Project 1 Report

October 26, 2020

1 Introduction

In this problem, we are given two 2-dimensional shapes A and B (A is similar to B,otherwise some path will never be matched), we should find the optimal match between points in A and B. Our solution must be monotonic, that is, if we find a matching $P = (m_1, n_1), (m_2, n_2)...(m_k, n_k)$, $m_1 < m_2 < ... < m_k$,$n_1 < n_2 < ... < n_k$must be satisfied and we should maxmize k.

To solve the problem, arrays and voting trees are supposed to used. If we just build the voting tree and then calculate the value of each node, the time complexity could be exponential. So we need to find some skills to decrease time complexity and the program can be run in polynomial time to solve the problem.

2 Algorithm Specification

2.1 Basic Idea

A simple solution is easy to find. We can enumerate every possible path in A and B, and check if every node in the path can be matched. Two paths $(m_1, m_2, ..., m_{k_1})$,$(n_1, n_2, ..., n_{k_2})$in A and B are valid if and only if:

- The length of path in A is equal to the length of path in B
- They should be monotonic.
- For all i from 1 to k_1,m_i can be matched with n_i at the same time.

The third restriction can be solved as follows: find a node in A which its interior angle is not equal to 180°, find its matching points, then we can calculate the similarity ratio, then check if each angle in A is equal to B and each length of path in A divide each length of path in B is equal to the similarity ratio.
If two paths are valid, they we add path from (m_i, n_i) to (m_{i+1}, n_{i+1}) from $i = 1$ to $k_1 - 1$. The vote of (m, n) is equal to the maximum distance if we walk from (m, n) along the path in the voting tree. In other words, it is equal to the maximum matches if we only consider the point numbered larger than m in A and numbered larger than n in B.
If we enumerate a subset of node, the order of the node can be determined.(must be increasing order) So we just need enumerate 2^{n+m} subsets and check if it is valid. Using the above method, the complexity of each check is $O(n + m)$. So the complexity of this algorithm is $O((n + m)2^{n+m})$ which is unacceptable.

2.2 Improvement

Considered that if a code is at the corner of the polygon, it must be matched. If there is no other point, we try to match the first node in B with every node in A, then other nodes in B have only one possible match with nodes in A. For example, if we have match $(3, 1)$,then the only way to match is $(3, 1), (4, 2), (5, 3)..., (n, n - 2), (1, n - 1), (2, n)$.
Now we should think how to deal with other points. Note that all the path also have a match relation from A to B. If path p in A is matched with path q in B, then the nodes on p can only

match the nodes on q. Now the problem is changed as given you two segment with some points on it, how many points coincide. It is a classical problem, solution will be shown later.
Now we have a much better solution for this problem:

a. find all nodes at the corner of the polygon

b. try to match the first node which is at the corner of B with all nodes in A which were found in step 2.

c. if all paths are matched, try to match the nodes on each segment, then we can find a longest path

d. modify the voting tree

e. find the answer

Algorithm 1 Main Function

```
1: function MAIN
2:     readpolygonA, B
3:     for i ← every point in A or B do
4:         if i is a corner point then
5:             key[i] ← true
6:         end if
7:     end for
8:     for i if keya[i] = 1 do
9:         if A and B are matched completely then
10:            find all nodes on lines which can be matched
11:        end if
12:    end for
13:    modify voting tree
14:    print the answer
15: end function
```

2.2.1 Geometry Skills

We prepare struct point and some fuctions for solving the problem. The defination is the same as geometry in math.

```
struct point
{
        double x,y;
        point(double x,double y):x(x),y(y){}//point(x,y)means a point located at (x,y)
        point(){}
        point operator - (const point &a) const
        {
                return point(x-a.x,y-a.y);//subtraction of vectors
        }
        double operator * (const point &a) const
        {
```

```
                return x*a.y-y*a.x;//vector cross product
        }
        double operator ^ (const point &a) const
        {
                return x*a.x+y*a.y;//vector dot product
        }
        double length() const
        {
                return sqrt(x*x+y*y);//vector's length
        }
};
const double pi=acos(-1.0);
// calculate pi(3.1415926....)
const double eps=1e-10;
//set precision
```

function angle is used to calculate the angle of two vectors. Calculate sin and cos, then use atan2()to get the angle.

```
double angle(const point &x,const point &y)// calculate angle of two vectors
{
        double sinx=(x*y)/x.length()/y.length();//|x*y|=|x||y|sina
        double cosx=(x^y)/x.length()/y.length();//|x^y|=|x||y|cosa
        double ans;//angle

        ans=atan2(sinx,cosx);
        return ans;
}
```

function dis is used to calculate the distance of two points.

```
double dis(const point &x,const point &y)//calculate distance from x to y
{
        return sqrt((x.x-y.x)*(x.x-y.x)+(x.y-y.y)*(x.y-y.y));
}
```

2.2.2 Find corner nodes

If the angle of two edges connected with the node is not 180°, this node is a corner node, we can use cross product to check.

```
for(int i=1;i<=m;i++)
{
        if(fabs((a[i-1]-a[i])*(a[i+1]-a[i]))>eps) keya[i]=1,posa[++cnta]=i;
}
for(int i=1;i<=n;i++)
{
```

```
        if(fabs((b[i-1]-b[i])*(b[i+1]-b[i]))>eps) keyb[i]=1,posb[++cntb]=i;
}
```

2.2.3 Try to match the corner node

We enumerate a node and check if all corner nodes can be matched, if it is possible, it is one possible match.

To check two polygons can be matched, we juast need to calculate each corresponding angles are equal and each corresponding paths are propotional.

```
for(int i=1;i<=cnta;i++)
{
        bool flag=1;//flag:A and B can matched completely
        point a1=a[posa[i]]-a[posa[i+1]];
        point b1=b[posb[1]]-b[posb[2]];
        double ratio=b1.length()/a1.length();//similarity ratio
        for(int j=1;j<=cntb;j++)
        {
                int id=i+j-1;if(id>cnta) id-=cnta;
                if(fabs(angle(a[posa[id]]-a[posa[id-1]],a[posa[id+1]]-a[posa[id]])
                -angle(b[posb[j]]-b[posb[j-1]],b[posb[j+1]]-b[posb[j]]))>eps)
                //if two angles are different
                {
                        flag=0;
                        break;
                }
                if(fabs(dis(b[posb[j]],b[posb[j-1]])
                -ratio*dis(a[posa[id]],a[posa[id-1]]))>eps)
                //if two lines are not propotional
                {
                        flag=0;
                        break;
                }
        }
        if(flag)
        {
                //...
        }
}
```

2.2.4 Find the longest path

Obviously, all the corner nodes are matched. Then for each segment, there are some nodes on it. We need to find how many of them are coincide. Each time we find the leftmost node of two

segments, if the leftmost node of the other segment has the same relative position, they should be matched. Else, the leftmost node will never matched, just ignore it.
There is a corner case. Before the first corner node, there may be some nodes. We should also deal with them.
As the code may be difficult to read, here is the pseudo-code.

Algorithm 2 Main Function

```
1: function MAIN
2:     for j ← 1 to cntb do
3:         match[pob[j]] = posa[i + j − 1]
4:     end for
5:     now ← i + 1
6:     pre ← posb[1]
7:     for j ← posb[1] + 1 to n do
8:         if match[j] then
9:             now ← match[j]
10:        else
11:            while dis(b[j], b[pre]) > ratio ∗ dis(a[now], a[match[pre]] do
12:                now move to next node
13:            end while
14:            if dis(b[j], b[pre]) = ratio ∗ dis(a[now], a[match[pre]]) then
15:                match[j] ← now
16:                now move to next node
17:            end if
18:        end if
19:    end for
20: end function
```

2.2.5 Modify the voting tree

As we get the array match, we can use it to build the voting tree.
Here are some useful properties to help us build the voting tree.

a. the matching array must be $[x, x+k_1, x+k_1+k_2, ..., x+k_1+k_2+...+k_s, l_1, l_1+l_2, ..., l_1+l_2+...+l_t](l_1+l_2+...+l_t < x, k_i > 0, l_i > 0)$.(ignore 0 in match[])

b. voting_tree[i][j] can only be modified once.

So we when we get a match[], add it from befind to front, if $match[i] > match[previous\_match\_node]$, set the vote value to 1, else the vote value plus 1.Then change voting_tree[match[i]][i] to the value.

```
int tmp=0;//vote value
pre=n+1;// the previous match node
for(int j=n;j;j--)
{
        if(!match[j]) continue;//ignore 0
        tmp++;//vote value plus 1
        if(match[j]>match[pre]) tmp=1;// set vote value to 1
```

```
        voting_tree[match[j]][j]=tmp;//change voting tree
        if(tmp!=1)
        {
                path[match[j]][j]=pii(match[pre],pre);//change path
        }
        pre=j;//set the previous match node
}
```

2.2.6 Print the answer

We find the maximum voting_tree(x,y),walk along the longest path, then we find the longest matching.

```
pii max_place;//where we can get the answer
int ans=0;//answer
for(int i=1;i<=m;i++)
{
        for(int j=1;j<=n;j++)
        {
                if(ans<voting_tree[i][j])//this place is larger than answer
                {
                        ans=voting_tree[i][j];//set answer
                        max_place=pii(i,j);//set position
                }
        }
}
while(max_place!=pii(0,0))
{
        printf("(%d,_%d)\n",max_place.first,max_place.second);
        // print the answer
        max_place=path[max_place.first][max_place.second];
        // walk along the path
}
```

3 Testing Results

3.1 Testing Code

Here are all the data generators:
gen_regular_polygon.cpp:

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstring>
```

```
#include<algorithm>
#define ll long long
using namespace std;
int main()
{
        long double pi=acos(-1.0);
        int n;long double len1,len2;
        cin>>n>>len1>>len2;
        long double angle=2.0*pi/n;
        freopen("regular_polygon_1.txt","w",stdout);
        long double x=0,y=0,anx=0;
        cout<<n<<" "<<n<<endl;
        for(int i=1;i<=n;i++)
        {
                printf("%.15Lf %.15Lf\n",x,y);
                x=x+len1*cos(anx);
                y=y+len1*sin(anx);
                anx-=angle;
        }
        x=233,y=233,anx=1;
        for(int i=1;i<=n;i++)
        {
                printf("%.15Lf %.15Lf\n",x,y);
                x=x+len2*cos(anx);
                y=y+len2*sin(anx);
                anx-=angle;
        }
}
```

gen_regular_polygon_point.cpp:

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstring>
#include<algorithm>
#define ll long long
using namespace std;
int main()
{
        long double pi=acos(-1.0);
        int n;long double len1,len2;
        cin>>n>>len1>>len2;
        long double angle=2.0*pi/n;
        freopen("regular_polygon_point3.txt","w",stdout);
```

```
        long double x=0,y=0,anx=0;
        cout<<2*n<<" "<<2*n<<endl;
        for(int i=1;i<=n;i++)
        {
                printf("%.15Lf %.15Lf\n",x,y);
                x=x+len1*cos(anx);
                y=y+len1*sin(anx);
                printf("%.15Lf %.15Lf\n",x,y);
                x=x+len1*cos(anx);
                y=y+len1*sin(anx);
                anx-=angle;
        }
        x=233,y=233,anx=1;
        for(int i=1;i<=n;i++)
        {
                printf("%.15Lf %.15Lf\n",x,y);
                x=x+len2*cos(anx);
                y=y+len2*sin(anx);
                printf("%.15Lf %.15Lf\n",x,y);
                x=x+len2*cos(anx);
                y=y+len2*sin(anx);
                anx-=angle;
        }
}
```

gen_triangle_point:

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstring>
#include<algorithm>
#include<ctime>
#define ll long long
using namespace std;
int vis[105],vis2[105];
int main()
{
        srand(time(0));
        long double pi=acos(-1.0);
        int n;long double len1,len2;
        cin>>n;
        long double angle=2.0*pi/n;
        freopen("triangle_point2.txt","w",stdout);
        int num=0,num2=0;
```

```
        for(int i=1;i<=n;i++)
        {
                vis[i]=rand()%2;
                if(vis[i]) num++;
        }
        for(int i=1;i<=n;i++)
        {
                vis2[i]=rand()%2;
                if(vis2[i]) num2++;
         }
        num+=3;num2+=3;
        cout<<num<<" "<<num2<<endl;
        cout<<"0 0\n0 1\n"<<n+1<<" "<<0<<endl;
        for(int i=n;i;i--)
        {
                if(vis[i]) cout<<i<<" "<<0<<endl;
        }
        cout<<"0 0\n0 1\n"<<n+1<<" "<<0<<endl;
        for(int i=n;i;i--)
        {
                if(vis2[i]) cout<<i<<" "<<0<<endl;
        }


}
```

gen_random.cpp:

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstring>
#include<algorithm>
#include<ctime>
#define ll long long
using namespace std;
struct point
{
        double x,y;
        point(double x,double y):x(x),y(y){}//point(x,y)means a point located at (x,y)
        point(){}
        point operator - (const point &a) const
        {
                return point(x-a.x,y-a.y);//subtraction of vectors
        }
```

```
        double operator * (const point &a) const
        {
                return x*a.y-y*a.x;//vector cross product
        }
        double operator ^ (const point &a) const
        {
                return x*a.x+y*a.y;//vector dot product
        }
        double length() const
        {
                return sqrt(x*x+y*y);//vector's length
        }
}a[105];
int num[105],cou[105];
bool vis[105];
int main()
{
        srand(time(0));
        long double pi=acos(-1.0);
        int n,m,inv;long double len1,len2;
        cin>>n>>m>>inv>>len1>>len2;
        long double angle=2.0*pi/n;
        //freopen("random10.txt","w",stdout);
        long double x=332,y=332,anx=2.33;
        cout<<n<<" "<<n<<endl;
        for(int i=1;i<=n-m;i++)
        {
                num[i]=rand()%m+1;
                while(cou[num[i]]==inv-1) num[i]=rand()%m+1;
                cou[num[i]]++;
        }
        for(int i=1;i<=m;i++)
        {
                a[i].x=x;a[i].y=y;
                printf("%.15Lf %.15Lf\n",x,y);
                x=x+len1*cos(anx);
                y=y+len1*sin(anx);
                anx-=angle;
                if(i==m) x=a[1].x,y=a[1].y;
                memset(vis,0,sizeof(vis));
                for(int j=1;j<=cou[i];j++)
                {
                        int x=rand()%(inv-1)+1;
                        while(vis[x]) x=rand()%(inv-1)+1;
                        vis[x]=1;
                }
```

```
                for(int j=1;j<=inv-1;j++)
                {
                        if(vis[j])
                        {
                                printf("%.15Lf %.15Lf\n",a[i].x+(x-a[i].x)/inv*j,a[i].y
                        }
                }
        }
        x=233,y=233,anx=1;
        memset(cou,0,sizeof(cou));
        for(int i=1;i<=n-m;i++)
        {
                num[i]=rand()%m+1;
                while(cou[num[i]]==inv-1) num[i]=rand()%m+1;
                cou[num[i]]++;
        }
        for(int i=1;i<=m;i++)
        {
                a[i].x=x;a[i].y=y;
                printf("%.15Lf %.15Lf\n",x,y);
                x=x+len2*cos(anx);
                y=y+len2*sin(anx);
                anx-=angle;
                if(i==m) x=a[1].x,y=a[1].y;
                memset(vis,0,sizeof(vis));
                for(int j=1;j<=cou[i];j++)
                {
                        int x=rand()%(inv-1)+1;
                        while(vis[x]) x=rand()%(inv-1)+1;
                        vis[x]=1;
                }
                for(int j=1;j<=inv-1;j++)
                {
                        if(vis[j])
                        {
                                printf("%.15Lf %.15Lf\n",a[i].x+(x-a[i].x)/inv*j,a[i].y
                        }
                }
        }
}
```

3.2 Sample Test

Table 1: Sample Test

Sample input	Sample output
3 4 0 4 3 0 0 0 8 4 8 1 4.25 6 8 6	(1, 2) (2, 3) (3, 4)

verdict:correct

3.3 regular polygon

Use "gen_regular_polygon" to generate two regular golygons, the expected maximum matching is n.This is used to check step a.
"regular_polygon_2.txt" is as follows:

Table 2: regular polygon

Sample input	Sample output
10 10	(1, 1)
0.000000000000000 0.000000000000000	(2, 2)
5.000000000000000 0.000000000000000	(3, 3)
9.045084971874737 -2.938926261462366	(4, 4)
10.590169943749475 -7.694208842938133	(5, 5)
9.045084971874738 -12.449491424413901	(6, 6)
5.000000000000001 -15.388417685876267	(7, 7)
0.000000000000001 -15.388417685876268	(8, 8)
-4.045084971874737 -12.449491424413903	(9, 9)
-5.59016994374947 -7.69420884293813	(10, 10)
-4.04508497187473 -2.93892626146236	
233.000000000000000 233.000000000000000	
236.782116141076978 238.890296893655276	
243.304142019622786 241.432575092353990	
250.074885425539320 239.655770733051092	
254.508152506870689 234.238562689641310	
254.910585919754355 227.250140310577863	
251.128469778677378 221.359843416922587	
244.606443900131570 218.817565218223872	
237.835700494215036 220.594369577526769	
233.402433412883666 226.011577620936551	

verdict: correct

You can use "regular_polygon_1.txt" and "regular_polygon_3.txt", they are also correct.

3.4 regular polygon with a node on each side

Use "gen_regular_polygon_point" to generate two regular golygons with a node on each side, the expected maximum matching is n.
"regular_polygon_point1.txt" is as follows:

Table 3: regular polygon with a node on each side

Sample input	Sample output
6 6 0.000000000000000 0.000000000000000 3.000000000000000 0.000000000000000 6.000000000000000 0.000000000000000 4.500000000000000 -2.598076211353316 3.000000000000000 -5.196152422706632 1.500000000000000 -2.598076211353316 233.000000000000000 233.000000000000000 234.080604611736279 234.682941969615793 235.161209223472559 236.365883939231586 236.078377416386715 234.588581909213452 236.995545609300871 232.811279879195317 234.997772804650436 232.905639939597658	(1, 1) (2, 2) (3, 3) (4, 4) (5, 5) (6, 6)

verict: correct

You can use "regular_polygon_point2.txt" and "regular_polygon_point3.txt", they are also correct.

3.5 Triangle with some nodes on one side

This test is used to test the correctness of step c. First, generate a triangle, then add some random nodes on one side."triangle_point1.txt" is as follows, it is checked to be correct.

Table 4: Triangle with some nodes on one side

Sample input	Sample output
8 8 0 0 0 1 11 0 6 0 5 0 3 0 2 0 1 0 0 0 0 1 11 0 10 0 9 0 4 0 3 0 2 0	(1, 1) (2, 2) (3, 3) (6, 7) (7, 8)

verdict:correct

You can use "triangle_point2.txt", it is also correct.

3.6 Corner case, some nodes is before the first keypoint

"corner.txt" is as follows, the expected maximum matching is 5.

Table 5: Add caption

Sample input	Sample output
5 5 0 2 0 3 0 4 5 0 0 0 0 4 0 6 0 8 10 0 0 0	(1, 1) (2, 2) (3, 3) (4, 4) (5, 5)

verdict: correct

3.7 Comprehensive Case

As the completely random testcases are really weak, we find a way to generate some common cases which can cover all types of points ant the matching is not easy to find. The generator is "gen_random.cpp".
"random1.txt" is shown as follows.

Table 6: Comprehensive Case

Sample input	Sample output
10 10	(1, 1)
332.000000000000000 332.000000000000000	(2, 2)
330.279139949001904 333.813460968667049	(3, 3)
328.558279898003808 335.626921937334098	(4, 4)
328.232000484795936 338.105538838857344	(5, 5)
327.905721071588040 340.584155740380599	(6, 7)
330.291578812784775 344.978206586555642	(8, 8)
332.548057684781852 346.054451493735620	(9, 9)
334.804536556778912 347.130696400915596	(10, 10)
337.262667195967811 346.675071817864261	
339.720797835156736 346.219447234812912	
233.000000000000000 233.000000000000000	
233.810453458802210 234.262206477211845	
234.620906917604419 235.524412954423690	
236.018483891578526 236.069186854144846	
237.416060865552623 236.613960753865996	
238.866934452534737 236.233216962586803	
240.317808039516851 235.852473171307611	
242.217779645801724 233.530812581274847	
242.304015377133940 232.033293500046971	
242.390251108466152 230.535774418819084	

verdict:correct

You can test "random2-10.txt" as well.

The program can pass all the tests, so we can conclude that the program can solve the problem correctly.

3.8 Efficiency Testing

We use regular polygons to test the efficiency of this program because it has the most ways that each corner point can be matched.
Use following codes to test the running time:

```
int main()
{
        int t=clock();
        Mymain();
        cout<<clock()-t<<endl;
}
```

Table 7: Add caption

n,m	time
10	0.003
20	0.006
30	0.008
40	0.019
50	0.038
60	0.046
70	0.057
80	0.066
90	0.076
100	0.082

As it is running too fast, it is meaningless to draw the time diagram.

4 Analysis and Comments

4.1 Analysis

4.1.1 Space complexity analysis

There are some 1-dimensional arrays($a, b, keya, keyb, posa, posb, match$), and the voting tree has nm nodes and nm paths at most, and there is no recursive function, so the space complexity is $O(nm + n + m) = O(nm)$.

4.1.2 Time complexity analysis

First, we use $O(n + m)$ to find the keypoints, then we enumerate a keypoint in A, use $O(n)$ time to check if this match is valid. If it is valid, we use $O(n + m)$ to enumerate each point on the polygon in clockwise. As there are $O(m)$ keypoints in A at most, this algorithm's time complexity is $O(n + m + m(n + m)) = O(m(n + m))$.

4.2 Comments

As the program's complexity is really fast, the constraints given in the problem can't show the running time changes clearly, in fact, the running time t can be evaluated as $t = k * n^2$(k is a constant).
Comparing with the complexity of basic algorithm($O((n+m)2^{n+m})$), this algorithm is much more efficient. Essentially, this algorithm is more efficient because the basic algorithm considers too many useless conditions , and for one condition, it may calculate many times.

Note that if the number of corner points is small, this algorithm runs much more quickly because the number that we try to find the optimal matching and find the maximum matching and modify the voting tree is no more than the number of corner points. , so when I generate testcases, I try to promise that the number of corner nodes is large enough.

If the polygons are not similar and we just want to let some of paths be matched, the voting tree may be much more complex and it has an exponential complexity, so I think using dynamic programming is much more compatible. Define f(x,y,a,b) denotes the previous path in A is (x,y) and the previous path in B is (a,b), we can transfer a state with time complexity $O(n^2)$, so the total time complexity is $O(n^6)$ with is much more suitable.

5 Appendix

Source code is given as follows:

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstring>
#include<algorithm>
#include<cstdlib>
#include<ctime>
#define MAXN 105
#define pii pair<int,int>
using namespace std;
//It is recommended to read the code and the report at the same time
struct point
{
        double x,y;
        point(double x,double y):x(x),y(y){}//point(x,y)means a point located at (x,y)
        point(){}
        point operator - (const point &a) const
        {
                return point(x-a.x,y-a.y);//subtraction of vectors
        }
```

```
        double operator * (const point &a) const
        {
                return x*a.y-y*a.x;//vector cross product
        }
        double operator ^ (const point &a) const
        {
                return x*a.x+y*a.y;//vector dot product
        }
        double length() const
        {
                return sqrt(x*x+y*y);//vector's length
        }
}a[MAXN*2],b[MAXN*2];//two initial polygons
bool keya[MAXN],keyb[MAXN];//1 if the point is a keypoint of the polygon
int posa[MAXN],posb[MAXN],cnta,cntb,match[MAXN];
/*
posa,posb:the position of keypoints in a and b
cnta,cntb:the number of keypoints in a and b
match:match[i] means the i-th point in b matches the match[i]-th point in a,0 if the po
*/
int voting_tree[MAXN][MAXN];
//voting_tree[x][y] means the maximum matches if we only consider the point numbered la
pii path[MAXN][MAXN];
// path[x][y] means the son of node (x,y) in voting trees which has the maximum value.
const double pi=acos(-1.0);
// calculate pi(3.1415926....)
const double eps=1e-6;
//set precision
double angle(const point &x,const point &y)// calculate angle of two vectors
{
        double sinx=(x*y)/x.length()/y.length();//|x*y|=|x||y|sina
        double cosx=(x^y)/x.length()/y.length();//|x^y|=|x||y|cosa
        double ans;//angle

        ans=atan2(sinx,cosx);//use arctan to calculate the angle,atan2() can return an
        return ans;
}


double dis(const point &x,const point &y)//calculate distance from x to y
{
        return sqrt((x.x-y.x)*(x.x-y.x)+(x.y-y.y)*(x.y-y.y));
}
int main()
{
        int n,m;
```

```
        scanf("%d%d",&m,&n);//read n and m
        for(int i=1;i<=m;i++)
        {
                scanf("%lf%lf",&a[i].x,&a[i].y);// read a
        }
        for(int i=1;i<=n;i++)
        {
                scanf("%lf%lf",&b[i].x,&b[i].y);// read b
        }
        a[0]=a[m];b[0]=b[n];// decrease some corner cases(a[1],a[m],b[1],b[n])
        a[m+1]=a[1];b[n+1]=b[1];
        double area=0;
        for(int i=1;i<=m;i++)
        {
                area+=a[i]*a[i+1];//if the sum of cross product>0, the input is read in
        }
        if(area>0)
        {
                puts("please input in clockwise order!");
                return 0;
        }
        area=0;
        for(int i=1;i<=n;i++)
        {
                area+=b[i]*b[i+1];//if the sum of cross product>0, the input is read in
        }
        if(area>0)
        {
                puts("please input in clockwise order!");
                return 0;
        }

        for(int i=1;i<=m;i++)
        {
                if(fabs((a[i-1]-a[i])*(a[i+1]-a[i]))>eps) keya[i]=1,posa[++cnta]=i;
                //if cross product equal to 0, the three points are in the same line,el
        }
        for(int i=1;i<=n;i++)
        {
                if(fabs((b[i-1]-b[i])*(b[i+1]-b[i]))>eps) keyb[i]=1,posb[++cntb]=i;
                //if cross product equal to 0, the three points are in the same line,el
        }
        posa[0]=posa[cnta];posb[0]=posb[cntb];// decrease some corner cases(posa[1],pos
        posa[cnta+1]=posa[1];posb[cntb+1]=posb[1];
        if(cnta!=cntb)// the number of keypoint is not equal
        {
```

```
            puts("invalid input!");
            return 0;// finish the program
        }
        for(int i=1;i<=cnta;i++)
        {
            bool flag=1;//flag:A and B can matched completely
            point a1=a[posa[i]]-a[posa[i+1]];
            point b1=b[posb[1]]-b[posb[2]];//two corresponding lines
            double ratio=b1.length()/a1.length();//similarity ratio
            //cout<<b1.length()<<" "<<a1.length()<<endl;
            //cout<<ratio<<endl;
            for(int j=1;j<=cntb;j++)
            {
                int id=i+j-1;if(id>cnta) id-=cnta;//corresponding nodes in A
                if(fabs(angle(a[posa[id]]-a[posa[id-1]],a[posa[id+1]]-a[posa[id
                -angle(b[posb[j]]-b[posb[j-1]],b[posb[j+1]]-b[posb[j]]))>eps)
                //if two angles are different
                {
                    flag=0;//illegal
                    break;
                }
                if(fabs(dis(b[posb[j]],b[posb[j-1]])-ratio*dis(a[posa[id]],a[po
                //if two lines are not propotional
                {
                    flag=0;//illegal
                    break;
                }
            }
            if(flag)//if it is a completely matching
            {
                memset(match,0,sizeof(match));//clear match
                //cout<<"ok";
                for(int j=1;j<=cntb;j++)
                {
                    int id=i+j-1;//corresponding nodes in A
                    match[posb[j]]=posa[id];//posb[j] matches posa[id]
                }

                int now=posa[i]+1,pre=posb[1];
                //now:now places of A pre:previous corner node
                for(int j=posb[1]+1;j<=n;j++)
                {
                    if(match[j])//corner node
                    {
                        while(now!=posa[i])//not at the last node
                        {
```

```
                        if(match[j]!=now)//now is not at the ne
                        {
                                now++;//move now
                                if(now>m) now-=m;// because it
                        }
                        else break;//finished
                }
                pre=j;//change previous corner node
        }
        else
        {
                while(now!=posa[i])// not at the last node
                {
                        //cout<<dis(b[j],b[pre])<<" "<< dis(a[n
                        if(fabs(dis(b[j],b[pre])-dis(a[now],a[m
                        //at the same point
                        {
                                match[j]=now;// add a match
                                now++;if(now>m) now-=m;
                                // because it is a circle,we sh
                        }
                        else if(dis(b[j],b[pre])-dis(a[now],a[m
                        // a is in front of b
                        {
                                now++;
                                if(now>m) now-=m;
                                // because it is a circle,we sh
                        }
                        else break;// a is beind b, there is no
                }
        }
}
for(int j=1;j<posb[1];j++)//corner case: some point in front of
{
        while(now!=posa[i])// not at the last node
        {
                if(fabs(dis(b[j],b[pre])-dis(a[now],a[match[pre
                //at the same point
                {
                        match[j]=now;//add a match
                        now++;if(now>m) now-=m;
                        // because it is a circle,we should cha
                }
                else if(dis(b[j],b[pre])-dis(a[now],a[match[pre
                {
                        now++;
```

```
                                        if(now>m) now-=m;
                                        // because it is a circle ,we should cha
                                }
                                else break;// a is beind b, there is no need to
                        }
                }
                int tmp=0;//vote value
                pre=n+1;// the previous match node
                for(int j=n;j;j--)
                {
                        if(!match[j]) continue;//ignore 0
                        tmp++;//vote value plus 1
                        if(match[j]>match[pre]) tmp=1;// set vote value to 1
                        voting_tree[match[j]][j]=tmp;//change voting tree
                        if(tmp!=1)
                        {
                                path[match[j]][j]=pii(match[pre],pre);//change
                        }
                        pre=j;//set the previous match node
                }
        }
    }
    pii max_place;//where we can get the answer
    int ans=0;//answer
    for(int i=1;i<=m;i++)
    {
        for(int j=1;j<=n;j++)
        {
            if(ans<voting_tree[i][j])//this place is larger than answer
            {
                ans=voting_tree[i][j];//set answer
                max_place=pii(i,j);//set position
            }
        }
    }
    while(max_place!=pii(0,0))
    {
        printf("(%d, %d)\n",max_place.first,max_place.second);// print the answ
        max_place=path[max_place.first][max_place.second];// walk along the pat
    }
    if(ans==0)//means two polygons are not similar
    {
        puts("Two polygons are no similar!");
    }
    return 0;
}
```

6 Declaration

I hereby declare that all the work done in this project titled "Project 1 Report" is of my independent effort.