VẬN DỤNG KIẾN THỨC HÌNH HỌC
TRONG VIỆC TỐI ƯU HÓA THUẬT TOÁN

A. MỞ ĐẦU

1. Lý do chọn đề tài

Hình học đối với giác quan của con người thì khá quen thuộc và dễ dàng. Nhưng hình học đối với máy tính thì lại là một vấn đề khác. Nhiều bài toán ta có thể giải ngay lập tức bằng cách "nhìn vào hình vẽ ta thấy", nhưng để thể hiện trên máy tính thì cần những chương trình không đơn giản chút nào. Các giải thuật hình học thường là các giải thuật đẹp và đôi khi là rất bất ngờ. Thực vậy, những tưởng có những bài toán ta phải giải quyết với chi phí thuật toán rất lớn (đôi khi không thể chấp nhận được), nhưng nhờ vào chính những tính chất đặc biệt của hình học mà ta lại có thể giải quyết nó một cách dễ dàng và "đẹp mắt".

Vì thế tôi chọn đề tài: "Vận dụng kiến thức hình học trong việc tối ưu hóa thuật toán", nhằm mục đích xây dựng một hệ thống bài tập liên quan đến hình học, và chỉ ra trong đó các kiến thức hình học giúp tối ưu hóa bài toán.

2. Mục đích chọn đề tài

Đề tài tôi chọn hướng tới các mục đích sau:

- Xây dựng được hệ thống bài tập đa dạng về hình học (nghĩa là có những bài tập hình học phải sử dụng phương pháp duyệt, phương pháp quay lui,số học…, có những bài tập hình học phải sử dụng các tính chất hình học nằm ngay trong hình mới có thể giải quyết được,……)

B. NỘI DUNG

Bài 1. Kiểm tra một điểm có nằm trong đa giác hay không

Cho một đa giác gồm có n đỉnh và m điểm. Nhiệm vụ của bạn là xác định xem điểm đó nằm bên trong, bên ngoài hay nằm trên các cạnh của đa giác. Đa giác bao gồm n đỉnh (x_1, y_1), (x_2, y_2),…,(x_n, y_n). Đỉnh (x_i, y_i) và (x_{i+1}, y_{i+1}) là liền kề với i=1,2,…n-1 và đỉnh (x_1, y_1) và (x_n, y_n) cũng liền kề

Input: Dòng đầu tiên gồm hai số nguyên n và m: số đỉnh của đa giác và số điểm. n dòng tiếp theo, mỗi dòng gồm hai số nguyên x_i, và y_i. Giả sử đa giác có các cạnh chỉ giao nhau tại các đỉnh của đa giác. M dòng tiếp theo mô tả các điểm, mỗi dòng gồm hai số nguyên x và y

Output: Với mỗi điểm, đưa ra "INSIDE", "OUTSIDE" hoặc "BOUNDARY"

Giới hạn:

$3 \le n, m \le 1000$

$1 \le m \le 1000$

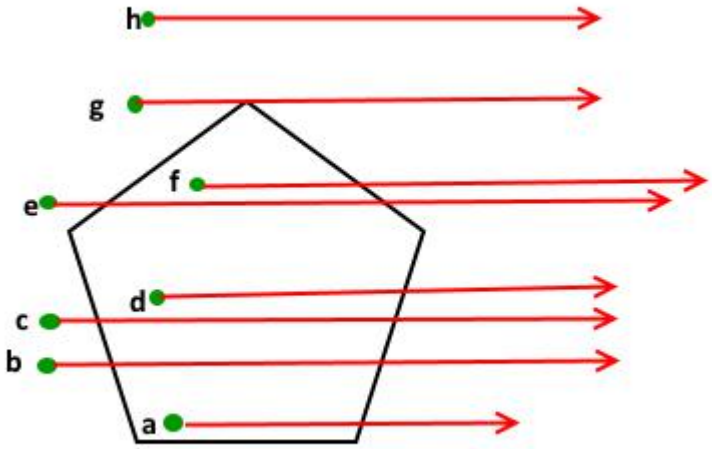
$-10^9 \le x_i, y_i \le 10^9$

$-10^9 \le x, y \le 10^9$

Input	Output
4 3 1 1 4 2 3 5 1 4 2 3 3 1 1 3	INSIDE OUTSIDE BOUNDARY

Thuật toán: Ta sẽ vẽ đoạn thẳng nằm ngang hướng sang bên phải nối giữa mỗi điểm và mở rộng tới một điểm ở vô cùng. Đếm số lần đoạn thẳng đó giao với các cạnh của

đa giác. Một điểm nằm trong đa giác nếu số lượng giao điểm là lẻ hoặc điểm nằm trên một cạnh của đa giác. Nếu không có điều kiện nào là đúng, thì điểm nằm bên ngoài.



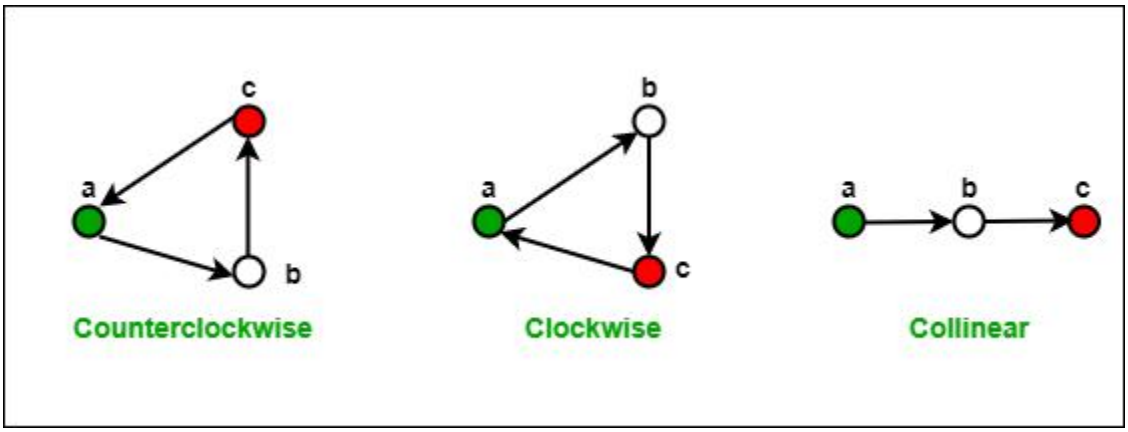
Lưu ý rằng ta sẽ trả về true nếu điểm nằm trên cạnh của đa giác hoặc là một trong các đỉnh của đa giác. Để xử lý điều này, sau khi kiểm tra đoạn thẳng nối từ điểm p tới điểm vô cùng có tuyến tính với cạnh đang xét đến của đa giác hay không. Nếu nó là tuyến tính, ta sẽ kiểm tra xem điểm p có nằm trên cạnh hiện tại của đa giác hay không, nếu nó nằm trên cạnh hiện tại của đa giác, ta sẽ trả về giá trị true, nếu không nằm trên cạnh của đa giác, ta sẽ trả về false

Như vậy bài toán ở trên sẽ quy về việc làm thế nào để kiểm tra xem hai đoạn thẳng có giao nhau hay không.

Bài toán con 1: Làm thế nào để kiểm tra xem hai đoạn thẳng có giao nhau hay không?

Trước khi đến việc kiểm tra hai đoạn thẳng có giao nhau hay không, ta cùng nhau tìm hiểu một số lý thuyết:

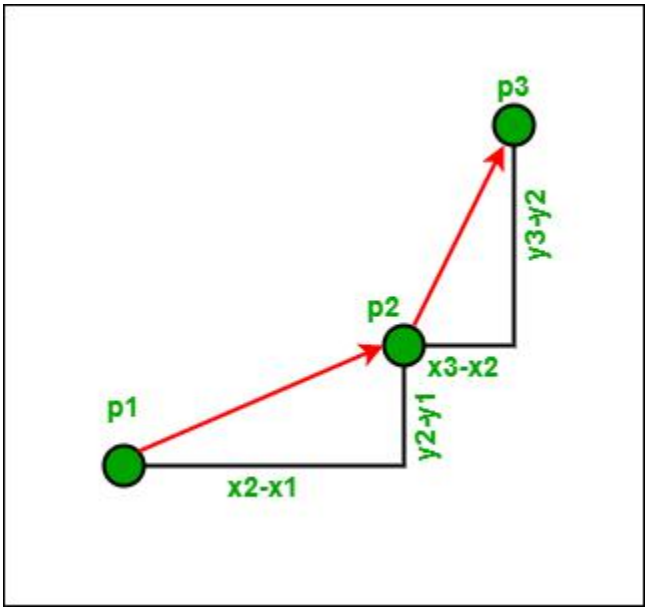
Hướng của 3 điểm theo thứ tự: Hướng của ba điểm có thứ tự có ba kiểu: ngược chiều kim đồng hồ (counterclockwise), cùng chiều kim đồng hồ (clock wise) hoặc tuyến tính hay thẳng hàng (collinear). Hình minh họa dưới đây thể hiện cho hướng của ba điểm:



Nếu hướng của ba điểm (p_1, p_2, p_3) là tuyến tính, khi đó hướng của (p_3, p_2, p_1) cũng tuyến tính

Nếu hướng của (p_1, p_2, p_3) là theo chiều kim đồng hồ, khi đó hướng của (p_3, p_2, p_1) là ngược chiều kim đồng hồ và ngược lại

Bài toán con 2: Cho ba điểm p_1, p_2 và p_3, tìm hướng của ba điểm (p_1, p_2, p_3). Hình minh họa dưới đây minh họa cho cách tìm ra hướng của ba điểm (p_1, p_2, p_3), ta dựa vào hệ số góc của đường thẳng:



Hệ số góc của đường thẳng (p_1, p_2) là $\alpha = (y_2 - y_1) / (x_2 - x_1)$

Hệ số góc của đường thẳng (p_2, p_3) là $\beta = (y_3 - y_2) / (x_3 - x_2)$

Nếu $\alpha > \beta$, hướng sẽ là theo chiều kim đồng hồ (xoay sang phải)

Sử dụng các giá trị trên, ta có thể tóm lại rằng, hướng của ba điểm phụ thuộc vào dấu của biểu thức sau: (y2 - y1)*(x3 - x2) - (y3 - y2)*(x2 - x1). Nếu biểu thức này âm, tức là ngược chiều kim đồng hồ. Nếu biểu thức này bằng 0, tức là cùng chiều kim đồng hồ.

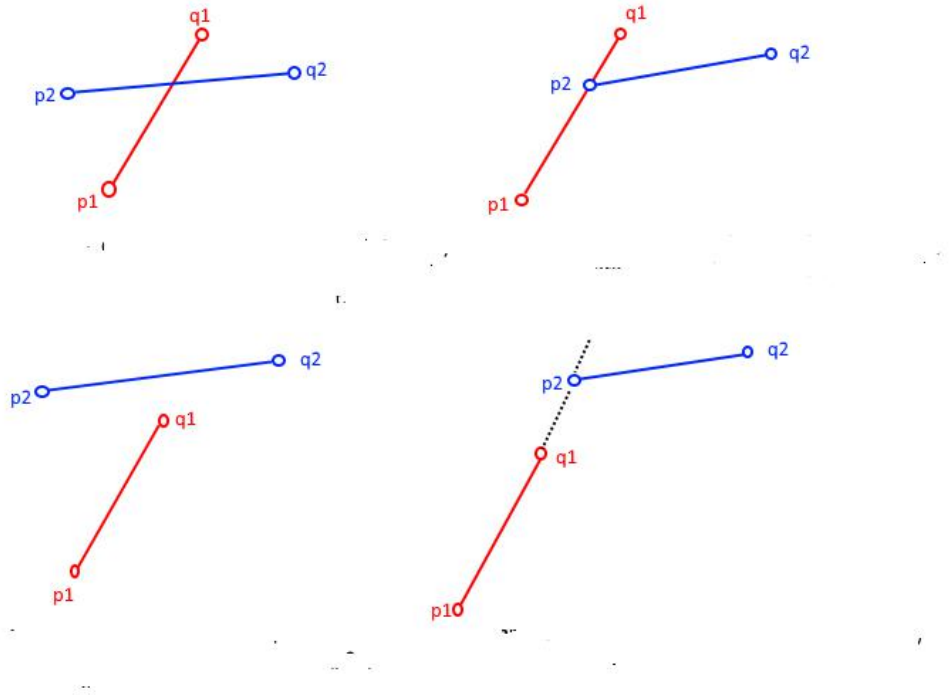
Trở lại với Bài toán con 1: Cho hai đoạn thẳng (p_1, q_1) và (p_2, q_2), tìm xem hai đoạn thẳng này có giao nhau hay không.

Bài toán con 2 có thể giúp ích cho bài toán con 1 như sau:

Hai đoạn thẳng (p_1, q_1) và (p_2, q_2) cắt nhau khi và chỉ khi một trong hai điều kiện sau là đúng:

Trường hợp 1: Trường hợp tổng quát:

- (p1, q1, p2) và (p1, q1, q2) có hướng khác nhau và
- (p2, q2, p1) and (p2, q2, q1) có hướng khác nhau



-
- Trong bức tranh trên có bốn tình huống:
- Tình huống 1: Hướng của (p1, q1, p2) và (p1, q1, q2) có hướng khác nhau và
- (p2, q2, p1) and (p2, q2, q1) có hướng khác nhau
- Tình huống 2: Hướng của (p1, q1, p2) và (p1, q1, q2) có hướng khác nhau và

- (p2, q2, p1) and (p2, q2, q1) có hướng khác nhau
- Tình huống 3: Hướng của (p1, q1, p2) và (p1, q1, q2) có hướng khác nhau và
- (p2, q2, p1) and (p2, q2, q1) là giống nhau
- Tình huống 4: Hướng của (p1, q1, p2) và (p1, q1, q2) có hướng khác nhau và
- (p2, q2, p1) and (p2, q2, q1) là giống nhau

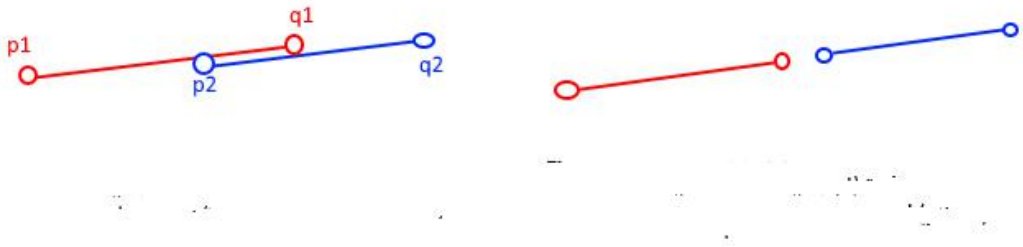
Sử dụng điều này, ta có thể kiểm tra hai đoạn thẳng cắt nhau hoặc giao nhau

Trường hợp 2: Trường hợp đặc biệt

(p1, q1, p2), (p1, q1, q2), (p2, q2, p1), và (p2, q2, q1) là tuyến tính.

Khi đó phép chiếu (p1, q1) và (p2, q2) trên trục Ox sẽ giao nhau hay tọa độ x của một điểm này sẽ nằm trong khoảng tọa độ của của hai đầu mút đoạn thẳng còn lại. Khi đó hai đoạn thẳng giao nhau.

Phép chiếu trên trục Oy của (p1, q1) và (p2, q2) giao nhau hay tọa độ y của một điểm này sẽ nằm trong khoảng tọa độ của hai đầu mút của đoạn thẳng còn lại. Khi đó hai đoạn thẳng không giao nhau.



Sau đây là đoạn chương trình cài đặt để kiểm tra xem hai đoạn thẳng có giao nhau hay không

```
// A C++ program to check if two given line segments intersect
#include <iostream>
using namespace std;


struct Point
```

```
{
   int x;
   int y;
};

// Given three colinear points p, q, r, the function checks if
// point q lies on line segment 'pr'
bool onSegment(Point p, Point q, Point r)
{
   if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
       q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
      return true;

   return false;
}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{       int val = (q.y - p.y) * (r.x - q.x) -
            (q.x - p.x) * (r.y - q.y);
```

```
    if (val == 0) return 0;  // colinear

    return (val > 0)? 1: 2; // clock or counterclock wise
}

// The main function that returns true if line segment 'p1q1'
// and 'p2q2' intersect.
bool doIntersect(Point p1, Point q1, Point p2, Point q2)
{
    // Find the four orientations needed for general and
    // special cases
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    // General case
    if (o1 != o2 && o3 != o4)
        return true;

    // Special Cases
    // p1, q1 and p2 are colinear and p2 lies on segment p1q1
    if (o1 == 0 && onSegment(p1, p2, q1)) return true;

    // p1, q1 and q2 are colinear and q2 lies on segment p1q1
```

```
    if (o2 == 0 && onSegment(p1, q2, q1)) return true;

    // p2, q2 and p1 are colinear and p1 lies on segment p2q2
    if (o3 == 0 && onSegment(p2, p1, q2)) return true;

     // p2, q2 and q1 are colinear and q1 lies on segment p2q2
    if (o4 == 0 && onSegment(p2, q1, q2)) return true;

    return false; // Doesn't fall in any of the above cases
}
 int main()
{
    struct Point p1 = {1, 1}, q1 = {10, 1};
    struct Point p2 = {1, 2}, q2 = {10, 2};

    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";

    p1 = {10, 0}, q1 = {0, 10};
    p2 = {0, 0}, q2 = {10, 10};
    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";

    p1 = {-5, -5}, q1 = {0, 0};
    p2 = {1, 1}, q2 = {10, 10};
    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";
```

```
    return 0;
}
```

Như vậy bài toán kiểm tra điểm nằm trong đa giác đã rõ ràng. Sau đây là đoạn chương trình cài đặt của bài toán này:

```
#include <bits/stdc++.h>
using namespace std;
int n, m;
struct Point{
    long long x, y;
    Point operator+(Point b){
        return {x+b.x, y+b.y};
    }
    Point operator-(Point b){
        return {x-b.x, y-b.y};
    }
}p, a[1005];
long long kc(Point a, Point b){
    return a.x*b.y - a.y*b.x;
}
long long dot(Point a, Point b){
    return a.x*b.x + a.y*b.y;
}
int sign(long long x){
    if (x > 0) return 1;
```

```
    else if (x < 0) return -1;
    else return 0;
}
bool onseg(Point a, Point b, Point c){
    if (kc(c-a, b-a) != 0) return false;
    if (0 <= dot(c-a, b-a) && dot(c-a, b-a) <= dot(b-a, b-a)) return true;
    return false;
}

bool intersec(Point a, Point b, Point c, Point d){
    int f1 = sign(kc(a-c, d-c)) * sign(kc(b-c, d-c));
    int f2 = sign(kc(c-a, b-a)) * sign(kc(d-a, b-a));
    if (f1 < 0 && f2 < 0) return true;
    if (f1 > 0 || f2 > 0) return false;
    if (onseg(a, b, c)) return true;
    if (onseg(a, b, d)) return true;
    if (onseg(c, d, a)) return true;
    if (onseg(c, d, b)) return true;
    return false;
}
void kt(Point x, Point a[]){
    for (int i = 1; i < n; i++){
        if (onseg(a[i-1], a[i], x)){
            cout << "BOUNDARY"<<'\n';
            return;
```

```
        }
    }
    if (onseg(a[n-1], a[0], x)){
        cout << "BOUNDARY"<<'\n';
        return;
    }
    Point y = x + Point{1, (long long)2e9+5};
    int kq = 0;
    for (int i = 1; i < n; i++){
        if (intersec(x, y, a[i-1], a[i])) kq++;
    }
    if (intersec(x, y, a[n-1], a[0])) kq++;
    if (kq % 2 == 1) cout << "INSIDE"<<'\n';
    else cout << "OUTSIDE"<<'\n';
}

int main()
{   ios_base::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);
    cin >> n >> m;
    for (int i = 0; i < n; i++){
        cin >> a[i].x >> a[i].y;
    }
    while(m--)
    {
```

```
        cin >> p.x >> p.y;
        kt(p, a);
    }
    return 0;
}
```

Link tải bộ test của bài toán:

https://drive.google.com/drive/folders/1IPXlpRsMBoT6aGjcs_Ps6X41fmyyUt9X?usp=sharing

Bài 2. Diện tích nhỏ nhất

Cho một mảng gồm có N có điểm có tọa độ (x_i, y_i) phân biệt. Yêu cầu bài toán: Tìm diện tích nhỏ nhất của hình chữ nhật được hình thành từ những điểm này.

Input: Dòng đầu tiên gồm số N. N dòng tiếp theo mỗi dòng gồm hai số nguyên x_i và y_i.

Output: Yêu cầu của bài toán. Nếu không có hình chữ nhật nào như vậy, đưa ra 0. Sai số của câu trả lời không vượt quá 10^{-5} .

Giới hạn: $1 \le N \le 10^5$

$0 \le x_i, y_i \le 4.10^4$

Input	Output
5 1 2 2 1 1 0 0 1	2.00000

Cách 1. Độ phức tạp $O(n^4)$

Ta sẽ duyệt qua 4 bốn điểm bất kỳ, sau đó kiểm tra xem bốn điểm này có tạo thành hình chữ nhật hay không, sau đó ta tìm max trong các hình chữ nhật này.

Cách 2. Độ phức tạp $O(n^3)$

Ta sẽ duyệt qua ba điểm, sau đó tìm tọa độ của điểm cuối cùng của hình chữ nhật, và tính toán diện tích của hình chữ nhật đó, sau đó tìm max trong các hình chữ nhật này

Cách 3. Độ phức tạp $O(n^2)$

Tính chất của hình chữ nhật:

Hai đường chéo của một hình chữ nhật có độ dài bằng nhau và cắt nhau tại trung điểm của mỗi đường. Do vậy, với mỗi cặp điểm, ta sẽ coi chúng là một đường chéo và tìm cách lưu lại khoảng cách giữa hai điểm, và tọa độ trung điểm của chúng. Ta có thể sử dụng thư viện map với khóa là chiều dài của hình chữ nhật và tọa độ tâm của chúng, còn giá trị của map là chỉ số của hai điểm tạo nên đường chéo đó.

Sau đây là đoạn chương trình cài đặt của bài toán:

```
#include<bits/stdc++.h>
double dist(vector<int> a, vector<int> b){
      double dx=a[0]-b[0];
      double dy=a[1]-b[1];
      return sqrt(dx*dx+dy*dy);
   }
   double minAreaFreeRect(vector<vector<int>>& points) {

unordered_map<int,unordered_map<int,unordered_map<int,vector<pair<int,int>>>>
> hm;
      int n=points.size();
      double ans=3e9;
      for(int i=0;i+1<n;i++){
         for(int j=i+1;j<n;j++){
                                                          int
midXTimesTwo=points[i][0]+points[j][0];
                int midYTimesTwo=points[i][1]+points[j][1];
                int distSqr=(points[i][0]-points[j][0])*(points[i][0]-
```

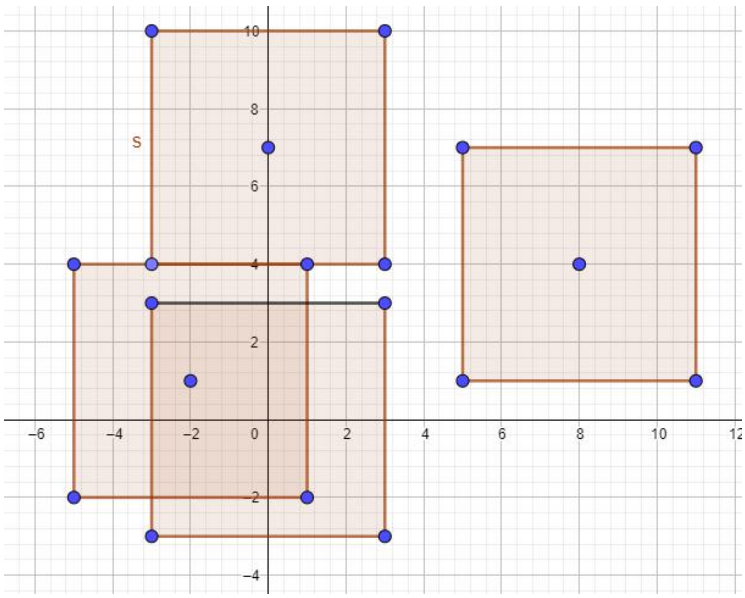
```
points[j][0])+(points[i][1]-points[j][1])*(points[i][1]-points[j][1]);
                if(hm[midXTimesTwo][midYTimesTwo].count(distSqr)){
                    vector<pair<int,int>>
diags=hm[midXTimesTwo][midYTimesTwo][distSqr];
                    for(auto p:diags){
                        double side1=dist(points[p.first],points[i]);
                        double side2=dist(points[p.first],points[j]);
                        ans=min(ans,side1*side2);
                    }
                }

hm[midXTimesTwo][midYTimesTwo][distSqr].push_back(make_pair(i,j));
            }
        }
        return ans>2e9?0.0:ans;
    }
```

Link bộ test của bài:

https://drive.google.com/drive/folders/1IPXlpRsMBoT6aGjcs_Ps6X41fmyyUt9X?usp=sharing

Bài 3. Hai hình vuông chồng chéo

Cho N hình vuông, mỗi hình vuông có kích thước là KxK. Hình vuông thứ i có tọa độ tâm là (x_i, y_i) với tọa độ số nguyên trong đoạn -1.000.000 ... 1.000.000. Hai hình vuông có thể chồng chéo lên nhau, nghĩa là chúng cùng nhau chia sẻ phần diện tích chung. Không có hai hình vuông nào có cùng tâm. Tính diện tích chung bởi hai hình vuông chồng chéo lên nhau. Đưa ra 0 nếu không có hai hình vuông chồng chéo lên nhau và đưa ra -1 nếu chồng chéo xảy ra giữa nhiều hình vuông



Input: Dòng 1: Hai số nguyên được phân tách bằng dấu cách, N và K. K được đảm bảo là chẵn. Dòng 2 đến N+1: dòng i + 1 chứa các số nguyên x_i và y_i, mô tả tâm của hình vuông thứ i

Output: Diện tích chung của hai hình vuông chồng lên nhau. Đưa ra số 0 nếu không có hai hình vuông nào trùng nhau và -1 nếu chồng chéo xảy ra giữa nhiều hơn một cặp hình vuông

Giới hạn: 2 <= N <= 50.000, 1 <= K <= 1.000.000

Input	Output
4 6 0 0 8 4 -2 1 0 7	20

Giải thích: Có 4 hình vuông, mỗi hình có kích thước 6 x 6. Hình vuông đầu tiên có tâm là (0,0), v.v. Hình vuông 1 và 3 chồng lên nhau với 20 là diện tích chung

Cách 1. Ta sẽ mượn ý tưởng của kỹ thuật sweep line (quét đường) vào bài này. Ban đầu ta sẽ tiến hành sắp xếp các tâm hình vuông tăng dần theo hoành độ x, sau đó tăng dần theo hoành độ y. Ta sẽ quét đường thẳng đứng từ trái sang bên phải để duyệt qua từng tâm hình vuông một, nếu hai hình vuông mà chia sẻ phần diện tích chung thì khoảng cách giữa hai hoành độ của tâm hình vuông phải nhỏ hơn k. Khi đó ta sẽ tiến hành tính diện tích của phần chung bằng cách tìm tọa độ các điểm trong phần diện tích chung này.

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
struct diem
{
   ll x, y;
};
bool ss(diem a, diem b)
{
   if(a.x == b.x) return a.y < b.y;
   return a.x < b.x;
}
diem p[50002];
ll n, k, dt = 0, d;
int main()
{
   ios_base::sync_with_stdio(0);
   cin.tie(0);
```

```
cout.tie(0);
cin >> n >> k;
for(int i = 1; i <= n; ++i)
    cin >> p[i].x >> p[i].y;

sort(p+1, p+1+n, ss);
for(int i = 2; i <= n; ++i)
    for(int j = i-1; j > 0; --j)
    if(abs(p[i].x-p[j].x) < k) {

        if(abs(p[i].y-p[j].y) < k)
        {
            diem a, b;
            if(p[j].y > p[i].y)
            {
                a.x = p[j].x+(k/2);
                a.y = p[j].y-(k/2);
                b.x = p[i].x-(k/2);
                b.y = p[i].y+(k/2);
            }
            else
            {
                a.x = p[j].x+(k/2);
                a.y = p[j].y+(k/2);
                b.x = p[i].x-(k/2);
```

```
                b.y = p[i].y-(k/2);
            }
            dt = abs((a.x-b.x)*(a.y-b.y));
            d++;
            if(d == 2)
            {
                cout << -1;
                return 0;
            }

        }
    }
    else break;
  cout << dt;
  return 0;
}
```

Độ phức tạp của thuật toán trên nhìn qua có thể là $O(n^2)$ nhưng thực tế chỉ là O(nlogn) vì khi biến đếm d tăng lên bằng 2 sẽ dừng chương trình luôn.

Cách 2: Vẫn sử dụng tư tưởng của thuật toán quét đường (sweep line), ta sẽ tìm kiếm hai điểm (x1,y2) và (x2,y2) thỏa mãn $|x_1 - x_2| \le k - 1$ và $|y_1 - y_2| \le k - 1$. Ta sẽ duy trì một đường thẳng quét với độ rộng là k-1, đi từ x-k+1 tới x. Khi x tăng, ta sẽ tạo ra một tập hợp (set) các điểm có mặt trong dải này. Mỗi lần một điểm (x_1, y_1) bước vào trong dải này, ta sẽ kiểm tra các điểm (x_2, y_2) trong dải này xem có thỏa mãn $|y_1 - y_2| \le k$, đồng thời chúng ta cũng phải xóa các điểm khi mà nó không nằm trong dải phạm vi có độ rộng k-1 do đường thẳng đó quét.

Đây là đoạn cài đặt cho cách này:

```
#include <bits/stdc++.h>
```

```
using namespace std;
struct Point {long long x, y;};
Point p[50001];
long long n, k;
vector < pair <Point, Point> > chong;
bool operator < (Point a, Point b)
{
   if (a.y == b.y) return a.x < b.x;
   return a.y < b.y;
}
bool cmpx (Point a, Point b)
{
   if (a.x == b.x) return a.y < b.y;
   return a.x < b.x;
}
multiset <Point> trai;
multiset <Point> ::iterator ptr;
long long dissq (Point a, Point b)
{
   return ((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
long long dtchong (pair <Point, Point> pp)
{
   Point p1 = pp.first;
   Point p2 = pp.second;
```

```
    long long rong = k - abs (p1.x - p2.x);
    long long dai = k - abs (p1.y - p2.y);
    return rong * dai;
}
long long closest()
{
    trai.insert (p[1]);
    int left = 1;
    for (int i = 2; i <= n; i++)
    {
        while (p[left].x < p[i].x - k) trai.erase (p[left++]);
        Point goc; goc.x = p[i].x - k; goc.y = p[i].y - k;
        if (!trai.empty())
            for (ptr = trai.lower_bound (goc); ptr != trai.end() && (*ptr).y <= p[i].y + k;
ptr++)
                if (abs (p[i].x - (*ptr).x) < k && abs (p[i].y - (*ptr).y) < k) chong.push_back
({p[i], *ptr});
        if (chong.size() > 1) return -1;
        trai.insert (p[i]);
    }
    if (chong.size() == 0) return 0;
    if (chong.size() > 1) return -1;
    return dtchong (chong[0]);
}
int main()
{
```

```
    ios_base :: sync_with_stdio(0);
    cin.tie (0);
    cout.tie (0);
    cin >> n >> k;
    for (int i = 1; i <= n; i++) cin >> p[i].x >> p[i].y;
    sort (p + 1, p + 1 + n, cmpx);
    cout << closest ();
    return 0;
}
```

Độ phức tạp của thuật toán này là O(nlogn)

Link bộ test của đề bài

https://drive.google.com/drive/folders/1IPXlpRsMBoT6aGjcs_Ps6X41fmyyUt9X?usp=sharing

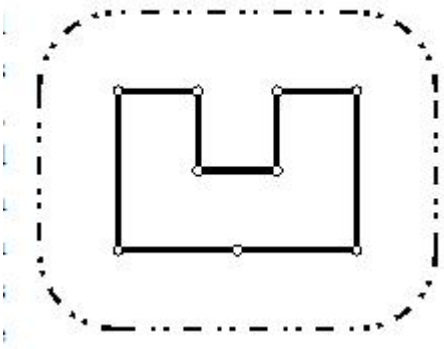
Bài 4. Xây tường thành

Cho lâu đài có dạng hình đa giác nằm trên trên khu đất bằng phẳng. Nhà vua cần xây dựng bức tường bao quanh toàn bộ lâu đài đó bằng cách sử dụng ít đá và nhân công nhất, nhưng yêu cầu bức tưởng phải cách lâu đài một khoảng cách ít nhất là L. Viết chương trình tìm ra tổng độ dài nhỏ nhất của mỗi bức tường mà anh ta có thể xây xung quanh lâu đài.

Input: Dòng đầu tiên gồm hai số nguyên N và L (3<=N<=1000) là số đỉnh trong lâu đài của vua và L (1<=L<=1000) khoảng cách tối thiểu từ bức tường tới tòa lâu đài

N dòng tiếp theo mô tả tọa độ các đỉnh của lâu đài theo chiều kim đồng hồ. Mỗi dòng gồm hai số nguyên X_i và Y_i cách nhau bởi một khoảng trắng (-10000<=x_i, y_i <=10000) thể hiện tọa độ của đỉnh thứ i. Tất cả các đỉnh đều khác nhau và các mặt của lâu đài chỉ giao nhau tại các đỉnh.

Output: Đưa ra chiều dài tối thiểu của bức tường. Nếu chiều dài là số thực thì lấy phần nguyên của nó



Input	Output
9 100 200 400 300 400 300 300 400 300 400 400 500 400 500 200 350 200 200 200	1628

Thuật toán: Ta sẽ tìm bao lồi của bức tường ban đầu, sau đó xây dựng bức tường bằng cách mở rộng bao lồi này ra một khoảng cách là L. Các khoảng trống còn lại chính là các cung tròn, có bán kính là L và góc ở tâm là pi/2, độ dài của mỗi cung tròn này là L*pi/2, do vậy tổng độ dài của bốn cung tròn là 4L*pi/2=2Lpi.

Độ phức tạp cả thuật toán là O(nlogn)

```
#include <bits/stdc++.h>

struct point
```

```
{
   double x, y;
   point(double x, double y) : x(x), y(y) { }
   point() {};
};

double dist(const point& a, const point& b)
{
   return std::sqrt((b.x-a.x)*(b.x-a.x)+(b.y-a.y)*(b.y-a.y));
}

bool rightTurn(const point& a, const point& b, const point& c)
{
   return (a.x-b.x)*(c.y-b.y)-(a.y-b.y)*(c.x-b.x) >= 0;
}

template <typename T> double calcHullPartLength(T begin, T end) // Graham scan
{   // s is the stack containing our tentative hull
   std::vector<point> s(std::distance(begin, end));
   int t; // Top of the stack
   for(t = 0; t < 2; t++)
      s[t] = *begin++;
   while(begin < end)
   {
      while(t >= 2 && !rightTurn(s[t-2], s[t-1], *begin))
```

```
        t--;        s[t++] = *begin++;
    }
    double ans = 0;
    for(int i = 1; i < t; i++)
        ans += dist(s[i], s[i-1]);
    return ans;
}

int main()
{
    int N, L;
    scanf("%d %d", &N, &L);
    std::vector<point> p;
    p.reserve(N);
    while(N--)
    {
        int x, y;
        scanf("%d %d", &x, &y);
        p.push_back(point(x, y));
    }
    std::sort(p.begin(), p.end(), [] (const point& a, const point& b)
                        { return (a.x == b.x) ? a.y < b.y : a.x < b.x; });
    double ans = calcHullPartLength(p.begin(), p.end());
    ans += calcHullPartLength(p.rbegin(), p.rend());
    printf("%.0lf", ans+4*acos(0.0)*L);    return 0;}
```

Link bộ test của đề bài:

https://drive.google.com/drive/folders/1IPXlpRsMBoT6aGjcs_Ps6X41fmyyUt9X?usp=sharing

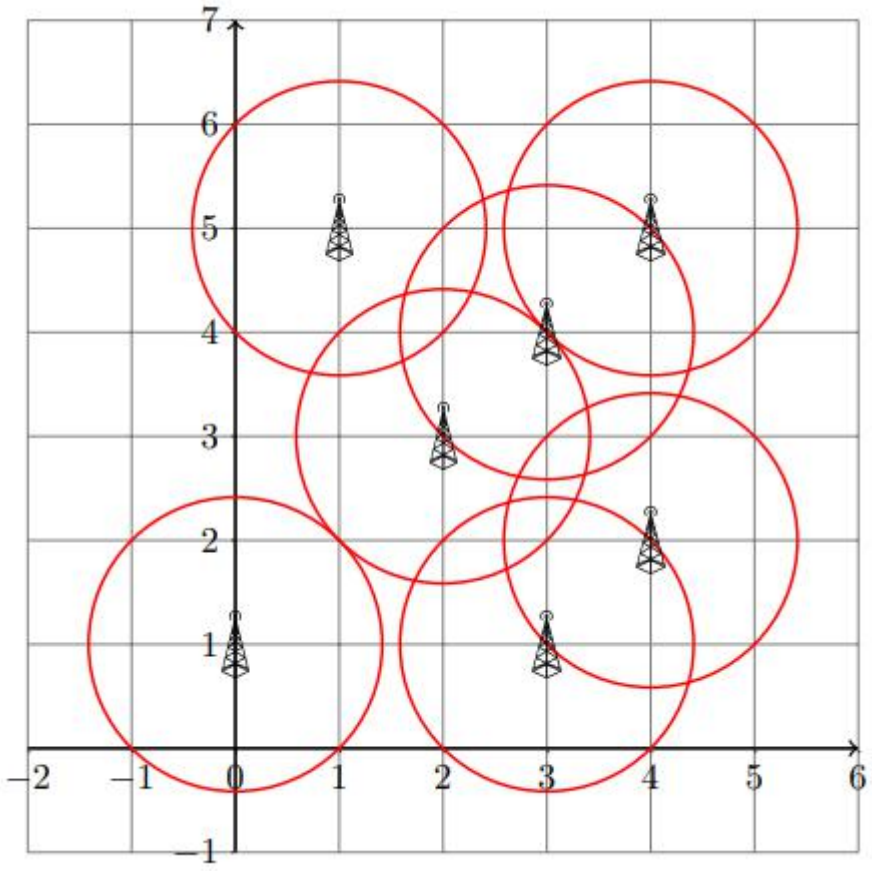
Bài 5. Kết nối thông tin

Nguyễn Chí Thanh là một nhà nghiên cứu thông tin. Anh đang được giao nhiệm vụ đảm bảo kết nối thông tin một cách tốt nhất giữa các tỉnh thành phố của quốc gia. Nhiệm vụ của Nguyễn Chí Thanh lần này là kết nối n ăng ten nằm rải rác trên các thành phố, các điểm đặt ăng ten có thể biểu diễn bằng một điểm trong hệ trục tọa độ oxy. Khi mỗi ăng ten phát sóng, nó sẽ phát sóng trong phạm vi là một hình tròn có bán kính không âm là r. Phạm vi của ăng ten được xác định là tập hợp tất cả các điểm có khoảng cách đến điểm đặt ăng ten tối đa là r. Nếu phạm vi của hai ăng ten có một điểm chung, các ăng ten đó là có thể giao tiếp trực tiếp với nhau. Nếu ăn ten A và B có thể giao tiếp trực tiếp với nhau, ăn ten B và C có thể giao tiếp trực tiếp với nhau, thì ăn ten A và C cũng có khả năng liên lạc với nhau, thông qua ăn ten B

Nhiệm vụ của Nguyễn Chí Thanh đó là tạo liên lạc giữa tất cả N ăng ten. Tuy nhiên, vì biết rằng kinh tế đất nước còn nghèo, việc mua được những ăng ten có khả năng kết nối xa là rất tốn tiền. Do vậy, Nguyễn Chí Thanh sẽ phải tìm cách chọn bán kính r tối thiểu để có thể để đảm bảo liên lạc giữa tất cả N ăng ten Em hãy giúp Nguyễn Chí Thanh giải quyết bài toán này cho quốc gia nhé

Input: Dòng đầu tiên gồm một số nguyên n (1<=n<=1000), số lượng ăng ten. Mỗi dòng trong số n dòng tiếp theo gồm các số nguyên xi và y I (0<=xi,yi<=10^9), đó là tọa độ của ăng ten thứ i

Output: Đưa ra bán kính r tối thiểu. Câu trả lời của bạn được coi là đúng nếu sai số tuyệt đối hoặc tương đối của nó không vượt quá 10^-6.



Thuật toán: Khoảng cách giữa hai đường tròn tiếp xúc ngoài là 2r. Do vậy, hai ăng ten có thể liên lạc trực tiếp với nhau nếu khoảng cách giữa chúng tối đa là 2r. Do vậy, ta có thể xây dựng một đồ thị bằng cách thêm các cạnh mà khoảng cách giữa chúng là nhỏ hơn hoặc bằng 2r. Sau đó ta sẽ dùng thuật toán dfs để kiểm tra đồ thị này có liên thông với nhau hay không. Do phải tìm bán kính r là nhỏ nhất, ta sẽ sử dụng tìm kiếm nhị phân, với bán kính r cao nhất là 10^9, nhỏ nhất là 0

```
#include<bits/stdc++.h>

using namespace std;

#define ll long long

#define pll pair<ll,ll>

double eps = 1e-7;

ll n, cnt;

pll a[1005];
```

```
bool kt[1005];


ll kc(pll x, pll y) {
   return (x.first - y.first) * (x.first - y.first) + (x.second - y.second) * (x.second - y.second);
}


void dfs(ll x, double r) {
   cnt++;
   kt[x] = 1;
   for (int i = 0; i < n; i++)
      if (i != x && !kt[i] && kc(a[x], a[i]) < (ll) 4 * r * r)
         dfs(i, r);
}


bool ok(double r) {
   memset(kt, 0, sizeof kt);
   cnt = 0;
   dfs(0, r);
   return cnt == n;
}


int main ()
{
   ios::sync_with_stdio(false);
```

```
    cin.tie(0);cout.tie(0);
    cin>>n;
    for (int i = 0; i < n; i++)
        cin>>a[i].first>>a[i].second;
    double kq = 0;
    double mn = 1e9;
    while (mn - kq > eps)
    {
        double mid = (kq + mn) / 2;
        if (ok(mid))
            mn = mid;
        else
            kq = mid;
    }
    cout<<fixed<<setprecision(7);
    cout<<kq;
    return 0;
}
```

Link bộ test của đề bài:

https://drive.google.com/drive/folders/1IPXlpRsMBoT6aGjcs_Ps6X41fmyyUt9X?usp=sharing

Bài 6. Diện tích của hợp các hình chữ nhật

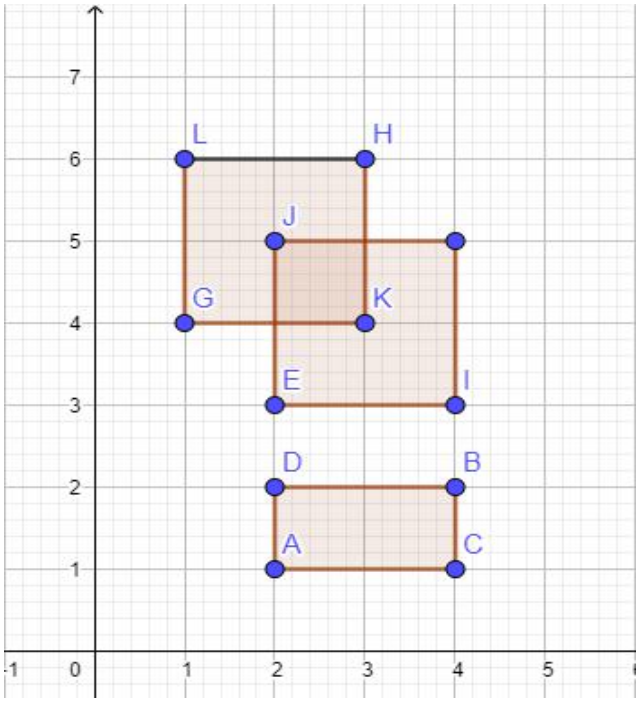
Diện tích của hình chữ nhật: Cho một tập gồm N hình chữ nhật có các cạnh song song với trục tọa độ, bạn cần tìm diện tích của hợp của chúng. Mỗi hình chữ nhật biểu thị bởi hai điểm, một điểm góc dưới bên trái, một điểm có tọa độ ở góc trên bên phải. Các tọa độ là các số nguyên

Input: Dòng đầu tiên gồm N biểu thị số hình chữ nhật N dòng tiếp theo bao gồm x1, y1, x2, y2 trong đó x1, y1 là tọa độ của điểm dưới bên trái và x2, y2 là tọa độ điểm trên bên phải của hình chữ nhật

Output: Đưa ra diện tích của hợp của các hình chữ nhật

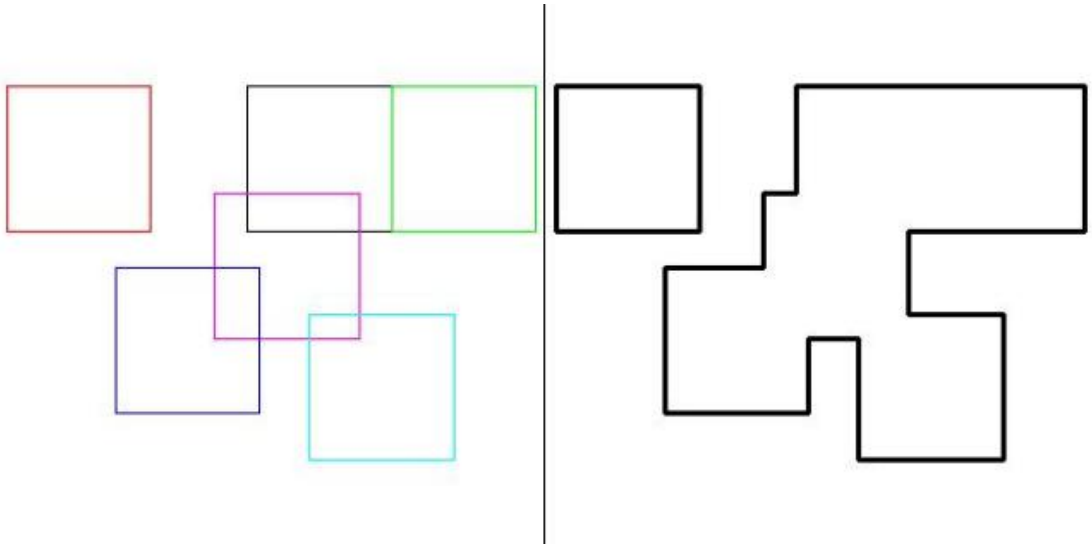
Giới hạn:

$1 \le n \le 10^4, 1 \le x_1 < x_2 \le 10^5, 1 \le y_1 < y_2 \le 10^5$



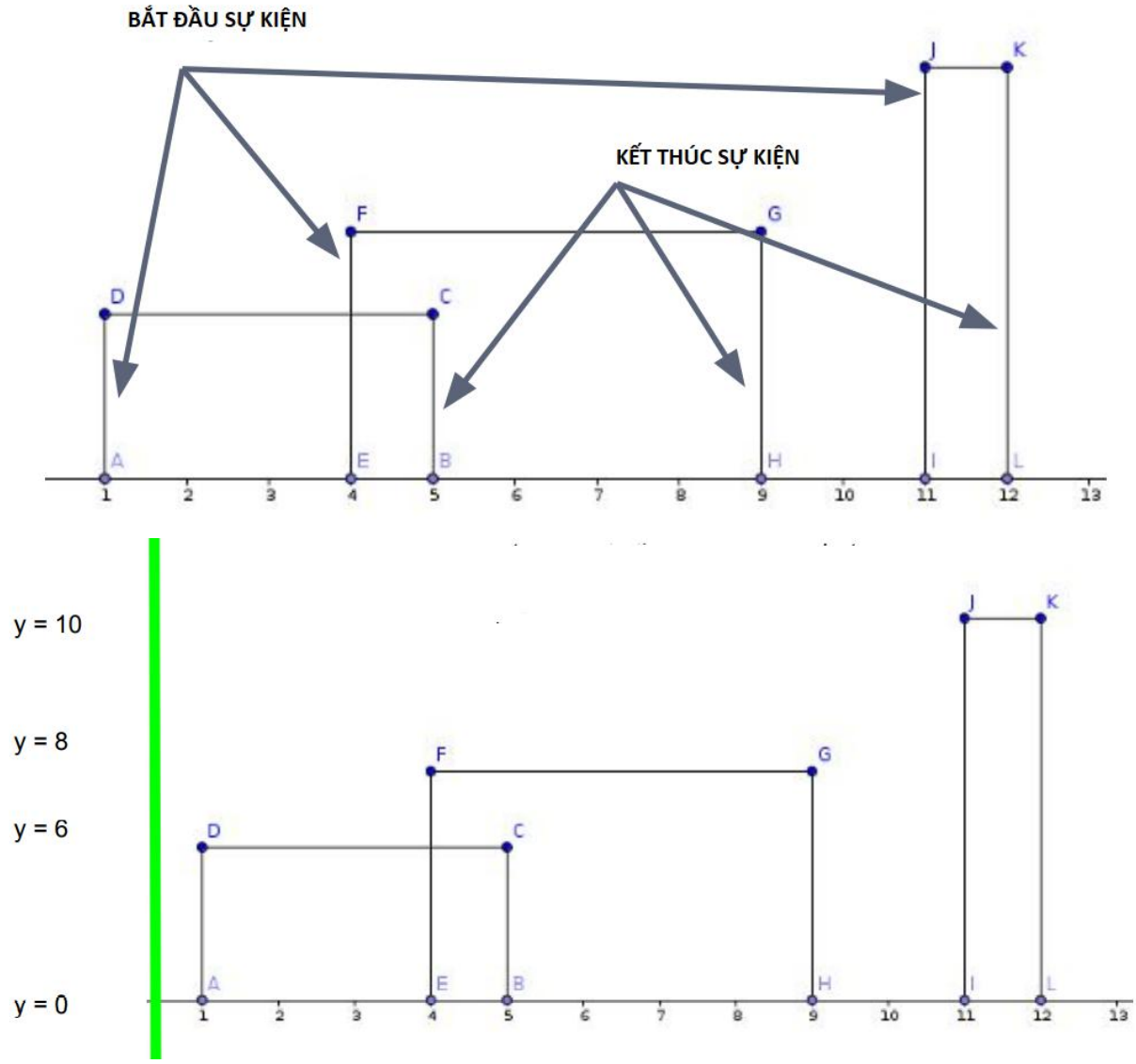
Input	Output
3 2 1 4 2 2 3 4 5 1 4 3 6	9

Thuật toán:

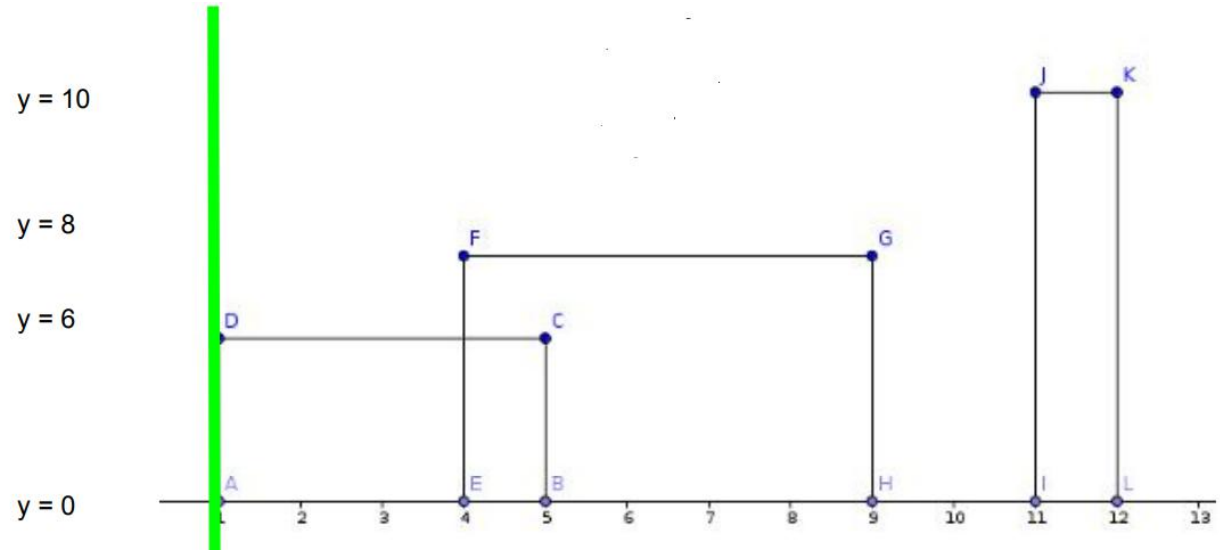


Hình minh họa hợp của các hình chữ nhật

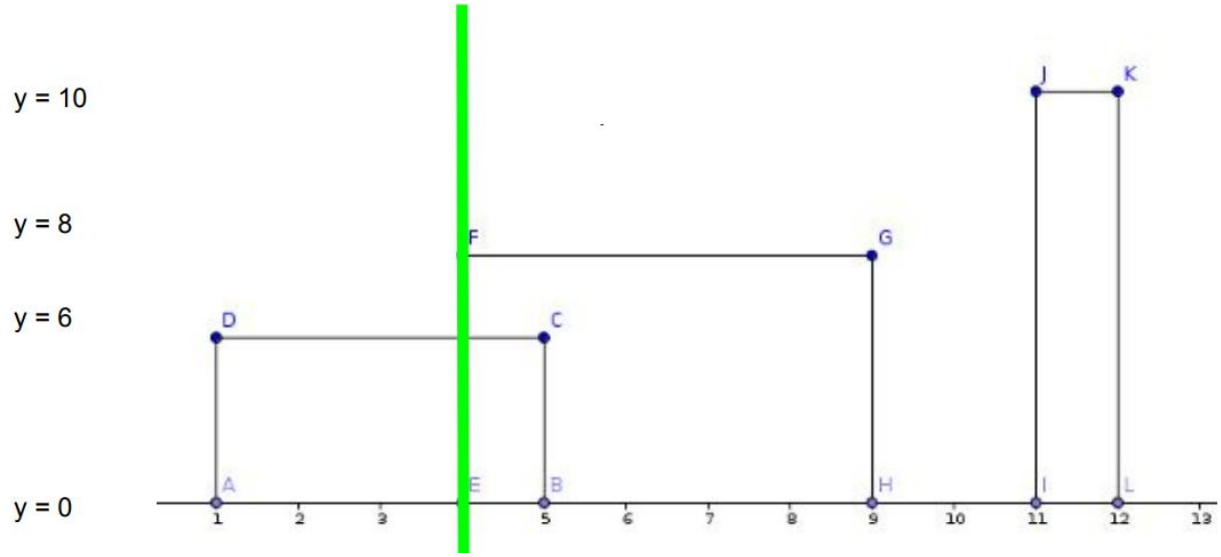
Bài này ta sẽ sử dụng kỹ thuật sweep line (đường thẳng quét), cho đường thẳng đứng quét từ trái qua phải, gặp cạnh đầu tiên bên trái của một hình chữ nhật được gọi là bắt đầu một sự kiện, và khi gặp cạnh bên phải của hình chữ nhật được gọi là kết thúc một sự kiện như hình minh họa dưới đây



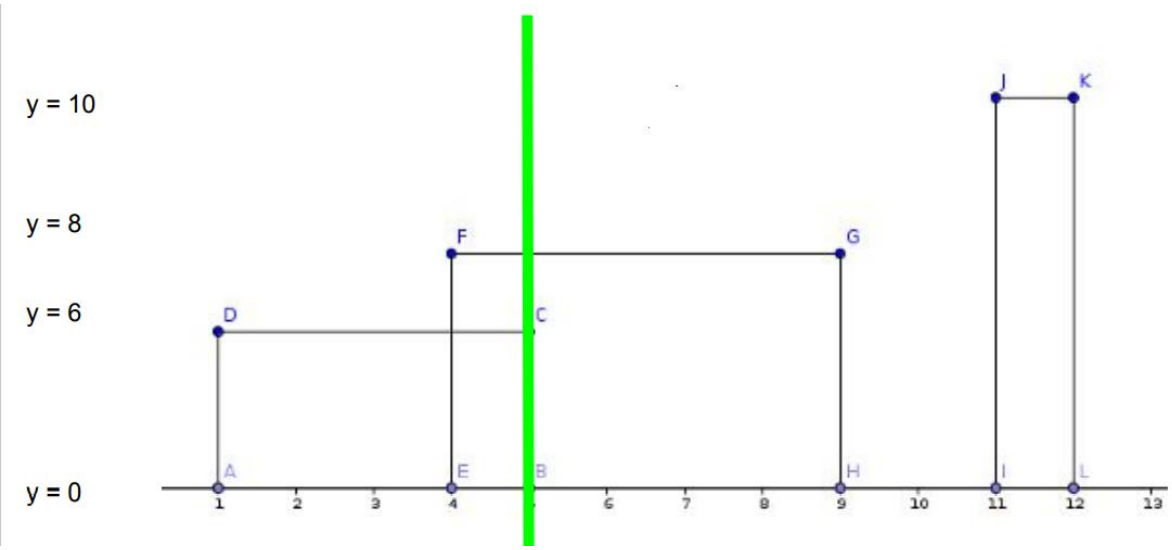
Hình minh họa đường thẳng quét màu xanh sẽ duyệt từ trái qua phải, sau đó tính toán sự chênh lệch về tọa độ x giữa những đường thẳng quét. Sau đó ta sẽ tìm y max. Để tính diện tích, ta tìm tích giữa độ chênh lệch về x giữa hai đường thẳng quét nhân với y max, hay viết gọn lại là x_diff*y_max



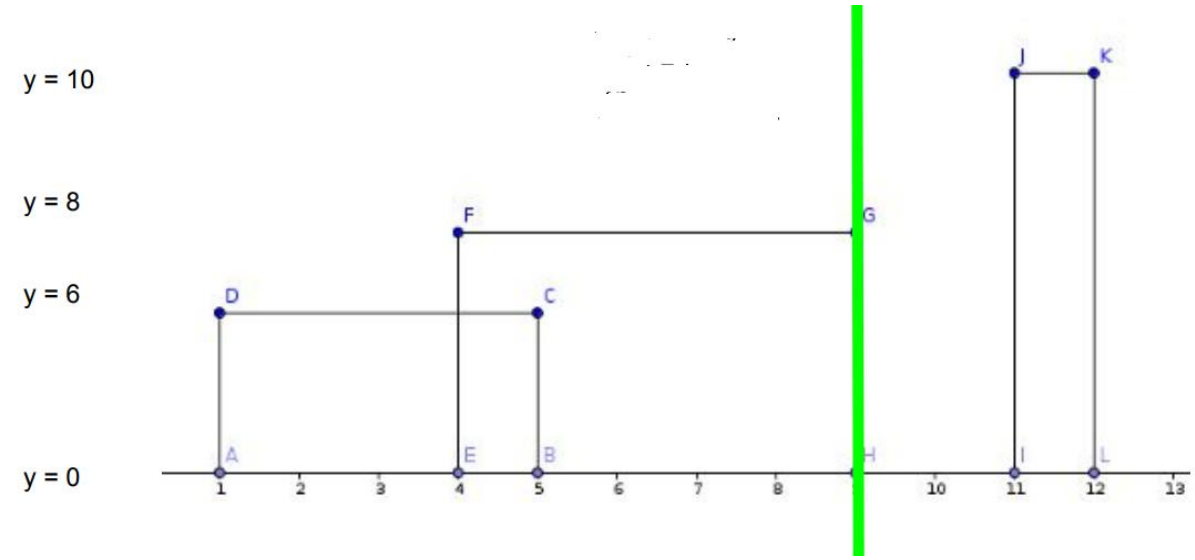
Trong hình minh họa ở trên, đường thẳng quét bắt đầu tại x=1, lúc này diện tích bằng 0, tập các hình chữ nhật đang xét đến lúc này là rỗng {}, ta bắt đầu bổ sung hình chữ nhật r1 vào tập hợp



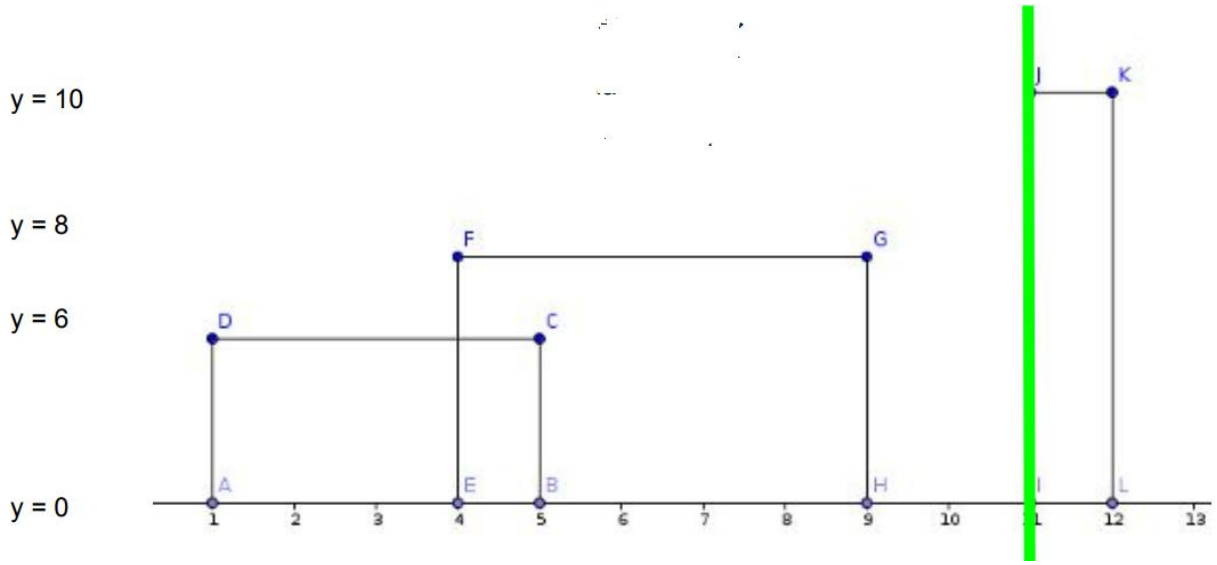
Trong hình minh họa ở trên, tập các hình chữ nhật đang xét đến là r1, đường thẳng quét đến x=4, bắt đầu cho hình chữ nhật r2, trước đó đường thẳng quét ở vị trí x=1, ta sẽ bổ sung hình chữ nhật r2 vào tập các hình chữ nhật đang xét đến. Khi đó, tập hợp là {r1,r2}. Diện tích của hình chữ nhật được công thêm một khoảng bằng (4-1)*6 với 4 và 1 là chênh lệch tọa độ x giữa hai đường thẳng quét,



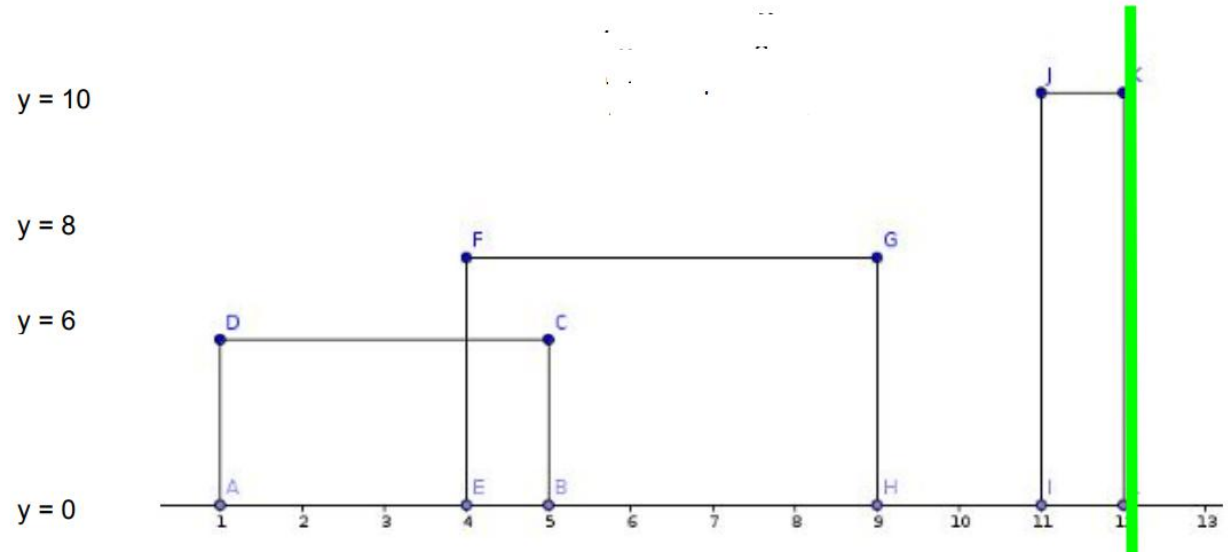
Trong hình minh họa ở trên, tập các hình chữ nhật đang xét đến là {r1,r2}, đường thẳng quét đến x=5, bắt đầu cho việc rời khỏi hình chữ nhật r1, trước đó đường thẳng quét ở vị trí x=4, ta sẽ xóa hình chữ nhật r1 khỏi tập các hình chữ nhật đang xét đến. Diện tích của hình chữ nhật được công thêm một khoảng bằng (5-4)*max(6,8)



Trong hình minh họa ở trên, tập các hình chữ nhật đang xét đến là {r2}, đường thẳng quét đến x=9, bắt đầu cho việc rời khỏi hình chữ nhật r2, trước đó đường thẳng quét ở vị trí x=5, ta sẽ xóa hình chữ nhật r2 ra khỏi tập các hình chữ nhật đang xét đến. Khi Diện tích của hình chữ nhật được công thêm một khoảng bằng (9-5)*8



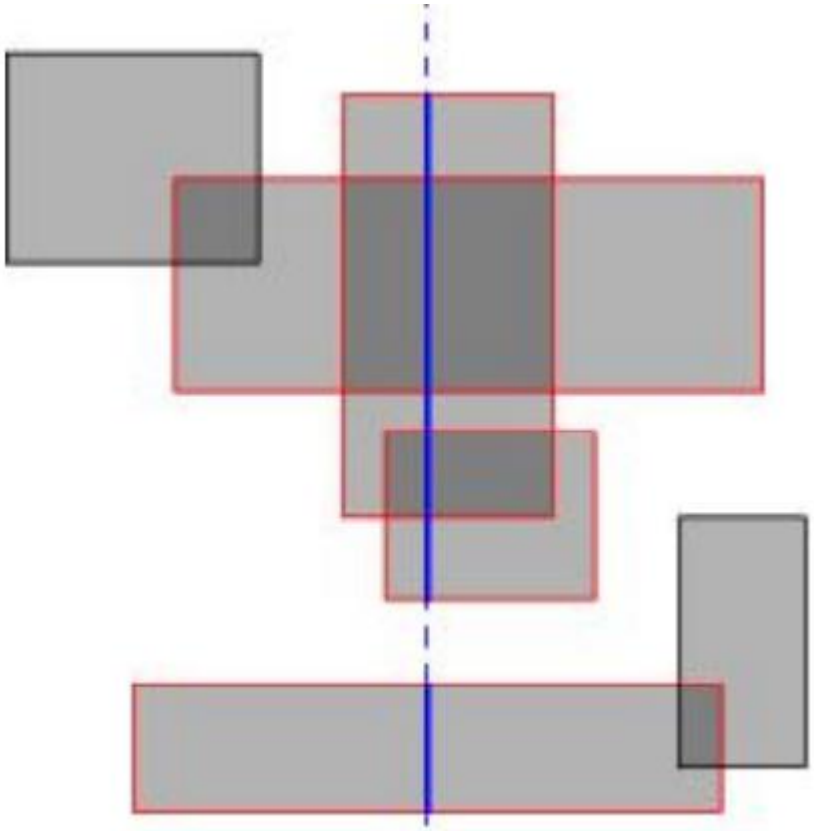
Trong hình minh họa ở trên, tập các hình chữ nhật đang xét đến là rỗng {}, đường thẳng quét đến x=11, trước đó đường thẳng quét ở vị trí x=9, ta sẽ thêm hình chữ nhật r3 tới tập các hình chữ nhật đang được xét đến. Khi đó diện tích của đa giác được cộng thêm một lượng là 0



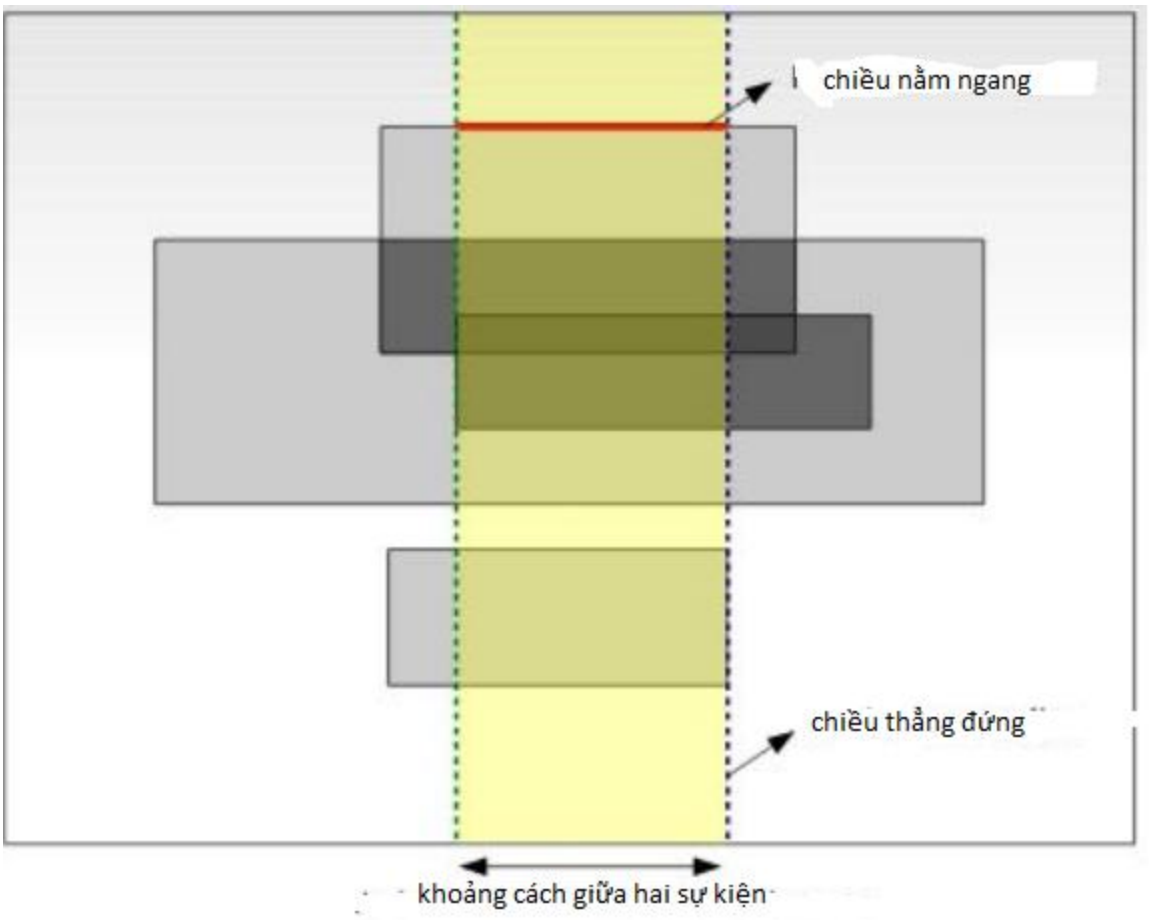
Trong hình minh họa ở trên, tập các hình chữ nhật đang xét đến là {r3}, đường thẳng quét đến x=12, bắt đầu cho sự kiện rời bỏ hình chữ nhật r3, trước đó đường thẳng quét ở vị trí x=11, ta sẽ loại bỏ hình chữ nhật r3 tới tập các hình chữ nhật đang được xét đến. Khi đó diện tích của đa giác được cộng thêm một lượng là (12-11)*10

Thuật toán ở trên là một trường hợp đơn giản cho bài toán hình chữ nhật. Trong tình huống tổng quát, có thể có nhiều hình chữ nhật ở phía trên. Khi đó, ta có bài toán con: Cho một tập hình chữ nhật, tìm chiều cao phủ bởi các hình chữ nhật từ trên

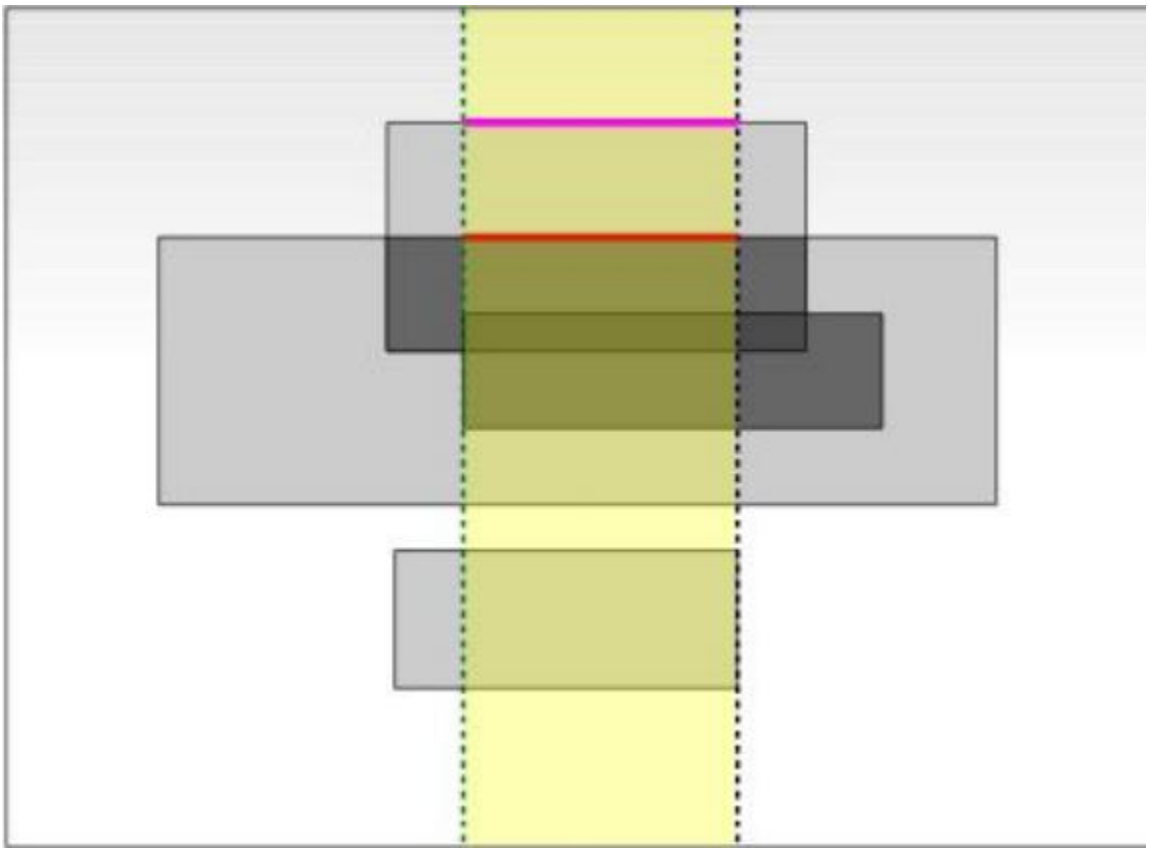
xuống dưới. Khi đó ta sẽ dùng đường thẳng quét từ trên xuống dưới, sau đó ta sẽ xác định khoảng cách giữa các hình chữ nhật bằng cách đếm xem có bao nhiêu hình chữ nhật ở trong đó.



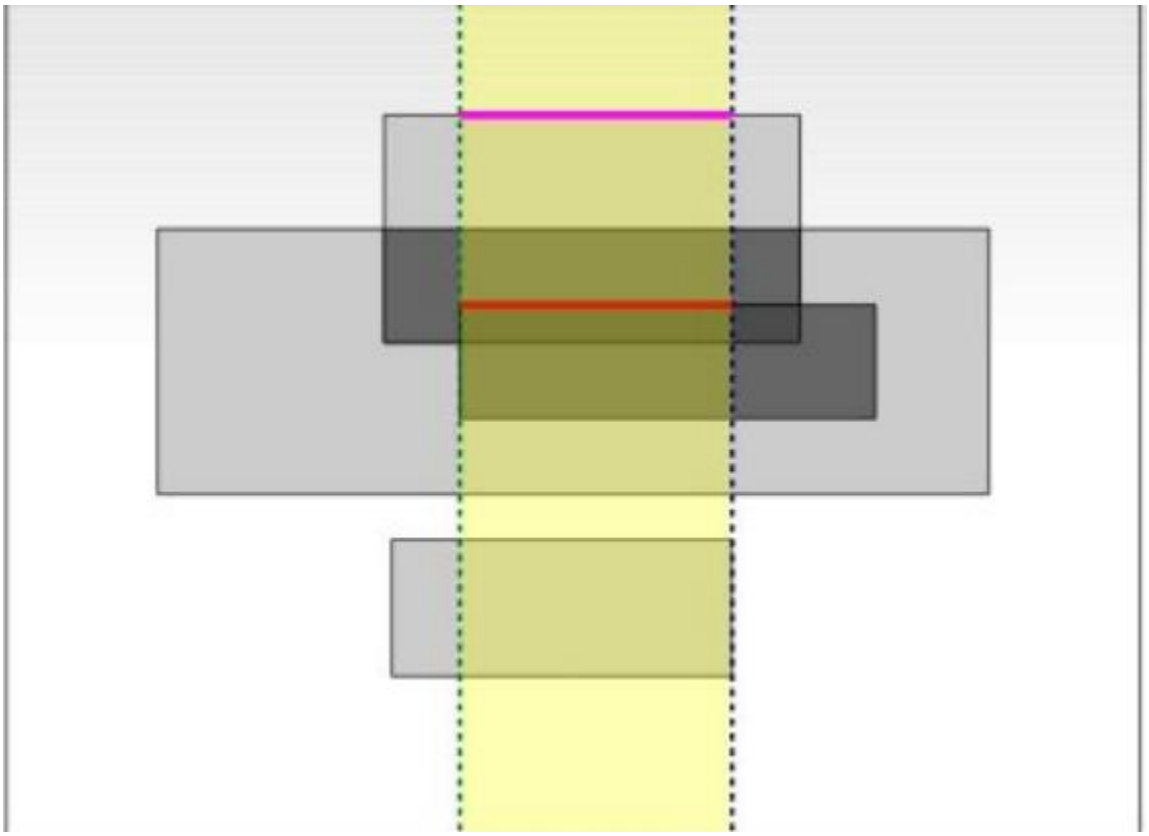
Ta sẽ quét theo chiều ngang và đếm xem có bao nhiêu hình chữ nhật ta có. Nếu bắt đầu một hình chữ nhật mới theo chiều ngang, ta sẽ tăng biến đếm số lượng các hình chữ nhật lên (dem++). Ta gọi First_Y là hình chữ nhật đầu tiên. Nếu kết thúc một hình chữ nhật mới theo chiều ngang, ta sẽ giảm biến đếm xuống (dem--). Nếu biến đếm sau khi giảm đúng bằng 0, nghĩa là lúc đó chúng ta phải đi tính khoảng cách Y_difference=First_Y-Current_Y



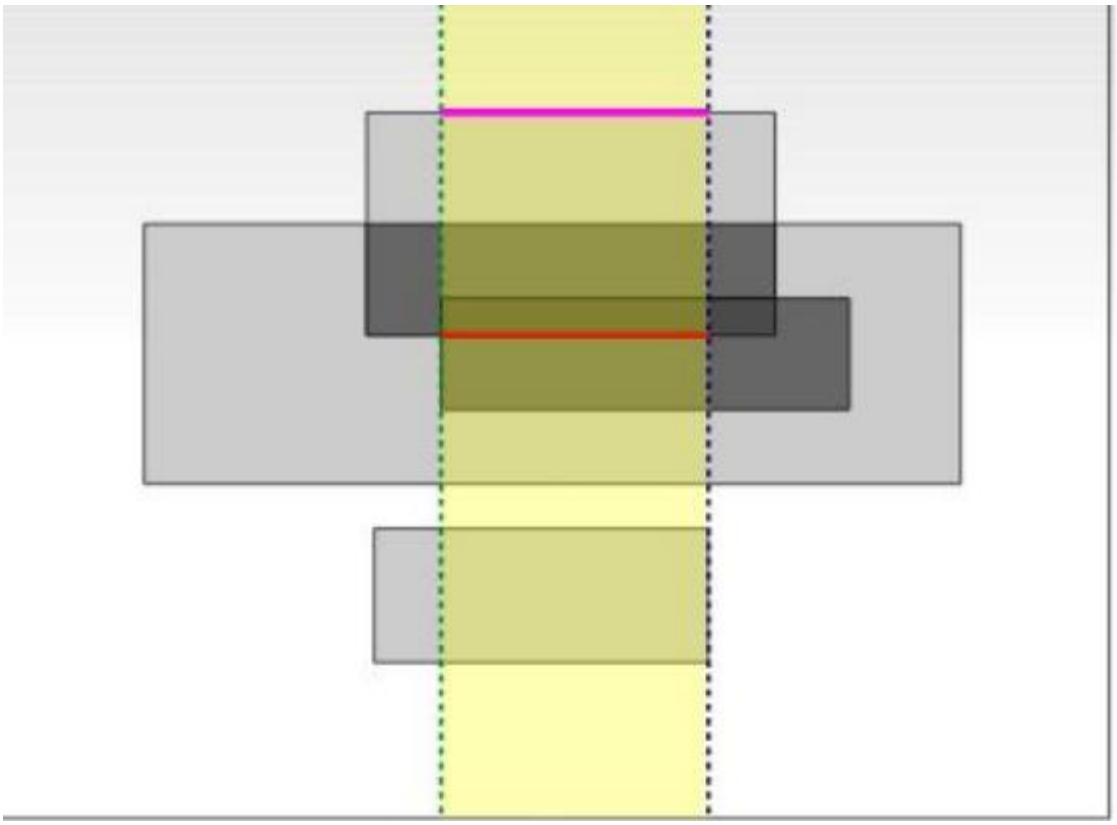
Đường màu đỏ thể hiện cho việc ta đang quét các hình chữ nhật từ trên xuống dưới, lúc này biến đếm số lượng các hình chữ nhật đang là 1



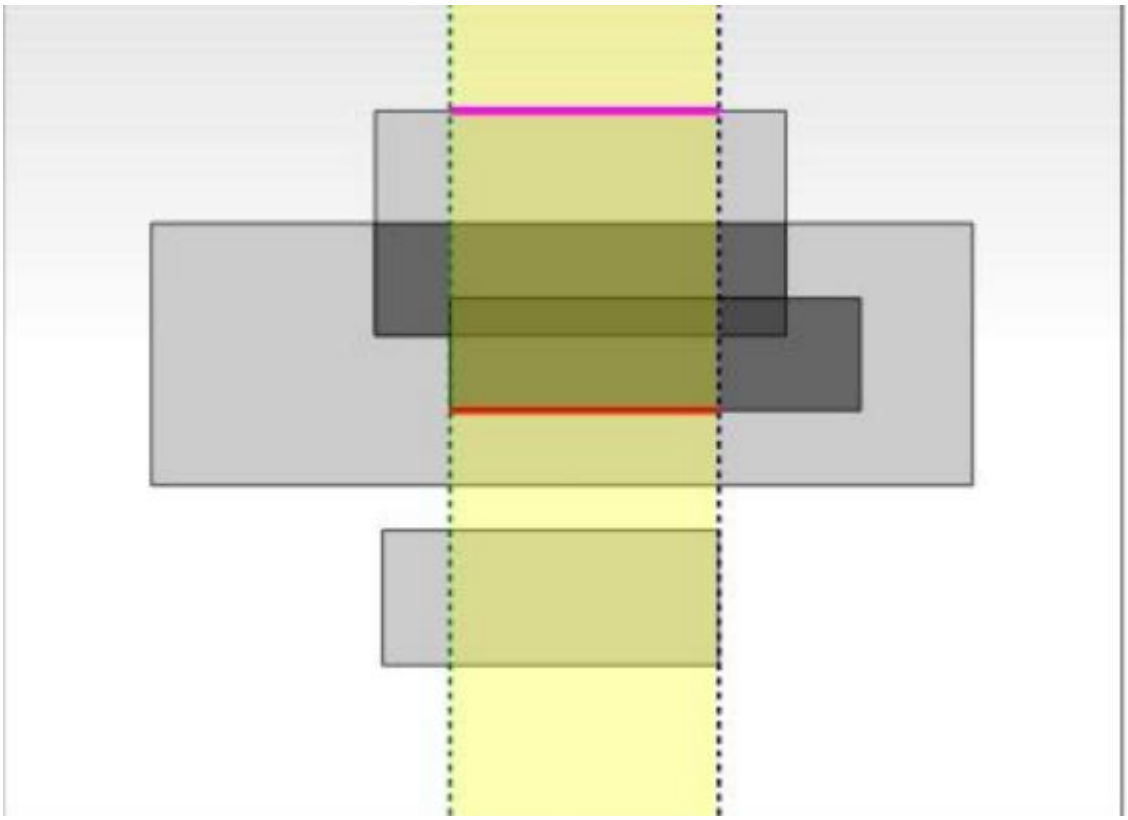
Đường màu đỏ nằm ngang tiếp tục di chuyển xuống dưới, lúc này gặp hình chữ nhật tiếp theo, ta tăng số lượng hình chữ nhật lên 1, hay biến đếm lúc này bằng 2



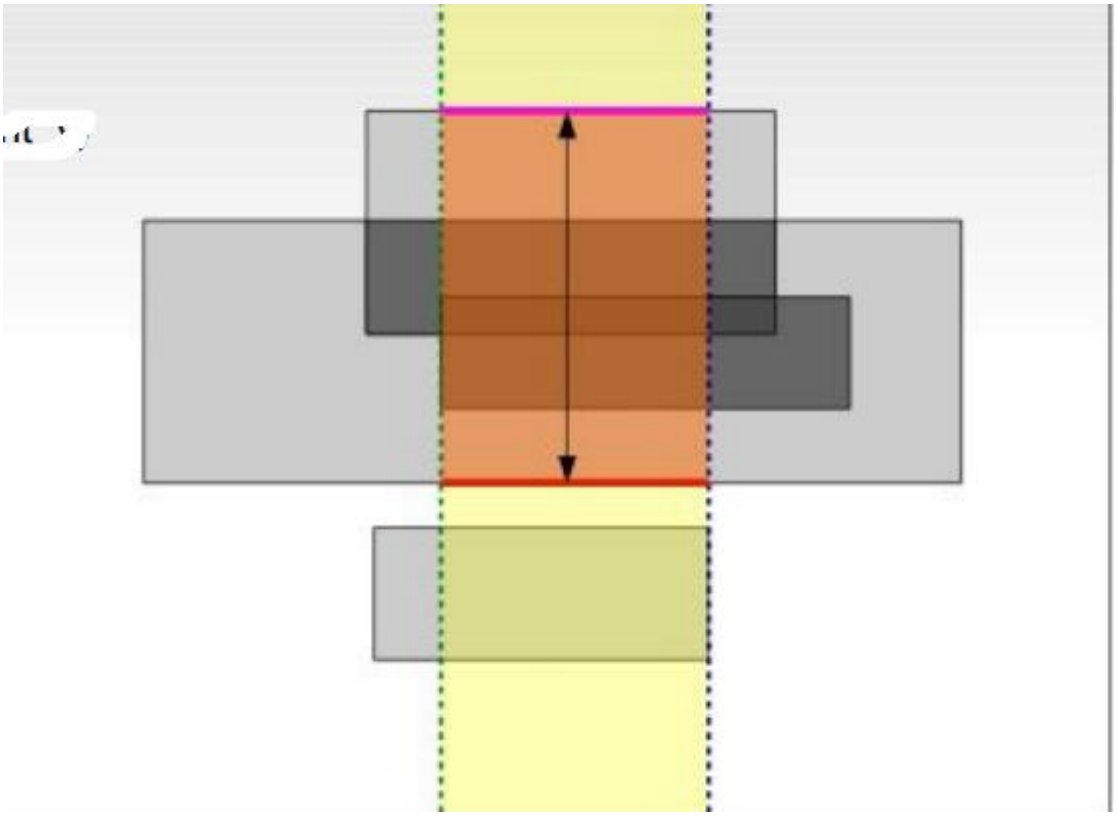
Đường màu đỏ nằm ngang tiếp tục di chuyển xuống dưới, lúc này gặp hình chữ nhật tiếp theo, ta tăng số lượng hình chữ nhật lên 1, hay biến đếm lúc này bằng 3



Đường màu đỏ nằm ngang tiếp tục di chuyển xuống dưới, lúc này ta rời khỏi hình chữ nhật trước đó, ta giảm số lượng hình chữ nhật đi 1, hay biến đếm lúc này bằng 2

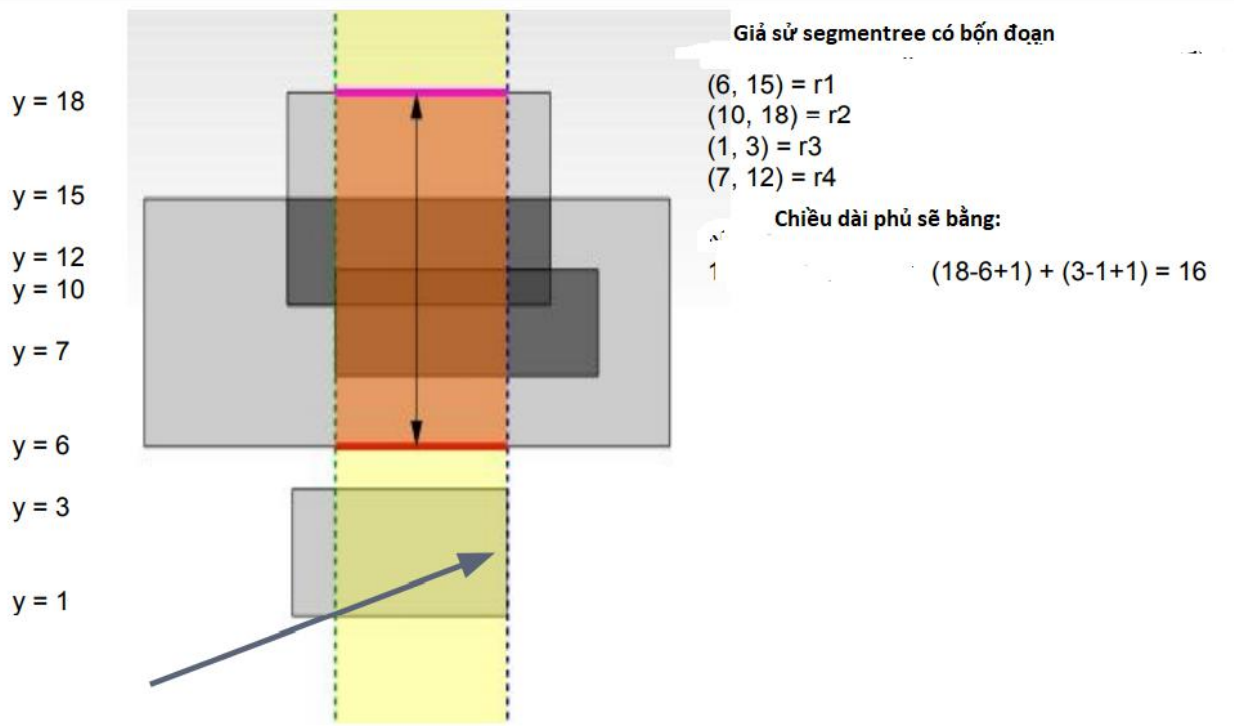


Đường màu đỏ nằm ngang tiếp tục di chuyển xuống dưới, lúc này ta rời khỏi hình chữ nhật trước đó, ta giảm số lượng hình chữ nhật đi 1, hay biến đếm lúc này bằng 1



Đường màu đỏ nằm ngang tiếp tục di chuyển xuống dưới, lúc này ta rời khỏi hình chữ nhật trước đó, ta giảm số lượng hình chữ nhật đi 1, hay biến đếm lúc này bằng 0. Lúc này diện tích sẽ được cộng thêm một lượng là delta_x*(first_y-current_y)

Như vậy khi quét theo chiều thẳng đứng, ta thao tác trên các các hình chữ nhật hay các khoảng đoạn của y, với y_start và y_end là bắt đầu và kết thúc cho một đoạn nào đó. Bài toán có thể quy về việc cho một vài khoảng [y_start, y_end], ta cần biết tổng chiều dài phủ trên đoạn đó là bao nhiêu?



Như vậy ta có thể thấy rằng trong từng đoạn ta có thể thêm 1 nếu gặp hình chữ nhật mới hoặc trong đoạn đó ta có thể bớt 1 đến rời khỏi hình chữ nhật mới. Khi đó ta quy về bài toán sử dụng cây phân đoạn (Segmentree) với việc cập nhật trên một đoạn, và ta có thể rút gọn bằng phương pháp Lazy Propagation.

```
#include <bits/stdc++.h>

using namespace std;

struct Edge

{

   bool open;

   int x, ymin, ymax;

   bool operator < (const Edge &e) const

   {

      return (x < e.x);

   }

};
```

```
vector <Edge> edges;
int x1, x2, y_1, y2, n, m;
long long sum[1000001], h[30001], dem[1000001];
void update (int v, int l, int r, int ymin, int ymax, bool open)
{
   if (ymax < h[l] || ymin > h[r]) return;
   if (ymin <= h[l] && h[r] <= ymax)
   {
      dem[v] += (open)? 1: -1;
      if (dem[v]) sum[v] = h[r] - h[l];
      else sum[v] = sum[v * 2] + sum[v * 2 + 1];
      return;
   }
   if (l + 1 >= r) return;
   int mid = (l + r) / 2;
   update (v * 2, l, mid, ymin, ymax, open);
   update (v * 2 + 1, mid, r, ymin, ymax, open);
   if (dem[v]) sum[v] = h[r] - h[l];
   else sum[v] = sum[v * 2] + sum[v * 2 + 1];
}
long long solve (long long h[], int m)
{
   long long ans = 0;
   int k = 1;
   update (1, 1, m, edges[0].ymin, edges[0].ymax, edges[0].open);
```

```
    for (int i = 1; i < edges.size(); i++)
    {
        ans = ans + sum[1] * (edges[i].x - edges[i - 1].x);
        update (1, 1, m, edges[i].ymin, edges[i].ymax, edges[i].open);
    }
    return ans;
}
int main()
{
    cin >> n;
    int m = 0;
    for (int i = 1; i <= n; i++)
    {
        cin >> x1 >> y_1 >> x2 >> y2;
        Edge e1, e2;
        e1.x = min (x1, x2); e1.ymin = min (y_1, y2); e1.ymax = max (y_1, y2); e1.open
= 1;
        e2.x = max (x1, x2); e2.ymin = min (y_1, y2); e2.ymax = max (y_1, y2); e2.open
= 0;
        edges.push_back (e1);
        edges.push_back (e2);
        h[++m] = y_1;
        h[++m] = y2;
    }
    sort (edges.begin(), edges.end());
    sort (h + 1, h + 1 + m);
```

```
    //for (int i = 1; i <= m; i++) cout << h[i] << " ";
    cout << solve (h, m);
    return 0;
}
```

Bài 7. Giao điểm giữa các đoạn thẳng

Cho n đoạn thẳng nằm ngang và thẳng đứng, nhiệm vụ của bạn là tính số lượng giao điểm của chúng. Bạn có thể giả sử rằng không không có hai đoạn thẳng song song nào giao nhau và không có điểm cuối của đoạn thẳng nào là giao điểm giữa hai đoạn thẳng

Input: Dòng đầu tiên gồm số nguyên n: số lượng đoạn thẳng. Có n dòng mô tả các đoạn thẳng. Mỗi dòng gồm bốn số nguyên x1,y1,x2,y2: một đoạn thẳng bắt đầu ở điểm (x1,y) và kết thúc ở điểm (x2,y2)

Output: Đưa ra số lượng giao điểm

Input	Output
3 2 3 7 3 3 1 3 5 6 2 6 6	2

Giới hạn: $1 \le n \le 10^5, -10^6 \le x_1 \le x_2 \le 10^6, -10^6 \le y_1 \le y_2 \le 10^6, (x_1, y_1) \ne (x_2, y_2)$

Cách 1. Độ phức tạp $O(n^2)$, ta sẽ lần lượt tìm giao điểm của từng cặp đường thẳng, với mỗi lần tìm được giao điểm giữa hai đường thẳng ta sẽ cộng 1 vào câu trả lời của bài toán. Sau đây là đoạn cài đặt chương trình:

```
#include <bits/stdc++.h>


using namespace std;
struct xx
```

```
{
   long a,c,d;
};
vector<xx> thang,ngang;
long long n,res=0;
int main()
{
   ios_base::sync_with_stdio(0);
   cin.tie(0);cout.tie(0);
  cin>>n;
  for(int i=1;i<=n;i++)
  {
     long a,b,c,d;
     cin>>a>>b>>c>>d;
     if(a==c)
     {
        xx u;
        u.a=a;u.c=b;u.d=d;
        if(u.c>u.d) swap(u.c,u.d);
        thang.push_back(u);
     }
     else
     {
        xx u;
        u.a=b;u.c=a;u.d=c;
```

```
            if(u.c>u.d) swap(u.c,u.d);
            ngang.push_back(u);
        }
    }
    for(xx a:thang)
    {
        for( xx b:ngang)
            if( (a.a>b.c&&a.a<b.d)&&(b.a>a.c&&b.a<a.d) )++res;
    }
    cout<<res;
    return 0;
}
```

Cách 2: Độ phức tạp O(nlogn)

Ta sẽ sử dụng tư tưởng của kỹ thuật sweep line như đã trình bày ở bài toán tìm hợp của các hình chữ nhật. Ban đầu ta sẽ sắp xếp các đoạn thẳng theo chiều dọc, sắp xếp các đoạn thẳng theo thứ tự tăng dần về chiều ngang, sau đó ta sẽ duyệt theo chiều dọc và chiều ngang tương ứng để quyết định số lượng giao điểm trong đoạn dọc. Khi đó các đoạn dọc giống như các đoạn ta cần quản lý trong cấu trúc dữ liệu Segment tree (Với tư tưởng giống như đã phân tích ở bài toán tìm hợp của các hình chữ nhật). Bài toán này khi dạy đạt được lợi ích sau:

- Một là củng cố cài đặt cấu trúc dữ liệu cây phân đoạn, và hiểu được ứng dụng của cây phân đoạn trong các bài toán tìm giao điểm giữa các hình,…
- Hai là có thể phát triển rất tốt tư duy học sinh về kỹ thuật sweep line sau khi dạy bài toán tìm hợp của các hình chữ nhật (đây là một bài toán rất khó).

Sau đây là hai phần cài đặt bằng cách sử dụng cây phân đoạn Segmentree và Binary Index Tree:

```
#include <bits/stdc++.h>
using namespace std;
```

```
struct Point
{
   bool open = 0;
   long long x, y;
   bool operator < (const Point &p) const
   {
      if (x == p.x) return y < p.y;
      return x < p.x;
   }
};
long long n, p1, p2, ans;
vector <Point> ngang, doc;
Point a, b;
long long t[10000007];
void update (long long v, long long l, long long r, long long k, bool openn)
{
   if (k < l || k > r) return;
   if (l == r)
   {
      t[v] = (openn == 1)? 1 : 0;
      return;
   }
   long long mid = (l + r) / 2;
   update (v * 2, l, mid, k, openn);
   update (v * 2 + 1, mid + 1, r, k, openn);
```

```
    t[v] = t[v * 2] + t[v * 2 + 1];
}
long long get (long long v, long long l, long long r, long long c, long long d)
{
    if (c > r || d < l) return 0;
    if (c <= l && r <= d) return t[v];
    long long mid = (l + r) / 2;
    return get (v * 2, l, mid, c, d) + get (v * 2 + 1, mid + 1, r, c, d);
}
int main()
{
    cin >> n;
    for (long long i = 1; i <= n; i++)
    {
        cin >> a.x >> a.y >> b.x >> b.y;
        a.x += 1000005;
        a.y += 1000005;
        b.x += 1000005;
        b.y += 1000005;
        if (a.x == b.x)
        {
            doc.push_back (a);
            doc.push_back (b);
        }
        else
```

```
        {
            if (a.x < b.x) a.open = 1; else b.open = 1;
            ngang.push_back (a);
            ngang.push_back (b);
        }
    }
    sort (doc.begin(), doc.end());
    sort (ngang.begin(), ngang.end());

    while (p1 < ngang.size() && p2 < doc.size())
    {
        while (p1 < ngang.size() && ngang[p1].x <= doc[p2].x)
        {
            update (1, 1, 2000015, ngang[p1].y, ngang[p1].open);
            p1++;
        }
        ans += get (1, 1, 2000015, doc[p2].y, doc[p2 + 1].y);
        p2 += 2;
    }
    cout << ans;
    return 0;
}
```

Cách cài đặt khác sử dụng Binary Index Tree:

```
#include <bits/stdc++.h>
using namespace std;
```

```
#define ll long long
struct diem
{
   ll x1, y1, x2, y2;
};
diem a[100002], b[100002], c[100002];
ll k, p1, h1, p2, h2, ans;
ll n, m, tmp = 1e6;
ll bit[2000002];
bool ss1(diem x, diem y)
{
   return x.x1 < y.x1;
}
bool ss2(diem x, diem y)
{
   return x.x2 < y.x2;
}
void update(int id, int so)
{
   for(int k = id; k <= tmp*2+1; k += (k&(-k)))
      bit[k] += so;
}
int getsum(int id)
{
```

```
    ll sum = 0;
    while(id > 0)
    {
        sum += bit[id];
        id -= (id&(-id));
    }
    return sum;
}
int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> k;
    for(int i = 1; i <= k; ++i)
    {
        cin >> p1 >> h1 >> p2 >> h2;
        if(h1 == h2)
        {
            a[++n].x1 = p1;
            a[n].y1 = h1;
            a[n].x2 = p2;
            a[n].y2 = h2;
            c[n].x1 = p1; c[n].y1 = h1; c[n].x2 = p2; c[n].y2 = h2;
        }
```

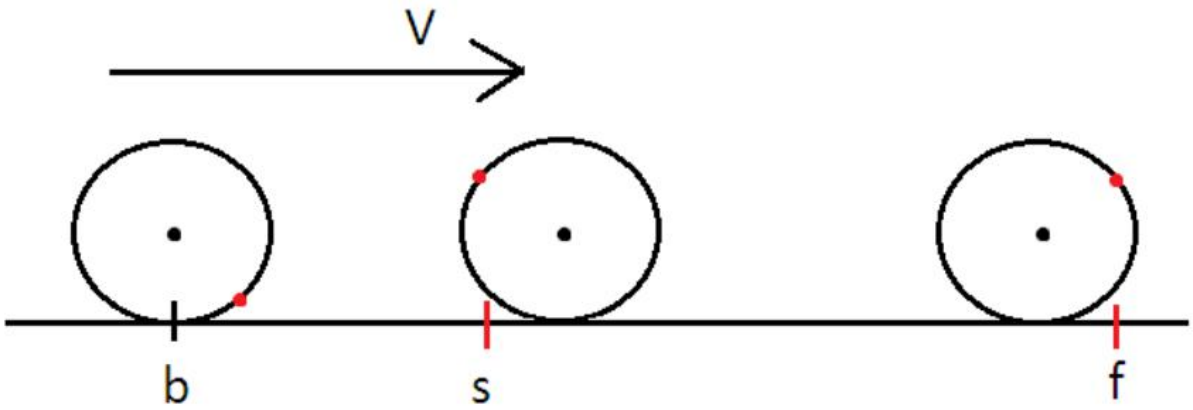
```
    else if(p1 == p2)
    {
        b[++m].x1 = p1;
        b[m].y1 = h1;
        b[m].x2 = p2;
        b[m].y2 = h2;
    }
}
sort(a+1, a+1+n, ss1);
sort(c+1, c+1+n, ss2);
sort(b+1, b+1+m, ss1);
int d = 1, h = 1, t = 1;
for(int x = -1e6; x <= 1e6; ++x)
{
    while(x == a[d].x1 && d <= n)
    {
        update(a[d].y1+tmp+1, 1);
        d++;
    }
    while(x == b[t].x1 && t <= m)
    {
        ans += getsum(b[t].y2+tmp+1)-getsum(b[t].y1-1+tmp+1);
        t++;
    }
    while(x == c[h].x2 && h <= n)
```

```
        {
            update(c[h].y1+tmp+1, -1);
            h++;
        }
    }
    cout << ans;
    return 0;
}
```

Bài 8. Cuộc đua xe đạp

Có n cuộc đua xe đạp. Trong cuộc đua xe đạp thứ i, người lái xe sẽ phải lái càng nhanh càng tốt từ điểm s_i đến điểm f_i ($s_i < f_i$). Việc đo thời gian trong cuộc thi xe đạp là một quá trình rất phức tạp liên quan đến việc sử dụng đúng cảm biến thời gian. Hãy tưởng tượng bánh trước xe đạp của người tham gia là một hình tròn bán kính là r. Ta tạm thời bỏ qua độ dày của lốp, kích thước xe đạp, độ trượt của bánh xe và các tác động vật lý khác. Cảm biến của bánh xe được đặt trên vành của bánh xe, tức là tại một điểm cố định nào đó trên đường tròn bán kính r. Sau đó, cảm biến di chuyển theo bánh xe.

Để bắt đầu, mỗi người tham giác có thể chọn bất kỳ điểm b_i sao cho xe của anh ta nằm hoàn toàn sau vạch xuất phát, tức là $b_i < s_i - r$. Sau đó, xe bắt đầu di chuyển, ngay lập tức tăng tốc đến tốc độ tối đa của nó, và tại thời điểm ts_i, khi cảm biến tọa độ trùng với điểm bắt đầu, thì bắt đầu tính thời gian, và ngay sau khi tọa độ cảm biến trùng với tọa độ kết thúc (thời gian tf_i), cảm biến sẽ tính tổng thời gian, và người đi xe đạp sẽ đi trong khoảng thời gian $tf_i - ts_i$.



Trong cuộc đua xe đạp đường trường nữ Olympic Tokyo, nữ tiến sỹ toán học Anna Kiesenhofer đã giành giải vô địch. Trước cuộc đua cô đã dự đoán rằng kết quả của cuộc đua không chỉ phụ thuộc vào tốc độ tối đa v của cô, mà còn phụ thuộc vào việc lựa chọn điểm xuất phát b_i. Yêu cầu đặt ra là xác định thời gian nhỏ nhất mà cô có thể đi được quãng đường từ vị trí s_i đến vị trí f_i.

Input: Dòng đầu tiên gồm ba số nguyên n, r và v (1<=n<=100000, 1<=r,v<=10^9), số lần thi đấu, bán kính của xe đạp và tốc độ tối đa của tay đua xe đạp.

N dòng tiếp theo chưa mô tả về các cuộc tih. Dòng thứ I gồm hai số nguyên s_i và f_i ($1<=s_i<f_i<10^9$), tọa độ điểm bắt đầu và tọa độ điểm kết thúc của cuộc thi thứ i tương ứng

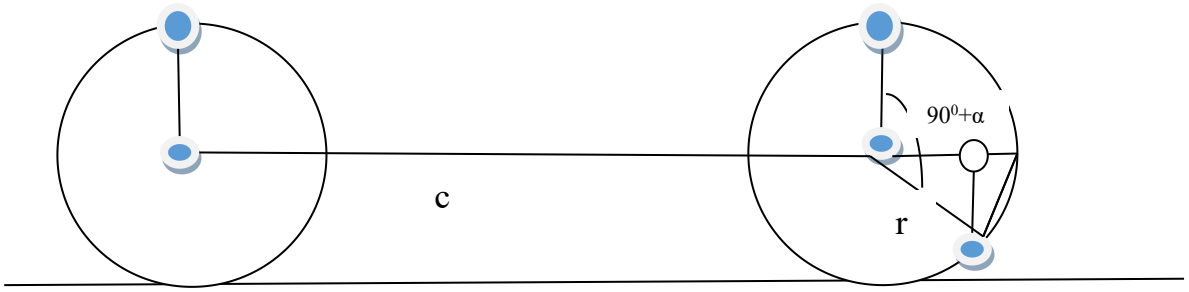
Output:

Đưa ra n số thực, số thứ i phải bằng thời gian tối thiểu mà tay đua xe đạp có thể đi được quãng đường của cuộc thi thứ i. Câu trả lời của bài toán được coi là đúng nếu sai số tuyệt đối hoặc tương đối của nó không vượt quá 10^{-6}.

Input	**Ouput**
2 1 2 1 10 5 9	3.849644710502 1.106060157705

Thuật toán: Để nữ tiến sỹ có thể về đích trong thời gian ngắn nhất, thì ở giữa khoảng cách của mỗi cuộc thi, cảm biến sẽ phải đặt ở điểm cao nhất của bánh xe hoặc ở điểm thấp nhất của bánh xe

Để tính toán câu trả lời. Ta sẽ sử dụng thuật toán tìm kiếm nhị phân. Nếu tâm của bánh xe đã đi được một quãng đường là c, thì cảm biến sẽ di chuyển đến khoảng cách c+ rsin(c/r), nếu cảm biến ở trên đầu bánh xe, hoặc cảm biến sẽ di chuyển đến khoảng cách c-rsin(c/r), nếu cảm biến nằm ở điểm chính giữa cuối cùng của bánh xe, trong đó r là bán kính của bánh xe. Để hiểu được điều này, ta có thể minh họa bằng hình ảnh sau:



Khi điểm nằm ở giữa đường tròn, mà tâm đi được một khoảng cách là c, giống như việc điểm này đã di chuyển một đường đi có độ dài một cung tròn có là c. Mà độ dài của một cung tròn sẽ bằng bán kính nhân với góc nội tiếp chắn cung tròn này. Hay ta có công thức $\sin(90^0+ \alpha)$ nhân bán kính r sẽ bằng c. Hay $\sin(90^0+ \alpha)=c/r$, hay $\cos \alpha=c/r$.

```
#include<cstdio>
#include<cmath>
#include<cstring>
#include<algorithm>
using namespace std;
const double eps=1e-9;
double ans,pi,d,r0,v;
int i,j,k,n,a,b;
double solve(double x,double w){
```

```
        double l=0,r=(x+4*pi*r0)/v,m,k;
        while ((r-l)/l>=eps){
                m=(l+r)/2;
                k=v*m+r0*cos(w-v/r0*m);
                if (k>=x) r=m;else l=m;
        }
        return r;
}
int main(){
        pi=acos(-1);
        scanf("%d%lf%lf",&n,&r0,&v);
        for (i=1;i<=n;i++){
                scanf("%d%d",&a,&b);d=(b-a)/2.0;
                ans=min(solve(d,pi/2),solve(d,-pi/2));
                printf("%.9f\n",ans*2);
        }
        return 0;
}
```

Link tải bộ test của bài toán:

https://drive.google.com/drive/folders/1IPXlpRsMBoT6aGjcs_Ps6X41fmyyUt9X?usp=sharing

C. KẾT LUẬN

Như vậy, qua phần bài tập mà tôi xây dựng, đề tài của tôi đã đạt được những kết quả sau:

- Xây dựng được một hệ thống bài tập đa dạng về hình học, trong đó có những bài tập phải sử dụng tính chất đặc biệt của hình học để giải quyết, có những bài tập hình học lồng vào trong đó là phương pháp duyệt, quay lui, tìm kiếm nhị phân,…
- Phát triển tư duy của học sinh từ những bài toán gốc

  Trong thời gian tới, tác giả sẽ cố gắng xây dựng thêm các bài tập vận dụng kiến thức hình học, để các bài tập trở nên đa dạng hơn.

  Kính mong các thầy cô góp ý để tác giả hoàn thiện đề tài của mình!

D. TÀI LIỆU THAM KHẢO VÀ BỘ TEST

1. Bộ test của bài toán được upload trên link:

https://drive.google.com/drive/folders/1IPXlpRsMBoT6aGjcs_Ps6X41fmyyUt9X?usp=sharing

2. https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/

3. https://www.geeksforgeeks.org/orientation-3-ordered-points/