

**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** | **实验五 贪心算法设计与应用** | | | **实验成绩** |  |
| **学生姓名** |  | **学 号** | **157** | **年级专业班级** |  |
| **小组成员** | **无** | | | **实验日期** | **2019年4 月25日** |

# 1. 实验目的和要求

## 1.1 实验目的

理解贪心算法的基本原理，掌握贪心算法设计的基本方法及其应用。

## 1.2 实验软硬件环境

① 操作系统：Windows 10中文家庭版

② 编译环境：jdk8，idea 2018

## 1.3 实验要求

1.3.1 问题一

* 1. **问题描述**

一个旅行家想驾驶汽车从城市A到城市B（***设出发时油箱是空的***）。给定两个城市之间的距离dis、汽车油箱的容量c、每升汽油能行驶的距离d、沿途油站数n、油站i离出发点的距离d[i]以及该站每升汽油的价格p[i],i=1,2,…,n。设d[1]=0<d[2]<…<d[n]。要花最少的油费从城市A到城市B，*在每个加油站应加多少油，最少花费为多少*？

* 1. **具体要求**

Input

输入的第一行是一个正整数k，表示测试例个数。接下来几行是k个测试例的数据，每个测试例的数据由三行组成，其中第一行含4个正整数，依次为A和B两个城市之间的距离d1、汽车油箱的容量c（以升为单位）、每升汽油能行驶的距离d2、沿途油站数n (1<=n<=200)；第二行含n个实数d1, d2 ,…, dn，表示各油站离出发点的距离（d1=0）；第三行含n个实数p1, p2 ,…, pn，表示各油站每升汽油的价格。同一行的数之间用一个空格隔开。

**Output**

对于每个测试例输出一行，含一个实数，表示从城市Ａ到城市Ｂ所要花费的最少油费（输出的结果精确到小数点后一位）。若问题无解，则输出“No Solution”。

* 1. **测试数据**

Sample Input

2

1500 50 10 4

0 300.0 800.0 1200.0

4.0 5.0 4.0 4.5

1000 40 10 3

0 500.0 750.0

4.5 5.0 4.2

Sample Output

640.0

No Solution

### 1.3.2 问题二

1. **问题描述**

设平面上分布着n个白点和n个黑点，每个点用一对坐标（x, y）表示。一个黑点b=（xb,yb）支配一个白点w=(xw, yw)当且仅当xb>=xw和yb>=yw。若黑点b支配白点w，则黑点b和白点w可匹配（可形成一个匹配对）。在一个黑点最多只能与一个白点匹配，一个白点最多只能与一个黑点匹配的前提下，求n个白点和n个黑点的最大匹配对数。

1. **具体要求**

Input

输入的第一行是一个正整数k，表示测试例个数。接下来几行是k个测试例的数据，每个测试例的数据由三行组成，其中第一行含1个正整数n(n<16)；第二行含2n个实数xb1, yb1,xb2, yb2,…, xbn, ybn， (xbi, ybi)，i=1, 2, …, n表示n个黑点的坐标；第三行含2n个实数xw1, yw1,xw2, yw2,…, xwn, ywn，(xwi, ywi)，i=1, 2, …, n表示n个白点的坐标。同一行的实数之间用一个空格隔开。

**Output**

对于每个测试例输出一行，含一个整数，表示n个白点和n个黑点的最大匹配对数。

1. **测试数据**

Sample Input

1

3

5.0 3.0 5.0 -1.0 4.0 4.0

2.0 3.5 2.0 2.0 -2.0 -2.0

Sample Output

3

# 2. 实验记录

## 2.2 实验过程

### 问题一

#### 2.2.1 实验思路

#### 2.2.1.1 问题一实现思路

这题的点就是：如果当前的油价低于下一站的油价，就在当前站点加满油，能开到哪里就开到哪里，到下一站再加油；如果当前的油价高于下一站，就刚刚好加油到可以开到下一站。各个站点之间就是执行上面这样的迭代操作。还需要注意这样：就是当前的油加满可以达到下一站但不可以到达下下一站，则就要在下一站加上适量的油，这些油可以让汽车开完剩下的路。对于该贪心算法的准确性：即局部最优解为全局最优解的证明：这里是每一次都选择最便宜的油价的加油站来加油，这样可以保证最后行驶到目的地的总花费最小。如果，当中，选择了一个比较贵的（相比起来比较贵），则最后，肯定总共的花费会大于每次都选择比较便宜油价加油的。对于这个算法的时间复杂度，在看完代码以后，在[总结](#_3.1问题一)部分说明。

#### 2.2.2 程序代码

#### 2.2.2.1 问题一程序代码

import java.util.Scanner;

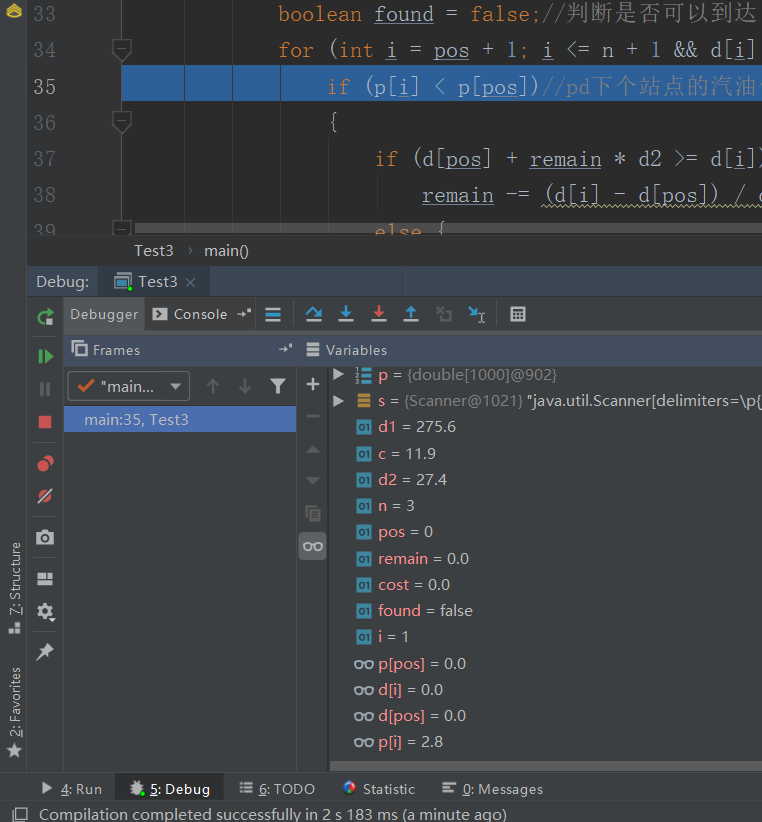
public class Test3 {  
  
 public static void main(String[] args) {  
 double[] d = new double[1000];//用来存放加油站  
 double[] p = new double[1000];//用来存放每一个加油站的油费  
 double d1;//两个城市之间的距离  
 double c;//汽车油箱的容量  
 double d2;//该汽车每一升汽油可以行使的距离  
 int n;//加油站的数目  
 Scanner s = new Scanner(System.*in*);  
 System.*out*.println("请输入两个城市之间的距离");  
 d1 = s.nextDouble();  
 System.*out*.println("请输入汽车油箱的体积");  
 c = s.nextDouble();  
 System.*out*.println("请输入该汽车每一升汽油可以行使的距离");  
 d2 = s.nextDouble();  
 System.*out*.println("请输入加油站的数目");  
 n = s.nextInt();  
 d[n + 1] = d1;//第n个加油站后即为最后的终点  
 for (int i = 1; i <= n; i++) {  
 System.*out*.println("请输入第" + i + "个加油站离出发点的距离：");  
 d[i] = s.nextDouble();  
 }  
 for (int i = 1; i <= n; i++) {  
 System.*out*.println("请输入第" + i + "个加油站的油价：");  
 p[i] = s.nextDouble();  
 }  
 int pos = 0;//当前加油站  
 double remain = 0, cost = 0;  
 do {  
 boolean found = false;//判断是否可以到达  
 for (int i = pos + 1; i <= n + 1 && d[i] <= d[pos] + c \* d2; i++) {  
 if (p[i] < p[pos])//pd下个站点的汽油价钱 和当前站点的比较  
 {  
 if (d[pos] + remain \* d2 >= d[i])//如果剩下的油量够就直接开过去  
 remain -= (d[i] - d[pos]) / d2;//剩下的油量减少  
 else {  
 //剩下的油量不够  
 cost += ((d[i] - d[pos]) / d2 - remain) \* p[pos];//还需要加的油发费的钱  
 remain = 0;//到达下一站油恰好用完了  
 }  
 pos = i;  
 found = true;  
 break;  
 }  
 }  
 if(!found){  
 cost+=(c-remain)\*p[pos];//不能到达前花费了多少钱  
 remain=c-(d[pos+1]-d[pos])/d2;//还剩下多少油  
 if(remain>=0)//大于0，油足够到达下一站  
 pos++;  
 else {  
 //小于0，油根本不可能满足到达下一站  
 System.*out*.println("No Solution");  
 }  
 }  
 } while (pos <= n);  
 System.*out*.println("最小花费为："+cost);  
  
  
 }  
}

#### 2.2.2.2问题一程序代码探究

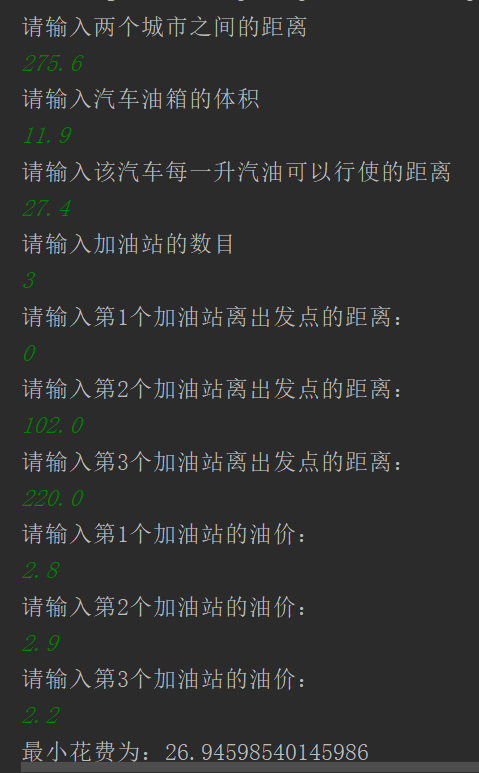
刚开始一直输出是0，开启idea的debug模式，终于找到了一个细节的错误：

如图，pos为0，显示d[pos]=0即d[0]=0，而我们这里输入的时候全部是从1开始的。

所以，这里所有的数组的下标均是从1开始的，我将其修改为pos=1，OK了：



#### 2.3 实现结果



### 问题二

#### 2.2.2 实验思路

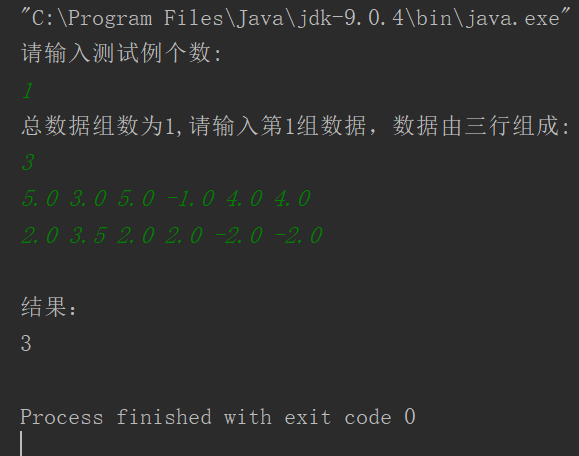
这里题目的意思是一个黑点需要匹配一个白点，需要满足黑点的恒纵坐标均要大于等于白点的横纵坐标。创建点对象，在该点对象里面实现类来继承comparetor接口，再重写compare方法，可以实现定制排序。这样，可以对拿到的黑点和白点对象可以分别进行排序。这里是实现从小到大的排序。这里其实需要注意到，一个点只能匹配一次，不能多次匹配，这个我一开始没有想到，后面试了几个样例以后就有问题了，我在下面的[**2.2.4 实现结果**](#_2.2.4_实现结果)**中对于这个问题的发现和解决过程进行了阐述，可以点击蓝色的链接查看。对于这个算法的时间复杂度以及我关于这个算法的优化的拙见可以在后面的**[**总结**](#_3.2_问题二)**中查看，这里先不阐述。**

#### 2.2.3 程序代码

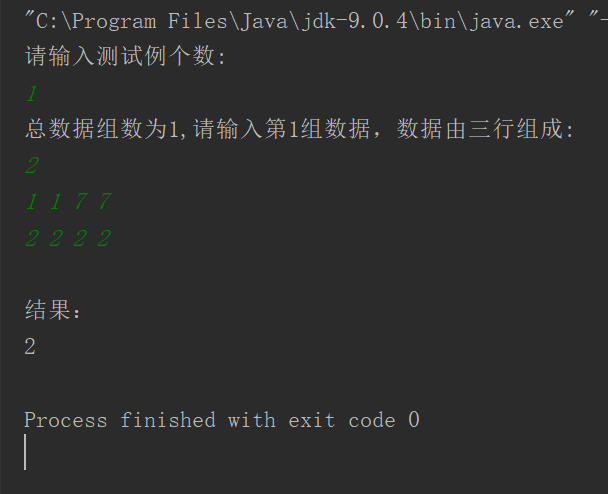
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.util.Collections;  
import java.util.Comparator;  
import java.util.Vector;  
  
public class BlackAndWhite {  
 public void Test(int m) throws IOException {  
 int[] result=new int[m];  
 for(int i=0;i<m;i++){  
 try {  
 BufferedReader br=new BufferedReader(new InputStreamReader(System.*in*));  
 System.*out*.println("总数据组数为"+m+",请输入第"+(i+1)+"组数据，数据由三行组成:");  
 String str1=br.readLine();  
 int n=Integer.*parseInt*(str1);//几个点  
 String str2=br.readLine();  
 String[] str3=str2.split(" ");//黑点的数据  
 String str4=br.readLine();  
 String[] str5=str4.split(" ");//白点的数据  
 //vecotr是线程安全的ArrayList，可以用在多线程中，但是效率不高。后面建议换成分ArrayList,单线程效率高  
 Vector b=new Vector();//黑  
 Vector w=new Vector();//白  
 //往创建的数据结构中添加点对象  
 for(int j=0;j<n;j++){  
 //(Float.parseFloat(str3[2\*j]),Float.parseFloat(str3[2\*j+1]))是每一个点的坐标  
 //依次实例化点对象  
 b.add(j,new MyPoint(Float.*parseFloat*(str3[2\*j]),Float.*parseFloat*(str3[2\*j+1])));  
 w.add(j,new MyPoint(Float.*parseFloat*(str5[2\*j]),Float.*parseFloat*(str5[2\*j+1])));  
 }  
 //进行。。。。。  
 result[i]=*WB*(w,b);  
 }catch(IOException e){  
 System.*out*.println(e);  
 }  
 }  
 System.*out*.println();  
 System.*out*.println("结果：");  
 for(int r=0;r<m;r++) {  
 System.*out*.println(result[r]);  
 }  
 }  
 //核心算法  
 public static int WB(Vector w,Vector b){  
 Collections.*sort*(w,new MyPoint.MyPointComparatorByXAC());//定制排序,从小到大  
 Collections.*sort*(b,new MyPoint.MyPointComparatorByXAC());//定制排序  
 int pp=0;  
 //自底向上（白棋)(最大的）  
 for(int i=w.size()-1;i>=0;i--){  
 boolean MATCH=false;  
 //自底向上（黑棋)（最大的）  
 for(int j=b.size()-1;j>=0;j--){  
 MyPoint mb=(MyPoint)b.get(j);//获得黑点对象  
 MyPoint mw=(MyPoint)w.get(i);//获得白点对象  
 if(mb.getX()<mw.getX()){  
 break;//如果黑点对象的x坐标最大都小于白点的化，就直接break，这一点nos（后面就不必要再下去啦)，去拿更小的白点来比较  
 }  
 if(mb.getY()<mw.getY()){  
 continue;//寻找下一个黑棋点  
 }  
 //有匹配的情况下需要给标志对象赋值  
 MATCH=true;  
 }  
 if (MATCH)//有匹配才加一。  
 pp++;  
  
 }  
 return pp;  
 }  
}  
  
class MyPoint{  
 private float x;  
 private float y;  
 public void setX(float x){  
 this.x=x;  
 }  
 public float getX(){  
 return x;  
 }  
 public void setY(float y){  
 this.y=y;  
 }  
 public float getY(){  
 return y;  
 }  
 //有参构造方法，用于实例化点对象  
 public MyPoint(float x,float y){  
 this.x=x;  
 this.y=y;  
 }  
 //通过继承Comparator来定制排序  
 public static class MyPointComparatorByXAC implements Comparator {  
 //这里是重写compare方法  
 public int compare(Object o1,Object o2){  
 MyPoint p1=(MyPoint)o1;  
 MyPoint p2=(MyPoint)o2;  
 int result;  
 if(p1.getX()>p2.getX()){  
 result=1;  
 }else if(p1.getX()==p2.getX()){  
 result=0;  
 }else{  
 result=-1;  
 }  
  
 return result;  
 }  
 }  
  
 public static void main(String[] args) throws Exception{  
 BlackAndWhite t = new BlackAndWhite();  
 BufferedReader br=new BufferedReader(new InputStreamReader(System.*in*));  
 System.*out*.println("请输入测试例个数:");  
 String str=br.readLine();  
 int m=Integer.*parseInt*(str);  
 t.Test(m);  
 }  
}

#### 2.2.4 实现结果

用上面的代码，输入样例，可以成功：



**但是，**又试了一下下面的这个数据，***gg了***：



看到没有，这里一个点匹配了两个！这和题目中的一个点仅仅能匹配一个点是不符合的！所以需要修改代码。

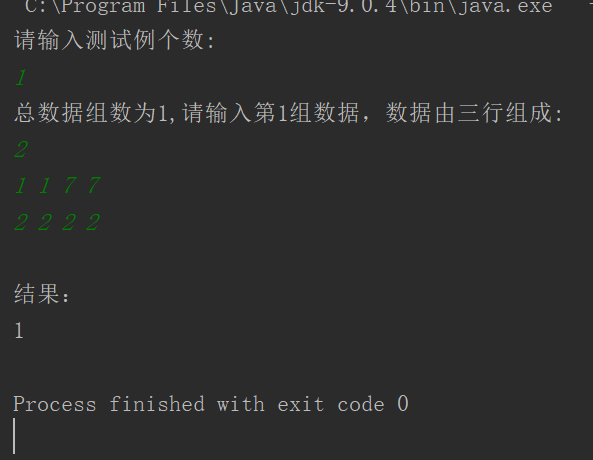
***既然一个点仅仅只能匹配一个，那就让匹配过的黑点变成最小不就完事了。***

修改代码：

在原来的代码的基础上加上：

if(mb.getY()<mw.getY()){  
 continue;//寻找下一个黑棋点  
 }  
 //有匹配的情况下需要给标志对象赋值  
 MATCH=true;  
 mb.setX(-32768);  
 mb.setY(-32768);  
}

这样在测试一遍：



可以看到，结果是正确的！

**ps：**这里为什么需要设置成-32768，这是-2^15。int需要两个字节，而一个字节是8位，所以共16位。在计算机组成原理里面学过，第一位是一个符号位（补码），所以是2^15，加上负号就为-2^15即-32768。

# 3. 实验总结

## 3.1问题一

这个实验，其实比较简单，比较容易想到思路，实现起来也比较容易。虽然一开使出现了一些小错误，但是通过idea的debug模式，迅速找到了错误所在，这里数组我从1开始，而平时一般这个从0开始，这个在上下代码种没有做到统一，导致了错误，好在只是小问题，很快解决了。现在我们来研究这个算法的时间复杂度：其实可以很明显的看出来，这个主程序中有2个循环（嵌套），所以很容易的得到其时间复杂度为O(n2)。

## 3.2 问题二

这个问题，刚刚拿到手，其实还是觉得比较容易上手的，不就是比较大小吗？其实也就是那样。。。这里需要注意的是：需要一个match匹配标志位，要不然总的结果会多一个1!这个刚刚解决了，以为万事俱备了，哈哈，后来又随便测试了一组数据，gg了（没毛病，就是随便测试的！），发现，一个点居然匹配了两个点，这可不行。那就让匹配后的点变成最小（没有人比他更小），不就完事了。这样就解决了。

现在来说说这个算法的时间复杂度：核心算法里面有两个循环，故时间复杂度为O(n2)。对于优化方面，我只能想出来***数据结构方面的优化***。上面的代码中，用到vector，这个东西，是适合多线程安全的（相当于多线程安全的ArrayList），但是，在多线程安全下，我们可以知道，其效率肯定没有单线程下的ArrayList的高，所以，我们可以选用ArrayList来代替Vector，效率方面应该会有所提高（这是从理论方面，因为如果没有多线程的情况下，ArrayList就是比Vector效率高，这两个的区别和使用在学习java集合的时候有学习过）（Vector的本质就是一个线程安全的ArrayList）。

--- 2019年5月23日星期四 江毅