MỤC LỤC

[1. Mở đầu 3](#_Toc16003834)

[2. Một số đối tượng cơ bản của hình học phẳng 3](#_Toc16003835)

[2.1. Hệ trục tọa độ Đề Các Oxy 3](#_Toc16003836)

[2.2. Điểm trên hệ trục tọa độ *Oxy*: 3](#_Toc16003837)

[2.3. Đường thẳng 4](#_Toc16003838)

[2.4. Tích chéo, tích vô hướng của 2 vector 4](#_Toc16003839)

[2.5. Góc 5](#_Toc16003840)

[2.6. Tam giác 5](#_Toc16003841)

[2.7. Đa giác 5](#_Toc16003842)

[2.8. Đường tròn 6](#_Toc16003843)

[3. Một số bài toán cơ bản của hình học phẳng 6](#_Toc16003844)

[3.1. Biểu diễn tuyến tính 6](#_Toc16003845)

[3.2. Giao điểm của hai đường thẳng 6](#_Toc16003846)

[3.3. Tìm giao điểm của hai đoạn thẳng 7](#_Toc16003847)

[3.4. Tìm hai điểm gần nhau nhất 8](#_Toc16003848)

[4. Bao lồi 8](#_Toc16003849)

[4.1. Thuật toán tìm bao lồi 8](#_Toc16003850)

[4.2. Kiểm tra điểm thuộc đa giác 9](#_Toc16003851)

[5. Một số bài tập 10](#_Toc16003852)

[Bài 1. METERAIN - Mưa thiên thạch 10](#_Toc16003853)

[Bài 2. MILITARY - Câu chuyện người lính 12](#_Toc16003854)

[Bài 3. NEAREST - Cặp điểm gần nhất 15](#_Toc16003855)

[Bài 4. QBPOINT - Bộ ba điểm thẳng hàng 17](#_Toc16003856)

[Bài 5. PRAVO - Tam giác vuông 19](#_Toc16003857)

[Bài 6. NKPOLI - Đa giác 22](#_Toc16003858)

[Bài 7. NKLAND - Mảnh đất tổ tiên 24](#_Toc16003859)

[Bài 8. LEM – Dòng sông 27](#_Toc16003860)

[Bài 9. GPMB - Giải phóng mặt bằng 29](#_Toc16003861)

[Bài 10 . BMB - Bắn máy bay 31](#_Toc16003862)

[6. Một số bài tự luyện khác: 33](#_Toc16003863)

[7. Kết luận 33](#_Toc16003864)

**CHUYÊN ĐỀ DUYÊN HẢI**

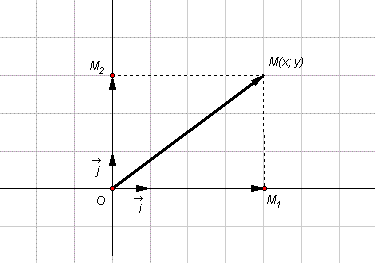
**HÌNH HỌC PHẲNG TRONG TIN HỌC**

# 1. Mở đầu

Bài toán trong tin học thường là rất đa dạng và có nhiều cách để giải bài toán đó. Việc áp dụng cách giải nào còn đòi hỏi sự sáng tạo và khả năng vận dụng của từng người. Để giải quyết những bài toán trong tin học ta có thể sử dụng các chiến lược như: Phương pháp quay lui, phương pháp nhánh và cận, phương pháp tham lam, phương pháp chia để chị, phương pháp quy hoạch động …, sử dụng cấu trúc dữ liệu đặc biệt như ngăn xếp, hàng đợi, cây … Ngoài ra còn có một số bài toán sử dụng nhiều kiến thức của toán ( số học, tổ hợp, hình học) ... Trong quá trình dạy tôi thấy rằng các em học sinh làm khá tốt những dạng bài trên tuy nhiên khi gặp một bài toán có liên quan đến kiến thức hình học các em thường gặp khó khăn, hoặc làm được nhưng kết quả sai so với đáp án. Lý do có thể là các em khó cài đặt, không tìm được thuật giải thích hợp, đôi lúc có thuật giải rồi nhưng kết quả vẫn sai do sai số trong quá trình tính toán. Để có thể giúp học sinh dễ dàng tiếp thu, giải quyết những dạng bài toán đó tôi đã tìm hiểu và viết chuyên đề **“Hình học phẳng trong Tin học”.**

Trong nội dung chuyên đề này tôi sẽ nhắc lại một số kiến thức hình học và xây dựng hệ thông các bài tập có hướng dẫn thuật toán chi tiết, đi kèm với lời giải để các em ôn tập và phần bài tập để các em tự luyện.

# 2. Một số đối tượng cơ bản của hình học phẳng

2.1. Hệ trục tọa độ Đề Các Oxy: Xem SGK Hình học 10, tiết 4.

## 2.2. Điểm trên hệ trục tọa độ *Oxy*:

Cho điểm *M(x,y)* như hình vẽ, *x* được gọi là hoành độ, *y* được gọi là tung độ của điểm M trong hệ trục tọa độ Đề Các *Oxy*, khi đó:



- Giả sử *x,y* là số nguyên, khi đó biểu diễn điểm M trong máy tính có thể dùng một trong hai cách sau:

|  |  |
| --- | --- |
| typedef pair<int,int> Point; | struct Point {  int x;  int y;  }; |

- Khoảng cách giữa 2 điểm  và , hoặc độ dài của vecctor  được tính bằng: 

|  |
| --- |
| double dist(Point A,Point B) {  return sqrt((B.x-A.x)\*(B.x-A.x)+(B.y-A.y)\*(B.y-A.y));  } |

## 2.3. Đường thẳng

Đường thẳng trong mặt phẳng xác định khi biết được 2 điểm *A, B* phân biệt nằm trên đường thẳng đó. Khi đó đường thẳng được xác định là tập hợp các điểm *M(x,y)* sao cho: 



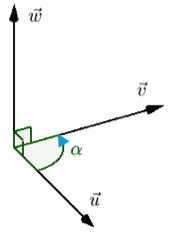
Nếu *t<0* thì *M* nằm ngoài *AB* về phía *A*

Nếu *0<t<1* thì M nằm giữa *A, B*

Nếu *t>1* thì *M* nằm ngoài *AB* về phía *B*.

## 2.4. Tích chéo, tích vô hướng của 2 vector

2.4.1. Tích chéo của hai vector:



Quay phải, tích chéo dương

Tích chéo của 2 vector  và (tích chéo là khái niệm được suy ra từ khái niệm tích có hướng trong không gian vector Ơclit nhiều chiều):

Giá trị của nó bằng định thức của ma trận  hoặc tính bằng . Trong đó góc là góc định hướng, có số đo từ  tới  🡺Giá trị lượng giác *sin* của góc định hướng = là:



Tích chéo có tác dụng để kiểm tra chiều quay từ  đến  là chiều quay phải, hay quay trái, ví dụ trong hình vẽ trên là quay trái khi đó ; chiều quay từ  đến  là quay phải nếu tích chéo có giá trị âm; và ,  thẳng hàng nếu tích chéo của chúng bằng 0.

|  |
| --- |
| int ccw(Point A, Point B, Point C) {  double t=(B.x-A.x)\*(C.y-A.y)-(B.y-A.y)\*(C.x-A.x);  if (t>0)  return 1; //quay trai  if (t<0)  return -1; //quay phai  return 0; //thang hang  } |

2.4.2. Tích vô hướng của 2 vector (hay còn gọi là tích chấm): Tích vô hướng của 2 vector là một số có giá trị là: 

Cài đặt tích vô hướng, tích có hướng bằng kĩ thuật chồng toán tử như sau:

|  |  |
| --- | --- |
| Tích vô hướng (tích chấm) | Tích chéo |
| int operator \*(Point u, Point v) {  return (u.x\*v.x+u.y\*v.y);  } | int operator ^(Point u, Point v) {  return (u.x\*v.y-u.y\*v.x);  } |

## 2.5. Góc

- Góc tạo bởi hai vector ,  có giá trị lượng giác .

- Góc tạo bởi tia *OA* và trục *Ox* có giá trị lượng giác **

|  |
| --- |
| double goc(Point A) {  double t=atan2(A.y,A.x);  if (t<0)  t=t+2\*acos(-1);  return t;  } |

2.6. Tam giác: Được xác định bởi 3 điểm A, B, C, có độ dài 3 cạnh thỏa mãn tất các điều kiện: *a+b>c; b+c>a; a+c>b.*

Diện tích tam giác được tính thông qua tích có hướng của vector như sau:



|  |
| --- |
| double sTriangle(Point A, Point B, Point C){  double s=(B.x-A.x)\*(C.y-A.y)-(B.y-A.y)\*(C.x-A.x);  return abs(s/2);  } |

Dựa vào diện tích tam giác, ta có thể tính khoảng cách từ điểm *C* đến đường thẳng *d* đi qua điểm *A, B* như sau:



|  |
| --- |
| double dist2(Point A, Point B, Point C){  return 2\*sTriangle(A,B,C)/dist(A,B);  } |

2.7. Đa giác: Đa giác là một đường gấp khúc khép kín. Trong lập trình, một đa giác được lưu bởi một dãy các đỉnh liên tiếp nhau A1, A2, … , AN. Khi đó **diện tích đại số** của một đa giác không tự cắt được định bởi công thức sau:



Nếu *S>0* thì ta đã liệt kê các đỉnh theo chiều ngược chiều kim đồng hồ.

Nếu *S<0* thì ta đã liệt kê các đỉnh theo chiều ngược chiều kim đồng hồ.

Còn *|S|* chính là diện tích của đa giác.

Công thức trên dễ dàng chứng minh bằng cách đi lần lượt theo các đỉnh biên của đa giác, tại mỗi đỉnh kẻ đường thẳng đứng xuống trục *Ox*, chia đa giác thành các hình thang vuông với hai đáy song song với trục tung *Oy* để tính diện tích.

2.8. Đường tròn: Tập hợp các điểm cách đều một điểm cho trước (gọi là tâm). Đường tròn biểu diễn thông qua tọa độ tâm và bán kính đường tròn. Đường tròn tâm *A* bán kính *r* kí hiệu toán học là: *(A,r).* Tương tự điểm thì có 2 cách biểu diễn với đường tròn như sau:

|  |  |
| --- | --- |
| struct circle {  Point A;  double r;  }; | typedef pair<pair<int,int>,double> circle; |

Một điểm nằm trong đường tròn khi khoảng cách của của điểm đó đến tâm đường tròn nhỏ hơn hoặc bằng bán kính. Ngược lại, khoảng cách tới tâm lớn hơn bán kính thì nó nằm ngoài đường tròn.

Hai đường tròn có điểm chung nếu khoảng cách giữa 2 tâm nhỏ hơn tổng hai bán kính và ngược lại.

Diện tích hình tròn: 

# 3. Một số bài toán cơ bản của hình học phẳng

## 3.1. Biểu diễn tuyến tính

Cho ba vectơ  hãy tìm hai số *p, q* để: 

Khi đó hai số *p, q* được tính bằng công thức sau:





+ Nếu *D* =  thì có duy nhất một cách biểu diễn tuyến tính vectơ qua vectơ 

+ Nếu *D = 0* thì hai vectơ song song với nhau, khi đó

*(p,q)* = *(nan; nan)* nếu  song song với 

*(p,q)* = *( inf, inf)* nếu  không song song với 

## 3.2. Giao điểm của hai đường thẳng

Trên mặt phẳng tọa độ Đề-các cho hai đường thẳng với phương trình tổng quát:





Hãy tìm giao điểm của hai đường thẳng đã cho

Đặt  ,  và  bài toán trở thành biểu diễn vectơ  qua tổ hợp tuyến tính của hai vectơ  và :  và biện luận cho giá trị giao điểm tìm được

## 3.3. Tìm giao điểm của hai đoạn thẳng

Cho bốn điểm *A, B, C, D* trên *Oxy*. Hãy cho biết hai đoạn thẳng AB và CD có giao nhau không, nếu có cho biết tọa độ giao điểm

Thuật toán 1:

+ Viết phương trình đường thẳng đi qua AB và CD

+ Tìm giao điểm M

+ Kiểm tra xem M có nằm trên AB hay không

Với  và  thì phương trình đường thẳng đi qua *A, B* là:







Tương tự như vậy ta tìm phương trình đường thẳng qua *CD* sau đó tìm giao điểm *M* bằng cách giải hệ hai phương trình bậc nhất hai ẩn sử dụng định thức, sau đó kiểm tra xem *M* có nằm giữa *A, B* hay không theo **(2.3).**

Nhận xét: Cách làm trên cần thực hiện nhiều phép tính nên không hiệu quả, dẫn đến tăng sai số.

Thuật toán 2:

Nếu M là giao điểm duy nhất của AB và CD thì  để: 



Như vậy, ta chỉ cần biểu diễn tuyến tính  qua  và , sau khi tìm được *p* và *q* (theo **3.1**) ta kiểm tra  và tìm tọa độ điểm *M* như sau:



Chương trình tìm giao điểm của 2 đoạn thẳng:

|  |
| --- |
| double p=(double)((C-A)^(C-D))/((B-A)^(C-D));  double q=(double)((C-A)^(B-A))/((C-D)^(B-A));  if (0<p & p<1 & 0<q & q<1) {  Point M=A+p\*(B-A); //Giao cua AB, CD là diem M  } |

Tương tự như trên ta có thể tìm giao điểm của 1 đường thẳng và 1 tia tương tự như tìm giao điểm của 2 đoạn thẳng như trên, chỉ khác điều kiện bây giờ là:



|  |
| --- |
| double p=(double)((C-A)^(C-D))/((B-A)^(C-D));  double q=(double)((C-A)^(B-A))/((C-D)^(B-A));  if (0<p & 0<q & q<1) {  Point M=A+p\*(B-A); //Giao cua tia AB voi doan CD là diem M  } |

## 3.4. Tìm hai điểm gần nhau nhất

Cho tập hợp Q gồm *n* điểm. Tìm cặp điểm trong Q có khoảng cách gần nhất.

Một thuật toán sơ đẳng nhất là ta đi tính tất cả các khoảng cách có thể được tạo ra từ 2 điểm trong Q, sau đó tìm khoảng cách lớn nhất trong đó. Độ phức tạp của thuật toán như vậy là O(*n2*).

|  |
| --- |
| double len=dist(a[1],a[2]);  for (int i=1; i<n; i++)  for (int j=i+1; j<=n; j++) {  if (dist(a[i],a[j])<len)  len=dist(a[i],a[j]);  } |

Có cách tiếp cận khác nhằm giảm độ phức tạp thuật toán còn O(*n*log*n*) sử dụng phương pháp chia để trị. Với bài toán kích thước *n* ta chia làm 2 bài toán con kích thước *n*/2 và kết hợp kết quả trong thời gian O(*n*).

# 4. Bao lồi

*Định nghĩa:* Cho tập hợp *n* điểm trên mặt phẳng. Bao lồi của tập này được định nghĩa là đa giác lồi có diện tích nhỏ nhất với các đỉnh thuộc tập đã cho và chứa tất cả *n* điểm ở bên trong hoặc biên của nó.

4.1. Thuật toán tìm bao lồi: Có nhiều thuật toán tìm bao lồi khác nhau như: thuật toán bọc gói, quét Graham,… Ở đây chúng ta xét một cách tìm bao lồi đơn giản hơn bằng Thuật toán **chuỗi đơn điệu** (**Monotone chain**) chỉ sử dụng các phép toán vector đơn giản.

Độ phức tạp chung của thuật toán O(*n*log*n*), trong đó sắp xếp mất O(*n*log*n*), xác định bao lồi mất O(*n*).



Giả sử *n* điểm đã cho là. Không mất tổng quát ta có thể xem các điểm này được xếp theo hoành độ tăng dần. Nếu hoành độ bằng nhau thì chúng được sắp xếp theo tung độ tăng dần. Khi đó ta chắc chắn *A1* (điểm cực trái) và *An* (điểm cực phải) thuộc bao lồi. Các điểm còn lại được chia thành hai phần:

- Các điểm ở nửa trên hay còn gọi là **chuỗi trên** (đường nét đứt trên hình), các điểm này thỏa mãn: 

- Các điểm ở nửa dưới hay còn gọi là **chuỗi dưới** (đường nét liền trên hình), các điểm này thỏa mãn:

|  |
| --- |
| //Point a[maxn]; đa giác ban đầu đã cho  //Point c[maxn]; các đỉnh bao lồi đánh số cùng chiều kim đồng hồ  sort(a+1,a+n+1);  // Tim bao loi  m=2;  c[1]=a[1];  c[2]=a[2];  for(int i=3; i<=n; i++) { ***//tìm chuỗi trên O(n)***  c[++m]=a[i];  while (m>2 && (c[m-1]-c[m-2])^(c[m]-c[m-1])>=0) {  m--;  c[m]=c[m+1];  }  }  int m0=m;  c[++m]=a[n-1];  for(int i=n-2; i>=1; i--) { ***//tìm chuỗi dưới O(n)***  c[++m]=a[i];  while (m-m0>1 && (c[m-1]-c[m-2])^(c[m]-c[m-1])>=0) {  m--;  c[m]=c[m+1];  }  }  --m; |

## 4.2. Kiểm tra điểm thuộc đa giác

Vấn đề đặt ra: Cho đa giác được xác định bởi *A*[maxn] được liệt kê lần lượt theo một thứ tự nào đó, kiểm tra xem điểm  có thuộc đa giác đó hay không.

Thuật toán 1: Áp dụng được với mọi loại đa giác không tự cắt.

- Nếu điểm *B* nằm trên biên của đa giác, được tính là thuộc đa giác.

- Nếu điểm *B* nằm trong đa giác thì kẻ một tia bất kì xuất phát từ *B* sao cho không đi qua đỉnh hoặc chứa cạnh nào của đa giác. Tia này sẽ cắt đa giác số lẻ lần. Để tìm giao điểm của tia và đoạn thẳng áp dụng phần 3.3.

Độ phức tạp thuật toán O(n).

Thuật toán 2: Chỉ áp dụng với đa giác lồi

- Nếu điểm *B* nằm trên biên của đa giác, được tính là thuộc đa giác.

- Nếu B nằm trong đa giác thì tổng diện tích tạo thành từ tam giác với 1 đỉnh là B và một cạnh là cạnh của đa giác bằng tổng diện tích đa giác.

Độ phức tạp là O(n).

Thuật toán 3: Chỉ áp dụng với đa giác lồi

Có cách khác để kiểm tra điểm có thuộc bao lồi hay không với độ phức tạp *O(logn)* bằng cách chặt nhị phân.

Thuật toán 4: Áp dụng đối với mọi loại đa giác không tự cắt

Dựa theo tổng góc định hướng tạo bởi B với các cặp đỉnh liên tiếp của đa giác. Nếu trị tuyệt đối của tổng đó bằng  thì B nằm trong đa giác.

Độ phức tạp: O(n).

# 5. Một số bài tập

## Bài 1. METERAIN - Mưa thiên thạch

Phú ông nhận được thông tin về một trận mưa thiên thạch sắp ập xuống trái đất. Không những thế, Phú ông còn biết tọa độ của vị trí điểm rơi của mỗi một thiên thạch. Phú ông nhờ Cuội xác định xem có bao nhiêu thiên thạch có thể rơi xuống cánh đồng của ông ta. Cánh đồng của Phú ông có dạng một hình đa giác lồi được xác định bởi danh sách các đỉnh được liệt kê theo thứ tự ngược chiều kim đồng hồ.

***Yêu cầu***: Xác định xem trong tập cho trước các điểm rơi của thiên thạch, có bao nhiêu điểm nằm trong cánh đồng của Phú ông. Các điểm nằm trên biên của cánh đồng không được tính là điểm nằm trong cánh đồng.

**Input**

* Dòng đầu tiên là số nguyên n (3 <= n <= 5000) là số đỉnh của đa giác lồi mô tả cánh đồng của Phú ông.
* Mỗi dòng trong n dòng tiếp theo chứa cặp tọa độ của một đỉnh của đa giác lồi.
* Dòng tiếp theo là số nguyên m (2 <= m <= 5000) - số thiên thạch rơi xuống.
* Mỗi dòng trong số m dòng cuối cùng chứa 2 số là tọa độ điểm rơi của một thiên thạch.

Các tọa độ là các số nguyên có trị tuyệt đối không quá 106.

**Output**

Ghi ra m dòng, mỗi dòng tương ứng với 1 điểm rơi của thiên thạch. Ghi "YES" nếu điểm rơi của thiên thạch nằm trong cánh đồng và ghi "NO" nếu trái lại.

Ví dụ:

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  2 4  8 4  6 8  4 6  4  3 5  4 7  5 5  6 7 | NO  NO  YES  YES |

**\* Hướng dẫn:**

Ta sử dụng tìm kiếm nhị phân cho mỗi truy vấn.

Gọi điểm đang xét là p. Ta chọn hai cạnh nối điểm 1 với điểm n và điểm 2 là 2 cạnh chốt. Sau đó, gọi l và r là 2 điểm mà cạnh nối từ 1 đến l và r thỏa mãn điểm p đang xét  nằm giữa 2 cạnh này (Ban đầu l = 2, r = n) . Ta sẽ kiểm tra cạnh nối điểm 1 với điểm  (mid) nằm cùng phía với cạnh nào. Nếu cùng phía với l thì l = mid, và ngược lại. Làm như vậy đến khi r – l = 1. Khi đó ta xét xem S1lr có bằng S1lp + S1rp +Splr  hay ko (S1lr là diện tích tạo bởi 3 điểm 1, điểm l và điểm r) . Đây chính là câu trả lời của bài toán.

**\* Chương trình mẫu:**

#include<bits/stdc++.h>

#define maxn 5005

#define maxC 1000000000

#define MOD (1e9 + 7)

#define mp make\_pair

#define PB push\_back

#define F first

#define S second

#define pii pair<int, int>

#define Task "metarain"

#define ll long long

using namespace std;

int n, Query;

pii a[maxn];

void setup() {

cin >> n;

for(int i=1 ; i<=n ; ++i) cin >> a[i].F >> a[i].S;

}

pii Vect(pii a, pii b) {

return mp(b.F - a.F, b.S - a.S);

}

ll CCW(pii x, pii y, pii mete) {

pii a = Vect(x, y);

pii b = Vect(y, mete);

return (1ll \* a.F \* b.S - 1ll \* a.S \* b.F);

}

ll dientich(pii a, pii b, pii c) {

return abs((b.F - a.F) \* (b.S + a.S) + (c.F - b.F) \* (c.S + b.S) + (a.F - c.F) \* (a.S + c.S));

}

bool calc(pii mete) {

int r = n, l = 2;

while(r - l > 1) {

int mid = (r + l) >> 1;

if(CCW(a[1], a[mid], mete) > 0) l = mid;

else r = mid;

}

if(l == 2 && CCW(a[1], a[2], mete) <= 0) return 0;

if(r == n && CCW(a[1], a[n], mete) >= 0) return 0;

ll s1 = dientich(a[1], mete, a[r]);

ll s2 = dientich(a[1], a[l], mete);

ll s3 = dientich(mete, a[l], a[r]);

ll S = dientich(a[1], a[l], a[r]);

if(!s3) return 0;

return (s1 + s2 + s3 == S);

}

int main() {

ios\_base::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);

// freopen(Task".inp", "r", stdin);

// freopen(Task".out", "w", stdout);

setup();

cin >> Query;

while(Query--) {

pii mete;

cin >> mete.F >> mete.S;

bool ok = calc(mete);

cout << (ok == true ? "YES" : "NO") << '\n';

}

return 0;

}

**\* Test:** <https://vn.spoj.com/problems/METERAIN/>

## Bài 2. MILITARY - Câu chuyện người lính

“Tôi vẫn nhớ chiến trường Điện Biên năm đó rất ác liệt, rất nhiều người lính đã ngã xuống. Tại vùng căn cứ này, địch cho xây dựng lô cốt, hàng rào dây thép gai rất nhiều , vòng trong nối vòng ngoài, tạo thành nhiều vòng bảo vệ … “Đó là dòng hồi tưởng của 1 người lính già đã từng tham gia chiến dịch Tây Bắc lịch sử. Lần theo những trang sử được ghi chép lại, người ta biết rằng tướng Đờ Cát lúc đầu chưa chọn vị trí để đặt sở chỉ huy mà tìm cách thiết lập các vòng bảo vệ bằng dây thép gai nối các cứ điểm lại với nhau, sau đó sẽ chọn đặt sở chỉ huy tại vị trí an toàn nhất là ở vị trí mà có nhiều vòng bảo vệ bao quanh nhất. Mỗi 1 vòng bảo vệ là 1 đa giác không tự cắt tạo thành bằng cách nối 1 số cứ điểm lại với nhau bằng dây thép gai, 1 cứ điểm thuộc về không quá 1 vòng bảo vệ, các vòng bảo vệ phải được thiết lập sao cho giữa 2 vòng bảo vệ bất kỳ X và Y thì phần diện tích chung của X và Y = Min( diện tích X, diện tích Y ) hoặc = 0. Trên mặt phẳng toạ độ, các cứ điểm được coi như các điểm có toạ độ nguyên. Bạn hãy xác định xem, sở chỉ huy của tướng Đờ Cát sẽ được bảo vệ tối đa bởi mấy vòng bảo vệ.

**Input**

Dòng 1: số nguyên N là số cứ điểm. ( 1 ≤ N ≤ 4000 ). N dòng tiếp theo, dòng thứ i gồm 2 số nguyên xi, yi tương ứng là toạ độ của cứ điểm i . Các toạ độ đều là số nguyên dương ≤ 10000 .

**Output**

Gồm 1 dòng duy nhất ghi ra số lượng vòng bảo vệ tối đa mà sở chỉ huy của tướng Đờ Cát có thể được bao bọc .

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  100 100  200 100  100 200  300 300 | 1 |

*Giải thích*: Ta nối cứ điểm 1, 2, 3, 4 lại tạo thành 1 vòng bảo vệ, đặt trụ sở chỉ huy bên trong thì ra được đáp án. Ngoài ra còn có các phương án khác là nối cứ điểm 1, 2, 3 tạo thành 1 vòng bảo vệ, nối cứ điểm 2, 3, 4 thành 1 vòng bảo vệ, … nhưng tất cả các phương án này thì khi chọn vị trí đặt trụ sở chỉ huy thì vẫn tối đa = 1.

**\* Hướng dẫn:**

Đầu tiên chúng ta tìm 1 bao lồi của tập hợp N điểm. Sau đó xóa nó đi khỏi mảng. Rồi lại tiếp tục tìm bao lồi cho những điểm còn lại cho đến khi chỉ còn 1 điểm hoặc các điểm còn lại tạo thành 1 đường thẳng(vì là 1 đường thẳng thì nó chẳng bao được cái gì cả

**\* Chương trình mẫu:**

#include <bits/stdc++.h>

using namespace std;

#define N 5010

struct point {

long long x, y;

}s[N];

int n;

bool cmp(const point &a, const point &b) {

return (a.y < b.y) || ((a.y == b.y) && a.x < b.x);

}

int ccw(const point &a, const point &b, const point &c) {

return a.x \* (b.y - c.y) + b.x \* (c.y - a.y) + c.x \* (a.y - b.y);

}

bool Straight() {

if (n<3) return true;

for (int i = 3; i<=n; i++) {

if (ccw(s[i - 2], s[i - 1], s[i]) != 0) return false;

}

return true;

}

int st[N];

bool fr[N];

void Convex\_hull() {

memset(fr, true, sizeof fr);

int m = 0;

//Up

for (int i = 1; i<=n; i++) {

while (m >= 2 && ccw(s[st[m - 1]], s[st[m]], s[i]) < 0) {

fr[st[m]] = true;

m--;

}

m++;st[m] = i;fr[i] = false;

}

// Down

int t = m + 1;

fr[1] = true;

for (int i = n; i>0; i--) {

if (!fr[i]) continue;

while (m >= t && ccw(s[st[m - 1]], s[st[m]], s[i])< 0 ) {

fr[st[m]] = true;

m--;

}

m++;st[m] = i;fr[i] = false;

}

int d = 0;

for (int i = 1; i<=n; i++) {

if (fr[i]) {

s[++d] = s[i];

}

}

n = d;

//cout<<n<<endl;

}

int main() {

// freopen("**MILITARY**.INP","w",stdin);

// freopen("**MILITARY**.OUT","w",stdout);

ios::sync\_with\_stdio(false);

cin.tie(0);cout.tie(0);

cin>>n;

for (int i = 1; i<=n; i++) {

cin>>s[i].x>>s[i].y;

}

sort(s + 1, s + n + 1, cmp);

int cnt = 0;

while (!Straight()){

Convex\_hull();

cnt++;

}

cout<<cnt;

}

**\*Test**: <https://vn.spoj.com/problems/MILITARY/>

## Bài 3. NEAREST - Cặp điểm gần nhất

Cho n (2 <= n <= 100,000) điểm trên mặt phẳng, hãy tìm cặp điểm có khoảng cách nhỏ nhất.

**Input**

* Dòng đầu tiên chứa số n.
* n dòng tiếp theo mỗi dòng chứa một cặp số thực (giá trị tuyệt đối không lớn hơn 107) biểu diễn tọa độ một điểm.

**Output**

Một số duy nhất (ghi chính xác đến 3 chữ số thập phân sau dấu phẩy) là khoảng cách nhỏ nhất tìm được.

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  1 1  2 2  3 3  4 4  5 5 | 1.414 |

**\* Hướng dẫn**: Sử dụng tư tưởng chia để trị

#include <algorithm>

#include <cassert>

#include <cstdio>

#include <cmath>

#include <set>

#include <bits/stdc++.h>

using namespace std;

const int N = (int) 1e5;

pair<double, double> p[N];

int n;

inline double sqr(double x) {

return x \* x;

}

inline double dist(const pair<double, double> &a, const pair<double, double> &b) {

return sqrt(sqr(a.first - b.first) + sqr(a.second - b.second));

}

int main() {

freopen("NEAREST.inp","r",stdin);

freopen("NEAREST.out","w",stdout);

assert(scanf("%d", &n) == 1);

for(int i = 0; i < n; ++i) {

assert(scanf("%lf %lf", &p[i].first, &p[i].second) == 2);

}

sort(p, p + n);

for(int i = 0; i + 1 < n; ++i) {

if (abs(p[i].first - p[i + 1].first) <= 1e-9 && abs(p[i].second - p[i + 1].second) <= 1e-9) {

puts("0.000");

return 0;

}

}

double best = 1e50;

set<pair<double, double> > s;

for(int i = 0, j = 0; i < n; ++i) {

set<pair<double, double> >::const\_iterator low = s.lower\_bound(make\_pair(p[i].second - best - 1e-3, -1e50));

set<pair<double, double> >::const\_iterator high = s.lower\_bound(make\_pair(p[i].second + best + 1e-3, 1e50));

for(set<pair<double, double> >::const\_iterator it = low; it != high; ++it)

best = min(best, dist(p[i], make\_pair(it->second, it->first)));

while(!s.empty() && p[i].first - p[j].first > best) {

s.erase(make\_pair(p[j].second, p[j].first));

++j;

}

s.insert(make\_pair(p[i].second, p[i].first));

}

printf("%.3f\n", best);

return 0;

}

**\*Test**: <https://vn.spoj.com/problems/NEAREST/>

## Bài 4. QBPOINT - Bộ ba điểm thẳng hàng

Trong các cuộc thi tin học, sự xuất hiện của những bài toán hình học làm đội tuyển CBQ khá lúng túng. Do đó thầy Thạch quyết định cho đội tuyển luyện tập các bài toán hình học. Bắt đầu từ điểm, thầy đưa ra bài toán:

Cho n điểm trong mặt phẳng Oxy, hãy đếm số bộ 3 điểm thằng hàng

Input

* Dòng thứ nhất ghi số N là số điểm trên mặt phẳng.
* N dòng tiếp theo, mỗi dòng ghi tọa độ của một điểm.

Output

* Một số duy nhất là số bộ 3 điểm thẳng hàng.

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6  0 0  0 1  0 2  1 1  2 0  2 2 | 3 |

*Giới hạn:*

1 ≤ N ≤ 2000. Tọa độ các điểm có trị tuyệt đối không quá 10000.

**\* Hướng dẫn:**

#include<cstdio>

#include<algorithm>

using namespace std;

#define N 2000

long long sqr(const int &x) { return (long long) x \* x; }

struct Point {

int x, y;

void read() {

scanf("%d%d", &x, &y);

}

} p[N];

struct Vector {

int x, y;

Vector() {}

Vector(const Point &A, const Point &B) {

int tx = B.x - A.x, ty = B.y - A.y;

if(ty < 0 || ty == 0 && tx < 0) { ty = -ty; tx = -tx; }

x = tx; y = ty;

}

inline bool operator < (const Vector &o) const {

return x \* o.y - y \* o.x > 0;

}

inline bool operator == (const Vector &o) const {

return x \* o.y == y \* o.x;

}

inline bool operator != (const Vector &o) const {

return x \* o.y != y \* o.x;

}

} u[N];

int main() {

//freopen("input.txt", "r", stdin);

int n; scanf("%d", &n);

for(int i = 0; i < n; p[i++].read());

int res = 0;

for(int i = 0; i < n; ++i) {

for(int j = i + 1; j < n; ++j) u[j] = Vector(p[i], p[j]);

sort(u+i+1, u+n); int begin = i+1;

for(int j = i + 1; j < n; ++j) if(u[j] != u[begin]) {

res += (j - begin) \* (j - begin - 1) / 2;

begin = j;

}

res += (n - begin) \* (n - begin - 1) / 2;

}

printf("%d\n", res);

return 0;

}

**\*Test**: <https://vn.spoj.com/problems/QBPOINT/>

## Bài 5. PRAVO - Tam giác vuông

Cho n điểm trên mặt phẳng. Hỏi có bao nhiêu tam giác vuông được tạo thành.

**Input**

Dòng đầu tiên chứa số nguyên dương n (3<=n<=1500), số điểm trên mặt phẳng

Dòng thứ i trong n dòng tiếp theo, mỗi dòng chứa 2 số nguyên xi, yi, tọa độ của một điểm (-109<=xi, yi<= 109). Không có hai điểm nào có cùng tọa độ.

**Output**

Gồm một dòng duy nhất là số lượng tam giác vuông tìm được.

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  4 2  2 1  1 3 | 1 |
| 4  5 0  2 6  8 6  5 7 | 0 |
| 5  -1 1  -1 0  0 0  1 0  1 1 | 7 |

**\* Hướng dẫn:**

#include<cstdio>

#include<cmath>

#include<cstring>

#include<cstdlib>

#include<cassert>

#include<ctime>

#include<algorithm>

#include<iterator>

#include<iostream>

#include<cctype>

#include<string>

#include<vector>

#include<map>

#include<set>

#include<queue>

#include<list>

//#include<conio.h>

#define maxn 1502

#define oo 1111111111

#define base 100000000

#define TR(c, it) for(typeof((c).begin()) it=(c).begin(); it!=(c).end(); it++)

long double const PI= acos((long double)(-1));

long double const ep = 0.000000000001;

using namespace std;

typedef pair<int, int> II;

typedef vector<int> VI;

typedef vector<II> VII;

typedef vector<VI> VVI;

typedef vector<VII> VVII;

typedef long double ld;

void OPEN(){

freopen("A.in", "r", stdin);

freopen("A.out", "w", stdout);

}

struct diem{

long long x,y;

};

struct vec\_tor{

long long x,y;

long double goc;

vec\_tor(){};

vec\_tor(diem D1,diem D2){

x = D2.x-D1.x;

y = D2.y-D1.y;

if(y<0){

x = -x;

y = -y;

}

if(y==0) x>0?x:-x;

goc = atan2((ld)y,(ld)x);

}

bool operator <(vec\_tor T)const{

return (x\*T.y>y\*T.x);// && x \* T.x >= 0 && y \* T.y >= 0);

}

bool operator ==(vec\_tor T)const{

return ( x\*T.y == y\*T.x && x \* T.x >= 0 && y \* T.y >= 0);

}

int vuonggoc(vec\_tor T) const{

if (x\*T.x==-y\*T.y) return 0;

if(x \* T.x + y \* T.y > 0) return 1;

return -1;

}

};

diem A[1511];

vec\_tor V[3011];

long long KQ = 0;

int n;

int main(){

//OPEN();

scanf("%d",&n);

for(int i = 0;i<n;i++)

scanf("%lld %lld",&A[i].x,&A[i].y);

for(int i = 0;i<n;i++){

int t = 0;

for(int j = 0;j<n;j++){

if(i!=j)

V[t++] = vec\_tor(A[i],A[j]);

}

sort(V,V+t);

int u=0,run = 0;

for(int j = 0;j<t;j++){

if(j>0 && V[j]==V[j-1]){

KQ+=run;

continue;

}

while(V[j].vuonggoc(V[u]) == 1 && u<t) u++;

run = 0;

while(V[j].vuonggoc(V[u]) == 0 && u<t){

u++;

run++;

KQ++;

}

}

}

printf("%lld",KQ);

//getch();

}

**\*Test**: <https://vn.spoj.com/problems/PRAVO/>

## Bài 6. NKPOLI - Đa giác

Có N điểm trên mặt phẳng với tọa độ là các số tự nhiên. Một *đa giác lồi nhiều đỉnh nhất*là một đa giác lồi có các đỉnh là gốc tọa độ và một số đỉnh trong các điểm đã cho, và có số đỉnh là nhiều nhất. Điểm gốc, nghĩa là điểm có tọa độ (0, 0), phải là một trong các đỉnh của *đa giác lồi nhiều đỉnh nhất*.

Viết chương trình xác định số đỉnh của đa giác này.

Một đa giác là *lồi*nếu mọi đoạn thẳng có đầu mút nằm trong đa giác đều nằm hoàn toàn trong đa giác đó. Các cạnh liên tiếp của một đa giác không được phép song song với nhau.

**Input**

Dòng đầu tiên chứa số tự nhiên N, 2 ≤ N ≤ 100, số điểm được cho.

Mỗi dòng trong số N dòng tiếp theo chứa 2 số tự nhiên X, Y, 1 ≤ X ≤ 100, 1 ≤ Y ≤ 100 cách nhau bởi khoảng trắng, cho biết tọa độ của một điểm. Các điểm đều phân biệt nhau.

**Output**

In ra một số nguyên duy nhất là số đỉnh của *đa giác lồi nhiều đỉnh nhất*.

*Lưu ý*: kết quả luôn không nhỏ hơn 3.

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  4 2  2 2  2 3  3 2  3 1 | 4 |
| 8  10 8  3 9  2 8  2 3  9 2  9 10  10 3  8 10 | 8 |
| 10  9 6  1 7  2 2  3 9  8 7  3 2  9 4  3 1  9 7  6 9 | 7 |

**\* Hướng dẫn:**

#include <algorithm>

#include <cmath>

#include <cstring>

#include <iostream>

#define pnt pair<int,int>

using namespace std;

pnt a[101];

int n;

int f[105][105];

bool angle\_cmp(pnt x,pnt y)

{

double a1 = atan2(1.0 \* x.second,1.0 \* x.first);

double a2 = atan2(1.0 \* y.second,1.0 \* y.first);

if (a1 != a2) return a1 < a2;

return (x.first \* x.first + x.second \* x.second < y.first \* y.first + y.second \* y.second);

}

bool ccw(int p,int q,int r)

{

int x1 = a[q].first - a[p].first,x2 = a[r].first - a[q].first,y1 = a[q].second - a[p].second,y2 = a[r].second - a[q].second;

return (x1 \* y2 - x2 \* y1 > 0);

}

int main()

{

// freopen("poly.in","r",stdin);

// freopen("poly.ou","w",stdout);

scanf("%d", &n);

for (int i = 1; i <= n; i++) scanf("%d %d", &a[i].first, &a[i].second);

sort(a + 1,a + n + 1,angle\_cmp);

memset(f,0,sizeof(f));

for (int j = 1; j <= n; j++)

{

f[0][j] = 2;

for (int i = 1; i < j; i++)

for (int k = 0; k < i; k++) if (ccw(k,i,j)) f[i][j] = max(f[i][j],f[k][i] + 1);

}

int best = 2;

for (int i = 1; i <= n; i++)

for (int j = i + 1; j <= n; j++) if (ccw(i,j,0)) best = max(best,f[i][j]);

printf("%d\n", best);

}

\*Test: <https://vn.spoj.com/problems/NKPOLI/>

## Bài 7. NKLAND - Mảnh đất tổ tiên

Bờm sống trên mảnh đất tổ tiên để lại từ xa xưa. Tuy nhiên, trải qua bao đời, mảnh đất của Bờm ngày nay có thể đã bị thay đổi vị trí, thậm chí còn có thể không giao với mảnh đất của tổ tiên! Một ngày nọ, Bờm phát hiện tấm bản đồ mô tả hình dạng mảnh đất của tổ tiên. Bờm muốn xác định xem mảnh đất ngày nay và mảnh đất tổ tiên có còn giao nhau hay không!

Yêu cầu: Biết mảnh đất ngày nay của Bờm và mảnh đất của tổ tiên đều có hình dạng đa giác lồi. Hãy giúp Bờm xác định 2 mảnh đất có giao nhau (nghĩa là có phần diện tích chung) hay không.

**Input**

Dòng đầu tiên chứa số nguyên t, cho biết số lượng test (t ≤ 10). t nhóm dòng tiếp theo mô tả các test, mỗi test có dạng như sau:

Dòng đầu tiên chứa số nguyên m, số đỉnh của đa giác lồi miêu tả mảnh đất của Bờm.

Dòng thứ 2 chứa 2m số nguyên cho biết tọa độ các đỉnh của mảnh đất của Bờm. Các đỉnh được liệt kê theo chiều kim đồng hồ.

Dòng thứ 3 chứa số nguyên n, số đỉnh của đa giác lồi miêu tả mảnh đất của tổ tiên.

Dòng thứ 4 chứa 2n số nguyên cho biết tọa độ các đỉnh của mảnh đất của tổ tiên Bờm. Các đỉnh được liệt kê theo chiều kim đồng hồ.

**Output**

Gồm t dòng, mỗi dòng ghi ra “YES” / “NO” nếu 2 mảnh đất giao nhau / không giao nhau trong test tương ứng.

*Giới hạn:* 3 ≤ m, n ≤ 1000. Tọa độ các đỉnh có giá trị tuyệt đối không vượt quá 1000000000.

Có 50% số test mà trong đó các số m,n đều có giá trị không vượt quá 100.

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  3  -6 3 -11 11 -10 6  3  -4 0 -3 5 -7 3  3  -3 4 -3 11 -6 9  3  0 -2 1 0 -2 -1 | YES  NO |

**\* Hướng dẫn:**

#include<cstdio>

#include<cmath>

#define MAX 1111

const double PI=acos(-1.0);

const double INF=2e9;

struct point {

double x,y;

point(){}

point(const double &\_x,const double &\_y) {

x=\_x;y=\_y;

}

point rotate(const double &alpha) {

return (point(x\*cos(alpha)-y\*sin(alpha),x\*sin(alpha)+y\*cos(alpha)));

}

};

double min(const double &x,const double &y) {

if (x<y) return (x); else return (y);

}

double max(const double &x,const double &y) {

if (x>y) return (x); else return (y);

}

int m,n;

point a[MAX];

point b[MAX];

void init(void) {

int i;

scanf("%d",&m);

for (i=1;i<=m;i=i+1) {

scanf("%lf",&a[i].x);

scanf("%lf",&a[i].y);

}

scanf("%d",&n);

for (i=1;i<=n;i=i+1) {

scanf("%lf",&b[i].x);

scanf("%lf",&b[i].y);

}

}

void check(void) {

int i,j;

double alpha;

double mxa,mna,mxb,mnb;

for (i=0;i<100;i=i+1) {

alpha=i\*PI/100.0;

mxa=-INF;mna=INF;

mxb=-INF;mnb=INF;

for (j=1;j<=m;j=j+1) {

mxa=max(mxa,a[j].rotate(alpha).x);

mna=min(mna,a[j].rotate(alpha).x);

}

for (j=1;j<=n;j=j+1) {

mxb=max(mxb,b[j].rotate(alpha).x);

mnb=min(mnb,b[j].rotate(alpha).x);

}

if (mxa<mnb) {

printf("NO\n");

return;

}

if (mxb<mna) {

printf("NO\n");

return;

}

}

printf("YES\n");

}

int main(void) {

int t,c;

scanf("%d",&t);

for (c=1;c<=t;c=c+1) {

init();

check();

}

return 0;

}

**\*Test:** <https://vn.spoj.com/problems/NKLAND/>

## Bài 8. LEM – Dòng sông

Do nhà và trường của Zone nằm ở hai bên bờ của một con sông nên hàng ngày cậu phải đi đò qua sông để đi học. Zone cảm thấy rằng hành trình của ông lái đò là không tối ưu và Zone muốn tìm ra một đường đi tốt hơn.

Sau khi quan sát và đo đạc, Zone có thể biểu diễn được con sông bằng cách chỉ ra 2 bờ của nó. Mỗi bờ sông sẽ được biểu diễn bằng 1 đường gấp khúc và được xác định bằng tọa độ của các điểm trên đường gấp khúc đó như sau:

Đường gấp khúc biểu diễn bờ phía Đông của sông có N điểm, điểm thứ i có tọa độ (x i, y i).

Đường gấp khúc biểu diễn bờ phía Tây của sông có M điểm, điểm thứ j có tọa độ (u j, v j).

Biết rằng 2 bờ sông không có điểm chung và y i< y i+1với mọi 1 ≤ i < N và v j< v j+1với mọi 1 ≤ j < M.

Bạn hãy tìm 2 điểm A và B sao cho A nằm trên bờ Đông của con sông và B nằm trên bờ Tây của con sông và khoảng cách AB là nhỏ nhất.

Input

- Dòng đầu ghi 2 số nguyên dương N và M.

- Dòng thứ i trong N dòng tiếp theo ghi 2 số nguyên x i, y i.

- Dòng thứ j trong M dòng tiếp theo ghi 2 số nguyên u j, v j.

Output

- Gồm 1 dòng duy nhất ghi khoảng cách ngắn nhất tìm được

( Các số thực được ghi với độ chính xác 3 chữ số sau dấu phẩy )

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 4  4 0  3 2  5 4  0 0  2 2  1 3  2 5 | 1.000 |

*Giới hạn*: 2 ≤ M, N ≤ 1000, |x i|, |yi|, |uj|, |vj| ≤ 10000

**\* Chương trình mẫu:**

#include <bits/stdc++.h>

#define sqr(x) ((x)\*(x))

const int N = 1111;

using namespace std;

struct P {

int x; int y;

};

P a[N], b[N];

double height(P a, P b, P c) {

double S = abs(a.x\*(b.y-c.y)+b.x\*(c.y-a.y)+c.x\*(a.y-b.y));

double AC2 = sqr(a.x-c.x)+sqr(a.y-c.y);

double AB2 = sqr(a.x-b.x)+sqr(a.y-b.y);

double BC2 = sqr(b.x-c.x)+sqr(b.y-c.y);

if (AB2 > AC2 + BC2) return sqrt(AC2);

if (AC2 > AB2 + BC2) return sqrt(AB2);

return S / (sqrt(BC2));

}

int main()

{

//freopen("LEM.in", "r", stdin);

int n, m, i, j; double res = 1e9;

scanf("%d %d\n", &n, &m);

for(i=1; i<=n; i++) scanf("%d %d\n", &a[i].x, &a[i].y);

for(i=1; i<=m; i++) scanf("%d %d\n", &b[i].x, &b[i].y);

for(i=1; i<=n; i++)

for(j=1; j<m; j++)

res = min(res, height(a[i], b[j], b[j+1]));

for(j=1; j<=m; j++)

for(i=1; i<n; i++)

res = min(res, height(b[j], a[i], a[i+1]));

printf("%.3f", res);

return 0;

}

**\*Test:** <https://vn.spoj.com/problems/LEM/>

## Bài 9. GPMB - Giải phóng mặt bằng

Chính quyền thành phố KN đang tiến hành mở thêm một tuyến đường mới trong thành phố. Chính quyền có bản đồ tọa độ của N hộ dân trong khu vực tuyến đường có thể đi qua. Tuyến đường là một đường thẳng đi qua tọa độ các hộ dân. Các hộ dân được đánh số từ 1 đến N; hộ dân thứ i có diện tích sử dụng là s i(m 2). Nếu tuyến đường đi ngang qua hộ dân thứ i, chính quyền cần phải đền bù cho hộ dân này si2+5 (đồng) tiền giải phóng mặt bằng. Hỏi chính quyền cần phải đền bù nhiều nhất bao nhiêu tiền khi xây dựng tuyến đường?

**Input**

Dòng 1: một số nguyên N là số hộ dân (1 ≤ N ≤ 1500).

Dòng thứ i trong N dòng tiếp theo chứa 3 số nguyên x i, y i, s icho biết tọa độ và diện tích của hộ dân thứ i (-50 ≤ x i, y i≤ 50, 30 ≤ s i≤ 500).

**Output**

In ra một số duy nhất là số tiền nhiều nhất chính quyền phải đền bù khi xây dựng tuyến đường.

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  0 0 1  1 1 2  2 2 4  0 1 5  1 0 3 | 51 |

**\* Hướng dẫn:**

#include<cstdio>

struct point {

int x,y;

point(){}

point(const int &\_x,const int &\_y) {

x=\_x;y=\_y;

}

};

point a[1515];

int c[111][111];

int f[111][111];

int n;

void init(void) {

scanf("%d",&n);

int i,x,y,s;

for (x=0;x<=100;x=x+1)

for (y=0;y<=100;y=y+1) c[x][y]=0;

for (i=1;i<=n;i=i+1) {

scanf("%d",&x);

scanf("%d",&y);

scanf("%d",&s);

c[x+50][y+50]+=s\*s+5;

a[i]=point(x+50,y+50);

}

}

int gcd(const int &a,const int &b) {

if (a==0) return (b);

if (b==0) return (a);

if (a%b==0) return (b);

if (b%a==0) return (a);

if (a>b) return (gcd(a%b,b));

if (b>a) return (gcd(a,b%a));

}

void count(void) {

int i,j;

for (i=0;i<=100;i=i+1)

for (j=0;j<=100;j=j+1)

f[i][j]=gcd(i,j);

}

bool inside(const int &x,const int &y) {

if (x<0) return (false);

if (y<0) return (false);

if (x>100) return (false);

if (y>100) return (false);

return (true);

}

int max(const int &x,const int &y) {

if (x>y) return (x); else return (y);

}

void process(void) {

int res=0;

int i,j,k,dx,dy,t,s;

for (i=0;i<=100;i=i+1)

for (j=0;j<=100;j=j+1)

if (f[i][j]==1)

for (k=1;k<=n;k=k+1) {

s=c[a[k].x][a[k].y];

dx=i;

dy=j;

for (t=1;t<=500;t=t+1) {

if (!inside(a[k].x+t\*dx,a[k].y+t\*dy)) break;

s+=c[a[k].x+t\*dx][a[k].y+t\*dy];

}

for (t=1;t<=500;t=t+1) {

if (!inside(a[k].x-t\*dx,a[k].y-t\*dy)) break;

s+=c[a[k].x-t\*dx][a[k].y-t\*dy];

}

res=max(res,s);

s=c[a[k].x][a[k].y];

dx=-i;

dy=j;

for (t=1;t<=500;t=t+1) {

if (!inside(a[k].x+t\*dx,a[k].y+t\*dy)) break;

s+=c[a[k].x+t\*dx][a[k].y+t\*dy];

}

for (t=1;t<=500;t=t+1) {

if (!inside(a[k].x-t\*dx,a[k].y-t\*dy)) break;

s+=c[a[k].x-t\*dx][a[k].y-t\*dy];

}

res=max(res,s);

}

printf("%d",res);

}

int main(void) {

init();

count();

process();

}

**\*Test:** [**https://vn.spoj.com/problems/GPMB/**](https://vn.spoj.com/problems/GPMB/)

## Bài 10 . BMB - Bắn máy bay

Vương quốc C11 sắp chuẩn bị gặp hiểm họa cực lớn từ vương quốc XYZ. Theo thông tin mật báo của điệp viên "Không Không Thấy", rằng XYZ sẽ gửi một Đại đội máy bay B52 ra thả bom tại HN (thủ đô của C11), và quyết đem HN về thời kỳ đồ đá. Và thật may mắn, điệp viên của ta đã rất nhanh tay lấy được sơ đồ chiến thuật tấn công của địch.

Đại đội máy bay bao gồm tổng cộng N chiếc, được đánh số từ 1-->N. Mỗi chiếc sẽ có một vị trí xác định trong đội hình, và được cụ thể bằng một cặp số (x,y) cho biết tọa độ tương đối của nó. Theo nghiên cứu, cứ mỗi 3 chiếc trong đội hình tạo thành một hình tam giác sẽ tăng chỉ số chắc chắn của đội hình lên một đơn vị. Và độ an toàn của một chiếc máy bay trong đội hình là số lượng đội hình tam giác mà nó tham gia.

Vì thế ta cũng đã đề ra một chiến thuật, mỗi lần ta sẽ bắn rơi chiếc máy bay có độ an toàn nhỏ nhất trong đội hình của chúng.

**Input**

* Dòng đầu tiên là số nguyên dương N (N<=200).
* N dòng tiếp theo là các cặp số nguyên (x,y) cho biết tọa độ của các chiếc máy bay. (|x|,|y|<=1000000000)

**Output**

Cho biết *chỉ số chắc chắn*của đội hình lúc đầu và số thứ tự của chiếc máy bay **được***bắn rơi đầu tiên*(nếu có nhiều máy bay cùng độ an toàn thì chọn chiếc có số thứ tự nhỏ nhất).

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  1 2  0 0  1 0  2 0 | 3 2 |

**\* Chương trình mẫu:**

#include <iostream>

using namespace std;

int isTriangle(int x1,int y1,int x2,int y2,int x3,int y3)

{

return 1LL\*(x1-x2)\*(y2-y3)!=1LL\*(y2-y1)\*(x3-x2);

}

int main()

{

int n,x[222],y[222],d[222]={0},total=0;

cin >> n;

for (int i=0;i<n;i++) cin >> x[i] >> y[i];

for (int i=0;i<n;i++)

for (int j=i+1;j<n;j++)

for (int k=j+1;k<n;k++)

if (isTriangle(x[i],y[i],x[j],y[j],x[k],y[k]))

d[i]++, d[j]++, d[k]++, total++;

int mn=1<<30,ans;

for (int i=0;i<n;i++)

if (d[i]<mn) mn=d[i], ans=i+1;

cout << total << ' ' << ans << endl;

}

**\*Test:** [**https://vn.spoj.com/problems/C11TRCNT/**](https://vn.spoj.com/problems/C11TRCNT/)

# 6. Một số bài tự luyện khác:

<https://vn.spoj.com/problems/POLY4/>

<https://vn.spoj.com/problems/CARPET/>

<https://vn.spoj.com/problems/RAOVUON/>

<https://vn.spoj.com/problems/QBCAKE/>

<https://vn.spoj.com/problems/SCIRCLE/>

<https://vn.spoj.com/problems/QBPIZZA/>

<https://vn.spoj.com/problems/FIRE/>

<https://vn.spoj.com/problems/TRIPOD/>

<https://vn.spoj.com/problems/RECT1/>

# 7. Kết luận

Bài tập hình học có thể có nhiều cách giải sáng tạo nếu học sinh được trang bị kiến thức toán học tốt. Chuyên đề Hình học trong Tin học nên dạy cho học sinh khi các em đã được học về vector trong Toán. Với những trường có lớp chuyên Toán- Tin có thể triển khai sớm ngay từ lớp 10, vì lớp 10 học sinh đã được trang bị kiến thức toán vector.

Trong giải toán hình học, cần chú ý việc xử lý sai số trong tính toán. Tránh tối đa tính toán với số thực, cố gắng đưa về tính toán với số nguyên lâu nhất có thể trong quá trình tính toán. Mặc dù rất cố gắng trong viết chuyên đề nhưng năng lực còn hạn chế rất mong sự góp ý của các thầy cô./.

**-----------HẾT-----------**