## poj 3278

### 1 原题目

**赶上那头牛**

|  |  |  |
| --- | --- | --- |
| **时限：** 2000MS |  | **内存限制：** 65536K |
| **提交总数：** 149574 |  | **接受的：** 45849 |

**描述** <http://poj.org/problem?id=3278>

农夫约翰已被告知一头逃犯的位置，并希望立即抓住她。他开始于一个点*Ñ*（0≤ *Ñ* ≤100,000）上的数线和牛是在点*ķ*（0≤ *ķ*上相同数目的线≤100,000）。农夫约翰有两种运输方式：步行和传送。

\*步行：FJ可以在一分钟内从任意点*X*移至点*X* -1或*X* + 1。  
\*传送：FJ可以在一分钟内从任意点*X*移至点2× *X*。

如果没有意识到它的追捕能力的母牛完全没有动弹，那么农夫约翰要花多长时间？

**输入项**

第1行：两个以空格分隔的整数：*N*和*K*

**输出量**

第1行：农夫约翰用最少的时间（以分钟为单位）捉住逃犯。

**样本输入**

5 17

**样本输出**

4

**暗示**

农夫约翰到达逃亡者牛的最快方法是沿着以下路径移动：5-10-9-18-17，这需要4分钟。

### 2 题目分析

农夫只能往前走一步，或往后走一步，或者直接到达他现在位置二倍的地方，让农夫走最少的步数来到达牛的位置。

### 3 基础知识

#### 3.1数据结构

无。

#### 3.2算法

广度优先搜索（解法一用到）

### 4 解法

#### 4.1 解法1（java已过）

利用广度优先搜索来解题，并利用标记数组来去重。

因为是一条直线，所以我们可以将农夫走过的位置进行标记，如果下次又到了标记的位置就不走这个位置。

借助队列，先将农夫的位置存进队列，然后将农夫在当前位置可以到达的所有位置存进队列中，并用-1分隔开农夫每步走到的位置，然后依次弹出队列的元素，重复操作直到走到了牛的位置。

本人代码：

**import** java.util.\*;//包含java.util包中所有的类

**public** **class** Main {//用广度优先搜索来做

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

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

**while**(in.hasNext()) {//输入多组数据

**int** map[] = **new** **int**[100001];//标记数组

Queue<Integer> queue = **new** LinkedList<Integer>();//队列

**int** n,k,num = 0;

n = in.nextInt();

k = in.nextInt();

queue.add(n);//压入队列

map[n] = 1;

queue.add(-1);//-1是用来将每一步可以到达的点隔开

**while**(!queue.isEmpty()) {

**int** temp = queue.poll();//从队列弹出

**if**(temp == -1) {

num++;//步数加1

queue.add(-1);

**continue**;

}

**else** {

**if**(temp == k)//等于k时说明找到了

**break**;

//存入当前点所有可能走过的点并进行去重

**if**(temp - 1 >= 0 && map[temp - 1] == 0) {

queue.add(temp - 1);

map[temp - 1] = 1;

}

**if**(temp + 1 <= k && map[temp + 1] == 0) {

queue.add(temp + 1);

map[temp + 1] = 1;

}

**if**(temp \* 2 <= 100000 && map[temp \* 2] == 0) {

queue.add(temp \* 2);

map[temp \* 2] = 1;

}

}

}

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

}

}

}

## poj 3126

### 1 原题目

**主要路径**

|  |  |  |
| --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 65536K |
| **提交总数：** 36147 |  | **接受：** 19404 |

**描述** <http://poj.org/problem?id=3126>

****内阁大臣对安全首长的信息感到非常不安，他们说他们都必须更改办公室四位数的房间号。  
—时不时地改变这些事情，以使敌人陷入黑暗是安全的问题。  
-但是，我有充分的理由选择了我的号码1033。我是总理，你知道的！  
—我知道，因此您的新电话号码8179也是素数。您只需要在办公室门上的四个旧数字上粘贴四个新数字。  
—不，不是那么简单。假设我将第一个数字更改为8，那么该数字将显示8033，这不是质数！  
—我知道，作为总理，即使坐了几秒钟，也无法忍受门口有非总理数字。  
—正确！因此，我必须发明一种方案，通过从质数到1033到8179的质数路径，其中只有一位从一个质数变为下一个质数。  
  
现在，一直在窃听的财政部长介入了。  
—请不要不必要的支出！我碰巧知道一个数字的价格是一磅。  
—嗯，在那种情况下，我需要一个计算机程序以最小化成本。您不了解一些非常便宜的软件专家，对吗？  
—事实上，我愿意。您会看到，这场编程竞赛正在进行中...帮助总理找到在两个给定的四位数素数之间最便宜的素数路径！当然，第一位必须为非零。这是上述情况的解决方案。

1033  
1733  
3733  
3739  
3779  
8779  
8179

该解决方案的成本为6磅。请注意，在步骤2中粘贴的数字1不能在最后一步中重复使用-必须购买新的1。

**输入项**

一行带有正数：测试用例的数量（最多100个）。然后，对于每个测试用例，用两个数字用空格分隔的一行。这两个数字都是四位数的质数（无前导零）。

**输出量**

每种情况只用一行，或者用数字表示最低费用，或包含“不可能”一词。

**样本输入**

3

1033 8179

1373 8017

1033 1033

**样本输出**

6

7

0

### 2 题目分析

给定两个素数a b，求a变幻到b需要几步，并且变幻时只有一个数字不同，并且是素数。

### 3 基础知识

#### 3.1数据结构

队列（解法一用到）

#### 3.2算法

广度优先搜索（解法一用到）

### 4 解法

#### 4.1 解法1（C++已过）

广度优先搜索，将a素数每一位依次用0到9替换，其他位暂且不变（要满足替换后的数字是素数这个条件）存进队列中，然后对队列中的数字进行同样的操作，直到找到素数b或者队列为空。

本人代码：

#include <iostream>

#include <string.h>

#include <queue>

using namespace std;

int prime[10005];//素数表

int Count[10005];//记录到当前素数的费用

int vis[10005];//标记素数是否已经走过

void Init()//对素数打表

{

int i,j;

for(i = 1000; i <= 10005; i++)

{

for(j = 2; j < i; j++)

if(i%j == 0)

{

prime[i] = 0;//不是素数置0

break;

}

if(j == i)

prime[i]=1;//是素数置1

}

}

int dfs(int st,int last)

{

queue<int>q;//初始化一个队列

memset(vis,0,sizeof(vis));//数组初始化

memset(Count,0,sizeof(Count));//数组初始化

int t[4] = {0},i,j,cur,temp,ntemp;

q.push(st);

vis[st] = 1;

while(!q.empty())

{

cur = q.front();//cur记录队头元素

if(cur == last)//找到最后的数字

return Count[cur];

q.pop();

t[0]=cur/1000;//将四位数cur拆开存储

t[1]=cur%1000/100;

t[2]=cur%100/10;

t[3]=cur%10;

for(i = 0; i < 4; i++)

{

temp = t[i];

for(j = 0; j < 10; j++)//cur的每一位的去尝试所有可能的数字

{

if(j != temp)//当前数字和当前位的数字不同时

{

t[i] = j;//取代当前位的数字

ntemp = t[0] \* 1000 + t[1] \* 100 + t[2] \* 10 + t[3];//计算新的数字

if(vis[ntemp] == 0 && prime[ntemp] == 1)//数字没走过且是素数

{

vis[ntemp] = 1;//标记

q.push(ntemp);//入队

Count[ntemp] = Count[cur] + 1;//计算费用

}

}

}

t[i] = temp;//位的数字复原

}

}

return -1;

}

int main()

{

memset(prime,0,sizeof(prime));//数组初始化

Init();

int n;

cin>>n;

while(n--)

{

int st,last;

cin>>st>>last;

int m = dfs(st,last);

if(m == -1)

cout<<"Impossible"<<endl;

else

cout<<m<<endl;

}

return 0;

}

## poj 2251

### 1 原题目

**地牢大师**

|  |  |  |
| --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 65536K |
| **提交总数：** 64938 |  | **接受：** 23649 |

**描述** <http://poj.org/problem?id=2251>

您被困在3D地牢中，需要找到最快的出路！地牢由可能填充或未填充岩石的单位立方体组成。向北，向南，向东，向西，向上或向下移动一个单元需要一分钟。您不能沿对角线移动，迷宫四面都是坚硬的岩石。  
  
有可能逃脱吗？如果是，需要多长时间？

**输入项**

输入包含多个地牢。每个地牢描述都以包含三个整数L，R和C（大小均限制为30）的行开头。  
L是组成地牢的层数。  
R和C是组成每个级别计划的行数和列数。  
然后将出现R行的L个块，每个块包含C个字符。每个角色描述一个地牢单元。充满岩石的单元格用“＃”表示，空单元格用“。”表示。您的起始位置以“ S”表示，退出位置以字母“ E”表示。每个级别之后只有一个空白行。输入以L，R和C的三个零终止。

**输出量**

每个迷宫产生一行输出。如果有可能到达出口，请打印表格的一行

在x分钟内转义。

其中x替换为最短逃生时间。  
如果无法逃脱，请打印该行

被困！

**样本输入**

3 4 5

S....

.###.

.##..

###.#

#####

#####

##.##

##...

#####

#####

#.###

####E

1 3 3

S##

#E#

###

0 0 0

**样本输出**

Escaped in 11 minute(s).

Trapped!

### 2 题目分析

给你一个三维数组构成的地牢（包含墙壁），并且给你一个起点和终点，让你判断由起点能否走到终点，如果可以请找出最短的路径。

### 3 基础知识

#### 3.1数据结构

队列（解法一用到）

#### 3.2算法

广度优先搜索（解法一用到）

### 4 解法

#### 4.1 解法1（C++已过）

最短路径的问题，直接广度优先搜索来做（要用到队列），具体请看代码和注释。

本人代码：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int dir[6][3]= {{0,0,-1},{0,0,1},{-1,0,0},{1,0,0},{0,-1,0},{0,1,0}};//下一步怎么走（前二是列数变，中二是层数变，后二是行数变）

char map[35][35][35];//存地图

int vis[35][35][35];//标记地图中的点是否被走过

int flag,x,y,z;//x，y，z代表层数，行数，列数

struct node//队列

{

int x,y,z;//x，y，z代表层数，行数，列数

int num;//num代表步数

} que[30010];

void bfs(int sx,int sy,int sz)

{

int head=0;//队列的头指针

int tail=1;//队列的尾指针

que[0].x=sx;

que[0].y=sy;

que[0].z=sz;

que[0].num=0;

vis[sx][sy][sz]=1;

int xx,yy,zz;

while(head<tail)//当对列不为空时

{

for(int i=0; i<6; i++)//下一步如何走（6种情况）

{

xx=que[head].x+dir[i][0];

yy=que[head].y+dir[i][1];

zz=que[head].z+dir[i][2];

if(map[xx][yy][zz]=='E') //搜到终点

{

printf("Escaped in %d minute(s).\n",que[head].num+1);

flag=1;

break;

}

if(xx < x && yy < y && zz < z && xx >= 0 && yy >= 0 && zz >= 0 && vis[xx][yy][zz] == 0 && map[xx][yy][zz] == '.') //如果在迷宫内，并且未走过

{

que[tail].x=xx;

que[tail].y=yy;

que[tail].z=zz;

que[tail].num=que[head].num+1;

vis[xx][yy][zz]=1;

tail++;

}

}

if(flag == 1)//flag为1说明已经找到了终点

break ;

head++;//flag不为1，那就要在继续寻找

}

if(flag == 0)

printf("Trapped!\n");

}

int main()

{

while(scanf("%d %d %d",&x,&y,&z) != EOF)

{

getchar();//吸收回车

if(x == 0 && y == 0 && z == 0)

break;

memset(vis,0,sizeof(vis));//标记数组置零

flag = 0;

int i,j,k,sx,sy,sz;

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

for(j = 0; j < y; j++)

{

for(k = 0; k < z; k++)

{

scanf("%c",&map[i][j][k]);//读入地图

if(map[i][j][k] == 'S')//找到起点的位置并记录下来

{

sx = i;

sy = j;

sz = k;

}

}

getchar();//吸收回车

if(j == y - 1 && i != x - 1)

getchar();//吸收空行（也就是回车）

}

bfs(sx,sy,sz);//广度优先搜索

}

return 0;

}

## poj 1753

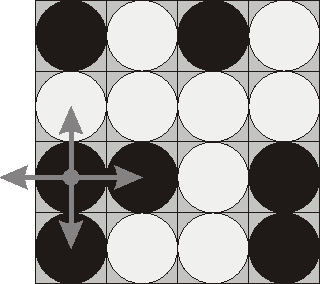
### 1 原题目

**翻转游戏**

|  |  |  |
| --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 65536K |
| **提交总数：** 60513 |  | **接受：** 25128 |

**描述** <http://poj.org/problem?id=1753>

翻转游戏是在一个矩形的4x4场上进行的，在其16个正方形的每个正方形上放置了两个棋子。每一块的一侧为白色，另一侧为黑色，并且每块都以黑色或白色的一面朝上。每轮翻转3到5片，从而将其上侧的颜色从黑色更改为白色，反之亦然。根据以下规则在每一轮中选择要翻转的棋子：

1. 选择16件中的任何一件。
2. 将选定的片段以及所有相邻片段翻转到选定片段的左侧，右侧，顶部和底部（如果有）。  
   以下面的位置为例：  
   bwbw  
   wwww  
   bbwb  
   bwwb  
   此处的“ b”表示其黑色面朝上的部分，“ w”表示其白色面朝上的部分。如果我们选择从第三行翻转第一个棋子（此选择如图所示），那么该字段将变为：  
   bwbw  
   bwww  
   wwwb  
   wwwb  
   游戏的目标是将所有棋子翻转白色或黑色朝上。您将编写一个程序，以搜索实现该目标所需的最小轮数。

**输入项**

输入包括4行，每行分别包含4个字符“ w”或“ b”，表示游戏场的位置。

**输出量**

向输出文件中写入一个整数-从给定位置达到游戏目标所需的最少回合数。如果最初达到目标，则写0。如果不可能达到目标，则写“不可能”（不带引号）。

**样本输入**

bwwb

bbwb

bwwb

bwww

**样本输出**

4

### 2 基础知识

枚举（解法一用到）

深度优先搜索（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

题目要你返回翻转棋子数量使满足全黑或全白，我们可以枚举出所有的情况，比如翻转一个棋子的时候有16种，翻转2个棋子有120种，…… ，翻转15个棋子有2种，最后翻转16个棋子有1种，我们枚举的过程利用深度优先来实现，具体看代码和注释。

本人代码：

|  |
| --- |
| #include <iostream>  #include<algorithm>  #include <cstring>  #include <cstdio>  #include <cstdlib>  #include <cmath>  using namespace std;  char s[10];  int mapp[20];//存储16个棋子的状态（1到16）  int flag;//标记是否找出符合条件的轮数  int pan()//判断是否全黑或全白  {      int x = mapp[1];      for(int i = 2; i <= 16; i++)          if(mapp[i] != x)              return 0;      return 1;  }  void fan(int x)//翻转棋子颜色  {      mapp[x] = !mapp[x];      if(x + 4 <= 16)          mapp[x + 4] = !mapp[x + 4];      if(x - 4 >= 1)          mapp[x - 4] = !mapp[x - 4];      if(x % 4 == 0)          mapp[x - 1] = !mapp[x - 1];      else if(x % 4 == 1)          mapp[x + 1] = !mapp[x + 1];      else      {          mapp[x - 1] = !mapp[x - 1];          mapp[x + 1] = !mapp[x + 1];      }  }  //x是下标，num是剩余翻转棋子数  void dfs(int x,int num)//利用dfs进行枚举  {      if(num == 0)//可翻转棋子数为0      {          if(pan())//如果符合条件              flag = 1;//标记          return ;      }      int i;      for(i = x + 1; i <= 16; i++)      {          if(i + num > 17)//深搜的剪枝（剩下的棋子数不够你翻转的，用前面的就会有重复的情况）              return ;          fan(i);//翻转          dfs(i,num - 1);          fan(i);//状态还原          if(flag)//找到满足的就没必要继续深搜了              return ;      }      return ;  }  int main()  {      int i,j;      flag = 0;      int bit = 1;      for (i = 0; i < 4; i++)//读入数据并存储      {          scanf("%s",s);          for(j = 0; j < 4; j++)          {              if(s[j] == 'b')                  mapp[bit++] = 0;              else                  mapp[bit++] = 1;          }      }      if(pan())//步骤为0先判断下          printf("0\n");      else      {          for(i = 1; i <= 16; i++)//1到16为可翻转棋子数（利用深搜枚举出翻转的所有组合）          {              dfs(0,i);//深度优先搜索              if(flag)//如果符合条件，不用继续了，直接跳出                  break;          }          if(!flag)//flag为0说明没有符合条件的翻转棋子数              printf("Impossible\n");          else              printf("%d\n",i);      }      return 0;  } |

## poj 1426

### 1 原题目

**寻找多重**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 10000K | | |
| **提交总数：** 52790 |  | **接受：** 22017 |  | **特别法官** |

**描述** <http://poj.org/problem?id=1426>

给定正整数n，编写一个程序以找出n的非零倍数m，其十进制表示形式仅包含数字0和1。您可以假定n不大于200，并且相应的m包含不超过100十进制数字。

**输入项**

输入文件可能包含多个测试用例。每行包含一个值n（1 <= n <= 200）。包含零的行将终止输入。

**输出量**

对于输入中n的每个值，打印一行包含m对应值的行。m的十进制表示形式不能超过100个数字。如果给定值n有多个解，则其中任何一个都是可接受的。

**样本输入**

2

6

19

0

**样本输出**

10

100100100100100100

111111111111111111

### 2 题目分析

找到n的整数倍的一个数字m（m只有0和1组成，且m的位数小于100）

### 3 基础知识

#### 3.1数据结构

无。

#### 3.2算法

深度优先搜索（解法一用到）

### 4 解法

#### 4.1 解法1（java已过）

m数字只能由0和1组成，所以我们可以利用10和11这两个数字进行深搜（在这两个数字后加0加1），所以有m \* 10 和 m \* 10 + 1 这两个式子

参考链接：<https://blog.csdn.net/qq_42964711/article/details/81938661>

本人代码：

**import** java.util.\*;//包含java.util包中所有的类

**public** **class** Main {//用深度优先搜索来做

**static** **int** *t*;//全局变量

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

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

**while**(in.hasNext()) {///多次输入

**int** n;

n = in.nextInt();

**if**(n == 0)

**break**;

Main a = **new** Main();

*t* = 0;//t用来判断是否已经找到了一个正确的解

**int** step = 1;//step记录数字m的位数

a.dfs(n,1,step);//调用dfs函数

}

}

**public** **void** dfs(**int** n,**long** m,**int** step) {

**if**(*t* == 1 || step > 19)//java中long型最大数字的位数为19位

**return** ;

**if**(m % n == 0) {//判断m是否为n的整数倍

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

*t* = 1;//找到正确的解，t标记为1

}

//尝试所有的只含01的十进制数字

dfs(n,m \* 10,step + 1);

dfs(n,m \* 10 + 1,step + 1);

}

}

## poj 3279

### 1 原题目

**翻板**

|  |  |  |
| --- | --- | --- |
| **时限：** 2000MS |  | **内存限制：** 65536K |
| **提交总数：** 28061 |  | **接受：** 9902 |

**描述 http://poj.org/problem?id=3279**

农夫约翰知道，一头知识上满意的母牛是一头快乐的母牛，它将提供更多的牛奶。他已安排一个聪明的活动中，他们操纵奶牛*中号*× *Ñ*栅格（1≤*中号*≤15; 1≤ *ñ* ≤15）正方形瓷砖，其中的每一个着色为黑色的一侧和白色的另一侧。

就像人们猜测的那样，当翻转单个白色图块时，它会变为黑色。翻转单个黑色图块时，它将变为白色。母牛在翻转砖块时会得到奖励，因此每块砖块的白色侧面都朝上。但是，母牛的蹄子相当大，当他们尝试翻转特定的瓷砖时，它们也会翻转所有相邻的瓷砖（与翻转的瓷砖共享完整边缘的瓷砖）。由于翻转很累，奶牛希望尽量减少翻转次数。

帮助母牛确定所需的最小翻转次数，以及达到该最小翻转的位置。如果有多种方法可以以最小的翻转次数来完成任务，则当将其视为字符串时，以输出中词典顺序最少的方式返回。如果无法完成任务，请用“ IMPOSSIBLE”一词打印一行。

**输入项**

第1行：两个以空格分隔的整数：*M*和*N*  
第2行*。M* +1：第*i* +1行描述了网格第i行的颜色（从左到右），其中*N个以*空格分隔的整数，黑色和1为1。 0代表白色

**输出量**

第1 .. *M*行：每行包含*N个以*空格分隔的整数，每个整数指定翻转该特定位置的次数。

**样本输入**

4 4

1 0 0 1

0 1 1 0

0 1 1 0

1 0 0 1

**样本输出**

0 0 0 0

1 0 0 1

1 0 0 1

0 0 0 0

### 2 基础知识

深度优先搜素（解法一用到）

枚举（解法一用到）

二进制（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

此题可以利用枚举来做，因为吧，棋子翻转的时候会影响到周围的棋子，所以我们可以枚举出第一行棋子的所有状态，实际上第一行的状态确定了，那么剩下行的状态也就随之确定了，只需对最后一行的棋子判断是否全为白色就可以了，如果全是白色就是可以的。此题目主要就是借助二进制的思想来枚举棋子状态。

本人代码：

|  |
| --- |
| #include <iostream>  #include<algorithm>  #include <cstring>  #include <cstdio>  #include <cstdlib>  #include <cmath>  #include <vector>  #include <queue>  #include <stack>  #include <map>  #include <set>  using namespace std;  int coor[5][2] = {0,0,0,1,0,-1,-1,0,1,0};//自身，上下左右  int M,N;  //mapp存放初始棋盘，turn存放每个棋子的翻转次数，ans存放最后的结果  int mapp[20][20],turn[20][20],ans[20][20];  //res存放最小翻转次数，cnt存放当前翻转的总次数  int res,cnt;  //获取当前点的颜色，0--白色，1--黑色  int getColor(int x,int y)  {      int temp = mapp[x][y];      //因为当前点的颜色受自身以及周围四个翻转次数的影响      for (int i = 0; i < 5; i++)      {          int xx = x + coor[i][0];          int yy = y + coor[i][1];          if (xx < 0 || yy < 0 || x >= M || yy >= N)              continue;          temp += turn[xx][yy];      }      return temp % 2;  }  //从第二层一直搜到最后一层  void dfs()  {      for (int i = 1; i < M; i++)      {          for (int j = 0; j < N; j++)          {              if (getColor(i - 1,j))              {                  //turn[i][j] = 1表示要翻转                  turn[i][j] = 1;                  cnt++;                  //剪枝                  if (cnt > res)                      return ;              }          }      }      //判断最后一层是否都是白色      for (int j = 0; j < N; j++)          if (getColor(M - 1,j))              return ;      //更新结果      if (cnt < res)      {          memcpy(ans,turn,sizeof(turn));          res = cnt;      }      return ;  }  int main()  {      while (scanf("%d %d",&M,&N) != EOF)      {          res = INT\_MAX;          for (int i = 0; i < M; i++)              for (int j = 0; j < N; j++)                  scanf("%d",&mapp[i][j]);          //利用二进制枚举出第一层的所有翻转情况          for (int i = 0; i < 1<<N; i++)          {              cnt = 0;              memset(turn,0,sizeof(turn));              for (int j = 0; j < N; j++)              {                  //因为要字典序，所以从后往前填                  //turn记录的就是当前点的翻转状态                  turn[0][N - 1 - j] = i>>j&1;                  if (turn[0][N - 1 - j])                      cnt++;              }              dfs();          }          if (res == INT\_MAX)              printf("IMPOSSIBLE");          else          {              for (int i = 0; i < M; i++)              {                  for (int j = 0; j < N; j++)                  {                      if (j == 0)                          printf("%d",ans[i][j]);                      else                          printf(" %d",ans[i][j]);                  }                  printf("\n");              }          }      }      return 0;  } |

## poj 2387

### 1 原题目

**直到母牛回家**

|  |  |  |
| --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 65536K |
| **提交总数：** 95828 |  | **接受的：** 31090 |

**描述** <http://poj.org/problem?id=2387>

贝西（Bessie）在野外，想回到谷仓以尽可能多地睡觉，直到农夫约翰（John Farmer）叫醒她早上进行挤奶。贝西需要美睡，所以她想尽快回来。  
  
农夫约翰的田地里有N（2 <= N <= 1000）个地标，唯一编号为1..N。地标1是谷仓；苹果树树林其中贝西代表整天使用地标之间的各种长度的T（1 <= T <= 2000）双向牛步道在现场的地标N.奶牛行进。贝西对自己的导航能力不抱有信心，因此，从开始到结束，她始终保持在步道上。  
  
给定地标之间的路径，确定Bessie返回谷仓必须行走的最小距离。可以保证存在这样的路由。

**输入项**

\*第1行：两个整数：T和N  
  
\*第2..T + 1行：每行将一个轨迹描述为三个以空格分隔的整数。前两个整数是小径在其间行进的地标。第三个整数是路径的长度，范围为1..100。

**输出量**

\*第1行：一个整数，是Bessie从地标N到地标1所必须经过的最小距离。

**样本输入**

5 5

1 2 20

2 3 30

3 4 20

4 5 20

1 5 100

**样本输出**

90

**暗示**

输入详细信息：  
  
有五个地标。  
  
输出详细信息：  
  
Bessie可以按照4、3、2和1的路线回家。

### 2 题目分析

让你找到一条从起点1到终点n的最短路径

### 3 基础知识

#### 3.1数据结构

有向图（解法一用到）

#### 3.2算法

迪杰斯特拉算法（解法一用到）

### 4 解法

#### 4.1 解法1（java已过）

最短路问题，利用迪杰斯特拉算法来求解，直接套用算法的模板就可以，具体请看代码及注释。

本人代码：

**import** java.util.\*;

**public** **class** Main {

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

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

**while**(in.hasNext()) {//多次输入

**int** t,n,i,j,row,col,dis;

t = in.nextInt();

n = in.nextInt();

**long** [][] map = **new** **long**[n + 1][n + 1];//由题中数据构建有向图

**long**[] vis = **new** **long**[n + 1];//记录是否已经确定了最短路径

**long**[] dist = **new** **long**[n + 1];//记录从源点到终点的最短路径长度

**for**(i = 0; i < n + 1; i++)

**for**(j = 0; j < n + 1; j++)

map[i][j] = Integer.***MAX\_VALUE***;//初始化

**for**(i = 0;i < n + 1; i++)

dist[i] = Integer.***MAX\_VALUE***;//初始化

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

row = in.nextInt();

col = in.nextInt();

dis = in.nextInt();

**if**(dis < map[row][col]) {//输入的数据存储较小的那个

map[row][col] = dis;

map[col][row] = dis;

}

}

Main dij = **new** Main();

dij.dijastra(map, vis, dist, n);

System.***out***.println(dist[n]);//输出结果

}

}

**public** **void** dijastra(**long**[][] map,**long**[] vis,**long**[] dist,**int** n) {//迪杰斯特拉算法

**int** v = 0;

**long** min;

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

dist[i] = map[1][i];//将起点到各个终点的最短路径长度初始化为弧上的权值

}

vis[1] = 1;

**for**(**int** j = 1; j <= n; j++) {//初始化结束，开始循环，每次求得起点到某个顶点v的最短路径，将v加入到vis中

min = Integer.***MAX\_VALUE***;

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

**if**(vis[k] == 0 && dist[k] < min) {//选择一条当前的最短路径，终点为v

v = k;

min = dist[k];

}

}

vis[v] = 1;//将v加入vis中

**for**(**int** w = 1; w <= n; w++) {//更新从起点出发所有顶点的最短路径长度

**if**(vis[w] == 0 && (dist[v] + map[v][w] < dist[w])) {

dist[w] = dist[v] + map[v][w];

}

}

}

**return** ;

}

}

## poj 3268

### 1 原题目

**银牛党**

|  |  |  |
| --- | --- | --- |
| **时限：** 2000MS |  | **内存限制：** 65536K |
| **提交总数：** 37274 |  | **接受：** 16663 |

**描述** <http://poj.org/problem?id=3268>

从每一个牛*ñ*农场（1≤ *ñ* ≤1000）方便地编号为1 .. *ñ*是要参加大牛党在农场＃举行*X*（1≤ *X* ≤ *ñ*）。共*中号*（1≤ *中号* ≤100,000）农场单向（单向道路所连接对;道路*我*需要*Ť 我*（1≤ *Ť 我* ≤100）的时间到横动单元。

每头母牛都必须走到聚会上，聚会结束后，要回到自己的农场。每头母牛都是懒惰的，因此会在最短的时间内选择一条最佳路线。由于道路是单向的，所以母牛的回程路线可能与原先到达聚会的路线有所不同。

在所有奶牛中，奶牛步行去聚会和返回所花费的最长时间是多少？

**输入值**

分别三空格隔开的整数，：1线*Ñ*，*中号*，和*X*  
线2 .. *中号* 1：线*我* 1描述了道路*我*有三个空格隔开的整数：*甲我*，*乙我*和*Ť 我*。所描述的道路从农场*A i*到农场*B i*，需要经过*T i个*时间单位。

**输出量**

第1行：一个整数：任何一头母牛必须行走的最长时间。

**样本输入**

4 8 2

1 2 4

1 3 2

1 4 7

2 1 1

2 3 5

3 1 2

3 4 4

4 2 3

**样本输出**

10

**暗示**

母牛4直接进入聚会（3个单位），并通过农场1和3（7个单位）返回，总共10个时间单位。

### 2 题目分析

有N个农场，每个农场一头牛，给你一个有向图表示农场与农场之间的距离。所有牛要去同一个农场X，然后再回到各自原来的农场，让你求解出所有母牛中一来一回的最长路程（母牛去和回来都走的是最短路径）

### 3 基础知识

#### 3.1数据结构

有向图（解法一用到）

#### 3.2算法

迪杰斯特拉算法（解法一用到）

### 4 解法

#### 4.1 解法1（C++已过）

母牛从X回到自己的农场的最短距离可以直接利用所给的有向图进行迪杰斯特拉算法来做。

不同母牛从不同农场分别去X的最短路径可以利用所给的有向图倒过来存储然后再迪杰斯特拉算法来求最短路径。

本人代码：

#include <iostream>

#include <cstring>

#include <climits>

using namespace std;

int map[1005][1005];//存正向的有向图

int map1[1005][1005];//存逆向的有向图（正向的倒过来存储）

int qusign[1005];//标记数组

int huisign[1005];

int qu\_len[1005];//记录母牛去的过程中从起点到当前位置的距离

int hui\_len[1005];//记录母牛回来的过程中从起点到当前位置的距离

void qu\_dij(int x,int n)

{

memset(qusign,0,sizeof(qusign));

int i,v,w,min;

for(v = 1; v <= n; v++)

{

qu\_len[v] = INT\_MAX;

if(map1[x][v] != -1)

qu\_len[v] = map1[x][v];

}

qu\_len[x] = 0;

qusign[x] = 1;

for(i = 2; i <= n; i++)

{

min = INT\_MAX;

for(w = 1; w <= n; w++)

{

if(qusign[w] == 0 && qu\_len[w] < min)

{

v = w;

min = qu\_len[w];

}

}

qusign[v] = 1;

for(w = 1; w <= n; w++)

{

if(map1[v][w] != -1 && qusign[w] == 0 && (qu\_len[v] + map1[v][w] < qu\_len[w]))

{

qu\_len[w] = qu\_len[v] + map1[v][w];

}

}

}

}

void hui\_dij(int x,int n)

{

memset(huisign,0,sizeof(huisign));//标记数组初始化

int i,v,w,min;

for(v = 1; v <= n; v++)//n个顶点依次初始化

{

hui\_len[v] = INT\_MAX;

if(map[x][v] != -1)

hui\_len[v] = map[x][v];//将x到各个终点的最短路径长度初始化为弧上的权值

}

hui\_len[x] = 0;//源点到源点的距离为0

huisign[x] = 1;//将源点加入到标记数组中

//初始化结束，开始循环，每次求得x到某个顶点v的最短路径，将v加入到标记数组中

for(i = 2; i <= n; i++)

{

min = INT\_MAX;

for(w = 1; w <= n; w++)

{

if(huisign[w] == 0 && hui\_len[w] < min)//选择一条当前的最短路径，终点为v

{

v = w;

min = hui\_len[w];

}

}

huisign[v] = 1;//将v加入标记数组中

for(w = 1; w <= n; w++)//更新从x出发到集合v-s上所有顶点的最短路径长度

{

if(map[v][w] != -1 && huisign[w] == 0 && (hui\_len[v] + map[v][w] < hui\_len[w]))

{

hui\_len[w] = hui\_len[v] + map[v][w];//更新hui\_len[w]

}

}

}

}

int main()

{

int N,M,X;

while(cin>>N>>M>>X)

{

memset(map,-1,sizeof(map));//有向图的初始化

memset(map1,-1,sizeof(map1));

int i,a,b,c;

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

{

cin>>a>>b>>c;

map[a][b] = c;//存储初始的有向图

map1[b][a] = c;//将有向图倒过来存储

}

hui\_dij(X,N);//母牛回来所走的路程

qu\_dij(X,N);//母牛过去所走的路程

int max = INT\_MIN;

for(i = 1; i <= N; i++)

{

if(i == X)

continue;

if(qu\_len[i] + hui\_len[i] > max)//计算所有母牛中的最大路程

max = qu\_len[i] + hui\_len[i];

}

cout<<max<<endl;

}

return 0;

}

## poj 2253

### 1 原题目

**青蛙**

|  |  |  |
| --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 65536K |
| **提交总数：** 75248 |  | **接受：** 22912 |

**描述**

弗雷迪·青蛙坐在湖中间的一块石头上。突然，他注意到坐在另一块石头上的菲奥娜·青蛙。他计划去探望她，但由于水很脏，到处都是游客的防晒霜，他想避免游泳，而是跳跳来到达她。  
不幸的是，菲奥娜的石头不在他的跳跃范围内。因此，弗雷迪（Freddy）考虑使用其他石头作为中间停靠点，并通过几次小跳来到达她。  
要执行给定的跳跃序列，青蛙的跳跃范围显然必须至少与序列中发生的最长跳跃一样长。  
因此，将两块石头之间的青蛙距离（人类也称为最小最大距离）定义为两块石头之间所有可能路径上的最小必要跳跃范围。  
  
您将获得弗雷迪石头，菲奥娜石头和湖中所有其他石头的坐标。您的工作是计算Freddy和Fiona的石头之间的青蛙距离。

**输入项**

输入将包含一个或多个测试用例。每个测试用例的第一行将包含结石数量n（2 <= n <= 200）。接下来的n条线分别包含两个整数xi，yi（0 <= xi，yi <= 1000），它们表示石头#i的坐标。第1块石头是弗雷迪的石头，第2块石头是菲奥娜的石头，其他n-2块石头没有人使用。每个测试用例后面都有一个空白行。输入以n的零（0）值终止。

**输出量**

对于每个测试用例，打印一行“ Scenario #x”和一行“ Frog Distance = y”，其中x替换为测试用例编号（从1开始编号），y替换为适当的实数，打印到三位小数。在每个测试用例之后，甚至在最后一个测试用例之后，都应留一个空白行。

**样本输入**

2

0 0

3 4

3

17 4

19 4

18 5

0

**样本输出**

Scenario #1

Frog Distance = 5.000

Scenario #2

Frog Distance = 1.414

### 2 题目分析

就是源点到终点有多条的路径，每一条路径中都有一段最大的距离！求这些路径中最大距离的最小值！

### 3 基础知识

#### 3.1数据结构

无。

#### 3.2算法

迪杰斯特拉算法（解法一用到）

### 4 解法

#### 4.1 解法1（C++已过）

一个简单的最短路径变形（用迪杰斯特拉算法），只要将length[i]的意义变为从0到i的最大边即可，然后修改下更新条件就可以了。

本人代码：

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <limits.h>

int sign[205];//标记点是否被走过

double length[205];//记录源点到当前点所有路径中的最大边中的最小边

double Graph[205][2];//存地图

int n;

double max(double a,double b)//比较大小

{

if(a > b)

return a;

return b;

}

double distance(double a,double b,double c,double d)//计算两点间的距离

{

return sqrt((a - c) \* (a - c) + (b - d) \* (b - d));

}

void dijkstra()

{

int i,v,w,min;

for(v = 0; v < n; v++)//初始化

{

sign[v] = 0;

length[v] = INT\_MAX;

}

length[0] = 0;//源点到源点最大边为0

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

{

min = INT\_MAX;

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

{

if(sign[w] == 0 && length[w] < min)//选择一条当前的最短路径，终点为v

{

v = w;

min = length[w];

}

}

sign[v] = 1;//将v加入sign中

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

{

if(sign[w] == 0 && max(length[v],distance(Graph[v][0],Graph[v][1],Graph[w][0],Graph[w][1])) < length[w])//因为题意找所有路径中最大边中的最小边，所以条件是这个

{

length[w]=max(length[v],distance(Graph[v][0],Graph[v][1],Graph[w][0],Graph[w][1]));

}

}

}

}

int main()

{

int m = 0;

while(scanf("%d",&n) != EOF)

{

if(n == 0)

break;

m++;

int i = 0;

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

scanf("%lf %lf",&Graph[i][0],&Graph[i][1]);

dijkstra();

printf("Scenario #%d\n",m);

printf("Frog Distance = %.3lf\n\n",length[1]);

getchar();

}

return 0;

}

## poj 1797

### 1 原题目

**重型运输**

|  |  |  |
| --- | --- | --- |
| **时限：** 3000MS |  | **内存限制：** 30000K |
| **提交总数：** 63325 |  | **接受：** 15635 |

**描述** <http://poj.org/problem?id=1797>

背景  
雨果重工很高兴。Cargolifter项目崩溃后，他现在可以扩展业务。但是他需要一个聪明的人，告诉他从客户建造巨型钢起重机的地方到需要所有街道都能承受重量的地方是否真的有办法。  
幸运的是，他已经制定了一个计划，包括所有街道和桥梁以及所有允许的重量。不幸的是，他不知道如何找到最大重量以告诉客户起重机的重量。但是你当然知道。  
  
问题  
系统会为您提供城市规划，由交叉口之间的街道（权重限制）描述，从1到n。您的任务是找到从交叉口1（雨果的住所）到交叉口n（客户的住所）的最大重量。您可以假设至少有一条路径。所有街道均可双向行驶。

**输入值**

第一行包含方案（城市计划）的数量。对于每个城市，第一行给出了路口的数量n（1 <= n <= 1000）和街道数量m。接下来的m行包含整数的三元组，这些整数指定街道的起点和终点交叉点以及允许的最大权重，其为正且不大于1000000。每对交叉点之间最多有一条街道。

**输出量**

每个方案的输出都以包含“方案#i：”的行开头，其中i是从1开始的方案编号。然后打印一行，其中包含Hugo可以运输给客户的最大允许重量。用空白行终止方案的输出。

**样本输入**

1

3 3

1 2 3

1 3 4

2 3 5

**样本输出**

Scenario #1:

4

### 2 题目分析

就是源点到终点有多条的路径，每一条路径中都有一段最小的承载量！求这些路径中最小承载量中的最大的承载量！

### 3 基础知识

#### 3.1数据结构

无。

#### 3.2算法

迪杰斯特拉算法（解法一用到）

### 4 解法

#### 4.1 解法1（java已过）

一个简单的最短路径变形（用迪杰斯特拉算法），只要将dist[i]的意义变为从1到n的最大承载量即可，然后修改下更新条件就可以了。

本人代码：

**import** java.util.\*;//包含java.util包中所有的类

**public** **class** Main {

**public** **void** dijastra(**int**[][] map, **int**[] vis, **int**[] dist, **int** n) {//迪杰斯特拉算法

**int** v = 0, max;

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

vis[i] = 0;//标记数组初始化

dist[i] = map[1][i];

}

dist[1] = 0;

vis[1] = 0;

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

max = Integer.***MIN\_VALUE***;

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

**if** (vis[k] == 0 && dist[k] > max) {//选择一条当前的最长路径，终点为v

v = k;

max = dist[k];

}

}

vis[v] = 1;//将v加入vis中

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

**if** (vis[w] == 0 && Math.*min*(dist[v], map[v][w]) > dist[w]) {//因为题意找所有路径中最小承载量中的最大承载量，所以条件是这个

dist[w] = Math.*min*(dist[v], map[v][w]);

}

}

}

**return**;

}

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

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

**int**[][] map = **new** **int**[1010][1010];//存地图

**int**[] vis = **new** **int**[1010];//标记点是否被走过

**int**[] dist = **new** **int**[1010];//表示从1到n的所有路径中的最大承载量

**while** (in.hasNext()) {

**int** z, m, n, k;

z = in.nextInt();

**for** (k = 1; k <= z; k++) {

n = in.nextInt();

m = in.nextInt();

**int** i, j, p, q, temp;

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

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

map[i][j] = 0;//地图初始化

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

p = in.nextInt();

q = in.nextInt();

temp = in.nextInt();

map[p][q] = temp;

map[q][p] = temp;

}

Main dij = **new** Main();

dij.dijastra( map, vis, dist, n);

System.***out***.println("Scenario #" + k + ":");

System.***out***.println(dist[n] + "\n");

}

}

}

}

## poj 1502

### 1 原题目

**MPI大漩涡**

|  |  |  |
| --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 10000K |
| **提交总数：** 15137 |  | **接受：** 9209 |

**描述** <http://poj.org/problem?id=1502>

BIT最近接受了他们的新超级计算机的交付，该超级计算机是具有分层通信子系统的32处理器Apollo Odyssey分布式共享内存计算机。Valentine McKee的研究顾问Jack Swigert已要求她对新系统进行基准测试。  
瓦伦丁对斯维格特说：``由于阿波罗是一台分布式共享存储机器，因此存储访问和通信时间并不统一。'' ``共享同一内存子系统的处理器之间的通信速度很快，但是不在同一子系统上的处理器之间的通信速度却较慢。阿波罗和我们实验室中的机器之间的通讯速度还慢。”“  
  
阿波罗的消息传递接口（MPI）的端口如何工作？”斯维格特问。  
  
``不太好，''瓦伦丁回答。``要将消息从一个处理器广播到所有其他n-1个处理器，它们只是执行一系列n-1发送。这真的使事情序列化并破坏了性能  
  
.````有什么可以解决的吗？''  
  
``是的，''情人微笑。``有。一旦第一个处理器将消息发送到另一个，这两个处理器就可以同时将消息发送到另外两个主机。然后将有四个可以发送的主机，以此类推  
  
.````啊，所以您可以像二叉树一样进行广播！''  
  
``不是真正的二叉树-我们应该利用我们网络的某些特殊功能。我们拥有的接口卡允许每个处理器同时将消息发送到与其连接的任意数量的其他处理器。但是，消息不一定要同时到达目的地-涉及通信成本。总的来说，我们需要考虑网络拓扑中每个链接的通信成本，并做出相应的计划以最大程度地减少广播所需的总时间。''

**输入项**

输入将描述连接n个处理器的网络的拓扑。输入的第一行将是n，即处理器的数量，因此1 <= n <=100。  
  
输入的其余部分定义一个邻接矩阵A。邻接矩阵是正方形，大小为nx n。它的每个条目都是整数或字符x。A（i，j）的值表示直接从节点i向节点j发送消息的开销。A（i，j）的x值表示无法将消息直接从节点i发送到节点j。  
  
请注意，对于节点向其自身发送消息的不需要网络通信，因此对于1 <= i <= n，A（i，i）= 0。此外，您可能会假定网络是无向的（消息可以以相同的开销在任一方向上传播），因此A（i，j）= A（j，i）。因此，将仅提供A的（严格）下部三角形部分上的条目。  
  
程序的输入将是A的下部三角形部分。也就是说，输入的第二行将包含一个条目A（2,1）。下一行将包含两个条目A（3,1）和A（3,2），依此类推。

**输出量**

您的程序应输出从第一个处理器向所有其他处理器广播消息所需的最短通信时间。

**样本输入**

5

50

30 5

100 20 50

10 x x 10

**样本输出**

35

### 2 题目分析

实际就是从起点到各个点的最短路中找出一个最大的。

### 3 基础知识

#### 3.1数据结构

无向图（解法一用到）

#### 3.2算法

迪杰斯特拉算法（解法一用到）

### 4 解法

#### 4.1 解法1（C++已过）

实际就是从起点到各个点的最短路中找出一个最大的。

本人代码：

#include <iostream>

#include <cstring>

#include <cstdlib>

using namespace std;

int map[105][105];

//之所以数组开辟成long long int是因为防止下面有可能出现越界的情况

int sign[105];//记录从源点v0到终点是否已经被确定最短路径

long long int length[105];//记录从源点v0到终点的当前最短路径长度

//用迪杰斯特拉算法求有向图的v0点到其余各点的最短路径

void dijkstra(int Graph[105][105],int v0,int n)//n为有向图顶点个数

{

memset(sign,0,sizeof(sign));//sign初始化为0

int i,v,w,min;

for(v = 1; v <= n; v++)//n个顶点依次初始化

{

length[v] = Graph[v0][v];//将v0到各个终点的最短路径长度初始化为弧上的权值

}

length[v0] = 0;//源点到源点的距离为0

sign[v0] = 1;//将v0加入到sign中

//初始化结束，开始循环，每次求得v0到某个顶点v的最短路径，将v加入到sign中

for(i = 2; i <= n; i++)//对其余n-1个顶点依次进行计算

{

min = INT\_MAX;

for(w = 1; w <= n; w++)

{

if(sign[w] == 0 && length[w] < min)//选择一条当前的最短路径，终点为v

{

v = w;

min = length[w];

}

}

sign[v] = 1;//将v加入sign中

for(w = 1; w <= n; w++)//更新从v0出发到集合v-s上所有顶点的最短路径长度

{

if(sign[w] == 0 && (length[v] + Graph[v][w] < length[w]))

{

length[w] = length[v] + Graph[v][w];//更新length[w]

}

}

}

}

int main()

{

int n;

while(cin>>n)

{

int i,j;

char s[10];

int max = INT\_MIN;

for(i = 1; i <= n; i++)//地图初始化

for(j = 1; j <= n; j++)

if(i != j)

map[i][j] = INT\_MAX;

else

map[i][j] = 0;

for(i = 2; i <= n; i++)

for(j = 1; j < i; j++)

{

cin>>s;

if(s[0] != 'x')

map[i][j] = map[j][i] = atoi(s);//将字符串转换为数字

}

dijkstra(map,1,n);//n为有向图顶点数，1为源点

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

if(length[i] > max)//在所有的最短路径中找出一个最大的

max = length[i];

cout<<max<<endl;

}

return 0;

}

## poj 1062

### 1 原题目

**昂贵的聘礼**

|  |  |  |
| --- | --- | --- |
| **Time Limit:** 1000MS |  | **Memory Limit:** 10000K |
| **Total Submissions:** 63565 |  | **Accepted:** 19284 |

**Description** <http://poj.org/problem?id=1062>

年轻的探险家来到了一个印第安部落里。在那里他和酋长的女儿相爱了，于是便向酋长去求亲。酋长要他用10000个金币作为聘礼才答应把女儿嫁给他。探险家拿不出这么多金币，便请求酋长降低要求。酋长说："嗯，如果你能够替我弄到大祭司的皮袄，我可以只要8000金币。如果你能够弄来他的水晶球，那么只要5000金币就行了。"探险家就跑到大祭司那里，向他要求皮袄或水晶球，大祭司要他用金币来换，或者替他弄来其他的东西，他可以降低价格。探险家于是又跑到其他地方，其他人也提出了类似的要求，或者直接用金币换，或者找到其他东西就可以降低价格。不过探险家没必要用多样东西去换一样东西，因为不会得到更低的价格。探险家现在很需要你的帮忙，让他用最少的金币娶到自己的心上人。另外他要告诉你的是，在这个部落里，等级观念十分森严。地位差距超过一定限制的两个人之间不会进行任何形式的直接接触，包括交易。他是一个外来人，所以可以不受这些限制。但是如果他和某个地位较低的人进行了交易，地位较高的的人不会再和他交易，他们认为这样等于是间接接触，反过来也一样。因此你需要在考虑所有的情况以后给他提供一个最好的方案。  
为了方便起见，我们把所有的物品从1开始进行编号，酋长的允诺也看作一个物品，并且编号总是1。每个物品都有对应的价格P，主人的地位等级L，以及一系列的替代品Ti和该替代品所对应的"优惠"Vi。如果两人地位等级差距超过了M，就不能"间接交易"。你必须根据这些数据来计算出探险家最少需要多少金币才能娶到酋长的女儿。

**Input**

输入第一行是两个整数M，N（1 <= N <= 100），依次表示地位等级差距限制和物品的总数。接下来按照编号从小到大依次给出了N个物品的描述。每个物品的描述开头是三个非负整数P、L、X（X < N），依次表示该物品的价格、主人的地位等级和替代品总数。接下来X行每行包括两个整数T和V，分别表示替代品的编号和"优惠价格"。

**Output**

输出最少需要的金币数。

**Sample Input**

1 4

10000 3 2

2 8000

3 5000

1000 2 1

4 200

3000 2 1

4 200

50 2 0

**Sample Output**

5250

### 2 基础知识

枚举（解法一用到）

迪杰斯特拉算法（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

此题看题目就知道是最短路算法，所以想到了用迪杰斯特拉算法，点和点之间的距离是优惠后的价格（建图成功）。但是问题来了，此题还有个等级差距，所以导致我们的算法受限。但是，我们可以利用枚举来解决此问题。假设酋长等级为3，最大差距为2，那么如果以酋长为起点，可以走的点就是1到5，但如果直接1到5进行迪杰斯特拉算法是不可取的，因为点1到点5的等级差大于2，所以利用枚举将区间划分为1～3，2～4，3～5，这样每个区间的任意两点的等级都小于等于最大差距，所以就可以每个区间进行迪杰斯特拉算法。然后取金币花费最少的就可以了。

参考链接：<https://blog.csdn.net/Jaster_wisdom/article/details/50849650>

本人代码：

|  |
| --- |
| #include <iostream>  #include<algorithm>  #include <cstring>  #include <cstdio>  #include <cstdlib>  #include <cmath>  using namespace std;  #define N 109  #define MIN(a,b) a>b?b:a  struct node//存储主人宝物的价格和主人的等级  {      int money;      int level;  } no[N];  int m,n;  int Graph[N][N];//存储点到点的信息  long long int length[N];//记录从源点到终点的当前最少金币数  int sign[N];//标记点是否访问过  int limit[N];//标记点是否被限制（不能参与交换）  int dijkastra()//迪杰斯特拉算法  {      memset(sign,0,sizeof(sign));      int i,v,w,minn;      for(v = 1; v <= n; v++)          length[v] = INT\_MAX;      length[1] = 0;      for(i = 2; i <= n; i++)      {          minn = INT\_MAX;          for(w = 1; w <= n; w++)          {              if(sign[w] == 0 && length[w] <= minn && limit[w])//选择一条当前的花费最少的路径，终点为v              {                  v = w;                  minn = length[w];              }          }          sign[v] = 1;          for(w = 1; w <= n; w++)//更新从起点出发到集合v-s上所有顶点的最少金币数          {              if((length[v] + Graph[v][w] < length[w]) && limit[w] && sign[w] == 0)              {                  length[w] = length[v] + Graph[v][w];//更新length[w]              }          }      }      int ans = INT\_MAX;      for(i = 1; i <= n; i++)//加上自身所花的金币      {          length[i] = length[i] + no[i].money;          ans = MIN(ans,length[i]);      }      return ans;  }  int main()  {      scanf("%d %d",&m,&n);      int i,j;      for(i = 0; i < N; i++)//初始化Graph为整型最大值，用以表示无连接          for(j = 0; j < N; j++)              Graph[i][j] = INT\_MAX;      for(i = 1; i <= n; i++)      {          int p,l,x;          scanf("%d %d %d",&p,&l,&x);          no[i].money = p;          no[i].level = l;          for(j = 0; j < x; j++)          {              int t,v;              scanf("%d %d",&t,&v);              Graph[i][t] = v;          }      }      int ans = INT\_MAX,kinglevel = no[1].level;      //枚举等级区间      for(i = 0; i <= m; i++)      {          memset(limit,0,sizeof(limit));          for(j = 1; j <= n; j++)          {              if((no[j].level >= kinglevel - m + i) && (no[j].level <= kinglevel + i))                  limit[j] = 1;          }          ans = MIN(ans,dijkastra());      }      printf("%d",ans);      return 0;  } |

## poj 1860

### 1 原题目

**货币兑换**

|  |  |  |
| --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 30000K |
| **提交总数：** 43787 |  | **接受的：** 16851 |

**描述** <http://poj.org/problem?id=1860>

我们的城市有几个货币兑换点。让我们假设每个点专门研究两种特定货币，并且仅使用这些货币执行兑换操作。同一对货币可能有多个要点。每个点都有自己的汇率，A到B的汇率是您获得1A时B的数量。另外，每个兑换点都有一定的佣金，您必须为兑换操作支付的总金额。佣金始终以原始货币收取。  
例如，如果您想在兑换点将100美元兑换成俄罗斯卢布，汇率为29.75，佣金为0.39，您将获得（100-0.39）\* 29.75 = 2963.3975RUR。  
您肯定知道我们城市可以处理N种不同的货币。让我们为每种货币分配从1到N的唯一整数。然后可以用6个数字描述每个交换点：整数A和B-​​交换的货币数量，以及实际R AB，C AB，R BA和C BA-分别将A兑换为B和B兑换为A时的汇率和佣金。  
尼克拥有货币S的钱，并且想知道是否可以在进行一些兑换操作后以某种方式增加其资本。当然，他希望最后用S货币来存钱。帮助他回答这个难题。尼克在开展业务时必须始终拥有非负数的资金。

**输入项**

输入的第一行包含四个数字：N-货币数量，M-兑换点数量，S-Nick拥有的货币数量，V-他拥有的货币单位数量。接下来的M行每个包含6个数字-相应交换点的描述-按上述指定的顺序。数字用一个或多个空格分隔。1 <= S <= N <= 100，1 <= M <= 100，V是实数，0 <= V <= 10 3。  
对于每个点的汇率和佣金是真实的，与小数点后至多两个数字，10给出-2 <=速率<= 10 2，0 <=佣金<= 10 2。  
如果在此序列中不使用任何交换点以上，则将交换操作的某些序列称为简单序列。您可以假定在任何简单的交换操作序列的末尾和开始时，总和的数值之比小于10 4。

**输出量**

如果Nick可以增加财富，则输出YES，否则输出NO到输出文件。

**样本输入**

3 2 1 20.0

1 2 1.00 1.00 1.00 1.00

2 3 1.10 1.00 1.10 1.00

**样本输出**

YES

### 2 题目分析

给你一些钱币的汇率和佣金，还有初始的钱币数量，让你通过不断换钱看是否能挣到钱（最后要换成初始钱的种类，要不然无法比较）。

### 3 基础知识

#### 3.1数据结构

前向星（解法一用到）

#### 3.2算法

SPFA算法（解法一用到）

### 4 解法

#### 4.1 解法1（C++已过）

SPFA算法的模板题，只不过此题的路径更新是往上更新，所以如果出现了正向回路也是返回true。

本人代码：

#include <iostream>

#include <climits>

#include <queue>

#include <cstring>

using namespace std;

const int MAXN = 1100;

struct Edge//边结构体

{

int next;//与第i条边同一起点的上一条边的位置

int to;//此边的终点

double yongjin;//佣金

double huilv;//汇率

} edge[MAXN];

int head[MAXN];//以i为起点的第一条边存储的位置（实际是最后输入的那个编号）

int index = 0;

bool vis[MAXN];//用来标记点是否在队列中

int num[MAXN];//每个点的入队列次数

double length[MAXN];//记录从起点到点n的钱数

void add(int start,int end,double huilv,double yongjin)//链式前向星的加边函数

{

edge[index].huilv = huilv;

edge[index].yongjin = yongjin;

edge[index].to = end;

edge[index].next = head[start];

head[start] = index++;

}

bool SPFA(int start,int n,double temp)//start是起始节点，n是节点总数，temp是初始钱数

{

int j;

memset(vis,false,sizeof(vis));

memset(num,0,sizeof(num));

memset(length,0,sizeof(length));

length[start] = temp;

queue<int>que;

while(!que.empty())

que.pop();

que.push(start);//入队

vis[start] = true;//标记此点入队列

num[start]++;//入队次数加1

while(!que.empty())

{

int u = que.front();

que.pop();//出队

vis[u] = false;//标记此点已经不在队列中

for(j = head[u]; j >= 0; j = edge[j].next)//遍历与u联通的点

{

int v = edge[j].to;

if(length[v] < (length[u] - edge[j].yongjin) \* edge[j].huilv)

{

length[v] = (length[u] - edge[j].yongjin) \* edge[j].huilv;//钱数更新

if(vis[v] == 0)//如果v点不在队列中

{

que.push(v);//入队

vis[v] = 1;//标记

num[v]++;//入队次数加1

if(num[v] > n)//如果这个点加入超过n次，说明存在正环回路，直接返回

return true;

}

}

if(length[start] > temp)//如果钱数大于初始值（同一种钱）

return true;

}

}

return false;

}

int main()

{

int n,m,s;

double v;

while(cin>>n>>m>>s>>v)

{

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

int i,a,b;

double c,d,e,f;

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

{

cin>>a>>b>>c>>d>>e>>f;

add(a,b,c,d);//构建前向星

add(b,a,e,f);

}

if(SPFA(s,n,v))

cout<<"YES"<<endl;

else

cout<<"NO"<<endl;

}

return 0;

}

## poj 3259

### 1 原题目

**虫洞**

|  |  |  |
| --- | --- | --- |
| **时限：** 2000MS |  | **内存限制：** 65536K |
| **提交总数：** 76938 |  | **接受：** 28602 |

**描述** <http://poj.org/problem?id=3259>

在探索他的许多农场时，农夫约翰发现了许多惊人的虫洞。虫洞非常特殊，因为它是一条单向路径，可在您进入虫洞之前将您送至目的地！每个FJ的农场包括*Ñ*（1≤ *ñ* ≤500）字段方便地编号为1 .. *Ñ*，*中号*（1≤ *中号* ≤2500）的路径，和*w ^*（1≤ *w ^* ≤200）虫洞。

由于FJ是一位热衷时间旅行的爱好者，因此他希望做到以下几点：从某个领域开始，经过一些路径和虫洞，并在他初次离开之前的某个时间返回到开始领域。也许他将能够见到自己：）。

为了帮助FJ找出这是否可行与否，他将与完整的地图供你*˚F*（1≤ *˚F* ≤5）他的农场。没有任何路径会花费超过10,000秒的时间，虫洞也不会使FJ的返回时间超过10,000秒。

**输入值**

第1行：一个整数，*˚F*。*F*服务器场说明如下。  
每个服务器场的第1行：三个空格分隔的整数：*N*，*M*和*W*  
第2行。每个服务器场的*M* +1：分别描述的三个空格分隔的数字（*S*，*E*，*T*）：双向路径在*S*和*E*之间需要经过*T*秒。两个字段可能通过一条以上的路径连接。  
线*中号* 2 .. *中号* + *w ^*每个场的1：三个空间分隔的数字（*S*，*E*，*T*）分别描述：从*S*到*E的*单向路径，也将旅行者向后移动*T*秒。

**输出量**

第1行*。F*：对于每个农场，如果FJ可以实现其目标，则输出“ YES”，否则输出“ NO”（不包括引号）。

**样本输入**

2

3 3 1

1 2 2

1 3 4

2 3 1

3 1 3

3 2 1

1 2 3

2 3 4

3 1 8

**样本输出**

NO

YES

**暗示**

对于农场1，FJ无法及时返回。  
对于农场2，FJ可以按1-> 2-> 3-> 1的周期返回时间，在离开前1秒钟回到起始位置。他可以从周期中的任何位置开始以实现此目的。

### 2 题目分析

这有一些虫洞，其中有一些虫洞是可以让你回到过去的，另外一些则是让你往前走，你可以选择任意的起点，题目问你是否可以通过穿越虫洞重新回到这个起点。

### 3 基础知识

#### 3.1数据结构

链式前向星（解法一用到）

#### 3.2算法

SPFA算法（解法一用到）

### 4 解法

#### 4.1 解法1（C++已过）

这道题让你求能否回到原点，又因为有些虫洞是回到未来，就相当于权值为负数，所以此题就是求解一个图中是否存在负权回路，直接套SPFA的模板就可以。

本人代码：

#include <iostream>

#include <climits>

#include <queue>

#include <cstring>

using namespace std;

const int MAXN = 7000;//可根据要求来改变大小

struct Edge//边结构体

{

int next;//与第i条边同一起点的上一条边的位置

int to;//此边的终点

int len;//边的权值

} edge[MAXN];

int head[MAXN];//以i为起点的第一条边存储的位置（实际是最后输入的那个编号）

int index = 0;

bool vis[MAXN];//用来标记点是否在队列中

int num[MAXN];//每个点的入队列次数

int length[MAXN];//记录从起点到点n的路径长

void add(int start,int end,int len)//链式前向星的加边函数

{

edge[index].len = len;

edge[index].to = end;

edge[index].next = head[start];

head[start] = index++;

}

bool SPFA(int start,int n)//start是起始节点，n是节点总数

{

int i,j;

memset(vis,false,sizeof(vis));

memset(num,0,sizeof(num));

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

{

length[i] = INT\_MAX;

}

queue<int>que;

while(!que.empty())

que.pop();

que.push(start);//入队

vis[start] = true;//标记此点入队列

length[start] = 0;//源点到源点的长度为0

num[start]++;//入队次数加1

while(!que.empty())

{

int u = que.front();

que.pop();//出队

vis[u] = false;//标记此点已经不在队列中

for(j = head[u]; j >= 0; j = edge[j].next)//遍历与u联通的点

{

int v = edge[j].to;

if(length[v] > length[u] + edge[j].len)

{

length[v] = length[u] + edge[j].len;//路径更新

if(vis[v] == 0)//如果v点不在队列中

{

que.push(v);//入队

vis[v] = 1;//标记

num[v]++;//入队次数加1

if(num[v] > n)//如果这个点加入超过n次，说明存在负权回路，直接返回

return true;

}

}

}

}

return false;

}

int main()

{

int F,N,M,W,S,E,T;

while(cin>>F)

{

int i;

while(F--)

{

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

index = 0;

cin>>N>>M>>W;

for(i = 1; i <= M; i++)

{

cin>>S>>E>>T;

add(S,E,T);

add(E,S,T);

}

for(; i <= M + W; i++)

{

cin>>S>>E>>T;

add(S,E,-T);

}

if(SPFA(1,N))

cout<<"YES"<<endl;

else

cout<<"NO"<<endl;

}

}

return 0;

}

## poj 2502

### 1 原题目

**地铁**

|  |  |  |
| --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 65536K |
| **提交总数：** 16826 |  | **接受的：** 5364 |

**描述** <http://poj.org/problem?id=2502>

您刚刚从一个安静的滑铁卢社区搬到了一个喧闹的大城市。现在，您不必步行每天骑自行车去上学，而是可以走路和乘地铁。因为您不想上课迟到，所以想知道您要花多长时间才能上学。  
您以10 km / h的速度行走。地铁以每小时40公里的速度行驶。假设您很幸运，每当您到达地铁站时，就有火车可以立即上车。您可以任意次数上下地铁，也可以根据需要在不同的地铁线路之间切换。所有地铁线路都是双向的。

**输入项**

输入内容包括您的房屋和学校的x，y坐标，以及几条地铁线路的规格。每条地铁线路按顺序包括该线路上每个车站的非负整数x，y坐标。您可以假设地铁在相邻站点之间成一直线，并且坐标表示整数米。每条线至少有两个站。每条地铁线的末端都跟着虚拟坐标对-1，-1。该城市总共最多有200个地铁站。

**输出量**

输出是您上学所需的分钟数，四舍五入到最接近的分钟，即最快的路线。

**样本输入**

0 0 10000 1000

0 200 5000 200 7000 200 -1 -1

2000 600 5000 600 10000 600 -1 -1

**样本输出**

21

### 2 题目分析

给你起点和终点的坐标，并且给你一些地铁的站点的位置（站点之间组成地铁线路），你可以从起点坐地铁或者走路到终点，你要保证你花的时间最短，步行速度和地铁的速度都已知。

### 3 基础知识

#### 3.1数据结构

无向图（解法一用到）

#### 3.2算法

弗洛伊德算法(floyd)（解法一用到）

### 4 解法

#### 4.1 解法1（G++已过）

此题就是弗洛伊德算法的模板题，难就难在无向图的构建上。

对于无向图的构建，我是将输入的一对坐标分别排上号（1，2，3……），这样坐标到坐标之间的时间就可以变成点到点的时间了，所以就可以用一个二维数组来表示。同一线路之间的站点所花的时间是按照地铁速度算，上一线路的终点到当前线路的起点的时间按照步行速度算，起点到第一条线路的第一个站点按照步行速度算，最后一条线路的最后一个站点到终点按照步行速度算，其他所有剩下的都按照步行的速度算，这样无向图就构建完成了。

本人代码：

#include <iostream>

#include <cstring>

#include <climits>

#include <cmath>

#include<cstdio>

using namespace std;

long double length[220][220];//记录点到点之间的分钟数

int x[220];//存所有的x坐标

int y[220];//存所有的y坐标

int ind = 1;//x，y数组的下标

long double len(int ax,int ay,int bx,int by)//算出两点之间的直线距离

{

return sqrt((bx - ax) \* (bx - ax) + (by - ay) \* (by - ay));

}

void floyd(int n)//n为图的顶点的个数

{

int i,j,k;

for(k = 1; k <= n; k++)

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

for(j = 1; j <= n; j++)

if(length[i][j] > length[i][k] + length[k][j])//从i经k到j的时间更短

{

length[i][j] = length[i][k] + length[k][j];//更新length[i][j]

}

}

int main()

{

int i,j,flag = 0;//flag标记是否输入的地铁站换到下一线路

double v1 = 10000.0 / 60;//步行的速度（m/min）

double v2 = 40000.0 / 60;//地铁的速度（m/min）

for(i = 0; i < 220; i++)//初始化

for(j = 0; j < 220; j++)

if(i == j)

length[i][j] = 0;

else

length[i][j] = INT\_MAX;

cin>>x[ind]>>y[ind];//输入起点

ind++;

cin>>x[ind]>>y[ind];//输入终点

ind++;

while(cin>>x[ind]>>y[ind])//读入所有站点坐标

{

if(x[ind] == -1 && y[ind] == -1)//代表换到下一线路

{

flag = 1;//标记

continue;

}

if(flag == 1)//前一线路的终点到此线路的起点需要步行

{

length[ind - 1][ind] = len(x[ind - 1],y[ind - 1],x[ind],y[ind]) / v1;//计算时间

length[ind][ind - 1] = length[ind - 1][ind];//无向图

flag = 0;//取消标记

ind++;

continue;

}

if(ind == 3)//起点到第一条线路的第一个站点也要步行

{

length[1][3] = len(x[1],y[1],x[3],y[3]) / v1;//计算时间

length[3][1] = length[1][3];

}

if(ind > 3)

{

length[ind - 1][ind] = len(x[ind - 1],y[ind - 1],x[ind],y[ind]) / v2;//计算地铁在两站点之间运行的时间（坐地铁）

length[ind][ind - 1] = length[ind - 1][ind];

}

ind++;

}

length[ind - 1][2] = len(x[ind - 1],y[ind - 1],x[2],y[2]) / v1;//计算时间（最后一条线路到终点需要步行）

length[2][ind - 1] = length[ind - 1][2];

for(i = 1; i < ind; i++)

for(j = 1; j < ind; j++)

if(length[i][j] == INT\_MAX)//地铁未连通的点都可以通过步行到达

{

length[i][j] = len(x[i],y[i],x[j],y[j]) / v1;//计算时间

length[j][i] = length[i][j];//无向图

}

floyd(ind - 1);//弗洛伊德算法

cout<<(int)(length[1][2] + 0.5)<<endl;//结果要四舍五入

return 0;

}

## poj 2240

### 1 原题目

**套利**

|  |  |  |
| --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 65536K |
| **提交总数：** 34469 |  | **接受的：** 14203 |

**描述** [**http://poj.org/problem?id=2240**](http://poj.org/problem?id=2240)

套利是利用货币汇率差异将一种货币单位转换成一种以上相同货币单位。例如，假设1美元买入0.5英镑，1英镑买入10.0法国法郎，1法国法郎买入0.21美元。然后，通过转换货币，聪明的交易者可以从1美元开始，买入0.5 \* 10.0 \* 0.21 = 1.05美元，获利5％。  
您的工作是编写一个程序，该程序将货币汇率列表作为输入，然后确定是否可以进行套利。

**输入值**

输入将包含一个或多个测试用例。在每个测试用例的第一行中，有一个整数n（1 <= n <= 30），代表不同货币的数量。接下来的n行分别包含一种货币的名称。名称中不会出现空格。下一行包含一个整数m，代表要遵循的表的长度。最后m行分别包含源货币的名称ci，代表ci到cj的汇率的实数rij和目标货币的名称cj。没有出现在表中的交换是不可能的。  
测试用例之间用空白行隔开。输入以n的零（0）值终止。

**输出量**

对于每个测试用例，分别以“案例：是”和“案例：否”的格式打印一行，说明是否可以套利。

**样本输入**

3

USDollar

BritishPound

FrenchFranc

3

USDollar 0.5 BritishPound

BritishPound 10.0 FrenchFranc

FrenchFranc 0.21 USDollar

3

USDollar

BritishPound

FrenchFranc

6

USDollar 0.5 BritishPound

USDollar 4.9 FrenchFranc

BritishPound 10.0 FrenchFranc

BritishPound 1.99 USDollar

FrenchFranc 0.09 BritishPound

FrenchFranc 0.19 USDollar

0

**样本输出**

Case 1: Yes

Case 2: No

### 2 基础知识

弗洛伊德算法(floyd)（解法一用到）

### 3 解法

#### 3.1 解法1（G++已过）

此题要注意可能出现a 1.1 a这种汇率的数据，就是某种钱对自身的汇率不同（这坑数据，纯纯恶心人）。然后就可以利用floyd算法来解题，三重for循环来解题（传递性，就是钱换钱），保证所有情况都能遍历到，最后只需要判断钱自身汇率是否大于1就可。（大于1说明可以套利）

本人代码：

|  |
| --- |
| #include <iostream>  #include<algorithm>  #include <cstring>  #include <cstdio>  #include <cstdlib>  #include <cmath>  #include <vector>  #include <queue>  #include <stack>  #include <map>  #include <set>  using namespace std;  //记录每种钱的编号  map<string,int> mp;  //记录i到j编号钱的汇率  double len[1005][1005];  void floyd(int n)  {      for(int k = 1; k <= n; k++)          for(int i = 1; i <= n; i++)              for(int j = 1; j <= n; j++)                  if(len[i][j] < len[i][k] \* len[k][j])//从i经k到j的钱数换的更多                  {                      len[i][j] = len[i][k] \* len[k][j];//更新len[i][j]                  }  }  int main()  {      int n,m;      double z;      string s1,s2;      int cas = 1;      int bit;      while (scanf("%d",&n) != EOF)      {          if (n == 0)              break;          mp.clear();          bit = 1;          for (int i = 0; i < n; i++)          {              cin>>s1;              mp[s1] = bit++;          }          scanf("%d",&m);          memset(len,0,sizeof(len));          for (int i = 0; i < m; i++)          {              cin>>s1>>z>>s2;              len[mp[s1]][mp[s2]] = z;          }          floyd(n);          int flag = 1;          for (int i = 1; i <= n; i++)          {              if (len[i][i] > 1)              {                  flag = 0;                  printf("Case %d: Yes\n",cas++);                  break;              }          }          if (flag)              printf("Case %d: No\n",cas++);      }      return 0;  } |

## poj 3087

### 1 原题目

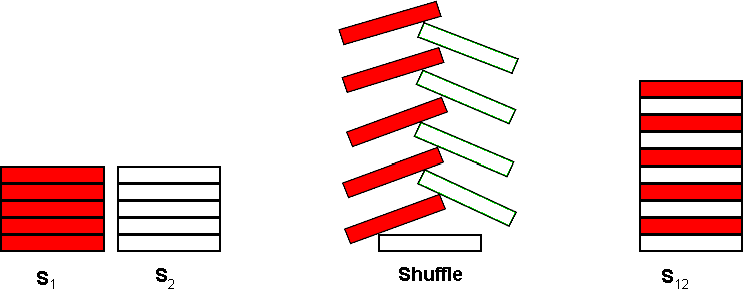
**随机播放**

|  |  |  |
| --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 65536K |
| **提交总数：** 19236 |  | **接受的：** 8686 |

**描述** <http://poj.org/problem?id=3087>

对于扑克玩家来说，在扑克桌上的常见消遣是洗牌。改组筹码通过从两叠扑克筹码**S 1**和**S 2开始进行**，每叠包含***C***筹码。每个堆叠可以包含几种不同颜色的芯片。

如下所示，对于***C*** = 5，通过将来自**S 1**的芯片与来自**S 2**的芯片交织来执行实际的混洗操作：



单个结果堆栈**S 12**包含2 \* ***C***芯片。的最底部芯片**小号12**是从最底部芯片**小号2**。在该芯片的顶部，是**S 1中**最底部的芯片。交织过程继续从**S 2**的底部取出第二个码片并将其放在**S 12上**，然后从**S 1**的底部取出第二个码片，依此类推，直到**S 1**的最上面的码片放在**S的**顶部**12**。

随机操作之后，通过从**S 12中**取出最低的***C***筹码以形成新的**S 1**并从**S 12中**取出最高的***C***筹码以形成新的**S 2**，将**S 12**分成2个新堆栈。然后可以重复执行随机操作以形成新的**S 12**。

对于此问题，您将编写一个程序来确定是否可以通过将两个堆栈改组几次来形成特定的结果堆栈**S 12**。

**输入项**

输入的第一行包含一个整数***Ñ***，（1≤ ***Ñ*** ≤1000），它是数据集的是后续的数量。

每个数据集由四行输入组成。数据集指定的第一行的整数***Ç***，（1≤ ***ç*** ≤100），这是在每个初始栈芯片（数**小号1**和**小号2**）。每个数据集的第二行指定堆栈**S 1**中每个***C***芯片的颜色，从最底端的芯片开始。每个数据集的第三行指定堆栈**S 2**中每个***C***芯片的颜色，从最底端的芯片开始。颜色表示为单个大写字母（**A**到**H**）。芯片颜色之间没有空白或分隔符。每个数据集的第四行包含2 \* ***C个***大写字母（**A**到**H**），代表将**S 1**和**S 2**改组零次或多次的预期结果的颜色。首先指定最底端的芯片颜色。

**输出量**

每个数据集的输出由单行组成，该行显示数据集编号（1到***N***），一个空格和一个整数值，该整数值是获得所需结果堆栈所需的最小随机操作数。如果使用数据集的输入无法达到所需的结果，则将随机操作次数显示为负1（**-1**）。

**样本输入**

2

4

AHAH

HAHA

HHAAAAHH

3

CDE

CDE

EEDDCC

**样本输出**

1 2

2 -1

### 2 题目分析

先给你两个长度一样的初始状态字符串s1，s2和一个大小为两倍的最终状态的s12，按照s2第一个先放在s12第一个，再依次是s1,s2,s1…，排完后再按照长度前一半是s1新状态，后一半是s2新状态， 重复以上操作。看得到s12是否有和 最初给的最终状态的s12相同的，有输出步数，没有就输出-1。

### 3 基础知识

#### 3.1数据结构

无。

#### 3.2算法

无。

### 4 解法

#### 4.1 解法1（G++已过）

题目做起来不难，难点可能在如何判断输出-1，我们只需要将每次洗牌后的字符串用map<string,int>book记录起来（记录成1），如果某次洗牌后发现此结果已经被标记过了，那就证明无法找到最后的结果，就输出-1。

本人代码：

#include <iostream>

#include <cstring>

#include <map>

using namespace std;

int main()

{

int n;

cin>>n;

for(int z = 1; z <= n; z++)

{

char s1[102] = {'0'},s2[102] = {'0'},s[204] = {'0'},res[204] = {'0'};

int c,i,num = 0;//num记录洗牌次数

map<string,int>book;

cin>>c>>s1>>s2>>res;

while(true)

{

int m = 0;

for(i = 0; i < c; i++)//洗牌

{

s[m++] = s2[i];

s[m++] = s1[i];

}

s[m] = '\0';

//判断此时洗牌后的情况之前是否出现过，出现过就说明找不到最后题意要的结果

if(book[s] == 1)

{

cout<<z<<' '<<-1<<endl;

break;

}

num++;//洗牌次数加1

book[s] = 1;

if(strcmp(s,res) == 0)//如果和题意的结果一样

{

cout<<z<<' '<<num<<endl;

break;

}

m = 0;

for(i = 0; i < c; i++)//重新分牌来进行下一次的洗牌

s1[m++] = s[i];

s1[m] = '\0';

m = 0;

for(; i < 2 \* c; i++)

s2[m++] = s[i];

s2[m] = '\0';

}

}

return 0;

}

## poj 2236

### 1 原题目

**无线网络**

|  |  |  |
| --- | --- | --- |
| **时限：** 10000MS |  | **内存限制：** 65536K |
| **提交总数：** 49908 |  | **接受：** 20431 |

**描述** <http://poj.org/problem?id=2236>

东南亚发生地震。亚洲合作医疗队（ACM）已与膝上计算机建立了无线网络，但由于意外的余震袭击，网络中的所有计算机都被破坏了。电脑被一一修复，网络逐渐恢复工作。由于硬件限制，每台计算机只能直接与不超过d米的计算机通信。但是，每台计算机都可以视为其他两台计算机之间通信的中介，也就是说，如果计算机A和计算机B可以直接通信，或者有计算机C可以同时与A和B通信，则计算机A和计算机B可以通信。 B．  
  
在修复网络的过程中，工作人员可以随时执行两种操作：修复计算机，或测试两台计算机是否可以通信。您的工作是回答所有测试操作。

**输入值**

第一行包含两个整数N和d（1 <= N <= 1001，0 <= d <= 20000）。N是计算机的数量，从1到N，而D是两台计算机可以直接通信的最大距离。在接下来的N行中，每行包含两个整数xi，yi（0 <= xi，yi <= 10000），这是N个计算机的坐标。从第（N + 1）行到输入的末尾，有一些操作，这些操作是一个接一个地执行的。每行包含以下两种格式之一的操作：  
1.“ O p”（1 <= p <= N），表示修复计算机p。  
2.“ S p q”（1 <= p，q <= N），表示测试计算机p和q是否可以通信。  
  
输入不会超过300000行。

**输出量**

对于每项测试操作，如果两台计算机可以通信，则打印“ SUCCESS”，否则，则打印“ FAIL”。

**样本输入**

4 1

0 1

0 2

0 3

0 4

O 1

O 2

O 4

S 1 4

O 3

S 1 4

**样本输出**

FAIL

SUCCESS

### 2 题目分析

题目给你一个d，两个计算机距离小于等于d即视为可以通信，起初计算机是全坏的。O代表修计算机，S代表问你两个编号的计算机是否可以联系？（两个计算机之间可以通过其他计算机相联系）

### 3 基础知识

#### 3.1数据结构

并查集（解法一用到）

#### 3.2算法

无。

### 4 解法

#### 4.1 解法1（G++已过）

就是并查集的运用，没什么好讲的。

本人代码：

#include <iostream>

#include <cstring>

#include <cmath>

using namespace std;

struct mp

{

int x,y;

} node[1005];//存储计算机的坐标

int vis[1005];//将修好的点放进此数组中

int father[1005];//记录父节点

int Find(int x)//查询x的根节点并路径压缩

{

if(father[x] != x)

father[x] = Find(father[x]);

return father[x];

}

void Union(int x,int y)//合并x和y的集合

{

int rx,ry;

rx=Find(x);

ry=Find(y);

if(rx!=ry)

father[rx]=ry;

}

double length(int x,int y)

{

return sqrt((node[x].x - node[y].x) \* (node[x].x - node[y].x) + (node[x].y - node[y].y) \* (node[x].y - node[y].y));

}

int main()//此题考查并查集的知识

{

int N,d,i,m,n,bit = 0;

cin>>N>>d;

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

for(i = 1; i <= N; i++)

{

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

father[i] = i;//并查集中记录父节点的数组初始化

}

char c;

while(cin>>c)

{

if(c == 'O')

{

cin>>m;

vis[bit++] = m;

if(bit >= 2)

{

for(i = 0; i < bit - 1; i++)

{

if(length(vis[i],m) <= d)//如果两个点之间的距离小于d

Union(vis[i],m);//进行连接

}

}

}

if(c == 'S')

{

cin>>m>>n;

if(Find(m) == Find(n))//如何两点的父亲是同一个点

cout<<"SUCCESS"<<endl;//就是成功

else

cout<<"FAIL"<<endl;

}

}

return 0;

}

## poj 1611

### 1 原题目

**嫌疑犯**

|  |  |  |
| --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 20000K |
| **提交总数：** 61216 |  | **接受：** 28903 |

**描述** <http://poj.org/problem?id=1611>

严重急性呼吸道综合症（SARS）是一种病因不明的非典型肺炎，在2003年3月中旬被认为是全球性威胁。为了最大程度地减少向他人的传播，最好的策略是将嫌疑犯与其他人分开。  
在不蔓延疾病大学（NSYSU）中，有很多学生团体。同一组中的学生经常互相交流，一个学生可以加入多个组。为了防止可能的SARS传播，NSYSU收集所有学生团体的成员列表，并在其标准操作程序（SOP）中制定以下规则。  
一旦组中的某个成员成为可疑对象，该组中的所有成员都将成为可疑对象。  
但是，他们发现，当一个学生被确认为犯罪嫌疑人时，要识别所有犯罪嫌疑人并不容易。您的工作是编写一个找到所有嫌疑犯的程序。

**输入值**

输入文件包含几种情况。每个测试用例都以一行中的两个整数n和m开头，其中n是学生数，m是组数。您可以假设0 <n <= 30000并且0 <= m <=500。每个学生都用0到n-1之间的唯一整数编号，并且最初在所有情况下，学生0都被视为犯罪嫌疑人。该行之后是组的m个成员列表，每组一行。每行以一个整数k开头，该整数k代表组中成员的数量。在成员数之后，有k个整数表示该组中的学生。一行中的所有整数至少间隔一个空格。  
n = 0和m = 0的情况表示输入的结尾，无需处理。

**输出量**

对于每种情况，在一行中输出可疑事件的数量。

**样本输入**

100 4

2 1 2

5 10 13 11 12 14

2 0 1

2 99 2

200 2

1 5

5 1 2 3 4 5

1 0

0 0

**样本输出**

4

1

1

### 2 题目分析

让你判断疾病的嫌疑人有多少（同学们分成不同的组，组里有一个嫌疑人的话那么组中其他同学全是嫌疑人），初始的时候只有0号是嫌疑人。

### 3 基础知识

#### 3.1数据结构

并查集（解法一用到）

#### 3.2算法

无。

### 4 解法

#### 4.1 解法1（C++已过）

就是并查集的运用，没什么好讲的。

本人代码：

#include <iostream>

using namespace std;

int father[30005];//记录父节点

int Find(int x)//查询x的根节点并路径压缩

{

if(father[x] != x)

father[x] = Find(father[x]);

return father[x];

}

void Union(int x,int y)//合并x和y的集合

{

int rx,ry;

rx=Find(x);

ry=Find(y);

if(rx!=ry)

father[rx]=ry;

}

int main()//考察并查集的知识

{

int n,m,i,j,k,d,p;

while(cin>>n>>m)

{

if(n == 0 && m == 0)

break;

if(m == 0)

cout<<1<<endl;

else

{

for(i = 0; i < n; i++)//数组初始化

father[i] = i;

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

{

cin>>k>>d;

for(j = 0; j < k - 1; j++)

{

cin>>p;

Union(d,p);

}

}

int num = 0;

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

if(Find(i) == Find(0))//查询是否在同一个集合中

num++;

cout<<++num<<endl;

}

}

return 0;

}

## poj 1182

### 1 原题目

**食物链**

|  |  |  |
| --- | --- | --- |
| **Time Limit:** 1000MS |  | **Memory Limit:** 10000K |
| **Total Submissions:** 112756 |  | **Accepted:** 34240 |

**Description** <http://poj.org/problem?id=1182>

动物王国中有三类动物A,B,C，这三类动物的食物链构成了有趣的环形。A吃B， B吃C，C吃A。  
现有N个动物，以1－N编号。每个动物都是A,B,C中的一种，但是我们并不知道它到底是哪一种。  
有人用两种说法对这N个动物所构成的食物链关系进行描述：  
第一种说法是"1 X Y"，表示X和Y是同类。  
第二种说法是"2 X Y"，表示X吃Y。  
此人对N个动物，用上述两种说法，一句接一句地说出K句话，这K句话有的是真的，有的是假的。当一句话满足下列三条之一时，这句话就是假话，否则就是真话。  
1） 当前的话与前面的某些真的话冲突，就是假话；  
2） 当前的话中X或Y比N大，就是假话；  
3） 当前的话表示X吃X，就是假话。  
你的任务是根据给定的N（1 <= N <= 50,000）和K句话（0 <= K <= 100,000），输出假话的总数。

**Input**

第一行是两个整数N和K，以一个空格分隔。  
以下K行每行是三个正整数 D，X，Y，两数之间用一个空格隔开，其中D表示说法的种类。  
若D=1，则表示X和Y是同类。  
若D=2，则表示X吃Y。

**Output**

只有一个整数，表示假话的数目。

**Sample Input**

100 7

1 101 1

2 1 2

2 2 3

2 3 3

1 1 3

2 3 1

1 5 5

**Sample Output**

3

### 2 基础知识

#### 2.1数据结构

带权并查集（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C++已过）

参考链接：<https://blog.csdn.net/niushuai666/article/details/6981689>

个人感觉就是主要就是理清向量关系，通过向量关系推出一些式子。

本人代码：

#include <iostream>

#include<cstdio>

using namespace std;

int father[50005];//记录当前节点的父节点

int sum[50005];//记录当前节点到父节点的关系

int Find(int x)//查询x的根节点，路径压缩，更新关系

{

if(father[x] != x)

{

int temp = father[x];

father[x] = Find(father[x]);

sum[x] = (sum[x] + sum[temp]) % 3;

}

return father[x];

}

int main()

{

int N,K,d,x,y,i,num = 0;

scanf("%d %d",&N,&K);

for(i = 0; i <= N; i++)//数组初始化

{

father[i] = i;

sum[i] = 0;

}

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

{

scanf("%d %d %d",&d,&x,&y);

if(x > N || y > N)//第二种情况的假话

{

num++;

continue;

}

if(d == 2 && x == y)//第三种情况的假话

{

num++;

continue;

}

int rx = Find(x);

int ry = Find(y);

if(rx != ry)//集合合并

{

father[rx] = ry;

sum[rx] = (d - 1 + sum[y] - sum[x] + 3) % 3;

}

else//第一种情况的假话

{

if(d == 1)

{

if(sum[x] != sum[y])//x和y是同类，根节点也一样，所以关系应该相同

{

num++;

continue;

}

}

if(d == 2)

{

if((sum[x] - sum[y] + 3) % 3 != d - 1)//x吃y，根节点一样，通过向量关系推出此式

{

num++;

continue;

}

}

}

}

printf("%d\n",num);

return 0;

}

## poj 3304

### 1 原题目

**区隔**

|  |  |  |
| --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 65536K |
| **提交总数：** 21047 |  | **接受的：** 6678 |

**描述** <http://poj.org/problem?id=3304>

给定二维空间中的*n个*段，编写一个程序，该程序确定是否存在一条线，以便在将这些段投影到其上之后，所有投影的段至少有一个共同点。

**输入值**

输入从显示测试用例数量的数字*T*开头，然后是*T个*测试用例。每个测试例就开始与包含一个正整数的线*Ñ* ≤100示出的段的数量。之后，*紧接着的n*行包含四个实数*x*1*y*1*x*2*y*2，其中（*x*1，*y*1）和（*x*2，*y*2）是段之一的两个端点的坐标。

**输出量**

对于每个测试用例，如果存在具有所需属性的行，则程序必须输出“是！”，并且必须输出“否！” 除此以外。如果| |，则必须假定两个浮点数*a*和*b*相等。*a* - *b* | <10 -8。

**样本输入**

3

2

1.0 2.0 3.0 4.0

4.0 5.0 6.0 7.0

3

0.0 0.0 0.0 1.0

0.0 1.0 0.0 2.0

1.0 1.0 2.0 1.0

3

0.0 0.0 0.0 1.0

0.0 2.0 0.0 3.0

1.0 1.0 2.0 1.0

**样本输出**

Yes!

Yes!

No!

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

计算几何——判断直线与线段相交问题

### 3 解法

#### 3.1 解法1（C++已过）

题目问你的问题可以转化为是否存在一条直线与所给的线段都相交，由于线段数量不多，所以我们可以枚举出来所有线段的两个端点组成的一条直线，判断直线是否与所有的线段相交。（要注意题目所说的如果两点间距小于10 -8则视为同一个点）。

线段相交判断的讲解链接：<https://www.cnblogs.com/tuyang1129/p/9390376.html>

本人代码：

#include <iostream>

#include <cstdio>

#include <cmath>

using namespace std;

int m;

const double n = 1e-8;

struct segment//存储线段的端点坐标

{

double x1,y1,x2,y2;

} s[110];

double dis(double x1,double y1,double x2,double y2)//两点间的距离

{

return sqrt((x2 - x1) \* (x2 - x1) + (y2 - y1) \* (y2 - y1));

}

double cross(double a,double b,double c,double d,double e,double f)//用向量叉乘判断直线与线段是否相交

{

return (a - c) \* (b - f) - (a - e) \* (b - d);

}

bool judge(double x1,double y1,double x2,double y2)

{

if(dis(x1,y1,x2,y2) < n)//如果是同一个点

return false;

for(int i = 0; i < m; i++)//传进来的直线要与所有线段相交

if(cross(x1,y1,x2,y2,s[i].x1,s[i].y1) \* cross(x1,y1,x2,y2,s[i].x2,s[i].y2) > 0)

return false;

return true;

}

int main()

{

int T,flag;

scanf("%d",&T);

while(T--)

{

int i,j;

scanf("%d",&m);

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

scanf("%lf %lf %lf %lf",&s[i].x1,&s[i].y1,&s[i].x2,&s[i].y2);

if(m == 1)//只有一条线段一定可以

{

printf("Yes!\n");

continue;

}

flag = 0;

for(i = 0; i < m; i++)//枚举出来所有端点构成的直线

{

for(j = i + 1; j < m; j++)

{

if(judge(s[i].x1,s[i].y1,s[j].x1,s[j].y1)||

judge(s[i].x1,s[i].y1,s[j].x2,s[j].y2)||

judge(s[i].x2,s[i].y2,s[j].x1,s[j].y1)||

judge(s[i].x2,s[i].y2,s[j].x2,s[j].y2))

{

flag = 1;

break;

}

}

if(flag == 1)

break;

}

if(flag == 1)

printf("Yes!\n");

else

printf("No!\n");

}

return 0;

}

## poj 2653

### 1 原题目

**捡起棍棒**

|  |  |  |
| --- | --- | --- |
| **时限：** 3000MS |  | **内存限制：** 65536K |
| **提交总数：** 16602 |  | **接受：** 6292 |

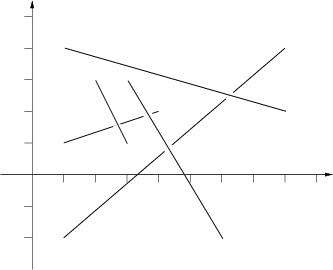
**描述** <http://poj.org/problem?id=2653>

斯坦有n种不同长度的木棍。他随意地一次将它们扔在地板上。投掷完成后，斯坦尝试找到最上面的棍棒，即这些棍棒上没有棍棒。斯坦（Stan）注意到最后扔出的棍子总是在最上面，但他想知道所有上面的棍子。Stan棍非常非常细，因此可以忽略其厚度。

**输入值**

输入包含多种情况。每种情况下的数据均以1 <= n <= 100000（此情况下的棒数）开头。接下来的n行每行包含四个数字，这些数字是一根棍子端点的平面坐标。棍子按Stan投掷的顺序列出。您可以假设顶杆不超过1000个。输入以n = 0的情况结束。这种情况不应该被处理。

**输出量**

对于每种输入情况，以示例中给定的格式打印输出的每一行，列出最上面的棍子。顶部棍棒应按其投掷顺序列出。  
  
下图为输入的第一种情况。****

**样本输入**

5

1 1 4 2

2 3 3 1

1 -2.0 8 4

1 4 8 2

3 3 6 -2.0

3

0 0 1 1

1 0 2 1

2 0 3 1

0

**样本输出**

Top sticks: 2, 4, 5.

Top sticks: 1, 2, 3.

**暗示**

大量输入，建议使用scanf。

### 2 基础知识

#### 2.1数据结构

无。

#### 2.2算法

计算几何——判断线段与线段相交问题

### 3 解法

#### 3.1 解法1（C++已过）

这道题就是求有多少条线段没有与编号大于它的线段相交，所以可以利用向量叉乘的知识来判断两个线段是否相交（这是本题重点）。剩下的就是暴力for循环来依次进行判断每一条线段是否与后面的线段相交。

本人代码：

#include <iostream>

#include <stdio.h>

using namespace std;

struct segment//存储线段端点坐标

{

double x1,y1,x2,y2;

} s[100005];

double cross(double a,double b,double c,double d,double e,double f)//用向量叉乘判断线段与线段是否相交

{

return (a - c) \* (b - f) - (a - e) \* (b - d);

}

int main()

{

int n;

while(scanf("%d",&n) != EOF)

{

if(n == 0)

break;

int i,j,res[1005] = {0},k = 0,flag;//res存储符合条件棍棒的编号

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

scanf("%lf %lf %lf %lf",&s[i].x1,&s[i].y1,&s[i].x2,&s[i].y2);

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

{

flag = 0;

for(j = i + 1; j <= n; j++)

{

if(cross(s[i].x1,s[i].y1,s[i].x2,s[i].y2,s[j].x1,s[j].y1) \* cross(s[i].x1,s[i].y1,s[i].x2,s[i].y2,s[j].x2,s[j].y2) < 0&&

cross(s[j].x1,s[j].y1,s[j].x2,s[j].y2,s[i].x1,s[i].y1) \* cross(s[j].x1,s[j].y1,s[j].x2,s[j].y2,s[i].x2,s[i].y2) < 0)//满足此条件就是两个线段相交

{

flag = 1;

break;

}

}

if(flag == 0)

res[k++] = i;

}

printf("Top sticks: ");

for(i = 0; i < k - 1; i++)

printf("%d, ",res[i]);

printf("%d.\n",res[k - 1]);

}

return 0;

}

## poj 3468

### 1 原题目

**一个整数的简单问题**

|  |  |  |
| --- | --- | --- |
| **时限：** 5000MS |  | **内存限制：** 131072K |
| **提交总数：** 178179 |  | **接受：** 54775 |
| **案件**期限**：** 2000MS | | |

**描述** <http://poj.org/problem?id=3468>

你有*ñ*整数，*一个*1，*一*2，...，*一ñ*。您需要处理两种操作。一种操作类型是在给定间隔内向每个数字添加一些给定数字。另一种是在给定的时间间隔内求和。

**输入值**

第一行包含两个数*Ñ*和*Q*。1≤ *Ñ*，*Q* ≤100000  
的第二行包含*Ñ*号码，的初始值*阿*1，*阿*2，...，*甲Ñ*。-1000000000≤ *甲我* ≤1000000000.  
每个下一个的*Q*线表示的操作。  
“ C *a b c* ”表示将*c*加到*A a*，*A a*+1，...，*A b中的*每*一个上*。-10000≤*Ç* ≤10000  
“Q *一* *b* ”的意思查询的总和*甲一个*，*甲一个*1，...，*甲b*。

**输出量**

您需要依次回答所有*Q*命令。一行回答。

**样本输入**

10 5

1 2 3 4 5 6 7 8 9 10

Q 4 4

Q 1 10

Q 2 4

C 3 6 3

Q 2 4

**样本输出**

4

55

9

15

**暗示**

总和可能超出32位整数的范围。

### 2 基础知识

#### 2.1数据结构

树状数组（解法一用到）

#### 2.2算法

无。

### 3 解法

#### 3.1 解法1（C++已过）

树状数组的区间更新区间查询的模板应用。

推荐树状数组的讲解链接：<https://www.cnblogs.com/xenny/p/9739600.html>

本人代码：

#include <iostream>

#include<algorithm>

#include <cstdio>

#include <cstdlib>

using namespace std;

typedef long long ll;

int N;//数组元素个数

ll a[100005];//原数组

ll c[100005];

ll c1[100005];

int lowbit(int x)

{

return x&(-x);

}

ll getsum(int i)//求前缀和（即a[1]到a[i]的和）

{

ll res = 0;

int x = i;

while(i > 0)

{

res += x \* c[i] - c1[i];

i -= lowbit(i);

}

return res;

}

void updata(int i,ll k)//在i位置加上k（此函数只更改树状数组的值，若要更改原数组的值需要在主函数中实现）

{

int x = i;//因为x不变，所以得先保存i值

while(i <= N)

{

c[i] += k;

c1[i] += k \* (x - 1);

i += lowbit(i);

}

}

int main()

{

int Q,i;

scanf("%d %d",&N,&Q);

for(i = 1; i <= N; i++)

{

scanf("%I64d",&a[i]);

updata(i,a[i] - a[i - 1]);//输入初值的时候，也相当于更新了值

}

char ch;

int x,y,z;

while(Q--)

{

getchar();

scanf("%c",&ch);

if(ch == 'Q')

{

scanf("%d %d",&x,&y);

printf("%I64d\n",getsum(y) - getsum(x - 1));//求[x,y]区间和

}

if(ch == 'C')

{

scanf("%d %d %d",&x,&y,&z);

//[x,y]区间内加上z

updata(x,z);//a[x] - a[x-1]增加z

updata(y + 1,-z);//a[y+1] - a[y]减少z

}

}

return 0;

}

## poj 1401

### 1 原题目

**阶乘**

|  |  |  |
| --- | --- | --- |
| **时限：** 1500MS |  | **内存限制：** 65536K |
| **提交总数：** 17965 |  | **接受：** 10857 |

**描述** <http://poj.org/problem?id=1401>

GSM网络最重要的部分是所谓的基站收发器（BTS）。这些收发器形成了一个称为单元的区域（此术语为蜂窝电话的名称），每部电话都以最强的信号连接到BTS（以简化的方式表示）。当然，防弹少年团需要引起注意，技术人员需要定期检查其功能。  
  
ACM技术人员最近面临一个非常有趣的问题。给定要访问的一组BTS，他们需要找到最短的路径来访问所有给定的点，然后返回到中心公司大楼。程序员花了几个月的时间研究这个问题，但没有结果。他们无法足够快地找到解决方案。长时间后，一位程序员在会议文章中发现了此问题。不幸的是，他发现该问题被称为“旅行商问题”，很难解决。如果我们要访问N个BTS，则可以按任意顺序访问它们，给我们N个！检查的可能性。表示该数字的函数称为阶乘，可以计算为乘积1.2.3.4 .... N。即使N相对较小，该数目也很高。  
  
程序员知道他们没有机会解决问题。但是，由于他们已经获得了政府的研究资助，因此他们需要继续他们的研究并至少取得一些成果。因此，他们开始研究阶乘函数的行为。  
  
例如，他们定义了函数Z。对于任何正整数N，Z（N）是数字N！的十进制形式末尾的零个数。他们注意到此功能永远不会降低。如果我们有两个数字N1 <N2，则Z（N1）<= Z（N2）。这是因为我们永远无法通过乘以任何正数来“丢失”任何尾随零。我们只能得到新的和新的零。函数Z非常有趣，因此我们需要一个可以有效确定其值的计算机程序。

**输入项**

输入的第一行上有一个正整数T。它代表要跟随的数字数量。然后是T行，每行仅包含一个正整数N，1 <= N <= 1000000000。

**输出量**

对于每个数字N，输出包含单个非负整数Z（N）的一行。

**样本输入**

6

3

60

100

1024

23456

8735373

**样本输出**

0

14

24

253

5861

2183837

### 2 基础知识

数学（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

题目让你算n！的结果末尾有多少0，我们知道10乘上任何数就会多一个0，而10是由2乘5来的，而我们只需要将所有数字的因子拆开来看，你会发现2的数量比5的数量多，所以我们只需要找有多少5，我们知道包含5这个因子的数字必须是5的倍数，所以我们只需要找到1-n这些数字中因子5的数量就是结果。那么问题就来了，5，10这种数字只包含一个因子5，而25包含2个因子5，所以就出来一个很妙的就是用n除i（看代码），可以计算出n以下的数字的数量，就比如50除5我们知道十个数字包含5，那么50除25就两个还包含5。大概就是这个意思。

本人代码：

|  |
| --- |
| #include <iostream>  #include<algorithm>  #include <cstring>  #include <cstdio>  #include <cstdlib>  #include <cmath>  using namespace std;  //求n！结果末尾0的个数就是求1-n中5这个因子的个数  int z(int n)  {  int ans = 0,i;  //n/i计算的是[1,n]中有多少数字是i的倍数（因子中包含i）  //就比如说1到50有10个数字包含因子5，2个数字因子包含25  //因为计算的是因子5的数量，5这种包含一次因子5，25包含两个因子5，所以这么计算  for(i = 5; i <= n; i \*= 5)  ans += n / i;  return ans;  }  int main()  {  int t,n;  scanf("%d",&t);  while(t--)  {  scanf("%d",&n);  printf("%d\n",z(n));  }  return 0;  } |

## poj 1328

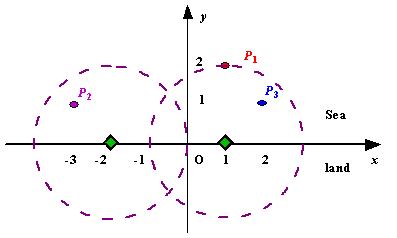
### 1 原题目

**雷达安装**

|  |  |  |
| --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 10000K |
| **提交总数：** 136202 |  | **接受：** 29943 |

**描述** <http://poj.org/problem?id=1328>

假设滑行是无限的直线。陆地在海岸的一侧，海洋在另一侧。每个小岛都是位于海边的一个地点。而且，位于海岸上的任何雷达装置只能覆盖d距离，因此，如果它们之间的距离最大为d，则可以用半径装置覆盖海中的一个岛。  
  
我们使用笛卡尔坐标系，定义了惯性运动为x轴。海洋一侧在x轴上方，陆地一侧在x轴上方。给定海洋中每个岛屿的位置，并确定雷达设备覆盖的距离，您的任务是编写一个程序，以找到覆盖所有岛屿的最少雷达设备数量。注意，岛的位置由其xy坐标表示。

  
图雷达输入示例

**输入项**

输入包含几个测试用例。每种情况的第一行都包含两个整数n（1 <= n <= 1000）和d，其中n是海洋中的岛屿数量，d是雷达设备的覆盖距离。这之后是n行，每行包含两个表示每个岛的位置坐标的整数。然后出现空白行以分隔个案。  
  
输入由包含零对的行终止

**输出量**

对于每个测试用例输出，一行由测试用例编号和所需的最少雷达安装数组成。“ -1”安装意味着在这种情况下没有解决方案。

**样本输入**

3 2

1 2

-3 1

2 1

1 2

0 2

0 0

**样本输出**

Case 1: 2

Case 2: 1

### 2 基础知识

贪心算法（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

根据题意可知，雷达要放在x轴上。所以我们可以先以小岛为圆心，d为半径，求得与x轴的两个交点。这样的目的是雷达只要在这两个交点之间就可以覆盖该小岛。我们将所有岛屿计算完毕后，在将交点组成的线段按照左端点大小从小到大进行排序（方便之后进行去重）

去重之后就是我们所求的雷达数量。

本人代码：

|  |
| --- |
| #include <iostream>  #include<algorithm>  #include <cstring>  #include <cstdio>  #include <cstdlib>  #include <cmath>  using namespace std;  int cmp(const void \*a,const void \*b)  {      const double p1 = \*(const double\*)a;      const double p2 = \*(const double\*)b;      if (p1 < p2)          return -1;      else if (p1 > p2)          return 1;      return 0;  }  int main()  {      int n,d,x,y;      int num = 0;      while(scanf("%d %d",&n,&d) != EOF)      {          if(n == 0 && d == 0)              break;          int flag = 0;          num++;          int i,res = -1;          double coor[1005][2],xx;          //以岛屿为圆心，d为半径做圆，求得与x轴的两个交点          for(i = 0; i < n; i++)          {              scanf("%d %d",&x,&y);              if(y > d)                  flag = 1;              xx = sqrt(d \* d - y \* y);              coor[i][0] = x - xx;              coor[i][1] = x + xx;          }          if(!flag)          {              qsort(coor,n,sizeof(coor[0]),cmp);//将二维数组按照头元素的大小来进行从小到大排序              res = 1;              i = 1;              double right = coor[0][1];              //去掉重读区间并计算所需雷达数目              while(i < n)              {                  if(coor[i][1] < right)                  {                      right = coor[i][1];                  }                  if(coor[i][0] > right)                  {                      res++;                      right = coor[i][1];                  }                  i++;              }          }          printf("Case %d: %d\n",num,res);          getchar();      }      return 0;  } |

## poj 3974

### 1 原题目

**回文**

|  |  |  |
| --- | --- | --- |
| **时限：** 15000MS |  | **内存限制：** 65536K |
| **提交总数：** 20081 |  | **接受：** 7666 |

**描述** <http://poj.org/problem?id=3974>

当教授问学生一个简单的问题时，聪明的计算机科学系学生安迪（Andy）正在参加算法课：“您能提出一种有效的算法来查找字符串中最大回文的长度吗？”  
  
如果字符串前后读取相同的字符串，则称其为回文，例如“女士”是回文，而“ acm”则不是。  
  
学生们意识到这是一个经典的问题，但是没有比遍历所有子字符串并检查它们是否是回文法更好的解决方案了，很明显，在一段时间后，Andy举起手来，该算法根本没有效率。说“好吧，我有一个更好的算法”，在他开始解释自己的想法之前，他停了片刻，然后说：“嗯，我  
  
如果您认为自己知道Andy的最终解决方案，那就证明一下！给定一个最多包含1000000个字符的字符串，请查找并打印该字符串内最大回文的长度。

**输入项**

您的程序最多将在30个测试用例上进行测试，每个测试用例本身在一行中以最多1000000个小写字符的字符串形式给出。输入以以字符串“ END”开头的行终止（为清楚起见用引号引起来）。

**输出量**

对于输入中的每个测试案例，请打印测试案例编号和最大回文的长度。

**样本输入**

abcbabcbabcba

bacababaaaab

END

**样本输出**

Case 1: 13

Case 2: 6

### 2 基础知识

Manacher马拉车算法（解法一用到）

### 3 解法

#### 3.1 解法1（C++已过）

马拉车算法的模板题目。

关于此算法参考参考以下两个链接：

1. <https://blog.csdn.net/Form_/article/details/79766986>

2. <https://www.cnblogs.com/czsharecode/p/9705358.html>

本人代码：

|  |
| --- |
| #include <iostream>  #include<algorithm>  #include <cstring>  #include <cstdio>  #include <cstdlib>  #include <cmath>  #include <vector>  using namespace std;  #define MAXN 1000005  char s[MAXN];  char t[MAXN \* 2];  int Manacher(char s[],int len)  {      int l = 0;      //开头加'$'是为了防止越界      t[l++] = '$';      t[l++] = '#';      //在原字符串开头结尾以及字符之间插入一个字符(未在串中出现过的)      for(int i = 0; i < len; i++)      {          t[l++] = s[i];          t[l++] = '#';      }      t[l++] = '\0';      //p[i] - 1就是t串中以i为中点的回文长度      vector<int> p(MAXN \* 2,0);      //mx(某回文串延伸到的最右边下标),id(mx所属回文串中心下标)      int mx = 0,id = 0;      //bit（结果最大回文串中心下标）,maxlen（最大回文长度）      int maxlen = 0;      for(int i = 1; i < l; i++)      {          //算法核心          p[i] = mx > i ? min(p[2 \* id - i],mx - i) : 1;          //一个个进行比较          //t数组最左边 = '$',最右边 = '\0'，无需判断边界          while(t[i + p[i]] == t[i - p[i]])              p[i]++;          //当t[i]匹配的 右边界超过mx时,mx和id就更新          if(i + p[i] > mx)          {              mx = i + p[i];              id = i;          }          //更新结果数据          if(p[i] > maxlen)          {              maxlen = p[i];          }      }      return maxlen - 1;  }  int main()  {      int sim = 1;      while(scanf("%s",s) != EOF)      {          if(s[0] == 'E')              break;          int len=strlen(s);          printf("Case %d: %d\n",sim++,Manacher(s,len));      }      return 0;  } |

## poj 3660

### 1 原题目

**牛比赛**

|  |  |  |
| --- | --- | --- |
| **时限：** 1000MS |  | **内存限制：** 65536K |
| **提交总数：** 21847 |  | **接受的：** 12134 |

**描述** [**http://poj.org/problem?id=3660**](http://poj.org/problem?id=3660)

*Ñ*（1≤ *ñ* ≤100）牛，方便地编号为1 .. *Ñ*，正在参与编程竞赛。众所周知，有些母牛的编码要好于其他。每头母牛都有一定的恒定技能等级，这在竞争者中是独一无二的。

比赛分几个头对头进行，每回合在两头母牛之间进行。如果牛*阿*具有更大的技能水平比牛*乙*（1≤*甲*≤ *Ñ* ; 1≤*乙*≤ *Ñ* ;*甲*≠*乙*），然后牛*甲*总是打牛*乙*。

农夫约翰试图按技能水平对母牛进行排名。给定一个列表的结果*中号*（1≤*中号*≤4500）二牛回合，确定奶牛他们的职级从结果可以精确地确定的数量。可以保证回合的结果不会矛盾。

**输入值**

\*第1行：两个以空格分隔的整数：*N*和*M*  
\*第2行*。M* +1：每行包含两个以空格分隔的整数，这些整数描述单个选手的竞争者和结果（第一个整数*A*是获胜者）一轮比赛：*A*和*B*

**输出量**

\*第1行：一个整数，代表可以确定其等级的母牛的数量

**样本输入**

5 5

4 3

4 2

3 2

1 2

2 5

**样本输出**

2

### 2 基础知识

### 3 解法

#### 3.1 解法1（C++已过）

一开始做这道题知道是找每个点比它大的和比它小的，然后二者和判断是否等于N-1，我是利用map和set容器做的，结果超时了（递归进行插入操作有太多得重复操作了）。可能是做题做傻了，三重for循环的暴力都忘了，哎，我看了下poj此题的讨论，都说是floyd，我看完全就是暴力，跟floyd算法扯不上关系啊（硬说有点关系难道是利用了弗洛伊德的传递性思想？？？？）。就是对每个点进行判断是否有比它大的点，有的话标记为1，这其中判断大小关系还可以利用点的传递性来判断，就是纯纯的暴力。

本人代码：

|  |
| --- |
| #include <iostream>  #include<algorithm>  #include <cstring>  #include <cstdio>  #include <cstdlib>  #include <cmath>  #include <vector>  #include <queue>  #include <stack>  #include <map>  #include <set>  using namespace std;  int N,M;  //v[i][j]表示i是否大于j  int v[110][110];  //直接暴力逐个判断大小关系  //借助中间值也是允许的，比如说，4>3，3>2，那么4>2  void floyd()  {      for(int k = 1; k <= N; k++)      {          for(int i = 1; i <= N; i++)          {              for(int j = 1; j <= N; j++)              {                  if(v[i][k] == 1 && v[k][j] == 1)                      v[i][j] = 1;              }          }      }      return ;  }  int main()  {      while (scanf("%d %d",&N,&M) != EOF)      {          memset(v,0,sizeof(v));          for (int i = 0; i < M; i++)          {              int x,y;              scanf("%d %d",&x,&y);              v[x][y] = 1;          }          floyd();          int res = 0;          for (int i = 1; i <= N; i++)          {              int temp = 0;              for (int j = 1; j <= N; j++)              {                  if (i == j)                      continue;                  //判断当前牛是否可以确定名次                  //如果当前牛比它大的和比它小的之和等于N-1那就可以确定名次                  //v[i][j]判断j是否比i小，v[j][i]判断j是否比i大，二者不可能同时为1（题意说不可能出现环）                  if (v[i][j] == 1 || v[j][i] == 1)                      temp++;              }              if (temp == N - 1)                  res++;          }          printf("%d\n",res);      }      return 0;  } |

## poj

### 1 原题目

### 2 基础知识

### 3 解法

#### 3.1 解法1（C++已过）

本人代码：

|  |
| --- |
|  |