

**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** | **148** | **年级专业班级** |  |
| **小组成员** | **无** | | | **实验日期** | **2019年5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：windows10操作系统

2.编译环境：CodeBlock（使用C++11）

## 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

**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

# 2. 实验记录

## 2.1 贪心算法的介绍

### 2.1.1贪心算法概念及理解

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） 。

**下面谈谈我自己对贪心算法的理解：**

贪心算法是我在低年级程序竞赛第一次接触到的算法，虽然很简单，但是很好用！贪心算法顾名思义，就是每次我都在使用最优步骤，以达到最终最优解的情况。但是！要注意，**贪心并不一定能从局部最优发展到总体最优**，最简单的例子就是01背包问题，假如我们按照性价比最高去选择最终并不一定会达到最优解（曾经第一次遇到01背包用贪心做错到自闭），这在课上已经讲过了。但是我改一下这个题面，就能用贪心求解了：每样商品都可以任意切割放到背包。显然这样局部最优最终会得到总体最优，直接贪心性价比。所以贪心的适用范围其实是有限的，用之前要谨慎考察部分最优和总体最优的关系。

### 2.1.2贪心算法实现方法和步骤

贪心算法的实现很简单，因为没有固定的模式，我们在使用时可以简化为两步：1.制定贪心策略；2.证明贪心的正确性。

首先，制定贪心策略就是制定你要贪心的步骤。比如在上面装背包问题，我是根据价值贪心，或是根据体积贪心，或是根据性价比贪心，这都是不同的贪心策略。证明贪心正确性是贪心方法能不能对的前提，当然，证明通常不会很简单，特别是对于某些情况很多的问题，例如在01背包问题上，乍一看好像性价比贪心是正确的，但是经过举证我们可以发现并不对。所以我通常试图举一些奇怪的反例，来检验自己的贪心，但这都是不严谨的方法。证明通常都是用数学归纳法，数学归纳法的证明过程详情百度。

## 2.2 实验过程

### 2.2.1加油问题的实验过程

1.实验思路：

首先，我们先制定贪心策略。我们要用最少的钱开到目的地，那么就要想办法在油价便宜的地方多加油，在油价贵的地方少加油。也就是说，我们在一个区间内相对便宜的地方，我们就尽可能加满油箱，在贵的地方我们只要保证油量足够到达便宜的地方就够了。

那么现在我们已经得到贪心策略了：我们从一个点开始往后遍历，找到第一个比当前油价便宜的地方，如果存在这样的点且能到达，那么我们把油加到刚好能到那个地方就行；如果不存在，我就把油加满，去下一个加油站。以上每一步都要先判断能不能到达下一个加油站，不能输出直接跳出程序并输出No solution。

那么我来**口头证明**一下我这个策略的正确性：

假如我现在在 i点，有一个j点满足“j是i后第一个油价比i便宜的点”，假设我的最优策略是选择在[i + 1，j – 1]的任意地点加油到达j，因为[i + 1，j – 1]的任意地点的油价都大于i的油价，显然此时不是最少花费，所以最优策略是在i加油加到刚好到j。

假如不存在这个j点，假设i点最远能到k，显然我能得到[i + 1，k]的任意地点油价都比i贵，那么我在这个区间内任意地点加油都比i贵，显然不是最小花费。以上，可以得出我贪心策略是正确的。

为了能到达目的地，应该设置目的地的油价最便宜。

2.程序代码：

**#include<set>**

**#include<map>**

**#include<stack>**

**#include<cmath>**

**#include<queue>**

**#include<vector>**

**#include<string>**

**#include<cstdio>**

**#include<cstring>**

**#include<sstream>**

**#include<iostream>**

**#include<algorithm>**

**typedef long longll;**

**using namespace std;**

**const int maxn = 5000 + 10;**

**const int MOD = 1e9 + 7;**

**const int INF = 0x3f3f3f3f;**

**double d[maxn], price[maxn];**

**//距离d，价格price**

**int main(){**

**int t;**

**scanf("%d", &t);**

**while(t--){**

**double D, c, per;**

**int n;**

**scanf("%lf%lf%lf%d", &D, &c, &per, &n);**

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

**scanf("%lf", &d[i]);**

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

**scanf("%lf", &price[i]);**

**d[++n] = D, price[n] = 0; //目的地设为加油站**

**int now = 1;**

**double cost = 0, oil = 0;**

**bool fail = false; //能否成功**

**while(now < n){**

**if(c \* per < d[now + 1] - d[now]){**

**//到不了下一个加油站**

**fail = true;**

**break;**

**}**

**int cheap = now + 1;**

**while(price[cheap] >= price[now] && cheap <= n)**

**cheap++; //找到第一家便宜的**

**if(cheap <= n && d[cheap] - d[now] <= c \* per){**

**//找到比当前便宜的加油站cheap**

**//且能到达cheap**

**if(oil < (d[cheap] - d[now]) / per){//油不够**

**cost += ((d[cheap] - d[now]) / per - oil) \* price[now];**

**oil = 0;**

**now = cheap;**

**}**

**else{**

**oil -= (d[cheap] - d[now]) / per;**

**now = cheap;**

**}**

**}**

**else{**

**//到不了cheap或者没有cheap**

**//在这里加满**

**cost += (c - oil) \* price[now];**

**oil = c - (d[now + 1] - d[now]) / per;**

**now = now + 1;**

**}**

**}**

**if(!fail) printf("%.1lf\n", cost);**

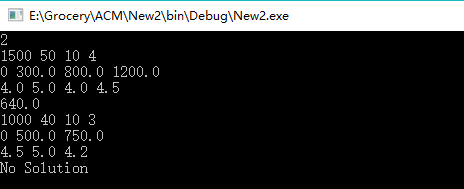
**else printf("No Solution\n");**

**}**

**return 0;**

**}**

3.实现结果：



实验结果正确无误。

### 2.2.2黑白点的匹配

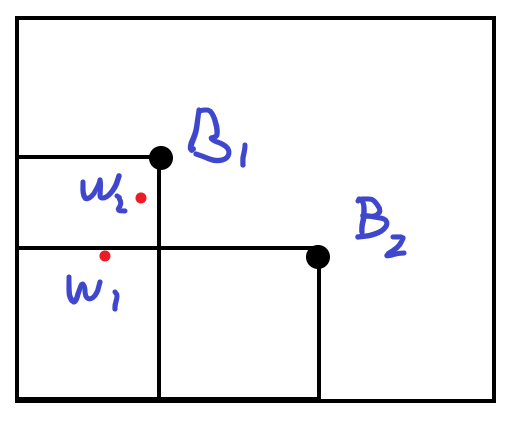
1.实验思路：

这道题被网上的题解搞了。我一开始根本没想到贪心怎么写，然后就去看网上的题解了。然后写出来的代码是这样写的：<https://pasteme.cn/7771>。

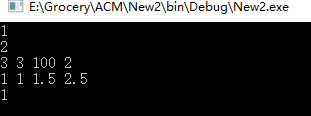
首先我制定出了**错误的**贪心策略：如果一个黑点要和一个白点匹配，怎么做才能最划算呢？在所有能匹配的点中找一个和自己差距最小的点匹配，那么我留下来的点和其他的点能和别人能匹配的可能性越高。所以我大致确定了一个贪心策略：先把白点按x升序排序，因为x越大能选择的点越多，所以要让x小的先选。然后我们遍历所有黑点，找出没被选过、符合条件且和当前白点距离最小的，那么这个点就是当前白点的最优选择。

但是我写完后又想了一下发现事情没那么简单。

我随手举了一个例子：



假设我现在有两个白点w1和w2满足w1x < w2x（如上图），那么我现在先匹配w1。显然按照距离最小原则，我w1选择的是B1。然而，我们能看出最好的选择其实是w1匹配B2，w2匹配B1。显然，这个算法无情淘汰。Pass。



（这代码果然错了）

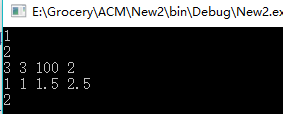
其实网上找的那些贪心算法基本上都是错的，感觉就是来坑害别人的……这就很尴尬了。网上的贪心版本有好几种，我也没全都看，可以自己证一下，证不出来估计就错了。

那么到底怎么做啊？其实这道题我一开始就想到用什么算法了，但是感觉不是贪心算法就没写，现在想一下好像也是贪心的应用吧。就像最短路的Dijkstra和SPFA，好像乍一看是图论的算法，但其实暗含贪心的思想。嗯，一定是这样，反正也想不出别的了。

这道题我的解法是用**二分图跑最大匹配**。

基本概念：二分图就是一个图能分成两个不相交的集合，显然这里的集合是白点和黑点。最大匹配是给一个条件使两个点集W集合和B集合有关联，然后能得到两个集合的最大二元组（Wi，Bi）的数量。显然和这道题完美应用。最大匹配的算法主要的原理其实就是匈牙利算法。匈牙利算法的主要原理是：我们按照一定顺序先逐一给集合W匹配一个集合B的点，如果出现Wi能匹配Bj但是Bj已经在之前被匹配过了，那么我们先假设Wi和Bj匹配，通过DFS让和Bj匹配的点Wk尝试和其他点匹配，如果可行，那么就得到了新的匹配。显然，这里贪的是匹配数量。

然后我只要直接把题目给的二维坐标映射成一个点的一维点坐标（怎么映射的可以看我代码，挺简单的），这样黑白点就变成两个点集了。然后跑二分匹配，完美解决。不得不吐槽word代码排版真丑。



这个答案就对了，应该没有问题。

2.程序代码（我把好看的排版贴网上了<https://pasteme.cn/7772>）：

**#include<cmath>**

**#include<set>**

**#include<map>**

**#include<queue>**

**#include<cstdio>**

**#include<vector>**

**#include<cstring>**

**#include <iostream>**

**#include<algorithm>**

**using namespace std;**

**typedef long long ll;**

**const int maxn = 20 + 10;**

**const int INF = 0x3f3f3f3f;**

**const int MOD = 1e9 + 7;**

**struct Edge{**

**int v, next;**

**}edge[maxn \* maxn \* 2];**

**int head[maxn], match[maxn], vis[maxn], tot;**

**void addEdge(int u, int v){ //建边**

**edge[tot].v = v;**

**edge[tot].next = head[u];**

**head[u] = tot++;**

**}**

**bool dfs(int u){**

**for(int i = head[u]; i != -1; i = edge[i].next){**

**int v = edge[i].v;**

**if(!vis[v]){**

**vis[v] = 1;**

**if(match[v] == -1 || dfs(match[v])){**

**match[v] = u;**

**match[u] = v;**

**return true;**

**}**

**}**

**}**

**return false;**

**}**

**map<pair<double, double>, int> id;**

**//map用来映射点**

**int solve(int n){**

**int ans = 0;**

**memset(match, -1, sizeof(match));**

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

**if(match[i] == -1){**

**memset(vis, 0, sizeof(vis));**

**if(dfs(i)) ans++;**

**}**

**}**

**return ans;**

**}**

**struct node{**

**double x, y;**

**}w[maxn], b[maxn];**

**int main(){**

**int t, n;**

**scanf("%d", &t);**

**while(t--){**

**int cnt = 0;**

**id.clear();**

**scanf("%d", &n);**

**memset(head, -1, sizeof(head));**

**tot = 0;**

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

**scanf("%lf%lf", &b[i].x ,&b[i].y);**

**id[make\_pair(b[i].x, b[i].y)] = ++cnt;**

**//x y映射成cnt**

**}**

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

**scanf("%lf%lf", &w[i].x ,&w[i].y);**

**id[make\_pair(w[i].x, w[i].y)] = ++cnt;**

**//x y映射成cnt**

**}**

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

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

**if(b[i].x >= w[j].x && b[i].y >= w[j].y){**

**addEdge(id[make\_pair(b[i].x, b[i].y)],**

**id[make\_pair(b[i].x, b[i].y)]);**

**}**

**}**

**}**

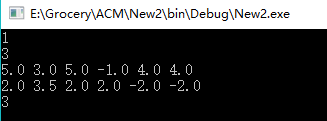
**printf("%d\n", solve(cnt));**

**}**

**return 0;**

**}**

3.实现结果：



解决。

# 3. 实验总结

原本以为贪心很简单，但好像也不简单。贪心算法的贪心策略可以随自己想，但是关键是还要证明贪心的正确性，这谁顶得住啊。第一题的证明还好，第二题根本就不知道怎么证，看了网上的题解用x，y坐标排序后贪心最小距离做的，结果自己一测就是错的。后来想了一下，到底是改进网上的贪心还是直接上匈牙利，因为不会直接贪心，只能借助匈牙利找增广路，但是用二分图的方法得出的结果肯定是正确的。至于匈牙利是不是贪心……因为匈牙利每次都是尽可能让每一个匹配都成立，用dfs验证成立与否，个人感觉还是有贪心的思维在里面的。

贪心应该算是一种思想，不能算是一种具体的算法。所以在做题的时候，瞎贪心就很容易错，因为没有具体的套路，也没有像Dijkstra一样有原理可以找，只能凭感觉先想一个贪心的策略，然后不断证明。然而通常情况都是还没证明出来就默认是贪对了或者根本不会证明，只有冥冥中有种力量告诉你这是对的。