

**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: .....**

**Đơn vị công tác: .....**

**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

# **SWEEP LINE**

*Tác giả: .....*

## **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  $10^9$ .

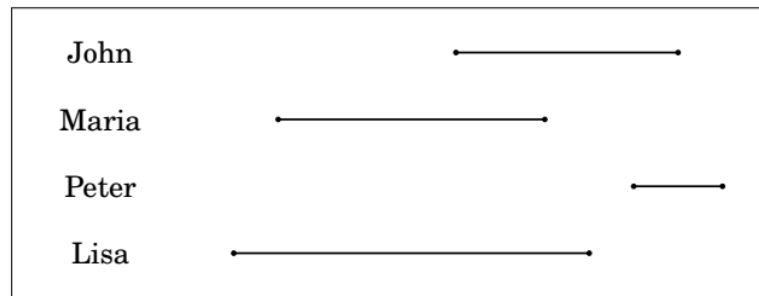
Ví dụ thời gian đến và về của các nhân viên được cho trong bảng sau:

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

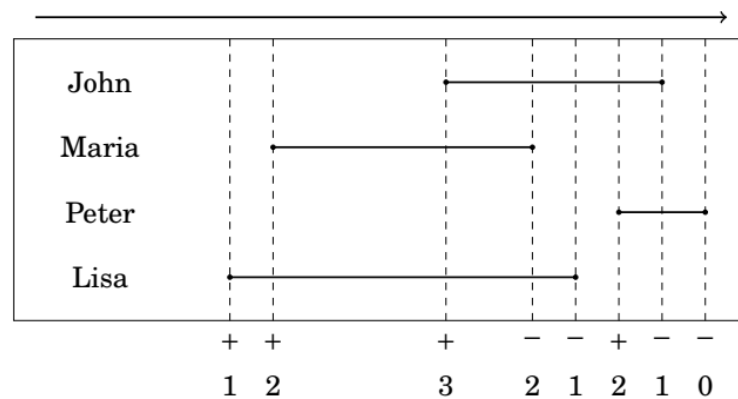
Ta có giải quyết rất tự nhiên như sau: Để biết tại mỗi thời điểm T bất kì, có bao nhiêu nhân viên đang có mặt tại công ty, cách đơn giản nhất là kiểm tra T có nằm trong đoạn thời gian đến và về của một người thì người đó đang ở công ty. Việc kiểm tra tất cả các thời điểm như vậy sẽ mất thời gian  $O(T_{\max}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

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:

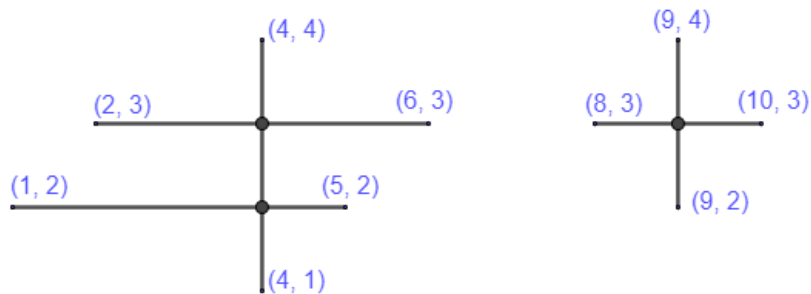
```
#include <bits/stdc++.h>
using namespace std;
int n, cnt, res;
vector<pair<int, int>> point;
int main() { //meeting
    cin >> n;
    for(int i=1; i<=n; i++) {
        int in, out; //Thoi gian den, di cua 1 nguoi
        cin >> in >> out;
        point.push_back({in, 0});
```

```
        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  $10^5$ .

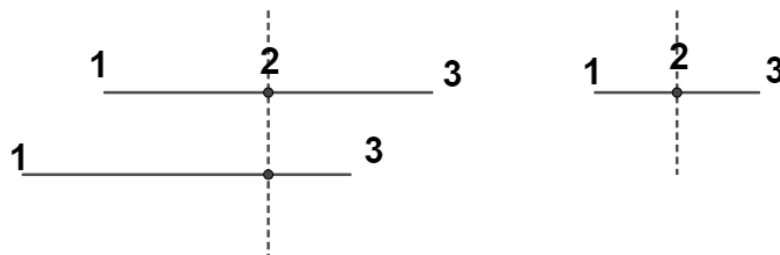
Ví dụ: Có 5 đoạn thẳng, biết tọa độ điểm đầu và cuối của chúng như hình bên dưới, dễ thấy chúng có 3 giao điểm.



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) {
```

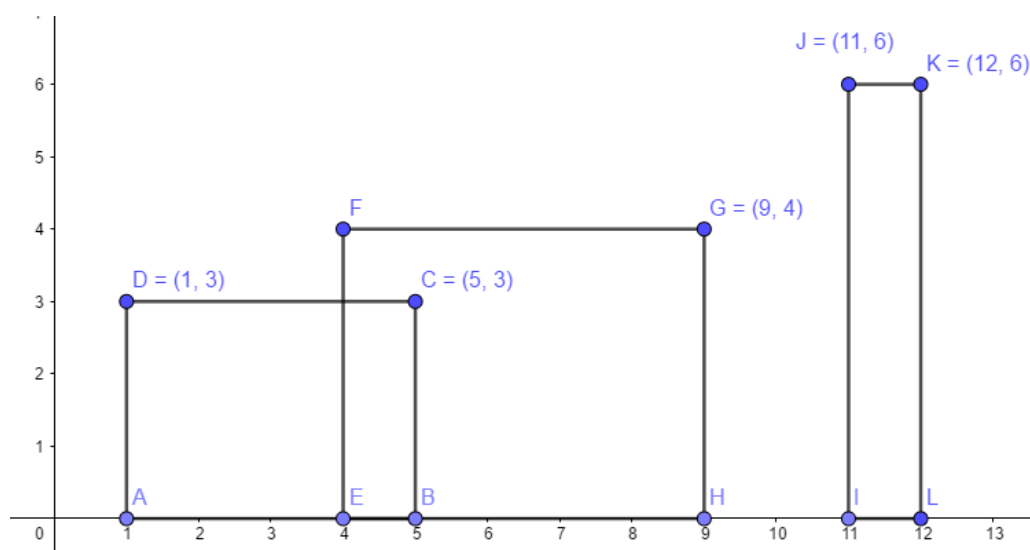
```

        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 miếng 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 tọa độ, các hình chữ nhật đặt ngẫu nhiên trong mặt phẳng. Mỗi hình chữ nhật cho tọa độ góc trái trên  $(x_1; y_1)$  và phải dưới  $(x_2; y_2)$ . Các tọa 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 lý 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 xóa 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:

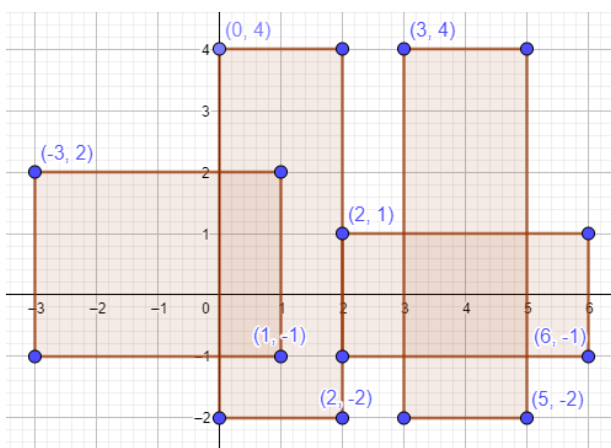
- cnt: Số hình chữ nhật đang che phủ hoàn toàn nút quản lý.
- cover: Độ dài đoạn tung độ bị che phủ thuộc nút quản lý

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
--	----	------------------------



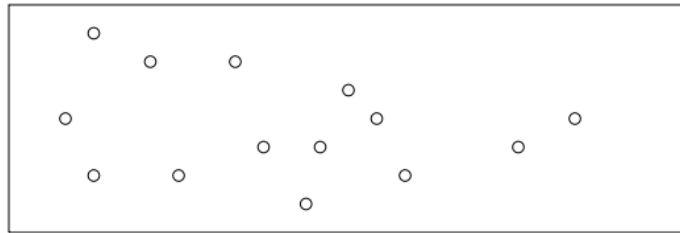
### Chương trình minh họa:

```
#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() {
```

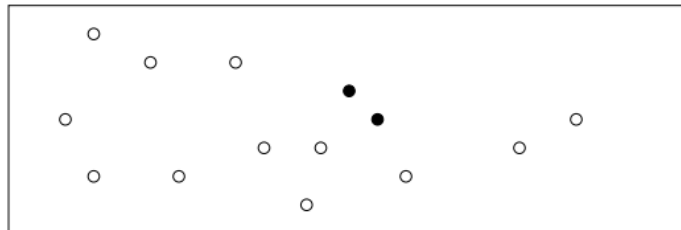
```
// freopen("HCN.inp", "r", stdin);
cin>>n;
for(int i=1; i<=n; i++) {
    int x1,y1,x2,y2;
    cin>>x1>>y1>>x2>>y2;
    E.push_back({x1,y2,y1,1});///open
    E.push_back({x2,y2,y1,-1});///close
}
sort(E.begin(),E.end());
for(int i=0; i<E.size(); i++) {
    res=res+ST[1].cover*(E[i].x-E[i-1].x);
 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



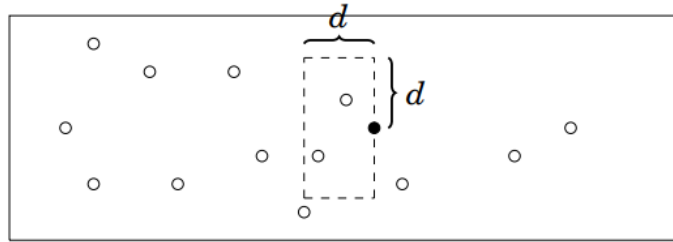
Cặp điểm gần nhau nhất là:



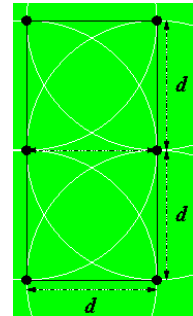
Cho biết 2 điểm  $A(x_A, y_A)$  và  $B(x_B, y_B)$  thì độ dài đoạn  $AB = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$ . Do đó khoảng cách giữa 2 điểm bất kì tính trong  $O(1)$ . Với cách tư duy đơn giản, ta đi kiểm tra tất cả các khoảng cách để tìm khoảng cách nhỏ nhất. Độ phức tạp:  $O(N^2)$ .

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;
}

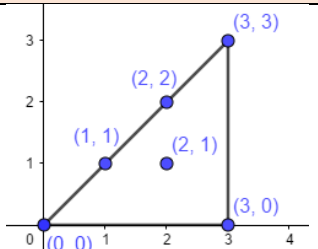
int main() {
    int n;
    while (cin >> n && n) {
        vector<point> eventPts(n);
        for (int i = 0; i < n; ++i) {
            double x, y;
            cin >> x >> y;
            eventPts[i] = point(x, y);
        }
        double d = closestPair(eventPts);
    }
    return 0;
}

```

## 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.