

**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** | **实验三 贪心算法设计与应用** | | | **实验成绩** |  |
| **学生姓名** |  | **学 号** | **116** | **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

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

② Intellij IDEA

## 1.3 实验要求

### 1.3.1加油问题（Problem Set 1702）：

#### 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？

#### ②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”。

#### ③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

#### ④设计与实现的提示

1. 注意考虑无解的情况
2. 对终点站可进行特殊处理

#### ⑤扩展内容

(1) 演示时建议采用可视化界面

(2) The Express Mail(Problem Set 1755)

### 1.3.2黑白点的匹配（Problem Set 1714）：

#### 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。

#### ②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。

#### ③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

#### ④扩展内容

(1) 建议采用可视化界面

# 2. 实验记录

## 2.1 贪心算法

### 2.1.1 基本原理

贪心法是一种算法设计技术，通常用于求解最优化问题。

通过一系列选择步骤来构造问题的解，每一步都是对当前部分解的一个扩展，直至获得问题的完整解。所做的每一步选择都必须满足：

1）可行的：必须满足问题的约束。

2）局部最优：当前所有可能的选择中最佳的局部选择。

3）不可取消: 选择一旦做出，在后面的步骤中就无法改变了。

要注意的是，贪心法不能保证总能得到最优解（一系列的局部最优选择不能保证最后得到整体最优解）

### 2.1.2该类算法设计与实现的要点

贪心算法往往效率高，一般时间复杂性为多项式阶。贪心算法一般较简单，其关键和难点在于贪心选择策略的确定，以及证明相应的贪心算法确实可求出最优解。

## 2.2 实验过程

### 2.2.1 程序代码

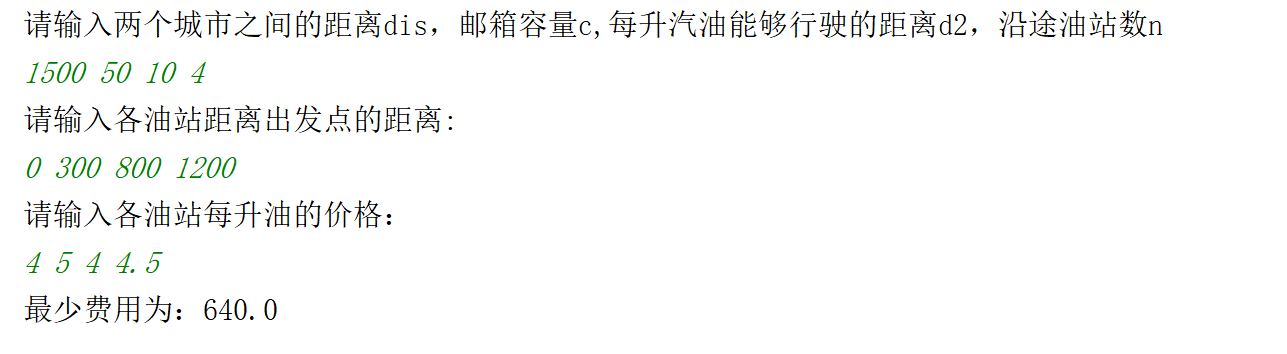
#### 2.2.1.1加油问题

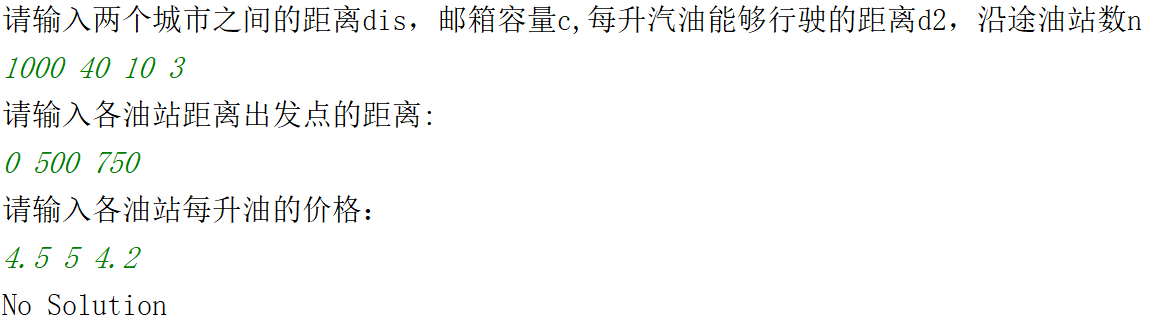
import java.util.Scanner;  
  
public class Test3\_1 {  
  
 public static void main(String[] args){  
 int dis,c,d2,n;  
 System.out.println("请输入两个城市之间的距离dis，邮箱容量c,每升汽油能够行驶的距离d2，沿途油站数n");  
 Scanner scan=new Scanner(System.in);  
 dis=scan.nextInt();  
 c=scan.nextInt();  
 d2=scan.nextInt();  
 n=scan.nextInt();  
 double[] str=new double[n+1];  
 double[] pay=new double[n];  
 System.out.println("请输入各油站距离出发点的距离:");  
 for(int i=0;i<n;i++)  
 str[i]=scan.nextDouble();  
 str[n]=dis;  
 System.out.println("请输入各油站每升油的价格：");  
 for(int i=0;i<n;i++)  
 pay[i]=scan.nextDouble();  
  
 refuel(dis,c,d2,n,str,pay);  
 }  
  
  
 **public static void refuel(double dis, double c, int d2, int n, double[] str, double[] pay) {  
 double distance=str[1]-str[0];  
 if (c \* d2 < distance) {  
 System.out.println("No Solution");  
 return;  
 }  
 double cost = 0;//设置初始费用  
 int flag=0;  
  
 //在出发点加油  
 if (pay[0] < pay[1]) {  
 cost += pay[0] \* c;  
 } else  
 cost += pay[0] \* ((str[1]-str[0]) / d2);  
  
 //设置剩余油量(第i段路后）  
 double c1 = c - (str[1]-str[0]) / d2;  
 //这里第一个问题在于 c1的值仍然为c没有变过  
  
  
 //System.out.println("dis:"+dis+" c:"+c+" d2:"+d2+" n:"+n+" str:"+str[0]+" pay:"+pay[0]+" c1:"+c1);  
  
 for (int i = 1; i < n; i++) {  
 flag=0;  
 //System.out.println("进入第"+i+"次大循环"+cost+" "+c1);  
 distance=str[i+1]-str[i];  
 if(distance>c\*d2){  
 System.out.println("No solution");  
 return;  
 }  
 if(i==n-1){  
 if(c1\*d2<distance){//如果不够跑最后一段路  
 cost+=(distance/d2-c1)\*pay[i];  
 System.out.println("最少费用为："+cost);  
 return;  
 }  
 System.out.println("最少费用为："+cost);  
 return;  
 }  
 //剩下的油够跑第i段路程  
 if (c1 \* d2 > distance) {  
 //System.out.println("进入第"+i+"次够不够跑判断");  
 if (pay[i] < pay[i + 1]){  
 c1 = c1 - distance / d2;  
 //System.out.println("进行第"+i+"次判断，不补充油");  
 }  
  
 else {  
 c1 = c1 - distance/ d2;  
 cost =cost+ (c - c1) \* pay[i];  
 c1 = c;  
 //System.out.println("补充油，油量为"+c1+" 花费为:"+cost);  
 }  
 continue;  
 }  
 //如果剩下的油不够跑第i段路程，且第i个加油站的邮费比后面的所有加油站都便宜，则直接补满  
 for(int j=i;j<n;j++){  
 //System.out.println("进入小循环"+i+"."+j);  
 if(pay[i]>pay[j])  
 flag=1;  
 //System.out.println("flag="+flag);  
 }  
 //System.out.println("c1:"+c1);  
 if(flag==0){  
 //System.out.println("进入flag=0判断,c1:"+c1);  
 cost+=(c-c1)\*pay[i];  
 c1=c-distance/d2;  
 //System.out.println("c1:"+c1+" cost:"+cost);  
 }  
 //如果剩下的油不够跑第i段路程，且第i个加油站的邮费不比所有的加油站都便宜，则只补满需要跑i段路程的油  
 else {  
 //System.out.println("进入flag=1判断前c1的值:"+c1+" cost:"+cost);  
 cost += pay[i] \* ((distance/d2) - c1);  
 //System.out.println(" distance:"+distance);  
 c1=0;  
 //System.out.println("进入flag=1判断后c1的值:"+c1);  
 }  
 }  
 }  
  
}**

#### 2.2.1.2 黑白点的匹配

## 2.3 实现结果

#### 2.3.1加油问题





实验结果符合预期。

#### 2.3.2 黑白点的匹配

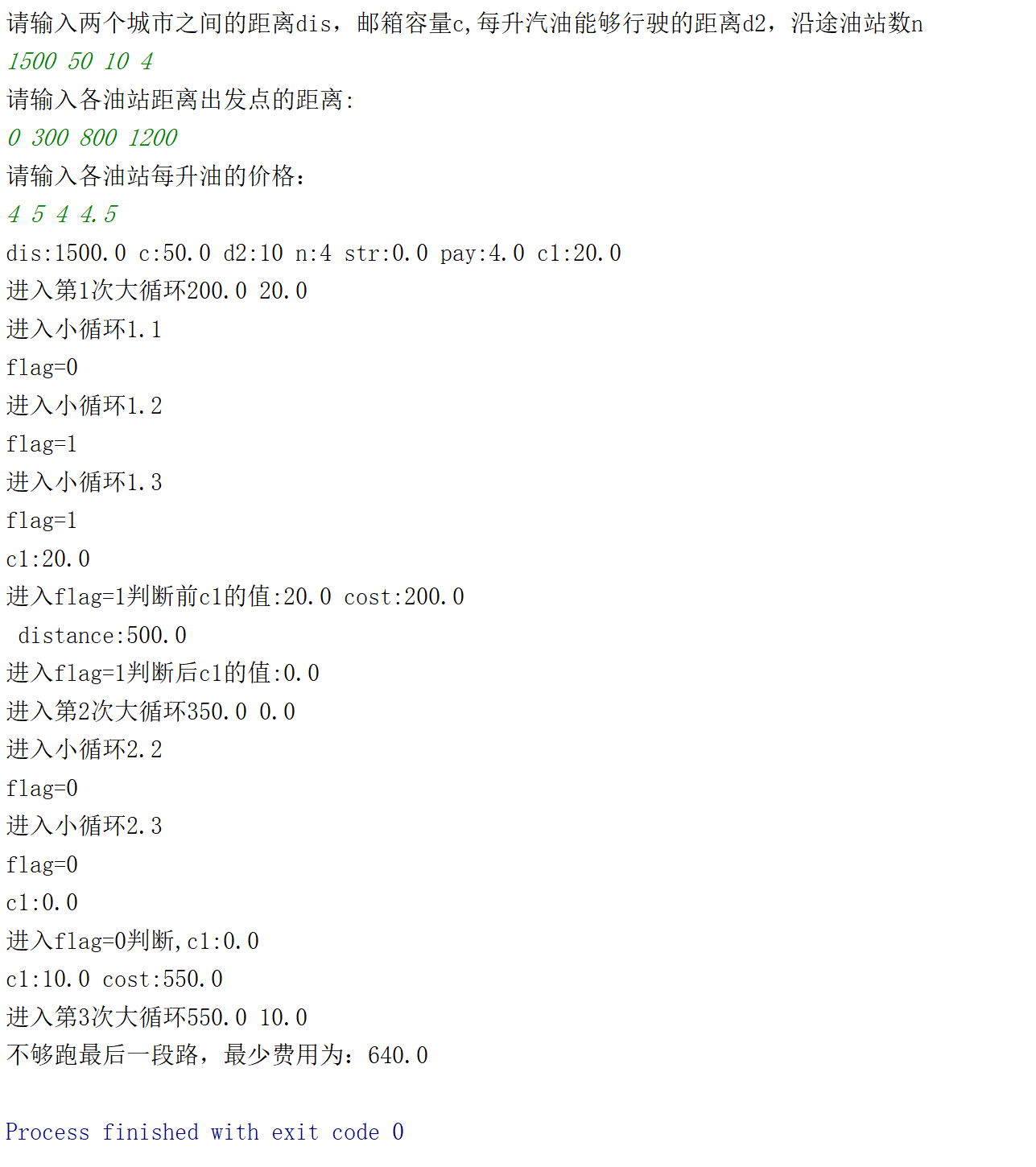
# 3. 实验总结

## 3.1 加油问题

加油问题是我第一个没有查找任何资料就直接开始上手的问题。从头到尾全部是我自己一个人瞎鼓捣鼓捣出来的。我以为贪心算法比较简单能够让我快乐一下，但是事实上并没有很快乐，想思路挺简单的，写代码也挺简单的，大概半个小时就写完了大概，结果查Bug查了三四个小时，全程保持着无能狂怒的状态。

第一次测试的时候发现自己把要输入的加油的邮费放到距离里去了，导致怎么出都没出结果。第二次就是发现完全把str，也就是离起点的距离当成了两个加油站之间的间隔。第三次是剩余油量c1的逻辑错误，我最开始设置的c1是在跑完第i段以后的c1，结果写着写着就写成了跑之前的c1。查这个错误的时候我差点把电脑从宿舍里丢出去，在做第一组数据测试的时候，我手算的第二段路程的费用是350，但是输出的数据是250，然后我在c1重置以后的基础上，算出了增加的油费是150，原来的油费是200，重复测试5次，每次我都以为是150+200=250,愤怒的我一直锤桌子，期间电脑还死机了一次……最后发现以后，感觉自己才是250。第四次就是，最后发现，我设置的距离distance并不是第i段的距离，而是第i-1段的distance，所以第一组测试数据最后一直是550而不是640……但是写完以后得出正确数字的时候真的很快乐。

下面这个图是我在插入了一堆输出测试结论终于调试成功的图，640一出来的瞬间我简直要抱头痛哭喜极而泣了：



## 3.2 黑白点的匹配