Tổ chức dữ liệu

1. Tổ chức

```
Type
        index=word;
        typeflag=integer; // trong trường hợp có hơn 2 trường hợp
        Typereal=real;
//Điểm:
        Point = record
                x, y: Typereal;
        end;
//Đường thẳng:
        Line = Record
                p1, p2: point;
        end;
//Đa giác:
        Polygon = array[1..n] of point;
```

- ***Để thuận lợi thì khi biểu diễn đa giác ta nên thêm hai đỉnh ở đầu và cuối: đỉnh 0 bằng đỉnh n và đỉnh n + 1 bằng đỉnh 1.***

- ***Từ đây ta cũng thống nhất với cách khai báo này cho các đoạn chương trình có thể dùng đến***.

2. Kiểu số thực

- Xử lý hình học hầu hết liên quan đến số thực.
- Bảng dưới đây là những kiểu số thực mà Pascal có sẵn:

Kiểu	Giới hạn	Chữ số có nghĩa	Kích thước (Byte)
Single	1.5e-45..3.4e38	7-8	4
Real	2.9e-39..1.7e38	11-12	6
Double	5.0e-324..1.7e308	15-16	8
Extended	3.4e-4932..1.1e4932	19-20	10

- Đặc biệt, mặc dù chỉ khi ta dùng Double hoặc Extended ta mới phải khai báo biên dịch {$N+}, nhưng ta nên lúc nào cũng làm như vậy. Vì khi đó máy tính sẽ dùng bộ đồng xử lý toán học, các phép toán với số thực sẽ thực hiện nhanh chẳng kém gì so với số nguyên (thậm chí còn nhanh hơn nếu ta dùng kiểu số thực Double).

- So sánh "=" giữa 2 số thực không thể thực hiện bằng phép "=" có sẵn.

```
Function equal(a,b:sothuc):Boolean;
Begin
        If abs(a-b)<esp then exit(true) // tùy theo bài toán mà chọn esp
        Exit(false);
End;
```

- Lưu ý: phép chia không (Dvision by zero)

  Ví dụ: điều kiện 3 điểm thẳng hàng $A(X_A,Y_A)$, $B(X_B,Y_B)$, $C(X_C,Y_C)$

$$\frac{X_A - X_B}{Y_A - Y_B} = \frac{X_A - X_C}{Y_A - Y_C} \ (*)$$

Khi lập trình, ta nên dùng:

$$(X_A - X_B) * (Y_A - Y_C) = (X_A - X_C) * (Y_A - Y_B) \ (**)$$

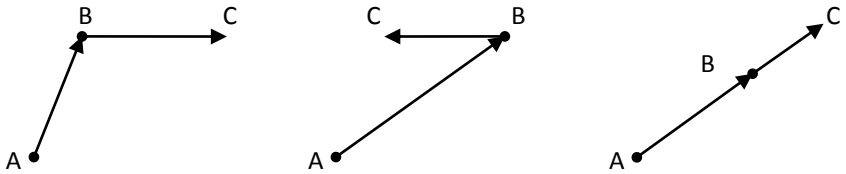
Phương pháp hình học

1. Khoảng cách giữa 2 điểm

```
function Dist(p1, p2: Point): typeReal;
begin
        Dist := Sqrt(Sqr(p1.x – p2.x) + Sqr(p1.y – p2.y));
end;
```

2. Vị trí tương đối giữa 3 Điểm

- 3 khả năng xãy ra:



$$k = \begin{vmatrix} B.x - A.x & C.x - B.x \\ B.y - A.y & C.y - B.y \end{vmatrix} = (B.x - A.x)(C.y - B.y) - (B.y - A.y)(C.x - B.x)$$

$$k = \begin{cases} 0 \rightarrow thẳng\ hàng \\ < 0 \rightarrow rẽ\ phải \\ > 0 \rightarrow rẽ\ trái \end{cases}$$

```
function CCW(A, B, C: Point): typeflag;
var
    k: typeReal;
begin
    k := (B.x − A.x) * (C.y − B.y) − (B.y − A.y) * (C.x − B.x);
    if equal(k,0) then CCW := 0 // c thẳng hàng ab
    else
            if k > 0 then CCW := 1 // c bên trái
            else CCW := -1; // c bên phải
end;
```

3. Phương trình đường thẳng tổng quát

- Phương trình đường thẳng đi qua hai điểm phân biệt p_1, p_2 có dạng:

$f(x,y) = (x - p_1.x)*(p_2.y - p_1.y) - (y - p_1.y)*(p_2.x - p_1.x) = 0$ (1)

- Viết dưới dạng tổng quát :

$Ax + By + C = 0$ (2)

Ta có: A= (p_2.y- p_1.y) ; B=(p_1. x –p_2 x) ; C=(p_2.x*p_1.y-p_1.x*p_2.y)

```
procedure Extract(p1, p2: Point; var a, b, c: typeReal);
begin
  a := p2.y - p1.y;
  b := p1.x - p2.x;
  C :=( p2.x*p1.y-p1.x*p2.y) ;
end;
```

⇨ Hàm tính f(x,y)

```
function ff(M, p1, p2: Point):typereal;
begin
  Extract(p1, p2, a, b, c);
  Exit(a*M.x+b*M.y-c);
end;
```

4. Khoảng cách giữa điểm và đường thẳng

- (d) là đường thẳng có phương trình: **Ax + By + C = 0**
- Khoảng cách từ điểm p đến đường thẳng (d) :

$$h = \frac{|Ap.x + Bp.y + C|}{\sqrt{A^2 + B^2}} = \frac{|f(p.x, p.y)|}{\sqrt{A^2 + B^2}} (3)$$

```
function DistPL(p: Point;a,b,c: typedata): typereal;
begin
        DistPoLi := abs(a*p.x+b*p.y+c)/Sqrt(Sqr(a)+ Sqr(b)));
end;
```

5. Vị trí tương đối giữa điểm và đường thẳng

- Cho 3 điểm p_1, p_2, M
- Vị trí tương đối giữa M và so với vector $\overrightarrow{p_1, p_2}$ xác định như sau:

VT=(p_2.x-p_1.x)(M.y-p_1.y)-(p_2.y-p_1.y)(M.x-p_1.x) (4)

- Nếu VT>0 thì M bên trái véctơ $\overrightarrow{p_1, p_2}$
- Nếu VT<0 thì M bên phải véctơ $\overrightarrow{p_1, p_2}$
- Nếu VT=0 thì M nằm trên véctơ $\overrightarrow{p_1, p_2}$

```
function PosPoVec(M,p1, p2: Point): typeflag;
var VT:typereal;
```

```
begin
        VT=(p2.x-p1.x)*(M.y-p1.y)-(p2.y-p1.y)*(M.x-p1.x);
        If equal(VT,0) then exit(0) // nằm trên vector
        Else if VT>0 then exit(-1) // điểm M bên trái vector
             Else exit(1); // điểm M bên phải vector
end;
```

⇨ **Xác định điểm M có thuộc đoạn thẳng p_1p_2**

- M thỏa 2 điều kiện sau:
  - M nằm trên đường thẳng p_1p_2
  - Tọa độ M thỏa : (M.x>=min(p_1.x,p_2.x)) and (M.x<=max(p_1.x,p_2.x)) and (M.y>=min(p_1.y,p_2.y)) and (M.y<=max(p_1.y,p_2.y))) ;

```
function PoInLi(M,p1, p2: Point): boolean;
var VT:typeflag;
begin
      VT:= PosPoVec(M,p1, p2);
      Exit((VT=0) and (M.x>=min(p1.x,p2.x)) and (M.x<=max(p1.x,p2.x)) and (M.y>=min(p1.y,p2.y))
and (M.y<=max(p1.y,p2.y))) ;
end;
```

⇨ **Xác định điểm M có thuộc tia AB**

- Điểm M thuộc tia AB nếu M thuộc đường thẳng AB và $\overrightarrow{AM} = k\overrightarrow{AB}$ $với\ k \geq 0$:
  F(M.x,M.y)=0, (M.x-A.x)(B.x-A.x)>=0 và (M.y-A.y)(B.y-A.y)>=0

```
function PoInRay(M,A,B: Point): boolean;
var VT:typeflag;
begin
      VT:= PosPoVec(M,A,B);
      Exit((VT=0) and ((M.x-A.x)*( B.x-A.x)>=0) and ((M.y-A.y)*( B.y-A.y)>=0));
end;
```

⇨ **Xác định vị trí tương đối giữa 2 điểm M_1,M_2 so với thẳng p_1p_2**

```
function Pos2PoLi(M1,M2,p1, p2,: Point): boolean;
var VT1, VT2:typeflag;
begin
      VT1:= PosPoVec(M1,p1, p2);
      VT2:= PosPoVec(M2,p1, p2);
      Exit(VT1 *VT2>=0); // nằm cùng phía
end;
```

6. Vị trí tương đối của 2 thẳng

- Cho 4 điểm A, B, C, D. Vị trí tương đối giữa 2 đường thẳng qua 2 điểm AB và qua 2 điểm CD được xác định như sau:
  - Tính hệ số A_1, B_1, C_1 của đường thẳng AB.

- Tính hệ số A_2, B_2, C_2 của đường thẳng CD.
- Tính

$$d = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}; d_x = \begin{vmatrix} -c_1 & b_1 \\ -c_2 & b_2 \end{vmatrix};\ d_y = \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}$$

- Nếu D<>0 thì cắt nhau
- Ngược lại
    - Nếu (dx=0) and (dy=0) thì trùng nhau
      Ngược lại song song

```
function Pos2Li(var I:Point;A,B,C,D: Point): integer;
var
    a1, b1, c1, a2, b2, c2:typereal;
    d, dx, dy: typereal;
Begin
    Extract(A,B,a1, b1, c1);
    Extract(C,D,a2, b2, c2);
    d:=a1*b2- a2*b1;
    dx:= c2*b1- c1*b2;
    dy:= a1*c2- a2*c1;
    If equal(d,0) then
            If equal(dx,0) and equal(dy,0) then exit(0) // trùng nhau
            Else exit(-1) // song song
    Else  // d<>0
        Begin
        I.x:=dx/d; I.y:=dy/d;
        exit(1); // cắt nhau tại 1 điểm
        end

End;
```

⇨ **<u>Xác định 2 đoạn thẳng có giao nhau?</u>**

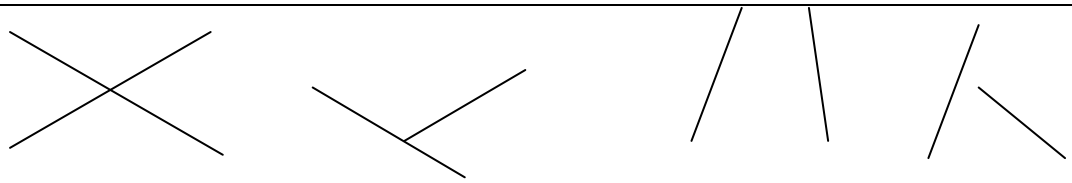
a. <u>Thuật toán 1</u>

2 đoạn thẳng giao nhau nếu thỏa điều kiện:

- Hai đường thẳng qua 2 điểm đó phải cắt nhau tại I
- Và I thuộc 2 đoạn thẳng

```
function Intersect1(A,B,C,D: Point; var I:Point): boolean;
Begin
    Exit((Pos2Li(I,A,B,C,D)=1) and PoInLi(I,A,B) and PoInLi(I,C,D);
End;
```

b. <u>Thuật toán 2</u>



- Trường hợp cắt nhau:
  - Đầu mút của đoạn thẳng nằm trên đoạn thẳng kia
  - 2 điểm của đoạn thẳng nằm khác phía với đường thẳng kia

```
function Intersect2(A,B,C,D: Point): boolean;
Begin
        If PoInLi(M,p1, p2: Point):
        If PoInLi (C,A,B) or PoInLi (D,A,B) or PoInLi(A,C,D) or PoInLi(B,C,D)
                Then exit(true);
        If (not Pos2PoLi(A,B,C,D)) and (not Pos2PoLi(C,D,A,B)) then exit(true);
        Exit(false);
End;
```

7. Tính góc

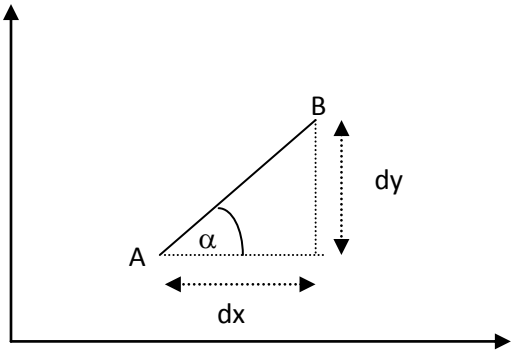
a. Góc giữa 2 đường thẳng

- Cho 2 đường thẳng: $a_1x+b_1x+c_1=0$ và $a_2x+b_2x+c_2=0$

$$\cos\alpha = \frac{|a_1a_2 + b_1b_2|}{\sqrt{{a_1}^2 + {b_1}^2}\sqrt{{a_2}^2 + {b_2}^2}} \; với \; \alpha\epsilon\left[0,\frac{\pi}{2}\right]$$

$$|a_1a_2 + b_1b_2| = 0 \; thì \; 2 \; đường \; thẳng \; vuông \; góc.$$

b. Góc giữa đường thẳng đi qua 2 điểm so với trục hoành



- Có thể tính góc này bằng cách dùng hàm arctan(dx/dy) có sẵn nhưng có vẻ hơi chậm hơn nữa không xét được góc từ [0..360]
- Chương trình sau đây giúp ta giải quyết các nhược điểm trên:

```
function theta(A,B: Point): typereal;
var goc,dx,dy:typereal;
Begin
        dx:=B.x-A.x; dy:=B.y-A.y;
        if equal(dx,0) and equal(dy,0) then goc:=0
```

```
        else goc:=dy/(abs(dx)+ abs(dy));
        if dx<0 then goc:=2-goc
        else if dy<0 then goc:=4+t;
        exit(goc*90);
End;
```

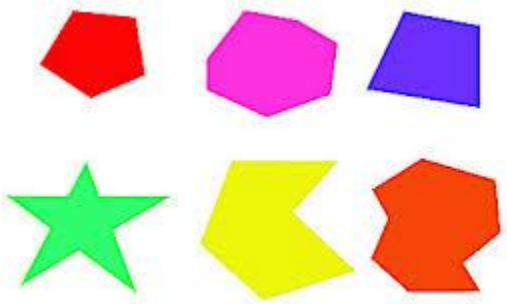
Đa giác

1. Một số định nghĩa

1.1 Đường gấp khúc

Một đường gấp khúc trên mặt phẳng gồm 1 dãy liên tiếp các đoạn thẳng $[A_1,A_2]$, $[A_2,A_3]$,…, $[A_{k-1},A_k]$, mỗi đoạn thẳng được gọi là cạnh, các đầu mút của các đoạn thẳng gọi là đỉnh.

1.2 Đa giác



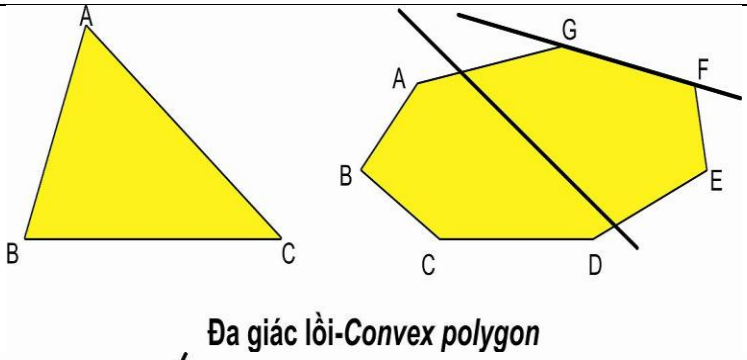
Một đa giác là một đường gấp khúc khép kín tức điểm A_k trùng với điểm A_1.

1.3 Đa giác tự cắt

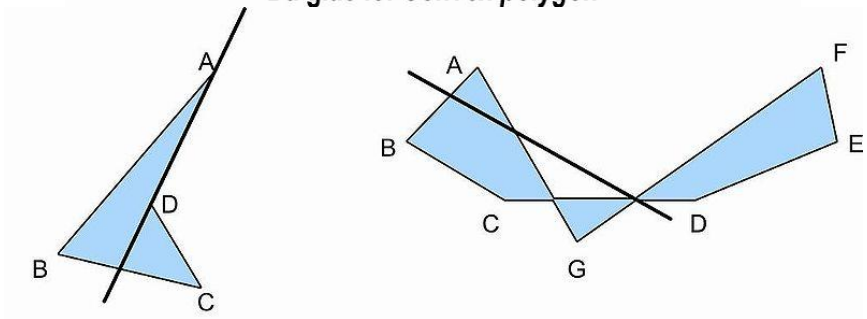
Một đa giác được gọi là tự cắt nếu có hai cạnh không liên tiếp có điểm chung.

1.4 Đa giác lồi

Một đa giác lồi được gọi là lồi nếu đa giác luôn nằm cùng một phía đối với đường thẳng đi qua một cạnh bất kỳ. Đa giác lồi là đa giác không tự cắt.



Đa giác lồi-*Convex polygon*



Đa giác lõm đơn
Simple concave polygon

Đa giác lõm phức
Simple complex polygon

Đa giác lõm-*Concave polygon*

2. Định lý về bao lồi

Với một tập hữu hạn M các điểm trên mặt phẳng ta luôn tìm được một con H của M sao cho H là tập các đỉnh của đa giác lồi P mà mọi điểm của M đều thuộc đa giác này.

3. Một số bài toán cơ sở

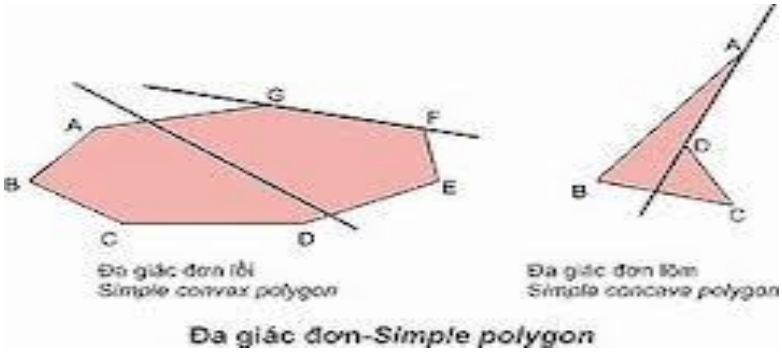
3.1 Tính diện tích một đa giác

$$S=\frac{1}{2}\left|\sum_{i=1}^{n}(x_{i+1}-x_i)(y_{i+1}+y_i)\right|$$

```
Function  Area(P:polygon;n:index):typeReal;
Var
   i:index;
   S:typeReal;
Begin
   S:=0;
   For i:=1 to n do
            S:= S+(P[i+1].x-P[i].x)*(P[i+1].y+P[i].y)/2;
   Exit(abs(S));
End;
```

3.2 Kiểm tra đa giác lồi

- Dựa theo định nghĩa



Đa giác đơn-Simple polygon

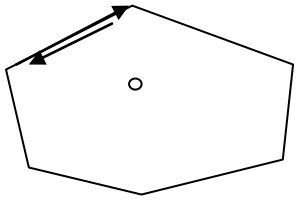
```
Function  Convex (P:polygon;n:index):boolean;
Var     i,j,l,k:index;
Begin
        For i:=1 to n do
                Begin
                l:=i+1; if l=n+1 then l:=1; // đỉnh kế với i
                k:=i+2; if l=n+1 then k:=1; // đỉnh xét cùng phía
                for j:=1 to n do // vét tất cả các đỉnh
                        if (j<>i) and (j<>k) and (j<>l) and (not Pos2PoLi(P[i],P[l],P[k],P[j]))
                                                                then exit(false);
                end;
         exit(true);
End;
```

3.3 Vị trí tương đối một điểm và đa giác

- Điểm thuộc đa giác nếu điểm nằm trên các cạnh hoặc thuộc miền đa giác.

c. Trường hợp đa giác lồi

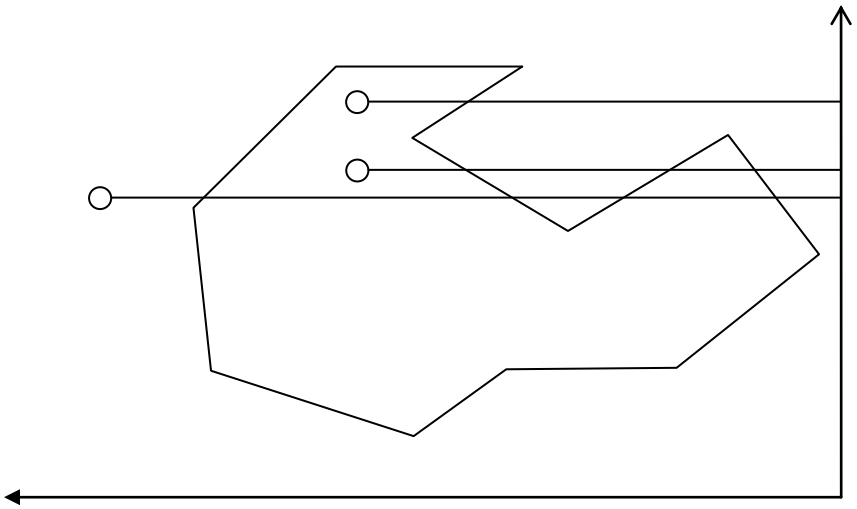
- Ta thấy nếu xét các cạnh của đa giác theo 1 chiều nào đó thì điểm M thuộc đa giác nếu nằm cùng 1 bên (trái hoặc phải) với mọi vector cạnh cuả đa giác.



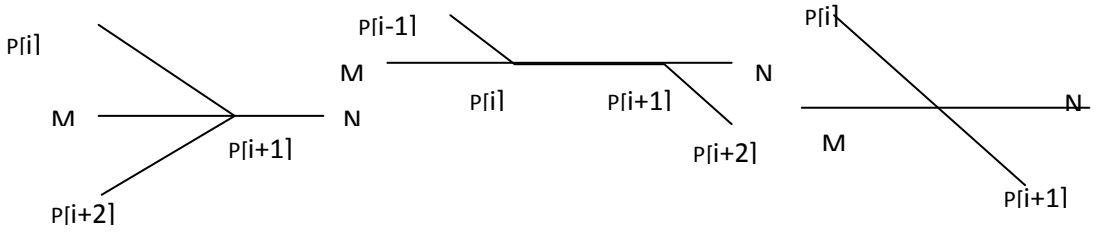
```
Function  Inside (P:polygon;n:index;M:point):boolean;
Var     i:index; VT, VT1:integer;
Begin
        If PoInLi(P[1], P[2],M) then exit(true);
```

```
        VT:=PosPoVec(P[1],P[2],M);
        For i:=2 to n do
                Begin
                If PoInLi(P[i], P[i+1],M) then exit(true);
                VT1:=PosPoVec(P[i],P[i+1],M);
                If (VT* VT1)<0 then exit(false);
                end;
         exit(true);
End;
```

d. Trường hợp đa giác bất kỳ



- Vẽ trục song với trục tung với tọa độ x=max{các hoành độ}+1
- Vẽ đoạn thẳng song song với trục hoàng và cắt trục vẽ bên trên. Ta nhận xét nếu số giao điểm với đa giác là số lẽ thì điểm thuộc đa giác. Còn ngược lại điểm nằm ngoài đa giác.
- Các trường hợp cắt sau ta chỉ tính cắt tại 1 giao điểm:



Trường hợp 1 **Trường hợp 2** **Trường hợp 3**

```
Function  Inside (P:polygon;n:index;M:point):boolean;
Var
        I,count:index; I,N point; xmax:typereal;
Begin
        P[n+2]:=P[2]; // phần tử cầm canh. Ta có sẵn P[0]:=P[n], P[n+1]:=P[1]
```

```
        Count:=0; // đếm số giao điểm
        x_max:=findxmax(P); // tìm hoành độ lớn nhất của đa giác
        N.x:=x_max+1; // điểm N
        N.y:=M.y;
        For i:=1 to n do
                Begin
                If PoInLi(P[i],P[i+1],M) then exit(true); // M thuộc cạnh
                If not PoInLi(M,N,P[i]) then
                        begin
                        If (not PoInLi(M,N, P[i+1])) and (Intersect(M,N,P[i],P[i+1],I))
                                then inc(count) // trường hợp 3
                        Else if not PoInLi(M,N, P[i+2]) and (not Pos2PoLi(M,N,P[i],P[i+2]))
                                then inc(count); // trường hợp 1
                        End
                Else  if PoInLi(M,N, P[i+1]) and (not Pos2PoLi(M,N,P[i-1],P[i+2]))
                                Then inc(count) // trường hợp 2
                End;
         If (count mod 2 <> 0) then exit(true);
        Exit(false);
End;
```

3.4 Tìm bao lồi có chu vi nhỏ nhất

- Cho tập hữu hạn M.
- Tìm H tập con của M và H là đa giác lồi.

a. Thuật toán 1

- Sắp xếp tăng dần theo hoàng độ. Bởi đỉnh đầu tiên này luôn thuộc tập H.
- H={M[1]}
- Lặp lại quá trình sau cho đến khi không còn chọn được
    - Giả sử A_1, A_2,...,A_i là các điểm được chọn thỏa yêu cầu: với mọi j<i thì tập M nằm cùng phía với A_jA_{j+1}.
    - Ta chọn A_{i+1} thỏa điều kiện: A_{i+1} chưa được chọn và tập M nằm cùng phía với A_iA_{i+1}

```
procedure  ConvexSet(M:SetPoint;n:index;var H:polygon; var k);
Var
        i,j:index;
        dd: array[1..n] of boolean;
Begin
        Sort_Y_Axis(M); // sắp tăng theo tung độ
        Fillchar(dd,sizeof(dd),false); // các đỉnh đã xét
        Stop:=false; k:=1; H[k]:=M[1]; dd[1]:=true;
        While not stop do
                Begin
                Stop:=true;
                // tìm điểm thỏa
                For i:=1 to n do
```

```
                If dd[i] then
                        Begin
                        Th:=true;
                        for j:=1 to n do // vét tất cả các đỉnh
                                if (j<>k) and (j<>i) and (not Pos2PoLi(H[k],M[i],H[1],M[j])
                                                                then
                                                                        begin
                                                                        th:=false;
                                                                        break;
                                                                        end;
                        if th then
                                begin
                                inc(k); H[k]:=M[i]; stop:=false; dd[i]:=true; break;
                                end;
                        end;
        end;
End;
```

b. <u>Thuật toán bọc gói</u>

* **<u>Điểm có tung độ nhỏ nhất luôn thuộc bao lồi</u>**

- po←min(trung độ);
- W={po};
- Lặp lại
    - Từ po Quét **ngược theo chiều kim đồng hồ** gặp đỉnh đầu tiên v
    - W=W ∪ v;
    - po←v
- Until v=đỉnh đầu tiên

Thuật toán bọc gói

```
function wrap:integer;
var
    M,min,i:integer; g,minangle,v:real; t:point;
begin
    min:=1; // chỉ số của điểm thuộc bao lồi đang xét
    for i:=2 to N do if p[i].y<p[min].y then min:=i;
    M:=0; p[N+1]:=p[min]; minangle:=0.0;
    repeat
        inc(M);
        t:=p[M];p[M]:=p[min];p[min]:=t;// doi cho
        min:=N+1; v:=minangle; minangle:=360.0;
        for i:=M+1 to N+1 do // chọn min(g>v) và g chưa trong bọc gói
            begin
            g:=theta(p[M],p[i]);
            if (g>v) and (g<minangle) then
                begin
                min:=i; minangle:=g;
                end;
```

```
            end;
        until min=N+1;
        exit(M); // những điểm từ 1 đến M của p là những điểm thuộc bao lồi
end;
```

Thuật toán bọc gói

c. **Grahamscan**

- Chọn điểm có tung độ nhỏ nhất. Nếu có tung độ bằng nhau thì chọn điểm có hoành độ lớn nhất. Điểm này thuộc bao lồi.
- Sắp xếp các điểm còn lại tăng dần về góc so với điểm trên.
- Lần lượt chọn các điểm theo trật tự sắp xếp
    - Nếu điểm được chọn không tạo thành đa giác lồi với các điểm đã chọn thì lần lượt bỏ các đỉnh đã chọn cho tới khi lập thành đa giác lồi thì thôi

    Bỏ điểm vào bao lồi

Thuật toán Grahamscan

```
function Grahamscan:integer;
var
    M,min,i:integer; t:point;
begin
    min:=1; // find min(p.y) and max(p.x)
    for i:=2 to N do
        if (p[i].y<p[min].y) or ((p[i].y=p[min].y) and (p[i].x>p[min].x))  then min:=i;
    t:=p[1];p[1]:=p[min];p[min]:=t;// điểm đầu tiên của bọc gói
    pa[1]:=0.0;
    for i:=2 to N do pa[i]:=theta1(p[1],p[i]); // tinh goc cac dinh so voi dinh 1
    quicksort;
    p[0]:=p[n];
    M:=3; \\ vì tam giác là đa giác lồi
    for i:=4 to N do
        begin
        while (ccw(p[M-1],p[M],p[i])<>-1) do dec(M);
        inc(M);
        t:=p[M];p[M]:=p[i];p[i]:=t;// bỏ điểm vào bao lồi
        end;
    grahamscan:=M;
end;
```

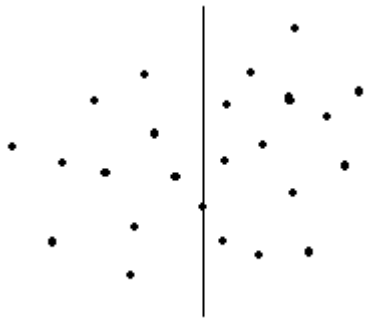
Thuật toán Grahamscan

3.5 Cặp điểm gần nhất.

Bài toán: Cho tập điểm, hãy tìm cặp điểm có khoảng cách nhỏ nhất trong tập điểm trên.

Một cách thô thiển ta có thể xét tất cả các cặp điểm và lưu lại cặp điểm có khoảng cách nhỏ nhất. Nhưng như vậy thì chi phí thuật toán sẽ là $O(N^2)$. Ta hoàn toàn có thể giải quết bài toán này với chi phí là O(NlogN) với việc áp dụng tư tưởng "chia để trị" của thuật toán Merge Sort (xin đọc phần "cấu trúc dữ liệu và giải thuật").

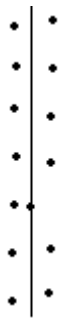
Ý tưởng thuật toán như sau: ta sắp xếp các điểm theo hoành độ (hoặc tung độ cũng được). Tại mỗi bước ta chia tập làm hai phần thì cặp điểm gần nhất sẽ nằm ở một trong hai phần hoặc là cặp điểm mà mỗi điểm thuộc một phần.



Chia để trị

Vấn đề là phải xử lý trường hợp mỗi điểm nằm ở một phần, còn trường hợp cả hai điểm đều thuộc một phần thì đã được giải quyết vì lời gọi đệ qui. Ta có thể sử dụng ngay thứ tự sắp xếp và khoảng cách min đã tìm được được để làm cận cho trường hợp xét trên hai phần. Khi xét mỗi điểm ở nửa này, nếu gặp điểm ở nửa kia có hiệu hoành độ đến nó không nhỏ hơn khoảng cách min đã tìm được thì ta dừng luôn vì tập điểm đã được sắp theo x.

Nhưng như vậy vẫn có thể gặp phải trường hợp xấu: các điểm nằm sát hai bên đường phân cách; trong trường hợp đó, nếu xử lý không tốt thì chi phí có thể sẽ là $O(N^2)$.



trường hợp xấu

Ta giải quyết vấn đề này bằng cách sắp các điểm theo tung độ y và xét tương tự như trên. Chú ý, nếu tại mỗi bước ta lại gọi thủ tục sắp xếp mỗi nửa thì chi phí thuật toán sẽ là $O(N\log^2 N)$, không phải là thực sự tốt lắm. Nhưng để ý một chút, ta thấy mỗi nửa đều được sắp xếp, hơn nữa cách làm của ta cũng đang dựa vào tư tưởng của Merge Sort, như vậy tại sao ta không sắp xếp theo kiểu Merge Sort ngay trong thủ tục đệ quy của mình. Như vậy hai công việc đều được sử lý đồng thời. Chi phí cho bài toán này giống như chi phí sắp xếp bằng Merge Sort, tỷ lệ với NlogN.

Cài đặt thuật toán này không khó nhưng khá tỉ mỉ. Ta bỏ qua bước sắp xếp tập điểm theo hoành độ x. Đầu tiên là thủ tục trộn hai phần, tiếp theo là thủ tục đệ quy tìm cặp điểm gần nhất.

Thuật toán cặp điểm gần nhất

```
procedure Merge(l, r: Integer);
var
        t: Polygon;
        i, j, m, c: Integer;
begin
        m := (l + r) div 2;
```

```
        i := l;
        j := m + 1;
        for c := 1 to r – l + 1 do
                if (j > r) or (p[i].y < p[j].y) then
                        begin
                        t[c] := p[i];
                        Inc(i);
                        end
                else
                        begin
                        t[c] := p[j];
                        Inc(j);
                        end;
                        for c := 1 to r – l + 1 do p[l + c - 1] := t[c];
end;

procedure MinDist(l, r: Integer);
var
        i, j, j1, m: Integer;
begin
        if l = r then Exit;
        m := (l + r) div 2;
        MinDist(l, m);
        MinDist(m + 1, r); {gọi đệ quy tìm min trên hai phần}
        j1 := m + 1;
        for i := l to m do
                begin
                while (j1 <= r)and(p[j1].y – p[i].y >= min) do Inc(j1);
                for j := j1 to m do
                        if p[j].y – p[i].y >= min then Break
                        else
                                if Dist(p[i], p[j]) < min then
                                        begin {cập nhật kết quả}
                                        min := Dist(p[i], p[j]);
                                        Result.x := p[i];
                                        Result.y := p[j];
                                        end;
                end;
        Merge(l, r); {trộn hai phần}
end;
```

Thuật toán cặp điểm gần nhất

Result là biến lưu kết quả, có cấu trúc giống như kiểu Line. Biến min dùng để lưu khoảng cách nhỏ nhất tìm được cho đến thời điểm hiện tại.

3.6 Tìm giao điểm của các đoạn thẳng nằm ngang và nằm dọc

Cho N đoạn thẳng ngang hoặc dọc. Hãy tìm các giao điểm của chúng.
Dữ liệu vào từ file văn bản GIAODIEM.INP.

- Dòng đầu là số N
- N dòng sau mỗi dòng 4 số nguyên là toạ độ 2 đầu mút của một đoạn thẳng.

Kết quả ghi ra file văn bản GIAODIEM.OUT. Mỗi dòng là 2 số thể hiện toạ độ một giao điểm.

Hướng dẫn:

Đặt các đoạn thẳng trong hệ trục toạ độ Đề Các. Giả sử các đoạn thẳng nằm ngang cùng phương trục hoành, các đoạn thẳng dọc cùng phương trục tung. Dùng một đường thẳng (Δ) nằm ngang quét từ dưới lên trên (từ đầu mút đoạn thẳng có tung độ thấp nhất tới đầu mút đoạn thẳng có tung độ cao nhất). Giao điểm xuất hiện khi (Δ) chứa 1 đoạn thẳng nằm ngang mà đoạn thẳng nằm ngang này lại chứa 1 điểm của một đoạn dọc nào đó. Điểm của đoạn thẳng dọc xuất hiện trên (Δ) khi (Δ) bắt đầu chạm tới đầu mút dưới của đoạn thẳng dọc và nó sẽ biến khỏi (Δ) khi (Δ) đi qua điểm mút trên của đoạn thẳng dọc này. Vì vậy có thể dùng dữ liệu kiểu TREE để quản lý hoành độ các đầu mút của các đoạn thẳng dọc (mỗi đoạn thẳng dọc 2 đầu mút): Nếu (Δ) gặp đầu thấp của đoạn thẳng dọc ta thêm hoành độ của đoạn này vào cây nhị phân tìm kiếm (nếu hoành độ này nhỏ hơn hoành độ nút x vừa dựng trước đó thì nó là con trái của nút x ngược lại nó là con phải của x), nếu (Δ) gặp đầu mút cao của đoạn thẳng dọc thì ta xoá nút là mút thấp của đoạn dọc này ra khỏi cây. Nếu (Δ) gặp đoạn thẳng ngang thì chính là lúc ta phải tìm kiếm giao điểm: giao điểm là điểm có hoành độ bằng một nút trên cây hiện tại có trong khoảng từ hoành độ nút trái của đoạn thẳng ngang cho đến hoành độ nút phải của đoạn thẳng ngang này, còn tung độ của giao điểm là tung độ của đoạn dọc ứng với nút đang xét.

Hướng dẫn cài đặt cụ thể:

Sắp tăng các điểm mút các đoạn thẳng theo tung độ của chúng.

Xây dựng hai cây nhị phân: Cây y chứa tung độ các điểm mút các đoạn thẳng. Cây x chứa hoành độ các điểm mút các đoạn thẳng dọc.

Xây dựng cây y như sau: Mỗi nút của cây tương ứng một đầu mút của một đoạn thẳng, đó là một record gồm các trường: dau (có giá trị bằng 1 nếu là đầu mút dưới của đoạn thẳng dọc, có giá trị bằng 2 nếu là đầu mút trên của đoạn thẳng dọc, có giá trị bằng 3 nếu là đầu của đoạn thẳng nằm ngang), stt (thuộc đoạn thẳng thứ mấy), key (tung độ của điểm mút), l (cây con trái), r (cây con phải). Mỗi khi đọc được 4 toạ độ x1, y1, x2, y2 của một đoạn thẳng i, thì chèn vào cây nhị phân y một nút tương ứng.

Sau đó duyệt cây y để xây dựng cây x và dùng cây x để tìm kiếm giao điểm. Duyệt cây y theo kiểu trung tự: vòng lặp gồm các bước: gọi duyệt cây trái, xử lý nút giữa, gọi duyệt cây phải), khi xử lý nút giữa (trên cây y) nếu gặp nút là đầu mút dưới của đoạn thẳng dọc thì chèn vào cây x, nếu là đầu mút trên của đoạn thẳng dọc thì xoá nó khỏi cây x, nếu gặp nút là đầu mút của đoạn thẳng ngang thì duyệt cây x hiện tại (duyệt kiểu tiền tự) để tìm giao điểm.