TOÁN HỌC VÀ TIN HỌC

www.thnt.com.vn

1. Phương pháp phân rã hình học

Trong các kỳ thi Tin học lập trình, tỉ lệ xuất hiện bài toán về hình học là rất cao. Mà đó lại thường là những bài mà học sinh vấp váp, vì một trong các lý do sau đây:

- Thuật giải quá khó, không nghĩ ra.

- Nghĩ ra được thuật giải, nhưng không cài đặt được vì quá phức tạp.

- Thuật giải tốt, cài đặt xong, nhưng vẫn không ổn do những lỗi nho nhỏ tinh vi và khó tránh.

Trong bài viết này, tôi xin được trình bày về một phương pháp có thể áp dụng cho một lớp rất lớn các bài toán tin có nội dung hình học: đó là phân rã bài toán ban đầu ra, đưa nó về một vài mô hình thật là đơn giản và cài đặt chỉ cần trình độ trung bình khá là ổn. Nội dung chính của phương pháp này mà tôi muốn nói cùng các bạn là:

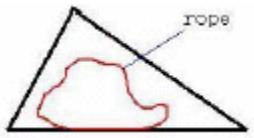
- Coi một góc là tập hợp vi phân các góc nhỏ liên tiếp. (1)

- Coi một bao hình là một tập hợp vi phân các điểm liên tiếp. (2)

Tất nhiên từ "vi phân" ở đây chỉ mang tính hình tượng, tức là một số vừa đủ lớn các góc vi phân, hay các điểm vi phân để cho (1) và (2) có thể coi như là đúng.

Chúng ta sẽ đưa vấn đề đi cụ thể hơn sau khi phân tích một số bài tin sau đây:

1. Diện tích trong tam giác (Problem G - The 2004 ACM Asia Programming Contest - Beijing):



Cho một tam giác và một vòng dây kín có độ dài biết trước. Hãy dùng vòng dây đó để khoanh một vùng kín nằm gọn trong tam giác sao cho diện tích phần thu được là lớn nhất.

Input: Gồm nhiều bộ test, mỗi bộ gồm đúng bốn số dương được viết trên cùng một dòng. Ba số đầu tiên là độ dài ba cạnh của tam giác, số cuối cùng là chu vi vòng dây. Độ dài các cạnh của mảnh vườn không quá 100. Độ dài vòng dây không lớn hơn chu vi tam giác.

Output: Gồm nhiều dòng, mỗi dòng ứng với một dòng trong input, chỉ ghi một số là diện tích lớn nhất có thể được, làm trọn với đúng hai chữ số sau dấu thập phân

Ví dụ:

Input:
12.0000 23.0000 17.0000
40.0000
84.0000 35.0000 91.0000
210.0000
100.0000 100.0000 100.0000
181.3800
Output:
89.35
1470.00
2618.00

Có thể không khó khăn lắm để nhận ra được thuật giải của bài toán này như sau: Tìm O là giao ba đường phân giác của tam giác đó. Ta gọi R là bán kính đường tròn tâm O (bạn cứ coi là có R rồi). Phần diện tích tối ưu là phần mà vừa nằm trong đường tròn vừa nằm trong tam giác, đồng thời chu vi của phần diện tích đó đúng bằng L là chu vi vòng dây. Còn để tìm R thì ta chặt nhị phân.

Chúng ta sẽ không bàn nhiều về thuật giải bài toán, cứ coi là bạn biết rồi đi. Vậy vấn đề là bạn sẽ cài đặt nó như thế nào? Quả thật rất khó để có thể hoàn thành bài này trong thời gian cho phép nếu như ta cứ cài đặt một cách thuần túy thông thường. Như vậy thì chẳng những mất thời gian mà hiệu quả đạt được còn là rất thấp.

Sở dĩ bài này khó cài đặt là bởi vì ứng với mỗi R ta có rất nhiều trường hợp có thể xảy ra về vị trí tương đối của hình tròn (O) và tam giác ABC. Số điểm giao nhau có thể không có, có thể có một, có hai,..., hoặc có sáu giao điểm. Các giao điểm lại không xếp theo quy luật gì nên lúc thực sự tính toán lại nảy sinh nhiều vấn đề rất phức tạp, ví dụ như trên mỗi cạnh có mấy giao điểm? Điều này phải xác định rõ ràng, nếu không thì không thể tính được chu vi và diện tích của hình cần thiết. Tính sơ sơ ra số trường hợp cần xét cũng quá lớn và trong mỗi trường hợp cũng đủ rắc rối để cho ta có thể giải quyết bài này một cách nhanh gọn và chính xác (Tôi đã làm thử

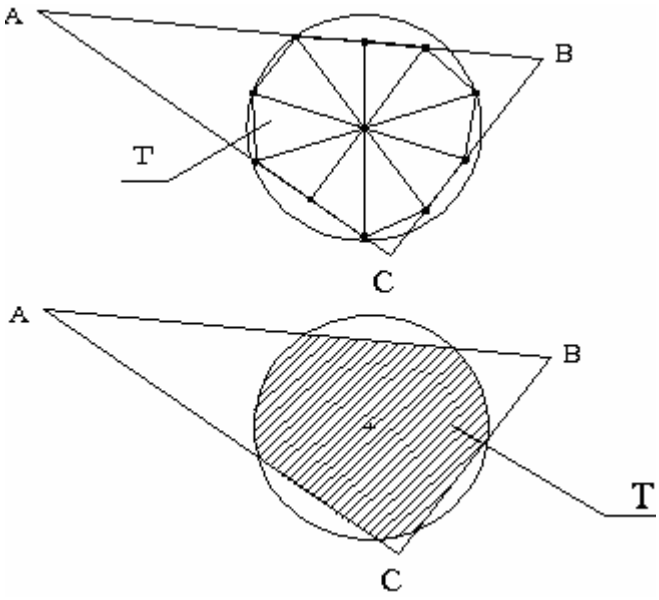
rồi). Vậy nếu đây là một bài thi quan trọng thì trong phòng thi liệu bạn có đủ bình tĩnh để làm một cách chính quy như trên? Tôi xin phát biểu phương án về cách phân rã bài toán này, để biến nó thành một bài toán đơn giản và rất dễ cài đặt, và từ đó tổng quát hóa lên một phương án tuyệt vời:

Đầu tiên phải thấy được là cái khó chỉ là làm sao xác định rõ những giao điểm cần thiết. Gọi (T) là hình tạo bởi (0) và ABC ứng với R biết trước. Rõ ràng là (T) là một hình gồm có những phần đoạn thẳng (thuộc tam giác ABC) và những phần đường tròn (thuộc (O). Bây giờ ta không quan niệm (T) như vậy nữa, ta coi nó gần đúng là một đa giác (T') gồm rất nhiều điểm Mi, với $1 \leq i \leq P$, với P bạn khai báo const ở đầu chương trình, P càng lớn càng tốt, nhưng không nên quá lớn vì chương trình sẽ chạy lâu hơn.

Xác định các điểm Mi: Ta chia góc 3600 tại O ra làm P góc đều nhau. Ứng với góc thứ i đó ta vẽ một tia Oi, sau đó ta xác định xem tia Oi cắt hình tròn trước hay cắt tam giác trước. Michính là giao điểm gần O nhất trong hai giao điểm đó.

Như vậy là thay vì phải tính toán với miền (T) vô cùng rắc rối, nếu ta coi nó như là một đa giác (T') gồm P đỉnh và xác định các điểm Mi dễ dàng như trên, thì công việc còn lại thật vô cùng đơn giản. Nhưng còn vấn đề sai số? Ta có thể khắc phục nó bằng một thủ thuật như sau:

Đặt M0 = MP và MP+1 = M1. Xác định 6 điểm i mà góc giữa Mi-1, Mi, Mi+1 là lớn nhất. Đó chính là sáu giao điểm gần như thực sự của (T). Bây giờ ta tính toán trực tiếp trên (T) bằng cách dùng các công thức chính xác cho cung tròn và đoạn thẳng thì vấn đề sai số coi như không còn.



Bạn thấy đó, cùng là một bài toán, nhưng nếu ta quan niệm nó khác đi một chút thì công việc sẽ được giảm tải đi rất nhiều lần. Không phải cứ cái gì tốt nhất về mặt lý thuyết cũng là tốt nhất với ta, nhất là trong các kỳ thi. Điều quan trọng là tính hiệu quả và thực tế của chương trình. Việc phân rã một hình (T) ra thành đa giác (T') cũng là điều thường gặp ở nhiều nơi, nhưng thủ thuật tách ra để rồi ghép lại thì quả là rất độc đáo. Thủ thuật này trong toán thường gặp khi phải tính tổng của một chuỗi số S, hay một chuỗi hàm f. Khi đó người ta hay vi phân vế trái, tính toán một hồi cho nó thật gọn rồi lại tích phân chính vế trái đó. Nếu như bạn để ý, về bản chất thì đó cũng là điều mà bài tin kia đã sử dụng, cho dù nó đã bị "tin hóa" đi bởi tham số P. Nhưng bạn cứ yên tâm, không có gì là tuyệt đối cả, nếu như P đủ lớn (khoảng vài nghìn) thì kết quả sẽ luôn ra tối ưu. Bài tin trên đã áp dụng cả (1) và (2) để giải quyết hiệu quả vấn đề. Có lẽ bạn vẫn chưa hình dung ra phương pháp này ra sao? Chúng ra hãy cùng bàn tiếp trong bài tiếp theo:

2. Chocolate

Nhà máy sản xuất bánh kẹo Fishburg sản xuất ra một loại bánh chocolate hình đa giác lồi. Kiđy và Carlson mua một cái và họ muốn cắt nó ra làm hai phần bằng nhau với một nhát cắt có độ dài nhỏ nhất. Viết chương trình tìm độ dài nhỏ nhất để cắt miếng bánh sử dụng những gì bạn được cho biết về miếng bánh. Tổng số đỉnh N là một số nguyên ($3 \le N \le 50$). Tọa độ của các đỉnh là tập hợp các cặp số thực $-100 \le xi, yi \le 100$.

Input: Dòng đầu của input file gồm số N - số lượng đỉnh của đa giác. N dòng sau gồm toạ độ của các đỉnh liên tiếp nhau của đa giác.

Output: gồm độ dài nhỏ nhất của đường cắt chính xác đến 0,0001.

Ví dụ:

Input:

```
4
0 0
0 3
4 3
4 0
```

Output:

```
3
```

Thuật giải tốt của bài này theo tôi là không phù hợp để bàn ra ở đây vì cái lõi toán của nó nhiều quá. Nói chung để mà vừa nghĩ thuật giải tốt hẳn và cài đặt xong nó

chắc cũng phải vất vả nhiều lắm mà cũng chẳng biết là liệu mình có kiểm soát được không nữa. Vậy chúng ta cùng bàn cách phân rã bài này ra cho nó đơn giản đi nhé.

Cách đơn giản nhất ai cũng có thể nghĩ ra là coi như đa giác này là một tập hợp các điểm rời rạc, sau đó ta lấy hai trong số những điểm đó, rồi kiểm tra xem đoạn thẳng nối hai điểm này có chia đôi được đa giác hay không, nếu như được rồi thì cập nhật kết quả thôi. Cải tiến của cách này là chỉ chọn một điểm thôi, điểm còn lại thì chặt nhị phân. Cách này về tư tưởng phân rã thì là tốt nhưng trên thực tế không phải là không có vấn đề. Điều kiện các tọa độ của đa giác có trị tuyệt đối không quá 100 cho nên sẽ vẫn có những test mà chu vi của nó sẽ khá là lớn (có thể lên tới hàng vạn). Mà theo như cách phân rã này thì độ sai số sẽ tỉ lệ theo chu vi của đa giác. Nếu chia đa giác thành P điểm, với độ phức tạp của cách này vượt quá Plog(P), thì rõ ràng muốn chạy nhanh được thì P không thể tới hàng vạn, cho nên khả năng chính xác tới nhiều chữ số sau dấu thập phân khi chu vi của đa giác quá lớn là điều không tưởng. Vậy (2) không dùng được ở đây.

Cách thứ hai có thể khắc phục nhược điểm trên, là ta phân rã các góc ra thành vô số các góc nhỏ vi phân. Giả sử số góc là P, thì một góc sẽ có giá trị là dP = 3600/P. Chắc hẳn đọc đề bài lần đầu ai cũng tưởng tượng là đa giác thì đứng yên, còn đường thẳng chia đôi đa giác sẽ xiên xiên. Ta hãy tưởng tượng ngược lại một chút như sau: Đường thẳng chia đôi đa giác thì luôn nằm ngang, có thể tịnh tiến lên xuống, còn đa giác thì có thể xoay. Về bản chất thì vẫn không có gì thay đổi, nhưng nó sẽ giúp ích nhiều cho ta. Tóm lại thuật giải sẽ như sau:

- Xoay đa giác đi một góc dP.
- Chặt nhị phân để tịnh tiến đường thẳng nằm ngang sao cho nó chia đôi đa giác kia. Nếu không chia vừa thì bằng cách chặt nhị phân ta có thể tịnh tiến đường thẳng này lên xuống đến bao giờ bằng nhau thì thôi.
- Sau khi đã chia đôi đa giác, cập nhật lại độ dài tốt nhất của nhát cắt. Sau khi xoay đi P lần, mỗi lần một góc dP thì đa giác sẽ quay về đúng vị trí ban đầu. Nếu như có bạn nào chưa biết công thức xoay hình, tôi xin được viết luôn ra đây:

xmới = xcũ.cos(alpha) – ycũ.sin(alpha).

ymới = xcũ.sin(alpha) + y cũ.cos(alpha).

Trong đó alpha là góc quay.

Tất nhiên P càng lớn thì càng tốt. Đối với bài này tôi để P = 35000/N và chạy đúng hết tất cả các test. Đó chính là kết quả của việc áp dụng tính phân rã thứ (1).

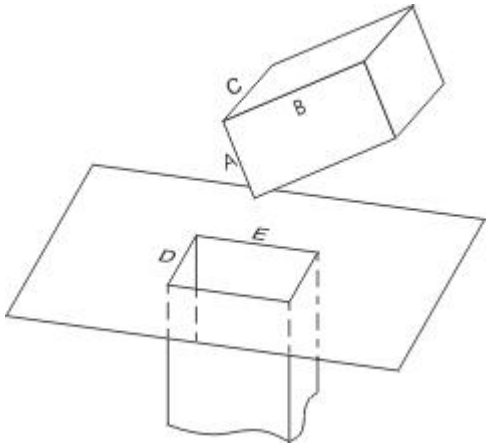
Kết luận: Có rất nhiều phương pháp để giải quyết một bài toán tin, chọn cách làm nào là tùy bạn. Một lời khuyên không bao giờ cũ của những người đi trước đối với tất cả chúng ta là: Không hẳn trong mọi sự lựa chọn ta đều lấy cái tốt nhất, hãy chọn cái phù hợp nhất đối với bạn. Trong phòng thi tâm lý ổn định là một điều rất quan trọng để dẫn tới thành công. Nhưng tâm lý sao ổn được khi bạn đang làm một việc quá sức mình? Nghệ thuật phân rã bài toán không phải là cái tốt nhất trong mọi trường, đó là điều chắc chắn. Nhưng nếu như trong một vài trường hợp cụ thể nào đó, có thể nó sẽ phù hợp với bạn đấy!

Và sau cùng là một bài tập luyện tập dành cho bạn.

Bricks - 2002-2003 ACM Northeastern European Regional Programming Contest

(Đề bài được tôi tóm tắt)

Có viên gạch kích thước A × B × C inches. Trên sàn nhà có một cái lỗ kích thước D × E inches, coi như cái lỗ là rất sâu. Hỏi liệu có thể xoay sở làm sao để nhét được viên gạch vào trong lỗ trên hay không?



Input: Gồm 5 số A, B, C, D, E, mỗi số không nhỏ hơn 1, không lớn hơn 10 và có nhiều nhất một chữ số sau dấu thập phân.

Output: Ghi “YES” nếu như có thể nhét gạch vào lỗ, ngược lại ghi “NO”.

Ví dụ:

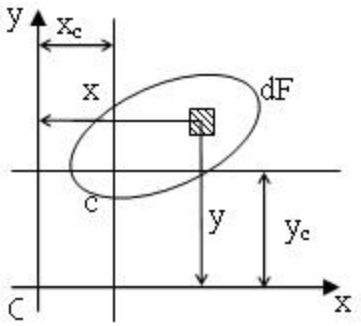
Input: 2.0 1.5 1.4 1.0
Output: NO
Input: 2.0 1.5 1.5 1.0
Output: YES

2.Xác định trọng tâm của một hình đa giác bất kỳ

Chắc đã có lần trong công việc hàng ngày, chúng ta đã gặp bài toán sau: "Trong mặt phẳng, cho một hình đa giác bất kì với toạ độ các đỉnh là số thực. Vấn đề đặt ra là xác định trọng tâm của hình đa giác đó".

Để làm được việc đó, sau đây xin tóm tắt lại lý thuyết đặc trưng hình học của mặt cắt ngang:

1. Mômen tĩnh của một hình phẳng F đối với hai trục Ox và Oy trong mặt phẳng của hình được định nghĩa lần lượt bằng biểu thức sau đây:



$$S_x = \int_F y dF$$

$$S_y = \int_F x dF$$

2. Trục trung tâm: Mômen tĩnh của một hình đối với một trục nào đó bằng không trục ấy gọi là trục trung tâm.

3. Trọng tâm: Giao điểm của hai trục trung tâm được gọi là trọng tâm mặt cắt. Trọng tâm là duy nhất đối với một hình phẳng.

4. Quan hệ giữa mômen tĩnh của một hình đối với một trục và khoảng cách từ trọng tâm của hình đến trục đó.

a) Giả sử có trục x bất kỳ và trục trung tâm xc (C là trọng tâm mặt cắt) song song với trục x. Ta có y = yc + y0.

Thay vào công thức định nghĩa, ta được:

$$S_x = \int_F y dF = \int_F (y_c + y_o) dF = y_c \int_F dF + \int_F y_o dF$$

Hay

$$S_x = y_c F + \int_F y_o dF$$

Theo định nghĩa số hạng thứ hai vế phải bằng không, do đó:

Sx = ycF

Hay

$y_c = \frac{S_x}{F}$

Tương tự ta tính được:

$S_y = x_c F \; ; \; x_c = \frac{S_y}{F}$

Như vậy là từ các công thức trên, ta có thể tính được mômen tĩnh của một hình nếu biết trọng tâm hoặc ngược lại xác định được trọng tâm nếu biết mômen tĩnh của hình mà không phải qua phép tính tích phân.

b) Từ đó ta có công thức tính trọng tâm hình ghép nếu biết trọng tâm của các hình thành phần.

$$y_c = \frac{\sum_{i=1}^{n} y_{ci} F_i}{F} \; ; \; x_c = \frac{\sum_{i=1}^{n} x_{ci} F_i}{F}$$

Nhận xét: Từ công thức này ta có thể tính được trọng tâm của một hình đa giác bất kỳ dựa vào các tam giác thành phần.

Công thức tính trọng tâm G, và diện tích F của hình tam giác biết toạ độ 3 đỉnh A (XA, YA), B (XB, YB) và C (XC, YC).

$$x_G = \frac{x_A + x_B + x_C}{3}$$

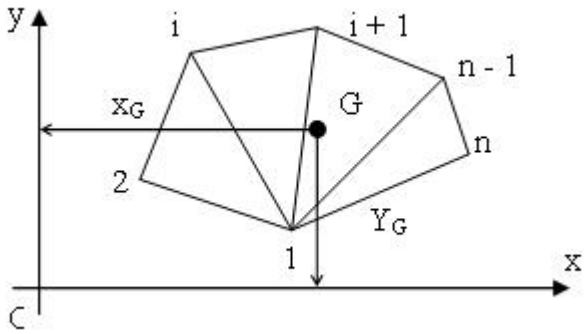
$$y_G = \frac{y_A + y_B + y_C}{3}$$

$$F = |(x_A - x_B)(y_A + y_B) + (x_B - x_C)(y_B + y_C) + (x_C - x_A)(y_C + y_A)|/2$$

Dựa vào nhận xét trên đây tôi xin giới thiệu chương trình tính trọng tâm của một hình đa giác lồi bất kỳ.

Dữ liệu vào là n (n > 2) điểm (trong mặt phẳng Oxy) – toạ độ n đỉnh liên tiếp nhau của đa giác lồi. Ta chia đa giác lồi này thành n-2 tam giác với 3 đỉnh của tam giác

lần lượt là đỉnh thứ 1, đỉnh thứ i và đỉnh thứ i + 1 ($2 \le i \le n - 1$). Dữ liệu vào là n (n > 2) điểm (trong mặt phẳng Oxy) – toạ độ n đỉnh liên tiếp nhau của đa giác lồi. Ta chia đa giác lồi này thành n-2 tam giác với 3 đỉnh của tam giác lần lượt là đỉnh thứ 1, đỉnh thứ i và đỉnh thứ i + 1 ($2 \le i \le n - 1$).



Từ đây ta có thể xây dựng chương trình, sau đây là toàn văn chương trình:

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+,Y+}
{$M 16384,0,655360}
Program Xac_dinh_trong_tam ;
Const
Maxn = 1000 ;
FileInp = 'TTAM.INP' ;
FileOut = 'TTAM.Out' ;
tp = 2 ; {So chu so thap phan can}
Type
Toado = Record
x, y : Real ;
End ;
Mang = Array [1.. Maxn] of Toado ;
Var
A : Mang ;
XG, YG : Real ;
tongx, tongy, tong : Real ;
N : Integer ;

Procedure Docfile ;
Var
f : Text ;
i : Integer ;
Begin
```

```
Assign (f, FileInp) ;
{$I-}
Reset (f) ;
{$I+}
If IOResult <> 0 then Halt ;
Readln (f, N) ;
FillChar (A, Sizeof (A), 0) ;
For i := 1 to N do
Readln (f, A [i].x, A [i].y) ;
Close (f) ;
tongx := 0 ;
tongy := 0 ;
tong := 0 ;
End ;

Function XAG (AA, BB, CC : Toado) : Real ;
Begin
XAG := (AA.x + BB.x + CC.x) / 3 ;
End ;

Function YAG (AA, BB, CC : Toado) : Real ;
Begin
YAG := (AA.y + BB.y + CC.y) / 3 ;
End ;

Function SA (AA, BB, CC : Toado) : Real ;
Var
tam : Real ;
Begin
tam := (AA.x - BB.x) * (AA.y + BB.y) +
(BB.x - CC.x) * (BB.y + CC.y) +
(CC.x - AA.x) * (CC.y + AA.y) ;
SA := Abs (tam) / 2 ;
End ;

Procedure Xuly ;
Var
i : Integer ;
tamx, tamy, tamS : Real ;
Begin
For i := 2 to n - 1 do
```

```
Begin
tamx := XAG (A [1], A [i], A [i + 1]) ;
tamy := YAG (A [1], A [i], A [i + 1]) ;
tongx := tongx + tamx * tamS ;
tongy := tongy + tamy * tamS ;
tong := tong + tamS ;
End ;
XG := tongx / tong ;
YG := tongy / tong ;
End ;

Procedure Ghifile ;
Var
f : Text ;
Begin
Assign (f, FileOut) ;
Rewrite (f) ;
Writeln (f, XG : 0 : tp, #32, YG : 0 : tp) ;
Close (f) ;
End ;
Begin
Docfile ;
Xuly ;
Ghifile ;
End.
```

File vào TTAM.INP

```
4
0 0
4 0
4 4
0 4
```

File ra TTAM.OUT

```
2.00 2.00
```

Bạn đọc có thể tìm hiểu thêm để xác định được trọng tâm của một hình bất kỳ (có cả phần khuyết bên trong) đồng thời có thể xác định thêm các đặc trưng hình học khác như mô men quán tính Jx, Jy, Jxy, bán kính quán tính ix, iy… Rất mong sự quan tâm và trao đổi của quý bạn đọc.

Bàn thêm về cặp ghép

Lưu anh tuấn

Bài toán cặp ghép là 1 bài toán rất cơ bản và cũng có rất nhiều ứng dụng trong thực tế. Trên ISM đã có rất nhiều bài viết viết về những vấn đề liên quan đến bài toán này. Bài viết của tôi chỉ xin nói thêm về 1 khía cạnh ít được đề cập đến. Đó là đếm số lượng cặp ghép.

Bài toán phát biểu như sau:

Cho N sinh viên(N<=12) và N vấn đề cần nghiên cứu. Mỗi sinh viên sẽ hứng thú với 1 số vấn đề, và khi sinh viên được giao vấn đề họ thích thì họ sẽ làm việc hiệu quả hơn rất nhiều. Ngài giáo sư đáng kính của chúng ta muốn biết có bao nhiêu cách ghép sao cho mỗi sinh viên sẽ giải quyết 1 vấn đề mà họ thích.

Giáo sư sẽ cung cấp cho chúng ta 1 ma trận A kích thước NxN trong file PROBLEM.TXT với
+ A[i,j]=1 khi sinh viên i thích vấn đề j.
+ A[i,j]=0 khi sinh viên i không thích vấn đề j.
Yêu cầu: Bạn hãy viết 1 chương trình tính số ghép thoả mãn yêu cầu của giáo sư và gửi file kết quả SOLVE.TXT cho giáo sư.

Ví dụ:

PROBLEM.TXT
3
1 1 1
1 1 1
1 1 0
SOLVE.TXT
4
Giải thích : 4 cặp ghép là
((1,2),(2,3),(3,1))
((1,1),(2,3),(3,2))
((1,3),(2,1),(3,2))
((1,3),(2,2),(3,1))

Bài toán trên ta có thể giải theo cách tầm thường là tìm toàn bộ cách khả năng có thể ghép bằng cách vét cạn, độ phức tạp là N!. Trong trường hợp ma trận A gồm toàn số 1, số cách chọn sẽ là N!. Dù N<=12 nhưng N! vẫn là 1 giá trị “khủng khiếp”.

Sau đây tôi xin đề xuất cách giải với thuật toán QHĐ trạng thái. Xin nói qua về QHĐ trạng thái. QHĐ trạng thái là QHĐ trên các trạng thái, các trang thái thường được biểu diễn bằng 1 dãy bít hoặc tính trước.

Ví dụ 1: Bài 1 thi QG năm 2006 bảng B (tôi không nói lại đề) : Ta dùng QHĐ trạng thái với 8 trạng thái cho mỗi dòng : (0,0),(0,1),(0,2),(0,3),(0,4),(1,3),(1,4),(2,4) với ý nghĩa (i,j) là chọn ô i và ô j, giá trị 0 là không chọn ô nào. Ví dụ 2: Bài viết "chia sẻ 1 thuật toán hay" của bạn Nguyễn Hiển. Bạn đã dùng 1 dãy bít với ý nghĩa là bít thứ i bằng 1 nếu công việc đó được chọn, bằng 0 nếu công việc đó không được chọn.

Trở lại bài toán của chúng ta. Ta biết: 1 cách ghép cặp là cách ghép 1 sinh viên và 1 vấn đề. Giả sử ta có 1 cách ghép cặp (x1,y1),(x2,y2),…,(xn,yn). Bây giờ ta bỏ đi 1 cặp (x1,y1). Cặp ghép còn lại là (x2,y2),(x3,y3),…,(xn,yn) vẫn là 1 cặp ghép, ta có bài toán với kích thước nhỏ hơn. Như vậy các bạn đã thấy rõ bản chất QHĐ của bài toán này. Để tìm số cách ghép của N sinh viên, ta phải tìm số cách ghép của N-1 sinh viên.

Ta định nghĩa 1 dãy bít X thay cho các trạng thái của các vấn đề. X[i]=1 nếu vấn đề i được chọn. X[i]=0 nếu vấn đề i không được chọn. Độ dài dãy bít tối đa là 12 nên ta thay 1 dãy bít X bằng 1 giá trị TX.

Vì cặp ghép là đầy đủ nên số sinh viên ghép với 1 trạng thái X là số giá trị 1 trong X. Ta cố định các sinh viên này và duyệt qua tất cả các trạng thái X. Gọi D[TX] là số cách ghép cặp 1 trạng thái X với sl sinh viên đầu tiên, sl là số bít 1 của trạng thái X. Ta có công thức QHĐ: D[TX] := D[TX]+D[TX xor (1 shl i)] với i thoả mãn X[i]=1 và có sinh viên sl thích vấn đề i. TX xor (1 shl i) có ý nghĩa là thay giá trị bít thứ i thành 0, ta đã giảm số vấn đề được chọn đi 1. Sau đây là chương trình:

```
{ Sử dụng Free Pascal }

Const max = 1 shl 12;
fi = 'PROBLEM.TXT';
fo = 'SOLVE.TXT';
Var n : Integer;
f ,g : text;
A : array[0..20,0..20] of Boolean;
D : array[0..max] of longInt; {Mảng D có ý nghĩa như trên }
T : array[0..20] of Integer; { T lưu lại vị trí các bít 1 để dễ dàng QHĐ hơn }

Procedure Tinh( TX : LongInt );
```

```
Var gt , j , i , sl : LongInt;
{sl là số lượng bít 1}
Begin
gt := TX;
i := -1;
sl := -1;
While gt> 0 do {vong while de tim cac bit 1 trong phan tich nhi phan so TX}
Begin
Inc( i );
If gt and 1 = 1 then {neu bít i là 1 }
Begin
Inc(sl);
T[pt]:=i; {luu lai vi tri cac bit 1}
End;
gt:= gt shr 1;
End;
D[TX]:=0;
For j :=0 to sl do
If A[ sl , T[j] ] then {Sinh viên sl thích vấn đề T[j]}
Inc( D[TX] , D[ TX xor (1 shl T[j])] );
{TX xor (1 shl T[j] là tắt bit thứ T[j]}
End;

Procedure Xuli;
Var TX:LongInt;
Begin
D[0]:=1;
For TX:=1 to (1 shl n)-1 do
Tinh(TX); {QHD voi so TX
Writeln(g, D[1 shl n-1] );
End;

Procedure Nhap;
Var i ,j,t:Integer;
Begin
Read(f,n);
For i:=0 to n-1 do
For j:=0 to n-1 do
Begin
Read(f,t);
A[i,j]:= t =1;
```

```
End;
End;

Begin

assign(f,fi);reset(f);
assign(g,fo);rewrite(g);
fillchar(d,sizeof(d),0);
Nhap;
Xuli;
close(f);close(g);
End.
```

Thuật toán trên có độ phức tạp khoảng 2^N, hiệu quả hơn rất nhiều so với cách duyệt bình thường.

Bài toán trên đã giải quyết xong. Bây giờ, ta sẽ thay đổi bài toán trên 1 chút:

Vị giáo sư đáng kính muốn biết có bao nhiêu cách ghép cặp mà trong đó có chứa cặp sinh viên x và vấn đề y.,/p>

Khi ta đã giải quyết được bài toán trên thì bài toán mở rộng trở nên quá dễ.

Trên đây, tôi xin bàn thêm về bài toán cặp ghép. Để nói hết thì thật là khó. Hi vọng các bạn sẽ cùng tôi khám phá những điều mới mẻ và lý thú từ những thuật toán hay.

Một vài bài tập về Palindrome

Palindrome hay còn gọi là xâu đối xứng, xâu đối gương là tên gọi của những xâu kí tự mà khi viết từ phải qua trái hay từ trái qua phải thì xâu đó không thay đổi. VD: MADAM, IOI,... Nhờ tính chất đặc biệt đó mà có khá nhiều bài tập có liên quan đến Palindrome, phần lớn trong chúng thường đi kèm với QHĐ. Tôi xin giới thiệu với các bạn một vài bài tập như vậy.

Bài 1: Xem một xâu có phải là Palindrome hay không?

Đây là một bài cơ bản, nhưng quan trọng vì nó được đề cập đến trong nhiều bài tập khác. Cách làm tốt nhất là duyệt đơn thuần mất O(N).

```
function is_palindrome(s: string): boolean;
var i, n : integer;
begin
n := length(s);
for i := 1 to (n div 2) do
if s[i] <> s[n+1-i] then
begin is_palindrome := false; exit; end;
is_palindrome := true;
end;
```

Một đoạn chương trình khác :

```
function is_palindrome(s : string) : boolean;
var i, j : integer;
begin
i := 1;
j := length(n);
while i
begin
if s[i] <> s[j] then
begin is_palindrome := false; exit; end;
inc(i);
dec(j);
end;
is_palindrome := true;
end;
```

Bài 2: Cho một xâu S <= 1000 kí tự; tìm palindrome dài nhất là xâu con của S (Xâu con là một dãy các kí tự liên tiếp).

Đây cũng là một bài cơ bản với nhiều cách làm.

Cách 1: QHĐ

Dùng mảng F[i, j] có ý nghĩa: F[i, j] = true/false nếu đoạn gồm các kí tự từ i đến j của S có/không là palindrome.

Ta có công thức là:

* F[i, i] = True
* F[i, j] = F[i+1, j-1]; (nếu s[i] = s[j])
* F[i, j] = False; (nếu s[i] <> s[j])

Đoạn chương trình như sau:

```
FillChar( F, sizeof(F), false );
for i := 1 to n do F[i, i] := True;
for k := 1 to (n-1) do
for i := 1 to (n-k) do
begin
j := i + k;
F[i, j] := ( F[i+1, j-1] ) and (s[i] = s[j] );
end;
```

i∀Kết quả là : Max(j-i+1) <=j thỏa F[i,j] = True.

Độ phức tạp thuật toán là 0(N2).

Chú ý : Với N lớn, ta phải thay mảng 2 chiều F bằng 3 mảng 1 chiều và dùng thêm biến max lưu giá trị tối ưu.

Cách 2: Duyệt có cận.

Ta xét từng vị trí i:

+ xem a[i] có phải là tâm của Palindrome có lẻ kí tự không?

(ví dụ Palindrome MADAM có tâm là kí tự D)

+ xem a[i] và a[i+1] có phải là tâm của Palindrome có chẵn kí tự không?

(ví dụ Palindrome ABBA có tâm là 2 kí tự BB)

với mỗi kí tự ta tìm palindrome dài nhất nhận nó là tâm, cập nhập lại kết quả khi duyệt. Ta duyệt từ giữa ra để dùng kết quả hiện tại làm cận.

Đoạn chương trình như sau:

```
procedure Lam;
var i, j : Longint ;
{ }
procedure try( first, last : Longint );
var đ : Longint;
begin
if first = last then
```

```
begin đ := 1; dec(first); inc(last); end
else đ := 0;
repeat
if (first < 1) or (last > N) then break;
if s[i] = s[j] then
begin
đ := đ + 2;
first := first - 1;
last := last + 1;
end
else break;
until false;
if max < dd then max := dd;
end;
{ }
begin
i := n div 2;
j := n div 2 + 1;
max := 1;
while (i > max div 2) and (j <= N-max div 2) do
begin
if i > max div 2 then
begin
try( i, i );
try( i, i+1 );
end;
if j <= N - max div 2 then
begin
try( j, j );
try( j, j+1 );
end;
i := i - 1;
j := j + 1;
end;
end;
```

Cách làm này có độ phức tạp: max*(N-max). Vì vậy nó chạy nhanh hơn cách QHĐ trên, thời gian chậm nhất khi max = N/2 cũng chỉ mất N2/4 nhanh gấp 4 lần cách dùng QHĐ. Nhờ vậy, chúng ta biết là: không phải lúc nào QHĐ cũng chấp nhận được về mặt thời gian và không phải lúc nào duyệt lúc nào cũng chậm.

Bài trên còn có một cách NlogN nữa là dùng Suffix Aray, thậm chí có cách O(N) là sử dụng Suffix Tree và thuật toán tìm LCA. Đương nhiên cách cài đặt không hề dễ dàng, tôi sẽ thảo luận với các bạn vào một dịp khác.

Bài 3: Chia một xâu thành ít nhất các Palindrome (độ dài <=1000). Bài này phức tạp hơn bài trên, cách làm thì vẫn là QHĐ.

Gọi F[i] là số palindrome ít nhất mà đoạn 1..j chia thành được.

Ta có công thức:

F[i] = max(F[j] + 1; j < i thỏa mãn:đoạn j+1..i là palindrome)

Đoạn chương trình như sau:

```
F[0] := 0;
for i := 1 to n do
begin
for j := i-1 downto 0 do
if (đoạn j+1..i là palindrome) then F[i] := max( F[i], F[j]+1 );
end;
```

Hai vòng for lồng nhau mất O(N2), phần kiểm tra đoạn j+1..i là palindrome hay không mất O(N), vậy độ phức tạp thuật toán là O(N3). Sẽ không được khả thi nếu N = 1000. Để giảm độ phức tạp thuật toán, ta sử dụng mảng L[i, j] có ý nghĩa tương tự như mảng F[i, j] ở bài 1. QHĐ lập mảng L[i, j] mất N2. Tổng cộng là O(N2) vì mỗi lần kiểm tra chỉ mất O(1).

Nhưng đến đây lại nảy sinh vấn đề: mảng L[i, j] không thể lưu được khi N=1000 vì bộ nhớ của chúng ta chỉ có 640KB. Một cách khắc phục là dùng xử lý bít. Nhưng có cách đơn giản hơn là dùng hai mảng một chiều L[i] và C[i] có ý nghĩa:

* L[i] là độ dài lớn nhất của palindrome độ dài lẻ nhận s[i] làm tâm;

* C[i] là độ dài lớn nhất của palindrome độ dài chẵn nhận s[i] và s[i+1] làm tâm;

L[i] và C[i] có thể tính được bằng cách 2 bài 2 trong O(N2). Phần kiểm tra ta viết lại như sau:

```
function is_palindrome(i, j : integer) : boolean;
var đ : integer;
begin
```

```
đ := j-i+1;
if ođ (đ) then is_palindrome := (L[(i+j) div 2] >= n)
else is_palindrome := (C[(i+j) div 2] >= n)
end;
```

Vậy thuật toán của chúng ta có độ phức tạp tính toán là O(N2), chi phí bộ nhớ là O(N).

Bài 4 : Pal - Ioicamp - Marathon 2005-2006- tuần 17

Cho một xâu, hỏi nó có bao nhiêu xâu con là palindrome; xâu con ở đây gồm các kí tự không cần liên tiếp (độ dài <= 120).

Ví Dụ:

Pal.inp
IOICAMP
Pal.out
9

Đây là một bài tập rất thú vị. Phương pháp là dùng QHĐ.

Gọi F[i, j] là số palindrome là xâu con của đoạn i..j.

Ta có công thức :

* F[i, i] = 1;
* F[i, j] = F[i+1, j] + F[i, j-1] - F[i+1, j-1] + T;
Nếu s[i] = s[j] thì T = F[i+1, j-1] + 1;
Nếu s[i] <> s[j] thì T = 0;

Đoạn chương trình như sau :

```
procedure lam;
var k, i, j : integer;
begin
n := length(s);
for i := 1 to n do F[i, i] := 1;
for k := 1 to n-1 do
for i := 1 to n-k do
begin
j := i+k;
```

```
F[i, j] := F[i, j-1] + F[i+1, j] - F[i+1, j-1];
if s[i] = s[j] then F[i, j] := F[i, j] + F[i+1, j-1] + 1;
end;
end;
```

Để chương trình chạy nhanh hơn, chúng ta sửa lại đoạn mã một chút như sau :

```
procedure lam2;
var k, i, j : integer;
begin
n := length(s);
for i := 1 to n do F[i, i] := 1;
for k := 1 to n do
for i := 1 to n-k do
begin
j := i+k;
F[i, j] := F[i, j-1] + F[i+1, j];
if s[i] = s[j] then F[i, j] := F[i, j] + 1
else F[i, j] := F[i, j] - F[i+1, j-1];
end;
end;
```

Đoạn chương trình trên chỉ có tính mô phỏng, muốn hoàn thiện bạn phải cài đặt các phép tính cộng trừ số lớn vì kết quả có thể lên tới 2n-1. Độ phức tạp của thuật toán là O(N2). Vì vậy, chúng ta hoàn toàn có thể làm với N = 1000, khí đó cần rút gọn mảng F thành ba mảng một chiều.

Bài 5: Palindrome - IOI 2000

Cho một xâu, hỏi phải thêm vào nó ít nhất bao nhiêu xâu kí tự để nó trở thành một palindrome (độ dài <= 500).

Bài này cũng sử dụng QHĐ:

Gọi F[i, j] là số phép biến đổi ít nhất cần thêm vào đoạn i..j để đoạn i..j trở thành palindrome.

Ta có công thức :

* F[i, i] = 0;
* Nếu s[i] = s[j] thì F[i, j] = F[i+1, j-1]
* Nếu s[i] <> s[j] thì F[i, j] = Min(F[i, j-1], F[i+1, j]) + 1;

Muốn chương trình chạy với n = 500 thì cần rút gọn F thành ba mảng một chiều. Muốn truy vết, bạn phải dùng mảng bít hoặc dùng dữ liệu động.

Để thực hành, bạn hãy làm bài tập sau :

Bài 6: The next palindrome - SPOJ

Cho nhiều số <= 106, với mỗi số, tìm số bé nhất có dạng palindrome lớn hơn số đã cho. Mở rộng với câu hỏi: Tìm số bé thứ k?

Ví Dụ :

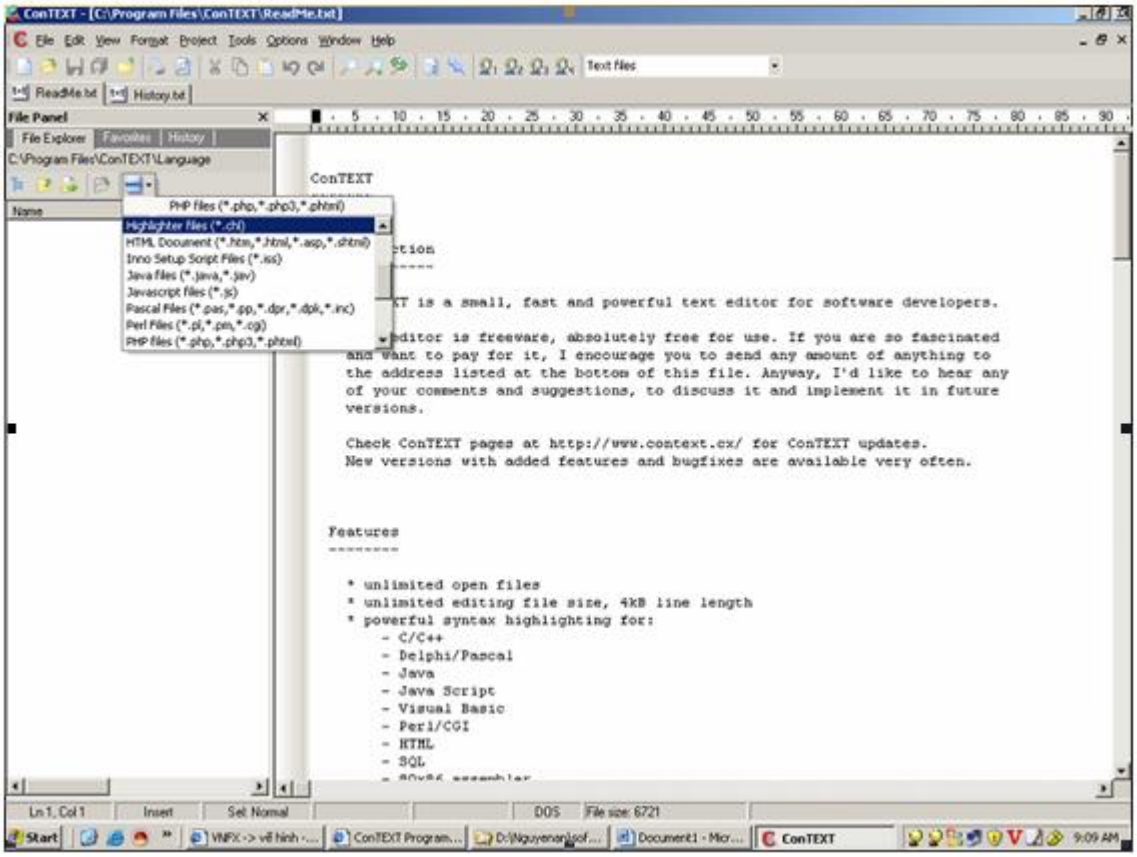
Input:
2
808
2133
Output:
818
2222

Gợi ý: dùng phương pháp đếm kết hợp QHĐ.

Context - Trình soạn mã nhỏ gọn và miễn phí cho dân lập trình

Nếu bạn là một lập trình viên, cùng lúc sử dụng nhiều ngôn ngữ, thì việc chuyển đổi qua lại giữa các môi trường trình biên dịch cũng đã khá là nhức đầu. Với phần mềm ConTEXT bạn sẽ thực sự thoải mái hơn trong việc soạn mã.

Tuy kích thước nhỏ gọn nhưng ConTEXT thật sự mạnh mẽ hỗ trợ tất cả ngôn ngữ mà bạn đã và sẽ học: C/C++, Delphi/Pascal, 80x86 assembler, Java, Java Script, Visual Basic, Perl/CGI, HTML, SQL, Python, PHP, Tcl/Tk.Vì vậy chỉ cần sử dụng ConTEXT bạn đã có thể cùng lúc soạn mã cho nhiều ngôn ngữ khác nhau. Riêng với các bạn sinh viên thì phần mềm ConTEXT thực sự hữu ích vì giúp họ thoát khỏi sự khó chịu khi phải sử dụng trình biên dịch Borland C++. Tuy nhiên phần mềm lại không có chức năng biên dịch nên hơi bất lợi đối những ai mới học lập trình còn có thói quen “viết tới đâu, chạy thử tới đó”.



Ngoài ra ConTEXT còn rất nhiều tính năng hữu ích,ví dụ như chuyển đổi văn bản giữa các dạng DOS,Unicode,UNIX, Macintosh, cả chức năng so sánh giữa các file và xuất ra file registry setting (.reg).... Chi tiết về phần mềm tôi xin để các bạn tự tìm hiểu và tôi hi vọng các bạn sẽ có thêm một công cụ đắc lực cho công việc của mình. Bạn có thể download miễn phí tại www.context.cx

3. Một bài toán về bàn cờ

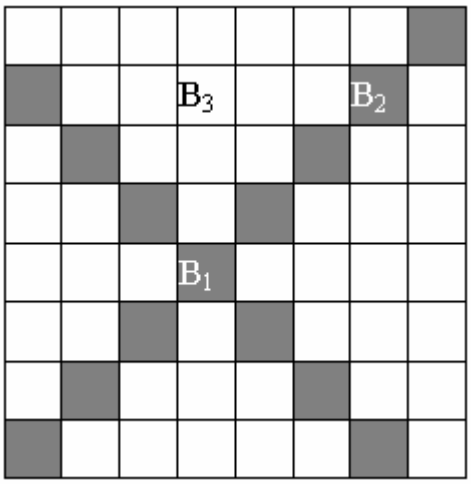
Đặng Chiến Công

Bài toán 1: Trên bàn cờ Quốc tế 8x8, cho 8 con hậu. Mỗi con hậu có thể khống chế (ăn) được tất cả các con hậu khác nằm trên cùng hàng, cùng cột, hoặc trên hai đường chéo đi qua vị trí của nó. Viết chương trình tính số các thế cờ chỉ gồm 8 con hậu trên bàn cờ sao cho không có hai con hậu nào có thể khống chế (ăn) được nhau.
Bài toán trên chính là bài toán con hậu nổi tiếng. Đây là một bài toán kinh điển và lời giải kinh điển của nó là thuật toán duyệt bằng phương pháp quay lui. Vì vậy nếu theo phương pháp này, bài toán con hậu khó có thể giải được với những dữ liệu lớn (bàn cờ kích thước lớn hơn, số con hậu nhiều hơn) vì độ phức tạp tính toán của thuật giải là một hàm số mũ. Tuy nhiên, bài báo này không có ý định tìm ra lời giải ưu việt cho bài toán con hậu (đây là bài toán khó khăn thực sự) mà thay vào đó, chúng

ta sẽ nghiên cứu một bài toán tương tự, đơn giản hơn nhưng cũng không kém phần thú vị, đó là bài toán:

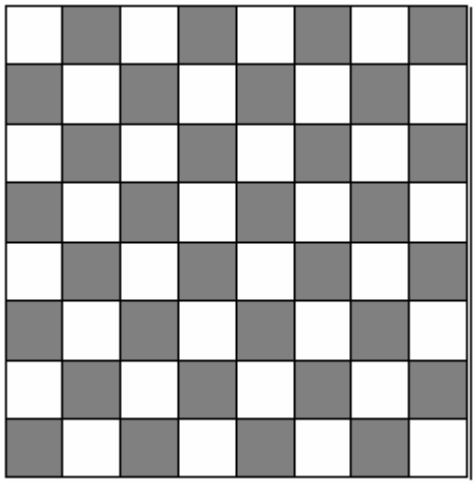
Bài toán 2: Trên bàn cờ vua, con tượng chỉ có thể di chuyển theo đường chéo và hai con tượng có thể khống chế (ăn) nhau nếu chúng nằm trên đường di chuyển của nhau. Trong hình sau, hình vuông tô đậm thể hiện các vị trí mà con tượng B1. Quân B1 và B2 khống chế (ăn) nhau, quân B1 và B3 không khống chế (ăn) nhau. Cho một bàn cờ kích thước NxN. Hãy tính số các thế cờ chỉ bao gồm K con tượng mà không có con tượng nào có thể khống chế nhau. (N, K là các số tự nhiên cho trước).

Bài toán 1 gọi là bài toán con hậu, bài toán 2 tạm gọi là bài toán con tượng. Dễ thấy con tượng có vùng "phủ sóng" hạn chế hơn con hậu. Con tượng chỉ có khả năng khống chế các quân cờ khác nằm trên cùng đường chéo với nó trong khi con hậu còn khống chế được cả thêm các quân nằm trên cùng hàng, cùng cột. Do đó, một cách hiển nhiên thì bài toán con tượng cũng có thể giải tương tự như bài toán con hậu bằng thuật toán quay lui. Tuy nhiên, ở đây chúng ta sẽ nghiên cứu một phương pháp khác để tính số thế cờ mà đề bài yêu cầu. Chúng ta sẽ cố gắng xây dựng một công thức tính số thế cờ đó.



I. Xây dựng công thức tính số cách sắp đặt các con tượng.

Giả sử chúng ta đang phải làm việc với một bàn cờ vuông kích thước NxN. Một bàn cờ vuông bình thường thì bao giờ mỗi ô trên đó cũng được tô bằng 2 màu: đen hoặc trắng (hoặc bằng cặp màu nào đó). Phương pháp tô theo cách sau: nếu một ô màu đen/trắng thì các các ô kề cạnh với nó sẽ có màu trắng/đen. Thí dụ theo hình vẽ dưới là một cách tô trong 2 cách có thể (chỉ cần tô một ô là xác định được màu của tất cả các ô còn lại, vì mỗi ô chỉ có thể là màu đen hoặc trắng nên cũng chỉ có 2 cách tô màu cho toàn bàn cờ, nói chung bàn cờ được tô thế nào cũng không quá quan trọng do những điều sắp được nói sau đây).



Sau khi đã tô màu cho tất cả các ô trên bàn cờ theo phương pháp trên, ta có thể rút ra một nhận xét: "các con tượng được đặt trên các ô màu đen sẽ không thể khống chế các con tượng nằm trên các ô màu trắng và ngược lại các con tượng được đặt trên các ô màu trắng cũng không thể khống chế các con tượng nằm trên các ô màu đen". Nhận xét này gợi ý cho chúng ta một phương pháp giải bài toán con tượng. Đó là coi tập các ô trắng và tập các ô đen trên bàn cờ là 2 bàn cờ con độc lập với nhau. Sau đó, ta tính số các thế cờ trên từng bàn cờ con đó rồi tổ hợp các kết quả với nhau thành công thức cuối cùng. Cách tính có thể được trình bày rõ ràng như sau:
Gọi DN(i), TN(i) (i=0, …, K) tương ứng là số các thế cờ chỉ bao gồm i con tượng trên bàn cờ các ô đen và bàn cờ các ô trắng của một bàn cờ vuông NxN và thỏa mãn không có hai con tượng nào có thể khống chế nhau. Như vậy, số các thế cờ cần tìm của bài toán 2 là:

$$S = \sum_{i+j=K} T_N(i) * D_N(j) \quad (1)$$

Với S là tổng số các thế cờ cần tính trên bàn cờ NxN.
Vấn đề còn lại giờ đây là tìm ra công thức tính TN(i) và DN(j).
Vẫn là bàn cờ được tô màu ở trên, nhưng chúng ta điền thêm các con số vào mỗi ô theo cách: ô ở dòng i cột j sẽ mang số j-i. Ta nhận thấy những ô cùng màu thì các số trên nó cũng cùng tính chẵn lẻ, những ô cùng nằm trên một đường chéo xuôi (đường chéo có hướng từ đỉnh trái-trên xuống đỉnh phải-dưới của bàn cờ) thì cùng mang những số có giá trị như nhau. Từ đây, chúng ta sẽ đánh số các đường chéo xuôi theo số của các ô trên nó, ví dụ ta có đường chéo số 0, số 1, -1,...

	1	2	3	4	5	6	7	8
1	0	1	2	3	4	5	6	7
2	-1	0	1	2	3	4	5	6
3	-2	-1	0	1	2	3	4	5
4	-3	-2	-1	0	1	2	3	4
5	-4	-3	-2	-1	0	1	2	3
6	-5	-4	-3	-2	-1	0	1	2
7	-6	-5	-4	-3	-2	-1	0	1
8	-7	-6	-5	-4	-3	-2	-1	0

Gọi F[m, k] (m, k ≥ 0) là số các thế cờ chỉ gồm k con tượng (hay số cách sắp đặt k con tượng) trên các ô thuộc các đường chéo số

$$\pm(m+2\times i) \text{ với } i=\overline{0,\left[\frac{N-m-1}{2}\right]}$$

của bàn cờ vuông NxN, sao cho không có hai con tượng nào có thể khống chế nhau (hay trên mỗi đường chéo, kể cả xuôi và ngược, chỉ có tối đa một con tượng).
Ví dụ, với bàn cờ 8x8 như hình vẽ trên, F[2, 4] là số cách sắp đặt 4 con tượng trên các ô thuộc các đường chéo -6, -4, -2, 2, 4, 6 sao cho không có hai con tượng nào có thể khống chế nhau.
Theo công thức trên, ta cũng thấy nếu đường chéo 0 được tô màu đen thì TN(i) = F[0,i] và DN(i) = F[1, i], thay vào biểu thức (1) ta có:

$$S=\sum_{i+j=K} F[0,i]*F[1,j] \quad (2)$$

Nếu ta tô màu ngược lại thì S vẫn có dạng như trên vì (hiển nhiên) S không phụ thuộc vào cách ta tô màu. Công thức trên đã gợi ý cho chúng ta đường lối rõ ràng hơn trong việc tìm S. Công việc duy nhất còn lại là tính các F[0, i] và F[1, j] hay nói một cách tổng quát hơn là ta phải đi tính F[m, k] với các số m và k cho trước.
Khi đối mặt với các công thức kiểu dạng F[m, k], phương án đầu tiên mà những người làm toán tin kinh nghiệm sẽ thường nghĩ đến là phương pháp tính truy hồi. Và quả như vậy, chúng ta sẽ tính công thức này theo công thức tính truy hồi.
Phương pháp tính truy hồi, xét về một khía cạnh nào đó thì cũng rất giống phương

pháp chứng minh quy nạp toán học. Do đó, bước đầu tiên, ta sẽ giả sử chúng ta đã tính được tất cả các F[i, j] với i = m+1, m+2,..., N-1 và j =0, 1,..., k. Và ta sẽ phải tính F[m, k] từ những công công thức đã đã tính được trước đó. Thực sự thì để tính F[m, k], điều giả sử của ta ở trên là hơi thừa. Thực tế, giả thiết chỉ cần đã tính được F[m+2, k], F[m+2, k-1] và F[m+2, k-2] là đủ.
Như đã nói ở trên, F[m, k] là số cách sắp đặt các con tượng hay là số các thế cờ (thỏa mãn các điều kiện của ta). Do đó, đối tượng mà ta quan tâm ở công thức F[m, k] là số các thế cờ, hay cụ thể là số các thế cờ gồm k quân tượng được sắt đặt trên các đường chéo ±m, ±(m+2), ±(m+4),… Ta sẽ chia tập các thế cờ này thành 4 tập con tương đương với 4 trường hợp sau:

TH1. Không có con tượng nào được đặt trên đường chéo m và –m. Số các thế cờ trong trường hợp này đơn giản là F[m+2, k].

TH2. Có 1 con tượng được đặt trên đường chéo m và không có con tượng nào trên đường chéo -m. Ta gọi tập các thế cờ trong trường hợp này là A. Sau đó ta xét các thế cờ đặt k-1 con tượng vào các đường chéo ±(m+2), ±(m+4), ±(m+6),… và ta lại gọi tập các thế cờ này là B. Với mỗi thế cờ thuộc tập B, ta sẽ đếm số ô không bị khống chế trên đường chéo m bởi k-1 con tượng trong thế cờ đó. Vì mỗi con tượng trong thế cờ chỉ có thể khống chế duy nhất một ô trên đường chéo m, nên sô ô bị khống chế trên đường chéo m là k-1. Đường chéo m có tổng cộng N-m ô, từ đây suy ra số ô không bị khống chế trên đường chéo m là N-m-k+1 ô. Trên bất kì N-m-k+1 ô này, ta có thể đặt thêm một con tượng vào thế cờ ta đang xét mà vẫn không có cặp con tượng nào không chế nhau. Như vậy với mỗi thế cờ thuộc tập B tương ứng sẽ có N-m-k+1 thế cờ thuộc tập A. Vậy số thế cờ thuộc tập A sẽ là (N-m-k+1)*F[m+2, k-1].

TH3. Có 1 con tượng được đặt trên đường chéo -m và không có con tượng nào trên đường chéo m. Lý luận tương tự như trường hợp vừa trên. Số thế cờ trong trường hợp này vân là (N-m-k+1)*F[m+2, k-1].

TH4. Trên mỗi đường chéo m và –m đều được đặt một con tượng. Lý luận tương tự như trường hợp 2. Trong trường hợp này, số ô không bị không chế trên mỗi đường chéo m và –m sẽ là N-m-k+2. Số ô không bị khống chế tăng lên một so với trường hợp 2 là do chỉ còn k-2 con tượng được đặt trên các đường chéo ±(m+2), ±(m+4), ±(m+6),… Nếu ta đặt một con tượng lên một trong N-m-k+2 ô không bị khống chế của đường chéo m thì trên đường chéo –m chỉ còn N-m-k+1 ô không bị không chế. Nên số cách đặt 2 con tượng lên 2 đường chéo m và –m là (N-m-k+2)*(N-m-k+1) và số thế cờ trong trường hợp 4 này sẽ là (N-m-k+2)*(N-m-k+1)*F[m+2, k-2].
Qua 4 trường hợp trên ta rút ra công thức cho F[m, k] sẽ là: F[m,k] = F[m+2,k] + (N – m – k + 1)*(2*F[m+2,k-1] + (N-m-k+2)*F[m+2,k-2]) (3)

Tuy nhiên không phải lúc nào ta cũng tính F[m, k] theo công thức (3). Vì với một số trường hợp đặc biệt, thì không phải lúc nào ta cũng có thể chia làm 4 trường hợp như trên. Cụ thể với m = 0 thì TH3 và TH4 không tính vì ta chỉ có một hàng 0 nên chỉ đặt được 1 con tượng. Trường hợp đặc biệt thứ 2 là k = 1, khi đó ta chỉ có thể đặt tối đa 1 con tượng lên hai đường chéo m và –m, nên TH4 sẽ không được tính. Nhóm các trường hợp đặc biệt còn lại là các trường hợp rất đơn giản mà ta có thể đưa ngay ra giá trị F[m, k] mà không cần tính toán nhiều. Tóm lại ta sẽ có các công thức tính F[m, k] cho từng trường hợp sau:
Nếu k = 0: F[m,0] = 1 với m >=0
Nếu m = N-1: F[N-1,1] =2; F[N-1,k] = 0 với k > 1
Nếu m = N-2: F[N-2,1] = 4, F[N-2,2] = 2, F[N-2, k] = 0 với k>2
Nếu m = 0: F[m,k] = F[m+2,k] + (N – m – k+1)*F[m+2,k-1]
Nếu k = 1: F[m,k] = F[m+2,k] + (N – m – k+1)*2*F[m+2,k-1]
Còn lại ta tính F[m, k] theo công thức (3).

II. Thuật toán – chương trình.
Về thuật toán có lẽ chúng ta không cần phải nói nhiều nữa. Ở đây tôi xin đưa ra một chương trình viết bằng C++ để minh họa. Chương trình này có thể chạy với các bộ dữ liệu n và k ở trong khoảng xung quanh 12, một tính với các bộ số lớn hơn ta phải thêm phần tính toán với số lớn.

III. Kết luận
Như vậy chúng ta đã giải quyết xong bài toán con tượng. Có thể nói bài toán con tượng này đã được giải quyết khá trọn vẹn. Nhờ có công thức tính mà bài toán con tượng đã trở nên "khả thi" hơn rất nhiều so với bài toán con hậu. Theo những ý tưởng từ bài toán con tượng trên, tôi cũng đã thử cải tiến phương pháp quay lui cho bài toán con hậu nhưng cho đến nay vẫn chưa thu được thành tựu nào đáng kể. Thực sự bài toán con hậu là một bài toán rất khó, nhưng rất mong có bạn nào đó sau khi đọc xong lời giải bài toán con tượng này có thể nghĩ ra được phương pháp có tính đột phá chẳng hạn? Chúc các bạn thành công.

Lập trình hiển thị công thức toán học

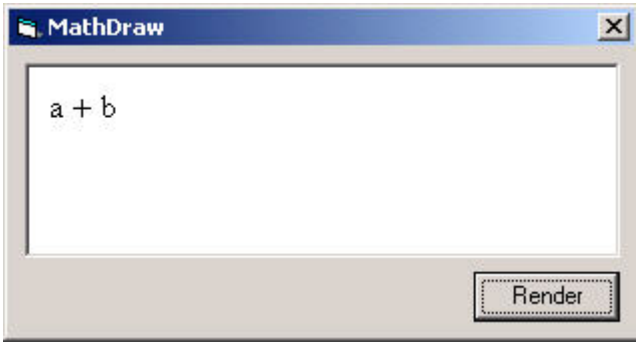
Thử nghiệm đối tượng Row

Chúng ta đã có thể chạy thử chương trình. Đoạn mã sau xử lý khi nhấn nút Render:

```
Private Sub cmdRender_Click()
Dim congthuc As Box
Dim so1 As New Text
```

```
Dim dau As New Operator
Dim so2 As New Text
Dim tong As New Row
so1.Content = 'á
so2.Content = 'b'
dau.Sign = '+'
tong.NumberElement = 3
Set tong.Member(1) = so1
Set tong.Member(2) = dau
Set tong.Member(3) = so2
Set congthuc = tong
congthuc.Layout picCanvas
congthuc.Draw picCanvas, 10, 10
End Sub
```

Chạy chương trình, nhấn nút Render, bạn sẽ nhận được kết quả:



Qua đoạn mã trên, ta thấy việc khởi tạo một đối tượng công thức khá rườm rà, dùng hơi nhiều biến trung gian. Cần một cách nào đó để thuận lợi hơn. Hãy chèn vào Project một Standard Module, đặt tên là Helper. Khai báo các hàm sau:

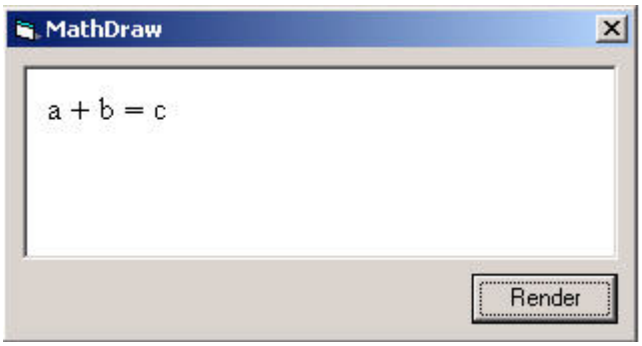
```
‘Standard Module Helper
Public Function Text(ByVal st As String) As Box
Dim t As New Text
t.Content = st
Set Text = t
End Function
‘----------------------------------------------------

Public Function Operator(ByVal st As String) As Box
Dim o As New Operator
o.Sign = st
Set Operator = o
```

```
End Function
‘--------------------------------------------------
Public Function Row3(box1 As Box, box2 As Box, box3 As Box) As Box
Dim r As New Row
r.NumberElement = 3
Set r.Member(1) = box1
Set r.Member(2) = box2
Set r.Member(3) = box3
Set Row3 = r
End Function
‘--------------------------------------------------
Public Function Row5(box1 As Box, box2 As Box, box3 As Box, box4 As Box,
box5 As Box) As Box
Dim r As New Row
r.NumberElement = 5
Set r.Member(1) = box1
Set r.Member(2) = box2
Set r.Member(3) = box3
Set r.Member(4) = box4
Set r.Member(5) = box5
Set Row5 = r
End Function
```

Hàm Text, Operator tạo các đối tượng kiểu Text và Operator tương ứng. Hàm Row3 tạo đối tượng kiểu Row có 3 thành phần (dạng x+y). Hàm Row5 tạo đối tượng kiểu Row có 5 thành phần (dạng x+y=z). Viết lại hàm xử lý nút Render như dưới đây:

```
Private Sub cmdRender_Click()
Dim congthuc As Box
Set congthuc = Row5(Text('á),Operator('+'),Text('b'),Operator('='),Text('c')
congthuc.Layout picCanvas
congthuc.Draw picCanvas, 10, 10
End Sub
```

Chạy chương trình, nhấn nút Render:



Rõ ràng, việc khởi tạo một công thức đã trở nên sáng sủa, dễ đọc hơn nhiều. Sau này, khi bổ sung thêm một dạng công thức mới, ta sẽ định nghĩa hàm tạo đối tượng đó trong module này.
Lưu ý: Các tham số của hàm Row 3 hay Row5 có kiểu bất kỳ, miễn là kế thừa từ lớp Box. Thứ tự các tham số không quan trọng, chương trình có thể hiển thị một công thức không đúng ý nghĩa toán học. Bạn cứ thử xem. Khai báo lớp Fraction
Bổ sung một lớp mới, đổi tên thành Fraction. Mã của lớp như sau:

```
‘Class Module Fraction
Option Explicit
Implements Box
Private m_Width As Long
Private m_Height As Long
Private m_Ascent As Long
Private m_Numerator As Box
Private m_Denominator As Box
‘-----------------------------------------
Private Sub Class_Initialize()
Set m_Numerator = Nothing
Set m_Denominator = Nothing
End Sub
‘-------------------------------------------------
Private Sub Class_Terminate()
Set m_Numerator = Nothing
Set m_Denominator = Nothing
End Sub
‘-------------------------------------------------
Private Property Get Box_Ascent() As Long
Box_Ascent = m_Ascent
End Property
‘-------------------------------------------------
Private Property Get Box_ClassName() As String
```

```
Box_ClassName = 'fraction'
End Property
‘-------------------------------------------------
Private Sub Box_Draw(pic As PictureBox, ByVal x As Long, ByVal y As Long)
Dim nX As Long
Dim nY As Long
Dim ls As Long
ls = 3
nX = x + (m_Width - m_Numerator.Width)/2
nY = y
m_Numerator.Draw pic, nX, nY
nY = nY + m_Numerator.Height + (ls/3)* 2
pic.Line (x, nY)-(x + m_Width, nY)
nY = nY + ls / 3
nX = x + (m_Width - m_Denominator.Width) / 2
m_Denominator.Draw pic, nX, nY
End Sub
‘-------------------------------------------------

Public Property Get Numerator() As Box
Set Numerator = m_Numerator
End Property
‘-------------------------------------------------
Public Property Set Numerator(ByVal vNewValue As Box)
Set m_Numerator = vNewValue
End Property

Public Property Get Denominator() As Box
Set Denominator = m_Denominator
End Property
‘-------------------------------------------------
Public Property Set Denominator(ByVal vNewValue As Box)
Set m_Denominator = vNewValue
End Property
Private Property Get Box_Height() As Long
Box_Height = m_Height
End Property
‘-------------------------------------------------
Private Sub Box_Layout(pic As PictureBox)
Dim ls As Long
m_Numerator.Layout pic
```

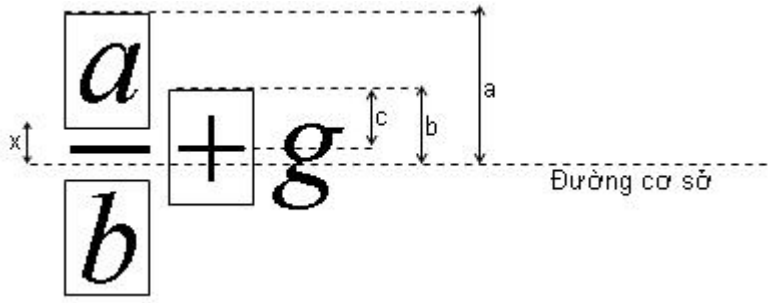
```
m_Denominator.Layout pic
ls = 3
If m_Numerator.Width > m_Denominator.Width Then
m_Width = m_Numerator.Width
Else
m_Width = m_Denominator.Width
End If
m_Height = m_Numerator.Height + m_Denominator.Height + ls
Dim hNewFont As Long
Dim hOldFont As Long
Dim hDC As Long
Dim lf As LOGFONT
Dim tm As TEXTMETRIC
hDC = pic.hDC
lf.lfCharSet = 2
lf.lfFaceName = 'Lucida Bright Math Symbol' & Chr(0)
lf.lfHeight = -MulDiv(12,GetDeviceCaps(hDC, LOGPIXELSY),72)
hNewFont = CreateFontIndirect(lf)
hOldFont = SelectObject(hDC, hNewFont)
GetTextMetrics hDC, tm
m_Ascent = m_Numerator.Height + (ls/3) * 2 + tm.tmAscent -
pic.TextHeight(Chr(&H21))2 SelectObject hDC, hOldFont
DeleteObject hNewFont
End Sub
‘--------------------------------------------------
Private Property Get Box_Width() As Long
Box_Width = m_Width
End Property
```

Tử số và mẫu số được lưu trong hai biến m_Numerator và m_Denominator. Hai thuộc tính tương ứng của lớp cho phép gán các giá trị đến hai biến này (Numerator, Denominator).

Phương thức Layout lần lượt gọi phương thức Layout của tử số và mẫu số. Biến ls (line spacing) lưu khoảng cách, nơi sẽ vẽ dòng kẻ ngang, giữa tử số và mẫu số (ls=3).

Độ rộng của phân số được xác định bằng độ rộng lớn nhất của tử số hoặc mẫu số. Chiều cao của phân số được xác định bằng chiều cao tử số cộng chiều cao mẫu số cộng chiều cao phần khoảng cách để vẽ đường nằm ngang (ls).

Mọi phức tạp nằm trong phần tính đường cơ sở của công thức. Để dễ quan sát, mời các bạn chú ý hình vẽ dưới đây:



Đường cơ sở hay ascent (a) cần tính bằng chiều cao của tử số cộng với x. Trong đó: x là khoảng cách từ tử số đến đường cơ sở. Khoảng cách phía trên dòng kẻ ngang ta chọn là 2 hay 2/3*ls. Vì vạch ngang của dấu + cần phải thẳng hàng với gạch ngang của phân số, do đó dựa vào hình vẽ ta có thể lập công thức tính x như sau:
x=2/3*ls+b-c
ls là khoảng cách từ tử số đến mẫu số
b là ascent của dấu +
c là ½ chiều cao của dấu +.
Ăp dụng công thức này vào chương trình ta sẽ có dòng mã:
x =(ls / 3) * 2 + tm.tmAscent - pic.TextHeight(Chr(&H21)) 2
Trước khi tính chiều cao cũng như ascent của dấu +, cần thiết lập lại các thuộc tính Font của Picture Box thành Lucida Bright Math Symbol. Các hàm CreateFontIndirect, SelectObject, DeleteObject…đều phục vụ quá trình này. Nếu bạn chưa hiểu về nó, nên tìm đọc thêm qua một số cuốn sách bất kỳ về lập trình trong Windows hay MSDN.
Mọi thao tác vẽ đều liên quan đến Device Context (DC). Khi cần thay đổi thuộc tính vẽ như Font chữ, nét vẽ…ta tạo các đối tượng tương ứng, tiếp theo chọn chúng cho DC bằng hàm SelectObject. Hàm SelectObject trả về thuộc tính vẽ hiện tại. Vẽ xong, cần khôi phục lại thuộc tính cũ của DC và xóa đối tượng vừa tạo trước đó bằng hàm DeleteObject.
Phương thức Draw hoạt động tương tự như lớp Row. Nó lần lượt gọi phương thức Draw của tử số, mẫu số, sau đó vẽ đường kẻ ngang bằng phương thức Line của đối tượng Picture Box.
Để có thể sử dụng được lớp Fraction, bạn phải khai báo thêm trong module API các cấu trúc và hàm sau:

```
Public Type LOGFONT
lfHeight As Long
lfWidth As Long
lfEscapement As Long
lfOrientation As Long
lfWeight As Long
lfItalic As Byte
```

```
lfUnderline As Byte
lfStrikeOut As Byte
lfCharSet As Byte
lfOutPrecision As Byte
lfClipPrecision As Byte
lfQuality As Byte
lfPitchAndFamily As Byte
lfFaceName As String * 32
End Type
Public Declare Function GetDeviceCaps…
Public Declare Function MulDiv…
Public Declare Function CreateFontIndirect…
Public Declare Function SelectObject…
Public Declare Function DeleteObject…
Public Const LOGPIXELSX = 88
Public Const LOGPIXELSY = 90
```

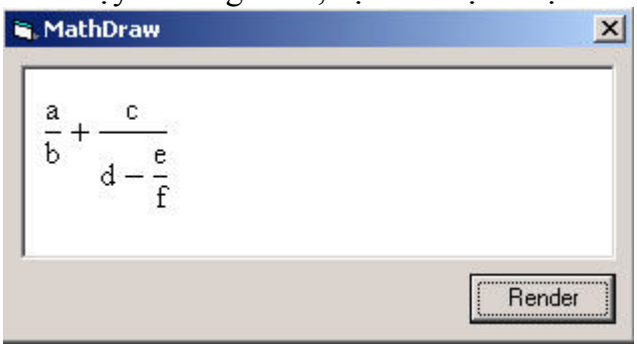
Tiếp theo, chèn thêm hàm Fraction vào module Helper:

```
Public Function Fraction(num As Box, den As Box) As Fraction
Dim f As New Fraction
Set f.Numerator = num
Set f.Denominator = den
Set Fraction = f
End Function
```

Sửa lại hàm xử lý sự kiến nhấn Render thành:

```
Private Sub cmdRender_Click()
Dim congthuc As Box
Set congthuc =
Row3(Fraction(Text('á),Text('b')),Operator('+'),Fraction(Text('c'),Row3(Text('d'),Op
erator('-'),Fraction(Text('é),Text('f')))))
congthuc.Layout picCanvas
congthuc.Draw picCanvas, 10, 10
End Sub
```

Khi chạy chương trình, bạn sẽ nhận được kết quả:



Theo cách đó, bạn có thể tạo một phân số có độ phức tạp, lồng nhau nhiều cấp bất kỳ…
Còn nữa

5.**Tính toán trước khi lập trình**
Tác giả: Nguyễn Duy Hàm

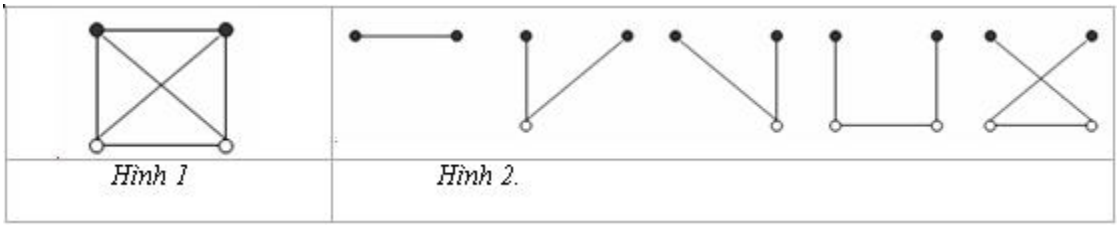
Trong một bài viết trên tạp chí Tin học và Nhà trường Thầy Nguyễn Xuân Huy đã nói về vấn đề đặt bút tính toán trước khi lập trình, Thầy đề cập đến bài toán đếm số ô vuông đơn vị bị cắt bởi 1 đường chéo của 1 hình chữ nhật có kích thước nxm. Qua đó chúng ta có thể phần nào hiểu được rằng việc tính toán, phân tích bài toán trước khi lập trình là vô cùng cần thiết, trong nhiều kì thi học sinh, sinh viên giỏi tin học có những bài toán tưởng chừng rất dễ, song nếu không được tiền xử lý tốt thì chỉ có thể vượt qua một vài test dễ ban đầu của BGK mà thôi. Lần này tôi cũng xin mạn phép được bàn về cùng vấn đề trong một bài toán khác, đó là một bài số 3 trong đề thi của khối chuyên tại kì thi OLIMPIC Tin học Sinh viên Việt Nam 2005 tại Đại học Khoa học Tự nhiên TP.Hồ Chí Minh vừa qua.

Bài toán

Một hệ thống giao thông gồm N nút (các nút được đánh số từ 1 đến N), trong đó bất kỳ hai nút nào cũng có đoạn đường hai chiều nối chúng. Ta gọi đường đi giữa hai nút là dãy các đoạn đường kế tiếp nhau, bắt đầu từ một nút và kết thúc tại nút kia, trong đó không có nút nào trên đường đi được lặp lại.

Yêu cầu: Cần đếm tất cả các đường đi khác nhau giữa hai nút bất kỳ của mạng giao thông đã cho.

Ví dụ: Với hệ thống giao thông 4 nút trong hình 1, ta có 5 đường đi nối giữa hai nút tô đen (xem hình 2).



Dữ liệu: Nhập từ file văn bản PCOUNT.INP gồm một số nguyên dương N (N

Kết quả: Ghi ra file PCOUNT.OUT gồm 1 dòng chứa số các đường đi khác nhau đếm được.

Ví dụ:

PCOUNT.INP	PCOUNT.OUT
4	5

Khi đọc bài toán này không ít bạn đã rất loay hoay, không biết phải làm thế nào, nhiều bạn đã sử dụng ngay đệ quy quay lui để duyệt tất cả các đường đi có thể từ đỉnh đầu đến đỉnh cuối trên 1 đồ thị ngầm định được xây dựng, vì cứ nói đến hệ thống giao thông là nhiều người nghĩ ngay tới cấu trúc đồ thị, với cách làm này chương trình không thể vượt qua test với n>15! (cần chú ý là bài toán này hạn chế thời gian chạy là 1 giây), nên việc duyệt như trên sẽ là một giải pháp khó có thể chấp nhận. Ta hãy cùng nhau phân tích bài toán

Cách giải quyết

Thuật toán:

Ta dễ thấy rằng: tổng các đường đi từ 2 đỉnh bất kì là tổng các đường đi có độ dài =1, các đường đi có độ dài =2, các đường đi có độ dài =3…,các đường đi có độ dài n-1. Cũng chính là tổng các đường đi không đi qua đỉnh trung gian nào, đi qua 1 đỉnh trung gian, qua 2 đỉnh trung gian,…qua n-2 đỉnh trung gian.

Tổng các đường đi không đi qua đỉnh trung gian nào là 1 (chính là đường nối trược tiếp 2 dỉnh đó). Tổng các đường đi đi qua 1 đỉnh trung gian là số cách chọn có thứ tự 1 đỉnh trong số n-2 đỉnh còn lại (A1n-2), tổng các đường đi qua 2 đỉnh trung gian là số cách chọn có thứ tự 2 đỉnh trong số n-2 đỉnh còn lại (A2n-2), tổng các đường đi qua 3 đỉnh trung gian là số cách chọn có tứ tự 3 đỉnh trên n-2 đỉnh còn lại (A3n-2).

Như vậy:

S=1+A1n-2 + A2n-2 + A3n-2+…+An-2n-2.(1)
(1)=1+(n-2)+(n-2)(n-3)+(n-2)(n-3)(n-4)+…+(n-2)!(2)
(2)=1+(n-2)[1+(n-3)[1+(n-4)[…[1+1]…]]].(3)
(3)=((..(1+1)2+1)3+1)4+1)5+1)…(n-3)+1)(n-2)+1(4)

Như vậy từ công thức (4) ta có thuật toán theo đoạn mã sau:

```
s:=1;
For i:=1 to n-2 do
Begin
s:=s*i;
s:=s+1;
End;
```

Cấu trúc dữ liệu:

Qua công thức trên ta thấy S là một số rất lớn (với n=1000 thì S có tới 2563 chữ số), không có kiểu số nào có thể lưu trữ được, ta phải sử dụng kiểu dữ liệu khác. Tốt nhất trong trường hợp này ta nên sử dụng mảng kiểu longint, với mỗi phần tử mảng sẽ lưu 1 số có 6 chữ số của chữ số S.

Để đơn giản trong xử lý ta nên lưu ngược, a[1] sẽ chứa 6 chữ số cuối cùng của S, a[2] sẽ chứa 6 chữ số tiếp theo…

Ví dụ s=1234031809283756.

Thì ta có mảng A như sau: a[1] = 283756; a[2]=031809; a[3]=001234;

Nhưng thực tế a[2] = 31809 và a[3]=1234(loại bỏ các số 0 ở đầu trái), vậy khi ghi kết quả ra file ta phải xử lý thêm chỗ này để đạt được kết quả đúng (thêm 0 vào trước cho đủ 6 chữ số rồi mới ghi ra file). Bây giờ vấn đề còn lại là lập trình. Chúng ta cùng theo dõi đoạn mã sau:

Chương trình

```
Const stout='PCOUNT.OUT';
stinp='PCOUNT.INP';
k=1000000;
d=6;
Var i,j,n,m,l,nho:longint;
s:string;
f:text;
ok:boolean;
a:array[1..500]of longint;//mảng chứa dãy số
Procedure Khoi_tao;
Begin
assign(f,stinp);
reset(f);
read(f,n);
```

```
close(f);
a[1]:=1;m:=1;
End;
Procedure Thuc_hien;
Begin
for i:=1 to n-2 do
begin
nho:=0;
for j:= 1 to m do
begin
nho:=nhơa[j]*i;
a[j]:=nho mod k;
nho:=nho div k ;
end;
if nho>0 then
begin
m:=m+1;
a[m]:=nho;
end;
nho:=1;ok:=true;j:=0;
while ok do
begin
j:=j+1;
nho:=a[j]+nho;
a[j]:=nho mod k;
nho:=nho div k;
if nho=0 then ok:=false;
end;
if j>m then m:=j;
end;
end;
Procedure In_ra;
Begin
assign(f,stout);
rewrite(f);
if n>1 then
begin
for i:=m downto 1 do
begin
str(a[i],s);
if i
```

```
begin
l:=length(s);
while l
begin
insert('0',s,1);
l:=l+1;
end;
end;
write(f,s);
end;
end
else write(f,’0’);
close(f);
End;
Begin
Khoi_tao;
Thuc_hien;
In_ra;
End.
```

Chú ý khi lập trình

- Không sử dụng các thủ tục inc(),dec() làm chậm chương trình.

- Trong chương trình có đoạn mã thêm số 0 vào trước các phần tử của mảng trước khi ghi lên file, không nên viết như sau mặc dù nó gọn hơn:

While length(s) < d do insert(‘0’,s,1); vì nó sẽ gọi hàm length(s) liên tục mỗi khi kiểm tra làm chậm chương trình của chúng ta.

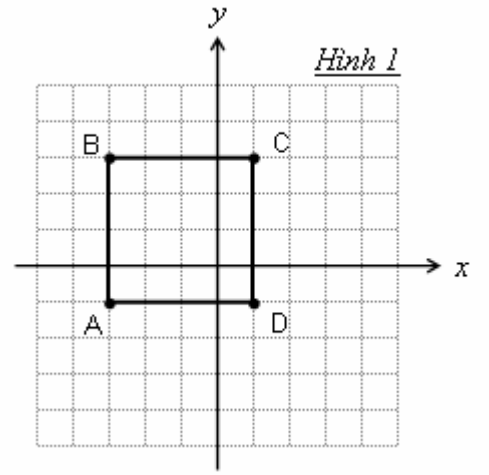
Kết luận

Cuối cùng ta thấy rằng bài toán này không phức tạp, không đòi hỏi phải tư duy thuật toán cao siêu, chỉ cần nắm vững kiến thức toán tổ hợp, biết cách xử lý trên số lớn, cùng với việc tính toán phân tích kĩ bài toán trước để tìm ra công thức đơn giản nhất, sao cho phép toán thực hiện là ít nhất sẽ đảm bảo chương trình chạy nhanh nhất!

Qua phần trình bày trên ta có thể thấy được sự liên quan mật thiết giữa toán học và tin học như thế nào, việc tính toán, phân tích kĩ bài toán, việc sử dụng thành thạo cấu trúc dữ liệu quan trọng như thế nào trong việc giải 1 bài toán trong tin học. Mọi góp ý, trao đổi xin gửi về duyhaman@yahoo.com. Xin cảm ơn!

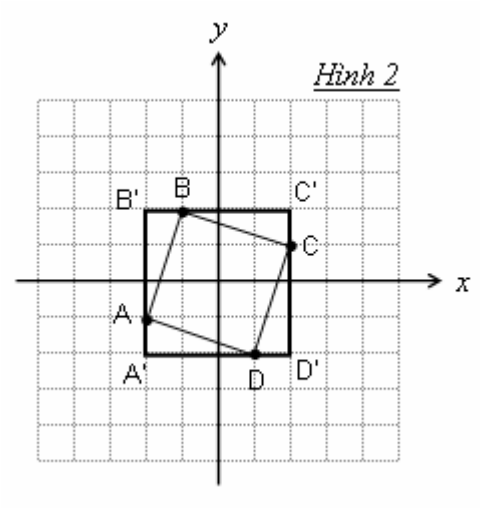
6.**Hình vuông cũng thú vị lắm chứ**
Tác giả: Bùi Tứ Quý

Hẳn các bạn đã biết đến hình vuông, một tứ giác đặc biệt có 4 cạnh bằng nhau và 4 góc bằng nhau (bằng 900), hình vuông là sự kết hợp hoàn hảo giữa hình chữ nhật và hình thoi. Trong Tin học, cũng có rất nhiều bài toán thú vị về hình vuông. Chúng ta hãy cùng tìm hiểu một trong số các bài toán ấy.
Trước tiên, chúng ta cùng xét bài toán:
Bài toán 1. Cho 4 điểm trên mặt phẳng. Hãy kiểm tra xem bốn điểm này có phải là đỉnh của một hình vuông có các cạnh song song với trục toạ độ hay không.
Nhận xét 1a. Hình vuông là hình chữ nhật có 4 cạnh bằng nhau. Nhận xét 1b. Nếu xA = xB, xC = xD, yA = yD và yB = yC thì ABCD là hình chữ nhật (có các cạnh song song với trục toạ độ).



Hình 1

Bước 1. Bạn sắp xếp toạ độ 4 điểm theo thứ tự tăng dần theo hoành độ (trong hình, thứ tự các điểm sau khi sắp xếp là A, B, D, C). Sau đó chúng ta kiểm tra nhận xét 1b. Nếu đúng chúng ta qua bước 2, nếu sai ta ghi kết quả 'SAI'. Bước 2. Tính độ dài AB, BD, CD, DA. Nếu AB = BC = CD = DA thì ta ghi 'DUNG', nếu sai ta ghi 'SAI'.
Như vậy chúng ta đã giải quyết xong bài toán 1. Tuy nhiên, nếu các cạnh hình vuông không song song với trục toạ độ thì bài toán 1 sẽ trở thành bài toán như sau:
Bài toán 2. Cho 4 điểm trên mặt phẳng. Hãy kiểm tra xem bốn điểm này có thể là đỉnh của 1 hình vuông hay không?
Input: File KTHV.INP gồm 4 dòng là toạ độ 4 điểm ban đầu. Output: File KTHV.OUT chứa số 1 (hoặc -1) nếu 4 điểm đó có thể (không thể) là 4 đỉnh một hình vuông.



Ví dụ:

KTHV.INP	KTHV.OUT
2.0 1.0 -2.0 -1.0 -1.0 2.0 1.0 -2.0	1

Nếu giải theo cách thông thường, ta không những kiểm tra độ dài 4 cạnh mà phải kiểm tra đến cả độ lớn 4 góc. Tôi đã nghĩ ra một giải thuật đơn giản hơn như sau: Chúng ta cũng sắp xếp 4 điểm ban đầu theo thứ tự tăng dần theo hoành độ, ta được 4 điểm đã sắp xếp theo thứ tự là A, B, D, C. Sau đó, chúng ta sẽ tạo ra 4 điểm mới là A', B', C', D' sao cho xA' = xB' = min (các hoành độ của A, B, C, D) ; xC' = xD' = max (các hoành độ của A, B, C, D); yA' = yD' = min (các tung độ của A, B, C, D); yB' = yC' = max (các tung độ của A, B, C, D).
Bây giờ chúng ta kiểm tra giá trị k = (A'B' = B'C' = C'D' = D'A') and (AB = BC = CD = DA). Nếu k = True thì ABCD là hình vuông, ngược lại thì ABCD không phải là hình vuông.
Sau đây là chương trình của bài 2.

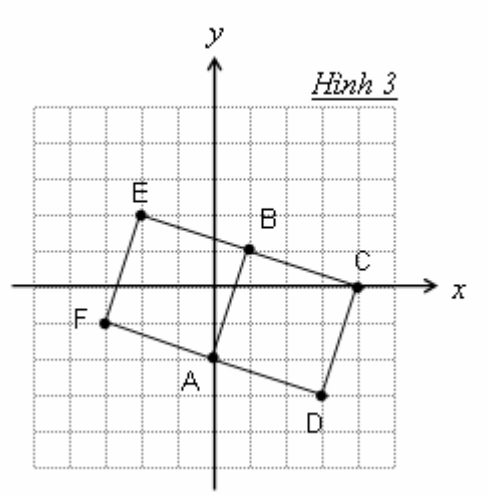
```
Program KiemTraHinhVuong;
Const fi = 'KTHV.INP';
fo = 'KTHV.OUT';
Type Point = Record
x,y:real;
End;
Var P : Array[1..8] of Point; {Các điểm A,B,D,C,A',B',C',D'}
A : Array[1..8] of Real; {AB,BC,CD,DA,A'B',B'C',C'D',D'A'}
i,j : byte;
```

```
f:text;

Function D(A,B:Point):real;
Begin
D:=sqrt(sqr(A.x-B.x)+sqr(A.y-B.y));
End;

Procedure DoiCho(Var A,B:Point);
Var k : Point;
Begin
k:=A; A:=B; B:=k;
End;

Procedure ReadInp;
Begin
Assign(f,fi); Reset(f);
For i:=1 to 4 do
Read(f,P[i].x,P[i].y);
Close(f);
End;
Procedure Solve;
Var minx,miny,maxx,maxy:real;
k:boolean;
Begin
For i:=1 to 4 do
For j:=i to 4 do
If P[i].x>P[j].x then DoiCho(P[i],P[j]);
minx:=P[1].x; miny:=P[1].y;
maxx:=P[1].x; maxy:=P[1].y;
For i:=2 to 4 do
Begin
If P[i].x
If P[i].y
If P[i].x>maxx then maxx:=P[i].x;
If P[i].y>maxy then maxy:=P[i].y;
End;
P[5].x:=minx; P[6].x:=minx;
P[7].x:=maxx; P[8].x:=maxx;
P[5].y:=miny; P[8].y:=miny;
P[6].y:=maxy; P[7].y:=maxy;
k:=false;
```

```
A[1]:=D(P[1],P[2]); A[2]:=D(P[2],P[4]);
A[3]:=D(P[4],P[3]); A[4]:=D(P[3],P[1]);
A[5]:=D(P[5],P[6]); A[6]:=D(P[6],P[7]);
A[7]:=D(P[7],P[8]); A[8]:=D(P[8],P[5]);
k:=(A[1]=A[2]) and (A[2]=A[3]) and (A[3]=A[4]);
k:= k and (A[5]=A[6]) and (A[6]=A[7]) and (A[7]=A[8]);
Assign(f,fo); Rewrite(f);
If k then Writeln(f,1) else Writeln(f,-1);
Close(f);
End;
Begin
ReadInp;
Solve;
End.
```

Bạn đọc có thể tự chứng minh thuật toán trên cho bài toán 2 là đúng. Nếu không thích, bạn cũng có thể kiểm tra góc vuông bằng cách kiểm tra 4 tam giác ABC, BCD, CDA, DAB có phải là 4 tam giác vuông lần lượt tại 4 đỉnh B, C, D, A hay không (bằng định lí Pythagore đảo)...

Bài toán 3. Cho hai điểm ĂxA, yA) và B(xB, yB) không trùng nhau. Hãy tìm toạ độ các điểm C, D, E, F sao cho ABCD và ABEF là các hình vuông.



Hình 3

Sau khi phân tích bài toán này, tôi được kết quả:
Nếu xA xB thì toạ độ 4 điểm C, D, E, F sẽ là:
xC = xB + (yB - yA) ; yC = yB – (xB – xA).
xD = xA + (yB - yA) ; yD = yA – (xB – xA).
xE = xB – (yB - yA) ; yE = yB + (xB – xA).
xF = xA – (yB - yA) ; yF = yA + (xB – xA).

Với xA > xB thì với tính toán như trên, ta sẽ được hai hình vuông ABDC, ABFE (không đúng với đề), do đó chúng ta chỉ cần đảo toạ độ hai điểm C và D cho nhau, E và F cho nhau.
Bài toán 3 là bài toán cho 2 đỉnh kề nhau, xác định 2 đỉnh còn lại của hình vuông. Sau đây là bài toán 4, cho 2 đỉnh đối nhau, xác định 2 đỉnh còn lại của hình vuông.
Bài toán 4. Cho hai điểm A (xA, yA) và C(xC, yC) là hai đỉnh đối nhau của một hình vuông. Hãy tìm toạ độ các điểm B và D sao cho ABCD là hình vuông.
Input : File HAIDINH.INP gồm 2 dòng là toạ độ 2 đỉnh A và C của hình vuông.
Output : File HAIDINH.OUT gồm 2 dòng là toạ độ 2 đỉnh B và D của hình vuông (mỗi toạ độ trong file output chính xác đến chữ số thứ 4 sau dấu phẩy).

HAIDINH.INP	HAIDINH.OUT
1.0 3.0	3.5000 4.5000
5.0 2.0	2.5000 0.5000

Ta có nhận xét: Hình vuông có hai đường chéo bằng nhau và vuông góc với nhau tại trung điểm mỗi đường. Ta cần tìm trung điểm của AC, đó là điểm O có toạ độ:

$$x_O = \frac{x_A + x_B}{2} \text{ và } y_O = \frac{y_A + y_B}{2}.$$

Sau đó, chúng ta phân tích tiếp, được kết quả:
xB = xO + (yA – yO) ; yB = yO + (xO – xA)
xD = xO – (yA – yO) ; yD = yO – (xO – xA)
Sau đây là chương trình của bài 4.

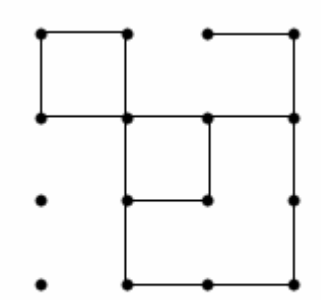
```
Program Tim2Dinh;
Const fi = 'HAIDINH.INP';
fo = 'HAIDINH.OUT';
Type Point = Record
x,y:real;
End;
Var P:Array[1..5] of Point;
f:text;

Procedure ReadInp;
Begin
Assign(f,fi); Reset(f);
Read(f,P[1].x,P[1].y);
Read(f,P[2].x,P[2].y);
```

```
Close(f);
End;

Procedure Solve;
Begin
P[5].x:=(P[1].x+P[2].x)/2;
P[5].y:=(P[1].y+P[2].y)/2;
P[3].x:=P[5].x+(P[1].y-P[5].y);
P[3].y:=P[5].y+(P[5].x-P[1].x);
P[4].x:=P[5].x-(P[1].y-P[5].y);
P[4].y:=P[5].y-(P[5].x-P[1].x);
End;

Procedure WriteOut;

Begin
Assign(f,fo); Rewrite(f);
Write(f,P[3].x:0:4,' ',P[3].y:0:4);
Writeln(f);
Write(f,P[4].x:0:4,' ',P[4].y:0:4);
Close(f);
End;
Begin
ReadInp;
Solve;
WriteOut;
End.
```

Chúng ta cùng xem sự phát triển của những bài toán về hình vuông:

Bài toán 5. Hình vuông (Đề thi Tin học trẻ không chuyên toàn quốc 2001 – bảng B). Cho một lưới N x N điểm gồm N dòng và N cột ($2 \le N \le 9$) là các điểm nút của một lưới ô vuông. Các dòng được đánh số từ trên xuống dưới, các cột được đánh số từ trái qua phải bắt đầu từ 1. Trên lưới điểm đó cho một số đoạn thẳng, mỗi đoạn nối một cặp điểm cạnh nhau trên cùng một dòng (đoạn ngang) hoặc trên cùng một cột (đoạn dọc). Cần phải đếm số các hình vuông với kích thước nhất định được tạo thành bởi các đoạn thẳng đã cho của lưới nêu trên.
Chẳng hạn ở hình 4, có 3 hình vuông: hai hình kích thước 1 và một hình kích thước 2 (kích thước của hình vuông là số các đoạn thẳng tạo thành 1 cạnh của hình vuông).
Yêu cầu: Hãy xác định số lượng các loại hình vuông và số hình vuông mỗi loại

trong lưới điểm đã cho (các hình vuông có cùng kích thước được xếp vào cùng một loại).



Hình 4

Input: SQUARE.INP
Dòng đầu tiên chứa 2 số nguyên N là số dòng (số cột) của lưới.
+ Dòng thứ hai chứa số nguyên M là số các đoạn thẳng được cho trên lưới.
+ M dòng tiếp theo, mỗi dòng biểu diễn một đoạn thẳng, có dạng:
- Hij chỉ một đoạn ngang trên dòng thứ i nối 2 điểm ở cột j và cột j+1
- hoặc Vij chỉ một đoạn dọc trên cột thứ i nối hai điểm ở dòng j và dòng j+1.
Output: SQUARE.OUT
+ Dòng đầu tiên ghi số nguyên P là số loại hình vuông có trên lưới.
+ P dòng tiếp theo, mỗi dòng ghi thông tin mô tả về một loại hình vuông và số lượng hình vuông loại đó, bao gồm hai số nguyên a và b cho biết có a hình vuông có cạnh độ dài b. Các thông tin về các loại hình vuông phải được đưa ra theo thứ tự tăng dần của độ dài cạnh.
+ Trong trường hợp không tìm được bất cứ một hình vuông nào thì ghi duy nhất một dòng thông báo: NO SQUARE.
Ví dụ: (Theo hình 4)

SQUARE.INP	SQUARE.OUT
4	2
16	2 1
H 1 1	1 2
H 1 3	
H 2 1	
H 2 2	
H 2 3	
H 3 2	
H 4 2	
H 4 3	
V 1 1	
V 2 1	
V 2 2	
V 2 3	
V 3 2	
V 4 1	
V 4 2	
V 4 3	

Ở bài này, chúng ta không chỉ kiểm tra hình vuông thông qua các đỉnh mà còn phải kiểm tra các đoạn nối giữa chúng nữa.
Việc đầu tiên là tổ chức dữ liệu như thế nào cho hợp lí. Chúng ta sẽ đặt toạ độ nút ở hàng i, cột j là (i, j) với $1 \le i, j \le N$. Như vậy nút ở góc trên trái có toạ độ (1, 1), nút góc dưới, phải sẽ là (N, N). Ta sẽ sử dụng 1 mảng C : Array[1..9,1..9,1..2] of boolean để lưu các cạnh, trong đó:
+ C[i,j,1] = True / False nếu có / không có đoạn nối ngang giữa nút (i, j) và nút (i, j+1).
+ C[i,j,2] = True / False nếu có / không có đoạn nối dọc giữa nút (i, j) và nút (i+1, j).
Hiển nhiên, C[i,N,1] = False với mọi i và C[N,i,2] = False với mọi i.
Ta sử dụng mảng HV[1..9] để chỉ số hình vuông mà cạnh có độ dài là 1 đoạn, 2 đoạn, …, N đoạn.
Một "ý tưởng táo bạo" là vét cạn toàn bộ để tìm ra tất cả các bộ 4 điểm rồi kiểm tra. Tuy nhiên, ý tưởng này không được khả thi cho lắm. Chúng ta có thể đi theo giải pháp sau:
Tại mỗi nút (i,j) với i < N và j < N, ta sẽ kiểm tra nó có phải là đỉnh trái trên của những hình vuông mà cạnh có độ dài là 1 đoạn, 2 đoạn,..., N-max(i,j) đoạn hay không:

```
For i:=1 to N do
For j:=1 to N do
Begin
max:=i;
If max
For d:=1 to N-max do
```

```
If KT(i,j,d) then Inc(HV[d]);
End;
```
Kèm theo đó, chúng ta phải có hàm KT(i,j,d) trả về giá trị Boolean để kiểm tra xem nút (i,j) có thể là đỉnh trên trái của hình vuông mà cạnh có độ dài d đoạn hay không.
Như vậy, chương trình hoàn chỉnh của bài 5 là:
```
Program Square;
Const fi = 'SQUARE.INP';
fo = 'SQUARE.OUT';
Var i,j,d,max,N,M,P:byte;
HV : Array[1..9] of integer;
C : Array[1..9,1..9,1..2] of boolean;
f:text;

Procedure ReadInp;
Var a,b:byte; ch:char;
Begin
Fillchar(C,sizeof(C),false);
Fillchar(HV,sizeof(HV),0);
P:=0;
Assign(f,fi); Reset(f);
Readln(f,N);
Readln(f,M);
For i:=1 to M do
Begin
Readln(f,ch,a,b);
If ch='H' then C[a,b,1]:=True;
If ch='V' then C[b,a,2]:=True;
End;
Close(f);
End;

Function KT(i,j,d:byte):boolean;
Label Ra;
Var a:byte; k:boolean;
Begin
k:=true;
For a:=i to i+d-1 do {Kiểm tra những đoạn dọc}
Begin
If C[a,j,2] = false then
Begin
k:=false; Goto Ra;
```

```
End;
If C[a,j+d,2] = false then
Begin
k:=false; Goto Ra;
End;
End;
For a:=j to j+d-1 do {Kiểm tra những đoạn ngang}
Begin
If C[i,a,1] = false then
Begin
k:=false; Goto Ra;
End;
If C[i+d,a,1] = false then
Begin
k:=false; Goto Ra;
End;
End;
Ra : KT:=k;
End;

Procedure Solve;
Begin
For i:=1 to N do
For j:=1 to N do
Begin
max:=i;
If max For d:=1 to N-max do
If KT(i,j,d) then Inc(HV[d]);
End;
End;

Procedure WriteOut;
Begin
For i:=1 to N do
If HV[i]>0 then inc(P);
Assign(f,fo); Rewrite(f);
Writeln(f,P);
For i:=1 to N do
If HV[i]>0 then Writeln(f,HV[i],' ',i);
Close(f);
End;
```

Begin
ReadInp; Solve; WriteOut;
End.
Sau đây là hai bài tập dành cho bạn tự luyện tập:

Bài 6. Hình vuông (Đề thi tuyển sinh lớp 10 chuyên Tin – Năm học 2003 – 2004 - PTNK).
Trên mặt phẳng, cho N hình vuông có cạnh song song với trục toạ độ được đánh số từ 1 đến N ($1 \le N \le 2000$). Hình vuông thứ i được cho bởi toạ độ góc dưới trái (xi, yi) và toạ độ đỉnh trên phải (zi, ti). Toạ độ các đỉnh là số nguyên trong phạm vi từ –10000 đến 10000. Khoảng cách giữa hai hình vuông A và B được định nghĩa là độ dài của đoạn thẳng ngắn nhất trong số các đoạn thẳng mà một đầu mút thuộc hình vuông A còn đầu mút kia thuộc hình vuông B.
Yêu cầu: Tìm 2 hình vuông xa nhau nhất trong N hình vuông cho trước.
Input: Vào từ file văn bản SQUARE.INP trong đó dòng đầu chứa số N, dòng thứ i trong N dòng tiếp theo chứa 4 số xi, yi, zi, ti.
Output: Ghi ra file văn bản SQUARE.OUT trong đó chứa chỉ số của 2 hình vuông tìm được.
Ví dụ:

SQUARE.INP	SQUARE.OUT
3 1 1 3 3 2 2 5 5 7 1 8 2	1 3

Bài 7. Lưới ô vuông (Đề thi Tin học trẻ không chuyên Toàn quốc 2002 - Bảng C)
Cho một lưới ô vuông có kích thước m dòng, n cột ($m, n \le 100$). Các dòng của lưới được đánh số từ 1 đến m từ trên xuống dưới, các cột của lưới được đánh số từ 1 đến n từ trái sang phải. Ô nằm trên dòng i, cột j là ô (i,j) của lưới và (i,j) gọi là toạ độ của ô này.
Một nhóm gồm p học sinh tổ chức trò chơi đánh dấu như sau : Mỗi học sinh đánh dấu các ô của một hình chữ nhật trên lưới. Các hình chữ nhật này có thể giao nhau.
Một tập S các ô của lưới ô vuông đã cho gọi là miền liên thông bậc k nếu thoả 2 điều kiện:
+ Mỗi ô thuộc S có số lượng học sinh đánh dấu nó đúng bằng k.
+ Hoặc S có đúng 1 ô, hoặc nếu S có nhiều hơn 1 ô thì đối với 2 ô bất kì của S luôn có cách di chuyển qua các ô chung cạnh thuộc S để từ ô này đến ô kia.
Yêu cầu: Hãy đếm số miền liên thông bậc $k = 1, 2, \ldots, p$ ($k \le p \le 20$)
Input: File SQNET.INP: Dòng đầu tiên chứa 2 số nguyên m, n là kích thước của lưới. Dòng thứ hai chứa p là số học sinh. Dòng thứ i trong số p dòng tiếp theo chứa thông tin về hình chữ nhật mà học sinh thứ i chọn, gồm 4 số nguyên dương x, y, u,

v, trong đó (x,y) là toạ độ ô góc trên trái và (u,v) là toạ độ ô góc dưới phải của hình chữ nhật.
Output: Ghi ra file SQNET.OUT, gồm p dòng, dòng thứ i chứa số lượng miền liên thông bậc i.
Ví dụ:

SQNET.INP	SQNET.OUT
100 100 3 10 10 30 30 20 20 50 50 40 40 60 60	3 2 0

SQNET.INP	SQNET.OUT
50 100 4 10 20 30 30 20 10 30 30 20 20 40 30 20 20 30 40	4 0 0 1

Mọi trao đổi góp ý, bạn có thể liên hệ với tôi tại địa chỉ email: qbuitu@yahoo.com.
Chân thành cám ơn!

7.Tìm bao lồi của một tập hợp điểm bằng phương pháp quét Graham

Tác giả: Tạ Tiến Đạt

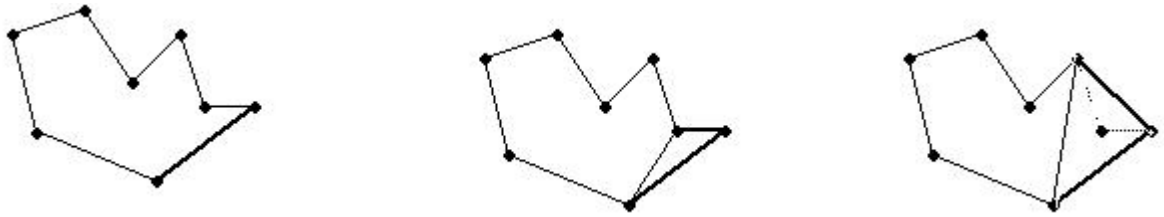
Các bạn thân mến!
Khi học toán chắc chắn bạn đã học môn hình học, nhưng hình học trong thực tế và trong tin học cũng có nhiều điểm khác nhau. Ví dụ như khi học hình bạn có thể nói 'nhìn hình vẽ ta có' nhưng đối với một bài toán tin thì không thể làm như vậy, có đúng không nào? Trong bài viết này tôi sẽ trình bày với các bạn một phương pháp để tìm bao lồi của một tập hợp điểm bằng phương pháp quét Graham − một bài toán khá hay.
Trước tiên chúng ta sẽ định nghĩa thế nào là một hình lồi ?
- Một hình được gọi là hình lồi khi và chỉ khi với hai điểm A và B thuộc nó thì đường thẳng AB cũng hoàn toàn nằm trong nó.
- Định nghĩa này không bị hạn chế đối với các hình không phải đa giác như hình tròn, ellipse.. như định nghĩa: 'Một đa giác là đa giác lồi khi nó nằm về một phía của đường thẳng đi qua cạnh bất kì'

Bây giờ chúng ta sẽ trở lại bài toán tìm bao lồi đã nói ở trên. Thứ nhất là về yêu cầu của bài toán: Chúng ta sẽ có đầu vào là một tập hợp điểm trên mặt phẳng và nhiệm vụ đặt ra là tìm một đa giác lồi có đỉnh là các điểm thuộc tập điểm đã cho và chứa toàn bộ các đỉnh của tập điểm bên trong nó.

Sau đó ta sẽ suy nghĩ về hướng giải của bài toán:
- Trước tiên là một cách giải rất 'con người': ta tìm một điểm có tung độ nhỏ nhất và hoành độ lớn nhất trong tập các điểm làm điểm chốt, sau đó thực hiện phép quay

ngược chiều kim đồng hồ, gặp điểm đầu tiên thì lại lấy điểm đó làm chốt và thực hiện phép quay như trên cho tới khi trở về điểm chốt ban đầu thì tập các điểm 'chốt' đó chính là bao lồi mà ta cần tìm. ứng với cách suy nghĩ này chúng ta có thuật toán bọc gói (Wrap) để tìm bao lồi của tập điểm.
- Nhưng cũng có một cách suy nghĩ khác đưa chúng ta đến với một giải thuật khác:

Chọn điểm chốt có hoành độ x lớn nhất trong các điểm có tung độ y nhỏ nhất (khi hiểu rõ thuật toán các bạn sẽ biết được nguyên nhân). Chuyển điểm chốt về vị trí 1. Ta sắp xếp các điểm theo khoá là góc tạo bởi điểm đó và điểm chốt với trục hoành theo thứ tự tăng dần. Khi đi theo thứ tự p[1], p[2],.. p[N], p[1] ta thu được một đa giác khép kín đơn.
Ta đi vòng qua đa giác này, thử đặt một điểm vào bao và kiểm tra xem các điểm trước đó có còn nằm trên bao hay không. Nếu không ta chỉ việc loại điểm đó ra khỏi bao thôi.
Việc kiểm ra một điểm có còn nằm trên bao hay không có thể làm như sau: khi cho một điểm mới vào bao, ta sẽ lần ngược lại những điểm đã nằm trong bao. Trong quá trình, nếu gặp một điểm là khúc rẽ phải thì điểm này sẽ không thuộc bao nữa, ta loại nó luôn. Quá trình kết thúc khi ta gặp một điểm là khúc rẽ trái, vì tất cả các điểm từ đó lùi về 1 chắc chắn sẽ thuộc bao.



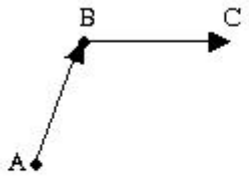
Các bước của thuật toán.
Khi đã có giải thuật trong tay, chúng ta sẽ nghĩ tới việc biểu diễn dữ liệu để đưa ra lời giải cuối cùng. Đối với một điểm, có một cách biểu diễn đơn giản như một record: TPoint = record x, y: Real; end;
và tập các điểm đã cho sẽ là một mảng [0..N] phần tử có kiểu TPoint. Kế đó chúng ta sẽ nghĩ tới cách sắp xếp các điểm từ 1..N dựa vào giá trị khóa là hệ số góc của đường thẳng nối điểm đó và điểm chốt.
Mặc dù có thể sử dụng hàm arctg để tính hệ số góc của đường thẳng (argtg(dy/dx)) tuy nhiên để tránh việc dx = 0 ta có thể sử dụng một hàm Theta có ý nghĩa tương tự như sau:

```
function Thetăp2, p1: TPoint): Real;
var
t: Real;
dx, dy, ax, ay: Real;
begin
```

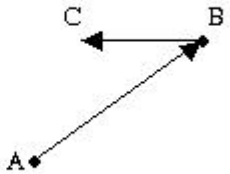
```
dx := p2.x - p1.x; ax := Abs(dx);
dy := p2.y - p1.y; ay := Abs(dy);
if (dx = 0) and (dy = 0) then t := 0
else
t := dy / (ax + ay);
if dx < 0 then t := 2 - t
else
if dy < 0 then t := 4 + t;
Theta := t * 90.0;
end;
```

Sau đó ta thực hiện việc sắp xếp các điểm theo giá trị tăng dần của hàm Theta (px, pchốt) (x = 1..N). (Lưu ý trước khi sắp xếp theo theta, ta thực hiện việc sắp các điểm theo thứ tự tăng dần của tung độ và hoành độ của các điểm - và phải sử dụng một thuật toán sắp xếp ổn định − ví dụ như Bubble Sort, Shell Sort.. các bạn thử suy nghĩ xem tại sao?).
Một điều nữa mà các bạn cần lưu ý là khi so sánh hai số thực a, b ta không thể viết a = b mà phải viết Abs(a − b) < epsilon trong đó epsilon là một giá trị dương nào đó đủ nhỏ mà ta quy định trước. (mặc dù trong bài toán này thì đó là một việc làm không cần thiết).

Để xét xem có loại một điểm ra khỏi bao hay không, với các điểm p[1..M] đã thuộc bao và điểm đang xét p[i] ta sử dụng hàm Ccw(p[m − 1], p[m], p[i]: TPoint). ở đây hàm Ccw của chúng ta khá đơn giản, giá trị trả về là 1 nếu đường đi ABC là rẽ phải, −1 nếu rẽ trái và 0 nếu 3 điểm thẳng hàng.



Rẽ phải



Rẽ trái

```
function Ccw(p0, p1, p2: TPoint): Integer;
var
a, b, c, d, dx1, dx2, dy1, dy2: Real;
begin
dx1 := p1.x - p0.x; dy1 := p1.y - p0.y;
dx2 := p2.x - p0.x; dy2 := p2.y - p0.y;
a := dx1 * dy2; b := dx2 * dy1;
if a > b then Ccw := -1;
if a < b then Ccw := 1;
```

```
if Abs(a - b) < epsilon then Ccw := 0;
end;
```

Việc quét tập hợp các diểm để tìm ra bao lồi được thực hiện bởi thủ tục GrahamScan. Kết thúc thủ tục này các điểm trong bao lồi theo thứ tự là các điểm p[1..M].
Demo là thủ tục thực hiện vẽ bao lồi và tập hợp điểm lên màn hình.

Sau đây là chương trình mẫu của thuật toán (có thể vẫn còn một số lỗi, mong các bạn góp ý để chương trình chạy nhanh hơn, bạn cũng có thể dùng Free Pascal để tăng giá trị max (nmax)):

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N+,O-,P-,Q-,R+,S+,T-,V+,X+,Y+}
{$M 16384,0,655360}
program Scan_by_Graham;
uses graph, crt;
const
max = 2000;
epsilon = 0.00001;
type
TPoint = record x, y: Real; end;
var
m, n: Integer;
p: array[0..max] of TPoint;
tt: array[1..max] of Real;

procedure Enter;
var
f: Text;
i: Integer;
begin
Assign(f, 'POINT.INP'); Reset(f);
Readln(f, n);
for i := 1 to n do Readln(f, p[i].x, p[i].y);
Close(f);
end;

procedure Swap(var p1, p2: TPoint);
var
temp: TPoint;
begin
```

```
temp := p1; p1 := p2; p2 := temp;
end;

procedure SortByXandY;
var
i, j: Integer;
begin
for i := 2 to n do
for j := n downto i do
if (p[j].y < p[j - 1].y) then Swap(p[j], p[j - 1]);
for i := 2 to n do
for j := n downto i do
if (p[j].x < p[j - 1].x) then Swap(p[j], p[j - 1]);
end;

function Ccw(p0, p1, p2: TPoint): Integer;
var
dx1, dy1, dx2, dy2, a, b: Real;
begin
dx1 := p1.x - p0.x; dy1 := p1.y - p0.y;
dx2 := p2.x - p0.x; dy2 := p2.y - p0.y;
a := dx1 * dy2;
b := dx2 * dy1;
if a > b then Ccw := -1;
if a < b then Ccw := 1;
if Abs(a - b) < epsilon then Ccw := 0;
end;

function Theta (p1, p0: TPoint): Real;
var
dx, dy, t: Real;
begin
dx := p1.x - p0.x;
dy := p1.y - p0.y;
if (dx = 0) and (dy = 0) then t := 0
else
t := dy / (Abs(dx) + Abs(dy));
if dx < 0 then t := 2 - t
else
if dy < 0 then t := 4 + t;
Theta := t * 90.0;
```

```
end;

procedure SortByTheta;
var
temp: Real;
i, j: Integer;
begin
for i := 2 to n do
for j := n downto i do
if (tt[j] < tt[j - 1]) then
begin
temp := tt[j]; tt[j] := tt[j - 1]; tt[j - 1] := temp;
Swap(p[j], p[j - 1]);
end;
end;

procedure Init;
var
min, i: Integer;
begin
SortByXandY;
min := 1;
for i := 2 to n do
if p[i].y < p[min].y then min := i;
for i := 1 to n do
if (p[i].y = p[min].y) and (p[i].x > p[min].x) then min := i;
Swap(p[min], p[1]);
for i := 1 to n do tt[i] := Thetăp[i], p[1]);
SortByTheta;
p[0] := p[n];
end;

procedure GrahamScan;
var
i: Integer;
begin
m := 3;
for i := 4 to n do
begin
while Ccw(p[m - 1], p[m], p[i]) = 1 do Dec(m);
m := m + 1;
```

```
Swap(p[i], p[m]);
end;
end;

procedure Demo;
var
td: array[1..max] of record x, y: Integer; end ;

cc, gd, gm, ge, i: Integer;
cmin, cmax, dmin, dmax, dx, dy: Integer;
pp, qq, xmin, ymin, xmax, ymax: Real;

procedure Oxy;
var
i: Integer;
begin
xmax := 2.9e-39; ymax := xmax;
xmin := 1.7e38; ymin := xmin;
for i := 1 to n do
begin
if p[i].x < xmin then xmin := p[i].x;
if p[i].x > xmax then xmax := p[i].x;
if p[i].y < ymin then ymin := p[i].y;
if p[i].y > ymax then ymax := p[i].y;
end;
if Abs(xmin) > Abs(xmax) then xmax := -xmin
else xmin := -xmax;
if Abs(ymin) > Abs(ymax) then ymax := -ymin
else ymin := -ymax;
cmin := 4; cmax := 630;
dmin := 4; dmax := 460;

pp := (cmax - cmin) / (xmax - xmin);
qq := (dmin - dmax) / (ymax - ymin);
dx := Round(pp * (-xmin)) + cmin;
dy := Round(qq * (-ymin)) + dmax;

Line(dx, dmin, dx, dmax);
Line(cmin, dy, cmax, dy);
Line(dx + 3, dmin + 5, dx, dmin);
Line(dx - 3, dmin + 5, dx, dmin);
```

```
Line(cmax, dy, cmax - 3, dy - 3);
Line(cmax, dy, cmax - 3, dy + 3);
end;

var
s1, s2, mstr: string;
begin
gd := 0; gm := 0; ge := m;
InitGraph(gd, gm, 'F:BPBGI');
Oxy;
for i := 1 to n do
begin
td[i].x := Round(pp * (p[i].x - xmin) + cmin);
td[i].y := Round(qq * (p[i].y - ymin) + dmax);
cc := GetColor;
SetColor(Red);
FillEllipse(td[i].x, td[i].y, 3, 3);
SetColor(cc);
if (i > 1) and (i <= ge) then
begin
cc := GetColor;
SetColor(White);
Line(td[i - 1].x, td[i - 1].y, td[i].x, td[i].y);
if i = ge then
Line(td[1].x, td[1].y, td[ge].x, td[ge].y);
SetColor(cc);
end;
end;
Readln;
CloseGraph;
end;

begin
Enter;
Init;
GrahamScan;
Demo;
end.
```

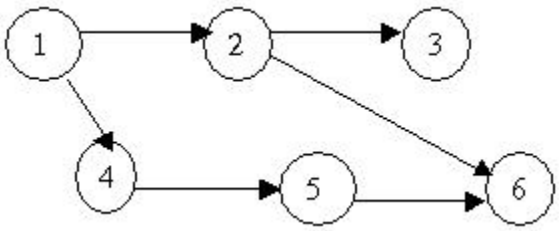
Nếu các bạn cảm thấy khi cài đặt mà không cần sử dụng thủ tục SortByXandY hay không dùng thuật toán sắp xếp ổn định mà vẫn chạy đúng thì hãy liên hệ với tôi, tôi

sẽ gửi cho bạn một vài test mà tôi đã thử qua và cho kết quả sai khi dùng các thuật toán sắp xếp khác như QuickSort).

8.Sắp xếp Topo trong đồ thị có hướng và một số ứng dụng

Tác giả: Nguyễn Xuân Sơn

I. Định nghĩa:
Cho G = <V,E> là một đồ thị có hướng. Sắp xếp topo các đỉnh V là quá trình gán một trật tự tuyến tính tới các đỉnh của G sao cho nếu có một cung (i,j) trong E thì i phải đứng trước j trong trật tự đó.
Nói cách khác: một trật tự topo của các đỉnh của một đồ thị có hướng G = <V,E> là một dãy (v1,v2,..,vn) sao cho V = {v1,v2,..,vn} và với mọi (vi,vj) trong E, vi đứng trước vj trong dãy.
Ví dụ: Đồ thị



Có 8 trật tự topo, đó là: 1 2 4 3 5 6
1 2 4 5 3 6
1 2 4 5 6 3
1 4 2 5 3 6
1 4 2 5 6 3
1 4 2 3 5 6
1 4 5 2 3 6
1 4 5 2 6 3

Dễ thấy: Nếu một đồ thị có hướng có chu trình, thì không thể có một trật tự topo của các đỉnh. Chúng ta có thể đặt câu hỏi ngược lại: Nếu đồ thị có hướng không có chu trình thì có luôn tồn tại một trật tự topo nào đó của các đỉnh hay không.
Trước tiên ta cùng xem xét định lý sau:
Định lý 1: Trong một đồ thị có hướng không chu trình với số đỉnh lớn hơn 1, tồn tại một đỉnh có bậc vào bằng 0.
Bậc vào của một đỉnh trong đồ thị có hướng được định nghĩa như sau: Đó là số cung có điểm cuối là đỉnh đó.
Chứng minh:

Phản chứng. Giả sử mọi đỉnh của G có bậc vào ≥ 1. Ta sẽ chứng minh G có chu trình.
Bắt đầu từ một đỉnh v1 nào đó, lần ngược lại một cung đi tới v1 tới một đỉnh v2, từ v2 lần ngược lại v3 và cứ tiếp tục như vậy. Do tất cả các đỉnh có bậc ≥ 1, quá trình này sẽ không bao giờ dừng. Nhưng G là hữu hạn nên chắc chắn một đỉnh nào đó phải được gặp lại trong quá trình lần ngược này và vì thế G có một chu trình.
Định lý trên dẫn tới một giải thuật hiệu quả giải quyết bài toán sắp xếp topo như sau:
Thuật toán:
Gọi S là tập chứa các đỉnh có bậc vào (trong quá trình tìm) bằng 0. Khởi tạo S là tập rỗng
Bước 1: Thêm vào S các đỉnh có bậc vào bằng 0
Bước 2: Xoá một đỉnh nào đó trong S và các cung đi ra từ nó
Bước 3: Lặp lại bước 1 cho tới khi mọi đỉnh của đồ thị đều bị xoá
Trật tự các đỉnh bị xoá chính là trật tự topo cần tìm.
Chứng minh tính đúng của thuật toán:
Giả sử tại thời điểm bắt đầu bước lặp thứ k ta đang có dãy Sk = (v1,v2,..,vk-1) và đồ thị là Gk. Ta chứng minh mệnh đề sau: Tại thời điểm bắt đầu bước lặp k của thuật toán
Sk = (v1,v2,..,vk-1), với mọi cạnh (vi,vj) trong E, nếu vj trong Sk thì vi trước vj trong Sk.
Quy nạp theo k: k=1 Hiển nhiên đúng
Giả sử mệnh đề đúng với k, ta chứng minh nó đúng với k+1 tức là Sk+1 = (v1,v2,..,vk), với mọi (vi,vk) trong E thì vi trong Sk.
Theo định lý 1: vk tồn tại (áp dụng định lý 1 với đồ thị Gk). Do vk có bậc bằng 0 trong Gk nên trong đồ thị ban đầu G phải xảy ra trường hợp với mọi cung (i,vk) thì i phải trong {v1,v2,..,vk-1} nghĩa là với mọi (vi,vk) trong E, vi đứng trước vk trong Sk+1.
Mệnh đề đúng với k+1
II. Một số yêu cầu liên quan
Bài toán 1: Tìm tất cả các trật tự topo của một đồ thị định hướng.
Tại mỗi bước chọn đỉnh để xoá ta có thể xoá tuỳ ý một đỉnh nào đó trong tập các đỉnh có bậc vào bằng 0. Nếu tại mỗi bước ta duyệt tất cả các cách xoá có thể thì sẽ thu được tất cả các trật tự topo cần tìm.
Bài toán 2: Tìm 2 trật tự topo nào đó (Nếu có).
Có thể tìm theo cách giải bài toán 1 và sau khi tìm được 2 trật tự topo thì dừng thuật toán. Tuy nhiên có thể tìm cách khác nhanh hơn bằng nhận xét sau:
Giả sử (v1,v2,..,vn) là một trật tự topo của G. Nếu tồn tại chỉ số i (i=1..n-1) sao cho (vi,vi+1) không là một cung trong E thì đổi chỗ vi và vi+1 trong dãy (v1,v2,..,vn) ta được một trật tự topo mới, nghĩa là
(v1,v2,..,vi-1,vi+1,vi,vi+2,..,vn) cũng là một trật tự topo.
Từ nhận xét trên ta có thuật toán cho bài toán 2 như sau: Sau khi tìm một trật tự

topo, giả sử là (v1,v2,..,vn) ta tìm chỉ số i (i=1..n-1) sao cho (vi,vi+1) không là một cung trong E. Nếu không tìm được chỉ số i như vậy, có nghĩa là với mọi i (i=1..n-1), (vi,vi+1) là một cung trong E. Khi đó (v1,v2,..,vn) là trật tự duy nhất, và không thể có một trật tự topo nào khác. Nếu có một chỉ số i thỏa mãn, ta đổi chỗ vi và vi+1 và thu được một trật tự topo thứ hai cần tìm.

III. ứ ng dụng

Bài toán 1:

1. Phát biểu bài toán :

Chúng ta có một dự án lớn. Dự án được chia thành các dự án nhỏ hơn, ta gọi là các dự án con. Mỗi dự án con cần một khoảng thời gian nào đó để hoàn thành, đồng thời mỗi dự án con không thể bắt đầu thực hiện cho đến khi một số dự án con nào đó được hoàn thành. Câu hỏi đặt ra là: Thời gian sớm nhất để hoàn thành dự án lớn. Giả sử thời điểm bắt đầu dự án lớn là 0.

2. Phân tích :

Bài toán có thể mô hình hoá dưới dạng đồ thị như sau:

Xây dựng đồ thị G = <V,E> với mỗi đỉnh của đồ thị đại diện cho một trạng thái nào đó của dự án lớn. Mỗi cung (i,j) biểu diễn một dự án con. Mỗi cung được gán một trọng số biểu diễn thời gian cần để hoàn thành dự án con đó. Với (i,j) thuộc E ta nói i là đỉnh liền trước j trong G và ngược lại, j là đỉnh liền sau i trong G. Đồ thị có hai đỉnh đặc biệt: Đỉnh chỉ có cung đi ra thể hiện trạng thái ban đầu của dự án lớn, khi chưa có công việc nào được thực hiện, ta gọi đó là đỉnh bắt đầu. Đỉnh chỉ có cung đi vào, thể hiện trạng thái kết thúc của dự án lớn, khi mọi công việc đã hoàn thành, ta gọi đó là đỉnh kết thúc.

Gọi a là đỉnh biểu diễn trạng thái ban đầu. Với mọi đỉnh v (trạng thái v của dự án lớn) của G, định nghĩa d(a,v) là thời gian sớm nhất có thể để đạt được trạng thái v. Một trạng thái w được đạt tới trong thời gian sớm nhất có thể nếu với mọi cung (i,w) thuộc E: i được đạt tới trong thời gian sớm nhất có thể và dự án con biểu diễn bởi cung (i,w) được thực hiện ngay sau đó. Tất cả các trạng thái i đó phải được đạt tới trước khi w đạt tới. Ta suy ra phương trình sau:

d(a,a) = 0

d(a,w) = max{d(a,v) + c[v,w]}

v thuộc P(w)

ở đây P(w) = {i / (i,w) thuộc E}

Yêu cầu của bài toán: Tính d(a,b) với b là đỉnh kết thúc.

Để tính d(a,w) với mọi w thuộc G, ta phải duyệt các đỉnh của G theo một trật tự topo nào đó, cách tính này đảm bảo d(a,w) được tính sau khi tất cả d(a,v) được tính, ở đây v là đỉnh liền trước w trong G. Để có thể tìm lại con đường từ trạng thái bắt đầu đến trạng thái kết thúc có tổng thời gian bằng d(a,b), tại mỗi bước ta ghi lại thông tin v là đỉnh trước w mà d(a,w) = d(a,v) + c[v,w].

Đến đây chúng ta có thể dễ dàng cài đặt bài toán trên.

3. Cài đặt :
Dữ liệu vào: Cho trong file time.inp với cấu trúc như sau:
- Dòng đầu tiên là hai số n,m: n là số các trạng thái của dự án lớn, m là số các dự án con. Giả sử các trạng thái và các dự án con được đánh số là các số tự nhiên liên tiếp bắt đầu từ 1.
- m dòng tiếp theo, dòng thứ i là 3 số x,y,z với ý nghĩa như sau: Dự án con i thực hiện từ trạng thái x sang trạng thái y mất khoảng thời gian là z.
Dữ liệu ra: Trong file time.out với một số duy nhất là thời gian cần tìm.
Trong cài đặt của tôi, các mảng c,a,b,bac dùng để lấy thông tin về các đỉnh liền sau một đỉnh v nào đó và trọng số của cung nối v với các đỉnh đó. (Nếu đồ thị được biểu diễn bằng ma trận kề thì ta không cần những mảng này). ở đây với mục đích mô tả thuật toán, tôi chỉ xét đồ thị có không quá 100 đỉnh. Tuy nhiên nếu khai báo dưới dạng mảng động hoặc dùng danh sách liên kết để lưu danh sách các đỉnh liền sau một đỉnh và trọng số của cung tương ứng thì có thể tăng kích thước của bài toán lên rất nhiều. Trong cài đặt tôi chỉ tính thời gian nhỏ nhất. Việc tìm lại con đường xin dành cho bạn đọc.
Bài toán 2: Cũng với giả thiết như bài toán trên nhưng yêu cầu như sau: Ta cần biết thời gian muộn nhất mỗi dự án con có thể bắt đầu mà vẫn đảm bảo thời gian thực hiện dự án lớn là sớm nhất có thể.
Xây dựng đồ thị G = <V,E> như cách giải bài toán 1. Gọi a là đỉnh bắt đầu, b là đỉnh kết thúc của G. Định nghĩa t[v,b] là khoảng thời gian sớm nhất để có thể từ trạng thái v đạt được trạng thái kết thúc. Bằng phân tích tương tự ta cũng có công thức sau:
t[b,b] = 0
t[v,b] = max{t[w,b] + c[v,w]}
w thuộc Q(v)
ở đây Q(v) = {i / (v,i) thuộc E}
Để tính t[v,b], ta phải duyệt các đỉnh của G theo một trật tự topo ngược nào đó. Cách tính này đảm bảo t[v,b] được tính sau khi tất cả t[w,b] được tính, ở đây w là đỉnh liền sau v trong G.
Giả sử (i,j) là cung biểu diễn dự án con đang xét. Khi đó yêu cầu của bài toán là: Tính d(a,b) − c[i,j] − t[j,b], d(a,b) được định nghĩa như bài toán 1.
Dữ liệu vào như bài toán 1 và thêm giả thiết: k là dự án con đang xét.
Phần cài đặt xin dành cho bạn đọc.
Trên đây tôi chỉ đưa ra một ứng dụng rất tiêu biểu của bài toán sắp xếp topo trong đồ thị có hướng. Trên thực tế còn rất nhiều ứng dụng khác của nó (như việc tính giá trị biểu thức,..). Bài viết sau tôi sẽ trình bày về một ứng dụng khác của bài toán trên vào việc tìm bao đóng bắc cầu của đồ thị với số đỉnh lớn. Bạn nào thắc mắc hay có nhu cầu trao đổi về bài toán sắp xếp topo trên đồ thị có hướng xin viết thư cho tôi theo địa chỉ sonnx10@yahoo.com. Tôi xin cảm ơn !

9 .**Ứng dụng lý thuyết Toán để giải các bài toán tin**

Tác giả: Lê Nguyễn Tuấn Thành

Như các bạn đã biết, toán học có ảnh hưởng rất lớn đến mọi lĩnh vực của cuộc sống. Các bài tin nếu có được thuật toán dựa trên cơ sở lí thuyết toán học vững chắc sẽ đem lại kết quả tốt hơn rất nhiều so với các thuật toán khác. Các bạn có thể tìm thấy nhiều bài tin hay ứng dụng toán học để giải trên các số báo trước. Trong bài viết này tôi sẽ trao đổi thêm với các bạn một vài bài tin với các thuật toán dựa trên cơ sở toán học. Các thuật toán tôi đưa ra có thể chưa tối ưu vì vậy rất mong nhận được sự góp ý của các bạn.
Chúng ta hãy bắt đầu bằng bài toán đơn giản sau:
Bài 1: Tìm tất cả các cặp số nguyên dương x,y,z sao cho x2 + y2 = z2. 0< x,y,z ≤10000.
Giải:
Đặt Max là giới hạn trên của x,y,z.
Với bài toán này, chúng ta có thể dùng hai vòng lặp x,y lồng nhau rồi kiểm tra nếu x2 + y2 là số chính phương thì ghi nhận kết quả. Để giảm bớt số lần lặp ta sẽ cho y>x, mỗi lần ghi nhận kết quả ta sẽ phải viết ra hai cặp nghiệm. Ta dùng một biến dem để ghi nhận số cặp x,y,z thoả mãn. Chương trình chính có thể viết như sau:

```
for x:=1 to Max-2 do
for y:=x+1 to Max-1 do
if trunc(sqrt(sqr(x)+sqr(y)))=sqrt(sqr(x)+sqr(y)) then
if trunc(sqrt(sqr(x)+sqr(y)))<=Max then
begin
z:=trunc(sqrt(sqr(x)+sqr(y)));
inc(dem);
writeln('Cach thu',dem);
write(' x=',x,' y=',y,' z=',z);
writeln;
inc(dem);
writeln('Cach thu ',dem);
write(' x=',y,' y=',x,' z=',z);
writeln;
end;
writeln('Co tat cac',dem,' cach');
```

Song ta có thể dùng cách khác để giải bài toán này. Các bạn hãy để ý điều kiện thoả mãn của x,y,z: x2+y2 = z2. Đó chính là phương trình Pitago. Nghiệm của phương trình này có dạng:

```
x=t*2*m*n
y=t*(m2-n2)
```

z=t*(m2+n2)
Với m,n,t là các số nguyên dương; m,n nguyên tố cùng nhau 1 chẵn,1 lẻ. Các bạn có thể tìm thấy chứng minh trên trong các tài liệu về số học, ở đây tôi không chứng minh lại. Từ công thức trên ta có một cách làm khác như sau: Ta đi tìm các giá trị có thể có của m,n,t. Với mỗi bộ ba số m,n,t ta sẽ được hai cặp nghiệm x,y,z. Giả sử n<m.

Ta có n22 ; t ≤ 1 mà t(n2 + m2) = z ≤ Max→ 2*n2 < Max $\Rightarrow n < \sqrt{\frac{Max}{2}}$

Lại có: t(n2+m2)=z ≤ Max → m2 ≤ Max-n2 $\Rightarrow m \leq \sqrt{Max - n^2}$

Và $t \leq \frac{Max}{n^2 + m^2}$

Với các phân tích như trên khoảng giới hạn của m,n,t đã nhỏ đi rất nhiều.Vấn đề còn lại là kiểm tra xem m,n có nguyên tố cùng nhau, có 1 chẵn,1 lẻ không. Để làm điều này ta sẽ viết một hàm tìm Ucln của hai số m,n để kiểm tra tính nguyên tố cùng nhau của m,n.
Còn một vấn đề cần giải quyết trước khi viết chương trình là:
Liệu với hai cặp số m1,n1,t1 và m2,n2,t2 tìm được thì hai cặp nghiệm x1,y1,z1 và x2,y2,z2 có trùng nhau không? Điều này được chứng minh như sau:
Chứng minh.
Giả sử tồn tại hai cặp số nguyên dương khác nhau m1,n1,t1 và m2,n2,t2 cho ta hai nghiệm x1,y1,z1 và x2,y2,z2 trùng nhau. Ta sẽ chứng minh điều giả sử là sai.
Thật vậy: do z được tính theo x,y nên ta xét hai trường hợp:
Th1: x1=x2 và y1=y2
Th2: x1=y2 và y1=x2
*) Trường hợp 1
x1=x2 → t1*m1*n1=t2*m2*n2 (1)
y1=y2 → t1*(m12 − n12)= t2*(m22 − n22) (2)
→ z1=z2 → t1*(m12 + n12)= t2*(m22 + n22) (3)
Từ (2),(3) → t1*m12 = t2*m22 (4)
và t1*n12 = t2*n22 (5)
+ Nếu t1=t2 thì từ (4),(5) → m1=m2 và n1=n2
Như vậy hai bộ m1,n1,t1 và m2,n2,t2 trùng nhau (Vôlý)
+ Nếu t1<>t2. Giả sử t1<t2 → t2>1
- nếu t1=1, gọi d là ước nguyên tố bất kì của t2. Từ (4) (5) → m1,n1 cùng chia hết cho d (Trái với điều kiện của m,n)
- nếu t1<1, gọi d=(t1,t2) đặt t1=t1/d,t2=t2/d → t2<1. Gọi d1 là ước nguyên tố bất kì của t2. Từ (4),(5) → m1,n1 cùng chia hết cho d1 (Vô lí).
*) Trường hợp 2
x1=y2 → t1*2*m1*n1= t2*(m22 − n22) (6)

y1=x2 → t1*(m12 − n12)= t2*2*m2*n2 (7)
→ z1=z2 → t1*(m12 + n12)= t2*(m22 + n22) (8)
Từ (6) → t2 chia hết cho 2 do m,n khác tính chẵn lẻ.
Đặt t2=2n*t (t không chia hết cho 2).
Từ (8) → t1=2n*k (k không chia hết cho 2)
Nhưng từ (6) → t2*(m22 − n22) chia hết cho 2n nhưng không chia hết cho 2n+1
còn t1*2*m1*n1 lại chia hết cho 2n+1 (Mâu thuẫn).
Như vậy bài toán được chứng minh.
Sau đây là chương trình với cách làm vừa phân tích ở trên:

```
uses crt;
const max=10000;
var m,n,k,dem,x,y,z:word;
time:longint;
{***********************************************}
{Tìm ước chung lớn nhất của hai số nguyên dương a,b}
function ucln(a,b:word):word;
var du:word;
begin
ucln:=1;
if (a=1) or (b=1) then exit;
if a mod b=0 then begin ucln:=b; exit; end;
if b mod a=0 then begin ucln:=a; exit; end;
while b<0 do
begin
du:=a mod b;
a:=b;
b:=du;
end;
ucln:=a;
end;
{***********************************************}
BEGIN
clrscr;
dem:=0;
time:=meml[0:$46c];
for n:=1 to trunc(sqrt(max/2)) do
for m:=n+1 to trunc(sqrt(max-sqr(n))) do
if ((m-n) mod 2=1) and (ucln(m,n)=1) then
for k:=1 to max div (sqr(m)+sqr(n)) do
begin
x:=k*2*m*n;
```

```
y:=k*(sqr(m)-sqr(n));
z:=k*(sqr(m)+sqr(n));
inc(dem);
writeln('Cach thu ',dem);
write(' x=',x,' y=',y,' z=',z);
writeln;
inc(dem);
writeln('Cach thu',dem);
write(' x=',y,' y=',x,' z=',z);
writeln;
end;
writeln('Co tat cac ',dem,' cach');
writeln('Thoi gian tinh: ',(meml[0:$46c]-time)/18.2:0:10,' giay');
readln;
END.
```

Các bạn có thể dùng hàm tính thời gian Meml[0:$46C] để so sánh hai cách làm. Tất nhiên với bài toán đơn giản này thì sự khác biệt là không lớn lắm nhất là với Max nhỏ. Nhưng qua đó các bạn cũng có thể thấy được hiệu quả việc áp dụng toán học vào trong các bài tin.

Bài 2: Tìm tất cả các số N có 5 chữ số sao cho 2*N cũng có 5 chữ số và tất các chữ số từ 0 đến 9 đều có mặt trong N và 2*N.

Giải:

Đây cũng là một bài toán khá đơn giản, các bạn có thể dùng quay lui để giải 1 cách dễ dàng, sự chênh lệch về thời gian trong những cài đặt khác nhau là không đáng kể. Nhưng điều tôi muốn nói khi đưa ra bài này là: các bạn hãy thử phân tích bài toán để tìm ra những điều kiện nhằm làm giảm bớt qúa trình quay lui.

Trước hết ta có thể thấy để N, 2*N là số có 5 chữ số thì chữ số hàng vạn của N phải lớn hơn 0 và nhỏ hơn 5. Thứ hai để các chữ số từ 0 đến 9 đều có mặt trong N và 2*N thì chữ số hàng đơn vị của N phải khác 0. Thứ ba, ta thấy nếu trong N có chứa số 9 và liền trước số 9 là một số lớn hơn 4 (dạng 9a với a>4) thì trong 2*N cũng sẽ có số 9, tương tự trong N có chứa số 0 và liền trước số 0 là một số nhỏ hơn 5 (dạng 0a với a<5) thì trong 2*N cũng có số 0.

Với ba nhận xét trên, ta có cách làm như sau: Dùng một mảng một chiều a: array [1..5] of 0..9 để lưu các chữ số của N (a[1] ứng với chữ số hàng đơn vị). Sau đó dùng quay lui để xét các trường hợp có thể có của các chữ số trong N. Chú ý nên dùng một mảng để đánh dấu những chữ số đã xuất hiện trong quá trình quay lui. Để làm được điều này, ta thấy rằng khi nhân một số với 2 thì nhớ nếu có từ một hàng lên hàng liền sau chỉ có thể là 1.Vì vậy trong quá trình quay lui ta có thể biết được những chữ số nào đã xuất hiện trong N và 2*N mà không cần phải đợi đến khi tìm đủ 5 chữ số của N.

Các bạn có thể tham khảo chương trình dưới đây. Hàm kt(j,i) để kiểm tra xem j có thể chọn vào vị trí thứ i của mảng a không?

```
uses crt;
type mt1 = array[1..5] of 0..9;
mt2 = array[0..9] of boolean;
var a : mt1;
đ : mt2;
dem : word;
time: longint;
{************************************************}
function kt(j,i:byte):boolean;
var kt1:boolean;
begin
if a[i-1]>=5 then
begin
kt1:=đ[(2*j+1) mod 10];
if j=9 then kt1:=false;
end
else
begin
kt1:=đ[(2*j) mod 10];
if j=0 then kt1:=false;
end;
kt:=kt1;
end;
{************************************************}
procedure viet;
var i:byte;
begin
if a[5]>=5 then exit;
if a[5]=0 then exit;
write('');
for i:=5 downto 1 do
write(a[i]);
writeln;
inc(dem);
end;
{************************************************}
procedure try(i:byte);
var j:byte;
begin
```

```
for j:=0 to 9 do
if đ[j] and kt(j,i) then
begin
đ[j]:=false;
a[i]:=j;
if a[i-1]>=5 then đ[(2*j+1) mod 10]:=false
else đ[(2*j) mod 10]:=false;

if i=5 then viet
else try(i+1);

đ[j]:=true;
if a[i-1]>=5 then đ[(2*j+1) mod 10]:=true
else đ[(2*j) mod 10]:=true;
end;
end;
{***********************************************}
procedure main;
var i:byte;
begin
dem:=0;
for i:=1 to 9 do
begin
fillchar(a,sizeof(a),0);
fillchar(đ,sizeof(đ),true);
a[1]:=i;
đ[i]:=false;
đ[(2*i) mod 10]:=false;
try(2);
end;
writeln(’’,dem);
end;
{***********************************************}
BEGIN
clrscr;
time:=meml[0:$46C];
main;
writeln(’Thoi gian chay : ’,(meml[0:$46C]-time)/18.2:0:20,’ giay’);
readln;
END.
```

Bài 3 : Cho phương trình A0+A1X+A2X2+ … +AnXn = 0.Với A0,A1,…,An là các

số nguyên. Tìm nghiệm nguyên của phương trình.
Giải
Bài này áp dụng tính chia hết để giải. Do A0,A1,…,An và X là các số nguyên nên từ phương trình ta suy ra A0 chia hết cho X. Vì vậy ta chỉ cần thử tất cả các ước nguyên của A0 để tìm giá trị của X.

Bài 4: Cho số thực R và số nguyên dương Max. Tìm phân số $\frac{P}{Q}$ tối giản (P, Q là hai số nguyên dương, Q ≤ Max) sao cho $\frac{P}{Q}$ gần R nhất.
File vào: PhanSo.inp
Gồm: R, Max nằm trên hai dòng khác nhau
File ra: PhanSo.out
Gồm: P,Q nằm trên cùng một dòng.
Giải
Để làm được bài này, ta thấy ứng với mỗi giá trị cụ thể của Q thì có một giá trị của P để $\frac{P}{Q}$ gần R nhất, giá trị đó là: P=[Q*R] ([x] là số nguyên lớn nhất không vượt quá x). Như vậy ta chỉ cần xét tất cả các giá trị của Q rồi so sánh các phân số lớn nhất tại mỗi giá trị của Q với nhau để tìm phân số gần R nhất.
Chương trình cho bài này như sau:

```
uses crt;
const fi='PhanSo.inp';
fo='PhanSo.out';
var r:real;
max,p,q:word;
{***********************************************}
procedure nhap;
var f:text;
begin
assign(f,fi);
reset(f);
readln(f,r);
readln(f,max);
close(f);
end;
{***********************************************}
{Tìm ước chung lớn nhất của hai số nguyên dương a,b}
function ucln(a,b:word):word;
var du:word;
begin
```

```
ucln:=1;
if (a=1) or (b=1) then exit;
if a mod b=0 then begin ucln:=b; exit; end;
if b mod a=0 then begin ucln:=a; exit; end;
while b>0 do
begin
du:=a mod b;
a:=b;
b:=du;
end;
ucln:=a;
end;
{************************************************}
procedure xuli;
var p1,q1:word;
begin
p:=0; q:=1;
for q1:=1 to max do
begin
p1:=trunc(r*q1);
if p*q1
begin
p:=p1;
q:=q1;
end;
end;
p1:=ucln(p,q);
p:=p div p1;
q:=q div p1;
end;
{************************************************}
procedure ghi_file;
var f:text;
begin
assign(f,fo);
rewrite(f);
writeln(f,p,'',q);
close(f);
end;
{************************************************}
procedure main;
```

```
begin
nhap;
xuli;
ghi_file;
end;
{************************************************}
BEGIN
clrscr;
main;
END.
```

Bài 5: Cho số thực R. Tìm các số thực a1,a2,a3,…,an sao cho $\sum_{1}^{n} a_i = \prod_{1}^{n} a_i = R$. (các số trong dãy a có thể bằng nhau)
File vào: Equal.inp
Gồm: Duy nhất một số thực R
File ra: Equal.out
Gồm: Các số thực a1,a2,a3,…,an mỗi số ghi trên một dòng. Nếu không tìm được số nào thì ghi 'Nó.
Ví dụ:
File vào: 4
File ra: 2
2
Giải
Đối với những bài như trên nếu ai không biết cách thì sẽ rất khó tìm được thuật toán. Với bài này ta sử dụng định lí Viet để giải.
Trước hết ta lập phương trình t2 − R*t + R = 0. (1)
Nếu phương trình (1) vô nghiệm thì không thể phân tích được R. Ngược lại, gọi hai nghiệm của (1) là t1,t2 (t1,t2 có thể bằng nhau). Rồi sau đó tiếp tục phân tích t1,t2. Như vậy ta phải cài đặt một chương trình đệ quy để tiến hành phân tích R.
Sau đây là chương trình mẫu các bạn có thể tham khảo

```
uses crt;
const fi=’Equal.inp’;
fo=’Equal.out’;
var r,delta:real;
g:text;
{************************************************}
Procedure nhap;
var f:text;
begin
assign(f,fi);
```

```
reset(f);
readln(f,r);
close(f);
assign(g,fo);
rewrite(g);
if r<4 then
begin
writeln(g,'No');
close(g);
halt;
end;
end;
{************************************************}
Procedure xuli(r:real);
var x1,x2:real;
begin
delta:=sqr(r)-4*r;
if delta>=0 then
begin
x1:=(r+sqrt(delta))/2;
x2:=(r-sqrt(delta))/2;
xuli(x1);
xuli(x2);
end
else writeln(g,r:0:5);
end;
{************************************************}
Procedure main;
begin
nhap;
xuli(r);
close(g);
end;
{************************************************}
BEGIN
clrscr;
main;
END.
```

Bài 6: Cho số nguyên dương K và chữ số a (a>0). Tìm một số nguyên dương chỉ gồm các chữ số a và 0 chia hết cho K

File vào: Chiahet.inp

Gồm: K,a (K ≤ 30000)
File ra: Số cần tìm.
Ví dụ:
File vào: 3 1
File ra: 111
Giải
Với bài này ta áp dụng nguyên lí Dirile để giải. Ta thành lập K số mỗi số gồm lần lượt 1,2,3,…,K số a.Tức là:
Số thứ nhất là a
Số thứ hai là aa
Số thứ ba là aaa
…
Nếu trong K số này có 1 số chia hết cho K thì ghi nhận và kết thúc. Nếu không thì K số này khi chia cho K sẽ nhận 1 trong K−1 số dư từ 1 đến K−1. Như vậy theo nguyên lí Dirile sẽ có ít nhất hai trong K số có cùng số dư khi chia cho K hiệu của chúng sẽ chia hết cho K đồng thời hiệu hai số này chỉ gồm các chữ số a và 0.
Để giải quyết bài toán một cách nhanh chóng ta phải nghĩ cách làm giảm bớt công sức tìm kiếm số dư của một số bất kì trong K số được thành lập như trên. Cách làm của tôi như sau:
Dùng một mảng du: array[1..max] of word để lưu số dư của các số khi chia cho K, một biến du10 để lưu số dư của 10i khi chia cho K. Ta sẽ cập nhật du10 sau mỗi lần tăng i như sau: du10:=du10*10 mod k. Do đó số dư của số thứ i khi chia cho K sẽ là:
du[sl]:=(du10*a mod k + du[sl-1]) mod k
Với thuật toán trên, ta có chương trình như sau:

```
uses crt;
const fi=’Chiahet.inp’;
fo=’Chiahet.out’;
max=32000;
var du : array[1..max] of word;
sl,k,a : word;
{***********************************************}
procedure nhap;
var f:text;
begin
fillchar(du,sizeof(du),0);
assign(f,fi);
reset(f);
read(f,k,a);
close(f);
end;
{***********************************************}
```

```
function kt(sl:word;var vt:word):boolean;
var i:word;
begin
kt:=true;
vt:=0;
for i:=1 to sl-1 do
if du[i]=du[sl] then
begin
vt:=i;
exit;
end;
kt:=false;
end;
{***********************************************}
procedure xuli;
var vt:word;
du10:longint;
g:text;
begin
sl:=1;
du[1]:=a mod k;
du10:=1;
if du[1]>0 then
begin
repeat
inc(sl);
du10:=du10*10 mod k;
du[sl]:=(du10*a mod k + du[sl-1]) mod k;
if du[sl]=0 then begin vt:=0; break; end;
until kt(sl,vt);
end
else vt:=0;
assign(g,fo);
rewrite(g);
for du10:=vt+1 to sl do write(g,a);
for du10:=1 to vt do write(g,0);
close(g);
end;
{***********************************************}
procedure main;
begin
```

```
nhap;
xuli;
end;
{***********************************************}
BEGIN
clrscr;
main;
END.
```

Kết qủa bài này có thể là một số rất dài. Trong chương trình trên tôi chỉ cho ghi trên một dòng vì vậy có thể gây ra lỗi tràn dòng, các bạn có thể sửa lại một chút để ghi trên nhiều dòng.

Bài 7: Cho số nguyên dương N (N>1). Hãy viết tất cả những phân số tối giản nhỏ hơn 1 có mẫu nhỏ hơn hoặc bằng N theo thứ tự tăng dần.

File vào: Faray.inp

Gồm: N

File ra: Faray.out

Gồm: Nhiều dòng, mỗi dòng gồm hai số nguyên dương biểu thị tử và mẫu cả một phân số.

Ví dụ:

File Faray.inp: 4

File Faray.out: 1 4

1 3

1 2

2 3

3 4

<u>Giải</u>

Đây là một bài khá hay.Thuật toán trước tiên nhiều bạn sẽ nghĩ khi đọc bài này là dùng hai vòng lặp lồng nhau để tìm tử và mẫu của các phân số, sau đó sắp xếp lại theo thứ tự tăng dần. Nhưng cách làm đó khá lâu và tốn bộ nhớ. Tôi sẽ giới thiệu với các bạn một thuật toán đã được nhà toán học người Anh là Faray tìm ra.

Chúng ta hãy để ý dãy phân số tìm được. $\frac{1}{4}\ \frac{1}{3}\ \frac{1}{2}\ \frac{2}{3}\ \frac{3}{4}$

Chúng ta sẽ thấy một tính chất thú vị sau: Phân số ở giữa có tử và mẫu theo thứ tự bằng tổng các tử và mẫu của hai phân số ở hai bên.

Ví dụ:

$\frac{1}{3}=\frac{1+1}{4+2}$ hoặc $\frac{1}{2}=\frac{1+2}{3+3}$ .....

Từ nhận xét đó nhà toán học người Anh đã đưa thuật toán để tìm tất cả các phân số như sau:

Bước 1: Bắt đầu bằng hai phân số $\frac{0}{1}$ và $\frac{1}{1}$

Bước 2 : Ta lấy tổng của tử và mẫu của hai phân số kề nhau trong dãy được tử và mẫu của một phân số mới. Nếu mẫu của phân số này nhỏ hơn hoặc bằng N thì ta đặt phân số mới này vào dãy ở giữa hai phân số ban đầu. Lặp lại Bước 2 cho đến khi không làm được nữa. Cuối cùng khi ghi ra ta bỏ đi hai phân số $\frac{0}{1}$ và $\frac{1}{1}$ là được dãy phân số cần tìm.

Cái hay của thuật toán trên là dãy phân số thành lập như trên luôn là dãy tăng và ta không phải kiểm tra tính tối giản của các phân số mà các phân số mới thêm vào luôn tối giản.

Để các bạn có thể hiểu kĩ hơn về cách làm tôi sẽ lấy ví dụ với N=4

Lúc đầu ta có hai phân số $\frac{0}{1}$ và $\frac{1}{1}$

Lần thứ 1:

Thêm phân số $\frac{1}{2}$ vào dãy $(\text{do } \frac{1}{2}=\frac{0+1}{1+1} \text{ và } 2<4)$ được dãy mới: $\frac{0}{1}\ \frac{1}{2}\ \frac{1}{1}$

Lần thứ 2: Thêm được hai phân số $\frac{1}{3}$ và $\frac{2}{3}$ vào dãy được dãy mới: $\frac{0}{1}\ \frac{1}{3}\ \frac{1}{2}\ \frac{2}{3}\ \frac{1}{1}$

Lần thứ 3: Lập được các phân số $\frac{1}{4}\ \frac{2}{5}\ \frac{3}{5}\ \frac{3}{4}$ nhưng ta chỉ thêm được vào dãy hai phân số $\frac{1}{4}$ và $\frac{3}{4}$ do có mẫu bằng 4,còn hai phân số kia có mẫu bằng 5>4. Lúc này dãy mới là:

$\frac{0}{1}\ \frac{1}{4}\ \frac{1}{3}\ \frac{1}{2}\ \frac{2}{3}\ \frac{3}{4}\ \frac{1}{1}$ Đến đây ta không thể làm tiếp được nữa,dãy phân số cần tìm là:

$\frac{1}{4}\ \frac{1}{3}\ \frac{1}{2}\ \frac{2}{3}\ \frac{3}{4}$ Với thuật toán trên ta sẽ cài đặt 1 chương trình đệ quy để làm bước thứ 2.

Các bạn có thể tham khảo chương trình sau:

```
uses crt;
const fi=’Faray.inp’;
fo=’Faray.out’;
var t1,m1,t2,m2,n:word;
g:text;
{***********************************************}
procedure nhap;
```

```
var f:text;
begin
assign(f,fi);
reset(f);
readln(f,n);
close(f);
assign(g,fo);
rewrite(g);
end;
{************************************************}
procedure viet(t1,m1:word);
begin
if t1>0 then
writeln(g,t1,' ',m1);
end;
{************************************************}
procedure xuli(t1,m1,t2,m2:word);
begin
if m1+m2<=n then
begin
xuli(t1,m1,t1+t2,m1+m2);
xuli(t1+t2,m1+m2,t2,m2);
end
else viet(t1,m1);
end;
{************************************************}
procedure main;
begin
nhap;
t1:=0; m1:=1;
t2:=1; m2:=1;
xuli(t1,m1,t2,m2);
close(g);
end;
{************************************************}
BEGIN
clrscr;
main;
END.
```

Dãy phân số tìm được có thể rất lớn, nhất là với N>100 vì vậy không thể ghi ra file được. Để khắc phục điều này các bạn có thể viết nhiều phân số trên 1 dòng. Nhưng

cách này cũng chỉ giải pháp tạm thời. Bài toán sẽ hay hơn nếu không bắt viết hết toàn bộ dãy phân số mà chỉ yêu cầu viết 1 hay vài phân số của dãy, lúc đó phải thêm vào một biến dem để ghi nhận vị trí của 1 phân số trong dãy.

10.Ứng dụng lý thuyết Toán để giải các bài Tin

Tác giả: Lê Nguyễn Tuấn Thành

(Tiếp theo số trước)

Bài 8

Câu 1 Cho một số thập phân vô hạn tuần hoàn dạng A,B(C) với A là phần trước dấu phẩy, B là phần sau dấu phẩy không tuần hoàn, C là phần thập phân tuần hoàn (A,B,C là các số nguyên dương, C<>9). Hãy viết phân số tối giản biểu diễn số thập phân đó.

File vào: Thapphan.inp

Gồm: 3 số A,B,C ghi trên 3 dòng khác nhau. Nếu B= -1 thì sau dấu phẩy của số đóự chỉ có phần vô hạn tuần hoàn.

File ra : Thapphan.out

Gồm : Tử và mẫu của phân số tìm được, ghi trên hai dòng

Câu 2: Cho phân số có tử và mẫu lần lượt là P,Q (P,Q là các số nguyên dương). Hãy viết dạng biểu diễn thập phân của phân số đó.

File vào : Fraction.inp

Gồm: P,Q ghi trên cùng một dòng cách nhau bởi dấu cách.

File ra: Fraction.out

Gồm: Nếu biểu diễn thập phân là vô hạn tuần hoàn thì ghi ra dạng A B C với ý nghĩa như câu 1.

Nếu biểu diễn thập phân là hữu hạn thì ghi ra

dạng A B.

Giải

Để giải quyết bài này trước hết tôi nhắc lại 2 định lí quan trọng về phân số được đề cập đến trong SGK lớp 7:

Định lí 1: Nếu một phân số tối giản mà mẫu lớn hơn 0 và mẫu không có ước nguyên tố nào khác 2 và 5 thì phân số đó được viết dưới dạng số thập phân hữu hạn.

Định lí 2: Nếu một phân số tối giản mà mẫu lớn hơn 0 và mẫu có ước nguyên tố khác 2 và 5 thì phân số đó được viết dưới dạng số thập phân vô hạn tuần hoàn. Bây giờ ta sẽ đi tìm phân số biểu diễn của một số thập phân vô hạn tuần hoàn dạng 0,(c). Gọi k là số chữ số của c

Ta có $0,(c)= \frac{cccc.....}{10^k*10^k*10^k*10^k.....}$ (n số c và n số 10^k)

hay $0,(c)= \frac{c*(10^{(n-1)*k}+10^{(n-2)*k}+.....+10^{2*k}+10^k+1)}{10^k*10^k*10^k*10^k.....*10^k}$ (có n số 10^k)

Mà $10^{(n-1)*k}+10^{(n-2)*k}+....+10^{2*k}+10^k+1=\frac{10^{n*k}-1}{10^k-1}$ (Cấp số nhân với công bội bằng 10^k)

$\Rightarrow 0,(c) = \frac{c*\frac{10^{n*k}-1}{10^k-1}}{10^{k*n}}$ (với n tiến đến vô cùng)

hay $0,(c)= \lim_{n\to\infty}\frac{c*\frac{10^{n*k}-1}{10^k-1}}{10^{k*n}} = \lim_{n\to\infty}\frac{c*(10^{n*k}-1)}{(10^k-1)*10^{k*n}}$

$= \lim_{n\to\infty}(\frac{c}{10^k-1}-\frac{c}{(10^k-1)*10^{n*k}}) = \lim_{n\to\infty}\frac{c}{10^k-1} - \lim_{n\to\infty}\frac{c}{(10^k-1)*10^{n*k}}$

$=\frac{c}{10^k-1} - 0 = \frac{c}{10^k-1}$

Vậy $0,(c) = \frac{c}{10^k-1}$ với k là số chữ số của c

Các bạn có thể kiểm chứng lại kết quả trên

Ví dụ: $0,(1)=\frac{1}{9}$

$0,(3)= \frac{3}{9} = \frac{1}{3}$

$0,(57)= \frac{57}{99}$

$0,(05)= \frac{5}{99}$

Nếu bạn nào tinh ý sẽ để ý thấy rằng số thập phân 0,(9) sẽ không có phân số nào biểu diễn, bởi vì theo trên $0,(9)=\frac{9}{9}=1$

Điều này không đúng. Trong chương trình tôi viết cho Câu 1 nếu nhập vào 0 −1 9 thì kết quả nhận được là $\frac{1}{1}$

Đây có lẽ là sự thú vị nhất về số thập phân vô hạn tuần hoàn.

Định lí 1 và 2 khẳng định mọi phân số đều có thể biểu diễn thành dạng thập phân hữu hạn hoặc vô hạn tuần hoàn, đến đây ta có thể thấy điều ngược lại không đúng.

Từ công thức tìm được ở trên, ta có thể dễ dàng giải quyết được Câu 1.
Gọi t là số chữ số của B.

Ta có: $0,(C)=\frac{C}{10^k-1}$ (k là số chữ số của C)

$$\Rightarrow 0,B(C)=\frac{B,(C)}{10^t}=\frac{B\frac{C}{10^k-1}}{10^t}=\frac{B*(10^k-1)+C}{10^t*(10^k-1)}$$

$$\Rightarrow \mathbf{A},B(C)=\mathbf{A}\frac{B*(10^k-1)+C}{10^t*(10^k-1)}=\frac{A*10^t*(10^k-1)+B*(10^k-1)+C}{10^t*(10^k-1)}$$

$$=\frac{A*10^{t+k}-A*10^t+B*10^k+C-B}{10^t*(10^k-1)}$$

Như vậy là ta đã tìm được phân số biểu diễn số thập phân vô hạn tuần hoàn A,B(C). Để giải quyết bài toán một cách trọn vẹn thì các bạn phải giải 2 trường hợp B = -1 và B ≠-1. Công thức ở trên mới chỉ giải trường hợp B≠-1. Đối với trường hợp B = -1, bằng cách làm tương tự như trên, ta có được công thức sau:

$\mathbf{A},(C)=\frac{A*10^k+C-A}{10^k-1}$ (k là số chữ số của C).

Còn một vấn đề cần giải quyết trước khi viết chương trình đó là: có trường hợp trong dạng biểu diễn của B,C có các chữ số 0 đứng đầu. Như vậy nếu ta đọc B,C từ file với dạng số thì sẽ làm mất đi các chữ số 0 đứng đầu, vì thế sẽ làm sai lệch số chữ số của B,C và khi thay vào công thức tính sẽ cho kết quả sai.
Tôi đã giải quyết trường hợp này bằng cách đọc giá trị của B,C vào các xâu, sau đó chuyển xâu sang dạng số để giải. Bằng cách này ta sẽ lưu được số chữ số thực của B,C.
Chương trình tôi viết cho hai câu trong bài này chỉ xử lí được các số trong phạm vi longint. Bạn nào muốn xử lí các số lớn hơn phải cài đặt các thuật toán xử lí số lớn. Các thuật toán này đã được giới thiệu trên các số báo trước. Các bạn có thể tham khảo lại để cài đặt.
Sau đây là chương trình cho Câu 1:

```
uses crt;
const fi='ThapPhan.inp';
fo='ThapPhan.out';
var a,b,c,nb,nc,p,q:longint;
xaub:string;
xauc:string;
{**********************************************}
procedure nhap;
var f:text;
```

```
z:integer;
begin
assign(f,fi);
reset(f);
readln(f,a);
readln(f,xaub);
readln(f,xauc);
close(f);
val(xaub,b,z);
val(xauc,c,z);
end;
{************************************************}
{Tìm ước chung lớn nhất của hai số nguyên dương a,b}
function ucln(a,b:longint):longint;
var du:longint;
begin
ucln:=1;
if (a=1) or (b=1) then exit;
if a mod b=0 then begin ucln:=b; exit; end;
if b mod a=0 then begin ucln:=a; exit; end;
while b>0 do
begin
du:=a mod b;
a:=b;
b:=du;
end;
ucln:=a;
end;
{************************************************}
{Tính 10a}
function mu10(a:longint):longint;
var i,tich:longint;
begin
tich:=1;
for i:=1 to a do
tich:=tich*10;
mu10:=tich;
end;
{************************************************}
{ Cho trường hợp B<>-1}
procedure xuli1;
```

```
begin
nb:=length(xaub);
nc:=length(xauc);
p:=a*mu10(nb)*mu10(nc)-a*mu10(nb)+b*mu10(nc)+c-b;
q:=mu10(nb)*(mu10(nc)-1);
end;
{***********************************************}
{ Cho trường hợp B=-1}
procedure xuli2;
begin
nc:=length(xauc);
p:=a*mu10(nc)+c-a;
q:=mu10(nc)-1;
end;
{***********************************************}
procedure main;
var g:text;
d:longint;
begin
nhap;
if b=-1 then xuli2
else xuli1;
d:=ucln(p,q);
p:=p div d;
q:=q div d;
assign(g,fo);
rewrite(g);
writeln(g,p);
writeln(g,q);
close(g);
end;
{***********************************************}
BEGIN
clrscr;
main;
END.
```

Bây giờ chúng ta sẽ giải quyết Câu 2.

Trước hết ta thấy rằng theo định lí 1 và 2 dạng biểu diễn của phân số $\frac{P}{Q}$ chỉ có thể là hữu hạn hoặc vô hạn tuần hoàn. Là hữu hạn hay vô hạn tuần hoàn phụ thuộc vào Q.

Để giải quyết bài toán với các số nhỏ hơn ta đưa phân số $\frac{P}{Q}$ về dạng tối giản
Ta có thể dễ dàng tìm được giá trị của A : A=P div Q.
Lúc đó ta thay P=P mod Q. Từ đây trở đi ta chỉ xét các phân số có tử nhỏ hơn mẫu.
Vì vậy chỉ cần đi tìm B và C.
Ta sẽ phân tích Q thành dạng: 2t*3k*5n*U (Với t,k,n,U là các số nguyên dương).
+ Trường hợp 1 Trong phân tích ra thừa số nguyên tố của Q chỉ chứa 2 và 5 mà không chứa bất kì số nguyên tố nào khác, tức là k=0 và U=1 (Q = 2t*5n)

Lúc này dạng biểu diễn của $\frac{P}{Q}$ sẽ là hữu hạn hay có dạng 0,B.
Để tìm được B ta làm như sau:
- Nếu t ≤ n ta đưa $\frac{P}{Q}$ về dạng $\frac{P*5^{t-n}}{2^t*5^t}=\frac{P*5^{t-n}}{10^t}$

Như vậy B=P*2n-t. Gọi nb là số chữ số của B.
Như vậy trong biểu diễn dạng số thập phân thì số chữ số 0 đứng liền sau dấu phẩy sẽ bằng: n-nb.
- Nếu t >n ta đưa $\frac{P}{Q}$ về dạng $\frac{P*5^{t-n}}{2^t*5^t}=\frac{P*5^{t-n}}{10^t}$
Như vậy B=P*5t-n. Gọi nb là số chữ số của B.
ị Trong biểu diễn dạng số thập phân thì số chữ số 0 đứng liền sau dấu phẩy sẽ bằng: t-nb.
+Trường hợp 2: Trong phân tích ra thừa số nguyên tố của Q có chứa các số nguyên tố khác 2 và 5. Q có dạng 2t*3k*5n*U.

Như vậy dạng biểu diễn của $\frac{P}{Q}$ sẽ là vô hạn tuần hoàn hay có dạng 0,B(C).

Theo chứng minh ở trên ta phải đưa $\frac{P}{Q}$ về dạng
$\frac{X}{10^i*(10^j-1)}$ hay dạng $\frac{X}{2^i*5^i*9*111..11}$ (X là một số nguyên dương và có j chữ số 1 dưới mẫu)
Ta sẽ dùng một biến thương để lưu lượng phải nhân thêm vào P và Q để đưa được về dạng $\frac{X}{2^i*5^i*9*111..11}$, một biến mu10 để lưu số mũ của 10 trong biểu diễn của Q ở trên (cụ thể mu10 =i), một biến sl1 để lưu số chữ số 1 trong biểu diễn của Q (cụ thể sl1 =j)
Để đơn giản, ta sẽ loại bỏ khỏi Q phần chứa 2i*5i*9 sau khi nhân thêm với thương. Khi đó Q mới sẽ là ước của 111..11 (j số 1).

Cách làm như sau:
Khởi tạo thương=1 ; Q:=Q div 2t*3k*5n
Lúc đầu thương, mu10 được tính như sau:
- Nếu t≤ n thì thương:= thương*2n-t, mu10:=n
Ngược lại nếu t > n thì thương :=<i>thương*5t-n, mu10:=t
- Nếu k >2 thì Q:=Q*3k-2
Vấn đề còn lại là đi tìm sl1 để 111..11 (có sl1 chữ số1, ta thay j ở trên bằng biến sl1) chia hết cho Q. Theo định lí 2 thì sẽ luôn tìm được sl1.
Ta sẽ dùng cách làm tương tự như cách đã làm ở bài 6, nhưng phải chú ý cập nhật thương. Tôi dùng cách làm này để các bạn có thể dễ dàng áp dụng khi cài đặt các thuật toán xử lí số lớn.
Thêm hai biến phu và thương10 có ý nghĩa sau:
phu để lưu thương của 111..11 khi chia cho Q
thương10 để lưu thương của 10sl1 khi chia cho Q, thương10
sẽ được cập nhật liên tục khi sl1 tăng lên.
Thủ tục tìm sl1 sẽ như sau:
Chú ý: k trong thủ tục thay cho Q.

```
Procedure so_luong_so_1(k:longint);
var du,du10,thuong10,phu:longint;
begin
sl1:=1;
if k=1 then exit;
phu:=0;
du:=1; du10:=1;
thuong10:=0;
repeat
inc(sl1);
thuong10:=thuong10*10+du10*10 div k;
du10:=du10*10 mod k;
phu:=phưthuong10+(dưdu10) div k;
du:= (dưdu10) mod k;
until du=0;
thuong:=thuong*phu;
end;
```

Biến du sẽ lưu số dư của 111..11 khi chia cho Q.
Biến thương sẽ được tính lại là: thuong:=thuong*phu
Chương trình bị hạn chế bởi thương, thương10, phu là dạng longint nên chỉ chạy được với Q khá nhỏ. Các bạn có thể bỏ các dòng lệnh chứa thương,thương10, phu để có thể tìm sl1 với Q lớn hơn.
Còn một lưu ý nữa là: khi mu10 = 0 thì B = -1, do đó phải xử lý thêm trường hợp này.

Sau khi tìm được thương thì P:=P*thương.
Để viết được dạng biểu diễn thập phân, chúng ta phải tìm được B và C. Ta thấy rằng nếu không tính các chữ số 0 đứng đầu thì số chữ số của B Ê mu10, số chữ số của C ≤ sl1. Theo trên P sẽ có dạng: B*(10sl1-1)+C.

Ta có: $P = B*(10^{sl1}-1)+C \Rightarrow B \le \frac{P}{10^{sl1}-1}$.

Từ đó ta sẽ duyệt toàn bộ các giá trị có thể có của B và C (với chú ý điều kiện về số chữ số của B,C).
Sau khi tìm được B,C cách tính số chữ số 0 đứng đầu trong B,C các bạn có thể làm theo cách đã giới thiệu ở trong trường hợp 1.
Sau đây là chương trình cho câu 2:

```
uses crt;
const fi='Fraction.inp';
fo='Fraction.out';
var p,q,a,b,c,thuong:longint;
mu2,mu3,mu5,mu10,sl1:word;
f:text;
{************************************************}
procedure nhap;
begin
assign(f,fi);
reset(f);
readln(f,p,q);
close(f);
assign(f,fo);
rewrite(f);
end;
{************************************************}
{Tìm số mũ của số nguyên tố a trong dạng phân tích ra thừa số nguyên tố của So}
function mu(a:word;var so:longint):word;
var k:word;
begin
k:=0;
while so mod a=0 do
begin
inc(k);
so:=so div a;
end;
mu:=k;
end;
{************************************************}
```

```
{Tìm ước chung lớn nhất của hai số nguyên dương a,b}
function ucln(a,b:longint):longint;
var du:longint;
begin
ucln:=1;
if (a=1) or (b=1) then exit;
if a mod b=0 then begin ucln:=b; exit; end;
if b mod a=0 then begin ucln:=a; exit; end;
while b>0 do
begin
du:=a mod b;
a:=b;
b:=du;
end;
ucln:=a;
end;
{***********************************************}
{Tinh as }
function mu_as(a,s:word):longint;
var i,tich:longint;
begin
tich:=1;
for i:=1 to s do
tich:=tich*a;
mu_as:=tich;
end;
{***********************************************}
{Tìm số chữ số cảu một số nguyên dương a}
function scs(a:longint):longint;
var k:longint;
begin
if a=0 then begin scs:=1; exit; end;
k:=0;
while a>0 do
begin
a:=a div 10;
inc(k);
end;
scs:=k;
end;
{***********************************************}
```

```
procedure huu_han;
var so,i:word;
begin
p:=p*thuong;
so:=scs(p);
if mu2>=mu5 then
begin
for i:=1 to mu2-so do write(f,0);
write(f,p);
end
else
begin
for i:=1 to mu5-so do write(f,0);
write(f,p);
end;
close(f);
halt;
end;
{***********************************************}
procedure so_luong_so_1(k:longint);
var du,du10,thuong10,phu:longint;
begin
sl1:=1;
if k=1 then exit;
phu:=0;
du:=1; du10:=1;
thuong10:=0;
repeat
inc(sl1);
thuong10:=thuong10*10+du10*10 div k;
du10:=du10*10 mod k;
phu:=phưthuong10+(dưdu10) div k;
du:=(dưdu10) mod k;
until du=0;
thuong:=thuong*phu;
end;
{***********************************************}
procedure vo_han_tuan_hoan;
var phu,lim:longint;
begin
so_luong_so_1(q);
```

```
p:=p*thuong;
if mu10=0 then
begin
write(f,-1,' ');
for phu:=1 to sl1-scs(p) do write(f,0);
write(f,p);
close(f);
halt;
end;
phu:=mu_as(10,sl1)-1;
lim:=p div phu;
for b:=0 to lim do
if scs(b)<=mu10 then
begin
c:=p-b*phu;
if scs(c)<=sl1 then break;
end;
for phu:=1 to mu10-scs(b) do write(f,0);
write(f,b,' ');
for phu:=1 to sl1-scs(c) do write(f,0);
write(f,c);
close(f);
end;
{***********************************************}
procedure xuli;
var d:longint;
begin
d:=ucln(p,q);
p:=p div d;
q:=q div d;
a:=p div q;
p:=p mod q;
write(f,a,' ');
mu2:= mu(2,q);
mu3:= mu(3,q);
mu5:= mu(5,q);
thuong:=1;
if mu2>=mu5 then begin thuong:=thuong*mu_as(5,mu2-mu5); mu10:=mu2; end
else begin thuong:=thuong*mu_as(2,mu5-mu2); mu10:=mu5; end;
if (mu3=0) and (q=1) then huu_han;
```

```
if mu3<2 then thuong:=thuong*mu_as(3,2-mu3)
else if mu3>2 then q:=q*mu_as(3,mu3-2);
vo_han_tuan_hoan;
end;
{***********************************************}
procedure main;
begin
nhap;
xuli;
end;
{***********************************************}
BEGIN
clrscr;
main;
END.
```

Trước khi kết thúc bài viết này, tôi muốn đưa ra cho các bạn một tính chất thú vị của số 7 về các số thập phân vô hạn tuần hoàn đó là:

$\frac{1}{7}$ =0,(142857)

$\frac{2}{7}$ =0,(285714)

$\frac{3}{7}$ =0,(428571)

$\frac{4}{7}$ =0,(571428)

$\frac{5}{7}$ =0,(714285)

$\frac{6}{7}$ =0,(857142)

Liệu còn số nào có tính chất như vậy không? Các bạn thử tìm xem.
Nếu bạn nào cần chương trình nguồn của tất cả các bài đã nêu trên xin hãy liên hệ với tôi.

11. **Trở lại bài toán tìm chữ số tận cùng khác 0 của n!**

Tác giả: Ngô Minh Đức

Trong bài "Tối ưu các bài tóan về số nguyên tố" (THNT 10/2003), tác giả Đinh Hữu Công đã nêu ra thuật toán "Tìm chữ số tận cùng khác 0 của n!" với n<231.
Trong bài này tôi xin nêu ra một cách khác giải quyết bài tóan này vô cùng nhanh

chóng. Cơ sở của nó hòan tòan từ bài viết "Tìm chữ số tận cùng khác 0 của n!" (Least Significant Non−Zero Digit of n!) của Mathpages site.
Sau đây là bài báo tiếng Anh trên đã được dịch và thay đổi một số đọan:
Gọi L(k) là chữ số tận cùng khác 0 của một số nguyên k bất kỳ. Các giá trị đầu tiên của dãy L(k!) với k=2,3,4,…là:
2,6,4,2,2,4,2,8,8,8,6,8,2,8,8,6,8,2,4,4,8,4,6,4,4,8,4,6,8,…
Chúng ta sẽ tìm cách xác định giá trị của L(k!) với mọi k là số tự nhiên cho trước.
Viết n! dưới dạng:
N!=2a2.3a3.5a5.7a7…
Chúng ta ký hiệu (n!)' để chỉ số n! đã bỏ đi hết các chữ số 0 tận cùng

Ta có định lý:
Trong phân tích của n! số mũ ap của p được tính theo công thức:

$$a_p = \left[\frac{n}{p}\right] + \left[\frac{n}{p^2}\right] + \left[\frac{n}{p^3}\right] + \ldots + \left[\frac{n}{p^k}\right]$$

(với k là số thỏa mãn $p^k \le n \le p^{k+1}$)

Dựa vào các tính chất về phần nguyên, dễ thấy nếu p1 < p2 thì ap1≥ ap2

Vậy thì a2 ≥ a5, từ đó suy ra số mũ của 10 trong n! là a10 = a5. Bỏ đi hết các lũy thừa của 10 trong n! ta được (n!)':
N!'=2a2.3a3.5a5.7a7…
Dựa vào định lý trên, bạn hãy chứng minh rằng với n>2, trong phân tích của n!, ta luôn có a2 > a5 (tức là dấu = không xảy ra) hay a2 − a5>0. Do đó (n!)' chia hết cho 2 và L(n!)=L(n!') sẽ nhận một trong các giá trị {2;4;6;8}
Mặt khác, ta có công thức: n!=n(n-1)!
Kết hợp hai điều này ta có thể suy ra:
L(n!)=L(L(n).L((n -1)!)) với mọi n không chia hết cho 5
Như vậy nếu đã biết được giá trị của L((5n)!) thì ta có thể tính được giá trị của L((5n+j)!) với j=0,1,2,3,4
Các giá trị của L(n!) được xếp cột thành từng chuỗi 5 chữ số như sau:
n: 01234 5 10 15 20 25 30 35 40 45
L(n!): 01264 22428 88682 88682 44846 44846 88682 22428 22428 66264 …

(ta bỏ qua một số giá trị đặc biệt như 0!, 1! − xem như L(0!)=0)
Tuy nhiên giá trị của L((5n)!) không thực sự rõ ràng. Nếu ta tiếp tục xếp cột các giá trị của L((5n)!) thì ta sẽ thấy chúng cũng lập thành những chuỗi 5 chữ số. Điều đó cho phép ta nêu một kết luận tương tự: chỉ cần biết giá trị của L((25n)!) thì sẽ biết được L((25n+5j)!) với j=0,1,2,3,4 (nghĩa là giá trị của L((25n+5j)!) chỉ phụ thuộc vào giá trị của L((25n)!)

Tiếp tục như thế, chúng ta có thể sắp xếp các giá trị của L((5kn)!) như bảng sau:

```
n       : 0 1 2 3 4 5   10  15  20  25  30  35  40  45

L(n!)   : 01264 22428 88682 88682 44846 44846 88682 22428 22428 66264
L((5n)!) : 02884 48226 24668 48226 48226 86442 24668 62884 24668 24668
L((25n)!) : 04244 82622 82622 28488 46866 64244 82622 82622 28488 46866
L((125n)!): 08824 68824 26648 68824 42286 26648 26648 42286 26648 84462
L((625n)!): 06264 22428 88682 88682 44846 44846 88682 22428 22428 66264
...
```

Ta rút ra được kết luận (không thấy tác giả chứng minh): giá trị của L((5kn+5k-1j)!) chỉ phụ thuộc vào giá trị của L((5kn)!) với j=0,1,2,3,4. Nếu giá trị của L((5kn)!) xác định được thì giá trị của L((5kn + 5k-1j)!) lập tức cũng xác định được.

Ví dụ:

Ta có L((125.3)!)=L(375!)=2 và L((125.3+25)!)=L(400!)=8

Thế thì L((125.8)!)=L(1000!)=2 nên L((125.8+25)!)=L(1025)! cũng bằng 8

và L((125.10)!=L(1250!)=2 nên L((125.10+25)!)=L(1275!) cũng bằng 8)…

Tóm lại ta khẳng định nếu L((125.k)!)=2 thì chắc chắn L((125.k+25)!)phải là chữ số thứ hai trong chuỗi 28488 (chữ số 8)

Mặt khác, ta nhận thấy các giá trị của L((625n)!) giống với L(n!). Điều đó cho thấy L((5kn)!)=L((5k+4n)!) với n≠1 (không thấy tác giả chứng minh). Trên mỗi dòng ta thấy có đúng 5 chuỗi phân biệt gồm 5 chữ số, mỗi chuỗi bắt đầu bằng một trong các số {0,2,4,6,8}. Ta chỉ cần tổng hợp tất cả các chuỗi này lại thành một bảng để sử dụng

Chúng ta rút ra được bảng sau dùng để tính giá trị của L((5kn +5k−1j)!) với j=0,1,2,3,4 khi đã biết giá trị của L((5kn)!):

```
   j    01234 01234 01234 01234 01234

k mod 4
-------
  0     06264 22428 44846 66264 88682
  1     02884 24668 48226 62884 86442
  2     04244 28488 46866 64244 82622
  3     08824 26648 42286 68824 84462
```

Chẳng hạn nếu đã biết L((125.2)!)=8 ta tính được L((125.2+25.2)!)=6 bằng cách nhìn vào hàng (2 mod 4)=2 (do 25=52), chuỗi bắt đầu bằng chữ số 8 (82622), vị trí j=2

Thuật toán tìm L(n!)

Đổi n ra cơ số 5, ta được:

$$n = d_n 5^n + d_{n-1}5^{n-1} + \ldots + d_2 5^2 + d_1 5 + d_0 = \overline{d_n d_{n-1} \ldots d_1 d_0}\ (5)$$

Dựa vào bảng trên, ta thấy nếu biết L((dn5n)!) ta sẽ tính được L((dn5n +dn-15n-1)!) Tương tự khi biết được L((dn5n +dn-15n-1)!), ta sẽ tính được L((dn5n +dn-15n-1 + dn-25n-2)!), v.v.. và cuối cùng sẽ tính được L(n!)

Tuy nhiên đầu tiên phải tính được L((dn5n)!), muốn vậy ta hãy hình dung:

$$n = d_{n+1}5^{n+1} + d_n 5^n + d_{n-1}5^{n-1} + \ldots + d_2 5^2 + d_1 5 + d_0 \text{ trong đó hệ số } d_{n+1} = 0$$

Do đó L((dn+15n+1)!)=0(một sự quy ước cho phù hợp với bảng □ để ta xét chuỗi có chữ số đầu tiên bằng 0). Ta dựa vào L((dn+15n+1)!)L((dn5n)!)

Ví dụ với n=1592:

Bước 1: đổi n ra cơ số 5, được $n = 2.5^4 + 2.5^3 + 3.5^2 + 3.5 + 2 = \overline{22332}\ (5)$

Bước 2: Xét dòng 0 (=4 mod 4), chuỗi có chữ số đầu là 0, tức là 06264.

Bước 3: chữ số đầu tiên của n (cơ số 5) là 2, do đó L((2.54)!) =2, là chữ số ở vị trí j=2.

Bước 4: Xét dòng 3 (=3 mod 4), chuỗi có chữ số đầu tiên là 2, tức là 26648

Bước 5: chữ số thứ hai của n (cơ số 5) là 2, do đó L((2.54+2.53)!) =6, là chữ số ở vị trí j=2

Bước 5: Xét dòng 2 (=2 mod 4), chuỗi có chữ số đầu tiên là 6, tức là 64244

Bước 6: chữ số thứ ba của n (cơ số 5) là 3, do đó L((2.54+2.53 +3.52)!)=4 là số ở vị trí j=3

…

Cuối cùng ta có:

L(1592!) = L((2.54+2.53 +3.52 + 3.5 + 2)!)=4

Chúng ta mô tả bảng trên bằng cách xây dựng một mảng Ă4,5,5). Trong đó chỉ số đầu tiên cho biết vị trí dòng (0,1,2,3); chỉ số thứ hai cho biết chữ số đầu tiên của chuỗi (0,2,4,6,8); chỉ số thứ ba cho biết vị trí j (0,1,2,3,4). Nếu gọi dk là chữ số thứ k trong hệ cơ số 5 của n, ta có thể mô tả thuật tóan ngắn gọn như sau:

Sn = Ăn mod 4, 0, dn)

Sn-1 = Ăn-1 mod 4, sn, dn-1)

…

S12, d1)

S01, d0)

Cuối cùng ta có S0 = L(n!)

Chương trình:

Dữ liệu: gồm số tự nhiên n đặt trong file NONZERO.INP

Quy định: Số có thể viết trên nhiều dòng (nếu qúa dài)

vd: NONZERO.INP

549485904089340974897979878989793489787935789973493489789438957897578
4789934768948459898973 48
Kết qủa: in ra màn hình L(n!)
Chương trình có thể tính L((1000!)!) = 6 trong thời gian chưa tới 0.5 giây

```
Program Least_Significant_Non_Zero_Digit_of_f_n;

(*==============KHAI=BAO==============*)
Const InputFile='NONZERO.INP';
Const Digits: array['0'..'9'] of byte=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
Table:array[0..3,0..4,0..4] Of Byte=
(((0,6,2,6,4), (2,2,4,2,8), (4,4,8,4,6), (6,6,2,6,4), (8,8,6,8,2)),
((0,2,8,8,4), (2,4,6,6,8), (4,8,2,2,6), (6,2,8,8,4), (8,6,4,4,2)),
((0,4,2,4,4), (2,8,4,8,8), (4,6,8,6,6), (6,4,2,4,4), (8,2,6,2,2)),
((0,8,8,2,4), (2,6,6,4,8), (4,2,2,8,6), (6,8,8,2,4), (8,4,4,6,2)));
Type BigInt= array[1..10000] of 0..9;
Var A: BigInt;
R5:BigInt;
N, N5:Word;
L: Byte;
(*===========DOC=DU=LIEU===========*)
Procedure Input;
Var I:Word;
F:Text;
Ch:Char;
B:BigInt;
Begin
Assign(F,InputFile);
Reset(F);
While Not EOF(F) Do
Begin
While not EOLN(F) Do
Begin
Read(F,Ch);
N:=N+1;
B[N]:=Digits[Ch];
End;
Readln(F);
End;
Close(F);
{Mang B chua cac chu so viet xuoi: vd so=1592, B=[1,5,9,2]}
```

```
{Mang A chua cac chu so viet nguoc: vd so=1592, A=[2,9,5,1]}
{Ta su dung mang A cho thuan tien ve sau}
For I:=1 To N Do A[I]:=B[N-I+1];

If (N=1) And ((A[1]=0) Or (A[1]=1)) Then {0! hay 1!}
Begin
Write(1); Readln; Halt(1);
End;
End;
(*==========DOI=SANG=CO=SO=5==========*)
Procedure Radix5;
Var I,J:Word; C:Byte;
Begin
C:=0;
N5:=0;
Repeat
N5:=N5+1;
R5[N5]:=A[N5] mod 5;
{Thay vi chia 5 ta nhan voi 2 roi chia cho 10 bang cach bo di chu so tan cung}
For I:=N5 To N Do
Begin
A[I]:=A[I]*2+C;
C:=A[I] div 10;
A[I]:=A[I] mod 10;
End;
If C=1 Then Begin
N:=N+1;
A[N]:=1;
C:=0;
End;
J:=J+1;
A[N5]:=0;
Until N5=N;
End;
(*==========TIM=CHU=SO=TAN=CUNG=KHAC=0=CUA=N!==========*)
Procedure Solve;
Var S:Byte;
I:Word;
Begin
S:=0;
For I:=N5 DownTo 1 Do
```

```
S:=Table[(I-1) Mod 4,S div 2,R5[I]];
L:=S;
End;
(*=========CHUONG=TRINH=CHINH=========*)
Begin
Input;
Radix5;
Solve;
Writeln(L);
Readln;
End.
```

12. Ứng dụng thuật toán tìm diện tích đa giác để lập trình giải một số bài toán về hình học

Tác giả: Phạm Mạnh Cường

Thuật toán hiệu quả tìm diện tích đa giác đã được nhiều tác giả nêu ra. Tuy nhiên trong bài viết này tôi chỉ xin nêu ra một số bài toán mà nếu ứng dụng thuật toán tìm diện tích đa giác để giải chúng sẽ rất hiệu quả. Trước hết, ta sẽ nhắc lại thuật toán này.

Bài toán: Tính diện tích đa giác (lồi hoặc lõm và không tự cắt) gồm n đỉnh A [1], A[2], …, A[n] (n>2).

Thuật toán: Ta có thể giải bài toán này bằng cách chia đa giác thành n − 2 tam giác rồi tính tổng diện tích của các tam giác ấy. Tuy nhiên phương pháp này phức tạp, ta làm cách khác như sau: chia đa giác thành các hình thang bằng cách chiếu các cạnh xuống trục hoành (hoặc trục tung). Hình thang được xác định bởi cạnh A[i]A[i+1] có diện tích là Abs(S) với:

S = (A[i].x - A [i+1].x).(A[i].y + A[i+1].y) /2

Sau khi gán đỉnh A[n+1] = A[1], ta tính diện tích toàn phần của đa giác như sau:

```
S: = 0;
For i: =1 to n do
S := S + (A[i].x - A [i+1].x).(A[i].y + A[i+1].y);
S: = (1/2) * Abs(S);
```

Sau đây là một số ví dụ áp dụng thuật toán vừa trình bày.

Bài 1. Kiểm tra tính lồi lõm của một đa giác.

Trong mặt phẳng toạ độ có n điểm có toạ độ tương ứng là (x1, y1), (x2, y2) ,…, (xn, yn). Biết rằng n điểm đã cho theo thứ tự tạo thành các đỉnh của một đa giác không tự cắt.

Yêu cầu: hãy kiểm tra đa giác lồi hay lõm.

Dữ liệu vào ghi trong file dagiac.inp: dòng đầu ghi một số n (n>2), n dòng tiếp theo,

trên dòng thứ i ghi hai số thực theo thứ tự là hoành độ và tung độ của đỉnh thứ i-1 của đa giác.
Dữ liệu ra ghi vào file dagiac.out gồm một số 0 hoặc 1, ghi 0 nếu đa giác lõm và ghi 1 nếu đa giác lồi.
Thuật toán: Gọi diện tích của đa giác là S0. Ta sẽ so sánh S0 với các Si, trong đó Si là diện tích của đa giác thu được từ đa giác ban đầu sau khi bỏ đi đỉnh thứ i (i=1, 2,…, n). Nếu $ i sao cho Si > S0 thì đa giác đã cho lõm (tại đỉnh thứ i), ngược lại thì đa giác đã cho lồi.
Toàn văn chương trình như dưới đây:

```
{Kiem tra mot da giac loi hay lom}
type
diem=record
x,y:real;
end;
var a:array [1..100] of diem;
n,vt:integer;
s0,s1:real;
procedure init;
var f:text;i:byte;
begin
assign(f,'dagiac.inp');
reset(f);
readln(f,n);
{while not eof(f) do}
for i:= 1 to n do
begin
read(f,a[i].x);
read(f,a[i].y);
end;
close(f);
a[n+1]:=a[1];
{Tinh dien tich da giac}
s0:=0;
for i:=1 to n do
s0:=s0+(a[i+1].x-a[i].x)*(a[i+1].y+a[i].y);
s0:=abs(s0)/2
end;
Procedure Inkq(i:byte);
var f:text;
begin
assign(f,'dagiac.out');
```

```
rewrite(f);
write(f,i);
close(f);
end;
Procedure resolve;
var b:array [1..50] of diem;
i,j,k:byte;
Begin
for i:=1 to n do
begin
for j:=1 to i-1 do b[j]:=a[j];
for j:=i+1 to n+1 do b[j-1]:=a[j];
s1:=0;
for j:=1 to n-1 do
s1:=s1+(b[j+1].x-b[j].x)*(b[j+1].y+b[j].y);
s1:=abs(s1)/2;
if s1>s0 then
begin
inkq(0);
halt
end;
end;
inkq(1);
End;
begin
init;
resolve;
end.
```

Bài 2. Chia đa giác (mô phỏng đề thi HS giỏi quốc gia lớp 12, bảng B, năm 1999-2000).

Trong mặt phẳng toạ độ có n điểm có toạ độ tương ứng là (x1, y1), (x2, y2),…,(xn, yn). Biết rằng n điểm đã cho theo thứ tự tạo thành các đỉnh của một đa giác không tự cắt. Yêu cầu: hãy chia đa giác đã cho thành hai đa giác bởi một cạnh nối hai đỉnh không kề nhau sao cho diện tích của chúng chênh lệch nhau ít nhất.

Dữ liệu vào ghi trong file dagiac.inp: dòng đầu ghi một số n (n>3), n dòng tiếp theo, trên dòng thứ i ghi hai số thực theo thứ tự là hoành độ và tung độ của đỉnh thứ i-1 của đa giác.

Dữ liệu ra ghi vào file dagiac.out gồm hai số nguyên dương chỉ số thứ tự của hai đỉnh được nối sao cho thoả mãn điều kiện đầu bài.

Thuật toán: Ta đưa ra thuật toán rất đơn giản như sau: thử tìm mọi cách chia và ghi nhận lại cách chia tốt nhất. Dễ thấy độ phức tạp của thuật toán trong trường hợp này

là O(n3).
Toàn văn chương trình như dưới đây:

```
Program vd;
type
diem=record
x,y:real;
end;
var a:array [1..100] of diem;
n,vt1,vt2:integer;
s0,s1,del:real;
procedure init;
var f:text;i:byte;
begin
assign(f,'dagiac.inp');
reset(f);
readln(f,n);
for i:= 1 to n do
begin
read(f,a[i].x);
read(f,a[i].y);
end;
close(f);
a[n+1]:=a[1];
s0:=0;
for i:=1 to n do
s0:=s0+(a[i+1].x-a[i].x)*(a[i+1].y+a[i].y);
s0:=abs(s0)/2;
del:=s0;
end;
Procedure resolve;
var b:array [1..50] of diem;
i,j,k,h:byte;
f:text;
Begin
for i:=1 to n-2 do
begin
for h:=i+2 to n-1 do
begin
k:=0;
for j:= i to h do
begin
```

```
inc(k);
b[k]:=a[j];
end;
b[k+1]:=b[1];
s1:=0;
for j:=1 to k do
s1:=s1+(b[j+1].x-b[j].x)*(b[j+1].y+b[j].y);
s1:=abs(s1)/2;
if del>abs(2*s1-s0) then
begin
vt1:=i;
vt2:=j;
del:=abs(2*s1-s0);
end;
end;
end;
assign(f,'dagiac.out');
rewrite(f);
write(f,vt1,' ',vt2);
close(f);
End;
begin
init;
resolve;
end.
```

Bài tập . Tứ giác bao n điểm (mô phỏng đề thi olimpic Sinh viên khối không chuyên 1999).

Cho n điểm có toạ độ (xi , yi), i = 1, 2 ,…, n với n > 4. Lập chương trình tìm 4 điểm sao cho 4 điểm này tạo thành một tứ giác có diện tích lớn nhất.

Bạn nào có nhu cầu tìm hiểu chương trình nguồn của bài tập trên hoặc muốn trao đổi thêm, có thể liên hệ với tác giả theo địa chỉ email: mhcuong2000@yahoo.com. Chúc các bạn có nhiều khám phá hay trong học tập lập trình nói chung cũng như lập trình với các bài toán về hình học nói riêng.

Phạm Mạnh Cường

13. Cặp ghép và luồng cực đại trên đồ thị hai phía

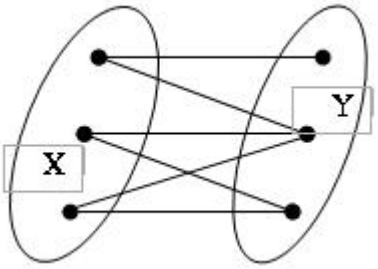
Tác giả: Nguyễn Thanh Tùng

Trong một số bài viết trước tôi có đề cập đến thuật toán 'cặp ghép' và 'luồng'. Sau khi bài viết được đăng, có bạn đọc phản hồi để nghị trình bày kĩ hơn về các thuật toán đó. Trong bài viết này tôi sẽ trình bày về thuật toán tìm cặp ghép cực đại và tìm luồng cực đại trên đồ thị hai phía.

1. Cặp ghép cực đại của đồ thị hai phía

1. Các định nghĩa

G được gọi là đồ thị hai phía nếu G=(X hợp Y,E); trong đó tập đỉnh V là hợp của 2 tập đỉnh con X,Y rời nhau và tập cạnh E chỉ chứa các cạnh có một đỉnh thuộc X và đỉnh còn lại thuộc Y. Như thế, về mặt trực quan đồ thị 2 phía có dạng:



Đồ thị hai phía có thể dùng biểu diễn nhiều mối quan hệ, thường là giữa 2 tập đối tượng khác nhau: chẳng hạn giữa công nhân và công việc hay vị trí trên dây truyền sản xuất, giáo viên và môn học,…

Thông thường đồ thị hai phía thường được biểu diễn bởi một ma trận quan hệ A (n,m). Khi đó:

- Tập X có n đỉnh, đánh số từ 1 đến n.
- Tập Y có m đỉnh, đánh số từ 1 đến m.
- Với 'i thuộc X, j thuộc Y nếu có cạnh (i,j) thì đặt A[i,j]=1; ngược lại thì A[i,j]=1.

Cặp ghép M là một tập con của E sao cho không có đỉnh nào của X hay Y thuộc nhiều hơn 1 phần tử của M (M chứa các cạnh đôi một không có đỉnh chung). Nói cách khác mỗi đỉnh thuộc X hay thuộc Y đều chỉ được tương ứng ('ghép') với nhiều nhất là một đỉnh khác thuộc tập kia, tất nhiên giữa cặp đỉnh đó phải có cạnh nối. Cặp ghép M được gọi là cặp ghép cực đại nếu nó có chứa nhiều cặp đỉnh nhất.

Khi X và Y có số phần tử bằng nhau và bằng N, ta có khái niệm cặp ghép đầy đủ là cặp ghép có đúng N phần tử. Nói một cách nôm na đồ thị có cặp ghép đầy đủ nếu mọi đỉnh i thuộc X đều được tương ứng với duy nhất một đỉnh j thuộc Y và ngược lại (tất nhiên giữa i và j phải có cạnh nối).

2. Ví dụ

Ví dụ một bài toán về tìm cặp ghép cực đại: Có N chàng trai và M cô gái. Mỗi chàng trai mến một hoặc có thể nhiều cô gái. Hãy tổ chức nhiều nhất các cặp nhảy, trong đó mỗi chàng trai sẽ nhảy với một cô gái mà mình mến.

Đồ thị 2 phía ở đây có tập đỉnh X là N chàng trai, tập đỉnh Y là M cô gái. Nếu chàng trai i mến cô gái j thì có cạnh (i,j). Việc tổ chức các cặp nhảy chính là tìm cặp ghép cực đại trên đồ thị.

Nếu N=M và yêu cầu của bài là tìm cách ghép để ai cũng có bạn nhảy thì bài toán

trở thành việc tìm cặp ghép đầy đủ.
3. Thuật toán và cài đặt
Với cặp ghép M, các đỉnh thuộc các cạnh trong M gọi là đỉnh đã ghép, các đỉnh còn lại gọi là đỉnh tự do.
Phương pháp xây dựng cặp ghép cực đại của đồ thị như sau:
- Khởi tạo M là rỗng.
- Với mỗi đỉnh x thuộc X, tìm cho nó một đường tăng cặp ghép. Nếu có thì tăng cặp ghép. Cho đến khi không tăng được nữa thì ta có cặp ghép cực đại.
Đường tăng cặp ghép xuất phát từ đỉnh x là một đường đi trên G có dạng như sau:
x , y1, x1, y2, x2, …, yk, xk, y.
Trong đó y là đỉnh tự do thuộc Y, các cạnh (xi,yi) thuộc M, i=1..k (k có thể bằng 0), tức là (xi,yi) là cặp đỉnh đã được ghép.
Với đường tăng cặp ghép trên ta tăng cặp ghép bằng cách: loại tất cả các cạnh (xi,yi) khỏi M và thay vào đó là các cạnh (x,y1), (x1,y2) … (xk,y). Dễ thấy nhờ đó M tăng thêm một phần tử.
Cài đặt:
Thuật toán tìm đường tăng cặp ghép cho đỉnh x dùng phương pháp BFS. Ta sử dụng một số cấu trúc dữ liệu sau:
queue: hàng đợi với các thao tác: put để thêm một phần tử vào queue, get để lấy ra một phần tử và hàm empty báo queue đã rỗng hay chưa.
mx, my là 2 mảng ghi nhận cặp ghép: mx[i], my[j] tương ứng là đỉnh ghép của i hay j. Bằng 0 tức là còn tự do, chưa được ghép với đỉnh nào.
tr là mảng lần vết, mô tả đỉnh trước của một đỉnh trên đường đi. tr[j]=0 tức là đỉnh j chưa thăm.

```
procedure find(x);
begin
queue :=  tr[j] := 0 với mọi j ;
put(x);
repeat
get(i);
for j := 1 to n do
if (tr[j] = 0) and (i,j) thuộc E then
tr[j] := i;
if my[j]=0 then begin
trace (j);
exit;
end
else put(my[j]);
until empty(queue);
end;
```

Thủ tục trace thực hiện việc lần vết để xây dựng đường tăng cặp ghép đồng thời tăng

cặp ghép.

```
procedure trace(j);
begin
repeat
i := tr[j];
t := mx[i];
mx[i] := j;
my[j] := i;
j := t;
until j=0;
end;
```

Và thuật toán tìm cặp ghép cực đại tiến hành như sau:

```
procedure match;
begin
for x := 1 to n do find(x);
end;
```

Đó là thuật toán kiểm tra tìm cặp ghép cực đại. Ta cũng có thể dùng nó để kiểm tra đồ thị có cặp ghép đầy đủ không (Nếu đỉnh nào cũng được ghép thì đồ thị có cặp ghép đầy đủ). Bạn đọc có thể tự lập trình giải bài toán ghép cặp nhảy ở trên.

Thuật toán cặp ghép áp dụng với kĩ thuật tìm kiếm nhị phân cho kết quả rất ấn tượng với nhiều bài toán. Các bạn có thể xem lại bài viết về Kĩ thuật tìm kiếm nhị phân của tôi trên số báo trước. Ngoài ra với cách nhìn nhận thích hợp ta có thể đưa một bài toán khó về một bài toán cặp ghép.

Tuy nhiên đôi khi trên đồ thị 2 phía nếu nhìn nhận bài toán theo kiểu cặp ghép thì độ phức tạp tính toán rất cao. Chẳng hạn ở bài xe bus, nếu số hành khách cỡ hàng ngàn và số chỗ trên xe cũng cỡ hàng trăm (điều đó đúng với thực tế hơn) thì giải bằng cặp ghép không tốt bằng giải bằng 'luồng'. Sau đây tôi sẽ trình bày về thuật toán luồng trên đồ thị hai phía.

II. Luồng cực đại của đồ thị hai phía

1. Các định nghĩa

Đồ thị hai phía G trong bài toán luồng vẫn có cấu trúc như đối với bài toán cặp ghép, nhưng có thêm trọng số đối với cả đỉnh và cạnh: mỗi đỉnh i thuộc X, j thuộc Y và cạnh (i,j) thuộc E đều có một trọng số kèm theo gọi là 'khả năng thông quá, được kí hiệu tương ứng là Cx(i), Cy(j) và C(i,j).

'Luồng' trên G bao gồm các giá trị kèm theo mỗi đỉnh i thuộc X, j thuộc Y và cạnh (i,j) thuộc E được kí hiệu tương ứng là Lx(i), Ly(j) và L(i,j) thoả mãn các điều kiện.

1. Điều kiện thông qua: '0 ≤ luồng ≤ khả năng thông quá.
2. Điều kiện cân bằng luồng: 'tổng luồng ra qua mỗi đỉnh bằng tổng luồng vào.

Các điều kiện trên có thể mô tả qua kí hiệu như sau:

$0 \le Lx(i) \le Cx(i)$; $0 \le Ly(i) \le Cy(i)$; $0 \le L(i,j) \le C(i,j)$ với mọi i,j.

$$Lx(i) = \sum_{j=1}^{m} L(i,j)\,;\; Ly(j) = \sum_{i=1}^{n} L(i,j)\,;$$

Ta sẽ hiểu rõ hơn những kí hiêu đó khi phân tích ví dụ sau: bài toán xe bus.

2. Ví dụ

Có K khách cần đến N khách sạn. Có M xe bus và xe bus thứ i có q[i] chỗ ngồi. Mỗi xe bus trên hành trình sẽ dừng trả khách tại một số khách sạn, và mỗi xe chạy một tuyến nên có thể đỗ tại các khách sạn không giống nhau. Hãy sắp xếp hành khách lên xe sao cho:

1. Số hành khách lên mỗi xe ≤ số ghế ngồi trên xe.
2. Mỗi hành khách đều ngồi lên xe có dừng tại khách sạn mình cần đến.

Phân tích:

Đồ thị hai phía được mô tả bởi:

- Tập X là tập N khách sạn. 'Khả năng thông quá của khách sạn i là: Cx(i) = số khách cần đến khách sạn i.
- Tập Y là tập M xe bus. 'Khả năng thông quá của xe bus j là số khách tối đa mà xe j có thể chở: Cy(j) = q[j].
- Nếu xe bus j dừng tại khách sạn i thì có cạnh (i,j), 'khả năng thông quá của cạnh này là C(i,j)=min(Cx(i),Cy(j)). Ngược lại ta đặt C(i,j)=0.
- Luồng trên cạnh (i,j): L(i,j) sẽ là số hành khách cần về khách sạn i được xếp lên xe j. Luồng ở đỉnh i thuộc X: Lx(i) là số hành khách được chở đến khách sạn i; luồng ở đỉnh j thuộc Y: Ly(j) là số khách được xếp lên xe j. Luồng có thể bằng 0.

Dễ dàng kiểm tra các điều kiện của luồng. (Ví dụ: số hành khách mà xe j chở = tổng số hành khách xe j chở đến từng khách sạn mà nó đi qua, tức là

$$Ly(j) = \sum_{i=1}^{n} L(i,j)\,,i=1..N).$$

).

Nếu luồng cực đại ở mỗi đỉnh bằng khả năng thông qua thì ta có cách sắp xếp khách thoả mãn 2 điều kiện đã cho. Chú ý là thứ tự khách không quan trọng (xếp lên xe nào cũng được, miễn là đi đến nơi cần đến). Chẳng hạn nếu L(1,1) = 3 thì chọn 3 khách chưa được xếp, cần về khách sạn 1 cho lên xe 1. Nếu còn 3 người nữa và có L(1,2) = 1 và L(1,3) = 2 thì xếp 1 người lên xe 2 và 2 người lên xe 3, người nào lên xe nào cũng được. Chú ý là $Cx(i)=Lx(i)=\sum_{j=1}^{m} L(i,j)\,,\,(j=1..M)$ nên luôn có đủ chỗ trên các xe, và ai cũng được lên xe.

Vậy, chỉ còn vấn đề tìm được luồng cực đại thôi. Thuật toán như sau:

3. Thuật toán

Phương pháp xây dựng cặp ghép cực đại của đồ thị như sau:

1. Khởi tạo luồng bằng 0.
2. Tìm cho nó một đường tăng luồng. Nếu có thì tăng. Nếu không tìm được một đường tăng luồng nào nữa thì ta được luồng cực đại.

Đường tăng luồng là một đường đi trên đồ thị hai phía có hướng Gx, có tập đỉnh là

X hợp Y, tập cạnh được xây dựng như sau:
- Nếu L(i,j)
- Nếu L(i,j)>0 thì có cung (j,i) từ Y sang X, được gọi là cung ngược.
Đường tăng luồng tìm bằng thuật toán BFS trên Gx, xuất phát từ một đỉnh x thuộc X mà Lx(x)
Giả sử các đỉnh trên đường đi đó như sau:
x j1 i1 y2 i2 … yk xk y.
Thế thì ta thấy:
Lx(x)
Để đảm bảo điều kiện cân bằng luồng tại x, ta phải tăng luồng trên cạnh (x,j1) lên một lượng d (chú ý đó là một cung xuôi của Gx). Để làm được điều đó thì d≤C(x,j1)-L(x,j1).
Do ta không được thay đổi luồng tại đỉnh j1 nên để đảm bảo điều kiện kiện cân bằng luồng tại j1 thì ta phải giảm luồng trên cạnh (i1,j1) là d, và như vậy d≤L(i1,j1). Chú ý (j1,i1) là cung ngược.
Suy luận tương tự ta thấy:
- Luồng trên cạnh chứa cung xuôi được tăng một lượng d.
- Luồng trên cạnh chứa cung ngược bị giảm một lượng d.
- Luồng tại x và y tăng một lượng d, tại các đỉnh khác thì không đổi.
Ta có d là giá trị lớn nhất có thể chọn mà vẫn đảm bảo điều kiện thông qua thì:
d = min(Cx(x)-Lx(x), Cy(y)-Ly(y),
C(i,j)-L(i,j) với (i,j) là cung xuôi,
L(i,j) với (j,i) là cung ngược).
Như vậy mỗi lần tìm được một đường tăng luồng thì tổng luồng trên toàn bộ đồ thị được tăng thêm một lượng d. Khi không tìm được đường tăng luồng nữa thì ta có luồng cực đại.
Sau đây là chương trình cài đặt thuật toán tìm luồng cực đại trên đồ thị hai phía. Các bạn hãy tập đọc để tìm hiểu cách cài đặt và rèn luyện kĩ năng đọc hiểu chương trình của người khác..

```
program bflow;
const
inp = 'bflow.inp';
out = 'bflow.out';
max = 100;
type
mang1 = array[1..2*max] of integer;
mang2 = array[1..max,1..max] of integer;
var
N,M,d:integer;
L,C:mang2;
Lx, Ly, Cx, Cy, Tx, Ty, Dt:mang1;
```

```
found:boolean;
(*********************)
procedure nhap;
var
i,j:integer;
f:text;
begin
assign(f, inp);
reset(f);
readln(f, N, M);
for i:=1 to N do read(f,Cx[i]);
readln(f);
for j:=1 to M do read(f,Cy[j]);
readln(f);
for i:=1 to N do begin
for j:=1 to M do
read(f,C[i,j]);
readln(f);
end;
close(f);
end;
(*********************)
function min(x,y:integer): integer;
begin
if x end;
procedure trace(j:integer);
var
i, t:integer;
begin
found:=true; t:=2;
Ly[j]:=Ly[j]+d;
repeat
if t = 2 then begin {tu Y ve X}
i:=Ty[j];
L[i,j]:=L[i,j]+d;
t:=1;
end
else begin {tu X ve Y}
i:=Tx[j];
if i = -1 then break;
L[j,i]:=L[j,i] - d;
```

```
t:=2;
end;
j:=i;
until false;
Lx[j]:=Lx[j]+d;
end;

procedure find(i:integer);
var
f,r,j,t:integer;
Qu,Qt,Qd:mang1;
begin
fillchar(Qu,sizeof(Tx),0);
fillchar(Qd,sizeof(Tx),0);
fillchar(Qt,sizeof(Tx),0);
fillchar(Tx,sizeof(Tx),0);
fillchar(Ty,sizeof(Ty),0);

f:=1; r:=1;
Qu[r]:=i; Tx[i]:=-1;
Qt[r]:=1; Qd[r]:=Cx[i] - Lx[i];
repeat
i:=Qu[f]; t:=Qt[f]; d:=Qd[f];
if t = 1 then begin {tim tu X sang Y}
for j:=1 to M do begin
if (L[i,j]
Ty[j]:=i; r:=r+1;
Qu[r]:=j; Qt[r]:=2;
Qd[r]:=min(d, C[i,j]-L[i,j]);
if Ly[j]
d:=min(Qd[r], Cy[j]-Ly[j]);
trace(j); exit;
end;
end
end
end
else begin {tim tu Y sang X}
j:=i;
for i:=1 to N do
if (L[i,j]>0) and (Tx[i] = 0) then begin
Tx[i]:=j; r:=r+1;
```

```
Qu[r]:=i; Qt[r]:=1;
Qd[r]:=min(d, L[i,j]);
end;
end;
f:=f+1;
until f > r;
end;
(**********************)
procedure xuly;
var
i:integer;
begin
repeat
found:=false;
for i:=1 to N do
if Lx[i] < Cx[i] then find(i);
until not found;
end;
(**********************)
procedure inkq;
var
f:text;
i,j:integer;
begin
assign(f, out);
rewrite(f);
for i:=1 to N do write(f,' ',Lx[i]);
writeln(f);
for j:=1 to M do write(f,' ',Ly[j]);
writeln(f);
for i:=1 to N do begin
for
```

14. **Một số kỹ thuật tinh chế mã**

Tác giả: Đặng Quang Hưng

Một phiên bản tốt hơn của tìm nhị phân
Trong số báo tháng trước tôi đã giới thiệu cho các bạn chương Tinh chế mã của tác giả Jon Bentley trong cuốn 'Programming Pearls' (Những viên ngọc trong kỹ thuật lập trình). Với số này chúng ta sẽ cùng nhau tìm hiểu một trong những ví dụ tinh tế

nhất của kỹ thuật tinh chế mã, đó là kỹ thuật tăng tốc độ cho chương trình tìm nhị phân.
Chương trình tìm nhị phân nguyên thủy

```
L := 1; R := N;
WHILE (L<=R) DO
BEGIN
M:=(L+R) DIV 2;
IF (A[M] < X) THEN
L:=M+1;
ELSE
IF (A[M] > X) THEN
R:=M-1;
ELSE
BREAK;
END;
IF (L<=R) HEN
P:=M;
ELSE
P := 0;
```

Chương trình tìm nhị phân tổng quát này đã rất hiệu quả, hiệu quả đến nỗi mọi nỗ lực tinh chế mã để nhằm tăng tốc nó đều không mang lại một sự khác biệt đáng kể nào! Như vậy thì tại sao chúng ta lại đề cập đến vấn đề cải thiện chương trình này ở đây? Chúng ta sẽ không tìm cách tinh chế chương trình tìm nhị phân tổng quát với giá trị N không giới hạn, mà chúng ta sẽ tinh chế chương trình tìm nhị phân với giá trị N = 1000.
Về mặt khoa học, bài toán với giá trị N = 1000 có thể được xem là đặc biệt nhưng trong thực tế chúng lại thường xảy ra. Chẳng hạn: tìm kiếm một mã số nhân viên trong một danh sách mã số nhân viên được sắp thứ tự, trong đó, số lượng nhân viên của một công ty trong đa số trường hợp thường nhỏ hơn 1000!
Phiên bản thứ nhất
ý tưởng cải tiến đầu tiên là giảm bớt số phép so sánh giữa X (phần tử cần tìm) với phần tử giữa A[M]. Trong chương trình tổng quát ở trên, ta nhận thấy rằng trong một vòng lặp, đôi lúc cần phải thực hiện hai phép so sánh. Chương trình dưới đây sẽ cải thiện được điều này.

```
L := 0; R := N+1;
WHILE (L+1 <> N) DO
BEGIN
M:=(L+R) DIV 2;
IF (A[M] < X) THEN
L:=M;
ELSE { X <= A[M] }
```

```
R:=M;
END;
P:=U;
IF (P>N) OR (X[P] <> T) THEN
P:=0;
```

Điểm đầu tiên cần lưu ý là chương trình có dùng thêm hai phần tử 'cầm canh' ở đầu và cuối mảng.

Quan sát đoạn chương trình ở trên, bạn cũng có thể dễ nhận thấy rằng trong mỗi lần lặp, chương trình chỉ phải thực hiện một phép so sánh duy nhất là A[M] < X.

Điểm khác biệt đầu tiên giữa hai chương trình là việc 'thu hẹp' khoảng tìm kiếm. Trong chương trình nguyên thủy, khoảng tìm kiếm mới không bao gồm phần tử giữa (A[M]). Còn trong chương trình ở trên, khoảng tìm kiếm mới vẫn bao gồm phần tử giữa.

Điểm khác biệt kế tiếp nằm ở việc xử lý trường hợp X[M] = A: chương trình tổng quát sẽ lập tức thoát khỏi vòng lặp (câu lệnh IF (A[M] = X) THEN BREAK;), còn chương trình mới sẽ không thoát khỏi vòng lặp mà vẫn tiếp tục thực hiện (câu lệnh ELSE R := M), nó chỉ thoát khi khoảng tìm kiếm đã được thu hẹp lại chỉ còn 2 phần tử kế nhau (điều kiện L+1 <> U của vòng lặp WHILE)

Để dễ hiểu hơn, mời các bạn quan sát ví dụ sau:

Tìm X = 8 trong mảng A: 1 3 5 7 8 10 20 22 23

Đối với chương trình nguyên thủy, ngay từ bước lặp đầu tiên (M= (1+9)/2 = 5), điều kiện A[M] = X đã được thỏa mãn, do đó, quá trình tìm kiếm kết thúc.

Đối với chương trình tiếp theo, trong vòng lặp đầu tiên (M = (0+10)/2 = 5), câu lệnh ELSE được thỏa mãn nên R được gán trị bằng 5. Vòng lặp tiếp tục với M = (0+5)/2 = 2, sau đó L được gán trị bằng 2. Bước kế tiếp, M = (2+5)/2 = 3, L sẽ có trị bằng 3. Cuối cùng, M=(3+5)/2= 4 , L có trị bằng 4. Lúc này, điều kiện L+1 <> R không còn thỏa mãn nữa nên quá trình tìm kiếm kết thúc.

Như vậy qua hoạt động của chương trình trên, ta có thể nói rằng nếu X có xuất hiện trong mảng A thì sau khi kết thúc vòng lặp thì A[R] = X. Nếu A[R] <> X thì suy ra X không có trong A. Cũng vậy, nếu X có trong A thì chắc chắn R phải nằm trong khoảng 1..N, ngược lại thì suy ra được X không có trong A.

Đến đây, ta thấy rằng, rằng chương trình mới không chắc là tốt hơn chương trình tổng quát trong mọi trường hợp. Chương trình mới, tuy giảm được số phép so sánh trong một bước lặp nhưng nó lại không dừng lại khi đã xác định được vị trí X trong mảng A. Kết quả là nó lại tiếp tục 'phung phí' các phép so sánh A[M] < X trong những lần lặp sau.

Vậy thì điểm tốt hơn của chương trình sau nằm ở điểm nào?

Phiên bản thứ hai

Version tiếp theo của chương trình sẽ dùng cách biểu diễn khác của giá trị L và R. Thay vì biểu diễn L và R gồm hai giá trị riêng rẽ, ta biểu diễn số R bằng một cận dưới và một gia số I nào đó. Nghĩa là R = L + I. Chúng ta sửa lại chương trình một tí

cho phù hợp với kiểu biểu diễn mới, thay thế các vị trí xuất hiện của R bằng biểu thức L+I và rút gọn biểu thức cuối cùng lại.

```
L := 0; I := N+1;
WHILE (I <> 1) DO
{ Điều diện I<>1 tương đương với L+1 <> R)
BEGIN
{(L+R) DIV 2 = (L + L + I) DIV 2 = (2L + I ) DIV 2
= L + (I DIV 2) }
I := I DIV 2; {Tương đương với M := (L+R) DIV 2 }
IF (A[L + I] < X) THEN
L:=L+I;
{ Đoạn lệnh này được bỏ đi vì giá trị I và L đã xác định R}
ELSE { X <= A[M] }
R:=M; }
END;
P:=L+1; { Tương đương với P := R }
IF (P>N) OR (A[P] <> X) THEN
P:=0;
```

Chương trình này lại cho ta một cách nhìn mới. Trong hai chương trình đầu, L và R đại diện cho vùng tìm kiếm còn lại. Còn trong chương trình trên, vùng tìm kiếm được đại diện bởi L là vị trí đầu khoảng tìm kiếm và I là độ dài (hay số phần tử) của khoảng tìm kiếm còn lại. Nếu độ dài này bằng 1 thì quá trình tìm kiếm kết thúc. Trong mỗi vòng lặp, độ dài I luôn được chia đôi và L được cập nhật hay không tùy thuộc vào phần tử giữa. Đoạn chương trình này không có gì khá hơn đoạn chương trình đầu tiên, chỉ khác là ta không cần phải quan tâm đến việc cập nhật lại giá trị của R (vì sự thay đổi của I và L cũng đã xác định R}. Nhờ đó, lệnh IF của chúng ta đã bỏ bớt được nhánh ELSE.

Đến đây, ta nhận xét rằng, nếu số phần tử của mảng A là một hằng số và là một lũy thừa của 2 thì các giá trị của I qua các vòng lặp cũng sẽ xác định.và phép chia nguyên luôn cho kết quả có số dư bằng 0. Chẳng hạn với trường hợp N = 1024 thì các giá trị của I qua các lần lặp sẽ lần lượt là: 512, 256, 128, 64, 32, 16, 8, 4, 2, 1. Với những giá trị xác định như thế, ta sẽ có thể bỏ đi phép toán DIV 2 và thay thế vòng lặp WHILE bằng một dãy lệnh IF tương ứng như chương trình sau

```
L:=0;
IF (A[512] < X) THEN L:=L + 512; { I = 512 }
IF (A[L+ 256] < X) THEN L:=L + 256; { I = 256 }
IF (A[L+ 128] < X) THEN L:=L + 128; { I = 128 }
IF (A[L+ 64] < X) THEN L:=L + 64; { I = 64 }
IF (A[L+ 32] < X) THEN L:=L + 32; { I = 32 }
IF (A[L+ 16] < X) THEN L:=L + 16; { I = 16 }
IF (A[L+ 8] < X) THEN L:=L + 8; { I = 8 }
```

```
IF (A[L+ 4] < X) THEN L:=L + 4; { I = 4 }
IF (A[L+ 2] < X) THEN L:=L + 2; { I = 2 }
IF (A[L+ 1] < X) THEN L:=L + 1; { I = 1 }
P:=L+1;
IF (P>1024) OR (A[P]<>X) THEN P :=0;
```

Vấn đề cuối cùng của chúng ta là xử lý trường hợp số phần tử N không phải là một lũy thừa của 2. Chẳng hạn N = 1000. ý tưởng rất đơn giản, nếu N không phải là một lũy thừa của 2 thì ta cố gắng đưa nó về lũy thừa của 2. Có hai cách để làm điều này. Cách đầu tiên là thêm các phần tử phụ vào các vị trí còn trống trong A sao cho các phần tử có giá trị tăng và luôn lớn hơn A[1000]. Cách thứ hai là chọn lựa khoảng tìm kiếm ở bước lặp thứ hai sao cho khoảng tìm kiếm này có độ dài bằng 512. Ban đầu, ta cũng so sánh X với phần tử A[512], nếu X < A[512] ta chọn khoảng tìm kiếm từ [0..512], ngược lại ta chọn khoảng tìm kiếm từ [489…1001].
Phương pháp này được thể hiện trong dòng lệnh IF đầu tiên

```
L:=0;
IF A[512] < X THEN L := 489; { = 1000 + 1 − 512 }
…
IF (P>1000) OR (A[P]<>X) THEN P:=0;
```

Với sửa đổi này, ta có được đoạn chương trình tìm nhị phân đã được tối ưu. Vậy thì hiệu quả của chương trình này so với chương trình ban đầu như thế nào? Nhận xét đầu tiên là một lần tìm kiếm sẽ luôn luôn dùng 10 phép so sánh A[L + …] < X và ít hơn 10 lệnh gán L:=L+….Điểm quan trọng nhất là đoạn chương trình này hoàn toàn không dùng phép chia và không có vòng lặp. Chúng tôi đã đo đạc và nhận thấy rằng chương trình trên thi hành nhanh gấp từ 2 cho đến 4.5 lần chương trình tìm nhị phân nguyên thuỷ!. Nếu không tận mắt chứng kiến kết quả này qua các công cụ đo đạc đáng tin cậy thì tôi và cả bạn đều cho rằng chương trình ở trên là một phép "ma thuật"!

Những lời cuối

Tuy chúng ta đã chứng kiến nhiều ví dụ rất đẹp mắt của kỹ thuật tinh chế mã nhưng một nguyên lý mà bạn vẫn cần phải nhớ là: đừng thực hiện việc tinh chế mã thường xuyên. Sau đây là một vài thông tin để bạn suy nghĩ:

Tính hiệu quả: bên cạnh tốc độ, nhiều tính chất khác của chương trình cũng rất quan trọng, nếu không muốn nói là quan trọng hơn. Don Knuth đã nhận xét rằng việc tối ưu hóa không đúng lúc là nguồn gốc của nhiều tai họa trong lập trình; nó có thể tổn thương tính đúng đắn của chương trình, làm cho chương trình khó hoạt động và khó bảo trì. (chẳng hạn việc tinh chế mã của chương trình nhị phân ở trên đã biến đổi hoàn toàn chương trình tìm kiếm nhị phân và làm cho chương trình trở nên khó hiểu đến mức bí ẩn và có lẽ bạn cũng sẽ lúng túng nếu không có giải thích chi tiết như trên!). Bạn chỉ nên quan tâm đến tốc độ khi và chỉ khi thực sự nó đã thực sự ảnh hưởng đến tính hữu ích của chương trình.

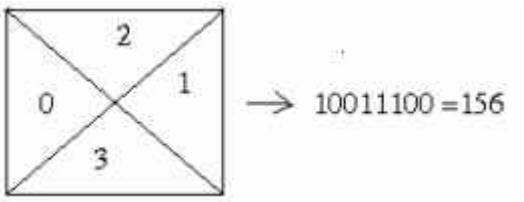
Thao tác profiling: Khi vấn đề tối ưu tốc độ đã được đặt ra, thao tác đầu tiên bạn cần

làm là thực hiện thao tác profile hệ thống để tìm ra những điểm nóng của hệ thống (là những điểm mà tại đó thời gian xử lý hay thi hành chương trình bị tiêu tốn phần lớn vào đó). Sau đó, tập trung cải tiến những vị trí này, còn những chỗ khác thì 'nếu nó không hỏng thì đừng chạm vào!'.
Các mức thiết kế: có nhiều phương pháp khác nhau để giải quyết bài toán tốc độ (thuật toán, nén không gian,…, bạn hãy thử tất cả các phương pháp khác trước, nếu tất cả đều không thành công khi đó mới nghĩ đến kỹ thuật tinh chế mã.
Qua các ví dụ đã trình bày, chúng ta rút ra một số quy tắc chính trong việc tinh chế mã:
Chương trình vẽ hình của Wan Wyk. Chiến lược chung là khai thác các trường hợp thông dụng. Điều đó dẫn đến việc caching các loại bản ghi thông dụng nhất.
Bài toán 1: phân loại ký tự. Trong lời giải của bài toán này, chúng ta sử dụng một mảng được đánh chỉ số bằng các ký tự, quy tắc rút ra từ bài toán này là: 'tính trước các hàm logic'
Bài toán 2: đếm số bit. Chúng ta đã sử dụng một mảng để lưu số các bit trong một con số. Quy tắc là 'Lưu trữ các kết quả đã được tính trước'.
Tìm nhị phân: Việc kết hợp các phép thử đã giảm đi một nửa số các lần so sánh mảng trong mỗi vòng lặp, việc khai thác tương đương đại số làm thay đổi cách biểu diễn cận dưới và cận trên thành cận dưới và độ dài, việc thay thế vòng lặp bằng dãy các lệnh tương đương giúp ta bỏ được nhiều thao tác không cần thiết (phép toán DIV).
Chúng ta đã xem qua kỹ thuật tinh chế mã để tăng tốc độ chương trình. Tuy nhiên, tinh chế mã còn có thể dùng vào nhiều mục đích khác, chẳng hạn như giảm thao tác phân trang bộ nhớ hoặc để tăng tỷ số cache hit trong bộ nhớ cache, ….Ngoài ra, nó còn có thể được sử dụng để làm giảm kích thước chương trình.

15. Cặp ghép với số đỉnh rất lớn

Tác giả: Võ Xuân Sơn

Trong số 26 (11/2001), tác giả Lê Văn Chương đã giới thiệu cho chúng ta thuật toán Kuhn − Munkres giải bài toán tìm cặp ghép có tổng trọng số lớn nhất, nhỏ nhất nhưng với số đỉnh rất bé. Sau đây, dựa theo thuật toán Kuhn − Munkres tôi trình bày bài toán cặp ghép với số đỉnh rất lớn (nhỏ hơn 5000 đỉnh).
Ví dụ minh hoạ.
Bài toán xếp hình:
1 trò chơi xếp hình trên máy tính từ những ô hình vuông kích thước giống nhau. Mỗi ô hình vuông này được chia thành 4 tam giác, mỗi tam giác có thể nhận 1 trong 4màu đỏ, xanh, vàng, trắng. (quy ước: 0 − đỏ; 1 − xanh; 2 − vàng; 3− trắng)
như hình vẽ:



Người chơi cần dùng chuột kéo những ô vuông đã cho vào một lưới ô vuông M hàng, N cột đến khi đầy lưới sau đó có thể nhấn chuột để quay ô. Mỗi lần nhấn chuột ô quay 1 góc 900 theo chiều kim đồng hồ. Giả thiết mỗi lần kéo một ô vuông vào một vị trí của lưới mất 1s và mỗi lần nhấn chuột mất 1s. Bằng những thao tác như vậy, người chơi cần tạo ra một hình giống như hình mẫu cho trước với thời gian nhanh nhất.

Các ô vuông đã cho được đánh số từ 1. Trạng thái của ô vuông được mã hoá bằng 1 byte (8 bit) như sau: hai bit trái nhất mô tả giá trị mầu của tam giác phía Bắc, lần lượt các nhóm 2 bit tiếp theo mô tả giá trị mầu của các tam giác phía Đông, Nam, Tây của ô vuông.

Dữ liệu vào: Xephinh.inp (m, n ≤ 50)

Dữ liệu ra: Xephinh.out

Nếu không có cách xếp thì ghi -1

Ví dụ:

xephinh.inp
2 3 (m, n) 183 237 183 108 27 198 (trạng thái của n*m vùng) 198 123 108 237 222 177 Hình mẫu cần xếp

xephinh.out
5 2 4 1 3 6 số hiệu của ô vùng được xếp vào vị trí tương ứng của lưới 2 2 1 2 4 2 số lần di chuyển và nhấn chuột

Tư tưởng thuật toán:

Đầu tiên chúng ta tìm các trạng thái có thể có được khi xoay các ô vuông theo chiều kim đồng hồ. Sau đó tiến hành ghép cặp các trạng thái đặt vào các ô của hình mẫu sao cho cách ghép đó là ghép được và thoả mãn với hình mẫu, Từ đó tiến hành lần lượt các bước như trong thuật toán Kuhn − Munkres với tổng thời gian (trọng số)

nhỏ nhất.
Các bạn có thể tham khảo thuật toán này qua chương trình sau:

```
Program xephinh;
const inp = ′xephinh.inp′;
out = ′xephinh.out′;
Ln = 2500;
Var c: array [1..ln, 0..3] of byte;
mau, tt: array [1..ln] of byte;
px, py: array [1..ln] of integer;
F, q, qu: array [1..ln*2] of integer;
m, n, mn, x, y, i, u, z: integer;
start: longint; g: text;
procedure input;
begin
assign (g,inp) ; reset (g);
readln (g,m,n); mn:= m*n;
for x:=1 to mn do read (g,tt[x]);
readln (g);
for x:=1 to mn do readln(g, mau[x]);
close (g);
start:= meml [0: $46c];
End;
procedure state;
var b1, b2: byte;
begin
for x:=1 to mn do
for y:=0 to 3 do
begin
c[x, y]:=tt[x];
b1:= tt[x] shr 2;
b2:= byte (Tt[x] shl 6);
Tt[x]:= b1 or b2;
end;
end;
procedure khoitao;
var ok: boolean;
begin
fillchar (f, sizeof (f), 0);
fillchar (px, sizeof (px), 0);
fillchar (py, sizeof (py), 0);
for x:=1 to mn do
```

```
begin
ok:= false;
for i:= 0 to 3 do
begin
for y:= 1 to mn do
If (py[y]=0) and (c[x,i] = mau [y]) then
begin
px [x]:= y;
py[y]:= x;
ok:=true;
break;
end;
if ok then
begin
F[x]:=i;
break;
end;
end;
end;
end;
Function liberty: boolean;
begin
for x:=1 to mn do;
if px[x]= 0 then
begin
liberty:= true;
u:=x; exit;
end;
liberty:= false;
end;
Function Findpath: boolean;
Var dau, cuoi, v,w,n: integer;
begin
fillchar (q, sizeof (q),0);
dau:=1; cuoi:= 1; qu[1]:=u; q[u]:=u;
while dau <= cuoi do
begin
v:= qu[dau]; inc(dau);
if v<= mn then;
for x:=0 to 3 do
for u:= mn + 1 to mn * 2 do
```

```
if c[v, x] =mau [u-mn] then
if (F[v] + F[u]= x) and (q[w] = 0) then
begin q[u]:= v;
inc (cuoi);
qu [cuoi]:= w;
end;
if v > mn then
if py[v - mn] = 0 then
begin
findpath:=true;
z:= v ;
exit;
end
else begin
u:=py[v-mn];
inc (cuoi);
qu [cuoi]:= w;
q[u]:= v;
end;
end;
findpath:= false;
end;
procedure tangcg;
var thuocy: boolean;
begin
y:= y ; thuocy:= true;
while y<> u do
begin x:= q[y];
if thuocy then
begin
px [x]:= y - mn;
py [y-mn]:= x;
end;
y:= x;
thuocy:= not thuocy;
end;
end;
procedure suanhan;
var d, h: integer;
begin
d:= maxint;
```

```
for x:= 1 to mn do
if q[x] > 0 then
for y:= mn + 1 to mn * 2 do
if q[y] = 0 then
for i:= 0 to 3 do
if c[x,i] =mau [y-mn] then
begin
h:= i - f[x] - f[y];
if d>h then d:= h;
break;
end;
for x:=1 to mn do
if q[x] > 0 then f[x]:= f[x] + d;
for y:= mn+1 to mn * 2 do
if q[y] > 0 then f[y]:= f[y] - d;
end;
procedure Virus;
begin
assign (g,out) ; rewrite (g);
write (g, -1); close (g); halt;
end;
procedure process;
begin
while liberty do
begin
while not findpath do
begin
if (meml [0: $46c] - start) / 18.22 > 1 then virus;
suanhan;
end;
tangcg;
end;
end;
procedure output;
begin
assign (g,out); rewrite (g);
for x:=1 to mn do
begin
write (g,py[x],' ');
if x mod n = 0 then writeln (g);
end;
```

```
for x:=1 to mn do
begin
for y:= 0 to 3 do
if c[py[x], y] = mau[x] then
begin
write (g, y+1, ' ');
break;
end;
if x mod n =0 then writeln(g);
end;
close (g);
end;
begin
input;
state;
khoitao;
process;
output;
end.
```

16. Tản man về số nguyên tố lớn nhất & thuật toán đa thức xác định số nguyên tố

Tác giả: Bùi Việt Hà

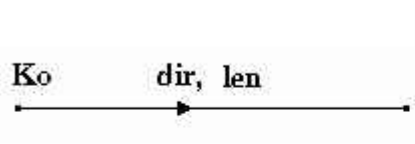
Trong số báo trước (tháng 4-2003) anh Vũ Đình Hòa đã có bài viết về tác giả và thuật toán đa thức xác định số nguyên tố. Chúng tôi xin cung cấp thêm một số thông tin sau:
Tác giả chính của bài báo là giáo sư Manindra Agrawal tại trường đại học Công nghệ Kanpur, ấn độ cùng với 2 học trò của mình là Neeraj Kayal và Nitin Saxena. Trước ông đã có nhiều tác giả khác đã đưa ra nhiều thuật toán xác định số nguyên tố với thời gian đa thức, tuy nhiên tất cả các thuật toán này đều hoặc phải dựa trên các giả thuyết chưa được chứng minh, hoặc chỉ là xấp xỉ chính xác, hoặc tương đối phức tạp. Còn với Manindra Agrawal thì khác hẳn: thuật toán rất đơn giản (đến không ngờ), không hề dựa trên một giả thuyết chưa được chứng minh nào và là một thuật toán xác định chính xác.
Với thành tích này tháng 10-2002, giáo sư Manindra Agrawal đã nhận được giải thưởng toán học Clay của Viện hàn lâm khoa họa Mỹ, một trong những giải thưởng danh giá nhất về Toán học của nước Mỹ. Cùng nhận giải năm nay với Manindra Agrawal còn có nhà toán học trẻ Vladimir Voedoevsky, người đã nhận giải thưởng Fields năm 2002 vừa qua. (Fields là giải thưởng lớn quốc tế lớn nhất dành cho các

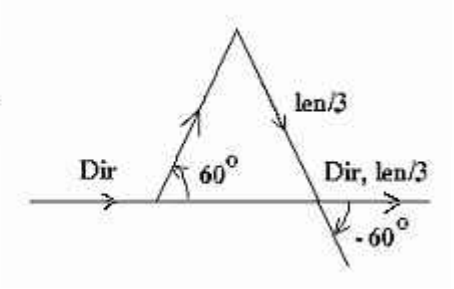
nhà Toán học trẻ dưới 40 tuổi, có thể coi như một thay thế của giải thưởng Nobel, được trao 4 năm một lần).
Tuy nhiên nhóm tác giả cũng thừa nhận rằng thuật toán của họ vẫn chưa có tính hiện thực cao vì bậc đa thức của thuật toán là 11, trong trường hợp tốt nhất chỉ có thể hạ xuống còn 6, là một số quá lớn. Để có thể áp dụng thành công trong thực tế, các tác giả sẽ còn có nhiệm vụ cải tiến để bậc đa thức giảm xuống dưới 4. Hy vọng với thuật toán mới này con người sẽ tiến thêm những bước mới trong việc chinh phục thế giới kỳ lạ của các số nguyên tố.
Tôi nhớ rằng trong một số tạp chí đầu tiên, tháng 11/1999, Tin học & Nhà trường đã có bài viết về số nguyên tố lớn nhất do con người tìm ra (bằng máy tính). Tại thời điểm tháng 11/1999, số nguyên tố lớn nhất đã tìm ra được là số Mersene M = 26972593 -1. Số này có 2098960 chữ số do nhóm tác giả Nayan Hajratwala tìm ra tháng 1-6-1999 cùng với sự hỗ trợ của dự án GIMS với hàng ngàn máy chủ cùng tính trên Internet. Đây là kỷ lục 1 triệu chữ số số nguyên tố đã được tìm ra. Trong bài báo trên cũng đưa ra thông tin rằng quĩ FFF đã đưa ra giải thưởng 100.000 USD cho người đầu tiên tìm ra được số nguyên tố với 10 triệu chữ số.
Vậy từ đó đến nay kỷ lục trên đã bị phá vỡ chưa? Đã có thêm số nguyên tố mới nào được đưa vào danh sách số nguyên tố lớn nhất do con người tìm ra chưa. Chúng tôi xin thông tin ngay cho bạn đọc: ngày 14-11-2001, nhóm tác giả Michael Cameron, George Woltman, Scott Kurowski đã tìm ra số nguyên tố lớn nhất là 213466917-1 với 4053946 chữ số. Đây là số Mersene nguyên tố thứ 39 đã tìm ra cho đến nay.
Cuối cùng tôi cung cấp cho các bạn địa chỉ một Web site rất hay, tập trung hầu như toàn bộ các thông tin về số nguyên tố: các thuật toán, định lý, ngân hàng dữ liệu số nguyên tố đã tìm được, các dự án, kỷ lục về số nguyên tố, trò chơi giải trí với số nguyên tố. Đó là địa chỉ http ://www.utm.edu/research/primes/

17. Cung được đĩnh nghĩa đệ quy, quy tắc và cách vẽ

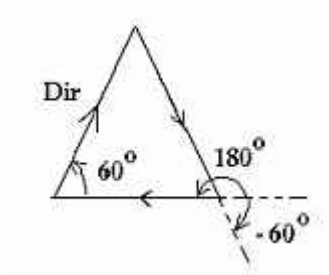
Tác giả: Phạn Xuân Bách

Cung là một đối tượng cơ bản trong đồ học vi tính. Cung cho ta những hình ảnh đơn giản nhưng khá bắt mắt (trừ những khung phức tạp như Fratal). Các cung thường có quy tắc và được định nghĩa dưới dạng tham số hay đệ quy.ở đây chúng ta chỉ xét các cung được định nghĩa đệ quy. Cũng có thể bạn không hứng thú lắm với hình ảnh các cung nhưng mình nghĩ thuật toán để vẽ có thể giúp ích được cho bạn trong khi lập trình đấy.
Chúng ta xét 4 cung cơ bản là cung koch, cung C, cung Hilbel và cung Sierpinski.
1. Cung Koch
Cung này được nhà toán học Helge von Koch tìm ra năm 1904, nó có thể tạo ra một đường dài vô hạn trong một vùng hữu hạn. Nó được định nghĩa như sau:
Từ một đoạn thẳng ban đầu, nó được thay thế 4 đoạn thẳng có cùng chiều dài và





đoạn thẳng ban đầu. Cách bố trí như hình



bằng 1/3 chiều dài vẽ.(hình) Đó là cung mỗi đoạn của Koch bậc 1. Để vẽ cung Koch bậc 2 ta thay cung Koch bậc 1 theo quy tắc đó. Cũng theo quy tắc đó cho mỗi đoạn của cung Koch bậc n-1, ta có cung bậc n. Đây là thuật giải để vẽ.

```
Procdure Koch(dir, len:real; n: byte);
Const rads=0,017453292; {pi/180}
Begin
If n>0 then
Begin
Koch(dir,len/3,n-1);{đoạn 1}
Koch(dir+60,len/3,n-1);{đoạn 2}
Koch(dir-60,len/3,n-1);{đoạn 3}
Koch(dir,len/3,n-1);{đoạn 4}
End
ElseLineRel(Round(len*cos(Rads*dir)),-Round(len*sin(Rads*dir)));
End;
```

Với dir,len là hướng (góc) và độ dài ban đầu; n: độ sâu của đệ quy.

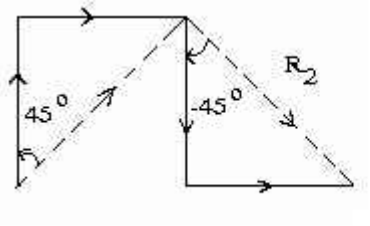
Một đoạn thẳng bất kỳ có thể vẽ được thành cung Koch. Chúng ta có thể vẽ cung Koch cho các cạch của một đa giác bất kỳ miễn là biết được đỉnh, góc bắt đầu và độ dài của một cạnh. Bằng hình học giải tích vectơ, bạn có thể biết được những yếu tố đó nhưng phải quy ước chiều quay của các cạnh là cùng chiều với kim đồng hồ. Và lúc đó bạn sửa tham số thế nào cho hợp?

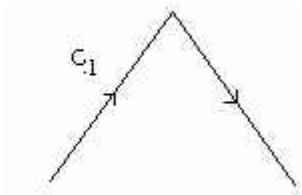
Đây là đoạn chương trình vẽ cung Koch trên ba cạnh của tam giác đều với đỉnh đầu là (xs,ys), độ dài len theo phương có góc bắt đầu là As

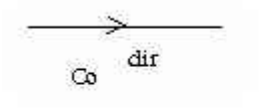
```
Procedure UDKoch(xs,ys:integer;As,len:real;n:byte);
Begin
MoveTo(xs,ys);
Koch(As+60,len,n);
Koch(As-60,len,n);
Koch(As+180,len,n);
End;
```

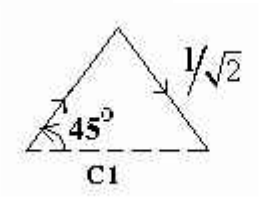
Bạn thử vẽ cung Koch cho đa giác đều m đỉnh thử xem.

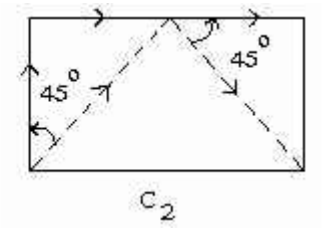
2. Cung C.











Cung C hình thành dựa vào việc tinh chế một đoạn thẳng và tiến trình đó được định nghĩa như sau:
Dựa vào cách định nghĩa, bạn thử xây dựng thủ tục vẽ cung C xem. Ta có được cung C bậc n bằng việc thay mỗi đoạn thẳng của cung C bậc n-1 theo cùng quy tắc.
Từ cung C, cung rồng được định nghĩa như sau: (hình)

Tinh chế đoạn thứ hai trong cung C theo chiều ngược với đoạn thứ nhất, ta thu được cung rồng.
Đây là thủ tục vẽ cung rồng với h là biểu điều khiển hướng quay.

```
Procedure CungR(dir, len: real;h :shortint; n:byte);
Const fct=0.707106781; {1/sqrt(2)}
Begin
If n>0 then
Begin
cungR(dir+h*45,len*fct,1,n-1);
cungR(dir-h*45,len*fct,-1,n-1);
end
else
LineRel(Round(len*cos(rads*dir)),-Round(len*sin(rads*dir)));
End;
```

Lúc gọi thủ tục, bạn gọi với tham số h=1; Khi vẽ bạn nhớ đến điểm khởi đầu, hướng bắt đầu, độ dài và độ sâu đệ quy. Bạn có thể tạo điểm dừng đệ quy cho chiều dài của cạnh, chỉ thực hiện khi len còn lớn hơn Minlen (tuỳ độ phân giải thiết bị).
Hẹn gặp lại bạn khi chúng ta tìm hiểu về cung Hilbert và cung SierpinaSki, những cung được định nghĩa phức tạp hơn.

Phạm Xuân Bách − 36/5K − Trần Phú − TP Huế;

18 .Suy diễn tiến - Một giải thuật thú vị

Tác giả: Lê Nhật Quang

Đã bao giờ bạn muốn xây dựng một chương trình giải toán tự động, chỉ cần nhập một đề toán, sau đó, chương trình trình sẽ đưa ra phương pháp giải, hoặc hơn nữa, chương trình sẽ tự động tính toán luôn kết quả cho bạn. Nếu câu trả lời là ″có″ thì thuật giải này sẽ là công cụ tuyệt vời để hỗ trợ cho bạn đấy! Bây giờ, chúng ta cùng

tìm hiểu về thuật giải này:

I/ Giới thiệu thuật giải:
1/ Hệ luật dẫn: là luật phát biểu dưới dạng:
If p1,p2,…,pn then q1,q2,…,qm
Trong đó, các ký hiệu pi,qj là các sự kiện nào đó.
VD : - If a>b, b>c then a>c
- If a=b then b=a
…
2/ Mô hình hệ luật dẫn: Gồm 2 thành phần cơ bản (F,R)
F là tập sự kiện , R là tập luật dẫn, mỗi luật có dạng: A---->B (A là giả thiết, B là kết luận của luật)
VD: Các liên hệ suy dẫn trên các yếu tố của một tam giác theo hệ luật dẫn:
(1) Tập sự kiện:
F={a,b,c,A,B,C,R,S,p,ha,hb,hc,…}
Trong đó: sự kiện a tương đương với ″biết cạnh a″
sự kiện b tương đương với ″biết cạnh b″
…
(2) Tập luật dẫn:
R={ r1 : A,B-> C,
r2: a,b,c->S,
…
}
3/ Vấn đề suy diễn:
Giả sử có hệ luật dẫn (F,R). Cho trước một tập sự kiện giả thiết GT và một tập sự kiện mục tiêu G.
Hỏi có thể suy ra các sự kiện mục tiêu G từ GT hay không?
4/ Suy diễn tiến:
Là quá trình suy ra các sự kiện mới từ những sự kiện đang có dự trên sự áp dụng của các luật dẫn, tập sự kiện xuất phát là các sự kiện trong giả thiết.
Quá trình suy diễn kết thúc khi đạt được các sự kiện mục tiêu hoặc khi không suy diễn thêm được sự kiện gì mớidựa trên các luật dẫn.
VD: GT={a,b,A} G={S}
Quá trình suy diễn:
- a,b,A -> B (luật a,b,A -> B dựa trên định lý hàm số Sin)
GT1={a,b,A,B}
- A,B-> C (luật A,B->C dựa trên định lý tổng các góc trong tam giác)
- C,a,b->S (luật C,a,b->S dựa theo công thức S=1/2abSinC)
-> từ a,b,A ta suy được S.

II/ Thuật giải suy diễn tiến:

Bước 1: Ghi nhận tập sự kiện ban đầu A=giả thiết và mục tiêu là B.
Bước 2: Tìm luật dẫn r: GT->KL sao cho GT thuộc A
Bươc 3: if (tìm được luật r) then
3.1 : Ghi nhớ luật r
3.2: Bổ sung luật r (KL của luật r) vào A.
3.3 if (B thuộc A) then Kết thúc
end
else Kết thúc: bị bế tắc.
Bươc 4: Trở lại bước 2.

Sau khi đọc xong thuật giải suy diễn tiến, bạn có thể tự xây dựng một chương trình giải toán đơn giản, ví dụ như giải toán hình học trong hệ luật dẫn tam giác.
VD:
Nhập giả thiết: a b c - mục tiêu: hc
Chương trình cho ra kết quả:
âb^c -> S (S=a*b*c/4*R)
S^c -> hc (hc=2*S/c)
Bạn nên tổ chức 1 file dữ liệu chứa tập sự kiện F và tập luật dẫn R. Chương trình sẽ dựa trên file dữ liệu này để giải một số bài toán. Đây là cấu trúc file dữ liệu đề nghị:
- Dòng 1: Chứa tập sự kiện của tam giác
a b c A B C ha hb hc p r R S
- K dòng tiếp theo chứa các luật dẫn và các chú thích :
ÂB -> C (C=180-B-A)
C,a,b->S (S=1/2abSinC)
…
Đây là một thuật giải đơn giản và hay,với một chút khéo léo trong việc xử lý, tôi nghĩ bạn sẽ dễ dàng xây dựng được chương trình này. Một điều lưu ý nữa là thuật giải này có thể sẽ làm một số bước dư thừa, bạn có thể dùng phương pháp quay lui hay tìm kiếm theo chiều rộng để tìm ra kết quả một cách tối ưu và công việc này xin nhường lại cho bạn! Nếu có thắc mắc hay cần chương trình ví dụ thì xin liên lạc với tôi tại địa chỉ: michealnest@hcmc.netnam.vn.

19. Các số Stirling

Tác giả: Vũ Đức Minh

Tin học với Toán học có một sự liên hệ rất mật thiết với nhau. Toán học có sự đóng góp lớn vào việc phát triển Tin học từ mặt lý thuyết, thuật toán đến các ứng dụng thực tế. Trong mục này tôi xin giới thiệu với các bạn các số Stirling, một khái niệm rất gần gũi với hệ số tổ hợp. Số Stirling được đặt tên bởi James Stirling (1692-1770). Mặc dù đã có một lịch sử lâu đời và có rất nhiều ứng dụng, các số Stirling vẫn chưa

có một ký hiệu chuẩn. Chúng ta dùng kí hiệu $\begin{bmatrix} n \\ k \end{bmatrix}$ cho các số Stirling loại 1 và $\begin{Bmatrix} n \\ k \end{Bmatrix}$ cho các số Stirling loại 2. Kí hiệu $\begin{Bmatrix} n \\ k \end{Bmatrix}$ tượng trưng cho số cách chia n đồ vật vào k tập con khác rỗng. Ví dụ, có 7 cách để chia 4 đồ vật vào 2 tập con: {1,2,3} υ{4}, {1,2,4} υ {3},{1,3,4} υ {2}, {2,3,4} υ {1},{1,2} υ {3,4},{1,3} υ {2,4}, {1,4} υ {2,3}, vì vậy $\begin{Bmatrix} 4 \\ 2 \end{Bmatrix}$ =7.Trước khi đi tìm công thức đệ quy cho $\begin{Bmatrix} n \\ k \end{Bmatrix}$ ta đồng ý rằng có một cách để chia một tập rỗng vào không tập rỗng, hay $\begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$ = 1, còn tất nhiên = 0 với n>0. Với một tập n>0 phần tử được chia thành k tập, ta có thể để phần tử cuối cùng vào riêng một tập hợp (có $\begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix}$ cách) hay đặt nó vào một tập con khác rỗng của n-1 phần tử đầu tiên với cách chia n-1 phần tử này ra k tập khác rỗng (có k cách), vì vậy ta có: $\begin{Bmatrix} n \\ k \end{Bmatrix} = k \begin{Bmatrix} n-1 \\ k \end{Bmatrix} + \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix}$ với n>0.

Còn $\begin{bmatrix} n \\ k \end{bmatrix}$ tượng trưng cho số cách sắp xếp n đồ vật vào k xích. Xích là một cách sắp xếp trên vòng tròn. Hai xích là giống nhau nếu có thể chặt xích ở vị trí nào đó và căng ra, ta thu được hai tập có thứ tự giống nhau. Với xích [A, B, C, D], ta có [A, B, C, D]= [B, C, D, A]= [C, D, A, B] = [D, A, B, C]. Nhưng xích [A, B, C, D] lại khác với [A, B, D, C] hay [D, C, B, A]. Và ta có 11 cách chia bốn phần tử thành hai xích [1,2,3][4]; [1,2,4][3]; [1,3,4][2]; [2,3,4] [1]; [1,3,2][4]; [1,4,2][3]; [1,4,3][2]; [2,3,4][1]; [1,2][3,4]; [1,3][2,4]; [1,4] [2,3]. Và ta dễ dàng nhận thấy rằng $\begin{bmatrix} n \\ k \end{bmatrix} \geq \begin{Bmatrix} n \\ k \end{Bmatrix}$ vì từ một tập hợp bất kỳ, ta có thể thu được nhiều xích hơn với các phần tử thuộc tập hợp ấy. Công thức đệ cho số Stirling loại 1 cũng được xây dựng tương tự như cho các số Stirling loại 2. Ta có thể xếp phần tử cuối cùng vào riêng một xích, n-1 phần tử còn lại vào k-1 xích (có $\begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$ cách) hay xếp phần tử cuối cùng vào một trong k xích của cách chia n-1 phần tử đầu tiên thành k xích (có (n-1) $\begin{bmatrix} n-1 \\ k \end{bmatrix}$ cách). Vì vậy ta có: $\begin{bmatrix} n \\ k \end{bmatrix} = (n-1) \begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$ với n>0.

Bên trên là khái niệm và công thức truy hồi cho các số Stirling loại 1 và loại 2. Tiếp theo tôi sẽ giới thiệu thêm một số tính chất cơ bản và nâng cao về hai dãy số này, và cũng là một bài tập để các bạn rèn luyện và ôn tập lại các kiến thức về tổ hợp. ở đây, đối với biểu thức logic A bất kỳ thì [A] sẽ nhận giá trị 1 nếu A đúng, ngược lại nhận giá trị 0 nếu A sai.

1. $\left\{{n \atop 0}\right\} = \left[{n \atop 0}\right] = [n=0]$

2. $\left\{{n \atop 1}\right\} = [n>0]$; $\left[{n \atop 1}\right] = (n-1)!\ [n>0]$.

   $\left\{{n \atop 2}\right\} = (2^{n-1}-1)[n>0]$; $\left[{n \atop 2}\right] = (n-1)!H_{n-1}[n>0]$

   $\left\{{n \atop n-1}\right\} = \left[{n \atop n-1}\right] = \binom{n}{2}$; $\left\{{n \atop n}\right\} = \left[{n \atop n}\right] = 1$

   $\left\{{n \atop k}\right\} = \left[{n \atop k}\right] = 0$ nếu $k>n$.

3. $\sum_{m \le k \le n} \left\{{n \atop k}\right\}\left[{k \atop m}\right](-1)^{n-k} = [m=n]$;

   $\sum_{m \le k \le n} \left[{k \atop m}\right]\left\{{n \atop k}\right\}(-1)^{n-k} = [m=n]$

4. $\left\{{n+1 \atop m+1}\right\} = \sum_k \binom{n}{k}\left\{{k \atop m}\right\}$

   $\left[{n+1 \atop m+1}\right] = \sum_{m \le k \le n} \left[{n \atop k}\right]\binom{k}{m}$

5. $\left\{{n \atop m}\right\} = \sum_{m \le k \le n} \binom{n}{k}\left\{{k+1 \atop m+1}\right\}(-1)^{n-k}$

   $\left[{n \atop m}\right] = \sum_{m \le k \le n} \left[{n+1 \atop k+1}\right]\binom{k}{m}(-1)^{k-m}$

6. $\left\{{n+1 \atop m+1}\right\} = \sum_{k=0}^{n} \left\{{k \atop n}\right\}(m+1)^{n-k}$

   $\left[{n+1 \atop m+1}\right] = n!\sum_{k=0}^{n} \frac{\left[{k \atop m}\right]}{k!}$

7. $\left\{{m+n+1 \atop m}\right\} = \sum_{k=0}^{m} k\left\{{n+k \atop k}\right\}$

   $\left[{m+n+1 \atop m}\right] = \sum_{k=0}^{m} (n+k)\left[{n+k \atop k}\right]$

8. $\binom{n}{m} = \sum_{m \le k \le n} \left\{{n+1 \atop k+1}\right\}\left[{k \atop m}\right](-1)^{k-m}$

Cuối cùng, tôi xin giới thiệu lại với các bạn một bài thi có sử dụng khái niệm và công thức truy hồi của số Stirling, đó là bài nằm trong đề thi ″lều chõng″ của Olympic Tin học sinh viên Toàn quốc lần thứ 12 tại Nha Trang. Bài toán tóm tắt

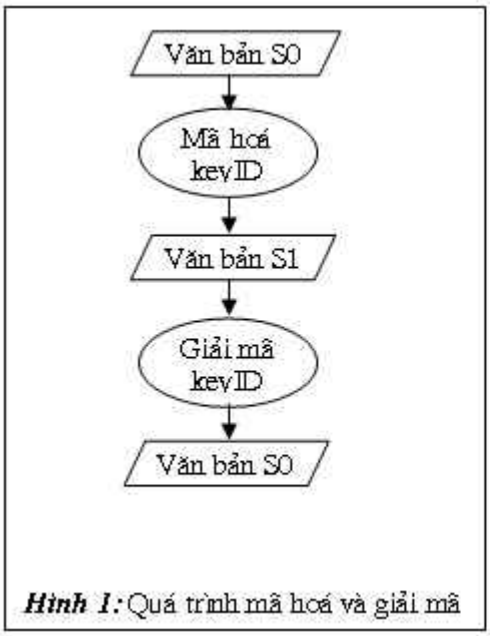
như sau: Cho số Stirling loại hai với công thức truy hồi
$\left\{ {n \atop k} \right\} = k\left\{ {n-1 \atop k} \right\} + \left\{ {n-1 \atop k-1} \right\}$, hãy xác định tính chẵn lẻ của số Stirling với n,k cho trước.

Toán học còn nhiều dãy số nữa mà được sử dụng nhiều trong Tin học, như các số Harmonic, Euclid, …Xin được giới thiệu với các bạn vào một dịp khác.

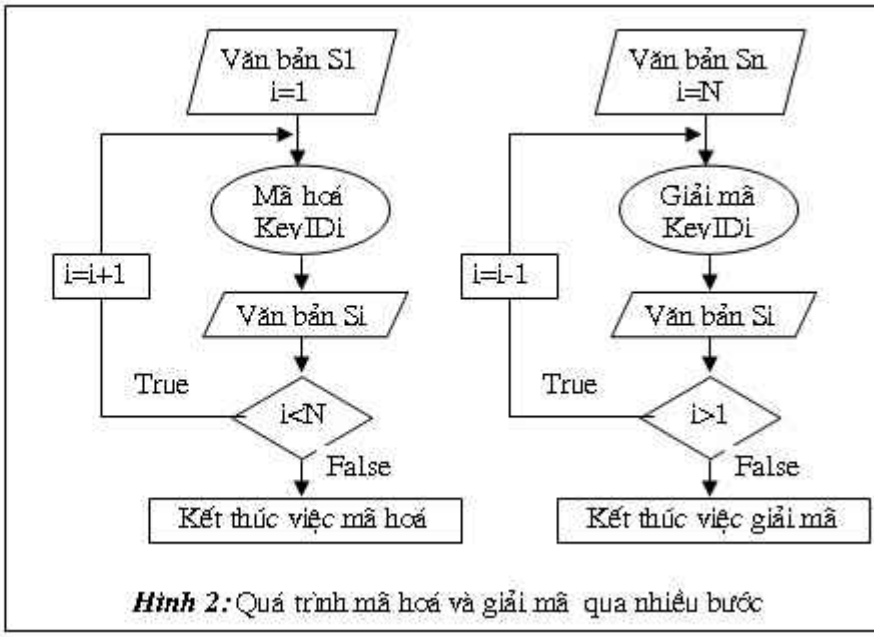
20 .Một phương pháp mã hoá và giải mã văn bản dạng text

Tác giả: Võ Công Chương

Phương pháp mã hoá văn bản mà tôi sắp nêu ra sau dây cũng là một trong những phương pháp mã hoá thông thường, hy vọng nó sẽ là một phương pháp mới trong sổ tay lập trình của các bạn yêu Tin học.
Trong phương pháp này, văn bản cần mã hoá hay giải mã của chúng ta là một dãy kí tự. Tuỳ thuộc vào khoá (keyID: là một số nguyên kiểu longInt) do chúng ta đưa vào mà kết quả của phương pháp sẽ khác nhau. Điều này nghĩa là nếu văn bản được mã hoá theo một khoá nào đó thì khi giải mã ta cũng phải dùng lại ngay chính khoá đó. Do vậy, mặc dầu phương pháp mã hoá này ai cũng biết nhưng họ chỉ giải mã được văn bản đã được mã hoá một khi họ biết được khoá dùng để mã hoá văn bản ấy. Mô hình mã hoá và giải mã được biểu diễn như hình 1.



Hình 1: Quá trình mã hoá và giải mã



Hình 2: Quá trình mã hoá và giải mã qua nhiều bước

Nhận xét: Ta cũng có thể mã hoá văn bản của ta qua N lần bởi N khoá khác nhau.

Khi đó, công việc giải mã phải làm ngược lại, nghĩa là phải giải mã N lần với N khoá tương ứng đã dùng. Biểu diễn quá trình này ở hình 2.
Như vậy, ta phải có hai hàm, hàm thứ nhất dùng để mã hoá, hàm thứ hai dùng để giải mã. Tuy nhiên, trong hàm mà tôi cài đặt sau đây nó đảm nhận cả hai chức năng trên (Bạn hãy để ý toán tử Xor trong giải thuật). Giải thuật trong hàm này không có gì phức tạp. Toàn bộ mã nguồn của hàm được cài đặt trong VB như sau:

```
Function MyEncrypt(MyStr As String, keyID As Long) As String
′Vào: MyStr là văn bản cần mã hoá hay giải mã.
′ keyID là khoá dùng cho việc mã hoá hay giải mã.
′Ra: Trả lại văn bản đã được mã hoá/giải mã.
Dim i As Integer
Dim stepNum As Integer
Dim keyIDtmp As Integer
Dim tmp As String
Dim ch As String
Dim byteArray() As Byte

tmp = Trim$(Str(keyID))
keyID = 0

For i = 1 To Len(tmp)
keyID = keyID + CLng(Mid(tmp, i, 1))
Next i

While (keyID > 255)
keyID = -1 + (keyID - 255)
Wend

ReDim Preserve byteArray(Len(MyStr))
For i = 1 To Len(MyStr)
byteArray(i) = Asc(Mid(MyStr, i, 1))
stepNum = keyID + i
While (stepNum > 255)
stepNum = -1 + (stepNum - 255)
Wend
byteArray(i) = byteArray(i) Xor CByte(stepNum)
Next i
tmp = ″ ″

For i = 1 To Len(MyStr)
```

```
tmp = tmp & Chr(byteArray(i))
Next i
MyEncrypt = tmp
End Function
```

Từ nay, sau khi có được phương pháp mã hoá này, chúng ta sẽ trao đổi tài liệu mật cho nhau mà không sợ bị người khác đọc được. Tất nhiên, hai người trao đổi tài liệu đó phải qui định khoá dùng cho cho công việc mã hoá và giải mã tài liệu của mình.

21 .Lý thuyết hình học trong toán tin

Tác giả: Nguyễn Thế Anh

1. Điểm, đường thẳng, đoạn thẳng:
a. Điểm (Point):
Trong hình học, chúng ta xét trong hệ Đề các xoy, thì một điểm có toạ độ: (x,y). Chính vì thế ta lưu tọa độ một điểm trong một bản ghi Record:

```
Type
point = Record
x , y : integer ;
End ;
Point_Chung = Record
x ,y : Real ;
End ;
```

Chính vì vậy khi xét tới toạ độ của P(x,y) thì ta xét P.x, P.y
Chúng ta biết khoảng cách giữa hai điểm P(x1,y1)và Q(x2,y2) trong mặt phẳng:

```
Function Khoang_Cach(P,Q: Point_Chung ) : Real ;
Begin
Khoang_Cach:=Sqrt(Sqr(P.x-Q.x)+Sqr(P.y-Q.y));
End ;
```

b. Đường thẳng (line):
Trong hình học, chúng ta có phương trình của một đường thẳng trong mặt phẳng: Ax+By+C =0. Chúng ta coi A, B, C là biểu diễn cho đường thẳng đó. Nếu một đường thẳng (d):
Ax + By + C = 0, đi qua 2 điểm A (x1,y1) và B(x2,y2) thì nó có:

```
A:= y1- y2;
B:= x2 - x1;
C:= -(A.x1+B.y1).
```

Chính vì thế chúng ta dùng thủ tục xác định A, B, C của một đường thẳng đi qua 2 điểm như sau:

```
Procedure Xac_DinhABC(P, Q: Point, var A , B , C: Longint );
```

```
Begin
A := P.y-Q.y ;
B := Q.x-P.x ;
C := -(A*P.x+B*P.y ) ;
End ;
```
Và chúng ta có thể coi đường thẳng là một kiểu:
```
Type
Lines = Record
a , b , c : Longint;
End ;
```
Chú ý:
* ″Lines″ chứ không phải là ″Line″, vì trong Pascal có thủ tục Line để vẽ đường thẳng. Thông thường chúng ta làm bài hình học để quan sát trực quan thì chúng ta thường biểu diễn lên hình vẽ. Chính vì thế chúng ta cần phải tránh những sai sót không đáng có.
* Sở dĩ chúng ta phải khai báo A, B, C trong Longint vì nếu chúng ta lưu A, B, C trong Longint thì lúc chúng ta tính toán phương trình không tràn bộ nhớ số học.

c. Đoạn thẳng:

Đoạn thẳng là một phần của đường thẳng, bị giới hạn x, y. Chúng ta xét đoạn thẳng thông thường cho đi qua 2 điểm. Chính vì thế toạ độ x, y bị giới hạn trong khoảng đó.

2. Phương trình tương quan giữa điểm và đường thẳng, đoạn thẳng:

a. Tương quan giữa điểm và đường thẳng:

Cho đường thẳng (d) có phương trình: Ax +By + C = 0 và P(x,y).

Thì phương trình:

F(x,y) = A*P.x + B*P.y + C, là phương trình tương quan của P với (d). Đúng vậy:
* Nếu F (x,y)=0 thì P thuộc (d). Ngược lại nếu F(x,y)<>0 thì P không thuộc (d).
* Nếu Q (x1,y1) mà nằm cùng phía không thuộc (d) thì F (x1,y1)*F(x,y)>0, và nếu khác phía thì F(x1,y1)*F(x,y)<0. Đây là một trong những điều kiện giúp ích cho ta rất nhiều trong giải toán tin hình học và cũng là một phương tiện thiết kế chương trình trong hình học dễ dàng hơn.

Chúng ta xây dựng hàm Phuong_Trinh để xác định mối tương quan của đường thẳng với một điểm:
```
Function Phuong_trinh (L : Lines ; P : Point_Chung ) : Real ;
Begin
Phuong_Trinh:=L.a*P.x+L.b*P.y+L.c;
End ;
```
b. Tương quan của điểm với đoạn thẳng.

Chúng ta biết rằng, đoạn thẳng là một phần đường thẳng. Nên mối tương quan giữa điểm P(x,y) với đoạn thẳng AB, (A (x1,y1), B(x2,y2)) là:
- Nếu P[AB] thì:

+ F(x,y)=0, tức là: a*x+b*y+c=0, với a=y1-y2; b=x2-x1 và c=- (a*x1+b*y1).
+ (x-x1)*(x-x2)<=0 và (y-y1)*(y-y2)<=0.
- Nếu P[AB] thì:
+ Nếu F(x,y)=0 thì: (x-x1)*(x-x2)>0 hoặc (y-y1)*(y-y2)>0.
+ Nếu F(x,y)<>0 thì P không thuộc đường thẳng qua A, B.
Ta có thể xây dựng hàm kiểm tra 1 điểm P có thuộc đoạn AB như sau:

```
Function thuoc_doan ( P : Point_Chung ; A, B : Point ) : Boolean ;
Var a , b , c : longint ; t : Real ;
Begin
xac_dinhABC (A, B ,a,b,c) ;
thuoc_doan:=false ;
t := a * P.x + b*P.y + c ;
if t<>0 then exit ;
if ( (P.x-A.x)*(P.x-B.x)>0)
r((P.y-A.y) * (P.y-B.y)>0)
then
Exit ;
thuoc_doan:=True;
End ;
```

3. Cắt nhau:
a. Đường thẳng cắt đường thẳng:
(1): A1*x+B1*y+C1=0 và (2): A2*x+B2*y+C2=0. Thì ta gọi mối tương quan giữa (1)và (2) được biểu diễn qua hệ phương trình sau:
A1*x+B1*y=-C1
A2*x+B2*y=-C2
đặt: D=A1*B2-A2*B1;
Dx=B1*C2 - B2*C1;
Dy=C1*B2-A1*C2
* Hai đường thẳng cắt nhau khi và chỉ khi: D≠0. Toạ độ điểm giao của hai đường thẳng này là: x=Dx/D, y= Dy/D
* Hai đường thẳng song song với nhau khi và chỉ khi: D=0 và Dx ≠ 0 hoặc Dy ≠ 0.
* Hai đường thẳng trùng nhau khi: D=Dx=Dy=0.
Chúng ta có thể xây dựng hàm kiểm tra cắt nhau của hai đường thẳng.
Hàm Lines_Cut có giá trị:
1: Nếu hai đường thẳng đó cắt nhau
2: Nếu hai đường thẳng đó song song nhau
3: Nếu hai đường thẳng đó trùng nhau.

```
Function Lines_cut ( L1 , L2 : Lines ) : Byte ;
var D, Dx,Dy : Longint ;
Begin
D := L1.a*L2.b -L2.a*L1.b;
```

```
Dx :=L1.b*L2.c-L1.c*L2.B ;
Dy:=L1.c*L2.b-L1.b*L2.c ;
If D<>0 then Lines_cut:=1
else If D=0 then
Begin
If (Dx<>0)Or(Dy<>0)Then
Lines_cut:=2 ;
If (Dx=0)And(Dy=0)
ThenLines_Cut:=3;
End ;
End ;
```

b. Đường thẳng cắt đoạn thẳng:

(1): A1*x+B1*y+C1=0 và đoạn AB, A (x1,y1), B(x2,y2).

Đặt A2:=y1-y2; B2:=x2-x1;

C2:=-(A2*x1+B2*y)

D=A1*B2-A2*B1; Dx=B1*C2 -B2*C1; Dy=C1*B2-A1*C2

Mối quan hệ giữa (1) và AB được thể hiện:

* Nếu D ≠ 0 và điểm P(Dx/D,Dy/D) nằm trên đoạn AB thì (1) cắt AB.
* Nếu D ≠ 0 và điểm P(Dx/D,Dy/D) nằm ngoài đoạn AB thì (1) cắt đường thẳng chứa AB nhưng không cắt AB.
* Nếu D=Dx=Dy=0 thì AB(1).
* Nếu D=0, Dx/D0 hoặc Dy/D0 thì AB song song với (1).

Chúng ta xây dựng hàm: Lines_Cut_AB

Lines_Cut_AB bằng:

1: Nếu đoạn thẳng cắt đường thẳng.
2: Nếu đoạn thẳng song song với đường thẳng
3: Nếu đoạn thẳng thuộc đường thẳng.
4: Nếu đường thẳng chứa đoạn thẳng cắt đường thẳng một điểm nằm ngoài đường thẳng.

```
Function Lines_Cut_AB (L1:Lines ; P,Q : Point ): Byte;
Var L2 : Lines;
Giao : Point_Chung ;
D , Dx , Dy : Longint ;
Begin
With L2 do Xac_DinhABC (P,Q,a,b,c);
D := L1.a*L2.b -L2.a*L1.b;
Dx :=L1.b*L2.c-L1.c*L2.B ;
Dy:=L1.c*L2.b-L1.b*L2.c ;
If D=0 Then
Begin If (Dx=0)And (Dy=0)Then Lines_Cut_AB:=3;
If ( Dx<>0)Or(Dy<>0)Then Lines_Cut_AB:=2 ;
```

```
End ;
If D<>0 Then
Begin
Giao.x := Dx/D ;
Giao.y := Dy/D ;
If Thuoc_Doan (Giao,P,Q) then
Lines_Cut_AB:=1
Else Lines_Cut_AB:=4 ;
End ;
End ;
```

c. Đoạn thẳng cắt đoạn thẳng:

Xét hai đoạn thẳng AB và CD thì chúng cắt nhau hay không thì chúng ta có thể xét theo hai cách. Trong bài này tôi xin đề nghị cả hai cách, mỗi cách có những ưu, nhược điểm khác nhau.

Cách 1:

Cách Theo Phương Trình Đường Thẳng:

```
Function Cat_Nhau1 (P,Q,M,N: Point) : Boolean ;
Var L : Lines ;
Begin
Cat_Nhau1:=False;
With L Do
Xac_DinhABC(P,Q,a,b,c);
If Lines_Cut_AB(L,M,N)<>1 then Exit;
With L Do
Xac_DinhABC(M,N,a,b,c);
If Lines_Cut_AB(L,P,Q)<>1 then Exit ;
Cat_Nhau1:=True ;
End ;
```

Cách 2:

(Tham khảo sách Cẩm nang thuật toán - tập 2)

```
Function ccw ( P0 , p1 , p2 : Point ) : Integer ;
Var Dx1,Dx2,Dy1,Dy2: Integer ;
Begin
Dx1 := P1. x -P0.x ;
Dy1 :=P1.y-P0.y ;
Dx2 := P2. x -P0.x ;
y2 :=P2.y-P0.y ;
If Dx1*Dy2>Dy1*Dx2 then
ccw :=1 ;
If Dx1*Dy2
ccw := -1 ;
```

```
If Dx1*Dy2=Dx2*Dy1 then
Begin
If (Dx1*Dx2<0) Or (Dy1*Dy2<0)
Then ccw:=-1
Else If (Dx1*Dx1+Dy1*Dy1) >= (Dx2*Dx2+Dy2*Dy2) then
ccw:=0
else ccw:=1 ;
End ;
End ;
Function Cat_Nhau2(P,Q,M,N : Point) : Boolean ;
Begin
Cat_Nhau2:=((ccw(P,Q,M)*ccw(P,Q,N))<=0)And
((ccw(M,N,P)*ccw(M,N,Q))<=0)
End ;
```

4. Đa giác:

a. Tam giác:

Một tam giác được định nghĩa là tập ba điểm không thẳng hàng: A (x1,y1); B(x2,y2); C(x3,y3).

Chúng ta có thể tính diện tích tam giác theo công thức tính diện tích đa giác (công thức hình thang hoặc công thức Pic mà tôi sẽ bàn sau). Hoặc chúng ta tính theo công thức Herong:

S:=Sqrt((p-a)*(p-b)*(p-c)*p);

Trong đó a, b, c là độ dài ba cạnh của tam giác. P=(a+b+c)/2;

b. Hình chữ nhật: (trường hợp cho cách cạnh song song các trục toạ độ).

Chúng ta xét trong hệ tọa độ một hình chữ nhật ABCD, A (x1,y1); B(x2,y2); C(x3,y3); D (x4,y4).

Nhưng có một điều đặc biệt là chúng ta chỉ cần xác định toạ độ đỉnh của hai đỉnh đối nhau thì xác định đợc một hình chữ nhật duy nhất. Chính vì thế thông thường chúng ta gọi toạ độ của điểm dưới trái và đỉnh trên phải là hai điểm đặc trưng cho hình chữ nhật đó.

c. Hình đa giác:

Một đa giác A1A2…An có toạ độ: Ai (xi ,yi).

* Người ta định nghĩa đa giác đó là lồi khi mọi điểm còn lại của đa giác nằm cùng phía với nhau so với một cạnh nào đó.

* Diện tích một đa giác được tính theo công thức hình thang như sau:

S : = |[(Xi-Xi+1)*(Yi+Yi+1)/2]|

d. Bao lồi:

(Chỉ giới thiệu, phần chi tiết đã đăng ở số báo trước).

Bao lồi của một tập điểm là một hình đa giác khép kín có các đỉnh là một trong các đỉnh của tập điểm đó, và thoả mãn đa giác lồi. Thông thường chúng ta cần phải tìm đa giác bao với chu vi nhỏ nhất. Có nhiều thuật toán để giải quyết bài toán này. Đặc

biệt phương pháp quét:
Phương pháp quét đường thẳng:
Chúng ta đi từ một đỉnh chắc chắn thuộc bao lồi (là những điểm có tung độ hoành độ lớn nhất hoặc nhỏ nhất). Chúng ta tìm các đỉnh tiếp theo, đỉnh nào thoả mãn chứa toàn bộ các đỉnh còn lại một bên mặt phẳng thì ta lấy điểm đó cho đến khi lặp lại điểm ban đầu.
Thủ tục giải quyết nó như sau:
Procedure Scan;
Begin
Xác định thuộc đa giác ;
Repeat
tìm đỉnh tiếp theo mà thoả mãn điều kiện lồi của đa giác ;
Until lặp lại đỉnh ban đầu ;
End ;
Những bài toán hình học thì không khó về giải thuật nhưng lại rờm rà về chương trình. Chính vì thế chỉ cần một sai sót nhỏ thì chúng ta sẽ không thể nào có thể sửa chương trình trong thời gian cho phép, đặc biệt là khi ngồi trong phòng thi tâm trạng không ổn định. Chính vì thế học phần này một cách bài bản, nghiêm túc thì chúng sẽ hẳn không khó đối với chúng ta. Không nên khinh thường những thứ sơ đẳng như trên. Nếu các bạn muốn có bộ bài tập về hình học thì cứ gửi mail cho mình. Rất mong nhận được thư của các bạn

22. **Reverse Polish Notation - Thuật toán tính giá trị biểu thức**

Tác giả: Ngô Minh Đức

Reverse Polish Notation (ký pháp nghịch đảo Ba Lan) là một loại ký pháp toán học dùng để biểu thị biểu thức đại số, rất thuận lợi trong giải thuật tính giá trị biểu thức, có nghĩa là bạn nhập vào một chuỗi, chẳng hạn ″(1+2)*3″, chương trình sẽ tính ra kết qủa bằng 9.
Reverse Polish Notation (gọi tắt là RPN) có hai loại, tiền tố (preffix) và hậu tố (suffix), trong bài này tôi chỉ đề cập đến dạng suffix.
Để hiểu rõ hơn và dễ dàng tiếp cận với thuật toán này trước tiên chúng ta xét một ví dụ đơn giản sau đây:
Chẳng hạn xét biểu thức sau:
(1 + 2) x 3 thì dạng RPN - hậu tố - của biểu thức trên là: 1 2 + 3 x
Chương trình có thể tính toán rất dễ dàng với biểu thức dạng RPN. Bạn để ý trong biểu thức RPN không còn dấu ngoặc nữa, khi tính toán chỉ cần duyệt các toán tử từ trái sang phải và thực hiện với các toán hạng đứng trước nó.
Đây là mô tả qúa trình tính toán biểu thức RPN trên:

Bước 1: Duyệt đến dấu +, ngừng lại.
Bước 2: Cộng hai toán hạng đứng trước nó: 1+2=3.
Bước 3: Xóa các phần tử 1, 2, + thay bằng số 3
Quay lại bước 1: Duyệt đến dấu x, ngừng lại
Quay lại bước 2: Nhân hai toán hạng đứng trước nó: 3x3=9
Quay lại bước 3: Thay các phần tử 3, 3, x bởi số 9. Chỉ còn một phần tử nên kết thúc.
Kết qủa chính là phần tử duy nhất còn lại: Đó là số 9
Do đó vấn đề chỉ còn là chuyển biểu thức từ dạng thông thường sang dạng RPN. Sau đây tôi sẽ trình bày giải thuật chuyển đổi dưới hai phần.
1. Phần cơ bản.
Giải thuật này do các bạn trên TTVNOnline (www.ttvnol.com) và Diễn Đàn Tin Học (www.diendantinhoc.com) cung cấp.
Xin các bạn chú ý một điều là: các phần tử trong biểu thức được chia ra làm hai loại: toán hạng (số) và toán tử (bao gồm dấu và hàm)
Trong các loại toán tử, chúng ta cần lưu ý đến toán tử cộng trừ một ngôi (unary plus/minus). Đây là một loại toán tử chỉ tác dụng lên một toán hạng đứng trước nó, khác với toán tử cộng trừ bình thường tác dụng lên hai toán hạng đứng trước nó
ví dụ: -2 + 5 thì ″-″ là toán tử một ngôi, ″+″ là toán tử bình thường
Để xác định ″+″, ″-″ là một ngôi hay hai ngôi, khi duyệt biểu thức ta chỉ cần xem phần tử đứng trước nó là số hay là một toán tử khác
Mức ưu tiên của các toán tử (từ nhỏ đến lớn): (, +, -, *, /, ^, + một ngôi, - một ngôi
Thuật toán:
Ta sử dụng hai stack (ngăn xếp): rpnStack dùng để lưu dạng RPN, oprStack dùng để tạm lưu các toán tử trong qúa trình xử lý. Đọc lần lượt từ đầu đến cuối biểu thức để xử lý theo từng loại:
Dấu mở ngoặc: đưa vào oprStack
Toán hạng: đưa vào rpnStack
Toán tử: Trong trường hợp này ta phải xét toán tử đang ở trên cùng (được đưa vào cuối cùng) trong oprStack: nếu mức ưu tiên cao hơn toán tử đang xét thì chuyển toán tử đó sang rpnStack; tiếp tục làm như vậy cho đến khi được toán tử có mức ưu tiên nhỏ hơn hoặc bằng toán tử đang xét; cuối cùng đưa toán tử đang xét vào oprStack
Dấu đóng ngoặc: lần lượt chuyển các toán tử được lưu trong oprStack sang rpnStack; cho đến khi gặp dấu mở ngoặc thì dừng lại và xóa dấu mở ngoặc đó khỏi oprStack
Lưu ý: Khi viết chương trình chỉ duyệt được từng ký tự, do đó phải thêm phần nhận biết nhóm ký tự hợp thành một số (hoặc một tên biến)
Ví dụ: 564 + 4, khi bắt đầu duyệt từ ký tự ″5″ sẽ nhận biết luôn số ″564″ và nhảy đến ký tự thứ tư.
Đây là những bước cơ bản của thuật toán, sau khi chuyển đổi xong, bạn đọc từ đầu đến cuối (từ dưới lên trên) trong rpnStack sẽ thu được dạng RPN.

Sau khi thu được dạng RPN, cách tính toán như sau:
Tìm toán tử đầu tiên trong rpnStack và áp dụng tính với các toán hạng đứng trước nó. Sau đó, xóa toán tử và các toán hạng đã tính, thay bằng kết qủa tính được.
Tiếp tục cho đến khi nào không còn toán tử để tính nữa, kết qủa của biểu thức chính là phần tử đầu tiên của rpnStack.
2. Phần mở rộng.
Trên đây chỉ là các bước cơ bản, tùy theo sự khéo léo của bạn mà có thể cải tiến giải thuật để tính được những biểu thức phức tạp hơn. Tôi xin trình bày một số mở rộng về giải thuật để tính phân số, hỗn số và tính hàm nhiều tham số (chẳng hạn max(1,2,3) =3)
* Tính phân số, hỗn số.
Ta đặt thêm hai toán tử, gọi là oprFraction (dấu phân số, giả sử là ″~″) và oprMixed (dấu hỗn số)
oprMixed chỉ là một toán tử ″ảo″ được thêm vào cho thuận lợi trong qúa trình xử lý.
Độ ưu tiên của toán tử: (, +, -, *, /, ~, ^, + một ngôi, - một ngôi).
Trước khi đưa toán tử phân số vào rpnStack ta cần xét phần tử trên cùng trong rpnStack như sau:
Nếu cũng là toán tử phân số: gộp toán tử này và toán tử phân số đang xét thành toán tử hỗn số.
Nếu là toán tử hỗn số: cho chương trình báo lỗi.
Trong trường hợp còn lại: đưa toán tử phân số vào rpnStack bình thường.
Thêm phần xử lý dấu phân số và dấu hỗn số vào thủ tục tính toán như sau:
Dấu phân số: gọi hàm khởi tạo fraction (gọi chung cho phân số/hỗn số) từ hai toán hạng đứng trước.
Dấu hỗn số: gọi hàm khởi tạo fraction từ ba toán hạng đứng trước.
Fraction có thể được quy định theo kiểu String (vd ″1~3″, ″17~18″,″1~1~2″….).
Trong khi tính toán với fraction nên đổi hết ra phân số (chỉ gồm tử và mẫu), đến khi xuất kết qủa mới đổi thành hỗn số hoặc giữ nguyên tùy theo lựa chọn của người dùng.
Lúc này, khi thực hiện các phép toán + - * /, có thể làm như sau (theo kiểu máy tính Casio):
Ta xét hai toán hạng của phép tính:
Nếu gồm một số lẻ thập phân: đổi hết ra số thập phân (nếu toán tử còn lại là phân số) và tính.
Nếu chỉ gồm số nguyên và phân số: tính theo phân số
Bạn phải tự viết một module để xử lý phân số (cộng, trừ, nhân, chia, khởi tạo,...)
Để xử lý loại biểu thức phức tạp hơn (chẳng hạn (1~3)~2 = 1~6) thì cần phải thêm một số bước nữa. Bạn có thể liên hệ với tôi để nhận được mã nguồn.
* Hàm nhiều tham số.
Phần này trình bày cách xử lý các hàm nhiều tham số, chẳng hạn như max(1,2,3) hay uscln(5,6,12,30),v.v…

Ta đặt thêm một toán tử gọi là oprComma (dấu phẩy), độ ưu tiên chỉ đứng trên dấu mở ngoặc (áp chót). Trong khi chuyển đổi cũng xử lý với oprComma như những toán tử khác
Để mô tả thuật giải, ta xét ví dụ sau: max(1,2,3).
Sau công đoạn chuyển đổi ta có: 1 2 3 , , max.
Khi gặp dấu ″,″ ta gộp hai toán hạng đứng trước nó vào một mảng.
Như vậy sau bước 1 ta có: 1(2,3), max (ký hiệu (2,3) là chỉ mảng).
Gặp dấu ″,″ tiếp theo ta gộp luôn phần tử đầu tiên vào phần tử ″mảng″ thứ hai:
Như vậy sau bước 2 ta có (1,2,3) max.
Do đó hàm ″max″ đã trở nên được thực hiện trên một tham số duy nhất, tham số đó lại là một mảng các đối số. Để tính toán được ta phải xây dựng các hàm xử lý trên mảng đối số.
Lưu ý: Cách làm này chưa hay và không thích hợp với những trình biên dịch như Turbo Pascal. Bạn có thể liên hệ với tôi theo địa chỉ: attilathehunvn @yahoo.com để nhận được mã nguồn (bằng VB và VB. NET). Tôi mô tả giải thuật này trong một class, gọi là ″Evaluator″, bao gồm một thủ tục chính là Eval(). Bạn có thể gọi thủ tục này từ bất kỳ module nào, chẳng hạn Eval(″1~3+2″) sẽ cho kết qủa là ″2~1~3″.
Class này được áp dụng trong chương trình Quick Calculator 1.0 của tác giả. Rất mong được các bạn giúp đỡ để hoàn thiện thêm giải thuật..

Ngô Minh Đức

23 .Một phương pháp lập bảng số nguyên tố

Tác giả: Ngô Minh Đức

Trong bài viết này tôi xin giới thiệu một phương pháp lập bảng số nguyên tố khác, ngoài phương pháp sàng Eratosthenes đã qúa quen thuộc với các bạn.
Xin nhắc lại về sàng Eratosthenes (mang tên nhà toán học Hy Lạp, 275 - 194 TCN):
- Ta sắp xếp các số tự nhiên từ 2 đến N vào một bảng theo thứ tự 2,3,4,5,... N.
- Lưu lại số 2, lần lượt xóa đi các bội số của 2.
- Sau số 2, số không bị xóa đầu tiên là số 3, lần lượt xóa đi các bội số của 3
- Sau số 3, số không bị xóa đầu tiên là số 5, lần lượt xóa đi các bội số của 5,...
- Lặp lại đến khi không thể xóa được nữa thì thôi
Như vậy các số không bị xóa lập thành bảng các số nguyên tố nhỏ hơn hoặc bằng N.
Trong bộ sách ″Câu Chuyện Toán Học″ (một bộ sách rất hay gồm 6 tập của các thầy Nguyễn Bá Đô, Đỗ Mạnh Hùng và Nguyễn Văn Túc), tập 3 có giới thiệu một phương pháp lập bảng số nguyên tố khác. Tôi xin trình bày lại ở đây:
Học sinh Sundaram người ấn Độ đã đưa ra phương pháp này năm 1934.

Trước tiên Sundaram liệt kê một bảng các số như sau:
4 7 10 13 16 19 22...
7 12 17 22 27 32 37...
10 17 24 31 38 45 52... (1)
13 22 31 40 48 58 67...
16 27 38 49 60 71 82...
.....
Bảng số này có tính đối xứng (số ở hàng m cột n bằng số hàng n cột m)
Các số này được chọn bằng cách lần lượt thực hiện các bước sau đây:
1. Viết số 4 vào góc bên trái
2. Các số tiếp theo của hàng 1 lần lượt là số liền trước cộng thêm 3 (các số trên hàng 1 lập thành cấp số cộng với công sai là 3)
3. Các số tiếp theo của hàng 2 lần lượt là số liền trước cộng thêm 5 (các số trên hàng 5 lập thành cấp số cộng với công sai là 5)
.....
Dễ dàng tìm được công thức để tính giá trị số thứ m hàng thứ n:
S m,n = (2m+1)n + m = 2mn + m + n
Sundaram chỉ ra rằng: Nếu N có trong bảng (1) thì 2N+1 là hợp số. Nếu N không có trong bảng (1) thì 2N+1 là số nguyên tố.
Để khẳng định điều đó, ta chứng minh mệnh đề tương đương sau: 2N+1 là hợp số khi và chỉ khi N thuộc bảng (1)
Chứng minh:
(2N+1=2(2mn+m+n)+1=(2m+1)(2n+1)
Do đó 2N+1 là hợp số
(=>) Nếu 2N+1 là hợp số, ta có: 2N+1=xy (x,y lẻ)
Đặt x=2m+1, y=2n+1
=> 2N+1=(2m+1)(2n+1)=2(2mn+m+n)+1
=> N=2mn+m+n
Suy ra N là số thứ m của hàng n trong bảng
Bây giờ ta thử xây dựng hai phương pháp trên thành chương trình.
Chương trình:
Vì Pascal không xây dựng được các mảng kích thước lớn (cỡ vài chục triệu phần tử), tôi xin chọn VB 6.0 để thể hiện thuật toán.
Bạn tạo một Project và mở Module mới (lưu ý chọn Project Properties − Startup Object − Sub Main()) để chạy chương trình.
Ta sử dụng mảng Mark() kiểu Byte để đánh dấu, lưu ý đối với mỗi phương pháp mảng này có ý nghĩa khác đôi chút:
- Sàng Eratosthenes: Mark(I)=0 nếu I là số nguyên tố, =1 nếu I là hợp số
- Phương pháp của Sundaram: Mark(I)=0 nếu 2*I+1 là số nguyên tố, =1 nếu 2*I+1 là hợp số
Khai báo như sau:

```
Option Explicit
Dim Mark() As Byte
Dim N As Long
```
Trong đó biến N xác định cần tìm các số nguyên tố từ 2 đến N.
Sàng Eratosthenes
Đây là thủ tục thể hiện sàng Eratosthenes, vì đã qúa quen thuộc nên có lẽ không cần giải thích nhiều với các bạn. Lưu ý một chút là ta chỉ cần xét các số từ 2 đến sqrt(N):
```
Private Sub Eratosthenes()
' ---------------------------------------------------------------------------------------------------
------
' Thủ tục: Eratosthenes()
' Lập bảng số nguyên tố bằng sàng Eratosthenes
' ---------------------------------------------------------------------------------------------------
------
Dim I As Long
Dim K As Long
ReDim Mark(N) ′ định lại dung lượng cho mảng
Mark(0) = 1: Mark(1) = 1 ' 0 và 1 không phải số nguyên tố
' sử dụng sàng Eratosthenes
For I = 2 To Int(Sqr(N))
If Mark(I) = 0 Then ' nếu i là số nguyên tố
K= I * 2 ' gạch các bội số của i
Do While K >N/2 thì dừng lại
```
Xét hàng 2, tất nhiên xét từ ô (2,2) để bỏ qua giá trị trùng lặp, cho đến số hạng > N/2 thì dừng lại...
Cho đến hàng n mà giá trị của ô (n,n)>N/2 thì dừng lại.
ở mỗi số hạng K tìm thấy, ta đều đánh dấu Mark(K)=1 để chỉ 2*K+1 là hợp số
Toàn bộ qúa trình trên là để xác định trong phạm vi 2->N/2, số nào có trong bảng, số nào không có trong bảng. Nhờ đó mà xác định được từ 2->N, số nào là số nguyên tố, số nào là hợp số.
Thủ tục như sau:
```
Private Sub Sundaram()
' ---------------------------------------------------------------------------------------------------
------
' Thủ tục: Sundaram()
' Tạo bảng số nguyên tố bằng phương pháp của Sundaram
' ---------------------------------------------------------------------------------------------------
------
Dim M As Long
Dim K As Long
Dim P As Long
```

```
' Lưu ý: Mark(0)=0 để chỉ 2 là số nguyên tố
ReDim Mark(N 2) ′ định lại dung lượng cho mảng
M = 0
Do M = M + 1
P = 2 * M + 1 ' các số trên hàng M lập thành cấp số cộng có công sai 2*M+1
K = M * (P +1) ' K là số nằm trên đường chéo của hàng M
Do While K> N/2 ' cho đến khi số trên đường chéo của hàng M+1 > N/2 thì dừng.
End Sub
```

Bạn có thể so sánh tốc độ hai thủ tục trên cùng một giá trị N như sau:

```
Dim t As Double
N = 10000000
```

So sánh giữa giá trị hiện ra trong hộp thông báo của đoạn mã:

```
t = Timer()
Call Sundaram
MsgBox Timer() − t
```

Và:

```
t = Timer()
Call Erastosthenes
MsgBox Timer() − t
```

Ngoài ra nếu thích bạn có thể trình bày bảng số nguyên tố ra file văn bản bằng thủ tục sau:

```
Private Sub PrintPrimes(ByVal FileName As String, ByVal Kind As Boolean)
' ---------------------------------------------------------------------------------------------------
------
' Thủ tục: PrintPrimes
' In bảng số nguyên tố đã tạo được ra text file
' Mô tả: FileName= tên File cần xuất
' Kind=True nếu sử dụng sàng Eratosthenes,
' = False nếu sử dụng sàng của Sundaram
' ---------------------------------------------------------------------------------------------------
------
Dim I As Long
Dim J As Long
Dim K As Long
Dim P As Long
Open FileName For Output As #1
' tiêu đề
Print #1, Space(20) & 'Bang cac so nguyen to tu 2 den ' & N
K = 6 ' viết một hàng 6 số
I = 0
Do
```

```
If Mark(I) = 0 Then
If Kind Then ' sàng Eratosthenes
P = I ' Mark(I)=0 thì số nguyên tố cần viết là I
Else ' sàng của Sundaram
P = IIf(I = 0, 2, 2 * I + 1) ' quy ước số nguyên tố cần viết = 2 nếu I=0,...
End If
If K = 6 Then
Print #1, ' xuống dòng
K = 1
Else K = K + 1
End If
Print #1, Space(11 - Len(CStr(P))) + CStr(P); ' căn lề phải sao cho mỗi số đươc 12 kí tự
End If
I = I + 1
Loop Until ((I > N/ 2) And (Not Kind)) Or (I > N) ' chỉ duyệt đến N/2 với phương pháp của Sundaram
Close #1
End Sub
```

Ví dụ để sử dụng thủ tục:

```
Sub Main()
N=10000000
Call Sundaram
PrintPrimes("D:Sundaram.txt",False)
End Sub
```

Bạn nên xem file văn bản bằng font Courier New (nếu xem trên Windows), vì font này có bề rộng các chữ bằng nhau (fixed-width font) nên các số sẽ được sắp thẳng cột

Tôi đã chạy thử thủ tục Sundaram() với N=100 000 000- số nguyên tố lớn nhất tìm được là 99 999 989, mất khoảng 88 giây (trên máy Celeron 700 Mhz, 128 MB Ram, Windows 98 − chương trình chạy trực tiếp, chưa biên dịch ra file.EXE), riêng file văn bản chiếm 60 MB dung lượng. Trong khi đó thủ tục Eratosthenes() mất tới 144 giây (thật không ngờ !!)

Có thể trong các trường hợp trên, cách viết mã cũng như thuật toán chưa đạt được hiệu qủa cao nhất, dẫn đến việc so sánh không hợp lý. Vì thế rất mong nhận được sự góp ý của các bạn, mọi trao đổi xin gửi về địa chỉ: attilathehunvn@yahoo.com.

Thân!

24. Tối ưu các bài toán về số nguyên tố

Tác giả: Đinh Hữu Công

Trong số 11-2001 tác giả Nguyễn Văn Trường đã giới thiệu các thuật toán về số nguyên tố. Nhưng một số thuật toán này còn bị hạn chế về thời gian và bộ nhớ với n lớn. Tuỳ vào từng bài toán cụ thể ta có thể tối ưu hoá những thuật toán này dựa vào những nhận xét và chứng minh toán học.

Bài toán 1: Cho số tự nhiên n. Hãy phân tích số n thành tích các thừa số nguyên tố

Input: file Phantich.inp với số n duy nhất (n < 231)

Ouput: file Phantich.out. Dòng 1 là k, số thừa số nguyên tố khác nhau trong phân tích. K dòng tiếp theo dòng thứ i gồm 1 cặp số (p,q) cách nhau 1 dấu trắng trong đó p là thừa số nguyên tố và q là số mũ tương ứng của nó trong phân tích (q > 0).

Thuật toán của bài này rất đơn giản:

+ Cho biến chạy Temp=0,
+ Lặp: Chừng nào temp≤n thì làm
- Temp:=số nguyên tố liền sau Temp;
- While (n mod Temp=0) do

Begin
n:=n div Temp;
Tăng số mũ của Temp;
End;

Hiệu quả của cài đặt chủ yếu phụ thuộc vào công đoạn tìm số nguyên tố liền sau Temp. Cách nhanh nhất là dùng 1 mảng để lưu trữ các số nguyên tố từ 1 đến n. Với n=1 tỷ ta phải lưu $\Pi(109) = 50.847.534$ số nguyên tố nên ta không thể có đủ bộ nhớ để thực hiện điều này. Một cách khác là ta tuần tự tăng biến Temp và kiểm tra Temp có nguyên tố không, cách này không tốn bộ nhớ nhưng không khả thi vì thời gian thực hiện quá lâu. Một cách tự nhiên, ta muốn dung hoà cả hai phương pháp trên. Vì vậy ta sẽ tìm cách giảm khối lượng lưu trữ và số lần kiểm tra nguyên tố:

Bước 1: Kiểm tra n. Nếu n nguyên tố hoặc n=1thì vấn đề trở nên quá đơn giản. Nếu n là hợp số thì chuyển sang bước 2.

Bước 2: Vì n là hợp số nên ta phải có n=a*b ($1<a \le b$) suy ra $a \le \text{sqrt}(n) \le 215,5 < 1$. Vậy ta chỉ cần lưu các số nguyên tố nhỏ hơn 6341

Vì n< 231 nên số thừa số nguyên tố nhỏ hơn 10 (2*3*5*..31 > 231), do đó ta chỉ phải kiểm tra nguyên tố khoảng 10 lần

Chương trình của bài này như sau:

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q+,R+,S+,T-,V+,X+}
{$M 65500,0,655360}
{Phan tich so n<2^31 thanh thua so nguyen to}
uses crt;
```

const maxPrime=46341; {max Prime = $\sqrt{2^{31}}$ }

```
Fin='phantich.inp';
Fou='phantich.out';
var p:array[1..10] of longint;
q:array[1..10] of integer; { Số thừa số nguyên tố lớn nhất là 10? }
```

```
Prime:array[0..maxPrime+1] of boolean;
n:longint;
count:integer;
temp:word;
f,g:text;
procedure Eratosten;
var i,j:word;
begin
Fillchar(Prime,sizeof(Prime),true);
Prime[0]:=false;
Prime[1]:=false;
for i:=2 to trunc(sqrt(maxPrime)) do
if Prime[i] then
for j:=2 to (maxPrime div i) do
Prime[i*j]:=false;
end;
function isPrime(n:longint):boolean;
var i,step:word;
begin
isPrime:=false;
if (n
begin
if Prime[n] then isPrime:=true;
exit;
end;
if ((n mod 2=0) or (n mod 3=0)) then Exit;
i:=5;
step:=2;
while (i≤sqrt(n)) do
begin
if (n mod i=0) then
exit;
i:=i+step;
step:=6-step;
end;
isPrime:=true;
end;
procedure Solve;
begin
Assign(f,Fin);
Reset(f);
```

```
Readln(f,n);
Close(f);
Assign(g,Fou);
Rewrite(g);
if n<2 then
begin
writeln(g,0);
Close(g);
Halt;
end;
temp:=2;
count:=0;
Fillchar(p,sizeof(p),0);
Fillchar(q,sizeof(q),0);
while (n<>1) do
begin
if isPrime(n) then
begin
Inc(count);
p[count]:=n;
q[count]:=1;
exit;
end
else
begin
while ((n mod temp<>0) and (temp
begin
Inc(temp);
while not Prime[temp] do
Inc(temp);
end;
Inc(count);
p[count]:=temp;
while (n mod temp=0) do
begin
Inc(q[count]);
n:=n div temp;
end;
end;
end;
end;
```

```
procedure Result;
var i:integer;
begin
Writeln(g,count);
for i:=1 to count do
Writeln(g,p[i],' ',q[i]);
Close(g);
end;
BEGIN
Eratosten;
Solve;
Result;
END.
```

Sau khi làm bài toán 1 ta sẽ dễ dàng làm được bài toán số 2 trong đề khối 10 cuộc thi Olympic 30-4 lần VIII.

Đề bài như sau:

Cho số nguyên dương n (1 < n < 231). Tìm số nguyên dương a nhỏ nhất sao cho a a chia hết cho n.

Input: file Chiahet.inp chứa 1 số duy nhất n

Output: file Chiahet.out chứa số a

Bài toán 2: Tìm số mũ của số nguyên tố p trong cách phân tích của n! ra thừa số nguyên tố (n ≤ 231)

Cũng như bài toán 1, việc duyệt các số nguyên là bội của p từ 1 đến n để chia cho p tìm số mũ rồi lấy tổng các số mũ này là không thể thực hiên được.

Ví dụ:

Nếu n=231 còn p = 2 thì ta phải duyệt qua (n div p) số, đại lượng này bằng 230 =1.073.741.824 tức là hơn 1 tỷ số cần phải duyệt. Nội chỉ việc cho biến i chạy qua 230 số này mà chưa làm gì cả cũng đã mất khoảng 10 giây rồi (nếu không tin bạn hãy viết chương trình chỉ gồm 1 lệnh cho i chạy từ 1 đến 1 tỉ không làm gì cả và viết hàm đếm thời gian xem).

Như vậy ta cần phải tìm 1 cách khác để tính số mũ của p. Gọi si là số số chia hết cho pi trong khoảng 1..n. Số này bằng tổng của số các số chia hết cho p, p2, p3… trong khoảng 1..n. Dễ thấy số mũ của p là s1+s2+…+sk với k lớn nhất sao cho pk ≤ n

Ví dụ: n=29 và p =3 thì k=3, s1 =9, s2 =3, s3 =1 → Số mũ của 3 là s1+s2+s3 =13

Mặt khác các số chia hết cho pi trong khoảng 1..n là: 1* pi, 2* pi, 3* pi, …, $\left[\frac{n}{p^i}\right]$* pi → có $\left[\frac{n}{p^i}\right]$ số. Đây chính là nội dung của định lý Lơgiăngđrơ: Số mũ của số nguyên tố p trong phân tích n! thành các thừa số nguyên tố là:

$$\left[\frac{n}{p}\right]+\left[\frac{n}{p^2}\right]+\left[\frac{n}{p^3}\right]+\left[\frac{n}{p^4}\right]+\cdots$$

$$\left[\frac{n}{p^{i+1}}\right]=\left[\frac{\left[\frac{n}{p}\right]}{p^i}\right]$$

Để ý rằng (Số số chia hết cho pi+1 trong khoảng 1..n bằng số số chia hết cho pi trong khoảng 1..(n div p)). áp dụng điều này với i=1 ta có được chương trình hàm tính số mũ của p trong phân tích ra thừa số nguyên tố của n!

```
function SoMu(p:longint;n:longint):integer;
var count:integer;
begin
count:=0;
while n<>0 do
begin
n:=n div p;
Count:=count + n;
end;
SoMu:=count;
end;
```

Với k đủ lớn ta có $\left[\frac{n}{p^k}\right]$ hay n < pk. Vì vậy hàm SoMu chỉ phải vào vòng lặp k lần với k là số nguyên nhỏ nhất sao cho n < pk

$\Rightarrow k = \log_p n$

Cụ thể trong bài này trường hợp lớn nhất là n = 231; p = 2 thì
k = log2231 = 31 so với 230 (nhỏ hơn 34 triệu lần) !

Bài toán 3: Tìm số chữ số 0 tận cùng và chữ số tận cùng khác 0 của n! (n < 231). Dữ liệu nhập từ bàn phím số n và in kết quả số chữ số 0 tận cùng và chữ số tận cùng khác 0 của n! ra màn hình

Cơ sở thuật toán: Ta thấy n! khi phân tích ra thừa số nguyên tố thì có dạng
n! = 2SoMu(2,n) * 5SoMu(5,n) *…
Trong công thức Lơgiăngđrơ nếu p1< p2 thì SoMu(p1,n) ≥ SoMu(p2,n) nên số chữ số 0 tận cùng của n! chính là SoMu(5,n).
Đặt m = SoMu(2,n) - SoMu(5,n). Bỏ đi các chữ số 0 tận cùng thì ta có: Chữ số tận cùng khác 0 cua n! = [(2m mod 10) * TanCung(n)] mod 10 với TanCung(n) là chữ số tận cùng của n! đã bỏ đi hết số mũ của 2 và 5. Dễ thấy 2m mod 10 chỉ phụ thuộc vào (m mod 4), nếu m mod 4 = 0, 1, 2, 3 thì 2m mod 10 = 6, 2, 4, 8 (trừ trường hợp m = 0). Công việc còn lại là tìm hàm TanCung(n) (Nếu bây giờ chỉ nói tận cùng thì ta hiểu là số đó đã được chia cho 2 và 5 đến dư).

Ví dụ: Tận cùng của 10 là 1, của 14 là 7 …
Nếu n tương đối nhỏ thì ta có thể duyệt bằng 1 vòng for để tìm tận cùng của n!.
Nhưng n có thể lên tới 2 tỷ nên ta xây dựng hàm TanCung(n) như sau (với ví dụ n = 27):
Nếu n < 10 thì duyệt bình thường. Xét n ≥10, ta tính tận cùng của tích (n mod 10) số liên tiếp từ $\left[\frac{n}{10}\right]$ * 10 + 1 đến n
(temp=21*22*23*24*25*26*27= 21*11*23*3*1*13*27=1*1*3*3*1*3*7=9)
```
temp:=1;
for i:=n+1-n mod 10 to n do
begin
tg:=i;
while not odd(tg) do
tg:=tg shr 1;
while (tg mod 5=0) do
tg:=tg div 5;
temp:=temp*(tg mod 10) mod 10;
end;
temp:=temp mod 10;
n:=n-n mod 10;
```
Vậy TanCung(n) = [temp*TanCung(n − n mod 10)] mod 10.
Gán lại n:= n − n mod 10(n = 20). Ta phân hoạch các số từ 1 đến n thành 3 phần:
- Các số không chia hết cho 2 và 5: Các số này có tận cùng là 1, 3, 7, 9 (1, 3, 7, 9, 11, 13, 17, 19) mỗi loại có (n div 10) số nên tận cùng của tích các số này bằng tận cùng của (1*3*7*9)n div 10 = 9n div 10. Nếu (n div 10) chẵn thì số này bằng 1, lẻ thì bằng 9 (n=20, số này là 1)
if odd(n div 10) then temp:=temp*9;{else temp:=temp*1}
(n=20 → temp=1*9=9)
- Các số là bội của 2 có dạng 2, 2*2, 2*3,…, 2*(n div 2) (2, 4, 6, 8, 10, 12, 14, 16, 18, 20) Tận cùng của tích này cũng chính là tận cùng của 1*2*3*…*(n div 2) =TanCung(n div 2) (Vì đã loại đi tất cả các thừa số 2)
(n=20 ta có 1*2*3*…*10 = TanCung(10))
- Các số là bội của 5 nhưng lẻ (không là bội của 2) có dạng 5, 5*3, 5*5…5*t (t lẻ sao cho 5*t < n) (n=20 → t=3; gồm 2 số: 5, 15). Tận cùng của tích này bằng tận cùng của tích các số lẻ 1*3*5*7*…*t (n=20 là 1*3) (vì đã loại đi tất cả các thừa số 5). Ta tính nó bằng hàm TanCungLe(n) (Tận cùng của các số lẻ không quá n)
Theo cách phân hoạch ta có (các số n sau là đã gán lại n = n − n mod 10):
TanCung(n):=temp*TanCung(n div 2)*TanCungLe(n div 5) mod 10
TanCung(27)=9*TanCung(10)*TanCungLe(4) mod 10
Cách xây dựng hàm TanCungLe(n) cũng tương tự như trên: n <10 duyệt bình thường. Xét n ≥10, tính tận cùng của các số lẻ trong khoảng (n-n mod 10)..n lưu vào

biến temp. Gán lại n:= n − n mod 10. Ta phân hoạch các số lẻ từ 1..n thành 2 phần:
- Các số không chia hết cho 5 có tận cùng la 1, 3, 7, 9 mỗi loại có (n div 10) số tính như phần trên
if ođ(n div 10) then temp:=temp*9; {else temp:=temp*1}
- Các số là bội của 5 có dạng 5, 5*3, 5*5,…5*t (t lẻ sao cho 5*t < n)
Tận cùng tích t số này chính là TanCungLe(n div 5).
Ta có TanCungLe(n)=temp * TanCungLe((n- n mod 10) div 5) mod 10;
Chương trình bài này cài đặt như sau:

```
{Tim chu so cuoi cung khac khong cua n!}
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q+,R+,S+,T-,V+,X+}
{$M 65500,0,655360}
program Chu_so_tan_cung_khac_0_cua_n_giai_thua;
uses crt;
var n,m,tc:longint;
function TanCungLe(n:longint):integer;
var temp,tg,i:longint;
begin
if n<=10 then
begin
temp:=1;
for i:=1 to n do
if (ođ(i) and (i<>5)) then
temp:=temp*i;
TanCungLe:=temp mod 10;
Exit;
end;
if not ođ(n) then n:=n-1;
if ođ(n div 10) then
temp:=9
else temp:=1;
for i:=n+1-n mod 10 to n do
if ođ(i) then
begin
tg:=i;
while (tg mod 5=0) do
tg:=tg div 5;
temp:=temp*(tg mod 10) mod 10;
end;
temp:=temp mod 10;
n:=n-n mod 10;
TanCungLe:=temp*TanCungLe(n div 5) mod 10;
```

```
end;
function TanCung(n:longint):integer;
var temp,tg,i:longint;
begin
if n<=10 then
begin
temp:=1;
for i:=1 to n do
begin
tg:=i;
while not ođ(tg) do
tg:=tg shr 1;
if (tg mod 5 =0) then
tg:=tg div 5;
temp:=(temp*tg);
end;
TanCung:=temp mod 10;
Exit;
end;
if ođ(n div 10) then temp:=9
else temp:=1;
for i:=n+1-n mod 10 to n do
begin
tg:=i;
while not ođ(tg) do
tg:=tg shr 1;
while (tg mod 5=0) do
tg:=tg div 5;
temp:=temp*(tg mod 10) mod 10;
end;
temp:=temp mod 10;
n:=n-n mod 10;
TanCung:=temp*TanCung(n div 2)*TanCungLe(n div 5) mod 10;
end;
function SoMu(p:longintr;n:longint):longint;
var count:longint;
begin
count:=0;
while n<>0 do
begin
n:=n div p;
```

```
Count:=count+n;
end;
SoMu:=count;
end;
procedure Solve;
begin
Write('N=');Readln(n);
m:=SoMu(2,n)-SoMu(5,n);
if m=0 then tc:=1
else
case (m mod 4) of
0 : tc:=6;
1 : tc:=2;
2 : tc:=4;
3 : tc:=8;
end;
tc:=(tc*TanCung(n)) mod 10;
Writeln('Chu so tan cung khac 0 cua n! la : ',tc);
end;
BEGIN
Clrscr;
Solve;
Readln;
END.
```

Với n < 231 bất kì thì thời gian tính toán không quá 1 giây. Nếu làm bình thường bằng cách duyệt như với n ≤ 10 ở cài đặt trên thì chạy n = 10 triệu mất khoảng 73 giây (Gấp nhau đến hàng trăm lần). Như đã thấy, với cùng một bài toán vấn đề nhưng thuật toán và cách cài đặt khác nhau thì hiệu quả của chương trình cài đặt có thể sẽ khác biệt nhau rất rõ ràng. Vì vậy những kiến thức toán học để chứng minh thuật toán và chương trình là rất cần thiết khi ta muốn tối ưu hoá một thuật toán nào đó. Lần sau tôi sẽ xin trình bày với các bạn về những kiến thức số học được ứng dụng trong giải các bài toán tin.

25 .Spigot - Thuật toán tính số Π

Tác giả: Nguyễn Hồng Vũ

Những điều trình bày dưới đây có thể không mới vì nó đã được phát hiện ra từ hơn 10 năm trước, tuy nhiên, chắc chắn chúng vẫn có rất nhiều giá trị hữu ích cho các bạn yêu thích thuật toán và lập trình.

Mọi sự tình cờ với một trang Web 'Computing decimals of PI in Java', trên đó miêu tả cách tính hàng ngàn chữ số của π chỉ với vài trăm ký tự, viết bằng C:

```
int a=10000,b,c=8400,d,e,f[8401],g;
main()
{ for(;b-c;)f[b++]=a/5;
for(;d=0,g=c*2;c-=14,printf( %.4d ,e+d/a),e=d%a)
for(b=c;d+=f[b]*a,f[b]=d%--g,d/=g--,--b;d*=b); }
```

Kèm theo đoạn mã trên là một lời giải thích quá ngắn gọn: 'Chương trình tính 1000 π thông qua thuật toán spigot bằng cách lần lượt tính 4 chữ số của π. Nhiệm vụ chính của chương trình là chuyển đổi từ cách biểu diễn π trong hệ đếm hỗn hợp về hệ đếm 10000. Để tính được 4*k số π, cần 14*k phần tử mảng f. '

Đây quả là điều hết sức thú vị. Chương trình đã tính trực tiếp các chữ số của π, số sau không cần dựa theo các số trước (nhưng vẫn phải tính các số trước). Phát hiện này được Stanley Rabinowitz và Stan Wagon đăng thành bài 'A Spigot Algorithm for the Digits of Pi' trên tạp chí 'American Mathematical Monthly', số 102 (1995). 'Spigot' là tên gọi một họ các thuật toán cho phép tính trực tiếp các chữ số của số vô tỷ. Đây chỉ là tên gọi chung nêu lên tư tưởng thuật toán. Việc cài đặt đối với từng trường hợp cụ thể sẽ khác nhau tùy cách tiếp cận vấn đề.

Trước khi đi sâu vào thuật toán, ta chuyển đổi chương trình C trên thành dạng Pascal quen thuộc:

```
var
a,b,c,d,e,g, kq:longint;
f:array[0..8400]of longint;
begin
a:=10000; b:=0; c:=8400; e:=0;
for b:=0 to c do f[b]:=a div 5;
while(c>0)do
begin
d:=0;
g:=c*2;
b:=c;
while(b>0) do
begin
d:=d+f[b]*a;
g:=g-1;
f[b]:=d mod g;
d:=d div g;
g:=g-1;
b:=b-1;
```

```
if(b>0)then d:=d*b;
end;
c:=c-14;
kq:=e+d div a;
if kq>999 then
write(kq)
else if kq>99 then
write('0',kq)
else if kq>9 then
write('00',kq)
else
write('000',kq);
e:=d mod a;
end;
end.
```

1. Mở rộng khái niệm hệ đếm.

Khi làm quen với máy tính, chắc chắn trong chúng ta ai cũng biết đến hệ đếm và thuật toán chuyển đổi giữa các hệ đếm. Tuy nhiên, chúng ta cũng chỉ quan tâm đến các hệ đếm nguyên như 2,3,10,16...Bây giờ, chúng ta làm quen với một dạng hệ đếm khác, tạm gọi là hệ đếm hỗn hợp (mixed-radix).

Ta nói số $s = \overline{a_0 a_1 a_2 a_3 a_4 \ldots}$ trong hệ đếm cơ số $b = \left(\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5} \ldots\right)$, ký hiệu $(a_0; a_1, a_2, a_3, a_4 \ldots)_b$ là số được biểu diễn như sau:

$$a_0 + \frac{1}{2}\left(a_1 + \frac{1}{3}\left(a_2 + \frac{1}{4}\left(a_3 + \frac{1}{5}(a_4 + \ldots)\right)\right)\right)$$

Nếu $0 \le a_i \le i$, với $i \ge 1$, biểu diễn đó gọi là chính quy (regular).

Người ta đã chứng minh được rằng:

Định lý 1: Nếu $i \ge 1$, $(0; 0, 0, \ldots 0, a_i, a_{i+1}, \ldots)_b < \frac{1}{i!}$ và $(0; a_1, a_2, \ldots)_b < 1$ thì phần nguyên của $(a_0; a_1, a_2, a_3, a_4 \ldots)_b$ là a0 còn phần thập phân là $(0; a_1, a_2, \ldots)_b$

Như vậy, để tính được các chữ số của s, trước hết lấy ra phần nguyên ao, tiếp theo nhân phần thập phân với 10 để lấy ra phần nguyên và tiếp tục lặp đi lặp lại như thế.

Ví dụ với số e, ta có: $e = \sum \frac{1}{i!}$ hay $e = 1 + \frac{1}{1}\left(1 + \frac{1}{2}\left(1 + \frac{1}{3}\left(1 + \frac{1}{4}(1 + \ldots)\right)\right)\right)$

Thực hiện các phép biến đổi sau đây:

$$e = 1 + \frac{1}{1}\left(1 + \frac{1}{2}\left(1 + \frac{1}{3}\left(1 + \frac{1}{4}\left(1 + \frac{1}{5} \times 1\right)\right)\right)\right)$$

$$e = \frac{1}{10}\left(10 + \frac{1}{1}\left(10 + \frac{1}{2}\left(10 + \frac{1}{3}\left(10 + \frac{1}{4}\left(10 + \frac{1}{5} \times 10\right)\right)\right)\right)\right)$$

$$e = \frac{1}{10}\left(10 + \frac{1}{1}\left(10 + \frac{1}{2}\left(10 + \frac{1}{3}\left(10 + \frac{1}{4}\left(12 + \frac{1}{5} \times 0\right)\right)\right)\right)\right)$$

$$e = \frac{1}{10}\left(10 + \frac{1}{1}\left(10 + \frac{1}{2}\left(10 + \frac{1}{3}\left(13 + \frac{1}{4}\left(0 + \frac{1}{5} \times 0\right)\right)\right)\right)\right)$$

$$e = \frac{1}{10}\left(10 + \frac{1}{1}\left(10 + \frac{1}{2}\left(14 + \frac{1}{3}\left(1 + \frac{1}{4}\left(0 + \frac{1}{5} \times 0\right)\right)\right)\right)\right)$$

$$e = \frac{1}{10}\left(10 + \frac{1}{1}\left(17 + \frac{1}{2}\left(0 + \frac{1}{3}\left(1 + \frac{1}{4}\left(0 + \frac{1}{5} \times 0\right)\right)\right)\right)\right)$$

$$e = \frac{1}{10}\left(20 + 7 + \frac{1}{1}\left(0 + \frac{1}{2}\left(0 + \frac{1}{3}\left(1 + \frac{1}{4}\left(0 + \frac{1}{5} \times 0\right)\right)\right)\right)\right)$$

$$e = 2 + \frac{1}{10}\left(7 + \frac{1}{1}\left(0 + \frac{1}{2}\left(0 + \frac{1}{3}\left(1 + \frac{1}{4}\left(0 + \frac{1}{5} \times 0\right)\right)\right)\right)\right)$$

Vậy chữ số thứ nhất của e là 2. Phần thập phân của e là

$7 + \frac{1}{1}\left(0 + \frac{1}{2}\left(0 + \frac{1}{3}\left(1 + \frac{1}{4}\left(0 + \frac{1}{5} \times 0\right)\right)\right)\right)$ hay $(7; 0, 0, 1, 0, 0)_b$

áp dụng lặp lại phép biến đổi như trên với phần thập phân, ta sẽ thu được lần lượt các chữ số tiếp theo của e. Hãy thử viết bằng Pascal:

```
var
a,b,c,d,g:longint;
f:array[1..6]of longint;
begin
a:=10;c:=6;g:=5;d:=0;
f[1]:=1;f[2]:=1;f[3]:=1;f[4]:=1;f[5]:=1;f[6]:=1;
for b:=c downto 2 do
begin
d:=d+f[b]*a;
f[b]:=d mod g;
d:=d div g;
g:=g-1;
end;
d:=d+f[1]*a;
```

f[1]:=d mod a;
writeln(d div a);
end.
Dễ dàng nhận thấy đoạn chương trình này tương tự với vòng lặp bên trong của thuật toán tính π và chương trình sẽ in ra 2, các giá trị tiếp theo của mảng f là (7,0,0,1,0,0).
Do e là số thập phân vô hạn, số chữ số tính được của e phải phụ thuộc vào chiều dài mảng f.
Định lý 2:Để tính được n chữ số của e theo hệ đếm cơ số 10, sẽ cần mảng f có chiều dài tối thiểu n+2 nếu n< 28 hoặc n nếu n>28 (chú ý: $28 = \lceil 10e \rceil$).
Như vậy bản chất của thuật toán tính e hay π là việc chuyển đổi từ hệ đếm cơ số hỗn hợp về hệ đếm cơ số 10. Mỗi lần lặp cho ta một chữ số.
2. Thuật toán dành cho π
Công thức dùng cho tính toán với π là chuỗi:

$$\frac{\pi}{2} = \sum_{i=0}^{\infty} \frac{i!}{(2i+1)!!} = 1 + \frac{1}{3}\left(1 + \frac{2}{5}\left(1 + \frac{3}{7}\left(1 + \frac{4}{9}(1 + \ldots)\right)\right)\right)$$

Trong đó k!! là ký hiệu của tích k số lẻ liên tiếp nhau từ 1 đến k: k!!=1.3.5..k. Chúng ta bỏ qua về nguồn gốc của chuỗi.
Từ công thức này có thể thấy π được biểu diễn dưới dạng cơ số hỗn hợp là:

$$(2;2,2,2\ldots)_c \text{ với } c = \left(\frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \ldots\right)$$

Các tính chất của hệ đếm c không hoàn toàn tương tự như đối với hệ đếm b.
Định lý 3:Số $(0;2,4,6,8,\ldots)_c$ với tối đa các chữ số biểu diễn số 2, do đó, số $(0;a,b,c,\ldots)_c$ theo dạng chính quy sẽ rơi vào trong khoảng 0 và 2.
Định lý trên cho thấy rằng phần nguyên của $(a_0;a_1,a_2,a_3,\ldots)_c$ có thể là ao hoặc aơ1 tùy theo $(0;a_1,a_2,a_3,\ldots)_c$ rơi vào khoảng $[0,1) \text{ hay } [1,2)$. Vì vậy, khi áp dụng các nguyên tắc của e cho π cần có một vài thay đổi:
Trước khi in ra một chữ số vừa tìm được, cần phải giữ lại để kiểm tra xem chữ số thứ tiếp theo rơi vào khoảng nào. Nếu cần thiết chữ số đó phải được điều chỉnh rồi mới in ra. Về thực tế, chữ số tìm được tiếp theo có thể lớn hơn 10, trong trường hợp này, phải cộng thêm 1 vào số vừa tìm được trước. Tuy nhiên, điều gì sẽ xảy ra nếu số trước đó đã là 9? Rõ ràng phải cộng dồn lên trước nữa 1 đơn vị và thay 9 thành 0. Các tác giả đề xuất phương án lưu chữ số tìm được trong một biến gọi là predigit, đồng thời sử dụng một biến đếm các chữ số 9 liên tiếp nhau. Khi xuất hiện tình huống chữ số tìm được tiếp theo lớn hơn 10, tiến hành biến đổi predigit và các số 9 đứng trước. Xin lưu ý thêm, khác với hệ đếm b, các phân số trong hệ đếm c có tử số lớn hơn 1, do đó phép biến đổi cũng hơi khác một chút. Sau đây là thuật toán tính n

chữ số của π:
Sử dụng biến predigit để lưu chữ số tìm được. nines để đếm số các số 9 liên tiếp. Khởi tạo bảng f với các phần tử có giá trị là 2. Lặp lại n lần: Nhân mỗi phần tử của bảng f với 10. Bắt đầu từ bên phải, chia f[i] cho 2*i-1, nhận được phần nguyên q và phần dư r. Gán r cho f[i], cộng dồn q*(i-1) vào f [i-1]. Lấy phần tử trái nhất của f chia 10, giữa lại phần dư trong f, được phần nguyên q. Nếu q≠9 và q≠10, in ra chữ số đang có trong predigit và lưu q vào predigit. Tiếp theo in nines số 9 và gán nines bằng 0. Nếu q =9, tăng nines lên 1. Nếu q =10: Tăng predigit hiện tại lên 1 và in giá trị ra. Gán predigit thành 0. In ra nines các số 0. Gán nines bằng 0. Đoạn chương trình sau bằng Pascal miêu tả thuật toán vừa rồi:

```
const n =1000;
len=10*n div 3;
var i, j, k, q, x, nines, predigit: integer;
a: array[1..len] of longint;
begin
for j:=1 to len do a[j]:=2;
nines:=0; predigit:=0;
for j:=1 to n do
begin
for i:=len downto 1 do
begin
x:=10*a[i]+q*i;
a[i]:=x mod (2*i-1);
q:=x div (2*i-1);
end;
a[1]:=q mod 10; q:=q div 10;
if q=9 then
nines:=nines+1
else
if q=10 then
begin
write(predigit+1);
for k:=1 to nines do write('0');
predigit:=0; nines:=0;
end
else
begin
write(predigit); predigit:=q;
for k:=1 to nines do write('9');
nines:=0;
```

```
end
end;
writeln(predigit);
end.
```

Định lý 4:Để tính được n chữ số của π theo thuật toán trên cần mảng f có $\lfloor 10n/3 \rfloor$ phần tử.

Nếu chương trình trên được thi hành, đến vòng lặp thứ 32, một tình huống cần xử lý predigit sẽ xuất hiện:

1	2	3																									30	31	32	33	34
30	13	41																									94	**49**	**102**	28	87
3	1	4																									9	**5**	**0**	2	8

Có thể cải tiến thêm tốc độ tính toán bằng cách thay vì mỗi vòng lặp tìm một chữ số, hãy tìm nhiều chữ số hơn. Việc này rất đơn giản, chỉ cần đổi sang hệ đếm lũy thừa của 10 như 10.000 chẳng hạn. Tuy nhiên, để có thêm 3 chữ số nữa, ta tính 1000 p. Mặt khác, nếu biết trước rằng trong quá trình tính toán không bao giờ xuất hiện số có một phần là 000 thì hoàn toàn có thể bỏ qua đoạn xử lý predigit.

Đến đây, chương trình C ban đầu đã trở nên dễ hiểu:

Vòng lặp thứ nhất khởi tạo các giá trị của mảng f thành 2000 (do tính 1000π). a là biến cho biết hệ đếm cần chuyển về. b, g là biến chỉ ra tử số, mẫu số của các phân số trong hệ đếm hỗn hợp. Điều băn khoăn cuối cùng có lẽ là con số 14. Như đã trình bày ở trên, người ta chứng minh rằng:

Để có thể tính chính xác p chữ số của π, với p đủ lớn, cần sử dụng n chữ số của π trong hệ đếm hỗn hợp sao cho 2n >10 p hay n/p >ln10/ln2. Trong trường hợp cụ thể này, vì tính với hệ đếm 10.000 nên n cần thỏa mãn: n/p>ln10000/ln2 xấp xỉ bằng 14 hay n>14p

3. Nói thêm

Mặc dù thuật toán trên không có tính cạnh tranh so với các thuật toán tính π đang được sử dụng rộng rãi nhưng nó rất đơn giản và có thể dễ dàng cài đặt trên các máy tính cá nhân. Tư tưởng của thuật toán này cũng có thể được áp dụng đối với nhiều loại số vô tỷ khác nữa như $\sqrt{2}$., ln2 ...Một câu hỏi cũng đặt ra là liệu có thể tìm được một cách biểu diễn π trong hệ đếm d nào đó sao cho ao luôn là phần nguyên của $(a_0; a_1, a_2, a_3, \ldots)_d$? Hay có thể xác định được trong những trường hợp nào thì phải xử lý predigit và khi nào thì có thể bỏ qua…?

Hiện nay người ta đã có rất nhiều cách để tính trực tiếp từng chữ số của π, thậm chí không cần tính các số trước đó, với độ phức tạp tương đối thấp. Ngoài hệ đếm 10 hay lũy thừa 10, π còn có thể được tính cho các hệ đếm khác như 2, 16.... Tất nhiên với cách này thì sẽ phải mất thêm một công đoạn chuyển đổi tiếp về hệ đếm thập phân.

Phần dưới đây xin được giới thiệu tiếp một số đoạn chương trình rất ngắn tính

những số vô tỷ khác. Mã nguồn được sao chép từ Internet và chưa có điều kiện để kiểm tra, đưa ra với mục đích tham khảo.
Chương trình tính π với 142 ký tự
```
main(){int a=1e4,c=3e3,b=c,d=0,e=0,f[3000],g=1,h=0;
for(;b;!--b?printf( %04d ,e+d/a),e=d%a,h=b=c-=15:f[b]=
(d=d/g*b+a*(h?f[b]:2e3))%(g=b*2-1));}
```
Chương trình tính e
```
main(){int N=9009,n=N,a[9009],x;while(--n)a[n]=1+1/n;
for(;N>9;printf( %d ,x))
for(n=N--;--n;a[n]=x%n,x=10*a[n-1]+x/n);}
```
Chương trình tính log10
```
main(){int a=1000,b=0,c=2658,d=75,f[2659],n=800,k;
for(;b
for(;n--;d+=*f*a,printf( %.3d ,d/a),*f=d%a)
for(d=0,k=c;--k;d/=b,d*=k)f[k]=(d+=f[k]*a)%(b=8*k+4);}
```
Chương trình tính $\sqrt{2}$
```
main(){int a=1000,b=0,c=1413,d,f[1414],n=800,k;
for(;b
for(;n--;d+=*f*a,printf( %.3d ,d/a),*f=d%a)
for(d=0,k=c;--k;d/=b,d*=2*k-1)f[k]=(d+=f[k]*a)%(b=100*k);}
```
Bài viết này được xây dựng từ một loạt các tài liệu và liên quan rất nhiều đến toán học. Các định lý trong bài chúng ta tạm công nhận mà bỏ qua phần chứng minh. Nếu có phần nào người viết hiểu sai vấn đề hay ngộ nhận thì rất mong được thông cảm. Xin tham khảo thêm phần tài liệu sử dụng dưới đây để có nhiều thông tin hơn.
Stanley Rabinowitz, Stan Wagon: 'A Spigot Algorithm for the Digits of Pi' − 'American Mathematical Monthly', 102 (1995)
Erik Bruchez: Computing decimals of PI in Java: www.bruchez.org/erik
Các tài liệu sau lấy từ: numbers.computation.free.fr/Constants
Xavier Gourdon: Computation of the n-th decimal digit of π with low memory
Xavier Gourdon and Pascal Sebah: N-th digit computation W. Hereman: Various Methods to Compute the Number π

26 .Ứng dụng Lý thuyết đồng dư trong Tin học

Tác giả: Đinh Đức Hùng

Như các bạn đã biết, trong thực tế có nhiều khi chúng cần quan tâm không chỉ các số nguyên mà cả số dư của số nguyên đó trong phép chia cho một số nguyên dương. Chẳng hạn ta muốn biết sau 50 giờ nữa sẽ là mấy giờ thì chỉ cần tính số dư của 50 cho 24 và cộng thêm giờ hiện thời rồi chia cho 24!

Lý thuyết đồng dư là một phần quan trọng trong toán phổ thông. Trong bài viết này, chúng tôi chỉ đề cập đến hai ứng dụng nhỏ của đồng dư: Bài toán tạo các số ngẫu nhiên bằng máy vi tính và Bài toán mã hoá thư từ.

Một số kiến thức về đồng dư có liên quan

Trước hết chúng ta cùng xem lại một số kiến thức về đồng dư:

Định nghĩa 1: Cho a là một số nguyên, m là một số nguyên dương. Khi đó, r được gọi là số dư trong phép chia a cho m nếu tồn tại số nguyên q, r sao cho: a = mq+r, ,0≤r.Trong lập trình Pascal để lấy số dư r chúng ta sử dụng phép toán mod: r:= a mod m;

Ví dụ: 3 mod 2=1; 4 mod 2=0; 5 mod 3=2.

Định nghĩa 2: Cho a, b là hai số nguyên, m là một số nguyên dương. Lúc đó, a được gọi là đồng dư với b theo môđun m nếu số dư trong phép chia a cho m bằng số dư trong phép chia b cho m. Kí hiệu: a≡b (mod m)

Từ định nghĩa này ta có: a ≡ b (mod m)□ a-b chia hết cho m;

Ví dụ: 5 ≡ 2 (mod 3)□ 5-2 chia hết cho 3; 11≡7 (mod 4)□11-7 chia hết cho 4;

Định lý 1: Cho m là số nguyên dương a đồng dư với b theo mođun m khi chỉ khi tồn tại số nguyên k sao cho a=b+km.

Định lý 2: Nếu a≡b (mod m); c≡d (mod m) thì a+c≡b+d (mod m) và ac≡bd (mod m).

Bài toán 1: Tạo các số ngẫu nhiên bằng máy vi tính

Thông thường, chúng ta dùng thuật ngữ ngẫu nhiên khi nói đến một sự tuỳ ý nào đó. Thí dụ khi bạn yêu cầu cho một số ngẫu nhiên nghĩa là cho một số nào cũng được tuỳ ý. Nhưng trong toán học, số ngẫu nhiên được định nghĩa là một số đều có khả năng xuất hiện tương đương nhaụ

Trong lập trình Pascal các bạn vần dùng hàm Random để sinh các số ngẫu nhiên trong một miền nào đó. Nhưng đã bao giờ các bạn đặt ra câu hỏi người ta sinh ra các số ngẫu nhiên đó bằng cách nàỏ Có nhiều phương pháp để tạo ra các số có tính chất gần giống với tính chất của số ngẫu nhiên bởi vì việc tạo ra các số ngẫu nhiên là điều không thể. Một chương trình do chúng ta viết thì chắc chắn là các số được tạo ra có thể suy luận được. Vì vậy, các số do chúng ta tạo ra chỉ có thể gọi là giả ngẫu

nhiên. Tuy vậy, chúng ta hãy bằng lòng với việc sử dụng những số giả ngẫu nhiên đó như những số ngẫu nhiên.

Một phương pháp phổ biến được đưa ra bởi D.Lehner năm 1951 để tạo ra các số giả ngẫu nhiên là phương pháp đồng dư tuyến tính.

Tư tưởng của phương pháp này có thể trình bày ngắn gọn như sau: Ta chọn 4 số nguyên là môđun m, nhân tử a, số gia c và số hạt giống x0 , với điều kiện 2 của m, $0 < c < m$ và $0 < x0 < m$.

Chúng ta sẽ tạo ra các dãy số ngẫu nhiên $\{xn\}$, $0 \leq xn < m$

bằng cách sử dụng liên tiếp các phép số dư: xn+1 = axn+c) mod m.

Ví dụ: Với m=9, a=7, c=4, x0 = 3 thì dãy số giả ngẫu nhiên được tạo ra như sau:

x1 = (7x0 + 4) mod 9 = (7.3 + 4) mod 9 = 25 mod 9 = 7.

x2 = (7x1 + 4) mod 9 = (7.7 + 4) mod 9 = 53 mod 9 = 8.

x3 = (7x2 + 4) mod 9 = (7.8 + 4) mod 9 = 60 mod 9 = 6.

x4 = (7x3 + 4) mod 9 = (7.6 + 4) mod 9 = 46 mod 9 = 1.

x5 = (7x4 + 4) mod 9 = (7.1 + 4) mod 9 = 11 mod 9 = 2.

x6 = (7x5 + 4) mod 9 = (7.2 + 4) mod 9 = 18 mod 9 = 0.

x7 = (7x6 + 4) mod 9 = (7.0 + 4) mod 9 = 4 mod 9 = 4.

x8 = (7x7 + 4) mod 9 = (7.4 + 4) mod 9 = 32 mod 9 = 5.

x9 = (7x8 + 4) mod 9 = (7.5 + 4) mod 9 = 39 mod 9 = 3.

x10 = (7x9 + 4) mod 9 = (7.3 + 4) mod 9 = 25 mod 9 = 7.

Dãy trên chứa 9 phần tử khác nhau trước khi lặp lại

Thực tế là khi chúng ta chọn các số m, a, c, x0 một cách tuỳ ý không theo một quy luật nào thì có thể dẫn đến trường hợp tạo ra được dãy số có chu kì quá nhỏ so với miền xác định của nó (mođun m). Một thí dụ trong trường hợp ta chọn m= 381, a=19, c=1, x0 = 0 dãy số giả ngẫu nhiên được tạo ra là: 0, 1, 20, 0, 1, 20,... Rõ ràng,

đây là dãy có chu kỳ 3, không ngẫu nhiên trong miền 0...381. Như vậy, làm thế nào để tạo ra một dãy số giả ngẫu nhiên tốt D.E.Knuth đã đưa ra và phát triển quy luật chọn m, a, c, x0 như sau:

+ Mođun m có thể lớn đến giá trị tối đa của một Word (thường là luỹ thừa của 10 hoặc 2);

+ Nhân tử a không quá lớn hay quá nhỏ (tốt hơn hết là ít hơn m một chữ số; a nên là một hằng số tuỳ ý không theo một mẫu riêng nào cả và nên kết thúc bởi x21 với x là một số chẵn.

+ Giá trị hạt giống x0 có thể lấy bất kỳ;

Người ta thấy rằng: với cách chọn đó dãy số tạo ra tương đối tốt theo nghĩa đã nói

Cài đặt: Ta có thể dùng mảng hoặc dùng một biến cập nhật liên tục các phần tử của dãy số giả ngẫu nhiên. Nhưng trong cả hai trường hợp đó để chương trình có thể chạy tốt ngay khi a và x0 tương đối lớn chúng ta cần giải quyết vấn đề tràn.

Sau đây chúng ta lấy một ví dụ sẽ thấy ngay điều đó: Giả sử máy tính của chúng ta có Word 32 bit nhưng ta chọn như sau: m=100000000, a=31415821, x0 = 1234567, c=1 đều là những giá trị nhỏ hơn giá trị tối đa của Word. Thế nhưng ngay giá trị của phép tính đầu tiên a.x 0 = 38784935884507 đã vượt quá giá trị giới hạn của Word (mà ở đây chúng ta chỉ quan tâm đến giá trị (a.x0 + 1) mod m nhỏ hơn giá trị Word).

Giải pháp: Giả sử P.Q vượt ra ngoài phạm vi m ở trên chúng ta sẽ làm như sau: Ta biểu diễn: P=104.P1+P0; Q=104.Q1+Q0; Bây giờ P.Q=(104.P1 + P0)(104.Q1 + Q0) =108.P1.Q1 + 104(P1.Q0 + P0.Q1) + P0.Q0. Trong tổng này chúng ta chỉ quan tâm đến 4 số cuối của số hạng 104(P1.Q0+P0.Q1) và ta có: P.Q mod m = (((P1.Q0+P0.Q1) mod 104).104 + P0.Q0) mod m. Các bạn hoàn toàn có thể chứng minh được điều này

Sau đây là một cách cài đặt hàm surplus(p,q) để lấy số dư của p.q cho 108:

```
Function Surplus(p,q:longint):longint;

Const m=100000000;

m1=10000;

Var q1,q0,p1,p0:longint;
```

Begin

Q1:=q div m1; q0:=q mod m1;

P1:=p div m1; q0:=q mod m1;

Surplus:=(((P1.Q0+P0.Q1) mod m1).m1 + P0.Q0) mod m;

End;

Cuối cùng ta có đoạn chương trình tạo n số ngẫu nhiên

For i:=1 to n do write((surplus(a,x0)+1) mod m);

Chương trình trên sẽ không bị tràn với $m \leq 1/2$ giới hạn tối đa của số nguyên. Muốn tạo các số ngẫu nhiên nằm trong khoảng (0,1) ta chỉ việc chia dãy đã tạo được ở trên cho mođun m. Mời các bạn hãy thử xem!

Bài toán 2: Bài toán mật mã

Trong nhiều lĩnh vực truyền thông thì việc giữ bí mật thông tin giữa người gửi và người nhận luôn được đặt ra Chẳng hạn những bức điện tín, thư tín trong quân sự. Mật mã học là một ngành khoa học chuyên nghiên cứu các thư từ bí mật. Một trong những người sử dụng mật mã được biết đến sớm nhất là Julius Caesar (Xê-da). Cách làm của Caesar như sau: ông dịch mỗi chữ cái ở bức thư gốc đi ba chữ về phía sau A -> D, B -> E,..., X -> A, Y -> B, Z -> C.

Chúng ta sẽ xem xét quá trình mã hoá của Caesar một cách toán học: Trước tiên ta thay các chữ cái bằng các số p từ 0 đến 25: A -> 0, B -> 1,..., Z -> 25. Để mã hoá bức thư ta thay p bằng giá trị f(p) thuộc {0,1, 2,..., 23, 24, 25} được xác định như sau: f(p) = (p+3) mod 26 và dịch ngược trở lại các kí tự.

Ví dụ: Ta phải mã hoá bức thư sau: "I LOVE YOU MORE THAN I CAN SAY" bằng mật mã Caesar ta tiến hành như sau:

- Chuyển các chữ thành các số: 8 11 14 21 4 24 14 20 12 14 17 4 19 7 0 13 8 2 0 13 18 0 24

- Thay p bởi f(p) ta được:11 14 17 24 17 1 17 23 15 17 20 7 22 10 3 16 11 5 3 16 21 3 1

- Thay ngược trở lại các chữ cái ta được bức thư bí mật như sau: L ORYH BRX PRUH WKDQ LXQ VDB

Người nhận bức thư này phải tiến hành giải mã bức thự Thực chất của việc giải mã bức thư được mã hoá bằng mật mã Caesar là sử dụng hàm ngược: f-1(p) = (p-3) mod 26. Nói cách khác là ta tiến về đầu 3 chữ cái

Chúng ta có thể tổng quát hoá phương pháp của Caesar bằng cách thay vì dịch 3 chữ cái ta dịch đi k chữ, lúc đó: f(p) = (p+k) mod 26 và f -1(p) =(p-k) mod 26. Các loại mật mã này được gọi là mật mã dịch. Độ an toàn của nó không cao

Một phương pháp nâng cao độ an toàn là sử dụng hàm f(p)=(ạp+b) mod 26. Trong đó a,b được chọn sao cho f là song ánh. Chẳng hạn ta có thể dùng hàm f(p)=(7.p+3) mod 26. Lúc đó, A -> D, B ->K,...,Y ->P, Z ->W. Trong trường hợp này để giải mã bức thư sẽ không đơn giản như trên. Thật vậy, muốn giải mã bức thư ta phải tiến hành như sau: giả sử f(p) là số biểu diễn chữ cái trong bức thư được mã hoá ta xét các khả năng số nguyên k =0, 1, 2,..., 6. Nếu tồn tại k sao cho (26.k-3 +f(p)) chia hết cho 7 thì chữ số biểu diễn chữ cái trong bức thư gốc là: p = (26.k-3 + f(p)) div 7. Sau đây là chương trình mật mã.

```
Program chuong_trinh_MAT_MA;

Var chon: integer;

Procedure Mahoa;

Var ch,kt:char;f,ma:text;i,ii:integer;s,ss:string;

Begin

writeln('cho ten buc thu can ma hoa:');readln(s);

writeln('thanh buc thu:');readln(ss);

assign(f,s);reset(f);

assign(ma,ss);rewrite(ma);

while not(eof(f)) do

begin
```

```
read(f,ch);
if ch<>' ' then
begin
i:=ord(upcase(ch))-65;
ii:=(7*i+3) mod 26;
kt:=chr(ii+65);
write(ma,kt);
end
else write(ma,ch);
end;
close(f);close(ma);
end;
Procedure giaimathu;
Var f,gma:text;ch,kt:char;
s,ss:string;i,ii,k:integer;ok:boolean;
Begin
writeln('hay cho ten cua buc thu can giai ma:');
readln(s);
writeln('tro thanh buc thu:');
readln(ss);
assign(f,s);
```

```
reset(f);
assign(gma,ss);
rewrite(gma);
while not(eof(f)) do
begin
read(f,ch);
if ch<>' ' then
begin
ii:=ord(ch)-65;
k:=0;ok:=false;
while (k<7) and (not ok) do
begin
if (26*k+ii-3) mod 7 =0 then
begin
i:= (26*k+ii-3) div 7;
kt:=chr(i+65);
write(gma,kt);
ok:=true;
end
else k:=k+1;
end;
```

```
end
else write(gma,ch);
end;
close(f);close(gma);
end;
{----------CHUONG TRINH CHINH----------------}
begin
Writeln('Moi ngai chon chuc nang can lam: ');
Writeln('1: Ma hoa thú);
Writeln('2: Giai ma thú);
Writeln('0: exit');
Write('Chon: ');
Readln(chon);
Case chon of
1: Mahoa;
2: giaimathu;
0:halt;
else writeln('moi chon laí);
end;
end.
```

27 .**Xử lý biểu thức số học**

Lê Văn Chương

Biểu thức số học được định nghĩa rất đơn giản và ngắn gọn như sau: Biểu thức số học là một chuỗi ký tự bao gồm:

- Một số nguyên không dấu hoặc có dấu, hoặc một tên biến đặt theo chuẩn của Pascal.

- Hai biểu thức số học nối với nhau bởi một dấu phép tính (+, -, *, hoặc /). Có thể đặt trong cặp dấu ngoặc đơn

Khi cho một biểu thức thì thông thường ta phải kiểm tra xem biểu thức đã cho có là một biểu thức đúng không. Để kiểm tra một biểu thức xem nó có đúng không ta cần kiểm tra xem các số của nó có đúng không?, các biến trong đó có được đặt tên theo chuẩn của pascal không? các phép tính có hợp logic toán học không? các dấu ngoặc đơn có đúng không?

Trong các điều kiện trên thì kiểm tra số, tên biến, các dấu chúng ta không để cập đến vì chúng quá dễ, một người mới bặt đầu học cũng có thể xử lý được. Chúng ta chỉ quan tâm đến xử lý các dấu phép tính và ngoặc đơn, đặc biệt là dấu ngoắc đơn bởi vì nó phức tạp và khó xử lý nhất.

Bây giờ, chúng ta sẽ xem xét một bài toán được gọi là đơn giản nhất trong xử lý biểu thức số học. Nội dung của bài toán như sau:

Bài toán 1: Lập trình chương trình nhập vào một xâu ký tự chỉ gồm các dấu mở ngoặc và đóng ngoặc đơn, sau đó kiển tra tính đúng đăn của cách đặt dấu ngoặc. Một xâu ký tự đúng là xâu thỏa mãn:

- Số lần mở ngoặc đúng bằng số lần đóng ngoặc.

- Dấu mở ngoặc phải đứng trước (ở phía bên trái) dấu đóng ngoặc tương ứng.

Bài toán xử lý không khó nhưng nó là nền tảng để chúng ta áp dụng vào các bài toán có mức độ cao hơn. Với bài này ta chỉ cần đọc dòng dữ liệu và tính số lần mở ngoặc có bằng số lần đóng ngoặc không? nếu không thì đó là biểu thức sai, nếu có thì kiểm tra tiếp dấu mở ngoặc có đứng trước dấu đóng ngoặc tương ứng không. Sau đây là đoạn chương trình chính.

```
For i:=1 to length(s) do

Begin
```

```
If s[i] in [ ( , ) ] then

Begin

If s[i]= ( then inc(mn) else dec(mn);

If mn<0 then begin Writeln( Bieu thuc sai );Halt;End;

End;

If mn<>0 then begin Writeln( Bieu thuc sai );Halt;End;

Writeln( Bieu thuc dung );

End;
```

Chúng ta có thể nâng cấp chúng trình trên để kiểm tra một biểu thức đầy đủ nhập vào có đúng không, đây chính là nội dung của bài toán 2.

Bài toán 2 : Cho một xâu ký tự hãy kiểm tra xem xâu ký tự đó có là một biểu thức số học đúng hay không, biểu thức được xem là đúng nếu nó thỏa mãn các điều kiện sau:

- Một số nguyên, hoặc một tên biến đặt theo chuẩn của pascal.

- Hai biểu thức số học nối với nhau bởi một dấu phép tính (+, -, *, hoặc /), có thể đặt trong một cặp dấu ngoặc đơn.

Với bài toán này, chúng ta không chỉ kiểm tra dấu ngoặc đơn như bài trước mà còn kiểm tra tính đúng đắn của các phép tính, các số, các biến, kiểm tra các dấu phép tính có phù hợp không?

Thuật toán: Sử dụng biến dem để kiểm tra các dấu ngoặc đơn có đúng không và biến dau để kiểm tra tính đúng đắn của dấu. Cách kiểm tra như sau:

Khởi tạo dem:=0 và dau:=1;

Gọi i là ký tự đang xét của xâu s:

- Nếu i là dấu '(' và biến dau có giá trị bằng 0 thì đặt biến tăng biến dem lên 1;

- Nếu i là dấu ')' và trước i là một dấu phép tính thì chắc chắn đây là một biểu thức sai, ngược lại giảm biến dem đi 1 để báo cho chương trình biết đã có 1 cặp dấu mở ngoặc đúng; nếu dem<0 thi` s la` mo^.t bie^?u thu+'c sai bo+?i vi` so^' ky' tu+. ddo'ng ngoa(.c nhie^`u ho+n so^' ky' tu+. mo+? ngoa(.c.

- Nếu i là một dấu phép tính(+, -, *, /) và dau=1, có nghĩa là có 2 dấu liên tiếp nhau => đây là biểu thức sai, ngược lại, nếu i là một dấu phép tính và trước đó chưa có dấu nào (dau = 0) thì đặt dau:=1 để báo là đang xử lý biểu thức có dấu;

- Nếu i là các chữ cái A.. Z và a.. z thì đây chính là bắt đầu của 1 tên biến, ta phải xét i ở vị trí cuối cùng của tên biến này rồi xử lý tiếp:

- Nếu dau=0: S là biểu thức sai, bởi vì: ta khởi tao dau=1 nên với biến đầu tiên chương trình sẽ cho nó đúng và gán lại dau=0 rồi xử lý tiếp, đến khi gặp một dấu phép tính khác thì mới khởi tạo dau=1 nên trong trường hợp này S là biểu thức sai.

- Nếu dau=1: Gán lại dau=0 và xử lý tiếp các ký tự sau tên biến đó.

Nếu i là các chữ số, ta phải xét vi trị mới của i là vị trí cuối cùng của số nguyên đó (nhiều số):

- Nếu dau=0: tương tự trên, đây là biểu thức sai

- Nếu dau=1: gán lại dau=0 và xử lý các ký tự sau số vừa đọc

Dưới đây là thủ tục kiểm tra một xâu có là biểu thức số học hay không:

```
Function test(s: String): boolean;

Var i,dem,dau: integer;

Begin

test:=false;

dem:=0;

dau:=1;

i:=1;

While (i<=length(s)) do
```

```
begin
Case s[i] of
'(' : begin If dau=0 then exit; inc(dem); end;
')' : begin
If dau=1 then exit;
Dec(dem);
If dem<0 then exit;
end;
'+','-','*','/' : If dau=1 then exit Else dau:=1;
'A'..'Z','á..'z': begin
If dau=0 then exit;
dau:=0;
While (i
['á..'z','A'..'Z','0'..'9']) do inc(i);
end;
'0'..'9' : begin
If dau=0 then exit;
dau:=0;
While (i
['0'..'9']) do inc(i);
end;
```

```
End;

Inc(i);

end;

If (dem=0)and(dau=0) then test:=true;

End;
```

Để kiểm tra một xâu s có là biểu thức số học hay không ta chỉ cần gọi:

```
If test(s) then Writeln( La bieu thuc so hoc ) else Writeln( Bieu thuc sai );
```

Thông thường sau khi kiểm tra tính đúng đắn của biểu thức thì người ra đề sẽ yêu cầu tính biểu thức đó với các giá trị biến cho trước. Có nhiều cách để tính giá trị biểu thức số học, sau đây tôi sẽ giới thiệu cho các bạn phương pháp Ký pháp nghịch đảo Balan.

Ký pháp nghịch đảo Balan là một cách biểu diễn biểu thức toán học thông thường theo một dạng khác, khá thuận lợi cho việc tính toán một biểu thức trong máy tính

Một biểu thức toán học thông thường có thể được biến đổi thành dãy gồm các toán tử và toán hạng, trong đó một toán tử được dùng để tính toán với 1 hoặc 2 toán hạng đứng trước nó trong dãy đó.

Ví dụ : (T+R)*A biến đổi thành TR+A*

T+E*A biến đổi thành TEA*+

Có hai dạng ký pháp nghịch đảo Balan, preffix và suffix. Dạng trên là suffix.

Thuật toán để biến đổi biểu thức toán học thành dạng ký pháp nghịch đảo Balan như sau :

Ta sử dụng hai stack. Một dùng để lưu dạng Balan (nói tắt cho gọn), gọi là BtBalan, một dùng để lưu các toán tử dùng trong quá trình chuyển biểu thức thành dạng Balan, gọi là pheptinh. Kết quả được đưa vào BtBalan. Chỉ việc đọc lần lượt từ đầu đến cuối của BtBalan thì sẽ được dạng ký pháp nghịch đảo Balan của biểu thức đã cho.

Ta sẽ đọc lần lượt từ đầu đến cuối biểu thức đã cho để xử lý:

- Nếu gặp dấu mở ngoặc thì Push nó vào stack pheptinh.

- Nếu gặp một toán hạng thì Push nó vào stack BtBalan.

- Nếu gặp một toán tử thì:

+ Nếu các toán tử được lưu cuối cùng trong pheptinh có mức ưu tiên cao hơn thì lần lượt Pop các toán tử đó ra khỏi pheptinh, Push nó vào BtBalan. Làm vậy cho đến khi gặp phải toán tử có mức ưu tiên bằng hoặc thấp hơn nó thì dừng.

+ Push toán tử đang xét vào pheptinh.

+ Mức ưu tiên của các toán tử được quy định như sau: + -, *, / Dấu mở ngoặc được coi là có mức ưu tiên thấp nhất.

- Nếu gặp dấu đóng ngoặc thì Pop toàn bộ các toán tử được lưu trong pheptinh, Push vào BtBalan, cho đến khi gặp dấu mở ngoặc (được lưu trong pheptinh). Sau đó Pop luôn dấu mở ngoặc đó ra, vứt đi.

Sau khi đã thu được dạng kpnđ Balan rồi, tính toán như sau: tìm trong BtBalan một phép toán đứng liền sau một toán hạng. Dùng phép toán đó áp dụng trên 1 hoặc 2 toán hạng liền trước nó. Thay một toán hạng bằng kết quả tìm được, xoá phần tử chứa toán hạng còn lại(nếu là 2 toán hạng) và phần tử chứa toán tử ra khỏi BtBalan. Tính đến khi không tìm được toán tử nào nữa thì thôi.

Ta có bài toán tổng quát như sau: Cho xâu S là biểu diễn của một biểu thức số học (với các định nghĩa biểu thức số học như trên). Hãy kiểm tra xem biểu thức đã cho có hợp lý không? Nếu hợp lý thì tính giá trị biểu thức đã cho

Dữ liệu vào quy ước như sau:

- dòng đầu là biểu thức cần tính toán

- Các dòng tiếp theo ghi giá trị các biến ở trong biểu thức cần tính (nếu có) theo cấu trúc: =

Dữ liệu ra: tệp bieuthuc.out gồm 1 dòng ghi biểu thức và giá trị của biểu thức đó sau khi tinh được.

Ví dụ:

((me*1984)+(tra*1983))/us

me=24

tra=29

us=04

Dưới đây là chương trình đầy đủ và chính xác về kiểm tra biểu thức số học và tính giá trị biểu thức đó:

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q+,R+,S+,T-,V+,X+}

{$M 16384,0,655360}

Program Bieuthucsohoc;

Uses Crt;

Const

MN = 200;

fi = 'bieuthuc.inp';

fo = 'bieuthuc.out';

Type

mangst = array [1..MN] of string[9];

manglg = array [1..MN] of longint;

mangit = array [1..MN] of integer;

Var

tenbien,ten : mangst;

Value,tri,stack : manglg;

p : mangit;

s,Hangso,Bienso : string;
```

```
so,sb,st,ip,sst,Sobien: longint;
sym : char;
f,g : text;
Procedure Docbien;
Var
s1: string;
i: integer;
Begin
Sobien:=0;
While not Eof(f) do
begin
Readln(f,s1);
i:=1; Inc(Sobien);
Tenbien[Sobien]:='';
While s1[i]<>'=' do
begin
If s1[i]<>' ' then Tenbien[Sobien]:=Tenbien[Sobien]+UpCase(s1[i]);
Inc(i);
end;
Value[Sobien]:=0;
While i<=length(s1) do
```

```
begin
If s1[i] in ['0'..'9'] then
Value[Sobien]:=Value[Sobien]*10+ord(s1[i])-ord('0');
Inc(i);
end;
end;
For i:=1 to Sobien do
While length(tenbien[i])<8 do tenbien[i]:=tenbien[i]+' ';
End;
Procedure Loi;
Begin
Writeln('LOI: Khong hop lý);
End;
Procedure Init;
Var
i: integer;
Begin
Assign(f,fi); Reset(f);
Readln(f,s);
Docbien;
Close(f);
```

```
For i:=1 to length(s) do s[i]:=UpCase(s[i]);
sb:=1; st:=0; ip:=0; sst:=0;
End;
Procedure Cach;
Begin
While s[sb]=' ' do Inc(sb);
End;
Function Doiso: longint;
Begin
so:=0;
While s[sb] in ['0'..'9'] do
begin
so:=so*10+(ord(s[sb])-ord('0'));
Inc(sb);
end;
End;
Procedure Nhanbien;
Begin
Bienso:='';
While s[sb] in ['A'..'Z','0'..'9'] do
begin
```

```
Bienso:=Bienso+s[sb]; Inc(sb);
end;
While length(Bienso)<8 do Bienso:=Bienso+' ';
End;
Procedure Pheptinh;
Begin
Cach;
Case s[sb] of
'(' : begin sym:=s[sb]; Inc(sb); end;
')' : begin sym:=s[sb]; Inc(sb); end;
'+' : begin sym:=s[sb]; Inc(sb); end;
'-' : begin sym:=s[sb]; Inc(sb); end;
'*' : begin sym:=s[sb]; Inc(sb); end;
'/' : begin sym:=s[sb]; Inc(sb); end;
'%' : begin sym:=s[sb]; Inc(sb); end;
'0'..'9': begin sym:='0'; Doiso; end;
'A'..'Z': begin sym:='A'; Nhanbien; end;
Else sym:=#0;
End;
End;
Function Timso: integer;
```

```
Var
i: integer;
Begin
For i:=1 to st do
If ten[i]=' ' then
If tri[i]=so then
begin
Timso:=i;
exit;
end;
Inc(st); ten[st]:=' ';
tri[st]:=so;
Timso:=st;
End;
Function Timbien: integer;
Var
i: integer;
Begin
For i:=1 to st do
If pos(Bienso,ten[i])>0 then
begin
```

```
Timbien:=i;
exit;
end;
Inc(st); ten[st]:='?'+Bienso;
Timbien:=st;
End;
Function Timdau(c: char): integer;
Begin
Case c of
'+':Timdau :=-1;
'-': Timdau:=-2;
'*': Timdau:=-3;
'/': Timdau:=-4;
'%': Timdau:=-5;
'#': Timdau:=-6;
End;
End;
Procedure Bieuthuc;
Var
i: integer; daubt: char;
Procedure Hangtu;
```

```
Var
i: integer;
Procedure Nhantu;
Var
i: integer;
Begin
While sym in ['0','A','('] do
begin
Case sym of
'0': begin i:=Timso; Inc(ip); p[ip]:=i; end;
'A': begin i:=Timbien; Inc(ip); p[ip]:=i; end;
'(': begin
Pheptinh; Bieuthuc;
If sym<>')' then Loi;
end;
End;
Pheptinh;
end;
End;
Begin
Nhantu;
```

```
While sym in ['*','/','%'] do
begin
i:=Timdau(sym);
Pheptinh;
Nhantu; Inc(ip); p[ip]:=i;
end;
End;
Begin
If sym in ['+','-'] then begin daubt:=sym; Pheptinh end
Else daubt:='+';
Hangtu;
If daubt='-' then begin i:=Timdau('#'); inc(ip); p[ip]:=i; end;
While sym in ['+','-'] do
begin
daubt:=sym;
Pheptinh;
Hangtu;
i:=Timdau(daubt); Inc(ip); p[ip]:=i;
end;
End;
Procedure Tinh(i: integer);
```

```
Begin
Case p[i] of
-1: begin stack[sst-1]:=stack[sst-1]+stack[sst]; Dec(sst); end;
-2: begin stack[sst-1]:=stack[sst-1]-stack[sst]; Dec(sst); end;
-3: begin stack[sst-1]:=stack[sst-1]*stack[sst]; Dec(sst); end;
-4: begin stack[sst-1]:=stack[sst-1] div stack[sst]; Dec(sst); end;
-5: begin stack[sst-1]:=stack[sst-1] mod stack[sst]; Dec(sst); end;
-6: stack[sst]:=-stack[sst];
End;
End;
Procedure Napso(i: integer);
Begin
Inc(sst);
stack[sst]:=tri[p[i]];
End;
Procedure Hoiso(i: integer);
Var
j: integer;
Begin
Delete(ten[p[i]],1,1);
tri[p[i]]:=0;
```

```
For j:=1 to Sobien do
If tenbien[j]=ten[p[i]] then
begin
tri[p[i]]:=Value[j];
break;
end;
Insert('&',ten[p[i]],1);
End;
Procedure Tinhtoan;
Var
i: integer;
Begin
For i:=1 to ip do
begin
If p[i]<0 then Tinh(i)
Else
Case ten[p[i],1] of
' ': Napso(i);
'?': begin Hoiso(i); Napso(i); end;
'&': Napso(i);
End;
```

```
end;
End;
Procedure Main;
Begin
Pheptinh;
Bieuthuc;
Tinhtoan;
End;
Procedure Done;
Var
i: integer;
Begin
Assign(g,fo); Rewrite(g);
Writeln(g,s,' = ',stack[1]);
Close(g);
End;
BEGIN
Clrscr;
Init;
Main;
Done;
```

END.

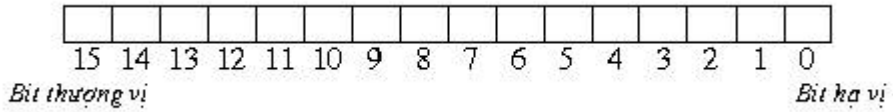
Chương trình trên chưa hoàn chỉnh, bởi vì nó chưa tính được các giá trị hàm lượng giác, hàm logarit, v..v.. Nếu muốn có chương trình 4Đ (đầy đủ, đúng đắn!) các bạn hãy liên hệ với tác giả theo địa chỉ email: chuong@bonbon.net hoặc theo đường bưu điện.

28 .**Các phép toán trên Bit**

Tác giả: Nguyễn Chí Thức

Trong máy tính, tất cả các kiểu dữ liệu đều được biểu diễn dưới dạng các bit. ở đây ta chỉ xét đến cách biểu diễn số nguyên trên máy tính. Trong Pascal, có các kiểu số nguyên: byte (8 bit), integer và word (16 bit) và longint (32 bit). Ta thử xét kiểu số nguyên 16 bit(integer và word).

Các bit được đánh số thứ tự từ 0 đến 15. Bit 15 còn được gọi là bit thượng vị, bit 0 là bit hạ vị.



Với 16 bit, ta biểu diễn được 216 = 65536 số khác nhau. Nếu dùng 16 bit để biểu diễn số tự nhiên (kiểu word), ta biểu diễn được từ 0 đến 65536. Còn đối với kiểu số nguyên (bao gồm cả nguyên âm và nguyên dương- kiểu integer), ta dùng bit thượng vị để diễn tả dấu của dữ liệu. Khi bit số 15 mang giá trị 0, quy định dữ liệ này là số dương và ngược lại. Như vậy 16 bit biểu diễn được các số từ -32768 đến 32767, cụ thể:

0	0000 0000 0000 0000
1	0000 0000 0000 0001
2	0000 0000 0000 0010
...	...
32767	0111 1111 1111 1111
-32768	1000 0000 0000 0000
-32767	1000 0000 0000 0001
...	...
-2	1111 1111 1111 1110
-1	1111 1111 1111 1111

Để biểu diễn số âm, người ta dùng phương pháp bổ số cấp 2.

Bổ số cấp 1 của số N(2) là số nhị phân nhận được bằng cách đảo các bit của số N: 0 thành 1, 1 thành 0. Ví dụ xét số 7 trong hệ cơ số 2: 0000 0000 0000 0111, bổ số cấp 1 của nó là: 1111 1111 1111 1000.

Bổ số cấp 2 của N(2) là số nhị phân nhận được khi lấy bổ số cấp 1 của N cộng thêm 1. Bổ số cấp 2 của 7 là: 1111 1111 1111 1001.

Bổ số cấp 2 của N dùng để chỉ số -N. Khi thực hiện phép cộng N với bổ số cấp 2 của N ta được kết quả là 0. Ví dụ:

$$\begin{array}{rl} 7: & 0000\ 0000\ 0000\ 0111 \\ \text{Bổ số cấp 2 của } 7: & +\ 1111\ 1111\ 1111\ 1000 \\ \hline & 0000\ 0000\ 0000\ 0000 \end{array}$$

Như vậy phép trừ được tiến hành bằng phép cộng với bổ số cấp 2 của số trừ.

Các phép tính trên đơn vị bit trong Pascal

- Các phép luận lí cơ bản trên đơn vị bit: And, Or, Not, Xor được định nghĩa qua bảng sau:

Bit a	Bit a	Not a	a Xor b	a Or b	a And b
0	1	1	1	1	0
0	0	1	0	0	0
1	1	0	0	1	1
1	0	0	1	1	0

Ví dụ: xét hai số 5 (0000 0000 0000 0101) và 7 (0000 0000 0000 0111):

Not 5 = 1111 1111 1111 1 010 = -6

Not 7 = 1111 1111 1111 1000 = -8

5 Or 7 = 0000 0000 0000 0010 = 7

5 Xor 7 = 0000 0000 0000 0010 = 2

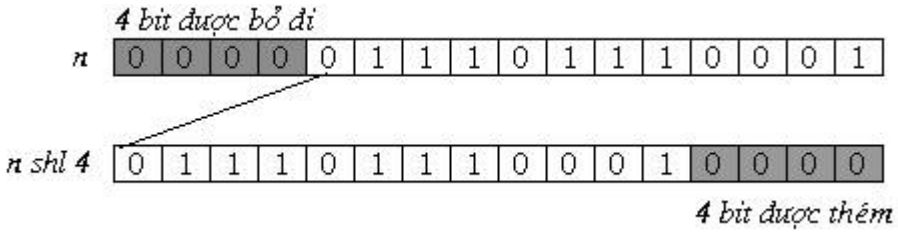
5 And 7 = 0000 0000 0000 0101 = 5

- Các phép xê dịch dữ liệu: (ta chỉ xét phép xê dịch đối với số nguyên dương)

Trong Pascal có hai phép xê dịch: dịch trái và dịch phải.

+ Phép dịch trái (shl - shift left):

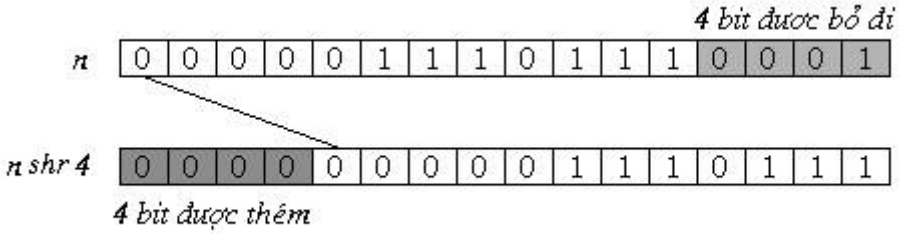
Cú pháp: n shl i - dịch các bit của n sang trái i vị trí, những bít trống được điền giá trị 0. Phép dịch trái được mô tả qua hình sau:



Như vậy n shl 4 = n*24 tổng quát ta có: n shl i = n*2i

+ Phép dịch phải (shr - shift right):

Cú pháp: n shr i - dịch các bit của n sang phải i vị trí, những bít trống được điền giá trị 0. Phép dịch phải được mô tả qua hình sau:



Ta có: n shr i = n div 2i

Sau đây là một số bài toán có liên quan đến các phép toán trên bit:

Bài 1: Cho một số tự nhiên n(kiểu word), in ra màn hình số n viết trong hệ nhị phân.

Giải: Với các phép toán trên bit, ta có rất nhiều cách để giải bài toán này, ví dụ:

Cách 1:

```
Procedure cach1;
Begin
for i:= 15 downto 0 do if n and ( 1 shl i)= 0 then write('0') else write('1');
End;
```

Cách2:

```
Procedure cach2;
Begin
for i:=0 to 15 do if ( n shl i) and 32768= 0 then write('0') else write('1');
End;
```

Cách3:

```
Procedure cach3;
Var k: word;
Begin
for i:=0 to 15 do
begin
k:= n; n:= n shl 1;
if n>= k then write(' 0') else write(' 1');
end;
End;
```

Bài 2: Cho số tự nhiên n (kiểu word), tìm số tự nhiên m lớn nhất sao cho 2m< n.

Giải:

```
Var n: word;

m: byte;

Begin

write(' Vao n: '); readln( n);

for m:=15 downto 0 do

if 1 shl m>n then continue

else

begin

writeln(' m=', m);

writeln(' 2^', m, ' <', n, '<', ' 2^', m+1);

exit;

end;

End.
```

Bài 3: Số chính

Cho file văn bản main. inp gồm n dòng chứa n số tự nhiên (kiểu word), mỗi số trên một dòng (n có thể rất lớn). Một số gọi là số chính nếu số đó xuất hiện ít nhất n div 2 + 1 lần trong file đó. Lập chương trình kiểm tra xem trong file main.inp có số chính không, nếu có, in ra số chính đó.

Giải:

Tạo hai mảng d0, d1: [0..15] of longint, ở đó d0[i] là số các số có bit thứ i là 0; dl[i] là số các số có bit thứ i là l.

Khi đó nếu tồn tại số chính m thì bit thứ i của m phải = max{d0[i], dl[i]} (do m xuất hiện > n div 2 lần). Dựa vào hai mảng d0, dl, ta xác định được m. Sau đó duyệt lại file văn bản và duyệt lại số lần xuất hiện của m.

Chương trình:

```
Const fi= ' main.inp';
Var
a, m, i: word;
n, c: longint;
d0, d1: array[ 0..15] of longint;
f: text;
mask: array[ 0..15] of word;
{ tính trước mảng mask để tăng tốc trong vòng lặp}
Begin
n:= 0; m:= 0; c:= 0;
for i:= 0 to 15 do mask[i]:= 1 shl i;
fillchar( d0, Sizeof( d0), 0);
fillchar( d1, Sizeof( d1), 0);
assign( f, fi>); reset( f);
while not eof(f) do
begin
inc( n); readln( f, a);
for i:=0 to 15 do if a and mask[i]= 0 then inc( d0[i]) else inc( d1[1]);
end;
close(f);
```

```
for i:= 0 to 15 do if d1>[i]> d0[i] then m:= n or mask[i];

assign( f, fi); reset( f);

while not eof( f) do

begin

readln( f, a);

if a= m then inc(c)

end;

close( f);

if c> n div 2 then writeln( 'co so chinh: ', m) else writeln( 'khong co!')

End.
```

Với việc sử dụng phép toán trên bit, chương trình trên cho phép chạy với n rất lớn và tốc độ nhanh (chưa đến hai giây với 100.000 số và chỉ hơn 10 giây với 1 triệu số khi test trên máy của tôi). Bạn có thể dùng hàm random để tạo file input gồm t số và viết thêm t+1 số giống nhau vào cuối file để có kết quả số chính.

Bài 4: Sắp xếp bằng cơ số

Có một thuật toán sắp xếp rất hiệu quả trên số nguyên bằng các phép toán trên bit như sau:

Đầu tiên, chia dãy thành hai phần: một phần có bit thượng vị (trong chương trình dưới đây là bit thứ 15- kiểu word) bằng 0, một phần có bit thượng vị bằng 1. Sau đó, chia tiếp mỗi phần thành hai phần: một phần có bit thứ 14 bằng 0, một phần có bit thứ 14 bằng 1... cứ như vậy đến bit hạ vị (bit 0) ta được một dãy số sắp xếp theo thứ tự tăng dần.

```
const

fi= 'sort. inp';

f0= 'sort. out';
```

```
Var
a: array[ 1..30000] of word;
mask: array[ 0..15] of word;
n, t: word; f: text;
Procedure init;
Begin
n:= 0;
for t:=0 to 15 do mask[t]:= 1 shl t;
assign( f, fi); reset( f);
while not eof( f) do
begin
inc( n);
readln( f, a[ n]);
end;
close( f);
End;
Procedure sort( l, h, b: word);
Var
i, j: word;
Begin
i:= l; j:= h;
```

```
repeat

while (i< j) and ( a[i] and mask[b]= 0) do inc(i);

while (i< j) and ( a[j] and mask[b]<> 0) do dec(j);

t:= a[i]; a[i]:= a[j]; a[j]:= t;

until i= j

if a[j] and mask[b]= 0 then inc(j);

if b> 0 then

begin

if l< j-1 then sort ( l, j-1, b-1)

if j< h then sort( j, h, b-1)

end;

End;

BEGIN

init;

sort( 1, n, 15);

assign( f, f0); rewrite( f);

for t:= 1 to n do writeln( f, a[t]);

close(f);

END.
```

Chương trình trên chạy với tốc độ tuyệt vời. Khi test với máy của tôi, với 30000 phần tử kiểu word chỉ chạy hết 0.77 giây.

Kết luận

Có rất nhiều bài toán nếu sử dụng linh hoạt, hợp lý các phép toán trên bit, ta sẽ có chương trình gọn và đặc biệt là tốc độ rất nhanh, vì vậy các bạn hãy tìm hiểu thêm nhiều nữa nhé!

29 .**Bài toán cái túi**

Tác giả: Võ Khắc Huấn

Bài toán : Cho một tập hữu hạn U = {ui; i = 1..n}$\Rightarrow$ mỗi phần tử ui $\in$U có kích cỡ S(ui) và số tự nhiên B.

Liệu có một tập con U' $\subseteq$ U sao cho $\sum$S(ui)= B. Trong đó: ui $\in$U'.

Để minh hoạ cho bài toán, chúng ta lấy ví dụ sau: giả sử tập u có 4 phần tử {u1, u2, u3, u4}, kích cỡ lần lượt là S(u1) = 1, S(u2) = 5, S(u3) = 2, S(u4) = 3 và B = 3. Bạn dễ dàng thấy rằng có hai phương án:

+ U1' = {u1, u3} vì S(u1)+ S(u3) = 3

+ U2' = {u4} vì S (u4) = 3

Đôi khi, tập U' được biểu diễn như là dãy có thứ tự các số nhị phân, phần tử là 1 - nếu nó thuộc U', hoặc 0 - nếu ngược lại là không thuộc. Như ví dụ trên ta có:

+ U1' = {u1, u3} $\Leftrightarrow$(1, 0, 1, 0)

+ U2' = {u4} $\Leftrightarrow$(0, 0, 0, 1)

Knapsack thuộc lớp bài toán NPC (không đa thức). Nghĩa là, nói chung không có thuật toán hữu hiệu nào để giải nó cho trường hợp bất kỳ. Điều này không có nghĩa là tất cả các trường hợp đều có cùng độ phức tạp. Chúng ta phát biểu lại bài toán dưới dạng có thể giải được bằng thuật toán có thời gian tuyến tính. Và gọi nó là bài toán Knapsack dễ.

Bài toán EKN: Cho n đồ vật, có khối lượng lần lượt là S1, S2,..., Sn sao cho:

$$S_{i+1} > \sum_{j=0}^{i} S_j$$

và một cái ba lô có sức chứa B.

Câu hỏi đặt ra là liệu có cách nào để bỏ vào ba lô một số vật để tổng khối lượng của chúng bằng B? Chúng ta minh hoạ bằng hai phương pháp sau:

Cho n = 6, B = 45

Và

$S_1 = 1$
$S_2 = 2$ $S_2 > S_1$
$S_3 = 4$ $S_3 > S_1 + S_2$
$S_4 = 8$ $S_4 > S_1 + S_2 + S_3$
$S_5 = 16$ $S_5 > S_1 + S_2 + S_3 + S_4$
$S_6 = 32$ $S_6 > S_1 + S_2 + S_3 + S_4 + S_5$

Bây giờ, chúng ta tìm một phương án V= (V1 , V2 ,..., V6). Trong đó Vi = 1 nếu vật thứ i được bỏ vào ba lô, Vi = 0 ngược lại vật bị để ngoài.

Chúng ta thấy:

V6 = 1, B - S6 = 45 - 32= 13

Kế tiếp V5 = 0 (vì S 5 = 16 > 13)

V4 = 1, B - S6 - S5 = 45 - 32 - 8= 5.

V3 = 1, B - S6 - S5 - S3 = 45 - 32 - 8 - 4 = 1

V2 = 0, (vì S2 = 2 > 1).

V1 = 1, B - S6 - S4 - S3 - S1 = 45- 32 - 8 - 4 - 1 = 0

Do đó, ta được: V= (1, 0, 1, 1, 0, 1).

Phân tích phương án trên chúng ta có thuật toán giải bài toán EKN như sau:

Input : S1, S2,..., Sn và B;

Output : V= (V1 , V2 ,..., Vn);

Actions :

for i:= n downto 1 do

```
begin

if B< Si then Vi = 0

else

begin

Vi = 1

B= B - Si

end;

end;
```

Sau đây là chương trình giải EKN với các khối lượng cho được bởi vectơ (1, 2, 4, 8, 16, 32, 64, 128).

```
Program EASYKNAPSACK;

Uses crt;

Const n = 8;

Type mang = array[1..n] of integer;

Var V, S: mang;

B, i: integer;

Begin

clrscr;

write(' Nhap B= '); readln(B);

S[1]:= 1;

for i:= 2 to n do S[i]:= S[i -1]* 2;

for i:= 1 to n-1 do
```

```
begin

if B< S[n-i] then V[n-i]:= 0

else

begin

V[n-i] :=1;

B := B- S[n-i];

end;

end;

if B= 0 then

For i:= 1 to n do writeln(' V', i, '=', V[i])

else writeln(' Khong co giai phap! ');

readln;

End.
```

Linh hoạt từ bài toán Knapsack chúng ta sẽ có nhiều bài toán dạng này. Mời các bạn thử bài toán sau:

Bài toán: Cho một cái cân hai đĩa và n quả cân với khối lượng tương ứng d1 , d2 ,..., dn. Ta nói trọng lượng y có thể cân được từ các quả cân di , i = 1..n, nếu Σxi di = y, với xi = {-1, 0, 1};

xi = -1, khi quả cân đặt cùng đĩa với vật cân;

xi = 0, khi quả cân không được sử dụng;

xi = 1, khi quả cân di không đặt cùng đĩa với vật cân.

Bạn hãy lập chương trình in ra mọi khối lượng có thể cân được từ những quả cân đó?

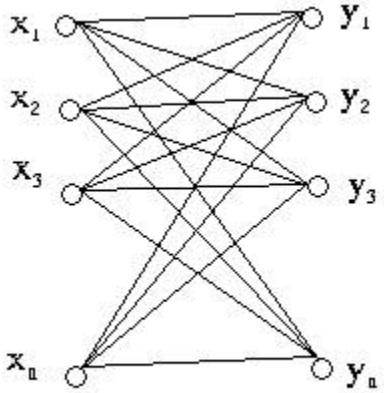
30 .**Bàn luận thêm về Cặp ghép**

Tác giả: Nguyễn Tuấn Dũng

'Ghép cặp' các đối tượng theo một quan hệ nào đó là một bài toán mang tính hết sức tự nhiên và có nhiều ý nghĩa trong ứng dụng thực tiễn. Chẳng hạn, các sinh viên ngành sư phạm do được nhà nước đào tạo miễn phí nên khi ra trường họ được phân công về dạy học ở các trường miền núi. Nhưng những sinh viên đó được đưa ra danh sách những trường mà mình muốn công tác với độ ưu tiên khác nhau. Một bài toán đặt ra là phải bố trí các sinh viên này sao cho phù hợp nhất (có thể được) với sở thích của mỗi người, tuy nhiên ở đây, sinh viên nào có kết quả học tập tốt hơn sẽ được ưu tiên hơn. Hay ta có thể gặp vấn đề ghép cặp trong các bài toán quen thuộc khác như: bài toán phân công công việc, bài toán hôn nhân bền vững, bài toán xếp thời khoá biểu...

Trong số 26 (11/2001), tác giả Lê Văn Chương đã giới thiệu với chúng ta thuật toán Kuhn-Munkres giải bài toán tìm cặp ghép có tổng trọng số lớn nhất. ở đây, không đi vào thuật toán tìm cặp ghép có tổng trọng số lớn nhất nữa vì điều đó khá rõ ràng trong số báo trước. Tuy nhiên, chúng ta sẽ xem xét thêm một chút để giúp các bạn phần nào tiếp cận gần bài toán hơn và đỡ nhầm lẫn trong lúc cài đặt chương trình. Để tiện theo dõi, ta tóm tắt lại bài toán:

Cho G = (X U Y, E) là đồ thị hai phía đầy đủ, trong đó: X, Y là hai tập hữu hạn gồm n phần tử, E là tập các cạnh của đồ thị và với mỗi cạnh được gán với một trọng số Cij. Cần tìm tập cặp ghép đầy đủ M có tổng trọng số lớn nhất.



Đối với bài toán này có thể sử dụng bài toán luồng cực đại để giải bằng cách thêm vào G hai đỉnh giả S và T, nối với các đỉnh xi thuộc X và nối T với các đỉnh yj thuộc Y bằng các cạnh có trọng số là 0. Tuy nhiên, bài toán cặp ghép cực đại là trường hợp

riêng của bài toán luồng nên nó có những đặc điểm riêng và do đó dẫn đến việc giải quyết nó thì cũng có những thuật toán đặc thù, mà tiêu biểu là thuật toán gán nhãn, trong đó ta có thể đi theo hai hướng chính:

Hướng thứ nhất, xuất phát từ một cặp ghép M đầy đủ bất kỳ của G, ta xây dựng một nhãn F tương ứng với M, nếu F chấp nhận được thì M là nghiệm cần tìm, ngược lại nhãn F là không chấp nhận được thì ta tiếp tục điều chỉnh.

Hướng thứ hai, xây dựng một nhãn F chấp nhận được, sau đó tìm M tương ứng với F bằng cách: khởi tạo M là tập rỗng, chừng nào mà M chưa đầy đủ thì ta còn tiếp tục điều chỉnh (tăng cặp ghép).

Thuật toán Kunh-Munkres trình bày trên số báo 26 là cách giải thứ hai, được trình bày khá rõ ràng. Tuy nhiên có điểm cần chú ý trong bước 5 (sửa nhãn), lượng sửa nhãn không phải:

d:= min {F(xi) + F (yj) - Cij, xi thuộc S, yj thuộc T} (có thể do lỗi khi in ấn) mà là:

d:= min {F(xi) + F(yj) - Cij, xi thuộc S, yj thuộc T } (*)

Bởi vì, trong bước 3 khi không tìm được đường tăng cặp ghép ta mới phải sửa nhãn sao cho:

- Nhãn Fmới vẫn chấp nhận được.

- Fmới vẫn tương ứng với M đang có, để cho số cạnh của G (F, M) không bị giảm đi.

- Tăng thêm số cạnh trong đồ thị cân bằng tương ứng G (F, M)

Muốn tăng số cạnh trong G(F, M) thì ta phải có thêm cặp xi', yj' khác, cụ thể (xi', yj') thuộc G(F, M), sao cho: F(xi') + F(yj') = C'ij. Đồng thời để cho sau khi thêm cạnh vào G(F, M) ta có thể tìm đường tăng cặp ghép thì cạnh cần thêm đó được chọn là cạnh nối giữa một đỉnh xi đã được đi đến trong quá trình tìm đường ở bước 3 với một đỉnh yj' chưa được đi đến trong quá trình đó.

Nếu lượng sửa nhãn là d:=min{F(xi) + F (yj) - Cij, xi thuộc S, yj thuộc T} thì sẽ dẫn đến những sai sót khiến chương trình không cho ra kết quả. Thật vậy, ta cho ra công thức sửa nhãn:

F(xi) := F(xi) - d với xi thuộc S

F(yj) := F(yj) + d với yj thuộc T

Xét một ví dụ đơn giản, sau khi sửa nhãn với lượng sửa nhãn d như trên, đối với những đỉnh xithuộc S và yj thuộc T, ta có:

F(xi) mới := F(xi) cũ - d

F(yj)mới := F(yj)cũ

Suy ra:

A = F(xi)mới + F(yj)mới - Cij = F(xi)cũ + F(yj)cũ - Cij - d

Nhận thấy A có thể dương, bằng không, hay âm tuỳ thuộc vào các xi, yj gán nhãn Fcũ vì d chỉ là min {F(xi)cũ + F(yj)cũ - Cij } đối với những đỉnh xi thuộc S, yj thuộc T. Do đó có thể xảy ra F(xi)mới + F(yj)mới < Cij với một vài cặp cạnh (xi,yj) nào đó mà xi thuộc S, yj thuộc T. Nói cách khác, nhãn Fmới sẽ là không chấp nhận được, ít nhất là đối với những đỉnh yj thuộc T.

Trong khi đó, theo như dưới đây ta sẽ thấy việc gán nhãn công thức (*) là hoàn toàn đúng đắn. Thật vậy:

Ta có công thức sửa nhãn:

F(xi) := F(xi) - d với xi thuộc S

F(yj) := F(yj) + d với yj thuộc T

Nhận thấy, do tính chất đặc biệt của G(F, M), chỉ có thể đi từ yj sang xi khi và chỉ khi (xi, yj) thuộc M nên tập S (tập các đỉnh đến được trong bước 3 tìm đường tăng cặp ghép) sẽ là tập các đỉnh của X đã được ghép cặp (trừ x0). Đồng thời, khi không có đường tăng cặp ghép nghĩa là trong khi tìm đường đi, ta chỉ đến được các yj đã bị ghép cặp. Do đó tập T cũng là tập các đỉnh của Y đã ghép cặp (với các đỉnh thuộc S)

Bây giờ ta sẽ xem xét các trường hợp:

1. Đối với những xi thuộc S: F(xi)mới := F(xi)cũ - D

- Xét các đỉnh yj thuộc T: F(yj)mới := F(yj)cũ

Suy ra: F(xi)mới + F(yj)mới - Cij = F(xi)cũ + F(yj)cũ - Cij - D >= 0

(vì D = min {F(xi)cũ + F (yj)cũ - Cij } với xi thuộc S, yj thuộc T)

do đó: F(xi)mới + F(yj)mới >= Cij , xi thuộc S, yj thuộc T (1)

Vậy nhãn F vẫn chấp nhận được đối với những xi thuộc S và yj thuộc T.

Ngoài ra ta còn có thêm đẳng thức F(xi) + F(yj) = Cij trong đó xithuộc S, yj thuộc T tương ứng với việc xảy ra dấu '=' trong bất đẳng thức lượng sửa nhãn (luôn luôn xảy ra). Do đó dẫn tới việc làm tăng thêm số cạnh của G(F, M).

- Xét yj thuộc T: F(yj)mới := F(yj)cũ + D

Do đó: F(xi)mới + F(yj)mới = F(xi)cũ - D + F(yj)cũ + D = F(xi)cũ + F(yj)cũ = Cij (2)

bởi vì với xi thuộc S và yj thuộc T trong bước 3 thì (xi, yj) là một cạnh của đồ thị cân bằng tương ứng G(F, Mcũ)

Như vậy, nhãn Fmới vẫn tương ứng tập cặp ghép Mcũ ngay cả với những đỉnh bị sửa nhãn. Đối với những đỉnh còn lại của Mcũ không bị sửa nhãn thì cũng vẫn tương ứng với Fmới vì Fmới là giữ nguyên giá trị của Fcũ đối với những đỉnh này.

2. Đối với xi thuộc S: F(xi)mới = F(xi)cũ

mà: F(yj)mới = F(yj)cũ với yj thuộc T

F(yj)mới = F(yj)cũ + D với yj thuộc T

thì: F(xi)mới + F(yj)mới >= F(xi)cũ + F(yj)cũ >= Cij (3)

(vì nhãn Fcũ là chấp nhận được)

Tóm lại, từ (1) (2) (3) ta kết luận sau khi sửa nhãn, nhãn Fmới vẫn chấp nhận được và tương ứng với Mcũ đang có, đồng thời thêm được cạnh trong G(F, M). Do đó, quay lại bước 3 ta vẫn có thể tìm được đường tăng cặp ghép và tiếp tục các bước tiếp theo của thuật toán.

Trên đây đã bàn luận một số chi tiết trong các bước thực hiện thuật toán Kuhn-Munkres. Tiếp theo, vấn đề đặt ra là đối với bài toán tìm cặp ghép đầy đủ có tổng trọng số nhỏ nhất thì sao?

Cặp ghép đầy đủ có tổng trọng số nhỏ nhất

Nếu bạn xem xét kỹ lưỡng một chút về thuật toán Kuhn-Munkres thì có thể thấy ngay bài toán này hoàn toàn tương tự với bài toán tìm cặp ghép có tổng trọng số lớn nhất. Chỉ cần sửa lại một chút như sau:

+ Thứ nhất, nhãn F được gọi là chấp nhận được nếu F(xi) + F(yj) <= Cij

Như vậy, có thể khởi tạo: F(xi) := 0

F(yj) := min {Cij , xi thuộc X }, yj thuộc Y

+ Thứ hai, lượng sửa nhãn: d:= max {F(xi) + F(yj) - Cij , xi thuộc S, yj thuộc T }

với nhận xét: d <= 0

Hoặc: d:=min {Cij - F(xi) - F(yj), xi thuộc S, yj thuộc T }

thì d >= 0 và khi đó, công thức sửa nhãn sẽ là: F(xi) := F(xi) + d

F(yj) := F(yj) - d

Cách khác, đơn giản hơn, chúng ta có thể thấy để tìm M có tổng trọng số nhỏ nhất, chỉ cần đổi dấu các phần tử của ma trận trọng số rồi tìm cặp ghép có tổng trọng số lớn nhất với ma trận trọng số mới này. Cặp ghép đó sẽ chính là cặp ghép có tổng trọng số nhỏ nhất cần tìm đối với ma trận trọng số ban đầu.

Như chúng ta đã biết bài toán cặp ghép được xem xét theo hai khía cạnh. Trường hợp thứ nhất, người ta quan tâm đến việc ghép cặp đầy đủ và thoả mãn tính tối ưu như bài toán Kuhn-Munkres ở trên. Nhưng trong trường hợp khác, người ta lại không quan tâm đến việc ghép cặp đầy đủ với tổng trọng số lớn nhất mà cần tìm một tập cặp ghép có số lượng cặp ghép là cực đại (không quan tâm đến trọng số trên các cạnh của cặp ghép). Khi đó chúng ta sẽ có một bài toán khác dưới đây:

Cặp ghép cực đại

Để minh hoạ, ta xét một bài toán cụ thể, bài toán Phân công thợ-việc: Có N người thợ và M công việc, mỗi công việc, mỗi một người thợ chỉ biết làm một số công việc nhất định. Cần phân công mỗi thợ chỉ làm một việc và mỗi việc chỉ được làm bởi một thợ sao cho có nhiều công việc được làm nhất. Khả năng làm việc của các thợ được cho trong ma trận Cij với Cij = 1 thì thợ i biết làm việc j, Cij =0 thì thợ i không biết làm việc j.

Ta nhận thấy hoàn toàn có thể giải bài toán này bằng thuật toán Kunh-Munkres. Tuy nhiên, vấn đề ở đây là bài toán của chúng ta đơn giản hơn nhiều và tất nhiên, có thể giải quyết nó bằng một cách riêng khá dễ dàng.

Nhắc lại bài toán phân công Thợ - Việc: Có N người thợ và M công việc, mỗi công việc, mỗi một người thợ chỉ biết làm một số công việc nhất định. Cần phân công mỗi thợ chỉ làm một việc và mỗi việc chỉ được làm bởi một thợ sao cho có nhiều công việc được làm nhất. Khả năng làm việc của các thợ được cho trong ma trận Cij với Cij = 1 thì thợ i biết làm việc j, Cij =0 thì thợ i không biết làm việc j.

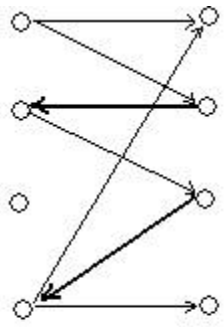
Ta nhận thấy hoàn toàn có thể giải bài toán này bằng thuật toán Kunh-Munkres. Tuy nhiên, vấn đề ở đây là bài toán của chúng ta đơn giản hơn nhiều và tất nhiên, có thể giải quyết nó bằng một cách riêng khá dễ dàng.

Với các khái niệm và định nghĩa như trong bài toán Kunh-Munkres (tuy nhiên, đồ thị hai phía G ở đây không nhất thiết đầy đủ), ta sử dụng định lý sau để có được thuật toán:

Tập cặp ghép M (có thể không đầy đủ) là cực đại khi và chỉ khi không tìm được đường đi bắt đầu từ một đỉnh tự do thuộc X và kết thúc tại một đỉnh tự do thuộc Y trên đồ thị G' = (X U Y, E'). Đường đi đó gọi là đường tăng cặp ghép, và G' nhận được từ G bằng cách định hướng lại các cạnh của G theo quy tắc:

- Những cạnh (x, y) thuộc M trong đồ thị G được định hướng ngược lại trở thành cung (y, x) trong đồ thị G'.

- Những cạnh (x, y) không thuộc M trong đồ thị G được định hướng trở thành cung (x, y) trong đồ thị G'.



Đồ thị G'(M)
M là tập các cung ngược

Như vậy, ta có nhận xét sau: giả sử đã xây dựng được một tập cặp ghép M nhưng vẫn có đường tăng cặp ghép từ đỉnh tự do x0 thuộc X đến đỉnh tự do y0 thuộc Y:

x0 --> y1 --> x1 --> y2 -->... --> xt --> y0

trong đó các cạnh đậm là các cạnh thuộc tập cặp ghép đã có. Dễ thấy vì đường tăng cặp ghép xuất phát từ đỉnh tự do x0 thuộc X đến đỉnh tự do y0 thuộc Y nên số cạnh nhạt trên đường đi sẽ lớn hơn số cạnh đậm. Khi đó nếu đổi chỗ các cạnh nhạt thành cạnh đậm, các cạnh đậm thành cạnh nhạt thì ta được tập cặp ghép mới M' có lực lượng lớn hơn M.

Tóm lại, sơ đồ thuật toán như sau:

Bước 1 : Khởi tạo tập cặp ghép M là rỗng.

Bước 2 : Tìm đường cặp ghép từ một đỉnh x0 tự do thuộc X đến một đỉnh y0 tự do thuộc Y:

+ Nếu có thì chuyển sang bước 3.

+ Nếu không có thì tập cặp ghép hiện thời là cực đại và kết thúc thuật toán.

Bước 3 : tăng cặp ghép:

Thực hiện việc đổi cạnh nhạt thành đậm và cạnh đậm thành nhạt. Tuy nhiên trong khi cài đặt không phải xây dựng cụ thể đồ thị G' và đổi màu cạnh.

Quay về bước 2.

Chương trình bài Phân công Thợ việc :

```
Program Capghepcucdai;

Uses Crt;

Const

Inf = 'Matching.inp';

Outf = 'Matching.out';

MaxN = 200;
```

```
MaxM = 200;
Var
A : array[0..MaxN,0..MaxM]of 0..1;
Tho: array[1..MaxN]of byte;
Viec : array[1..MaxM]of byte;
Pred : array[1..MaxN]of byte;
N,M : byte;
X0, Y0 : byte;
PathFound:boolean;
Procedure ReadInp;
var f:text; i,x:byte;
begin
fillchar(A, sizeof (A),0);
assign(f,inf); reset(f);
readln(f,N,M);
for i:=1 to n do
begin
while not (seekeoln(f)) do
begin
read(f,x);
A[i,x]:=1;
```

```
end;
readln(f);
end;
close(f);
end;
Procedure WriteOut;
var f:text; i, socv:byte;
begin
socv:=0;
for i:=1 to n do
if viec[i]>0 then inc(socv);
assign(f,outf); rewrite(f);
writeln(f,socv);
for i:=1 to n do
if viec[i]>0 then writeln(f,{'Thó,}i:3,{' - viec',}viec[i]:3);
close(f);
end;
Procedure FindPath(x:byte);
var y:byte;
begin
for y:=1 to m do
```

```
begin
if (Pred[y]=0)and (A[x,y]=1) then
begin
Pred[y]:=x;
if Tho[y]=0 then
begin
Y0:=y;
PathFound:=true;
Exit;
end
else
begin
FindPath(Tho[y]);
end;
end;
if PathFound then Exit;
end;
end;
Procedure IncM;
var x,y,y1:byte;
begin
```

```
y:=y0;
while pred[y]<>x0 do
begin
x:=pred[y];
tho[y]:=x;
y1:=viec[x];
viec[x]:=y;
y:=y1;
end;
viec[X0]:=y; tho[y]:=X0;
end;
Procedure Matching;
var Stop: boolean; x:byte;
begin
fillchar(Tho, sizeof(Tho),0);
fillchar(Viec, sizeof(Viec),0);
Stop:=false;
While not(Stop) do
begin
PathFound:=false;
fillchar(pred, sizeof(pred),0);
```

```
for x:=1 to n do

if viec[x]=0 then

begin

X0:=x;

FindPath(x);

if PathFound then Break;

end;

if PathFound then IncM {TangCG} else Stop:=true;

end;

end;

BEGIN

clrscr;

ReadInp;

Matching;

WriteOut;

END.
```

Có rất nhiều bài toán được giải quyết bằng việc ghép cặp sao cho số cặp ghép là cực đại. Chẳng hạn như bài Xếp các con xe trên bàn cờ quốc tế: Trên bàn cờ quốc tế kích thước N x N (N < 251), ngu+o+`i ta dda'nh da^'u mo^.t o^ ba(`ng ca'ch ghi tre^n ddo' so^' 1, ca'c o co`n la.i ghi so^' 0. Ha~y ti`m ca'ch dda(.t nhie^`u nha^'t ca'c con xe va`o ca'c o^ ddu+o+.c dda'nh da^'u sao cho kho^ng co' hai con xe na`o a(n ddu+o+.c nhau (tu+'c la` kho^ng co' hai con na`o xe^'p tre^n cu`ng mo^.t ha`ng hay cu`ng mo^.t co^.t).

Tuy nhiên, có một dạng bài toán khác khá hay và thực tế mà cũng được giải bằng thuật toán ghép cặp cực đại. Đó là bài toán ta sẽ trình bày dưới đây:

Hệ đại diện phân biệt

Định nghĩa : Cho một họ tập con S1, S2,..., Sm thuộc X.

Một bộ sắp thứ tự gồm m phần tử (a1, a2,..., am) với a1 thuộc S1, a2 thuộc S2,..., am thuộc Sm, trong đó: ai khác aj với i khác j (i, j = 1, 2,..., m) được gọi là một hệ đại diện phân biệt của họ S1, S2,..., Sm.

Bài toán đặt ra là cần phải xây dựng một hệ đại diện phân biệt của một họ cho trước.

Ví dụ, nhà trường tổ chức một buổi họp cán bộ của trường, đề nghị mỗi tổ chức cử một người đi họp và mỗi người chỉ được đại diện cho một tổ chức. Tuy nhiên, một người có thể giữ nhiều chức vụ trong các tổ chức khác nhau. Chẳng hạn, một bí thư liên chi có thể vừa là một lớp trưởng, một phó bí thư của lớp cũng có thể là hội trưởng hội sinh viên của khoa... Cần lập chương trình sắp xếp mọi người đi họp sao cho thích hợp.

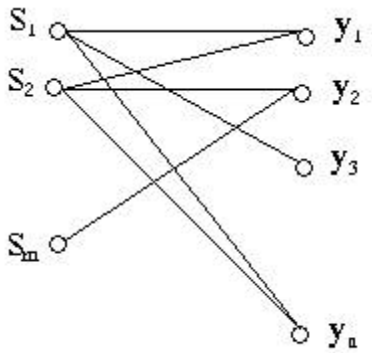
Quay lại bài toán, trước hết, gọi tập Y = S1 U S2 U S3 U... U Sm là hợp của tất cả các Si (chính là tập các phần tử khác nhau của họ S). Chúng ta có nhận xét sau về điều kiện để tồn tại một hệ đại diện phân biệt là:

+ Nếu lực lượng (số phần tử) của Y là n thì n tối thiểu phải bằng m. Nghĩa là n >= m.

Để giải bài toán hệ đại diện phân biệt, ta xây dựng đồ thị hai phía biểu diễn mối quan hệ giữa các tập Si thuộc X và các phần tử yj thuộc Y của họ (j = 1, 2,..., n) bằng ma trận Cij được định nghĩa như sau:

Cij = 1 tương đương với phần tử yj thuộc tập Si

Cij = 0 tương đương với phần tử yj không thuộc Si



Từ đây ta thấy bài toán đã được đưa về việc tìm một tập cặp ghép cực đại M (có lực lượng lớn nhất) của đồ thị G. Nếu lực lượng của M tìm được bằng đúng m thì ta đã tìm được một hệ đại diện phân biệt. Còn nếu không thì kết luận là không tồn tại một hệ đại diện phân biệt nào của họ S.

Trên đây chỉ là vài vấn đề bổ sung thêm cho bài toán cặp ghép mà tôi xin được giới thiệu để các bạn tham khảo và xem xét. Còn vô số bài toán khác liên quan đến việc 'ghép cặp' trong lý thuyết đồ thị cũng như trong thực tế phức tạp. Mong được bạn đọc góp ý thêm.

31 .**Ma Phương**

Tác giả: Nguyễn Trường Đức Trí

Dân tộc Shang chia thế giới làm chín phần. Chính giữa là vương quốc của họ. Những phần khác là núi, sa mạc và những vùng đất con người chưa biết đến. Nếu vị hoàng đế giữ cho phần đất của mình luôn hoà bình và thịnh vượng, thượng đế sẽ làm cho những phần đất còn lại phải cùng làm việc để xây dựng một thế giới hoàn thiện.

Sau này, người Trung Quốc sử dụng sáng kiến này để xây dựng nên ma phương số. Nếu mỗi ô vuông đứng đúng vị trí của nó, thì các hàng ngang, dọc, chéo đều có tổng giống nhau. Đây là dấu hiệu của một thế giới hoàn thiện. Chúng ta không biết mỗi số phải đứng ở đâu. Nhưng nếu một phần của thế giới đứng sai vị trí thì cả thế giới sẽ bị lộn xộn.

Nhiệm vụ của chúng ta là sắp xếp các số theo trật tự của nó để tạo thành một thế giới hoàn thiện.

Các khái niệm về ma phương

Trước khi bắt đầu công việc của mình, chúng ta phải nắm một số khái niệm:

- Ma phương số cấp n là một hình vuông gồm n2 số nguyên khác nhau được sắp xếp sao cho tổng các số theo hàng ngang, cột dọc, hoặc theo đường chéo là bằng nhau, và bằng một hằng số gọi là hằng số ma của hình vuông.

Ma phương gọi là chuẩn khi n2 số là n2 số nguyên dương đầu tiên.

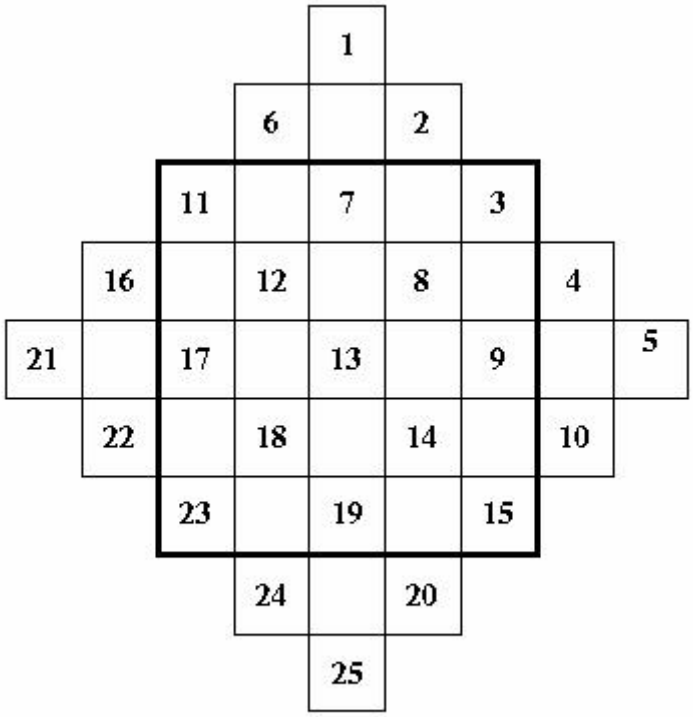
Người ta chứng minh được rằng hằng số ma của ma phương chuẩn cấp n là

$$\frac{n(n^2+1)}{2}$$

Trong giới hạn của bài viết này, tôi chỉ xin trình bày những hiểu biết của mình về cách xây dựng một ma phương chuẩn cấp n.

Thuật toán và cách giải ma phương bậc n

Chắc các bạn ai cũng biết các giải ma phương bậc ba, năm, bảy các số lẻ theo toán học. Chúng ta sẽ vẽ thêm các tháp vào 4 cạnh của hình vuông như sau:



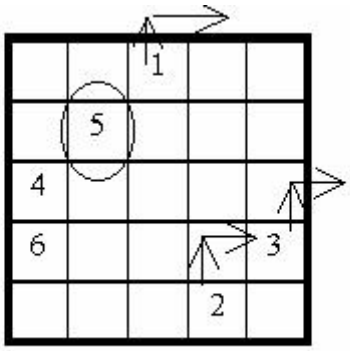
Ví dụ về ma phương bậc 5

Sau đó điền các số trên các tháp vào hình vuông. Cách giải này chắc ai cũng biết nên tôi xin không nhắc lại.

Thế nhưng để mô phỏng cách giải này trên máy vi tính thì ta phải mở rộng ma trận của mình và như thế rất tốn bộ nhớ. Ta sẽ phải làm gì với những ô không chứa số Quả là một lượng bộ nhớ quá lớn sẽ không được sử dụng một cách đáng giá phải không nào Tôi xin trình bày cách giải ma phương lẻ này mà không cần mở rộng mảng:

Bắt đầu số 1 sẽ đứng ở hàng đầu tiên, cột chính giữa. Bạn sẽ phải đi như đi một quân cờ. Thuật toán của chúng ta là đi lên, qua phải, nếu đụng nóc thì xuống dưới, dụng tường phải thì qua tường trái. Còn nếu bạn đi phải vào một ô đã có số rồi thì xin mời

bạn lùi lại chỗ cũ và xuống một ô. Xong một quá trình như vậy bạn sẽ có một ô dừng chân, hãy điền số tiếp theo vào ô đó. Thuật toán trên có thể giúp giải bài toán ma phương bậc lẻ trở nên rất dễ dàng hơn rất nhiều mà không cần mở rộng mảng.



Hình vẽ mô phỏng cho thuật toán trên

(Trường hợp số 5, lên qua phải, nhưng do ô đó đã có số 1 nên phải trở về vị trí của số 4 và xuống 1 ô)

Cứ như vậy cho đến khi bạn đã đi được hết mảng. Nếu làm ma phương bậc 3 bạn sẽ thu được kết quả sau

8	1	6
3	5	7
4	9	2

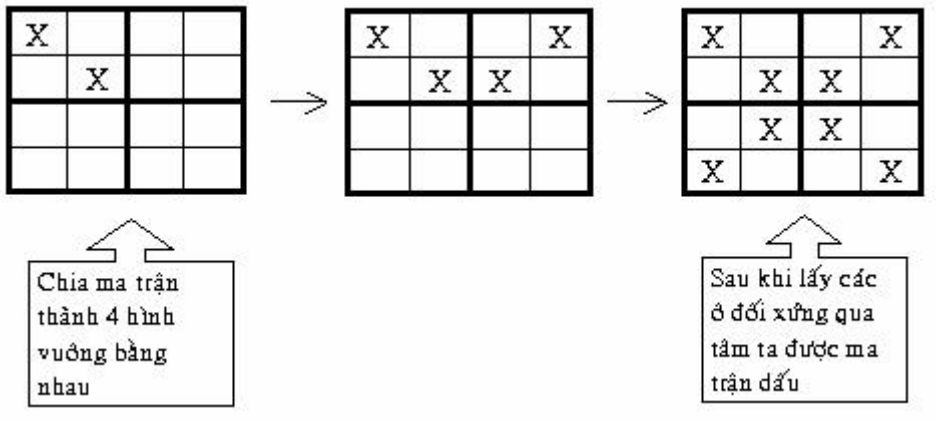
Tôi nghĩ việc làm một chương trình để giải ma phương bậc lẻ là một việc không khó đối với chúng ta phải không nào

```
Procedure MaPhuongLe;

var dem:integer;

luui,luuj:integer;

begin

fillchar(a,sizeof(a),0);
```

```
dem:=1; i:=1; j:=n div 2+1;

repeat

a[i,j]:=dem;

inc(dem);

luui:=i; luuj:=j; i:=i-1; j:=j+1;

if i<1 then i:=n;

if j>n then j:=1;

if a[i,j]<>0 then

begin

i:=luui+1; j:=luuj;

end;

until dem>n*n;

end;
```

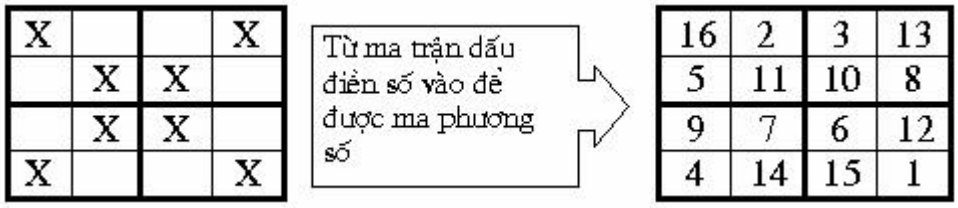
Rõ ràng, chúng ta không thể áp dụng thuật toán trên cho ma phương bậc chẵn.

Với bậc chia hết cho 4 thì chúng ta cần đánh dấu một số ô tạm gọi là ô đối xứng tâm (kí hiệu trong bài viết là chữ x) vì những số khi gặp ô này sẽ đi qua ô đối xứng tâm với nó. Chia ma trận của ta ra làm bốn phần. Số lần xuất hiện kí hiệu x trên cùng một dòng, một cột trong một phần đều không vượt quá n/4. Chẳng hạn với n=4 thì mỗi hàng, của mảng nhỏ 2x2 chỉ được có 1 dấu x. Sau đó lấy đối xứng các kí hiệu qua tâm. (Xem sơ đồ)



Hình 1

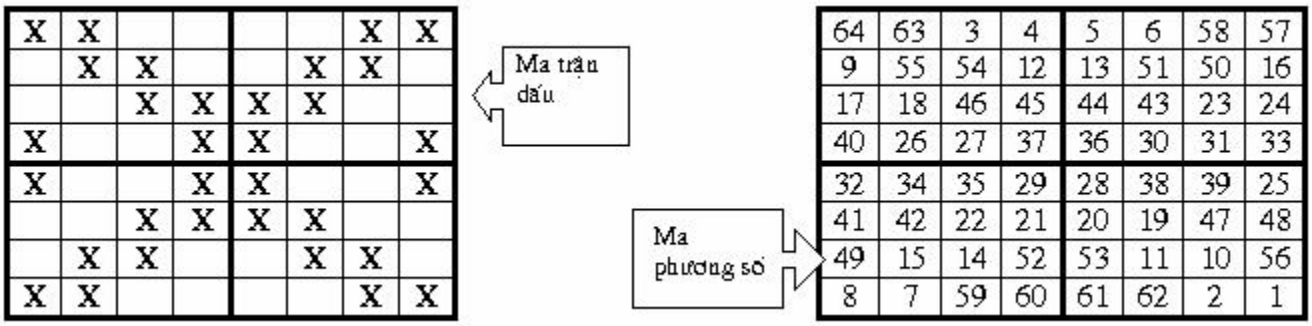
Bắt đầu từ số 1 ở hàng và cột đầu tiên, điền các số từ trái sang phải, từ trên xuống dưới, các ô nào có đánh dấu x hoặc có số điền rồi thì chuyển số đó qua ô đối xứng tâm với nó.



16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1

Hình 2

Ví dụ về ma phương bậc 8

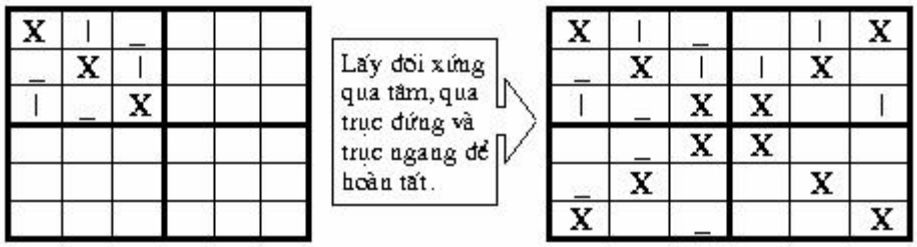


64	63	3	4	5	6	58	57
9	55	54	12	13	51	50	16
17	18	46	45	44	43	23	24
40	26	27	37	36	30	31	33
32	34	35	29	28	38	39	25
41	42	22	21	20	19	47	48
49	15	14	52	53	11	10	56
8	7	59	60	61	62	2	1

Hình 3

Còn ở ma phương bậc chẵn không chia hết cho 4, ngoài việc đánh dấu các ô đối xứng qua tâm, chúng ta còn phải đánh dấu các ô đối xứng qua 2 trục (trục đứng và

trục ngang). Qui tắc đánh dấu các ô đối xứng tâm vẫn giống như ở trên nhưng do n không chia hết cho 4 nên chỉ lấy phần nguyên trong phép chia n cho 4. Còn các ô đối xứng qua các trục (kí hiệu | : đối xứng qua trục đứng; _: đối xứng qua trục ngang). Qui tắc đánh dấu các ô đối xứng trục: Mỗi hàng, mỗi cột trong mỗi hình vuông nhỏ đều đúng 1 kí hiệu mỗi loại. Ví dụ với n=6, mỗi hàng, mỗi cột chỉ được có 1 dấu X, 1 dấu | và 1 dấu _. Sau đó lấy đối xứng qua trục đứng, trục ngang các dấu | và _.



Sau đó điền giống như cách đã nói ở trên, nhưng nhớ rằng ở đây còn có các số đối xứng theo trục ngang và trục đứng đấy nhé. Với ma trận dấu ở trên chúng ta sẽ hoàn tất được ma phương 6 một cách dễ dàng:

36	5	33	4	2	31
25	29	10	9	26	12
18	20	22	21	17	13
19	14	16	15	23	24
7	11	27	28	8	30
6	32	3	34	35	1

Chương trình con để tạo ma trận dấu:

```
procedure DanhDau;

var n2,n3,k,x:integer;

begin

n2:=n div 2;

n3:=n2 div 2;
```

```
j:=0;
fillchar(dau,sizeof(dau),' ');
for i:=1 to n2 do
begin
for k:=1 to n3 do
begin
x:=(j+k-1) mod n2 +1;
dau[i,x]:='x';
dau[i,n-x+1]:='x';
dau[n-i+1,x]:='x';
dau[n-i+1,n-x+1]:='x';
end;
inc(j);
end;
if n mod 4 <>0 then
begin
j:=n3; k:=n3+1;
for i:=1 to n2 do
begin
j:=j mod n2 +1;
dau[i,j]:='|';
```

```
dau[i,n-j+1]:='|';
k:=k mod n2 +1;
dau[i,k]:='-';
dau[n-i+1,k]:='-';
end;
end;
end;
```

Chương trình con để điền các số vào ma trận số để được một ma phương.

```
procedure DienSo;
var dem:integer;
begin
dem:=1;
for i:=1 to n do
for j:=1 to n do
begin
case dau[i,j] of
'x': a[n-i+1,n-j+1]:=dem;
'|': a[i,n-j+1]:=dem;
'-': a[n-i+1,j]:=dem;
' ': a[i,j]:=dem;
end;
```

```
inc(dem);

end;

end;
```

Lưu ý: Thuật toán trên có thể áp dụng cho một dãy số tăng dần đều bất kỳ. VD: dãy các số chẵn, các số lẻ Bạn thử xem!

Phần còn lại viết một chương trình chính để xử lý các chương trình con ở trên xin dành cho các bạn. Chào tạm biệt và chúc thành công.

32 .Hệ đếm nhị phân trong máy tính những bài toán hay về xử lý bít

Tác giả: Nguyễn Phi Hùng

Con người thường xử lý những bài toán bằng hệ đếm thập phân, nhưng có lẽ hệ đếm nhị phân lại là hệ đếm được ưa thích hơn với những chiếc máy tính. Trong PC của bạn, các con số được thể hiện bằng các bit dưới dạng nhị phân (0,1). Khi chúng ta nhập các con số dưới dạng thập phân, máy sẽ xẽ xử lý và đưa về hệ nhị phân dưới dạng các bit mà chúng ta có thể tạm hiểu là 0 và 1. Và chỉ tới khi xuất dữ liệu ra (màn hình hoặc tệp văn bản), các số mới được thể hiện lại ở dạng thập phân. Mọi tệp trong máy thực chất là dòng các byte, trong đó có những byte giữ vị trí đặc biệt dùng để điều khiển việc hiển thị.

Một số người có thể đã hiểu ít nhiều về bit và các bài toán xử lý bit, một số người thì không. Nhưng điều đó không phải là quan trọng, khi mà điều quan trọng là chúng ta sẽ đề cập đến một vấn đề tưởng chừng như rất phức tạp (đối với những người chưa biết nhiều về nó), nhưng thật ra lại thật là đơn giản mà hiệu quả lại rất cao trong khi xử lý những bài toán hay và khó. Sau đây là những thông tin sơ lược về bit trong PC.

- Trong PC, mỗi số nguyên đều được biểu diễn ở hệ nhị phân bởi một dãy các bit 0 và 1.

- Các bit được mã số với những số hiệu từ phải sang trái.

7 6 5 4 3 2 1 0

- 1byte=8bit

- 1word=16bit

- Hàm sizeof (x) = số byte của số nguyên x.

- Các phép toán xử lý bit:

+ NOT x : phép đảo bit, đổi các giá trị trong mỗi bit của x từ 0->1,1->0.

+ x OR y : Phép cộng logic trên các bit.Thực hiện trên từng cặp bit tương ứng của các toán hạng theo bảng sau.

a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

+ x AND y : Phép nhân logic trên các bit. Thực hiện trên từng cặp bit tương ứng của các toán hạng theo bảng sau.

a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1

+ x XOR y : Phép cộng logic trên các bit.Thực hiện trên từng cặp bit tương ứng của các toán hạng theo bảng sau.

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

+ x SHR i : Phép dịch phải, cho giá trị có được từ số nguyên x sau khi dịch sang phải i bit.

+ x SHL i : Phép dịch trái, cho giá trị có được từ số nguyên x sau khi dịch sang trái i bit.

Trên đây là một số phép toán làm việc trên các bit mà ta hay dùng, trên cơ sở đó, ta xây dựng được một số hàm, thủ tục hay dùng sau.

- Hàm lấy bit.

```
Function LayBit(x:word; i:byte):byte;
Begin
LayBit:=(x SHR i) and 1
End;
```

Hàm trả về giá trị là bit thứ i của số nguyên x.

- Thủ tục bật bit.

```
Procedure BatBit(Var x:word; i:byte);
Begin
X:=x OR (1 SHL i)
End;
```

Thủ tục gán trị 1 cho bit thứ i trong số nguyên x.

- Thủ tục tắt bit.

```
Procedure TatBit(Var x:word; i:byte);
Begin
X:=x AND (NOT(1 SHL i))
End;
```

Thủ tục gán trị 0 cho bit thứ i trong số nguyên x.

- Hàm Lấy số.

```
Function LaySo(x:word;j,i:byte):byte;
Var
```

```
K:byte;

Begin

K:= (8*sizeof(x) - j - 1);

LaySo := (x SHL k) SHR (i+k)

End;
```

Hàm trả về giá trị là số tạo bởi các bit từ trái sang phải, từ j -> i của số x với điều kiện 8*sizeof(x)> j ≥ i ≥ 0.

Tiếp đến, chúng ta sẽ giải quyết một số bài toán khá hay về xử lý bit.

Bài toán: Sắp xếp dãy.

Cho tệp Data.dat gồm các số nguyên không âm nhỏ hơn 500 000 đôi một khác nhau. Hãy xắp tệp theo thứ tự tăng dần và đưa ra tệp Result.dat.

Bài toán trên khiến ta cảm thấy khó chịu về giới hạn dữ liệu của nó, nếu ta nghĩ đến việc dùng mảng để lưu trữ và xắp xếp thuần tuý thì quả là ngớ ngẩn bởi vì bộ nhớ 64K chẳng thấm tháp vào đâu và riêng việc xắp xếp đã chiếm mức thời gian kỉ lục rồi. Ta cũng có thể dùng phương pháp xắp xếp ngoài Merge Sort (dùng file để sắp xếp), tuy nhiên, đây không phải là giải pháp hữu hiệu vì vấn đề thời gian. Tuy nhiên, phương pháp sắp xếp đánh dấu bằng một vài thủ thuật dùng bit thô sơ lại cho ta một kết quả khá là mĩ mãn. Ta chỉ cần dùng một mảng A kiểu byte với khoảng 63000 phần tử là thừa đủ để xử lý bài toán này. Ta tưởng tượng mảng A là một đoàn tầu, và mỗi toa tầu là một phần tử có 8 chỗ ngồi, chỗ ngồi được đánh dấu là 1 nếu có người ngồi, là 0 nếu ngược lại. Ban đầu đoàn tầu không có người ngồi, ta gán mảng bằng 0. Và với mỗi phần tử đọc từ file ra, ta đánh dấu bit tương ứng trong mảng cho tới khi hết file bằng thủ tục batbit. Để lấy kết quả ghi ra file, ta dùng function laybit. Sau đây là chương trình thể hiện thuật toán:

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q+,R+,S+,T-,V+,X+}

PROGRAM XAP_XEP_DAY_VOI_DU_LIEU_LON;

Const

Datin = 'Data.dat';
```

```
Datout = 'Result.dat';
Bt=8;
MAX=63000;
Var
a:array[0..MAX] of byte;
f:text;
Procedure Finish;
Var
i,j:longint;
Function LayBit(x,j:longint):byte;
Begin
LayBit:=(x shr j) and 1;
End;
Begin
Assign(f,DATOUT);rewrite(f);
For i:=0 to max do
For j:=0 to bt-1 do
If LayBit(a[i],j)=1 then Write(f,(i*bt+j),' ');
Close(f)
End;
Procedure Start;
```

```
Var
i,tg:longint;
Procedure BatBit(x,y:longint);
Begin
A[x]:=a[x] or (1 shl y)
End;
Begin
Fillchar(a, sizeof (a), 0);
Assign(f,DATIN);reset(f);
While not seekeof(f) do
Begin
Read(f,tg);
BatBit(tg div bt,tg mod bt)
End;
Close(f)
End;
BEGIN
Start;
Finish
END.
```

Một số bài tập tương tự.

Bài số 1: Trộn tệp.

Cho 2 tệp 'In1.dat' và 'In2.dat' chứa các số nguyên không âm có thể rất lớn (không lớn hơn 500 000). Hãy tạo tệp 'Out.dat' chứa các số có mặt trong 2 tệp trên, tuy nhiên, không được trùng nhau.

Bài số 2: Tỉa xâu.

Cho file văn bản Str.inp gồm nhiều dòng, mỗi dòng không quá 30 kí tự, chỉ chứa hai kí tự " * " và " ". Hãy loại bỏ bớt các dòng giống nhau rồi in kết quả ra file "Str.out" gồm các dòng khác nhau.

Các bạn thấy chưa, kĩ thuật xử lý bit đâu phải là quá khó, và sức mạnh của nó cũng đâu đến nỗi tồi !. Sử dụng kĩ thuật xử lý bit để giải toán, bạn sẽ có được hai cái lợi.

- Thứ nhất, với những bài toán đánh dấu, bạn sẽ thừa hưởng bộ nhớ lớn gấp 8 lần bình thường, không nhỏ đâu!.

- Thứ hai, tốc độ chương trình của bạn sẽ nhanh hơn do làm việc trực tiếp với ngôn ngữ máy, không phải thông qua hệ đếm cơ số 10.

33 .**Một số bài toán và thuật toán liên quan tới số nguyên tố**

Tác giả: Nguyễn Văn Trường

Trong các đề thi Olimpic tin học và trong nhiều bài toán tin chúng ta thường gặp dạng bài tập có liên quan trực tiếp hay gián tiếp đến số nguyên tố. Các dạng bài tập này có thể là tìm số nguyên tố thoả mãn một điều kiện nào đó hay kiểm tra tính nguyên tố của một số cho trước. Bên cạnh đó, vấn đề tìm các số nguyên tố rất lớn đang được rất nhiều người quan tâm vì sự ứng dụng thực tế của nó; đặc biệt là trong lĩnh vực mã hoá và an toàn thông tin, việc tìm ra và sử dụng các số nguyên tố lớn đảm bảo tính an toàn rất cao cho hệ thống mã hoá công khai nổi tiếng RSA. Bài viết này xin được giới thiệu với bạn đọc một số bài tập và một số thuật toán hiệu quả có liên quan tới số nguyên tố. Trước hết chúng ta cùng nghiên cứu một vài kiến thức làm cơ sở toán học cho các thuật toán sẽ trình bày
Định nghĩa: Một số nguyên dương khác 1 được gọi là số nguyên tố nếu nó chỉ có hai ước số dương là 1 và chính nó. Một số nguyên dương khác 1 không là số nguyên tố thì được gọi là hợp số.
Định lý 1. Ước số dương bé nhất d khác 1 của một số nguyên dương a lớn hơn 1 là một số nguyên tố.
Chứng minh Ta chứng minh bằng phản chứng như sau:
Giả sử d là hợp số => d = qd' , với 1 < q, d' < d. Mặt khác do a = dp theo giả thiết

nên a = qd'p => a có một ước số dương là d' với 1< d'< d. điều này là vô lý vì vậy d là số nguyên tố.
Bài 1. Hãy phân tích một số nguyên n, n > 1 thành tích của các thừa số nguyên tố.
Ví dụ với n = 15 ta có dạng phân tích là n = 3.5
Theo định lý 1, ta có thể dễ dàng đưa ra thuật giải như sau:
B1.Tìm ước số dương d nhỏ nhất của n, (d >1) và gán n = n div d;
B2. Lặp lại B1 nếu n >1, ngược lại thì tập hợp các ước số đã tìm được chính là các thừa số nguyên tố của n.
Định lý 2. Ước số dương bé nhất khác 1 của một hợp số a > 1 là một số nguyên tố không vượt quá căn bậc hai của ạ
Định lý này cho phép ta kết luận: Một số nguyên dương n lớn hơn 1 là số nguyên tố nếu nó không có ước số dương lớn hơn 1 và nhỏ hơn hoặc bằng $\sqrt{n}$
Chứng minh Giả sử p là ước số dương bé nhất khác 1 nhất của a =>p là số nguyên tố (Theo định lý 1) và a = pd, với d≥ P => pd≥ p2 =>p2 ≤a =>p ≤$\sqrt{a}$
Định lý 3. (Định lý cơ bản của số học) Mọi số tự nhiên lớn hơn 1 đều phân tích được thành tích của các thừa số nguyên tố và sự phân tích này là duy nhất nếu không kể đến thứ tự của các thừa số.
Chứng minh Giả sử 1 < a$\in$ N. Vì a > 1 nên a có ước nguyên tố là p1. Do đó a = p1a1, 1 ≤ a1 < a.
Nếu a1 = 1 => a= p1.
Nếu a1 > 1 => a1 có ước nguyên tố là p2. Do đó a1 = p2a2, 1≤ a2 < a 1.
Nếu a2 = 1 => a1 = p2 => a = p1p2.
Nếu a2 > 1 => a2 có ước nguyên tố là p3. Do đó a2 = p3a3, 1≤ a3 < a2. Cứ tiếp tục như vậy thì quá trình sẽ phải dừng lại và an = 1 (vì a > a1 > a2 >...> an =1). Khi đó a = p1p2p3...pn (n > 0).
Phần chứng minh tính duy nhất của phân tích trên xin dành cho bạn đọc.
Dạng phân tích chính tắc của một số tự nhiên a > 1.
Cho a là một số tự nhiên lớn hơn 1, khi ta phân tích số a thành tích các thừa số nguyên tố thì có thể gặp cùng một thừa số nhiều lần. Nếu p1, p2,...,pn xuất hiện theo thứ tự m1, m2,..., mn lần thì ta có thể viết a dưới dạng:
$a=p_1^{m_1}p_2^{m_2}\dots p_n^{m_n}$ và dạng này được gọi là dạng phân tích chính tắc hay dạng phân tích tiêu chuẩn của ạ
Ví dụ 720 = 24.32.5
Bài 2. Cho trước số nguyên dương N, hãy tìm dạng phân tích chính tắc của N!=1.2.3...N. Dữ liệu vào trong file văn bản NUM.INP trong đó chứa số nguyên dương N (2≤N≤ 10 000). Dữ liệu đưa ra file NUM.OUT bao gồm nhiều dòng, mỗi dòng gồm hai số ghi cách nhau một dấu cách: số thứ nhất là thừa số nguyên tố của N và số thứ hai là số mũ tương ứng.
Ví dụ:
NUM.INP

4
NUM.OUT
2 3
3 1
Sàng Eratosten (Sieve of Eratosthenes). Sử dụng định lý 2 chúng ta có thể tìm tất cả các số nguyên tố nhỏ hơn hoặc bằng một số nguyên dương n cho trước. Phương pháp này được gọi là sàng Eratosten vì nó được nhà toán học cổ Hy Lạp Eratosten (276-194 T.C.N) phát minh rạ Trước hết ta thấy rằng, mọi hợp số nhỏ hơn n sẽ có ước nguyên tố không vượt quá $\sqrt{n}$. Do đó các số nhỏ hơn hoặc bằng n, sau khi loại bỏ đi những số là bội của các số nguyên tố nhỏ hơn hay bằng $\sqrt{n}$, sẽ đều là các số nguyên tố. Phương pháp này được trình bày cụ thể như sau:
B1.Viết các số từ 1 tới n thành một dãy
B2.Số 1 bị xoá.
B3.Tìm số đầu tiên m không bị xoá trong dãy và kiểm tra số này: nếu m > $\sqrt{n}$thì dừng thuật toán và các số không bị xoá chính là các số nguyên tố cần tìm, ngược lại thì ta xoá các số là bội số của m (ngoại trừ m).
B4. Lặp lại từ bước 3.
Bài 3. Cho trước một số nguyên dương N, 1< N <10 000, hãy viết chương trình tìm tất cả các số nguyên tố theo phương pháp sàng Eratosten.
Chúng ta đã quen thuộc với một bài toán kinh điển sau: Cho trước một số nguyên dương n, cho biết n có phải là số nguyên tố hay không? Với bài toán này, đa số chúng ta dùng ngay một thuật giải quen thuộc là kiểm tra tính chia hết của n cho các số nguyên lớn hơn 1 và nhỏ hơn hay bằng $\sqrt{n}$. Tuy nhiên cách làm này rất tốn kém về thời gian, và thường chỉ có ý nghĩa khi ta thao tác với những số n nhỏ. Để tăng tốc độ cho thuật toán cho bài toán này, ta cần có những giải thuật hiệu quả hơn. Dưới đây trình bày 2 định lý quan trọng góp phần cải tiến thuật toán tìm số nguyên tố và kiểm tra tính nguyên tố của một số nguyên dương cho trước.
Định lý 4. Nếu n là một số nguyên tố lớn hơn 5 thì số dư của n cho 6 phải là 1 hoặc 5.
Định lý này cho phép chúng ta tìm các số nguyên tố theo bước tăng không phải là đơn vị mà là 2, 4, 2, 4...
Chứng minh
Để chứng minh định lý ta chỉ cần chứng tỏ rằng nếu n chia cho 6 dư 2, 3 hoặc 4 thì n phải là hợp số. Thật vậy, giả sử n chia cho 6 dư 2 => n = 6q+2 = 2(3q+1), q $\in$ Z + . Do đó n là hợp số. Nếu n chia cho 6 dư 3 hoặc 4 thì ta cũng chứng minh được n là hợp số.
Định lý 5. Một số nguyên dương n là số nguyên tố nếu nó không chia hết cho bất kỳ số nào trong dãy các số nguyên tố nhỏ hơn hoặc bằng $\sqrt{n}$.
Chứng minh Từ định lý 2 ta thấy rằng để chỉ ra n là số nguyên tố thì ta chỉ cần

chứng tỏ rằng nếu n không chia hết bất kỳ số nào trong dãy các số nguyên tố nhỏ hơn hoặc bằng $\sqrt{n}$thì n cũng không chia hết cho bất kỳ số nào trong dãy các số nguyên nhỏ hơn hoặc bằng $\sqrt{n}$. Thật vậy, giả sử n chia hết cho một hợp số m, m ≤ $\sqrt{n}$tức là n = mk, 1< m,k < n. Vì m là hợp số nên tồn tại một ước nguyên tố d của m sao cho m = qd, 1< q,d < m => n = mk = qdk, hay m chia hết cho một số nguyên tố d với d < m < n, điều này là mâu thuẫn với giả thiết.
Sử dụng định lý này ta thấy rằng để kiểm tra một số nguyên n có phải là số nguyên tố hay không ta chỉ cần kiểm tra n không chia hết cho bất kỳ số nào trong dãy các số nguyên tố nhỏ hơn hay bằng $\sqrt{n}$.
Từ hai định lý trên ta xây dựng một thuật toán tạm chấp nhận được để kiểm tra tính nguyên tố của một số nguyên dương n thông qua một hàm như sau:

```
function isPrime(n:longint):boolean;
var i:longint;j:byte;
begin
isPrime:=true;
if n<2 then begin isPrime:=false;exit end;
if (n=2)or(n=3)then begin isPrime:=true; exit end;
if n mod 2=0 then begin isPrime:=false;exit end;
if n mod 3=0 then begin isPrime:=false;exit end;
i:=5;j:=2;
while i<=trunc(sqrt(n))do
begin
if n mod i =0 then begin isPrime:=false; exit end;
i:=i+j; j:=6-j
end;M
end;
```

Ta có thể cải tiến thuật toán bằng cách lưu các số nguyên tố hỏ hơn hay bằng $\sqrt{n}$vào một mảng a trước khi tiến hành kiểm tra với n. Thuật toán được mô tả như sau:

```
function isPrime(n:longint):boolean;
var i,j:longint;
begin
isPrime:=true;
i:=1;j:=round(sqrt(n));
{mang a chua cac so nguyen to da sap xep tang dan}
while a[i]<=j do
begin
if n mod a[i] = 0 then
begin isPrime:=false;exit end;
```

```
i:=i+1;
end;
end;
```

Cách làm này chỉ tối ưu khi bài toán liên quan đến tìm nhiều số nguyên tố (nhỏ). Tuy nhiên cách làm này tốn kém về bộ nhớ vì ta cần sử dụng mảng a để lưu các số nguyên tố nhỏ hơn hoặc bằng $\sqrt{n}$

Định nghĩa Số các số nguyên tố không vượt quá n ký hiệu là π (n).

Ta hãy quan sát giá trị của hàm π với một vài giá trị đặc biệt của n như bảng sau:

N	π(n)
10^3	168
10^4	1229
10^5	9592
10^6	78498
10^7	664579
10^8	5761455
10^9	50847534
10^{10}	455052512
10^{11}	4118054813
10^{12}	37607912018
10^{13}	346065536839
10^{14}	3204941750802

Ta thử ước lượng chi phí bộ nhớ để lưu tất cả các số nguyên tố không vượt quá 1014: số các số nguyên tố đó là 3 204 941 750 802 số, ta giả sử trung bình mỗi số đó dài 7 ký tự. Như vậy cần 3204941750802 x 7 byte bộ nhớ hay tương đương với khoảng 20 000 Gb, thật là một chi phí quá lớn ! Vì vậy, cần nhấn mạnh rằng: tuỳ thuộc vào đầu bài để lựa chọn phương pháp nào tối ưu

Bài 5. Cho trước một số nguyên dương N, 1< N< 500 000 hãy tìm giá trị của hàm π(n).

Bài 6. Cho trước một số nguyên dương N, 1< N< 1018 hãy tìm một số nguyên tố M nhỏ nhất và lớn hơn N. Ví dụ với N = 1000 thì M= 1009 với N = 10000000000000000 thì M = 10000000000000061.

Phân tích và hướng dẫn giải: thuật toán đưa ra rất đơn giản, tuy nhiên cần tìm ra thuật toán hiệu quả để chạy trong thời gian chấp nhận được. Giải pháp đưa ra như

sau:
Xuất phát từ N, tìm cách tăng N theo bước tăng 2, 4, 2, 4... Mỗi khi tăng N, kiểm tra xem N có phải là số nguyên tố không bằng cách duyệt các ước số của N cũng theo bước tăng 2, 4, 2, 4... và các ước đó không vượt quá căn bậc hai của N.
Vấn đề là cần cài đặt các phép toán phù hợp để thoả mãn yêu cầu dữ liệu của đầu bài là N có thể lớn tới 17 chữ số, M có thể lớn tới 18 chữ số. Như vậy, để làm được bài này ta cần sử dụng cấu trúc dữ liệu là số thực và xây dựng tập hợp một vài phép toán như DIV (phép chia lấy phần nguyên), MOD (phép chia lấy phần dư)... và ta gọi cấu trúc dữ liệu đó là số giả nguyên. Khi có số giả nguyên thì ta có thể thao tác với các số nguyên lớn tới 20-21 chữ số (bằng số chữ số có nghĩa của kiểu dữ liệu thực).
Thuật toán được minh hoạ qua chương trình sau:

```
{$N+,B-}
uses crt;
type bigInt=comp;
function DivB(a,b:bigint):bigint;
begin
DivB:=int(a/b);
{ham cho phan nguyen cua mot so thuc va
tra ve la kieu so thuc}
end;
function ModB(a,b:bigint):bigint;
begin
ModB:=a-b*int(a/b);
end;
function EvenBig(n:Bigint):boolean;
begin
EvenBig:=ModB(n,2)=0;
end;
function prime(n:bigInt):boolean;
var i,j:longint;c:real;
begin
if (n<2) then
begin prime:=false;exit; end;
if (n=2) or (n=3) then
begin prime:=true;exit; end;
if EvenBig(n) or(ModB(n,3)=0) then
begin prime:=false;exit; end;
prime:=true;
c:=sqrt(n);
i:=5;j:=2;
```

```
while (i<=c) do
begin
if ModB(n,i)=0 then
begin prime:=false;exit; end;
i:=i+j;j:=6-j;
end;
end;
function NextPrim(n:bigint):bigint;
var i:bigint;j:byte;
begin
if n<2 then
begin NextPrim:=2; exit; end;
if (n=2) or (n=4) then
begin Nextprim:=n+1;exit; end;
if (n=3) or (n=5) then
begin Nextprim:=n+2;exit; end;
i:=ModB(n,6);
if (i=0)then
begin n:=n-1;j:=2 end
else if i=5 then j:=2
else begin n:=n-i+1;j:=4 end;
While true do
begin
n:=n+j;
if Prime(n) then
begin
nextPrim:=n;
exit;
end;
j:=6-j;
End;
end;
procedure GenP;
var f:text;
n:bigint;
begin
assign(f,'Num.txt');
reset(f);
readln(f,n);
close(f);
assign(f,'Primętxt');
```

```
rewrite(f);
writeln(f,Nextprim(N):0:0);
close(f);
end;
begin
genP
end.
```

Tuy nhiên, cách làm như trên vẫn có thể được cải tiến để tăng tốc độ nếu ta khai thác triệt để các hàm về số học có sẵn của Pascal để không tốn thời gian gọi hàm. Cụ thể ta có thể không dùng các hàm DIV và MOD như trong chương trình trên và cải tiến bằng cách: để kiểm tra một số m có chia hết cho n hay không tương đương với kiểm tra hàm frac(m/n) có bằng 0 không, và để tìm phần dư của m/n ta lấy kết quả của phép tính frac(m/n)*n (Hàm frac có đối số là số thực và cho kết quả là phần phân của đối số, ví dụ Frac(12.123) sẽ cho kết quả là 0.123). Bạn đọc có thể tự sửa lại chương trình trên và kiểm nghiệm sự cải thiện tốc độ của thuật toán (nếu có nhu cầu các bạn có thể liên hệ với tác giả để có chương trình nguồn).

Bài 7. Số nguyên tố 86311 có dạng đặc biệt là nếu đem chia nó cho 10000, 1000 và 100 thì ta được các số dư 8311, 311 và 11 đều là các số nguyên tố. Hãy tìm tất cả các số nguyên tố có N chữ số ($N \leq 9$) có dạng trên (tức là mọi số dư của số đó cho 10^i, $1 < i < N$ đều là số nguyên tố)

Hiện nay, với những kiến thức của toán học hiện đại, người ta đã xây dựng những thuật toán hiệu quả để giải quyết bài toán với các số nguyên tố lớn tới hàng trăm chữ số với thời gian tính toán chỉ vài chục giây Phần tiếp theo trình bày một số thuật toán đề cập tới vấn đề trên.

Còn nữa

34. **Một số bài toán và thuật toán liên quan tới số nguyên tố**

Tác giả: Nguyễn Văn Trường

Định nghĩa (Số nguyên tố Mersen): Nếu p là số nguyên tố và $Mp = 2p - 1$ cũng là số nguyên tố thì Mpgọi là số nguyên tố Mersen.

Định lý 6 . (The Luacas Lehmer Test) P là một số nguyên, đặt r1 = 4 và với mọi k > 1, rk $r2k - 1 - 2$ (mod Mp), khi đó Mp là số nguyên tố khi và chỉ khi $rp - 1 \equiv 0$ (mod Mp).

Chú ý: ký hiệu $' \equiv '$ ở trên để chỉ ký hiệu đồng dư, một khái niệm phổ biến trong số học. Nếu a và b chia cho m có cùng số dư thì ta gọi là a đồng dư với b theo modun m và ký hiệu $a \equiv b$ (mod m).

Định lý này dùng để kiểm tra Mp có là số nguyên tố hay không với độ phức tạp thuật toán là O(p3).

Ví dụ: cho $M5 = 25 - 1 = 31$

Ta có r1 = 4
r2 ≡ 42 − 2 = 14 (mod 31)
r3 ≡ 142 − 2 = 8 (mod 31)
r2 ≡ 82 − 2 = 0 (mod 31)
Do đó M5 = 31 là số nguyên tố.
Bằng cách này có thể sinh ra các số nguyên tố 2p −1 với p là các số nguyên tố: 3, 5, 7, 13, 17, 19, …
Thuật toán được mô tả như sau:
Input: Số nguyên tố p
Output: Mp = 2p −1 có phải là số nguyên tố không?
B1. Tính Mp = 2p −1
B2. r:=4;
B3. For i:=2 to p − 1 do
R:= r2 − 2 mod Mp
B5. if r = 0 then {kết luận Mp là số nguyên tố }
else{kết luận Mp không phải là số nguyên tố}.
Bằng phương pháp này, người ta có thể tìm ra các số nguyên tố lớn đến hàng trăm, thậm chí hàng ngàn chữ số. Bảng dưới đây tổng kết một số kết quả đã đạt được:

P	Số chữ số thập phân trong M_p	Năm tìm ra
2	1	Từ thời cổ đại
3	1	Từ thời cổ đại
5	2	Từ thời cổ đại
7	3	Từ thời cổ đại
13	4	Thế kỷ 15
17	6	1603
19	6	1603
31	10	1772
61	19	1883
89	27	1911
107	33	1914
127	39	1976
521	157	1952
…		
216091	65050	1985
756839	227832	1992

Bài 8 . Hãy lập trình tìm tất cả các số nguyên tố Mersen nhỏ hơn 1040. Kết quả ghi vào file text Mersen.txt gồm nhiều dòng: mỗi dòng gồm hai số ghi cách nhau một dấu cách, số thứ nhất là số nguyên tố p, số thứ hai là số nguyên tố Mersen Mp.

Ví dụ một số dòng đầu tiên của file Mersen.txt

```
2 3
3 7
5 31
7 127
13 8191
17 131071
19 524287
31 2147483647
61 2305843009213693951
89 618970019642690137449562111
107 162259276829213363391578010288127
```

Thuật toán để giải bài toán này không khó, cái khó là sử dụng cấu trúc dữ liệu nào cho phù hợp. Một phương pháp có thể chấp nhận được là sử dụng cấu trúc dữ liệu kiểu mảng hoặc kiểu xâu. Bạn đọc có thể tham khảo chương trình minh hoạ cho thuật toán được trình bày như sau:

```
{$B-}
type big = array[0..100] of byte;
function isPrime(n:longint):boolean;
var i,j:longint;
begin
isPrime:=true;
ifn<2 then begin isPrime:=false;exit end;
if(n=2)or(n=3)then begin isPrime:=true; exit end;
ifn mod 2=0 then begin isPrime:=false;exit end;
ifn mod 3=0 then begin isPrime:=false;exit end;
i:=5;j:=2;
whilei<=trunc(sqrt(n))do
begin
if n mod i =0 then begin isPrime:=false; exit end;
i:=i+j; j:=6-j
end;
end;
function sodu(s1,s2:string):string;
var a,b:array[0..200] of byte;
l,la,lb,lc,i:byte;s:string;
function ss(d,k:byte):boolean;
var ok,kt:boolean;i,j:byte;
begin
Ifk-d+1>lb then ok:=true
elseif (k-d+1la) then ok:=false
else
```

```
begin
ok:=true;kt:=true;i:=d;j:=1;
while kt and(i<=k) do
if a[i] >b[j] then kt:=false
else
if a[i]
Begin kt:=false;ok:=false End
else
Begin i:=i+1;j:=j+1;end;
end; ss:=ok
end;
procedure tru(d,k:byte);
var tg,phu:byte;j,i:byte;
begin
phu:=0;j:=lb;
fori:=k downto d do
begin
if a[i]-b[j]- phu<0 then
begin
a[i]:=a[i]+10-b[j]-phu; phu:=1;
end
else
begin
a[i]:=a[i]-b[j]-phu; phu:=0;
end; j:=j-1;
end;
end;
procedure xu_ly;
var k,i:byte;
begin
lc:=0;l:=1;k:=lb;
ifnot ss(l,k)and(k>=la) then exit;
ifnot ss(l,k) then k:=k+1;
k:=k-1;
repeat
k:=k+1;
while (a[l]=0) and(l while ss(l,k) do
begin
tru(l,k);
if (a[l]=0) and(l<=la) then l:=l+1;
end;
```

```
untilnot ss(l,la)or(k>la);
while(l end;
procedure test;
var f:text;kk,x,h,i:byte;s:string;j:integer;
begin
la:=0;lb:=0;
for i:=1 to length(s1) do
begin
val(s1[i],x,j);
la:=la+1;a[la]:=x;
end;
for i:=1 to length(s2) do
begin
val(s2[i],x,j);
lb:=lb+1;b[lb]:=x;
end;
a[0]:=0;b[0]:=0;
xu_ly;
end;
begin
test;
ifl>la then Sodu:='0'
else
begin
s:='';
for i:=l to la do s:=s+chr(a[i]+48);
sodu:=s;
end;
end;
function mu2(p:longint):string;
var du,i,j,l:longint;m:array[1..300] of byte;s:string;
begin
m[300]:=1;l:=300;
forj:=1 to p do
begin
du:=0;
fori:=300 downto l do
begin
du:=2*m[i]+du;
m[i]:=du mod 10;
du:=du div 10;
```

```
end;
ifdu>0 then
begin
l:=l-1;
m[l]:=du;
end;
end;
s:='';
for i:=l to 300 do s:=s+chr(ord(m[i])+48);
mu2:=s;
end;
procedure tru(var s:string;x:byte);
var t,i:byte; c:char;
begin
i:=length(s);t:=x;
while(i>0)and(t>0)do
iford(s[i])-48-t<0 then
begin
s[i]:=char(ord(s[i])+10-t);t:=1;i:=i-1;
end
else
begin
s[i]:=char(ord(s[i])-t);t:=0;
end;
end;
function square(s:string):string;{binh phuong}
var x,y:string; a:array[1..300] of byte;
l,i,j,p:integer;
begin
l:=length(s);
forj:=l downto 1 do
a[j]:=ord(s[j])-48;
fori:=l downto 1 do
begin
x:='';p:=0;
for j:=l downto 1 do
begin
p:=p+a[i]*a[j];
x:=chr(p mod 10 +48)+x;
p:=p div 10;
end;
```

```
if p>0 then x:=chr(p+48)+x;
if i=l then y:=x
else
begin
p:=0;
for j:=1 to l-i do x:=x+'0';
forj:=length(y) to length(x)-1 do y:='0'+y;
for j:=length(x) to length(y)-1 do x:='0'+x;
for j:=length(x) downto 1 do
begin
p:=ord(x[j])+ord(y[j])+p-96;
y[j]:=chr(p mod 10 +48);
p:=p div 10;
end;
if p>0 then y:=chr(p+48)+y;
end;
END;
square:=y;
end;
function TestPrim(p:byte; var m:string):boolean;
{m^p-2 co phai la so ng to}
var r:string;j:longint;
begin
ifp = 2 then
begin m:='3'; testprim:=true; exit end;
m:=mu2(p);
tru(m,1);
r:='4';
forj:=2 to p-1 do
begin
r:=square(r);
Tru(r,2);
r:=sodu(r,m);
end;
testprim:=(r='0');
end;
procedure GenP;
var f:text;
n:longint;
m:string;
begin
```

```
assign(f,'Mersen.txt');
rewrite(f);
forn:=1 to 150 do
ifisprime(n) then
begin
if testPrim(n,m)then
writeln(f,n,' ',m);
end;
close(f);
end;
begin
genP
end.
```

Ngoài các thuật toán phổ biến như đã đượctrình bày ở trên, hiện nay người ta đã phát triển nhiều thuật toán khác với một vài ưu điểm. Tuy nhiên trong phạm vi bài viết này không thể trình bày quá chi tiết về các thuật toán đó. Nếu bạn đọc nào quan tâm có thể tìm hiểu trong cuốn sách Elementary number theory andits applications, của tác giả Kenneth H. Rosen.

Bài 9 . (Bài 3 trong đề thi Tin học quốc tế 1994)

Cho một bảng hình vuông gồm 5 x 5 hình vuông nhỏ, trên mỗi ô vuông ghi một chữ số thập phân như hình vẽ.

1	1	3	5	1
3	3	2	0	3
3	0	3	2	3
1	4	0	3	3
3	3	3	1	1

Các chữ số được ghi vào phải thoả mãn các điều kiện sau:

- 12 số gồm: 5 số năm chữ số theo từng hàng đọc từ trái qua phải, 5 số năm chữ số theo từng cột đọc từ trên xuống dưới, 2 số năm chữ số theo hai đường chéo đọc từ trái sang phải đều là các số nguyên tố với năm chữ số có nghĩa.
- Tổng các chữ số của mỗi một trong mười hai số đều bằng một số cho trước. Trong ví dụ trên tổng các chữ số đều bằng 11.

Dữ liệu vào được cho bởi file DS.INP trong đó dòng thứ nhất ghi số là tổng các chữ số của các số cần xây dựng, dòng thứ hai ghi chữ số ở góc trái trên của hình vuông. Kết quả ghi ra file DS.OUT như sau: với mỗi cách điền các chữ số vào các ô thoả mãn các yêu cầu, ghi ra 5 dòng, mỗi dòng ghi năm chữ số. Hai cách điền chữ số xem

như là khác nhau nếu chúng có ít nhất hai chữ số ở hai ô tương ứng là khác nhau.
Ví dụ:
DS.INP
11
1
DS.OUT
1 1 3 5 1
1 4 0 3 3
3 0 3 2 3
5 3 2 0 1
1 3 3 1 3
1 1 3 5 1
3 3 2 0 3
3 0 3 2 3
1 4 0 3 3
3 3 3 1 1
1 3 3 1 3
1 3 0 4 3
3 2 3 0 3
5 0 2 3 1
1 3 3 3 1
Nguyễnvăn Trường

35 .**Cặp ghép tối ưu lực lượng không bằng nhau**

Tác giả: Trần Đức Thiện

Cho G = (X υY, E) là đồ thị hai phía đầy đủ, trong đó: X , Y là hai tập hữu hạn gồm n phần tử, E là tập các cạnh của đồ thị và với mỗi cạnh được gán với một trọng số C ij . Cần tìm tập cặp ghép đầy đủ M có tổng trọng số lớn nhất.
Về mặt thuật toán các tác giả đã trình bày rất kĩ, nhưng chương trình phần lớn còn chưa được tối ưu, khó hiểu, khó cài đặt. Bằng kinh nghiệm lập trình tôi xin đưa ra chương trình cài đặt thuật toán tìm cặp ghép tối ưu đạt max. Trước khi theo dõi chương trình, xin các bạn hãy chú ý những điểm sau:
+) Các đỉnh thuộc tập Y được đánh số từ n+1 đến n+n (hay 2n) để dễ quản lí và cài đặt.
+) Ta vẫn dùng các biến quen thuộc với ý nghĩa đã được các tác giả của các số báo trước đã trình bày, nếu không hiểu các bạn xem lại các số trước hoặc liên hệ với tôi.
+) Trong bài số 11-2001 tác giả Lê Văn Chương dùng 3 mảng là: Tr (trước), S, T (theo định nghĩa). ở đây tôi chỉ cần dùng mảng truoc (Trước) là được, bởi vì mảng S, T cũng chỉ là để biết xem đỉnh u thuộc X hay v thuộc Y có đi đến được hay không

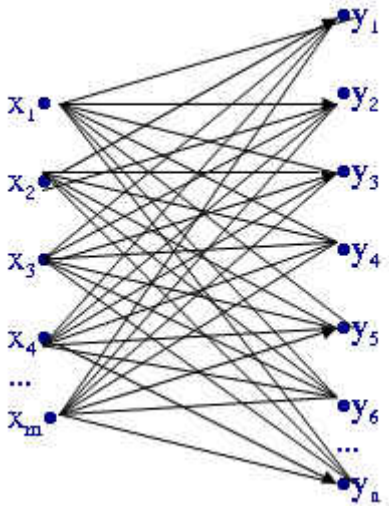
mà thôi. Trong chương trình của tôi thì đỉnh u được loang tới (đi tới được) nếu truoc[u] > 0 (cho cả tập X và Y). Và còn nhiều mảng nữa không cần thiết được tôi bỏ đi.
+)Trong thủ tục tìm đường đi khi lấy đỉnh v ra khỏi hàng đợi, nếu v > n (v thuộc Y) thì tác giả phải quét tất cả các đỉnh của X để lấy đỉnh kết nạp. Tôi chỉ cần kết nạp ngay y[v-n] bởi vì: Khi chúng ta đi đến v và (u,v) thuộc tập cặp ghép hiện thời thì u chắc chắn chưa đi tới và B[v] + A[u] luôn bằng C[u,v] (là Fx[u] + Fy[v] = C[u,v] trong chương trình của tôi). Tôi dùng mảng y lưu cặp ghép mà không dùng mảng x(Cg) giống tác giả là vì lí do này.
+) Các bạn sẽ thấy rằng trong chương trình của tôi tổng lượng bộ nhớ được dùng là rất ít, đặc biệt là các biến được phân bố cả vào chương trình con, khiến chương trình chính dôi ra rất nhiều.
(Tôi khai integer là để cho maxn = 180!)
+) Tôi hoàn toàn đồng ý với tác giả Nguyễn Tấn Dũng, chỉ cần sửa như tác giả nói là có thể được chương trình cặp ghép tối ưu đạt min, ở đây tôi đưa ra chương trình max để các bạn dễ so sánh với chương trình ở số 11-2001.
{Dữ liệu vào: Tệp CAPGHEP.INP chứa n và ma trận C ij }
{Dữ liệu ra: Tệp CAPGHEP.OUT chứa tập cặp ghép cần tìm (dạng xi -yj)}

```
PROGRAM CHUONG_TRINH_TOI_UU;
uses crt;
const finp = 'capghep.inp'; fout = 'capghep.out';
maxn = 100;
var f :text;
c:array[1..maxn,1..maxn] of integer; { Mảngtrọng số }
y:array[1..maxn] of integer{Byte};{ y[v]= u nghĩa là (u,v) đang thuộc tập cặp ghép hiện thời}
fx,fy:array[1..maxn] of longint; {}
truoc:array[1..2*maxn] of integer; {Byte} { Lưu trữ đỉnh trước trong đường tăng cặp ghép }
n,u,v:byte;
procedure nhap; { = ReadF }
var i,j :byte;
begin
assign(f,finp); reset(f);
readln(f,n);
for i:= 1 to n do
begin
for j:= 1 to n doread(f,c[i,j]);
readln(f);
end; close(f);
end;
```

```
procedure khoitao; { Khởi tạo tập cặp ghép rỗng }
var i,j :byte;
begin
for i := 1 to n do
begin
y[i] := 0; fy[i] := 0;
end;
for i:= 1 to n do
begin
fx[i]:= c[i,1];
for j:= 2 to n do if c[i,j] > fx[i] then fx[i]:= c[i,j];
end;
end;
procedure loang(u :byte); { = FindPath(u). Tìm đường tăng cặp ghép từ đỉnh tự do u
thuộc X}
var i,x,dq,cq :integer;
q:array[1..2*maxn] of integer;
begin
for i:= 1 to 2*n do truoc[i]:= 0; { Khoi tao }
truoc[u]:= u;
dq:= 1; cq:= 1; q[cq]:= u;
while dq <= cq do
begin
x:= q[dq]; inc(dq);
if x > n then begin inc(cq); q[cq]:= y[x-n]; truoc[y[x-n]]:= x; end
else
begin
for i:= n+1 to 2*n do if truoc[i] = 0 then
if fx[x] + fy[i-n] = c[x,i-n] then
begin
inc(cq); q[cq]:= i; truoc[i]:= x;
if y[i-n] = 0 then begin v:= i; exit; end;
end;
end;
end;
end;
procedure SuaNhan; { = Change }
var d :longint;
i,j:byte;
begin
d:= maxlongint;
```

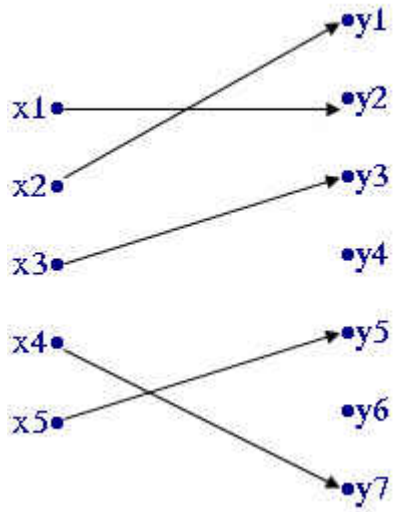
```
for i:= 1 to n do if truoc[i] > 0 then
for j:= 1 to n do if truoc[j+n] = 0 then
if fx[i] + fy[j] - c[i,j] < d then d:= fx[i]+fy[j] - c[i,j];
for i:= 1 to n do if truoc[i] > 0 then dec(fx[i],d);
for j:= 1 to n do if truoc[j+n] > 0 then inc(fy[j],d);
end;
procedure tangcapghep; { = IncCg. X âydựng tập cặp ghép mới từ đường tăng cặp
ghép. Quá ngắn phải không bạn }
begin
while v <> u do
begin
y[v-n]:= truoc[v];
v:= truoc[truoc[v]];
end;
end;
procedure ghep;
begin
for u:= 1 to n do
begin
v:= 0;
loang(u);
while v = 0 do begin suanhan; loang(u); end;
tangcapghep;
end;
end;
procedure print;
var tong :longint;
i:byte;
x:array[1..maxn] of integer;
begin
tong := 0;
for i:= 1 to n do begin x[y[i]]:= i; inc(tong,c[y[i],i]); end;
assign(f,fout); rewrite(F);
writeln(f,tong);
for i:= 1 to n do writeln(f,i,' ',x[i]);
close(f);
end;
Begin
nhap;
khoitao;
ghep;
```

print;
End.
Mở rộng bài toán cặp ghép:
Bài toán cặp ghép bình thường (I X I=I Y I= n) đã được giải quyết triệt để. Nhưng nếu bây giờ ta thay tập X gồm m đỉnh, Y gồm n đỉnh($m \leq n$), vẫn ma trận trọng số là C ij với i từ 1 tới m , j từ 1 tới n , và cần ghép mỗi đỉnh tập X với 1 đỉnh tập Y để được tập cạnh M có tổng trọng số lớn nhất (mỗi đỉnh tập Y ghép với tối đa 1 đỉnh của tập X (có thể không được ghép)). Khi đó thuật toán cặp ghép đã nêu không thể giải quyết được vấn đề này. Kể cả khi bạn dùng đến luồng cũng phải bó tay mà thôi.



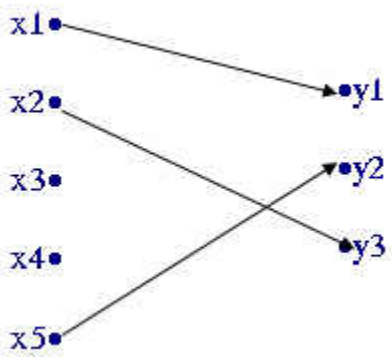
Có thể lấy ví dụ với bài toán sau:
Bài toán: Có m linh kiện chức năng giống nhau và cón máy. Nhưng mỗi linh kiện được sản xuất ở một công ty khác nhau và có đặc tính khác nhau. Giám đốc thống kê được một bảng C có m dòng, n cột với ý nghĩa: số ở dòng i, cột j là mức độ phù hợp của linh kiện i với máy j. Biết rằng năng suất tỉ lệ với mức độ phù hợp sau khi lắp, tỉ lệ của các máy là như nhau. Nhiệm vụ của bạn là phân công mỗi linh kiện vào một máy để tổng mức phù hợp là lớn nhất (số linh kiện có thể thiếu hoặc thừa).
Hay ở dạng tổng quát bài toán phát biểu như sau: Cho ma trận m*n các số nguyên dương, tìm trên mỗi dòng, mỗi cột nhiều nhất một số sao cho tổng các sô được chọn là lớn nhất có thể. Kết quả đưa ra vị trí các ô được chọn bao gồm chỉ số dòng và chỉ số cột của nó.
Cách thứ nhất giải bài toán này là ta cho vào tập X thêm (n-m) đỉnh mới và gán trọng số các đỉnh này với mỗi đỉnh thuộc Y là một số t (t cố định chọn bất kì) - Để được ma trận trọng số là C[1..n,1..n]. Sau đó tìm cặp ghép tối ưu đạt max bình thường trên đồ thị mới và chỉ cần sửa lúc xuất kết quả ta chỉ lấy những cặp chứa các đỉnh ban đầu của X và tổng trọng số phải trừ đi (n-m)*t.
*Chứng minh: Giả sử ta tìm được tập cặp ghép tối ưu M trên đồ thị mới với tổng

trọng số tìm được là C.
Khi đó tập cặp ghép này được chia làm hai phần: Phần một là tập M1 các cặp chứa các đỉnh ban đầu của tập X (ghép với các đỉnh của tập Y), tập này có tổng trọng số là C1 = C - (n-m)*t, tập còn lại chứa các đỉnh mới của tập X ghép với các đỉnh còn lại của tập Y (có tổng trọng số là (n-m)*t). Bây giờ ta giả sử tìm được tập M1′ gồm m cặp chỉ chứa các đỉnh ban đầu của tập X (và ghép với các đỉnh của tập Y) mà có tổng trọng số C1′ > C1 thì trong đồ thị mới, từ M1′ ta luôn tìm được cách ghép các đỉnh mới thêm vào với các đỉnh còn lại chưa được ghép của tập Y để được tập cặp ghép mới M′có tổng trọng số là C′ = C1′+(n-m)*t> C. Tức là ta tìm được một tập cặp ghép khác có tổng trọng số lớn hơn tổng trọng số lớn nhất mà thuật toán tìm cặp ghép ban đầu tìm ra. Điều này mâu thuẫn, thuật toán được chứng minh.



Một số điểm chú ý:
+ Lúc đầu tôi quan niệm phải chọn t rất nhỏ (với bài toán tổng trọng số min thì ngược lại, chọn số t rất lớn), nhưng theo cách chứng minh trên ta có thể chọn t bất kì, vì vậy để đơn giản ta chọn t = 0.
+ Chương trình trên tôi giải quyết cả trường hợp m > n. Tức là ta chỉ in những cặp chứa cả x và y cũ.
Tuy nhiên cách trên vẫn chưa phải là cách tối ưu, có nhược điểm là lãng phí bộ nhớ (do phải thêm đỉnh và các cạnh tương ứng), làm giảm tốc độ chạy,làm cho việc lập trình trở lên phức tạp, thêm nhiều công đoạn,...Chúng ta để ý thấy rằng thuật toán cặp ghép cực đại (hay cặp ghép tối đa) làm việc trên đồ thị có số đỉnh tập X khác số đỉnh tập Y. Vì vậy ý tưởng đưa ra là ta cải tiến thuật toán cặp ghép tối ưu theo hướng giống thuật toán cặp ghép tối đa. Tức là trong các biểu thức ta lấy số đỉnh tập X là m chứ không phải n. Chúng ta chỉ cần sửa thuật toán chuẩn một chút là có thể được chương trình cho thuật toán mới này, nhưng bạn phải chúý tới nhập dữ liệu và in kết quả, đây là hai công đoạn có thay đổi nhiều nhất. Tôi xin được đưa ra chương

trình để các bạn tham khảo: (Tôi không giải quyết trường hợp m >n, để dành cho các bạn)




```
PROGRAMCHUONG_TRINH_HOAN_CHINH_CAP_GHEP_TOI_UU_DAT_MAX_LUC_LUONG_KHAC_NHAU;
uses crt;
const inp = 'capghep.inp'; out = 'capghep.out';
maxmn = 100;
type index = 1..maxmn;
var f :text;
c:array[index,index] of integer;
y:array[index] of integer;
fx,fy:array[index] of longint;
truoc:array[1..2*maxmn] of integer;
m,n,u,v:byte;
procedure nhap;
var i,j :byte;
begin
assign(f,inp); reset(f);
readln(f,m,n);
for i:= 1 to m do
begin
for j:= 1 to n doread(f,c[i,j]);
readln(F);
end;
close(f);
end;
procedure khoitao;
var i,j :byte;
```

```
begin
for i := 1 to n do
begin
y[i] := 0;
fy[i] := 0;
end;
for i:= 1 to m do
begin
fx[i]:= c[i,1];
for j:= 2 to n do if c[i,j] > fx[i] then fx[i]:= c[i,j];
end;
end;
procedure loang(u :byte);
var i,x,dq,cq :integer;
q:array[1..2*maxmn] of integer;
begin
for i:= 1 to m+n do truoc[i]:= 0;
truoc[u]:= u;
dq:= 1; cq:= 1; q[cq]:= u;
while dq <= cq do
begin
x:= q[dq]; inc(dq);
if x > m then begin inc(cq); q[cq]:= y[x-m]; truoc[y[x-m]]:= x; end
else
begin
for i:= m+1 to m+n do if truoc[i] = 0 then
if fx[x] + fy[i-m] = c[x,i-m] then
begin
inc(cq); q[cq]:= i; truoc[i]:= x;
if y[i-m] = 0 then begin v:= i; exit; end;
end;
end;
end;
end;
procedure suanhan;
var d :longint;
i,j:byte;
begin
d:= maxlongint;
for i:= 1 to m do if truoc[i] > 0 then
for j:= 1 to n do if truoc[m+j] = 0 then
```

```
if fx[i] + fy[j] - c[i,j] < d then d:= fx[i]+fy[j] - c[i,j];
for i:= 1 to m do if truoc[i] > 0 then dec(fx[i],d);
for j:= 1 to n do if truoc[m+j] > 0 then inc(fy[j],d);
end;
procedure tangcapghep;
begin
while v <> u do
begin
y[v-m]:= truoc[v];
v:= truoc[truoc[v]];
end;
end;
procedure ghep;
begin
for u:= 1 to m do
begin
v:= 0;
loang(u);
while v = 0 do begin suanhan;loang(u); end;
tangcapghep;
end;
end;
procedure print;
var tong :longint;
i:byte;
x:array[index] of integer;
begin
tong := 0;
for i:= 1 to n do if y[i] <> 0 then { y[v] = 0 co nghia la dinh vkhong duoc ghep }
begin x[y[i]]:= i; inc(tong,c[y[i],i]); end;
assign(f,out); rewrite(f);
writeln(f,tong);
for i:= 1 to m do writeln(f,i,' ',x[i]);
close(f);
end;
Begin
clrscr;
nhap;
khoitao;
ghep;
print;
```

End.
Tôi xin gợi ý cho các bạn với trường hợp m >n thì ta đưa về bài toán trên bằng cách
+ Ghi nhận lại việc m>n. Nếu không thì giải quyết bình thường.
+ Đổi mảng C sao cho Ci,j mới = Cj,i cũ.
+ Lúc in kết quả ta không write(f,x[i],y[i]) nữa mà viết ngược lại: write(f,y[i], x[i]);
Tôi chưa cải tiến cho chương trình đúng với cả m > n (hoàn toàn được) mong bạn suy nghĩ thêm.
Có rất nhiều bài toán đưa về bài toán phải ghép cặp giữa hai tập đỉnh với số đỉnh không bằng nhau, lấy ví dụ đơn giản là ta sửa đề bài của các bài toán cặp ghép đi một chút ta sẽ gặp bài toán khác, và ta cũng chẳng phải suy nghĩ nhiều, chỉ cần sửa chương trình cũ đi một chút thế là xong. Hi vọng bạn tích luỹ thêm được một chút gì đó vào cái kho ″các thuật toán kinh điển″ của mình. Chào tạm biệt và chúc bạn thành công!

36 .**Lịch sử ra đời số PI**

Tác giả: Nguyễn Lê Anh

Tính 2000 chữ số sau dấu phảy của số Π

{Tính được 2*n chữ số Pi. Có thể cho tăng n=500}

```
uses crt;

const n=100;

cs=10000;

type sl=array[0..n+2] of longint;

var a,b,c,x,y,z:sl;

Procedure cong(a,b:sl;var c:sl);

var i:longint;

d:longint;

Begin

d:=0;
```

```
for i:=n downto 0 do
begin
d:=d+a[i]+b[i];
c[i]:=d mod cs;d:=d div cs;
end;
End;
Procedure nhan(a,b:sl;var c:sl);
var i,j:longint;
d,e:comp;
Begin
d:=0;
for i:=n+1 downto 0 do
begin
for j:=0 to i do
if (j<=n) and (i-j<=n) then d:=d+a[i-j]*b[j];
e:=trunc(d/cs);
c[i]:=round(d-e*cs);d:=e;
end;
End;
Procedure nhan2(var a:sl);
var i:longint;
```

```
Begin
a[n+1]:=0;
for i:=n downto 0 do
begin
a[i]:=a[i+1] div cs+a[i]*2;
a[i+1]:=a[i+1] mod cs;
end;
End;
Procedure Dao(a:sl;var b:sl);
var i,j,k,eps:longint;
h,l,r:longint;
c,v:array[0..n+2] of longint;
Begin
fillchar(v,sizeof(v),0);
for i:=0 to n do
begin
H:=cs;L:=0;c:=v;
Repeat
R:=(H+L) div 2;
for k:=n+2 downto 0 do
if k>=i then v[k]:=c[k]+R*a[k-i] else v[k]:=c[k];
```

```
for k:=n+2 downto 1 do
begin
v[k-1]:=v[k-1]+v[k] div cs;
v[k]:=v[k] mod cs;
end;
eps:=0;
for k:=n+2 downto 1 do if v[k]>0 then eps:=1;
if H-L<=1 then begin b[i]:=R;break;end;
if v[0]>1 then eps:=1;
if v[0]<1 then eps:=-1;
Case eps of
1: H:=R;
-1: L:=R;
0: begin b[i]:=R; break; end;
End;
Until 0=1;
end;
End;
Procedure Can(a:sl;var b:sl);
var i,j,k,eps:longint;
h,l,r:longint;
```

```
c,v:array[0..n+2] of longint;
Begin
fillchar(v,sizeof(v),0);
for i:=0 to n do
begin
H:=cs;L:=0;c:=v;
Repeat
b[i]:=(H+L) div 2;
for k:=n+2 downto 0 do
if k=2*i then v[k]:=b[i]*b[i]
else if (k<2*i)and(k>=i)then v[k]:=c[k]+2*b[i]*b[k-i]
else v[k]:=c[k];
for k:=n+2 downto 1 do
begin
v[k-1]:=v[k-1]+v[k] div cs;
v[k]:=v[k] mod cs;
end;
eps:=0;
for k:=n+2 downto 0 do
if v[k]>a[k]then eps:=1
else if v[k]
```

```
if H-L<=1 then break;
Case eps of
1: H:=b[i];
-1: L:=b[i];
0: break;
End;
Until 0=1;
end;
End;
var i,j:longint;
t:string;
BEGIN
for i:=0 to n+2 do a[i]:=0;a[0]:=4;
for i:=0 to n+2 do b[i]:=0;b[0]:=2;
can(b,b);nhan2(b);
Repeat
Dao(a,x);Dao(b,y);cong(x,y,z);Dao(z,a);nhan2(a);
nhan(a,b,c);can(c,b);
gotoxy(1,5); write('3.');
for j:=1 to n do begin str(a[j],t);while length(t)<4 do t:='0'+t; write(t); end;
writeln;writeln; write('3.');
```

for j:=1 to n do begin str(b[j],t);while length(t)<4 do t:='0'+t;write(t); end;

Until 1=0;

END.

Bài tập.

Sử dụng công thức gần đúng

$$\arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \ldots$$

và hằng đẳng thức

$$\frac{\pi}{4} = 4.\arctan(\frac{1}{5}) - \arctan(\frac{1}{239})$$

để tính nhanh giá trị số Π.

```
var s1,s2,x:real;
begin
x:=1/5;
s1:=x-x*x*x/3+x*x*x*x*x/5-x*x*x*x*x*x*x/7+x*x*x*x*x*x*x*x/9;
x:=1/239;
s2:=x-x*x*x/3+x*x*x*x*x/5-x*x*x*x*x*x*x/7+x*x*x*x*x*x*x*x/9;
writeln(16*s1-4*s2,' ',pi);
end.
```

$$x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} = x(1 - x^2(\frac{1}{3} - x^2(\frac{1}{5} - x^2\frac{1}{9})))$$

Giá trị tính được 3.1415926...

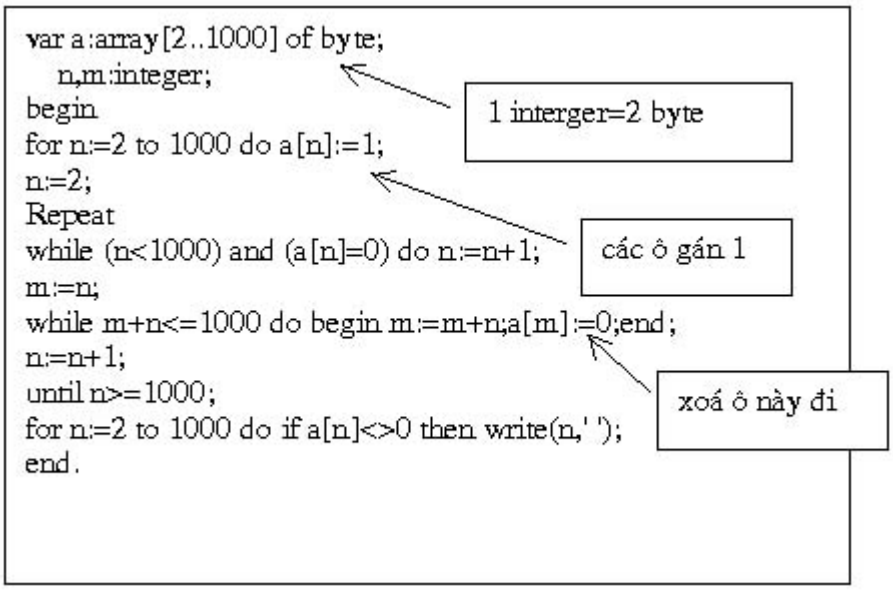
```
var s,x:real;

n:integer;

begin

x:=1/5;

s:=0; for n:=10 downto 1 do s:= 1/(2*n-1) -x*x*s;

s1:=x*s;

x:=1/239;

s:=0; for n:=10 downto 1 do s:=-x*x*s+1/(2*n-1);s2:=x*s;

writeln(16*s1-4*s2,' ',pi);

end.
```

Bài tập. Viết trình tìm các số nguyên tố nhỏ hơn 1000 theo phương pháp Eratosthenes.

Thuật toán:

xếp các ô thành dãy đánh số 2.. 1000. Gán tất cả 1. Bắt đầu từ ô đầu tiên nếu giá trị khác 0 chưa bị xoá - là số nguyên tố. Xoá gán 0 vào đến hết các ô là bội của nó và tìm ô khác 0 tiếp theo.

Trình:



Ngày xuân nói chuyện về số giả nguyên và số nguyên tố Mecxen

Một số tự nhiên lớn hơn 1 được gọi là số nguyên tố nếu nó không chia hết cho bất kỳ số tự nhiên nào ngoài 1 và chính bản thân số đó. Những số có dạng 2n - 1 được mang tên nhà toán học Mecxen là người đầu tiên nghiên cứu chúng. Nếu 2n - 1 là số nguyên tố thì chúng được gọi là số nguyên tố Mecxen. Trong thời gian suốt hai thế kỷ, các nhà toán học "ngờ" rằng số m = 267 - 1 là số nguyên tố Mecxen. Vì số này quá lớn nên mọi người rất ngại khi phải nói đến nó. Người đầu tiên cho biết m không phải là nguyên tố là nhà toán học Mỹ Coun. Tại phiên họp tháng 10 năm 1903 của Hội Toán học Mỹ, Coun lặng lẽ bước đến tấm bảng đen đặt trên đại lễ đường. Ông lấy 67 số 2 nhân với nhau, sau đó trừ đi 1. Vẫn không nói một lời nào, ông bước sang phần bảng bên cạnh làm phép tính nhân hai số: 193707721 x 761838257287. Kết quả này đúng bằng kết quả của 267 - 1 !... Lần đầu tiên trong lịch sử của Hội Toán học Mỹ người ta hoan hô lâu đến như vậy. Coun, vẫn không hề nói một lời nào, thong thả trở về chỗ ngồi. Biết Coun là người ít nói nên không ai hỏi ông câu gì.

Sau đó vài năm, người bạn thân của Coun hỏi rằng ông phải mất bao nhiêu thời gian cho kết quả đó, Coun trả lời: "Tất cả các ngày chủ nhật trong vòng ba năm".

Mất 144 ngày để phân tích số 267 - 1 ra thừa số nguyên tố! Phải là một người kiên nhẫn phi thường và tính toán cực nhanh thì mới làm nổi công việc chán ngắt đó.

Chúng ta thử lập trình để khảo sát các số Mecxen m = 2n - 1 với n = 2..59. Với n = 2 ta có m = 3 là số nguyên tố. Với n = 59 ta có m = 576460752303423487. Số này có 18 chữ số do đó nó vượt quá giới hạn của các số nguyên có thể biểu diễn trong môi trường Turbo Pascal của bạn. Chúng ta thử tìm cách khắc phục bằng cách sử dụng kiểu số thực mở rộng extended. Đó là kiểu số thực có đến 19-20 chữ số, sử dụng 10 byte cho biểu diễn trong, do đó có thể biểu diễn các số tự nhiên trong khoảng 0.. 576460752303423488 = 259.

Số giả nguyên

Do muốn sử dụng kiểu extended để biểu diễn các số tự nhiên nên chúng ta phải tự viết một số thao tác trên kiểu số giả nguyên này. Trước hết bạn cần đặt chế độ dịch {$N+} để báo cho chương trình dịch xử lý các số lớn theo chế độ đồng xử lý của các bộ xử lý 8087/80287. Sau đó bạn khai báo kiểu giả nguyên như sau:

Type GiaNguyen = extended;

Tiếp đến bạn triển khai hàm Nguyen: lấy phần nguyên của số kiểu extended, thí dụ:

Nguyen(12345.79) = 12345.

Để có hàm này bạn lưu ý đến hai thủ tục chuyển đổi số sang xâu ký tự có trong Turbo Pascal là:

str(e:n:t,s) và val(s,e,c)

Thủ tục str(e:n:t,s) chuyển số thực e sang dạng xâu, trong đó tham biến phụ trợ n cho biết tổng số chữ số cần lấy, t cho biết số chữ số cần lấy ở phần thập phân. Với thí dụ trên nếu e = 12345.79 thì sau khi gọi:

str(e:20:0,s) ta sẽ thu được s = "12345".

Thủ tục val(s,e,c) đối xứng với thủ tục str(e,s), tức là val chuyển số dạng xâu ký tự s thành số (kiểu nguyên hoặc thực) e. Tham biến c dùng để bắt lỗi: nếu c=0 tức là phép chuyển thực hiện tốt, c > 0 là có lỗi. Tiếp tục với thí dụ trên, nếu s = "12345", thì sau khi gọi val(s,e,c) ta sẽ thu được e=12345. Kết hợp hai thủ tục này ta sẽ lấy được phần nguyên của số giả nguyên e như trong hàm dưới đây:

function Nguyen(e: GiaNguyen): GiaNguyen;

var s: string[30];

```
c: integer;

begin

str(e:20:0,s);

val(s,e,c);

Nguyen:=e;

end;
```

Đến đây bạn cần viết thủ tục DivMod(a,b,t,d) tìm thương t và dư d khi chia số giả nguyên a cho số giả nguyên b như sau:

```
procedure DivMod(a,b:GiaNguyen;var t,d:GiaNguyen);

begin

t := Nguyen(a/b);

d := a-b*t;

end;
```

Tiếp theo chúng ta viết hàm kiểm tra số giả nguyên e có phải là số nguyên tố hay không. Để tăng tốc cho chương trình chúng ta coi e là biến tổng thể. Số e là nguyên tố khi và chỉ khi e không chia hết cho các số không vượt quá căn bậc hai c của e. Chỉ có duy nhất một số nguyên tố chẵn là 2, do đó để kiểm tra tính chia hết của số lẻ e cho các số trong khoảng 1..c ta chỉ cần xét các số lẻ từ 3 trở đi.

```
function ELaNguyenTo: Boolean;

var d,r,t,du: GiaNguyen;

begin

ELaNguyenTo:=false;

r:=Nguyen(sqrt(e));

d := 3.0;
```

```
while d <= r do

begin

DivMod(e,d,t,du);

if du = 0.0 then exit;

d := d+2.0;

end;

ELaNguyenTo:=true;

end;
```

Chú ý: Vì kiểu giả nguyên về bản chất là số thực nên bạn không thể dùng biến giả nguyên làm biến điều khiển cho vòng lặp for.

Chương trình của chúng ta sẽ gồm hai thủ tục lớn. Thủ tục thứ nhất khảo sát các số Mecxen m = 2i - 1, với i = 2..31. Khi đó có thể dùng biến longint để biểu diễn m. Với các số Mecxen ứng với i = 32..59 chúng ta dùng biến giả nguyên. Kết quả khảo sát sẽ được ghi vào tệp văn bản có tên MECXEN.DAT. Số nào là nguyên tố sẽ được đánh dấu *.

Chú ý: nếu đã biết m=2i thì ta tính được 2i+1 =m+m.

```
(* MECXEN.PAS *)

(* xac dinh cac so nguyen to Mecxen dang 2^n - 1 *)

{$N+}

uses crt;

const fn = 'MECXEN.DAT';

se = '2147483647';

type GiaNguyen = extended;

var p,q: longint;
```

```
n: longint;
f: text;
e: GiaNguyen;
procedure Init;
begin
assign(f,fn);
rewrite(f);
end;
procedure Finit;
begin
close(f);
end;
function PLaNguyenTo: Boolean;
var d,r: longint;
begin
PLaNguyenTo:=false;
r:=round(sqrt(p));
d := 3;
while d <= r do
begin
if p mod d = 0 then exit;
```

```
d := d+2;
end;
PLaNguyenTo:=true;
end;
procedure SmallMecxen;
var i: byte;
begin
writeln(f,'Cac so Mecxen nho:');
p := 1; n:=0;
p := (p shl 1) or 1;
inc(n); i:=2;
writeln(f,i:3,' ',p,' *');
for i := 3 to 31 do
begin
p := (p shl 1) or 1;
write(f,i:3,' ',p);
if od(i) then
if PLaNguyenTo then
begin
inc(n);
writeln(f,' *');
```

```
end
else writeln(f)
else writeln(f);
end;
writeln(f,'Tong cong: ',n,' so nguyen to Mecxen nhó);
end;
function Nguyen(e: GiaNguyen): GiaNguyen;
var s: string[30];
c: integer;
begin
str(e:20:0,s);
val(s,e,c);
Nguyen:=e;
end;
procedure DivMod(a,b:GiaNguyen;var t,d:GiaNguyen);
begin
t := Nguyen(a/b);
d := a-b*t;
end;
function ELaNguyenTo: Boolean;
var d,r,t,du: GiaNguyen;
```

```
begin
ELaNguyenTo:=false;
r:=Nguyen(sqrt(e));
d := 3.0;
while d <= r do
begin
DivMod(e,d,t,du);
if du = 0.0 then exit;
d := d+2;
end;
ELaNguyenTo:=true;
end;
procedure BigMecxen;
var i: byte;
c: integer;
m: GiaNguyen;
begin
writeln(f,'Cac so Mecxen tren ',se,':');
m := 2147483648.0;
n:=0;
for i := 32 to 59 do
```

```
begin
m := m + m;
e := m-1;
write(f,i:3,' ',e:20:0);
if ođ(i) then
if ELaNguyenTo then
begin
inc(n);
writeln(f,' *');
end
else writeln(f)
else writeln(f);
end;
writeln(f,'Tong cong: ',n,' so nguyen to Mecxen tren ',se);
end;
BEGIN
Clrscr;
Init;
SmallMecXen;
BigMecxen;
Finit;
```

END.

Chạy chương trình trên bạn sẽ phát hiện ra rằng trong khoảng 1.. 231 - 1 chỉ có 8 số nguyên tố Mecxen, còn trong khoảng 232.. 259 không có số Mecxen nào.

Bài tập: Số giả nguyên dương extended chỉ có thể đảm bảo 18 chữ số có nghĩa. Bạn thử nghĩ cách biểu diễn số m = 267 - 1 trong hệ thập phân xem sao?.

37 .Thuật toán chia kẹo và ứng dụng giải lớp bài toán chia nhóm

Tác giả: Lã Thành Công

Xét một bài toán chia nhóm tổng quát như sau:

Cho n số a1, a2,..., an (n ' NI*) và m số b1, b2, ..., bm (m < n,m ' NI*).

Yêu cầu: Hãy chia n số a1, a2, ..., an thành m nhómsao cho tổng các phần tử của nhóm i bằng bi (i = 1,m).

Trước khi đưa ra phương pháp giải bài toán trên ta xét bài toán đơn giản nhưng thú vị đó là bài "chia kẹo":

Có n gói kẹo mỗi gói cóai cái kẹo (i= 1,n). Hãy tìm cách chia n gói kẹo thành 2 nhóm sao cho chênh lệch giữa 2 nhóm là ít nhất.

Bài toán có thể giải theo thuật toán sau:

Ta đi xây dựng các tổng có thể có được từ gói kẹo. Ta thấy nếu tổng nào gồm (n div 2) nhất thì đó là giá trị của nhóm 1. Phần còn lại sẽ thuộc vào nhóm 2. Để xây dựng được ta làm như sau:

Ta dùng 3 mảng làm các công việc sau:

Mảng A dùng để ghi lại các giá trị của tổng

Mảng B dùng để ghi lại các phần tử cuối cùng cho vào để đạt giá trị ở A.

Mảng C dùng để ghi lại các vị trí của tổng mà không có phần tử ở B. Mảng này dùng để lần lại các phần tử có để đạt giá trị ở A.

Thực hiện như sau:

```
A[0]:= 0; Max := 0; B [0]:= 0; C [0]:= 0;

For i:= 1 to n do

For j:= 0 to max do

Begin

D:= True; m:= max;

For k:= 0 to max do if (A[j] +ai) = A[k] then D:= false;

If D then

Begin

inc (m);

A[m] := A[j] + ai;

B[m] := i;

C[m] := j;

end;
```

Thuật giải trên đã xây dựng được các tổng có thể có từ n gói kẹo. Khi chọn được A[i] gần với (n div 2) nhất thì ta thực hiện thủ tục lần ngược như sau:

```
Fillchar (Logic, size of (logic), false);

While A[i] > 0 do

Begin

Logic [B[i]] := true; (cho phần tử B[i] vào nghiệm)

i:= c[i];

End;
```

Khi đó các phần tửmang giá trị Logic là true sẽ tạo thành một nhóm có tổng A[i] ban đầu.

Xét tiếp một bài toán ở mức độ cao hơn như sau:

Cho n số a1, a2,..., an (n ' NI*).

Tìm cách chia số trên thành m nhóm sao cho mỗi nhóm đều có tổng bằng nhau và m tìm được là lớn nhất.

Cách giải : Bài toán trên ta có thể duyệt mọi m là ước số của T = Σ ai. Mà ở đó m chạy từ T về 1.

Với mỗi m có được ta có tổng các nhóm sẽ là D = T/m. Với mỗi dự tuyển (m,D):

Khi đó ta sẽ dùng thuật toán trên để tìm m nhóm có tổng là D.

+ Lưu ý một điều đó là:

Ta phải sắp xếp ai (i=1,n) theo thứ tự giảm dần để khi duyệt ta được kết quả tối ưu.

Ta sẽ dùng thuật toán trên tìm lần lượt m nhóm tổng là D. Mỗi lần tìm được một nhóm ta sẽ nhấc các phần tử thuộc nhóm đó ra và thực hiện lại với các phần tử còn lại.

Khi tìm được một dự tuyển (m,D) thoả mãn thìđó sẽ là tối ưu của lời giải.

Nếu ta không sắp xếp ai (i=1, n) theo thứ tự giảm dần thì ta sẽ thấy trường hợp sau bị vướng mắc:

VD: dãy n số là: n = 7

8 2 1 3 9 10 11

Ta thấy mmax không sắp xếp thì lần đầu máy sẽ cho (8,2,1) vào một nhóm và như vậy không làm tiếp được.

Qua 2 ví dụ ta đã phần nào hiểu được ưu thế của thuật toán trên trong một lớp các bài toán chia nhóm, một lớp bài toán mà nếu không có thuật toán và nắm bắt bản chất ta sẽ rất khó giải.

Tuy nhiên có thể còn nhiều thuật toán khác hay hơn mà tôi chưa biết. Tôi hy vọng các bạn có thểtìm ra nhiều nhiều thuật toán hay hơn.

Để luyện tập, tôi xin đưa ra một bài toán sau (Thi chọn đội tuyển quốc gia năm 1995):

Một quầy trả tiền có n loại tiền với mệnh giá (giá trị tiền ghi trên tờ tiền) làA[1], A[2],... A[n] (các A[i] lànguyên dương và khác nhau từng đôi). Giả thiết loại tiền mệnh giá A[i] có B[i] tờ (i := 1, n). Có M khách (đánh số hiệu tử 1 -> m) cần lấy tiền. Số tiền khách i cần lấy là K[j], K[j] nguyên dương, 1 <= j <= một. Quy định rằng với mỗi khách hoặc quầy từ chối trả tiền hoặcphải trả đúng số tiền mà khách cần lấy.

Dữ liệu vào file INP.TXT dòng đầu ghi giá trị N (N <= 10), dòng tiếp theo ghi các giá trị A[1], A[2],..., A[n] dòng tiếp theo ghi các giá trị B[1], B[2],..., B[n]. Sau đó là dòng ghi giá trị M và dòng ghi các giá trị K[1], K[2], K[3]..., K[M]. Tất cả đều nguyên dương.

Yêu cầu:

Tìm cách trả sao cho trả được nhiều khách nhất. Thông báo ra file OU1.TXT, trong dòng đầu ghi số khách được trả tiền, trong các dòng tiếp theo mỗi dòng ghi thông tin về một khách, số tiền phải trả và dãy các số X[1], X[2],..., X[n] trong đó X[i] là số tờ của loại tiền mệnh giá A[i] (1 <=i <= n) được trả cho khách.

Tìm cách trả sao cho trả được nhiều tiền nhất. Thông báo ra file OU2.TXT số tiền dòng đầu tiếp theo là thông tin về các khách được trả tiền theo quy ước giống câu 1.

Có thể lấy chương trình các ví dụ và bài tập theo địa chỉ của tôi:

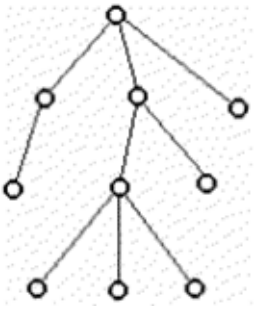
Lã Thành Công
Lớp 12A2 - chuyên tintrường PTTH chuyên Vĩnh Phúc

38 .**Mô hình cây và các bài toán quản lý**

Tác giả: Đỗ Quang Tiến

Trước tiên ta nhắc lại sơ lược những khái niệm về mô hình cây trong lý thuyết đồ thị. Cây là một đồ thị liên thông không có chu trình, nhiều cây hợp lại thành rừng. Nếu giả sử G là một cây có n nút thì G có những tính chất sau:
1. G có n-1 cạnh, G không có chu trình và liên thông.
2. Mỗi cạnh của G đều là một cầu trong đó cầu là cạnh mà nếu xoá nó đi đồ thị

không còn liên thông.
3. Hai nút bất kỳ trong G được nối với nhau bởi đúng một đường đi đơn là đường đi mà mỗi đỉnh trên đường đi được đi qua không quá một lần.
4. Nếu thêm vào G một cạnh giữa hai nút bất kỳ thì ta được đúng một chu trình.
Như đã biết, mô hình cây là một mô hình khá cơ bản trong lý thuyết đồ thị mà ứng dụng của nó rất rộng rãi, trong đó điển hình là các ứng dụng của bài toán cây khung (cây bao trùm), cây khung tối tiểu (cây bao trùm có tổng trọng số nhỏ nhất) quen thuộc.
Bên cạnh những ứng dụng đó, mô hình cây còn được sử dụng để giải quyết các bài toán quản lý có đặc trưng như sau:
"Chúng ta phải quản lý một tập các đối tượng là các xâu kí tự hoặc các chuỗi số được sắp xếp theo các ưu tiên nào đó. Số thứ tự của đối tượng sau khi sắp xếp gọi là mã của đối tượng. Như vậy giữa đối tượng và mã của nó có quan hệ tương ứng 1-1 và để có thể sử dụng đối tượng trong những công việc cụ thể nào đó thì vấn đề đặt ra là với một đối tượng phải biết được mã của nó, ngược lại, khi có một mã xác định phải tìm được đối tượng tương ứng."
Dường như những bài toán có đặc trưng như vậy thì cách giải quyết sẽ không khó nếu số lượng các đối tượng cần quản lý không lớn. Tuy nhiên vấn đề sẽ phức tạp lên nhiều nếu tập các đối tượng cần quản lý quá lớn (thường tuân theo hàm giai thừa hay hàm luỹ thừa) không thể lưu trữ đầy đủ dưới bất kỳ một hình thức nào. Vì vậy cần tìm một phương pháp hiệu quả để giải quyết và có một cách thức được đề xuất giải những bài toán có đặc trưng trên là sử dụng mô hình cây và phương pháp duyệt cây.



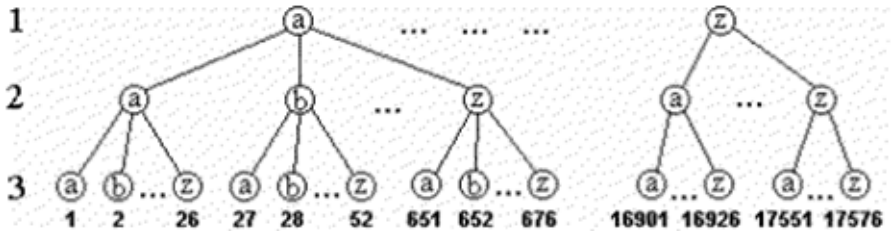
Hình 1. Ví dụ về cây

Mô hình cây mà ta xét ở đây có những đặc điểm sau:
- Cây được chia làm N lớp đánh số từ 1 đến N theo chiều từ trên xuống.
- Những nút ở lớp thứ nhất gọi là gốc của cây.
- Những nút mà chỉ có cạnh nối với nút ở lớp trên gọi là lá.
Từ đó ta định nghĩa phương pháp duyệt cây là phương pháp xuất phát từ gốc của cây đi qua các nút trên cây mỗi nút một lần cho đến lá.

Để nắm được và hiểu cách thức này, cách tốt nhất là ta phải phân tích trên những bài toán cụ thể. Trước tiên xét một bài toán khá quen thuộc, bài toán quản lý xâu, có đề cụ thể như sau:
Bài 1. Với mọi xâu (chỉ gồm các chữ cái in thường) có độ dài N (N£10), người ta sắp xếp theo qui tắc ưu tiên thứ tự bảng chữ cái, tức là sắp xếp theo thứ tự từ điển. Mỗi xâu có một số đặc trưng là số thứ tự của xâu sau khi sắp xếp, số này gọi là mã của xâu. Sau đây là một số xâu và mã của nó:

Với N = 2:		
	aa	1
	ab	2
	...	
	zy	675
	zz	676
Với N=3		
	abb	28
	azb	652
	zza	17551

Yêu cầu đặt ra dành cho bạn như sau: nhập giá trị N, nếu cho một xâu, bạn hãy cho biết mã của xâu, ngược lại nếu cho một mã, bạn hãy cho biết xâu tương ứng với mã nhận được.
Ta thấy tập đối tượng cần quản lý của ta là tập các xâu độ dài N và không khó khăn lắm khi chứng minh rằng với N xác định thì có 26N xâu độ dài N tức là 26N đối tượng cần quản lý, đây quả là một con số không hề nhỏ. Vậy giải quyết bài toán này như thế nào?
Sau khi đánh giá những đặc điểm của đối tượng (là các xâu), ta xây dựng một mô hình cây ứng với giá trị cụ thể N=3 như sau:



Hình 2. Mô hình cây của bài quản lý xâu

Mỗi nút trên cây ứng với một chữ cái, nút ở lớp cuối của cây được gọi là lá. Số nút con của một nút luôn là 26 và các nút con này được xếp đặt tăng dần từ a đến z. Nếu xuất phát từ gốc (từ một nút ở lớp trên cùng), đi theo các nhánh của cây theo chiều từ trên xuống dưới cho đến lá và ghi nhận các chữ cái tại các nút đã đi qua thì ta sẽ được một xâu. Dễ thấy, mỗi xâu tạo được theo cách trên là một xâu trong tập chúng

ta cần quản lý. Theo cách thực hiện như vậy, mỗi lá ứng với một xâu, tổng số lá chính là tổng số xâu. Đặc biệt số ghi dưới lá là số thứ tự của xâu (các số này đánh tăng dần từ trái sang).

Bây giờ ta đề xuất phương pháp tìm kiếm (duyệt) trên cây: từ một xâu cho trước, xuất phát từ gốc và duyệt qua các nút của cây theo chiều từ trên xuống sao cho thứ tự các nút này ứng với thứ tự các chữ cái của xâu, cuối cùng kết thúc ở một lá, ghi nhận số tương ứng ở lá này, đó chính là số thứ tự của xâu.
Ngược lại, nếu biết số thứ tự của xâu thì ta biết được vị trí của lá tương ứng, sau đó lần ngược từ lá cho đến gốc (luôn đi từ dưới lên), ghi nhận chữ cái tại các nút đã đi qua, cho đến gốc thì ta đã được xâu tương ứng với số thứ tự đã cho.

Trên đây là nội dung cơ bản của phương pháp duyệt cây mà chúng ta sử dụng để giải quyết các bài toán quản lý có đặc trưng đã nêu ở trên. Rõ ràng không khó để chứng minh tính đúng đắn của phương pháp này, vấn đề ở chỗ thể hiện nó. Vì vậy tiếp theo đây sẽ mô tả cách thức chuyển phương pháp duyệt cây trên thành chương trình tính toán cụ thể sao cho thật đơn giản và hiệu quả.

Bắt đầu bằng việc tại mỗi nút của cây ta đánh một số là số lá mà từ nút này có thể đi tới theo chiều từ trên xuống dưới, số này gọi là số lá con của nút. Như đã biết, mỗi xâu ứng với một nhánh của mô hình cây, thế thì nếu có thể đếm được số lá bên trái nhánh là có thể biết được số thứ tự của xâu và cách làm của ta là đếm số lá này. Có một đặc điểm của mô hình cây cần đặc biệt chú ý là tất cả các nút trên cây trừ ở lớp cuối đều có cùng số nút con là 26. Ngoài ra, còn có thể thấy số lá con của một nút bất kỳ tại lớp thứ i (i=1..N-1) là 26N-i. Vì vậy ta lập mảng một chiều A, với Ai (i=1..N-1) cho biết số lá con của một nút tại lớp thứ i. Và mảng A được tính như sau:

```
A[N]:=0;
A[N-1]:=26;
For i:=N-2 downto 1 do A[i]:=A[i+1]*26;
```

Khi có một xâu St độ dài N, cần phải đưa ra mã số Stt của xâu. Thế thì đầu tiên khởi tạo Stt=0. Với ký tự đầu tiên của xâu ta tìm được một gốc tương ứng với ký tự này, đem số lá con của tất cả các gốc khác xếp bên trái gốc tìm được cộng vào Stt, sau đó chuyển việc xử lý tới ký tự tiếp theo của xâu. Với ký tự thứ i của xâu đã cho (i=1..N) ta lại tìm được một nút trên cây ứng với ký tự này (nút này thoả mãn theo cách thức duyệt cây đã trình bầy ở trên). Đem số lá con của tất cả các nút khác - xếp bên trái, có cùng một nút cha với nút vừa xác định - cộng vào Stt. Sau khi xét qua i=N, ta cộng 1 thêm vào Stt và đây chính là số thứ tự của xâu đã cho.

Dựa vào đặc điểm đã nhận xét và đã thể hiện trong mảng A, các câu lệnh sau sẽ tìm ra Stt từ xâu St cho trước:

```
Stt:=1;
For i:=1 to length(St) do
For j:=1 to (ord(St[i])-ord('a')-1) do Stt:=Stt+A[i];
```

Như đã thấy, chỉ cần hai vòng for đơn giản là có thể biết số thứ tự của một xâu. Bây giờ sau khi thực hiện bài toán xuôi - tức là cho một xâu tìm số thứ tự - ta sẽ giải quyết bài toán ngược - nếu như từ một mã số Stt cho trước tìm ra xâu St tương ứng. Trước tiên là tìm ký tự đầu tiên của xâu bằng việc chọn các gốc liên tiếp từ gốc trái nhất sao cho tổng lá con của các gốc này là lớn nhất mà không vượt quá Stt, gán Stt bằng Stt trừ đi tổng này, ký tự đầu tiên của xâu cần tìm là ký tự ứng với gốc tiếp theo ngay bên phải các gốc đã chọn. Để tìm ký tự thứ i (i=2..N) ta tìm tập các nút liên tiếp từ trái trong lớp thứ i sao cho tổng lá con của các nút này lớn nhất và không vượt quá Stt, gán Stt bằng Stt trừ đi tổng vừa tìm được rồi ghi nhận ký tự ứng với nút tiếp theo ngay bên phải tập các nút vừa xác định làm ký tự thứ i của xâu. Tiếp tục sử dụng đặc điểm thể hiện trong mảng A ta có đoạn chương trình sau tìm ra xâu St:

```
St:='';
For i:=1 to N do
Begin
Temp:=0;
j:=0;
Repeat
j:=j+1;
Temp:=Temp+A[i];
Until Temp>=Stt;
St:=St+char(j+ord('a'));
Stt:=Stt-Temp-A[i];
End;
```

Hãy xét kỹ đoạn chương trình trên thì quả là từ một số thứ tự (mã của xâu) Stt sẽ cho ra xâu cần tìm tương ứng chứa trong St.

Như vậy phương pháp duyệt cây của chúng ta rất đơn giản khi mô tả cũng như khi thể hiện bằng chương trình và lại có hiệu quả khá rõ rệt, khi này vấn đề kích thước của tập đối tượng cần quản lý không còn là một trở ngại quá quan trọng nữa.

Với bài toán cụ thể vừa nêu ta hoàn toàn có thể mở rộng giới hạn của N, N£52 chẳng hạn, ngoài ra các xâu không chỉ chứa chữ cái in thường mà còn có thể chứa

những ký tự bất kì. Khi đó chương trình chỉ thay đổi một chút: các phép tính phải thực hiện trên các số rất lớn và việc xử lý tính toán trên số lớn không khó nếu ta lưu trữ các số trên xâu và thực hiện cộng, trừ trên xâu.

Và Bài toán quản lý xâu trên sẽ được mở thành bài toán sau với yêu cầu cao hơn:

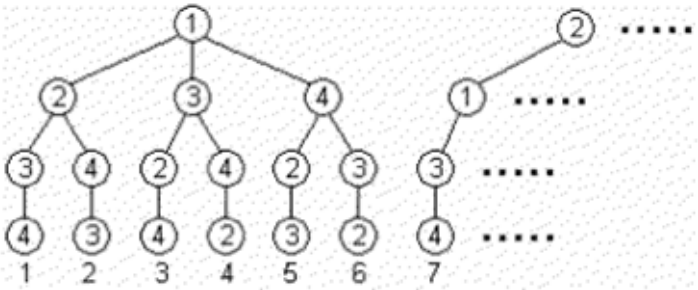
Bài 1a. Ta vẫn quản lý các xâu nhưng lần này xét với tất cả các xâu chứa chữ cái cả in thường lẫn in hoa sao cho một ký tự bất kỳ trong xâu (trừ ký tự đầu tiên) luôn có mã ASCII lớn hơn mã ASCII của kí tự đứng trước nó. Các xâu được xét có độ dài bất kỳ nhưng không vượt quá 52, tập các xâu này cũng được sắp xếp ưu tiên thứ tự từ điển (không ưu tiên độ dài), mã của xâu chính là số thứ tự của xâu sau khi sắp xếp.

Yêu cầu: với một mã hãy cho biết xâu tương ứng và với một xâu hãy cho biết mã của nó.

Ta tiếp tục phân tích các ứng dụng của mô hình cây bằng một bài toán quen thuộc khác, bài toán quản lý hoán vị của N số tự nhiên đầu tiên.

Bài 2. Giả sử ta có tất cả hoán vị của N số tự nhiên đầu tiên, N nhập từ bàn phím (tuỳ khả năng tính toán có thể tự giới hạn N £12 hoặc N £50). Tất cả các hoán vị này được sắp xếp theo thứ tự từ điển. Như vậy mỗi hoán vị được ứng với một mã số là số thứ tự của hoán vị trong dãy các hoán vị đã sắp xếp. Yêu cầu đặt ra là: với một mã số hãy đưa ra hoán vị tương ứng và ngược lại, cho một hoán vị hãy đưa ra mã số của hoán vị này.

Phân tích các đặc trưng của bài toán, ta xây dựng lên mô hình cây như hình sau (ứng với giá trị cụ thể N=4):



Hình 3. Mô hình cây của bài quản lý hoán vị

Hãy xét kỹ mô hình cây trên và dễ thấy bài toán có thể giải được bằng phương pháp duyệt cây đã mô tả. Bây giờ ta cùng đánh giá những đặc điểm riêng của mô hình cây

này để thể hiện bằng một chương trình thật hiệu quả:
- Cây được phân thành N lớp theo thứ tự từ trên xuống
- Nút tại lớp thứ i ứng với một số là số thứ i trong một hoán vị (i=1..N)
- Số lá con của một nút bất kì tại lớp thứ i (i=1..N-1) là (N-i)!
- Các nút con của một nút được xếp đặt tăng dần. Và ta thấy tập các nút con của một nút bất kì với tập các nút ở lớp trên mà từ các nút con này có thể lần ngược lên hợp thành một hoán vị, vì vậy có thể xác định được tập nút con của một nút.
Dựa vào các đặc điểm trên ta xây dựng mảng một chiều Giai_Thua, mà Giai_Thua[k] = k!, k=1..N. Giả sử ta có một hoán vị của N số tự nhiên đầu tiên chứa trong bảng một chiều Hv[1..N], cần tìm số thứ tự Stt của hoán vị này. Đoạn chương trình sau cho cách tính Stt:

```
Stt := 0;
for i := 1 to N do
begin
Temp := Hv[i];
for j := 1 to i do
if Hv[j] <= Hv[i] then dec(Temp);
Stt := Stt + Temp*Giai_Thua[N-i];

end;
```

Biến Temp cho biết số nút xếp bên trái và cùng nút cha của nút đang xét. Và khi cho một số thứ tự Stt mà lần ra hoán vị đặt vào mảng Hv thì cũng không khó:

```
procedure Xac_Dinh(k:byte);
var i,j,t:byte;
begin
for i:=N downto 1 do
begin
t:=i-1;
for j:=1 to k-1 do
if Hv[j] < i then dec(t);
if t*Giai_Thua[N-k]<=Stt then
begin
Stt:=Stt-t*Giai_Thua[N-k];
Hv[k]:=i;
exit;
end;
end;
end;
```

```
Procedure Tim_Hoan_Vi;
var k:byte;
begin
for k:=1 to N do Xac_Dinh(k);
end;
```

Có một bài áp dụng quản lý hoán vị là bài Magic trong đề thi Tin học quốc tế năm 1997, đề bài cụ thể đã được TH&NT đăng trong số tháng 9. Dễ thấy rằng trạng thái của một bảng số ứng với một hoán vị của 8 số tự nhiên đầu tiên, như vậy số trạng thái của bảng có tối đa là 8!. Vậy ta cho mỗi trạng thái tương ứng với một số tự nhiên và số tự nhiên này lại tương ứng với một đỉnh của đồ thị thì sẽ được đồ thị có 8! đỉnh. Sau đó thực hiện loang tìm kiếm trên đồ thị 8! đỉnh này cùng với một số kỹ thuật xử lý khác mà đặc biệt là áp dụng việc quản lý các hoán vị của 8 số tự nhiên đầu tiên ta hoàn toàn có thể giải được bài Magic.

Để tiếp tục làm rõ ứng dụng của mô hình cây chúng ta xét thêm một bài toán quản lý khác rất có ý nghĩa trong thực tiễn sản xuất hàng hoá. Bài toán quản lý mã vạch.

Còn nữa

39. **Mô hình cây và các bài toán quản lý**

Tác giả: Đỗ Quang Tiến

(Tiếp theo)

Bài 3: Để quản lý xuất xứ, chất lượng, giá cả của hàng hoá người ta sử dụng hệ thống mã vạch. Một mã vạch là một dãy các vạch đen trắng xen kẽ nhau bắt đầu bởi vạch đen bên trái, mỗi vạch có độ rộng được tính bằng một số nguyên dương.

Tập mã vạch Mv(n,k,m) là một tập hợp chứa mọi mã vạch gồm k vạch trải trên n đơn vị độ rộng và mỗi vạch có độ rộng không quá m. Để đơn giản, người ta biểu diễn mối mã vạch bằng một dãy nhị phân trong đó số 1 biểu thị một đơn vị rộng màu đen, số 0 thể hiện một đơn vị rộng màu trắng.

Với biểu diễn nhị phân như vậy, mỗi tập Mv(n,k,m) sẽ là một tập các xâu có độ dài n chỉ gồm các ký tự 0 và 1. Ta có thể liệt kê tập đó theo thứ tự từ điển với qui ước ký tự 0 đứng trước ký tự 1. Từ đó mỗi mã vạch tương ứng với một số hiệu chính là số thứ tự của xâu nhị phân biểu diễn nó.

Xét tập Mv(7,4,3) gồm 16 mã vạch được biểu diễn bởi 16 xâu nhị phân độ dài 7, sau đây là liệt kê 16 xâu theo thứ tự từ điển, số thứ tự của xâu là số hiệu của mã vạch tương ứng:

01. 1000100	09. 1100100
02. 1000110	10. 1100110
03. 1001000	11. 1101000
04. 1001100	12. 1101100
05. 1001110	13. 1101110
06. 1011000	14. 1110010
07. 101100	15. 1110100
08. 1100010	16. 1110110

Vấn đề đặt ra (khi có ba số n, k, m xác định) là:

a. Với một xâu biểu diễn một mã vạch, hãy chỉ số hiệu của mã vạch đó.

b. Với một số hiệu, hãy chỉ ra xâu biểu diễn mã vạch tương ứng với số hiệu này.

Dữ liệu vào: từ file văn bản MAVACH.OUT:

- Dòng đầu tiên ghi tổng số mã vạch trong tập Mv(n,k,m).

- Với mỗi cặp dòng của file dữ liệu, ghi ra kết quả tương ứng trên hai dòng của file kết quả.

Ví dụ về file kết quả và file dữ liệu:

MAVACH.IN	MAVACH.OUT
7 4 3 1 1001110 2 16	16 1 5 2 1110110

2 1 1 1001100	2 1000100 1 4

Trước tiên ta thiết lập mô hình cây cho bài toán mã vạch.

Nhắc lại, một tập Mv(n,k,m) là tập mã vạch có k vạch trải trên n đơn vị rộng, mỗi vạch không rộng quá m đơn vị.

Dễ dàng nhận ra nếu hàm Mv(n,k,m) cho biết số phần tử của tập mã vạch tương ứng thì:

trong đó Mv(0,0,m)=1, Mv(i,0,m)=0, i=1..n. Vậy, việc tính số lượng mã vạch đã có công thức truy hồi trên.

Bây giờ cho một mã vạch, ta sẽ tìm số hiệu của mã vạch đó. Xét mã vạch dạng (v1,v2,...,vk) với $1 \leq vi \leq m$, i=1..k, vi là độ rộng của vạch thứ i. Ta cần tìm F là số hiệu của mã vạch, khi đó trước tiên khởi tạo F=0 sau đó với mỗi t từ 1 đến k:

- Nếu t lẻ:

- Nếu t chẵn:

Sử dụng công thức trên ta tìm được số hiệu tương ứng của một mã vạch. Bây giờ ngược lại, cho một số hiệu ta sẽ tìm mã vạch tương ứng với số hiệu đó. Gọi F là số hiệu của mã vạch có dạng: (v1,v2,...,vk). Với F xác định ta phải tìm (v1,v2,...,vk). Và sẽ làm như sau mỗi t từ 1 đến k, xét tìm vt:

1. Nếu t lẻ. Ta xách định v' sao cho tổng:

là lớn nhất mà không vượt quá F:

+ Gán F=F-S

+ Nếu S=F thì Vt =v' ngược lại Vt = v'+1

2. Nếu t chẵn. Ta xác định v' sao cho tổng:

là lớn nhất không vượt quá F:

+ Gán F=F-S

+ Nếu S=F thì Vt = v' ngược lại Vt = v'-1

Bạn có thể tham khảo toàn văn chương trình giải bài Mã vạch:

```
{$N+}
Uses crt;
Const
inp='mavach.in';
out='mavach.out';
ktmax=50;
Var
f,g: text;
SoHieu: extended;
mavach: string[ktmax+1];
n,k,m: interger;
CanTren: interger;
V: array[1..ktmax] of interger;
{Ma_Vach[i,j] = Mv(i,j,m)}
Ma_Vach: array[0..ktmax, 0..ktmax] of extended;
Procedure XayDung_Ma_Vach;
var i,j,t: interger;
```

```
s: extended;
begin
fillchar(Ma_Vach, sizeof(Ma_Vach),0);
Ma_Vach[0,0] : = 1;
for i:=1 to k do begin
for j:=1 to n do begin
s:=0;
for t:=1 to CanTren do
if j-t>=0 then
s:=s+Ma_Vach[j-t,i-1];
Ma_Vach[j,i]:=s;
end;
end;
Procedure Tim_So_Hieu;
var i,j,t,s: interger;
begin
MaVach:=MaVach+'2';
fillchar(v,sizeof(v),0);
t:=1;
for i:=1 to n do begin
inc(v[t]);
```

```
if MaVach[i] <> MaVach[i+1]
then inc(t); end;
{-}
SoHieu:=0; s:=0;
for i:=1 to k-1 do begin
if od(i) then begin
for j:=1 to V[i] -1 do
if n-s-j>=0 then
SoHieu:=SoHieuMa_Vach[n-s-j,k-i];
end;
s:=s+V[i]; end;
SoHieu:=SoHieu1;
writeln(g,1);
writeln(g,SoHieu:0:0);
end;
Procedure Tim_Ma_Vach;
var i,j,t,s: interger;
s2, temp: extended;
begin
fillchar(v,sizeof(v),0);
s:=0;
```

```
for i:=1 to k-1 do
begin
s2:=0;
if od(i) then begin
temp:=0; V[i]:=1;
for j:=1 to CanTren do
if n-s-j>=0 then begin
s2:=s2+Ma_Vach[n-s-j,k-i];
if s2<=SoHieu then begin
temp:=2;
if s2=SoHieu then V[i]:=j
else V[i]:=j+1
end
else break; end;
end else
begin
temp:=0; V[i]:=CanTren;
for i:=CanTren downto 0
do if n-s-j>=0 then
begin
s2:=s2+Ma_Vach[n-s-j,k-i];
```

```
if s2<=SoHieu then begin
temp:=s2;
if s2=SoHieu then V[i]:=j
else break;end;end;
s:=s+V[i];
SoHieu:=SoHieu-temp;
end;
{-}
V[k] : =n-s;
writeln(g,2);
for i:=1 to k do
if od(i) then
for j:=1 to V[i] do
write(g,1)
else
for j:=1 to V[i] do
write(g,0); writeln(g);
end;
Procedure Nhap_Xuly_Inkq;
var i: interger;
begin
```

```
assign(f,inp); reset(f);
assign(g,out); rewrite(g);
readln(f,n,k,m);
if n-k+1
CanTren:=n-k+1
else CanTren:=m;
XayDung_Ma_Vach;
while not eof(f) do begin
readln(f,i);
if i=1 then begin
readln(f,MaVach);
Tim_So_Hieu; end
else begin
readln(f,SoHieu);
Tim_Ma_Vach; end;
end;
close(g); close(f);
end;
Begin
clrscr;
Nhap_XuLy_Inkq;
```

End.

Cuối cùng xin đưa ra một bài toán khá hay và cũng khá khó nếu không áp dụng mô hình cây và kèm theo là gợi ý nho nhỏ cho hai bài toán này. Bạn hãy tự phát hiện đặc điểm của mô hình cây dựa vào các điều kiện của đề bài, sau đó tìm ra cách thức cụ thể đẻ giải quyết bằng chương trình.

Bài 4: Quản lý xâu ngoặc

Xét các xâu có độ dài N ($N \leq 200$) có tính chất sau:

- Xâu chỉ chứa các ký tự '(' và ')'

- Số ký tự '(' luôn bằng số ký tự ')'

- Nếu tính từ trái qua phải thì số '(' bao giừo cũng lớn hơn hoặc bằng số ')'.

Tất cả các xâu thoả mãn tính chất nêu trên được sắp xếp theo thứ tự từ điển và được đánh số thứ tự, số thứ tự này gọi là mã của xâu. Từ gía trị N xác định, yêu cầu hãy:

a. Cho biết số lượng các xâu thoả mãn 3 tính chất nêu trên.

b. Cho biết mã của một xâu cho trước.

c. Cho biết xâu ứng với một mã cho trước.

Dữ liệu vào từ file văn bản BRACKET.IN:

- Dòng đầu tiên chứa số nguyên dương N

- Các dòng tiếp theo của file, mỗi dòng hoặc chứa một xâu hoặc chứa một mã số.

Kết quả ghi ra file BRACKET.OUT:

- Dòng đầu tiên ghi số lượng các xâu độ dài N thoả mãn các tính chất của đề bài.

- Tương ứng với mỗi dòng của file dữ liệu hãy tìm kết quả theo yêu cầu và ghi ra trên một dòng của file kết quả.

Gợi ý cho bài Quản lý xâu ngoặc. Nhận xét thấy số ký tự '(' luôn bằng số ký tự ')' nên N chẵn và N/2 nguyên, vì vậy ta lập mô hình cây có N/2 lớp và mỗi nút ở lớp

thứ i trên cây tương ứng với một số, số này là vị trí của dấu '(' thứ i (i=1..N/2) tính từ trái sang.

Tóm lại, bài viết này đề xuất thêm một ứng dụng của lý thuyết đồ thị nói chung và lý thuyết cây nói riêng. Xin nhắc lại, những bài toán có đặc trưng nêu ở phần đầu bài viết nếu bằng cách thiết lập mô hình cây hợp lý thì hoàn toàn có thể giải được bằng một chương trình đơn giản, ngắn gọn, hiệu quả.

40 . **Giải quyết bài toán QHTT bằng bảng tính Excel**

Tác giả: Trịnh Thanh Hải

Bài viết cho mục Toán - Tin dành cho các bạn Sinh Viên:

Quy hoạch tuyến tính (QHTT) là một bộ phận của Tối ưu hoá. Nó có rất nhiều ứng dụng trong thực tiễn. QHTT bắt nguồn từ những nghiên cứu của Viện sỹ Kantorovich L.V (Nga) từ những năm 1937.

Bài toán QHTT tổng quát

Bài toán QHTT tổng quát thường được phát biểu như sau:

$$f(x) = \sum_{j=1}^{n} c_j x_j \quad (1)$$

Tìm véc tơ x=(x1,x2...xn) sao cho hàm:

Trong đó : I là tập con của M và M ={1,2..m}

J là tập con của N và N ={1,2..n}

I= MI

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad (i \in I)$$

đạt giá trị nhỏ nhất với các ràng buộc:

$$x_j \geq 0 \quad (j \in J) \quad (4)$$

Vì mỗi bất phương trình trong ràng buộc bao giờ cũng đưa được về dạng (2) và

f(x*) = max(f(x)) <=> -f(x*) = min(-f(x)).

Hàm f (x) được gọi là hàm mục tiêu; các điều kiện 2,3,4 được gọi các điều kiện ràng buộc.

Ta gọi mỗi véc tơ x=(x1,x2...xn) thoả mãn điều kiện ràng buộc 2,3,4 là một phương án. Phương án tối ưu (nghiệm của bài toán) là phương án làm cho hàm mục tiêu f đạt giá trị nhỏ nhất.

Điều kiện cần và đủ để bài toán QHTT có phương án tối ưu là tập các phương án khác trống và hàm mục tiêu f bị chặn dưới nếu là bài toán cực tiểu hoá và bị chặn trên nếu là bài toán cực đại hóa.

Phương án x=(x1,x2...xn) gọi là không suy biến nếu x có đúng m thành phần dương (m là số phương trình trong hệ ràng buộc)

Thuật toán giải bài toán QHTT tổng quát

Hiện nay có nhiều phương pháp để giải bài toán QHTT, một trong các phương pháp đó là Phương pháp đơn hình của G.Dantzig đề xướng vào năm 1949 như sau:

* Bước 1: Biến đổi đưa hệ thống các điều kiện ràng buộc về dạng giải ra đối với một cơ sở xuất phát nào đó.

* Bước 2: Biểu diễn hàm mục tiêu f qua các ẩn không thuộc cơ sở, gọi hệ số của ẩn xj trong biểu diễn của f là D j

* Bước 3: Xét nếu trong biểu diễn của f mà tất cả các hệ số Dj của các ẩn không thuộc cơ sở đều không dương ($D_j \leq 0$) thì phương án tương ứng là phương án tối ưu. Kết thúc thuật toán. Nếu tồn tại các hệ số biểu diễn D j dương chuyển sang bước 4:

* Bước 4: Xét

+ Trường hợp 1: Nếu có một trong các hệ số dương: giả sử là hệ số của xj là Dj > 0 mà cột hệ số của xj trong biểu diễn hệ ràng buộc (aij) đều âm. Thì min (f) = âm vô cùng , bài toán không có nghiệm.

+ Trường hợp 2: với mỗi hệ số D j >0 đều tồn tại ít nhất một hệ số aij >0 thì chuyển sang bước 5.

* Bước 5: Tương ứng với các hệ số Dj >0, ta tính min{bk/akj | aij>0}: bk là số hạng tự do tương ứng với dòng biểu diễn của xk trong hệ ràng buộc. Giả sử giá trị nhỏ nhất tương ứng với giá trị k=i. Phần tử aij gọi là phần tử giải được, ta chuyển sang bước 6.

* Bước 6 : Ta chuyển sang cơ sở mới bằng cách khử xi trong cơ sở cũ và đưa xj vào cơ sở mới. Quay về bước 2.

Giải bài toán QHTT trên máy tính

Bài toán QHTT có độ phức tạp và khối lượng tính toán rất lớn. Ngay từ những năm 1952 người ta đã sử dụng MTĐT để giải quyết các bài QHTT cỡ lớn. Ngày nay với các ngôn ngữ lập trình cấp cao như TURBO PASCAL, C... ta có thể lập các chương trình để giải quyết bài toán tổng quát. Với một bài toán cụ thể, ta chỉ việc nhập bộ dữ liệu tương ứng (Trong giáo trình Tối ưu hoá [2] đã trình bày chương trình giải bài toán QHTT bằng phương pháp đơn hình). Trong bài viết này chúng tôi xin giới thiệu một cách giải bài toán QHTT bằng cách sử dụng bảng tính điện tử EXCEL - là một phần mềm hầu như không một máy tính nào không cài đặt.

Xin mô tả cách làm qua một ví dụ cụ thể:

Giải bài toán QHTT:

Đây là trường hợp bài toán QHTT không có sẵn cơ sở đơn vị. Để giải quyết bài toán ta phải sử dụng phương pháp hai pha (sử dụng ẩn giả - bài toán M). Nếu sử dụng bảng tính điện tự Excel thì công việc rất đơn giản.

$$\begin{cases} x_1 + 2x_2 - x_3 + x_4 = 2 \\ 2x_1 - 6x_2 + 3x_3 + 3x_4 = 9 \\ x_1 - x_2 + x_3 - x_4 = 12 \\ x_j \geq 0, j = 1,2,3,4 \end{cases}$$

f(x) = -3x1+x2+3x3-x4 -> Min

- Trước tiên ta cần sử dụng 4 ô để chứa các giá trị của 4 ẩn xj.

Ta chọn vùng B3:B6 tương ứng các ẩn x1, x2, x3, x4.

- Sử dụng một ô để lưu trữ biểu thức của '>f. Ta chọn ô D6.

Ta nhập vào ô D6 công thức : = -3*b3+b4+3*b5-b6="">=> (f(x) = -3x1+x2+3x3-x4)

- Vì hệ ràng buộc có 3 phương trình, nên ta sử dụng 3 ô D3,D4, D5 để ghi 3 biểu thức ràng buộc thành phần.

Tại ô D3 ta nhập công thức: =b3+2*b4-b5+b6>=b3+2*b4-b5+b6>(x1+2x2-x3+x4)

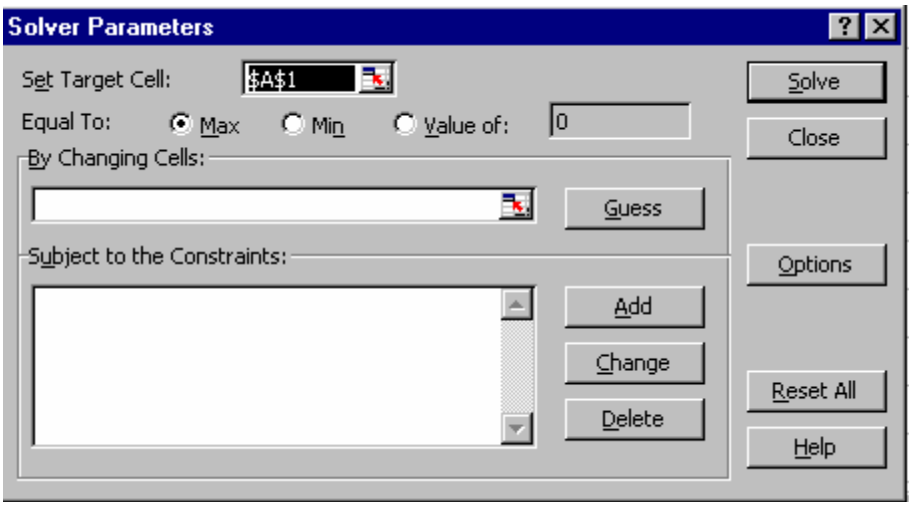
Tại ô D4 ta nhập công thức: =2*b3-6*b4+3*b5+3*b6>=2*b3-6*b4+3*b5+3*b6>(2x1-6x2+3x3+3x4)

Tại ô D5 ta nhập công thức:= B3-B4+B5-B6 (x1 - x2 +x3 - x4)

Giải bài toán QHTT bằng bảng tính điện tử EXCEL

Các ẩn xi		Hệ điều kiện ràng buộc	
ẩn x1	ô B3	Phương trình thứ nhất	ô D3 =b3+2*b4-b5+b6>=b3+2*b4-b5+b6>
ẩn x2	ô B4	Phương trình thứ hai	ô D4 =2*b3-6*b4+3*b5+3*b6>=2*b3-6*b4+3*b5+3*b6>
ẩn x3	ô B5	Phương trình thứ ba	ô D5 = B3-B4+B5-B6
ẩn x4	ô B6	Hàm mục tiêu :	ô D6 =-3*b3+b4+3*b5-b6>=-3*b3+b4+3*b5-b6>

- Thực hiện lệnh Tool Solver->Tool ->Solver.. xuất hiện hộp thoại (hình 1)



Hình 1

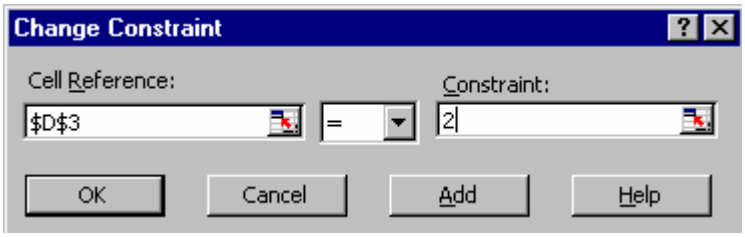
Ta khai báo như sau:

- Nhập địa chỉ ô chứa hàm mục tiêu f : $D$6 vào hộp Set Target Cell.

- Vì bài toán yêu cầu tìn min(f) nên ta bấm chọn Min trong mục Equal to.

- Ta chọn vùng từ B3 đến B6 để lưa trữ các ẩn xưi nên tại hộp By Changing Cells ta nhập địa chỉ: $B$3:$B$6.

-



Hình 2

Tiếp theo ta phải nhập các điều kiện ràng buộc vào hộp: Subject to the Constraints bằng cách: Bấm vào nút lệnh Ađ, xuất hiện hộp thoại:

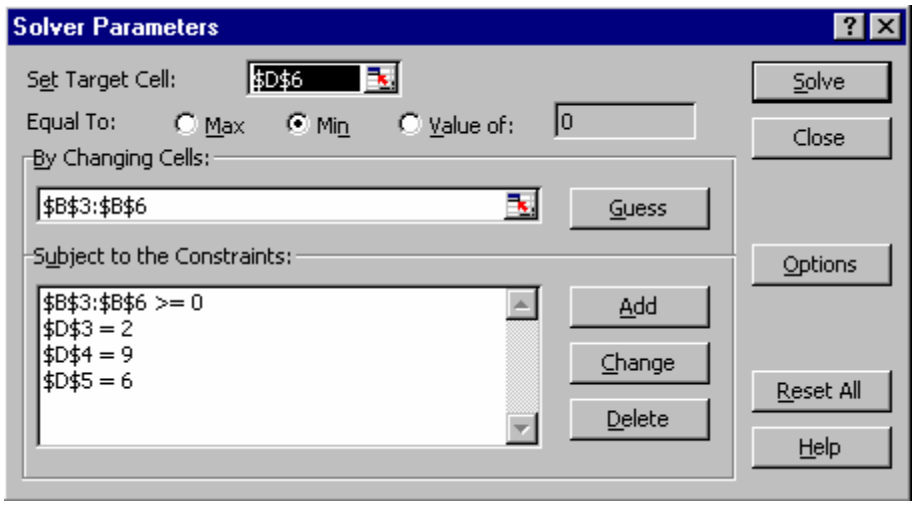
Tại hộp Cell Reference (hình 2): ta nhập địa chỉ $D$3 (để con trỏ tại ô này sau đó bấm chuột vào ô D3 trên bảng tính). Tại hộp Constraint ta chọn phép toán = (bấm chuột vào mũi tên, sau đó bấm vào dấu =), tiếp theo nhập số 2 vào ô kế tiếp (hệ số tự do của phương trình thứ nhất của hệ điều kiện).

Nhập song bấm vào nút lệnh Ađ.

Tương tự ta nhập 2 ràng buộc còn lại : $D$4=9 và $D$5=6.

Cuối cùng ta nhập điều kiện các ẩn không âm: $B$3:$B$6 ≥0 (để con trỏ tại ô Cell Reference, sau đó dùng chuột bôi đen vùng B3:B6, sau đó chọn phép toán ≥ và nhập giá trị 0 tại ô Constraint.

Kết thúc khai báo, hộp thoại Tool Solver có dạng sau (hình 3):



Hình 3

Kết thúc công việc, ta bấm chuột vào nút lệnh Solve, hộp thoại Solver Results hiện ra, ta chọn mục Keep Solver Solution, sau đó chọn OK. Kết quả bài toán là:

Giải bài toán QHTT bằng bảng tính điện tử EXCEL

Các ẩn xi		Hệ điều kiện ràng buộc	
ẩn x1	3	Phương trình thứ nhất	2
ẩn x2	2	Phương trình thứ hai	9
ẩn x3	5	Phương trình thứ ba	6
ẩn x4	0	Hàm mục tiêu:	8

Phương án tối ưu của bài toán là: X(3,2,5,0) và min (f) = 8. Nếu cần chi tiết, tại hộp thoại Solver Results, ta đánh dấu chọn các mục Answer, Sensitivity, Limits trong mục Report, Excel sẽ cho ta bảng báo cáo chi tiết. So với lập trình hoặc tính bằng bảng đơn hình thì thật là đơn giản.

Trong phạm vi một bài báo không đề cập một cách chi tiết về bài toán QHTT và trình bày chi tiết các thao tác trên bảng tính. Rất mong các bạn có quan tâm tiếp tục khai thác chức năng Tool Solver của bảng tính điện tử Excel để giải bài toán QHTT và các chức năng tuyệt vời khác của nó mà bài báo không có điều kiện đề cập đến.

/* Tài liệu tham khảo:

[1]. Giáo trình QHTT - Phí Mạnh Ban - NXB Giáo dục - 1999.

[2]. Giáo trình Tối Ưu hoá - Bùi Thế Tâm - Bùi Minh Trí - NXB GTVT 1996 */

41 .**Một số bài toán quy hoạch động**

Tác giả: Đỗ Quang Tiến

Khi gặp một bài toán tin có yêu cầu tìm kết quả tối ưu về một hay nhiều tính chất nào đấy, hẳn không ít người nghĩ ngay đến sử dụng giải thuật quy hoạch động để giải bài toán. Tại sao lại vậy Bởi vì quy hoạch động thường có độ phức tạp tính toán không cao nghĩa là chương trình sẽ chạy cho ra kết quả đúng trong thời gian ngắn cho phép. Tuy nhiên, không phải bài toán với yêu cầu tối ưu nào cũng có thể giải bằng quy hoạch động, mặt khác cũng có không ít bài toán đúng là có thể giải bằng quy hoạch động nhưng việc phát hiện và áp dụng phương pháp này để giải là không đơn giản.

Việc phát hiện cũng như áp dụng quy hoạch động để giải bài toán phụ thuộc rất lớn vào khả năng tư duy của bạn và đặc biệt là những kinh nghiệm mà bạn có.

Bài viết này sẽ không đề cập đến những khái niệm cơ bản của quy hoạch động vì những khái niệm này đã quá quen thuộc với mọi người. Bài viết chỉ dừng ở mức phân tích cụ thể lời giải của một số bài toán khá hay, từ đó hy vọng ít nhiều giúp bạn có thêm một chút kinh nghiệm trong lập trình giải quyết các bài toán tin.

Trước tiên ta cùng xét một bài toán đã được sử dụng trong kỳ thi Olympic Tin học sinh viên Thủ đô năm 1998.

Bài 1. Giá trị biểu thức

Giả thiết X,Y là hai số nguyên dương. Kí hiệu Sx là tổng các chữ số trong dạng biểu diễn cơ số 10 của X, Dmax_y là chữ số lớn nhất và Dmin_y là chữ số nhỏ nhất trong dạng biểu diễn cơ số 10 của Y. Phép tính hai ngôi # với các toán hạng nguyên dương X,Y được định nghĩa như sau:

(X#Y) = Sx*Dmax_y + Dmin_y

Ví dụ:

(30#9) = 3*9 + 9 = 36

(9#30) = 9*3 + 0 = 27

Với X cho trước, một số biểu thức hợp lệ là:

(X#X)

((X#X)#X)

(X#(X#X)#(X#X)#X)

Ký hiệu kết quả biểu thức là K. Cho X và K (0 < X,K < 109-1) cần xác định số ít nhất m các phép # để từ đó có thể xây dựng biểu thức thuộc dạng đang xét với X cho kết quả K và biểu diễn của biểu thức.

Dữ liệu vào từ file văn bản BT.IN, dòng thứ nhất chứa X, dòng thứ hai chứa K.

Kết quả ra file văn bản BT.OUT, dòng thứ nhất chứa m, dòng thứ hai chứa biểu thức.

Ví dụ:

BT.IN	BT.OUT
718 81	3 ((718 #(718 #718)) #718)

* * *

Thực ra đề bài này cũng dễ hiểu không phức tạp lắm, bây giờ ta đi vào phân tích tìm lời giải cho bài toán. Trước tiên nhận xét rằng cho 0 < X, K < 109 nên:

$1 \le Sx \le 9*9$

$1 \le Dmax\_x \le 9$

$0 \le Dmin\_x \le 9$

Từ đó $1 \le X\#X \le 9*9*9+9 = 738$, tức là biểu thức có một dấu # luôn mang giá trị trong đoạn [1,738]. Suy rộng hơn thì giá trị của một biểu thức hợp lệ bất kỳ phải nằm trong đoạn [1,738], đây chính là cốt lõi lời giải cho bài toán.

Rõ ràng ta chỉ chấp nhận các giá trị K thoả mãn $1 \le K \le 738$, nếu K nằm ngoài khoảng này thì chắc chắn vô nghiệm.

Xét biểu thức X#X có 1 dấu #, dễ thấy có 3 cách mở rộng biểu thức 1 dấu # này là:

- X#(X#X)

- (X#X)#X

- (X#X)#(X#X)

Giả sử B là một biểu thức tạo bởi X và n dấu # thế thì có 3 cách mở rộng biểu thức này là:

- X#B (n+1 dấu #)

- B#X (n+1 dấu #)

- B#B (2*n+1 dấu #)

Ta lập mảng một chiều L[1..738] trong đó L[i] cho biết số phép # ít nhất để từ X tạo ra kết quả là i, i=1..738. Dễ thấy mảng L mang tính truy hồi, lần ngược và khi làm cụ thể ta thực hiện như sau:

1. Khởi tạo mảng A[1..738] := 0, mảng A đánh dấu các giá trị đã tạo được từ biểu thức có X và #; khởi tạo mảng L nhận các giá trị Maxint.

2. Tìm T=X#X; A[T]:=1; L[T]:=1

3. Thực hiện bước 3 cho đến khi mảng L không còn bị thay đổi:

For i:=1 to 738

Nếu A[i]=1 thì

t:=X#i;

Nếu L[t]>L[i]+1 thì

L[t]:=L[i]+1; A[t]:=1;

t:=i#X;

Nếu L[t]>L[i]+1 thì

L[t]:=L[i]+1; A[t]:=1;

t:=i#i;

Nếu L[t]>2*L[i]+1 thì

L[t] :=2*L[i]+1; A[t]:=1;

L[K] cho số phép # tối thiểu của biểu thức cần tìm.

Để hoàn thiện chương trình tất nhiên phải thiết kế thêm mảng lưu trữ thế nào để sau này khi in kết quả có thể đưa ra được cả vị trí các dấu # và các dấu ngoặc của biểu thức. Trong chương trình dưới đây, mảng Pre và D có chức năng này. Sau đây là toàn văn chương trình giải bài "Giá trị biểu thức":

```
{Gia Tri Bieu Thuc - Qui hoach dong DQT }

uses crt;

const

nmax=738;

inp='bt.in';

out='bt.out';

var

f:text;
```

```
x,k :longint;
stop :boolean;
a,d :array[1..nmax]of byte;
l,pre:array[1..nmax]of integer;
procedure Nhap;
begin
assign(f,inp);
reset(f);
readln(f,x); readln(f,k);
close(f);
end;
function TinhGiaTri(x,y:longint):integer;
var i,k,dmax,dmin,sx:byte;
st:string;
c:integer;
begin
str(y,st);
dmax:=0; dmin:=9;
for i:=1 to length(st) do
begin
val(st[i],k,c);
```

```
if kthen dmin:=k;
if k>dmax then dmax:=k;
end;
str(x,st);
sx:=0;
for i:=1 to length(st) do
begin
val(st[i],k,c);
sx:=sx+k;
end;
TinhGiaTri:=sx*dmax+dmin;
end;
procedure TimBieuThuc;
var i,t,top:integer;
begin
fillchar(a,sizeof(a),0);
for i:=1 to 738 do L[i]:=maxint;
{---}
t:=TinhGiaTri(x,x);
a[t]:=1; l[t]:=1;
stop:=false;
```

```
while not stop do
begin
stop:=true;
for i:=1 to nmax do
if a[i]=1 then
begin
t:=TinhGiaTri(x,i);
if l[t]>l[i]+1 then
begin
a[t]:=1; l[t]:=l[i]+1;
pre[t]:=i; d[t]:=1; stop:=false;
end;
{---}
t:=TinhGiaTri(i,x);
if l[t]>l[i]+1 then
begin
a[t]:=1; l[t]:=l[i]+1;
pre[t]:=i; d[t]:=2; stop:=false;
end;
{---}
t:=TinhGiaTri(i,i);
```

```
if l[t]>2*l[i]+1 then
begin
a[t]:=1; l[t]:=2*l[i]+1;
pre[t]:=i; d[t]:=3; stop:=false;
end;
end;
end;
procedure path(t:integer);
begin
if pre[t]<>0 then
begin
write(f,'(');
case d[t] of
1: begin
write(f,x,'#'); path(pre[t]);
end;
2: begin
path(pre[t]); write(f,'#',x);
end;
3: begin
path(pre[t]); write(f,'# '); path(pre[t]);
```

```
end;
end;
write(f,')');
end
else write(f,'(',x,'#',x,')');
end;
procedure Xuly_Inkq;
begin
assign(f,out);
rewrite(f);
if k>738 then writeln(f,'0')
else
begin
TimBieuThuc;
if a[k]=0 then writeln(f,'0')
else
begin
writeln(f,l[k]); path(k);
end;
end;
close(f);
```

```
end;

BEGIN

clrscr;

Nhap;

Xuly_Inkq;

END.
```

Nhận thấy rằng cái khoá của bài trên là giới hạn của kết quả biểu thức từ đó nếu chịu khó suy nghĩ thì từ cái khoá này có thể mở ra, phát triển thành rất nhiều bài toán tương tự. Tiếp theo ta phân tích một bài toán khác có cách giải, dĩ nhiên, vẫn là qui hoạch động nhưng ở một hình thức biểu hiện khác.

Bài 2 : Lịch thuê nhân công

Có một dự án kéo dài trong T tháng và người quản lý cần phải lập lịch sử dụng công nhân trong dự án, anh ta biết số công nhân tối thiểu cần trong mỗi tháng. Mỗi khi thuê hay sa thải một công nhân thì đều phải mất một chi phí xác định, mỗi công nhân được thuê sẽ vẫn nhận được lương tháng ngay cả khi không sử dụng anh ta làm việc.

Với mỗi công nhân, người quản lý biết chi phí thuê, chi phí sa thải và tiền lương phải trả cho công nhân đó trong 1 tháng. Và bài toán đặt ra như sau: Cần phải thuê hay sa thải bao nhiêu công nhân mỗi tháng để tổng chi phí dành cho nhân công của dự án là nhỏ nhất, tức là giảm tối đa chi phí của dự án.

Dữ liệu vào từ file văn bản EMPLOY.IN có cấu trúc như sau:

- Dòng đầu ghi T là số tháng diễn ra dự án (T=<100).

- Dòng thứ hai ghi 3 số lần lượt là chi phí thuê, lương tháng, chi phí sa thải mỗi công nhân.

- Dòng cuối cùng là T số nguyên dương, số thứ i cho biết số công nhân tối thiểu cần cho tháng thứ i, các giá trị số không quá 150.

Kết quả ra file văn bản EMPLOY.OUT theo định dạng:

- Dòng thứ nhất ghi tổng chi phí nhỏ nhất tìm được.

- Dòng thứ hai ghi T số, số thứ i là số công nhân hoạt động trong dự án tại tháng thứ i.

Ví dụ về file dữ liệu vào và file kết quả ra:

EMPLOY.IN	EMPLOY.OUT
3 4 5 6 10 9 11	265 10 10 11

* * *

Đề bài này hơi rắc rối khó hiểu một chút vì vậy cần phải đọc kỹ, nắm chắc. Rõ ràng việc thuê thêm hay sa thải công nhân đều phải chịu phí tổn nên thấy: để đảm bảo chi phí ít nhất cho việc thuê nhân công trong cả dự án thì số công nhân đang thuê trong một tháng không nhất thiết phải là số công nhân tối thiểu cần cho tháng đấy. Ví dụ, phí tổn để thuê cũng như sa thải công nhân rất lớn thì không dại gì ta lại thường xuyên thuê rồi lại sa thải người trong từng tháng, tốt nhất nên giữ họ và trả lương (dù họ không làm gì). Nhiệm vụ của chúng ta trong mỗi tháng phải quyết định có bao nhiêu công nhân trong biên chế thuê, nghĩa là phải quyết định xem cần thuê hay cần sa thải bao nhiêu công nhân trong biên chế của tháng trước. Nếu gọi T_max là số công nhân của tháng cần nhiều người nhất thì rõ ràng số công nhân trong biên chế thuê của một tháng bất kỳ trong dự án không bao giờ vượt quá T_max. Xét một tháng nào đó, biết rằng không nên sử dụng quá T_max người và cũng không được phép sử dụng ít hơn số người tối thiểu cần cho tháng đấy, nhưng phải chọn giá trị nào trong khoảng giới hạn này Đến đây nếu các giá trị của T và T_max là nhỏ thì có thể duyệt tìm ra kết quả tối ưu nhưng do các giá trị này lại có thể khá lớn nên buộc phải tìm cách khác hiệu quả hơn.

Lập mảng Scn[1..T], Scn[i] cho biết số công nhân tối thiểu cần cho tháng thứ i, i=1..T (mảng này nhập vào từ file input). Lập thêm mảng C[T,T_max] trong đó C[i,j] cho biết chi phí tối thiểu của i tháng đầu tiên của dự án nếu tại tháng thứ i có j công nhân trong biên chế thuê (i=1..T, j=1..T_max). Thấy là giá trị C[i,j] có thể xác định thông qua các giá trị C tại tháng i-1 (là tháng trước):

C[i,j] = Min{ C[i-1,k] + chi phí để từ k người thành j người }

(i=1..T; j=Scn[i]..T_max; k=Scn[i-1]..T_max)

Bài toán đã đơn giản hơn nhiều khi có được công thức truy hồi và vì độ phức tạp tính toán không lớn nên chắc chắn chương trình sẽ ngắn gọn, cho kết quả tối ưu trong thời gian rất ngắn. Kết quả tối ưu cuối cùng là:

Kq = Min{C[T,j] + chi phí sa thải j người}

(j=Scn[T]..T_max)

Để có thể đưa ra kết quả đầy đủ (số công nhân sử dụng mỗi tháng) thì cần thêm mảng hai chiều Pre, trong đó Pre[i,j] := k, với i=1..T; j=Scn[i]..T_max; k thoả mãn tổng (C[i-1,k] + chi phí để từ k người thành j người) nhỏ nhất. Việc lần ngược tìm kết quả trong mảng Pre không khó.

Nếu bạn từng làm nhiều bài Tin dùng qui hoạch động để giải thì chắc bạn thấy rằng qui hoạch động chỉ là một phương pháp rất chung, khi áp dụng vào mỗi bài một khác, không bài nào giống bài nào. Để minh họa điều này, mời các bạn đón đọc tiếp trong số sau ta cùng xét thêm một số bài toán tin khá hấp dẫn nữa.

Bài 4. Xoay ô

(Đề bài đã được đăng trên số báo trước)

Thực ra đây không phải là một bài dễ dàng ngay cả với những người nhiều kinh nghiệm giải toán tin. Không ít người gặp bài này phải bó tay hay buộc phải dùng duyệt để giải quyết cho dù duyệt tốt đến mấy cũng không thể chạy được với những test có kích thước tối đa. Thế nhưng bài này lại có thể dùng qui hoạch động giải quyết với độ phức tạp tính toán không cao lắm cỡ O(2N*M). Và ta sẽ phân tích cụ thể cách giải bài này bằng qui hoạch động.

Đầu tiên phải chú ý đến một đặc điểm bất thường của đề bài: giới hạn số ô tối đa trên một dòng là 8 trong khi số dòng tối đa lại là 50, chắc không phải ngẫu nhiên lại có giới hạn này. Như đề bài cho biết, mỗi ô được chia làm 4 miền: trên, trái, dưới, phải; mỗi miền lại được tô một trong hai màu đen hoặc trắng, tô màu đen ứng với một số 1, tô màu trắng ứng với số 0. Giả sử ta xét tại một dòng nào đó, miền trên của các ô trên dòng này theo thứ tự từ trái sang phải tạo lên một xâu nhị phân; miền dưới của các ô trên dòng này cũng vậy, cũng tạo nên một chuỗi nhị phân. Nếu phải thoả mãn điều kiện các miền chung cạnh ô phải cùng màu thì dễ thấy chuỗi nhị phân của miền dưới của một dòng chính là chuỗi nhị phân của miền trên của dòng tiếp theo trong bảng. Từ nhận xét này ta đang dần hình dung ra cách giải. Nhắc lại rằng số ô tối đa trên một dòng là 8 tức là chuỗi nhị phân của miền trên hoặc dưới của một

dòng có độ dài không quá 8. Hiển nhiên mỗi chuỗi nhị phân này có thể ứng với một số trong hệ thập phân và số này nằm trong khoảng (0,2N), N =< 8.

Bây giờ ta xét một bài toán con: ta sẽ đánh giá cụ thể 1 dòng của bảng này và biết rằng các miền trên của dòng có thể ứng với một số, miền dưới cũng ứng được với một số; nếu giả sử cho hai số Mt, Md, phải tìm cách xoay một số ít nhất các ô trên dòng sao cho dòng thoả mãn: các miền trên của dòng phải tương ứng tạo nên Mt, các miền dưới tương ứng tạo thành Md, các miền còn lại nếu chung cạnh ô thì phải cùng màu, ngoài ra còn cần xác định những ô nào được xoay nữa. Thực ra với bài toán con có số lượng ô trên một dòng không quá 8 thì có thể dùng duyệt chạy cũng khá hiệu quả. Nhưng nếu ta mở rộng bài toán này thành một bài khác, giả sử số ô xét không quá 200 chẳng hạn, thì chắc chắn phải tìm cách khác thôi. Và cách được đề xuất cho bài mở rộng này lại là qui hoạch động. Tuy nhiên nó cũng không quá phức tạp lắm, bạn hãy thử tự giải quyết xem.

Quay trở lại bài xoay ô của chúng ta, vấn đề trạng thái của một dòng đã được giải quyết, giờ là kết hợp trạng thái các dòng để đưa ra lời giải chung. Bắt đầu bằng việc thiết lập mảng hai chiều D[1..M, 1..2N], D[i,j] cho biết số ô tối thiểu đã xoay để i dòng đầu tiên của mảng thoả mãn yêu cầu đề bài và miền dưới của dòng thứ i tạo lên số j (i=1..M, j=1..2N), thế thì dễ thấy:

D[i,j] = Min { D[i-1,k] + Xikj }

i=1..M; j=1..2N; k=1..2N

Xikj là số tối thiểu các ô trên dòng i được xoay để dòng i có miền trên tương ứng k, miền dưới tương ứng là j.

Gọi KqMin là kết quả tối ưu nhất tức là số tối thiểu các ô phải xoay đối với cả bảng M dòng thì:

KqMin = Min { D[M,k] , k=1..2N }

Như vậy dùng mảng D là biết số ô tối thiểu cần xoay sao cho thoả mãn điều kiện đặt ra của đề bài nhưng chúng ta còn một yêu cầu nữa chưa thực hiện được là cần cho biết những ô nào được xoay. Yêu cầu này thì mảng D chịu, không thể cho biết được, vì vậy ta thiết kế thêm một mảng nữa, mảng hai chiều Pre[1..M,1..2N]. Do biết:

D[i,j] = Min { D[i-1,k] + Xikj }

nên ta gán Pre[i,j]=k trong đó k là giá trị mà D[i-1,k] + Xikj nhỏ nhất. Khi viết ra kết quả vào file output ta chỉ cần lần ngược trong mảng Pre, bạn hãy tự giải quyết lấy, nó không quá phức tạp đâu.

Nếu muốn, bạn hãy tham khảo toàn bộ chương trình giải bài Xoay ô ở sau, về cơ bản thì chương trình đã thể hiện được giải thuật nhưng cụ thể từng bước cài đặt chưa chắc đã tối ưu, bạn có thể thay đổi, chỉnh sửa cho hợp lý.

```
{ Xoay Cac O - Qui hoach dong - DQT }

uses crt;

const

inp='xoay.in';

out='xoay.out';

mmax=50;

nmax=8;

tt :array[1..8]of byte=(1,3,7,15,31,63,127,255);

x :array[1..4]of byte=(0,1,1,1);

type

m_int=array[0..mmax,0..255]of word;

m_byt=array[0..mmax,0..255]of byte;

tto=array[1..4]of byte; {trang thai o}

var

f:text;

m,n,ttmax:byte; {trang thai Max}

a:array[1..mmax,1..nmax]of tto; {trang thai dau nhap tu input}
```

```
pre:m_byt; {so ung voi cac mien tren tren 1 dong}
d:m_int;
{d[i,j] - so o xoay it nhat de dua dong i ve trang thai j}
{Danh cho tim cach xoay}
bt,bd :byte; {bit tren, bit duoi}
xmo :array[1..4]of tto; {xet 4 trang thai tai mot o}
trai,sbd,prex:array[1..nmax,1..4]of word; {sbd - so bien doi}
procedure Nhap;
var i,j,k:byte;
begin
assign(f,inp);
reset(f);
readln(f,m,n);
for i:=1 to m do
begin
for j:=1 to n do
for k:=1 to 4 do read(f,a[i,j,k]);
readln(f);
end;
close(f);
end;
```

```
procedure KhoiTao;
var i:byte;
begin
ttmax:=tt[n];
for i:=0 to ttmax do
begin
d[0,i]:=0;
pre[0,i]:=0;
end;
end;
procedure GetBit(so,vt:byte;var bit:byte);
begin
bit:=(so shr vt) and 1;
end;
procedure xoay90(tto1:tto;var tto2:tto);
begin
tto2[1]:=tto1[2]; tto2[2]:=tto1[3];
tto2[3]:=tto1[4]; tto2[4]:=tto1[1];
end;
procedure xoay180(tto1:tto;var tto2:tto);
begin
```

```
tto2[1]:=tto1[3]; tto2[2]:=tto1[4];
tto2[3]:=tto1[1]; tto2[4]:=tto1[2];
end;
procedure xoay270(tto1:tto;var tto2:tto);
begin
tto2[1]:=tto1[4]; tto2[2]:=tto1[1];
tto2[3]:=tto1[2]; tto2[4]:=tto1[3];
end;
procedure tinh_xoay(i,st,sd:byte;var temp:integer;var ttc:byte);
var
k,t,s:byte;
begin
xmo[1]:=a[i,1]; xoay90(a[i,1],xmo[2]);
xoay180(a[i,1],xmo[3]); xoay270(a[i,1],xmo[4]);
getbit(st,0,bt); getbit(sd,0,bd);
for t:=1 to 4 do
if (xmo[t,1]=bt)and(xmo[t,3]=bd) then
begin
trai[1,t]:=xmo[t,4]; sbd[1,t]:=x[t];
end
else trai[1,t]:=2;
```

```
{}
for k:=2 to n do
begin
xmo[1]:=a[i,k]; xoay90(xmo[1],xmo[2]);
xoay90(xmo[2],xmo[3]); xoay90(xmo[3],xmo[4]);
getbit(st,k-1,bt); getbit(sd,k-1,bd);
for t:=1 to 4 do
begin
sbd[k,t]:=maxint;
for s:=1 to 4 do
if (xmo[t,1]=bt)and(xmo[t,2]=trai[k-1,s])and(xmo[t,3]=bd)
and(sbd[k,t]>sbd[k-1,s]+x[t]) then
begin
sbd[k,t]:=sbd[k-1,s]+x[t];
trai[k,t]:=xmo[t,4];
prex[k,t]:=s;
end;
end;
end;
temp:=maxint;
for k:=1 to 4 do
```

```
if temp>sbd[n,k] then
begin
temp:=sbd[n,k];
ttc:=k;
end;
end;
procedure Xuly;
var i,j,k,ttc:byte;
temp:integer;
begin
for i:=1 to m do
begin
for j:=0 to ttmax do
begin
d[i,j]:=maxint;
for k:=0 to ttmax do
begin
Tinh_xoay(i,k,j,temp,ttc);
if d[i,j]>d[i-1,k]+temp then
begin
d[i,j]:=d[i-1,k]+temp;
```

```
pre[i,j]:=k;
end;
end;
end;
end;
end;
procedure Sua_Mang_Ăi,ttc:byte);
var k,t:byte;
begin
for k:=n downto 1 do
begin
case ttc of
2:xoay90(a[i,k],a[i,k]);
3:xoay180(a[i,k],a[i,k]);
4:xoay270(a[i,k],a[i,k]);
end;
ttc:=prex[k,ttc];
end;
end;
procedure Inkq;
var
```

```
min,temp:integer;
i,j,k,ttc,vt:byte;
begin
min:=maxint;
for i:=0 to ttmax do
if d[m,i]
begin
min:=d[m,i];
vt:=i;
end;
{---}
assign(f,out);
rewrite(f);
if min<>maxint then
begin
writeln(f,min);
for i:=m downto 1 do
begin
Tinh_Xoay(i,pre[i,vt],vt,temp,ttc);
Sua_mang_Ăi,ttc);
vt:=pre[i,vt];
```

```
end;
for i:=1 to m do
begin
for j:=1 to n do
for k:=1 to 4 do write(f,a[i,j,k]:2);
writeln(f);
end;
end
else writeln(f,'No Solution.');
close(f);
end;
BEGIN
clrscr;
Nhap;
KhoiTao;
XuLy;
Inkq;
END.
```

Phải nói rằng "Xoay ô" là một bài qui hoạch động hay, đặc biệt nó lại chứa đựng bên trong một bài toán con khác cũng dùng qui hoạch động để giải. Tuy nhiên khi N=8 thì chương trình chạy cũng hơi lâu, và có hạn chế là cách giải này không thể thực thi khi $N \geq 9$.

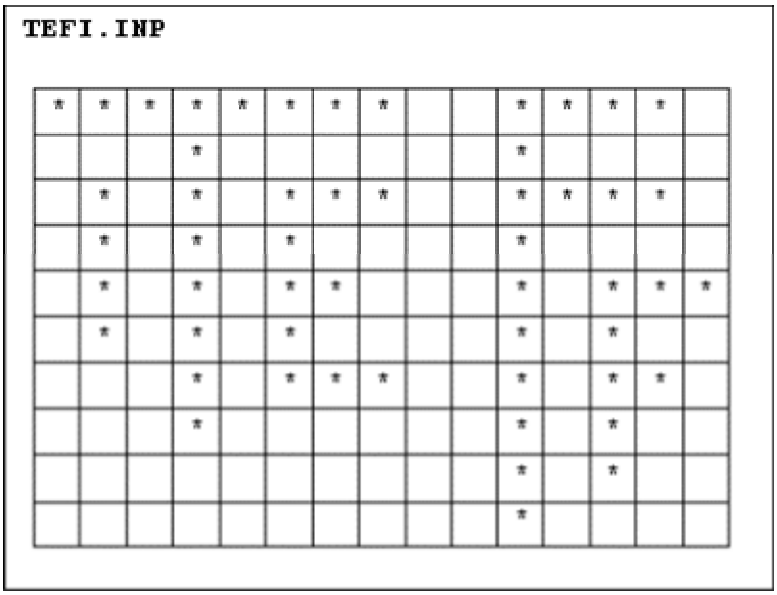
Các bài toán tin giải bằng phương pháp qui hoạch động rất đa dạng, phong phú cả về nội dung lẫn về mô hình lời giải. Hi vọng rằng bài viết có thể ít nhiều giúp các bạn hiểu rõ hơn việc đánh giá, phân tích bài toán và đưa những đánh giá phân tích này thành lời giải cụ thể.

42 .Các bài toán có yếu tố nhận dạng

Tác giả: Nguyễn Xuân Huy

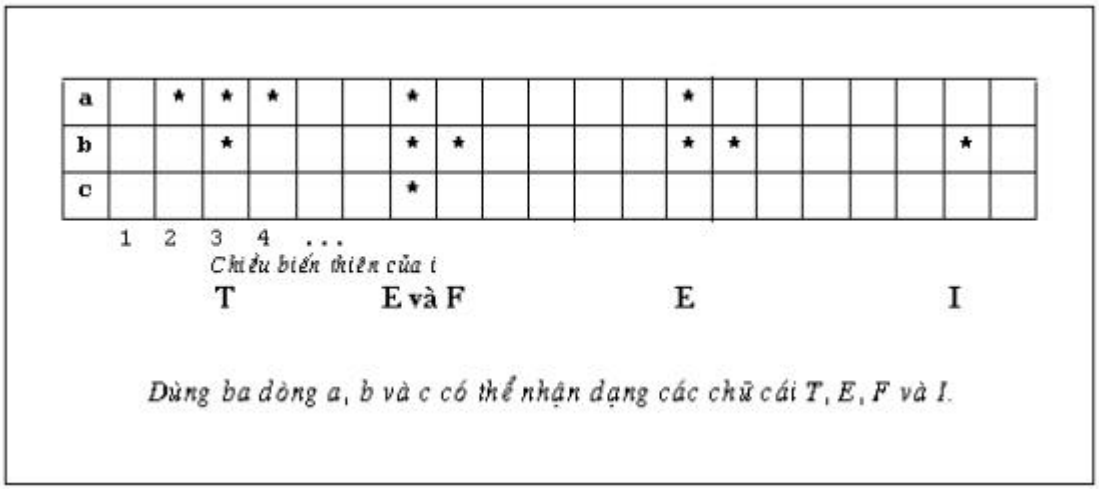
Một số bài toán tin đòi hỏi sự phân biệt giữa đối tượng này với các đối tượng khác. Đó là lớp các bài toán nhận dạng. Nhận dạng một đối tượng đòi hỏi liệt kê các đặc trưng cho đối tượng đó sao cho ta có thể nhận biết được chúng. Việc liệt kê như vậy gọi là đặc tả đối tượng. Các dấu hiệu liệt kê trong đặc tả phải đủ nhỏ, đặc trưng và càng dễ thực hiện càng tốt. Bức tranh con kiến và bức tranh con voi khác nhau không phải ở kích thước. Người ta có thể vẽ một con voi nhỏ như hạt gạo và một con kiến to như cái thúng, vậy mà người xem vẫn nhận biết đâu là kiến đâu là voi. Chúng ta thử tìm các dấu hiệu đặc tả cho bài toán sau:

Bài toán 1: (TEFI) Hãy đếm số chữ cái mỗi loại trong số 4 chữ cái T, E, F và I. Các chữ cái nói trên được viết trong một tệp văn bản có tên TEFI.INP theo các quy tắc sau:
1. Các chữ đều thuộc loại IN HOA không chân.
2. Các nét chữ được viết bằng các dấu hoa thị (*) và là các nét đơn, tức là có độ dày là một dấu *.
3. Chiều dài của mỗi nét tối thiểu là 2 dấu *.
4. Các chữ không dính nhau: nét của chữ này cách nét của chữ kia ít nhất là một dấu cách.
5. Các nét ngang của cùng một chữ không dính nhau.
6. Tệp chỉ chứa các dấu hoa thị và dấu cách.
7. Chiều dài tối đa của mỗi dòng trong tệp là 70 ký tự.
8. Tổng số chữ cái có trong tệp có thể đạt tới 60000.
Dữ liệu ra ghi trong tệp văn bản TEFI.OUT gồm 4 số dt, de, df và di được viết trên một dòng cách nhau bởi dấu cách và biểu thị số lượng chữ cái mỗi loại theo thứ tự T, E, F và I.
Thí dụ:




```
TEFI.OUT
1 1 2 1
```

Bài giải:



Gọi dt, de, df và di lần lượt là các biến đếm số lượng các chữ cái T, E, F và I có trong văn bản. Ta để ý rằng chữ T có chứa một "ngã ba" dọc để phân biệt với chữ E và chữ F cùng chứa một "ngã ba" ngang. Chữ I thì có "đầu nhọn". Vậy ta có đặc tả đầu tiên sau đây:

CHỮ CÁI	NGÃ BA	ĐẦU NHỌN	GÓC VUÔNG DƯỚI
T	1 dọc	không	không
E	1 ngang	không	có
F	1 ngang	không	không
I	không	có	không

Từ đó rút ra:
dt = số lượng các "ngã ba" dọc trong văn bản
di = số lượng các "đầu nhọn" trong văn bản
E và F đều có 1 ngã ba ngang do đó tổng số chữ E và F chính là tổng số ngã ba ngang. Ta dùng biến dEF để đếm tổng này. Ta có:
dEF = số lượng các "ngã ba" ngang = tổng số chữ E và F.
Riêng chữ E có thêm góc vuông dưới nên
dE = số lượng góc vuông dưới.
Khi đó số lượng chữ F sẽ là hiệu của dEF và dE,
dF = dEF - dE.
Tổng hợp lại ta chỉ cần quản lý ba dòng đọc gần nhất là a, b và c theo thứ tự từ trên xuống. Gọi ss là hằng chứa dấu *, bl là hằng chứa dấu cách khi đó với mỗi vị trí i trên các dòng ta có thể đặc tả mỗi chữ như sau:
ChuT := (a[i]=ss) and (b[i]=ss) and (a[i-1]=ss) and (a[i+1]=ss);
ChuEF := (a[i]=ss) and (b[i]=ss) and (c[i]=ss) and (b[i+1]=ss);
ChuE := (a[i]=ss) and (b[i]=ss) and (c[i]=bl) and (b[i+1]=ss);
ChuI := (a[i]=bl) and (b[i]=ss) and (b[i-1]=bl) and (b[i+1]=bl);
Lúc đầu ta khởi trị cho 2 dòng a và b toàn dấu cách. Mỗi lần ta đọc một dòng của tệp vào biến c. Sau khi xử lý xong bộ ba a, b, c ta chuyển b vào a, c vào b để chuẩn bị đọc dòng mới vào c ở bước kế tiếp.
Ta cần chú ý rằng sau khi đọc xong và xử lý dòng cuối cùng của tệp ta còn phải làm một công việc nhỏ nữa. Đó là phát hiện góc vuông cuối cùng của chữ E. Do đó ta phải gán trị một dòng c chứa toàn dấu cách để xử lý bộ ba a, b, c cuối cùng này với mục đích duy nhất là đếm thêm các chữ E cuối cùng nếu có. Thủ tục cơ bản được mô tả trong đoạn trình dưới đây:

```
procedure run;
var i: byte;
begin
clrscr;
init;
while not eof(f) do
begin
docdong(c);
```

```
for i:=1 to mn do
if chuT(i) then inc(dT)
else if chuI(i) then inc(dI)
else if chuEF(i) then inc(dEF)
else if chuE(i) then inc(dE);
a:=b;
b:=c;
end;
fillchar(c,sizeof(c),BL);
for i:=1 to mn do
if chuE(i) then inc(dE);
finit;
end;
```

Chương trình thu được sẽ như sau:

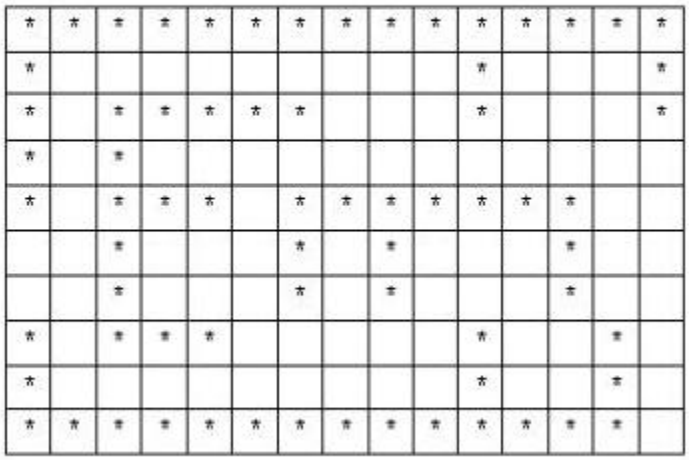
```
uses crt;
const
fn = 'TEFI.IN4';
bl = ' '; { Dau cach }
mn = 70;
mn1 = mn+1;
ss = '*';
type
mang = array[0..mn1] of char;
var f: text;
dt,de,di,def: longint;
a,b,c: mang;
s: string[MN1];

procedure init;
begin
assign(f,fn);
reset(f);
dt:=0;
de:=0;
di:=0;
def:=0;
fillchar(a,sizeof(a),BL);
fillchar(b,sizeof(b),BL);
end;
```

```
procedure finit;
begin
close(f);
writeln(dt,BL,de,BL,def-de,BL,di);
end;

procedure docdong(var x: mang);
var i: byte;
begin
readln(f,s);
for i:=1 to length(s) do
x[i]:=s[i];
for i:=length(s)+1 to mn1 do
x[i]:=bl;
end;

function ChuT(i: byte): Boolean;
begin
ChuT := (a[i]=ss) and (b[i]=ss) and (a[i-1]=ss) and (a[i+1]=ss);
end;
function ChuEF(i: byte): Boolean;
begin
ChuEF := (a[i]=ss) and (b[i]=ss) and (c[i]=ss) and (b[i+1]=ss);
end;
function ChuE(i: byte): Boolean;
begin
ChuE := (a[i]=ss) and (b[i]=ss) and (c[i]=bl) and (b[i+1]=ss);
end;

function ChuI(i: byte): Boolean;
begin
ChuI := (a[i]=bl) and (b[i]=ss) and (b[i-1]=bl) and (b[i+1]=bl);
end;
procedure run;
var i: byte;
begin
clrscr;
init;
while not eof(f) do
begin
docdong(c);
```

```
for i:=1 to mn do
if chuT(i) then inc(dT)
else if chuI(i) then inc(dI)
else if chuEF(i) then inc(dEF)
else if chuE(i) then inc(dE);
a:=b;
b:=c;
end;
fillchar(c,sizeof(c),BL);
for i:=1 to mn do
if chuT(i) then inc(dT)
else if chuI(i) then inc(dI)
else if chuEF(i) then inc(dEF)
else if chuE(i) then inc(dE);
finit;
end;

begin
run;
readln;
end.
```

Bàii toán 2: (E xiếc) Hãy đếm số chữ E mỗi loại được viết chân phương hoặc quay đi một góc là bội của 90 độ. Các quy tắc viết được mô tả như trong bài toán 1. Dữ liệu vào được ghi trong tệp văn bản EXIEC.INP. Dữ liệu ra ghi trong tệp văn bản EXIEC.OUT gồm 4 số ep (E phải), et (E trái), el (E lên) và ex (E xuống) được viết trên một dòng cách nhau bởi dấu cách và biểu thị số lượng chữ cái mỗi loại. Các từ ghép phải, trái, lên và xuống được tính theo hướng trỏ của 3 nét ngang.

TEFI.INP



TEFI.OUT

1 0 1 2

43 .**Lập trình giải một số bài toán về hình học**

Tác giả: Nguyễn Văn Trường

Kiến thức sử dụng trong bài viết dưới đây thuộc chương trình hình học ở phổ thông. Vì vậy, nội dung là dễ hiểu đối với học sinh từ lớp 10 trở lên và phục vụ cho các độc giả yêu thích môn tin học, học sinh, sinh viên, giáo viên chuyên ngành tin trong quá trình học tập nâng cao kiến thức.

Bài 1 . Tìm giao điểm (nếu có) của hai đường thẳng trong mặt phẳng tọa độ có phương trình lần lượt là a1x + b1y = c1 và a2x + b2y = c2.

Bài toán này có thể dễ dàng giải được bằng việc sử dụng phương pháp đã biết ở phổ thông. Cụ thể:

D: = a1*b2 - a2*b1; Dx : = c1*b2 - c2*b1; Dy: = a1*c2 - a2*c1;

If D < > 0 then Writeln (' x = ', Dx/D : 4:2, 'y = ', Dy/D: 4:2)

else If (Dx = 0) and (Dy = 0) then Writeln (' H ai đường thẳng trùng nhaú)

else Writeln (' Hai đường thẳng song song');

Bài 2 . Kiểm tra xem 2 đoạn thẳng AB và CD có cắt nhau hay không.

Để giải bài toán này ta nhớ lại một kiến thức đã được học ở phổ thông: 'Trong mặt phẳng tọa độ cho đường thẳng d: ax + by + c = 0. Khi ấy, một trong hai nửa mặt phẳng bờ d, ngoại trừ d, gồm các điểm có tọa độ thoả mãn bất phương trình: ax + by + c > 0. Nửa mặt phẳng kia gồm các điểm có tọa độ thoả mãn bất phương trình: ax + by + c < 0'.

Từ đó ta dễ dàng chứng minh được kết quả sau: 'Điều kiện cần và đủ để hai đoạn thẳng AB và CD cắt nhau là: A, B nằm khác phía so với đoạn CD và C, D nằm khác phía so với đoạn AB'.

Ngoài ra, cũng không được quên một trong những kiến thức cơ bản là: phương trình đường thẳng đi qua hai điểm phân biệt Ăx1 , y1), B(x2 , y2) có dạng:

f(x,y) = (x- x1)(y2 - y1) - (y - y1)(x2 - x1) = 0.

Vậy nếu xét hai điểm M(x3, y3), N (x4 , y4) thì điều kiện cần và đủ để M, N nằm cùng phía (khác phía) so với đường thẳng có phương trình f(x,y) = 0 là:

f(x3,y3)f(x4 , y4) >= 0 (<= 0).

Dựa trên cơ sở đó chương trình giải bài toán trên được cài đặt như sau:

```
Type Diem = Record

X,y: Real;

End;

Var A,B,C,D: Diem;

Function F(A,B,M: Diem): Real;

{ Tính giá trị hàm F tại điểm M với F là hàm ứng với phương trình

đường thẳng đi qua AB }

Begin

F: = (M.x - A.x) * (B.y - A.y) - (M.y - A.y)* (B.x - A.x);

End;
```

```
Function KhacPhia (A,B,C,D:Diem):Boolean;
{ Kiểm tra xem C,D có nằm khác phía so với đường thẳng đi qua A, B hay không? }
Begin
If F(A,B,C) * F(A,B,D) < = 0 then KhacPhia: = True
else khacphia: = False;
End;
BEGIN

Readln(A.x,A.y,B.x,B.y,C.x,C.y,D.x,D.y);
If Khacphia (A,B,C,D) and Khacphia (C,D,A,B) then
Writeln ('Hai đoan thang cat nhaú);
else Writeln ('Khong cat nhaú);
Readln;
END.
```

Bài 3. Kiểm tra điểm M có nằm trên đoạn thẳng AB hay không?

Điểm M nằm trên đoạn thẳng AB <=> 2 điều kiện sau đều thoả mãn:

1. Điểm M nằm trên đường thẳng AB <=> F (A,B,M) = 0;

2. M nằm giữa Avà B <=> (xm - xa). (xm - xb) <= 0 và (ym - ya). (ym -yb) <= 0.

Bài 4 . Kiểm tra điểm M có thuộc tia AB hay không?

Điểm M thuộc tia AB <=> 2 điều kiện sau đều thoả mãn:

1. M,A,B thẳng hàng <=> F(A,B,M) = 0.

2. A không nằm giữa B,M <=> (xm - xa). (xb - xa) >= 0 và (ym - ya). (yb - ya) >= 0.

Bài 5. Kiểm tra tia AB có cắt đoạn CD không?

Có thể đưa ra 2 cách giải như sau:

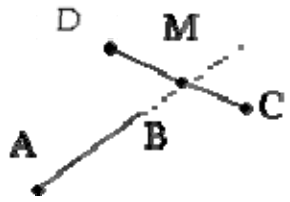
Cách 1 : Trước tiên tìm giao điểm M (nếu có) của 2 đường thẳng AB và CD, khi đó tia AB cắt đoạn CD <=> 2 điều kiện sau đều thoả mãn:

1. M nằm giữa C, D <=> (xm - xc). (xm - xd) <= 0 và (ym - yc). (ym - yd) <= 0.

2. A không nằm giữa B, M <=> (xm - xa). (xb - xa) >= 0 và (ym - ya). (yb - ya) >= 0.

Cách 2 : Tia AB cắt đoạn CD <=> 2 điều kiện sau thoả mãn:

1. C, D nằm khác phía so với AB <=> Khacphia (A,B,C,D) = True.

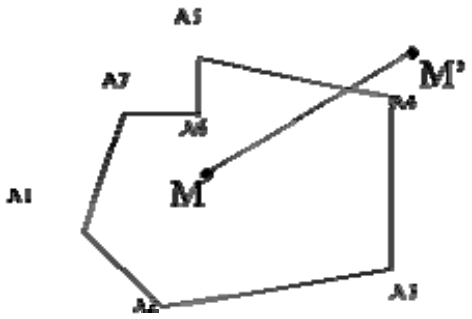


2. A, M nằm khác phía so với CD <=> Khacphia (C,D,A,M) = True. Trong đó M (x,y) là điểm thuộc tia AB và thoả mãn điều kiện:

| xm| > Max (| xc |,|xd |) hoặc |ym | > Max (|yc |,|yd |);

Có thể cài đặt chương trình như sau:

```
Procedure Tim _ M;

Var i: integer;

Xmax,Ymax: Real;

Begin

i =1; Xmax := Max (abs(C. x), abs (D. x));

Ymax : = Max (abs(C. y), abs (D. y)); { Giả sử hàm Max đã xây dựng }

While (abs (A. x + i * (B.x - A.x)) <= Xmax) and (Abs (A. y + i * (B. y - A. y)) <= Ymax) do inc(i);

M.x : = A. x + i * (B.x - A.x);

M. y : = A. y + i * (B.y - A.y);

End;

Procedure Ket_qua;

Begin

If Khacphia (A,B,C,D) then

begin
```



```
Tim_M;

If Khacphia (C,D,A,M) then Writeln ('Tia AB cắt đoạn CD')

else Writeln ('Khong cat ' )

end

else Writeln ('Khong cat ');

BEGIN

< Nha^.p A,B,C,D >;

Ket_ qua;

Readln

END.
```

Bài 6 . Kiểm tra vị trí của một điểm M so với một đa giác (đa giác có thể lồi hoặc lõm).

Để giải bài toán này, trước hết ta tìm M' (x,y) sao cho x > max { xi : xi là hoành độ một đỉnh i của đa giác ; i=1, 2... } . Còn y có giá trị sao cho đoạn MM' không chứa đỉnh nào của đa giác, y có thể tìm bằng phương pháp duyệt trung điểm của các cạnh hay lựa chọn ngẫu nhiên. Tiếp đó ta đếm số giao điểm của đoạn MM' so với tất cả các cạnh của đa giác (sử dụng bài 2).

+ Nếu số giao điểm là lẻ thì M nằm trong đa giác.

+ Nếu số giao điểm là chẵn thì M nằm ngoài đa giác.

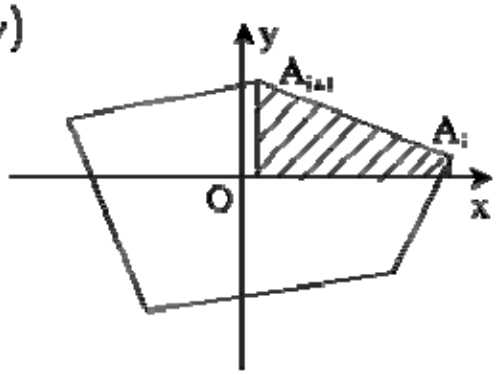
Chú ý : Nếu M thuộc một cạnh hay một đỉnh nào đó của đa giác thì ta cần xử lý riêng.

Bài 7. Tính diện tích đa giác (lồi hoặc lõm và không tự cắt) gồm n đỉnh A [1], A[2],... A[n].

Ta có thể giải bài toán này bằng cách chia đa giác thành n-2 tam giác rồi tính tổng diện tích của các tam giác ấy. Tuy nhiên phương pháp này dài dòng, ta làm cách khác như sau: chia đa giác thành các hình thang bằng cách chiếu các cạnh xuống

trục hoành (hình vẽ). Hình thang được xác lập bởi cạnh A [i] A[i+1] có diện tích là Abs (S) với :

$$S = \frac{1}{2} * (A[i] * x - A[i+1] * x) * (A[i] * y + A[i+1] * y)$$



Sau khi gán đỉnh A[n+1]:=A[1] ta tính diện tích toàn phần của đa giác như sau:

S: = 0;

For i: =1 to N do

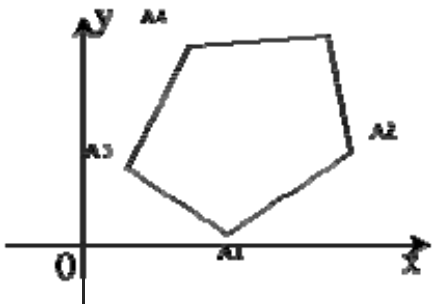
$$S := S + (A[i].x - A[i+1].x) * (A[i].y + A[i+1].y);$$

S: = (1/2) * Abs(S);

Chú ý :

1. Hoàn toàn tương tự ta có thể tìm diện tích của đa giác bằng cách chiếu các cạnh xuống trục tung.

2. Nếu thay bước gán S: = (1/2) * abs (S) bởi S: = S/2 thì dấu của S là dương hay âm sẽ cho ta biết chiều đánh số của các đỉnh theo thứ tự từ 1, 2,.... N là ngược hay xuôi chiều kim đồng hồ.



Bài 8 . (Bài toán tìm bao lồi): Trong mặt phẳng tọa độ cho tập gồm N điểm. Hãy chọn ra M điểm A1, ..., Am (3<= M<= N) sao cho chúng tạo thành một đa giác lồi chứa N - M điểm còn lại của tập N điểm ban đầu.

Có nhiều thuật toán giải bài toán này. Sau đây là một ví dụ.

Bước 1: Chọn A1 là đỉnh hoành độ nhỏ nhất.

> Bước 2 : Trong số các đỉnh chưa chọn, ta chọn ra một đỉnh Ai thoả mãn điều kiện: mọi đỉnh còn lại đều phải nằm về cùng một phía của cạnh tạo bởi 2 đỉnh là Ai và đỉnh được chọn ngay trước Ai (là đỉnh Ai-1).

Lặp lại bước 2 cho tới khi không chọn được đỉnh nào nữa.

Dễ thấy các đỉnh (theo thứ tự) được chọn sẽ tạo ra đa giác thoả mãn điều kiện đầu bài.

Chú ý : Cần xử lý khéo léo khi có một số điểm thẳng hàng.

Thông qua những bài toán cơ bản ở trên, chúng ta thấy rằng các bài toán có liên quan đến kiến thức hình học khá phong phú và hấp dẫn, bạn đọc có thể tự suy ra những bài toán tương tự hoặc phức tạp hơn.

Mời các bạn thử sức với một vài bài toán dưới đây:

1. Kiểm tra hai tia có giao nhau hay không?

2. Tìm tâm và tỉ số vị tự biến đường tròn này thành đường tròn kia khi biết tọa độ hai tâm của hai đường tròn và bán kính của chúng.

3. (Đề thi Học sinh giỏi quốc gia 2000): Trong mặt phẳng tọa độ cho một đa giác lồi gồm N đỉnh. Hãy chia đa giác đó thành hai đa giác bởi một đoạn thẳng nối hai đỉnh sao cho diện tích của chúng chênh lệch nhau ít nhất (gợi ý sử dụng bài 7).

4. Cho trước tọa độ các đỉnh của n tam giác. Hãy tìm phần diện tích của mặt phẳng bị bao phủ bởi các tam giác đó (lưu ý là các tam giác có thể giao nhau).

5. Kiểm tra một đa giác có phải là đa giác lồi hay không.

6. (Đề thi olympic Sinh viên 99): Hãy chọn ra 4 điểm theo thứ tự tạo thành các đỉnh của một tứ giác có diện tích lớn nhất trong số N điểm cho trước.

Trong khuôn khổ bài báo có hạn, rất mong bạn đọc thông cảm, trao đổi và góp thêm nhiều ý kiến về chuyên đề này để mỗi chúng ta ngày càng yêu thích môn tin học lập trình nói chung, cũng như các bài toán tin về hình học nói riêng.

44 .**Nhân và chia hai đa thức**

Tác giả: Lê Mạnh Dũng

Nhân dịp khai giảng năm học mới 2001-2002, Tạp chí Tin học & Nhà trường và tác giả gửi tặng các em học sinh PTCS hai chương trình tin học được viết bằng Turbo Pascal : nhân và chia 2 đa thức mà các em sẽ học ở lớp 8 phổ thông bằng kiến thức lập trình Pascal thông thường, các em đều có thể tự viết cho mình giải các bài toán bằng ngôn ngữ lập trình Pascal. Riêng với hai chương trình nhân và chia hai đa thức một ẩn này, giúp các em kiểm tra lại các kết quả tính toán của mình, làm thêm bài tập, chứ không phải là 'phao cấp cứu, càng không thể thay thế cho tư duy học toán của mình! Hai chương trình hữu ích này còn có thể giúp các em kiểm tra việc phân tích một đa thức thành các nhân tử chung. Còn các chương trình cộng, trừ đa thức xin giành cho các em, xem như bài tập. Các thầy cô giáo có thể ra thêm bài tập, bài kiểm tra dựa vào kết quả các chương trình chạy trên máy tính. Sau đây là 2 chương trình:

```
Program Nhan_hai_da_thuc;

type ma=array[0..50] of real;

var a,b,c : ma;

n,m : integer;

Procedure TieuDe;
```

```
Begin
writeln('CHUONG TRINH NHAN 2 DA THUC ');
writeln('Dang ăx)=an*x^n+...+a1*x+a0');
writeln(' b(x)=bm*x^m+...+b1*x+b0');
writeln('Da thuc tich co dang c(x)=ăx)*b(x)');
writeln('Tim da thuc c(x) ');
End;
Procedure Nhan(s1,s2 : ma;h1,h2 : integer;var s :ma);
var i,j,k : integer;
Begin
write('Vao so bac cua da thuc ăx) ,n = ');readln(h1);
writeln('Vao cac so hang :');
for i:=h1 downto 0 do
begin write('a[',i,']= ');readln(s1[i]); end;
write('Vao so bac cua da thuc b(x) ,m= ');readln(h2);
writeln('Vao cac so hang :');
for i:=h2 downto 0 do
begin write('b[',i,']= ');readln(s2[i]); end;
for i:=h1+h2 downto 0 do s[i]:=0;
for i:=h1 downto 0 do
for j:=h2 downto 0 do
```

```
for k:=h1+h2 downto 0 do
if k=i+j then s[k]:=s[k]+s1[i]*s2[j];
writeln;
writeln('KET QUA cac so hang cua c(x) la :');
for i:=h1+h2 downto 0 do begin writeln('c[',i,']= ',s[i]:3:2);end;
End;
BEGIN
TieuDe;
Nhan(a,b,n,m,c);
readln
END.
Program Chia_hai_da_thuc;
const p='DA THUC c(x) co cac he so tim duoc la :';
q='DA THUC du tim duoc :';
var a,b,c : array[0..50] of real ;
i,j,k,n,m : integer;
BEGIN
writeln('CHUONG TRINH CHIA 2 DA THUC ');
writeln('Dang ăx)=an.x^n+...+a1x+a0');
writeln(' b(x)=bm.x^m+...+b1x+b0');
writeln('Ket qua co dang ăx)=b(x)*c(x)+d(x)');
```

```
writeln('Tim da thuc thuong c(x) va da thuc du d(x)');
writeln(' Mo tep C:Ketqua.out de xem ket qua □);
write('Vao so bac cua da thuc ăx) ,n = ');readln(n);
write('Vao so bac cua da thuc b(x) ,m = ');readln(m);
writeln('Vao cac he so cua da thuc ăx) :');
for i:=n downto 0 do
begin write('a[',i,']= ');readln(a[i]); end;
writeln('Vao cac he so cua da thuc b(x) :');
for i:=m downto 0 do
begin write('b[',i,']= ');readln(b[i]); end;
writeln;
assign(Output,'C:Dulieu.inp');
rewrite(Output);
writeln('Vao cac he so cua da thuc ăx) :');
for i:=n downto 0 do
begin writeln('a[',i,']= ',(a[i]):3:2); end;
writeln('Vao cac he so cua da thuc b(x) :');
for i:=m downto 0 do
begin writeln('b[',i,']= ',(b[i]):3:2); end;
close(Output);
assign(Output,'C:Ketqua.out');
```

```
rewrite(Output);
writeln(p);
if (n>=m) then
begin
while (n>=m) do
begin
c[n-m]:= a[n]/b[m] ;
writeln('c[',n-m,']= ',c[n-m]:3:2);
for i:=m downto 0 do
begin a[n-(m-i)]:=a[n-(m-i)]-c[n-m]*b[i]; end;
n:=n-1;
end;
for i:=n-m downto 0 do writeln('c[',i,'] = ',c[i]:3:2);
writeln(q);
for i:=m-1 downto 0 do writeln('d[',i,'] = ',a[i]:3:2);
end
else
begin
writeln(p);
writeln('c[i]= ',0);
writeln(q);
```

```
for i:=m downto 0 do writeln('d[',i,'] = ',b[i]:3:2);

end;

close(Output);

readln

END.
```

45 .**Một cách khác để khai triển đa thức**

Tác giả: Trương Văn Thuận

Ngoài các cách khai triển hệ số của đa thức (x+y)n mà các bạn đã biết. Tôi muốn trình bày với các bạn một cách khác, xong xem có dễ nhớ, dễ học không thì do bạn nhận định. Cách khai triển trình bày như sau: ta cần khai triển (x+y)n với (n >= 0, nguyên) thành đa thức a0xn + a1*xn-1*y +... + an-1*x*yn-1+ anyn luôn cho ta n+1 hệ số, rõ ràng ta đã biết cách khai triển tam giác Pascal hay dùng cách khai triển Newton

$$(x+y)^n = \sum_{i=1}^{n} C_n^i * x^i * y^{n-i}$$

ta ngại động chạm đến Cni = n! / i!*(n-i)!, một cách khác để ta xác định các ai thông qua ai-1 với 0<= i <= n như sau:

a0 = an = 1

ai i-1*(n-i+1)/ i (i chạy từ 1 đến n-1)

Ví dụ cụ thể : (x+y)44 + 4*x3*y + 6*x2*y2 + 4*x1*y3 + y4

Thuật toán trên chẳng qua là thay vì tính giai thừa để tìm từng hệ số ta chuyển sang t hệ số đi sau bằng hệ số trước đó nhân với số mũ của x (của hệ số ỏ trước đó) và chia cho số mũ của y (của hệ số ỏ trước đó) cộng 1. Lưu ý rằng: số mũ của y của hệ số trước đó cộng 1 bằng số mũ của y đang tìm, số mũ của x thì ngược lại. Đây là cách tính rất đơn giản.

Mô phỏng cài đặt chương trình tính hệ số đa thức như cách trên:

```
Procedure tinh_hso;
```

```
Var A:array[1..n] of integer;

i:integer;

Begin

A[0]:=1;

A[n]:=1;

i:=1;

While ( i<= n ) do

begin

A[i]:= A[i-1 ]*(n - i + 1)/i ;

i:=i+1;

end;

End.
```

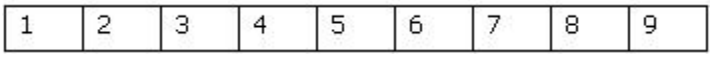
Một cách tương tự bạn có thể khai triển (x-y)n.

46 .**Phép đối xứng gương và các thao tác**

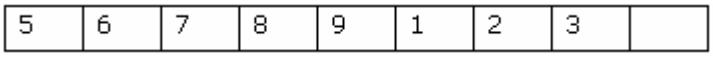
Tác giả: Nguyễn Xuân Huy

Bài toán: Người ta dùng N khối bê tông có kích thước bằng nhau để lát một đường băng cho máy bay. Sau khi đặt xong, người ta mới phát hiện ra rằng các khối đã đặt không đúng như bản thiết kế. K khối đầu tiên phải được chuyển về cuối đường băng. Để đặt lại, người ta dùng một cần trục có sức nâng 1 tấm bê tông mỗi lần và một giá đỡ phụ trên mặt đất để đặt tạm tấm bê tông gỡ ra khỏi đường băng. Hãy xác định một phương án giải quyết đỡ tốn kém với lưu ý rằng vị trí của đường băng sau khi sửa không được thay đổi.

Để hiểu rõ yêu cầu của bài toán xét một ví dụ với N=9 và k=4. Trong hình 1, (a) là phương án đặt sai, (b) là phương án đúng theo bản thiết kế. Theo phương án (b), đầu tiên phải đặt lần lượt các tấm bê tông 5,6,7,8 và 9, sau đó đặt tiếp các tấm 1,2,3 và 4.



(a)phương án đặt sai



Giá đỡ phụ

(a) Bản thiết kế: Phương án đúng

Hình 1. Giả thiết và kết luận của bài toán.

Vì chỉ có một giá đỡ phụ và phải tôn trọng yêu cầu không làm thay đổi vị trí cảu đường băng nên lời giải tự nhiên cho bài toán như sau.

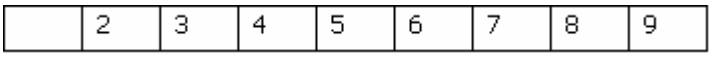
Thực hiện k lần, mỗi lần sẽ chuyển 1 khối bê tông từ đầu đường bằng về cuối đường băng theo 3 bước sau đây (xem hình 2):

Bước 1: Trục khối bê tông từ đầu đường băng ra giá đỡ.

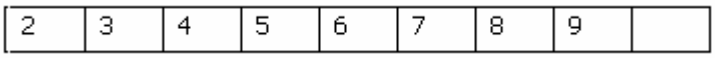
Bước 2: Dịch chuyển các khối bê tông về phía đầu đường băng: mỗi khối dịch lên 1 vị trí và đặt đúng vào vị trí của khối đã chuyển trước.

Bước 3: Trục khối bê tông từ giá đỡ vào vị trí cuối đường băng

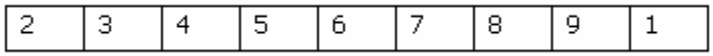
(a) cấu hình ban đầu



(b) Sau khi chuyển khối bê tông thứ nhất ra giá đỡ (Bước 1): 1 thao tác



c) Sau khi dịch chuyển các khối bê tông trên đường băng sang trái 1 vị trí (Bước 2): (N-1) thao tác



(d) Sau khi chuyển khối bê tông thứ nhất từ giá đỡ vào vị trí cuối đường băng (Bước 3): 1 thao tác.

Hình 2: Lần thực hiện thứ nhất của giải thuật tự nhiên.

Để so sánh với phương án cải tiến sẽ trình bày sau đây chúng ta hãy tính xem giải thuật tự nhiên đòi hỏi bao nhiêu thao tác. Ta hiểu mỗi lần trục một khối bê tông từ vị trí này đến vị trí khác là một thao tác. Dễ thấy bước 1 và 3 cần 1 thao tác cho mỗi bước. Bước 2 đòi hỏi (N-1) thao tác để dịch chuyển số bê tông còn lại trên đường băng. Vậy giải thuật tự nhiên cần 1+ (N-1)+1=N+1 thao tác cho mỗi lần chuyển một khối bê tông từ đầu đường băng về cuối đường băng. Tất cả có K khối bê tông cần chuyển, vậy tổng số thao tác sẽ là K(N+1). Theo ví dụ trên với N=9, K=4 ta tính được số thao tác cần thiết là K(N+1) = 4.10=40.

Khi K>N/2, thay vì chuyển K khối từ đầu về cuối đường băng ta có thể chuyển ngược N-K từ cuối về đầu đường băng. Thí dụ với N=9 và K=6, số thao tác cần thiết sẽ là (N-K)(N+1)=(9-6)10=30. Trước khi tìm một phương án tốt hơn ta hãy nhắc lại định nghĩa và liệt kê vài tính chất quan trọng của phép đối xứng gương.

Đối xứng gương của một dãy các phần tử:

Cho dãy m phần tử A=(a1,a2,.....,am). Phép đối xứng gương của dãy A cho ta dãy các phần tử của A theo thứ tự đảo. Ký hiệu A’ là “ảnh soi trong gương của A” ta có:

A’=(am,am-1,....a2,a1).

Thí dụ với A=(1,2,3,4) ta có A’=(4,3,2,1)

Phép đối xứng gương có hai tính chất quan trọng sau đây:

1. A”=A. Sau hai phép đối xứng gương liên tiếp ta thu lại được đối tượng ban đầu.

2. (AB)”=B”A”. Đối xứng gương của một dãy ghép nối từ hai A và B là một dãy được ghép nối từ hai dãy đối xứng gương của B và A.

Thí dụ, nếu A =(1,2,3,4), B=(5,6,7,8,9) thì
AB=(1,2,3,4)(5,6,7,8,9)=(1,2,3,4,5,6,7,8,9) và do đó.

(AB)’=(1,2,3,4,5,6,7,8,9)’Û(9,8,7,6,5,4,3,2,1). Mặt khác

B’A’Û(5,6,7,8,9)’(1,2,3,4)’=(9,8,7,6,5)(4,3,2,1)=(AB)’

Trở lại bài toán sửa đường băng, ta ký hiệu đoạn gồm K khối bê tông đầu tiên là A, đoạn còn lại của đoạn băng là B. Bài toán đặt ra khi đó sẽ được phát biểu gọn như sau: ABÞBA, trong đó AB là cấu hình đặt sai, còn BA là phương án thiết kế của đường băng. Bây giờ ta vận dụng hai tính chất của đối xứng gương để biến đổi AB thành BA. Ta có (A’B’)=B”A”=BA. Bí mật nằm ở đây. Theo công thức này ta sẽ thực hiện phương án sửa đường băng như sau:

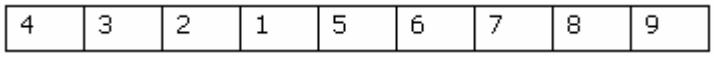
- Bước 1: Thực hiện đối xứng gương cho đoạn gồm K khối bê tông đầu đường băng mà ta ký hiệu là [1..K].

- Bước 2: Thực hiện đối xứng gương cho đoạn gồm các khối bê tông còn lại mà ta ký hiệu là [K+1..N]

- Bước 3: Thực hiện đối xứng gương cho toàn bộ đường băng.

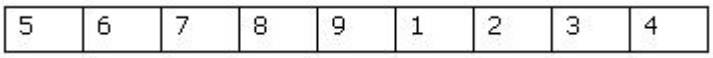
Hình 3 mô tả cách làm (mà ta gọi là giải thuật đối xứng gương) với thí dụ đã nêu, N=9, K=4

(a) Cấu hình ban đầu



(b) Đối xứng gương đoạn [1..4]

(c) Đối xứng gương đoạn [5..9]



(d) Đối xứng gương trền toàn bộ đường băng

Hình 3: Giải thuật đối xứng gương

Để thực hiện phép đối xứng trên gương đoạn gồm m phần tử với một giá đỡ ta đổi chỗ từng cặp phần tử thứ nhất và phần tử thứ m, sau đó đổi chỗ phần tử x và y được trợ giúp của giá đỡ d được thực hiện thông qua 3 thao tác như sau: Trước hết đưa x vào d, sau đó đưa y vào x, cuối cùng đưa d vào y.

Dưới đây là hai thí dụ thực hiện đối xứng gương đối với hai dãy (1,2,3,4) và (1,2,3,4,5)

- Cấu hình ban đầu:(1,2,3,4)

- Đổi chỗ 1 và 4:(4,2,3,1)

- Đổi chỗ 2 và 3: (4,3,2,1) cần 3 thao tác

Tổng cộng: 6 thao tác

- Cấu hình ban đầu: (1,2,3,4,5)

- Đổi chỗ 1 và 5:(5,2,3,4,1)

- Đổi chỗ 2 và 4: (5,4,3,2,1)

Tổng cộng: 6 thao tác

Ta để ý rằng khi m lẻ thì chỉ phải đổi chỗ [m/2] cặp, trong đó [x] biểu thị phần nguyên của số dương x. Tóm lại thực hiện đối xứng gương dãy m phần tử đòi hỏi 3[m/2] thao tác.

Bây giờ chúng ta tính tổng số thao tác của phép dịch chuyển ABÞBA. Để ý rằng A chứa K phần tử và B chứa (N - K) phần tử, ta có vì (A’B’)’=BA.

Thực hiện A’ đòi hỏi 3[K/2] thao tác.

Thực hiện B’ đòi hỏi 3[(N-K)/2] thao tác.

Thực hiện (...)’ trên toàn bộ đường băng đòi hỏi 3[N/2] thao tác.

Tổng cộng ta cần 3[K/2]+3[(N-K)/2] + 3[N/2]. Kết quả không phụ thuộc vào giá trị K. Để thấy hiệu quả của phương án cải tiến ta giả sử đường băng gồm N=1000 khối bê tông và cần chuyển K=500 khối từ đầu về cuối đường băng. Ngoài ra ta giả sử rằng mỗi ngày công trường có thể thực hiện được 100 thao tác trục các khối bê tông. Khi đó phương án thứ nhất cần:

K(N+1)=500.1001/100=5001/ngày » 13 năm 8 tháng và 20 ngày, và phương án thứ hai cần: 3N=3.1000/100=30 ngày.

Luyện tập: Hãy xây dựng phương án thực hiện phép chuyển ABCÞCBA.

47 .**Một ý tưởng ngông cuồng**

Tác giả: Nguyễn Lê Anh

Các em hẳn đã làm quen với bài toán cổ sau đây. Có một bác nông dân nuôi trâu để kéo cầy. Sau một ngày làm việc, tối về nhà bác cho chúng ăn cỏ. Con nào khỏe hơn thì ăn nhiều hơn còn con nào yếu hơn thì ăn ít hơn. Bác quan sát và nhận thấy có 3 loại trâu. Loại thứ nhất là gồm những con đứng là những con còn rất khỏe. Loại thứ hai là các nằm chúng đã thấm mệt sau một ngày làm việc. Loại thứ ba gồm những con đã già yếu. Bác nông dân nhẩm tính.

Trăm con trâu trăm bó cỏ. Con Đứng ăn 5 bó Nằm ăn 3. Ba con trâu Già chung nhau 1 bó. Hỏi mỗi loại có mấy con?

Một cậu hoc trò nghèo nghe được và nhẩm tính. Các em có biết cậu học trò nghèo ấy tính thế nào không?

Cậu bé học trò nghèo ấy tính thế này nhé.

Số trâu già không thể nhiều hơn 100 con vì thế ăn hết nhiều nhất là 100/3 bó cỏ tức là 33 bó cỏ. Như vậy số cỏ trâu đứng và trâu nằm ăn không ít hơn 67 bó. Số trâu đứng và số trâu nằm không thể ít hơn 67/5 con tức là không ít hơn 14 con. Như thế số trâu già không lớn hơn 86 con và chúng ăn nhiều nhất là 28 bó.

Nghiệm 1.

Nếu các con già ăn hết 28 bó thì số trâu già là 84 con. Số trâu đứng và số trâu nằm là 16 con và ăn hết 72 bó cỏ. Nếu cho mỗi con trâu nằm trong số 16 con này ăn thêm 2 bó nữa thì cả 16 con này ăn hết 80 bó, tức là cần thêm 8 bó nữa. Số này chính là số cỏ cần thêm để mỗi con nằm được ăn thêm 2 bó. Như vậy có 4 con trâu nằm, 12 trâu đứng, 84 con trâu già.

Nghiệm 2:

Nếu các con già ăn hết 27 bó thì số trâu già là 81 con, số trâu nằm là 11 con, số trâu đứng là 8 con.

Nghiệm 3:

Nếu các con già ăn hết 26 bó thì số trâu già là 78 con, số trâu nằm là 18, số trâu đứng là 4 con.

Nghiệm 4:

Nếu các con già ăn hết 25 bó thì số trâu già là 75 con, số trâu nằm là 25 con, số trâu đứng là 0 con.

Chỉ có các trường hợp chúng ta nêu trên thôi vì: nếu các con trâu già ăn hết 24 bó hay ít hơn, thì có ít hơn 72 con. Như thế số lượng trâu đứng và trâu nằm không ít hơn 28 con, và chúng ăn không ít hơn 28x3=84 bó cỏ. Như vậy trên thực tế số con già ăn không quá 100-84=16 bó cỏ. Và do đó số con già có ít hơn 16x3=54 con. Tức là số lượng con đứng và số con nằm không ít hơn 46 con, và số cỏ tối thiểu để chúng ăn hết là 46x3=138 bó. Điều này vô lý

Cách giải trên rất hay nhưng tìm ra lời giải ấy thế nào? Chúng ta rơi vào tuyệt vọng chăng. Các bài toán thì có rất nhiều, không có lẽ cứ mỗi khi có một bài toán thì chúng ta lại phải suy nghĩ để tìm ra lời giải cho chúng! Nào thử giải bài toán sau?Một cốc nước sôi dìm trong chậu nước đá đang tan hỏi sau bao lâu thì nhiệt độ trong lòng cốc nước là 30?C.? ... Quá đơn giản! Quá đơn giản! Hãy cứ lấy một cốc nước sôi dìm vào trong chậu nước đá đang tan và đo nhiệt độ của nó. Khi nào nhiệt độ trong lòng cốc nước là 30 ?C thì thông báo thời gian. Ơ-ric-ca ?Bài toán tự giải rồi thông báo kế quả?.

Với bài toán?trăm trâu, trăm cỏ?. Có 100 con, vừa trâu đứng, vừa trâu nằm vừa trâu già. Các con trâu này ăn theo khẩu phần con đứng ăn 5 bó cỏ, con nằm ăn 3 bó, 3 con già ăn một bó cỏ. Trâu của gia đình nào ăn hết đúng 100 bó cỏ thì số trâu mỗi loại của gia đình ấy là một lời giải cho bài toán. Như vậy chỉ cần có thật nhiều gia đình nôi trâu và có đủ số quan sát viên để kiểm tra số lượng cỏ trâu ăn có đúng bằng 100 bó không?

Không nhất thiết phải nuôi trâu thật, mà có thể mô phỏng việc nuôi và cho trâu ăn bằng máy tính.

Mỗi máy tính thay mặt cho một hộ nuôi trâu. Nó nhớ số trâu mỗi loại: x số con đứng, y số con nằm, z số con già, thì nó sẽ tính ngay ra số cỏ cần là 5x+3y+z/3. Mỗi quan sát viên cũng được thay bằng một máy tính nó kiểm tra xem 5x+3y+z/3 có bằng 100 hay không. Nếu bằng thì thông báo kết quả.

Như vậy mô hình hoá như sau:

Với mỗi trường hợp (x,y,z) mà {0<= x<=100; 0<=y<= 100, 0<=z<= 100 và x+y+z=100}

nếu 5x+3y+z/3=100 thì thông báo kết quả (x,y,z)

Chương trình.

for all 0<=x<= 100; 0<=y<= 100, 0<=z<= 100 do {Mọi khả năng nuôi}

if x+y+z=100 then {100 con trâu}

if 5x+3y+z/3=100 then print (x,y,z). {ăn hết 100 bó cỏ}

Như chúng ta thấy khó khăn không phải là tìm ra lời giải cho bài toán mà là tìm cho được đủ số gia đình nuôi trâu và quan sát viên. Số hộ nuôi trâu đồng thời nuôi và đồng thời cho trâu ăn. Các quan sát viên đồng thời xem và thông báo kết quả. Sơ đồ ý tưởng như vậy người ta gọi là lập trình song song. Công nghệ lập trình song song dựa trên khả năng của công nghệ có thể tạo ra một máy cái có rất nhiều máy tính con chúng sử lý song song và thông báo kết quả. Hiện nay công nghệ này đang rất phát triển.

* Các bạn học sinh cấp 2 đã làm quen với hệ phương trình thì lý luận như sau.

Giả sử X, Y, Z tương ứng là số trâu đứng, nằm, già. Khi đó chúng ta có quan hệ như sau. X+ Y+ Z =100; 5X+ 3Y+ Z/3=100

Biến đổi X+ Y + Z = 100; 15X + 9Y + Z = 300 -> 14X+8Y=200 -> 7X+4Y=100.

X phải chia hết cho 4 và X<100/7. Nhu+ va^.y X< 15 va` chi? co' the^? nha^.n ca'c gia' tri. 0, 4, 8, 12.

Chúng ta tìm được 4 nghiệm như sau (X, Y, Z) =(0, 25, 75) (X, Y, Z) =(4, 18, 78) (X, Y, Z) =(8, 11, 81) (X, Y, Z) =(12, 4, 84)

** Ngày nay người ta không những mô phỏng các bài toán kỹ thuật, kinh tế trên máy tính như dự báo thời tiết, hay dự báo cổ phiếu giá thị trường chứng khoán, mà ngay cả thử các loại vũ khí cũng thử trên máy tính. Cho mổ thử bom nguyên tử trong máy tính là bước mở màn cho một cuộc chạy đua vũ trang không tiếng động.

48 .**Tản mạn về suy luận lập trình**

Tác giả: Nguyễn Lê Anh

Nguyên lý tuần tự

Các máy tính mà chúng ta hiện đang sự dụng tại các gia đình lại dựa vào một nguyên lý mà người ta gọi là nguyên lý tuần tự. Có thể hình dung như sau. Chỉ cần một hộ nuôi trâu thôi. Nhưng hộ ấy nuôi lần lượt hết lứa trâu này đến lứa trâu khác. Mỗi lứa có đúng 100 con vừa trâu đứng, vừa trâu già, vừa trâu nằm. Và chỉ cần 1 quan sát viên mà thôi.

Thuật toán

Liệt kê lần lượt các trường hợp. Mỗi trường hợp được kiểm tra xem có thoả mãn điều kiện hay không. Nếu có thì thông báo kết quả.

Mọi sử lý đều gồm 2 phần.

Phần A. Liệt kê các trường hợp.

Phần B. Xử lý cho mỗi trường hợp.

Liệt kê các khả năng x+y+z=100 với x>=0, y>=0, z>=0 có thể thực hiện theo trình tự như sau như sau.

Chọn x tăng dần từ 0 đến 100. Nếu đã chọn x rồi thì khả năng của y là từ 0 tới 100-x. Nếu đã chọn x, y rồi thì z=100-(x+y). Như thế bài toán liệt kê được lệnh hóa như sau.

```
for x=0 to 100 do

for y=0 to 100-x do

begin

z=100-x-y;
```

{mọi khả năng có thể x+y+z=100 mà x>=0;y>=0;z>=0}

end;

Phần liệt kê cho bài toán □100 trâu 100 cỏ□ như vậy là rõ ràng. Phần kiểm tra mỗi trường hợp được thực hiện như sau.

Cứ mỗi trường hợp (x,y,z) chúng ta kiểm tra đẳng thức

5x+3y+(100-x-y)/3=100. Nếu thoả mãn chúng ta thông báo kết quả.

Như vậy chúng ta có đoạn trình sau.

Var x,y,z:integer; {Báo cho máy tính các biến cần sử dụng}

begin

for x:=0 to 100 do for y:=0 to 100-x do {Liệt kê lần lượt mọi trường hợp}

if 5*x+3*y+(100-x-y)/3=100 then {Kiểm tra xem có thỏa mãn điều kiện}

writeln(x,y,z); {Nếu có thì in kết quả}

end.

Thời gian tính

Trở lại bài toán □trăm trâu, trăm cỏ□. Có hai cách liệt kê thể hiện bằng hai khúc trình sau đây.

Thuật giải 1:

Var x,y,z:integer; {Báo cho máy tính các biến cần sử dụng}

begin

for x:=0 to 100 do for y:=0 to 100-x do {Liệt kê lần lượt mọi trường hợp}

if 5*x+3*y+(100-x-y)/3=100 then {Kiểm tra xem có thỏa mãn điều kiện}

writeln(x,y,z); {Nếu có thì in kết quả}

end.

Thuật giải 2:

```
Var x,y,z:integer; {Báo cho máy tính các biến cần sử dụng}
Begin style='mso-tab-count:5'> {Liệt kê lần lượt mọi trường hợp}
for x:=0 to 100 do for y:=0 to 100 do for z:=0 to 100 do
                                        {Kiểm tra xem có thỏa mãn điều kiện}
if (x+y+z=100) and (5*x+3*y+(100-x-y)/3=100) then
writeln(x,y,z); {Nếu có thì in kết quả}
end.
```

Hai thuật giải trên khác nhau thế nào.

Cả hai thuật giải đều cùng tìm ra kết quả như nhau, nhưng thời gian tính cáo khác nhau. Thuật giải thứ nhất nhanh hơn. Thật vậy. Theo như thuật giải I cần phải liệt kê 5000 trường hợp. Đối với thuật giải II thì cần tới 1000000 trường hợp. Nếu bài toán □trăm trâu, trăm cỏ□ mà đổi thành □nghìn trâu, nghìn cỏ□ thì các em có đủ kiên nhẫn chờ thuật giải II in ra kết quả không? (xin hãy cầm lấy búa!).

Như vậy điều quan trọng để giải một bài toán không phải là lời giải, mà là tìm ra một thuật giải có thể thực hiện được trong một thời gian không lâu lắm.

Làm thế nào để tìm được một thuật giải nhanh. Chúng ta sẽ đề cập tới trong các số báo tiếp theo.

Bài tập về nhà

1. Tìm tất cả các tam giác vuông có các cạnh là các số nguyên và tổng của chúng nằm trong khoảng từ 1000 đến 1500.

2. Trong số các tam giác có các cạnh là các số nguyên với chu vi 100 thì những tam giác nào có diện tích lớn hơn 450.

3. Cho biết vòng tròn tâm ở gốc toạ độ và đi qua điểm nguyên (77,33). Tìm tất cả các điểm nguyên khác mà nó đi qua.

49 .**Số nguyên tố lớn nhất**

Tác giả: Bùi Việt Hà

Các bạn có thấy ngạc nhiên không? Có đấy. Ngay từ những năm học phổ thông cấp 2, chắc các bạn ai cũng biết rằng số các số nguyên tố là vô hạn. Euler là người đầu tiên đã chứng minh được định lý đẹp tuyệt vời đó. Như vậy với mọi số tự nhiên N sẽ phải tồn tại số nguyên tố lớn hơn N. Thế thì "Số nguyên tố lớn nhất" ở đây là gì? Chẳng nhẽ lại có sự nhầm lẫn nào chăng, hay tác giả của bài viết này không biết gì về định lý Euler đó?

Không đâu. Các bạn hãy chú ý nhé: Định lý Euler về sự tồn tại vô hạn các số nguyên tố chỉ chứng minh sự tồn tại của các số nguyên tố "lớn" này nhưng không hề chỉ ra cụ thể vậy thì các số nguyên tố này là những số nào? Do vậy câu hỏi đặt ra ở bài viết này hoàn toàn có ý nghĩa: Số nguyên tố lớn nhất mà con người có thể viết ra được trên giấy (hay còn gọi là hai năm rõ mười) là số nào vậy? Điều mà chúng ta cần là chỉ ra bằng xương bằng thịt, tức là bằng phép toán số học và bằng các số 0, 1,.., 9 để chỉ rõ số nguyên tố đó, chứ không phải bằng những ký hiệu "mập mờ" như n, k, i, j, p, q nào đó.

Như vậy đã từ lâu, con người luôn có mong muốn tìm và chỉ ra các số nguyên tố lớn và bài toán ghi lại kỷ lục về số nguyên tố lớn nhất đã theo đuổi con người hàng trăm năm nay. Vì sao các số nguyên tố lớn lại mang lại niềm say mê như vậy đối với con người? Bên cạnh bản năng của tính "tò mò", các số nguyên tố lớn trên thực tế đóng vai trò rất quan trọng trong nhiều ngành khoa học, đặc biệt là tin học. Chúng tôi mong rằng sẽ có dịp quay trở lại với các bạn trẻ về đề tài đầy hấp dẫn này trong các số sắp tới của Tin học & Nhà trường.

Mersenne

Còn bây giờ chúng ta quay trở lại với những số nguyên tố lớn. Có lẽ người khởi đầu cho sự tìm kiếm đầy chông gai và hấp dẫn này là Marin Mersenne (1588-1648). Mersenne là một nhà toán học người Pháp, ông là người đầu tiên đưa ra các số Mp = 2p -1 mà sau này người ta gọi nó là các số Mersenne. Năm 1644, trong một quyển sách ông đã phát biểu hai điều gây chấn động giới toán học đương thời:

- Thứ nhất, ông khẳng định rằng các số Mp với p = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257 là các số nguyên tố và với p <=257 chỉ có các số Mp với p như trên là số nguyên tố.

- Thứ hai, ông khẳng định "Với các số tự nhiên có 15 hay 20 chữ số, có thể mất cả một đời người mà vẫn không thể kiểm tra được xem số đó có là số nguyên tố hay không".

Những khẳng định trên của Mersenne đã làm đau đầu bao nhiêu thế hệ các nhà toán học. Khẳng định đầu tiên về các số nguyên tố Mersenne đã được coi là bí ẩn trong suốt gần 300 năm. Còn về khẳng định thứ hai, quả thật nó sẽ còn đúng... vĩnh viễn... nếu như con người không phát minh ra máy tính điện tử để thực hiện các phép thử như Mersenne đã nói.

Về giả thuyết thứ nhất của Mersenne

Vào thời gian của Mersenne, người ta đã biết được các số Mp = 2p -1 là nguyên tố với p=2, 3, 5, 7, 13 và 19. Còn các số tiếp theo như Mersenne đã khẳng định?

Mãi đến năm 1772, Euler chứng minh được M31 =231-1 =2147483647 (số chỉ có 10 chữ số!) là số nguyên tố.

Gần 100 năm sau nữa, năm 1876, Lucas đã chứng minh rằng M127 là số nguyên tố (bỏ qua 67). Đến lúc này người ta đã nghĩ rằng Mersenne hoàn toàn đúng.

Năm 1883, Pervushin đã chứng minh rằng M61 là số nguyên tố! Lúc này do quá tin vào Mersenne, có người đã khẳng định rằng xưa kia, Mersenne đã ghi nhầm 61 thành 67!

Thế nhưng mọi sự đã diễn ra không như mong đợi.

Năm 1911, R.E. Powers chứng minh được rằng M 89 là số nguyên tố. Ba năm sau cũng Powers phát hiện thêm M 107 là số nguyên tố. Hai số này hoàn toàn không nằm trong giả thuyết của Mersenne. Và cuối cùng năm 1924, M. Kraitchik chứng minh được M 257 là một hợp số, không phải là số nguyên tố. Mặc dù chứng minh của Kraitchik có sai sót, nhưng sau này với sự giúp đỡ của máy tính, khẳng định của Kraitchik đã được chứng minh là đúng.

Như vậy về cái gọi là "giả thuyết Mersenne" sau gần 300 đã được làm sáng tỏ hoàn toàn.

Các số nguyên tố lớn do con người tính tay

Số M17 = 217-1 = 131071 do Pietro Cataldi tìm ra năm 1588.

Số M19 = 219-1 = 524287 cũng do Pietro Cataldi tìm ra năm 1588.

Số M31 = 231 -1 = 2147483647 do Euler tìm ra năm 1772.

Số (259-1)/179951 = 3203431780337 do Landry tìm ra năm 1867.

Số M127 = 2127 -1 = 170141183460469231731687303715884105727 do Lucas tìm ra năm 1876. Số Lucas này đã được ghi vào kỷ lục số nguyên tố lớn nhất trong suốt 75 năm.

Năm 1951, Ferrier tìm ra được một kỷ lục mới là số nguyên tố có 44 chữ số:

(2148 +1) /17 = 20988936657440586486151264256610222593863921

Kỷ lục về số nguyên tố lớn do con người dùng tay tìm ra được có lẽ tới đây kết thúc.

Năm 1945, chiếc máy tính điện tử đầu tiên ra đời... và thế giới đã thay đổi. Với sự giúp đỡ của máy tính, con người đã liên tục đột phá kỷ lục về các số nguyên tố lớn. Danh sách các số kỷ lục này ngày một dài ra.

Với sự giúp đỡ của máy tính

Miller và Wheeler có lẽ là những người đầu tiên sử dụng máy tính tấn công vào bức tường kỷ lục về các số nguyên tố. Năm 1951, hai ông này đã dùng máy tính kiểm tra và tìm ra hàng loạt số nguyên tố dạng kM127 + 1. Hai ông đã tìm ra các số nguyên tố từ dãy này với k = 114, 124, 388, 408, 696, 738, 744, 780, 934, 978. Cũng năm đó hai ông đã đạt một kỷ lục xuất sắc khi tìm ra được số nguyên tố có 79 chữ số là 180(M127)2 + 1.

Trong năm sau đó 1952, Robinson đã viết một chương trình ngoạn mục và đã liên tục tìm ra tiếp 5 kỷ lục nữa, đó là các số M521, M607, M1279, M2203, 2281.

Danh sách các số nguyên tố lớn do máy tính tìm ra cứ ngày một lớn lên và dài ra mãi. Chúng ta hãy cùng nhìn vào danh sách kỷ lục đó.

Các kỷ lục do máy tính tìm ra

Số nguyên tố	Số chữ số	Năm tìm ra	Máy tính	Tác giả
$180(M_{127})^2+1$	79	1951	EDSAC1	Miller & Wheeler
M_{521}	157	1952	SWAC	Robinson (30-1)
M_{607}	183	1952	SWAC	Robinson (30-1)
M_{1279}	386	1952	SWAC	Robinson (25-6)
M_{2203}	664	1952	SWAC	Robinson (7-10)
M_{2281}	687	1952	SWAC	Robinson (9-10)
M_{3217}	969	1957	BESK	Riesel
M_{4423}	1332	1961	IBM7090	Hurwitz
M_{9689}	2917	1963	ILLIAC 2	Gillies
M_{9941}	2993	1963	ILLIAC 2	Gillies
M_{11213}	3376	1963	ILLIAC 2	Gillies
M_{19937}	6002	1971	IBM360/91	Tuckerman
M_{21701}	6533	1978	Cyber 174	Noll & Nickel
M_{23209}	6987	1979	Cyber 174	Noll
M_{44497}	13395	1979	Cray 1	Nelson & Slowinski
M_{86243}	25962	1982	Cray 1	Slowinski
M_{132049}	39751	1983	Cray X-MP	Slowinski
M_{216091}	65050	1985	Cray X-MP	Slowinski
$391581*2^{216193}-1$	65087	1989	Amdahl 1200	Amdahl Six
M_{756839}	227832	1992	Cray-2	Slowinski & Gace
M_{859433}	258715	1994	Cray C90	Slowinski & Gace
$M_{1257787}$	378632	1996	Cray T94	Slowinski & Gace
$M_{1398269}$	420921	1996	Pentium (90 Mhz)	Armengaud, Woltman, ...
$M_{2976221}$	895932	1997	Pentium (100 Mhz)	Spence, Woltman, ...
$M_{3021377}$	909526	1998	Pentium (200 Mhz)	Clarkson, Woltman, Kurowski, ...

Có rất nhiều giai thoại hấp dẫn và ly kỳ về danh sách này và về các máy tính và tác giả của chúng nữa.

Có lẽ Robinson đã sung sướng phát điên lên mất vì trong một ngày 30-1-1951 đã tìm ra được 2 kỷ lục, đó là các số M521 và M607.

Còn Hurwitz sử dụng máy tính IBM7090 sau khi đã biết chắc chắn rằng số M4253 là số nguyên tố. Đang chuẩn bị chờ đợi máy in ra kết quả cuối cùng thì máy lại thông báo kết quả của số M4423 trước đúng 1 giây! Hóa ra phần bộ nhớ in kết quả của số M4253 đã bị trục trặc nên in kết quả chậm. Như vậy kỷ lục về số M4253 đã bị xóa ngay trước khi được công bố trên toàn thế giới.

Như các em nhìn thấy từ bảng trên, số nguyên tố lớn nhất M3021377 có tất cả 909526 chữ số. Như vậy chúng ta đã ở gần ngưỡng cửa của việc tìm ra số nguyên tố có 1 triệu chữ số đầu tiên. Quĩ Điện tử Cấp tiến (FFF - Electronic Frontier Foundation) đã đưa ra giải thưởng 50000$ cho ai tìm ra số nguyên tố đầu tiên có 1 triệu chữ số. Theo các dự đoán của nhiều chuyên gia, kỷ lục mới này sẽ được tìm ra vào tháng giêng năm 1999. Chúng ta hãy chờ xem.

Số nguyên tố lớn nhất

Bây giờ chắc các bạn đã sốt ruột. Vậy thì số nguyên tố lớn nhất hiện nay là số nào. Xin thông báo ngay đây:

Ngày 1 tháng 6 năm 1999, Nayan Hajratwala cùng với nhóm nghiên cứu GIMS đã tìm ra số nguyên tố lớn nhất thế giới mà con người biết đến, đó là số M6972593 = 2 6972593 -1. Số này có 2098960 (hai triệu chín mươi tám ngàn chín trăm sáu mươi) chữ số. Như vậy kỷ lục về số nguyên tố có 1 triệu chữ số đã bị phá vỡ. Và đây cũng là số nguyên tố Mersenne thứ 38 mà con người biết đến. Nayan Hajratwala đã sử dụng máy tính Pentium II, 350 MHz chạy trong 111 ngày (hay tương đương liên tục trong 3 tuần lễ) để tìm ra được số nguyên tố này. Tuy nhiên, Nayan Hajratwala tham gia trong đề án GIMS đã sử dụng kết quả trung gian của rất nhiều các máy tính khác nằm rải rác trên toàn cầu thông qua mạng Internet. Hy vọng chúng tôi sẽ được trình bày kỹ với các bạn về đề án GIMS thú vị này trong số báo sau.

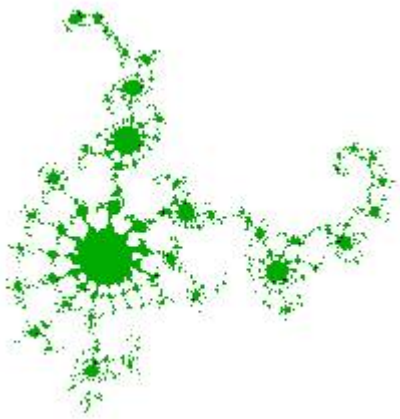
Và trong tương lai

Danh sách kỷ lục số nguyên tố đã và đang lôi cuốn hàng ngàn nhà Toán học và Tin học tham gia vào quá trình tìm kiếm đầy hấp dẫn này. Các bạn trẻ Việt nam cũng không nên đứng ngoài cuộc. Các bạn hoàn toàn có thể tham gia, nghiên cứu, tìm hiểu các thuật toán, các chương trình để tìm ra các số nguyên tố kỷ lục. Hiện tại quĩ FFF đã trao giải thưởng 100000$ cho người tìm ra số nguyên tố đầu tiên có 10 triệu

chữ số. Liệu kỷ lục mới này khi nào sẽ bị phá? Năm 2001 hay 2000? Tất cả còn đang ở phía trước và phụ thuộc rất nhiều vào chính các bạn trẻ, trong đó có người Việt nam chúng ta. Chúc các bạn có nhiều hoài bão lớn, nhiều mong ước lớn trong việc học tập, tìm hiểu và nghiên cứu khoa học của mình.

50 .FRACTAL là gì?

1. Thế giới Fractal



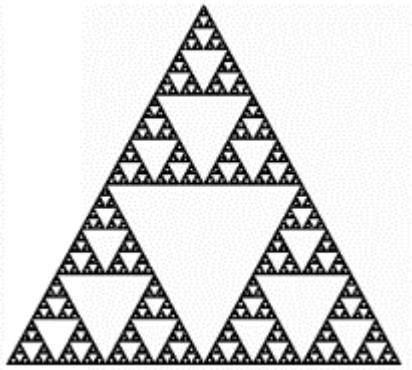
Các bạn hãy nhìn vào "hình vẽ" dưới đây, hãy quan sát thật kỹ, chắc chắn bạn sẽ thấy nhiều điều kỳ lạ và thú vị. Đó chính là các hình Fractal do nhà bác học người Mỹ là benoit Mandelbrot phát hiện ra lần đầu tiên vào năm 1980. Sự kỳ lạ và những tính chất đặc biệt của hình này đã thu hút sự quan tâm của rất nhiều các nhà Toán học đương thời. Các ứng dụng thực sự của Fractal đã đi liền ngay sau đó. Fractal là hình ảnh mô phỏng của rất nhiều hiện tượng thiên nhiên, môi trường, xã hội. Khoa học về các Fractal và các hệ động đã phát triển nhanh chóng, trở thành một trong những khoa học mũi nhọn của các ngành Toán học, Cơ học, Vật lý học và Tin học hiện nay. Chúng ta hãy hình dung các bờ biển, các đường phân hoạch miền sinh thái, đường đi của các cơn bão, gió xoáy, các vành đai động đất,... Tất cả chúng đều có hình dáng của những Fractal (!). Hôm nay tôi muốn mời các bạn cùng tìm hiểu sâu hơn một vài đặc tính cơ bản của cái gọi là hình học Fractal, các tính chất đơn giản và thuật toán sinh ra chúng.

2. Hệ thống "Chaos"

Ta hãy xét một ví dụ đầu tiên của Fractal từ một "Chaos game" (trò chơi hỗn loạn). Trên mặt phẳng cho 3 đỉnh của một tam giác đều A, B, C và một điểm P0. Xuất phát từ P0, ta chọn ngẫu nhiên một trong 3 đỉnh A, B, C và gọi P1 là trung điểm của P0 và đỉnh đã được chọn đó. Bây giờ lấy P1 làm vị trí xuất phát, chọn ngẫu nhiên một trong 3 đỉnh A, B, C và ta thu được điểm thứ hai P2. Quá trình trên được lặp lại liên tiếp và ta thu được dãy điểm P0, P1, P2,... Việc tạo ra dãy này, còn được gọi là quá trình lặp liên tục (Iteration Process), là một trong những mô hình chính cho việc nghiên cứu các hệ hình học Fractal. Dãy điểm P0, P1, P2,... được gọi là quĩ đạo lặp của điểm P0. Một điều vô cùng kỳ lạ là của hình dáng của quĩ đạo này trên mặt phẳng sẽ không phụ thuộc vào cách chọn điểm ban đầu P0. Hình dưới đây mô phỏng

hình được tạo bởi quĩ đạo trên khi số bước lặp đủ lớn. Đó chính là một hình Fractal đầy bí ẩn mà chúng ta có dịp xem xét hôm nay.

Các quá trình "Chaos Games" tương tự sẽ đều tạo ra các hình ảnh Fractal gần giống như trên.



Nếu các bạn quan sát kỹ các hình ảnh Fractal chúng ta sẽ phát hiện ra những đặc điểm quan trọng sau về các hình Fractal:

1. Tính tương tự. Tính tương tự được thể hiện ở chỗ nếu ta phóng to một phần nhỏ của hình ta sẽ thu được chính hình ảnh của hình ban đầu. Thật vậy chúng ta hãy quan sát kỹ góc phía trên của tam giác trong hình trên. Nếu phóng to ra ta thu được chính hình ảnh ban đầu của hình.

2. Khái niệm về độ "rộng" của các hình fractal. Một trong những đặc thù kỳ lạ nhất của hình học Fractal chính là khái niệm về chiều, hay "độ rộng" của chúng. Nếu như với các đường thẳng, đường cong tự nhiên ta có thể nói ngay đó là các hình 1 chiều vì trên các đường này chỉ có 1 chiều để đi lại, hay nói cách khác chúng có độ rộng bằng 0, các hình phẳng như hình chữ nhật, hình tròn, hình ellips là hình 2 chiều. Còn đối với các hình Fractal, ví dụ như hình vẽ trên thì chúng có số chiều là bao nhiêu. Câu hỏi đặt ra là tự nhiên. Một điều có thể hình dung ngay là các hình này phải có "độ rộng" rất đặc biệt. Chúng ta sẽ thấy ngay sau đây rằng quả thật khái niệm về chiều của các hình Fractal là rất đặc biệt và vô cùng thú vị.

3. Thế nào là Fractal

Vậy thế nào là một hình Fractal. Từ các nhận xét trên chúng ta có thể đưa ra ở đây một định nghĩa cho hình học Fractal.

Định nghĩa: Một hình được gọi là Fractal nếu nó có tính chất tương tự và có số chiều fractal lớn hơn số chiều topo của mình.

Hình có tính chất tương tự là hình có tính chất khi phóng to một phần không gian của mình sẽ thu được chính bản thân hình đó.

Số chiều topo là khái niệm chiều quen thuộc của hình học mà chúng ta đã quen.

Số chiều Fractal được đo bởi logarit của tỷ lệ phóng to đã được nêu ở trên.

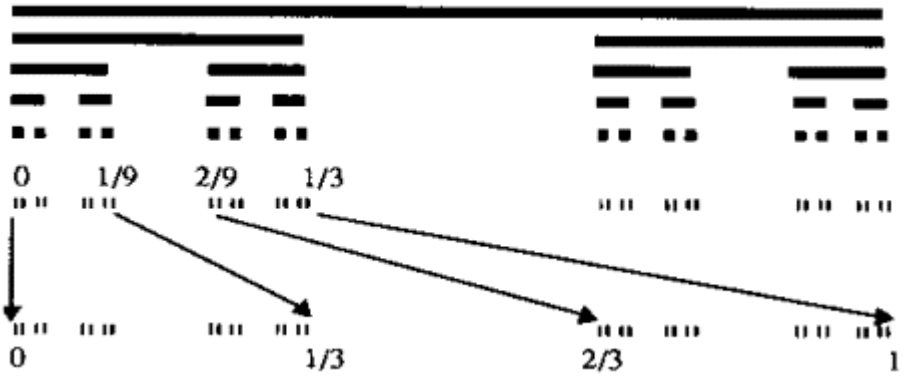
4. Các ví dụ

Để hiểu rõ hơn định nghĩa Fractal đã nêu trên, ta xét trong mục này một vài ví dụ của các Fractal điển hình.

Ví dụ 1: Tập hợp Cantor

Tập hợp Cantor theo định nghĩa là tập hợp thu được từ đoạn thẳng [0,1] sau khi liên tiếp bỏ đi các khoảng 1/3 nằm giữa các đoạn thẳng còn lại của tập này. Hình sau mô tả quá trình lược bỏ khoảng 1/3 và thu được tập Cantor.

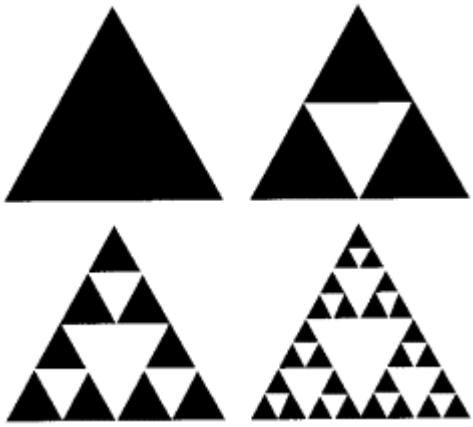
Gọi C là tập Cantor. Dễ thấy C được chia làm 2 phần: phần nằm trong đoạn [0, 1/3] và phần nằm trong đoạn [2/3, 1]. Nếu chúng ta áp dụng phép biến đổi L(x) = 3x lên đoạn [0,1/3] thì dễ dàng chứng minh được L sẽ ánh xạ phần tập C trên [0, 1/3] lên toàn bộ C. Thật vậy khi đó L sẽ ánh xạ phần [1/9, 2/9] lên [1/3, 2/3], [1/27, 2/27] lên [1/9, 2/9],... Do đó L sẽ ánh xạ mỗi khoảng trống nằm trong phần [0, 1/3] lên khoảng trống của C. Tương tự ta có ánh xạ L(x) = 3x -2 sẽ ánh xạ phần [2/3, 1] lên toàn bộ C. Bằng cách xét tương tự ta có thể thấy có thể sử dụng các ánh xạ tuyến tính để "phóng to" một miền nhỏ tuỳ ý của C lên toàn bộ C. Đó chính là tính chất tương tự của tập Cantor, một trong hai tính chất quan trọng nhất của hình học Fractal.



Chú ý rằng tại thời điểm xét ánh xạ tuyến tính như trên, nếu tập Cantor được chia thành 2n phần nhỏ thì ánh xạ tuyến tính sẽ có hệ số phóng đại là 3n.

Ví dụ 2: Tam giác Sierpinski

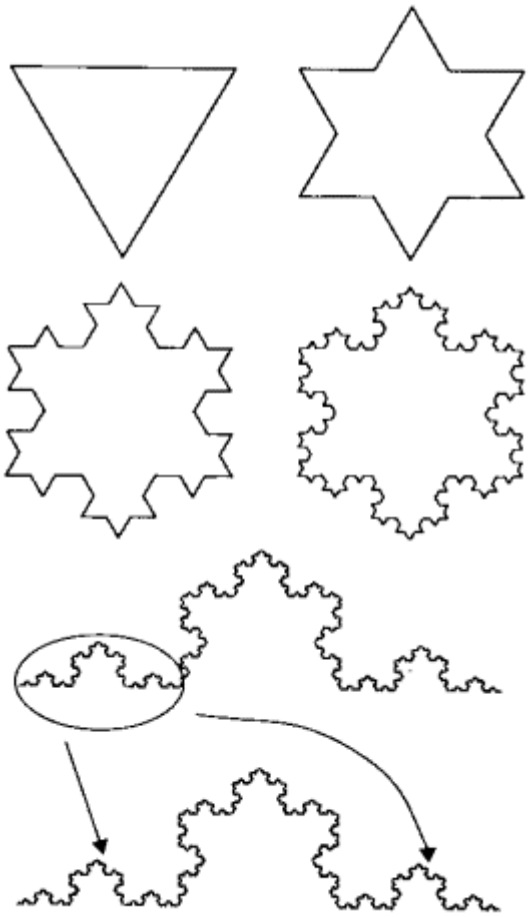
Tam giác Sierpinski đã được mô tả trong mục "Chaos games". Tuy nhiên tam giác Sierpinski có một cách khác mô tả như sau: Bắt đầu từ một tam giác đều chúng ta lần lượt khoét bỏ đi một tam giác con nằm chính giữa có kích thước đúng bằng một nửa tam giác ban đầu. Quá trình khoét bỏ trên kéo dài đến vô tận và hình thu được còn lại chính là tam giác Sierpinski. Hình vẽ sau mô tả các bước ban đầu khi xây dựng tam giác Sierpinski.



Suy luận hoàn toàn tương tự như trong trường hợp tập Cantor, tam giác Sierpinski cũng có tính chất tương tự như tập Cantor: Tồn tại ánh xạ tuyến tính phóng to một phần nhỏ bất kỳ của tam giác Sierpinski lên toàn bộ tam giác này. Đối với tam giác Sierpinski dễ thấy tại thời điểm đang xét nếu tam giác Sierpinski được chia thành 3n tập con bằng nhau thì hệ số phóng to sẽ là 2n.

Ví dụ 3: Bông tuyết Koch

Ngược lại so với tập Cantor và tam giác Sierpinski, bông tuyết Koch được xây dựng bằng cách bổ sung lần lượt các hình. Xuất phát từ các cạnh của một tam giác đều, chúng ta chia mỗi cạnh làm 3 phần bằng nhau, lấy mỗi phần này làm một cạnh chúng ta xây dựng ra "phía ngoài" các tam giác đều. Quá trình bổ sung như vậy sẽ lặp lại đến vô tận. Hình vẽ dưới đây mô phỏng 4 bước của quá trình bổ sung để tạo ra bông tuyết Koch. Đường bao quanh hình còn được gọi là đường cong Koch.



Tính tương tự của bông tuyết Koch có thể dễ dàng kiểm chứng. Hình vẽ dưới đây là một gợi ý tốt cho các bạn tự thiết lập lấy các ánh xạ phóng to cho tính tương tự của hình này.

Nhận xét rằng tại thời điểm đang xét bông tuyết được chia thành 4 phần nhỏ, mỗi phần có thể được phóng to với hệ số 3 để trùng khít với hình ban đầu.

Ta sẽ chứng minh ở đây một tính chất đặc biệt lạ lùng của bông tuyết Koch: Bông tuyết Koch có diện tích hữu hạn nhưng có chu vi vô hạn.

Thật vậy tính hữu hạn của bông tuyết Koch là hiển nhiên vì ta luôn nhìn thấy toàn bộ chúng trong tầm mắt. Bây giờ ta hãy tính thử chu vi của nó xem sao. Giả sử C0, C1, ... là số các cạnh của bông tuyết trong quá trình xây dựng như đã nêu trên. Dễ thấy:

C0 = 3

C1 = 4.3

C2 = 4.12 = 42.3

.....

Cn = 4C n-1 = 4 n-1 .3

Gọi Lk là độ dài một cạnh của bông tuyết Koch tại thời điểm lặp k. Giả sử tam giác ban đầu có cạnh bằng 1. Khi đó dễ thấy Lk = L k-1 /3 = 1/3 k .

Do đó chu vi của đường cong Koch là giới hạn của dãy sau:

P k = C k .L k = 4 k .3.1/3 k = (4/3) k .3

Rõ ràng là độ dài sẽ tiến đến vô hạn.

5. Cơ sở toán học của Fractal

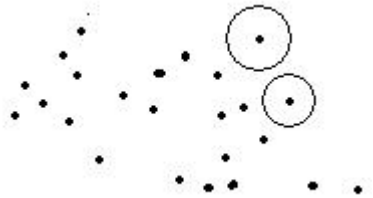
Trong phần này chúng ta sẽ có được những định nghĩa chính xác hơn của các hình Fractal. Chúng tôi sẽ cố gắng trình bày chúng một cách đơn giản nhất để tất cả các bạn đều có thể hiểu chúng.

Khái niệm chiều Topo

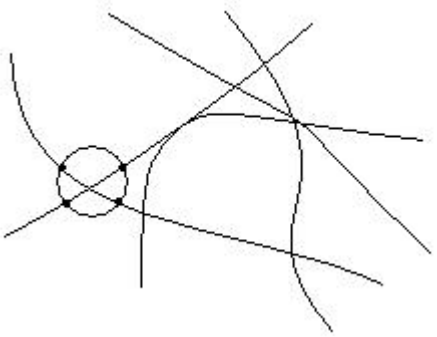
Khi nhắc đến số chiều topo (hay số chiều tự nhiên) của một đường thẳng hoặc đường cong, ta khẳng định ngay chúng là 1 chiều. Các hình phẳng như hình chữ nhật, tròn sẽ có số chiều là 2, các hình khối như hộp, khối tròn sẽ là các vật thể 3 chiều. Để có thể xác định chính xác ta có định nghĩa chiều topo như sau:

Tập hợp S được gọi là có chiều topo 0 nếu với mọi điểm của S tồn tại một lân cận đủ nhỏ của điểm này sao cho S không có giao điểm với viền (khung) của lân cận này. Ta hiểu lân cận ở đây theo nghĩa nếu S là miền trên mặt phẳng thì lân cận của điểm là một hình tròn lấy điểm này làm tâm, nếu S là miền trong không gian thì lân cận của một điểm là hình cầu lấy điểm này làm tâm, khái niệm đủ nhỏ hiểu là các lân cận này có thể chọn với bán kính nhỏ tuỳ ý.

Tập hợp S được gọi là có chiều topo k nếu mọi điểm của S đều có lân cận đủ nhỏ và giao của S với viền (khung) của lân cận này có chiều topo k-1.



Hình có chiều topo 0



Hình có chiều topo 1

Xét chiều topo của các fractal đã biết:

- Tập hợp Cantor, dễ thấy có chiều topo 0. Thật vậy nếu x là một điểm trên tập Cantor, khi đó với mọi n đủ lớn, từ cách dựng ta có ngay x sẽ phải nằm trong một đoạn có dạng [k/3n, (k+1)/3n] và kề với hai khoảng rỗng độ dài 1/3n xung quanh. Do đó lân cận bán kính 1/3n của x sẽ có viền không giao với tập Cantor. Do đó theo định nghĩa tập Cantor có chiều topo 0.

- Tam giác Seirpinski có chiều topo 1. Thật vậy mọi lân cận của mọi điểm của hình này có viền giao với chính nó chỉ tại hữu hạn điểm (bạn hãy chứng minh nhận xét này).

- Bông tuyết Koch có chiều topo 1. Điều này dễ dàng chứng minh bởi thực chất bông tuyết Koch là một đường cong liên tục khép kín trên mặt phẳng.

<u>Khái niệm Tương tự (Self Similarity)</u>

Trong các phần trên của bài viết chúng ta đã nhắc đến khái niệm tương tự của các hình Fractal. Bây giờ chúng ta sẽ xét các định nghĩa này một cách chính xác hơn.

Xét tập các điểm trên mặt phẳng dạng $p = \begin{pmatrix} x \\ y \end{pmatrix}$ ở đây (x,y) là toạ độ mặt phẳng của điểm p. Một ánh xạ tuyến tính (hay còn gọi biến đổi affine) là một phép biến đổi A trên mặt phẳng có dạng sau:

$$A\begin{pmatrix} x \\ y \end{pmatrix} = \beta\begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

Hằng số β được gọi là Hệ số tương tự của phép biến đổi trên.

Định nghĩa: Tập S được gọi là tương tự affine (hay đơn giản là tương tự) nếu S có thể chia thành k tập con, mỗi tập con có thể biến đổi tuyến tính với cùng một hệ số tương tự T để thu được toàn bộ S.

Dễ thấy rằng các hình đơn giản như đoạn thẳng, hình vuông, chữ nhật, hình tròn,... đều là những hình có tính chất tương tự. Chẳng hạn với đoạn thẳng, đoạn này có thể chia thành N phần và với phép biến đổi tuyến tính với hệ số tương tự N. Hình vuông được chia thành N2 phần nhỏ với hệ số biến đổi N.

Các hình Fractal được mô tả ở trên đều là các tập tương tự theo định nghĩa trên.

<u>Khái niệm chiều Fractal (chiều tương tự)</u>

Bây giờ ta định nghĩa một khái niệm quan trọng nhất là số chiều fractal.

Định nghĩa: Giả sử S là tập tương tự được chia thành k phần với hệ số biến đổi tương tự T. Khi đó số chiều fractal của S được định nghĩa bởi công thức:

$$D = \frac{\log(k)}{\log(T)} = \frac{\log(\text{số phần})}{\log(\text{hệ số tương tự})}$$

Ta hãy tính chiều fractal của một vài hình quen thuộc:

- Đoạn thẳng: số phần chia là N, hệ số phóng to là N, do đó số chiều fractal của đoạn thẳng sẽ là $D = \frac{\log(n)}{\log(n)} = 1$

- Hình vuông: số phần chia là N2, hệ số phóng to là N, do đó $D = \frac{\log(N^2)}{\log(N)} = 2$

- Tập Cantor, số phần chia ở mỗi giai đoạn là 2n, hệ số phóng to cho mỗi phần là 3n. do đó chiều fractal của hình này sẽ bằng $D = \frac{\log 2^n}{\log 3^n} = \frac{n \log 2}{n \log 3} = \frac{\log 2}{\log 3} = 0.6309297536...$

- Tam giác Sierpinski, số phần chia là 3, hệ số phóng to là 2, do đó chiều fractal của hình này sẽ là $D = \frac{\log 3^n}{\log 2^n} = \frac{\log 3}{\log 2} = 1.584962501...$

- Tương tự trên, chiều fractal của bông tuyết Koch sẽ là $D = \frac{\log 4}{\log 3} = 1.261859507...$

Các bạn đang được chứng kiến những điều hết sức mới mẻ lạ lùng của khái niệm fractal: số chiều của chúng không phải là số nguyên, chúng là những số thập phân, số vô tỉ thực sự. Quay trở lại với định nghĩa của fractal ta thấy chỉ có tập Cantor, tam giác Sierpinski và bông tuyết Koch là các hình fractal: chúng có số chiều fractal lớn hơn số chiều topo của mình.

6. Hệ thống hàm lặp và thuật toán

Trong phần này chúng tôi sẽ mô tả một vài thuật toán dùng để tạo ra các hình fractal bí ẩn. Các bạn có thể sử dụng chúng để tạo ra cho riêng mình các hình ảnh kỳ lạ này.

Xét một phép biến đổi tuyến tính dạng sau:

$$A\begin{pmatrix} x \\ y \end{pmatrix} = \beta\begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

hay có thể dưới dạng rút gọn sau:

A (P) = β (P-P0) + P0, ở đây β là một hằng số thỏa mãn 0 < β < 1 va` P0 là một điểm cho trước trên mặt phẳng.

Với mỗi điểm P trên mặt phẳng ta xét quĩ đạo của P định nghĩa bởi dãy vô hạn các điểm sau là điểm lặp của ánh xạ A:

P, A (P), A2(P)=A (A (P)), ..., Ak(P) = A (A (....A (P)...),....

Định nghĩa: Giả sử 0 < β < 1 va` P1, P2,..., Pn là n điểm bất kỳ trên mặt phẳng. Giả sử A1, A2,..., An là các phép biến đổi tuyến tính dạng A (P) = β (P-Pi) + Pi, với i = 1, 2,.., n. Hệ (A1, A2,..., An) được gọi là hệ thống hàm lặp tương ứng với hệ số β và hệ các điểm P1, P2,..., Pn.

Chú ý: các hàm tuyến tính có thể thay thế bằng các biến đổi

$$A\begin{pmatrix} x \\ y \end{pmatrix} = \beta\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}\begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

với θ là góc quay của biến đổi A.

Thuật toán sinh fractal: Giả sử (A1, A2, ..., An) là một hệ hàm lặp bất kỳ. Từ một điểm bất kỳ trên mặt phẳng xét "quĩ đạo" của điểm này bằng cách áp dụng ngẫu nhiên một trong cac phép biến đổi Ai. Hình ảnh của quĩ đạo này khi số bước lặp tiến ra vô cùng chính là một hình fractal hoàn chỉnh.

Ta thể hiện thuật toán này như sau:

P = Giá trị ban đầu ngẫu nhiên

Repeat

i = random(n)

P = Ai(P)

until false

Với chương trình rất đơn giản trên, các bạn có thể tạo cho mình rất nhiều các hình ảnh lý thú của fractal.

51 .**Phân hoạch tập hợp hữu hạn phần tử**

Tác giả: Ngọc Điền

Định nghĩa
Cho tập hợp A. Một họ {A1, A2,..., An} gồm các tập con khác rỗng của A được gọi là một phân hoạch của tập A nếu nó thoả mãn:
i) Ai □ Aj = □ với i ≠ j

ii) $\bigcup_{i=1}^{n} Ai = A$

Như vậy cho tập A hữu hạn phần tử, ta có ngay hai phân hoạch tầm thường:
- Họ chỉ gồm một tập tất cả các phần tử của A.
- Họ gồm tất cả các tập một phần tử của A.
Cho tập hợp n phần tử, ta biết rằng số các tập con của nó là

$$\sum_{k=0}^{n} c_n^k = 2^n$$

Vấn đề chúng ta quan tâm ở đây là với tập hợp như vậy , ta có bao nhiêu phân hoạch và cụ thể mỗi phân hoạch đó như thế nào?

Số phân hoạch của một tập hợp
Ta có thể thấy ngay rằng:
- Nếu B là tập con của A thì một phân hoạch của tập hợp (AB) cùng với tập B sẽ là một phân hoạch của tập A.
- Ta hoàn toàn xây dựng được các phân hoạch của tập gồm n phần tử từ các phân hoạch của tập gồm (n-1) phần tử bằng cách thêm vào mỗi phân hoạch này một phần tử, sau đó lại tiếp tục phân hoạch các tập con mới.
Gọi T(n) là số phân hoạch của tập hợp gồm n phần tử. Ta qui ước T(0)=1.
Với những nhận xét như trên, các bạn sẽ chứng minh được rằng số các phân hoạch của tập hợp n phần tử được tính theo công thức:

$$T(n)=\sum_{k=1}^{n} C_{n-1}^{k-1}.T(n-k)$$

Rõ ràng ta có: T(1)=1; T(2)=2.
Trong trường hợp n=3 ta có:
Cụ thể, với A={a,b,c}, ta có các phân hoạch sau:
{a}, {b}, {c}
{a,b}, {c}
{a}, {b,c}
{b}, {c,a}
{a,b,c}

Các phân hoạch của một tập hợp:
Để liệt kê được toàn bộ các đối tượng của một tập hợp, người ta thường sử dụng một cách nào đó để có thể từ một đối tượng của tập hợp ta suy được tất cả các đối tượng khác. Ví dụ để liệt kê các dãy nhị phân có độ dài bằng 2, người ta có thể làm như sau:
- Bắt đầu từ dãy 00.
- Cộng theo hệ nhị phân (có nhớ) với 1.
- Thực hiện lại bước 2 đến khi được dãy 11 thì dừng lại.
Cuối cùng ta sẽ thu được kết quả như sau:

0	0
0	1
1	0
1	1

Để thực hiện được phương pháp này, tập các đối tượng trên phải thoả mãn hai yêu cầu sau:
- Trên tập đối tượng mà ta cần liệt kê, có một quan hệ thứ tự. Ta xác định được đối tượng đầu tiên và đối tượng cuối cùng theo quan hệ thứ tự đó.

- Tại mỗi thời điểm mà chưa phải là kết thúc của quá trình liệt kê, ta có thể suy được đối tượng tiếp theo.
Thuật toán thực hiện ý tưởng này gọi là thuật toán sinh tập hợp.
Bằng cách này, ta có thể liệt kê được tất cả các tập con k phần tử của tập gồm n phần tử. Thật vậy, không mất tính tổng quát, ta giả sử A={1, 2,..., n}.
Tập a={a1, a2,..., ak} với $1 \le a1 < a2 <... \le ak \le n$ là tập con gồm k (với $1 \le k \le n$) phần tử của tập A.
Trên tập hợp tất cả các tập con gồm k phần tử ta xác định một quan hệ thứ tự như sau: Với á={á1, á2,..., ák} cũng là tập con gồm k phần tử của A, Ta coi a
+) ai = ái nếu i=1, 2,.., j-1.
+) aj < a'j.
Ta có thể thấy tập các đối tượng (chính là các tập con gồm k phần tử của tập A) có đối tượng đầu tiên là {1, 2,..., k}, đối tượng cuối cùng là {n-k+1, n-k+2,..., n}.
Rõ ràng nếu ta có tập a={a1, a2,..., ak }, ta hoàn toàn chỉ ra được tập con kế tiếp bằng cách sau:
- Tìm từ phải sang phần tử $ai \ne n-k+i$.
- Thay ai bằng ai+1.
- Thay aj bằng ai + j-1, với j=i+1, i+2,..., k.

Ta lấy ví dụ:
Tập A={1, 2, 3, 4} có 4 tập con gồm 3 phần tử. Các tập con đó là:
{1, 2, 3}
{1, 2, 4}
{1, 3, 4}
{2, 3, 4}
Để có thể liệt kê được hết các tập con của A, ta sẽ lần lượt liệt kê các tập con gồm k phần tử với (k=1, 2,..., n).
Gọi Ai, với i=1, 2,..., 2n-1 là các tập con khác rỗng của A (tập A có một tập con là tập □). Ta liệt kê các phân hoạch của A theo cách sau:
Gọi □ là một phân hoạch.
1- Cho □ = Ai với i=1, 2,..., 2n-1. 2- Với j=1, 2,..., 2n-1 mà □ □ Aj = □ thì P:= □ +Aj.
3- Nếu □ chưa có trước đó thì P là phân hoạch mới.

Các bạn hãy theo dõi ví dụ sau:
Cho A={1, 2, 3, 4}, số phân hoạch:
Ta có:
+ Các tập con 1 phần tử: {1}, {2}, {3}, {4}
+ Các tập con 2 phần tử: {1,2}, {1,3}, {1,4}, {2,3}, {2,4}, {3,4}
+ Các tập con 3 phần tử: {1,2,3}, {1,2,4}, {1,3,4}, {2,3,4}
+ Các tập con 4 phần tử: {1,2,3,4}

Cuối cùng ta được các phân hoạch như sau:
1. {1}, {2}, {3}, {4}
2. {1}, {2}, {3, 4}
3. {1}, {2, 3, 4}
4. {1}, {2, 3}, {4}
5. {2}, {1, 3, 4}
6. {2}, {1, 3}, {4}
7. {2}, {3}, {1, 4}
8. {3}, {2, 1}, {4}
9. {3}, {1}, {2, 4}
10. {3}, {1, 2, 4}
11. {4}, {1, 2, 3}
12. {1, 2}, {3, 4}
13. {1, 4}, {2, 3}
14. {1, 3}, {2, 4}
15. {1, 2, 3, 4}
Các bạn hãy cùng giải các bài tập sau:

Bài 1. Trong một hộp có 5 tờ tiền loại 500 đồng, 1000 đồng, 2000 đồng, 5000 đồng, 10000 đồng. Bạn có thể lấy một số. Hãy cho biết bạn có thể có bao nhiêu tiền và cụ thể số tiền đó?
Bài 2. Với 5 loại tiền như trên số lượng tuỳ ý, có thể có bao nhiêu cách chọn để được số tiền là 20000 đồng?

52 .<u>Phương pháp hợp giải</u>

Logic mệnh đề, chứng minh định lý bằng phương pháp hợp giải

Trước hết ta cần phải nắm vững một số khái niệm:

1.Logic mệnh đề

Mệnh đề

Mệnh đề là một phát biểu chỉ có thể đúng hoặc sai. Trong thực tế chúng ta gặp rất nhiều mệnh đề, ví dụ:

-2 cộng 2 bằng 4.

-Toán học và tin học có mối quan hệ với nhau..

Một mệnh đề thường được ký hiệu bởi các chữ La tinh: a,b,c,..

Người ta qui ước rằng: Mệnh đề đúng có giá trị là 1 và mệnh đề sai có giá trị là 0 gọi là giá trị chân lý của mệnh đề (truth value). Trong logic mệnh đề, khi xét một mệnh đề, điều ta quan tâm là giá trị của mệnh đề.

Các phép kết nối logic

Người ta xây dựng các biểu thức trong logic mệnh đề trên cơ sở các mệnh đề và các phép kết nối logic. Giá trị chân lý của một biểu thức sẽ được tính dựa theo cấu trúc của các phép kết nối có trong biểu thức và giá trị của các mệnh đề có trong biểu thức. Với mỗi giá trị xác định của mệnh đề (hoặc là 0, hoặc là 1), các phép kết nối có trong biểu thức cho ta một giá trị hoàn toàn xác định của biểu thức. Khi đó, thay đổi giá trị của các mệnh đề trong biểu thức và tính giá trị của biểu thức theo các phép kết nối trong đó ta có một bảng gọi là bảng chân lý (truth table). Một biểu thức hay một cấu trúc kết nối sẽ được xác định khi ta viết bảng chân lý của nó. Các phép kết nối mệnh đề được định nghĩa dưới đây:

Phép hội (conjunction): Hội của hai mệnh đề a và b cho ta một biểu thức, ký hiệu âb, nhận giá trị là 1 khi a và b cùng đúng và nhận giá trị 0 trong các trường hợp ngược lại.

Phép tuyển (dijunction): Tuyển của hai mệnh đề a và b cho ta một biểu thức, ký hiệu avb, nhận giá trị là 0 khi a và b cùng sai và nhận giá trị 1 trong các trường hợp ngược lại.

Phép phủ định (negation): Phủ định của mệnh đề a cho ta một biểu thức, ký hiệu là ¬ a, nhận giá trị là 1 khi a đúng và nhận giá trị là 0 trong trường hợp ngược lại.

Phép kéo theo (implication): Mệnh đề a kéo theo mệnh đề b, ký hiệu a->b, là biểu thức nhận giá trị 0 chỉ khi a đúng và b sai và nhận giá trị 1 trong các trường hợp ngược lại.

Phép tương đương (equivalence): Mệnh đề a tương đương với mệnh đề b, ký hiệu a<->b, là biểu thức nhận giá trị 1 khi a và b đều đúng hoặc cùng đều sai và nhận giá trị 0 trong các trường hợp ngược lại.

Bảng chân lý mô tả các phép kết nối được cho dưới đây.

-Bảng chân lý mô tả phép phủ định:

a	¬ a
0	1

1	0

-Bảng chân lý mô tả các phép toán khác:

a	b	âb	avb	a->b	a<->b
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Các bạn có thể thấy ngay các phép kết nối trong logic mệnh đề có cấu trúc giống một số phép toán trong các hàm rời rạc. Tuy nhiên, trong toán rời rạc, số lượng các phép toán lớn hơn. Nếu số mệnh đề trong biểu thức là n thì chúng ta sẽ có 2n bộ giá trị cụ thể của các mệnh đề. Các bạn cần chú ý liệt kê hết các bộ giá trị này (có thể sử dụng thuật toán sinh tập hợp để thực hiện điều này). Để hiểu rõ các phép toán và biểu thức mệnh đề, ta hãy xét ví dụ sau:

Ví dụ: Xây dựng bảng giá trị của biểu thức (âb) ->c.

Biểu thức trên có ba mệnh đề a,b,c. Như vậy ta có 8 bộ giá trị của các mệnh đề này. Bảng chân lý của biểu thức như sau:

a	b	c	(âb) ->c
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Chúng ta lưu ý rằng một mệnh đề cũng được coi là một biểu thức và trong biểu thức mệnh đề nếu không có dấu (.) thì thứ tự ưu tiên các phép như sau:

-Phép phủ định.

-Phép kéo theo.

-Phép tương đương.

-Phép hội.

-Phép tuyển. (*)

Ngoài ra, chúng ta có khái niệm hai biểu thức tương đương:

Biểu thức A được gọi là tương đương với biểu thức B, ta viết là A<->B, nếu chúng có cùng bảng chân lý (nghĩa là với mỗi bộ giá trị cụ thể của các mệnh đề cùng có trong A và B, biểu thức A và B cùng nhận một giá trị như nhau). (**)

Chúng ta dễ dàng kiểm tra được rằng các phép toán hội và tuyển có các tính chất: Giao hoán, kết hợp, phân phối và lũy đẳng (tức là âa<->a, ava<->a). Một tính chất mà chúng ta cần quan tâm là tính chất đối ngẫu:

i) ¬ (avb) <-> (¬ a¬ b)

ii) ¬ (âb) <-> (¬ av¬ b)

Tính chất này được mở rộng:

¬ (a1 ^a2^^an) <-> (¬ a1v¬ a2vv ¬ an).

¬ (a1 v a2vvan) <-> (¬ a1 ^¬ a2 ^^¬ an).

Các bạn lưu ý rằng trên đây là các phép kết nối cơ bản, trong thực tế người ta đưa ra nhiều cấu trúc kết nối khác.

Mệnh đề: Mọi biểu thức logic mệnh đề đều có biểu thức tương đương với nó, chỉ chứa các phép toán hội, tuyển và phủ định. Các bạn có thể chứng minh mệnh đề này dựa theo nhận xét sau:

*) a->b <-> ¬ a v b.

*) a<->b <-> a->b^b->a.

2. Phương pháp hợp giải.

Phương pháp hợp giải (Resolution) là phương pháp chứng minh các định lý trong logic mệnh đề. Đây cũng là một trong những cơ sở để người ta xây dựng phương pháp chứng minh này là phương pháp chứng minh bằng phản chứng mà chúng ta vẫn sử dụng.

Chúng ta có một số phép biến đổi sơ cấp (các bạn hãy tự kiểm tra), chúng sẽ được sử dụng nhiều lần trong quá trình biến đổi và đó là các công cụ quan trọng để thực hiện chứng minh mệnh đề bằng phương pháp này. Với mọi a, b, c ta có:

- Phép nuốt sơ cấp: (âb) v a<->a.

- Phép dán: (âb) v (â¬ b)<-> a

- Phép dán không đầy đủ: (âb) v(â¬ b) <-> a v (âb)v(â¬ b).

- Phép dán mở rộng: (âc) v (b^¬ c) <-> (âc) v (b^¬ c) v (âb).

Một trong những cơ sở để xây dựng phương pháp Hợp giải là nhận xét:

+) Với mọi a, b:¬ (a->b) <-> (¬ b^a).

+) Biểu thức (¬ ÂA) luôn có giá trị bằng 0 và (¬ AvA) luôn có giá trị bằng 1.

ý tưởng cơ bản của phương pháp này và cũng là mục đích của chúng ta là: Giả sử ta có A phải chứng minh B nghĩa là chứng minh biểu thức (A->B) có giá trị bằng 1 khi đó ta sẽ tìm cách suy ra có giá trị bằng 0. Bài toán tổng quát mà chúng ta cần giải quyết được phát biểu như sau:

Bài toán: Giả sử ta có các giả thiết: G1, G2,Gm. Từ các giả thiết này ta hãy suy ra một trong các kết luận: K1, K2, Kn. (Các giả thiết và các kết luận ở đây đều là các mệnh đề)

Thực chất bài toán yêu cầu chúng ta chứng minh biểu thức mệnh đề:

(G1^G2 ^Gm) -> (K1 v K2 v Kn) có giá trị bằng 1.

Vận dụng phương pháp Hợp giải để giải bài toán, ta chứng minh biểu thức:

¬ (K1 v K2 v vKn)^(G1^G2 ^Gm) (1) có giá trị bằng 0. Ta có:

¬ (K1 v K2 v vKn) ^ (G1 ^G2 ^^ Gm)= (¬ K1^¬ K2^^Kn) ^(G1 ^ G2 ^^Gm)

= (¬ K1^G1)^(¬ K1^Gm) ^(¬ K2^ G1) ^^(¬ K2^ Gm) ^(¬ Kn^G1) ^^(¬ ^Gm).

Ta nhận thấy rằng nếu cặp (¬ Kp^Gq) và (¬ Ks^Gt) với p,s thuộc {1,,n}, q,t thuộc {1,,m} mâu thuẫn với nhau thì biểu thức (1) có giá trị bằng 0. Do vậy để giải bài toán chúng ta lấy phủ định của: K1, K2,,Kn , sau đó kết hợp với các giả thiết

G1,G2,,Gm. Khi mâu thuẫn xuất hiện thì bài toán được giải xong. Cụ thể bài toán được giải quyết bằng cách sau:

Bước 1:

Xây dựng tập S gồm các giả thiết được viết dưới dạng biểu thức chỉ gồm các phép toán ^, v, ¬ và phủ định của kết luận. Như vậy ban đầu tập S sẽ gồm các dòng như sau:

1) GT1

2) GT2

m) GTm

m+1) ¬ KL1.

m+2) ¬ KL2.

m+n) ¬ KLn.

Trong đó GTi và KLi là các biểu thức chỉ chứa các phép toán của ^, v, ¬ của Gi và Kj với i thuộc {1,,m}, j thuộc {1,,n}.

Bước 2:

Thực hiện việc hợp các giả thiết. Đây là giai đoạn quan trọng nhất của quá trình hợp giải. Mỗi thao tác hợp hai giả thiết cho ta một biểu thức, ta coi đó là giả thiết mới. Việc sắp xếp các giả thiết mới. Việc sắp xếp các giả thiết thành các dòng cho phép ta ký hiệu Hop-giai (i.kt, j.kt) là thao tác kết hợp hai giả thiết. Trong đó i và j là chỉ số các dòng được kết hợp, kt là các chữ cái biểu diễn thứ tự của thành phần kết hợp trong dòng thứ tự này là thứ tự của chữ cái trong bảng chữ cái (ví dụ: A chỉ thành phần đầu tiên của dòng, B chỉ thành phần thứ hai của dòng,). Các bạn lưu ý, ta phải thực hiện nhiều thao tác lặp đi lặp lại. Quá trình kết hợp này sẽ dừng lại khi xuất hiện mâu thuẫn hoặc tập S không thay đổi.

Bước 3: Xuất hiện mâu thuẫn, ta kết luận bài toán được chứng minh xong.

Một trong những hạn chế của phương pháp này là trong quá trình hợp các giả thiết, số lượng các biểu thức mới hình thành (đó cũng là các giả thiết) tăng nhanh, tức là tập S ngày càng lớn do sự bùng nổ tổ hợp.

Để nắm vững phương pháp này hơn, các bạn xét ví dụ cho dưới đây.

Ví dụ: Giả sử ta có

Bài toán được giải như sau:

Bước 1: Xây dựng tập S:

1.¬ av¬ bvc.

2. ¬ bv¬ cvd.

3. a.

4. b.

5. ¬ d.

Bước 2:

Tiến hành việc hợp các giả thiết. Các kết quả như sau:

6.¬ bvc, (Hop_giai (1.A, 3.A)).

7.¬ avc (Hop_giai (1.A, 4.A)).

8. ¬ cvd, (Hopưư_giai (2.A,4.A)).

9. ¬ bvc, (Hopư_giai (2.C,5.A)).

10. c, (Hop_giai (3.A,7.A)).

11.¬ c, (Hop_giai (4.A, 9.A)).

Bước 3: Xuất hiện mâu thuẫn ở dòng thứ 10 (và dòng thứ 11), bài toán được chứng minh xong.

Lưu ý: Thao tác hợp giải trên đây thực chất là thực hiện phép hội các biểu thức mệnh đề. Khi thực hiện thao tác hợp giải, ví dụ: Hop_giai (1.A,3.A), ta được kết quả như sau:

(¬ av¬ bvc) ^a<-> ¬ âa) v ((¬ ^a) <-> (¬ bvc) ^a.

Nhưng vì chúng ta đã có giả thiết (3.) nên ở dòng thứ sáu ta chỉ viết (¬ bvc). Các trường hợp khác tương tự.

Trong thực tế, Trí tuệ nhân tạo còn rất nhiều vấn đề cần giải quyết. Ngay vấn đề trình bày ở trên, việc biểu diễn một mệnh đề và từ một biểu thức diễn đạt được theo nghĩa tự nhiên là vấn đề rất nan giải mà bất cứ ai quan tâm đến Trí tuệ nhân tạo đều không thể bỏ qua. Trên đây, chúng ta đã làm quen một số khái niệm cơ bản của logic toán học. Phần nào các bạn hình dung được ứng dụng của nó trong việc xây dựng và phát triển các ngành khoa học khác đặc biệt là Công nghệ thông tin.

(*) Các dấu (.) trong các biểu thức trong bài đôi khi dùng để nhận dạng các biểu thức rõ hơn.

(**) Có một số sách coi là mệnh đề bằng nhau, ký hiệu: =.

Ngọc Điền

53 .**Phương pháp tính Horner**

Tác giả: Trần Quang Minh

Trong quá trình học tập và nguyên cứu các bài toán tin học, chúng ta thường gặp một dạng bài toán như:

Một bộ xử lí (processor) gồm các phép biển đổi

-command 1

-command 2

.....

-command N

Cho trước một số (gọi là StartNumber),là Input của bộ xử lí và một số (gọi là DestinationNumber) là kết quả Output của bộ xử lí.

Hãy chỉ ra trình tựu các lệnh (command) mà bộ xử lí cần thực hiện để đổi từ Inout thành output (nhiều bài toán còn đòi hỏi số phép biển đổi là ít nhất)

Trong bài viết này, xin trình bày một phương pháp rất hay, khả năng giải quyết một lớp khá rộng này. Đó là việc sử dụng phương pháp tính Horner.

Vậy phương trình tính Horner là gì?

P(x)= aNxN+aN-1xN-1+...+a1x+a0

Thực chất, đó là phương pháp tính nhanh giá trị của một đa thức P(x) tại một giá trị k nào đó. Bằng cách viết P(x) dưới dạng:

P(x)=x(aNxN-1+aN-1xN-2+...+a2x+a1)+ a0

NxN-2+aN-1xN-3+...+a2)+a1)+ a0

Nx+aN-1)+aN-2)+...)+ a0

thì phương pháp tính Horner cho chúng ta một cách tính nhanh trị số p(x)

P:=aN

For i:=n-1 Downto 0 Do p:=x*P+ai;

Để hiểu cách ứng dụng phương pháp tính Horner và dạng toán trên chúng ta sẽ lần lượt xét các ví dụ:

Ví dụ 1: Cho một bộ xử lý tạo hằng nguyên. Bộ xử lý chỉ bao gồm 4 lệnh:

-PLUS1 Khởi tạo hằng +1

-MINUS Khởi tạo hằng -1

-INC Tăng hằng nhận được lên 1

-DUP Nhân đôi hằng nhận được

Với hằng nguyên cho trước (trong phạm vi LongInt), hãy nêu chương trình ít câu lệnh nhất để tạo hằng đó.

Dữ liệu: Nhập từ bàn phím một số N (N thuộc phạm vi LongInt).

Kết quả: xuất ra màn hình trình tự các lệnh mà bộ xử lý cần thực hiện để đặt được số đó. Yêu cầu là số phép biến đổi là ít nhất.

Bài toán trên sẽ được giải quyết trong hai trường hợp sau:

Trường hợp N là số nguyên dương

Đây là trường hợp mà phương pháp tính Horner của chín ta được ứng dụng để giải quyết.

Với bộ lệnh của bộ xử lý là: DUP (nhân đôi) và INC(cộng 1) gợi cho chúng ta nghĩ đến cách biểu diễn nhị phân của N

Giả sử N được biểu diễn dưới dạng P(N)= aN2N+aN-12N-1+...+a12+a0

N2+aN-1)+aN-2)+...)+ a0

(trong đó các ai nhận giái trị 0 và 1 và aN =1)

Đến đây, chắc các bạn có thể hình dung con đườn giải quyết cài toán này.

Chúng ta đã nhìn thấy cách mà bộ xử lí sẽ thực hiện:

Đầu tiên là khởi tạo hằng +1(PLUS1) {ứng với aN =1}

Sau đó nhân 2(DUP) { ứng với 2*aN}

Cứ tiếp tục như vậy trình tự lệnh mà bộ xử lý thực hiện chính là việc tính dần từ trong ra ngoài của đa thức theo phương pháp tính Horner. Chú ý ở đây thì:

Đầu tiên là thực hiện lệnh DUP {tương ứng 2*P}

Nếu ai =1 thì thực hiện INC, còn trường hợp ai =0 thì không thực hiện lệnh N mà tiếp tục tính ra vòng ngoài.

Ta có chương trình thể hiện dãy lệnh của bộ xử lý sẽ thực hiện đơn giản như sau

PLUS1

For i:=N-1 Downto 0 Do

Begin

DUP

If ai =1 Then INC;

End;

Như vậy ý tưởng chính của phương pháp này là:

Ta đưa trình tự lệnh thực hiện của bộ xử lý về trình tự tính giá trị của một đa thức P(x) nào đó tại giá trị a mà P(a)= DestinationNumber.

Trường hợp N là số nguyên âm

Dễ thấy rằng:

1) Trường hợp này chúng ta không thể làm như trên được.

2) Lệnh đàu tiên của bộ xử lý phải là MINUS1

Có nhiều phương pháp để giải quyết bài toán trong trường hợp này. Nhưng phương pháp đơn giản dễ cài đặt hơn cả là chúng ta sẽ sử dụng một mẹo: Thay vì tìm cách biến đổi từ số ban đầu đến N là làm ngược lại là lần ngược từ N về số ban đầu để tìm số trược số i cần tạo trên đường xây dựng số N là số bào!?. Cụ thể, cách làm của tôi ở đây là:

Dùng một mảng B để lưu lại quá trình lần ngược đó.

B[1]:=N;

Chừng nào N khác dạng -2K thì thực hiện:

Nếu N chẵn thì N:=N Div 2 và nạp N vào mảng B.

Ngược lại (N lẻ) thì N:=N-1 và nạp N vào mảng B.

Như vậy, đầu tiên bộ xử lý sử dụng k lệnh DUP để tạo số N=-2k trước sau đó nhờ và bảng lần ngược B để xây dựng tiếp tục tạo các số.

Ví dụ 1: Trình tự xây dựng cuối về đầu của -6 là: -6<=-3<=-4 (dừng vì -4=-22)

Khi đó mảng B sẽ là B=(-6,3) và bộ xử lý sẽ thực hiện như sau:

Lệnh	Giá trị
-MINUS1	-1

-2 lệnh DUP -4

-INC -3

-DUP -6

Khi quá trình lần ngược kết thúc. Ta dễ dàng viết được trình tự lệnh thực hiện của bộ xử lý như sau:

MINUS1

For i:=1 to k Do DUP { k được tìm như trên}

For i:=Spt(B) Downto 1 do {số phần tử B}

Begin

If B[i] mod 2=0 Then DUP

Else INC;

End;

Chúng ta sét tiếp ví dụ sau:

Ví dụ 2: Mởy máy tính xử lý các số nguyên không âm. Máy tính chỉ có 3 ô nhớ A,X,Y và 5 lệnh sau:

R1: Biến đổi một cặp số (m,n) trong (X,Y) thành cặp số (m+n,m-n) và đặt lại trong (X,Y).

R2: Biến đổi cặp số (m,n) trong (X,Y) thành cặp số (m*2, n div 2) và đặt vào (X,Y).

R3: Đổi chỗ 2 số trong (X,Y).

R4: Sao chép A vào X (giá trị có trước của X bị thay thế).

R5: Sao chép X vào A (giá trị có trước của A bị thay thế).

Chương trình của máy là một dãy các lệnh trong 5 bộ lệnh kể trên.

Bài toán đặt ra: Cho một cặo số x,y không đồng thời bằng 0 đặt vào hai ô nhớ Xvà Y và một số a xem như là đích cần xây dựng. Hãy lập trình sinh ra một chương trình của máy sao cho khi thực hiện chương trình đó thì số a được tạo trong ô nhớ A.

Input: file Prog.Inp gồm 1 dòng ghi 3 số x,y,a cách nhau bởi kí tự trống.

Output: kết quả trả lời ghi thành bản Debug gồm chương trình và kế đó là kết quả khi thực hiện các phép toán trong file Prog.Out (gồm thông tin về giá trị trong các ô nhớ A,X,Y). Nếu trong A chưa có giá trị thì thay vào đó là dấu #. Dong cuối cùng là từ END báo hiệu kết thúc chương trình.

Bài toán này trông có vẻ phức tạp song nếu nhìn một cách "phóng khoáng" nó không khó hơn bài toán ở ví dụ 1.

Tương tự như ví dụ 1. Ta đưa trình tự lệnh về cách tính Horner lần lượt trong ô nhớ A. Các lệnh cần thiết là: nhân đôi giá trị ô nhớ A, cộng một vào ô nhớ A dễ dàng có được nhờ vào 5 bộ lệnh đã cho.

Với bài viết này mong các bạn sẽ áp dụng sáng tạo chương phương pháp trên vào nhiều bào toán. Chẳng hạn, tuỳ bộ lệnh của bộ vi xử lý mà ta đổi về hệ cơ số cho thích hợp (chứ không nhất thiết là cơ số 2).

54 .**Bài toán quay vecto với một cách làm khác**

Tác giả: Nguyễn Xuân Tài

Nếu bạn không thể tìm thấy số báo đó, bạn có thể xem đề ra như sau:

Bài toán (trên báo số 3-1999):

Cho mảng nguyên a[1..N] ($0 \le N \le 43212$) và một số tự nhiên K ($0 \le K \le N$). Hãy chuyển K phần tử từ đầu mảng về cuối mảng

Bài toán trên có thể nêu ở dạng khác như sau (gọi là bài toán quay véctơ):

Cho vectơ A có 2 đoạn MN, hãy quay véctơ đó thành véctơ NM. Với đoạn M là K phần tử đầu của véctơ A"

Theo quy luật, bài toán hẹp hơn bao giờ cũng có cách hay hơn. Nhưng bài toán này thì sao? Biết đâu lại chẳng có cách nào khác? Chúngta thử tìm hiểu xem.

Thuật toán (A'B')' là một thuật toán tinh tế và dễ hiểu. Nhưng cũng như biết bao bài toán khác, chúng ta có quyền phân vân "Có cách giải nào nữa chăng?". Đối với bài toán này, có một thuật toán cũng dễ hiểu không kém:

Không mất tính tổng quát, ta giả sử đoạn M không lớn hơn đoạn N (tức là $K \leq N/2$), ta chia đoạn N ra làm 2 phần. N1 và N2 sao cho N2=M. đôi chỗ hai đoa.n M, N2 cho nhau, vậy là M đã về đúng vị trí. Bây giờ véctơ A là N2N1M và chúng ta lại phải đổi chỗ của N2 và N1.

Vậy là đã xong, đó là một thuật toán đệ quy. Nhưng bài toán này đâu có dễ đến thế, chúng ta còn một vấn đề nữa cần phải giải quyết đó là viết chương trình thế nào đây. Nếu các bạn cứ viết như bình thường thì quả thuật toán trên chẳng có gì đáng nói cả vì nó chạy chậm hơn thuật toán (A'B')' nhiều.

Trước khi vào viết chương trình cụ thể, chúng ta hãy cùng tìm hiểu thủ tục Move. Đây là thủ tục "linh hồn" cho chương trình của chúng ta.

Cú pháp:

Move (var Source, Destination; count:word);

Source, destination: là mảng, biến hoặc đặc biệt hơn nữa là phần tử đầu của một số byte liên tiếp trong mảng.

Count: là số byte liên tiếp của biến được chỉ rõ trong Source và destination.

Bạn có thể tự hỏi vậy thì dùng move và dùng phép gán có khác nhau không? Chúng chênh nhau nhiều về thời gian thực hiện. Nếu bạn không tin chúng ta cùng làm thí nghiệm nhỏ sau:

```
uses crt;

const

limit=1000;

var

a:array[1..1000] of byte;

b:array[1..1000] of byte;
```

```
time:longint;
i,j,k:longint;
begin
clrscr;
for k:=1 to 8 do
begin
writeln('Test ',k, ' ');
time:=meml[0:$46c];
for i:=1 to limit do
move(b[1],a[1],1000);
writeln('MOVE:',((meml[0:$46c]-time)/18.21):10:10);
time:=meml[0:$46c];
for i:=1 to limit do
for j:=1 to 1000 do
b[j]:=a[j];
writeln('Gan: ',((meml[0:$46c]-time)/18.21):10:10);
end;
readln;
end.
```

Thế nào? Giá trị limit mới có 1000 thôi đấy nhé. Nếu lỡ tay cho thêm vài con số 0 vào sau số 1 của hằng limit nữa thì bạn nên đi uống một vài tách trà, sau khi quay lại bạn sẽ thấy trên màn hình một con số tương tự như 53.5969 (Tính bằng giây đấy! Trời ơi). Ngay cả tôi cũng không đợi nổi đến test thứ 2 và phải Ctrl+Break.

Những kiến thức trên thật là sơ đẳng và chính kiến thức sơ đẳng ấy sẽ giúp chúng ta giải quyết bài toán một cách gọn đẹp. Bây giờ bạn đã có đủ "vũ khí để ra chiến trường" rồi, chúng ta bắt đầu viết chương trình:

Khai báo:

```
A: Array[1..43212] of byte; {chuẩn bị cho những Test lớn}

B: Array [1..21606] of byte; {mảng phụ}

N,K : word;
```

Thủ tục sau đây sẽ tiến hành mọi công việc cần thiết:

```
procedure change(i1,j1,i2,j2:word);

begin

if (j1 then

begin

writeln(((meml[0:$46c]-timesaved)/18.21):12:12);

exit;

end;

if j1-i1+1then

begin

d:=j1-i1+1; {D là biến toàn cục}

move (a[i1],b[1],d);

move (a[j2-d+1],a[i1],d);

change(i1,j1,j1+1,j2-d);

end; {of if}
```

```
else

begin

d:=j2-i2+1;

move(a[i1],b[1],d);

move(a[i2],a[i1],d);

move(b[1],a[i2],d);

change(i1+d,j1,i2,j2);

end;

end; {of procedure}
```

Đối với những Test lớn, chương trình này làm việc hiệu quả hơn nhiều so với chương trình theo thuật toán (A'B')'. Hầu hết các Test lớn thậm chí cả Test có độ phức tạp cao nhất nó cũng chỉ tốn mất 0.0000 giây trên máy 300 Mhz. Nhược điểm của chương trình này là nó có thể gây ra lỗi tràn Stack. Bạn có thể giải bài toán này một cách tối ưu bằng cách phối hợp cả hai thuật toán, còn cách phối hợp như thế nào thì tuỳ vào sự sáng tạo của riêng bạn. Nhưng đầu tiên hãy kiểm tra xem chương trình này có đạt tiêu chuẩn không nhé. Chúc bạn thành công.

55 .**Nhìn lưới ô vuông dưới con mắt Tin học như thế nào?**

Tác giả: Trần Minh Quang

Trong các kỳ thi học Tin học, dạng toán về lưới ô vuông chiếm tỉ lệ khá nhiều. Cách ra đề thường lắt léo khiến chúng ta mất phương hướng dẫn đến "mù tịt" về thuật giải. Vì vậy việc tập nhìn nhận những lưới ô vuông này thật đáng cho chúng ta bỏ ít thời gian để nghiên cứu.
Có một điều chắc chắn rằng, mỗi một chuyên gia ở các ngành nghề khác nhau sẽ nhìn lưới ô vuông này bằng những "con mắt" khác nhau:
Một nhà toán học có thể nhìn nó như một mặt phẳng toạ độ Đề Các trong khi nhà địa lý lại xem nó như một phần trải thẳng của bản đồ mà các đường ngang dọc là các vĩ tuyến và kinh tuyến...!!!. Còn dưới con mắt chúng ta - những " chuyên gia Tin học", chúng ta thấy những gì? Vâng, một đồ thị cũng nên !!?
Ta sẽ điểm qua một số cách nhìn về chúng.
I. Là một mảng hai chiều MxN.

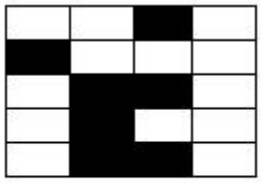
Rõ ràng đây là cách nhìn "toán học". Hầu hết các bài toán dễ thường sử dụng cách nhìn này. Lấy một ví dụ đơn giản sau:

Ví dụ: Tìm kiếm mẫu

Cho một hình chữ nhật kích thước MxN được chia thành các ô vuông như mô tả trong hình 1a dưới đẫy. Mỗi ô vuông được đánh số liên tiếp sao cho ô ở góc trên bên trái được đánh số 1, ô bên phải nó được đánh số 2 và ô ở góc dưới phải được đánh số MxN.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	16	16
17	18	19	20

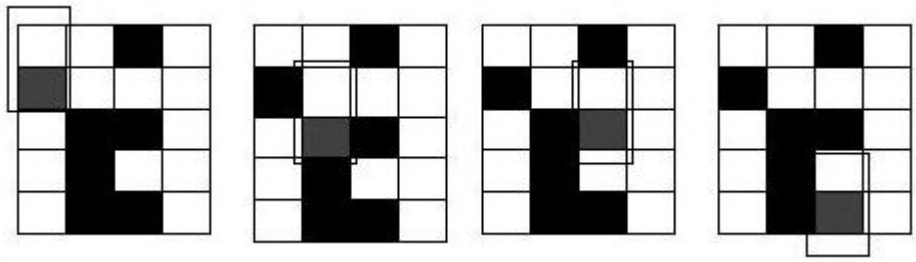
Hình 1a



Hình 1b

Hình 1c

Hình 1b cho thấy lưới ô vuông với một số ô được sơn đen. Xét một hình chữ nhật con với ô ở góc trên trái chính là ô góc trên trái của hình chữ nhật đã cho còn góc dưới phải có toạ độ (p,q). Chẳng hạn hình chữ nhật con xác định bởi ô dưới phải (2,1) của hình chữ nhật cho trong hình 1b có dạng cho trong hình 1c. Hình chữ nhật con này xuất hiện ở 4 vị trí khác nhau trong hình chữ nhật đã cho trong hình 1b (xem các hình dưới đây)



Bạn cần viết chương trình nhập thông tin về hình chữ nhật đã cho và một hình chữ nhật con của nó, sau đó đưa ra số lần xuất hiện của hình chữ nhật con trong hình chữ nhật đã cho. Bạn không cần tính các phép quay của hình chữ nhật con. Lưu ý rằng hình chữ nhật con luôn được chọn ở góc trên trái của hình chữ nhật đã cho.

Dữ liệu: Từ file văn bản RPART.INP

· Dòng đầu tiên chữ bốn số nguyên M,N ($1 \le M,N \le 20$), p ($1 \le p \le M$) và q ($1 \le q \le N$).

· Dòng thứ hai cho biết số lượng ô được bôi đen x, ($0 \leq x \leq M,N$). Mỗi một trong số x dòng tiếp theo chứa vị trí của một ô được sơn đen (cách đánh số vị trí được mô tả trong hình 1a).

Kết quả: Ghi ra file văn bản RPART.OUT số lần xuất hiện của hình chữ nhật con trong hình chữ nhật đã cho (tính cả vị trí ban đầu ở góc trên trái của nó).

Ví dụ:

RPART.INP	RPART.OUT
5 4 2 1 7 3 5 10 11 14 18 19	4

RPART.INP	RPART.OUT
4 5 2 2 2 2 5	2

Lời giải bài toán này không có gì hơn là: Dùng một mảng hai chiều A[1..20,1..20] lưu trạng thái của hình chữ nhật rồi tiến hành dò từ trái qua phải, từ trên xuống dưới để đếm số lượng các mẫu xuất hiện. (Cần chú ý việc đổi toạ độ của ô đen vào bảng)

II. Là một mảng một chiều MxN phần tử

Thông thường cách tiếp cận này dùng để tránh lãng phí bố nhớ trong một số trường hợp như: dữ liệu dưới dạng bảng chỉ chứa trong một nửa bảng.

III. Là một đồ thị 2 phía

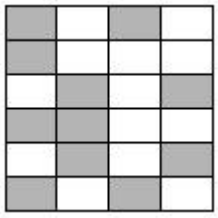
Đây chính là cách nhìn rất Tin học và quan trọng nhất. Chúng ta sẽ đi sâu vào cách nhìn này bởi kinh nghiệm cho thấy các bài toán khó gặp trong thực tế thường liên quan đến cách nhìn này. Vài ví dụ sau đây sẽ cho chúng ta cách nhìn cụ thể hoá hơn:

Ví dụ 1: Cho một bảng gồm các ô vuông kích thước MxN (M,N ≤100), trong đó có một số ô đen, còn lại là ô trắng. Hãy tô màu tất cả các ô trắng bằng hai màu Xanh và Đỏ sao cho trên mỗi dòng cũng như trên mỗi cột các ô màu Xanh và Đỏ lệch nhau không quá 1.

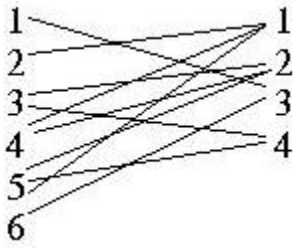
Quả thực, nếu nhìn nhận bài toán với mô hình bảng chữ nhật thì không có thuật toán nào hay cho bài toán cả. Song nếu áp dụng cách nhìn nhận dưới góc độ lý thuyết đồ thị thì bài toán trở nên khá đơn giản và có thể giải quyết không mấy khó khăn.

Ta xây dựng một đồ thị hai phía gồn M+N đỉnh mà mỗi đỉnh tương ứng với mỗi dòng và mỗi cột của bảng đã cho. Các đỉnh phía trái 1, 2,..., M tương ứng với dòng thứ nhất, thứ hai,..., thứ M. Các đỉnh phía phải 1, 2,..., N tương ứng với cột thứ nhất, thứ hai,..., thứ N. Nếu ô (I,J) là ô đen ta có cung nối đỉnh thứ I thuộc bên trái với đỉnh thứ J thuộc bên phải.

Ví dụ ta có bảng sau:



thì đồ thị 2 phía tương ứng là:



Lúc này, bài toán của chúng ta được phát biểu lại như sau:

Hãy tô màu các cung của đồ thị bằng hai màu Xanh và Đỏ sao cho khi tô hết tất cả các cung thì tại mỗi đỉnh của đồ thị số cung Xanh và Đỏ hơn kém nhau không quá 1.

Với cách nhìn mới dưới dạng đồ thị chúng ta có thuật toán tốt để giải quyết nhờ vào hai nhận xét sau:

Nhận xét 1: Nếu xuất phát từ một đỉnh bậc lẻ và đi một cách bất kỳ theo các cung của đồ thị, mỗi cung đi qua một lần thì trạng thái "tắc đường" phải xẩy ra tại một đỉnh bậc lẻ khác (cần chú ý: số đỉnh bậc lẻ (nếu có) trong đồ thị phải là một số chẵn).

Nhận xét 2: Nếu xuất phát từ một đỉnh bậc chẵn trong đồ thị không có đỉnh bậc lẻ và đi một cách bất kỳ theo các cung của đồ thị, mỗi cung đi qua chỉ một lần thì trạng thái "tắc đường" phải xảy ra chính tại đỉnh xuất phát.

(Trạng thái "tắc đường" là trạng thái mà tại đỉnh vừa tới không còn cung nào chưa đi qua).

Với hai nhận xét mấu chốt trên ta có thuật toán để giải quyết bài toán:

Trường hợp 1: Khi đồ thị vẫn còn đỉnh bậc lẻ

Chọn một đỉnh bậc lẻ bất kỳ để xuất phát. Bằng cách đi bất kỳ qua các cung của đồ thị mà chưa được tô màu, mỗi cung đi qua ta tô xen kẽ bằng hai màu Xanh và Đỏ cho đến khi "tắc đường". Trong trường hợp này, tại đỉnh bậc lẻ kết thúc đường đi trên, không còn cung nào chứa nó chưa được tô, đồng thời tại đỉnh xuất phát số cung còn lại chưa tô (nếu có) là một số chẵn, còn tại các đỉnh còn lại trên đường đi số cung được tô bằng màu Xanh và Đỏ bằng nhau.

Trường hợp 2: Khi đồ thị chỉ còn đỉnh bậc chẵn

Trường hợp này, tất cả các cung kề với các đỉnh bậc lẻ (nếu có) của đồ thị ban đầu đều đã được tô. Chọn một đỉnh nào đó còn cung chưa tô chứa nó là đỉnh xuất phát và đi một cách bất kỳ theo cung chưa được tô cho đến khi đạt được trạng thái kết thúc (tại đỉnh xuất phát). Bằng cách tô màu các cung xen kẽ trên đường đã đi qua. Khi đó số lượng các cung được tô màu Xanh và Đỏ kề với mỗi đỉnh trên đường đi là bằng nhau.

Cứ thực hiện như trên cho đến khi mọi cung đều được tô màu ta được lời giải của bài toán.

Như vậy, chúng ta vừa mới tiếp cận một cách nhìn lưới ô vuông khá mới mẻ và thú vị. Bây giờ ta sẽ nhìn nhận một bài toán khác về lưới ô vuông mà để giải nó ngoài việc "nhìn" nó tốt còn phải phải nắm vững thuật toán tìm luồng cực đại trong mạng.

Ví dụ 2: Đề thi học sinh giỏi toàn quốc 1994-1995

Cho một bảng chữ nhật kích thước MxN (M,N <100) ca'c o^ vuo^ng. Trong ddo' co' mo^.t so^' o^ tra('ng co`n la.i la` dden. Ha~y cho.n ra 2M o^ dden trong ba?ng sao cho:

Mỗi dòng chọn đúng 2 ô đen

· Số ô đen được chọn trong cột chọn nhiều ô nhất là nhỏ nhất.

Theo kinh nghiệm của bài trên, chúng ta thành lập một mạng gồm M+N+2 đỉnh (2 đỉnh giả S và T đóng vai trò đỉnh phát và đỉnh thu). M đỉnh phía trái ứng với M dòng đều được nối với đỉnh phát S và N đỉnh phía phải ứng với N cột của bảng đều được nối với đỉnh thu T.

Nếu ô (I,J) là ô đen thì ta nối đỉnh I phía trái với đỉnh J phía phải. Độ thông qua được xác định như sau:

- Mọi cung xuất phát từ đỉnh S có độ thông qua bằng 2.

- Mọi cung nối cặp đỉnh phía trái với phía phải có độ thông qua bằng 1

- Mọi cung nối với đỉnh thu T có độ thông qua đều bằng k (k sẽ thay đổi trong quá trình tính toán). Ban đầu k = 1; sau mỗi bước tìm luồng cực đại nếu giá trị luồng bằng 2M thì kết thúc và in lời giải còn nếu nhỏ hơn ta sẽ tăng k lên 1 đơn vị (chú ý ở đây có thể dùng phép tìm kiếm nhị phân).

Thưa các bạn ! Chúng ta đã biết thêm được khá nhiều điều mới mẻ và bổ ích phải không ạ? Tất nhiên còn rất nhiều cách nhìn khác dưới "con mắt Tin học" của chúng ta đối với các lưới ô vuông mà mỗi cách nhìn mới lại cho ta những lời giải mới, độc đáo hơn.

Tôi mong rằng các bạn sẽ phát hiện nhiều cái nhìn mới hơn trong các lớp các bài toán - tin về lưới ô vuông. Thiết nghĩ, cách nhìn nhận một vấn đề dưới nhiều hình thức khác nhau như vậy âu cũng là một cách học tập tốt.

Để luyện tập, các bạn hãy thử sức với một bài toán "khá mềm" sau:

Bài tập: Cho một ma trận MxN (M,N ≤ 100) và 2 dãy số nguyên a1,a2...,aN và b1,b2,...bN thoả mãn điều kiện ∑ ai = ∑bi. Hãy viết chương trình lấp đầy ma trận bằng các số 0 hoặc 1 sao cho tổng các số các số trên hàng i là ai, tổng các số trên cột j là bj.

Nếu cần chương trình giải các ví dụ trên hãy liên hệ với tôi !

Trên đây là một vài ý kiến nhỏ của tôi, rất vui lòng được các bạn góp ý.

Xin chân thành cảm ơn.

56 .**Tin học lý thuyết và toán học**

Tác giả: GS. Jean Eric Pin

Tin học và Toán học là hai bộ môn khác biệt nhưng không độc lập với nhau. Ngay những vấn đề đơn giản như các phép tính số học cũng đặt ra những bài toán rất khó, thậm chí không giải được. Lại nữa, giải thuật nhân nhanh hai số nguyên, được phát minh bởi Schõnhage và Strassen vào năm 1971 là dựa trên ý tưởng được lấy từ giải tích Fourier. Điều đáng ngạc nhiên là, cho tới nay vẫn còn chưa biết số tối thiểu các phép nhân cần thiết để thực hiện phép nhân hai ma trận cấp n. Chỉ biết được số đó cỡ nr, trong đó giá trị của r đã được làm giảm dần:

Vào năm 1969, r = 2,81.

năm 1978, r = 2,79.

năm 1979, r = 2,78.

năm 1981, r = 2,55 rồi 2,53.

năm 1982, r = 2,50.

năm 1987, r = 2,48 rồi 2,38 v.v...

Mặt khác việc xuất hiện các máy tính song song đã làm thay đổi các dữ kiện của bài toán. Các giải thuật có hiệu lực trên kiểu máy tính này không nhất thiết là các giải thuật có hiệu lực trên máy tính chỉ có một bộ xử lí.

Kinh nghiệm đã nhanh chóng chứng tỏ cho các nhà Tin học rằng, mặc dù các máy của họ có công suất lớn, vẫn chưa giải được một số bài toán. Điều đó khiến cho các nhà nghiên cứu phải cho một định nghĩa hình thức về độ khó của một bài toán. Đó là điểm xuất phát của lý thuyết độ phức tạp. Một trong những thành công của lý thuyết này là đã chứng minh rằng nhiều bài toán cụ thể có bản chất rất khác nhau nhưng lại có cùng cấp độ khó.

Bài toán người bán hàng rong và bài toán xếp ba lô (cho trước một tập các vật có trọng lượng đã biết, hỏi xem có thể chọn được một tập con các vật có tổng trọng lượng đúng bằng trọng lượng giới hạn cho phép của ba lô) là các thí dụ. Khi gặp các

bài toán thực sự quá khó để có thể giải quyết được trọn vẹn, ta có thể tìm lời giải gần đúng. Một giải pháp khác là việc sử dụng các thuật toán xác suất có tham gia một hay nhiều phép chọn ngẫu nhiên. Loại giải thuật này rất thông dụng trong lý thuyết mật mã. Lý thuyết đồ thị và Toán học rời rạc tham gia vào nhiều lĩnh vực của Tin học: trong khoa học giải thuật (algorithmic) là đương nhiên, mà còn trong việc mô hình hoá các mạng, trong lý thuyết biên dịch, trong việc thiết kế các mạch, v..v... Hơn nữa các giải thuật trên đồ thị làm thành một chương quan trọng của môn giải thuật học và có nguồn gốc từ nhiều bài toán thực hành.

Thoạt đâu, các máy tính hầu như chỉ thao tác với các chữ cái và chữ số. Ngày nay chúng thao tác cả với những đối tượng ba chiều và hoạt hình theo thời gian thực. Tiến trình đó đã dẫn tới việc sáng tạo ra một môn học mới là hình học tính toán (còn gọi là hình học giải thuật) mà các ứng dụng đụng chạm tới tất cả các khía cạnh hình học của Tin học: thị giác máy tính, thực hiện các mạch với độ tích hợp rất cao, khoa học người máy (robotics), v..v...

Một trong các hoạt động cơ bản của các nhà Tin học là viết các chương trình. Để làm được điều đó, họ sử dụng ngôn ngữ lập trình hiện đã có với một số lượng rất lớn. Hàng trăm nhà nghiên cứu đang tập trung vào việc xây dựng các ngôn ngữ cho tương lai. Phần lớn các công việc đó được xây dựng trên cơ sở của lôgic toán. Một trong những mục tiêu được tuyên bố là nhằm làm thuận tiện việc "chứng minh" chương trình, có nghĩa kiểm chứng, nếu có thể, một cách tự động, xem chương trình có thực hiện đúng cái mà ta mong đợi ở nó không.

Phép tính hình thức cần phải rất khác với phép tính số. Nó thao tác trên các đối tượng Toán học không nhất thiết là các số. Một hệ tính hình thức có khả năng phân tích các số nguyên, các đa thức, khai triển một hàm thành chuỗi, tính đạo hàm, giải các phương trình vi phân, nói tóm lại thực hiện loại bài tập về toán mà ta thường ra cho sinh viên. Nhưng việc dùng phép tính hình thức không chỉ dừng lại ở mức độ đó. Ta cũng có thể thao tác các khái niệm Toán học trìu tượng hơn, hay tạo ra những đối tượng mà muốn làm việc trên chúng.

Lý thuyết Ôtômat ra đời vào những năm 50 nhằm: một mặt hình thức hoá các ngôn ngữ tự nhiên, mặt khác hình thức hoá các nơ ron. Sau đó lý thuyết Ôtômat đã được phát triển thành một môn tự trị và đã thiết lập được những cơ sở lý thuyết vững chắc, cho phép mô hình hoá những tình huống cực kì khác nhau. Những ứng dụng của lý thuyết Ôtômat được trải rộng từ lý thuyết chương trình dịch tới ngữ nghĩa của các ngôn ngữ đồng bộ, rồi chuyển qua lý thuyết mã hoá, nén dữ liệu hay việc thao tác trên các tài liệu. Nó cũng được áp dụng tốt cho việc phân tích các dây chuyền phân tử gặp trong sinh học giống như việc xử lí các xâu kí tự.

Như vậy, các nhà Tin học đã sử dụng Toán học rất nhiều. Họ không những hướng nhiều về những bộ môn của Toán học đã được thiết lập tốt như: đại số, giải tích hay xác suất để nghiên cứu hình học tính toán, như lý thuyết số dùng cho lý thuyết mật mã, như lôgic toán dùng cho nghiên cứu ngữ nghĩa các ngôn ngữ, mà còn hướng về các bộ môn hiện đại hơn của Toán học như: Lý thuyết đồ thị hay Toán học rời rạc. Và khi những cái đó còn chưa đủ, các nhà Tin học không ngần ngại gì trong việc sáng tạo ra một lý thuyết theo kiểu "may đo". Đó là điều đã xảy ra đối với lý thuyết độ phức tạp và lý thuyết Ôtômat.

Xem xét ảnh hưởng ngược lại của Tin học lên Toán học cũng là điều lý thú. Các nhà lôgic hợp tác rất thường xuyên với các phòng thí nghiệm Tin học (khi mà bản thân họ chưa trở thành những nhà Tin học). Các nhà Toán học lỗi lạc như S.Eilenberg và J.H.Conway, mỗi người đều đã viết sách về lý thuyết Ôtômat và S.Smale đã đề xuất một sự mở rộng của lý thuyết độ phức tạp.

Sau cùng, từ nay về sau, cứ bốn năm một lần, ngoài việc trao những huy chương Fields truyền thống cho các nhà Toán học trẻ xuất sắc, còn trao một giải thưởng Nevanlinna cho một nhà Toán học làm việc trong lĩnh vực những vấn đề cơ sở của Tin học.

Hồ Thuần lược dịch
GS. Jean Eric Pin
(Trung tâm nghiên cứu Khoa học quốc gia Pháp)

57 .**Tản mạn xung quanh bài toán mã đi tuần**

Tác giả: Trịnh Thanh Hải

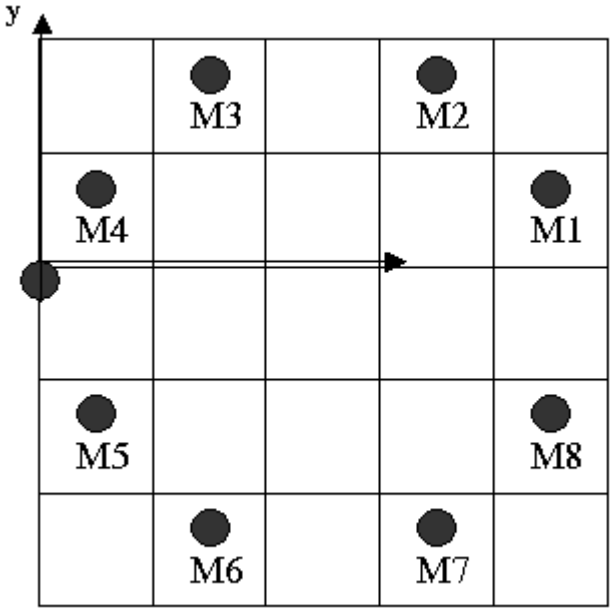
Đối với các bạn say mê môn Tin học thì bài toán phỏng theo quy luật di chuyển của con mã trên bàn cờ quốc tế rất quen thuộc. Đã có nhiều thuật toán để giải bài toán này. Trong bài viết này chúng tôi muốn trình bày thêm một số suy luận nhằm sáng tỏ câu hỏi:?Từ một vị trí bất kỳ trên bàn cờ, liệu con mã có thể đi đến tất cả các vị trí trên bàn cờ được không?.

Trước tiên ta xét bài toán:Nếu đặt con mã ở một vị trí bất kỳ trên bàn cờ và chọn một vị trí đích thì liệu con mã có thể đi đến được vị trí đó không? Nếu đi được thì số bước đi tối thiểu là bao nhiêu?

Để minh hoạ đường đi của con mã ta lập một hệ toạ độ mà gốc toạ độ O(0,0) là vị trí ban đầu của con mã, vị trí kết thúc sau khi di chuyển của con mã là M(x,y).

Ta xét từng trường hợp

+ Số bước đi là 1: từ quy luật di chuyển của con mã trên bàn cờ ta thấy ngay con mã sẽ có 8 khả năng di chuyển khác nhau. Khi đó toạ độ điểm đích của nó sẽ là một trong các trường hợp sau: M1(2,1); M2(1,2); M3(-1,2); M4(-2,1); M5(-2,-1); M6(-1,-2); M7(1,-2); M8(2,-1). Xét toạ độ x và y ta có: $|x| + |y| = 3$. Mặt khác khi di chuyển một bước, số ô theo bất kỳ hàng hoặc cột đều không quá 2, so với vị trí xuất phát số hàng, cột chênh lệch chỉ là 1 hoặc 2: $|x| \le 2$ và $|y| \le 2$. Hơn nữa x,y không cùng lẻ hoặc chẵn. Vậy với x, y không cùng chẵn hoặc lẻ thỏa mãn $\{|x| \le 2, |y| \le 2; |x|+|y| = 3\}$ thì con mã chỉ cần di chuyển một bước để đến vị trí M(x,y).



Hình 1

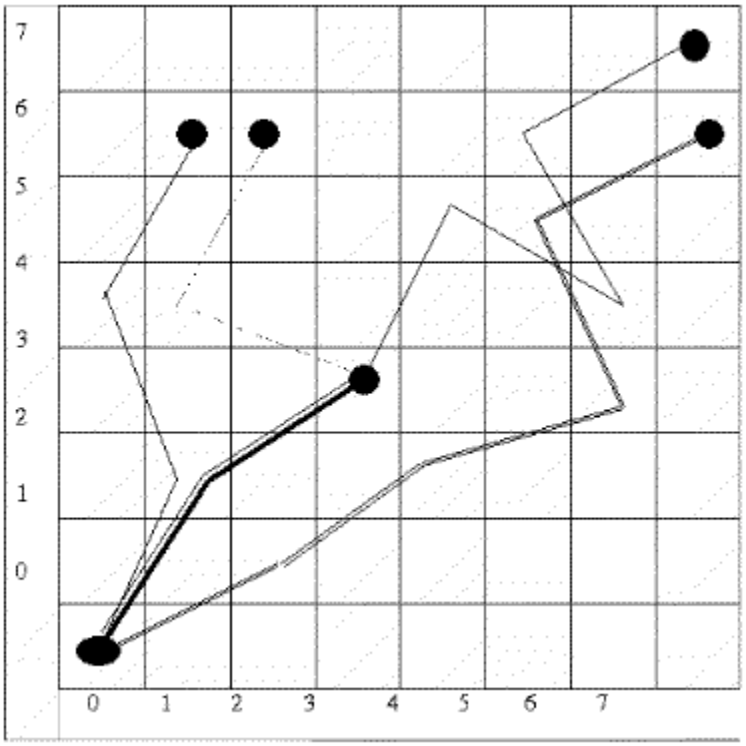
+ Số bước đi là 2 Sau 2 bước đi toạ độ x, y chỉ có một trong hai khả năng: x, y cùng lẻ hoặc x, y cùng chẵn và thoả mãn: $\{|x| \le 4 ; |y| \le 4; |x|+|y| \le 6\}$. Ngược lại nếu ta cho trước tọa độ điểm đích M(x,y) thỏa mãn điều kiện trên thì con mã chỉ cần sau 2 bước là có thể đến được điểm M (riêng để di chuyển tới vị trí M(2,2), N(-2,-2), P(2,2), Q(2,-2) cần 4 lần di chuyển.

+ Số bước đi là 3: Sau 3 bước di chuyển toạ độ x, y không cùng lẻ hoặc cùng chẵn và thoả mãn: {|x| ≤ 6; |y| ≤ 6; 3 * |x|+|y| ≤ 9} và đây cũng là điều kiện tọa độ x,y của điểm M để con mã có thể đến được sau 3 bước di chuyển

Tiếp tục quá trình xét tương tự, chẳng hạn sau 5 bước di chuyển của con mã ta có |x| +|y| ≤ 15 trong đó x,y không cùng lẻ hoặc cùng chẵn... Ta có bảng sau:

Điều kiện ràng buộc với toạ độ x, y của điểm đích M(x,y)	Số bước ít nhất để con mã đi tới đích
|x|+|y| = 3; |x| =< 2; |y| =< 2; x, y không cùng lẻ hoặc chẵn	1
|x|+|y| =< 6; |x| =< 4; |y| =< 4; x,y cùng chẵn hoặc lẻ	2
3 * |x|+|y| =< 9; |x| =< 6; |y| =< 6; x,y không cùng chẵn hoặc lẻ	3
6< |x|+|y| =< 12; |x| =< 6; |y| =< 6; x,y cùng chẵn hoặc lẻ	4
9<|x|+|y| =< 15; |x| =< 7; |y| =< 7; x,y không cùng chẵn hoặc lẻ	5
12< |x|+|y| =< 15; |x| =< 7; |y| =< 7; x,y cùng chẵn hoặc lẻ	6

Ta minh hoạ bảng trên qua một vài ví dụ cụ thể:



Hình 2

Giả sử vị trí xuất phát của con mã là góc dưới bên trái, như vậy các ô được gắn toạ độ từ (0,0)... (7,7) (hình 2)

+ Vị trí ô đích là (3,3). Vì |3|+|3| = 6, hơn nữa cả hai toạ độ x, y đều lẻ nên ta chỉ cần 2 bước để di chuyển từ (0,0) đến (3,3).

+ Vị trí đích là ô (1,6). Ta có |1| + |6| = 7, mặt khác trong 2 số x, y có một số lẻ, một số chẵn. Vậy con mã chỉ cần 3 bước để từ (0,0) đến (1,6)

+ Vị trí đích là ô (2,6). Ta có |2|+|6| = 8, mặt khác x, y cùng chẵn. Vậy để đi từ (0,0) đến (2,6) cần tối thiểu 4 lần di chuyển.

+ Vị trí ô đích là (7,6). Ta có |7| + |6| = 13, mặt khác x, y một số chẵn, một số lẻ. Căn cứ vào bảng trên ta có cố bước di chuyển tối thiểu là 5.

+ Ví trí ô đích là (7,7). Vì |7| + |7| = 14 và x, y cùng là số lẻ nên có bước tối thiểu để đi từ (0,0) đến (7,7) là 6 bước.

Chú ý rằng đây là số bước đi tối thiểu chứ không phải là cách di chuyển duy nhất. Có thể có nhiều cách đi với số bước trên để đến vị trí đích.

Vậy là nếu đặt con mã vào một vị trí bất kỳ trên bàn cờ (ta coi là gốc toạ độ) và chọn một ô bất kỳ (có toạ độ (x,y) với hệ toạ độ đã xác định) làm ô đích thì ta xét thấy x,y chỉ có thể là một trong các khả năng đã liệt kê trong bảng. Do đó con mã hoàn toàn có thể di chuyển tới đích và số lần di chuyển tối thiểu đã được nêu trong bảng.

Căn cứ quy luật di chuyển của con mã sau một bước, so với vị trí xuất phát số hàng, cột chênh lệch chỉ là 1 hoặc 2. Hơn nữa x, y không cùng lẻ hoặc chẵn. |x|+|y|=3; |x| Ê 2; |y| Ê 2 điều này tương đương với điều kiện x2 + y2 = 5, nên ta có thuật toán tìm vị trí ô có toạ độ (x[i],y[i]) cho bước di chuyển thứ i (i>=1) của con mã là:

```
for dong:=- 2 to 2 do

for cot:=- 2 to 2 do

if (dong*dong+cot*cot=5) then

begin

a:=x[i-1]+dong;

b:=y[i-1]+cot;

if (ô( a,b) vẫn thuộc bàn cờ) then
```

```
Begin

x[i]:=a;

y[i]:=b;

end;

end;
```

Cuối cùng ta giải quyết bài toán:?Đặt con mã vào một vị trí bất kỳ trên bàn cờ. Tìm chu trình của con mã để ghé thăm một lần duy nhất tất cả các ô còn lại?. Kết quả ta tìm được nhiều chu trình đi khác nhau nhưng số bước đi đều là 63.

Để đơn giản ta chọn ô góc dưới bên trái là gốc toạ độ. Như vậy các cột, các dòng được gắn số thứ tự từ 0 đến 7 (hình 2). Theo trên ta có thủ tục Try_ma như sau:

```
Procedure Try_măi:integer);

var dong,cot:-2..2;

a,b:integer;

Begin

for dong:=-2 to 2 do

for cot:=-2 to 2 do

if (dong*dong+cot*cot=5) then

begin

a:=x[i-1]+dong;

b:=y[i-1]+cot;

if (0=<a)and(a<=7)and(0=<b)and(b<=7)and(bc[a,b]) then

begin

x[i]:=a;
```

```
y[i]:=b;

bc[a,b]:=false;

if i< 63 then Try_ma(i+1)

else xuat;{thủ tục Xuat viết chu trình con mã ra màn hình}

bc[a,b]:=true

end;

end;

end;
```

Trong bài trên sử dụng bc là mảng 2 chiều kiểu Boolean: bc[x,y]:=true nếu con mã chưa đi qua ô [x,y]. Trong thủ tục này, số bước duyệt cho một lần lựa chọn là 25, để giảm bớt số lần duyệt ta có thể căn cứ vào quy luật di chuyển cuả con mã sau một bước đã phân tích ở trên. Thiết lập mảng C, D lưu trữ toạ độ tương ứng 8 khả năng di chuyển của con mã:

```
c[1]:=2;c[2]:=1;c[3]:=-1;c[4]:=-2;c[5]:=-2;c[6]:=-1;c[7]:= 1; c[8]:= 2;

d[1]:=1;d[2]:=2; d[3]:=2;d[4]:= 1;d[5]:=-1;d[6]:=-2;d[7]:=-2; d[8]:=-1;
```

Mảng 2 chiều bc[x,y] ghi nhận con mã đã đi đến vị trí (x,y) sau bước di chuyển thứ i. Khởi đầu các giá trị của mảng bc[x,y]=0. Vị trí ban đầu con mã (io,jo).

```
Procedure Try_măi:byte);

begin

if i>63 then xuat {thủ tục Xuat viết chu trình con mã ra màn hình}

else

for j:=1 to 8 do

begin

x:=i0+c[j];
```

```
y:=j0+d[j];

then

begin

bc[x,y]:=i;

i0:=x;

j0:=y;

Try_măi+1);

i0:=x-c[j];

j0:=y-d[j];

bc[x,y]:=0;

end;

end;

end;
```

Từ đây ta đặt ra các bài toán khác, chẳng hạn:? Trên bàn cờ n x n ô ($n \geq 3$) ta đặt con mã vào một vị trí (io,jo) bất kỳ trên bàn cờ. Hỏi trên bàn cờ đó còn sắp được thêm bao nhiêu con mã để không con nào ăn được con nào?. Hoặc cho trước vị trí xuất phát (io,jo) và một ô đích cần tới (x,y) trên bàn cờ. Liệu có thể tính trước được sẽ có bao nhiêu đường đi khác nhau với cùng một số bước tối thiểu?... Rất mong được trao đổi với các bạn.