

Bảo Lộc – Lâm Đồng 08/11/2020

MỤC LỤC

HÌNH HỌC TÍNH TOÁN	3
Hợp nhất các đoạn thẳng	3
Bài toán	3
Giải thuật	3
Chương trình	4
Diện tích đa giác cạnh không tự cắt	5
Bài toán	5
Giải thuật	5
Chương trình	6
Định lý Pick	7
Định lý	7
Chứng minh	7
Tập điểm phủ các đoạn thẳng	9
Bài toán 1	9
Giải thuật	9
Chương trình	10
Bài toán 2	11
Giải thuật	11
Chương trình	18
Bao lồi	20
Bài toán	20
Giải thuật Graham	20
Chương trình	22
Kiểm tra điểm trong	24
Bài toán	24
Giải thuật	25
Chương trình	27
Tìm cặp điểm gần nhất	29

Bài toán	29
Giải thuật	30
Chương trình	32
Đường thẳng quét	33
Bài toán	33
Giải thuật	34
Chương trình:	40
BÀI TẬP NỘI DUNG HÌNH HỌC	43
WB45. LÁT VỈA HÈ <i>Tên chương trình: SIDEWALK.CPP</i>	43
WB47. ỐP GẠCH MEN <i>Tên chương trình: TILES.CPP</i>	44
WA43. KHU CÔNG NGHIỆP <i>Tên chương trình: INDZONE.CPP</i>	47
WA48. PHỦ BÓNG <i>Tên chương trình: OVERLAP.CPP</i>	50
WA49. TAM GIÁC CÂN <i>Tên chương trình: ISOSCELE.CPP</i>	55
WB33. CỬA TRƯỢT <i>Tên chương trình: MOVEMENT.CPP</i>	57
VZ33. IN PHUN 3D <i>Tên chương trình: P3D.CPP</i>	59
VZ34. HÌNH VUÔNG KHÁC NHAU <i>Tên chương trình: DIFFERENT.CPP</i>	61
VZ45. CẤP ĐIỂM <i>Tên chương trình: POINTS.CPP</i>	63
WA07. TAM GIÁC <i>Tên chương trình: TRIANGLE.CPP</i>	69
WA18. KÉO CẮT GIẤY <i>Tên chương trình: SCISSOR.CPP</i>	72
VZ18. BÃI SÌNH <i>Tên chương trình: MUD.CPP</i>	76
VZ24. BẰNG NHAU <i>Tên chương trình: EQ.CPP</i>	86
VZ25. ĐƠN VI <i>Tên chương trình: UNIT.CPP</i>	92
VY35. PHONG BÌ <i>Tên chương trình: ENVELOPE.CPP</i>	94
VZ01. NGHỆ THUẬT ĐƯỜNG PHỐ <i>Tên chương trình: ART.CPP</i>	96
B12. BỨC TRẠNH <i>Tên chương trình: PICTURE.CPP</i>	97
VX14. ĐÁNH LUỐNG <i>Tên chương trình: DRILL.CPP</i>	100
VX15. QUE NHÚA <i>Tên chương trình: STICKS.CPP</i>	103
WB29. VẮC XIN <i>Tên chương trình: VACCINE.CPP</i>	109

HÌNH HỌC TÍNH TOÁN

Hợp nhất các đoạn thẳng

Bài toán

Cho n đoạn thẳng trên đường thẳng, đoạn thứ i có tọa độ các điểm đầu và cuối là a_i, b_i . Hãy tính độ dài của hợp các đoạn đã cho.

Dữ liệu: Vào từ file văn bản MERGE_SEG.INP:

Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^5$),

Dòng thứ i trong n dòng sau chứa 2 số nguyên a_i và b_i ($-10^9 \leq a_i \leq b_i \leq 10^9$).

Kết quả: Đưa ra file văn bản MERGE_SEG.OUT một số nguyên – độ dài tính được.

Ví dụ:

MERGE_SEG.INP
4
1 5
3 8
14 16
10 20

MERGE_SEG.OUT
17

Giải thuật giải bài toán đã nêu được Klee công bố năm 1977 với độ phức tạp O(nlogn) và là giải thuật nhanh nhất hiện nay.

Giải thuật

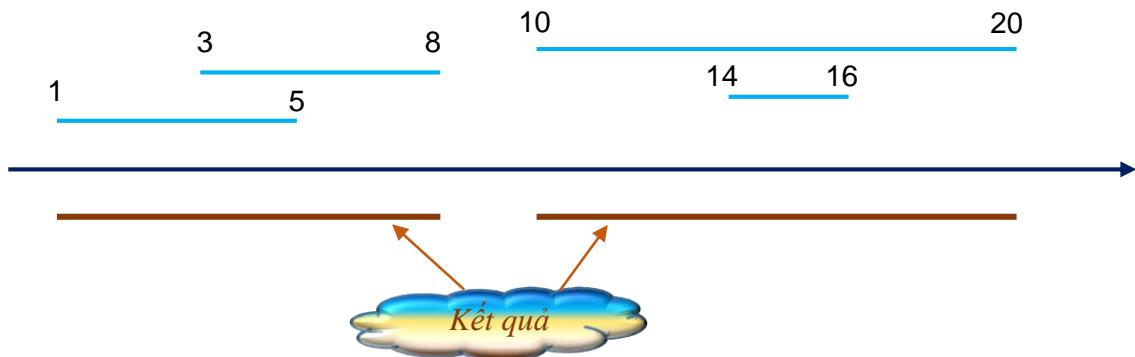
Lưu tất cả các tọa độ vào mảng \mathbf{x} ,

Với mỗi phần tử của \mathbf{x} : đánh dấu là điểm đầu hay điểm cuối của đoạn thẳng,

Sắp xếp \mathbf{x} theo thứ tự tăng dần, với các điểm cùng tọa độ – ưu tiên điểm đầu,

Sử dụng bộ đếm c thống kê số lượng đoạn thẳng giao nhau,

Duyệt mảng \mathbf{x} , nếu $c \neq 0$: cộng thêm vào kết quả hiệu $\mathbf{x}_i - \mathbf{x}_{i-1}$, nếu gặp điểm đầu – tăng c lên 1, gặp điểm cuối – giảm c .



Chương trình

```
#include <bits/stdc++.h>
#define NAME "merge_seg."
using namespace std;
typedef pair<int,bool> pib;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,a,b,c=1,ans=0;

int main()
{
    fi>>n;
    vector<pib> x(2*n);
    for(int i=0;i<n;++i)
    {
        fi>>a>>b;
        x[2*i] = {a,0};
        x[2*i+1] = {b,1};
    }
    sort(x.begin(),x.end());
    for(int i=1;i<2*n;++i)
    {
        if(c) ans+=x[i].first-x[i-1].first;
        if(x[i].second) --c; else ++c;
    }
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Diện tích đa giác cạnh không tự cắt

Bài toán

Cho đường gấp khúc khép kín n đỉnh không tự cắt, đỉnh thứ i có tọa độ thực (x_i, y_i) ($|x_i|, |y_i| \leq 10^6$, $i = 1 \div n$). Các đỉnh được liệt kê theo một trình tự nào đó.

Hãy tính diện tích của đa giác tạo bởi đường gấp khúc này.

Dữ liệu: vào từ file POLYG.INP:

- Dòng đầu tiên chữ số nguyên n ($2 < n \leq 10^5$),
- Dòng thứ i trong n dòng sau chứa 2 số thực x_i và y_i .

Kết quả: đưa ra file POLYG.OUT với độ chính xác 6 chữ số sau dấu chấm thập phân.

Ví dụ:

POLYGON.INP
6
2 1
0 3
3 6
6 5
8 1
4 3

POLYGON.OUT
20.000000

Giải thuật

Công thức tính diện tích đa giác:

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

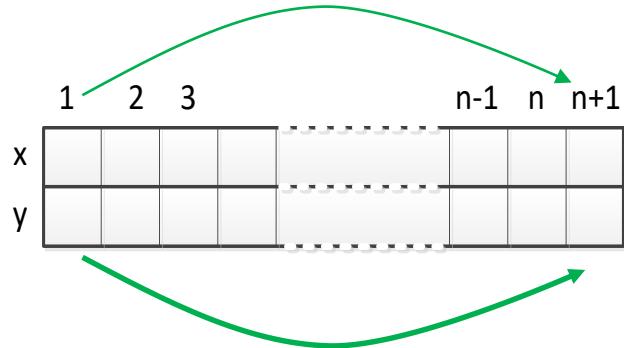
Trong đó $x_{n+1} \equiv x_1$, $y_{n+1} \equiv y_1$.

Lưu ý: biểu thức $\sum_{i=1}^n (x_i y_{i+1} - x_{i+1} y_i)$ cho giá trị dương khi các đỉnh được liệt kê theo chiều ngược kim đồng hồ và giá trị âm trong trường hợp ngược lại.

Dữ liệu vòng tròn:

Tạo $x_{n+1} = x_1$,

$y_{n+1} = y_1$.



Chương trình

```
#include <bits/stdc++.h>
#define NAME "polyg."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n;
double s=0, a, b;

int main ()
{
    fi>>n;
    vector<double>x(n+1), y(n+1);
    for(int i=0; i<n; ++i) fi>>x[i]>>y[i];
    x[n]=x[0]; y[n]=y[0];
    for(int i=0; i<n; ++i)
        s+=x[i]*y[i+1]-x[i+1]*y[i];
    s=abs(s)/2;
    fo<<fixed<<setprecision(6)<<s;
}
```



Định lý Pick

Xét đa giác diện tích khác 0, cạnh không tự cắt và trong mặt phẳng với hệ tọa độ Đè các tọa độ các đỉnh đa giác đều nguyên.

Định lý

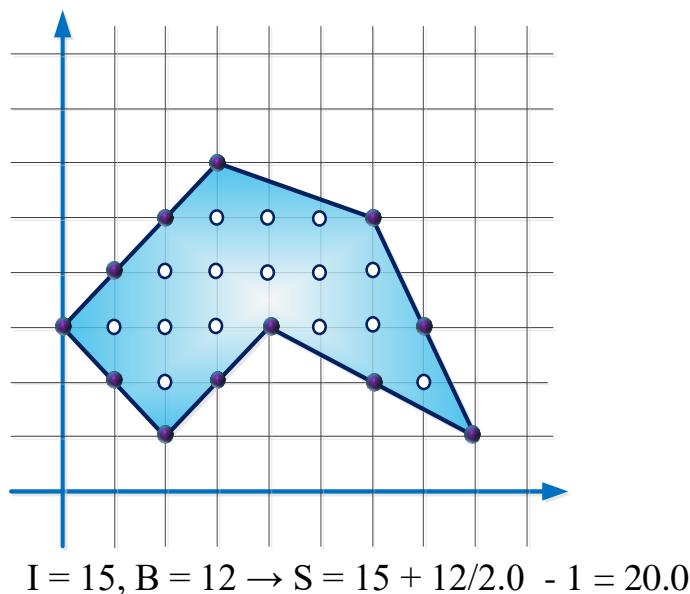
Gọi S là diện tích của đa giác, B – số điểm có tọa độ nguyên nằm trên cạnh của đa giác, I – số điểm trong có tọa độ nguyên. Khi đó tồn tại quan hệ

$$S = I + \frac{B}{2.0} - 1$$

Định này do nhà toán học Áo G. A. Pick phát biểu và chứng minh năm 1899.

Dựa vào định lý này, nếu biết B và I ta có thể tính diện tích đa giác mà không cần quan tâm đến tọa độ cụ thể của các đỉnh.

Ví dụ, với đa giác của hình dưới đây ta có :



Chứng minh

Định lý đúng với trường hợp hình vuông đơn vị,

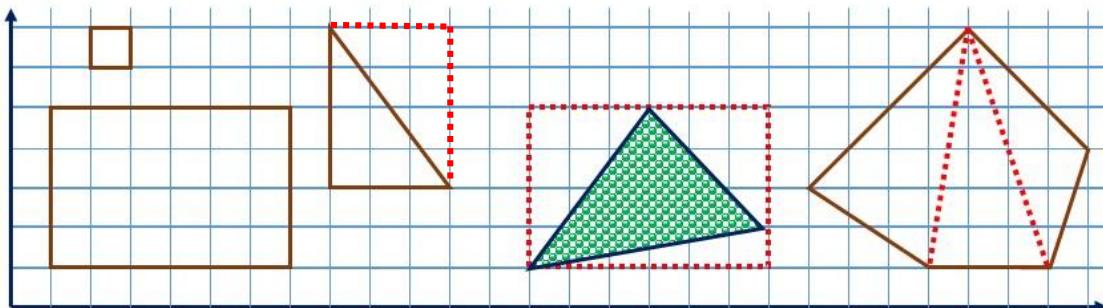
Nếu đa giác là hình chữ nhật cạnh độ dài a, b song song với trục tọa độ, số điểm trong là $I = (a-1) \times (b-1)$, số điểm trên cạnh là $B = 2 \times (a+b)$. Dễ dàng thấy được là diện tích $S = a \times b = I + B/2.0 - 1$.

Trường hợp đa giác là tam giác vuông có các cạnh góc vuông song song với trục tọa độ: ta giác là một nửa hình chữ nhật nếu ta cắt hình chữ nhật theo đường chéo. Gọi

c là số điểm có tọa độ nguyên trên đường chéo này. Để dàng chứng minh diện tích hình chữ nhật (và từ đó – diện tích tam giác vuông) không phụ thuộc vào c.

Trường hợp tam giác bất kỳ: bằng cách ghép thêm tối đa 3 tam giác vuông có các cạnh góc vuông song song với trực tọa độ ta đưa tam giác về hình chữ nhật cạnh song song với trực tọa độ. Lưu ý là diện tích các tam giác bổ sung không phụ thuộc vào số điểm nguyên trên cạnh huyền ta có điều cần chứng minh.

Với đa giác bất kỳ: chia thành các tam giác.



Lưu ý: Định lý Pick không áp dụng với trường hợp 3 hay nhiều chiều.



Tập điểm phủ các đoạn thẳng

Bài toán 1

Cho n đoạn thẳng trên đường thẳng, đoạn thứ i có tọa độ các điểm đầu và cuối là a_i, b_i . Hãy xác định số lượng ít nhất các điểm cần chọn để mỗi đoạn thẳng đã cho chứa ít nhất một điểm trong số đã chọn.

Dữ liệu: Vào từ file văn bản COVERING.INP:

- ⊕ Dòng đầu tiên chứa một số nguyên n ($1 < n \leq 10^5$),
- ⊕ Dòng thứ i trong n dòng sau chứa 2 số nguyên a_i và b_i ($-10^9 \leq a_i \leq b_i \leq 10^9$).

Kết quả: Đưa ra file văn bản COVERING.OUT một số nguyên – số điểm ít nhất cần chọn.

Ví dụ:

COVERING.INP
7
1 5
3 8
30 32
25 35
33 33
10 20
14 16

COVERING.OUT
4

Đây là mô hình toán học của một dạng bài toán lập lịch.

Giải thuật

Dữ liệu được lưu trữ vào vector \mathbf{x} , mỗi tọa độ được lưu dưới dạng nhóm 3 số

(tọa độ, dấu hiệu cuối đoạn, số thứ tự của đoạn)

a_i hoặc b_i

True nếu tọa độ là b_i

i

Dùng mảng $\text{flg}[]$ để đánh dấu, $\text{flg}[i] = \text{true}$ nếu đã có điểm đại diện và bằng false trong trường hợp ngược lại,

Sắp xếp \mathbf{x} theo thứ tự tăng dần,

Duyệt \mathbf{x} từ đầu đến cuối:

- Gặp điểm đầu: nạp số thứ tự vào vector \mathbf{v} ,

- Gặp điểm cuối: kiểm tra **flg** tương ứng, nếu đoạn tương ứng chưa có đại diện thì tăng số lượng đại diện lên 1, đánh dấu có đại diện cho tất cả các đoạn có số thứ tự lưu trữ trong **v** và xóa **v**.

Độ phức tạp của giải thuật: $O(n \log n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "covering."
using namespace std;
typedef tuple<int,bool,int> tibi;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,ans=0,m,a,b,k,pv=0;
bool c;

int main()
{
    fi>>n;
    vector<tibi> x(2*n);
    vector<bool> flg(n,0);
    for(int i=0;i<n;++i)
    {
        fi>>a>>b;
        x[2*i]= make_tuple(a,0,i);
        x[2*i+1] = make_tuple(b,1,i);
    }
    sort(x.begin(),x.end());
    vector<int> v(n);
    for(int i=0;i<2*n;++i)
    {
        tie(a,c,m)=x[i];
        if(c==0)v[pv++]=m;
        if(c && (flg[m]==0))
        {
            for(int j=0;j<pv;++j)flg[v[j]]=true;
            flg[m]=true; ++ans; pv=0;
        }
    }
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double) 1000<<" sec";
}
```

Bài toán 2

Cho n đoạn thẳng trên đường thẳng, đoạn thứ i có tọa độ các điểm đầu và cuối là a_i, b_i . Mỗi đoạn thẳng thuộc một trong 2 loại 0 hoặc 1. Hãy xác định số lượng ít nhất các điểm cần chọn để mỗi đoạn thẳng loại 1 đã cho chứa ít nhất một điểm trong số đã chọn và không có điểm chọn nào thuộc đoạn thẳng loại 0, chỉ ra một phương án tọa độ các điểm được chọn.

Dữ liệu: Vào từ file văn bản COVR_2.INP:

- ⊕ Dòng đầu tiên chứa một số nguyên n ($1 < n \leq 10^5$),
- ⊕ Dòng thứ i trong n dòng sau chứa 3 số nguyên a_i, b_i và t_i ($-10^9 \leq a_i \leq b_i \leq 10^9, 0 \leq t_i \leq 1$).

Kết quả: Đưa ra file văn bản COVR_2.OUT, dòng thứ nhất chứa một số nguyên – số điểm ít nhất cần chọn, dòng thứ hai – tọa độ các điểm được chọn.

Ví dụ:

COVR_2.INP
7
1 5 1
3 8 0
30 32 0
25 35 1
33 33 0
10 20 1
14 16 1

COVR_2.OUT
3
2 16 35

Giải thuật

Bài toán *có thể vô nghiệm* vì có các đoạn không được phép lấy đại diện,

Việc chọn điểm đại diện *không có định ở điểm cuối* của khoảng (vì có thể thuộc vùng cấm),

Cần phân biệt 2 loại đánh giá:

- ⊕ Độ phức tạp của *giải thuật*,
- ⊕ Độ phức tạp *lập trình*.

Độ phức tạp của giải thuật:

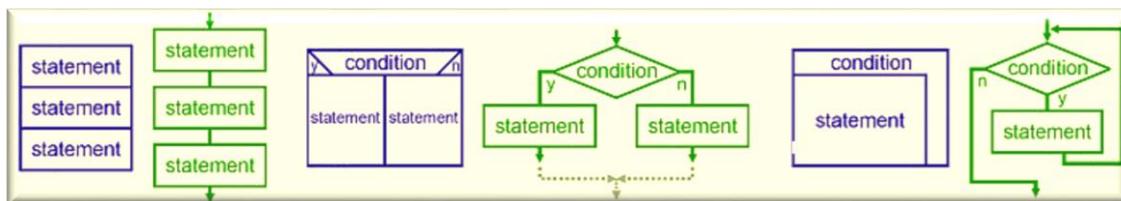
- Mỗi quan hệ giữa *thời gian thực hiện* chương trình với các tham số của bài toán,
- Sản phẩm lập trình được *nghiệm thu theo độ phức tạp của giải thuật*,
- Đây là *tiêu chuẩn định lượng*.

Độ phức tạp lập trình:

- Chương trình có cấu trúc tuyến tính là đơn giản nhất, dễ hiểu, dễ hiệu chỉnh và bảo trì (nâng cấp, cải tiến giải thuật, sửa đổi và mở rộng chương trình khi yêu cầu xử lý và cấu trúc dữ liệu vào thay đổi),
- Chương trình có độ phức tạp lập trình thấp khi nó có cấu trúc gần với tuyến tính nhất,
- Thời gian lập trình nhỏ nhất,
- Đây là **tiêu chuẩn định tính**.

Lý thuyết Lập trình cấu trúc (*Structured Programming*) do Niklaus Wirth, Dijkstra, Robert W. Floyd, Tony Hoare, Ole-Johan Dahl và David Gries đề xuất được truyền bá rộng rãi trên thế giới và những năm của thập kỷ 70 – 80 của thế kỷ XX với tham vọng biến lập trình từ nghệ thuật (*Art*) thành khoa học (*Discipline*).

Lập trình cấu trúc dựa trên 3 loại cấu trúc: tuyến tính, rẽ nhánh và chu trình, trong đó rẽ nhánh có thể là loại rẽ 2 nhánh (**if**) hoặc nhiều nhánh (**case/switch**), cấu trúc chu trình có thể là loại có số lần lặp biết trước (**for**) hoặc không biết trước (**while**, **repeat-until/do-while**).



Tuy vậy, dần dần người ta cũng nhận thấy lập trình vẫn mang trong mình **yếu tố nghệ thuật: nghệ thuật tổ chức dữ liệu** và **nghệ thuật thể hiện giải thuật**.

Ở bài toán đang xét có hai loại đoạn cho trước: loại bị cầm chọn điểm đại diện và loại cần chọn điểm đại diện. Hai loại đoạn này tác động lên việc tìm lời giải theo các nguyên tắc khác nhau: một điểm đã bị cầm thì bị cầm một lần hay k lần cũng như nhau. Nhưng điểm đại diện phải có mặt trong từng đoạn đã cho.

Chương trình sẽ đơn giản hơn nếu:

- ✓ Chọn đơn vị xử lý: điểm và các thuộc tính của nó,
- ✓ Phân loại và xử lý sơ bộ các đoạn cầm để đơn giản hóa yêu cầu cầm chọn,
- ✓ Lưu trữ dữ liệu xử lý theo cùng một quy cách.

Dữ liệu về các đoạn cầm chọn được lưu trữ dưới dạng nhóm 4 số:



Thuộc tính điểm:

- 0 – điểm đầu,
- 1 – điểm cuối đoạn cần chọn điểm đại diện,
- 2 – điểm cuối đoạn bị cấm.

Thuộc tính đoạn: 0 hoặc 1 như đã cho. Thuộc tính có thể lưu trữ kiểu bool.

Số thứ tự của đoạn:

Với đoạn cần chọn: lưu đúng số thứ tự của nó trong dữ liệu vào,

Với các đoạn cấm chọn: không cần phân biệt, vì vậy có chứa giá trị bất kỳ!

Với cách lưu trữ dữ liệu như vậy, khi sắp xếp lại, nếu có các điểm trùng tọa độ thì điểm đầu của đoạn cấm sẽ đứng trái nhất, sau đó là các điểm đầu của những đoạn cần chọn, tiếp sau – các điểm cuối của những đoạn cần chọn và cuối cùng – điểm cuối đoạn cấm.

Xử lý sơ bộ: tổng hợp thông tin về các đoạn cấm. Đây là vấn đề đơn giản, có nhiều cách xử lý tương đương với độ phức tạp giải thuật $O(n)$. Dưới đây là một trong số các cách đó, phục vụ việc **rèn luyện kỹ năng tổng quát** xử lý các vấn đề về đoạn thẳng.

Thông tin về các đoạn cấm được lưu trữ dưới dạng cặp dữ liệu (*tọa độ*, *thuộc tính*). Thuộc tính nhận giá trị 0 nếu là điểm đầu và 1 – cho điểm cuối.

Dữ liệu được sắp xếp theo thứ tự tăng dần. Như vậy, nếu trùng tọa độ thì các điểm đầu sẽ đứng trước các điểm cuối.

Dùng biến cnt tính số lượng đoạn cấm đè lên một điểm. Bắt đầu từ $cnt = 0$, khi gặp điểm đầu – tăng cnt, gặp điểm cuối – giảm cnt. Khi cnt trở lại 0 – ta có điểm cuối của đoạn hợp nhất của một số đoạn cấm.

Lưu ý: Cần hợp nhất 2 đoạn cấm đi liên tiếp nhau, tức là nếu có các đoạn cấm $[a, b]$ và $[b+1, c]$ thì kết quả hợp nhất sẽ là $[a, c]$.

Sau khi hợp nhất các đoạn cấm, thông tin tổng hợp sẽ được ghi nhận vào mảng chứa thông tin về các đoạn cần chọn đại diện theo quy cách đã nêu.

Tại đây, quá trình nhập, ghi nhận và chuẩn bị dữ liệu kết thúc.

Tìm các điểm đại diện: tìm điểm tự do cuối cùng phải nhất. Khi gặp điểm cuối đoạn cần chọn, ghi nhận điểm tự do cuối cùng tìm được làm điểm đại diện và đánh dấu tất cả các đoạn đã có đại diện. Nếu gặp điểm cuối một đoạn mà điểm tự do nhỏ hơn điểm đầu – bài toán vô nghiệm.

Tổ chức dữ liệu

- Mảng **flg[]** để đánh dấu, **flg[i] = true** nếu đã có điểm đại diện và bằng **false** trong trường hợp ngược lại,
- Mảng **y[]** kiểu **pair<int, bool>** lưu thông tin về các đoạn cấm,
- Mảng **x[]** kiểu **tuple<int, int, bool, int>** lưu thông tin các điểm,
- Biến **int free_p** lưu tọa độ điểm tự do cuối cùng,
- Mảng **res[]** kiểu **int** lưu tọa độ các điểm được chọn,
- Mảng **vt[]** lưu số hiệu đoạn của các đoạn chắc chắn đã có đại diện nhưng chưa được ghi nhận,
- Cấu trúc **map<int, int> mp** lưu điểm đầu các đoạn đã gấp và chưa có đại diện.

Xử lý

Đọc và phân loại dữ liệu:

```
fi>>n;
vector<tiibi> x;
vector<pib>y;
vector<bool> flg(n,0);
for(int i=0;i<n;++i)
{
    fi>>a>>b>>t;
    if(t==0){y.push_back({a,0}); y.push_back({b,1});}
    else {x.push_back(make_tuple(a,0,t,i));
          x.push_back(make_tuple(b,1,t,i));}
}
```

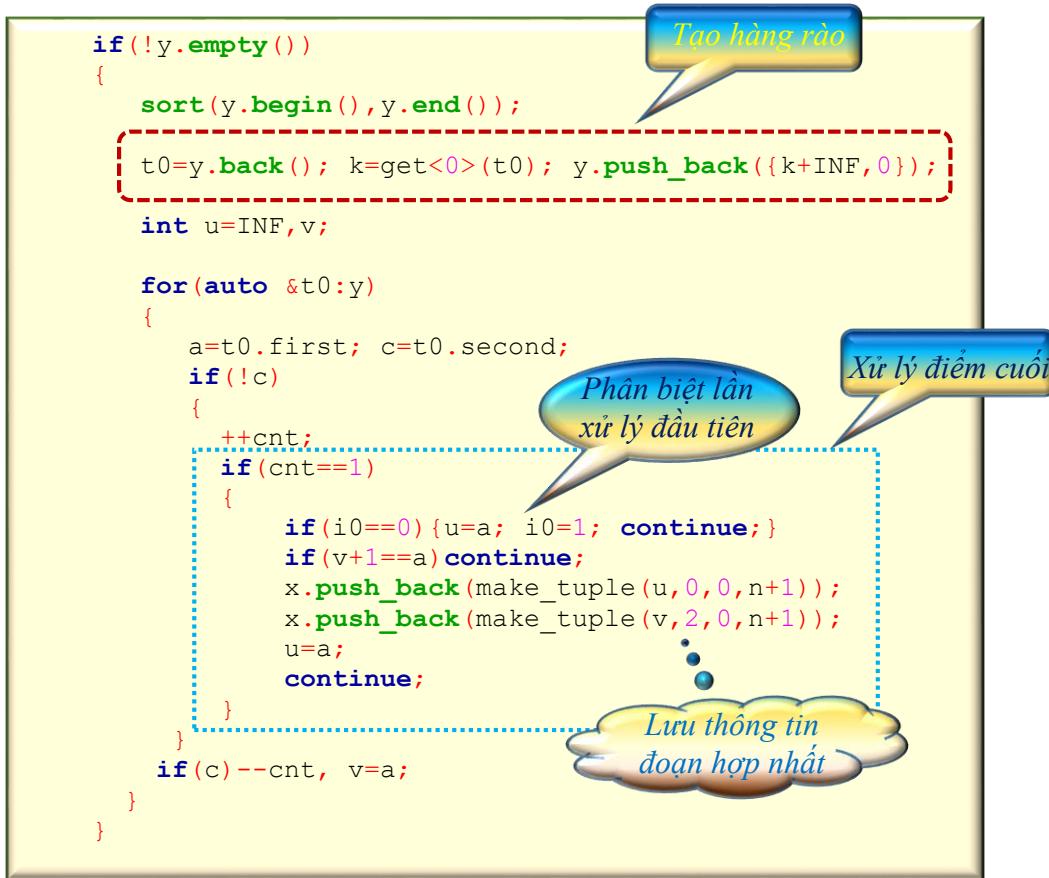
Hợp nhất các đoạn bị cấm chọn:

Chỉ xử lý khi trong dữ liệu có đoạn cầm,

Sắp xếp theo tọa độ tăng dần,

Thông tin về đoạn hợp nhát phụ thuộc vào điểm đầu của đoạn tiếp theo vì vậy có thể dùng kỹ thuật hàng rào để tiện xử lý: thêm một điểm đầu với tọa độ vô cực,

Thông tin về đoạn cầm được ghi nhận với số thứ tự đoạn là **n+1**.



Tìm điểm đại diện:

Xác định điểm tự do **free_p** phải nhất:

$$\text{free_p} = \begin{cases} \mathbf{a}, & \text{nếu } \mathbf{a} \text{ là điểm đầu của đoạn loại 1 và không bị phủ bởi đoạn loại 0,} \\ \mathbf{a}-1, & \text{nếu } \mathbf{a} \text{ là điểm đầu của đoạn loại} \\ \mathbf{a}+1, & \text{nếu } \mathbf{a} \text{ là điểm cuối của đoạn loại 0,} \end{cases}$$

Xử lý điểm đầu đoạn loại 1:

- ❖ \mathbf{a} – điểm đầu đoạn số thứ tự \mathbf{m} ,
- ❖ Trường hợp không bị phủ: nạp \mathbf{m} vào vector \mathbf{vt} , ghi nhận **free_p**,
- ❖ Trường hợp bị phủ: nạp \mathbf{a} vào \mathbf{mp} : $\mathbf{mp}[\mathbf{m}] = \mathbf{a}$;

Xử lý điểm cuối đoạn loại 1:

- ▲ \mathbf{a} – điểm cuối đoạn số thứ tự \mathbf{m} ,
- ▲ Trường hợp vô nghiệm: đoạn \mathbf{m} chưa được đánh dấu, điểm cuối bị phủ ($\mathbf{cnt} == 1$) và $\text{free_p} < \mathbf{mp}[\mathbf{m}]$,
- ▲ Nếu không vô nghiệm: chỉnh lý **free_p** khi cần thiết, nạp **free_p** vào mảng kết quả **res**, nạp \mathbf{m} vào \mathbf{vt} (nếu cần), đánh dấu đã có đại diện các đoạn lưu trong \mathbf{vt} và xóa rỗng \mathbf{vt} .

Đưa ra kết quả:

- ♣ Số lượng điểm: kích thước của **res**,

♣ Các điểm đại diện lưu trong **res**.

```
cnt=0;
for(auto & z:x)
{
    tie(a,c,t,m)=z;
    if(c==0)
    {
        if(t==0) {cnt=1;free_p=a-1; continue;}
        if(t==1 && cnt==0){free_p=a;vt.push_back(m); continue;}
        if(t==1 && cnt==1 && flg[m]==0){mp[m]=a; continue;}
    }
    else
    {
        if(c==2){cnt=0; free_p =a+1; continue;}
        if(cnt==1 && flg[m]==0)
            if(free_p<mp[m]) {fo<<'0';return 0;}
        if(cnt==0 && flg[m]==0){free_p=a;
            flg[m]=1;res.push_back(free_p);}
        if(flg[m]==0)res.push_back(free_p); vt.push_back(m);
        for(auto & iv:vt) flg[iv]=1; vt.clear();
    }
}
```

Độ phức tạp của giải thuật: $O(n \log n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "covr_2."
using namespace std;
typedef tuple<int,int,bool,int> tiibi;
typedef pair<int,bool> pib;
typedef pair<int, int> pii;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int INF = 2000000000;
int n,ans=0,cnt=0,a,b,c,m,k,free_p =INF,i0=0,i1=0;
bool t;
pib t0;
pii tq;

int main()
{
    fi>>n;
    vector<tiibi> x;
    vector<pib>y;
    vector<bool> flg(n,0);
    for(int i=0;i<n;++i)
    {
        fi>>a>>b>>t;
        if(t==0){y.push_back({a,0}); y.push_back({b,1});}
        else {x.push_back(make_tuple(a,0,t,i));
              x.push_back(make_tuple(b,1,t,i)); }
    }
    if(!y.empty())
    {
        sort(y.begin(),y.end());
        t0=y.back(); k=get<0>(t0); y.push_back({k+INF,0});
        int u=INF,v;

        for(auto &t0:y)
        {
            a=t0.first; c=t0.second;
            if(!c)
            {
                ++cnt;
                if(cnt==1)
                {
                    if(i0==0) {u=a; i0=1; continue;}
                    if(v+1==a)continue;
                    x.push_back(make_tuple(u,0,0,n+1));
                    x.push_back(make_tuple(v,2,0,n+1));
                    u=a;
                    continue;
                }
            }
            if(c)--cnt, v=a;
        }
        sort(x.begin(),x.end());
    }
}
```

```

vector<int> vt;
vector<int>res;
map<int,int> mp;
cnt=0;
for(auto & z:x)
{
    tie(a,c,t,m)=z;
    if(c==0)
    {
        if(t==0 ) {cnt=1; free_p=a-1; continue;}
        if(t==1 && cnt==0) {free_p=a; vt.push_back(m); continue;}
        if(t==1 && cnt==1 && flg[m]==0) {mp[m]=a; continue;}
    }
    else
    {
        if(c==2) {cnt=0; free_p =a+1; continue;}
        if(cnt==1 && flg[m]==0)
            if(free_p<mp[m]) {fo<<'0';return 0;}
        if(cnt==0 && flg[m]==0) {free_p=a;
                                    flg[m]=1;res.push_back(free_p);}
        if(flg[m]==0) res.push_back(free_p); vt.push_back(m);
        for(auto & iv:vt) flg[iv]=1; vt.clear();
    }
}
fo<<res.size()<<'\n';
for(int i:res) fo<<i<<' ';
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```



Bao lồi

Bài toán

Cho n điểm trên mặt phẳng, điểm thứ i có tọa độ $(\mathbf{x}_i, \mathbf{y}_i)$. Hãy xác định bao lồi (hay nói cách khác – xác định đa giác lồi nhỏ nhất) chứa các điểm đã cho.

Dữ liệu: Vào từ file văn bản CONVEX.INP:

- ✚ Dòng đầu tiên chứa một số nguyên n ($3 \leq n \leq 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số thực \mathbf{x}_i và \mathbf{y}_i ($0 \leq |\mathbf{x}_i|, |\mathbf{y}_i| \leq 10^9$).

Kết quả: Đưa ra file văn bản CONVEX.OUT, dòng đầu tiên chứa số nguyên m – số đỉnh của bao lồi, dòng thứ j trong m dòng sau chứa 2 số thực \mathbf{u}_j và \mathbf{v}_j xác định đỉnh của bao lồi với độ chính xác 6 chữ số sau dấu chấm thập phân. Các đỉnh liệt kê theo chiều kim đồng hồ.

Ví dụ:

CONVEX.INP	CONVEX.OUT
9	5
3 3	1.000000 1.000000
7 2	3.000000 5.000000
5 3	6.000000 5.000000
4 0	7.000000 2.000000
3 5	4.000000 0.000000
5 2	
1 1	
6 5	
5 1	

Giải thuật Graham

Giải thuật tìm bao lồi được Graham công bố năm 1972 và được Andrew cải tiến năm 1979. Bao lồi được tìm với độ phức tạp $O(n \log n)$ và chỉ cần dùng các phép cộng, trừ, nhân và so sánh.

Đây là giải thuật tìm bao lồi có độ phức tạp tốt nhất. Tuy nhiên giải thuật này chỉ thích hợp với việc xử lý dữ liệu offline.

Tọa độ các điểm được lưu trữ dưới dạng mảng cặp tọa độ. Mảng được sắp xếp theo trình tự tăng dần. Gọi P_0 và P_1 tương ứng là điểm đầu và cuối trong mảng đã sắp xếp. P_0 sẽ là điểm có tọa độ \mathbf{x} nhỏ nhất và có tọa độ \mathbf{y} nhỏ nhất trong số các điểm có cùng tọa độ \mathbf{x} nhỏ nhất. P_1 sẽ là điểm có tọa độ \mathbf{x} lớn nhất và có tọa độ \mathbf{y} lớn nhất trong số các điểm có cùng tọa độ \mathbf{x} lớn nhất. Rõ ràng bao lồi cần tìm phải chứa các điểm P_0 và P_1 .

Đường thẳng d nối 2 điểm P_0 và P_1 sẽ chia các điểm đã cho thành 2 tập S_{up} và S_{down} . Tập S_{up} chứa các điểm nằm bên trái đường thẳng d , tập S_{down} chứa các điểm nằm bên phải đường thẳng d . Các điểm P_0 và P_1 tham gia vào cả 2 tập. Các điểm nằm trên đường thẳng có thể cho vào tập bất kỳ, dù sao chúng cũng không phải là những điểm tạo nên bao lồi.

Xây dựng phần trên của bao lồi từ các điểm thuộc S_{up} và phần dưới của bao lồi từ các điểm thuộc S_{down} .

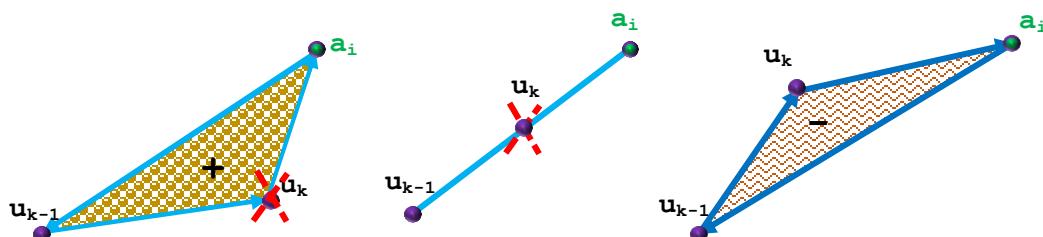
Xét việc xây dựng phần trên của bao lồi (phần dưới – xây dựng tương tự).

Từ tập up rỗng kết nạp P_0 và sau đó P_1 vào up .

Lần lượt duyệt các điểm còn lại của S_{up} theo trình tự đã sắp xếp chung.

Với mỗi điểm mới tính diện tích có dấu của tam giác có các đỉnh là điểm gần cuối cùng u_{pk-1} , điểm cuối cùng up_k trong up và điểm mới theo đúng trình tự đã liệt kê. Nếu diện tích là dương – điểm mới nằm bên trái đường nối u_{pk-1} và up_k , như vậy điểm up_k sẽ không tham gia vào bao lồi và sẽ bị loại. Nếu diện tích bằng 0 – ba điểm nằm trên đường thẳng nhưng điểm up_k nằm ở giữa và cũng sẽ bị loại.

Cuối cùng bổ sung điểm mới vào cuối up .



Sau khi xử lý tất cả các điểm trong S_{up} ta có phần trên của bao lồi. Xử lý tương tự - có phần dưới bao lồi. Kết quả cần tìm sẽ là sự liên kết 2 phần nhận được.

Lưu ý:

Ta không cần xây dựng tường minh các tập S_{up} và S_{down} ,

Gọi S là diện tích có dấu của tam giác ABC theo trình tự liệt kê $A \rightarrow B \rightarrow C$, có

$$2 \times S = \mathbf{x}_a \times (\mathbf{y}_b - \mathbf{y}_c) + \mathbf{x}_b \times (\mathbf{y}_c - \mathbf{y}_a) + \mathbf{x}_c \times (\mathbf{y}_a - \mathbf{y}_b)$$

Độ phức tạp của giải thuật: O(nlogn).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "convex."
using namespace std;
typedef pair<double, double> pdd;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n;

double area(pdd a, pdd b, pdd c)
{
    return (a.first*(b.second-c.second) +
            b.first*(c.second-a.second) +
            c.first*(a.second-b.second));
}

int main()
{
    fi>>n;
    vector<pdd> a(n);
    vector<pdd> up, down;
    for(int i=0;i<n;++i) fi>>a[i].first>>a[i].second;
    sort(a.begin(),a.end());
    pdd p1=a[0], p2=a[n-1];
    up.push_back(p1); down.push_back(p1);
    for(int i=1;i<a.size();++i)
    {
        if(i==a.size()-1 || area(p1,a[i],p2)<0)
        {
            while(up.size()>=2 &&
                  area(up[up.size()-2],up[up.size()-1],a[i])>=0) up.pop_back();
            up.push_back(a[i]);
        }
        if(i==a.size()-1 || area(p1,a[i],p2)>0)
        {
            while(down.size()>=2 &&
                  area(down[down.size()-2],down[down.size()-1],a[i])<0)
                down.pop_back();
            down.push_back(a[i]);
        }
    }
    a.clear();
    for(int i=0;i<up.size();++i) a.push_back(up[i]);
    for(int i=down.size()-2;i>0;--i) a.push_back(down[i]);

    fo<<a.size()<<'\n';
    for(int i=0;i<a.size();++i) fo<<fixed<<a[i].first<<' '<<a[i].second<<'\n';

    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```

}



Kiểm tra điểm trong

Bài toán

Cho đa giác lồi n đỉnh, đỉnh thứ i có tọa độ nguyên (a_i, b_i) , $i = 1 \div n$. Các đỉnh được liệt kê theo trình tự ngược kim đồng hồ.

Với mỗi điểm trong số m điểm cho trước, điểm thứ j có tọa độ nguyên (x_j, y_j) , $j = 1 \div m$, hãy xác định điểm nằm trong hay ngoài đa giác lồi đã cho và đưa ra thông báo tương ứng “**Inside**” hoặc “**Outside**”. Một điểm được gọi là nằm trong đa giác nếu nó nằm hoàn toàn trong miền trong của đa giác hay nằm trên cạnh hoặc đỉnh của đa giác.

Dữ liệu: Vào từ file văn bản INSIDE.INP:

- ✚ Dòng đầu tiên chứa một số nguyên n ($3 \leq n \leq 10^5$),
- ✚ Dòng thứ thứ i trong n dòng tiếp theo chứa 2 số nguyên a_i và b_i ,
- ✚ Dòng tiếp theo chứa số nguyên m ,
- ✚ Dòng thứ thứ j trong m dòng tiếp theo chứa 2 số nguyên x_j và y_j .

Các tọa độ đều nguyên và có giá trị tuyệt đối không vượt quá 10^9 .

Kết quả: Đưa ra file văn bản INSIDE.OUT các thông báo tìm được, mỗi thông báo đưa ra trên một dòng.

Ví dụ:

INSIDE.INP
5
6 5
3 5
1 1
4 0
7 2
7
4 2
2 3
1 3
4 5
2 2
11 8
5 2

INSIDE OUT
Inside
Inside
Outside
Inside
Inside
Outside
Inside

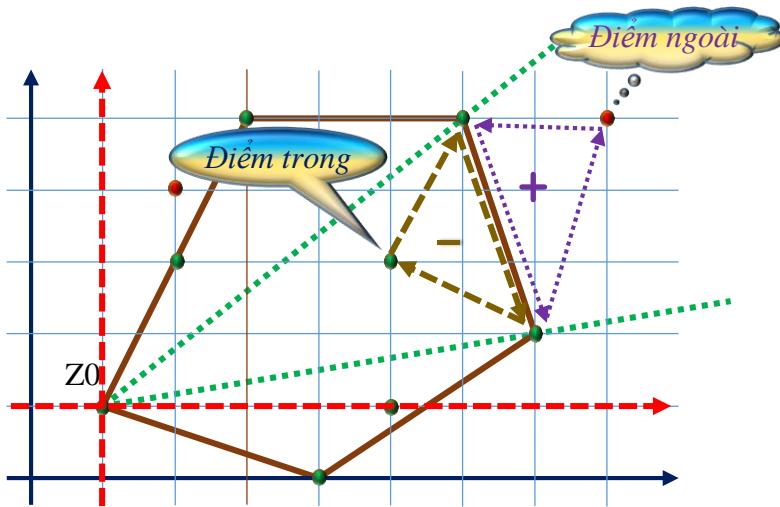
Giải thuật

Việc xác định một điểm có nằm trong đa giác lồi đã cho hay không được thực hiện bằng phương pháp tìm kiếm nhị phân theo góc.

Tìm đỉnh \mathbf{z}_0 trái nhất và thấp nhất của đa giác, đánh số lại các đỉnh đa giác bắt đầu từ \mathbf{z}_0 ,

Tịnh tiến trực tọa độ, đưa \mathbf{z}_0 về gốc tọa độ,

Như vậy các đỉnh đa giác đều nằm ở nửa phải của mặt phẳng tọa độ,



Các góc giữa tia nối đỉnh đa giác với gốc tọa độ và trục Ox được gọi là góc tọa độ cực, *tăng dần* khi di chuyển từ một đỉnh sang đỉnh khác theo chiều ngược kim đồng hồ,

Các góc có giá trị trong đoạn $[-\frac{\pi}{2}, \frac{\pi}{2}]$, có thể tính bằng hàm **atan2**, tuy nhiên hàm này cho giá trị thực và việc tích lũy sai số làm tròn có thể dẫn đến kết quả sai lệch.

Ở đây ta không cần bắn thân giá trị góc mà chỉ cần so sánh 2 góc, vì vậy có thể hoàn toàn xác định được quan hệ so sánh góc với các tọa độ nguyên: với 2 điểm tọa độ $(\mathbf{x}_1, \mathbf{y}_1)$ và $(\mathbf{x}_2, \mathbf{y}_2)$ cùng ở nửa phải mặt phẳng tọa độ tang của góc tọa độ cực tương ứng sẽ là $\mathbf{y}_1/\mathbf{x}_1$ và $\mathbf{y}_2/\mathbf{x}_2$, như vậy, chỉ cần *xử lý 2 phân số* này là có được quan hệ so sánh.

Với điểm **A** tọa độ (\mathbf{x}, \mathbf{y}) , bằng phương pháp tìm kiếm nhị phân, ta có thể xác định hai đỉnh liên tiếp nhau **P1** và **P2** của đa giác, **P1** có góc tọa độ cực nhỏ hơn hoặc bằng góc tọa độ cực của **A**, **P2** có góc tọa độ cực lớn hơn góc tọa độ cực của **A**.

Xét tam giác với các đỉnh **P1**, **P2**, **A** theo đúng trình tự đỉnh đã liệt kê ta có thể dễ dàng xác định **A** nằm bên trái hay phải của đoạn **P1**→**P2**. Nếu **A** nằm bên trái hoặc

trên cạnh – **A** thuộc đa giác, trong trường hợp ngược lại – ngoài đa giác. Điều này có thể thực hiện bằng cách tính diện tích có dấu của tam giác.

Việc xác định một đỉnh có nằm trong đa giác hay không có độ phức tạp $O(\log n)$. Các công việc chuẩn bị chung cho mọi lần tìm kiếm có độ phức tạp $O(n)$.

Xử lý

Nhập dữ liệu về đa giác và đánh số lại:

The diagram shows a yellow rectangular box containing C++ code. A blue speech bubble labeled "Tim vị trí z0" points to the first few lines of code. A blue cloud-like shape labeled "Đẩy vòng tròn, đưa z0 về đầu" points to the rotation part. A brown rounded rectangle at the bottom right contains the text "z0 không tham gia tính góc".

```
fi>>n;
vector<pll>p(n);iz0=0;
for(int i=0;i<n;++i)
{
    fi>>p[i].first>>p[i].second;
    if(p[i]<p[iz0])iz0=i;
}
pll z0=p[iz0];
rotate(p.begin(),p.begin()+iz0,p.end());
p.erase(p.begin());--n;
```

Tính tiền tọa độ và xác định tham số tang các góc:

The diagram shows a yellow rectangular box containing C++ code. A blue speech bubble labeled "Trường hợp điểm trên trực Oy" points to the angle calculation part.

```
vector<pll>a(n);
for(int i=0;i<n;++i)
{
    a[i].first = p[i].second - z0.second;
    a[i].second = p[i].first - z0.first;
    if(a[i].first==0)
        a[i].second=(a[i].second<0)? -1: 1;
}
```

Xử lý truy vấn:

Chương trình

```
#include <bits/stdc++.h>
#define NAME "inside."
using namespace std;
typedef int64_t ll;
typedef pair<ll, ll> pll;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n, m, iz0;
ll x, y;

fi>>x>>y; pll q={x, y};
bool in=false;
if (q==z0) in=true;
else
{
    pll mq={y-z0.second, x-z0.first};
    if (mq.first==0) mq.second=(mq.second<0) ? -1 : 1;
    auto it=upper_bound(a.begin(), a.end(), mq,
                         [&](pll a, pll b)
                         {return (a.first*b.second <=a.second*b.first); });
    if (it==a.end() && mq==a[n-1]) it=a.end()-1;
    if (it!=a.end() && it!=a.begin())
    {
        int p1=int(it-a.begin());
        in=(area(p[p1], p[p1-1], q)<=0);
    }
    fo<<((in) ? "Inside\n" : "Outside\n");
}
```

Tính tiền và tính góc

Tiêu chuẩn tìm kiếm nhị phân

Xác định P1

Kiểm tra điểm trong

```
ll area(pll a, pll b, pll c)
{
    return (a.first*(b.second-c.second) +
            b.first*(c.second-a.second) +
            c.first*(a.second-b.second) );
}
int main()
{
    fi>>n;
    vector<pll>p(n); iz0=0;
    for(int i=0; i<n; ++i)
    {
        fi>>p[i].first>>p[i].second;
        if (p[i]<p[iz0]) iz0=i;
    }
    pll z0=p[iz0];
    rotate(p.begin(), p.begin()+iz0, p.end());
    p.erase(p.begin()); --n;
    vector<pll>a(n);
    for(int i=0; i<n; ++i)
```

```

{
    a[i].first = p[i].second - z0.second;
    a[i].second = p[i].first - z0.first;
    if(a[i].first==0) a[i].second=(a[i].second<0)? -1: 1;
}
fi>>m;
for(int i=0;i<m;++i)
{
    fi>>x>>y; pll q={x,y}; bool in=false;
    if(q==z0) in=true;
    else
    {
        pll mq={y-z0.second,x-z0.first};
        if(mq.first==0) mq.second=(mq.second<0)? -1: 1;
        auto it=upper_bound(a.begin(),a.end(),mq,
            [&](pll a, pll b)
            {return (a.first*b.second <=a.second*b.first);});
        if(it==a.end() && mq==a[n-1]) it=a.end()-1;
        if(it!=a.end() && it!=a.begin())
        {
            int p1=int(it-a.begin());
            in=(area(p[p1],p[p1-1],q)<=0);
        }
    }
    fo<<((in)? "Inside\n" : "Outside\n");
}
fo<<"\nTime: "<<clock() / (double) 1000<<" sec";
}

```



Tìm cặp điểm gần nhất

Bài toán

Cho n điểm trên mặt phẳng, điểm thứ i có tọa độ $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1 \div n$. Khoảng cách d giữa 2 điểm i và j được tính theo công thức $d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ (*Khoảng cách Euclide*).

Hãy xác định khoảng cách ngắn nhất giữa 2 điểm trong số các điểm đã cho và chỉ ra một cặp điểm có khoảng cách ngắn nhất.

Dữ liệu: Vào từ file văn bản DISTANCE.INP:

- ✚ Dòng đầu tiên chứa một số nguyên n ($2 \leq n \leq 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên \mathbf{x}_i và \mathbf{y}_i ($|\mathbf{x}_i|, |\mathbf{y}_i| \leq 10^9$).

Kết quả: Đưa ra file văn bản DISTANCE.OUT, dòng đầu tiên là khoảng cách ngắn nhất tìm được với độ chính xác 6 chữ số sau dấu chấm thập phân, dòng thứ 2 chứa 2 số nguyên là số thứ tự của một cặp điểm có khoảng cách nhỏ nhất.

Ví dụ:

DISTANCE.INP
8
1 1
4 0
5 2
3 5
8 5
0 5
3 2
8 2

DISTANCE .OUT
2.000000
3 7

Giải thuật

Phương pháp giải: Chia và trị,

Kỹ thuật lập trình: đệ quy.

Sơ đồ xử lý là chia tập các điểm thành 2 phần, giải bài toán tương tự ở mỗi phần,

Xác định một dải đệm kết nối 2 phần, giải bài toán tương tự ở lớp đệm và tổng hợp kết quả từ các kết quả nhận được ở ba phần đã nêu.

Việc phân chia tập ban đầu thành 2 phần không quá khó khăn:

- ✚ Sắp xếp các điểm đã cho theo trình tự tăng dần của x ,
- ✚ Lấy các điểm ở nửa đầu của dãy cho vào tập **A1**, phần còn lại – vào tập **A2**.

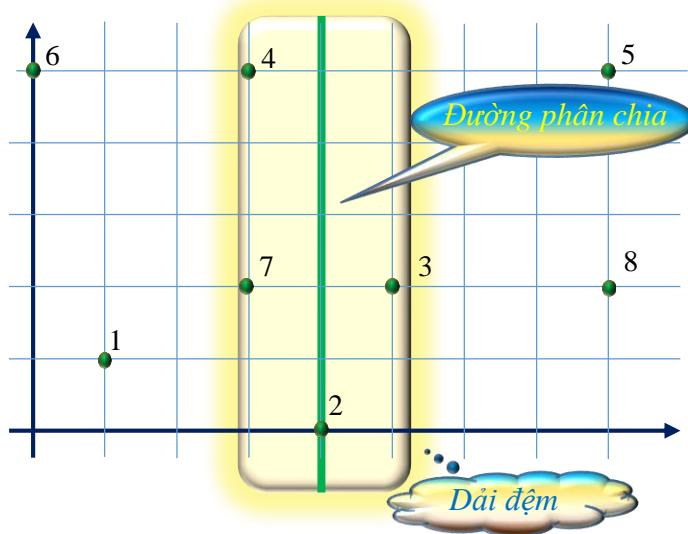
Giả thiết ta đã giải được bài toán ở các tập **A1** và **A2** với khoảng cách nhỏ nhất tìm được tương ứng là **d1** và **d2**.

Gọi $d = \min\{d1, d2\}$.

Rõ ràng tập các điểm **B** ở dải đệm chỉ chứa các điểm thỏa mãn điều kiện khoảng cách tới đường phân chia nhỏ hơn d

$$|x_v - x_B| < d,$$

trong đó x_B – tọa độ xác định đường phân chia.



Tập **C** các điểm cần xét trong dải đệm còn hẹp hơn, với mỗi điểm p_i trong **B** chỉ cần xét các điểm có chênh lệch theo y tới p_i không quá d . Gọi tập điểm cần xét với điểm

\mathbf{p}_i là $\mathbf{C}(\mathbf{p}_i)$. $\mathbf{C}(\mathbf{p}_i)$ dễ dàng xác định được nếu các điểm trong \mathbf{B} được sắp xếp theo giá trị tọa độ y .

Xử lý đệ quy:

Hàm **rec(int l, int r)** tìm khoảng cách nhỏ nhất trong số các điểm từ **l** đến **r** của dãy đã sắp xếp:

- ⊕ Nếu **r-l** là đủ bé – tiến hành vét cạn các cặp điểm, sắp xếp dữ liệu trong khoảng theo y,
- ⊕ Trong trường hợp ngược lại:
 - Chia đôi khoảng và lần lượt gọi rec với các khoảng nhận được,
 - Hợp nhất khoảng và sắp xếp theo y,
 - Xử lý dãy đệm.

Lưu ý: dùng mảng dữ liệu trung gian kiểu static để tránh việc phải phân nhiều lần, điều có thể dẫn đến hiện tượng thiếu bộ nhớ làm việc.

Độ phức tạp của giải thuật: O($n \log n$).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "distance."
using namespace std;
typedef pair<double, double> pdd;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int MAXN=100001;
int n;
struct pt {int x, y, id; } a[MAXN];
inline bool cmp_x (const pt & a, const pt & b)
    { return a.x < b.x || a.x == b.x && a.y < b.y; }
inline bool cmp_y (const pt & a, const pt & b) { return a.y < b.y; }
double mindist; int ansa, ansb;
inline void upd_ans (const pt & a, const pt & b)
    {double dist = sqrt ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y) + .0);
     if (dist < mindist) mindist = dist, ansa = a.id, ansb = b.id; }

void rec (int l, int r)
{if (r - l <= 3)
{
    for (int i=l; i<=r; ++i)
        for (int j=i+1; j<=r; ++j)
            upd_ans (a[i], a[j]);
    sort (a+l, a+r+1, &cmp_y);
    return;
}
int m = (l + r) >> 1;
int midx = a[m].x;
rec (l, m), rec (m+1, r);
static pt t[MAXN];
merge (a+l, a+m+1, a+m+1, a+r+1, t, &cmp_y);
copy (t, t+r-l+1, a+l);
int tsz = 0;
for (int i=l; i<=r; ++i)
    if (abs (a[i].x - midx) < mindist)
    {
        for (int j=tsz-1; j>=0 && a[i].y - t[j].y < mindist; --j)
            upd_ans (a[i], t[j]);
        t[tsz++] = a[i];
    }
}
int main()
{
    fi>>n;
    for(int i=0;i<n;++i)
    {
        fi>>a[i].x>>a[i].y; a[i].id=i;
    }
    sort(a,a+n,&cmp_x);
    mindist=1e19;
    rec(0,n-1);
    fo<<fixed<<setprecision(6)<<mindist<<' \n'<<ansa+1<<' ' <<ansb+1;
    fo<<"\nTime: "<<clock () / (double) 1000<<" sec";
}
```



Đường thẳng quét

Bài toán

Cho n đoạn thẳng trên mặt phẳng, đoạn thứ i được xác định bởi điểm đầu tọa độ $(\mathbf{x}_i, \mathbf{y}_i)$ và điểm cuối tọa độ $(\mathbf{u}_i, \mathbf{v}_i)$, $i = 1 \dots n$.

Hãy xác định có tồn tại một cặp đoạn thẳng nào đó giao nhau hay không và đưa ra số thứ tự của các đoạn đó. Nếu không tồn tại cặp đoạn thẳng giao nhau thì đưa ra 2 số -1 và -1 . Hai đoạn thẳng được gọi là giao nhau nếu có ít nhất một điểm chung.

Dữ liệu: Vào từ file văn bản INTERSECT.INP:

- ✚ Dòng đầu tiên chứa một số nguyên n ($2 \leq n \leq 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 4 số nguyên $\mathbf{x}_i, \mathbf{y}_i, \mathbf{u}_i, \mathbf{v}_i$, các tọa độ có giá trị tuyệt đối không vượt quá 10^9 .

Kết quả: Đưa ra file văn bản INTERSECT.OUT 2 số nguyên – số thứ tự của 2 đoạn thẳng giao nhau. Nếu tồn tại nhiều cặp đoạn thẳng giao nhau thì đưa ra một kết quả tùy chọn. Nếu không tồn tại cặp đoạn thẳng giao nhau thì đưa ra -1 và -1 .

Ví dụ:

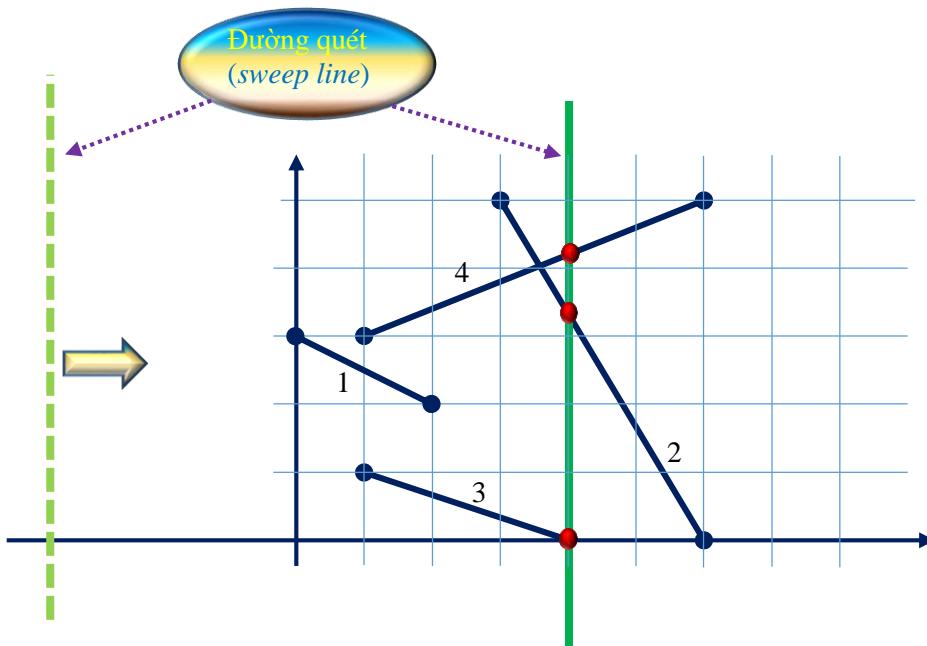
INTERSECT.INP
4
0 3 2 2
3 5 6 0
1 1 4 0
1 3 6 5

INTERSECT .OUT
4 2

Giải thuật

Phương pháp: giải thuật *đường thẳng quét* (*Sweep line*).

Đầu tiên xét trường hợp không có đoạn thẳng nào song song với trục Oy. Hình dung ta có đường thẳng $x = -\infty$ (đường thẳng vuông góc với trục O). Tịnh tiến đường thẳng này sang phải. Đến một lúc nào đó nó sẽ cắt một hoặc một số đoạn thẳng đã cho, tức



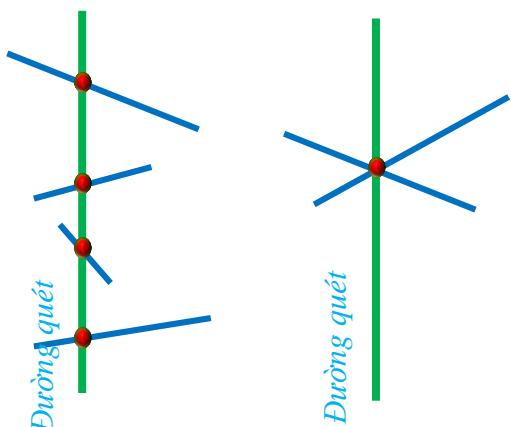
là có với mỗi đoạn thẳng đó một điểm chung và đến một lúc nào đó – không cắt các đoạn thẳng đó nữa!

Điều mà ta quan tâm là vị trí tương đối của các đoạn thẳng bị đường quét cắt, tức là tọa độ y của giao điểm với đường quét.

Nếu 2 đoạn thẳng giao nhau thì sớm hay muộn giao điểm đó sẽ nằm trên đường quét.

Có thể rút ra một số kết luận:

- ✚ Để tìm cặp đoạn thẳng cắt nhau chỉ cần *kiểm tra cặp đoạn thẳng liên tiếp nhau* theo vị trí cắt trên đường quét,
- ✚ Không cần xét mọi vị trí của đường quét, chỉ cần quan tâm tới các vị trí khi nó *bắt đầu có điểm chung với đoạn mới* hoặc *khi một điểm chung bị mất*,
- ✚ Khi có đoạn mới cắt đường quét, cần nạp nó vào *danh sách quản lý các đoạn*



bị cắt theo *trình tự từ dưới lên trên*,

- ✚ Chỉ cần kiểm tra cắt nhau giữa đoạn mới với *hai đoạn trên* và *dưới* nó,
- ✚ Khi loại một đoạn thẳng khỏi danh sách quản lý, cần *kiểm tra* khả năng cắt nhau của 2 đoạn *trên* và *dưới* đoạn bị loại,
- ✚ Không cần lưu tọa độ điểm cắt với đường quét cũng như thực hiện các loại kiểm tra nào khác.

Các kết luận trên được rút ra từ các nhận xét:

Với hai đoạn không cắt nhau *vị trí tương đối* của chúng trong tập quản lý *không thay đổi*, thật vậy, nếu một đoạn lúc đầu đứng trước đoạn kia, tức là giao với đường quét cao hơn, nhưng nếu sau đó – lại đứng thấp hơn thì giao của 2 đoạn với đường quét đã có lúc gặp nhau trên đường quét, tức là 2 đoạn giao nhau, đó là điều trái với giả thiết ban đầu!

Hệ quả từ nhận xét trên: Trong tập các đoạn đang được quản lý không tồn tại đồng thời 2 điểm giao cùng độ cao y trên đường quét với 2 đoạn không giao nhau,

Như vậy với các đoạn không giao nhau, sau khi đã ghi nhận trình tự xuất hiện trên đường quét ta chỉ phải chờ đợi khi nó ra khỏi đường quét,

Với các đoạn giao nhau: ở thời điểm giao nhau chúng *đứng cạnh nhau trong dòng xếp hàng quản lý* các đoạn giao với đường quét,

Như vậy ta chỉ cần *kiểm tra 2 đoạn liên tiếp* nhau trong dòng xếp hàng và việc kiểm tra chỉ cần thực hiện *khi xuất hiện đoạn mới* hay *một đoạn rời khỏi dòng xếp hàng* (thay đổi láng giềng).

Lưu ý:

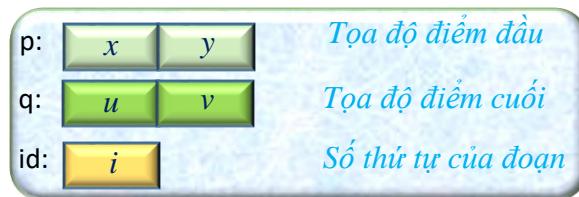
- ✚ Nếu đồng thời xuất hiện đoạn mới và loại bỏ đoạn cũ thì phải *thực hiện các phép xử lý liên quan tới bỏ sung trước*, sau đó mới xử lý sự kiện loại bỏ. Quy trình này cho phép *xử lý cả các đoạn song song với trực Oy*. Với những đoạn loại này thời điểm bỏ sung trùng với thời điểm loại bỏ, nhưng nó vẫn được kết nạp vào dòng xếp hàng và *xử lý trước khi loại bỏ*. Đoạn song song với Oy có nhiều điểm giao với đường quét và có thể có nhiều hơn 2 láng giềng, nhưng ta chỉ cần làm việc với 2 trong số đó là đủ!
- ✚ Các đại lượng thực trung gian trong giải thuật được dẫn xuất trực tiếp từ các dữ liệu nguyên, vì vậy không sẽ không có hiện tượng tích lũy sai số làm tròn. Tuy nhiên, do đại lượng thực được lưu trữ gần đúng nên 2 đại lượng được coi là bằng nhau nếu nó chênh lệch không quá EPS đủ nhỏ.

Đánh giá độ phức tạp của giải thuật:

- Độ phức tạp O(1) để ghi nhận hoặc loại bỏ một đoạn vào dòng quản lý,
- Tối đa phải thực hiện n lần ghi nhận/loại bỏ,
- Thực hiện kiểm tra giao nhau không quá $2n$ lần, mỗi lần – O(log n),
- Tổng hợp: Độ phức tạp của giải thuật: O(nlog n).

Tổ chức dữ liệu và xử lý:

Thông tin về các đoạn thẳng lưu dưới dạng mảng bản ghi:



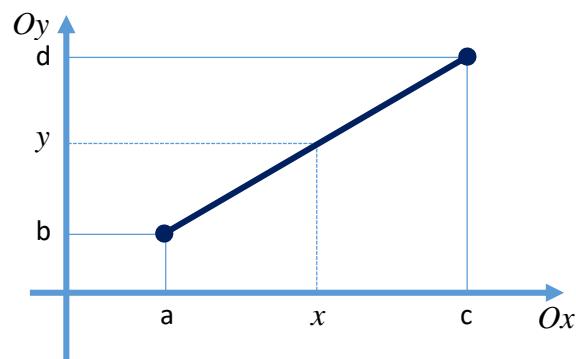
C++ là ngôn ngữ lập trình hướng đối tượng (*OOP – Object Oriented Programming*), nó cho phép gắn phép xử lý với đối tượng.

Với đoạn thẳng điểm đầu (a, b), điểm cuối (u, v), phương trình đường thẳng chứa đoạn thẳng sẽ có dạng:

$$\frac{x - a}{c - a} = \frac{y - b}{d - b}$$

Từ đây có

$$y = b + (d - b) \times \frac{x - a}{c - a}$$



```

struct pt {
    double x, y;
};

struct seg
{
    pt p, q;
    int id;
    double get_y (double x) const
    {
        if (abs (p.x - q.x) < EPS) return p.y;
        return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
    }
};

```

Phép xử lý

Đoạn thẳng là một điểm

Trả về tọa độ y

Với đường quét đi qua điểm $(x, 0)$ ta cần tính tung độ giao điểm với đoạn thẳng, vì vậy *khai báo kiểu dữ liệu* có dạng:

Kiểm tra 2 đoạn thẳng cắt nhau:

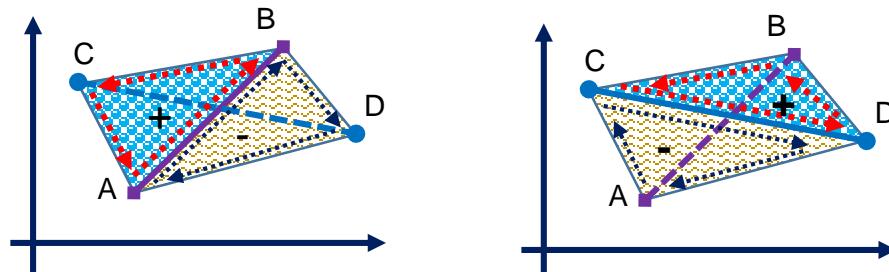
Hai đoạn thẳng AB và CD cắt nhau ở *điểm trong* của mỗi đoạn khi và chỉ khi đồng thời thỏa mãn 2 điều kiện: A và B nằm ở 2 phía của đường thẳng chứa CD và C, D nằm ở 2 phía của đường thẳng chứa AB.

Tuy nhiên, ta còn phải lưu ý trường hợp điểm cắt nhau là điểm đầu hay cuối của một đoạn thẳng nào đó.

Điều kiện cần của việc 2 đoạn thẳng có điểm chung là các *hình chiếu* của chúng trên trực tọa độ phải *có điểm chung*.

Việc giải *hệ phương trình tuyến tính 2 ẩn số* tìm điểm chung đòi hỏi phải xử lý trường hợp *hệ phương trình vô nghiệm* hay có *vô số nghiệm*.

Để tránh các phép kiểm tra phức tạp ta tính diện tích có dấu ΔABC theo trình tự duyệt $A \rightarrow B \rightarrow C$ và diện tích có dấu ΔABD theo trình tự duyệt $A \rightarrow B \rightarrow D$. Nếu 2 diện tích này trái dấu – C và D nằm ở 2 phía của AB. Tương tự như vậy đối với đoạn thẳng CD và các điểm A, B.



Trường hợp tồn tại diện tích bằng 0 – kết quả kiểm tra hình chiếu sẽ hỗ trợ việc rút ra kết luận.

Nhóm hàm sau thực hiện các công việc kiểm tra nêu trên:

*Macro kiểm tra
hình chiếu*

```
inline bool intersect1d (double l1, double r1, double l2, double r2)
{
    if (l1 > r1) swap (l1, r1);
    if (l2 > r2) swap (l2, r2);
    return max (l1, l2) <= min (r1, r2) + EPS;
}

inline int vec (const pt & a, const pt & b, const pt & c)
{
    double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
    return abs(s)<EPS ? 0 : s>0 ? +1 : -1;
}

bool intersect (const seg & a, const seg & b)
{
    return intersect1d (a.p.x, a.q.x, b.p.x, b.q.x)
        && intersect1d (a.p.y, a.q.y, b.p.y, b.q.y)
        && vec (a.p, a.q, b.p) * vec (a.p, a.q, b.q) <= 0
        && vec (b.p, b.q, a.p) * vec (b.p, b.q, a.q) <= 0;
}
```

*Macro tính
diện tích*

Macro tương tự như hàm, nhưng sẽ được triển khai ngay tại nơi gọi.

Mảng **vector<event>** e ghi nhận sự kiện đường quét có còn cắt đoạn thẳng thứ i hay không và nếu còn thì cắt bắt đầu và kết thúc ở các hoành độ nào.

Cấu trúc **event** được kết gắn với phép xử lý của 2 cấu trúc **event**, khác với cách kết gắn xử lý ở cấu trúc **seg** – xử lý *dữ liệu nội bộ* một cấu trúc.

```
struct event
{
    double x;
    int tp, id;
    event() { }
    event (double x, int tp, int id)
        : x(x), tp(tp), id(id)
    { }

    bool operator< (const event & e) const
    {
        if (abs (x - e.x) > EPS) return x < e.x;
        return tp > e.tp;
    }
};
```

Tập hợp `set<seg>` s quản lý các đoạn bị đường quét cắt. Với tập hợp này có 2 macro hỗ trợ `next` và `prev` xác định 2 đoạn láng giềng của đoạn đang xét.

Mảng `vector < set<seg>::iterator >` where lưu các con trỏ chỉ tới đoạn tương ứng trong s.

Tìm đoạn giao nhau:

- Khi đoạn mới xuất hiện: kiểm tra đoạn mới với đoạn trên và với đoạn dưới trong s,
- Khi một đoạn bị loại: kiểm tra 2 láng giềng của nó,
- Lưu ý trường hợp khi chỉ có một láng giềng.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "intersect."
using namespace std;
typedef pair<int,int> pii;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n;
pii ans;
const double EPS = 1E-9;
struct pt {
    double x, y;
};
struct seg
{
    pt p, q;
    int id;
    double get_y (double x) const
    {
        if (abs (p.x - q.x) < EPS)  return p.y;
        return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
    }
};

vector<seg> a;

inline bool intersect1d (double l1, double r1, double l2, double r2)
{
    if (l1 > r1)  swap (l1, r1);
    if (l2 > r2)  swap (l2, r2);
    return max (l1, l2) <= min (r1, r2) + EPS;
}

inline int vec (const pt & a, const pt & b, const pt & c)
{
    double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
    return abs(s)<EPS ? 0 : s>0 ? +1 : -1;
}

bool intersect (const seg & a, const seg & b)
{
    return intersect1d (a.p.x, a.q.x, b.p.x, b.q.x)
        && intersect1d (a.p.y, a.q.y, b.p.y, b.q.y)
        && vec (a.p, a.q, b.p) * vec (a.p, a.q, b.q) <= 0
        && vec (b.p, b.q, a.p) * vec (b.p, b.q, a.q) <= 0;
}

bool operator< (const seg & a, const seg & b)
{
    double x = max (min (a.p.x, a.q.x), min (b.p.x, b.q.x));
    return a.get_y(x) < b.get_y(x) - EPS;
}

struct event
```

```

{
    double x;
    int tp, id;
    event() { }
    event (double x, int tp, int id)
        : x(x), tp(tp), id(id)
    { }

    bool operator< (const event & e) const
    {
        if (abs (x - e.x) > EPS) return x < e.x;
        return tp > e.tp;
    }
};

set<seg> s;
vector < set<seg>::iterator > where;

inline set<seg>::iterator prev (set<seg>::iterator it)
{
    return it == s.begin() ? s.end() : --it;
}
inline set<seg>::iterator next (set<seg>::iterator it)
{
    return ++it;
}

pair<int,int> solve (const vector<seg> & a)
{
    int n = (int) a.size();
    vector<event> e;
    for (int i=0; i<n; ++i)
    {
        e.push_back (event (min (a[i].p.x, a[i].q.x), +1, i));
        e.push_back (event (max (a[i].p.x, a[i].q.x), -1, i));
    }
    sort (e.begin(), e.end());
    s.clear();
    where.resize (a.size());
    for (size_t i=0; i<e.size(); ++i)
    {
        int id = e[i].id;
        if (e[i].tp == +1)
        {
            set<seg>::iterator nxt = s.lower_bound (a[id]), prv=prev (nxt);
            if (nxt != s.end() && intersect (*nxt, a[id]))
                return make_pair (nxt->id, id);
            if (prv != s.end() && intersect (*prv, a[id]))
                return make_pair (prv->id, id);
            where[id] = s.insert (nxt, a[id]);
        }
        else
        {
            set<seg>::iterator nxt=next(where[id]), prv=prev (where[id]);
            if (nxt != s.end() && prv != s.end() && intersect (*nxt, *prv))
                return make_pair (prv->id, nxt->id);
        }
    }
}

```

```

        s.erase (where[id]);
    }
}
return make_pair (-1, -1);
}

int main()
{
    fi>>n;
    a.resize(n);
    for(int i=0; i<n; ++i)
    {
        fi>>a[i].p.x>>a[i].p.y>>a[i].q.x>>a[i].q.y;
        a[i].id=i;
    }
    ans=solve(a);
    fo<<ans.first+1<<' '<<ans.second+1;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```



BÀI TẬP NỘI DUNG HÌNH HỌC

WB45. LÁT VỈA HÈ

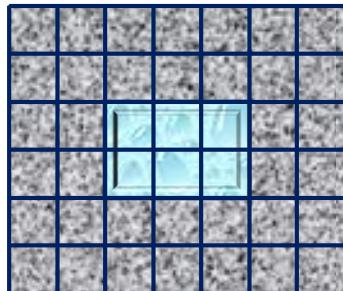
Tên chương trình: SIDEWALK.CPP

Người ta dùng các viên đá nguyên khối kích thước 1×1 để lát đoạn vỉa hè hình chữ nhật kích thước $n \times m$. Đáng tiếc, số đá được cung cấp không đủ để lát kín toàn bộ vỉa hè. Mọi người quyết định sẽ để một mảnh đất hình chữ nhật ở giữa để trồng hoa, phần còn lại sẽ lát để đi bộ. Độ rộng của đường đi bộ phải như nhau dọc theo cạnh của vỉa hè (Xem hình vẽ) và có gác lát được nhiều nhất có thể, sao cho số đá còn thừa là ít nhất.

Hãy xác định độ rộng lớn nhất có thể của đường đi bộ được lát đá.

Dữ liệu: Vào từ file SIDEWALK.INP:

- ✚ Dòng đầu tiên chứa số nguyên n và m ($3 \leq n, m \leq 10^5$),
- ✚ Dòng thứ 2 chứa số nguyên s – số viên đá được cung cấp ($1 \leq s \leq n \times m$).



Kết quả: Đưa ra file văn bản SIDEWALK.OUT một số nguyên – độ rộng lớn nhất có thể của đường đi bộ được lát đá.

Ví dụ:

SIDEWALK. INP
6 7
38

SIDEWALK. OUT
2

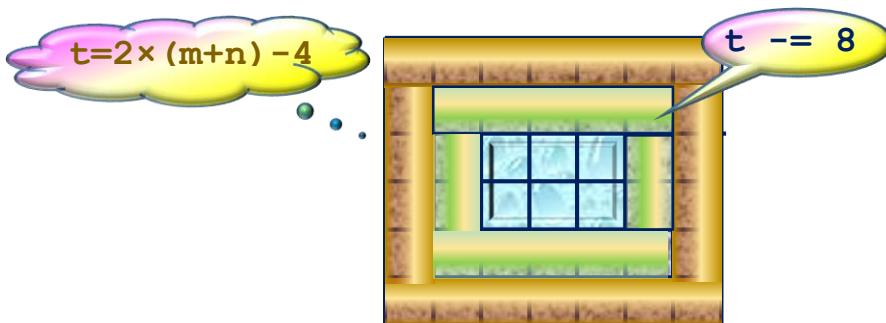


Giải thuật: Cơ sở lập trình .

Lần lượt xét số gạch cần thiết để lát vỉa hè độ rộng 1, 2, 3, ... và dừng lại khi không còn đủ số gạch.

Dễ dàng thấy rằng độ dài đường đi độ rộng 1 là $t=2 \times (m+n) - 4$.

Khi mở rộng đường thêm 1, số gạch cần thiết sẽ tăng thêm một lượng là $t-=8$.



Vì việc tăng độ rộng đường đi sẽ được thực hiện chừng nào còn đủ gạch.

Độ phức tạp của giải thuật: O(min(n,m)).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int,int> pii;
ifstream fi ("sidewalk.inp");
ofstream fo ("sidewalk.out");

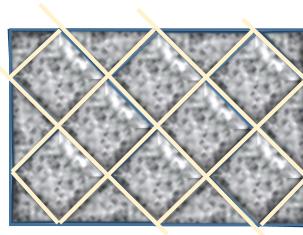
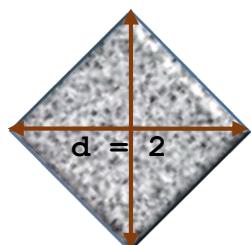
int main()
{
    int64_t n,m, s,t,d;
    fi>>n>>m>>s;
    t = 2*(n+m) - 4;
    d = 0;
    while(t <= s && t>0)
    {
        ++d; s -= t; t -= 8;
    }
    fo<<d;
    fo<<"\nTime: "<<clock() / (double) 1000<<" sec";
}
```



WB47. ỐP GẠCH MEN

Tên chương trình: *TILES.CPP*

Bức tường trong sảnh chờ của một khách sạn có hình chữ nhật kích thước $n \times m$, trong đó n và m là các số chẵn. Tường được ốp bằng gạch men hoa hình vuông với đường chéo của viên gạch là 2. Các viên gạch được đặt sao cho đường chéo song song với cạnh của bức tường. Để lát kín được toàn bộ bức tường người ta phải cưa đôi hoặc cưa 4 một số viên gạch.



Tường được lát với số viên gạch bị cưa là ít nhất.

Hãy xác định số viên gạch nguyên vẹn và số viên gạch bị cưa cần dùng để lát kín tường.

Dữ liệu: Vào từ file *TILES.INP* gồm một dòng chứa 2 số nguyên n và m , $2 \leq n, m \leq 10^5$.

Kết quả: Đưa ra file văn bản *TILES.OUT* trên một dòng 2 số nguyên – số viên gạch nguyên và số viên gạch bị cưa cần dùng.

Ví dụ:

TILES.INP
4 6

TILES.OUT
8 4

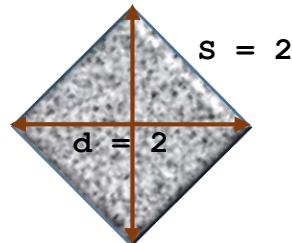
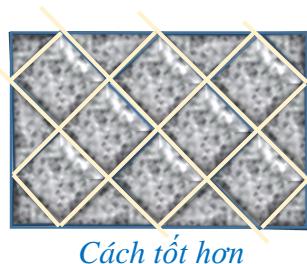
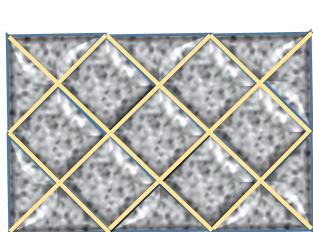


WA27 Mo_8_20192710_6_A XX

Giải thuật: Cơ sở lập trình .

Dễ dàng thấy rằng diện tích mỗi viên gạch bằng 2.

Có 2 cách lát:



Số viên gạch cần phải cắt:

$$\text{res2} = (m-2)/2 + (n-2)/2 + 1 = (m+n)/2 - 1$$

Số viên gạch nguyên:

$$\text{res1} = m \times n / 2 - \text{res2}$$

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int,int> pii;
ifstream fi ("tiles.inp");
ofstream fo ("tiles.out");

int main()
{
    int64_t n,m, res1, res2;
    fi>>n>>m;
    res2 = (m+n)/2-1;
    res1 = m*n/2 - res2;
    fo<<res1<<'\n'<<res2;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



WA43. KHU CÔNG NGHIỆP

Tên chương trình: `INDZONE.CPP`

Bước đầu hoạt động Khu công nghiệp có hình vuông diện tích 1. Sau một thời gian hoạt động có hiệu quả Khu công nghiệp mở rộng thêm một diện mạo có hình vuông diện tích 1 kề cạnh với diện tích cũ, tạo thành hình chữ nhật kích thước 1×2 . Lần mở rộng thứ i – bổ sung thêm một diện tích hình vuông cạnh bằng cạnh lớn của hình chữ nhật ở bước trước và toàn bộ diện tích luôn là một hình chữ nhật.

Hãy xác định diện tích của Khu công nghiệp sau lần thứ mở rộng thứ n và đưa ra theo mô đun 998244353.

Dữ liệu: Vào từ file input chuẩn của hệ thống, gồm một dòng chứa số nguyên n ($1 \leq n \leq 10^{18}$).

Kết quả: Đưa ra file output chuẩn của hệ thống diện tích tìm được theo mô đun 998244353.

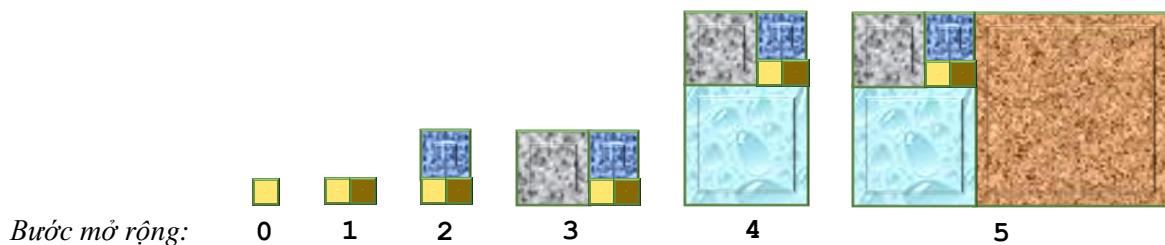
Ví dụ:

INPUT	OUTPUT
4	40



Giải thuật: Tính nhanh lũy thừa ma trận.

Hình khu công nghiệp ở các bước phát triển đầu tiên:



Dễ dàng thấy rằng sau bước mở rộng thứ i , Khu công nghiệp có hình chữ nhật với hai cạnh là các số Fibonacci F_i và F_{i+1} .

Như vậy, sau bước mở rộng thứ n diện tích s của khu công nghiệp sẽ là $F_n \times F_{n+1}$.

Với mọi n ta có

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

Việc tính F_n và F_{n+1} có thể dễ dàng thực hiện theo sơ đồ tính nhanh lũy thừa.

Độ phức tạp của giải thuật: O(log n).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "indzone."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int64_t p = 998244353;
vector<int64_t> x(4),y(4),z(4);
int64_t n,ans;

vector<int64_t> mult(vector<int64_t> a,vector<int64_t> b)
{
    vector<int64_t> t(4);
    t[0]=(a[0]*b[0]%p+a[1]*b[2]%p)%p;
    t[1]=(a[0]*b[1]%p+a[1]*b[3]%p)%p;
    t[2]=(a[2]*b[0]%p+a[3]*b[2]%p)%p;
    t[3]=(a[2]*b[1]%p+a[3]*b[3]%p)%p;
    return t;
}
int main()
{
    fi>>n;
    z[0]=z[3]=1; z[1]=z[2]=0;
    x[0]=x[1]=x[2]=1; x[3]=0;
    while(n>0)
    {
        if(n&1) z=mult(x,z);
        x=mult(x,x);
        n>>=1;
    }
    ans=z[0]*z[2]%p;

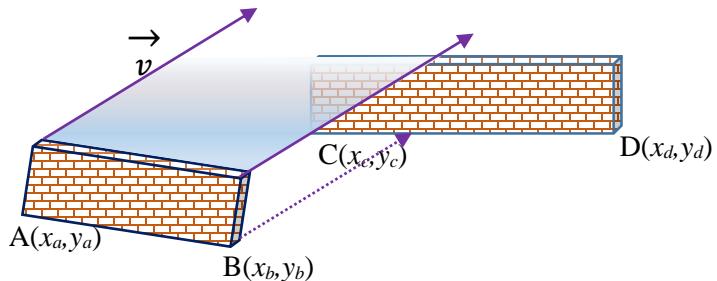
    fo<<ans;
    cerr<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



WA48. PHỦ BÓNG

Tên chương trình: OVERLAP.CPP

Có hai bức tường mỏng hình chữ nhật cùng độ cao. Chân của bức tường thứ nhất là đoạn thẳng AB có tọa độ các điểm đầu và cuối tương ứng là (x_a, y_a) và (x_b, y_b) . Chân của bức tường thứ hai là đoạn thẳng CD có tọa độ các điểm đầu và cuối tương ứng là (x_c, y_c) và (x_d, y_d) . Các bức tường không có điểm chung.



Một nguồn sáng chiếu vào bức tường thứ nhất theo hướng $\vec{v} = (\mathbf{v}_x, \mathbf{v}_y)$ tạo thành một bóng râm hình hộp chữ nhật kéo sau bức tường.

Nói bức tường thứ nhất phủ bóng lên bức tường thứ hai nếu bóng râm đè lên ít nhất một điểm của bức tường thứ hai.

Hãy xác định bức tường thứ nhất có phủ bóng lên bức tường thứ hai hay không và đưa ra thông báo **Yes** hoặc **No** tương ứng.

Dữ liệu: Vào từ file OVERLAP.INP:

- ⊕ Dòng đầu tiên chứa số nguyên n – số lượng tests cần kiểm tra ($1 \leq n \leq 50\,000$),
- ⊕ Mỗi dòng trong n dòng thiếp theo chứa thông tin về một test, bao gồm 10 số nguyên $x_a, y_a, x_b, y_b, x_c, y_c, x_d, y_d, v_x, v_y$, các số có giá trị tuyệt đối không vượt quá 10^6 , vevc to v khác 0.

Kết quả: Đưa ra file văn bản OVERLAP.OUT, kết quả mỗi test là **Yes** hoặc **No** và được đưa ra trên một dòng.

Ví dụ:

OVERLAP.INP
2
0 2 1 1 2 2 3 1 1 1
0 2 1 1 2 2 3 1 -1 -1

OVERLAP.OUT
Yes
No

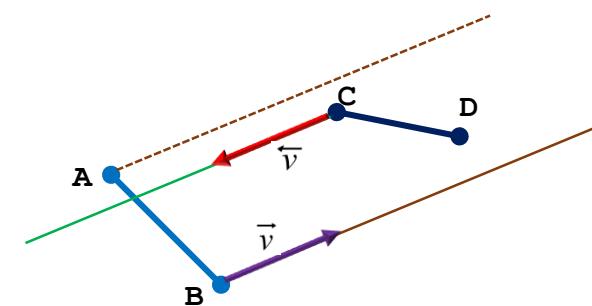
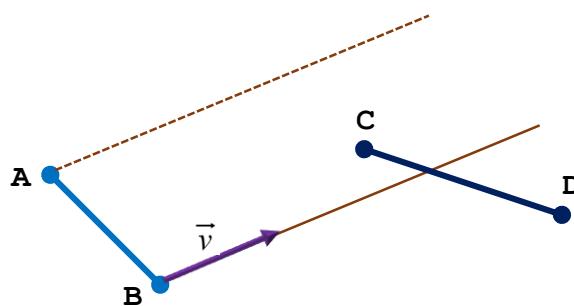


Giải thuật: Phương trình đường thẳng.

Vấn đề phải xét ở đây là kiểm tra **một tia** có **cắt đoạn thẳng** cho trước hay không.

Đoạn AB phủ bóng lên đoạn CD theo hướng \vec{v} khi thỏa mãn một trong 2 điều kiện sau:

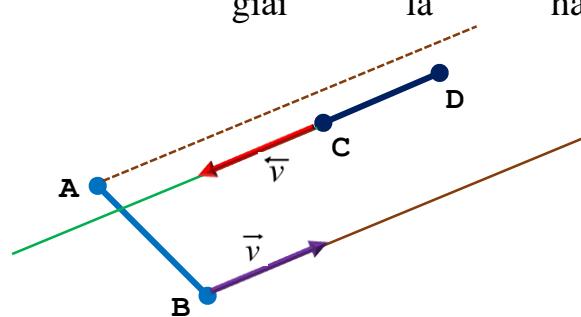
- Tia theo hướng \vec{v} từ A hoặc B có điểm chung với đoạn CD,
- Tia theo hướng \vec{v} từ C hoặc D có điểm chung với đoạn AB.



Trường hợp **CD** không đồng phương với \vec{v}

Cơ sở của lời

giải là hàm

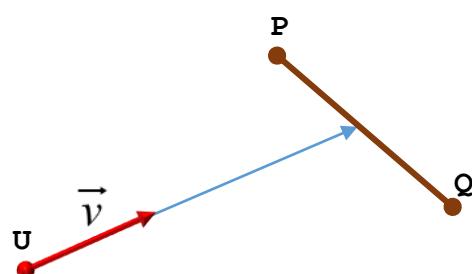


Trường hợp **CD** đồng phương với \vec{v}

f (u, v, p, q) kiểm tra tia xuất phát từ điểm **U** có \vec{v} cắt đoạn thẳng **PQ** hay không. Hàm **f** cho giá trị **true** nếu tia cắt đoạn thẳng và cho giá trị **false** trong trường hợp ngược lại.

Việc kiểm tra được thực hiện với 4 trường hợp:

- $U = A, P = C, Q = D$ và $V = \vec{v}$
- $U = B, P = C, Q = D$ và $V = \vec{v}$
- $U = C, P = A, Q = B$ và $V = \vec{v}$
- $U = D, P = A, Q = B$ và $V = \vec{v}$



Xây dựng hàm **f (u, v, p, q)**:

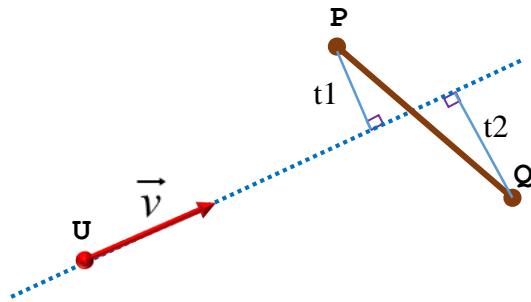
Ký hiệu tọa độ các điểm cần xét tương ứng là $(\mathbf{u}_x, \mathbf{u}_y)$, $(\mathbf{p}_x, \mathbf{p}_y)$, $(\mathbf{q}_x, \mathbf{q}_y)$ và phương $(\mathbf{v}_x, \mathbf{v}_y)$.

Phương trình đường thẳng đi qua điểm \mathbf{U} theo phương \mathbf{v} :

$$\frac{x - u_x}{v_x} = \frac{y - u_y}{v_y}$$

$$x \times v_y - y \times v_x + u_y \times v_x - u_x \times v_y = 0$$

Khoảng cách có dấu *chưa chuẩn hóa* từ \mathbf{P} và \mathbf{Q} tới đường thẳng trên sẽ là:



$$t1 = p_x \times v_y - p_y \times v_x + u_y \times v_x - u_x \times v_y$$

$$t2 = q_x \times v_y - q_y \times v_x + u_y \times v_x - u_x \times v_y$$

Nếu $t1$ và $t2$ khác 0 và có cùng dấu – tia đang xét không cắt đoạn thẳng.

Nếu $t1 = 0$ và $t2 = 0$ ta có một trong hai trường hợp:



Xét phương trình của tia xuất phát từ \mathbf{U} theo hướng \vec{v} :

$$\mathbf{x} = \mathbf{u}_x + t * \mathbf{v}_x ,$$

$$\mathbf{y} = \mathbf{u}_y + t * \mathbf{v}_y , \quad (Phương trình dưới dạng tham số)$$

$$t \in [0, \infty] .$$

Giải phương trình để xác định t cho tọa độ của tia ứng với điểm \mathbf{P} :

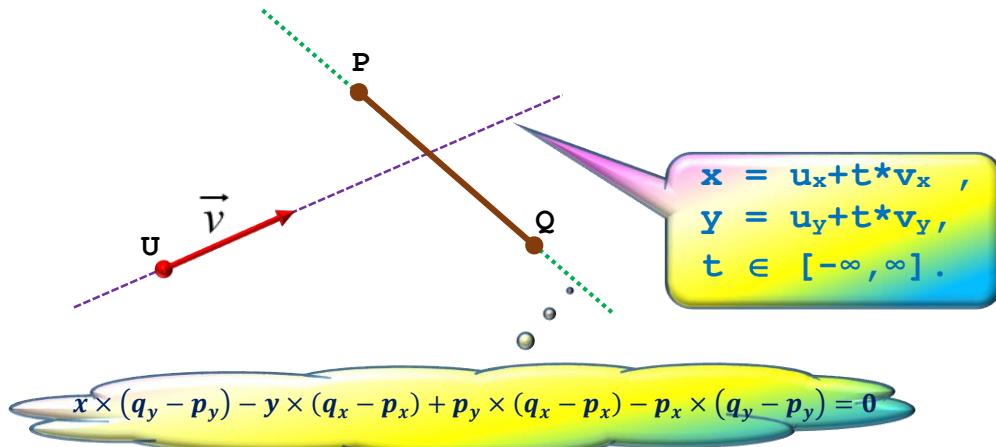
$$\mathbf{p}_x = \mathbf{u}_x + t * \mathbf{v}_x \rightarrow t * \mathbf{v}_x = \mathbf{p}_x - \mathbf{u}_x$$

$$\mathbf{p}_y = \mathbf{u}_y + t * \mathbf{v}_y \rightarrow t * \mathbf{v}_y = \mathbf{p}_y - \mathbf{u}_y$$

Cần xét cả 2 phương trình vì \mathbf{v}_x hoặc \mathbf{v}_y có thể bằng 0.

Tồn tại nghiệm t không âm khi hệ số ở vế trái và giá trị ở vế phải cùng dấu.

Trường hợp đường thẳng đi qua U , có phương v cắt PQ :



Phương trình đường thẳng chứa đoạn PQ :

$$\frac{x - p_x}{q_x - p_x} = \frac{y - p_y}{q_y - p_y}$$

$$x \times (q_y - p_y) - y \times (q_x - p_x) + p_y \times (q_x - p_x) - p_x \times (q_y - p_y) = 0$$

Phương trình xác định giá trị t tương ứng với giao điểm của 2 đường:

$$(u_x + t*v_x) * (q_y - p_y) - (u_y + t*v_y) * (q_x - p_x) + p_y * (q_x - p_x) - p_x * (q_y - p_y) = 0$$

$$t * (v_x * (q_y - p_y) - v_y * (q_x - p_x)) = (u_y - p_y) * (q_x - p_x) - (u_x - p_x) * (q_y - p_y)$$

$$t * (v_x * (q_y - p_y) - v_y * (q_x - p_x)) = (u_y - p_y) * (q_x - p_x) - (u_x - p_x) * (q_y - p_y)$$

t_3

t_4

Đoạn PQ cắt tia đang xét nếu phương trình trên có nghiệm t không âm, tức là t_3 và t_4 phải cùng dấu.

Từ kết quả gọi hàm `f` 4 lần với các tham số đã nêu ở trên ta có thể xác định AB có phủ bóng lên CD theo hướng \vec{v} hay không.

Tổ chức dữ liệu: Tọa độ mỗi điểm đã cho có thể lưu dưới dạng cặp dữ liệu `pair<int, int>`.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
ifstream fi ("overlap.inp");
ofstream fo ("overlap.out");
typedef pair<int64_t,int64_t> pii;
int n;
pii a1, a2, b1, b2, vv, vr;

bool f(pii u, pii v, pii p, pii q)
{
    int64_t t1,t2,t3,t4,t33,t44;
    t3=u.ss*v.ff-u.ff*v.ss;
    t1=v.ss*p.ff-v.ff*p.ss+t3;
    t2=v.ss*q.ff-v.ff*q.ss+t3;
    if((t1>0 && t2>0) || (t1<0 && t2<0)) return false;

    if(t1==0 && t2==0)
        if((v.ff>=0 && p.ff-u.ff>=0 && v.ss>=0 && p.ss-u.ss>=0) ||
           (v.ff<=0 && p.ff-u.ff<=0 && v.ss<=0 && p.ss-u.ss<=0)) return true;
        else return false;

    t3=v.ff*(q.ss-p.ss)-v.ss*(q.ff-p.ff);
    t4=(u.ff-p.ff)*(q.ss-p.ss)+(u.ss-p.ss)*(q.ff-p.ff);

    if(t3>=0) t33=1; else t33=-1; if(t4>=0) t44=1; else t44=-1;
    if(t4==0 || (t33*t44>0)) return true; else return false;
}

int main()
{
    fi>>n;
    for(int i=0; i<n; ++i)
    {
        fi>>a1.ff>>a1.ss>>a2.ff>>a2.ss>>b1.ff>>b1.ss>>
            b2.ff>>b2.ss>>vv.ff>>vv.ss;
        vr.ff=-vv.ff; vr.ss=-vv.ss;
        if(f(a1,vv,b1,b2) || f(a2,vv,b1,b2) ||
           f(b1,vr,a1,a2) || f(b2,vr,a1,a2))
            fo<<"Yes\n";
        else fo<<"No\n";
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```

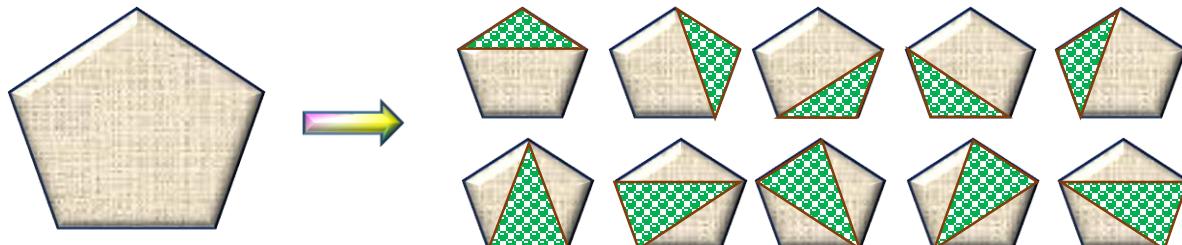


WA49. TAM GIÁC CÂN

Tên chương trình: ISOSCELE.CPP

Tam giác cân là tam giác có 2 cạnh bằng nhau. Tam giác đều là trường hợp riêng của tam giác cân.

Cho đa giác đều n đỉnh. Hãy xác định số tam giác cân có đỉnh đồng thời là đỉnh của đa giác đều.



Ví dụ, với $n = 5$ ta có 10 tam giác cân.

Dữ liệu: Vào từ file ISOSCELE.INP gồm một dòng chứa số nguyên n ($3 \leq n \leq 10^9$).

Kết quả: Đưa ra file văn bản ISOSCELE.OUT một số nguyên – số tam giác cân xác định được.

Ví dụ:

ISOSCELE.INP	ISOSCELE.OUT
5	10



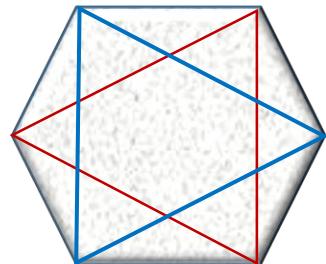
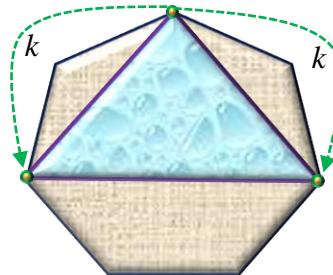
Giải thuật: Thống kê đơn giản.

Để có tam giác cân, từ một đỉnh xuất phát ta phải nối với 2 đỉnh bên phải và bên trái cách đều k đỉnh so với đỉnh xuất phát.

Từ một đỉnh xuất phát ta có $(n-1)/2$ cách chọn cặp đỉnh còn lại.

Có tất cả n cách chọn đỉnh xuất phát.

Như vậy, số tam giác cân có thể nhận được là $(n-1)/2 * n$.



Tuy vậy khi n chia hết cho 3 sẽ có $n/3$ tam giác đều, mỗi tam giác đều xuất hiện 3 lần trùng nhau, vì vậy cần phai bớt đi $2*n/3$ số lần tính lặp với các tam giác đều.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("isoscele.inp");
ofstream fo ("isoscele.out");

int main()
{
    int64_t n;
    fi >> n;
    int64_t ans = n * ((n - 1) / 2);
    if (n % 3 == 0)
        ans -= 2 * n / 3;
    fo << ans << endl;
}
```

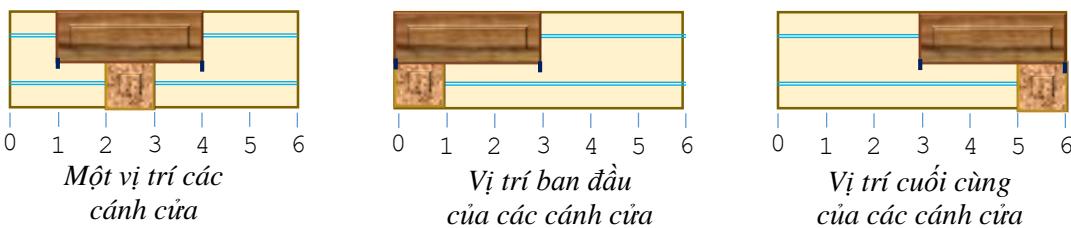


WB33. CỬA TRƯỢT

Tên chương trình: MOVEMENT.CPP

Alice học ngành thiết kế mỹ thuật. Nhiệm vụ đầu tiên của cô là thiết kế cánh cửa trượt cho một tủ tường.

Mặt tủ có hình chữ nhật kích thước $2 \times n$, có 2 cánh cửa hình chữ nhật, cánh ngắn kích thước $1 \times a$ lắp ở dưới, cánh dài kích thước $1 \times b$ lắp ở trên, có 2 thanh trượt để đẩy các cánh cửa chuyển động sang phải hoặc trái. Phạm vi chuyển động của cánh cửa dài là suốt toàn bộ chiều dài cửa tủ, cho đến khi một đầu chạm tường. Hai đầu của cánh cửa dài có các mấu giữ không cho cánh cửa ngắn trượt ra ngoài, vì vậy phạm vi chuyển động của cánh cửa ngắn là đoạn nằm trong phạm vi giữa 2 mấu giữ của cửa dài. Chỉ có thể đẩy riêng từng cánh cửa.



Ban đầu các cánh cửa nằm ở sát mép trái của cửa. Để kiểm tra độ trơn chuyen động cần đẩy các cánh cửa về vị trí sát với mép phải của cửa.

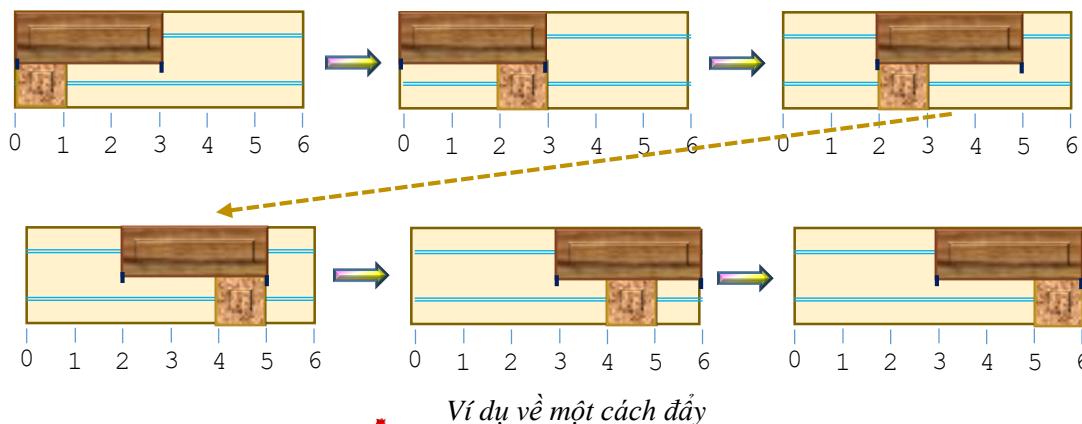
Hãy xác định số thao tác đẩy ít nhất cần thực hiện.

Dữ liệu: Vào từ file văn bản MOVEMENT.INP gồm một dòng chứa 3 số nguyên a , b và n ($1 \leq a < b \leq n \leq 10^7$).

Kết quả: Đưa ra file văn bản MOVEMENT.OUT một số nguyên – số thao tác đẩy ít nhất cần thực hiện.

Ví dụ:

MOVEMENT.INP	MOVEMENT.OUT
1 3 6	5



WB33 NWRC 2019 A XVII

Giải thuật: Cơ sở lập trình .

Khi di chuyển, cần đẩy cánh cửa sang phải nhiều nhất có thể.

Một lần di chuyển, mỗi cánh cửa lần lượt đi được một quãng đường tối đa là $b-a$,

Ban đầu cần chuyển cánh cửa ngắn sang phải tối đa có thể,

Sau đó lặp lại nhiều lần thao tác:

- Ⓐ Di chuyển cánh cửa dài,
- Ⓑ Di chuyển cánh cửa ngắn.

Số lần di chuyển cánh cửa dài là $2 \times \lceil \frac{n-b}{b-a} \rceil + 1$.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "Movement."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    int a, b, n;
    fi >> a >> b >> n;
    fo<< 1 + 2 * ((n - b + (b - a) - 1) / (b - a));
}
```



VZ33. IN PHUN 3D

Tên chương trình: P3D.CPP

Bài tập sử dụng máy in phun 3D là tạo ra các khối hộp chữ nhật từ thạch cao nhão. Hình hộp tạo ra còn khá mềm nên không thể lật hay dựng đứng hình.

Dựa vào kích thước chiều rộng, chiều dài của đáy và chiều cao của khối hộp sản phẩm sẽ được đánh giá là “**good**” hoặc “**bad**”.

Sản phẩm được đánh giá là “**good**” nếu tỷ lệ giữa cạnh bé của đáy với chiều cao phải ít nhất là 2 và tỷ lệ giữa cạnh lớn của đáy với cạnh bé của đáy không vượt quá 2.

Cho 2 kích thước đáy là **w**, **l** và chiều cao **h**.

Hãy đưa ra đánh giá đối với hình hộp chữ nhật đã cho.

Dữ liệu: Vào từ file văn bản P3D.INP gồm 3 số nguyên **w**, **l** và **h** ($10^3 \leq w, l, h \leq 10^4$), mỗi số trên một dòng.

Kết quả: Đưa ra file văn bản P3D.OUT thông báo **good** hoặc **bad**.

Ví dụ:

P3D.INP
4600
8600
1600

P3D.OUT
good



VZ33 SptM20181210 A A XVI



Giải thuật: Cơ sở lập trình.

Chuẩn hóa dữ liệu: đưa về trường hợp $w \leq l$,

Thay việc tính trực tiếp tỷ lệ bằng phép so sánh tương đương chỉ sử dụng phép nhân.

Độ phức tạp của giải thuật: $O(1)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "P3D."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    int w, l, h;
    fi >> w >> l >> h;
    if (w > l) swap(w, l);
    if (w >= h * 2 && l <= w * 2)
        fo << "good\n";
    else
        fo << "bad\n";
    return 0;
}
```



VZ34. HÌNH VUÔNG KHÁC NHAU

Tên chương trình: DIFFERENT.CPP

Để lát vỉa hè hay quảng trường người ta ép bột đá tạo thành các hình vuông đơn vị. Từ các hình vuông đơn vị này người ta ghép dán thành các hình vuông kích thước lớn hơn, tạo thành những viên gạch lát chịu lực tốt và vẫn để nước thẩm qua mặt lát.

Để quảng cáo cho mặt hàng mới này tại Hội chợ Công nghiệp hàng năm người ta làm các viên đá lát hình vuông kích thước khác nhau từ n viên đơn vị đang có sẵn để tạo thành sản phẩm mang ra trưng bày. Không nhất thiết phải sử dụng hết số viên đơn vị.

Hãy xác định nhiều nhất có bao nhiêu sản phẩm được mang ra trưng bày.

Dữ liệu: Vào từ file văn bản DIFFERENT.INP gồm một dòng chứa số nguyên n ($1 \leq n \leq 10^{18}$).

Kết quả: Đưa ra file văn bản DIFFERENT.OUT: một số nguyên – số lượng nhiều nhất các sản phẩm có thể mang ra trưng bày.

Ví dụ:

DIFFERENT.INP
15

DIFFERENT.OUT
3



VZ34 SptM20181210 B A XVI



Giải thuật: Cơ sở lập trình .

Để có nhiều hình vuông nhất cần bắt đầu từ những hình có kích thước nhỏ nhất,

Cần tìm k lớn nhất thỏa mãn điều kiện

$$\frac{k \times (k + 1) \times (2k + 1)}{6} \leq n$$

Sử dụng công thức trên để tìm k sẽ mất nhiều thời gian hơn việc tích lũy trực tiếp tổng bình phương các số tự nhiên đầu tiên và so sánh với n ,

Để tránh khả năng tràn ô ở bước kiểm tra cuối cùng nên so sánh k^2 với phần còn lại của n sau mỗi lần xác định được hình vuông tiếp theo.

Độ phức tạp của giải thuật: $O(\sqrt[3]{n})$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "Different."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    int64_t n, k=0, s;
    fi >> n;
    s = n;
    while (s >=0)
    {
        k++;
        s -= k * k;
    }
    fo << k - 1 << "\n";
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VZ45. CẤP ĐIỂM

Tên chương trình: POINTS.CPP

Với nhiều học sinh hình học thường mang lại nỗi khiếp sợ vô hình. Để chứng minh rằng cái đáng sợ là cấu trúc dữ liệu và giải thuật chứ không phải hình học thầy giáo ra một bài có nội dung hình học: Cho n điểm trên trực hoành, điểm thứ i có tọa độ $(x_i, 0)$ và n điểm trên trực tung, điểm thứ i có tọa độ $(0, y_i)$, $i = 1 \dots n$. Tất cả các điểm đều có tọa độ nguyên và không có điểm nào trùng với gốc tọa độ. Khi nối một điểm trên trực hoành với một điểm trên trực tung ta có một đoạn thẳng.

Hãy xác định có bao nhiêu cách nối mỗi điểm trên trực hoành với một điểm trên trực tung sao cho không có hai đoạn thẳng nào cắt nhau và đưa ra theo mô đun 998244353.

Dữ liệu vào từ file văn bản POINTS.INP:

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên x_1, x_2, \dots, x_n ($-10^9 \leq x_1 < x_2 < \dots < x_n \leq 10^9$),
- ✚ Dòng thứ 3 chứa n số nguyên y_1, y_2, \dots, y_n ($-10^9 \leq y_1 < y_2 < \dots < y_n \leq 10^9$).

Kết quả: Đưa ra file văn bản POINTS.OUT số nguyên tính được.

Ví dụ:

POINTS.INP	POINTS.OUT
2 -1 1 -1 2	2



Giải thuật: Số tổ hợp, tính nhanh lũy thừa và nghịch đảo mò đun .

Gọi số lượng điểm trên trực hoành có tọa độ dương là $\mathbf{x+}$, số điểm có tọa độ âm là $\mathbf{x-}$, số lượng điểm trên trực tung có tọa độ dương là $\mathbf{y+}$, số điểm có tọa độ âm là $\mathbf{y-}$.

Giả thiết ta có $\mathbf{P++}$ là số lượng cặp điểm được chọn, trong đó cả 2 tọa độ đều dương. Khi đó có thể tính được số đoạn thẳng $\mathbf{P+-}$ nối điểm hoành độ dương với điểm tung độ âm: $\mathbf{P+- = x+ - P++}$ (vì $\mathbf{P++ + P+- = x+}$).

Tương tự như vậy, có thể tính $\mathbf{P-+}$ – số đoạn thẳng nối điểm hoành độ âm với điểm tung độ dương: $\mathbf{P-+ = y++ - P++}$ và $\mathbf{P--}$ – số đoạn thẳng nối điểm hoành độ âm với điểm tung độ âm: $\mathbf{P-- = x- - P-+}$.

Biết các giá trị $\mathbf{P??}$, trong đó ? là ký tự + hoặc -, ta có thể xác định số cách chọn điểm cho mỗi nhóm.

Ký hiệu $\mathbf{C_u^v}$ là tổ hợp chập \mathbf{v} từ \mathbf{u} phần tử, ta có số cách chọn các điểm hoành độ dương để có $\mathbf{P++}$ là $\mathbf{C_{x+}^{P++}}$ và số cách chọn các điểm tung độ dương để có $\mathbf{P++}$ là $\mathbf{C_{y+}^{P++}}$.

Gọi $\mathbf{A+}$ là tập các điểm đã chọn các điểm trong $\mathbf{x+}$ và $\mathbf{B+}$ là tập các điểm trong $\mathbf{y+}$ tham gia tạo $\mathbf{P++}$, cách nối thỏa mãn điều kiện bài toán là nối điểm có *giá trị lớn nhất trong A+* với *điểm giá trị lớn nhất trong B+*, nối điểm có *giá trị lớn thứ hai trong A+* với *điểm giá trị lớn thứ hai trong B+*, ... Các đoạn thẳng nối như vậy sẽ *không cắt nhau* và nằm gọn trong *phản tư I* của mặt phẳng tọa độ.

Các điểm tham gia tạo $\mathbf{P+-}$ cũng được nối theo cách tương tự: nối điểm có *giá trị lớn nhất trong A+* với *điểm giá trị tuyệt đối lớn nhất trong B-*, nối điểm có *giá trị lớn thứ hai trong A+* với *điểm giá trị tuyệt đối lớn thứ hai trong B-*, ... Các đoạn thẳng nối như vậy sẽ *không cắt nhau* và nằm gọn trong *phản tư IV* của mặt phẳng tọa độ.

Các đoạn thẳng tạo $\mathbf{P-+}$ và $\mathbf{P--}$ cũng được tạo theo cách nối điểm tương tự đã nêu, chúng nằm gọn trong phản tư II và phản tư III, do đó không có đoạn thẳng nào giao nhau.

Một khi đã chọn các điểm thuộc $\mathbf{y+}$ tham gia tạo $\mathbf{P++}$, số điểm thuộc $\mathbf{y+}$ tham gia tạo $\mathbf{P-+}$ trở nên tiền định và $\mathbf{P-+}$ chỉ còn phụ thuộc các chọn điểm trong $\mathbf{x-}$. Để có $\mathbf{P-+}$, số cách chọn điểm có hoành độ âm là $\mathbf{C_{x-}^{P-+}}$.

Số cách chọn điểm có tung độ âm để có $\mathbf{P-+}$ chỉ còn phụ thuộc vào cách chọn điểm trong $\mathbf{y-}$ và sẽ là $\mathbf{C_{y-}^{P-+}}$.

Với mỗi cách chọn điểm để có $P++$, nếu các giá trị $P+-$, $P--$ và $P-$ không âm thì tổng số cách nối điểm sẽ tăng thêm một lượng là $C_{X+}^{P++} \times C_{Y+}^{P++} \times C_{X-}^{P-+} \times C_{Y-}^{P+-}$.

Tính $C_u^v \ mod \ q$, trong đó q – số nguyên tố:

$$C_u^v = x \rightarrow \frac{u!}{v!(u-v)!} = x \rightarrow u! = dv \times x, \text{ trong đó } dv = v! \times (u-v)!$$

Từ đây có:

$$\begin{aligned} u! \times dv^{q-2} &= dv^{q-1} \times x \\ (u! \times dv^{q-2}) \ mod \ q &= (dv^{q-1} \times x) \ mod \ q \\ &= (dv^{q-1} \ mod \ q) \times (x \ mod \ q) \\ &= 1 \times (x \ mod \ q) \\ &= x \ mod \ q \end{aligned}$$

Như vậy để tính nhanh theo mô đun q số tổ hợp C_u^v ta cần chuẩn bị bảng giá trị các giai thừa theo mô đun q và sơ đồ nâng nhanh lũy thừa một số (tính theo mô đun q).

Tổ chức dữ liệu:

- ▀ Mảng `int64_t f[maxn]` – Lưu giá trị các giai thừa theo mô đun q ,
- ▀ Các biến `pos`, `diff`, `next` quản lý số điểm đang xét tương ứng thuộc $\mathbf{X+}$, $\mathbf{Y-}$ và $\mathbf{X-}$.

Xử lý:

Sơ đồ đệ quy tính nhanh $a^b \ mod \ MOD$:

```
int64_t power(int64_t a, int64_t b)
{
    if (b == 0) return 1;
    if (b % 2 == 0)
    {
        int64_t x = power(a, b / 2);
        return (x * x) % MOD;
    } else return (power(a, b - 1) * a) % MOD;
}
```

Tính số tổ hợp C_n^k theo mô đun **MOD**:

```

int64_t inv(int64_t a, int64_t mod)
{
    return power(a, mod - 2);
}

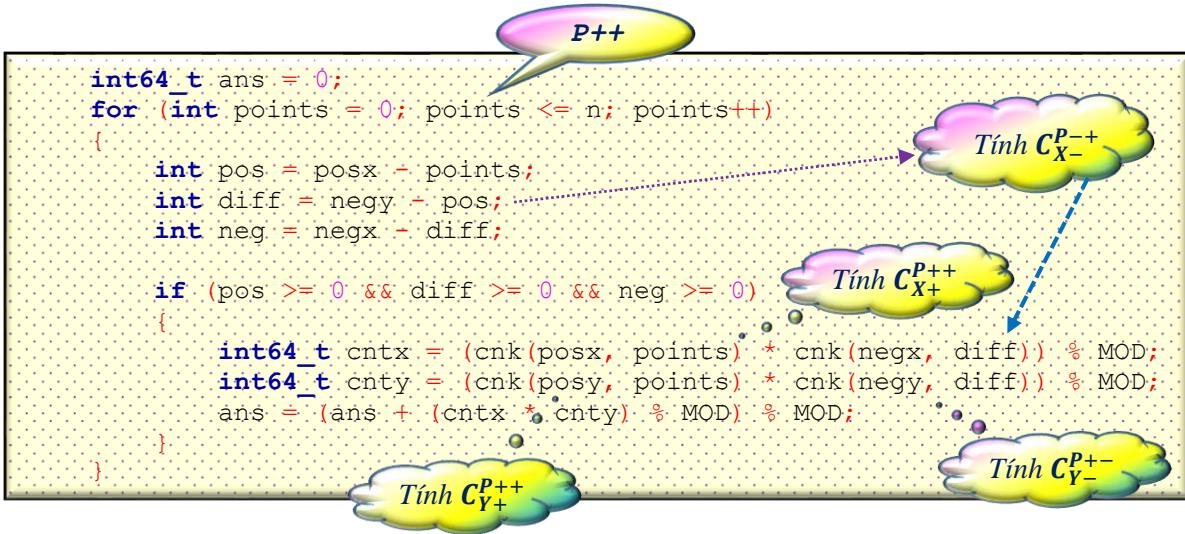
int64_t divide(int64_t a, int64_t b)
{
    return (a * inv(b, MOD)) % MOD;
}

int64_t cnk(int n, int k)
{
    if (0 <= k && k <= n)
    {
        int64_t b = (f[k] * f[n - k]) % MOD;
        return divide(f[n], b);
    }
    return 0;
}

```

Tính nghịch đảo
theo mô đun

Xác định tham số các tập điểm và chỉnh lý kết quả:



Dộ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
const int MOD = 998244353;
const int maxn = 1e5 + 10;

int64_t f[maxn];

int64_t power(int64_t a, int64_t b)
{
    if (b == 0) return 1;
    if (b % 2 == 0)
    {
        int64_t x = power(a, b / 2);
        return (x * x) % MOD;
    } else return (power(a, b - 1) * a) % MOD;
}

int64_t inv(int64_t a, int64_t mod)
{
    return power(a, mod - 2);
}

int64_t divide(int64_t a, int64_t b)
{
    return (a * inv(b, MOD)) % MOD;
}

int64_t cnk(int n, int k)
{
    if (0 <= k && k <= n)
    {
        int64_t b = (f[k] * f[n - k]) % MOD;
        return divide(f[n], b);
    }
    return 0;
}

int main()
{
    ifstream cin ("points.inp");
    ofstream cout ("points.out");
    f[0] = 1;
    for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % MOD;
    int n;
    cin >> n;
    int posx = 0, negx = 0, posy = 0, negy = 0;
    for (int i = 0; i < n; i++)
    {
        int x;
        cin >> x;
        if (x > 0) posx++; else negx++;
    }
    for (int i = 0; i < n; i++)
    {
```

```

        int y;
        cin >> y;
        if (y > 0) posy++; else negy++;
    }

int64_t ans = 0;
for (int points = 0; points <= n; points++)
{
    int pos = posx - points;
    int diff = negy - pos;
    int neg = negx - diff;

    if (pos >= 0 && diff >= 0 && neg >= 0)
    {
        int64_t cntx = (cnk(posx, points) * cnk(negx, diff)) % MOD;
        int64_t cnty = (cnk(posy, points) * cnk(negy, diff)) % MOD;
        ans = (ans + (cntx * cnty) % MOD) % MOD;
    }
}

cout << ans << '\n';
cout<<"\nTime: "<<clock()/(double)1000<<" sec";
}

```



WA07. TAM GIÁC

Tên chương trình: TRIANGLE.CPP

Alice có 3 thanh nhựa độ dài tương ứng là a , b và c . Alice định lắp một hình tam giác diện tích khác 0 có cạnh là các thanh nói trên, mỗi thanh là một cạnh.

Nếu không thể làm điều đó từ các thanh ban đầu thì Alice có thể hơ nóng và kéo dài một hoặc vài thanh để lắp. Cứ mỗi phút Alice kéo dài thanh được chọn thêm 1cm.

Hãy xác định thời gian tối thiểu cần thiết cho việc kéo dài thanh để lắp được tam giác.

Dữ liệu: Vào từ file văn bản TRIANGLE.INP gồm một dòng chứa 3 số nguyên a , b , c ($1 \leq a, b, c \leq 10^9$).

Kết quả: Đưa ra file văn bản TRIANGLE.OUT một số nguyên – thời gian tối thiểu (tính theo số phút) cần thiết cho việc kéo dài thanh để lắp được tam giác.

Ví dụ:

TRIANGLE.INP
100 10 10

TRIANGLE.OUT
81



WA07 Mos KB 20180110 BC V16

Giải thuật: Cơ sở lập trình, Kỹ năng phân tích giải thuật.

Chuẩn hóa dữ liệu: Đưa về trạng thái $\mathbf{a} \geq \mathbf{b} \geq \mathbf{c}$.

Điều kiện để lắp được tam giác: $\mathbf{b+c > a}$.

Nếu $\mathbf{b+c > a} \rightarrow$ Thời gian kéo dài thanh bằng 0.

Trong trường hợp ngược lại $\mathbf{ans = a - (b+c) + 1}$.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("triangle.inp");
ofstream fo ("triangle.out");
int a,b,c,ans=0;

int main()
{
    fi>>a>>b>>c;
    if(a<b) swap(a,b);
    if(a<c) swap(a,c);
    if(a>=b+c) ans = a-b-c+1;
    fo<<ans;
}
```

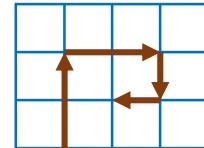


WA13. KÉO CẮT GIẤY

Tên chương trình: SCISSOR.CPP

Để trang trí phòng phục vụ tổ chức sinh nhật cho một người bạn Alice lấy một tờ giấy màu thủ công kẻ ô vuông kích thước $n \times m$ (n hàng và m cột), cắt thành hình lò xo xoắn theo hướng phải sang trái và có độ rộng của đường bằng 1:

- ▣ Bắt đầu từ biên phải cột 0 cắt lên trên cho đến khi cách lề trên một ô,
- ▣ Cắt sang phải theo đường biên dưới cho đến khi cách lề phải một ô,
- ▣ Cắt xuống dưới, rồi sang trái, sau đó lên trên, . . . để có băng giấy độ rộng 1 ô,
- ▣ Quá trình cắt sẽ dừng khi không cách cắt tiếp mà không làm đứt băng giấy.



Hãy tính tổng độ dài đường cắt theo đơn vị ô.

Dữ liệu: Vào từ file SCISSOR.INP gồm một dòng chứa 2 số nguyên n và m ($2 \leq n, m \leq 10^9$).

Kết quả: Đưa ra file văn bản SCISSOR.OUT một số nguyên – độ dài đường cắt.

Ví dụ:

SCISSOR.INP
3 4

SCISSOR.OUT
6



WA13 Io20191005 G V16

Giải thuật: Phân tích mô hình toán học.

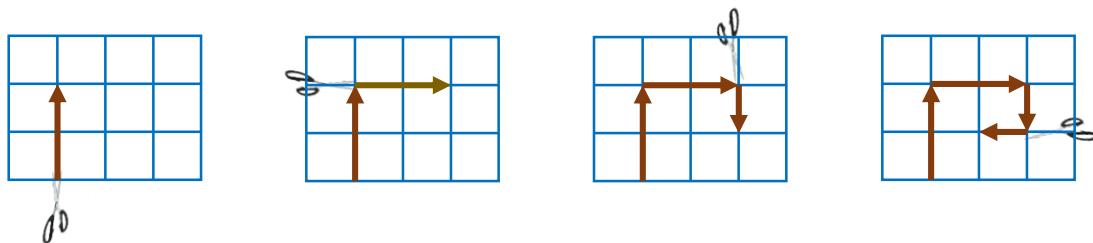
Đầu tiên xét trường hợp $n < m$.

Nhát cắt đầu tiên trước khi đổi hướng cắt có độ dài $n-1$.

Độ rộng của giấy phần bên phải của kéo là $m-1$,

Xoay kéo và ta có độ dài đường cắt tiếp theo là $m-2$,

Sau mỗi lần xoay kéo độ dài đường cắt sẽ giảm 1 so với độ dài đường cắt cùng hướng trước đó.



Việc cắt sẽ được thực hiện cho đến khi phần độ rộng bên phải kéo (sau khi cắt) có độ rộng 1.

Lần cắt	Độ dài lát cắt dọc	Độ dài lát cắt ngang
1	$n-1$	$m-1-1$
2	$n-2$	$m-1-2$
3	$n-3$	$m-1-3$
...
i	$n-i$	$m-1-i$
...
$n-1$	1	$m-1-(n-1)$

Như vậy số lần cắt theo chiều rộng và theo chiều dài là $n-1$.

Tổng độ dài các đường cắt sẽ là

$$n-1+m-2+n-2+m-3+\dots+1+(m-1-(n-1)) = \sum_{i=1}^{n-1} i + \sum_{m-2}^{m-n} i = (n-1) \times (m-1)$$

Kết quả vai trò của n và m là đối xứng, không phụ thuộc vào việc n lớn nhỏ hay bằng m .

Độ phức tạp của giải thuật: O(1).



Chương trình

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ifstream fi ("scissor.inp");
ofstream fo ("scissor.out");

int main()
{
    int64_t n,m,ans;
    fi>>n>>m;
    ans = (n-1) * (m-1);
    fo<<ans;
}
```



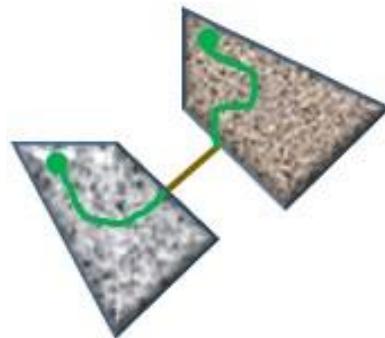
VZ18. BÃI SÌNH

Tên chương trình: MUD.CPP

Khu bảo tồn sinh thái vùng đất ngập mặn có n hòn đảo. Khi thủy triều rút toàn bộ khu bảo tồn sẽ là một bãi sinh lầy với các đảo nằm rải rác. Trên bản đồ, mỗi hòn đảo có hình đa giác lồi và không có đa giác lồi nào giao nhau, các đảo được đánh số từ 1 đến n . Đảo thứ i là đa giác lồi có k_i đỉnh, đỉnh thứ j có tọa độ $(x_{k_{i,j}}, y_{k_{i,j}})$, $j = 1, 2, \dots, k_i$. Các tọa độ đều nguyên.

Cơ sở cứu hộ và nuôi dưỡng động vật hoang dã được xây dựng ở đảo a . Người ta nhận được thông báo phải giải cứu một động vật quý hiếm bị kẹt ở đảo b . Việc di chuyển xuyên qua các hòn đảo không thành vấn đề, nhưng để vượt qua sinh lầy để tới đảo khác với các thiết bị lính kính trên lưng là vô cùng vất vả, vì vậy bao giờ người ta cũng phải tìm cách đi sao cho tổng độ dài các đoạn lội bùn là ngắn nhất.

Hãy xác định tổng độ dài ngắn nhất của các đoạn lội bùn.



Dữ liệu: Vào từ file văn bản MUD.INP:

- ✚ Dòng đầu tiên chứa 3 số nguyên n , a và b ($1 \leq n \leq 200$, $1 \leq a, b \leq n$),
- ✚ Tiếp sau là n nhóm dữ liệu, nhóm i mô tả một hòn đảo thứ i :
 - Dòng đầu tiên trong nhóm chứa số nguyên k_i ($3 \leq k_i \leq 500$),
 - Mỗi dòng trong kí dòng tiếp theo chứa 2 số nguyên x và y xác định tọa độ một đỉnh ($|x|, |y| \leq 10^9$).
 - Các đỉnh được liệt kê theo chiều ngược kim đồng hồ và không có 3 đỉnh liên tiếp nằm trên một đường thẳng.

Kết quả: Đưa ra file văn bản MUD.OUT: một số thực với độ chính xác không ít hơn 9 chữ số sau dấu chấm thập phân.

Ví dụ:

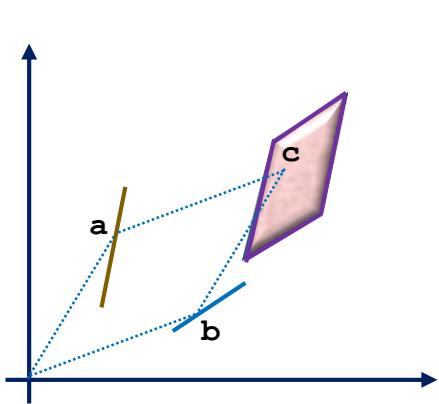
MUD.INP	MUD.OUT
<pre>2 1 2 4 2 1 3 2 2 3 1 3 4 4 2 5 2 4 4 3 3</pre>	<pre>0.707106781186548</pre>



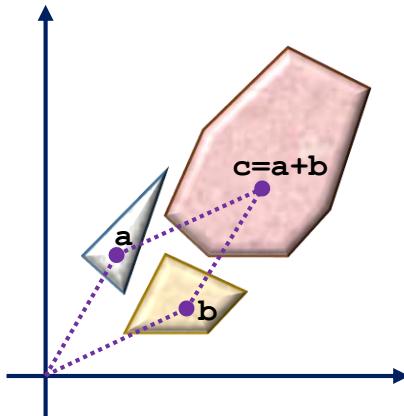
Giải thuật: Tổng Minkowski.

Cho 2 điểm trên mặt phẳng $\mathbf{a}(\mathbf{x}_a, \mathbf{y}_a)$ và $\mathbf{b}(\mathbf{x}_b, \mathbf{y}_b)$, tổng $\mathbf{a}+\mathbf{b}$ theo Minkowski là điểm $\mathbf{c}(\mathbf{x}_c, \mathbf{y}_c)$, trong đó $\mathbf{x}_c = \mathbf{x}_a + \mathbf{x}_b$, $\mathbf{y}_c = \mathbf{y}_a + \mathbf{y}_b$.

Tổng Minkowski của 2 tập \mathbf{A} và \mathbf{B} là tập $\mathbf{C} = \{\mathbf{a}+\mathbf{b}: \mathbf{a} \in \mathbf{A}, \mathbf{b} \in \mathbf{B}\}$.



Tổng 2 đoạn thẳng



Tổng 2 đa giác lồi

Có thể chứng minh được rằng tổng Minkowski của 2 đa giác lồi là một đa giác lồi.

Một trong số các ứng dụng của tổng Minkowski là tìm khoảng cách giữa 2 tập lồi.

Gọi $dist(\mathbf{A}, \mathbf{B})$ là khoảng cách giữa 2 tập \mathbf{A} và \mathbf{B} .

$$dist(\mathbf{A}, \mathbf{B}) = \min_{a \in A, b \in B} |a - b|$$

Xét trường hợp \mathbf{A}, \mathbf{B} – hai đa giác lồi. Dễ dàng xác định được \mathbf{B}' – hình *đối xứng qua tâm* của \mathbf{B} với *tâm đối xứng* là *góc tọa độ*.

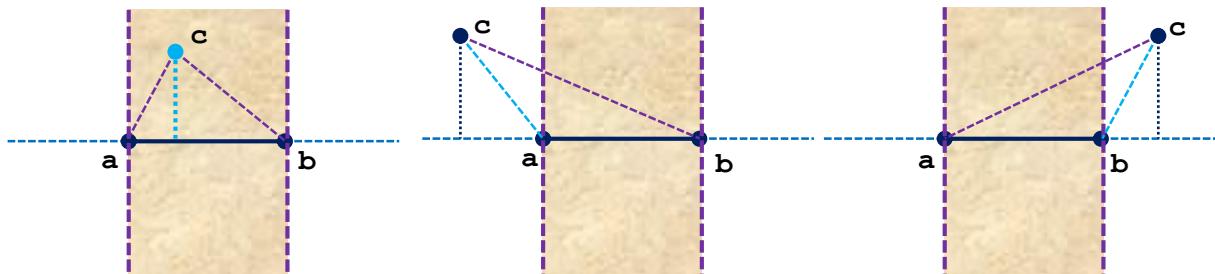
Gọi \mathbf{C} là tổng Minkowski của \mathbf{A} và \mathbf{B}' : $\mathbf{C} = \mathbf{A} + \mathbf{B}'$.

Ta có

$$dist(\mathbf{A}, \mathbf{B}) = \min_{a \in A, b \in B} |a - b| = \min_{a \in A, b \in B'} |a + b| = \min_{a \in A + B'} |c|$$

Từ đây suy ra khoảng cách giữa 2 đa giác lồi \mathbf{A}, \mathbf{B} bằng *khoảng cách từ góc tọa độ tới các cạnh của $\mathbf{A} + \mathbf{B}'$* hoặc *bằng 0 nếu \mathbf{A} và \mathbf{B} giao nhau* (trong trường hợp này $\mathbf{A} + \mathbf{B}'$ chứa góc tọa độ).

Như vậy ta phải tính khoảng cách từ *một điểm* **c** tới *đoạn thẳng* **[a,b]**.



Có hai trường hợp:

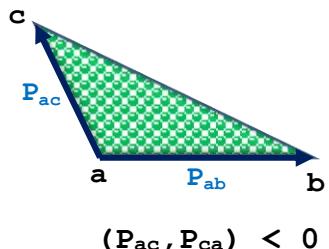
- Chân đường vuông góc hạ từ **c** xuống đường thẳng chứa **a** và **b** nằm trong đoạn **[a,b]**. Khi đó khoảng cách từ **c** tới **[a,b]** sẽ là

$$\frac{|(y_b - y_a) \times x_c - (x_b - x_a) \times y_c + x_b \times y_a - x_a \times y_b|}{\sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}}$$

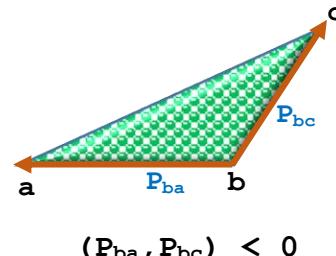
- Chân đường vuông góc hạ từ **c** xuống đường thẳng chứa **a** và **b** nằm ngoài đoạn **[a,b]**. Khi đó khoảng cách từ **c** tới **[a,b]** sẽ là *min* khoảng cách từ **c** tới **a** và khoảng cách từ **c** tới **b**.

Dấu hiệu nhận biết trường hợp 2:

Có nhiều cách để kiểm tra, một trong số đó là tính tích vô hướng



$$(x_b - x_a)(x_c - x_a) + (y_b - y_a)(y_c - y_a) < 0$$



$$(x_a - x_b)(x_c - x_b) + (y_a - y_b)(y_c - y_b) < 0$$

Lưu ý:

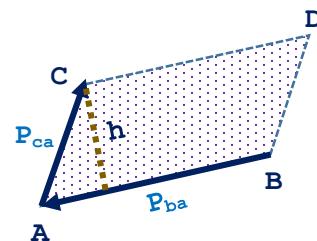
Có nhiều công thức tương đương dẫn xuất độ dài đường cao của tam giác. Tồn tại các công thức dễ nhớ và dễ lập trình. Một trong những công thức đó là biểu diễn độ dài đường cao thông qua *tích giả vô hướng* (*pseudoscalar*) 2 véc tơ cạnh.

Xét $\mathbf{P} = (\mathbf{x}_p, \mathbf{y}_p)$, $\mathbf{Q} = (\mathbf{x}_q, \mathbf{y}_q)$ – 2 véc tơ trên mặt phẳng. Ký hiệu $\mathbf{P} \wedge \mathbf{Q}$ – tích giả vô hướng của \mathbf{P} và \mathbf{Q} .

$$\mathbf{P} \wedge \mathbf{Q} = \begin{vmatrix} x_p & y_p \\ x_q & y_q \end{vmatrix} = \mathbf{x}_p \times \mathbf{y}_q - \mathbf{y}_p \times \mathbf{x}_q$$

Tích giả vô hướng $\mathbf{P}_{ba} \wedge \mathbf{P}_{ca}$ cho diện tích có hướng của hình bình hành ABCD.

Độ cao \mathbf{h} từ đỉnh C của hình bình hành ABCD sẽ là:



Khi tính toán theo công thức trên cần chú ý tính phản đối xứng của tích giả vô hướng, tức là $\mathbf{P}_{ba} \wedge \mathbf{P}_{ca} = -(\mathbf{P}_{ca} \wedge \mathbf{P}_{ba})$.

Với số lượng đa giác không nhiều ($n \leq 200$) ta có thể tính khoảng cách giữa 2 đa giác i và j ($1 \leq i, j \leq n$) với mọi i, j và bằng quy hoạch động – xác định khoảng cách ngắn nhất giữa a và b .

Tổ chức dữ liệu:

- Bản ghi **struct point** với phép xử lý gắn kèm xử lý cặp điểm: +, -, < và tính độ dài đoạn thẳng,
- Mảng 2 chiều **vector<vector<point>> polys** – lưu đỉnh các đa giác,
- Mảng 2 chiều **vector<vector<point>> ipolys** – lưu đỉnh các đa giác đối xứng qua gốc tọa độ,
- Mảng 2 chiều **vector<vector<double>> graph(n, vector<double>(n))**
 - ghi nhận khoảng cách ngắn nhất trực tiếp giữa 2 đảo,
- Mảng **vector<bool> used(n)** – đánh dấu đảo đã xét,
- Mảng **vector<double> dist(n, 1e100)** – ghi nhận khoảng cách ngắn nhất từ a .

Xử lý:

Nhập tọa độ đỉnh các đa giác và chuẩn bị dữ liệu để tính khoảng cách:

```
for (int i = 0; i < n; ++i)
{
    polys.push_back({});
    ipolys.push_back({});
    int k;
    fi >> k;
    int mem = 0;
    for (int j = 0; j < k; ++j)
    {
        int x, y;
        fi >> x >> y;
        polys.back().push_back(point(x, y));
        ipolys.back().push_back(point(-x, -y));
        if (polys[i][j] < polys[i][mem]) mem = j;
    }
    rotate(polys[i].begin(), polys[i].begin() + mem, polys[i].end());
    mem = 0;
    for (int j = 0; j < k; ++j)
        if (ipolys[i][j] < ipolys[i][mem]) mem = j;
    rotate(ipolys[i].begin(), ipolys[i].begin() + mem, ipolys[i].end());
}
```

Khởi tạo dòng mới

Tìm đỉnh trái nhất và thấp nhất

Danh số lại các đỉnh

Các phép tính gắn với điểm

```
struct point
{
    int x, y;

    point(int x, int y) : x(x), y(y) {}

    point operator+(point const& other) const
    {
        return point(x + other.x, y + other.y);
    }

    point operator-(point const& other) const
    {
        return point(x - other.x, y - other.y);
    }

    double len() const
    {
        return sqrt((ll)x * x + (ll)y * y);
    }

    bool operator<(point const& other) const
    {
        return x < other.x || (x == other.x && y < other.y);
    }
};
```

Tính diện tích hình bình hành theo tích giả vô hướng:

```
ll p_scalar(point const& p1, point const& p2)
{
    return (ll) p1.x * p2.y - (ll) p1.y * p2.x;
}
```

Tính khoảng cách từ điểm **C** tới đoạn thẳng **AB**:

```
vector<double> dist(n, 1e100);
dist[a] = 0;
vector<bool> used(n, 0);

for (int i = 0; i < n; ++i)
{
    int mem = -1;
    for (int j = 0; j < n; ++j)
        if (!used[j])
            if (mem == -1 || dist[mem] > dist[j]) mem = j;
    used[mem] = true;
    if (mem == b)
    {
        fo<<fixed<<setprecision(15)<<dist[b]<<!'\n';
        fo<<"\nTime: "<<clock()/(double)1000<<" sec";
        return 0;
    }
    for (int j = 0; j < n; ++j)
        dist[j] = min(dist[j], dist[mem] + graph[mem][j]);
}
```

Tính khoảng cách ngắn nhất từ gốc tọa độ tới **A+B'**:

```

double calc_dist(vector<point> const& poly1, vector<point> const& poly2)
{
    int c1 = 0, c2 = 0;
    point cur = poly1[0] + poly2[0];
    double ans = 1e100;
    while (c1 < poly1.size() || c2 < poly2.size())
    {
        point shift(0, 0);
        int next1 = c1 + 1;
        if (next1 == poly1.size()) next1 = 0;
        int next2 = c2 + 1;
        if (next2 == poly2.size()) next2 = 0;

        if (c1 == poly1.size() || (c2 == poly2.size() &&
            p_scalar(poly1[next1] - poly1[c1], poly2[next2] - poly2[c2]) < 0))
        {
            shift = poly2[next2] - poly2[c2];
            c2++;
        }
        else
        {
            shift = poly1[next1] - poly1[c1];
            c1++;
        }
        ans = min(ans, get_dist(cur, cur + shift, point(0, 0)));
        cur = cur + shift;
    }
    return ans;
}

```

Tính độ dài đường đi ngắn nhất từ **a** tới **b**:

```

double get_dist(point const& a, point const& b, point const& c)
{
    if (scalar(b - a, c - a) < 0 || scalar(a - b, c - b) < 0)
        return min((c - a).len(), (c - b).len());
    return abs(p_scalar(b - a, c - a) / (b - a).len());
}

```

*Chân đường cao nằm
ngoài đoạn thẳng*

Độ phức tạp của giải thuật: $O(k \times n^2)$.

Chương trình

```
#include <bits/stdc++.h>
#define ff first
#define ss second

using namespace std;
typedef long long ll;
typedef long double ld;
typedef pair<int, int> pii;
ifstream fi ("mud.inp");
//ifstream fi ("85");
ofstream fo ("mud.out");

struct point
{
    int x, y;

    point(int _x, int _y) : x(_x), y(_y) {}

    point operator+(point const& other) const
    {
        return point(x + other.x, y + other.y);
    }

    point operator-(point const& other) const
    {
        return point(x - other.x, y - other.y);
    }

    double len() const
    {
        return sqrt((ll) x * x + (ll) y * y);
    }

    bool operator<(point const& other) const
    {
        return x < other.x || (x == other.x && y < other.y);
    }
};

ll p_scalar(point const& p1, point const& p2)
{
    return (ll) p1.x * p2.y - (ll) p1.y * p2.x;
}

ll scalar(point const& p1, point const& p2)
{
    return (ll) p1.x * p2.x + (ll) p1.y * p2.y;
}

double get_dist(point const& a, point const& b, point const& c)
{
    if (scalar(b - a, c - a) < 0 || scalar(a - b, c - b) < 0)
        return min((c - a).len(), (c - b).len());
    return abs(p_scalar (b - a, c - a) / (b - a).len());
}
```

```

}

double calc_dist(vector<point> const& poly1, vector<point> const& poly2)
{
    int c1 = 0, c2 = 0;
    point cur = poly1[0] + poly2[0];
    double ans = 1e100;
    while (c1 < poly1.size() || c2 < poly2.size())
    {
        point shift(0, 0);
        int next1 = c1 + 1;
        if (next1 == poly1.size()) next1 = 0;
        int next2 = c2 + 1;
        if (next2 == poly2.size()) next2 = 0;

        if (c1==poly1.size() || (c2!=poly2.size() &&
            p_scalar(poly1[next1]-poly1[c1], poly2[next2]-poly2[c2])<0))
        {
            shift = poly2[next2] - poly2[c2];
            c2++;
        } else
        {
            shift = poly1[next1] - poly1[c1];
            c1++;
        }
        ans = min(ans, get_dist(cur, cur + shift, point(0, 0)));
        cur = cur + shift;
    }
    return ans;
}

int main()
{
    int n, a, b;
    fi >> n >> a >> b;
    --a; --b;
    vector<vector<double>> graph(n, vector<double>(n));
    vector<vector<point>> polys;
    vector<vector<point>> ipolys;
    for (int i = 0; i < n; ++i)
    {
        polys.push_back({});
        ipolys.push_back({});
        int k;
        fi >> k;
        int mem = 0;
        for (int j = 0; j < k; ++j)
        {
            int x, y;
            fi >> x >> y;
            polys.back().push_back(point(x, y));
            ipolys.back().push_back(point(-x, -y));
            if (polys[i][j] < polys[i][mem]) mem = j;
        }
        rotate(polys[i].begin(), polys[i].begin() + mem, polys[i].end());
        mem = 0;
    }
}

```

```

    for (int j = 0; j < k; ++j)
        if (ipolys[i][j] < ipolys[i][mem]) mem = j;
    rotate(ipolys[i].begin(), ipolys[i].begin() + mem, ipolys[i].end());
}

for (int i = 0; i < n; ++i)
    for (int j = i + 1; j < n; ++j)
        graph[i][j] = graph[j][i] = calc_dist(polys[i], ipolys[j]);

vector<double> dist(n, 1e100);
dist[a] = 0;
vector<bool> used(n, 0);

for (int i = 0; i < n; ++i)
{
    int mem = -1;
    for (int j = 0; j < n; ++j)
        if (!used[j])
            if (mem == -1 || dist[mem] > dist[j]) mem = j;

    used[mem] = true;
    if (mem == b) {
        fo<<fixed<<setprecision(15) << dist[b] << '\n';
        fo<<"\nTime: "<<clock() / (double)1000<<" sec";
        return 0;
    }
    for (int j = 0; j < n; ++j)
        dist[j] = min(dist[j], dist[mem] + graph[mem][j]);
}
}

```



VZ24. BẰNG NHAU

Tên chương trình: EQ.CPP

Có một vấn đề liên quan tới n điểm trên mặt phẳng, các điểm đều có tọa độ nguyên, tọa độ điểm thứ i là (x_i, y_i) , $i = 1 \dots n$.

Để giải quyết vấn đề giáo sư Braun sử dụng khoảng cách Manhattan, theo đó khoảng cách giữa 2 điểm i và j là $|x_i - x_j| + |y_i - y_j|$. Sau đó ông trình bày lại các lập luận của mình dựa trên cơ sở khoảng cách Euclid $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ để cho mọi người thấy là các số liệu tính toán có thể khác nhau khi sử dụng công thức tính khoảng cách khác nhau, nhưng những tính chất tìm thấy được ở một khái niệm khoảng cách vẫn đúng với khái niệm khoảng cách khác! Đó là một trong số các đặc điểm quan trọng của không gian Metric.

Tuy vậy với một số cặp điểm (i, j) giáo sư Braun không phải trình bày lại các dẫn xuất vì đơn giản là khoảng cách giữa chúng thính theo 2 cách đều bằng nhau.

Hãy xác định có bao nhiêu cặp số có khoảng cách giống nhau không phụ thuộc vào công thức áp dụng.

Dữ liệu: Vào từ file văn bản EQ.INP:

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 2 \times 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên x_i và y_i ($|x_i|, |y_i| \leq 10^9$).

Kết quả: Đưa ra file văn bản EQ.OUT một số nguyên – số lượng cặp tìm được.

Ví dụ:

EQ.INP
6
0 0
0 1
0 2
-1 1
0 1
1 1

EQ.OUT
11



VZ24 Io201900324_Z_A_XI

Giải thuật: Nguyên lý bù trừ, Kỹ thuật 2 con trỏ .

Lưu tọa độ các điểm đã cho vào mảng **a**, mỗi phần tử của mảng có kiểu **pair<int,int>**,

Khoảng cách giữa 2 điểm **i** và **j** bằng nhau theo 2 cách tính khi và chỉ khi **a[i].first==a[j].first** hoặc **a[i].second==a[j].second**.

Sắp xếp mảng **a** theo thứ tự tăng dần,

Các điểm có tọa độ đầu tiên bằng nhau sẽ tạo thành nhóm các phần tử liên tiếp,

Gọi **p** là chỉ số của phần tử đầu tiên trong nhóm,

Gọi **q** là chỉ số của phần tử cuối cùng trong nhóm,

Số cặp điểm trong nhóm cho khoảng cách như nhau theo 2 cách tính là

$$(q-p) \times (q-p+1) / 2$$

Việc tìm nhóm tiếp theo được thực hiện bắt đầu từ phần tử **q+1** trong dãy **a**.

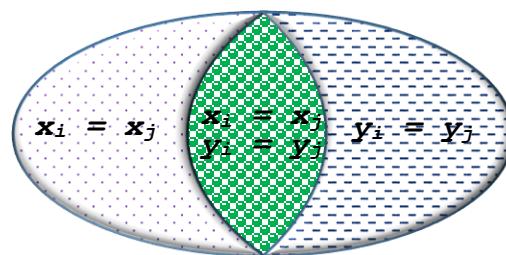
Đổi chỗ tọa độ **x** và **y** trong mỗi điểm, theo sơ đồ trên dễ dàng tính được số cặp điểm cần tìm theo tọa độ **y**.

Số cặp điểm có đồng thời tọa độ **x** bằng nhau và tọa độ **y** bằng nhau được tính 2 lần, vì vậy cần tìm các nhóm điểm này, tính số cặp tạo thành và giảm tương ứng tổng số lượng cặp đã tích lũy khi tính riêng theo các tọa độ **x** và **y**.

Tổ chức dữ liệu

Mảng **vector<pii> a** – lưu tọa độ các điểm đã cho.

Xử lý



Nhập dữ liệu và chuẩn bị:

```
fi>>n;
a.resize(n);
for(int i=0; i<n; ++i)
{
    fi>>x>>y;
    a[i] = {x, y};
}
x=(int)1e9+1;
a.push_back({x, x+1});
sort(a.begin(), a.end());
```

Phản tử hàng rào

Tìm điểm đầu đoạn các tọa độ thứ I bằng nhau:

```
int get_p()
{
    for(int i=q; i<n; ++i)
        if(a[i].ff==a[i+1].ff){p=i; return p;}
    p=n+1; return p;
}
```

Điểm xuất phát

Dấu hiệu điểm đầu

Tìm điểm cuối đoạn các tọa độ thứ I bằng nhau:

```
int get_q()
{
    for(int i=p; i<n; ++i)
        if(a[i].ff!=a[i+1].ff){q=i; return q;}
    q=n+1; return q;
}
```

Điểm xuất phát

Dấu hiệu điểm cuối

Tìm điểm đầu đoạn tọa độ trùng nhau:

```
int get_ep()
{
    for(int i=q; i<n; ++i)
        if(a[i]==a[i+1]){p=i; return p;}
    p=n+1; return p;
}
```

Dấu hiệu điểm đầu

Tính số lượng các cặp điểm có tọa độ thứ I bằng nhau:

```
int64_t calc()
{
    int64_t t, r=0;
    x=(int)1e9+1; p=0; q=0;
    while(p<n)
    {
        get_p();
        if(p<n)
        {
            get_q();
            t = (int64_t)q-p;
            r+= t*(t+1)/2;
        }
    }
    return r;
}
```

Tổ hợp chap 2

Độ phức tạp của giải thuật: $O(n \log n)$.

Chương trình

```
#include <bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
typedef pair<int,int> pii;
ifstream fi ("eq.inp");
ofstream fo ("eq.out");
int n,x,y,p,q;
vector<pii> a;

int64_t ans;

int get_q()
{
    for(int i=p; i<n; ++i)
        if(a[i].ff!=a[i+1].ff) {q=i; return q;}
}

int get_p()
{
    for(int i=q; i<n; ++i)
        if(a[i].ff==a[i+1].ff) {p=i; return p;}
    p=n+1; return p;
}

int64_t calc()
{
    int64_t t,r=0;
    x=(int)1e9+1; p=0; q=0;
    while(p<n)
    {
        get_p();
        if(p<n)
        {
            get_q();
            t = (int64_t)q-p;
            r+= t*(t+1)/2;
        }
    }
    return r;
}

int get_ep()
{
    for(int i=q; i<n; ++i)
        if(a[i]==a[i+1]) {p=i; return p;}
    p=n+1; return p;
}

int get_eq()
{
    for(int i=p; i<n; ++i)
        if(a[i]!=a[i+1]) {q=i; return q;}
}
```

```

int64_t calc_e()
{
    int64_t t, r=0;
    x=(int)1e9+1; p=0; q=0;
    while(p<n)
    {
        get_ep();
        if(p<n)
        {
            get_eq();
            t = (int64_t)q-p;
            r+= t*(t+1)/2;
        }
    }
    return r;
}

int main()
{
    fi>>n;
    a.resize(n);
    for(int i=0; i<n; ++i)
    {
        fi>>x>>y;
        a[i] = {x,y};
    }
    x=(int)1e9+1;
    a.push_back({x,x+1});
    sort(a.begin(),a.end());
    ans=calc();
    for(auto &i:a) swap(i.ff,i.ss);
    sort(a.begin(),a.end());
    p=0; q=0;
    ans+=calc();
    p=0; q=0;
    ans -= calc_e();
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```



VZ25. ĐƠN VỊ

Tên chương trình: UNIT.CPP

Toàn bộ khu đất quy hoạch là khu công nghiệp có diện tích n . Người ta chia nó thành các phần bằng nhau, mỗi phần là một hình vuông để là đơn vị đấu thầu cho các công ty muốn thuê đất. Bạn quản lý Khu công nghiệp muốn đơn vị đấu thầu phải càng lớn càng tốt.

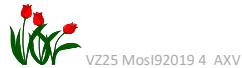
Hãy xác định diện tích lớn nhất có thể sử dụng làm đơn vị đấu thầu.

Dữ liệu: Vào từ file văn bản UNIT.INP gồm một dòng chứa số nguyên n ($1 \leq n \leq 2 \times 10^9$).

Kết quả: Đưa ra file văn bản UNIT.OUT một số nguyên – giá trị lớn nhất của đơn vị đấu thầu.

Ví dụ:

UNIT.INP	UNIT.OUT
180	36



Giải thuật: Phân tích ra thừa số nguyên tố.

Gọi **s** – diện tích lớn nhất có thể sử dụng làm đơn vị đầu thầu.

Các tính chất của **s**:

- **n** chia hết cho **s**,
- Nếu phân tích **s** ra thừa số nguyên tố, các thừa số chỉ chứa số mũ chẵn.

Từ đây suy ra $t = n/s$ chỉ chứa các thừa số nguyên tố với số mũ là 1.

Sơ đồ tính **t**:

- Phân tích **n** ra thừa số nguyên tố và giữ lại các thừa số với số mũ lẻ,
- **t** là tích các thừa số được giữ lại, số thừa số được giữ lại sẽ không quá 10.

Kết quả cần tìm sẽ là n/t .

Độ phức tạp của giải thuật: $O(n^{0.5})$.

Tổ chức dữ liệu: Mảng **vector<int>** **pr** – lưu các thừa số nguyên tố của **t**.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("unit.inp");
ofstream fo ("unit.out");
int n,k,p,ip,ans=1;

int main()
{
    fi>>n; k=n;
    vector<int> pr;
    p=2;
    while (p*p<=n)
    {
        if (n%p==0)
        {
            ip=0;
            while (n%p==0) ++ip, n/=p;
            ip&=1;
            if (ip) pr.push_back(p);
        }
        ++p;
    }
    if (n>1) pr.push_back(n);

    for (int i:pr) ans*=i;
    ans=k/ans;
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Ngày nay hiếm có sinh viên nào lại không có máy tính cá nhân, việc soạn thảo văn bản thì ngay đến học sinh PTCS cũng đã thành thạo. Vì vậy một số nơi trước khi tới phòng vấn xin vào làm người ta yêu cầu gửi trước đơn xin việc viết tay. “*Nét chữ - Nét người*”, nhìn vào trang giấy viết tay người ta cũng có thể đoán được nhiều tính cách của người xin việc, thậm chí, có một số trường hợp, biết được tình trạng sức khỏe!

Alice phải gửi một đơn xin việc viết tay. Theo quy định, đơn viết trên giấy hình chữ nhật kích thước $n \times m$. Alice chỉ có trong tay phong bì hình chữ nhật kích thước $h \times w$. Nếu tờ giấy không cho vừa vào phong bì Alice gấp đôi lá đơn theo chiều dọc hoặc chiều ngang một hoặc vài lần. Đơn được bỏ vào phong bì với các cạnh song song với các cạnh của phong bì. Đơn bỏ được vào phong bì nếu các cạnh không lớn hơn cạnh tương ứng của phong bì. Đơn có thể được xoay 90° trước khi bỏ vào phong bì.

Là người cẩn thận, Alice không muốn lá đơn bị nhau quá mức và vì vậy cố gắng thực hiện việc gấp một cách ít nhất có thể.

Hãy xác định số lần gấp ít nhất cần thực hiện.

Dữ liệu: Vào từ file văn bản ENVELOPE.INP gồm một dòng chứa 4 số nguyên n, m, h, w ($1 \leq n, m, h, w \leq 10^{18}$).

Kết quả: Đưa ra file văn bản ENVELOPE.OUT một số nguyên – số lần gấp ít nhất cần thực hiện.

Ví dụ:

ENVELOPE.INP	ENVELOPE.OUT
3 3 2 2	2



Giải thuật: Cơ sở lập trình.

Việc gấp đôi tờ giấy sẽ dẫn đến kích thước phải biểu diễn dưới dạng số thực và cần chống tích lũy sai số làm tròn.

Thay vì giảm kích thước tờ giấy – phóng to gấp đôi kích thước phong bì!

Kiểu dữ liệu `int64_t` vẫn lưu được giá trị 2×10^{18} – vượt quá kích thước tối đa của tờ giấy, vì vậy sẽ không có hiện tượng tràn ô (overflow).

Xét 2 trường hợp:

- Không xoay tờ giấy,
- Xoay tờ giấy 90° .

Kết quả: số phép biến đổi nhỏ nhất trong 2 trường hợp nói trên.

Độ phức tạp của giải thuật: bậc $O(\log n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "envelope."
using namespace std;

int64_t solve(int64_t n, int64_t m, int64_t h, int64_t w)
{
    int ans = 0;
    while (h < n)
    {
        ans++;
        h *= 2;
    }
    while (w < m)
    {
        ans++;
        w *= 2;
    }
    return ans;
}

int main()
{
    freopen(NAME"inp", "r", stdin);
    //freopen("27", "r", stdin);
    freopen(NAME"out", "w", stdout);
    int64_t n,m,h,w;
    cin >> n >> m >> h >> w;
    cout << min(solve(n, m, h, w), solve(m, n, h, w)) << "\n";
}
```



VZ01. NGHỆ THUẬT ĐƯỜNG PHỐ

Tên chương trình: ART.CPP

Tồn tại trào lưu nghệ thuật mang hội họa xuống đường phố. Người ta vẽ tranh lên phía ngoài các tường nhà để chúng không còn là các bê tông đơn điệu, buồn tẻ.

Một khu nhà được vây quanh bằng các bức tường song song với trục tọa độ, tạo thành đường gấp khúc khép kín cạnh không tự cắt n đỉnh có tọa độ nguyên, đỉnh thứ i có tọa độ (x_i, y_i) , $i = 1 \div n$. Phía ngoài bức tường rào được trang trí bằng các hình vẽ theo những trường phái nghệ thuật khác nhau và thu hút sự hiếu kỳ của những ai đi ngang qua.

Nhưng khu nhà lại nằm giữa 4 đường cao tốc không cho phép dừng xe: hai đường theo hướng bắc – nam ở bên phải và bên trái, hai đường theo hướng đông – tây ở trên và ở dưới. Như vậy người ta chỉ nhìn thấy các phần của bức tường khi quan sát vuông góc theo hướng từ bắc xuống, từ nam lên, từ đông hoặc tây sang.

Không ít người đã bỏ nhiều thời gian đi vòng quanh khu nhà để ngắm nhìn các tác phẩm hội họa và ai cũng tiếc là không được chiêm ngưỡng toàn bộ tác phẩm trên tường.

Hãy xác định tổng độ dài phần tranh bị khuất.

Dữ liệu: Vào từ file văn bản ART.INP:

- + Dòng đầu tiên chứa số nguyên n ($4 \leq n \leq 1000$),
- + Dòng thứ i trong n dòng sau chứa 2 số nguyên x_i và y_i ($-10^6 \leq x_i, y_i \leq 10^6$). Các đỉnh được liệt kê theo một chiều nào đó.

Kết quả: Đưa ra file văn bản ART.OUT một số nguyên – tổng độ dài phần tranh bị khuất.

Ví dụ:

ART.INP
10
1 1
6 1
6 4
3 4
3 3
5 3
5 2
2 2
2 3
1 3

ART.OUT
6

B12. BỨC TRANH

Tên chương trình: PICTURE.CPP

Huấn luyện viên bóng đá thường không ở lâu một nơi vì hợp đồng có thể không được gia hạn hay thậm chí – bị sa thải sớm, vì vậy họ thường không mua nhà mà chỉ thuê để ở.

Tường căn phòng mà huấn luyện viên đội bóng thành phố thuê được lát bằng gạch men nghệ thuật (gạch có in hình), viên gạch có hình vuông cạnh m cm, trong rất đẹp nhưng khá đon điệu. Ông mua một bức tranh hình chữ nhật về treo lên tường để phủ được các nhiều viên gạch càng tốt. Các cạnh của bức tranh phải song song với các cạnh của viên gạch. Bức tranh có kích thước $w \times h$. Một viên gạch gọi là được phủ nếu nó có diện tích chung khác 0 với bức tranh.



Hãy xác định với cách treo hợp lý số viên gạch nhiều nhất được phủ là bao nhiêu.

Dữ liệu: Vào từ file văn bản PICTURE.INP gồm một dòng chứa 3 số nguyên m , w và h ($1 \leq m, w, h \leq 2 \times 10^9$).

Kết quả: Đưa ra file văn bản PICTURE.OUT một số nguyên – số viên gạch nhiều nhất được phủ.

Ví dụ:

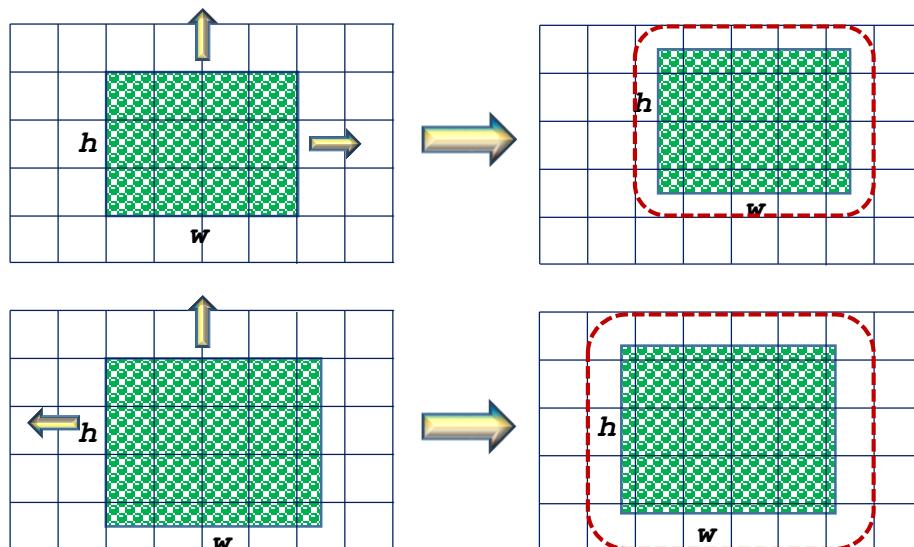
PICTURE.INP	PICTURE.OUT
10 30 20	12



Giải thuật: Cơ sở lập trình, Khả năng phân tích giải thuật.

Khi treo bức tranh để đinh trên trái của nó trùng với một đinh của viên gạch, nếu w chia hết cho m thì ở mỗi hàng, số viên gạch bị phủ là w/m . *Tịnh tiến bức tranh sang phải một chút* để đinh trên trái của nó không trùng với đinh của viên gạch, ở mỗi hàng số viên gạch bị phủ sẽ là $w/m+1$.

Trường hợp w không chia hết cho m , khi treo bức tranh để đinh trên trái trùng với đinh một viên gạch thì ở mỗi hàng, số viên gạch bị phủ là $w/m+1$. *Tịnh tiến bức tranh sang trái một chút* để đinh trên trái của nó không trùng với đinh của viên gạch, ở mỗi hàng số viên gạch bị phủ sẽ là $w/m+2 = w/m+1+w\%m$.



Lập luận tương tự như vậy với h để tính số lượng hàng bị phủ, ta có số lượng hàng bị phủ là $h/m+1+h\%m$.

Kết quả cần tìm là tích 2 đại lượng đã tính được nói trên.

Chú ý: Khai báo kiểu dữ liệu thích hợp.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("picture.inp");
ofstream fo ("picture.out");
int64_t m,w,h,ans, t1,t2;

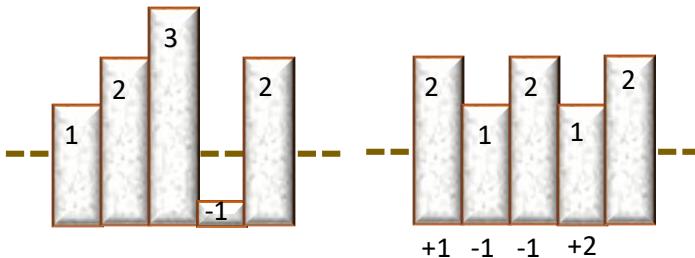
int main()
{
    fi>>m>>w>>h;
    t1=w/m+1+(w%m>0);
    t2=h/m+1+(h%m>0);
    ans=t1*t2;
    fo<<ans;
}
```



VX14. ĐÁNH LUỐNG

Tên chương trình: DRILL.CPP

Mảnh vườn của Viện Nghiên cứu Giống và Cây trồng được đánh thành n luồng, luồng thứ i có độ cao a_i so với mốc tính, a_i có thể âm, $i = 1 \div n$, phù hợp cây trồng trên luồng. Nhiệm vụ sắp tới của Viện là cung cấp cây giống cho 2 loại cây có tác dụng hỗ trợ nhau khi trồng xen. Vì vậy người ta phải cải tạo lại cách đánh luồng để các luồng ở vị trí chẵn có cùng độ cao, các luồng ở vị trí lẻ có cùng độ cao và chênh lệch độ cao giữ 2 luồng liên tiếp là k .



Máy đánh luồng chạy dọc theo luồng và mỗi lần chạy có thể bóc đất bè mặt, giảm độ cao luồng 1 đơn vị hoặc đắp thêm đất để độ cao luồng tăng thêm 1.

Hãy xác định số lần vận hành máy ít nhất để có mảnh vườn với các luồng có độ cao thỏa mãn yêu cầu mới.

Dữ liệu: Vào từ file văn bản DRILL.INP:

- ⊕ Dòng đầu tiên chứa 2 số nguyên n và k ($1 \leq n \leq 10^5$, $0 \leq k \leq 10^9$),
- ⊕ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($|a_i| \leq 10^9$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản DRILL.OUT một số nguyên – số lần vận hành máy ít nhất.

Ví dụ:

DRILL.INP
5 1
1 2 3 -1 2

DRILL.OUT
5

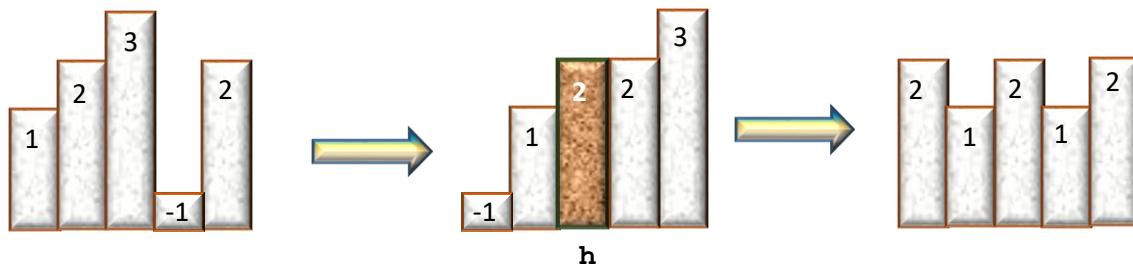


VX14 Io20180325 B AXIII

Giải thuật: Kỹ thuật lập trình.

Xét trường hợp $k = 0$ (Trường hợp san bằng).

Gọi độ cao của luồng sau khi san bằng là \mathbf{h} . Số lần san sẽ là $\sum_{i=1}^n |a_i - h|$. Tổng này nhận giá trị nhỏ nhất khi \mathbf{h} bằng giá trị a_i nằm ở điểm giữa trong dãy $\{a_i\}$ đã sắp xếp tăng dần.



Xét trường hợp $k > 0$.

Có 2 khả năng:

- ✚ Luồng 1 là luồng cao,
- ✚ Luồng 1 là luồng thấp.

Xét trường hợp *luồng 1 là luồng cao*: Nếu ta lấy độ cao tất cả các luồng ở *vị trí lẻ* trừ đi \mathbf{k} thì bài toán cần giải tương đương với bài toán san bằng với các giá trị a_i mới!

Xét trường hợp *luồng 1 là luồng thấp*: Nếu ta lấy độ cao tất cả các luồng ở *vị trí chẵn* trừ đi \mathbf{k} thì bài toán cần giải tương đương với bài toán san bằng với các giá trị a_i mới! Có thể thay vì giảm độ cao các vị trí chẵn ta có thể *tăng k ở độ cao các luồng vị trí lẻ* (giữ nguyên độ cao các luồng ở vị trí chẵn), kết quả cần tìm không thay đổi!

Tổ chức dữ liệu:

- ─ Mảng `vector<int>` $\mathbf{a}(n)$ – lưu các độ cao a_i ,
- ─ Các mảng `vector<int>` $\mathbf{tmp}, \mathbf{tmp2}$ – lưu a_i mới phục vụ tìm kiếm \mathbf{h} .

Xử lý:

Không cần sắp xếp tường minh dãy các giá trị a_i mới vì ta chỉ cần tìm phần tử đứng giữa trong dãy đã sắp xếp.

Độ phức tạp của giải thuật: $O(nlnn)$.

Chương trình

```
#include "bits/stdc++.h"
#define NAME "drill."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef int64_t ll;
const ll INFL = 1e18 + 123;

int main()
{
    int n, k;
    fi >> n >> k;
    vector<int> a(n);
    for (int i = 0; i < n; ++i) fi >> a[i];

    ll best = INFL;
    for (int i = -1; i <= 1; i += 2)
    {
        vector<int> tmp = a;
        for (int j = 0; j < n; j += 2)
            tmp[j] += k * i;
        auto tmp2 = tmp;

        ll sum = 0;
        nth_element(tmp.begin(), tmp.begin() + tmp.size() / 2, tmp.end());
        int val = tmp[tmp.size() / 2];
        for (int num : tmp) sum += abs((ll) num - val);
        if (sum < best)
            best = sum;
    }

    fo << best << "\n";
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VX15. QUE NHỰA

Tên chương trình: STICKS.CPP

Jimmy tìm thấy trong đống đồ chơi của mình 4 que nhựa độ dài nguyên dương là a , b , c và d . Trên mỗi que nhựa có các khắc đánh dấu từng đoạn độ dài đơn vị, vì vậy có thể dễ dàng bẻ mỗi que thành nhiều que nhỏ độ dài nguyên.

Jimmy muốn có 4 que để làm khung ảnh hình chữ nhật, mỗi cạnh là một que và rõ nhiên, hình chữ nhật phải có diện tích lớn nhất có thể.

Hãy xác định độ dài 2 cạnh (chiều dài và chiều rộng) của hình chữ nhật có diện tích lớn nhất.

Dữ liệu: Vào từ file văn bản STICKS.INP gồm một dòng chứa 4 số nguyên a , b , c và d ($1 \leq a, b, c, d \leq 10^{15}$).

Kết quả: Đưa ra file văn bản STICKS.OUT trên một dòng 2 số nguyên xác định kích thước của hình chữ nhật diện tích lớn nhất. Nếu có nhiều đáp án thì đưa lời giải tùy chọn.

Ví dụ:

STICKS.INP
1 8 6 19

STICKS.OUT
9 6



Giải thuật: Kỹ thuật bảng phương án.

Chuẩn hóa dữ liệu: đưa về câu hình $\mathbf{a} \geq \mathbf{b} \geq \mathbf{c} \geq \mathbf{d}$.

Xét các tình huống tạo hình chữ nhật với diện tích là lớn nhất:

- ⊕ Mỗi cạnh hình chữ nhật từ một que,
- ⊕ Hai cạnh hình chữ nhật từ một que, hai cạnh kia – mỗi cạnh từ một que trong số còn lại,
- ⊕ Ba cạnh hình chữ nhật từ một que, cạnh thứ tư – từ một trong số các que còn lại,
- ⊕ Bốn cạnh hình chữ nhật – từ một que.

Que ngắn nhất sẽ không bị bẻ.

Sau một số lần bẻ, ta có các que với độ dài $\mathbf{s}\mathbf{t}_0, \mathbf{s}\mathbf{t}_1, \dots, \mathbf{s}\mathbf{t}_{k-1}$,

Sắp xếp $\mathbf{s}\mathbf{t}_i$ giảm dần, diện tích hình chữ nhật lớn nhất tạo được với bộ que này là $\mathbf{s}\mathbf{t}_1 \times \mathbf{s}\mathbf{t}_3$ (giá trị lớn thứ nhì và thứ tư),

Dễ dàng thấy rằng $k \leq 7$ (trường hợp bẻ que độ dài \mathbf{a} thành 4 phần). Trên thực tế, để tìm số lớn thứ nhì và thứ tư ta không cần giữ hết các giá trị giống nhau, vì vậy $k < 7$.

Số lượng trường hợp cần xét là không lớn, vì vậy mỗi lần bẻ que (hoặc các que), để tìm giá trị lớn thứ nhì và thứ tư đơn giản nhất là sắp xếp lại bộ giá trị nhận được.

Duyệt tất cả các trường hợp bẻ cần xét, so sánh diện tích nhận được và lưu kết quả tối ưu.

Cần thành lập bảng ghi nhận cách bẻ cần xét để tránh bỏ sót hoặc trùng lặp.

Lưu ý các trường hợp $\mathbf{a} = \mathbf{b}$ hoặc $\mathbf{a} = \mathbf{b} = \mathbf{c}$. Bảng ghi nhận các khả năng cần bẻ phải bao quát các trường hợp này nhưng **không cần kiểm tra** nhận dạng. Việc kiểm tra sẽ làm **độ phức tạp lập trình tăng** còn **độ phức tạp giải thuật giảm không đáng kể**.

Diện tích hình chữ nhật có thể rất lớn (bậc 10^{30}) vì vậy cần tổ chức thực hiện phép **nhân số lớn**.

Khi bẻ một que thành nhiều đoạn, các đoạn con phải bằng nhau hoặc gần bằng nhau (trường hợp độ dài không chia hết cho số cách bẻ), ví dụ, với $\mathbf{a} = 11$, khi bẻ thành 4 đoạn ta có các độ dài 2, 3, 3 và 3.

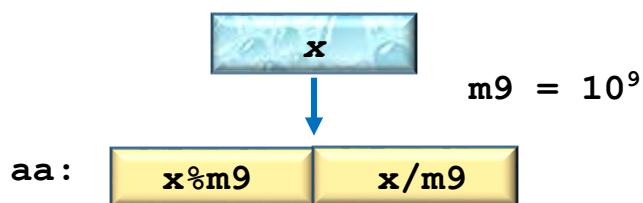
Các trường hợp cần xét:

$i \setminus st_j$	0	1	2	3	4	5
1	d	c	b	a		
2	d	c	$b/2$	$a/2$	$a-a/2$	$b-b/2$
3	d	$c/2$	b	$a/2$	$a-a/2$	$c-c/2$
4	d	c	b	$a-b$	b	
5	d	c	b	$a-c$	c	
6	d	c	b	$a/4$	$(a-2 \times (a/4))/2$	$a-st_3-st_4$
7	d	c	b	$(a-b)/2$	$a-st_3$	
8	d	c	b	$b-1$	$(a-b+1)/2$	$a-st_3-st_4$
9	d	c	b	b	$(a-b)/2$	$a-st_3-st_4$
10	d	c	$b/2$	a	$b-b/2$	
11	d	$c/2$	b	a	$c-c/2$	
12	d	c	b	$a/2$	$(a-a/2)/2$	$a-st_3-st_4$
13	d	c	b	$a/2+1$	$(a-a/2-1)/2$	$a-st_3-st_4$
14	d	c	b	$a/2$	$a-st_3$	

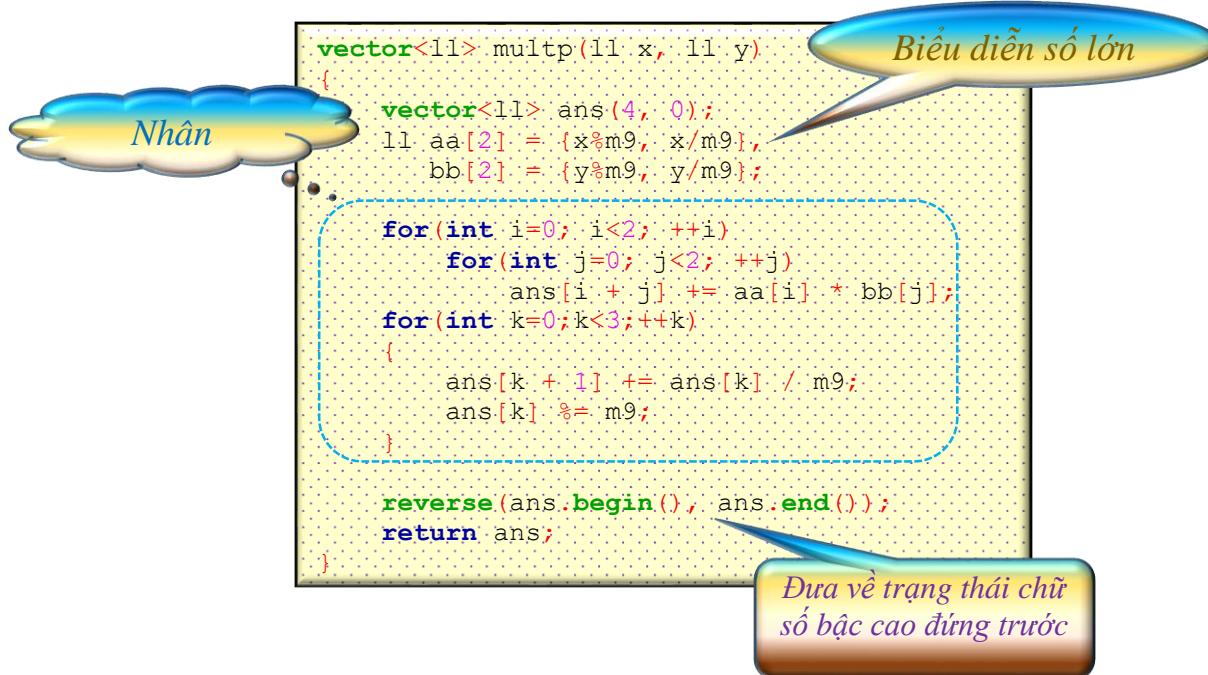
Tổ chức dữ liệu:

- Mảng `int64_t st[8]` – lưu độ dài các đoạn cần xét,
- Mảng `vector<int64_t> area(4, OLL)` – lưu diện tích lớn nhất của hình chữ nhật có thể tạo.

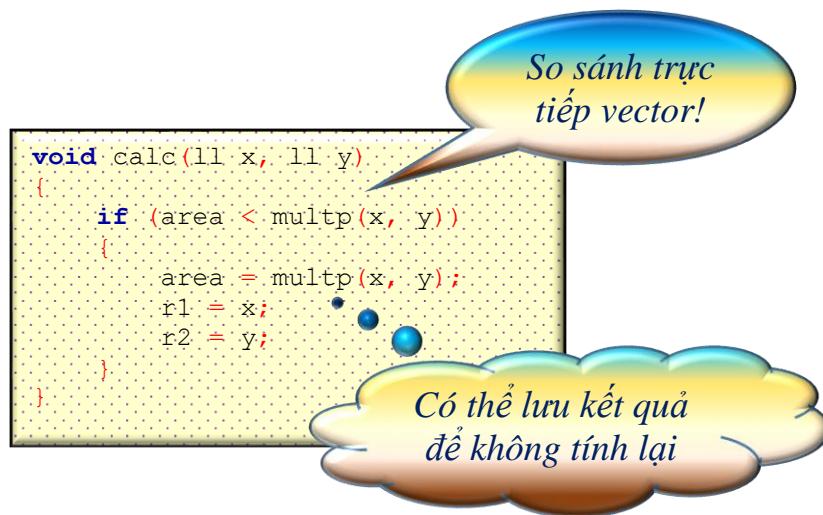
Xử lý:



Nhân số lớn:



Chọn kết quả tối ưu:



Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "sticks."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef int64_t ll;
const ll m9=(ll)1e9;
ll r1,r2,st[8],a,b,c,d,e;
vector<ll> area(4,0LL);

vector<ll> multp(ll x, ll y)
{
    vector<ll> ans(4, 0);
    ll aa[2] = {x%m9, x/m9},
        bb[2] = {y%m9, y/m9};
    for(int i=0; i<2; ++i)
        for(int j=0; j<2; ++j)
            ans[i + j] += aa[i] * bb[j];
    for(int k=0; k<3; ++k)
    {
        ans[k + 1] += ans[k] / m9;
        ans[k] %= m9;
    }
    reverse(ans.begin(), ans.end());
    return ans;
}

void calc(ll x, ll y)
{
    if (area < multp(x, y))
    {
        area = multp(x, y);
        r1 = x;
        r2 = y;
    }
}

void init()
{
    st[0]=d, st[1]=c; st[2]=b; st[3]=a;
}

int main()
{
    for(int i=0;i<4;++i) fi>>st[i];
    sort(st,st+4); a=st[3]; b=st[2]; c=st[1]; d=st[0]; calc(b, d);

    init(); st[3]/=2; st[4]=a-st[3]; st[1]/=2; st[5]=c-st[1];
    sort(st,st+6,greater<ll>()); calc(st[1],st[3]);
    init(); st[3]/=2; st[4]=a-st[3]; st[2]/=2; st[5]=b-st[2];
    sort(st,st+6,greater<ll>()); calc(st[1],st[3]);
    init(); st[3]=a-b; st[4]=b;
    sort(st,st+5,greater<ll>()); calc(st[1],st[3]);
    init(); st[3]=a-c; st[4]=c;
```

```

        sort(st,st+5,greater<ll>()); calc(st[1],st[3]);
init(); st[3]=a/4; st[4]=(a - 2*(a/4))/2; st[5]=a-st[3]-st[4];
        sort(st,st+6,greater<ll>()); calc(st[1],st[3]);
init(); st[3]=(a-b)/2; st[4]=a-st[3];
        sort(st,st+5,greater<ll>()); calc(st[1],st[3]);
init(); st[3]=b-1; st[4]=(a-b+1)/2; st[5]=a-st[3]-st[4];
        sort(st,st+6,greater<ll>()); calc(st[1],st[3]);
init(); st[3]=b; st[4]=(a-b)/2; st[5]=a-st[3]-st[4];
        sort(st,st+6,greater<ll>()); calc(st[1],st[3]);
init(); st[2]/=2; st[4]=b-st[2];
        sort(st,st+5,greater<ll>()); calc(st[1],st[3]);
init(); st[1]/=2; st[4]=c-st[1];
        sort(st,st+5,greater<ll>()); calc(st[1],st[3]);
init(); st[3]/=2; st[4]=(a-a/2)/2; st[5]= a-st[3]-st[4];
        sort(st,st+6,greater<ll>()); calc(st[1],st[3]);
init(); st[3]=a/2+1; st[4]=(a-a/2-1)/2; st[5]= a-st[3]-st[4];
        sort(st,st+6,greater<ll>()); calc(st[1],st[3]);
init(); st[3]=a/2; st[4]=a-st[3];
        sort(st,st+5,greater<ll>()); calc(st[1],st[3]);

fo << r1 << ' ' << r2 ;
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```



WB29. VẮC XIN

Tên chương trình: VACCINE.CPP

Để chuẩn bị sản xuất đại trà vắc xin chống một loại đại dịch đang hoành hành trên thế giới người ta cần có một trại nuôi ngựa lấy huyết thanh và một phòng thí nghiệm sản xuất vắc xin. Hai cơ sở này phải tách rời, nhưng ở càng gần nhau càng tốt.

Có n địa điểm có thể có thể đặt các cơ sở đó. Địa điểm thứ i được xác định bởi điểm có tọa độ (x_i, y_i) , $i = 1 \dots n$. Khoảng cách d giữa 2 điểm i và j được tính theo công thức $d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ (Khoảng cách Euclidean).

Hãy xác định khoảng cách ngắn nhất giữa 2 điểm trong số các điểm đã cho và chỉ ra một cặp điểm có khoảng cách ngắn nhất.

Dữ liệu: Vào từ file văn bản VACCINE.INP:

- ✚ Dòng đầu tiên chứa một số nguyên n ($2 \leq n \leq 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên x_i và y_i ($|x_i|, |y_i| \leq 10^9$).

Kết quả: Đưa ra file văn bản VACCINE.OUT, dòng đầu tiên là bình phương khoảng cách ngắn nhất tìm được dòng thứ 2 chứa 2 số nguyên là số thứ tự của một cặp điểm có khoảng cách nhỏ nhất.

Ví dụ:

VACCINE.INP
3
1 1
3 1
1 2

VACCINE .OUT
4
1 3



WB29 Azr2020 E A XVII

Giải thuật: Chia để trị.

Sơ đồ xử lý là chia tập các điểm thành 2 phần, giải bài toán tương tự ở mỗi phần, xác định một dải đệm kết nối 2 phần, giải bài toán tương tự ở lớp đệm và tổng hợp kết quả từ các kết quả nhận được ở ba phần đã nêu.

Để phục vụ xác định điểm có thể chọn, mỗi điểm tương ứng với cụm 3 đại lượng nguyên ($\mathbf{x}_i, \mathbf{y}_i, i$).

Việc phân chia tập ban đầu thành 2 phần không quá khó khăn:

- ✚ Sắp xếp các điểm đã cho theo trình tự tăng dần của \mathbf{x} ,
- ✚ Lấy các điểm ở nửa đầu của dãy cho vào tập **A1**, phần còn lại – vào tập **A2**.

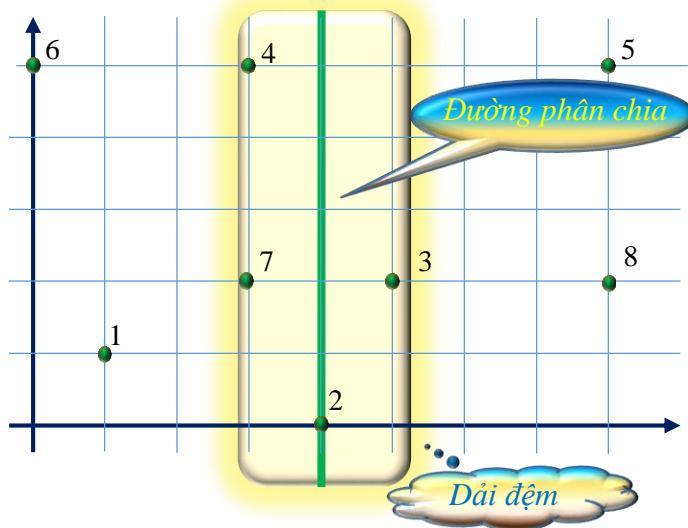
Giả thiết ta đã giải được bài toán ở các tập **A1** và **A2** với khoảng cách nhỏ nhất tìm được tương ứng là **d1** và **d2**.

Gọi $\mathbf{d} = \min\{\mathbf{d1}, \mathbf{d2}\}$.

Rõ ràng tập các điểm **B** ở dải đệm chỉ chứa các điểm thỏa mãn điều kiện khoảng cách tới đường phân chia nhỏ hơn **d**

$$|\mathbf{x}_v - \mathbf{x}_B| < \mathbf{d},$$

trong đó \mathbf{x}_B – tọa độ xác định đường phân chia.



Tập **C** các điểm cần xét trong dải đệm còn hẹp hơn, với mỗi điểm \mathbf{p}_i trong **B** chỉ cần xét các điểm có chênh lệch theo \mathbf{y} tới \mathbf{p}_i không quá **d**. Gọi tập điểm cần xét với điểm

\mathbf{p}_i là $\mathbf{C}(\mathbf{p}_i)$. $\mathbf{C}(\mathbf{p}_i)$ dễ dàng xác định được nếu các điểm trong \mathbf{B} được sắp xếp theo giá trị tọa độ y .

Xử lý để quy:

Hàm **rec(int l, int r)** tìm khoảng cách nhỏ nhất trong số các điểm từ **l** đến **r** của dãy đã sắp xếp:

- ⊕ Nếu **r-l** là đủ bé – tiến hành vét cạn các cặp điểm, sắp xếp dữ liệu trong khoảng theo y,
- ⊕ Trong trường hợp ngược lại:
 - Chia đôi khoảng và lần lượt gọi rec với các khoảng nhận được,
 - Hợp nhất khoảng và sắp xếp theo y,
 - Xử lý dãy đệm.

Lưu ý: dùng mảng dữ liệu trung gian kiểu static để tránh việc phải phân nhiều lần, điều có thể dẫn đến hiện tượng thiếu bộ nhớ làm việc.

Tổ chức dữ liệu:

Mảng **vector<point>** a – Lưu tọa độ các điểm từ dữ liệu vào,

Mảng **static t[MAXN]** – Phục vụ lưu dữ liệu hợp nhất các vùng đã khảo sát.

Độ phức tạp của giải thuật: $O(n \log n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "Vaccine."
using namespace std;
typedef pair<double, double> pdd;
typedef tuple<int, int, int>tiii;
typedef int64_t ll;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int MAXN=100001;
int n;
vector<tiii> a;

inline bool cmp_y (const tiii & a, const tiii & b) { return
get<1>(a) < get<1>(b); }
ll mindist;
int ansa, ansb;
inline void upd_ans (const tiii & a, const tiii & b)
{ll dist = 111*(get<0>(a)-get<0>(b))* (get<0>(a)-get<0>(b))
+ 111*(get<1>(a)-get<1>(b))* (get<1>(a)-get<1>(b));
if(dist<mindist)
{mindist=dist;ansa= get<2>(a);ansb=get<2>(b);}
}

void rec (int l, int r)
{if (r - l <= 3)
{
    for (int i=l; i<=r; ++i)
    for (int j=i+1; j<=r; ++j)
    upd_ans (a[i], a[j]);
    sort (a.begin()+l, a.begin()+r+1, &cmp_y);
    return;
}
int m = (l + r) >> 1;
int midx = get<0>(a[m]);
rec (l, m), rec (m+1, r);
static tiii t[MAXN];
merge (a.begin()+l, a.begin()+m+1,
       a.begin()+m+1, a.begin()+r+1, t);

copy (t, t+r-l+1, a.begin());
int tsz = 0;
for (int i=l; i<=r; ++i)
if (abs (get<0>(a[i]) - midx) < mindist)
{
    for (int j=tsz-1;
```

```

        j>=0 && get<1>(a[i]) - get<1>(t[j]) < mindist; --j)
                                upd_ans (a[i], t[j]);
    t[tsz++] = a[i];
}
}

int main()
{
    fi>>n;
    int x,y,k;
    for(int i=0;i<n;++i)
    {
        fi>>x>>y;
        a.push_back(make_tuple(x,y,i));
    }
    sort(a.begin(),a.end());
    mindist=111*1e18;
    rec(0,n-1);
    fo<<mindist<<' \n'<<ansa+1<<' ' <<ansb+1;

    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```

