

**数学与信息学院学生实验报告**

**实验课程名称：** 算法分析与设计基础 **教师： \_\_**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验项目名称** | **实验二 动态规划算法设计与应用** | | | **实验成绩** |  |
| **学生姓名** |  | **学 号** | **158** | **年级专业班级** |  |
| **小组成员** | **无** | | | **实验日期** | **2019年4 月** |

# 1. 实验目的和要求

## 1.1 实验目的

1.加深对动态规划算法的基本原理的理解，掌握用动态规划方法求解最优化问题的方法步骤及应用；

2.用动态规划设计整数序列的最长递增子序列问题的算法，分析其复杂性，并实现；

3.用动态规划设计求凸多边形的三角剖分问题的算法，分析其复杂性，并实现。

4.选做题：用动态规划设计求解0/1背包问题的算法，分析其复杂性，并实现。

## 1.2 实验软硬件环境

① 操作系统

WIN10操作系统

② 编译环境

Java语言，Eclipse编译器编译

## 1.3 实验要求

1.3.1 基本原理

动态规划是一种非常重要的程序设计方法，常用于求解最优化问题。最优化问题：给定若干个约束条件和一个目标函数，在某指定集合中求满足所有约束条件的且使得目标函数值达最大或最小的元素和相应的目标函数值，即：问题的最优值和最优解。

适用动态规划求解的问题的基本要素：

(1)满足最优性原理：即

一个最优化问题的最优解包含了其子问题的最优解。

(2)

无后向性：即某阶段状态一旦确定，就不受这个状态以后决策的影响。也即，某状态以后的过程不会影响以前的状态，只与当前状态有关，这种特性也被称为无后效性。

(2)具有重叠的子问题：即问题被分解成的子问题存在互相重叠。动态规划方法对于这些重叠的子问题只求解一次，以提高算法的效率。

1.3.2 该类算法设计与实现的要点

动态规划算法求解最优化问题的步骤：

(1) 找出问题的最优子结构。分析问题的最优解（最优值）的结构特征。

(2) 递归地定义最优值。 根据最优子结构，确定最优值所满足的递归公式。

(3) 计算最优值。根据最优值的递归公式，采用自底向上的迭代或自顶向下的递归，计算最优值。

(4) 构造最优解。在求解最优值的过程中要记录下得到最优值的相应最优解的信息，并根据该信息构造最优解。

注意：在计算最优值时应保存相应的信息：

(a) 已经求出的子问题的最优值（避免重复计算）。

(b) 最优解的有关信息。

动态规划算法求解其它问题的步骤：

(1) 根据最优化原理分析问题的解的结构。

(2) 递归地定义问题的解。

(3) 计算问题的解。 根据解的递归公式，自底向上或自顶向下地计算解，计算过程中注意保存已经求出的子问题的解。

其中，自底向上方法通过迭代来实现，适用于所有的子问题都需要解的情况，实现时要注意根据递归公式正确确定子问题的求解顺序。自顶向下方法通过递归来实现，适用于不必解所有的子问题的情况，实现时要注意标记子问题是否计算过，同一个子问题只在第一次递归调用时计算并存储结果。

## 1.4 实验要求

(一) 最长递增子序列问题

1.问题描述

求一个由n个整数组成的整数序列的最长递增子序列。一个整数序列的递增子序列可以是序列中非连续的数按照原序列顺序排列而成的。 最长递增子序列是其递增子序列中长度最长的。

2. 具体要求（若在ACM平台上提交程序，必须按此要求）――平台上1700题

输入：输入的第一行是一个正整数n，表示测试例个数。接下来几行是n个测试例的数据，每个测试例的数据由两行组成，其中第一行为一个正整数k (k<=500)，表示整数序列的长度，第二行给出整数序列，整数之间用一个空格隔开。（设给出的每个整数序列的最长递增子序列都是唯一的。）

输出：对于每个测试例输出两行，第一行为最长递增子序列的长度，第二行为最长递增子序列，整数之间用一个空格隔开。两个测试例的输出数据之间用一个空行隔开，最后一个测试例后无空行。

3. 测试数据

输入：3

5

3 1 4 2 3

6

1 3 9 5 2 6

20

1 2 7 13 3 5 10 24 12 4 9 16 53 6 83 8 23 11 31 47

输出：3

1 2 3

4

1 3 5 6

10

1 2 3 5 10 12 16 23 31 47

4. 设计与实现的提示

(1) 寻找最优子结构、写出递归方程是问题的关键。

(2) 以Ai为末元素的最长递增子序列(记为S(i))，等于以使S(j), (j=1～i), 最大的那个Aj为末元素的递增子序列最末再加上Ai；如果这样的元素不存在，那么Ai自身构成一个长度为1的以Ai为末元素的递增子序列。

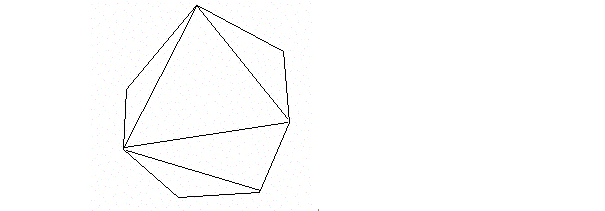
(3) 最优解的信息在此是以Ai为末元素的最长递增子序列的前驱元素，应当记录下来。

5. 扩展内容

本题可以采用多种方法求解，可以尝试用不同思路求解。

(二) 凸多边形的三角剖分

1. 问题描述

设P是一个有n个顶点的凸多边形，P中的弦是P中连接两个非相邻顶点的线段。用P中的(n-3)条弦将P剖分成(n-2)个三角形（如下图所示）。使得(n-3)条弦的长度之和最小的三角形剖分称为最优三角剖分。

2. 具体要求（若在ACM平台上提交程序，必须按此要求）――平台上1701题

输入：输入的第一行是一个正整数m，表示测试例个数，接下来几行是m个测试例的数据，每个测试例的数据由两行组成，第一行含一个正整数n (n<=500)，表示凸多边形的顶点个数；第二行含2n个实数x1 , y1 , x2 , y2 , …xn , yn ，按顺时针方向依次给出n个顶点的坐标值(xi, yi) i=1, 2, …, n，整数之间用一个空格隔开。

输出：对于每个测试例输出一行，含一个实数（精确到小数点后三位），表示最优三角剖分的n-3条弦的长度之和。两个测试例的输出数据之间用一个空行隔开，最后一个测试例后无空行。

3. 测试数据

输入：

2

6

1 2 2 1.5 2 0.5 1 0 0 0.5 0 1.5

9

723 1220 463 1074 370 842 317 534 524 192 992 87 1378 355 1683 855 1301 1131

输出：

5.606

4928.722

4. 设计与实现的提示

(1) 凸(n+1)边形的最优三角剖分是该凸多边形的所有三角剖分中弦长最短的那个剖分。

(2) 本题与课本7.3节中的矩阵链相乘问题非常相似。寻找最优子结构、写出递归方程是问题的关键，注意递归方程的参数的设置。

(3) 将凸(n+1)边形三角剖分时，注意弦长不要重复计算。

5. 扩展内容

本题只要求计算出最优值。如果要求最优解，在求最优值过程中，要记录下改最优值对应的三角剖分的信息。

(三) 选做题――0/1背包问题

1. 问题描述

设有一个容量为C的背包，n个物品的集合U={u1, u2, …, un}，物品uj的体积和价值分别为sj和vj，C, sj, vj都是正整数。在U中选择物品装入背包，使得装入背包的物品总价值最大。设每种物品或完全装入或完全不装入背包。

2. 具体要求

输入：输入的第一行是一个正整数m，表示测试例个数，接下来几行是m个测试例的数据。每个测试例的数据由三行组成，第一行含两个正整数n和C，其中， n (n<=100)表示给定的是n个物品的集合U={u1, u2, …, un}，C(C<=5000)表示背包的容量；第二行含n个整数s1 , s2 , …sn，表示n个物品的体积；第三行含n个整数v1 , v2 , …vn，表示n个物品的价值，整数之间用一个空格隔开。

输出：对于每个测试例输出2行数据，其中，第一行含一个整数，表示装入背包物品的最大总价值；第2行含n个整数x1 , x2 , …xn，表示u1, u2, …, un 这n个物品是否放入背包。其中xi ={1, 0} , (i=1,2, …,n)。 如果xi =1，表示物品ui放入背包；如果xi =0，表示物品ui不放入背包。每个xi之间用一个空格隔开，两个测试例的输出数据之间用一个空行隔开，最后一个测试例后无空行。

3. 测试数据

输入：2

5 20

7 13 6 4 3

1. 7 3 2 3
2. 100

7 13 45 25 16 75 48 32

6 5 20 10 7 32 22 13

输出：13

1 0 1 1

48

1 0 1 0 0 0 1 0

4. 设计与实现的提示

(1) 寻找最优子结构、写出递归方程仍是问题求解的关键。

(2) 对于物品是否装入背包，可以有多种表示方式，这里仅采用0，1的方式，即xi=1表示物品ui装入背包；xi=0表示物品ui不装入背包。

(3) 由于在求解最优值时，并非所有的子问题都需要解，因此可以采用自顶向下的递归方法。

(4) 在求解最优值的过程中，注意最优解的相应信息的表示和记录。

5. 扩展内容

(1) 可以练习实现课本第7章的练习7.27可复选的背包问题，并与本题进行比较。

(2) 可以练习实现课本第7章的练习7.26 。

要求：取不同K值，求0/1背包问题的近似最优值，观察分析不同的K值对最优值的精确性的影响。

(3) 可以将本题与一般背包问题（一个物品可以部分装入背包）相比较，可以问题的复杂性和求解方法上有何不同。

# 2. 实验记录

## 2.1 最长递增子序列问题

### 2.1.1 最长递增子序列（longest increasing subsequence）

① 概述

在[计算机科学](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6)中，**最长递增子序列**（**longest increasing subsequence**）问题是指，在一个给定的数值序列中，找到一个子序列，使得这个子序列元素的数值依次递增，并且这个子序列的长度尽可能地大。最长递增子序列中的元素在原序列中不一定是连续的。

### 2.1.2 动态规划（recursion）

① 概述

多阶段过程转化为一系列单阶段问题，利用各阶段之间的关系，逐个求解，创立了解决这类过程优化问题的新方法——动态规划

② 基本思想

动态规划算法通常用于求解具有某种最优性质的问题。在这类问题中，可能会有许多可行解。每一个解都对应于一个值，我们希望找到具有[最优值](https://baike.baidu.com/item/%E6%9C%80%E4%BC%98%E5%80%BC)的解。动态规划算法与[分治法](https://baike.baidu.com/item/%E5%88%86%E6%B2%BB%E6%B3%95)类似，其基本思想也是将待求解问题分解成若干个子问题，先求解子问题，然后从[这些子](https://baike.baidu.com/item/%E8%BF%99%E4%BA%9B%E5%AD%90)问题的解得到原问题的解。与分治法不同的是，适合于用动态规划求解的问题，经分解得到子问题往往不是互相独立的。若用分治法来解这类问题，则分解得到的子问题数目太多，有些子问题被重复计算了很多次。如果我们能够保存已解决的子问题的答案，而在需要时再找出已求得的答案，这样就可以避免大量的重复计算，节省时间。我们可以用一个表来记录所有已解的子问题的答案。不管该子问题以后是否被用到，只要它被计算过，就将其结果填入表中。这就是动态规划法的基本思路。具体的动态规划算法多种多样，但它们具有相同的填表格式。

具体参见：

百度百科《动态规划》

<https://baike.baidu.com/item/%E5%8A%A8%E6%80%81%E8%A7%84%E5%88%92/529408?fr=aladdin>

### 2.1.3 实验思路 一个大的长序列X=<b1,b2,…,bn>，我们可以再找出一个L=<a1,a2,…,an> ，这个序列L即是对序列X按递增排序后的序列。此时显然我们要找的最长递增序列就是序列X与序列Y的最长公共子序列。 在《算法设计技巧与分析》一书中给出了最长公共子序列（LCS）的公式： 公式以图表展示：

**2.1.4 程序代码**

**Solution.java**

**package** Solution;

**import** java.util.Scanner;

**public** **class** Solution {

**private** **static** **int** *NUM* = 100;

//进行排序

**public** **static** **void** Sort(**int** r[], **int** n , **int** C[]){

**int** temp;

//冒泡排序

**for**(**int** i=0; i<n; i++){

**for**(**int** j=0; j<n-i-1; j++){

**if**(r[j]>r[j+1]){

temp = r[j];

r[j] = r[j+1];

r[j+1] = temp;

}

}

}

//把冒泡排序好的子序列放进一维数组中

**for**(**int** i=0; i<n; i++){

C[i] = r[i];

}

}

//计算子序列的长度

**public** **static** **void** Lcslength(**int** H[][], **int** n, **int** A[], **int** L, **int** B[]){

**int** i = n;

**int** j = n;

**int** k = L;

**int**[] C = **new** **int**[*NUM*];

**while**(i>0 && j>0){

**if**(H[i][j] == 0){

C[k--] = A[i];

i--;

j--;

} **else**{

**if**(H[i][j] == 1){

i--;

} **else**{

j--;

}

C[0] = B[0];

}

}

**for**(i=0; i<L; i++){

System.***out***.print(C[i] + " ");

}

}

//输出公共子序列所有所有元素，即最长递增序列

**public** **static** **void** Lcs(**int** A[], **int** B[], **int** n, **int** num){

**int** length;

**int**[][] L = **new** **int**[50][50];

**int**[][] H = **new** **int**[50][50];

//第一列为0

**for**(**int** i=0; i<n; i++){

L[i][0] = 0;

}

//第一行为0

**for**(**int** j=0; j<n; j++){

L[0][j] = 0;

}

//将公式转化为代码表示

**for**(**int** i=1; i<=n; i++){

**for**(**int** j=1; j<=n; j++){

**if**(A[i] == B[j]){

L[i][j] = L[i-1][j-1] + 1;

H[i][j] = 0;

} **else**{

**if**(L[i-1][j] >= L[i][j-1]){

L[i][j] = L[i-1][j];

H[i][j] = 1;

} **else**{

L[i][j] = L[i][j-1];

H[i][j] = 2;

}

}

}

}

System.***out***.println(L[n][n]);

length = L[n][n];

*Lcslength*(H, n, A, length, B);

}

**public** **static** **void** getdata(**int** []A, **int** [][]B,**int** n ) {

Scanner scan = **new** Scanner(System.***in***);

//进行循环输入

**for**(**int** i=0; i<n; i++){

A[i]= scan.nextInt();

**for**(**int** k=0;k<A[i];k++) {

B[i][k] = scan.nextInt();

}

}

}

**public** **static** **void** printfdata(**int** []E, **int** n, **int** []C, **int** i,**int** []D) {

*Sort*(E, n, C);

System.***out***.println("输出：");

*Lcs*(D,C,n,i);

System.***out***.println("");

System.***out***.println("");

}

**public** **static** **void** main(String[] args) {

//定义输入函数

Scanner scan = **new** Scanner(System.***in***);

//定向相关参数

**int** n,m;

**int**[] A = **new** **int**[*NUM*];

**int**[][] B = **new** **int**[*NUM*][*NUM*];

**int**[] C = **new** **int**[*NUM*];

**int**[] D = **new** **int**[*NUM*];

**int**[] E = **new** **int**[*NUM*];

System.***out***.print("请输入测试例的个数n：");

n = scan.nextInt();

*getdata*(A,B,n); //调用输入函数数据方法

**for**(**int** i=0; i<n; i++) {

**for**(**int** j=0; j<B[i].length; j++) {

E[j] = B[i][j];

D[j] = E[j];

}

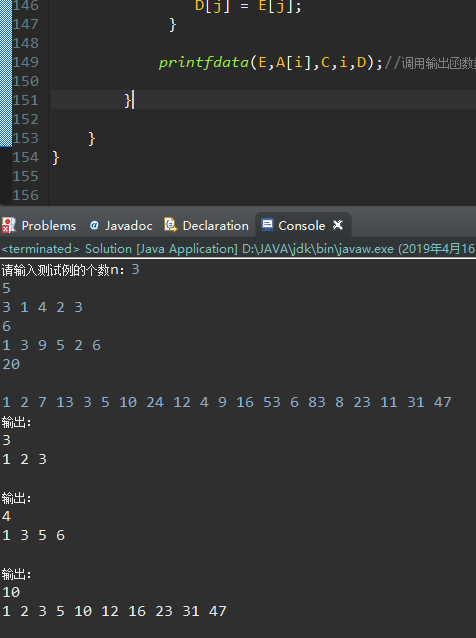
*printfdata*(E,A[i],C,i,D);//调用输出函数数据方法

}

}

}

## 2.1.5 实验结果



顺利输出实验预期结果

## 2.2 凸多边形的三角剖分

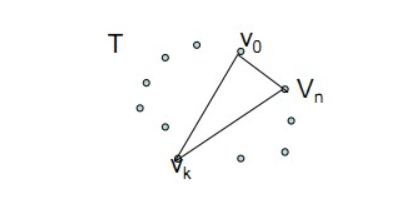
# 2.2.1 凸多边形的三角部分问题

① 问题相关  
(1)凸多边形的三角剖分：将凸多边形分割成互不相交的三角形的弦的集合T。

(2)最优剖分：给定凸多边形P，以及定义在由多边形的边和弦组成的三角形上的权函数w。要求确定该凸多边形的三角剖分，使得该三角剖分中诸三角形上权之和为最小。

②最优子结构性质

若凸(n+1)边形P={V0,V1……Vn}的最优三角剖分T包含三角形V0VkVn,1<=k<=n，则T的权为三个部分权之和：三角形V0VkVn的权，多边形{V0,V1……Vk}的权和多边形{Vk,Vk+1……Vn}的权之和。如下图所示：

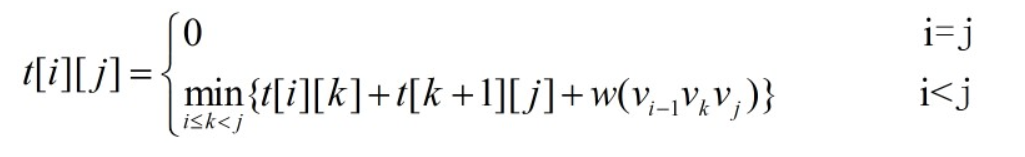


  可以断言，由T确定的这两个子多边形的三角剖分也是最优的。因为若有{V0,V1……Vk}和{V0,V1……Vk}更小权的三角剖分，将导致T不是最优三角剖分的矛盾。因此，凸多边形的三角剖分问题具有最优子结构性质。

③递推关系

设t[i][j],1<=i<j<=n为凸多边形{Vi-1,Vi……Vj}的最优三角剖分所对应的权值函数值，即其最优值。最优剖分包含三角形Vi-1VkVj的权，子多边形{Vi-1,Vi……Vk}的权，子多边形{Vk，Vk+1……Vj}的权之和。

因此，可得递推关系式：



# 2.2.2 实验思路 设置数组接收多边形个数，每个多边形每个顶点的（x,y）→定义length方法通过勾股定理X²+Y²=边长²→ 运用递归函数，不断调用自身，求出所有能分割的小三角权值→for循环找出最小值→四舍五入输出得数

# 2.2.3实验代码 Solution1.java

package Solution;

import java.text.DecimalFormat;

import java.util.Scanner;

public class solution1 {

private static float[][] length = new float[100][100];

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

int m;

int[] A = new int[100];

float[] x = new float[100];

float[] y = new float[100];

float[] s = new float[100];

System.out.print("输入：");

m = scan.nextInt();//要计算几个凸多边形

getdata(A,m,x,y,s);//调用输入数据函数

System.out.println("输出：");

printdata(m,s);//调用输出数据函数

}

public static void getdata(int []A,int m , float []x ,float []y , float[]s ) {

Scanner scan = new Scanner(System.in);

for(int i=0; i<m; i++) {

A[i] = scan.nextInt();//凸多边形的边数

for(int j=0; j<A[i]; j++) {

//输入凸多边形的各个点的坐标

x[j] = scan.nextFloat();

y[j] = scan.nextFloat();

}

length(A[i], x, y);//调用长度函数，计算每条边的长度

s[i] = Triangle(0, A[i]-1);//多个子三角形计算权值

}

}

public static void printdata(int m,float []length) {

for(int i=0;i<m;i++) {

//四舍五入得数，保留三位小数

String answer = new DecimalFormat("#.000").format(length[i]);

System.out.println(answer);

}

}

//求出每条边的长度

public static void length(int count, float []x, float []y){

//令空数组全部为0

for(int i=0; i<100; i++){

for(int j=0; j<100; j++){

length[i][j] = 0;

}

}

for(int i=0; i<count; i++){

for(int j=0; j<count; j++){

//求出每条边的长度

length[i][j] = (float) Math.sqrt((x[i]-x[j])\*(x[i]-x[j])+(y[i]-y[j])\*(y[i]-y[j]));

}

}

}

//找出最优三角形与最优解

public static float Triangle(int zero, int num){

int m = 0;

float answer = 40000;

float[] A = new float[100];

if(num-zero ==2){

return 0;

}

//最优三角部分

else{

for(int k=zero+1; k<num; k++){

//递归调用自身，求出所有小三角形的权值

if(k==zero+1){

A[m] = Triangle(k, num) + length[zero+1][num];

} else if(k==num-1){

A[m] = Triangle(zero, k) + length[zero][num-1];

} else{

A[m] = Triangle(k, num) + length[k][num] + Triangle(zero, k) +length[zero][k];

}

m++;

}

//不断覆盖，找出最优解

for(m=0; m<num-zero-1; m++){

if(answer>A[m]){

answer = A[m];

}

}

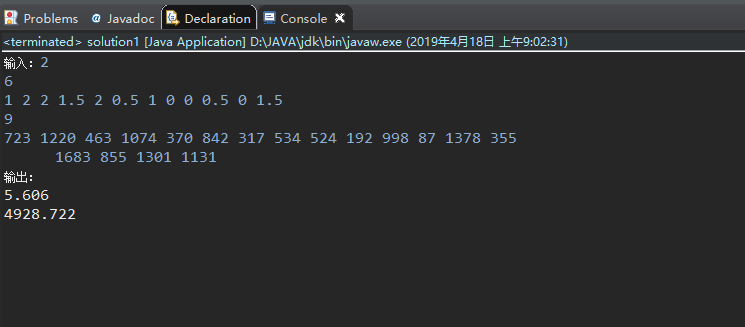
return answer;

}

}

}

# 2.2.4 实验结果

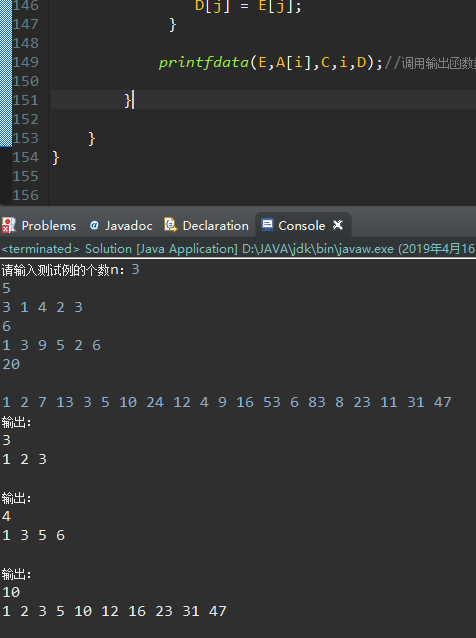


实验结果符合预期

# 3. 实验总结

# 3.1 最长递增子序列问题

# 3.1.1 实验结果展示



# 3.1.2 关键代码

//第一列为0

**for**(**int** i=0; i<n; i++){

L[i][0] = 0;

}

//第一行为0

**for**(**int** j=0; j<n; j++){

L[0][j] = 0;

}

//将公式转化为代码表示

**for**(**int** i=1; i<=n; i++){

**for**(**int** j=1; j<=n; j++){

**if**(A[i] == B[j]){

L[i][j] = L[i-1][j-1] + 1;

H[i][j] = 0;

} **else**{

**if**(L[i-1][j] >= L[i][j-1]){

L[i][j] = L[i-1][j];

H[i][j] = 1;

} **else**{

L[i][j] = L[i][j-1];

H[i][j] = 2;

}

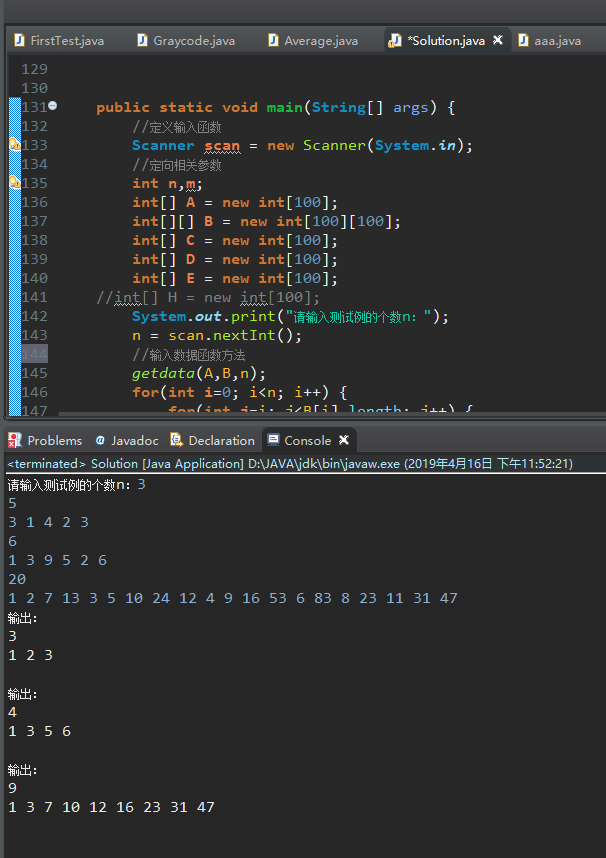
}

}

这段代码就是前文写的公式的体现，把他用图表示出来即为前文问题分析的图

# 3.1.3 实验过程错误分析与总结

实验过程中遇到过错误，先放图



可以看到，在第三行的递增子序列处出了问题，原本输出应为10 1 2 3 5 10 12 16 23 31 47.观察到是从10前面开始出错的，推断是for循环的循环控制出错了。

经过调试，终于发现是在二维数组传入一维数组时出错了，出错代码如下：

**for**(**int** i=0; i<n; i++) {

**for**(**int** j=i; j<B[i].length; j++) {

E[j] = B[i][j];

D[j] = E[j];

}

printfdata(E,A[i],C,i,D);

}

正确代码如下：

**for**(**int** i=0; i<n; i++) {

**for**(**int** j=0; j<B[i].length; j++) {

E[j] = B[i][j];

D[j] = E[j];

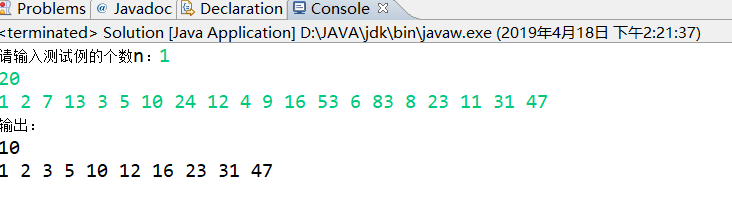
}

printfdata(E,A[i],C,i,D);

}

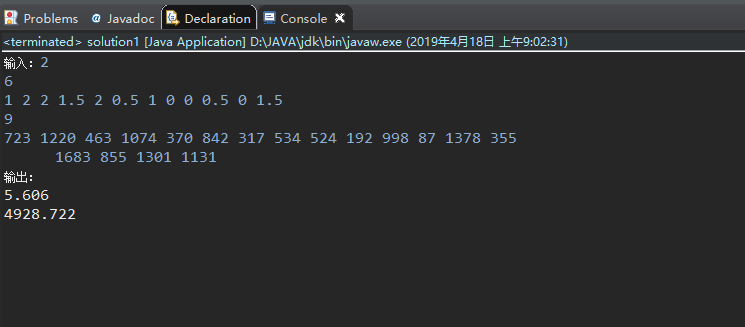
这段代码的功能是将我之前getdata的数据（之前用二维数组存储的）放到一个一维数组中，用for循环加以控制，但是原本我在写的时候k从i开始，这就漏掉了前几个数，导致后面再遍历输出时出错了。问题结束。

但是有趣的是我用错误代码单独运行输入第三行数据时，答案是对的，如图所示：



# 3.2 凸多边形三角部分问题

# 3.2.1实验结果展示



# 3.2.2 关键代码

求出凸多边形的每条边的长度，直角坐标系中两点坐标差的勾股定理可以得出。

**for**(**int** i=0; i<count; i++){

**for**(**int** j=0; j<count; j++){

//求出每条边的长度

*length*[i][j] = (**float**) Math.*sqrt*((x[i]-x[j])\*(x[i]-x[j])+(y[i]-y[j])\*(y[i]-y[j]));

}

}

调用自身，分割成多个小三角形，将权值存入一维数组中。

**if**(num-zero ==2){

**return** 0;

}

**else**{

**for**(**int** k=zero+1; k<num; k++){

//递归调用自身，求出所有小三角形的权值

**if**(k==zero+1){

A[m] = *Triangle*(k, num) + *length*[zero+1][num];

} **else** **if**(k==num-1){

A[m] = *Triangle*(zero, k) + *length*[zero][num-1];

} **else**{

A[m] = *Triangle*(k, num) + *length*[k][num] + *Triangle*(zero, k) +*length*[zero][k];

}

m++;

}

不断覆盖，找出最小的权值，即最优解

//不断覆盖，找出最优解

**for**(m=0; m<num-zero-1; m++){

**if**(answer>A[m]){

answer = A[m];

}

}

**return** answer;

}

四舍五入方法输出结果：

**for**(**int** i=0;i<m;i++) {

//四舍五入得数，保留三位小数

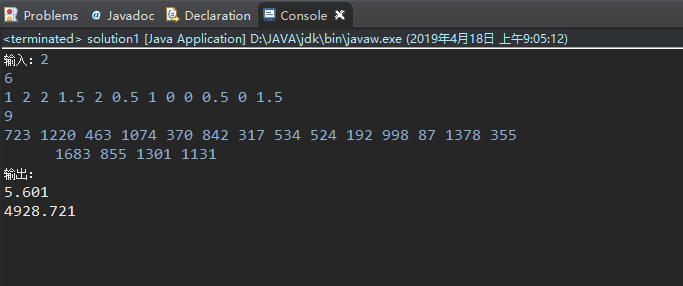
String answer = **new** DecimalFormat("#.000").format(length[i]);

System.***out***.println(answer);

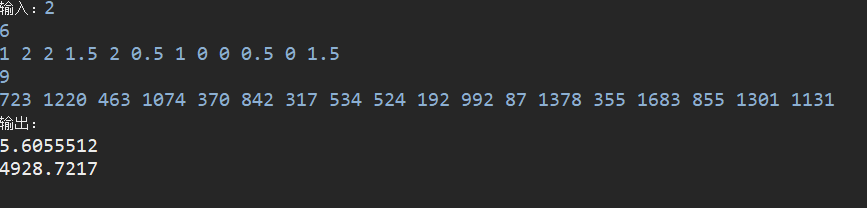
}

# 3.1.3 实验过程错误分析与总结

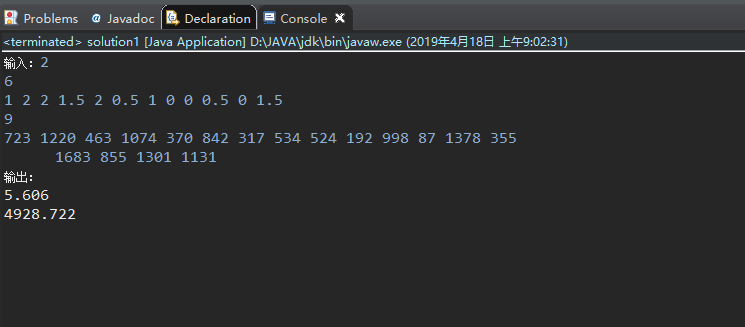
在实验中，刚开始完成代码时四舍五入的方法用错了结果得数出错，但是又跟老师的得数十分接近，所以甚至一度怀疑老师的答案给的是错误的，错误截图如下：



后来不四舍五入打出得数查看得到的确是我四舍五入时出错了：



后来改善了我的四舍五入输出代码才得以输出正确结果：



# 3.3 实验总结

这次的实验运用到了动态规划的相关知识，实验完成的时间相比实验一大大增加了，难度也增加了许多，但是我在这次实验中学习到了很多。

其实这两道问题就是将问题不断变小，然后通过若干个小问题找出最优解，从而达到目的。对我来说难度挺大，以至于在完成前两题时，剩余时间已经不够支持我在完成选做题了。还有就是代码量太大了，很多都是冗余的代码，没有能更好的做到精简代码。这点做的不是很好。最后，对于时间复杂度，我认为最长递增子序列算法的时间复杂度应为：

O(nlogn)，凸多边形三角部分的时间复杂度为：O（n^3）。

这次的实验深刻体会到了痛苦是发现了BUG，快乐是解决了BUG。随着一个个错误被解决，在收获知识的同时还收获了快乐。

最后，长篇实验报告完成，辛苦老师批阅。