L28 : Computational Geometry

1-Tut : Area Of Convex Polygon

Send Feedback

A convex polygon is a set of n vertices that are joined by n edges, such that no two edges intersect and all angles are less than 180 degrees. We can represent a polygon by listing all the vertices, starting at one vertex and following the edges until that vertex is reached again. Thus, element 0 in the array represents the first vertex. The first vertex is connected to the second vertex (element 1), the second vertex is connected to the third vertex (element 2) and so on. The last element represents the last vertex, which is connected to the first vertex.

Given the vertices of a polygon, return its exact area.

Note: Get the integer part of the area. (It can be long). So get your answer in double, and typecast it into long.

Input Format:

First line of input will contain T(number of test cases), each test case follows.
Line 1: Integer N denoting the number of points.
Next N lines will denote the N cordinates (x,y) in a anticlockwise order.

Constraints:

1 <= T <= 10^5
1 <= N <= 50
1 <= X <= Y <= 10^5
The given polygon is guranted to be convex.

Output Format:

For each test case, print the area of polygon in new line.

Sample Input 1:

```
1
4
1 5
2 2
9 2
7 5
```

Sample Output 1:

```
19
```

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define double long double

int32_t main()
{
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

    int t; cin >> t;
    while(t--)
    {
        int n; cin >> n;

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

        double area = 0.0;

        for(int i = 1 ; i < n-1 ; ++i) {
            double x1 = x[i] - x[0];
            double y1 = y[i] - y[0];
            double x2 = x[i+1] - x[0];
            double y2 = y[i+1] - y[0];

            area += (x1 * y2) - (y1 * x2);
        }

        cout << (int) abs(area)/2 << '\n';
    }
    return 0;
}

```

2-Tut : Surveyor

Send Feedback

A plot has been surveyed. Its boundary consists of segments that form a polygon. Each segment runs either North-South or East-West. Calculate the enclosed area.

The i-th character of direction and the i-th element of length describe the direction and length of the i-th segment of the polygon's boundary as the surveyor walked it. If you start at the surveyor's starting point and walk according to the sequence of directions and lengths reported by the surveyor, you will traverse the boundary of the polygon, ending up back at the starting point. This traversal described by the given segments may be either clockwise or counterclockwise.

Input Format:

First line of input contains a string s

Second line of input contains an array (space separated), with the length same as string s.

Constraints:

Direction string will have between 4 and 50 characters inclusive.
Length will have the same number of elements as the number of characters in direction.
Each element of direction will be an uppercase letter 'N', 'E', 'S', or 'W'.
Each element of length will be between 1 and 1000 inclusive.
The segments will represent a simple polygon. No two segments will touch or intersect (except that the last point of a segment is the first point of the next segment, and the last point of the final segment is the first point of the first segment).

Output Format:

For each input, print the area calculated.

Sample Test Case

NWWSE
10 3 7 10 10

Sample Ouput:

100

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define double long double

int32_t main()
{
	ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

	string s; cin >> s;
	const int n = s.size();

	vector<int> points(n);
	for(int i = 0 ; i < n ; ++i) {
		cin >> points[i];
	}

	int ans = 0;

	vector<int> x(n+1, 0), y(n+1, 0);
	for(int i = 0 ; i < n ; ++i) {
		x[i+1] = x[i];
		y[i+1] = y[i];

		if(s[i] == 'N') {
			y[i+1] = y[i] + points[i];
		}
		if(s[i] == 'S') {
```

```
                    y[i+1] = y[i] - points[i];
                }
                if(s[i] == 'E') {
                    x[i+1] = x[i] + points[i];
                }
                if(s[i] == 'W') {
                    x[i+1] = x[i] - points[i];
                }
                ans += (x[i+1] - x[i]) * y[i];
        }

        cout << abs(ans) << '\n';

        return 0;
}
```

3-Tut : Convex Hull

Send Feedback

Given a set of points in the plane, the Convex Hull of the set is the smallest convex polygon that contains all the points of it.

Find out the convex hull for the given set of points.

Input Format:

First line of input contains integer N, representing number of points.
Second line contains N space separated integers, which are the X coordinates.
Third line contains N space separated integers, which are the Y coordinates.

Constraints:

1 <= N <= 10^5
1 <= X[i] <= 10^4
1 <= Y[i] <= 10^4
Also input is given in such a way that the number of points on the hull doesn't exceed 50

Output Format:

Print the X coordinate and the Y coordinate of all the points seperated by space. Order doesn't matter.

Sample Input:

```
5
2 1 7 9 7
2 5 5 2 4
```

Sample Output:

```
1 5
2 2
9 2

7 5
```

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

struct Point
{
    int x, y;
};

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;
    return (val > 0)? 1: 2;
}


void convexHull(Point points[], int n)
{

    if (n < 3) return;

    vector<Point> hull;


    int l = 0;
    for (int i = 1; i < n; i++)
        if (points[i].x < points[l].x)
            l = i;


    int p = l, q;
    do
    {

        hull.push_back(points[p]);


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

          if (orientation(points[p], points[i], points[q]) == 2)
```

```
            q = i;
        }


        p = q;

    } while (p != l);


    for (int i = 0; i < hull.size(); i++)
        cout << hull[i].x << " " << hull[i].y << "\n";
}

int main()
{
    int n;
    cin >> n;
    Point points[n];

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

    convexHull(points, n);
    return 0;
}
```

4-Ass : Collinear Points

Send Feedback

You are given a set of 3 points in a 2-D plane. You have to find out whether they are collinear or not.

Input Format:

First line of input will contain T, representing the number of test cases.
Each test case contains three lines containing two space separated integers x and y, respectively, denoting the points in 2-D plane.

Constraints:

1 <= T <= 10^5
-10^6 <= x, y <= 10^6

Ouput Format:

For each test case output "YES" if they are collinear and "NO" if they are not.

Sample Input:

```
2
1 3
1 4
1 5
-1 0
```

```
0 1
1 0
```
Sample Output:

```
YES

NO
```

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define double long double

int32_t main()
{
	ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

	int t; cin >> t;
	while(t--)
	{
		int x1, y1; cin >> x1 >> y1;
		int x2, y2; cin >> x2 >> y2;
		int x3, y3; cin >> x3 >> y3;

		int determinant = x1*(y2-y3) + x2*(y3-y1) + x3*(y1-y2);

		if(determinant == 0) {
			cout << "YES\n";
		}
		else {
			cout << "NO\n";
		}
	}
	return 0;
}
```

5-Ass : Ninja Investigations

Send Feedback

A recent non violent protest in Uptown Plaza took a drastic turn as a bomb was blasted by some resurgent. Ninja Investigations was hired to determine the responsible party. Using advanced satellite systems and GPS tracking of cell phones, the investigators determined the coordinates of every person in the plaza. Using some advanced algorithms, the investigative team has also determined the approximated coordinates of the bomb and the vicinity in which a person had to be if he planted the bomb. You have been given a very important task - narrow down the suspect pool. To do this, you have to find out whether a given person was in the vicinity of the bomb. Also print the probability that they planted the bomb. The probability can be calculated using the following formula:

100 - (distance of point to be checked with approx coordinates of bomb / total area of the vicinity).

Note: A person in the plaza may not be carrying a phone and their coordinates might be unknown. But we can disregard such people, because the bomb was cell phone triggered, hence it would be impossible for a person not carrying the phone to plant and trigger the bomb.

Note: Get the answer of distance and area in double, then typecast it in int. Print probability in int also.

Input Format:

First line contains integer t, representing the number of test cases.
For each test case
Line 1: contains integer n, number of vertices of vicinity.
Next n lines contain coordinates of vertices of vicinity.
Next line contains approximate coordinates of the bomb.
Next line contains integer q, number of coordinates to be checked.
Next q lines contain coordinates of the people to be checked.

Output Format:

For each test case, print “Yes” if the person is present in the vicinity of the bomb and the probability that he planted the bomb in the next line, otherwise print “No”.

Sample Case 1:

```
1
3
1 10
9 1
10 7
5 6
2
8 6
8 8
```

Sample Output 1:

```
Yes
90

No
```

```
#include<bits/stdc++.h>
using namespace std;
#define INF 50000
#define MIN(x,y) (x < y ? x : y)
#define MAX(x,y) (x > y ? x : y)

struct Point
{
    double x,y;
    Point() {}
    Point(double a, double b)
    {
        x = a;
        y = b;
    }
    double cross(Point p)
    {
        return x*p.y - y*p.x;
    }
    double distance(Point p)
    {
        double dis = sqrt((pow(p.x - x, 2.0) + pow(p.y - y,2.0)) * 1.0);
        return sqrt(pow(p.x - x, 2) + pow(p.y - y, 2) * 1.0);
    }
};

double polygonArea(Point arr[], int n)
{
    double a = arr[n - 1].cross(arr[0]);

    for(int i=0;i<int(n) - 1;i++)
    {
        a += arr[i].cross(arr[i + 1]);
    }
    return a/2;
}

int probability(Point arr[], int n, Point p, Point app)
{
    int area = polygonArea(arr, n);
    int dis = app.distance(p);
    int prob = (dis*1.0 / area*1.0) * 100;
    return 100 - prob;
}

int isInside(Point polygon [], int N, Point p)
{
    int counter = 0;
    int i;
    double xinters;
    Point p1,p2;

    p1 = polygon[0];
    for(i = 1;i<=N;i++)
    {
        p2 = polygon[i % N];
        if(p.y > MIN(p1.y,p2.y))
        {
            if(p.y <= MAX(p1.y,p2.y))
            {
                if(p.x <= MAX(p1.x,p2.x))
                {
                    if(p1.y != p2.y)
                    {
                        xinters = (p.y-p1.y)*(p2.x-p1.x)/(p2.y-p1.y)+p1.x;
                        if(p1.x == p2.x || p.x <= xinters)
                            counter++;
                    }
                }
            }
        }
        p1 = p2;
    }

    if(counter % 2 == 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        int n;
        cin>>n;
        Point arr[n];
        for(int i=0;i<n;i++)
        {
            double a,b;
            cin>> a>> b;
            arr[i] = Point(a, b);
        }

        double a, b;
        cin>> a >> b;
        Point approx(a,b);

        int q;
        cin >> q;

        for(int i = 0;i < q;i++)
        {
            int a, b;
            cin>> a >>b;
            Point ch(a,b);
            if(!isInside(arr,n,ch))
            {
                cout<<"Yes"<<endl;
                cout<<probability(arr,n,ch,approx)<<endl;
            }
            else
            {
                cout<<"No"<<endl;
            }
        }
    }
    return 0;
}
```