Project4-The Best Polygon

Chapter 1 : Introduction

1.1 Problem Description

Given a series of points and their 2-dimensional positions, our purpose would be finding the choices of specific number of points inside the polygon to make the area as large as possible.

1.2 Report Purpose

In our project, we thought of two kind of algorithms. We would illustrate the background of the data structures and algorithm is this chapter. The report also contains the details of our implementation of the project and our application of the alogrithms.

To claim the correctness of our code, we not only test on the online judge but also create some specific examples and get the sample output. You can see that our program got accepted on the online judge in Chapter 3 testing results. The analysis such as time complexity and the space complexity are also contained in the report.

1.3 Background

Here is one of the little knowledge point for this project, which is referenced from wikipedia. This is how we calculate the item's area of the n-polygon.

Area and centroid [edit]

Simple polygons [edit]

For a non-self-intersecting (simple) polygon with vertices $\{(x_i, y_i)\}_{i=0}^{n-1}$ the signed area and the Cartesian coordinates of the centroid are given by:[3]

$$A = \frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i),\ where (x_n, y_n) = (x_0, y_0),$$

$$16A^2 = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\begin{vmatrix} Q_{i,j} & Q_{i,j+1} \\ Q_{i+1,j} & Q_{i+1,j+1} \end{vmatrix},$$

where $Q_{i,j}$ is the squared distance between (x_i, y_i) and (x_j, y_j); [4] and

Chapter 2 : Data Structure / Algorithm Specification

2.1 Backtracking Version

2.1.1 Background

In order to get the solution of the problem, backtracking needs to take a number of steps, each step is the same, each step is done on the basis of the previous step, and it needs to record the previous trajectory until the end situation, no more likely to be correct or possibly a mistake.

Compare with DP: here is the recursion of DP from up to bottom, and the two all involve recursion, but there is a difference, the recursion of DP is a small recursion, that is, a subproblem. But backtracking recursion is the choice of every step, which can be considered as a parallel and a step in a complete search. Moreover, DP is from high to low recursion, and backtracking is from beginning to end, it's like low to high recursion.

2.1.2 Data Structure and Algorithm

- Data Structure

We use linkedList to store the polygon

```
public LinkedList<Vertex> vertexList = new LinkedList<>();
static class Vertex
{
  public int index;
  public double xVal;
  public double yVal;
  public boolean isUsed = false;
```

```
}
```

- Algorithm

  Step 1 : Each point could be the start point of the chosen polygon

  Step 2 : Select the next point from the point set until n points had been chosen

  Step 3 : If the current area is bigger than the stored area, update the points and update the area

  Step 4 : Track back, delete the last chosen point, and get back to step 2

  Step 5 : Finally output the chosen points' result

2.1.3 Pseudocode

```
choose each point as the start point
  choose the next n - 1 point in the point set
    if the area is bigger than the max area
      update the stored point
      update the max area
    delete the last point from the list and put in the next point
  after all the possible results are traversed
output the points chosen to get the max area
```

2.2 Dynamic Programming Version

2.2.1 Background

Dynamic programming conditions :

1. Optimization principle: if the solution of the subproblem contained in the optimal solution of the problem is also optimal, it is called the optimal substructure, that is, to satisfy the optimization principle.
2. No aftereffect: that is, once a certain stage of state is determined, it will not be affected by the subsequent decision of this state. That is to say, the subsequent process of a certain state will not affect the previous state, but only the current state.
3. There is a problem of overlapping subproblems: that is, the sub problem is not independent, and a sub problem may be used many times in the decision of the next stage (the nature is not the necessary condition for the application of

dynamic programming, but if it does not, the dynamic programming algorithm has no advantages compared with other algorithms).

2.2.2 Data Structure and Algorithm

- Data Structure

We store the points as the node structure.

```
struct node
{
    double x,y;
    int id;
    friend double operator * (node a,node b)
    {
        return a.x*b.y-a.y*b.x;
    }
    friend node operator + (node a,node b)
    {
        return node{a.x+b.x,a.y+b.y};
    }
    friend node operator - (node a,node b)
    {
        return node{a.x-b.x,a.y-b.y};
    }
}P[305];

double S[305][305];
double f[305][15];
int fz[305][15];
int cnt;int choose[305];
```

- Algorithm

The first version of dp algorithm, we used this state equation :

  - f(i, j, n) = Max(f(i, j - 1, n), Max(Area(x, y, k) + f(i, j - 1, n - 1)))

After the optimization, we used this one :

  - F[i][j] = min(f[k][j - 1] + the cut-off area by the lines between point i and point j)

2.2.3 Pseudocode

```
choose each point as the start point
  S[i][j] store the cut-off area by the line between point i and point j
  use the linked list to represent the polygon
```

```
the area would be the total area minus the cut-off area
the next point best strategy would be best results from the enumeration of the current point
```

Chapter 3 : Testing Results

3.1 BackTracking Version

When we using the backtracking version based on Java, we got 3 test points to be out of runtime.

On small data set, our output was good.

```
10 6
133.0 1.0
544.0 71.0
558.0 206.0
536.0 338.0
463.0 436.0
330.0 503.0
188.0 499.0
305.0 2.0
55.0 410.0
2.0 140.0
9 8 5 3 1 0
Process finished with exit code 0
```

3.2 Dynamic Programming Version

Our dynamic programming version based on C++ got accepted on the data set.

查看提交

评测结果

时间	结果	得分	题目	语言	用时(ms)	内存(kB)	用户
5月06日 16:59	答案正确	35	1023	C++ (g++ 4.7.2)	108	1132	[redacted]

测试点

测试点	结果	用时(ms)	内存(kB)	得分/满分
0	答案正确	2	364	18/18
1	答案正确	2	484	5/5
2	答案正确	2	612	3/3
3	答案正确	2	356	1/1
4	答案正确	64	1132	2/2
5	答案正确	108	1124	6/6

查看代码

Chapter 4 : Analysis and Comments

3.1 BackTracking Version

When we using the backtracking version based on Java, we got 3 test points to be out of runtime.

In this version, our

- Space complexity would be one linked list, that is $O(N)$, and using backTracking,
- Time complexity would be $O(2^N)$

It is not acceptable when N grows too big. So we change our strategy and using dynamic programming to implement this project and finally got accepted.

3.2 Dynamic Programming Version

- At the first time we used 3-dimension dynamic programming :
  - $f(i, j, n) = Max(f(i, j - 1, n), Max(Area(x, y, k) + f(i , j - 1, n - 1)))$
- After the optimization
  - $F[i][j] = min(f[k][j - 1]$ + the area cut off by the line between point i and point k)

- [ ] Time complexity : $O(n*N*3)$

- We store the area item in the two dimension array S[i][j].

- [ ] Space complexity : $O(n^2)$

Appendix : Source Code

```
//dynamic version
#include<iostream>
#include<cstdlib>
#include<cstdio>
#include<cmath>
#define alpah 1e-9
using namespace std;
const double inf=999999999;
double ans=inf;
struct node // represent the 2-D point
{
```

```
    double x,y; // x position y position
    int id; // represent the index of the point, used for sort the points
    friend double operator * (node a,node b)
    {
        return a.x*b.y-a.y*b.x; // the vertor operation
    }
    friend node operator + (node a,node b)
    {
        return node{a.x+b.x,a.y+b.y}; // the vector operation
    }
    friend node operator - (node a,node b)
    {
        return node{a.x-b.x,a.y-b.y}; // the vector operation
    }
}P[305];
int N,n;
double S[305][305]; // store the area of the cut-off area between point i and point j
double f[305][15];  // store the result
int fz[305][15];
int cnt;int choose[305];
void qsort(int l,int r) // implement the quick sort
{
    int x,y;node temp,mid;
    x=l;y=r;mid=(P[(l+r)/2]-P[1]);
    while(x<=y)
    {
        while((P[x]-P[1])*mid>0)x++;
        while((P[y]-P[1])*mid<0)y--;
        if(x<=y)
        {
            temp=P[x];P[x]=P[y];P[y]=temp;
            x++;y--;
        }
    }
    if(l<y)qsort(l,y);
    if(x<r)qsort(x,r);
}
int main()
{
    int a,b,c,d,e;
    scanf("%d%d",&N,&n);  //input N and n
    for(a=1;a<=N;a++)
    {
        scanf("%lf%lf",&P[a].x,&P[a].y);  //input the positions of the points
        P[a].id=a-1;  //input the index of the point
    }
    qsort(2,N); // sort the points according to their x position and y position
    for(a=1;a<=N;a++)
    {
        S[a][a]=0;S[a][a+1]=0;  // initialize the item area
        for(b=a+2;b<=N;b++)S[a][b]=S[a][b-1]+((P[b-1]-P[a])*(P[b]-P[a]))*0.5; // calculate the area
    }
    for(a=1;a<=N;a++)for(b=a-1;b>=1;b--)S[a][b]=S[1][N]-S[b][a];
    for(a=1;a<=N;a++)
    {
```

```
        for(b=0;b<=N;b++)for(c=0;c<=n;c++)f[b][c]=inf;
        f[a][1]=0;
        for(b=2;b<n;b++)
        {
            e=N-(n-b);
            for(c=a+b-1;c<=e;c++)
            for(d=a+b-2;d<c;d++)
            {
                if(f[c][b]>f[d][b-1]+S[d][c])
                {
                    f[c][b]=f[d][b-1]+S[d][c];
                    fz[c][b]=d;
                }
            }
        }
        int FROM=-1;
        for(c=a+n-1;c<=N;c++)
        {
            for(d=a+n-2;d<c;d++)
            {
                if(f[c][n]>f[d][n-1]+S[d][c]+S[c][a])
                {
                    f[c][n]=f[d][n-1]+S[d][c]+S[c][a];
                    fz[c][n]=d;
                }
            }
            if(f[c][n]<ans)
            {
                ans=f[c][n];
                FROM=c;
            }
        }
        if(FROM!=-1)
        {
            cnt=0;
            while(FROM!=a)
            {
                cnt++;
                choose[cnt]=FROM;
                FROM=fz[FROM][n-cnt+1];
            }
            cnt++;
            choose[cnt]=a;
        }
    }
    for(a=1;a<cnt;a++)cout<<P[choose[a]].id<<" "; // output the result
    cout<<choose[cnt]-1;
    return 0;
}

```

```
//backtracking Version
import java.util.LinkedList;
import java.util.Scanner;
import static java.awt.geom.Line2D.linesIntersect;

public class Main {
```

```java
    public static int verticeNum;   //total number of vertices of the convex 6 <= N <= 300
    public static int usedVerticeNum;   //approximating convex n-gon 6 <= n <= min(10, N)
    public static Polygon inputPolygon;

    static class Vertex
    {
        public int index;
        public double xVal;
        public double yVal;
        public boolean isUsed = false;  // true : the points that are choosed

        public int GetIndex() // get the index of the point
        {
            return index;
        }
        public double GetX()  // get its x position
        {
            return xVal;
        }
        public double GetY()  // get its y position
        {
            return yVal;
        }
        public void SetVertex(int index, double xVal, double yVal)  // set the position of the vertex
        {
            this.index = index;
            this.xVal = xVal;
            this.yVal = yVal;
        }
        public Vertex(int index, double xVal, double yVal)
        {
            this.index = index;
            this.xVal = xVal;
            this.yVal = yVal;
        }
    }

    static class Polygon {
        public LinkedList<Vertex> vertexList = new LinkedList<>();  // the list of all the input points
        public LinkedList<Integer> resultList = new LinkedList<>(); // the list to store the index of the points that area choosed
        //public LinkedList<Vertex> currentList = new LinkedList<>();
        public static double maxArea; // the area

        public double GetSingleArea(double x0, double y0, double x1, double y1, double x2, double y2)
        { // calculate the area by the vectore operation
            double singleArea = (x0 * y1 + y0 * x2 + x1 * y2 - x2 * y1 - x0 * y2 - x1 * y0) / 2.0;
            return singleArea;
        }
        //resultList.get(0) as the first point
        //TODO : make the points in the correct order
        public void CalculateMaxArea(int vertexVal, int chooseIndex)
```

```java
    {
        if(vertexVal <= 0)
        {
            //TODO : previous area plus the current 3-polygon
            double currentArea = GetTotalArea(vertexList);
            if(maxArea < currentArea)
            {
              // if current area is bigger than the sotred max area
              // update
                resultList = new LinkedList<>();  // update the resultlist
                for(int i = 0; i < vertexList.size(); i++)  // update the resultlist by traverse the total points list
                {
                    if(!vertexList.get(i).isUsed) //only the node that marked true would be put in the result list
                        continue;
                    resultList.add(vertexList.get(i).index);
                }
                // update the max area
                maxArea = currentArea;
            }
        }
        else
        {
            for(int i = chooseIndex + 1; i <= vertexList.size() - vertexVal; i++)
            {
              // backtracking method
                vertexList.get(i).isUsed = true;
                vertexVal--;
                CalculateMaxArea(vertexVal, i);
                vertexVal++;
                vertexList.get(i).isUsed = false;
            }
        }
        return;
    }

    public double MaxArea(int vertexVal)
    {
        for(int i = 0; i <= vertexList.size() - vertexVal; i++)
        {
          // each point could be the start point
            vertexList.get(i).isUsed = true;
            vertexVal--;
          // the entry of the backtracking method
            CalculateMaxArea(vertexVal, i);
            vertexVal++;
            vertexList.get(i).isUsed = false;
        }

        return maxArea;
    }

    public double GetTotalArea(LinkedList<Vertex> vList)
    {
        int startIndex = 0; // the first point
```

```
            while(!vList.get(startIndex).isUsed)
            {
                startIndex++;
            }
            double x0 = vList.get(startIndex).xVal; // get the first chosen point int the list
            double y0 = vList.get(startIndex).yVal;
            startIndex++;
            while(!vList.get(startIndex).isUsed)
            {
                startIndex++;
            }
            double x1 = vList.get(startIndex).xVal; // get the second chosen point int the list
            double y1 = vList.get(startIndex).yVal;
            startIndex++;
            while(!vList.get(startIndex).isUsed)
            {
                startIndex++;
            }
            double x2 = vList.get(startIndex).xVal; // get the third chosen point int the list
            double y2 = vList.get(startIndex).yVal;
            startIndex++;
            double totalArea = GetSingleArea(x0, y0, x1, y1, x2, y2);
            for(int i = startIndex; i < vList.size(); i++)
            {
                if(!vList.get(i).isUsed)
                {
                    continue;
                }
                x1 = x2;
                y1 = y2;
                x2 = vList.get(i).xVal;
                y2 = vList.get(i).yVal;
                totalArea += GetSingleArea(x0, y0, x1, y1, x2, y2); // calculate the n-polygon area by add up all the little 3-polygon area

            }
            return Math.abs(totalArea);
        }
    }

    public static boolean checkIntersection(double x0, double y0, double x1, double y1, double x2, double y2, double x3, double y3)
    {
        return linesIntersect(x0, y0, x1, y1, x2, y2, x3, y3);  // true : the two lined intersect each other
    }
    public static void inputValue()
    {
        inputPolygon = new Polygon();
        Scanner scanner = new Scanner(System.in); // input the n and N
        if(scanner.hasNextInt())
        {
            verticeNum = scanner.nextInt();
```

```java
            usedVerticeNum = scanner.nextInt();
        }

        usedVerticeNum = Math.min(usedVerticeNum,Math.min(10, verticeNum));
        for(int i = 0; i < verticeNum; i++) // input the position of the points
        {
            double inputX = scanner.nextDouble();
            double inputY = scanner.nextDouble();
            Vertex vertex = new Vertex(i, inputX, inputY);
            //TODO : fix the order
            if(i > 1 && inputPolygon.GetSingleArea(inputPolygon.vertexList.get(0).xVal, inputPolygon.vertexList.get(0).yVal, inputPolygon.vertexList.getLast().xVal, inputPolygon.vertexList.getLast().yVal,
                    inputX, inputY) < 0)
            {
                int j;
                for(j = 0; j < verticeNum; j++) // if the lines got intersected, the points should be sorted
                {
                    if(checkIntersection(inputPolygon.vertexList.get(j).xVal, inputPolygon.vertexList.get(j).yVal,
                            inputPolygon.vertexList.get(j + 1).xVal, inputPolygon.vertexList.get(j + 1).yVal,
                            inputPolygon.vertexList.getLast().xVal, inputPolygon.vertexList.getLast().yVal,
                            inputX, inputY))
                        break;
                }
                inputPolygon.vertexList.add(j + 1, vertex); // add in the vertex
            }
            else
            {
                inputPolygon.vertexList.add(vertex);
            }
        }
    }
    public static void main(String[] args) {
        // write your code here
        inputValue();

        // the input didn't obey the rules
        if(verticeNum < 6 || verticeNum > 300 || usedVerticeNum < 6 || usedVerticeNum > Math.min(10, verticeNum))
        {
            return;
        }

        double maxArea = inputPolygon.MaxArea(usedVerticeNum);
        //System.out.println("max Area : " + maxArea);
        for(int i = inputPolygon.resultList.size() - 1; i >= 0 ; i--)
        { //output the points chosen to get the max area of n-polygon
            System.out.print(inputPolygon.resultList.get(i));
            if(i > 0)
                System.out.print(" ");
        }
    }
```

```
}

```

References

[1] https://en.wikipedia.org/wiki/Polygon