**实验三 贪心算法设计与应用**

目录

[一．贪心算法详解 1](#_Toc8315011)

[二．该类算法设计与实现的要点 2](#_Toc8315012)

[三．实验目的和要求 2](#_Toc8315013)

[四．实验一内容 2](#_Toc8315014)

[加油问题（Problem Set 1702）： 2](#_Toc8315015)

[五：实验一步骤分析 3](#_Toc8315016)

[（一）初步设想，思考 3](#_Toc8315017)

[（二）看网上大佬思路 4](#_Toc8315018)

[（三）代码实现 5](#_Toc8315019)

[（四）证明其为最优解 7](#_Toc8315020)

[（五）实验结果展示 7](#_Toc8315021)

[（六）复杂度分析 7](#_Toc8315022)

[（七）问题一总结 7](#_Toc8315023)

[六：实验二内容 7](#_Toc8315024)

[黑白点的匹配（Problem Set 1714）： 7](#_Toc8315025)

[Input 7](#_Toc8315026)

[Sample Input 7](#_Toc8315027)

[Sample Output 8](#_Toc8315028)

[七.实验二步骤分析 8](#_Toc8315029)

[（一）初步设想，思考 8](#_Toc8315030)

[（二）看网上大佬思路 9](#_Toc8315031)

[（三）代码实现 9](#_Toc8315032)

[（四）证明其为最优解 9](#_Toc8315033)

[（五）实验结果展示 9](#_Toc8315034)

[（六）复杂度分析 9](#_Toc8315035)

[（七）问题一总结 9](#_Toc8315036)

## 一．贪心算法详解

（一）基础原理

贪心法用来求解最优化问题。

通常包含一个用以寻找局部最优解的迭代过程，选择是局部最优的

（二）求解步骤

通过一系列选择步骤来构造问题的解（局部最优），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），但因其每一步的工作量很小且基于少量信息，所以算法可以在花更少的时间内获得较好的解。

（三）优势

1.效率高，时间复杂度是多项式级的

2.较容易理解，怀着贪婪的心，提出选择策略

3.正确性容易证明：即证明贪心算法可求出最优解

（四）个人理解

与动态规划又不一样的是：每一步的确定不依赖于子问题的解。

感觉跟 神经网络之类的计算智能算法（将指数式的复杂度降到多项式级）很像，都是通过模拟，近似这样的感觉，但又具有有效性，实用性。

## 二．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。

算法由一个简单的迭代过程构成，在维持可行性的前提下它选择能产生最大直接利益的项

## 三．实验目的和要求

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；

## 四．实验一内容

#### 加油问题（Problem Set 1702）：

* 1. **问题描述**
  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“NoSolution”。

* 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

**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)

## 五：实验一步骤分析

#### （一）初步设想，思考

已知量：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 两个城市距离 | 汽车油箱的容量 | 每升汽油能行驶的距离 | 沿途油站数 | 油站i离出发点的距离 | 油站i每升汽油的价格 |
| dis | C | d | n | d[i] | p[i] |

求：从城市A到城市B最少耗油量 设为 ocost

在后面过程中发现所需变量 ：

邮箱中油的剩余量：r

**初始条件：**

出发时邮箱是空的： 设 邮箱剩余量 r = 0

**特殊情况：**

加满油仍不能到达下一家油站： 即 dC < d[i+1]-[d[i]

**考虑出现情况：**

1.下一家油站汽油更加便宜： 即 p[i+1] < p[i]

只需加汽油刚好 到下一家油站的汽油量：

两个油站间距离： idis = d[i+1]-d[i]

加油量： r = r - (idis-dr)/d

所需邮费 ocost += r\*p[i]

2.下一家的油站更贵： 即 p[i+1] > p[i]

需要加满油量：

加油量 ：r = C-r

所需油费： ocost += r\*p[i]

3.因为考虑到每次到一个油站需要考虑下一家油站的油费，所以应该将油站 n+1

并设置 p[n+1] = 0 ;d[n+1] = 0

4.若汽车剩余油费可支撑汽车走过两个汽油站，那该怎样考虑？

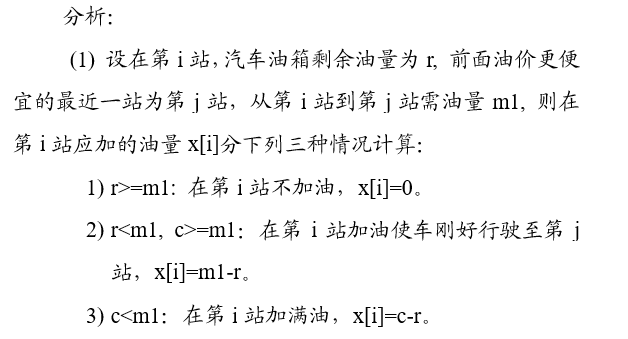
是否每次需要判断油量是否可支撑两个油站？

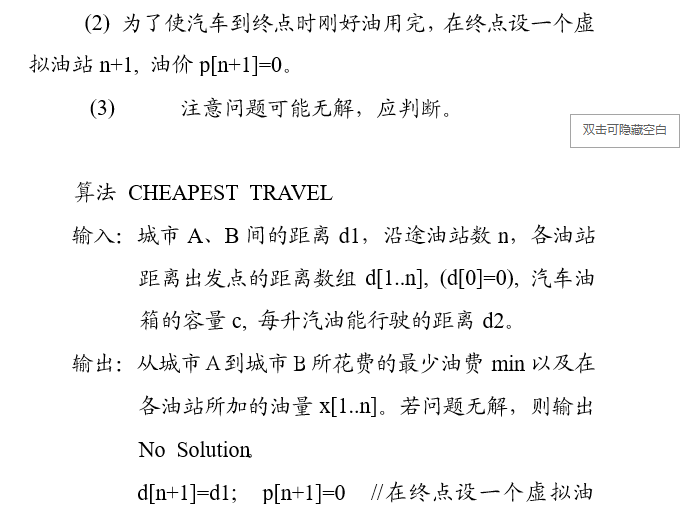
是否要考虑三家油站油费的大小？

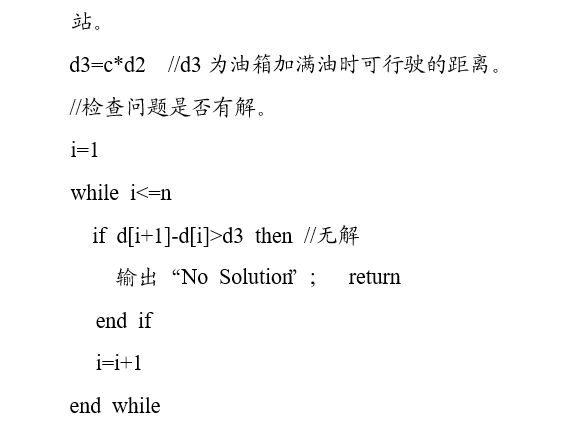
越想越复杂

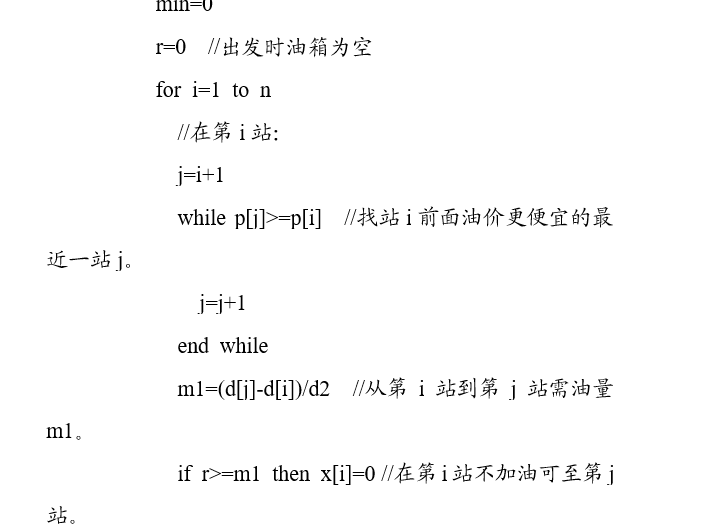
我的自我解决到此为止

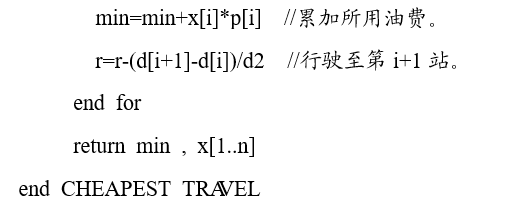
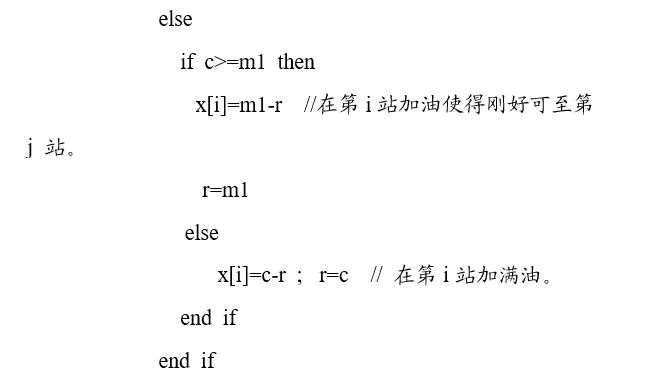
#### （二）看网上大佬思路











**（三）代码实现**

**3.1**

**3.2 Python 代码实现**

**import numpy as np**

**def deal(Dis,C,d,n,dis,p):**

**r = 0 #邮箱中油的剩余量**

**ocost = 0 #城市A到城市B最少耗油量所需费用**

**p.append(0)**

**dis.append(Dis)**

**for i in range(0,n):**

**idis = dis[i+1] - dis[i] # 两个油站间距离**

**if d\*C < idis:**

**return -1**

**j = i+1;**

**while p[j]>= p[i]:**

**j += 1;**

**idis = dis[j] - dis[i] #两油站之间的距离**

**if idis <= d\*r:**

**r = r -idis/d**

**else:**

**if d\*C >= idis: #邮箱油量刚好到J站**

**ocost += ((idis - d\*r)/d)\*p[i]**

**r = idis/d - (dis[i+1]-dis[i])/d #剩余量**

**else: #加满油**

**ocost += (C- r)\*p[i]**

**r = C - (dis[i+1]-dis[i])/d**

**return ocost**

**'''**

**#定义变量**

**Dis = 0 #两个城市距离**

**C = 0 #汽车油箱的容量**

**d = 0 #每升汽油能行驶的距离**

**n = 0 #沿途油站数**

**dis = [] #油站i离出发点的距离**

**p = [] #油站i每升汽油的价格**

**'''**

**count = int(input())**

**for i in range(0,count):**

**dis = [] #油站i离出发点的距离**

**p = [] #油站i每升汽油的价格**

**num = list()**

**Dis,C,d,n = map(int ,input().split(' '))**

**dis = list(map(float,input().split(' ')))**

**p = list(map(float,input().split(' ')))**

**num.append(deal(Dis,C,d,n,dis,p))**

**for i in range(0,count):**

**if num[i] == -1:**

**print("No solution")**

**else:**

**print("%.1f"%num[i])**

#### 证明其为最优解

证明：

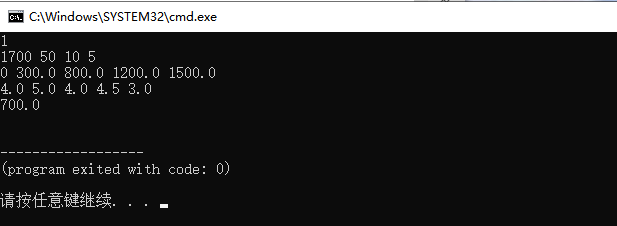
当没有油站的时候，即n=0,假设Dis= 100，C= 50，d = 10

可知该算法成立。

假设对任意自然数 n 都成立：

对 n =5；采用题目所给数据得出 640.0

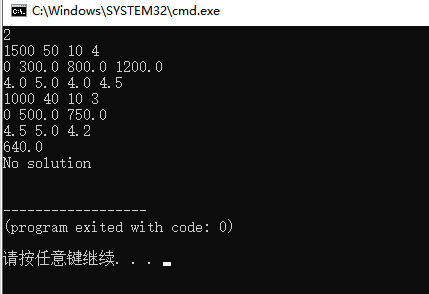
则n+1 假设 Dis = 1700,C = 50,d = 10,n =6 如图



通过手算，和程序运行可知，对n+1也成立

所以该贪心算法思路没有问题

#### （五）实验结果展示



#### （六）复杂度分析

#### 根据deal()函数可知，时间复杂度最小为O(n),最差（即最少的油价是倒数第一个油站）为O(n\*n)，平均时间复杂度为O(n\*n).

因为只需几个变量储存空间，所以空间复杂度为O(n)

#### （七）问题一总结

这个问题，刚开始没带电脑，就对着手机上的实验进行思路分析，刚开始觉得很像是最短路径问题，但不一样的是：条件限制不一样，后面越发觉得不是一种类型。这道题最难的部分是每到一个油站，怎样判断加多少油的解决方案。

最终这个贪心，贪的是：每到达一站i，找到前面油价更便宜的最近的油站j，在控制所加的油量至多只能行驶到j站的前提下，尽可能多加油。

针对我之前的问题，大佬直接判断出找出最近的更便宜的一油站，这一点我在犹豫，刚开始没有想到可以这样判断。

## 六：实验二内容

#### 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。

* 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

* 1. **扩展内容**

(1) 建议采用可视化界面

## 七.实验二步骤分析

#### （一）初步设想，思考

已知量：

|  |  |
| --- | --- |
| 黑点 | 白点 |
| 坐标（xb,yb） | 坐标（xw,yw） |

已知：黑点 匹配 白点 条件：

xb >= xw && yb >= yw

求 n个白点与黑点的最大匹配

初始条件：

已知所有黑点，白点的准确坐标

贪心想法：

有题意可知：即需要 每个白点从坐标系中右上角 四分之一的位置里 查询离她最近的黑点 作为匹配对象 应该最为贪心

考虑做法：

1.对每个白点 ，先查询出符合条件（xb >= xw && yb >= yw）的所有黑点

求出白点与所有对象之间的距离，取最小的

2.在1.的基础上，还需要考虑两个方向上的距离：

因为存在 该白点取了一个其他某个白点唯一的取的黑点

所以还要考虑方向，引入 P(以该白点为原点，x轴为出发点，判断黑点与白点的角度 Cover)

从x方向开始旋转，考虑黑点

即：找 距离又近，

好像有问题，截断，考虑另一种

3.

3.1先对所有白点xw 进行排序（降序）

在此基础上，对所有xw相等的白点的 yw 进行降序排列

最终可得到 一个类似 y = x线性相似度的列表

3.2 从第一个白点开始：搜索所有满足条件xb >= xw && yb >= yw的黑点，选距离最近的一个

这样好像没问题了。

因为若对最远的白点都满足的黑点，对其余白点也满足，不会出现交叉的情况

#### （二）看网上大佬思路

#### 代码实现

#include<stdio.h>

#include<iostream>

#include<math.h>

#include<iomanip>

#include<time.h>

#include<string.h>

#include<cctype>

using namespace std;

#define PI acos(-1)

#define demax 300

#define demin 1e-6

struct point{

float x;

float y;

int tag;

}w[demax],b[demax];

int match(point w[],point b[],int n)

{

int i,j,minflag,minp,count;

count = 0;

float miny;

for(i = n-1;i>= 0 ;i--)

{

minflag =0;

minp = 0;

miny = 10000;

for(j = n-1;j>= 0;j--)

{

if(b[j].tag)

continue;

if(b[j].x <w[i].x)

continue;

if(b[j].y >= w[i].y)

{

minflag = 1;

if(miny > b[j].y)

{

miny = b[j].y;

minp = j;

}

}

}

if(minflag)

{

count++;

b[minp].tag = 1;

}

}

return count;

}

void sort(point w[],int n)

{

int j,k,h;

point t;

for(h = n-1;h>0 ;h = k)

{

for(j =0,k=0;j<h;j++)

{

if(w[j].x > w[j+1].x)

{

t = w[j];

w[j] = w[j+1];

w[j+1] = t;

k = j;

}

}

}

}

int main()

{

int k,n,i,count[k],m;

cin>>k;

m = k;

while(m--)

{

cin>>n;

for(i =0;i<n;i++)

cin>>b[i].x>>b[i].y;

for(i=0;i<n;i++)

cin>>w[i].x>>w[i].y;

for(i =0;i<n;i++)

w[i].tag = b[i].tag = 0;

sort(w,n);

count[k-m] = match(w,b,n);

cout<<count[k - m];

}

return 0;

}

#### 证明其为最优解

证明：

当黑点的时候，即ba;ck[] = 空

可知该算法成立。

假设对任意自然数 n 都成立：

对 n =3；采用题目所给数据得出 3

则n+1，

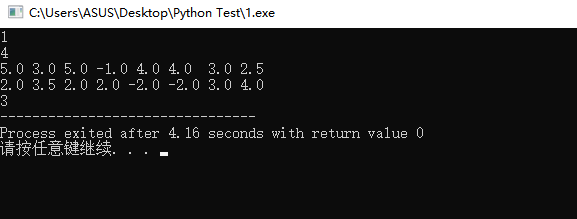
1

4

5.0 3.0 5.0 -1.0 4.0 4.0 3.0 2.5

2.0 3.5 2.0 2.0 -2.0 -2.0 3.0 4.0

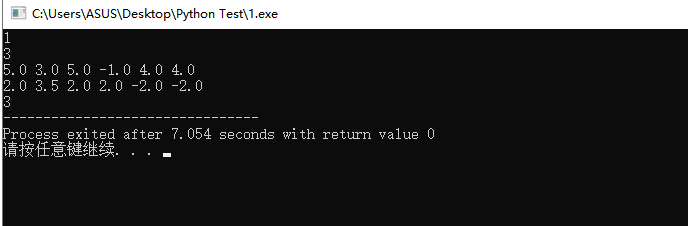
如图



经过手算可知，对 n+1也成立

所以该贪心算法思路没有问题

#### （五）实验结果展示



#### （六）复杂度分析

#### （七）问题一总结