# CHUYÊN ĐỀ: ỨNG DỤNG MỘT SỐ ĐỐI TƯỢNG HÌNH HỌC PHẲNG TRONG TIN HỌC

## I. CƠ SỞ LÝ THUYẾT

### 1. Biểu diễn các đối tượng hình học cơ bản trong máy tính

Như chúng ta đã biết, các khái niệm: điểm, đoạn thẳng, đường thẳng là những khái niệm cơ sở nhất trong hình học nói chung. Để biểu diễn các đối tượng nói trên trong tin học ta thường sử dụng cách biểu diễn phổ biến như sau:

#### 1.1. Điểm

```
Point=record
           x,y:real;
       end;
```

#### 1.2. Đường thẳng

```
Line=record
         P1,P2:Point;
     end;
```

#### 1.3. 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. Khi đó, biểu diễn đa giác lại như sau:

```
  Polygon=Array[0..n+1] of Point;
```

**Mộ số khái niệm**

***Đường gấp khúc***

Một đường gấp khúc trên mặt phẳng gồm một dãy liên tiếp các đoạn thẳng $[P_1,P_2]$, $[P_2,P_3]$,…, $[P_{k-1},P_k]$, mỗi đoạn thẳng được gọi là cạnh, các đầu mút của các đoạn thẳng gọi là đỉnh.

***Đa giác***

Một đa giác là một đường gấp khúc khép kín tức điểm $P_k$ trùng với điểm $P_1$.

***Đa giác tự cắt***

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

***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 kì của đa giác. Đa giác lồi là đa giác không tự cắt.

Trong các bài toán hình học, phần lớn các đối tượng đều được thể hiện trên hệ trục tọa độ Descartes, việc biểu diễn các thành phần tọa độ có thể sử dụng cả kiểu số thực và kiểu số nguyên của ngôn ngữ lập trình. Một số kiểu dữ liệu của Pascal hay sử dụng.

+ Kiểu số nguyên:

| ***Tên kiểu*** | ***Phạm vi*** | ***Dung lượng*** |
|---|---|---|
| ***Shortint*** | -128 → 127 | **1 byte** |
| ***Byte*** | 0 → 255 | **1 byte** |
| ***Integer*** | -32768 → 32767 | **2 byte** |
| ***Word*** | 0 → 65535 | **2 byte** |
| ***LongInt*** | -2147483648 → 2147483647 | **4 byte** |

+ Kiểu số thực:

| ***Tên kiểu*** | ***Phạm vi*** | ***Dung lượng*** |
|---|---|---|
| ***Single*** | $1.5\times10^{-45} \rightarrow 3.4\times10^{+38}$ | **4 byte** |
| ***Real*** | $2.9\times10^{-39} \rightarrow 1.7\times10^{+38}$ | **6 byte** |
| ***Double*** | $5.0\times10^{-324} \rightarrow 1.7\times10^{+308}$ | **8 byte** |
| ***Extended*** | $3.4\times10^{-4932} \rightarrow 1.1\times10^{+4932}$ | **10 byte** |

Một trong những khó khăn gặp phải khi giải quyết các bài toán hình học là ta phải làm việc với số thực. Khi làm việc với số thực bao giờ ta cũng phải chấp nhận với những sai số nhất định. Vì vậy khi so sánh hai giá trị với nhau ta chú ý không được dùng dấu "=", mà phải xét trị tuyệt đối hiệu hai giá trị với một giá trị Epsilon nào đó. Ở đây, Epsilon là một số tương đối bé, tuỳ vào yêu cầu của bài toán mà ta có chọn lựa về giá trị của nó.

```
Function Equal(x,y:real):Boolean;
Begin
   Equal:= abs(x-y)<=Eps;
end;
```

**2. Các phương pháp hình học**

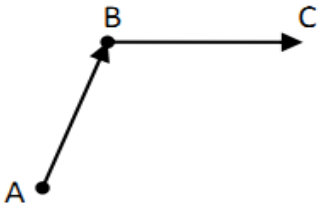
**2.1. Khoảng cách giữa hai điểm**

Cho 2 điểm $A(x_A;y_A)$ và $B(x_B;y_B)$, khoảng cách giữa hai điểm A, B được tính: $AB=\sqrt{(x_A-x_B)^2+(y_A-y_B)^2}$

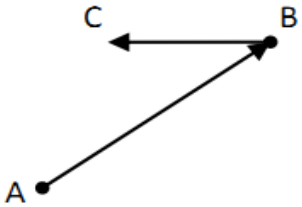
```
Function Dist(p1,p2:Point):Real;
Begin
        Dist:=sqrt(sqr(p1.x-p2.x)+sqr(p1.y-p2.y));
end;
```

## 2.2. Vị trí tương đối giữa 3 Điểm

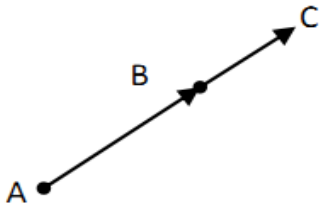
Cho 3 điểm $A(x_A;y_A)$, $B(x_B;y_B)$ và $C(x_C,y_C)$, Có 3 khả năng xảy ra:




Rẽ phải




Rẽ trái




Đi thẳng

$$k=\begin{vmatrix} x_B-x_A & x_C-x_B \\ y_B-y_A & y_C-y_B \end{vmatrix}=(x_B-x_A)(y_C-y_B)-(y_B-y_A)(x_C-x_B)$$

Nếu *k = 0* thì 3 điểm A, B, C thẳng hàng

Nếu k < 0 thì rẽ phải

Nếu k > 0 thì rẽ trái

Hàm CCW sau trả ra -1 nếu là rẽ phải, 1 nếu rẽ trái, 0 nếu 3 điểm thẳng hàng.

```
Function CCW(p1,p2,p3:Point):Integer;
Var    a1, b1, a2, b2, t: Real;
begin
      a1 := p2.x - p1.x;
      b1 := p2.y - p1.y;
      a2 := p3.x - p2.x;
      b2 := p3.y - p2.y;
      t := a1*b2 - a2*b1;
      if Abs(t) < Eps then CCW := 0
      else    if t > 0 then CCW := 1
              else CCW := -1;
end;
```

## 2.3. Phương trình đường thẳng qua hai điểm

Phương trình đường thẳng qua 2 điểm $A(x_A;y_A)$ và $B(x_B;y_B)$ có dạng:

$$\frac{x-x_A}{x_B-x_A}=\frac{y-y_B}{y_B-y_A}\Leftrightarrow(x-x_A)(y_B-y_A)-(y-y_B)(x_B-x_A)=0$$

Để biểu diễn đường thẳng ta có thể biểu diễn bằng toạ độ hai điểm trên đường thẳng đó. Nhưng đôi khi để tiện lợi cho tính toán, ta phải có được phương trình dưới dạng tổng quát của nó.

```
Procedure PointToLine(p1,p2:Point; Var a,b,c:Real);
Begin
        a:=p2.y-p1.y;
        b:=p1.x-p2.x;
        c:=-(a*p1.x+b*p1.y);
end;
```

### *2.4. Diện tích đa giác*

Trong mặt phẳng với hệ tọa độ Đề-các vuông góc, Cho đa giác $P=P_1P_2...P_n$ không tự cắt, trong đó điểm $P_i$ có tọa độ $(x_i, y_i)$. Bổ sung thêm điểm $P_0$ trùng với $P_n$ và điểm $P_{n+1}$ trùng với $P_1$, khi đó diện tích đa giác được tính: $S=\frac{1}{2}\left|\sum_{i=1}^{n}(x_{i+1}-x_i)(y_{i+1}+y_i)\right|$

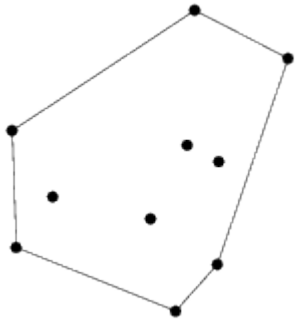
```
Function Area(P:Polygon):Real;
Var i:longint; S:real;
begin
        P[0]:=P[n];
        P[n+1]:=P[1];
        S:=0;
        for i:=1 to n do
           S:=S+(P[i+1].x-P[i].x)*(P[i+1].y+P[i].y);
        Area:=abs(S)/2;
end;
```

***Chú ý:*** - Ta có thể xác định phép đánh số các đỉnh đa giác là thuận hay nghịch dựa vào công thức tính diện tích đa giác: Bỏ dấu giá trị tuyệt đối trong công thức khi đó S>0 tương ứng với phép đánh số thuận và S<0 tương ứng với phép đánh số nghịch.

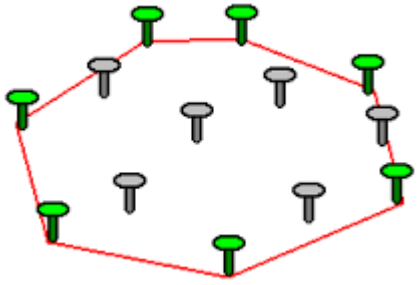
- Đường biên đa giác phải là một đường gấp khúc khép kín, không tự cắt. Công thức tính diện tích trên sẽ sai nếu dữ liệu vào không thỏa mãn điều kiện này. Ví dụ một đa giác gồm 4 đỉnh (0;0), (1;1), (0; 1) và (1;0), áp dụng công thức tính diện tích trên cho đa giác này là 0, tuy nhiên trên thực tế thì diện tích miền này là 0.5.

### *2.5. Bao lồi*

Trong hình học tính toán**,** bao lồi **(**convex hull**)** của một tập điểm là tập lồi nhỏ nhất (theo diện tích, thể tích, ...) mà tất cả các điểm đều nằm trong tập đó.



Một cách trực quan, nếu ta coi các điểm trong một tập hợp là các cái đinh đóng trên một tấm gỗ, bao lồi của tập điểm đó có viền ngoài tạo bởi sợi dây chun mắc vào các cái đinh sau khi bị kéo căng về các phía như hình vẽ sau:



**Thuật toán tìm bao lồi trên mặt phẳng**

Bài toán tìm bao lồi của một tập điểm trên mặt phẳng là một trong những bài toán được nghiên cứu nhiều nhất trong hình học tính toán và có rất nhiều thuật toán để giải bài toán này. Sau đây là hai thuật toán phổ biến nhất:

**Thuật toán bọc gói**

Thuật toán bọc gói **(**Gift wrapping algorithm**)** hay còn gọi là thuật toán Jarvis march là một trong những thuật toán tìm bao lồi đơn giản và dễ hiểu nhất. Tên thuật toán xuất phát từ sự tương tự của thuật toán với việc đi bộ xung quanh các điểm và cầm theo một dải băng gói quà.

- Bước đầu tiên của thuật toán là chọn một điểm chắc chắn nằm trong bao lồi, ví dụ, điểm có tung độ lớn nhất (nếu có nhiều điểm cùng có tung độ lớn nhất thì có thể chọn điểm tung độ lớn nhất và hoành độ lớn nhất).
- Xuất phát từ điểm này, mục tiêu của ta là sẽ lần lượt đi đến các điểm khác cho đến khi quay trở lại điểm ta chọn lúc đầu.
- Ban đầu, ta nhìn về phía bên phải. Khi đi đến các điểm khác, ta sẽ lưu lại:
  + Điểm P mà ta đang chọn
  + Vector $\vec{v}$ chỉ hướng ta đang nhìn.
- Tiếp theo, thuật toán sẽ lặp lại liên tục các bước sau cho đến khi tìm được bao lồi.
  + Ta quay mặt theo chiều kim đồng hồ cho đến khi ta nhìn thấy một điểm, gọi điểm đó là Q.
  + Rồi ta cầm theo dải băng và đi đến điểm Q. Khi ta đến điểm đấy, ta thay:
    - $\vec{v}$ thành $\vec{PQ}$
    - P thành Q
  - Thuật toán kết thúc, khi ta quay trở về điểm ban đầu. Lúc này ta đã đi đến tất cả các đỉnh của bao lồi theo chiều kim đồng hồ.

Để xác định điểm ta nhìn thấy đầu tiên khi ta quay mặt theo chiều kim đồng hồ, ta duyệt tất cả các điểm R trong tập, ngoại trừ điểm P. Với mỗi điểm, ta xét vector $\vec{u} = \vec{PR}$, $\vec{u}$ tạo với $\vec{v}$ một góc θ nhỏ nhất sẽ tương ứng với điểm. Để tìm $\theta$ nhỏ nhất, ta tìm $cos\theta$ lớn nhất, với $cos\theta = \frac{\vec{u}.\vec{v}}{\left|\vec{u}\right|.\left|\vec{v}\right|}$

Với mỗi lần tìm điểm tiếp theo, ta cần duyệt qua tất cả các điểm trong tập, vì vậy độ phức tạp của mỗi lần tìm điểm là O(n) với n là số lượng điểm trong tập. Số lần tìm điểm tiếp theo phụ thuộc vào số lượng điểm là đỉnh của bao lồi, gọi số lượng điểm đó là h, khi đó độ phức tạp của cả thuật toán là O(n×h). Trong trường hợp xấu nhất, h=n hay tất cả các điểm trong dữ liệu vào tạo thành một đa giác lồi, độ phức tạp của thuật toán là $O(n^2)$, không đủ nhanh khi n>5000.

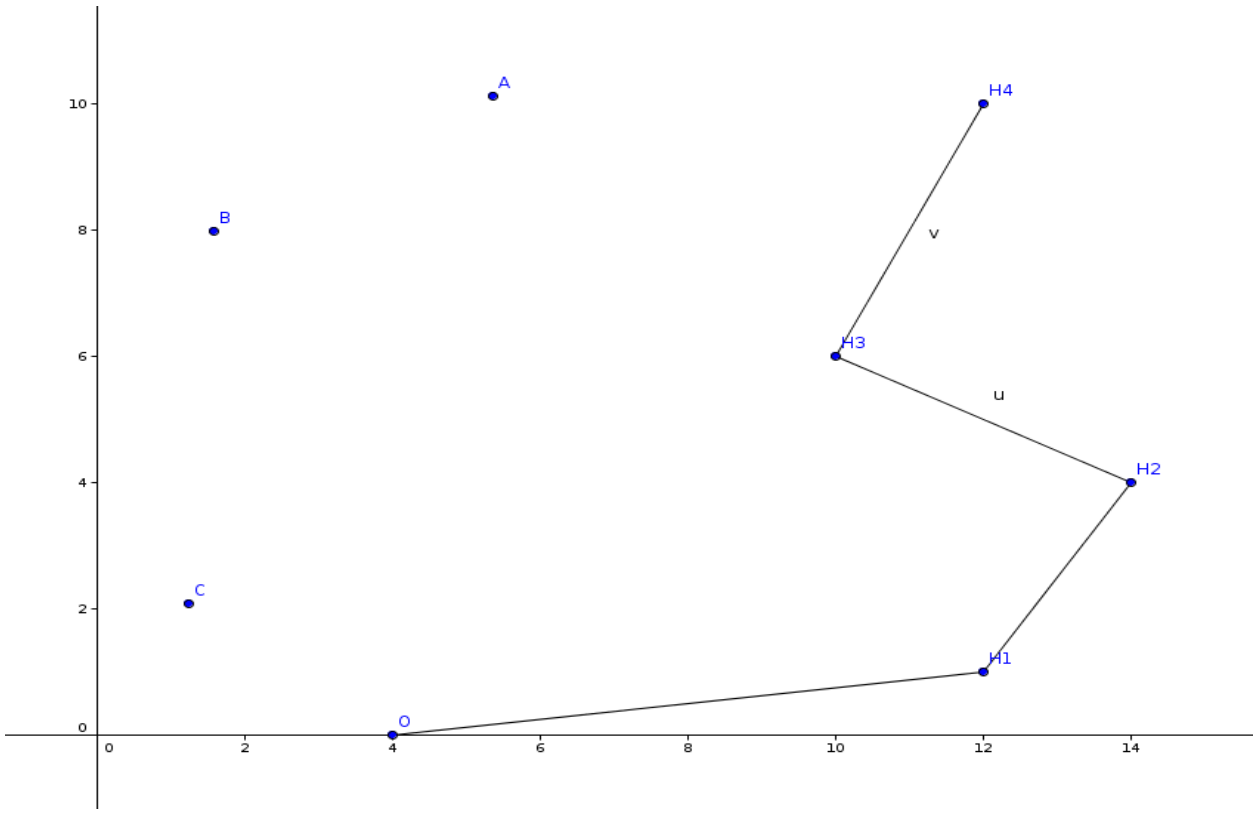
**Thuật toán Graham**

Thuật toán Graham có độ phức tạp trong trường hợp xấu nhất nhỏ hơn thuật toán bọc gói, song thuật toán Graham lại phức tạp hơn.

- Đầu tiên, ta xác định một điểm mà chắc chắn thuộc bao lồi. Thông thường, khi cài đặt người ta chọn điểm có tung độ nhỏ nhất (nếu có nhiều điểm như vậy thì chọn điểm trái nhất). Gọi điểm này là điểm O.
- Chọn hệ trục tọa độ có gốc là điểm vừa chọn, đổi các tọa độ các điểm còn lại theo hệ trục tọa độ mới (chú ý lúc cài đặt thường ta không đổi trục tọa độ, nhưng khi tính góc hoặc sắp xếp ở bước tiếp theo cần chú ý tránh nhầm lẫn).
- Tiếp theo, ta sắp xếp các điểm còn lại theo thứ tự tăng dần của góc tạo bởi trục hoành theo chiều dương và $\vec{OI}$ với I là một trong các điểm còn lại.
- Ta xét các điểm theo thứ tự ta vừa sắp xếp, với mỗi điểm ta sửa lại bao lồi H. Gọi điểm đầu tiên được cho vào bao lồi H là $H_1$, điểm cuối cùng là $H_h$ (ban đầu h=0). Khi xét mỗi điểm ta làm như sau:

1. Thêm điểm P vào cuối bao lồi H. Tức là ta tăng h lên 1 và đặt $H_h$=P.
2. Nếu h<3, xét tiếp điểm tiếp theo, ngược lại làm bước 3.
3. Xét 3 điểm $H_h$,$H_{h-1}$ và $H_{h-2}$. Có thể sau khi cho thêm điểm $H_h$, ta biết được điểm $H_{h-1}$ chắc chắn không nằm trong bao. Gọi $\vec{u} = \overrightarrow{H_{h-2}H_{h-1}}$ và $\vec{v} = \overrightarrow{H_{h-1}H_h}$. Nếu khi đi theo hướng $\vec{u}$ rồi đi theo hướng $\vec{v}$ là ta đã bẻ góc ngược chiều kim đồng hồ, hay $\vec{u} \times \vec{v} > 0$, thì cả ba điểm đều tạm thuộc bao, và ta xét tiếp điểm tiếp theo. Nhưng nếu $\vec{u} \times \vec{v} < 0$, thì góc $H_{h-2}\hat{H}_{h-1}H_h$ sẽ tạo ra đa giác lõm và điểm $H_{h-1}$ phải bị loại bỏ, có nghĩa là $H_{h-1}$ được đặt là $H_h$ và h giảm đi 1. Sau đó quay lại bước 2 cho đến khi xét hết các điểm.

Ví dụ:




- Ta đang xây dựng bao lồi, đến vị trí h=4

- Góc $H_{h-2}\hat{H}_{h-1}H_h$ lõm, nên ta cần bỏ điểm $H_3$ khỏi bao lồi

Sau quá trình trên, ta đã có một bao lồi $H_1,H_2,...,H_h$ sắp xếp ngược chiều kim đồng hồ.

Để đảm bảo ta loại bỏ điểm và thêm điểm với độ phức tạp O(1), ta có thể dùng cấu trúc dữ liệu stack.

Về độ phức tạp của thuật toán, ta thấy bước sắp xếp các điểm có độ phức tạp O(nlogn). Mỗi điểm được cho vào bao nhiều nhất một lần nên tổng độ phức tạp của các bước thêm điểm là O(n), và mỗi điểm bị loại ra khỏi bao nhiều nhất một lần nên tổng độ phức tạp của các bước xóa điểm là O(n), do đó độ phức tạp của bước xét các điểm là O(n). Vậy, độ phức tạp của thuật toán Graham là O(nlogn), phù hợp cho hầu hết các bài toán.

Cài đặt:

```
Function CCW(P1,P2,P3:Point):Integer;
Var k:real;
Begin
     k:=(P2.x-P1.x)*(P3.y-P2.y)-(P3.x-P2.x)*(P2.y-P1.y);
     if abs(k)<=Eps then CCW:=0
     else
        If k>0 then CCW:=1
```

```
        else CCW:=-1;
End;

Procedure Doicho(Var p1,p2:Point);
var Tg:Point;
begin
        tg:=p1;
        p1:=p2;
        p2:=tg;
end;

Function Check(p1,p2:Point):Boolean;
var c:integer;
begin
        c:=CCW(a[1],p1,p2);
        Check:=false;
        if c>0 then Check:=true
        else
      if   (c=0)   and   ((p1.x<p2.x)   or   ((p1.x=p2.x)   and
(p1.y<p2.y)))
           then Check:=true;
end;

Procedure QuickSort(L,R:longint);
var  x:Point; i,j:longint;
begin
        i:=L;
        j:=R;
        x:=a[(i+j) div 2];
        repeat
           while Check(a[i],x) do inc(i);
           While Check(x,a[j]) do dec(j);
           if i<=j then
              begin
                   Doicho(a[i],a[j]);
                   inc(i);
                   dec(j);
              end;
        until i>j;
        if i<R then Quicksort(i,R);
        if L<j then Quicksort(L,j);
end;

Procedure graham;
var i,k:longint;
begin
      // Tim diem co tung do nho nhat,
      // neu co nhieu tung do nho nhat, chon diem co hoanh do
nho nhat
        k:=1;
        for i:=2 to n do
          if    (a[i].y<a[k].y)    or    ((a[i].y=a[k].y)    and
(a[i].x<a[k].x)) then k:=i;
```

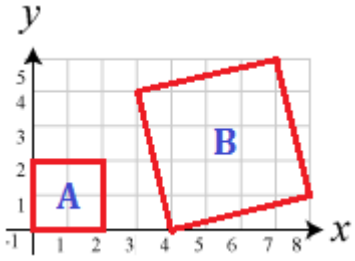
```
          Doicho(a[1],a[k]);
          Quicksort(2,n);
          a[n+1]:=a[1];
          a[0]:=a[n];
          b[1]:=a[1];
          b[2]:=a[2];
          m:=2;
          for i:=3 to n+1 do
          begin
             while   (m>=2)   and   (CCW(b[m-1],b[m],a[i])<=0)   do
dec(m);
              inc(m);
              b[m]:=a[i];
          end;
          dec(m);
end;
```

## II. MỘT SỐ BÀI TẬP MINH HỌA

### Bài 1. Đếm hình vuông

Trong mặt phẳng tọa độ, cho một lưới các điểm nguyên có kích thước n+1 dòng và m+1 cột. Điều đó có nghĩa là nếu (x, y) là một điểm nguyên trong lưới thì

$0 \le x \le m$ và $0 \le y \le n$. Người ta có thể chọn 4 điểm nào đó trong lưới để tạo thành các hình vuông như hình bên dưới (n=5 và m=8).




Trong số các hình vuông này, diện tích của một số hình vuông là lẻ (diện tích của B là 17), hình vuông còn lại có diện tích là chẵn (diện tích của A là 4). Hãy đếm xem tổng cộng có bao nhiêu hình vuông có diện tích lẻ.

**Input: Cho bởi file văn bản HVUONG.INP**

- Là hai số nguyên n và m cách nhau một khoảng trắng ($1 \le n, m \le 10^5$)

**Output: Ghi vào tập văn bản HVUONG.OUT**

- Là số lượng số hình vuông có diện tích lẻ. Kết quả đảm bảo số lượng này là số nguyên 64 bít.

**Ví dụ:**

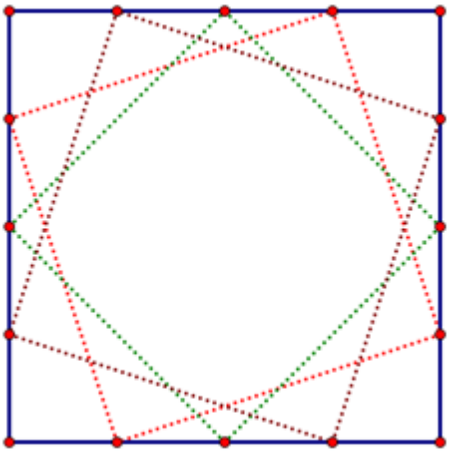
| HVUONG.INP | HVUONG.OUT |
|---|---|
| 1 2 | 2 |
| 3 3 | 12 |

***Hướng dẫn thuật toán:***

- Hình vuông được xét phải có cạnh là một số lẻ.

- Một hình vuông có độ dài với L điểm ở trên một cạnh và các cạnh song song với trục tọa độ, gọi độ dài đó là $i$ có thể được di chuyển theo $(n-i+1) \times (m-i+1)$ vị trí khác nhau trên lưới. $(i = L-1)$

- Chúng ta xét trường hợp số hình vuông có các cạnh song song với các đường chéo trên lưới vuông có L điểm. Dễ dàng thấy được nếu lưới vuông có $L$ điểm mỗi cạnh thì sẽ có $L-2$ hình vuông (Hay nói cách khác có i -1 hình vuông) có cạnh song song với các đường chéo. Minh họa bằng hình vẽ sau:



Có 3 hình vuông có các cạnh song song với các đường chéo trên lưới vuông có 5 điểm với độ dài 4.

Từ đó ta có công thức tổng quát cho trường hợp N×M như sau:

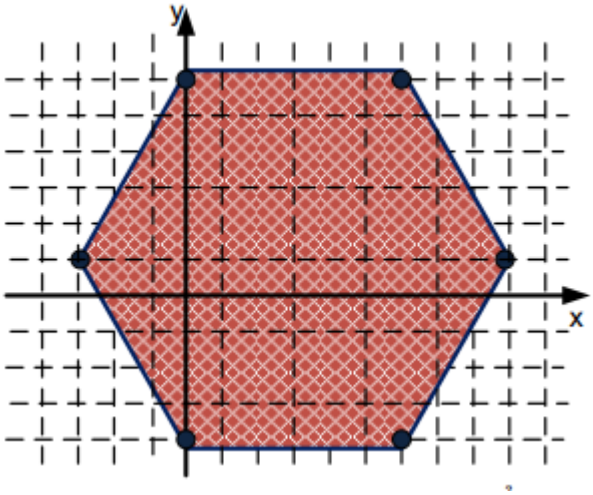
$$= \sum_{i=1}^{min(n,m)} ((n-i+1)\times(m-i+1)+(i-1)\times(n-i+1)\times(m-i+1))$$

$$= \sum_{i=1}^{min(n,m)} (i\times(n-i+1)\times(m-i+1)) \text{ Với i lẻ}$$

Độ phức tạp thuật toán: O(min(n,m))

**Bài 2. Lục giác đều (OLP 2008)**

Lục giác đều là một dạng cấu trúc đặc biệt trong thiên nhiên. Bạn có thể gặp lục giác đều khi quan sát cách bố trí cánh của nhiều loại hoa, khi quan sát cấu trúc của tổ ong, khi nghiên cứu sơ đồ liên kết giữa Các bon và Ôxy trong các hợp chất hữu cơ và vô cơ. Mũ đinh ốc cũng tạo thành một lục giác đều. Lục giác đều là một trong số hiếm hoi các loại đa giác đều có thể phủ kín mặt phẳng.




Một bạn sinh viên quyết định chọn "Vai trò và vị trí của lục giác đều trong thiên nhiên" làm đề tài báo cáo trong một buổi sinh hoạt ngoại khóa. Để chuẩn bị số liệu cho bản thuyết trình của mình bạn đó đã khảo sát rất nhiều dữ liệu về cấu trúc lục giác gặp trong thiên nhiên và cuộc sống. Mỗi dữ liệu khảo sát là một dãy tọa độ 6 đỉnh trong mặt phẳng của lục giác. Bạn sinh viên muốn biết 6 điểm này có thể là đỉnh của một lục giác đều hay không. Ví dụ, nếu tọa độ của 6 điểm nhận được là (-3,1), (6,6.19615), (0,6.19615), (9,1), (0, -4.19615), (6, -4.19615) thì câu trả lời là có. Với dữ liệu phong phú thu thập được, việc kiểm tra trở thành một công việc nặng nề và tẻ nhạt nếu không sử dụng máy tính.

**Yêu cầu:** Cho tọa độ 06 đỉnh $(x_i, y_i)$. Hãy kiểm tra xem 06 đỉnh trên có tạo thành một hình lục giác đều hay không.

**Input: Cho bởi file văn bản LUCGIAC.INP**

- Là 06 cặp số thực $(x_i, y_i)$, mỗi số cách nhau một khoảng trắng ($-10^3 \le x_i, y_i \le 10^3$).

**Output: Ghi vào file văn bản LUCGIAC.OUT**

- Nếu 06 đỉnh trên tạo thành hình lục giác đều, in ra YES.

- Nếu không, in ra NO.

*Lưu ý*: Các giá trị thực được so sánh với độ chính xác $10^{-4}$.

**Ví dụ:**

| **LUCGIAC.INP** | **LUCGIAC.OUT** |
|---|---|
| -3 1 6 6.19615 0 6.19615 9 1 0 -4.19615 6 -4.19615 | YES |
| 0 6 0 -4 6 6 6 -4 -1 1 9 1 | NO |

***Hướng dẫn thuật toán***

- Với mỗi đỉnh p[i], tính khoảng cách từ nó đến các đỉnh còn lại lưu vào mảng kc. *Chú ý: khoảng cách (p[i], p[j]) = khoảng cách (p[j], p[i]) được tính là một.*
- Sắp xếp mảng kc theo chiều tăng dần.
- Điều kiện để 6 đỉnh tạo thành lục giác đều là: *(kc[1]=kc[6]) và (kc[7]=kc[12) và (kc[13]=kc[15]).*
- Chú ý : Khi so sánh hai số thực x, y có bằng nhau hay không ta sử dụng `abs(x-y)<=Eps`, với Eps chọn một số thực nhỏ nào đó.

Độ phức tạp thuật toán: $O(n^2)$, với n = 15.