

**HỘI THẢO KHOA HỌC  
CÁC TRƯỜNG THPT CHUYÊN  
KHU VỰC DUYÊN HẢI VÀ ĐỒNG BẰNG BẮC BỘ  
NĂM 2021**

**Môn: Tin học**

**KĨ THUẬT SWEEP LINE**

**Người viết: Nguyễn Như Thắng**

**Đơn vị công tác: Trường THPT Chuyên Lào Cai**

**Tháng 9/2021**

## **MỤC LỤC**

1. MỞ ĐẦU .....	5
2. MỘT SỐ BÀI TOÁN VÍ DỤ .....	6
2.1. Ví dụ 1: Thời điểm gặp mặt .....	6
2.2. Ví dụ 2: Đếm số giao điểm .....	8
2.3. Ví dụ 3: Diện tích bao phủ .....	10
2.4. Ví dụ 4: Diện tích che phủ bởi các hình chữ nhật.....	12
2.5. Ví dụ 5: Cặp điểm gần nhau nhất.....	14
2.6. Ví dụ 6: Bao lồi của tập hợp điểm .....	16
3. BÀI TẬP THỰC HÀNH.....	17
3.1. Bài toán 1: Chia kẹo .....	17
3.1.1. Đề bài .....	17
3.1.2. Phân tích bài toán .....	18
3.1.3. Chương trình minh họa .....	18
3.1.4. Test.....	19
3.2. Bài toán 2: Số lần độ cao lặp lại nhiều nhất.....	19
3.2.1. Đề bài .....	19
3.2.2. Phân tích bài toán .....	19
3.2.3. Chương trình minh họa .....	20
3.2.4. Test.....	20
3.3. Bài toán 3: Tổng khoảng cách Manhattan .....	20
3.3.1. Đề bài: .....	20
3.3.2. Phân tích bài toán .....	21
3.3.3. Chương trình minh họa .....	22
3.3.4. Test.....	22
3.4. Bài toán 4: Chiếu sáng .....	22
3.4.1. Đề bài .....	22
3.4.2. Phân tích bài toán .....	23
3.4.3. Chương trình minh họa .....	23
3.4.4. Test.....	24
3.5. Bài toán 5: Những toà nhà bị cô lập.....	24
3.5.1. Đề bài: .....	24
3.5.2. Phân tích bài toán .....	25
3.5.3. Chương trình minh họa .....	25
3.5.4. Test:.....	26
3.6. Bài toán 6: Hội thảo Khoa học (Seminar).....	26
3.6.1. Đề bài .....	26
3.6.2. Phân tích bài toán .....	27

3.6.3. Chương trình minh hoạ .....	28
3.6.4. Test .....	29
3.7. Bài toán 7: Truy vấn hình vuông.....	29
3.7.1. Đề bài .....	29
3.7.2. Phân tích bài toán .....	30
3.7.3. Chương trình minh hoạ .....	31
3.7.4. Test .....	32
3.8. Bài toán 8: Truy vấn tam giác .....	33
3.8.1. Đề bài: .....	33
3.8.2. Phân tích bài toán .....	33
3.8.3. Chương trình minh hoạ .....	34
3.8.4. Test .....	35
3.9. Bài toán 9: Quân xe trên bàn cờ.....	36
3.9.1. Đề bài .....	36
3.9.2. Phân tích bài toán .....	36
3.9.3. Chương trình minh hoạ .....	37
3.9.4. Test .....	38
3.10. Bài toán 10: Đường tròn Manhattan .....	39
3.10.1. Đề bài .....	39
3.10.2. Phân tích bài toán .....	39
3.10.3. Chương trình minh hoạ .....	40
3.10.4. Test .....	41
3.11. Bài toán 11: Tòa nhà (Building – VOI 2020) .....	41
3.11.1. Đề bài .....	41
3.11.2. Phân tích bài toán .....	42
3.11.3. Chương trình minh hoạ .....	43
3.11.4. Test .....	44
3.12. Bài toán 12: Kẹo ngọt (CANDY) .....	45
3.12.1. Đề bài .....	45
3.12.2. Phân tích bài toán .....	45
3.12.3. Chương trình minh hoạ .....	46
3.12.4. Test .....	47
3.13. Bài toán 13: Robot AI .....	47
3.13.1. Đề bài .....	47
3.13.2. Phân tích bài toán .....	48
3.13.3. Chương trình minh hoạ .....	49
3.13.4. Test .....	50
4. BÀI TẬP THẢO LUẬN .....	50
4.1. Thảo luận 1: Hình vuông lớn nhất, bài E – Bubble cup 2013 .....	50
4.1.1. Đề bài .....	50
4.1.2. Thảo luận thuật toán.....	51

4.2. Thảo luận 2: Bảo vệ nhà kho (Bubble cup 2019, bài H).....	52
4.2.1. Đề bài .....	52
4.2.2. Thảo luận huật toán.....	53
5. MỘT SỐ BÀI TỰ LUYỆN.....	54
6. TÀI LIỆU THAM KHẢO.....	55

nhuthangbk@gmail.com

# **SWEEP LINE**

*Tác giả: Nguyễn Như Thắng – THPT Chuyên Lào Cai*

## **1. MỞ ĐẦU**

Qua một số năm dạy HSG môn Tin học, tôi nhận thấy để học sinh có đam mê môn học thì người thầy cần có tâm với nghề, có năng lực chuyên môn tốt, có kĩ năng sư phạm, nhưng bên cạnh đó việc giáo viên luôn tìm ra những cái mới để kích thích sự sáng tạo của học sinh cũng là điều hết sức quan trọng. Học sinh có đam mê, có sáng tạo thì chắc chắn sẽ gặt hái nhiều thành công, không chỉ trong học tập mà còn sẽ thành công trong cuộc sống sau này.

Trong quá trình giảng dạy của mình tôi thấy có một chủ đề về Sweep line có vẻ thú vị, nhưng tìm kiếm tài liệu tiếng Việt rất hiếm, nên trong chuyên đề này tôi xin chia sẻ một số nội dung về sweep line mà tôi đã dạy cho học sinh của mình.

Một hình ảnh rất thú vị là khi bạn quét một cái sân hình chữ nhật (sân nhà, sân trường) thì bạn sẽ có cách quét thế nào, thông thường bạn sẽ quét ngang hoặc quét dọc theo các đường thẳng. Dựa trên mô hình đó, ta có phương pháp xử lí và cách làm về sweep line trong một số bài toán trong Tin học. Kĩ thuật sweep line được sử dụng trong các bài toán hình học hoặc các bài toán được mô hình hoá bằng cách quy về bài toán hình học.

Khi bài toán giải được bằng sweep line thì học sinh có thể quét ngang hoặc dọc, xuôi hoặc ngược,...đều được. Nó phụ thuộc chủ yếu về cách mô hình hoá bài toán và xử lí các sự kiện gặp phải khi quét qua nó.

Một bài toán có thể có rất nhiều cách giải quyết khác nhau. Trong quá trình cho học sinh làm bài, cũng có nhiều hướng giải quyết khác nhau mà học sinh đã đưa ra. Đó là điều tôi rất mong muốn ở học sinh của mình. Có một số bài toán đã quen thuộc, tuy vậy nếu nhìn theo cách sweep line thì mọi việc được xử lí khoa học, dễ hiểu, đưa ra lời giải hay.

Việc đánh giá độ phức tạp thuật toán trong các bài của chuyên đề chỉ đánh giá về thời gian, còn bộ nhớ do chưa quan trọng nên tôi không xem xét.

Do kiến thức, kinh nghiệm chưa nhiều, nên tôi rất chân thành mong các thầy cô góp ý, bổ sung nhiều ý kiến cho tôi để tôi có thể hoàn thiện thêm về chuyên đề này từ đó tạo ra một chuyên đề hoàn chỉnh hơn, để chia sẻ cho đồng nghiệp và học sinh của mình. Rất mong các thầy cô khi đọc chuyên đề của tôi cũng như chuyên đề của các đồng nghiệp khác hãy tương tác hai chiều để chúng ta hoàn thiện mình hơn. Như vậy cũng là cách để ghi nhận, đánh giá công sức của người viết chuyên đề.

Cá nhân tôi rất mong nhận được nhiều lời phê bình của đồng nghiệp.

Trân trọng cảm ơn!

## 2. MỘT SỐ BÀI TOÁN VÍ DỤ

Nhiều bài toán hình học có thể được giải quyết bằng cách sử dụng kỹ thuật đường quét (**sweep line**). Ý tưởng trong các thuật toán này là biểu diễn các đối tượng của bài toán như là một tập các sự kiện tương ứng với các điểm trong mặt phẳng. Các sự kiện được xử lý theo thứ tự tăng dần theo tọa độ x hoặc y của chúng. Trong các bài toán ví dụ, tôi trình bày theo một mô hình lí tưởng. Từ đó học sinh hiểu rõ việc xử lí các sự kiện sau khi được mô hình hoá, trong các ví dụ. Sau đó học sinh sẽ phát triển tự xử lí trên các mô hình không chuẩn, cần kết hợp nhiều kĩ thuật hoặc phương pháp khác.

Sweep line có nhiều tài liệu gọi là thuật toán, nhưng cá nhân tôi nghĩ nó nên gọi là kĩ thuật hợp lí hơn, vì nó gợi ý cho chúng ta một ý tưởng về xử lí các sự kiện, xử lí các thông tin của bài toán theo một thứ tự nhất định.

Các bài tập được phân tích, thiết kế theo nhiều cấp độ, nhiều bài có kết hợp cấu trúc dữ liệu, thuật toán khác. Nên kiến thức nền để học sinh lĩnh hội hết nội dung chuyên đề này khá rộng. Bài tập được đưa ra bao gồm đề bài, phân tích thuật toán, chương trình minh hoạ (có comment để thể hiện ý tưởng), có test cho bài.

Để hình dung rõ về kĩ thuật này, chúng ta đi vào một số ví dụ sau:

### 2.1. Ví dụ 1: Thời điểm gặp mặt

Ta xem xét bài toán sau: Có một công ty có N nhân viên, chúng ta biết rằng mỗi nhân viên có thời gian đến và về trong một ngày nhất định. Nhiệm vụ của bạn là tính số lượng tối đa nhân viên có trong văn phòng tại một thời điểm. Cho  $N \leq 10^5$ , các thời điểm không quá  $10^9$ .

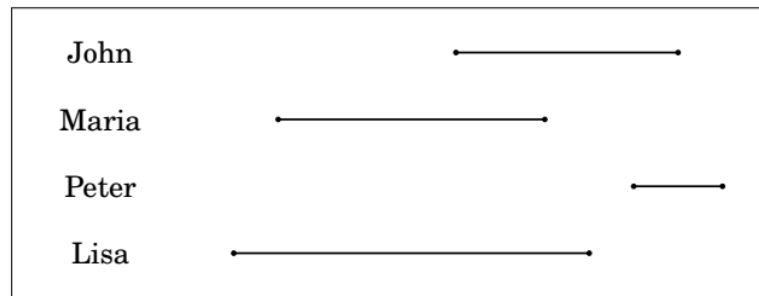
Ví dụ thời gian đến và về của các nhân viên được cho trong bảng sau:

Nhân viên	Thời gian đến	Thời gian về
John	10	15
Maria	6	12
Peter	14	16
Lisa	5	13

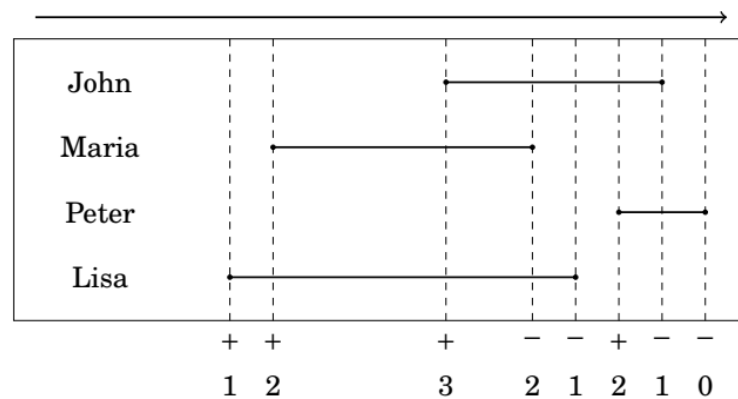
Ta có giải quyết rất tự nhiên như sau: Để biết tại mỗi thời điểm T bất kì, có bao nhiêu nhân viên đang có mặt tại công ty, cách đơn giản nhất là kiểm tra T có nằm trong đoạn thời gian đến và về của một người thì người đó đang ở công ty. Việc kiểm tra tất cả các thời điểm như vậy sẽ mất thời gian  $O(T_{\max} * N)$ .

Rõ ràng với cách làm trên sẽ khá lâu khi  $T_{\max}$  lớn. Ta xét mô hình sử dụng kĩ thuật sweep line như sau:

Với mỗi nhân viên ta sẽ biểu diễn quãng thời gian làm việc của họ như một đoạn thẳng trên mặt phẳng, các sự kiện tương ứng đó là thời điểm đến, thời điểm về của từng người, như sau:



Sau khi mô hình hoá như vậy, nhận xét rằng mỗi thời điểm  $T$ , ta coi nó là một đường thẳng đứng quét từ trái sang phải của mô hình. Với khi gặp sự kiện đến thì ta tăng số nhân viên lên 1, còn gặp sự kiện đi thì giảm số nhân viên đi 1. Còn trong các thời điểm  $T$  quét qua giữa đoạn, không gặp sự kiện nào thì số nhân viên không đổi. Từ đó dẫn đến là ta chỉ cần xử lí các sự kiện đến và đi là đủ.



Trên hình, các đường nét đứt là thời gian  $T$  để quét xem số người có thể gặp là bao nhiêu, ta quét  $T$  từ trái sang phải, theo chiều tăng dần của thời gian. Dễ thấy trong khoảng thời điểm đến của John và thời điểm rời đi của Maria thì số người có mặt tại công ty là lớn nhất (3 người).

Độ phức tạp thuật toán trên là  $O(N \cdot \log(N))$  là thời gian sắp xếp các sự kiện đi và đến tăng dần theo thời gian. Thời gian đọc dữ liệu và xử lí sự kiện ( $N$ ). Rõ ràng cách xử lí này là hiệu quả hơn nhiều, không còn phụ thuộc vào  $T_{\max}$ .

**Chú ý:** Trong trường hợp tại thời điểm  $T$  có cả sự kiện đến và đi thì ta sẽ ưu tiên xử lí sự kiện đến trước.

Chương trình minh hoạ:

```
#include <bits/stdc++.h>
using namespace std;
int n, cnt, res;
vector<pair<int, int>> point;
int main() { //meeting
    cin >> n;
    for(int i=1; i<=n; i++) {
        int in, out; //Thoi gian den, di cua 1 nguoi
        cin >> in >> out;
        point.push_back({in, 0});
```

```

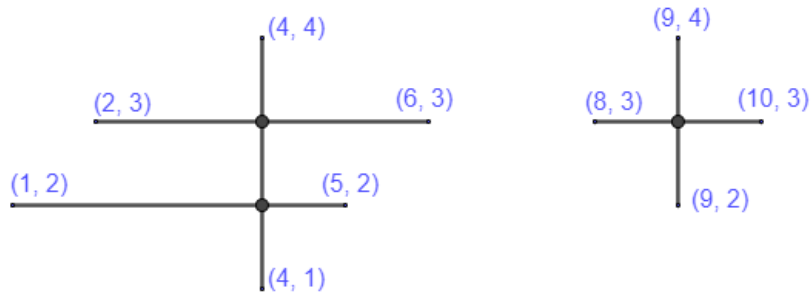
        point.push_back({out, 1});
    }
    sort(point.begin(), point.end());
    for(int i=0; i< point.size(); i++) { ///sweep line
        if(point[i].second==0)
            res=max(res, ++cnt);
        else
            cnt--;
    }
    cout<<res;
}

```

## 2.2. Ví dụ 2: Đếm số giao điểm

Cho  $N$  đoạn thẳng thuộc góc phần tư thứ nhất của mặt phẳng  $Oxy$ , mỗi đoạn thẳng dạng thẳng đứng hoặc nằm ngang, tọa độ 2 đầu đều là các số nguyên. Hãy tính số giao điểm giữa các đoạn thẳng đó. Cho  $N \leq 10^5$ , các tọa độ nguyên không quá  $10^5$ .

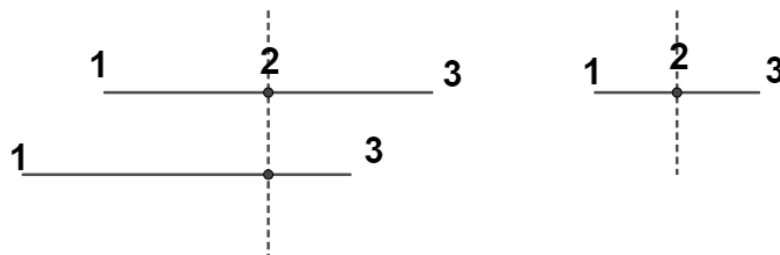
Ví dụ: Có 5 đoạn thẳng, biết tọa độ điểm đầu và cuối của chúng như hình bên dưới, dễ thấy chúng có 3 giao điểm.



Cách xử lý đơn giản nhất: Với mỗi đoạn thẳng bất kì, kiểm tra xem nó cắt bao nhiêu đoạn thẳng khác. Cách kiểm tra 2 đoạn thẳng có cắt nhau hay không trong  $O(1)$  coi như bài tập cho bạn đọc. Độ phức tạp của cách xử lý trên là  $O(N^2)$ .

Để xử lý bằng sweep line, ta mô hình hoá các loại sự kiện như sau:

- + Sự kiện loại 1: Gặp đầu bên trái của đoạn thẳng ngang (Điểm bắt đầu)
- + Sự kiện loại 2: Gặp đoạn thẳng đứng (Đếm số giao điểm)
- + Sự kiện loại 3: Gặp điểm bên phải của đoạn thẳng ngang (Điểm kết thúc).



Chúng ta sẽ xử lý các sự kiện từ trái qua phải, kết hợp với cấu trúc dữ liệu để lưu trữ tung độ của các đoạn thẳng đang được sweep line quét đến.



- Nếu gặp sự kiện điểm bắt đầu đoạn nằm ngang, ta sẽ đưa tung độ của nó vào tập các tung độ đang xét.

- Nếu gặp sự kiện điểm kết thúc đoạn nằm ngang thì ta xoá tung độ của nó ra khỏi tập các tung độ đang xét.

- Nếu gặp sự kiện đoạn thẳng đứng, ta đếm xem có bao nhiêu tung độ trong tập đang xét có giá trị trong **đoạn tung độ** của đoạn thẳng đứng đang xét (cấu trúc dữ liệu để xử lý bài toán đếm có thể dùng segment tree, binary index tree, có thể cần nén số nếu tung độ y lớn).

Độ phức tạp sắp xếp các sự kiện:  $O(N \cdot \log(N))$ ; Độ phức tạp xử lý mỗi sự kiện  $O(\log(N))$ , do đó tất cả các sự kiện là:  $O(N \cdot \log(N))$ .

Độ phức tạp bài toán là:  $O(N \cdot \log(N))$

**Chú ý:** Nếu gặp 3 loại sự kiện (bắt đầu, kết thúc, đếm giao điểm) đồng thời thì ta sẽ xử lý ưu theo thứ tự bắt đầu, đếm, kết thúc (hay  $1 \rightarrow 2 \rightarrow 3$ ).

Trong chương trình minh họa, tôi đưa ra mô hình lí tưởng, các đoạn thẳng đều nằm trong góc phần tư thứ nhất, các đỉnh đều có tọa độ thuộc đoạn  $[1, N]$  (nếu không thì dễ dàng đưa về dạng trên bằng cách dùng nén số).

Dữ liệu vào	Kết quả ra	Giải thích
5 1 2 5 2 2 3 6 3 4 1 4 4 9 2 9 4 8 3 10 3	3	Xem hình vẽ 

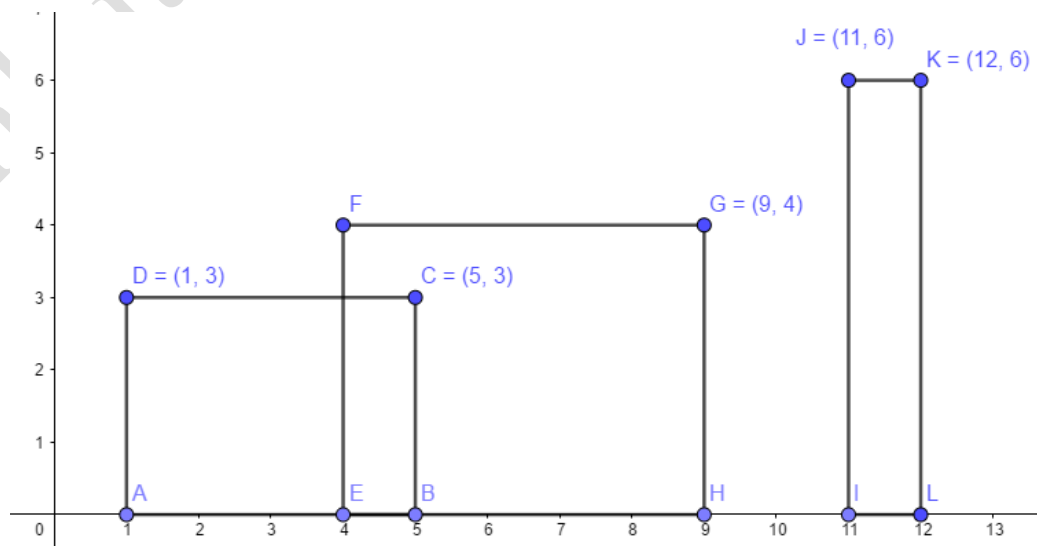
**Chương trình minh họa:**

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
struct P {
    int x,t,y1,y2;
};
vector<P> point;
bool cmp(P a, P b) {
    return ((a.x<b.x) || (a.x==b.x && a.t<b.t));
}
int n,res,BIT[100005];
void upd(int pos,int val) {
    while(pos<n) {
        BIT[pos]+=val;
        pos=pos+(pos&-pos);
    }
}
int get(int pos) {
    int s=0;
    while(pos>0) {
```

```
s=s+BIT[pos];
pos=pos-(pos&-pos);
}
return s;
}
int32_t main() { ///intersection
ios_base::sync_with_stdio(0);
cin>>n;
for(int i=1; i<=n; i++) {
int x1,x2,y1,y2;
cin>>x1>>y1>>x2>>y2;///read data input
if(y1==y2) {
point.push_back({x1,1,y1,0});///begin
point.push_back({x2,3,y1,0});///end
}
if(x1==x2) {
point.push_back({x1,2,y1,y2});///Vertical
}
}
sort(point.begin(),point.end(),cmp);
for(P a:point) {
if(a.t==1)
upd(a.y1,1); ///Begin of segment
if(a.t==3)
upd(a.y1,-1); ///End of segment
if(a.t==2) /// Count the intersection
res=res+get(a.y2)-get(a.y1-1);
}
cout<<res;
}
```

### 2.3. Ví dụ 3: Diện tích bao phủ

Cho  $N$  hình miểng vải hình chữ nhật đặt trên trục  $Ox$  có tọa độ đều là số nguyên, hỏi diện tích phần mà chúng che phủ trên mặt đất là bao nhiêu? Cho  $N \leq 10^5$ , các tọa độ nguyên không quá  $10^5$ .



Xét tất cả các ô vuông đơn vị trong mặt phẳng, đếm số ô nằm trong một trong các hình chữ nhật đưa ra. Như vậy độ phức tạp khá lớn, do phụ thuộc vào kích thước các hình và số hình.

Với sweep line ta xử lí như sau:

- Sắp xếp các đỉnh phía trên theo thứ tự tăng dần của hoành độ, nếu cùng hoành độ thì ưu tiên điểm kết thúc trước điểm bắt đầu, nếu cùng là điểm kết thúc thì ưu tiên điểm có tung độ lớn hơn trước.

- Mô hoá các sự kiện gồm điểm trái trên bắt đầu, và trái trên kết thúc. Do các hình được đặt lên trục  $Ox$  nên ta chỉ quan tâm cạnh phía trên của hình.

- Khi có sự kiện gặp đỉnh trái trên hoặc phải trên, ta kiểm tra xem hình chữ nhật nào có độ cao lớn nhất đang phủ để đó, khi đó diện tích bao phủ là  **$maxY * (\text{Khoảng cách giữa sweep line đang xét với sweep line gần nhất bên trái})$** .

Để tìm max, và xoá một giá trị tung độ thì ta có thể dùng multiset.

Độ phức tạp sắp xếp:  $O(N \cdot \log(N))$ . Thời gian xử lí  $O(N)$ . Độ phức tạp toàn thuật toán:  $O(N \cdot \log(N))$ .

Dữ liệu mẫu:

Cover.inp	Cover.out	Giải thích
3 1 3 5 3 4 4 9 4 11 6 12 6	35	Do các hình chữ nhật đặt trên $Ox$ , nên dữ liệu vào chỉ có 2 đỉnh trái trên, phải trên của hình chữ nhật. Đây là dữ liệu mẫu của hình bên trên.

Chương trình minh hoạ:

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
struct Point {
    int x, y;
    bool t; ///type
};
operator <(Point a, Point b) {
    return (a.x<b.x || (a.x==b.x && a.t>b.t));
}
vector<Point> P;
multiset<int,greater<int>> S;
int n,res;
int32_t main() {
    // freopen("cover.inp","r",stdin);
    // freopen("cover.out","w",stdout);
    ios_base::sync_with_stdio(0);
    cin>>n;
    for(int i=1; i<=n; i++) {
        int x1,y1,x2,y2;
```

```
cin>>x1>>y1>>x2>>y2;
P.push_back({x1,y1,0});
P.push_back({x2,y2,1});
}
sort(P.begin(),P.end());
int x=0;
for(int i=0; i<P.size(); i++) {
    if(!P[i].t) {
        int l= *S.begin();
        res=res+ l * (P[i].x-x);
        x=P[i].x;
        S.insert(P[i].y);
    } else {
        int l= *S.begin();
        res=res+ l * (P[i].x-x);
        x=P[i].x;
        S.erase(S.lower_bound(P[i].y));
    }
}
cout<<res<<"\n";
return 0;
}
```

## 2.4. Ví dụ 4: Diện tích che phủ bởi các hình chữ nhật

Mở rộng của ví dụ 3. Cho N hình chữ nhật có cạnh song song với hệ trục toạ độ, các hình chữ nhật đặt ngẫu nhiên trong mặt phẳng. Mỗi hình chữ nhật cho toạ độ góc trái trên  $(x_1; y_1)$  và phải dưới  $(x_2; y_2)$ . Các toạ độ là số nguyên và có trị tuyệt đối không quá 50000.

Hãy tính diện tích bị che phủ trên mặt phẳng của N hình chữ nhật đó?

Lời giải: Bằng cách xử lí tương tự như ví dụ trên. Khi quét từ trái qua phải, nếu gặp cạnh bên trái của hình chữ nhật (gọi là cạnh mở), ta cần cập nhật đoạn trên tung độ bị che phủ. Khi gặp cạnh bên phải (gọi là cạnh đóng), ta phải xoá thông tin mà nó che phủ trên đoạn tung độ. Khi gặp bất kì sự kiện nào thì cần tính diện tích sau đó mới cập nhật thông tin của cả khe.

Việc tính diện tích được tính toán theo khe, giữa 2 sweep line liên tiếp, diện tích che phủ được tăng thêm lượng:

**(Độ dài đoạn tung độ bị che phủ) \* (khoảng cách 2 sweep line).**

Để giải quyết việc cập nhật đoạn tung độ bị che phủ, ta sử dụng cấu trúc Segment tree, mỗi nút trên cây quản lý 2 thông tin:

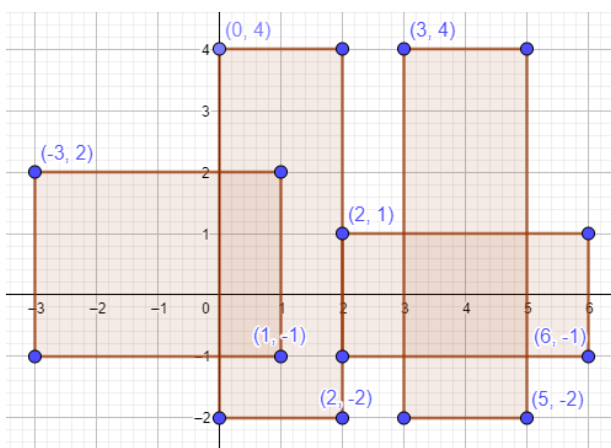
- cnt: Số hình chữ nhật đang che phủ hoàn toàn nút quản lý.
- cover: Độ dài đoạn tung độ bị che phủ thuộc nút quản lý

Nếu cnt=0 thì độ dài đoạn tung độ nút quản lý bằng tổng độ dài đoạn tung độ do 2 nút con của nó quản lý.

Dữ liệu mẫu:

HCN.inp	HCN.out	Giải thích
---------	---------	------------

4 -3 2 1 -1 0 4 2 -2 3 4 5 -2 2 1 6 -1	37	Quan sát hình bên dưới
--	----	------------------------



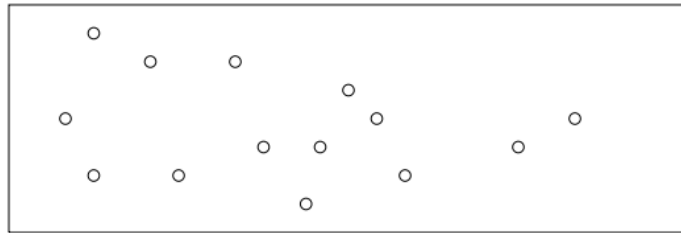
### Chương trình minh họa:

```
#include<bits/stdc++.h>
#define int long long
const int maxn = 50005;
using namespace std;
int n,res;
struct Event {
    int x,low,high,t; ///-1: close, 1: open
    bool operator <(Event &a) {
        return (x<a.x) || ((x==a.x) && t<a.t);
    }
};
struct Node {
    int cnt, cover;
} ST[16*maxn];
void update(int id, int L, int R, int u, int v, int val) {
    if(v<=L || R<=u)
        return;
    if(u<=L && R<=v) {
        ST[id].cnt+=val;
        if(ST[id].cnt==0)
            ST[id].cover=ST[2*id].cover+ST[2*id+1].cover;
        else
            ST[id].cover = R-L;
        return;
    }
    int mid=(L+R)/2;
    update(2*id,L,mid,u,v,val);
    update(2*id+1,mid,R,u,v,val);
    if(ST[id].cnt==0)
        ST[id].cover=ST[2*id].cover+ST[2*id+1].cover;
}
vector<Event> E;
main() {
```

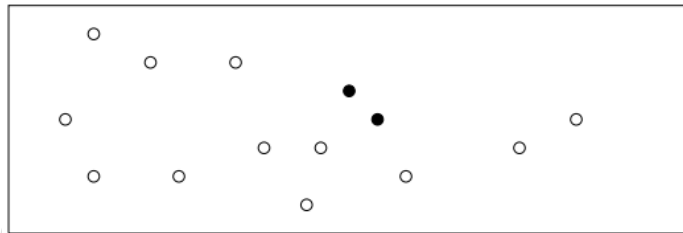
```
// freopen("HCN.inp", "r", stdin);
cin>>n;
for(int i=1; i<=n; i++) {
    int x1,y1,x2,y2;
    cin>>x1>>y1>>x2>>y2;
    E.push_back({x1,y2,y1,1});///open
    E.push_back({x2,y2,y1,-1});///close
}
sort(E.begin(),E.end());
for(int i=0; i<E.size(); i++) {
    res=res+ST[1].cover*(E[i].x-E[i-1].x);
    update(1,-maxn,maxn,E[i].low,E[i].high,E[i].t);
}
cout<<res;
}
```

## 2.5. Ví dụ 5: Cặp điểm gần nhau nhất

Cho  $N$  điểm trên mặt phẳng, hãy xác định 2 điểm có khoảng cách gần nhau nhất. Ví dụ: có các điểm sau:



Cặp điểm gần nhau nhất là:

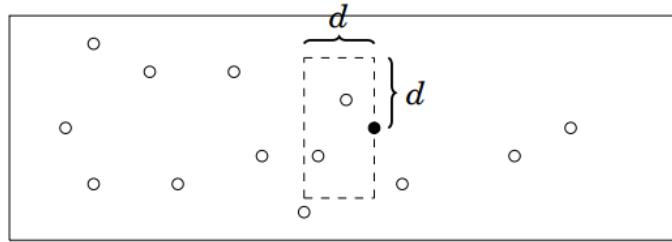


Cho biết 2 điểm  $A(x_A, y_A)$  và  $B(x_B, y_B)$  thì độ dài đoạn  $AB = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$ . Do đó khoảng cách giữa 2 điểm bất kì tính trong  $O(1)$ . Với cách tư duy đơn giản, ta đi kiểm tra tất cả các khoảng cách để tìm khoảng cách nhỏ nhất. Độ phức tạp:  $O(N^2)$ .

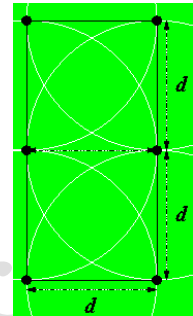
Bằng cách dùng sweep line ta hoàn toàn có thể xử lí trong thời gian  $O(N \cdot \log(N))$  như sau:

- Gọi sự kiện là khi sweep line quét tới điểm mới.

- Ta luôn duy trì khoảng cách  $d$  nhỏ nhất giữa các điểm ở bên trái (các điểm mà sweep line đã quét qua. Giả sử điểm đang xét có tọa độ  $(x, y)$ . Khi đó các điểm có khả năng tạo ra khoảng cách nhỏ hơn  $d$  đến  $(x, y)$  là các điểm có hoành độ thuộc  $[x - d, x]$  và tung độ trong đoạn  $[y - d, y + d]$ .



**Nhận xét:** Trong hình chữ nhật kích thước  $d * 2d$  ta có thể đặt vào đó không quá 6 điểm sao cho khoảng cách giữa 2 điểm bất kì không nhỏ hơn  $d$ . Để dàng chứng minh điều đó, bạn đọc có thể tham khảo cách chứng minh tại: <https://www.cs.mcgill.ca/~cs251/ClosestPair/proofbox.html>. Khi đó cách bố trí 6 điểm tối ưu như sau:



Để xác định nhanh các điểm có trong hình chữ nhật này, ta có thể dùng chặt nhị phân trên cấu trúc dữ liệu **multiset** có sẵn trong C++.

Độ phức tạp để xác định khoảng cách nhỏ nhất từ 1 điểm bất kì với các điểm bên trái nó là  $O(\log(N))$ . Do đó độ phức tạp về thời gian của bài toán là:  $O(N \cdot \log(N))$

#### Chương trình minh họa:

```
#include <bits/stdc++.h>
using namespace std;
#define rep(i, j, n) for(int i=(j); i<(int)(n); ++i)
#define lp(i, cnt) rep(i, 0, cnt)
const double OO = 1e8;
const double EPS = (1e-8);
int dcmp(double x, double y) {
    return abs(x - y) <= EPS ? 0 : x < y ? -1 : 1;
}
typedef complex<double> point;
#define X real()
#define Y imag()
#define vec(a,b) ((b)-(a))
#define length(a) (hypot((a).imag(), (a).real()))
struct cmpX {
    bool operator()(const point &a, const point &b) {
        if(dcmp(a.X, b.X) != 0)
            return dcmp(a.X, b.X) < 0;
        return dcmp(a.Y, b.Y) < 0;
    }
};
struct cmpY {
    bool operator()(const point &a, const point &b) {
        if(dcmp(a.Y, b.Y) != 0)
            return dcmp(a.Y, b.Y) < 0;
        return dcmp(a.X, b.X) < 0;
    }
};
double closestPair(vector<point> &eventPts) {
    double d = OO;
    multiset<point, cmpY> active;
    sort(eventPts.begin(), eventPts.end(), cmpX());
    int left = 0;
    for(int right=0; right<(int)eventPts.size(); ++right) {
```

```

while(left<right&&eventPts[right].X-eventPts[left].X>d)
    active.erase(active.find(eventPts[left++]));
auto asIt=active.lower_bound(point(-OO,eventPts[right].Y-d));
auto aeIt=active.upper_bound(point(-OO,eventPts[right].Y+d));
for(; asIt != aeIt; asIt++)
    d=min(d, length(eventPts[right]-*asIt));
active.insert(eventPts[right]);
}
return d;
}
int main() {
    int n;
    while(cin >> n && n) {
        vector<point> eventPts(n);
        for(int i = 0; i < n; ++i) {
            double x, y;
            cin >> x >> y;
            eventPts[i] = point(x, y);
        }
        double d = closestPair(eventPts);
    }
    return 0;
}

```

## 2.6. Ví dụ 6: Bao lồi của tập hợp điểm

Bao lồi của tập hợp điểm là đa giác lồi nhỏ nhất mà chứa toàn bộ tập hợp điểm. Bao lồi được chỉ ra bằng các đỉnh liên tiếp của nó theo chiều kim đồng hồ (hoặc ngược lại).

Ví dụ:

Dữ liệu vào	Kết quả ra	Giải thích
6 2 2 1 1 2 1 3 0 0 0 3 3	0 0 3 3 3 0	

Để tìm bao lồi ta sắp xếp các đỉnh theo thứ tự tăng dần của hoành độ, nếu cùng hoành độ thì tăng dần theo tung độ.

Dùng sweep line quét từ trái sang phải để xác định các đỉnh thuộc nửa trên của bao lồi. Sau đó lại quét lại từ phải qua trái qua phải để xác định các đỉnh thuộc nửa dưới của bao lồi.

Xét việc xác định nửa trên của bao lồi như sau:

- Sự kiện là khi gặp một đỉnh theo thứ tự sắp xếp từ trái qua phải.
- Mỗi khi gặp sự kiện, nếu điểm đó tạo ra đoạn gấp khúc quay ngược chiều kim đồng hồ thì điểm trước đó sẽ bị loại khỏi bao lồi. Cứ như vậy ta xây dựng xong các đỉnh thuộc nửa trên của bao lồi. Làm tương tự ta có nửa dưới bao lồi.

Độ phức tạp sắp xếp:  $O(N \cdot \log(N))$ . Độ phức tạp tìm bao lồi:  $O(N)$ .



Độ phức tạp chung của thuật toán:  $O(N \cdot \log(N))$

Chương trình minh họa:

```
#include<bits/stdc++.h>
#define pii pair<int,int>
using namespace std;
int n;
vector<pii> point,convex;
bool ccw(pii a, pii b, pii c) {
    int x1=b.first-a.first;
    int y1=b.second-a.second;
    int x2=c.first-b.first;
    int y2=c.second-b.second;
    return (x1*y2-y1*x2>=0);
}
int main() {
    // freopen("Convex.inp","r",stdin);
    cin>>n;
    for(int i=1; i<=n; i++) {
        int x,y;
        cin>>x>>y;
        point.push_back({x,y});
    }
    sort(point.begin(),point.end());
    convex.push_back(point[0]);
    convex.push_back(point[1]);
    for(int i=2; i<point.size(); i++) {
        while(convex.size()>=2 &&
            ccw(convex[convex.size()-2], convex[convex.size()-1],point[i])) {
            convex.erase(convex.end());
        }
        convex.push_back(point[i]);
    }
    convex.push_back(point[point.size()-2]);
    for(int i=point.size()-3; i>=0; i--) {
        while(ccw(convex[convex.size()-2],convex[convex.size()-1],
point[i])) {
            convex.erase(convex.end());
        }
        convex.push_back(point[i]);
    }
    convex.erase(convex.end()); ///xoa diem dau do lap lai
}
```

### 3. BÀI TẬP THỰC HÀNH

#### 3.1. Bài toán 1: Chia kẹo

##### 3.1.1. Đề bài

Có  $N$  em bé được xếp thành một vòng tròn, được đánh số từ 1 đến  $N$  theo chiều kim đồng hồ. Người ta tổ chức chia kẹo cho các em bé trong  $M$  lượt, mỗi lượt xác định một cặp chỉ số  $L, R$ . Tất cả các em bé được đánh số từ  $L$  đến  $R$  theo chiều kim đồng hồ sẽ được nhận 1 cái kẹo. Hỏi sau  $M$  lượt chia kẹo, thì số kẹo lớn nhất mà một em bé có thể nhận được là bao nhiêu và có bao nhiêu em bé nhận được số kẹo như vậy?

Dữ liệu vào: Đọc vào từ tệp CHIAKEO.inp

Dòng đầu tiên là số  $N, M$  ( $1 \leq N \leq 10^9; 1 \leq M \leq 10^5$ ) là số em bé và số lần chia kẹo.

M dòng tiếp theo là cặp chỉ số  $L, R$  ( $1 \leq L, R \leq N$ ).

**Kết quả ra:** Ghi ra tệp CHIAKEO.out

Ghi 2 số là đáp số của yêu cầu trong đề bài, số kẹo lớn nhất và số em bé nhận được số kẹo đó.

**Ví dụ:**

CHIAKEO.inp	CHIAKEO.out	Giải thích
5 2 1 5 4 2	2 4	Ban đầu các em bé đều có 0 kẹo. Sau lượt 1, số kẹo là: 1, 1, 1, 1, 1 Sau lượt 2, số kẹo là: 2, 2, 1, 2, 2

Subtask 1: 60% test có  $N, M \leq 10^3$ .

Subtask 2: 20% test có  $N, M \leq 10^5$ ;

Subtask 3: 20% test còn lại không có thêm ràng buộc gì.

### 3.1.2. Phân tích bài toán

**Với subtask 1:** Hoàn toàn có thể làm bằng duyệt toàn bộ. Với mỗi truy vấn  $\{L, R\}$ , nếu  $L > R$  thì chia thành 2 đoạn  $[L, N], [1, R]$ . Sử dụng vòng lặp để tăng số kẹo của các em bé. Độ phức tạp thuật toán  $O(N * M)$ .

**Với subtask 2:** Dùng đánh dấu, cộng dồn. Với mỗi truy vấn  $\{L, R\}$  nếu  $L \leq R$  thì đánh dấu +1 tại vị trí  $L$ , trừ 1 tại vị trí  $R+1$ ; Nếu  $L > R$  thì ta xử lí thành 2 đoạn  $[L, N], [1, R]$ . Sau đó cộng dồn từ 1 đến  $N$  và xử lí bài toán đếm sơ đẳng. Độ phức tạp:  $O(N)$ .

**Với subtask 3:** Khi  $N$  lớn thì không thể xử lí như Subtask 2, nên khi đó cải tiến bằng sweep line. Độ phức tạp  $O(M \cdot \log(M))$ .

### 3.1.3. Chương trình minh họa

```
#include<bits/stdc++.h>
using namespace std;
int n,m,L,R,cnt,res,res2;
vector<pair<int,int>> point;
int main() {
    // freopen("CHIAKEO.inp","r",stdin);
    cin>>n>>m;
    for(int i=1; i<=m; i++) {
        cin>>L>>R;
        if(L<=R) {
            point.push_back({L,0});///begin
            point.push_back({R,1});///end
        }
        if(L>R) {
            point.push_back({1,0});
            point.push_back({R,1});
            point.push_back({L,0});
            point.push_back({n,1});
        }
    }
    sort(point.begin(),point.end());
```

```
for(int i=0; i<point.size(); i++)
    if(point[i].second==0) {
        cnt++;
        if(cnt>res) {
            res=cnt;
            res2=point[i+1].first-point[i].first+1;
        } else if(cnt==res)
            res2+=point[i+1].first-point[i].first+1;
    } else {
        cnt--;
    }
cout<<res<<" "<<res2;
}
```

### 3.1.4. Test

<https://drive.google.com/drive/folders/1ofheTovqzJrTWXeCXBxQA-lH00qqCmT?usp=sharing>

## 3.2. Bài toán 2: Số lần độ cao lặp lại nhiều nhất

### 3.2.1. Đề bài

Hệ thống Kiểm soát không lưu có trách nhiệm điều hướng, đảm bảo an toàn cho các chuyến bay từ lúc cất cánh đến khi hạ cánh. Một máy bay nhận được các tín hiệu điều khiển từ Hệ thống yêu cầu tăng, giảm độ cao đến độ cao  $H_i$  để tránh va chạm, các độ cao liên tiếp đảm bảo khác nhau. Cơ trưởng ghi lại nhật kí điều khiển liên tiếp trong một khoảng thời gian. Hỏi số lần nhiều nhất có thể mà máy bay bay qua độ cao nào đó.

Dữ liệu vào: Đọc vào từ tệp FLYMODE.inp

Dòng đầu ghi số  $N$  ( $2 \leq N \leq 10^5$ ) là số lệnh điều khiển.

Dòng hai ghi  $N$  số nguyên là độ cao  $H_i$  của lệnh điều khiển ( $1 \leq H_i \leq 10^9$ ).

Kết quả ra: Ghi ra tệp FLYMODE.out

Ghi số lần nhiều nhất có thể mà máy bay bay qua độ cao nào đó.

Ví dụ:

FLYMODE.inp	FLYMODE.out	Giải thích
5 1 2 3 2 3	3	Các độ cao trong khoảng (2,3) đều được lặp lại 3 lần

Subtask 1: 50% test có  $1 \leq N, H_i \leq 1000$ ;

Subtask 2: 50% test có  $1 \leq N, H_i \leq 10^5$ .

### 3.2.2. Phân tích bài toán

Rõ ràng là độ cao bay qua nhiều nhất chắc chắn là số nguyên. Nên ta chỉ cần xét các độ cao là nguyên, các độ cao là số thực không cần quan tâm.

Với subtask 1: Tạo ra  $N-1$  đoạn, sau đó xét tất cả các độ cao và đếm số đoạn mà chứa độ cao đó. Độ phức tạp:  $O(N \cdot H)$

**Với subtask 2:** Đánh dấu bằng map<int,int>; hoặc nén số và sau đó dùng mảng đánh dấu, đếm. Tương tự cách làm bài CHIAKEO, MEETING.

Độ phức tạp:  $O(N \cdot \log(N))$

### 3.2.3. Chương trình minh họa

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    // FLYMODE
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    int n;
    cin >> n;
    map<int, long long int> umap;
    int arr[n];
    for(int i=0; i<n; i++)
        cin >> arr[i];
    for(int i=0; i<n-1; i++) {
        int x=max(arr[i], arr[i+1]);
        int y=min(arr[i], arr[i+1]);
        umap[y]++;
        umap[x]--;
    }
    long long int sum=0, ans=0;
    for(auto it=umap.begin(); it!=umap.end(); it++) {
        sum+=it->second;
        ans=max(sum, ans);
    }
    cout << ans << endl;
    return 0;
}
```

### 3.2.4. Test

[https://drive.google.com/drive/folders/1ofheTovqzJrTW\\_XeCXBxQA-IH00qqCmT?usp=sharing](https://drive.google.com/drive/folders/1ofheTovqzJrTW_XeCXBxQA-IH00qqCmT?usp=sharing)

## 3.3. Bài toán 3: Tổng khoảng cách Manhattan

### 3.3.1. Đề bài:

Cho  $N$  điểm trong mặt phẳng tọa độ  $Oxy$ , điểm thứ  $i$  có tọa độ nguyên  $(x_i; y_i)$ . Ta định nghĩa khoảng cách Manhattan giữa hai điểm  $i, j$  là  $|x_i - x_j| + |y_i - y_j|$ . Hãy tính tổng khoảng cách Manhattan giữa mọi cặp điểm. Khoảng cách Manhattan còn được gọi là khoảng cách trong thành phố, ở đó một thành phố được chia thành các dãy nhà, khu phố hình ô vuông, khoảng cách Manhattan chính là khoảng cách ngắn để di chuyển từ điểm A đến điểm B trong thành phố (không tính đường chim bay, đường thẳng trực tiếp nối A với B).

**Dữ liệu vào:** Đọc vào từ tệp Manhattan.inp

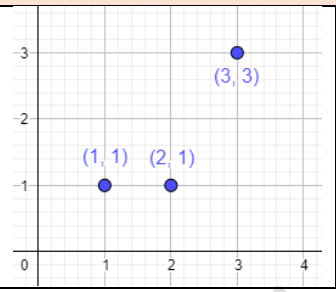
Dòng đầu tiên là số  $N$  ( $1 \leq N \leq 10^5$ ), số cặp điểm

N dòng sau, dòng thứ  $i$  ghi cặp số nguyên  $(x_i, y_i)$  ( với  $-10^9 \leq x_i, y_i \leq 10^9$ ) là toạ độ điểm thứ  $i$  trên mặt phẳng.

**Kết quả ra:** Ghi ra tệp Manhattan.out

Tổng khoảng cách Manhattan giữa mọi cặp điểm

**Ví dụ:**

Manhattan.inp	Manhattan.out	Giải thích	
3 1 1 1 2 3 3	8		Khoảng cách đỉnh 1,2 là: 1 Khoảng cách đỉnh 2,3 là: 3. Khoảng cách đỉnh 1,3 là: 4
1 1 1	0		

Subtask 1: 50% test có  $N \leq 10^3$ ;

Subtask 2: 50% test có  $N \leq 10^5$ .

### 3.3.2. Phân tích bài toán

Subtask 1: Dễ dàng duyệt toàn bộ các khoảng cách giữa các điểm. Độ phức tạp thuật toán  $O(N^2)$ .

Subtask 2: Thông thường, nhiều khi giải bài toán mới, có không gian nhiều chiều ta có thể xét từ không gian ít chiều hơn. Nếu giải được thì ta có thể phát triển để giải bài toán nhiều chiều.

Xét bài toán có  $N$  điểm trên trục  $Ox$ . Tính tổng tất cả các khoảng cách giữa chúng.

Ta thấy, với  $A, B, C$  nằm trên trục  $Ox$  như hình vẽ, thì khoảng cách từ điểm  $B$  đến các điểm bên trái nó là  $A$  tính như sau:  $AB = OB - OA$

Khoảng cách từ  $C$  đến 2 điểm bên trái nó là  $A, B$  được tính như sau:

$$AC + BC = (OC - OA) + (OC - OB) = 2 * OC - (OA + OB).$$



Từ đó có có sweep line quét dọc theo trục  $Ox$  từ trái qua phải. Với mỗi sự kiện gặp một điểm mới, ta tính khoảng cách từ nó đến tất cả các điểm bên trái nó như cách mô tả trên.

Trở lại bài toán, thực chất ta đi tính các khoảng cách hình chiếu trên  $Ox, Oy$  tương ứng của tất cả các điểm. Vậy, chúng ta đi giải 2 bài toán rời nhau.

Độ phức tạp thuật toán bằng thời gian sắp xếp hoành độ, tung độ các điểm tăng dần là  $O(N \cdot \log(N))$ .

### 3.3.3. Chương trình minh họa

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
int n,x[100005],y[100005],res,s;
int32_t main() {
    // freopen("MANHATTAN.inp","r",stdin);
    cin>>n;
    for(int i=1; i<=n; i++) {
        cin>>x[i]>>y[i];
    }
    sort(x+1,x+1+n);
    sort(y+1,y+1+n);
    s=0;
    for(int i=1; i<=n; i++) {
        res=res+x[i]*(i-1)-s;
        s=s+x[i];
    }
    s=0;
    for(int i=1; i<=n; i++) {
        res=res+y[i]*(i-1)-s;
        s=s+y[i];
    }
    cout<<res;
}
```

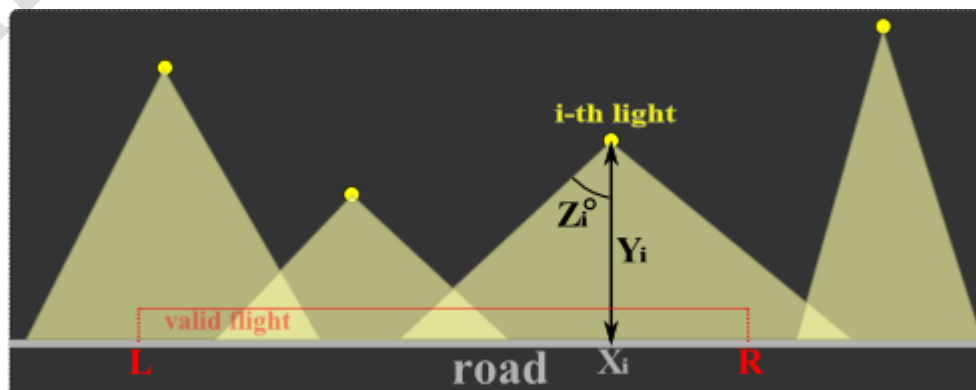
### 3.3.4. Test

[https://drive.google.com/drive/folders/1ofheTovqzJrTW\\_XeCXBxQA-lH00qqCmT?usp=sharing](https://drive.google.com/drive/folders/1ofheTovqzJrTW_XeCXBxQA-lH00qqCmT?usp=sharing)

## 3.4. Bài toán 4: Chiếu sáng

### 3.4.1. Đề bài

Có  $N$  bóng đèn được đặt dọc con đường theo trục  $Ox$ . Bóng đèn thứ  $i$  có hoành độ  $X_i$ , tung độ  $Y_i$  và chiếu sáng một vùng hình tam giác cân có nửa góc ở đỉnh tính theo đơn vị độ là  $Z_i^\circ$ . Có một thiết bị bay không người lái muốn bay tại độ cao  $H$  dọc theo đoạn từ  $L$  đến  $R$ , nhưng phải dưới phạm vi được chiếu sáng của các bóng đèn. Tìm  $H$  lớn nhất có thể? Coi các bóng đèn không gây cản trở với thiết bị bay.



Dữ liệu vào: Đọc vào từ tệp LIGHT.inp

Dòng đầu là số  $N$ ,  $L$ ,  $R$  là số bóng đèn, hoành độ của đoạn giám sát.

$N$  dòng tiếp theo ghi  $X_i, Y_i, Z_i$  toạ độ và nửa góc chiếu của đèn từng vị trí.

**Kết quả ra:** Ghi ra tệp LIGHT.out

Ghi độ cao  $H$  lớn nhất. Do  $H$  có thể là số thực nên sai số cho phép là  $10^{-6}$ .

**Ví dụ:**

LIGHT.inp	LIGHT.out
2 3.2 7.3 3.2 4.7 28 7.3 4.2 75	3.30075964

Ràng buộc:  $1 \leq N \leq 50000$ ;  $0 \leq L < R \leq 2000$ ;

$0 < X_i, Y_i \leq 2000$ ,  $15 \leq Z_i \leq 75$ ;

$N$  là số nguyên nhưng các số khác có thể không nguyên.

### 3.4.2. Phân tích bài toán

Xét các sweep line nằm ngang, quét từ trên xuống, với mỗi sweep line, sẽ kiểm tra xe toàn bộ đoạn có toạ độ  $(L, H)$  đến  $(R, H)$  trên sweep line có nằm trọn trong vùng được chiếu sáng hay không. Nếu không ta lại xét sweep line tiếp theo.

Nhận thấy nếu độ cao  $H_1$  thoả mãn thì độ cao  $H_2 < H_1$  chắc chắn thoả mãn yêu cầu máy bay không người lái phải bay dưới vùng được đèn chiếu sáng. Do đó ta có thể chặt nhị phân độ cao  $H$  để tìm đáp số.

Với mỗi độ cao  $H$  ta kiểm tra như sau:

- Mỗi đèn nếu được đặt ở độ cao lớn hơn  $H$  thì nó chiếu sáng lên sweep line một đoạn có độ dài nửa đáy là:  $Base = (Y_i - H) * \tan(Z_i)$ .

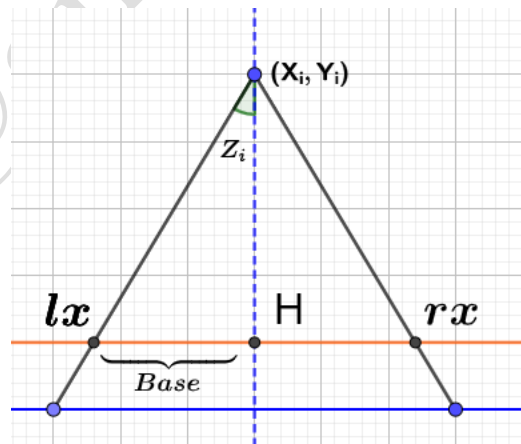
Khi đó điểm bên trái nó chiếu sáng đến là:  $lx = X_i - Base$

Điểm bên phải là:  $rx = X_i + Base$

Việc còn lại là kiểm tra các đoạn  $[lx, rx]$  có phủ hết đoạn  $L, R$  khá dễ dàng trong  $O(N)$

### 3.4.3. Chương trình minh hoạ

```
#include<bits/stdc++.h>
using namespace std;
#define N 50005
#define eps 1e-9
double x[N], y[N], z[N], l, r;
int n;
bool check(double ym) {
    vector<pair<double, double>> p;
```



```
for(int i = 0 ; i < n ; i++) {
    if(y[i]+eps < ym)
        continue;
    double s = tan(z[i]*acos(-1)/180.0)*(y[i]-ym);
    if(x[i]+s+eps < l || x[i]-s > r+eps)
        continue;
    p.push_back({x[i]-s,x[i]+s});
}
p.push_back({l,l});
p.push_back({r,r});
sort(p.begin(),p.end());
double fin = LLONG_MIN;
for(int i = 0 ; i < p.size() ; i++) {
    if(p[i].first > fin+eps && fin > INT_MIN)
        return true;
    fin = max(fin,p[i].second);
}
return false;
}

int main() {
    ios_base::sync_with_stdio(0), cin.tie(0);
    // freopen("LIGHT.inp","r",stdin);
    cin >> n >> l >> r;
    for(int i = 0 ; i < n ; i++)
        cin >> x[i] >> y[i] >> z[i];
    double l = 0, r = 1000, med;
    while(r-l > eps) {
        med = (l+r)/2;
        if(check(med))
            r = med;
        else
            l = med;
    }
    printf("%.9f\n",med);
    return 0;
}
```

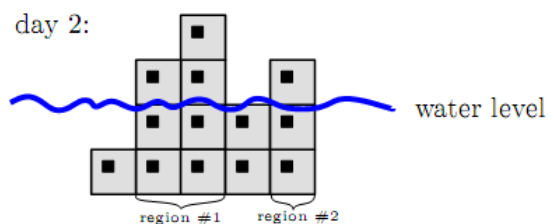
#### 3.4.4. Test

[https://drive.google.com/drive/folders/1ofheTovqzJrTW\\_XeCXBxQA-IH00qqCmT?usp=sharing](https://drive.google.com/drive/folders/1ofheTovqzJrTW_XeCXBxQA-IH00qqCmT?usp=sharing)

### 3.5. Bài toán 5: Những toà nhà bị cô lập

**3.5.1. Đề bài:** Dọc theo đại lộ xinh đẹp của thành phố sát bờ biển, có rất nhiều toà nhà cao chọc trời được xây san sát nhau. Mỗi toà nhà có chiều rộng 100 (mét) và chiều cao nào đó. Thật không may, do hiện tượng nóng lên của trái

đất, mực nước biển bắt đầu tăng mỗi ngày 1 mét. Nếu toà nhà không cao hơn mực nước biển thì nó gọi là bị ngập. Người ta định nghĩa một vùng cô lập là tập hợp tối đa các toà nhà liên tiếp mà không bị ngập. Việc tính toán số vùng cô lập





để thành phố bố trí phương tiện cung cấp hàng hoá cho người dân là hết sức quan trọng để vượt qua giai đoạn khó khăn này. Bạn hãy giúp thành phố tính toán số vùng cô lập sau một số ngày? Hình bên, có 5 toà nhà, và 2 vùng cô lập sau ngày thứ 2.

Dữ liệu vào: Đọc vào từ tệp Isolation.inp

Dòng đầu là số  $N, Q$  ( $N, Q \leq 10^5$ ) là số toà nhà và số ngày cần tính toán số vùng cô lập.

Dòng thứ 2 chứa  $N$  số  $a_1, a_2, \dots, a_N$  ( $1 \leq a_i \leq 10^9$ ) là độ cao các toà nhà.

Dòng thứ 3 chứa  $Q$  số  $t_1, t_2, \dots, t_Q$  ( $1 \leq t_i \leq 10^9$ ) là ngày mà cần tính toán số vùng cô lập.

Kết quả ra: Ghi ra tệp Isolation.out

In ra câu trả lời cho  $Q$  câu hỏi đưa ra

Ví dụ:

Isolation.inp	Isolation.out	Giải thích
5 3 1 3 4 2 3 1 2 3	1 2 1	Quan sát hình trong đề bài

Subtask 1: 50% test có  $N, Q \leq 10^3$ ;

Subtask 2: 50% test có  $N, Q \leq 10^5$ .

### 3.5.2. Phân tích bài toán

Với subtask 1: Với mỗi ngày, ta đi duyệt tất cả các toà nhà từ 1 đến  $N$  để đếm số vùng bị cô lập. Với ngày  $i$ , theo điều kiện là các toà nhà liên tiếp nhau đều cao hơn  $t_i$  và cực đại nên bên trái và bên phải đều bị ngập. Độ phức tạp  $O(N * Q)$ .

Với subtask 2: Dùng một sweep line nằm ngang, quét từ trên xuống dưới. Sweep line quét qua các  $t_1, t_2, \dots, t_Q$ .

Sự kiện ở đây là sweep line gặp đỉnh của toà nhà. Mỗi khi có toà nhà cao hơn sweep line thì ta có thêm một vùng cô lập, tuy vậy, nếu toà nhà bên trái hoặc bên phải đã tạo ra vùng cô lập rồi, thì số vùng cô lập lại bị giảm đi 1.

Mỗi toà nhà cũng chỉ xét 1 lần, mỗi thời điểm cũng chỉ xét một lần. Ta cần sắp xếp độ cao các toà nhà và các truy vấn, sau đó xử lý từng truy vấn theo thứ tự giảm dần.

Độ phức tạp thuật toán:  $O(N \cdot \log N + Q \cdot \log Q)$

### 3.5.3. Chương trình minh hoạ

```
#include<bits/stdc++.h>
using namespace std;
int N,Q,ans[100005],cnt;
bool use[100005];
pair<int,int> a[100005],q[100005];
```

```
int main() {
    // freopen("Isolation.inp", "r", stdin);
    cin>>N>>Q;
    for(int i=1; i<=N; i++) {
        cin>>a[i].first;
        a[i].second=i;
    }
    for(int i=1; i<=Q; i++) {
        cin>>q[i].first;
        q[i].second=i;
    }
    sort(a+1,a+1+N);
    sort(q+1,q+1+Q);
    int d=N;
    for(int i=Q; i>=1; i--) {
        while(d>=1) {
            if(a[d].first>q[i].first) {
                cnt++;
                use[a[d].second]=1;
                if(use[a[d].second-1])
                    cnt--;
                if(use[a[d].second+1])
                    cnt--;
                d--;
            } else
                break;
        }
        ans[q[i].second]=cnt;
    }
    for(int i=1; i<=Q; i++)
        cout<<ans[i]<<" ";
}
```

#### 3.5.4. Test:

[https://drive.google.com/drive/folders/1ofheTovqzJrTW\\_XeCXBxQA-lH00qqCmT?usp=sharing](https://drive.google.com/drive/folders/1ofheTovqzJrTW_XeCXBxQA-lH00qqCmT?usp=sharing)

### 3.6. Bài toán 6: Hội thảo Khoa học (Seminar)

#### 3.6.1. Đề bài

Do tình hình dịch Covid-19 diễn ra phức tạp, nên lịch Hội thảo 2021 phải tổ chức tại hai địa điểm A, B khác nhau để đảm bảo giãn cách. Ban tổ chức dự kiến có  $N$  bài báo cáo có thể diễn ra tại cả hai địa điểm. Bài báo cáo thứ  $i$  dự kiến diễn ra tại A trong thời gian  $[sa_i, ea_i]$  và tại B trong thời gian  $[sb_i, eb_i]$ . Hai bài báo gọi là cắt nhau nếu chúng có điểm chung về thời gian, tính cả thời gian lúc bắt đầu và lúc kết thúc.

Ban tổ chức muốn chọn ra một số bài báo để chúng có thể được diễn ra ở tại cả địa điểm A và địa điểm B. Ban tổ chức nhận thấy có một số bài báo cáo có thể diễn ra ở A lại không thể diễn ra tại B (do bài báo cắt nhau). Trong tất cả các cách chọn bài báo cáo, hỏi có cách chọn danh sách các bài báo cáo nào

mà nó chỉ có thể diễn ra tại một trong 2 địa điểm. Nếu có thì in ra YES, ngược lại in ra NO.

Dữ liệu vào: Đọc vào từ tệp Seminar.inp

Dòng đầu tiên là số bài báo có thể được chọn để báo cáo

Dòng thứ  $i$  trong  $N$  dòng tiếp theo mô tả bài báo cáo thứ  $i$  bởi bộ 4 số tương ứng  $sa_i, ea_i, sb_i, eb_i$  (với  $1 \leq sa_i \leq ea_i \leq 10^9; 1 \leq sb_i \leq eb_i \leq 10^9$ ).

Kết quả ra: Ghi ra tệp Seminar.out

YES nếu tồn tại danh sách bài báo cáo mà không thể diễn ra tại cả A,B

Ngược lại in ra NO

Ví dụ:

Seminar.inp	Seminar.out	Giải thích
2 1 2 3 6 3 4 7 8	NO	Tất cả danh sách các bài, nếu diễn ra ở A được thì đều diễn ra ở B được
3 1 3 2 4 4 5 6 7 3 4 5 5	YES	Chọn bài báo cáo 1, 3. Nó diễn ra ở B được, nhưng không diễn ra ở A được thì thời gian kết thúc của bài 1 trùng thời gian bắt đầu của bài 3.
6 1 5 2 9 2 4 5 8 3 6 7 11 7 10 12 16 8 11 13 17 9 12 14 18	NO	Tất cả danh sách các bài, nếu diễn ra ở A được thì đều diễn ra ở B được

Subtask 1: 20% test có  $N \leq 20$

Subtask 2: 30% test có  $N \leq 5000$

Subtask 3: 50% test còn lại không có thêm ràng buộc gì.

### 3.6.2. Phân tích bài toán

Với subtask 1: Duyệt tất cả  $2^N$  khả năng của chọn danh sách bài báo cáo. Nếu tồn tại danh sách nào mà chỉ diễn ra tại một trong 2 địa điểm thì in ra YES.

Với subtask 2: Ta chỉ cần quan tâm đến cách chọn đúng 2 bài báo cáo là đủ. Nếu tồn tại nhiều bài báo cáo chỉ có thể diễn ra tại một trong 2 điểm A,B. Thì tồn tại 2 bài báo cáo không giao nhau tại A mà giao nhau tại B hoặc ngược lại là đủ. ĐPT:  $O(N^2)$

Với subtask 3: Ta duy trì một tập S quản lý các đoạn thời gian bắt đầu và kết thúc của các bài báo.

Ta sắp xếp các bài báo theo thứ tự tăng dần theo thời gian bắt đầu tại A. Xét bài báo thứ  $i$  có khoảng thời gian  $(sa_i, ea_i, sb_i, eb_i)$ . Ta tách làm 2 sự kiện:

+ Thêm vào S đoạn  $[sb_i, eb_i]$  khi gặp  $sa_i$  (thời gian bắt đầu của  $i$ ).

+ Bỏ ra khỏi S đoạn  $[sb_i, eb_i]$  khi gặp  $ea_i$  (thời gian kết thúc của  $i$ )

Với các bài báo đang xét thì nó cắt nhau tại A, nên bắt buộc  $[sb_i, eb_i]$  phải cắt tất cả các đoạn trong S. Điều này chỉ xảy ra khi  $sb_i$  nhỏ hơn thời gian bắt đầu sớm nhất của bài báo cáo đang lưu trong S, hay  $sb_i \leq e_{min}$ . Tương tự  $eb_i \geq s_{max}$ .

Chúng ta sẽ sử dụng cấu trúc dữ liệu multiset để thực hiện bài này.

Độ phức tạp:  $O(N \cdot \log(N))$

### 3.6.3. Chương trình minh họa

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 200005;
struct Time {
    int x, y1, y2, op;
    bool operator < (const Time &P) {
        if(x == P.x)
            return op > P.op;
        return x < P.x;
    }
};
vector<Time> Event;
multiset<int> cory1, cory2;
bool solve() {
    for(auto u : Event) {
        if(u.op == 1) {
            if(cory2.size() > 0) {
                auto minY2 = cory2.begin();
                auto maxY1 = cory1.rbegin();
                if(u.y1 <= *minY2 && u.y2 >= *maxY1);
            } else
                return false;
            cory1.insert(u.y1);
            cory2.insert(u.y2);
        } else {
            cory1.erase(cory1.find(u.y1));
            cory2.erase(cory2.find(u.y2));
        }
    }
    return true;
}
int n, sa[maxn], ea[maxn], sb[maxn], eb[maxn];
bool ans;
main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    // freopen("Seminar.inp", "r", stdin);
    cin >> n;
    for(int i = 1; i <= n; i++) {
        cin >> sa[i] >> ea[i] >> sb[i] >> eb[i];
        Event.push_back({sa[i], sb[i], eb[i], 1});
    }
}
```

```

        Event.push_back({ea[i], sb[i], eb[i], -1});
    }
    sort(Event.begin(), Event.end());
    ans = solve();
    Event.clear();
    for(int i = 1; i <= n; i++) {
        Event.push_back({sb[i], sa[i], ea[i], 1});
        Event.push_back({eb[i], sa[i], ea[i], -1});
    }
    sort(Event.begin(), Event.end());
    ans &= solve();
    if(ans == true)
        cout << "YES";
    else
        cout << "NO";
    return 0;
}

```

### 3.6.4. Test

[https://drive.google.com/drive/folders/1ofheTovqzJrTW\\_XeCXBxQA-IH00qqCmT?usp=sharing](https://drive.google.com/drive/folders/1ofheTovqzJrTW_XeCXBxQA-IH00qqCmT?usp=sharing)

## 3.7. Bài toán 7: Truy vấn hình vuông

### 3.7.1. Đề bài

Có  $N$  điểm trên mặt phẳng vô hạn, điểm thứ  $i$  có tọa độ  $(x_i, y_i)$  và giá trị  $c_i$ . Bạn cần chọn một hình vuông có các cạnh song song với trục tọa độ sao cho góc trái dưới và góc phải trên nằm trên đường thẳng  $y = x$ .

Điểm của cách chọn bằng tổng tất cả các điểm nằm trong và trên cạnh của hình vuông đó trừ đi độ dài 1 cạnh của hình vuông. Chú ý có thể chọn hình vuông cạnh là 0, tức là bạn chọn 1 điểm nằm trên đường thẳng  $y = x$ .

Bạn cần tính số điểm cao nhất có thể nhận được sau khi chọn một hình vuông bất kì.

Dữ liệu vào: Đọc vào từ tệp QUADQUERY.inp

Dòng đầu tiên ghi số nguyên dương  $N$  ( $1 \leq N \leq 10^5$ ) là số điểm trên mặt phẳng.

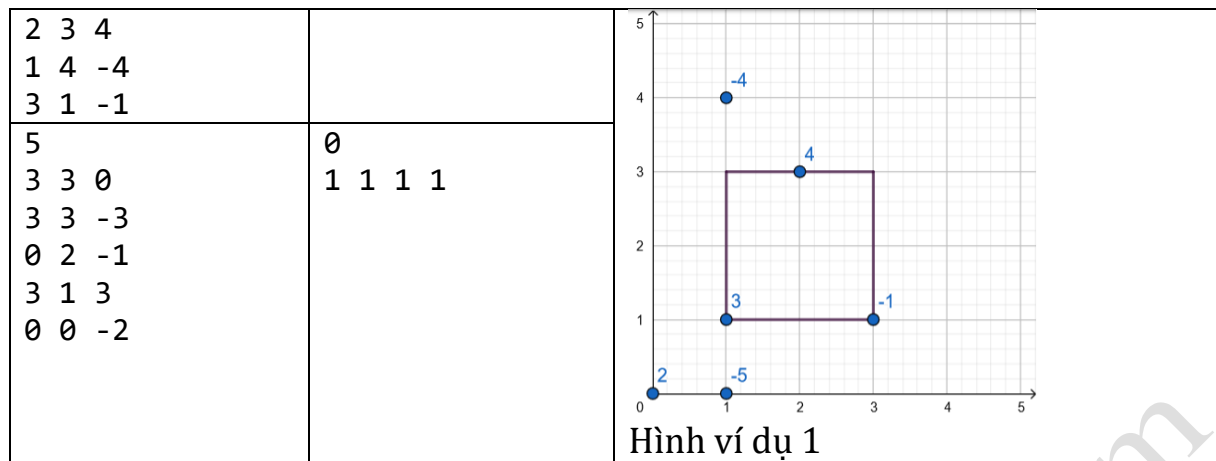
$N$  dòng sau, mỗi dòng gồm 3 số nguyên  $x_i, y_i, c_i$  ( $0 \leq x_i, y_i \leq 10^9, -10^6 \leq c_i \leq 10^6$ ) là tọa độ và giá trị của điểm thứ  $i$ .

Kết quả ra: Ghi ra tệp QUADQUERY.out

Ghi ra điểm lớn nhất có thể đạt được.

Ví dụ:

QUADQUERY.inp	QUADQUERY.out	Giải thích
6 0 0 2 1 0 -5 1 1 3	4	



Subtask 1: 30% test có  $N \leq 100$ ;

Subtask 2: 20% test có  $N \leq 1000$ ;

Subtask 3: 30% test có các toạ độ  $x_i, y_i$  nhỏ hơn  $10^5$ ;

Subtask 4: 20% test không có điều kiện gì thêm.

### 3.7.2. Phân tích bài toán

Với subtask 1: Ta sinh ra tất cả các hình vuông rồi kiểm tra. Chú ý rằng cạnh phải và cạnh trên của hình vuông phải có ít nhất một điểm nằm trên đó; cạnh trái và cạnh dưới của hình vuông phải có ít nhất một điểm nằm trên đó (vì nếu không thoả mãn 2 điều kiện này, ta luôn tìm được một hình vuông khác nhỏ hơn nhưng số điểm bị bao không thay đổi). Do đó, số hình vuông cần xét là  $O(n^2)$ . Với mỗi hình vuông, ta kiểm tra có bao nhiêu điểm nằm trong hình vuông đó rồi tính điểm. ĐPT:  $O(n^3)$ .

Với subtask 2: Tương tự như subtask 1, nhưng ta có thể đếm được số điểm nằm trong hình vuông trong  $O(1)$  (có thể tính trong lúc duyệt, hoặc tính bằng tổng cộng dồn,...). ĐPT:  $O(n^2)$ .

Với subtask 3: Nhận xét: hình vuông có góc trái dưới  $(l; l)$  góc phải trên  $(r; r)$  chứa điểm  $M(x; y)$  khi  $l \leq \min(x, y) \leq \max(x, y) \leq r$ . Bài toán đặt ra thách thức cho chúng ta làm sao để tính tổng lớn nhất của đoạn  $[l; r]$ .

Đặt  $r_{\max} = \max\{x_i, y_i\}$ . Gọi  $f(l, r)$  là giá trị lớn nhất của tất cả các đoạn đầu mút trái  $l \in [1, r]$  và đầu mút phải là  $r$ . Khi đó ta duyệt qua mỗi  $l$  giảm dần, với mỗi  $l$  cố định thì ta có kết quả:

$$res = \max\{res, \max_{r_i \in [l, r_{\max}]} (f(l, r_i) - (r_i - l))\}$$

Sử dụng sweep line để các sự kiện gặp điểm từ  $l = r_{\max}$  đến  $l = 0$ .

Mỗi khi gặp một điểm  $M(x; y)$  thì ta cần cập nhật tất cả các hình vuông có thể chứa nó từ  $r = \max(x, y)$  đến  $r_{\max}$ . Để việc cập nhật và tìm max hiệu quả, ta cần sử dụng Segment tree có lazy propagation.

Ta lại có:  $f(l, r_i) - (r_i - l) = [f(l, r_i) - r_i] + l$

Nên ban đầu ta dựng Segment tree có  $f(r, r) = -r$ . Mỗi khi tìm max thì ta nhớ cộng thêm  $l$  tại sweep line đang xét.

Độ phức tạp thuật toán:  $O(N * \log(N))$ .

Với subtask 4: Bổ sung thêm nén số.

### 3.7.3. Chương trình minh họa

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e5;
const long long INF = 1e13 + 1;
const int INFi = 1e9 + 1;
struct point {
    int x, y;
    long long cost;
    operator <(point &a) {
        return x<a.x;
    }
} a[maxn];
vector<int> Ev; ///Event = point
struct node {
    long long val, lazy;
} st[4 * maxn];
void buildST(int id, int l, int r) {
    if(l > r)
        return;
    if(l == r) {
        st[id].val = -Ev[l];///trừ độ dài phải
        return;
    }
    buildST(id * 2, l, (l + r) / 2);
    buildST(id * 2 + 1, (l + r) / 2 + 1, r);
    st[id].val = max(st[id * 2].val, st[id * 2 + 1].val);
}
void down(int id) { ///lazy propagation
    for(int i = id * 2; i <= id * 2 + 1; i++) {
        st[i].val += st[id].lazy;
        st[i].lazy += st[id].lazy;
    }
    st[id].lazy = 0;
}
void updateST(int id, int l, int r, int u, int v, int val) {
    if(v < l || r < u)
        return ;
    if(u <= l && r <= v) {
        st[id].val += 1LL * val;
        st[id].lazy += 1LL * val;
        return ;
    }
    down(id);
    updateST(id*2, l, (l+r)/2, u, v, val);
    updateST(id*2+1, (l+r)/2+1, r, u, v, val);
    st[id].val=max(st[id*2].val, st[id*2+1].val);
}
```

```
}
long long getST(int id, int l, int r, int L, int R) {
    if(R < l || r < L)
        return -INF;
    if(L <= l && r <= R)
        return st[id].val;
    down(id);
    return max(getST(id*2,l,(l+r)/2,L,R),
               getST(id*2+1,(l+r)/2+1,r,L,R));
}
int n;
void compress() {
    Ev.push_back(-1);
    Ev.push_back(INFi);
    sort(Ev.begin(), Ev.end());
    Ev.erase(unique(Ev.begin(), Ev.end()), Ev.end());
    for(int i=1; i<=n; i++) {
        a[i].x=lower_bound(Ev.begin(),Ev.end(),a[i].x)-Ev.begin();
        a[i].y=lower_bound(Ev.begin(),Ev.end(),a[i].y)-Ev.begin();
    }
}
main() {
    ios_base::sync_with_stdio(false);
    cin >> n;
    for(int i = 1; i <= n; i++) {
        cin >> a[i].x >> a[i].y >> a[i].cost;
        if(a[i].x>a[i].y)
            swap(a[i].x,a[i].y);
        Ev.push_back(a[i].x), Ev.push_back(a[i].y);
    }
    compress();///nén số
    int rmax=Ev.size()-1;
    sort(a+1,a+n+1);
    buildST(1,1,rmax);
    int id=n;
    long long ans = -1;
    for(int l=rmax;l>0;l--) {
        for(;min(a[id].x,a[id].y)==l && id>0;id--)
            updateST(1,1,rmax,max(a[id].x,a[id].y),rmax,
a[id].cost);
        ans = max(ans,getST(1,1,rmax,l,rmax)+Ev[l]);
    }
    ///cộng thêm trái
    cout << ans;
}
```

### 3.7.4. Test

[https://drive.google.com/drive/folders/1ofheTovqzJrTW\\_XeCXBxQA-lH00qqCmT?usp=sharing](https://drive.google.com/drive/folders/1ofheTovqzJrTW_XeCXBxQA-lH00qqCmT?usp=sharing)



### 3.8. Bài toán 8: Truy vấn tam giác

#### 3.8.1. Đề bài:

Cho  $N$  điểm có tọa độ nguyên  $(x_i, y_i)$  với  $i \in [1, N]$  trên mặt phẳng tọa độ  $Oxy$ . Bạn cần trả lời  $Q$  truy vấn, mỗi truy vấn cho bởi bộ 3 số  $(x, y, d)$ , hãy xác định số điểm có trong hình tam giác  $ABC$  được xác định bởi đỉnh  $A(x, y)$ ,  $B(x + d, y)$  và  $C(x, y + d)$ .

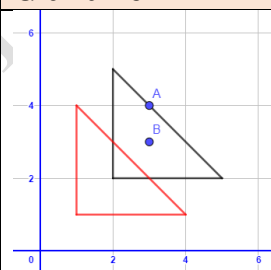
Dữ liệu vào: Đọc vào từ tệp TRIQUERY.inp

- Dòng đầu tiên ghi 2 số  $N, Q$  ( $1 \leq N, Q \leq 2 \cdot 10^5$ )
- $N$  dòng tiếp theo ghi các tọa độ của các điểm  $1 \leq x_i, y_i \leq 2 \cdot 10^5$ .
- $Q$  dòng tiếp theo, mỗi dòng mô tả một truy vấn  $1 \leq x, y, d \leq 2 \cdot 10^5$

Kết quả ra: Ghi ra tệp TRIQUERY.out

- Ghi  $Q$  số trả lời cho  $Q$  truy vấn, mỗi số trên một dòng.

Ví dụ:

TRIQUERY.inp	TRIQUERY.out	Giải thích
2 2 3 3 3 4 1 1 3 2 2 3	0 2	

Subtask 1: 50% test có  $N, Q \leq 10^3$ ;

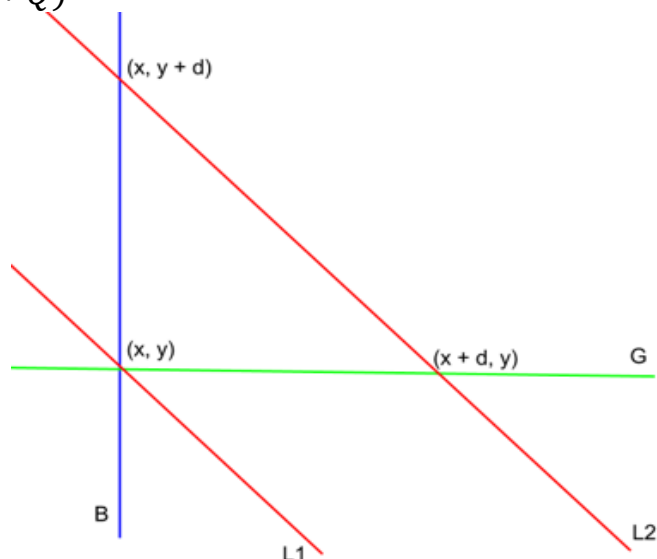
Subtask 2: 50% test có  $N, Q \leq 2 \cdot 10^5$ .

#### 3.8.2. Phân tích bài toán

Với subtask 1: Với mỗi truy vấn, duyệt qua tất cả  $N$  điểm để kiểm tra xem nó có thuộc tam giác của truy vấn. Kiểm tra mỗi điểm trong thời gian  $O(1)$ . Độ phức tạp của cách làm này:  $O(N * Q)$

Với subtask 2: Ta dùng sweep line nghiêng góc  $45^\circ$  quét từ trái qua phải như hình bên dưới. Vấn đề bây giờ là ta phải đếm số điểm nằm trong mỗi tam giác truy vấn được xác định bởi đường B, G, L2.

Gọi số điểm bên dưới L2 là Num[L2], số điểm bên trái B là Left[B], số điểm bên dưới G là Down[G], số điểm vừa bên trái



B và bên dưới G là  $\text{Num}[\text{Left}[B] \& \text{Down}[G]]$ .

Khi đó kết quả truy vấn  $(x, y, d)$  được tính theo phương pháp bao hàm-loại trừ như sau:

$$\text{Answer} = \text{Num}[L2] - \text{Left}[B-1] - \text{Down}[G-1] + \text{Num}[\text{Left}[B-1] \& \text{Down}[G-1]]$$

Trong đó:  $\text{Num}[L2]$  dễ dàng có được vì chúng ta xử lí lần lượt các điểm trên mặt phẳng theo thứ tự của sweep line.

$\text{Left}[B-1]$  là các điểm đã được duyệt mà có hoành độ nhỏ hơn B

$\text{Down}[G-1]$  là các điểm có tung độ nhỏ hơn G.

$\text{Num}[\text{Left}[B-1] \& \text{Down}[G-1]]$ : Được tính khi xử quét sweep line của thời điểm  $(x, y)$ . Giá trị này lại được tính theo phương pháp bao hàm – loại trừ khi xét sự kiện gặp điểm  $(x, y)$  của truy vấn.

Xét tại thời điểm  $(x, y)$  thì:

$$\text{Num}[\text{Left}[B-1] \& \text{Down}[G-1]] = \text{Left}[B] + \text{Down}[G] - \text{Num}[L1]$$

Ta sẽ mô hình hoá sự kiện như sau:

- Sự kiện loại 0: Gặp điểm trên mặt phẳng. Khi gặp sự kiện 0 thì ta tăng đếm số điểm dưới G và trái B.

- Sự kiện loại 1: Khi gặp điểm  $(x, y)$  của truy vấn ta tính số điểm vừa nằm bên trái B, vừa nằm bên dưới G.

- Sự kiện loại 2: Khi gặp điểm  $(x + d, y)$  ta tính số điểm thuộc truy vấn  $(x, y, d)$ .

Các sự kiện được xử lí theo thứ tự sweep line, các sweep line đều có phương trình dạng  $x + y = \text{const}$ . Sweep line được sắp xếp tăng dần theo  $x + y$ . Ưu tiên sự kiện theo thứ tự  $0 \rightarrow 1 \rightarrow 2$ .

Độ phức tạp thuật toán là  $O(N * \log(N))$ .

### 3.8.3. Chương trình minh hoạ

```
#include <bits/stdc++.h>
#define endl '\n'
#define ii pair<int,int>
using namespace std;
struct BIT {
    vector<int> bit;
    BIT(int n) {
        bit.assign(n + 1, 0);
    }
    void upd(int k, int x) {
        for(; k < int(bit.size()); k += k & -k)
            bit[k] += x;
    }
    int get(int k) {
        int ans=0;
        for(; k > 0; k -= k & -k)
```

```
        ans += bit[k];
        return ans;
    }
};

struct Event {
    int x, y, type, L, D, pos; ///Left, Down
};

bool operator<(Event e1, Event e2) {
    return (e1.x+e1.y<e2.x+e2.y) || ((e1.x+e1.y==e2.x+e2.y)
&& (e1.type<e2.type));
}

int cnt,n,q;
vector<Event> E;
map<ii,int> dem; ///dem Num[B&G] loai 1
int ans[200005];
int main() {
    // freopen("TRIQUERY.inp","r",stdin);
    ios_base::sync_with_stdio(0);
    cin>>n>>q;
    for(int i=1; i<=n; i++) {
        int x, y;
        cin>>x>>y;
        E.push_back({x,y,0,0,0,-1}); ///Loai 0
    }
    for(int i=1; i<=q; i++) {
        int x,y,d;
        cin>>x>>y>>d;
        E.push_back({x-1,y-1,1,0,0,-1});///Loai 1
        E.push_back({x+d,y,2,x,y,i}); ///Loai 2
    }
    sort(E.begin(),E.end());
    BIT x(200005),y(200005);
    for(auto e:E) { ///xu li cac su kien
        if(e.type==0) {
            y.upd(e.y,1);
            x.upd(e.x,1);
            cnt++;
        } else if(e.type==1) {
            dem[ {e.x,e.y}]=x.get(e.x)+y.get(e.y)-cnt;
        } else if(e.type==2) {
            ans[e.pos]=cnt-x.get(e.L-1)-y.get(e.D-1)+ dem[
{e.L-1,e.D-1}];
        }
    }
    for(int i=1; i<=q; i++)
        cout<<ans[i]<<endl;
}
```

### 3.8.4. Test

[https://drive.google.com/drive/folders/1ofheTovqzJrTW\\_XeCXBxQA-lH00qqCmT?usp=sharing](https://drive.google.com/drive/folders/1ofheTovqzJrTW_XeCXBxQA-lH00qqCmT?usp=sharing)

### 3.9. Bài toán 9: Quân xe trên bàn cờ

#### 3.9.1. Đề bài

Cho một bàn cờ hình chữ nhật gồm  $N$  cột và  $M$  hàng, được đặt gắn với hệ trục tọa độ  $Oxy$ , các cột được đánh số từ 1 đến  $N$  từ trái qua phải, các hàng được đánh số từ 1 đến  $M$  từ dưới lên trên. Có  $K$  quân xe và  $Q$  hình chữ nhật nhỏ bên trong bàn cờ. Một hình chữ nhật được gọi là bị phong tỏa nếu tất cả các ô trong của nó bị phong tỏa bởi các quân xe nằm trong hình chữ nhật đó.

Hãy xác định xem  $Q$  hình chữ nhật đó bị phong tỏa hay không?

Dữ liệu vào: Đọc vào từ tệp ROOK.inp

Dòng đầu tiên ghi số  $N, M, K, Q$  ( $1 \leq N, M, K, Q \leq 10^5$ ) tương ứng là số cột, số hàng, số quân xe, số hình chữ nhật.




$K$  dòng tiếp theo ghi tọa độ các quân xe trên bàn cờ, dòng thứ  $i$  ghi 2 số nguyên dương  $x_i, y_i$  là tọa độ quân xe thứ  $i$  trên bàn cờ,  $1 \leq x_i \leq N, 1 \leq y_i \leq M$ .

$Q$  dòng tiếp theo mỗi dòng ghi 4 số nguyên dương  $x_1, y_1, x_2, y_2$  là tọa độ góc trái dưới và phải trên của hình chữ nhật. Dữ liệu đảm bảo:  $1 \leq x_1 \leq x_2 \leq N; 1 \leq y_1 \leq y_2 \leq M$ ;

Kết quả ra: Ghi ra tệp ROOK.out

Ghi ra  $Q$  dòng là tương ứng  $Q$  câu trả lời kiểm tra xem hình chữ nhật nhỏ có bị phong tỏa hay không. Nếu có ghi ra 1, ngược lại ghi ra 0.

Ví dụ:

ROOK.inp	ROOK.out	Giải thích				
4 3 3 3	1	3				Test thứ 3 thì ô (1;2) không bị phong tỏa
1 1	1	2				
3 2	0	1				
2 3						
2 3 2 3						
2 1 3 3						
1 2 2 3						
		$y \backslash x$	1	2	3	4

Subtask 1: 50% test có  $1 \leq N, M, Q \leq 10^3$ ;

Subtask 2: 50% test có  $10^3 < N, M, Q \leq 10^5$ .

#### 3.9.2. Phân tích bài toán

Với subtask 1: Một hình chữ nhật không bị phong tỏa nếu tồn tại một ô không bị phong tỏa, tức là tồn tại một hàng và một cột không có quân xe nào.

Để kiểm tra điều trên, ta thấy phức tạp hơn là ta kiểm tra điều ngược lại là một hình chữ nhật bị phong tỏa thì một trong hai điều kiện sau đây xảy ra:

- Hàng nào cũng có ít nhất một quân xe

- Cột nào cũng có ít nhất một quân xe

Việc kiểm tra một trong hai điều kiện trên được thực hiện hoàn toàn độc lập. Có thể dễ dàng đếm nhanh nếu có cộng dồn và mỗi truy vấn sẽ xử lý trong  $O(N)$ . Độ phức tạp thuật toán là  $O(N * Q)$

Với subtask 2: Khi kích thước bàn cờ lớn hơn, việc cộng dồn là không thể. Ta phải sửa đổi cách tiếp cận bài toán.

Xét một hình chữ nhật nằm từ hàng  $a$  đến hàng  $b$ , cột bên trái là cột  $x$ , cột bên phải đang xét là  $y$ . Gọi  $Last[i]$  là vị trí gần nhất có quân xe trên dòng  $i$  khi xét đến biên bên phải  $y$ . Khi đó nếu hình chữ nhật bị phong tỏa thì:

$$\min_{a \leq i \leq b} (Last[i]) \geq x \quad (*)$$

Nếu điều kiện trên không xảy ra, tức là có dòng không bị phong tỏa, khi đó ta xét đến kiểm tra cột bằng cách tương tự.

Ta có thể dễ dàng kiểm tra điều kiện (\*) trong  $O(\log(N))$  bằng sử dụng Segment tree.

Mô hình hoá các sự kiện khi xét dòng như sau:

- Sự kiện loại 0: Gặp quân xe tại vị trí  $(x; y)$ , cập nhật lại  $x$  vào Segment tree quản lý min của đoạn tại nút quản lý đoạn  $[y, y]$ .

- Sự kiện loại 1: Gặp biên phải của một hình chữ nhật có tung độ  $y_1, y_2$ . Kiểm tra xem điều kiện (\*) có thoả mãn hay không.

- Sweep line quét từ trái qua phải theo hoành độ tăng dần. Mỗi sự kiện được xử lý trong  $O(\log(N))$ . Tương tự làm một lần nữa quét từ dưới lên trên.

Có tất cả  $Q + K$  sự kiện, và ta phải quét ngang và dọc. Độ phức tạp thuật toán là:  $O((Q + K) * \log(M * N))$

### 3.9.3. Chương trình minh họa

```
#include <bits/stdc++.h>
using namespace std;
const int N = 2e5+5;
vector<int> xx[N], yy[N], v[N], u[N];
int x1[N], x2[N], y2[N], y[N];
int ans[N], tree[N*2], c, L, R;
void update(int pos, int l, int r, int v) {
    if(c < l || r < c)
        return;
    if(l == r && l == c) {
        tree[pos] = v;
        return;
    }
    int mid = (l+r)/2;
    update(pos<<1, l, mid, v);
    update(pos<<1|1, mid+1, r, v);
    tree[pos] = min(tree[pos<<1], tree[pos<<1|1]);
}
```

```
int get(int pos,int l,int r) {
    if(r<L || R<l)
        return 1e9;
    int mid=(l+r)/2;
    if(L<=l && r<=R)
        return tree[pos];
    return min(get(pos<<1,l,mid),get(pos<<1|1,mid+1,r));
}
int main() { //ROOK
    int n,m,k,q,a,b,x;
    scanf("%d%d%d%d",&n,&m,&k,&q);
    for(int i=0; i<k; ++i) {
        scanf("%d%d",&a,&b);
        xx[a].push_back(b);
        yy[b].push_back(a);
    }
    for(int i=0; i<q; ++i) {
        scanf("%d%d%d%d",&x1[i],&y[i],&x2[i],&y2[i]);
        u[x2[i]].push_back(i);
        v[y2[i]].push_back(i);
    }
    for(int i=1; i<=n; ++i) { //quet doc
        for(int j=0; j<xx[i].size(); ++j)
            c= xx[i][j],update(1,1,m,i);
        for(int j=0; j<u[i].size(); ++j) {
            x= u[i][j];
            L= y[x],R= y2[x];
            if(get(1,1,m)>=x1[x])
                ans[x]|=1;
        }
    }
    memset(tree,0,sizeof(tree));
    for(int i=1; i<=m; ++i) { //quet ngang
        for(int j=0; j<yy[i].size(); ++j)
            c= yy[i][j],update(1,1,n,i);
        for(int j=0; j<v[i].size(); ++j) {
            x= v[i][j];
            L= x1[x],R= x2[x];
            if(get(1,1,n)>=y[x])
                ans[x]|=1;
        }
    }
    for(int i=0; i<q; ++i)
        if(ans[i])
            printf("1\n");
        else
            printf("0\n");
}
```

### 3.9.4. Test

[https://drive.google.com/drive/folders/1ofheTovqzJrTW\\_XeCXBxQA-lH00qqCmT?usp=sharing](https://drive.google.com/drive/folders/1ofheTovqzJrTW_XeCXBxQA-lH00qqCmT?usp=sharing)

### 3.10. Bài toán 10: Đường tròn Manhattan

#### 3.10.1. Đề bài

Định nghĩa đường tròn Manhattan bán kính  $R$ , tâm  $I$ . Là tập hợp các điểm trên mặt phẳng có khoảng cách Manhattan đến tâm  $I$  nhỏ hơn hoặc bằng  $R$ . Trong đó khoảng cách Manhattan giữa hai điểm có tọa độ  $(x_0, y_0), (x_1, y_1)$  được tính bằng:  $|x_0 - x_1| + |y_0 - y_1|$ .

Cho  $N$  điểm có tọa độ nguyên trên mặt phẳng  $Oxy$  và bán kính  $R$ . Hãy xác định số điểm lớn nhất nằm trong đường tròn Manhattan bán kính  $R$ .

Dữ liệu vào: Đọc vào từ tệp Circle.inp

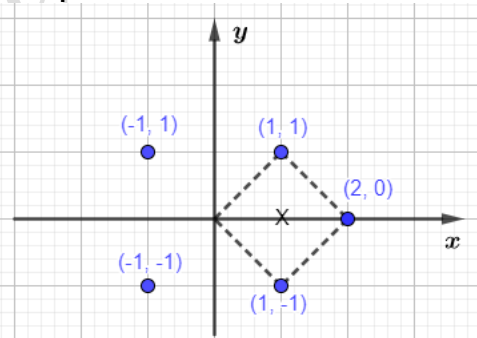
Dòng đầu tiên ghi 2 số nguyên dương  $N, R$  ( $1 \leq N \leq 10^5, 1 \leq R \leq 10^5$ ).

$N$  dòng tiếp theo ghi, mỗi dòng ghi 2 số nguyên  $x_i, y_i$  ( $-10^6 \leq x_i, y_i \leq 10^6$ ) là tọa độ của điểm thứ  $i$ .

Kết quả ra: Ghi ra tệp Circle.out

Ghi một số duy nhất là số điểm nằm trong đường tròn Manhattan có bán kính  $R$ .

Ví dụ:

Circle.inp	Circle.out	Giải thích
5 1 1 1 1 -1 -1 1 -1 -1 2 0	3	Ví dụ 1 có 3 điểm. 
5 2 1 1 1 -1 -1 1 -1 -1 2 0	5	

Subtask 1: 50% test có  $1 \leq N, R \leq 10^3, |x_i| \leq 10^3, |y_i| \leq 10^3$ .

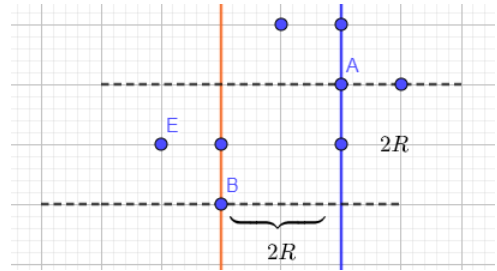
Subtask 2: 50% test còn lại không có giới hạn thêm.

#### 3.10.2. Phân tích bài toán

Nhận xét thấy các điểm nằm trong cùng đường tròn Manhattan bán kính  $R$  thì đều nằm trong hình vuông có cạnh là  $R\sqrt{2}$ , các hình vuông này tạo với trục  $Ox$  hoặc  $Oy$  góc  $45^\circ$ . Nên nếu ta dùng phép biến hình là phép quay quanh gốc tọa độ đi  $45^\circ$  và vị tự tâm  $O$  với tỉ số  $\sqrt{2}$  thì ta sẽ biến một điểm có tọa độ  $(x; y)$  thành điểm có tọa độ  $(x - y; x + y)$ , khi đó đường tròn Manhattan có bán kính là  $R\sqrt{2}$ , đồng thời hình vuông có cạnh  $R\sqrt{2}$  sẽ biến thành hình vuông có cạnh  $2 * R$ .

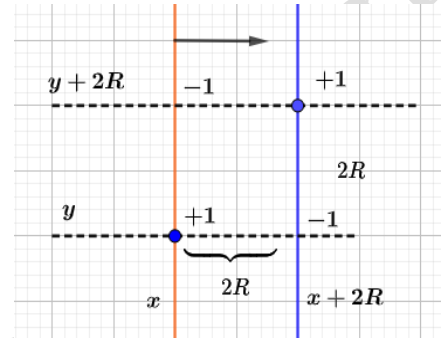


**Với subtask 1:** Việc trên dẫn đến làm cho bài toán dễ dàng xử lý hơn. Bài toán của chúng ta sẽ trở thành tìm số điểm nhiều nhất có thể nằm trong hình vuông cạnh  $2 * R$ , các cạnh đều song song với trục tọa độ.



Sau phép biến hình như trên, nếu sử dụng kỹ thuật cộng dồn, chúng ta sẽ dễ dàng đếm được số điểm nằm trong một hình vuông cạnh  $2 * R$  bất kì trong  $O(1)$ . Khi đó độ phức tạp thuật toán này là  $O(\max\{|x_i| * |y_i|\})$

**Với subtask 2:** Khi tọa độ lớn hơn, ta không thể dùng cộng dồn nữa, mà khi đó ta sẽ mô hình hoá các điểm như các sự kiện và sweep line quét từ trái qua phải.



Mỗi khi gặp một điểm  $(x; y)$ , thì điểm đó tác động lên hình vuông có góc trái dưới là  $(x; y)$  góc phải trên là  $(x + 2R; y + 2R)$ . Vậy ta sẽ xử lý bài toán là tìm vị trí trên mặt phẳng được bao phủ bởi nhiều hình vuông nhất.

Bài toán đếm như sau: gặp cạnh bên trái tại hoành độ  $x$ , ta cộng toàn bộ đoạn có tung độ  $y$  đến  $y + 2R$  lên 1, khi sang đến cạnh bên phải tại hoành độ  $x + 2R + 1$  ta lại trừ toàn bộ đoạn có tung độ từ  $y$  đến  $y + 2R$  đi 1. Khi xử lý xong các sự kiện trên một sweep line thì ta lại cập nhật lại kết quả.

Độ phức tạp thuật toán là  $O(4 * N * \log(\max\{|x_i|, |y_i|\}))$

Bài toán có thể mở rộng thêm không gian về tọa độ, số điểm, tuy vậy cách xử lý vẫn như trên, chỉ kết hợp thêm kỹ thuật khác như nén số.

### 3.10.3 Chương trình minh hoạ

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 6e5+10; ///max sweep line
vector<pair<int,int>> event[maxn];
const int n = 1<<19;
int ST[2*n], value[2*n];
void add(int i, int v) {
    for(i+=n; i>1; i>>=1) {
        if(i&1)
            value[i-1] += v;
        ST[i>>1] = max(ST[i]+value[i], ST[i^1]+value[i^1]);
    }
}
int main() {
    // freopen("Circle.inp", "r", stdin);
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n, r;
```



```
cin >> n >> r;
for(int k = 0; k < n; k++) {
    int x, y;
    cin >> x >> y;
    ///xoay 45, va vi tu sqrt(2)
    int i = x-y+2e5;
    int j = x+y+2e5;
    event[i].push_back({j,-1});
    event[i].push_back({j+r*2+1,1});
    event[i+r*2+1].push_back({j,1});
    event[i+r*2+1].push_back({j+r*2+1,-1});
}
int ans = 0;
for(int i = 0; i < maxn; i++) {
    for(auto p : event[i]) ///Xet sweep line i
        add(p.first, p.second);
    ans = max(ans,ST[1]);///cap nhat lai kq
}
cout << ans << endl;
}
```

### 3.10.4. Test

[https://drive.google.com/drive/folders/1ofheTovqzJrTW\\_XeCXBxQA-IH00qqCmT?usp=sharing](https://drive.google.com/drive/folders/1ofheTovqzJrTW_XeCXBxQA-IH00qqCmT?usp=sharing)

### 3.11. Bài toán 11: Toà nhà (Building - VOI 2020)

#### 3.11.1. Đề bài

Tóm tắt: Có  $N$  hình chữ nhật trong mặt phẳng tọa độ Oxy, các hình chữ nhật có thể chạm vào nhau nhưng không đè lên nhau. Ta tạo một đồ thị vô hướng, với các đỉnh là các hình chữ nhật, 2 đỉnh có cạnh nối khi và chỉ khi 2 hình chữ nhật tương ứng chạm nhau. Trong đồ thị mới có thể có một số cầu, khi xóa cây cầu đó đi thì một thành phần liên thông sẽ bị tách ra làm 2, gọi số lượng đỉnh 2 bên lần lượt là  $A$  và  $B$ , ta cần tìm  $|A - B|$  nhỏ nhất.

Dữ liệu vào: Đọc vào từ tệp BUILDING.inp

Dòng đầu chứa số lượng toà nhà  $N$  ( $1 \leq N \leq 10^5$ )

$N$  dòng tiếp theo, mỗi dòng ghi 4 số  $x, y, p, q$  ( $1 \leq x \leq p \leq 10^9, 1 \leq q \leq y \leq 10^9$ ) là tọa độ góc trái trên và phải dưới của hình chữ nhật. Dữ liệu cho đảm bảo các hình chữ nhật không đè lên nhau, nhưng có thể tiếp xúc nhau.

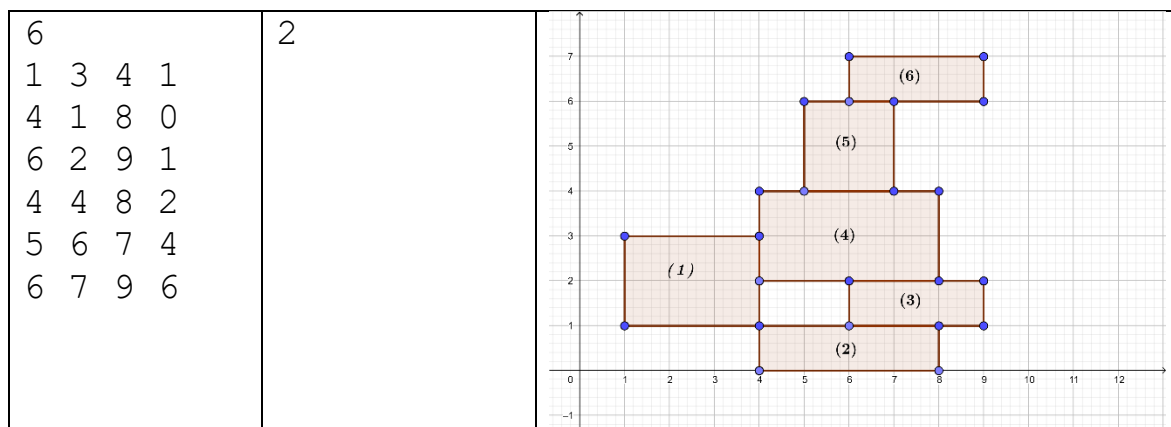
Kết quả ra: Ghi ra tệp BUILDING.out

Ghi ra chênh lệch nhỏ nhất tìm được, nếu không có lối đi nào như vậy thì in ra -1.

Ràng buộc: 30% test có  $N \leq 10^3$ ; 30% test có  $N \leq 10^5$ , tọa độ các hình vuông không quá  $10^3$ ; 40% test còn lại không có thêm giới hạn.

Ví dụ:

BUILDING.inp	BUILDING.out	Giải thích
--------------	--------------	------------



Xóa cầu giữa 4-5: khi đó chênh lệch giữa 2 vùng là  $|4 - 2| = 2$

Xoá cầu giữa 5-6: khi đó chênh lệch giữa 2 vùng là  $|5 - 1| = 4$

### 3.11.2. Phân tích bài toán

Nhận xét:

- Việc xây dựng đồ thị là việc khó khăn và mất nhiều thời gian hơn. Trong khi đó việc tìm  $|A - B|$  là khá cơ bản.
- Đồ thị dựng được là đơn đồ thị, vô hướng.
- Đồ thị dựng được là đồ thị phẳng, nên số cạnh không quá  $3 * N$  (nhận xét này rất quan trọng).
- Một toạ độ có không quá 4 hình vuông chứa nó.

Subtask 1: Duyệt trâu toàn bộ các hình chữ nhật để tạo đồ thị. Nếu hình chữ nhật nào tiếp xúc nhau thì thêm cạnh đồ thị. Sau đó duyệt đồ thị, tìm cầu và tính chênh lệch số đỉnh giữa 2 bên. Độ phức tạp  $O(N^2)$ .

Subtask 2: Số đỉnh nhiều, tuy vậy kích thước toạ độ lại không lớn, nên tại mỗi toạ độ, ta sẽ xác định được xem có bao nhiêu hình vuông chứa nó, từ đó chúng ta dựng cạnh. Để đánh dấu mỗi hình chữ nhật chứa các toạ độ nào, ta dùng kĩ thuật tổng cộng dồn 2D, nhưng giờ là mỗi vị trí là set, chứ không còn là cộng dồn 2D.

Subtask 3: Khi toạ độ lớn, ta cần sửa đổi một chút cách làm trên. Xử lí bằng sweep line như sau:

Xét bài toán có  $N$  đoạn thẳng đặt trên trục  $Ox$ . Hãy xác định các cặp đoạn thẳng có điểm chung.

Mỗi đoạn ta sẽ tách thành 2 điểm, điểm mở và điểm đóng của đoạn

Mỗi khi gặp điểm mở, ta đưa nó vào set, khi gặp điểm đóng thì toàn bộ các điểm trong set có điểm chung với đoạn có điểm đóng đang xét. Sau đó bỏ điểm mở của nó khỏi set.

Với cách làm như trên ta sẽ xử lí bài toán ban đầu bằng cách tách mỗi hình chữ nhật thành 4 đoạn thẳng tương ứng 4 cạnh. Khi 2 hình chữ nhật có điểm chung thì chúng sẽ có điểm chung trên cạnh song song với  $Ox$ , hoặc song

song với  $Oy$ . Do vậy ta dùng sweep line quét theo trục  $Ox$  và cả  $Oy$  để tạo được các cạnh của đồ thị.

Độ phức tạp thuật toán trên là  $O(N * \log(N))$ , trong đó sắp xếp các cạnh tăng dần theo toạ độ là  $O(N * \log(N))$ , xử lý sweep line là  $O(N)$ , tính toán chênh lệch giữa 2 thành phần khi loại bỏ cạnh cầu là  $O(N)$ .

### 3.11.3. Chương trình minh hoạ

```
#include <bits/stdc++.h>
using namespace std;
const int maxn=1e5 + 5;
struct TPoint {///rectangle
    int x,y,u,v;
} rec[maxn];
struct TData {
    int val,typ,id;///typ=0: open; typ=1: close
    bool operator<(TData &other) {
        return val<other.val||(val==other.val&&typ<other.typ);
    }
};
int n;
vector<TData> adjX[maxn << 1],adjY[maxn << 1];
set<int> adj[maxn];
int res;
int num[maxn],low[maxn],cnt,nChild[maxn];
vector<int> br;
void DFS(int u,int p=-1) {///Find bridge
    num[u]=low[u]=++cnt;
    nChild[u]=1;
    for(int v : adj[u]) {
        if(v==p)
            continue;
        if(num[v])
            low[u]=min(low[u],num[v]);
        else {
            DFS(v,u);
            nChild[u]+=nChild[v];
            low[u]=min(low[u],low[v]);
            if(low[v] >= num[v]) {
                br.push_back(v);
            }
        }
    }
}
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin>>n;
    vector<int> X,Y;
    for(int i=1; i <= n; ++i) {
        int x,y,u,v;
        cin>>x>>y>>u>>v;
        swap(y,v);
        assert(x<u);
        assert(y<v);
        rec[i]= {x,y,u,v};
        X.push_back(x),X.push_back(u);///Vertical edge
        Y.push_back(y),Y.push_back(v);///Horizontal edge
    }
    sort(X.begin(),X.end()), sort(Y.begin(),Y.end());
    X.erase(unique(X.begin(),X.end()),X.end());
    Y.erase(unique(Y.begin(),Y.end()),Y.end());
    for(int i=1; i <= n; ++i) {
        rec[i].x=lower_bound(X.begin(),X.end(),rec[i].x)-X.begin();
        rec[i].u=lower_bound(X.begin(),X.end(),rec[i].u)-X.begin();
    }
}
```

```
rec[i].y=lower_bound(Y.begin(),Y.end(),rec[i].y)-Y.begin();
rec[i].v=lower_bound(Y.begin(),Y.end(),rec[i].v)-Y.begin();
adjX[rec[i].x].push_back({rec[i].y,-1,i}); ///y : close
adjX[rec[i].x].push_back({rec[i].v,1,i}); ///v: open
adjX[rec[i].u].push_back({rec[i].y,-1,i});
adjX[rec[i].u].push_back({rec[i].v,1,i});
adjY[rec[i].y].push_back({rec[i].x,-1,i}); ///x: close
adjY[rec[i].y].push_back({rec[i].u,1,i}); ///u: open
adjY[rec[i].v].push_back({rec[i].x,-1,i});
adjY[rec[i].v].push_back({rec[i].u,1,i});
}
vector<pair<int,int>>E; ///danh sach canh
for(int i=0; i<X.size(); ++i) {///tao ds canh
    set<int> myset;
    sort(adjX[i].begin(),adjX[i].end());
    for(auto &cur : adjX[i]) {
        if(cur.typ==1)
            myset.erase(cur.id);
        else {
            for(int id : myset)
                E.push_back({min(id,cur.id),max(id,cur.id)});
            myset.insert(cur.id);
        }
    }
}
for(int i=0; i<Y.size(); ++i) {///tao ds canh
    set<int> myset;
    sort(adjY[i].begin(),adjY[i].end());
    for(auto &cur : adjY[i]) {
        if(cur.typ==1)
            myset.erase(cur.id);
        else {
            for(int id : myset)
                E.push_back({min(id,cur.id),max(id,cur.id)});
            myset.insert(cur.id);
        }
    }
}
sort(E.begin(),E.end());
E.erase(unique(E.begin(),E.end()),E.end());
for(auto x : E) { ///tao danh sach ke
    int u=x.first,v=x.second;
    adj[u].insert(v);
    adj[v].insert(u);
}
res=1e9;
for(int i=1; i <= n; ++i)
    if(num[i]==0) {
        br.clear();
        DFS(i);
        for(int v : br)
            res=min(res,abs(nChild[v]-(nChild[i]-nChild[v])));
    }
if(res>n)
    res=-1;
cout << res;
return 0;
}
```

#### 3.11.4. Test

[https://drive.google.com/drive/folders/1ofheTovqzJrTW\\_XeCXBxQA-IHO0qqCmT?usp=sharing](https://drive.google.com/drive/folders/1ofheTovqzJrTW_XeCXBxQA-IHO0qqCmT?usp=sharing)

### 3.12. Bài toán 12: Kẹo ngọt (CANDY)

#### 3.12.1. Đề bài

Nhân dịp Trung thu đặc biệt năm 2021, đất nước của CANDY đã lên kế hoạch phát kẹo cho các em bé chăm chỉ trong học tập. Có  $10^9$  em bé được đánh số từ 1 đến  $10^9$ . Họ đã lập ra  $N$  kế hoạch phát kẹo cho các em bé, mỗi kế hoạch sẽ phát kẹo cho các em bé có chỉ số từ  $L$  đến  $R$  một cái kẹo. Nhưng do ăn kẹo nhiều cũng không tốt, nên các em bé quyết định chỉ ăn một cái, còn lại sẽ cất đi, nếu số kẹo cất đi là số chẵn thì em bé sẽ vui. Biết mỗi em bé được nhận không nhiều hơn 8 cái.

Hỏi CANDY có thể mang niềm vui đến cho nhiều nhất bao nhiêu em bé, nếu CANDY được phép chọn các kế hoạch phát kẹo.

Dữ liệu vào: Đọc vào từ tệp CANDY.inp

Dòng đầu tiên ghi số  $N$  ( $1 \leq N \leq 10^5$ ) là số kế hoạch phát kẹo

Dòng thứ  $i$  trong  $N$  dòng tiếp theo ghi  $L_i, R_i$  ( $1 \leq L_i \leq R_i \leq 10^9$ ) là đoạn chỉ số của các em bé nhận được 1 kẹo trong kế hoạch đó.

Kết quả ra: Ghi ra tệp CANDY.out

Số em bé nhiều nhất sẽ có niềm vui.

Ví dụ:

CANDY.inp	CANDY.out	Giải thích
3	4	Sau khi chọn kế hoạch 1 và 3 thì số kẹo các em bé nhận được là: 1, 1, 2, 1, 1
1 3		Số em bé vui là: 4, các em nhận được 1 kẹo
2 4		
3 5		

Subtask 1: 30% test có  $N \leq 20$ ;

Subtask 2: 70% test còn lại có  $N \leq 10^5$ .

#### 3.12.2. Phân tích bài toán

Với subtask 1: Do có  $N$  phương án, nên chúng ta sẽ có trạng thái chọn hoặc không chọn. Ta duyệt toàn bộ  $2^N$  phương án, sau đó đếm số em bé nhận được số lẻ kẹo bằng sweep line như các bài cơ bản phía trên. ĐPT:  $O(2^N)$

Với subtask 2: Kết hợp sweep line và quy hoạch động trạng thái.

Tách mỗi đoạn  $[L_i, R_i]$  thành 2 sự kiện là mở đoạn tại  $L_i$  và đóng đoạn tại  $R_i$ . Khi gặp sự kiện mở đoạn  $\{L_i; i\}$  thì nếu ta chọn đoạn  $i$  thì cần cập nhật trạng thái bit còn trống để giúp kiểm tra số kẹo là chẵn hay lẻ. Việc cập nhật thực hiện trên bất kì bit nào đang không bị chiếm.

Khi gặp sự kiện đóng đoạn  $\{R_i; -i\}$  thì ta tính toán số em bé có số lẻ kẹo dựa sự kiện ngay trước nó, sau đó huỷ trạng thái bit mà nó đang giữ.

Do mỗi em bé chỉ nhận được tối đa 8 kẹo, nên mask là  $2^8$ .

\*) Nếu chọn đoạn  $i$  thì  $dp[i+1][new\_mask]$  được cập nhật qua  $dp[i][mask]$ .

\*) Nếu không chọn đoạn  $i$  thì  $dp[i+1][mask]$  cập nhật dựa trên  $dp[i][mask]$ .

### 3.12.3. Chương trình minh họa

```
#include <bits/stdc++.h>
using namespace std;
#define ff first
#define ss second
#define pii pair<int,int>
#define pb push_back
const int N = 2e5 + 5;
int n, x;
int dp[N][(1<<8)],
    g2[N]; ///danh dau vi tri bit ma su kien i dang chiem
vector< pii > listt; ///Luu cac su kien mo, dong doan
int get(int id, int mask, vector<int> &avail) {
    if(id == x)
        return 0;
    int &ret = dp[id][mask];
    if(ret != -1) ///độ quy có nhớ
        return ret;
    int p = 0,
        cur = listt[id].ff, /// vị trí sweep line đang xét
        segn = listt[id].ss, /// Thông tin id thuộc đoạn nào
        a1 = 0, a2 = 0; ///a1 là chọn, a2 là không chọn
    if(id == 0)
        p = 1;
    else
        p = listt[id-1].ff; /// sweep line ngay trước nó
    int x = __builtin_popcount(mask);
    ///Dem so bit 1 voi mask hien tai
    if(segn > 0) { /// Neu la mo doan (open)
        int y = avail.back(); ///lay mot bit trống
        avail.pop_back();
        g2[ segn ] = y; /// Luu lai bit ma id chiem
        /// Neu lay suu kien id
        int new_mask = (mask|(1<<y));
        if(x&1)
            a1 = (cur - p) + get(id+1, new_mask, avail);
        else
            a1 = get(id+1, new_mask, avail);
        /// Neu khong lay su kien id
        if(x&1)
            a2 = (cur - p) + get(id+1, mask, avail);
        else
            a2 = get(id+1, mask, avail);
        avail.pb(y);
    } /// khi xu li xong thi dua bit ma segn dang giu tro lai vail
    } else { /// Neu la dong doan
        segn *= -1;
```

```
int y = g2[segn]; /
//vi tri bit ma su kien segn dang chiem la y
avail.pb(y);      ///giai phong bit y
if((mask&(1<<y))) { ///Nếu đoạn được đưa vào mask
    int new_mask = (mask^(1<<y)); /// tắt bit y
    if(x&1) ///nếu số kẹo là lẻ
        a1 = (cur - p) + get(id+1, new_mask, avail);
///tăng số em bé
    else
        a1 = get(id+1, new_mask, avail);
} else {
///Nếu không được đưa vào mask thì không cần tắt bit
if(x&1)
    a1 = (cur - p) + get(id+1, mask, avail);
else
    a1 = get(id+1, mask, avail);
}
avail.pop_back();
}
return ret = max(a1, a2);
}
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int k, ans;
    cin >> n;
    for(int i=1; i<=n; i++) {
        int l, r;
        cin >> l >> r;
        listt.pb({l, i});          /// đầu đoạn
        listt.pb({r+1, -i});       /// cuối đoạn
    }
    sort(listt.begin(), listt.end());
    x = listt.size();              /// tổng số sự kiện
    vector<int> avail(8); /// Đánh dấu bit còn trống ban đầu
    for(int i=0; i<8; i++)
        avail[i] = i; /// vector
    memset(dp, -1, sizeof dp);
    cout << get(0, 0, avail);
}
```

### 3.12.4. Test

[https://drive.google.com/drive/folders/1ofheTovqzJrTW\\_XeCXBxQA-IH00qqCmT?usp=sharing](https://drive.google.com/drive/folders/1ofheTovqzJrTW_XeCXBxQA-IH00qqCmT?usp=sharing)

## 3.13. Bài toán 13: Robot AI

### 3.13.1. Đề bài

Sau khi hoàn thành nhiệm vụ cuối cùng về sweep line, NASA quyết định đưa N robot đi thám hiểm một hành tinh X. Khi thám hiểm các robot đồng loạt xuất phát, di chuyển theo các đường thẳng song song, vận tốc như nhau. Mỗi

robot được mô tả bởi bộ 3 số  $x_i, r_i, q_i$  là vị trí trên trục  $Ox$ , phạm vi chúng có thể quan sát, và độ thông minh (IQ) của robot. Các robot chỉ giao tiếp với nhau khi cả 2 cùng nhìn thấy nhau và có độ chênh lệch về IQ không quá  $K$ .

Hỏi có bao nhiêu cặp robot có thể giao tiếp với nhau trong chuyển thám hiểm hành tinh  $X$  này?

Dữ liệu vào: ROBOT.inp

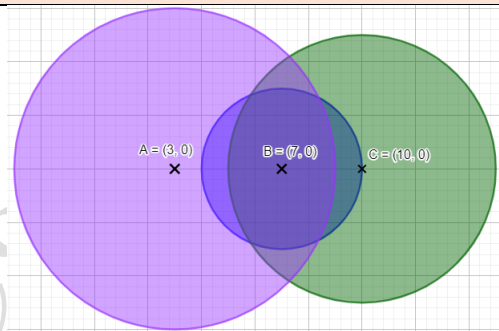
Dòng đầu tiên ghi 2 số  $N, K$  ( $1 \leq N \leq 10^5; 0 \leq K \leq 20$ ;

$N$  dòng tiếp theo mô tả thông tin của robot, dòng thứ  $i$  trong đó ghi 3 số  $x_i, r_i, q_i$  ( $0 \leq x_i, r_i, q_i \leq 10^9$ )

Kết quả ra: ROBOT.out

Ví dụ:

ROBOT.inp	ROBOT.out	Giải thích
3 2 3 6 1 7 3 10 10 5 8	1	Chỉ có robot 1 và robot 3 vừa nhìn thấy nhau, đồng thời chênh lệch IQ không quá $K=2$



Subtask 1: 50% test có  $N \leq 10^3$ ;

Subtask 2: 50% test còn lại không có thêm ràng buộc nào.

### 3.13.2. Phân tích bài toán

Với subtask 1: Điều kiện để robot  $i, j$  giao tiếp được với nhau thì  $\min(r_i, r_j) \geq |x_i - x_j|$  và  $|q_i - q_j| \leq K$ . Duyệt trâu toàn bộ các cặp điểm sau đó kiểm tra điều kiện đề bài. Độ phức tạp  $O(N^2)$ .

Với subtask 2: Với mỗi robot ta mô hình hoá các sự kiện như sau

- Sự kiện loại 0: Gặp vị trí  $x_i - r_i$ , lưu thông tin robot tại vị trí  $x_i$
- Sự kiện loại 2: Gặp vị trí  $x_i + r_i$ , xoá thông tin robot tại vị trí  $x_i$
- Sự kiện loại 1: Gặp vị trí  $x_i$  thì đếm xem có bao nhiêu robot trong phạm vi  $[x_i - r_i, x_i + r_i]$  đang có.

\* Vấn đề 1: Xác định tất cả các cặp robot nhìn thấy nhau

Dựa trên mô hình hoá các sự kiện như trên, sử dụng cấu trúc dữ liệu segment tree để giải bài toán đếm là khá dễ dàng.

\* Vấn đề 2: Xác định các robot nhìn thấy nhau mà giao tiếp với nhau

Do chênh lệch IQ của chúng không được quá  $K$ , mà  $K \leq 20$  nên với mỗi robot ta cập nhật thông tin lên các segment tree có IQ tương ứng khi gặp sự



kiện loại 0, loại 2. Khi gặp sự kiện loại 1 thì đếm các vị trí trên các cây có IQ chênh lệch không quá K.

Tuy vậy, vấn đề đặt ra là giới hạn IQ quá lớn, không thể lưu trữ hết các cây IQ được. Có thể xử lý bằng nén IQ, chỉ lưu các thông tin cần thiết. Đây là cách làm cần thiết, rất hay tránh việc tràn bộ nhớ.

Độ phức tạp thuật toán là:  $O(N * \log(N))$

**Cách xử lý subtask 2 khác:** Mỗi robot sẽ bao phủ một đường tròn có bán kính  $r$ . Nên robot có bán kính nhỏ hơn nếu bao phủ robot có bán kính lớn hơn thì mặc định chiều ngược lại đúng. Nên ta sẽ xét các robot có bán kính bao phủ giảm dần, sau đó cập nhật như cách làm trên.

Độ phức tạp thuật toán:  $O(N * \log(X_{max}))$ .

### 3.13.3. Chương trình minh họa

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5,M=5e6+5,inf=1e9;//Xmax=inf
int n,k,tt;
long long ans;
map<int,int>rt; ///root tree
struct qq {
    int x,r,q;
    bool operator <(qq &a) {
        return r>a.r;
    }
} a[N];
struct st {
    int ls,rs,s;
} t[M]; /// tree, chỉ lưu các node cần thiết
bool cmp(qq x,qq y) {
    return x.r>y.r;
}
void upd(int &x,int l,int r,int p) {
    if(!x) ///chỉ lưu node nút cần thiết
        x=++tt;
    t[x].s++;
    if(l==r)
        return;
    int mid=l+r>>1;
    if(p<=mid)
        upd(t[x].ls,l,mid,p);
    else
        upd(t[x].rs,mid+1,r,p);
}
int qry(int x,int l,int r,int tl,int tr) {
    if(!x)
        return 0;
    if(tl<=l&&r<=tr)
        return t[x].s;
    int mid=l+r>>1,res=0;
    if(tl<=mid)
```

```
        res+=qry(t[x].ls,l,mid,tl,tr);
        if(tr>mid)
            res+=qry(t[x].rs,mid+1,r,tl,tr);
        return res;
    }
    int main() {
        // freopen("robot.inp","r",stdin);
        scanf("%d%d",&n,&k);
        for(int i=1; i<=n; i++)
            scanf("%d%d%d",&a[i].x,&a[i].r,&a[i].q);
        sort(a+1,a+n+1);
        for(int i=1; i<=n; i++) {
            for(int j=max(0,a[i].q-k); j<=a[i].q+k; j++)
                ans+=qry(rt[j],-inf,inf,a[i].x-
a[i].r,a[i].x+a[i].r);
            upd(rt[a[i].q],-inf,inf,a[i].x);
        }
        printf("%lld",ans);
    }
```

### 3.13.4. Test

[https://drive.google.com/drive/folders/1ofheTovqzJrTW\\_XeCXBxQA-IHO0qqCmT?usp=sharing](https://drive.google.com/drive/folders/1ofheTovqzJrTW_XeCXBxQA-IHO0qqCmT?usp=sharing)

## 4. BÀI TẬP THẢO LUẬN

### 4.1. Thảo luận 1: Hình vuông lớn nhất, bài E – Bubble cup 2013

#### 4.1.1. Đề bài

*Bài E: Hacker – Bubble cup 2013*

Cho một hình chữ nhật kích thước  $W * H$  (chiều rộng  $W$  và chiều cao  $H$ ) gắn một hệ trục tọa độ vuông góc  $Oxy$  với gốc  $O$  trùng với góc trái dưới.

Đặt  $N$  hình chữ nhật nhỏ, hình thứ  $i$  cho bởi bộ 4 số  $x_i, y_i, w_i, h_i$  là góc tọa độ góc trái dưới  $(x_i, y_i)$ , chiều rộng  $w_i$ , chiều cao  $h_i$  của hình. Hỏi có thể đặt thêm một hình vuông kích thước lớn nhất trên hình chữ nhật lớn ban đầu, sao cho nó không che khuất bất cứ hình chữ nhật nhỏ nào? Cho phép được chung cạnh, chung đỉnh.

Dữ liệu vào: SQUARE.inp

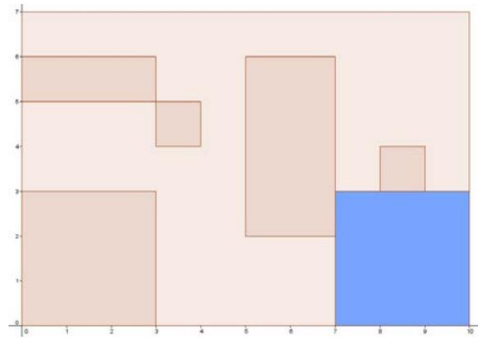
Dòng đầu tiên ghi 2 số  $W, H, N$  ( $1 \leq W, H \leq 10^9$ ;  $1 \leq N \leq 4 \cdot 10^4$ ) là kích thước hình chữ nhật lớn là  $W, H$ ; số hình chữ nhật nhỏ là  $N$ .

$N$  dòng tiếp theo ghi  $N$  bộ 4 **số nguyên**  $x_i, y_i, w_i, h_i$  ( $1 \leq x_i, y_i, w_i, h_i \leq 10^9$ ;  $x_i + w_i \leq W$ ;  $y_i + h_i \leq H$ ) thể hiện thông tin về hình chữ nhật nhỏ.

Kết quả ra: SQUARE.out

Ghi ra kích thước lớn nhất mà hình vuông có thể đặt thêm vào để không che khuất bất kì hình chữ nhật nhỏ nào.

Ví dụ:

SQUARE.inp	SQUARE.out	Giải thích	
<pre> 10 7 5 0 0 3 3 3 4 1 1 0 5 3 1 5 2 2 4 8 3 1 1 </pre>	3		Hình màu xanh là hình vuông lớn nhất.

Subtask 1: 50% test có  $1 \leq W, H \leq 10^3$ ;

Subtask 2: 50% test còn lại không có thêm điều kiện.

#### 4.1.2. Thảo luận thuật toán

Với subtask 1: Khi kích thước của hình chữ nhật ban đầu là nhỏ, ta đánh dấu các vị trí bị che phủ bởi các hình chữ nhật. Xét từng vị trí góc trái dưới của hình vuông, chặt nhị phân kết quả kích thước hình vuông không che lấp hình nào ban đầu.

Độ phức tạp:  $O(N^2 \cdot \log(N))$ .

Với subtask 2: Sử dụng sweep line

Quan sát 1: Nếu bài toán chỉ có một hình vuông  $R(0,0,x,y)$ , ta mở rộng hình vuông thành  $R_1(0,0,x+k,y+k)$ , khi đó với mọi điểm A không nằm trong  $R_1$  ta đều có thể dựng một hình vuông  $k \times k$  nhận A là đỉnh góc phải trên mà không đè lên hình  $R$ . Ngược lại, nếu mọi điểm B nằm trong  $R_1$  đều không thể dựng hình vuông  $k \times k$  nhận B là góc phải trên mà không đè lên  $R$ .



Nếu mở rộng hình  $R(0,0,x,y)$  lên  $k$  nhưng đảm bảo có 1 cạnh chưa ra ngoài hình chữ nhật ban đầu thì ta hoàn toàn có thể dựng được hình vuông  $k \times k$ . Vậy nên ta mở rộng  $k$  đến khi nào vẫn còn cạnh nằm trong hình chữ nhật ban đầu.

Quay trở lại bài toán ban đầu. Nếu sau khi mở rộng mà vẫn tồn tại một điểm A không nằm trong bất kì hình chữ nhật nào thì ta luôn dựng được một hình vuông có kích thước  $k \times k$  có góc phải trên là A (như cách dựng ở Quan sát 1). Nếu hình chữ nhật được mở rộng mà có kích thước vượt qua hình chữ nhật ban đầu thì ta xoá phần vượt quá đi. Nếu mở rộng tất cả N hình nhỏ lên K về cả 4 phía mà diện tích che phủ của nó chưa quá diện tích hình chữ nhật ban đầu thì ta luôn đặt được một hình chữ nhật kích thước  $K+1$  vào trong đó.

Để tìm ra K ta cần chặt nhị phân. Để tính diện tích N hình sau khi mở rộng ta sử dụng bài **Ví dụ 4**.

Độ phức tạp thuật toán:  $O(N * \log(N) * \log(\min(W, H)))$ . Trong đó  $O(\log(\min(W, H)))$  là thời gian chặt nhị phân tìm kích thước hình vuông.

## 4.2. Thảo luận 2: Bảo vệ nhà kho (Bubble cup 2019, bài H)

### 4.2.1. Đề bài

Bob Bubblestrong mới nhận được công việc làm bảo vệ cho hệ thống nhà kho, nhiệm vụ là phát hiện kẻ đột nhập và báo cảnh sát. Nhìn từ trên xuống, mỗi nhà kho coi như là một đa giác lồi, không có 2 nhà kho nào giao nhau, không nhà kho nào xây dựng trong nhà kho khác.

Bob khá lười nên anh ta đã sử dụng một hệ thống kính siêu X quang có thể nhìn xuyên qua mọi bức tường. Không may, hôm nay nó gặp sự cố nên chỉ có khả năng nhìn xuyên qua 1 bức tường. Hỏi diện tích nhà kho mà Bob có thể quản lý được là bao nhiêu? Bob cần tính số này, để từ đó biết diện tích phần còn lại mình cần phải tự đi kiểm tra để đảm bảo an toàn.

Dữ liệu vào: Đọc vào từ tệp Warehouse.inp

Dòng đầu tiên ghi số N là số nhà kho

N dòng tiếp theo mô tả nhà kho, dòng thứ  $i$  trong đó định dạng như sau: số đầu tiên  $c_i$  là số đỉnh của nhà kho, tiếp theo là  $c_i$  cặp tọa độ  $(x_j, y_j)$  là số nguyên là các đỉnh của nhà kho  $i$  được liệt kê theo chiều kim đồng hồ.

Biết Bob đặt hệ thống theo dõi tại tọa độ (0,0).

Kết quả ra: Ghi ra tệp Warehouse.out

Một số là số diện tích phần nhà kho theo dõi được ở dạng thập phân, được sai số  $10^{-4}$ .

Ràng buộc:

$$1 \leq N \leq 10^4; 3 \leq c_i \leq 10^4; \sum_{i=1}^n c_i \leq 5 \cdot 10^4; |x_j|, |y_j| \leq 3 \cdot 10^4$$

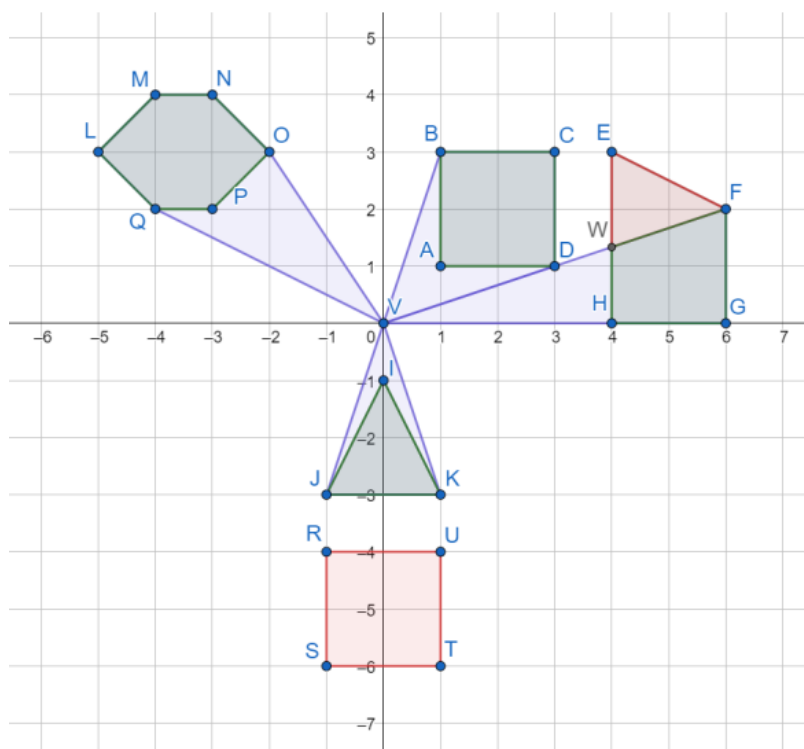
Ví dụ:

Warehouse.inp	Warehouse.out
5 4 1 1 1 3 3 3 3 1 4 4 3 6 2 6 0 4 0 6 -5 3 -4 4 -3 4 -2 3 -3 2 -4 2 3 0 -1 1 -3 -1 -3 4 1 -4 1 -6 -1 -6 -1 -4	13.333333333333

### **Giải thích:**

Phần diện tích tô màu hồng là vùng không nhìn thấy, còn lại các vùng nhìn thấy.

$$P = P_{ABCD} + P_{FGHW} + P_{LMNOPQ} = 4 + 3.33333333333 + 2 + 13.3333333333$$



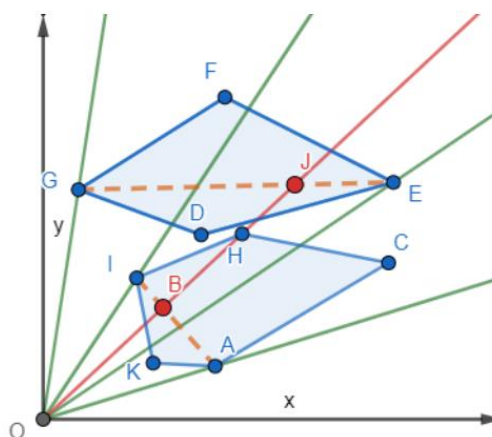
$P_{IJK}$

$4 =$

### **4.2.2. Thảo luận huật toán**

Sử dụng sweep line có gốc đặt tại  $O(0;0)$  theo chiều ngược chiều kim đồng hồ để đi qua tất cả các đỉnh của các đa giác.

Để thấy đa giác có đỉnh gần nhất, gần  $O$  hơn thì sẽ ưu tiên nhìn thấy, có thể che khuất đa giác sau nó.



Ta biết để tính diện tích của một tam giác ABC khi biết tọa độ 3 đỉnh A, B, C ta có thể sử dụng một trong hai công thức đơn giản sau:

$$S_{ABC} = \sqrt{p(p-a)(p-b)(p-c)} = \frac{1}{2} * |\vec{AB} \times \vec{AC}|$$

Xét trường hợp chỉ có 1 đa giác:

Khi đó các đỉnh của hình trên được quét theo thứ tự  $A \rightarrow B \rightarrow C \rightarrow D$ . Cứ có 3 đỉnh ta sẽ có 1 tam giác và cộng diện tích của nó vào kết quả.

$$S_{(\text{vùng nhìn thấy})} = S(CBA) + S(DCA)$$

Khi gặp đỉnh đầu tiên của đa giác gần hơn: như hình a) thì ta xác định giao của OA (sweep line đang xét) với đa giác phía sau là K, L. Khi đó diện tích phần nhìn thấy cộng thêm  $S(KLE)$ . Khi quét tiếp đa giác phía trước thì các đỉnh nằm trong đỉnh đầu tiên và cuối của của đa giác phía trước của các đa giác nằm sau đều bị bỏ qua. Vùng nhìn thấy sẽ là:

$$S(KLE) + S(CBA) + S(JCA)$$

Khi gặp đỉnh cuối cùng của đa giác phía trước như hình bên thì ta phải tìm giao của OB với đa giác sau tại I, J

Vùng nhìn thấy là:

$$S(BCA) + S(EJI) + S(GEI)$$

Việc tìm giao điểm của sweep line với các đa giác phía sau khi sweep line quét từ đỉnh bắt đầu đến kết thúc của đa giác phía trước là không cần thiết.

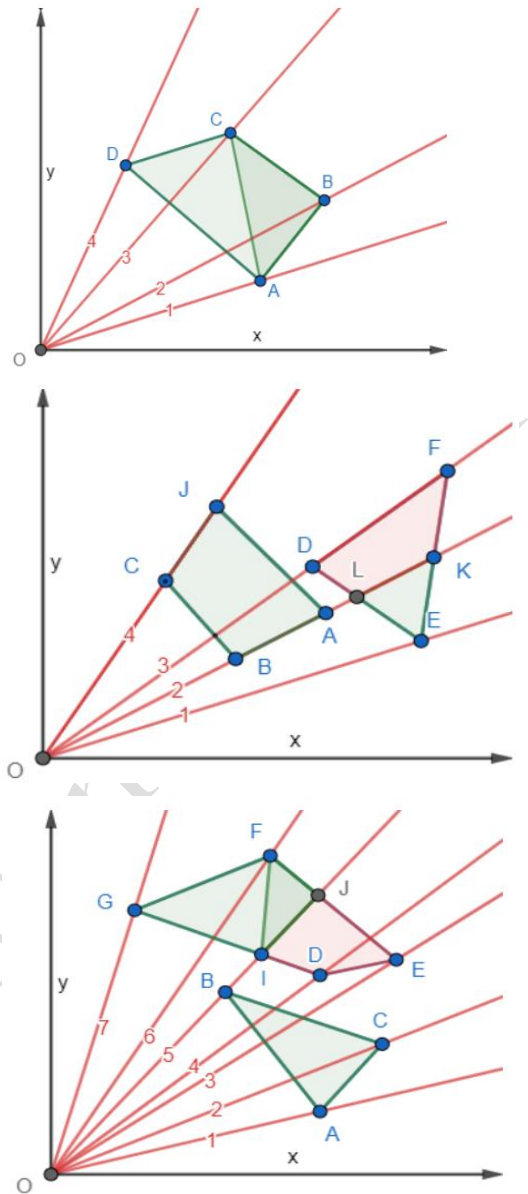
Bài toán của chúng ta chỉ cần sắp xếp các đỉnh theo thứ tự vòng tròn, ngược chiều kim đồng hồ. Nếu cùng góc đo thì đỉnh gần hơn sẽ ưu tiên xử lý trước, việc tìm giao điểm của đường thẳng với đa giác có đỉnh đang xét trong  $O(1)$ , tính diện tích tam giác trong  $O(1)$ , sắp xếp các đỉnh trong  $O(N \cdot \log(N))$ , xử lý các đỉnh trong  $O(N)$  cần tránh việc tìm một số giao điểm không cần thiết của các đa giác phía sau với sweep line.

Độ phức tạp thuật toán:  $O(N * \log(N))$

## 5. MỘT SỐ BÀI TỰ LUYỆN

Các bài tự luyện, được sưu tầm từ các trang mạng như Codechef, Codeforces, các cuộc thi khác có chủ đề về hình học. Từ khoá tìm kiếm sẽ là sweep line, scan line, line sweep. Rất nhiều bài thuộc mức độ trung bình, khó.

Bài 1: <https://www.codechef.com/problems/INSQ16G> (Flash and Zoom)



Bài 2: <https://www.codechef.com/problems/URBANDEV>

Bài 3: <https://www.codechef.com/problems/CR192>

Bài 4: <https://www.spoj.com/problems/RAIN1/> (November rain)

Bài 5: <https://www.codechef.com/problems/CHEFCIRC> (Chef and Circle)

## **6. TÀI LIỆU THAM KHẢO**

[1] - <https://www.codechef.com>

[2] - <https://codeforces.com>

[3] - <https://vnoi.info>

[4] - <https://www.spoj.com>

[5] - <https://bubblecup.org/>

-----HẾT-----