**经验总结**

[§1 常见错误 4](#_Toc19379513)

[§2 常用算法 6](#_Toc19379514)

[2.1最大公因数和最小公倍数 6](#_Toc19379515)

[2.2连通无环无向图判断(并查集) 7](#_Toc19379516)

[2.3筛法获取素数 8](#_Toc19379517)

[2.4最小生成树(Kruskal+Prim) 10](#_Toc19379518)

[题目： 10](#_Toc19379519)

[Kruskal 10](#_Toc19379520)

[Prim 12](#_Toc19379521)

[2.5最短路径 13](#_Toc19379522)

[题目（变形）： 14](#_Toc19379523)

[题解： 14](#_Toc19379524)

[题目（普通）： 16](#_Toc19379525)

[题解: 17](#_Toc19379526)

[2.6拓扑排序 19](#_Toc19379527)

[2.7最优二叉搜索树 21](#_Toc19379528)

[2.8闰年判断 23](#_Toc19379529)

[2.9快速模除 23](#_Toc19379530)

[2.10网络流(EK算法) 24](#_Toc19379531)

[2.11费用流 26](#_Toc19379532)

[2.12记忆性搜索(矩阵中的最长路径) 29](#_Toc19379533)

[2.13两个日期间的差值（日期计算） 32](#_Toc19379534)

[§3 历年习题 34](#_Toc19379535)

[3.1有趣的数(动态规划) 34](#_Toc19379536)

[3.2无线网络(BFS+long long) 35](#_Toc19379537)

[3.3命令行选项(字符串处理) 37](#_Toc19379538)

[3.4无线网络(BFS+注意溢出) 40](#_Toc19379539)

[3.5最优配餐(带障碍的BFS) 42](#_Toc19379540)

[3.6集合竞价(暴力模拟) 46](#_Toc19379541)

[3.7 Z字形扫描(找规律扫描) 49](#_Toc19379542)

[3.8最优灌溉(最下生成树Kruskal) 51](#_Toc19379543)

[3.9节日(日期计算) 53](#_Toc19379544)

[3.10网络延时(树的直径+2次DFS) 55](#_Toc19379545)

[3.11模板生成系统(字符串查找+替换) 59](#_Toc19379546)

[3.12高速公路(强连通分量) 62](#_Toc19379547)

[3.12送货(欧拉路径+并查集判断连通性) 65](#_Toc19379548)

[3.13路径解析(字符串处理) 69](#_Toc19379549)

[3.14游戏(带时间维的BFS) 72](#_Toc19379550)

[3.15炉石传说(游戏模拟) 74](#_Toc19379551)

[3.16交通规划(变形最短路径Dijkstra) 80](#_Toc19379552)

[3.17压缩编码(类似矩阵链乘+动态规划) 82](#_Toc19379553)

[3.18地铁修建(Kruskal最小生成树) 84](#_Toc19379554)

[3.19通信网络(多次DFS) 87](#_Toc19379555)

[3.20行车路线(SPFA+FLOYD) 89](#_Toc19379556)

[3.21棋局评估(暴力DFS+博弈搜索) 92](#_Toc19379557)

[3.22再卖菜(回溯法) 96](#_Toc19379558)

[3.23 A1003 Emergency 100](#_Toc19379559)

[Dijkstra 100](#_Toc19379560)

[BellmanFord 101](#_Toc19379561)

[SPFA 103](#_Toc19379562)

[3.24 A1030 Travel Plan 107](#_Toc19379563)

[Dijkstra 107](#_Toc19379564)

[BellmanFord 109](#_Toc19379565)

[SPFA 111](#_Toc19379566)

# 常见错误

1.DFS、BFS时注意边界值，CCF中下标一般从1开始，因此判断边界时与1比较（矩阵）

2.并查集合并：root[roota]=rootb

3.输出要看全，有可能根据输入不同有不同的输出形式

4.不要写a[i]=b[++i]之类的代码

5.使用fgets、getline等函数时，注意输入之间的换行符

6.命令输入的顺序要注意，如果某一条命令没处理可能会改变整个命令序列的顺序，比如集合竞价一题。

7.使用while循环时不要忘记循环变量++

8.输入的两个数num1，num2不一定满足num1<num2!!!

9.数字问题可能会输入负数

10.解决图论问题时注意空图、自环、多边（树同理）,还要考虑**是否连通**

11.声明的变量注意范围（少了一个）

12.输出的字符串注意格式（空格、小数点等），因此最好CV

13.图论问题使用scanf和printf输入输出，速度极快

14.认真审题（比如传送门必须传送）

15.变量一定要**初始化**！！！

16.使用**long long**类型！！！！！！

17.再次强调！**小心溢出**

18.注意输出格式！！是否包含**前导0**、小数**保留位数**等

19.并查集的find操作

20.sort使用小于比较，priority\_queue使用大于比较

21.注意**时间格式的输出**，比如hh:mm:ss要用%02d控制输出格式

22.**进制转换需要特殊处理0**的情况，比如A+B转换为D进制，当A+B=0时要输出0.

23.字典序比较时小于号写成了大于号

24.注意输出格式，结尾**不要多了空格**！

25.反转链表时要考虑可能存在**无效节点**的情况，比如1→2→3→4，规定头节点为3，则1、2为无效节点。

26.注意**边界点**的测试！一定要保证所有的边界情况不会出错！

27.正数、正整数、整数一定要看清！

28.INT\_MIN，INT\_MAX定义在limits.h文件中

29.BFS遍历时使用增量矩阵可简化遍历过程

30.在使用BFS计算树的每一层的节点数是，注意不要遗漏最后一层！（即跳出循环后还要处理一下）

31.前序遍历即插入节点的顺序

32.double有效数字15位，long double有效数字18位，int有效数字10位，long long有效数字19位

33.采用mid = left+(right-left)/2 代替mid=(left + right)/2可以避免int溢出带来的问题

34.greater<>()定义在functional头文件中

35.**查找字符串a是否包含子串b,不是用 strA.find(strB) > 0 而是 strA.find(strB) != string:npos**

String:

查找操作

size type find( const basic string sstr, size _ type index ) ; 
: npos 
size type find( 
size type find ( 
Ét, KEh1ength) 
size_type ( 
string: :npos 
const char •str, size type index ) ; 
const char •str, size type index, size type length ; 
char ch. size _ type index ) ; 

string &insert(int p,const string &s); //在p位置插入字符串s

string &replace(int p, int n,const char \*s); //删除从p开始的n个字符，然后在p处插入串s

string &erase(int p, int n); //删除p开始的n个字符，返回修改后的字符串

string substr(int pos = 0,int n = npos) const; //返回pos开始的n个字符组成的字符串

void swap(string &s2); //交换当前字符串与s2的值

string &append(const char \*s); //把字符串s连接到当前字符串结尾

void push\_back(char c) //当前字符串尾部加一个字符c

const char \*data()const; //返回一个非null终止的c字符数组，**data():与c\_str()**类似，用于string转const char\*其中它返回的数组是不以空字符终止,

const char \*c\_str()const; //返回一个以null终止的c字符串，即c\_str()函数返回一个指向正规C字符串的指针, 内容与本string串相同,用于string转const char\*

strtok(char \*s,char \*t);//第一次分割传入s，之后的分割传入NULL，分割到结尾时返回NULL

2.scanf格式化读取

比如2009/11/07-12:12:12可以使用scanf("%d/%d/%d-%d:%d:%d", &year, &month, &day, &t1, &t2, &t3);进行读取。

# 常用算法

## 2.1最大公因数和最小公倍数

int GreatestCommonDivisor(int a, int b)//求a，b的最大公因数

{

    int t;

    if (a < b)

    {

        t = a; a = b; b = t;

    }

    while (b != 0)

    {

        t = a % b; a = b; b = t;

    }

    return a;

}

int LeastCommonMultiple(int a, int b,int c)//c是a、b的最大公因数

{

    int t = a \* b / c;

    return t;

}

unsigned long long powermod(unsigned long long a, unsigned long long b, unsigned long long mod)//快速模除

{

    unsigned long long sum = 1;

    while (b)

    {

        if (b & 1)//b是奇数

        {

            sum = (sum\*a) % mod;

        }

        b /= 2;

        a = (a\*a) % mod;

    }

    return sum;

}

## 2.2连通无环无向图判断(并查集)

无向图判断环：BFS

无向图判断连通性：并查集、DFS、树的性质，例如并查集判断：

#include<iostream>

#include<vector>

#include<algorithm>

#include<map>

using namespace std;

#define SIZE 100001

map<int, int> root;

int find(int x)

{

    if (root[x] == 0) return x;

    else return root[x] = find(root[x]);

}

int main(void)

{

    int a, b;

    while (true)

    {

        cin >> a >> b;

        if (a == -1 && b == -1) break;

        root.clear();

        int cnt = 0;

        while (true)//录入迷宫

        {

            if (a == 0 && b == 0) break;

            int roota, rootb;

            roota = find(a); rootb = find(b);

            if (roota == rootb)

            {

                cnt = 2; //存在环路

            }

            else

            {

                root[roota] = rootb;

            }

            cin >> a >> b;

        }

        if (cnt == 0)

        {

            map<int, int>::iterator it = root.begin();

            int pin = find(it->first);

            while (it != root.end())

            {

                int roota = find(it->first);

                if (roota != pin)

                {

                    cnt = 3; break;//不连通

                }

                it++;

            }

        }

        if (cnt >= 2)

        {

            cout << "No" << endl;

        }

        else cout << "Yes" << endl;

    }

}

## 2.3筛法获取素数

int prime[SIZE];

void getnum()

{

    for (int i = 0; i < SIZE; i++)//初始化

    {

        prime[i] = 1;

    }

    prime[0] = prime[1] = 0;

    for (int i = 2; i < SIZE; i++)

    {

        if (prime[i])

        {

            for (int j = i+i; j < SIZE; j += i)

            {

                prime[j] = 0;

            }

        }

    }

    return;

}

## 2.4最小生成树(Kruskal+Prim)

### 题目：

**Problem Description**

省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要能间接通过公路可达即可）。经过调查评估，得到的统计表中列出了有可能建设公路的若干条道路的成本。现请你编写程序，计算出全省畅通需要的最低成本。

**Input**

测试输入包含若干测试用例。每个测试用例的第1行给出评估的道路条数 N、村庄数目M ( < 100 )；随后的 N   
行对应村庄间道路的成本，每行给出一对正整数，分别是两个村庄的编号，以及此两村庄间道路的成本（也是正整数）。为简单起见，村庄从1到M编号。当N为0时，全部输入结束，相应的结果不要输出。

**Output**

对每个测试用例，在1行里输出全省畅通需要的最低成本。若统计数据不足以保证畅通，则输出“?”。

**Sample Input**

3 3

1 2 1

1 3 2

2 3 4

1 3

2 3 2

0 100

**Sample Output**

3

?

### Kruskal

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<vector>

#include<stdio.h>

#include<algorithm>

using namespace std;

struct Node

{

    int a;

    int b;

    int cost;

    Node() {};

    Node(int a1, int b1, int c1) :a(a1), b(b1), cost(c1) {};

    bool operator<(Node &p)

    {

        return cost < p.cost;

    }

};

int N, M;

int find(vector<int> &root, int x)

{

    if (root[x] == -1) return x;

    else return root[x] = find(root, root[x]);

}

int main(void)

{

    while (true)

    {

        cin >> N >> M;

        if (N == 0) break;

        vector<Node> Arc;

        for (int i = 0; i < N; i++)

        {

            int a, b, c;

            scanf("%d %d %d", &a, &b, &c);

            Arc.push\_back(Node(a, b, c));

        }

        sort(Arc.begin(), Arc.end());

        vector<int> root(M + 1, -1);

        int num = 0;//添加的边数

        int cost = 0;//添加的边的权值和

        for (int i = 0; i < Arc.size(); i++)

        {

            int roota, rootb;

            roota = find(root, Arc[i].a); rootb = find(root, Arc[i].b);

            if (roota == rootb) continue;

            cost += Arc[i].cost;

            root[roota] = rootb;

            num++;

            if (num == M - 1) break;

        }

        if (num == M - 1) cout << cost << endl;

        else cout << "?" << endl;//不存在最小生成树

    }

}

### Prim

#include<iostream>

#include<vector>

#include<queue>

#include<algorithm>

using namespace std;

struct ArcNode {//边

    int a;

    int b;

    int info;

    ArcNode(int a, int b, int info) :a(a), b(b), info(info) {};

    ArcNode() {};

};

struct cmp {

    bool operator()(ArcNode &a, ArcNode &b) {

        return a.info > b.info;

    }

};

bool cmp\_info(ArcNode &a, ArcNode &b)

{

    return a.info < b.info;

}

struct AdjNode {//邻接边节点

    int adj\_vex;

    int info;

    AdjNode(int v, int w) :adj\_vex(v), info(w) {};

};

    //输入：邻接表表示的无向图G(顶点从1开始编号)，顶点数vex\_num

    //操作：求图G的最小生成树

int Prim(vector<vector<AdjNode>> &G, int vex\_num) {

    vector<int> visited(vex\_num + 1, 0);//标为访问过的顶点

    priority\_queue<ArcNode, vector<ArcNode>, cmp> q;

    int i;

    for (i = 0; i < G[1].size(); i++) {//第一个顶点的邻接点入队

        q.push(ArcNode(1, G[1][i].adj\_vex, G[1][i].info));

    }

    visited[1] = true;

    //vector<ArcNode> ans;//最小生成树的边

    int num = 0;//边数

    int sum = 0;//权值和

    while (!q.empty())

    {

        ArcNode top;

        top = q.top(); q.pop();

        if (visited[top.b])

            continue;

        for (int i = 0; i < G[top.b].size(); i++) {//top.b未被访问的邻接点入队

            if (!visited[G[top.b][i].adj\_vex])

                q.push(ArcNode(top.b, G[top.b][i].adj\_vex, G[top.b][i].info));

        }

        visited[top.b] = true;

        //ans.push\_back(top);//添加最小生成树的边

        num++;

        sum += top.info;

        if (num == vex\_num - 1) break;

        //if (ans.size() == vex\_num - 1) break;

    }

    if (num == vex\_num - 1) return sum;

    else return -1;

    //if (ans.size() != vex\_num - 1) ans.clear();

}

int main(void)

{

    int VexNum, EdgeNum;

    while (true)

    {

        cin >> EdgeNum >> VexNum;

        if (EdgeNum == 0) break;

        vector<vector<AdjNode>> G(VexNum + 1);

        int u, v, w;

        while (EdgeNum--)

        {

            scanf("%d %d %d", &u, &v, &w);

            G[u].push\_back(AdjNode(v, w));

            G[v].push\_back(AdjNode(u, w));

        }

        int sum = Prim(G, VexNum);

        if (sum < 0) cout << "?" << endl;

        else cout << sum << endl;

    }

}

## 2.5最短路径

### 题目（变形）：

**Problem Description**

给你n个点，m条无向边，每条边都有长度d和花费p，给你起点s终点t，要求输出起点到终点的最短距离及其花费，如果最短距离有多条路线，则输出花费最少的。

**Input**

输入n,m，点的编号是1~n,然后是m行，每行4个数 a,b,d,p，表示a和b之间有一条边，且其长度为d，花费为p。最后一行是两个数 s,t;起点s，终点。n和m为0时输入结束。  
(1<n<=1000, 0<m<100000, s != t)

**Output**

输出 一行有两个数， 最短距离及其花费。

**Sample Input**

3 2

1 2 5 6

2 3 4 5

1 3

0 0

**Sample Output**

9 11

### 题解：

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<vector>

#include<stdio.h>

#include<queue>

#include<iostream>

using namespace std;

struct ArcNode {

    int v;//邻接点

    int len;//长度

    int cost;//花费

    ArcNode() {};

    ArcNode(int v1, int len1, int cost1) :v(v1), len(len1), cost(cost1) {};

};

struct GNode {

    int v;//结点v

    int len;//源点到结点v的距离

    int cost;//源点到结点v的花费

    GNode() {};

    GNode(int v1, int len1, int cost1) :v(v1), len(len1), cost(cost1) {};

    friend bool operator<(const GNode &node1, const GNode &node2);

};

bool operator<(const GNode &node1, const GNode &node2)

{

    if (node1.len == node2.len)

        return node1.cost > node2.cost;

    return node1.len > node2.len;

}

int s, t;//起点s，终点t

vector<int> visited;//标记结点是否被访问

vector<int> len;//len[v]表示源点到结点v的距离

vector<int> cost;//cost[v]表示源点到结点v的花费

void Dijkstra(vector<vector<ArcNode> > &G)

{

    priority\_queue<GNode> q;

    len[s] = 0;

    cost[s] = 0;

    q.push(GNode(s, 0, 0));

    while (!q.empty())

    {

        GNode front;

        front = q.top(); q.pop();

        if (visited[front.v]) continue;//结点已被访问

        visited[front.v] = 1;

        //special

        if (front.v == t)//已经获取到终点的路径长度

        {

            return;

        }//special

        for (int i = 0; i < G[front.v].size(); i++)//访问v的所有邻接点

        {

            int adjv = G[front.v][i].v; //v的第i个邻接点

            int adjlen = G[front.v][i].len;

            int adjcost = G[front.v][i].cost;

            if (!visited[adjv])//若v的第i个邻接点未被访问

            {

                if (len[front.v] + adjlen < len[adjv])//有更短的路径，更新len和cost

                {

                    len[adjv] = len[front.v] + adjlen;

                    cost[adjv] = cost[front.v] + adjcost;

q.push(GNode(adjv, len[adjv], cost[adjv]));

                }

                else if (len[front.v] + adjlen == len[adjv])//路径一样长，选择cost小的那一条路

                {

                    if (cost[front.v] + adjcost < cost[adjv])

                        {

cost[adjv] = cost[front.v] + adjcost;//若需要前驱结点也要更新

q.push(GNode(adjv, len[adjv], cost[adjv]));

}

                }

            }

        }

    }

}

int main(void)

{

    int n, m;//节点数，边数

    while (true)

    {

        scanf("%d %d", &n, &m);

        if (n == 0 && m == 0) break;

        vector<vector<ArcNode> >G(n + 1);

        visited.assign(n + 1, 0);

        len.assign(n + 1, INT\_MAX);

        cost.assign(n + 1, INT\_MAX);

        for (int i = 0; i < m; i++)

        {

            int a, b, d, p;//边(a,b)长d，花费p

            scanf("%d %d %d %d", &a, &b, &d, &p);

            G[a].push\_back(ArcNode(b, d, p));

            G[b].push\_back(ArcNode(a, d, p));

        }

        scanf("%d %d", &s, &t);

        Dijkstra(G);

        printf("%d %d\n", len[t], cost[t]);

    }

}

### 题目（普通）：

**Problem Description**

在每年的校赛里，所有进入决赛的同学都会获得一件很漂亮的t-shirt。但是每当我们的工作人员把上百件的衣服从商店运回到赛场的时候，却是非常累的！所以现在他们想要寻找最短的从商店到赛场的路线，你可以帮助他们吗？

**Input**

输入包括多组数据。每组数据第一行是两个整数N、M（N<=100，M<=10000），N表示成都的大街上有几个路口，标号为1的路口是商店所在地，标号为N的路口是赛场所在地，M则表示在成都有几条路。N=M=0表示输入结束。接下来M行，每行包括3个整数A，B，C（1<=A,B<=N,1<=C<=1000）,表示在路口A与路口B之间有一条路，我们的工作人员需要C分钟的时间走过这条路。  
输入保证至少存在1条商店到赛场的路线。

**Output**

对于每组输入，输出一行，表示工作人员从商店走到赛场的最短时间

**Sample Input**

2 1

1 2 3

3 3

1 2 5

2 3 5

3 1 2

0 0

**Sample Output**

3

2

### 题解:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<vector>

#include<queue>

#include<stdio.h>

using namespace std;

int N, M;//N个路口，M条边

vector<int> visited;

vector<int> cost;

struct ArcNode

{

    int v;

    int cost;

    ArcNode() {};

    ArcNode(int v1, int cost1) :v(v1), cost(cost1) {};

};

bool operator<(const ArcNode &node1, const ArcNode &node2)

{

    return node1.cost > node2.cost;

}

void Dijkstra(vector<vector<ArcNode> > &G,int src)

{

    priority\_queue<ArcNode> q;

    cost[src] = 0;

    q.push(ArcNode(src, 0));

    while (!q.empty())

    {

        ArcNode front;

        front = q.top(); q.pop();

        if (!visited[front.v])

        {

            if (front.v == N) return;//找到到达终点的路线

            visited[front.v] = 1;

            int adjv, adjcost;

            for (int i = 0; i < G[front.v].size(); i++)//访问所有邻接点

            {

                adjv = G[front.v][i].v; adjcost = G[front.v][i].cost;

                if (visited[adjv]) continue;

                if (cost[front.v] + adjcost < cost[adjv])

                {

                    cost[adjv] = cost[front.v] + adjcost;

q.push(ArcNode(adjv, cost[adjv]));

                }

            }

        }

    }

}

int main(void)

{

    while (true)

    {

        scanf("%d %d", &N, &M);

        if (N == 0 && M == 0) break;

        vector<vector<ArcNode> > G(N + 1);

        visited.assign(N + 1, 0);

        cost.assign(N + 1, INT\_MAX);

        for (int i = 0; i < M; i++)

        {

            int a, b, c;

            scanf("%d %d %d", &a, &b, &c);

            G[a].push\_back(ArcNode(b, c));

            G[b].push\_back(ArcNode(a, c));

        }

        Dijkstra(G, 1);

        cout << cost[N] << endl;

    }

}

## 2.6拓扑排序

//输入：邻接表表示的图G(节点从0开始顺序编号)，存放数据的容器sorted

    //操作：对G进行拓扑排序，将排序后的序列存放到sorted容器中

    //返回值：true：成功排序；false：图中存在环，拓扑排序失败

    //限制条件：无

    bool TopologicalSort(vector<vector<ArcNode>>& G,vector<int>& sorted)

    {

        vector<int> InDgree(G.size(), 0);

        int i, j;

        /\*统计所有顶点的入度\*/

        for(i=0;i<G.size();i++)

            for (j = 0; j < G[i].size(); j++)

                InDgree[G[i][j].adjVex]++;

        queue<int> q;

        /\*入度为0的顶点入队\*/

        for (i = 0; i < G.size(); i++)

            if (InDgree[i] == 0)

                q.push(i);

        vector<int> res;

        int front;

        while (!q.empty())

        {

            front = q.front(); q.pop();

            res.push\_back(front);

            /\*front的邻接点入度-1\*/

            for (int j = 0; j < G[front].size(); j++)

                if (--InDgree[G[front][j].adjVex] == 0)

                    q.push(G[front][j].adjVex);

        }

        if (res.size() < G.size()) return false;

        sorted = res;

        return true;

    }

//leetcode AC拓扑排序，numCOurses是节点数，prerequisited的second是first的先修课

#include<vector>

#include<queue>

#include<iostream>

using namespace std;

class Solution {

public:

    vector<int> findOrder(int numCourses, vector<pair<int, int>>& prerequisites) {

        vector<int> inDegree(numCourses,0);//保存结点的入度

        vector<vector<int> >G(numCourses);//保存拓扑图

        vector<bool> visited(numCourses, 0);

        queue<int> q;// 保存入度为0的结点

        for (int i = 0; i < prerequisites.size(); i++)

        {

            G[prerequisites[i].second].push\_back(prerequisites[i].first);//创建拓扑图

            inDegree[prerequisites[i].first]++;

        }

        /\*将入度为0的结点入队\*/

        for (int i = 0; i < inDegree.size(); i++)

        {

            if (inDegree[i] == 0)

                q.push(i);

        }

        /\*执行拓扑排序\*/

        vector<int> ans;

        while (!q.empty())

        {

            int front = q.front(); q.pop();

            ans.push\_back(front);

            for (int i = 0; i < G[front].size(); i++)//删除与该结点相连的边(等效于与该点的邻接点的入度-1)

            {

                if (--inDegree[G[front][i]] == 0)//入度减位0则入队

                    q.push(G[front][i]);

            }

        }

        if (ans.size() != numCourses)//拓扑排序失败

            ans.clear();

        return ans;

    }

};

int main(void)

{

    Solution s;

    int n, m;

    vector<pair<int, int>> v;

    cin >> n >> m;

    for (int i = 0; i < m; i++)

    {

        int a, b;

        cin >> a >> b;

        v.push\_back(pair<int, int>(a, b));

    }

    s.findOrder(n, v);

}

## 2.7最优二叉搜索树

#include<iostream>

#include<vector>

using namespace std;

#define MAX\_SIZE 1000

double p[MAX\_SIZE], q[MAX\_SIZE];

double e[MAX\_SIZE][MAX\_SIZE], w[MAX\_SIZE][MAX\_SIZE];

int root[MAX\_SIZE][MAX\_SIZE];

//输入：关键字概率p，伪关键字概率q,关键字个数n，关键字代价矩阵e，关键字分割矩阵root

//操作：求最优二叉树，将分割代价保存在e中，分割点保存在root中

//返回值：无

void OPTIMAL\_BST(int n)

{

    for (int i = 1; i <= n + 1; i++)

    {

        e[i][i - 1] = q[i - 1];

        w[i][i - 1] = q[i - 1];

    }//初始条件，只包含伪关键字,相当于l=0，

    for (int l = 1; l <= n; l++)

    {//子树包含的关键字数量从1到n逐次枚举

        for (int i = 1; i <= n - l + 1; i++)

        {//子树包含关键字ki,ki+1,ki+2...kj

            int j = i + l - 1;

            e[i][j] = INT\_MAX;

            w[i][j] = w[i][j - 1] + p[j] + q[j];//计算子树概率

            for (int r = i; r <= j; r++)//枚举所有的根

            {

                double temp = e[i][r - 1] + e[r + 1][j] + w[i][j];

                if (temp < e[i][j])//更新代价和根

                {

                    e[i][j] = temp;

                    root[i][j] = r;

                }

            }

        }

    }

    return;

}

int main(void)

{

    system("color F0\n");

    int N;

    cout << "输入关键字个数:" << endl;

    cin >> N;

    cout << "输入" << N<<"个关键字的概率：" << endl;

    for (int i = 1; i <= N; i++)

        cin >> p[i];

    cout << "输入" << N+1 << "个伪关键字的概率：" << endl;

    for (int i = 0; i <= N; i++)

        cin >> q[i];

    OPTIMAL\_BST(N);

    cout << "代价" << "e[i][j]:" << endl;

    for (int l = 0; l <= N; l++)

    {

        int begin, end;

        for (begin = 1; begin <= N-l+1; begin++)

        {

            end = begin + l - 1;

            printf("%-8.3f ", e[begin][end]);

        }

        cout << endl;

    }

    cout << "概率" << "w[i][j]:" << endl;

    for (int l = 0; l <= N; l++)

    {

        int begin, end;

        for (begin = 1; begin <= N - l + 1; begin++)

        {

            end = begin + l - 1;

            printf("%-8.3f ", w[begin][end]);

        }

        cout << endl;

    }

    cout << "根" << "root[i][j]:" << endl;

    for (int l = 1; l <= N; l++)

    {

        int begin, end;

        for (begin = 1; begin <= N - l + 1; begin++)

        {

            end = begin + l - 1;

            printf("%-8d ", root[begin][end]);

        }

        cout << endl;

    }

    system("pause");

}

## 2.8闰年判断

1800年1月1日周三

int isSpecialYear(int year)

{

    if (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0))

        return 366;

    return 365;

}

## 2.9快速模除

//求出(a^b)%mod

unsigned long long powermod(unsigned long long a, unsigned long long b, unsigned long long mod)

{

    unsigned long long sum = 1;

    while (b)

    {

        if (b & 1)//b是奇数

        {

            sum = (sum\*a) % mod;

        }

        b /= 2;

        a = (a\*a) % mod;

    }

    return sum;

}

## 2.10网络流(EK算法)

**输入描述 Input Description**

第1行: 两个用空格分开的整数N (0 <= N <= 200) 和 M (2 <= M <= 200)。N是农夫John已经挖好的排水沟的数量，M是排水沟交叉点的数量。交点1是水潭，交点M是小溪。

第二行到第N+1行: 每行有三个整数，Si, Ei, 和 Ci。Si 和 Ei (1 <= Si, Ei <= M) 指明排水沟两端的交点，雨水从Si 流向Ei。Ci (0 <= Ci <= 10,000,000)是这条排水沟的最大容量。

**输出描述 Output Description**

输出一个整数，即排水的最大流量。

**样例输入 Sample Input**

5 4

1 2 40

1 4 20

2 4 20

2 3 30

3 4 10

**样例输出 Sample Output**

50

//codevs 1993

#include<iostream>

#include<stdio.h>

#include<vector>

#include<algorithm>

#include<queue>

using namespace std;

const int INF = 0x7ffffff;

int n, m;//n是节点数，m是边数，1是源点，n是汇点

vector<int> pre(201, -1);//bfs时保存前驱结点

vector<int> flow(201);//flow[i]表示源点到结点i的流量

int g[201][201], maxflow;

int bfs(int s, int t)

{

    queue <int> q;

    pre.assign(201, -1);//置空

    pre[s] = 0;

    q.push(s);

    flow[s] = INF;//初始流量无穷大

    while (!q.empty())

    {

        int x = q.front();

        q.pop();

        if (x == t) break;//到达汇点

        for (int i = 1; i <= n; i++)

            //EK一次只找一个增广路

            if (g[x][i] > 0 && pre[i] == -1)

            {

                pre[i] = x;

                flow[i] = min(flow[x], g[x][i]);

                q.push(i);

            }

    }

    if (pre[t] == -1) return -1;//未找到到达汇点的路径

    else return flow[t];

}

//increase为增广的流量

void EK(int s, int t)

{

    int increase = 0;

    while ((increase = bfs(s, t)) != -1)//bfs获取一条增广路径

    {//迭代

        int k = t;

        while (k != s)//从汇点回溯到源点

        {

            int last = pre[k];//从后往前找路径

            g[last][k] -= increase;

            g[k][last] += increase;

            k = last;

        }

        maxflow += increase;

    }

}

int main()

{

    cin >> m >> n;//边数，节点数

    for (int i = 1; i <= m; i++)

    {

        int a, b, c;

        cin >> a >> b >> c;//(a,b)=c

        g[a][b] += c;//此处不可直接输入，要+=，可能存在多重边

    }

    EK(1, n);

    printf("%d", maxflow);

    return 0;

}

## 2.11费用流

**题目描述**

如题，给出一个网络图，以及其源点和汇点，每条边已知其最大流量和单位流量费用，求出其网络最大流和在最大流情况下的最小费用。

**输入输出格式**

**输入格式：**

第一行包含四个正整数N、M、S、T，分别表示点的个数、有向边的个数、源点序号、汇点序号。

接下来M行每行包含四个正整数ui、vi、wi、fi，表示第i条有向边从ui出发，到达vi，边权为wi（即该边最大流量为wi），单位流量的费用为fi。

**输出格式：**

一行，包含两个整数，依次为最大流量和在最大流量情况下的最小费用。

**输入输出样例**

**输入样例#1：** 复制

* 4 5 4 3
* 4 2 30 2
* 4 3 20 3
* 2 3 20 1
* 2 1 30 9
* 1 3 40 5

**输出样例#1：** 复制

* 50 280

**说明**

时空限制：1000ms,128M

（BYX：最后两个点改成了1200ms）

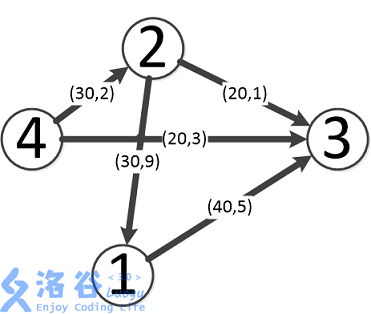
数据规模：

对于30%的数据：N<=10，M<=10

对于70%的数据：N<=1000，M<=1000

对于100%的数据：N<=5000，M<=50000

样例说明：



如图，最优方案如下：

第一条流为4-->3，流量为20，费用为3\*20=60。

第二条流为4-->2-->3，流量为20，费用为（2+1）\*20=60。

第三条流为4-->2-->1-->3，流量为10，费用为（2+9+5）\*10=160。

故最大流量为50，在此状况下最小费用为60+60+160=280。

故输出50 280。

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<queue>

using namespace std;

const int maxn=100010;

bool vis[maxn];

int n,m,s,t,x,y,z,f,dis[maxn],pre[maxn],last[maxn],flow[maxn],maxflow,mincost;

//dis最小花费;pre每个点的前驱；last每个点的所连的前一条边；flow源点到此处的流量

struct Edge{

int to,next,flow,dis;//flow流量 dis花费

}edge[maxn];

int head[maxn],num\_edge;

queue <int> q;

void add\_edge(int from,int to,int flow,int dis)

{

edge[++num\_edge].next=head[from];

edge[num\_edge].to=to;

edge[num\_edge].flow=flow;

edge[num\_edge].dis=dis;

head[from]=num\_edge;

}

bool spfa(int s,int t)

{

memset(dis,0x7f,sizeof(dis));

memset(flow,0x7f,sizeof(flow));

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

q.push(s); vis[s]=1; dis[s]=0; pre[t]=-1;

while (!q.empty())

{

int now=q.front();

q.pop();

vis[now]=0;

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

{

if (edge[i].flow>0 && dis[edge[i].to]>dis[now]+edge[i].dis)//正边

{

dis[edge[i].to]=dis[now]+edge[i].dis;

pre[edge[i].to]=now;

last[edge[i].to]=i;

flow[edge[i].to]=min(flow[now],edge[i].flow);//

if (!vis[edge[i].to])

{

vis[edge[i].to]=1;

q.push(edge[i].to);

}

}

}

}

return pre[t]!=-1;

}

void MCMF()

{

while (spfa(s,t))

{

int now=t;

maxflow+=flow[t];

mincost+=flow[t]\*dis[t];

while (now!=s)

{//从源点一直回溯到汇点

edge[last[now]].flow-=flow[t];//flow和dis容易搞混

edge[last[now]^1].flow+=flow[t];

now=pre[now];

}

}

}

int main()

{

memset(head,-1,sizeof(head)); num\_edge=-1;//初始化

scanf("%d%d%d%d",&n,&m,&s,&t);

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

{

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

add\_edge(x,y,z,f); add\_edge(y,x,0,-f);

//反边的流量为0，花费是相反数

}

MCMF();

printf("%d %d",maxflow,mincost);

return 0;

}

## 2.12记忆性搜索(矩阵中的最长路径)

给定一个整数矩阵，找出最长递增路径的长度。

对于每个单元格，你可以往上，下，左，右四个方向移动。 你不能在对角线方向上移动或移动到边界外（即不允许环绕）。

**示例 1:**

**输入:** nums =

[

[**9**,9,4],

[**6**,6,8],

[**2**,**1**,1]

]

**输出:** 4

**解释:** 最长递增路径为 [1, 2, 6, 9]。

**示例 2:**

**输入:** nums =

[

[**3**,**4**,**5**],

[3,2,**6**],

[2,2,1]

]

**输出:** 4

**解释:** 最长递增路径是 [3, 4, 5, 6]。注意不允许在对角线方向上移动。

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

class Solution

{

public:

int longestIncreasingPath(vector<vector<int>> &matrix)

{

if (matrix.empty())

return 0;

int raw, col;

raw = matrix.size();

col = matrix[0].size();

vector<vector<int>> path; //path[i][j]表示从i到j的最长路径的长度

path.resize(raw);

for (int i = 0; i < raw; i++)

{

path[i].resize(col);

for (int j = 0; j < col; j++)

path[i][j] = -1;

} //初始化路径

int res = 0;

for (int i = 0; i < raw; i++)

for (int j = 0; j < col; j++)

{

int temp;

temp = DFS(matrix, path, i, j, raw, col);

res = max(temp, res);

}

return res;

}

private:

int DFS(vector<vector<int>> &matrix, vector<vector<int>> &path, int i, int j, int &raw, int &col)

{

if (path[i][j] != -1)

return path[i][j]; //该节点的最长路径已经求出

int up, down, left, right; //matrix[i][j]上下左右的元素大小，若不存在则标记为-1

up = i - 1 >= 0 ? matrix[i - 1][j] : -1;

down = i + 1 < raw ? matrix[i + 1][j] : -1;

left = j - 1 >= 0 ? matrix[i][j - 1] : -1;

right = j + 1 < col ? matrix[i][j + 1] : -1;

int up\_path, down\_path, left\_path, right\_path; //从matrix[i][j]出发，分别沿4个方向获取的最长路径

up\_path = matrix[i][j] < up ? 1 + DFS(matrix, path, i - 1, j, raw, col) : 1;

down\_path = matrix[i][j] < down ? 1 + DFS(matrix, path, i + 1, j, raw, col) : 1;

left\_path = matrix[i][j] < left ? 1 + DFS(matrix, path, i, j - 1, raw, col) : 1;

right\_path = matrix[i][j] < right ? 1 + DFS(matrix, path, i, j + 1, raw, col) : 1;

int m1, m2;

m1 = max(up\_path, down\_path);

m2 = max(left\_path, right\_path);

m1 = max(m1, m2); //在四个方向的路径中找到最长的那个

path[i][j] = m1;

return m1;

}

};

## 2.13两个日期间的差值（日期计算）

有两个日期，求两个日期之间的天数，如果两个日期是连续的我们规定他们之间的天数为两天。输入格式为YYYYMMDD。

#include <iostream>

#include <string>

using namespace std;

int daysOfMonth[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

int daysSum[13];

//计算到达该月经过了几天

void getDaysSum()

{

daysSum[0] = 0;

for (int i = 1; i <= 12; i++) {

daysSum[i] = daysSum[i - 1] + daysOfMonth[i - 1];

}

}

//计算该年的总天数

int leapYear(int year)

{

if (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0))

return 366;

return 365;

}

//计算当前日期是该年的第几天

int daysToCurrent(int year, int month, int day)

{

int total = daysSum[month - 1] + day;

if (leapYear(year) == 366 && month > 2) {

total++;

}

return total;

}

//将字符串中的日期分割为数字

void splitDate(const string& date, int& year, int& month, int& day)

{

year = (date[0] - '0') \* 1000 + (date[1] - '0') \* 100 + (date[2] - '0') \* 10 + (date[3] - '0');

month = (date[4] - '0') \* 10 + (date[5] - '0');

day = (date[6] - '0') \* 10 + (date[7] - '0');

}

int main(void)

{

getDaysSum();

int year1, month1, day1;

int year2, month2, day2;

string date1, date2;

while (cin >> date1 >> date2) {

if (date1 > date2) { //保证date1日期小

date1.swap(date2);

}

splitDate(date1, year1, month1, day1);

splitDate(date2, year2, month2, day2);

//计算两个日期距离‘year1年1月1日的天数‘

int sum1 = 0;

int sum2 = 0;

sum1 = daysToCurrent(year1, month1, day1);

for (int i = year1; i < year2; i++) {

sum2 += leapYear(i);

}

sum2 += daysToCurrent(year2, month2, day2);

cout << sum2 - sum1 + 1 << endl;

}

return 0;

}

# 历年习题

## 3.1有趣的数(动态规划)

**问题描述**

　　我们把一个数称为有趣的，当且仅当：  
　　1. 它的数字只包含0, 1, 2, 3，且这四个数字都出现过至少一次。  
　　2. 所有的0都出现在所有的1之前，而所有的2都出现在所有的3之前。  
　　3. 最高位数字不为0。  
　　因此，符合我们定义的最小的有趣的数是2013。除此以外，4位的有趣的数还有两个：2031和2301。  
　　请计算恰好有n位的有趣的数的个数。由于答案可能非常大，只需要输出答案除以1000000007的余数。

**输入格式**

　　输入只有一行，包括恰好一个正整数n (4 ≤ n ≤ 1000)。

**输出格式**

　　输出只有一行，包括恰好n 位的整数中有趣的数的个数除以1000000007的余数。

**样例输入**

4

**样例输出**

3

#define MOD 1000000007

#include<iostream>

using namespace std;

unsigned long long D[1001][6];

int main(void)

{

    D[1][0] = 1;

    int n;

    cin >> n;

    for (int i = 2; i <= n; i++)

    {

        D[i][0] = (D[i - 1][0]) % MOD;

        D[i][1] = (D[i - 1][0] + 2 \* D[i - 1][1]) % MOD;

        D[i][2] = (D[i - 1][0] + D[i - 1][2]) % MOD;

        D[i][3] = (D[i - 1][1] + 2 \* D[i - 1][3]) % MOD;

        D[i][4] = (D[i - 1][1] + D[i - 1][2] + 2 \* D[i - 1][4]) % MOD;

        D[i][5] = (D[i - 1][3] + D[i - 1][4] + 2 \* D[i - 1][5]) % MOD;

    }

    cout << D[n][5] << endl;

}

## 3.2无线网络(BFS+long long)

**问题描述**

　　目前在一个很大的平面房间里有 n 个无线路由器,每个无线路由器都固定在某个点上。任何两个无线路由器只要距离不超过 r 就能互相建立网络连接。  
　　除此以外,另有 m 个可以摆放无线路由器的位置。你可以在这些位置中选择至多 k 个增设新的路由器。  
　　你的目标是使得第 1 个路由器和第 2 个路由器之间的网络连接经过尽量少的中转路由器。请问在最优方案下中转路由器的最少个数是多少?

**输入格式**

　　第一行包含四个正整数 n,m,k,r。(2 ≤ n ≤ 100,1 ≤ k ≤ m ≤ 100, 1 ≤ r ≤ 108)。  
　　接下来 n 行,每行包含两个整数 xi 和 yi,表示一个已经放置好的无线 路由器在 (xi, yi) 点处。输入数据保证第 1 和第 2 个路由器在仅有这 n 个路由器的情况下已经可以互相连接(经过一系列的中转路由器)。  
　　接下来 m 行,每行包含两个整数 xi 和 yi,表示 (xi, yi) 点处可以增设 一个路由器。  
　　输入中所有的坐标的绝对值不超过 108,保证输入中的坐标各不相同。

**输出格式**

　　输出只有一个数,即在指定的位置中增设 k 个路由器后,从第 1 个路 由器到第 2 个路由器最少经过的中转路由器的个数。

**样例输入**

5 3 1 3  
0 0  
5 5  
0 3  
0 5  
3 5  
3 3  
4 4  
3 0

**样例输出**

2

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

struct Node

{

    long long x, y;//坐标+可额外安放的路由器数目

    Node() {};

    Node(long long x1, long long y1) :x(x1), y(y1) {};

};

struct Router

{

    int id;//路由器编号

    int k;//还可安放的额外路由器编号

    int deep;//BFS的深度

    Router() {};

    Router(int id1, int k1, int deep1) :id(id1), k(k1), deep(deep1) {};

};

int visited[201][101];//visited[i]表示到达到达第i个路由器时还可以安放visited[i]个额外路由器

int n, m, k;

unsigned long long r;

bool isOK(Node &a, Node &b)

{

    if ((a.x - b.x)\*(a.x - b.x) + (a.y - b.y)\*(a.y - b.y) <= r\*r)

    {

        return true;

    }

    return false;

}

int main(void)

{

    cin >> n >> m >> k >> r;

    vector<Node> v(n + m + 1);//n个路由器+m个可安放路由器的位置

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

    {

        cin >> v[i].x >> v[i].y;

    }

    queue<Router> q;

    q.push(Router(1, k, 0));

    int ans = 0;

    while (!q.empty())

    {

        Router front = q.front(); q.pop();

        if (front.id==2)//到达第二个路由器

        {

            ans = front.deep; break;

        }

        int Max = (front.k == 0 ? n : m + n);

        for (int i = 1; i <= Max; i++)

        {

            if (i <= n)//不使用额外的路由器

            {

                if (!visited[i][front.k] &&isOK(v[front.id], v[i]))//当前路由器可以到达第i个路由器

                {

                    visited[i][front.k] = 1;

                    q.push(Router(i, front.k, front.deep + 1));

                }

            }

            else//使用额外的路由器

            {

                if (!visited[i][front.k-1] && isOK(v[front.id], v[i]))//当前路由器可以到达第i个路由器

                {

                    visited[i][front.k - 1] = 1;

                    q.push(Router(i, front.k - 1, front.deep + 1));

                }

            }

        }

    }

    cout << ans - 1 << endl;

}

## 3.3命令行选项(字符串处理)

|  |
| --- |
| 201403-3 |
| 试题名称： | 命令行选项 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　请你写一个命令行分析程序,用以分析给定的命令行里包含哪些选项。每个命令行由若干个字符串组成,它们之间恰好由一个空格分隔。这些字符串中的第一个为该命令行工具的名字,由小写字母组成,你的程序不用对它进行处理。在工具名字之后可能会包含若干选项,然后可能会包含一 些不是选项的参数。 　　选项有两类:带参数的选项和不带参数的选项。一个合法的无参数选项的形式是一个减号后面跟单个小写字母,如"-a" 或"-b"。而带参数选项则由两个由空格分隔的字符串构成,前者的格式要求与无参数选项相同,后者则是该选项的参数,是由小写字母,数字和减号组成的非空字符串。 　　该命令行工具的作者提供给你一个格式字符串以指定他的命令行工具需要接受哪些选项。这个字符串由若干小写字母和冒号组成,其中的每个小写字母表示一个该程序接受的选项。如果该小写字母后面紧跟了一个冒号,它就表示一个带参数的选项,否则则为不带参数的选项。例如, "ab:m:" 表示该程序接受三种选项,即"-a"(不带参数),"-b"(带参数), 以及"-m"(带参数)。 　　命令行工具的作者准备了若干条命令行用以测试你的程序。对于每个命令行,你的工具应当一直向后分析。当你的工具遇到某个字符串既不是合法的选项,又不是某个合法选项的参数时,分析就停止。命令行剩余的未分析部分不构成该命令的选项,因此你的程序应当忽略它们。  **输入格式**  　　输入的第一行是一个格式字符串,它至少包含一个字符,且长度不超过 52。格式字符串只包含小写字母和冒号,保证每个小写字母至多出现一次,不会有两个相邻的冒号,也不会以冒号开头。 　　输入的第二行是一个正整数 N(1 ≤ N ≤ 20),表示你需要处理的命令行的个数。 　　接下来有 N 行,每行是一个待处理的命令行,它包括不超过 256 个字符。该命令行一定是若干个由单个空格分隔的字符串构成,每个字符串里只包含小写字母,数字和减号。  **输出格式**  　　输出有 N 行。其中第 i 行以"Case i:" 开始,然后应当有恰好一个空格,然后应当按照字母升序输出该命令行中用到的所有选项的名称,对于带参数的选项,在输出它的名称之后还要输出它的参数。如果一个选项在命令行中出现了多次,只输出一次。如果一个带参数的选项在命令行中出 现了多次,只输出最后一次出现时所带的参数。  **样例输入**  albw:x 4 ls -a -l -a documents -b ls ls -w 10 -x -w 15 ls -a -b -c -d -e -l  **样例输出**  Case 1: -a -l Case 2: Case 3: -w 15 -x Case 4: -a -b |

#include<iostream>

#include<string>

#include<vector>

#include<map>

#include<sstream>

using namespace std;

string format;

int main(void)

{

    cin >> format;

    int N;

    cin >> N;

    getchar();

    for (int i = 1; i <= N; i++)

    {

        string line;

        getline(cin, line);

        map<string, string> mp;

        istringstream instr(line);

        string opt;

        instr >> opt;//忽略程序名

        while (instr >> opt)

        {

            if (opt.length() == 2 && opt[0] == '-')//是一个选项

            {

                int pos = format.find(opt[1]);

                if (pos == -1)

                {

                    break;//非法选项

                }

                if (format[pos + 1] == ':')//是带参数的选项

                {

                    string arg;

                    if(instr >> arg)//读取参数

                        mp[opt] = arg;

                }

                else//不带参数的选项

                {

                    mp[opt] = "";

                }

            }

            else break;//既不是选项也不是参数

        }

        map<string, string>::iterator it = mp.begin();

        cout << "Case " << i << ":";

        while (it != mp.end())

        {

            if (it->second == "")

            {

                cout << " " << it->first;

            }

            else cout << " " << it->first << " " << it->second;

            it++;

        }

        cout << endl;

    }

    return 0;

}

## 3.4无线网络(BFS+注意溢出)

|  |  |
| --- | --- |
| 试题编号： | 201403-4 |
| 试题名称： | 无线网络 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　目前在一个很大的平面房间里有 n 个无线路由器,每个无线路由器都固定在某个点上。任何两个无线路由器只要距离不超过 r 就能互相建立网络连接。 　　除此以外,另有 m 个可以摆放无线路由器的位置。你可以在这些位置中选择至多 k 个增设新的路由器。 　　你的目标是使得第 1 个路由器和第 2 个路由器之间的网络连接经过尽量少的中转路由器。请问在最优方案下中转路由器的最少个数是多少?  **输入格式**  　　第一行包含四个正整数 n,m,k,r。(2 ≤ n ≤ 100,1 ≤ k ≤ m ≤ 100, 1 ≤ r ≤ 108)。 　　接下来 n 行,每行包含两个整数 xi 和 yi,表示一个已经放置好的无线 路由器在 (xi, yi) 点处。输入数据保证第 1 和第 2 个路由器在仅有这 n 个路由器的情况下已经可以互相连接(经过一系列的中转路由器)。 　　接下来 m 行,每行包含两个整数 xi 和 yi,表示 (xi, yi) 点处可以增设 一个路由器。 　　输入中所有的坐标的绝对值不超过 108,保证输入中的坐标各不相同。  **输出格式**  　　输出只有一个数,即在指定的位置中增设 k 个路由器后,从第 1 个路 由器到第 2 个路由器最少经过的中转路由器的个数。  **样例输入**  5 3 1 3 0 0 5 5 0 3 0 5 3 5 3 3 4 4 3 0  **样例输出**  2 |

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

struct Node

{

    long long x, y;//坐标+可额外安放的路由器数目

    Node() {};

    Node(long long x1, long long y1) :x(x1), y(y1) {};

};

struct Router

{

    int id;//路由器编号

    int k;//还可安放的额外路由器编号

    int deep;//BFS的深度

    Router() {};

    Router(int id1, int k1, int deep1) :id(id1), k(k1), deep(deep1) {};

};

int visited[201][101];//visited[i]表示到达到达第i个路由器时还可以安放visited[i]个额外路由器

int n, m, k;

unsigned long long r;

bool isOK(Node &a, Node &b)

{

    if ((a.x - b.x)\*(a.x - b.x) + (a.y - b.y)\*(a.y - b.y) <= r\*r)

    {

        return true;

    }

    return false;

}

int main(void)

{

    cin >> n >> m >> k >> r;

    vector<Node> v(n + m + 1);//n个路由器+m个可安放路由器的位置

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

    {

        cin >> v[i].x >> v[i].y;

    }

    queue<Router> q;

    q.push(Router(1, k, 0));

    int ans = 0;

    while (!q.empty())

    {

        Router front = q.front(); q.pop();

        if (front.id==2)//到达第二个路由器

        {

            ans = front.deep; break;

        }

        int Max = (front.k == 0 ? n : m + n);

        for (int i = 1; i <= Max; i++)

        {

            if (i <= n)//不使用额外的路由器

            {

                if (!visited[i][front.k] &&isOK(v[front.id], v[i]))//当前路由器可以到达第i个路由器

                {

                    visited[i][front.k] = 1;

                    q.push(Router(i, front.k, front.deep + 1));

                }

            }

            else//使用额外的路由器

            {

                if (!visited[i][front.k-1] && isOK(v[front.id], v[i]))//当前路由器可以到达第i个路由器

                {

                    visited[i][front.k - 1] = 1;

                    q.push(Router(i, front.k - 1, front.deep + 1));

                }

            }

        }

    }

    cout << ans - 1 << endl;

}

## 3.5最优配餐(带障碍的BFS)

|  |  |
| --- | --- |
| 试题编号： | 201409-4 |
| 试题名称： | 最优配餐 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　栋栋最近开了一家餐饮连锁店，提供外卖服务。随着连锁店越来越多，怎么合理的给客户送餐成为了一个急需解决的问题。 　　栋栋的连锁店所在的区域可以看成是一个n×n的方格图（如下图所示），方格的格点上的位置上可能包含栋栋的分店（绿色标注）或者客户（蓝色标注），有一些格点是不能经过的（红色标注）。 　　方格图中的线表示可以行走的道路，相邻两个格点的距离为1。栋栋要送餐必须走可以行走的道路，而且不能经过红色标注的点。  http://118.190.20.162/RequireFile.do?fid=383qHJjQ 　　送餐的主要成本体现在路上所花的时间，每一份餐每走一个单位的距离需要花费1块钱。每个客户的需求都可以由栋栋的任意分店配送，每个分店没有配送总量的限制。 　　现在你得到了栋栋的客户的需求，请问在最优的送餐方式下，送这些餐需要花费多大的成本。  **输入格式**  　　输入的第一行包含四个整数n, m, k, d，分别表示方格图的大小、栋栋的分店数量、客户的数量，以及不能经过的点的数量。 　　接下来m行，每行两个整数xi, yi，表示栋栋的一个分店在方格图中的横坐标和纵坐标。 　　接下来k行，每行三个整数xi, yi, ci，分别表示每个客户在方格图中的横坐标、纵坐标和订餐的量。（注意，可能有多个客户在方格图中的同一个位置） 　　接下来d行，每行两个整数，分别表示每个不能经过的点的横坐标和纵坐标。  **输出格式**  　　输出一个整数，表示最优送餐方式下所需要花费的成本。  **样例输入**  10 2 3 3 1 1 8 8 1 5 1 2 3 3 6 7 2 1 2 2 2 6 8  **样例输出**  29  **评测用例规模与约定**  　　前30%的评测用例满足：1<=n <=20。 　　前60%的评测用例满足：1<=n<=100。 　　所有评测用例都满足：1<=n<=1000，1<=m, k, d<=n^2。可能有多个客户在同一个格点上。每个客户的订餐量不超过1000，每个客户所需要的餐都能被送到。 |

#include<iostream>

#include<stdio.h>

#include<vector>

#include<queue>

using namespace std;

struct Node

{

    int x;

    int y;

    int num;

    Node(int x1, int y1, int n1) :x(x1), y(y1), num(n1) {};

    Node() {};

};

int n, m, k, d;

int matrix[1001][1001];

int visited[1001][1001];

int main(void)

{

    queue<Node> q;

    cin >> n >> m >> k >> d;//方格图大小、分店数量、客户数量、障碍数量

    vector<Node> v;

    for (int i = 0; i < m; i++)//录入分店

    {

        int x, y;

        scanf("%d %d", &x, &y);

        q.push(Node(x, y, 0));

        visited[x][y] = 1;

    }

    for (int i = 0; i < k; i++)//录入客户

    {

        int x, y, n;

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

        v.push\_back(Node(x, y, n));

    }

    for (int i = 0; i < d; i++)//录入障碍

    {

        int x, y;

        scanf("%d %d", &x, &y);

        visited[x][y] = 1;

    }

    while (!q.empty())

    {

        Node front = q.front(); q.pop();

        if (front.x - 1 >= 1 && !visited[front.x - 1][front.y])//上

        {

            visited[front.x - 1][front.y] = 1;

            matrix[front.x - 1][front.y] = front.num + 1;

            q.push(Node(front.x - 1, front.y, front.num + 1));

        }

        if (front.x + 1 <= n && !visited[front.x + 1][front.y])//下

        {

            visited[front.x + 1][front.y] = 1;

            matrix[front.x + 1][front.y] = front.num + 1;

            q.push(Node(front.x + 1, front.y, front.num + 1));

        }

        if (front.y - 1 >= 1 && !visited[front.x][front.y - 1])//左

        {

            visited[front.x][front.y - 1] = 1;

            matrix[front.x][front.y - 1] = front.num + 1;

            q.push(Node(front.x, front.y - 1, front.num + 1));

        }

        if (front.y + 1 <= n && !visited[front.x][front.y + 1])//右

        {

            visited[front.x][front.y + 1] = 1;

            matrix[front.x][front.y + 1] = front.num + 1;

            q.push(Node(front.x, front.y + 1, front.num + 1));

        }

    }

    long long ans = 0;

    for (int i = 0; i < v.size(); i++)

    {

        ans = ans + matrix[v[i].x][v[i].y] \* v[i].num;

    }

    cout << ans << endl;

}

## 3.6集合竞价(暴力模拟)

|  |  |
| --- | --- |
| 试题编号： | 201412-3 |
| 试题名称： | 集合竞价 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　某股票交易所请你编写一个程序，根据开盘前客户提交的订单来确定某特定股票的开盘价和开盘成交量。 　　该程序的输入由很多行构成，每一行为一条记录，记录可能有以下几种： 　　1. buy p s 表示一个购买股票的买单，每手出价为p，购买股数为s。 　　2. sell p s 表示一个出售股票的卖单，每手出价为p，出售股数为s。 　　3. cancel i表示撤销第i行的记录。 　　如果开盘价为p0，则系统可以将所有出价至少为p0的买单和所有出价至多为p0的卖单进行匹配。因此，此时的开盘成交量为出价至少为p0的买单的总股数和所有出价至多为p0的卖单的总股数之间的较小值。 　　你的程序需要确定一个开盘价，使得开盘成交量尽可能地大。如果有多个符合条件的开盘价，你的程序应当输出最高的那一个。  **输入格式**  　　输入数据有任意多行，每一行是一条记录。保证输入合法。股数为不超过108的正整数，出价为精确到恰好小数点后两位的正实数，且不超过10000.00。  **输出格式**  　　你需要输出一行，包含两个数，以一个空格分隔。第一个数是开盘价，第二个是此开盘价下的成交量。开盘价需要精确到小数点后恰好两位。  **样例输入**  buy 9.25 100 buy 8.88 175 sell 9.00 1000 buy 9.00 400 sell 8.92 400 cancel 1 buy 100.00 50  **样例输出**  9.00 450  **评测用例规模与约定**  　　对于100%的数据，输入的行数不超过5000。 |

#include<iostream>

#include<string>

#include<vector>

#include<stdio.h>

#include<set>

#include<algorithm>

using namespace std;

struct Node

{

    string type;

    double price;

    int num;

    bool valid;//记录是否有效

    Node() { valid = true; };

};

struct Record

{

    double price;

    int num;

    bool type;

    Record() {};

    Record(double price1, int num1, bool type1) :price(price1), num(num1), type(type1) {};

};

int main(void)

{

    vector<Node> v(5001);//保存记录

    string line;

    double price; int num;

    int index = 1;

    /\*读取记录\*/

    while (cin>>line)

    {

        if (line == "buy")

        {

            cin >> v[index].price >> v[index].num;

            v[index].type = "buy";

        }

        else if (line == "sell")

        {

            cin >> v[index].price >> v[index].num;

            v[index].type = "sell";

        }

        else

        {

            cin >> v[index].num;

            v[index].type = "cancel";

            v[v[index].num].valid = false;

        }

        index++;

    }

    /\*处理记录\*/

    vector<Record> R;//保存有效交易记录

    set<double> s;//保存可能的开盘价

    for (int i = 1; i <index; i++)

    {

        if (v[i].valid)

        {

            if (v[i].type == "buy")//有效的购买记录

            {

                R.push\_back(Record(v[i].price, v[i].num, 0));

                s.insert(v[i].price);

            }

            else if (v[i].type == "sell")//有效的售出记录

            {

                R.push\_back(Record(v[i].price, v[i].num, 1));

                s.insert(v[i].price);

            }

        }

    }

    /\*计算开盘价\*/

    double ans = 0;//最高开盘价

    long long max\_num = 0;//成交数

    double cur\_ans;//当前开盘价

    long long cur\_num;//当前成交记录

    set<double>::iterator it = s.begin();

    while (it != s.end())

    {

        cur\_ans = \*it;

        long long num1 = 0, num2 = 0;

        for (int i = 0; i < R.size(); i++)

        {

            if (R[i].type == 0)//是买家

            {

                if (R[i].price >= cur\_ans) num1 += R[i].num;

            }

            else//是卖家

            {

                if (R[i].price <= cur\_ans) num2 += R[i].num;

            }

        }

        cur\_num = min(num1, num2);

        if (cur\_num >= max\_num)

        {

            ans = cur\_ans;

            max\_num = cur\_num;

        }

        it++;

    }

    printf("%.2lf %lld", ans, max\_num);

    return 0;

}

## 3.7 Z字形扫描(找规律扫描)

|  |  |
| --- | --- |
| 试题编号： | 201412-2 |
| 试题名称： | Z字形扫描 |
| 时间限制： | 2.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　在图像编码的算法中，需要将一个给定的方形矩阵进行Z字形扫描(Zigzag Scan)。给定一个n×n的矩阵，Z字形扫描的过程如下图所示： http://118.190.20.162/RequireFile.do?fid=Nad2EgMG 　　对于下面的4×4的矩阵， 　　1 5 3 9 　　3 7 5 6 　　9 4 6 4 　　7 3 1 3 　　对其进行Z字形扫描后得到长度为16的序列： 　　1 5 3 9 7 3 9 5 4 7 3 6 6 4 1 3 　　请实现一个Z字形扫描的程序，给定一个n×n的矩阵，输出对这个矩阵进行Z字形扫描的结果。  **输入格式**  　　输入的第一行包含一个整数n，表示矩阵的大小。 　　输入的第二行到第n+1行每行包含n个正整数，由空格分隔，表示给定的矩阵。  **输出格式**  　　输出一行，包含n×n个整数，由空格分隔，表示输入的矩阵经过Z字形扫描后的结果。  **样例输入**  4 1 5 3 9 3 7 5 6 9 4 6 4 7 3 1 3  **样例输出**  1 5 3 9 7 3 9 5 4 7 3 6 6 4 1 3  **评测用例规模与约定**  　　1≤n≤500，矩阵元素为不超过1000的正整数。 |

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int n;

int matrix[501][501];

int main(void)

{

    cin >> n;

    for (int i = 0; i < n; i++)

        for (int j = 0; j < n; j++)

            cin >> matrix[i][j];

    int row, col;

    row = col = 0;

    bool direction = 1;//direction=1,左下到右上。direction=0，右上到左下

    for (int i = 1; i <= n; i++)

    {

        for (int j = 1; j <= i;)

        {

            cout << matrix[row][col] << ' ';

            if (++j > i)

                break;

            if (direction) { row--; col++; }

            else { row++; col--; }

        }

        if (direction) col++;

        else row++;

        direction = !direction;

    }

    if (direction) { row--; col++; }

    else { row++; col--; }

    for (int i = n-1; i >= 1; i--)

    {

        for (int j = 1; j <= i;)

        {

            cout << matrix[row][col] << ' ';

            if (++j > i) break;

            if (direction)

            {

                row--; col++;

            }

            else

            {

                row++; col--;

            }

        }

        if (direction) row++;

        else col++;

        direction = !direction;

    }

}

## 3.8最优灌溉(最下生成树Kruskal)

|  |  |
| --- | --- |
| 试题编号： | 201412-4 |
| 试题名称： | 最优灌溉 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　雷雷承包了很多片麦田，为了灌溉这些麦田，雷雷在第一个麦田挖了一口很深的水井，所有的麦田都从这口井来引水灌溉。 　　为了灌溉，雷雷需要建立一些水渠，以连接水井和麦田，雷雷也可以利用部分麦田作为“中转站”，利用水渠连接不同的麦田，这样只要一片麦田能被灌溉，则与其连接的麦田也能被灌溉。 　　现在雷雷知道哪些麦田之间可以建设水渠和建设每个水渠所需要的费用（注意不是所有麦田之间都可以建立水渠）。请问灌溉所有麦田最少需要多少费用来修建水渠。  **输入格式**  　　输入的第一行包含两个正整数n, m，分别表示麦田的片数和雷雷可以建立的水渠的数量。麦田使用1, 2, 3, ……依次标号。 　　接下来m行，每行包含三个整数ai, bi, ci，表示第ai片麦田与第bi片麦田之间可以建立一条水渠，所需要的费用为ci。  **输出格式**  　　输出一行，包含一个整数，表示灌溉所有麦田所需要的最小费用。  **样例输入**  4 4 1 2 1 2 3 4 2 4 2 3 4 3  **样例输出**  6  **样例说明**  　　建立以下三条水渠：麦田1与麦田2、麦田2与麦田4、麦田4与麦田3。  **评测用例规模与约定**  　　前20%的评测用例满足：n≤5。 　　前40%的评测用例满足：n≤20。 　　前60%的评测用例满足：n≤100。 　　所有评测用例都满足：1≤n≤1000，1≤m≤100,000，1≤ci≤10,000。 |

#include <iostream>

#include <vector>

#include <algorithm>

#include <stdio.h>

using namespace std;

struct node

{

    int a, b;

    int cost;

    node() {};

    node(int &a1, int &b1, int &cost1) : a(a1), b(b1), cost(cost1) {};

    bool operator<(node &node)

    {

        return cost < node.cost;

    }

};

int find(vector<int> &root, int x)

{

    if (root[x] == -1)

        return x;

    else

        return root[x] = find(root, root[x]);

}

int main(void)

{

    int n, m;

    cin >> n >> m;

    vector<int> root(n + 1, -1);

    vector<node> g(m);

    int i;

    for (i = 0; i < m; i++)

    {

        scanf("%d %d %d", &g[i].a, &g[i].b, &g[i].cost);

    }

    sort(g.begin(), g.end());

    int num = 0;

    int total\_cost = 0;

    for (int i = 0; i < g.size(); i++)

    {

        int roota, rootb;

        roota = find(root, g[i].a);

        rootb = find(root, g[i].b);

        if (roota == rootb)

            continue;

        else

        {

            root[roota] = rootb;

            total\_cost += g[i].cost;

            num++;

            if (num == n - 1) break;

        }

    }

    cout << total\_cost << endl;

}

## 3.9节日(日期计算)

|  |  |
| --- | --- |
| 试题编号： | 201503-3 |
| 试题名称： | 节日 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　有一类节日的日期并不是固定的，而是以“*a*月的第*b*个星期*c*”的形式定下来的，比如说母亲节就定为每年的五月的第二个星期日。 　　现在，给你*a*，*b*，*c*和*y*1, *y*2(1850 ≤ *y*1, *y*2 ≤ 2050)，希望你输出从公元*y*1年到公元*y*2年间的每年的*a*月的第*b*个星期*c*的日期。 　　提示：关于闰年的规则：年份是400的整数倍时是闰年，否则年份是4的倍数并且不是100的倍数时是闰年，其他年份都不是闰年。例如1900年就不是闰年，而2000年是闰年。 　　为了方便你推算，已知1850年1月1日是星期二。  **输入格式**  　　输入包含恰好一行，有五个整数*a*, *b*, *c*, *y*1, *y*2。其中*c*=1, 2, ……, 6, 7分别表示星期一、二、……、六、日。  **输出格式**  　　对于*y*1和*y*2之间的每一个年份，包括*y*1和*y*2，按照年份从小到大的顺序输出一行。 　　如果该年的*a*月第*b*个星期*c*确实存在，则以"yyyy/mm/dd"的格式输出，即输出四位数的年份，两位数的月份，两位数的日期，中间用斜杠“/”分隔，位数不足时前补零。 　　如果该年的*a*月第*b*个星期*c*并不存在，则输出"none"（不包含双引号)。  **样例输入**  5 2 7 2014 2015  **样例输出**  2014/05/11 2015/05/10  **评测用例规模与约定**  　　所有评测用例都满足：1 ≤ *a* ≤ 12，1 ≤ *b* ≤ 5，1 ≤ *c* ≤ 7，1850 ≤ *y*1, *y*2 ≤ 2050。 |

#include <iostream>

#include <stdio.h>

using namespace std;

int months[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

int months1[] = { 0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

int isSpecialYear(int year)

{

    if (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0))

        return 366;

    return 365;

}

int main(void)

{

    int a, b, c, y1, y2; //a月的第b个星期c

    cin >> a >> b >> c >> y1 >> y2;

    int days = 0; //y年1月1日距离1850.1.1的天数

    for (int i = 1850; i < y1; i++)

        days += isSpecialYear(i);

    c %= 7;

    for (int y = y1; y <= y2; y++)

    {

        int temp = days;

        if (isSpecialYear(y) == 365)

        {

            for (int i = 1; i < a; i++)

                temp += months[i]; //y1年a月1日距离1850.1.1的天数

        }

        else

        {

            for (int i = 1; i < a; i++)

                temp += months1[i]; //y1年a月1日距离1850.1.1的天数

        }

        temp += 2;

        temp %= 7; //计算出y1年a月1日的星期数

        int m\_day = 1; //月中的第几天

        while (temp != c)

        {

            m\_day++;

            temp = (temp + 1) % 7;

        }

        m\_day += (b - 1) \* 7;

        if (isSpecialYear(y) == 366)

        {

            if (m\_day > months1[a])

                cout << "none" << endl;

            else printf("%04d/%02d/%02d\n", y, a, m\_day);

        }

        else

        {

            if (m\_day > months[a])

                cout << "none" << endl;

            else printf("%04d/%02d/%02d\n", y, a, m\_day);

        }

        days += isSpecialYear(y);

    }

}

## 3.10网络延时(树的直径+2次DFS)

|  |  |
| --- | --- |
| 试题编号： | 201503-4 |
| 试题名称： | 网络延时 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　给定一个公司的网络，由*n*台交换机和*m*台终端电脑组成，交换机与交换机、交换机与电脑之间使用网络连接。交换机按层级设置，编号为1的交换机为根交换机，层级为1。其他的交换机都连接到一台比自己上一层的交换机上，其层级为对应交换机的层级加1。所有的终端电脑都直接连接到交换机上。 　　当信息在电脑、交换机之间传递时，每一步只能通过自己传递到自己所连接的另一台电脑或交换机。请问，电脑与电脑之间传递消息、或者电脑与交换机之间传递消息、或者交换机与交换机之间传递消息最多需要多少步。  **输入格式**  　　输入的第一行包含两个整数*n*, *m*，分别表示交换机的台数和终端电脑的台数。 　　第二行包含*n* - 1个整数，分别表示第2、3、……、*n*台交换机所连接的比自己上一层的交换机的编号。第*i*台交换机所连接的上一层的交换机编号一定比自己的编号小。 　　第三行包含*m*个整数，分别表示第1、2、……、*m*台终端电脑所连接的交换机的编号。  **输出格式**  　　输出一个整数，表示消息传递最多需要的步数。  **样例输入**  4 2 1 1 3 2 1  **样例输出**  4  **样例说明**  　　样例的网络连接模式如下，其中圆圈表示交换机，方框表示电脑： http://118.190.20.162/RequireFile.do?fid=F9GfBRHL 　　其中电脑1与交换机4之间的消息传递花费的时间最长，为4个单位时间。  **样例输入**  4 4 1 2 2 3 4 4 4  **样例输出**  4  **样例说明**  　　样例的网络连接模式如下： http://118.190.20.162/RequireFile.do?fid=LYDFDEbt 　　其中电脑1与电脑4之间的消息传递花费的时间最长，为4个单位时间。  **评测用例规模与约定**  　　前30%的评测用例满足：*n* ≤ 5, *m* ≤ 5。 　　前50%的评测用例满足：*n* ≤ 20, *m* ≤ 20。 　　前70%的评测用例满足：*n* ≤ 100, *m* ≤ 100。 　　所有评测用例都满足：1 ≤ *n* ≤ 10000，1 ≤ *m* ≤ 10000。 |

#include<iostream>

#include<vector>

#include<memory.h>

using namespace std;

int n, m;//交换机、台式机个数

bool visited[20001];

int DFS(vector<vector<int> > &G, int index, int &seq)//返回最深深度和该深度下的终点

{

    visited[index] = true;

    int deep = 0, num = index;

    for (int i = 0; i < G[index].size(); i++)

    {

        if (!visited[G[index][i]])

        {

            int deep1, num1;

            deep1 = 1 + DFS(G, G[index][i], num1);

            if (deep < deep1)

            {

                deep = deep1;

                num = num1;

            }

        }

    }

    seq = num;

    return deep;

}

int main(void)

{

    cin >> n >> m;

    vector<vector<int> > G(m + n + 1);

    //交换机编号1-n，台式机编号n+1-n+m

    for (int i = 2; i <= n + m; i++)

    {

        int a;

        scanf("%d", &a);

        G[i].push\_back(a);

        G[a].push\_back(i);

    }

    int num, deep;

    DFS(G, 1, num);

    memset(visited, 0, sizeof(visited));

    deep = DFS(G, num, num);

    cout << deep << endl;

}

## 3.11模板生成系统(字符串查找+替换)

|  |  |
| --- | --- |
| 试题编号： | 201509-3 |
| 试题名称： | 模板生成系统 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　成成最近在搭建一个网站，其中一些页面的部分内容来自数据库中不同的数据记录，但是页面的基本结构是相同的。例如，对于展示用户信息的页面，当用户为 Tom 时，网页的源代码是 http://118.190.20.162/RequireFile.do?fid=Mbg3eNaq  　　而当用户为 Jerry 时，网页的源代码是 http://118.190.20.162/RequireFile.do?fid=69mbB6TA  　　这样的例子在包含动态内容的网站中还有很多。为了简化生成网页的工作，成成觉得他需要引入一套模板生成系统。 　　模板是包含特殊标记的文本。成成用到的模板只包含一种特殊标记，格式为 {{ VAR }}，其中 VAR 是一个变量。该标记在模板生成时会被变量 VAR 的值所替代。例如，如果变量 name = "Tom"，则 {{ name }} 会生成 Tom。具体的规则如下： 　　·变量名由大小写字母、数字和下划线 (\_) 构成，且第一个字符不是数字，长度不超过 16 个字符。 　　·变量名是大小写敏感的，Name 和 name 是两个不同的变量。 　　·变量的值是字符串。 　　·如果标记中的变量没有定义，则生成空串，相当于把标记从模板中删除。 　　·模板不递归生成。也就是说，如果变量的值中包含形如 {{ VAR }} 的内容，不再做进一步的替换。  **输入格式**  　　输入的第一行包含两个整数 *m*, *n*，分别表示模板的行数和模板生成时给出的变量个数。 　　接下来 *m* 行，每行是一个字符串，表示模板。 　　接下来 *n* 行，每行表示一个变量和它的值，中间用一个空格分隔。值是字符串，用双引号 (") 括起来，内容可包含除双引号以外的任意可打印 ASCII 字符（ASCII 码范围 32, 33, 35-126）。  **输出格式**  　　输出包含若干行，表示模板生成的结果。  **样例输入**  11 2 <!DOCTYPE html> <html> <head> <title>User {{ name }}</title> </head> <body> <h1>{{ name }}</h1> <p>Email: <a href="mailto:{{ email }}">{{ email }}</a></p> <p>Address: {{ address }}</p> </body> </html> name "David Beckham" email "david@beckham.com"  **样例输出**  <!DOCTYPE html> <html> <head> <title>User David Beckham</title> </head> <body> <h1>David Beckham</h1> <p>Email: <a href="mailto:david@beckham.com">david@beckham.com</a></p> <p>Address: </p> </body> </html>  **评测用例规模与约定**  　　0 ≤ *m* ≤ 100 　　0 ≤ *n* ≤ 100 　　输入的模板每行长度不超过 80 个字符（不包含换行符）。 　　输入保证模板中所有以 {{ 开始的子串都是合法的标记，开始是两个左大括号和一个空格，然后是变量名，结尾是一个空格和两个右大括号。 　　输入中所有变量的值字符串长度不超过 100 个字符（不包括双引号）。 　　保证输入的所有变量的名字各不相同。 |

#include<string>

#include<vector>

#include<iostream>

#include<map>

using namespace std;

int main(void)

{

    int m, n;

    cin >> m >> n;

    getchar();

    vector<string> v(m);//保存读入的字符串

    map<string, string> mp;//变量映射

    for (int i = 0; i < m; i++)

    {

        getline(cin, v[i]);

    }

    string line, s1, s2;

    for (int i = 0; i < n; i++)

    {

        getline(cin, line);

        int pos = line.find(' ');

        string s1, s2;

        s1 = line.substr(0, pos);

        s2 = line.substr(pos + 1, line.size() - pos - 1);

        s2 = s2.substr(1, s2.length() - 2);//取出引号

        mp[s1] = s2;//插入变量

    }

    for (int i = 0; i < v.size(); i++)

    {

        string replace;

        int begin, end;

        begin = end = 0;

        begin = v[i].find("{{ ");

        if (begin == -1) continue;//没有变量

        replace = v[i].substr(0, begin);

        while (begin!=-1)

        {

            end = v[i].find(" }}", begin);

            string key = v[i].substr(begin + 3, end - begin - 3);

            replace += mp[key];

            begin = v[i].find("{{ ", end);

            end += 3;

            if (begin == -1) replace += v[i].substr(end);

            else replace += v[i].substr(end, begin - end);

        }

        v[i].swap(replace);

    }

    for (int i = 0; i < v.size(); i++)

        cout << v[i] << endl;

    //system("pause");

}

## 3.12高速公路(强连通分量)

|  |  |
| --- | --- |
| 试题编号： | 201509-4 |
| 试题名称： | 高速公路 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　某国有*n*个城市，为了使得城市间的交通更便利，该国国王打算在城市之间修一些高速公路，由于经费限制，国王打算第一阶段先在部分城市之间修一些单向的高速公路。 　　现在，大臣们帮国王拟了一个修高速公路的计划。看了计划后，国王发现，有些城市之间可以通过高速公路直接（不经过其他城市）或间接（经过一个或多个其他城市）到达，而有的却不能。如果城市A可以通过高速公路到达城市B，而且城市B也可以通过高速公路到达城市A，则这两个城市被称为便利城市对。 　　国王想知道，在大臣们给他的计划中，有多少个便利城市对。  **输入格式**  　　输入的第一行包含两个整数*n*, *m*，分别表示城市和单向高速公路的数量。 　　接下来*m*行，每行两个整数*a*, *b*，表示城市*a*有一条单向的高速公路连向城市*b*。  **输出格式**  　　输出一行，包含一个整数，表示便利城市对的数量。  **样例输入**  5 5 1 2 2 3 3 4 4 2 3 5  **样例输出**  3  **样例说明**  http://118.190.20.162/RequireFile.do?fid=4HG9GgbF 　　城市间的连接如图所示。有3个便利城市对，它们分别是(2, 3), (2, 4), (3, 4)，请注意(2, 3)和(3, 2)看成同一个便利城市对。  **评测用例规模与约定**  　　前30%的评测用例满足1 ≤ *n* ≤ 100, 1 ≤ *m* ≤ 1000； 　　前60%的评测用例满足1 ≤ *n* ≤ 1000, 1 ≤ *m* ≤ 10000； 　　所有评测用例满足1 ≤ *n* ≤ 10000, 1 ≤ *m* ≤ 100000。 |

#include<iostream>

#include<stack>

#include<vector>

using namespace std;

vector<int> visited;

vector<int> ans;

void DFS\_G(vector<vector<int> >&G, int src, stack<int> &s)//深搜原图

{

    visited[src] = true;

    for (int i = 0; i < G[src].size(); i++)

    {

        if (!visited[G[src][i]])

            DFS\_G(G, G[src][i], s);

    }

    s.push(src);

}

void DFS\_GT(vector<vector<int> > &GT, int src, stack<int> &s, int flag)//深搜逆图

{

    visited[src] = true;

    ans[flag]++;

    for (int i = 0; i < GT[src].size(); i++)

    {

        if (!visited[GT[src][i]])

            DFS\_GT(GT, GT[src][i], s, flag);

    }

}

int main(void)

{

    int n, m;//城市数量+单向路数量

    cin >> n >> m;

    vector<vector<int> > G(n + 1), GT(n + 1);//图G和图G的逆图GT

    for (int i = 0; i < m; i++)

    {

        int a, b;

        cin >> a >> b;

        G[a].push\_back(b); GT[b].push\_back(a);

    }

    stack<int> s;

    visited.assign(n + 1, 0);

    for (int i = 1; i <=n; i++)

    {

        if (!visited[i])

            DFS\_G(G, i, s);

    }

    ans.assign(n + 1, 0);

    visited.assign(n + 1, 0);

    int index = 1;

    while (!s.empty())

    {

        int i = s.top(); s.pop();

        if (!visited[i])

        {

            DFS\_GT(GT, i, s, index);

            index++;

        }

    }

    long long res = 0;

    for (int i = 1; i < index; i++)

        res += ans[i] \* (ans[i] - 1) / 2;

    cout << res;

}

## 3.12送货(欧拉路径+并查集判断连通性)

|  |  |
| --- | --- |
| 试题编号： | 201512-4 |
| 试题名称： | 送货 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　为了增加公司收入，F公司新开设了物流业务。由于F公司在业界的良好口碑，物流业务一开通即受到了消费者的欢迎，物流业务马上遍及了城市的每条街道。然而，F公司现在只安排了小明一个人负责所有街道的服务。 　　任务虽然繁重，但是小明有足够的信心，他拿到了城市的地图，准备研究最好的方案。城市中有*n*个交叉路口，*m*条街道连接在这些交叉路口之间，每条街道的首尾都正好连接着一个交叉路口。除开街道的首尾端点，街道不会在其他位置与其他街道相交。每个交叉路口都至少连接着一条街道，有的交叉路口可能只连接着一条或两条街道。 　　小明希望设计一个方案，从编号为1的交叉路口出发，每次必须沿街道去往街道另一端的路口，再从新的路口出发去往下一个路口，直到所有的街道都经过了正好一次。  **输入格式**  　　输入的第一行包含两个整数*n*, *m*，表示交叉路口的数量和街道的数量，交叉路口从1到*n*标号。 　　接下来*m*行，每行两个整数*a*, *b*，表示和标号为*a*的交叉路口和标号为*b*的交叉路口之间有一条街道，街道是双向的，小明可以从任意一端走向另一端。两个路口之间最多有一条街道。  **输出格式**  　　如果小明可以经过每条街道正好一次，则输出一行包含*m*+1个整数*p*1, *p*2, *p*3, ..., *pm*+1，表示小明经过的路口的顺序，相邻两个整数之间用一个空格分隔。如果有多种方案满足条件，则输出字典序最小的一种方案，即首先保证*p*1最小，*p*1最小的前提下再保证*p*2最小，依此类推。 　　如果不存在方案使得小明经过每条街道正好一次，则输出一个整数-1。  **样例输入**  4 5 1 2 1 3 1 4 2 4 3 4  **样例输出**  1 2 4 1 3 4  **样例说明**  　　城市的地图和小明的路径如下图所示。 http://118.190.20.162/RequireFile.do?fid=HgNYQ5G9  **样例输入**  4 6 1 2 1 3 1 4 2 4 3 4 2 3  **样例输出**  -1  **样例说明**  　　城市的地图如下图所示，不存在满足条件的路径。 http://118.190.20.162/RequireFile.do?fid=67NLAqAY  **评测用例规模与约定**  　　前30%的评测用例满足：1 ≤ *n* ≤ 10, *n*-1 ≤ *m* ≤ 20。 　　前50%的评测用例满足：1 ≤ *n* ≤ 100, *n*-1 ≤ *m* ≤ 10000。 　　所有评测用例满足：1 ≤ *n* ≤ 10000，*n*-1 ≤ *m* ≤ 100000。 |

#include<iostream>

#include<vector>

#include<set>

#include<stack>

using namespace std;

vector<int> root(10001, -1);//并查集判断连通性

bool visited[10001][10001];

int num = 0;//统计走过的街道数量

int find(int x)

{

    if (root[x] == -1) return x;

    else return root[x] = find(root[x]);

}

void DFS(vector<set<int> > &G, int src,stack<int> &s)

{

    set<int>::iterator it = G[src].begin();

    while (it!=G[src].end())

    {

        if (!visited[src][\*it])

        {

            visited[src][\*it] = visited[\*it][src] = true;

            DFS(G, \*it, s);

            num++;

        }

        it++;

    }

    s.push(src);

}

int main(void)

{

    int n, m;

    cin >> n >> m;//路口数量和街道数量

    vector<set<int> > G(n + 1);//保存图，并且使邻接点有序

    vector<int> degree(n + 1);//计算结点的度

    for (int i = 0; i < m; i++)

    {

        int a, b;

        cin >> a >> b;

        int roota, rootb;

        roota = find(a); rootb = find(b);

        if (roota != rootb) root[roota] = rootb;//并查集合并

        G[a].insert(b); G[b].insert(a);

        degree[a]++; degree[b]++;

    }

    /\*判断连通性\*/

    int flag = find(1);

    for (int i = 1; i <= n; i++)

    {

        if (find(i) != flag)//不连通

        {

            cout << -1 << endl;

            return 0;

        }

    }

    int cnt = 0;//统计奇数度的结点

    int even[2];//保存奇数度的结点

    for(int i=1;i<=n;i++)

        if (degree[i] % 2 == 1)

        {

            even[cnt] = i;

            cnt++;

            if (cnt == 3) break;

        }

    if (cnt == 3 || cnt == 1) cout << -1 << endl;//不存在欧拉路径

    else if (cnt == 2 && (even[0] != 1 && even[1] != 1)) cout << -1 << endl;//1号路口的度不为奇数

    else

    {

        stack<int> s;

        DFS(G, 1, s);

        while (!s.empty())

        {

            cout << s.top() << ' '; s.pop();

        }

    }

    return 0;

}

## 3.13路径解析(字符串处理)

|  |  |
| --- | --- |
| 试题编号： | 201604-3 |
| 试题名称： | 路径解析 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　在操作系统中，数据通常以文件的形式存储在文件系统中。文件系统一般采用层次化的组织形式，由目录（或者文件夹）和文件构成，形成一棵树的形状。文件有内容，用于存储数据。目录是容器，可包含文件或其他目录。同一个目录下的所有文件和目录的名字各不相同，不同目录下可以有名字相同的文件或目录。 　　为了指定文件系统中的某个文件，需要用**路径**来定位。在类 Unix 系统（Linux、Max OS X、FreeBSD等）中，路径由若干**部分**构成，每个部分是一个目录或者文件的名字，相邻两个部分之间用 / 符号分隔。 　　有一个特殊的目录被称为**根目录**，是整个文件系统形成的这棵树的根节点，用一个单独的 / 符号表示。在操作系统中，有**当前目录**的概念，表示用户目前正在工作的目录。根据出发点可以把路径分为两类： 　　 **绝对路径**：以 / 符号开头，表示从根目录开始构建的路径。 　　 **相对路径**：不以 / 符号开头，表示从当前目录开始构建的路径。  　　例如，有一个文件系统的结构如下图所示。在这个文件系统中，有根目录 / 和其他普通目录 d1、d2、d3、d4，以及文件 f1、f2、f3、f1、f4。其中，两个 f1 是同名文件，但在不同的目录下。 http://118.190.20.162/RequireFile.do?fid=aMLNafRq 　　对于 d4 目录下的 f1 文件，可以用绝对路径 /d2/d4/f1 来指定。如果当前目录是 /d2/d3，这个文件也可以用相对路径 ../d4/f1 来指定，这里 .. 表示上一级目录（注意，根目录的上一级目录是它本身）。还有 . 表示本目录，例如 /d1/./f1 指定的就是 /d1/f1。注意，如果有多个连续的 / 出现，其效果等同于一个 /，例如 /d1///f1 指定的也是 /d1/f1。 　　本题会给出一些路径，要求对于每个路径，给出**正规化**以后的形式。一个路径经过正规化操作后，其指定的文件不变，但是会变成一个不包含 . 和 .. 的绝对路径，且不包含连续多个 / 符号。如果一个路径以 / 结尾，那么它代表的一定是一个目录，正规化操作要去掉结尾的 /。若这个路径代表根目录，则正规化操作的结果是 /。若路径为空字符串，则正规化操作的结果是当前目录。  **输入格式**  　　第一行包含一个整数 *P*，表示需要进行正规化操作的路径个数。 　　第二行包含一个字符串，表示当前目录。 　　以下 *P* 行，每行包含一个字符串，表示需要进行正规化操作的路径。  **输出格式**  　　共 *P* 行，每行一个字符串，表示经过正规化操作后的路径，顺序与输入对应。  **样例输入**  7 /d2/d3 /d2/d4/f1 ../d4/f1 /d1/./f1 /d1///f1 /d1/ /// /d1/../../d2  **样例输出**  /d2/d4/f1 /d2/d4/f1 /d1/f1 /d1/f1 /d1 / /d2  **评测用例规模与约定**  　　1 ≤ *P* ≤ 10。 　　文件和目录的名字只包含大小写字母、数字和小数点 .、减号 - 以及下划线 \_。 　　不会有文件或目录的名字是 . 或 .. ，它们具有题目描述中给出的特殊含义。 　　输入的所有路径每个长度不超过 1000 个字符。 　　输入的当前目录保证是一个经过正规化操作后的路径。 　　对于前 30% 的测试用例，需要正规化的路径的组成部分不包含 . 和 .. 。 　　对于前 60% 的测试用例，需要正规化的路径都是绝对路径。 |

#include<iostream>

#include<string>

#include<vector>

#include<sstream>

using namespace std;

string cur;

int main(void)

{

    int p;

    string line;

    cin >> p;

    cin >> cur;//获取当前目录

    getchar();

    for (int i = 0; i < cur.size(); i++)

        if (cur[i] == '/')

            cur[i] = ' ';

    vector<string> curdir;//拆分当前目录

    istringstream is(cur);

    string temp;

    while (is>>temp)

    {

        curdir.push\_back(temp);

    }

    while (p--)

    {

        getline(cin, line);

        vector<string> v;

        if (line == "")

        {

            v = curdir;

        }

        else {

            if (line[0] != '/') v = curdir;//使用相对路径

            for (int i = 0; i < line.size(); i++)

                if (line[i] == '/')

                    line[i] = ' ';

            istringstream inStr(line);

            while (inStr >> temp)

            {

                if (temp == ".") continue;

                else if (temp == "..") {

                    if (!v.empty())

                        v.pop\_back();

                }

                else v.push\_back(temp);

            }

        }

        if (v.empty()) cout << "/" << endl;

        else

        {

            for (int i = 0; i < v.size(); i++)

                cout << "/" << v[i];

            cout << endl;

        }

    }

    return 0;

}

## 3.14游戏(带时间维的BFS)

|  |  |
| --- | --- |
| 试题编号： | 201604-4 |
| 试题名称： | 游戏 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　小明在玩一个电脑游戏，游戏在一个*n*×*m*的方格图上进行，小明控制的角色开始的时候站在第一行第一列，目标是前往第*n*行第*m*列。 　　方格图上有一些方格是始终安全的，有一些在一段时间是危险的，如果小明控制的角色到达一个方格的时候方格是危险的，则小明输掉了游戏，如果小明的角色到达了第*n*行第*m*列，则小明过关。第一行第一列和第*n*行第*m*列永远都是安全的。 　　每个单位时间，小明的角色必须向上下左右四个方向相邻的方格中的一个移动一格。 　　经过很多次尝试，小明掌握了方格图的安全和危险的规律：每一个方格出现危险的时间一定是连续的。并且，小明还掌握了每个方格在哪段时间是危险的。 　　现在，小明想知道，自己最快经过几个时间单位可以达到第*n*行第*m*列过关。  **输入格式**  　　输入的第一行包含三个整数*n*, *m*, *t*，用一个空格分隔，表示方格图的行数*n*、列数*m*，以及方格图中有危险的方格数量。 　　接下来*t*行，每行4个整数*r*, *c*, *a*, *b*，表示第*r*行第*c*列的方格在第*a*个时刻到第*b*个时刻之间是危险的，包括*a*和*b*。游戏开始时的时刻为0。输入数据保证*r*和*c*不同时为1，而且当*r*为*n*时*c*不为*m*。一个方格只有一段时间是危险的（或者说不会出现两行拥有相同的*r*和*c*）。  **输出格式**  　　输出一个整数，表示小明最快经过几个时间单位可以过关。输入数据保证小明一定可以过关。  **样例输入**  3 3 3 2 1 1 1 1 3 2 10 2 2 2 10  **样例输出**  6  **样例说明**  　　第2行第1列时刻1是危险的，因此第一步必须走到第1行第2列。 　　第二步可以走到第1行第1列，第三步走到第2行第1列，后面经过第3行第1列、第3行第2列到达第3行第3列。  **评测用例规模与约定**  　　前30%的评测用例满足：0 < *n*, *m* ≤ 10，0 ≤ *t* < 99。 　　所有评测用例满足：0 < *n*, *m* ≤ 100，0 ≤ *t* < 9999，1 ≤ *r* ≤ *n*，1 ≤ *c* ≤ *m*，0 ≤ *a* ≤ *b* ≤ 100。 |

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

struct Node {

    int row, col, time;

    Node() {};

    Node(int row1, int col1, int time1) :row(row1), col(col1), time(time1) {};

};

int n, m, t;

bool visited[104][104][304];//visited[i][j][t]表示在时刻t第i行第j列是否被访问（或者是危险的)。最慢到达出口的时间是300

int BFS(int beginRow,int beginCol,int endRow,int endCol)

{

    queue<Node> q;

    q.push(Node(beginRow, beginCol, 0));

    visited[beginRow][beginCol][0] = true;

    while (!q.empty())

    {

        Node front = q.front(); q.pop();

        if (front.row == endRow && front.col == endCol) return front.time;

        //上

        if (front.row - 1 >= 1 && !visited[front.row - 1][front.col][front.time + 1])

        {

            visited[front.row - 1][front.col][front.time + 1] = true;

            q.push(Node(front.row - 1, front.col, front.time + 1));

        }

        //下

        if (front.row + 1 <= n && !visited[front.row + 1][front.col][front.time + 1])

        {

            visited[front.row + 1][front.col][front.time + 1] = true;

            q.push(Node(front.row + 1, front.col, front.time + 1));

        }

        //左

        if (front.col - 1 >= 1 && !visited[front.row][front.col - 1][front.time + 1])

        {

            visited[front.row][front.col - 1][front.time + 1] = true;

            q.push(Node(front.row, front.col - 1, front.time + 1));

        }

        //右

        if (front.col + 1 <= m && !visited[front.row][front.col + 1][front.time + 1])

        {

            visited[front.row][front.col + 1][front.time + 1] = true;

            q.push(Node(front.row, front.col + 1, front.time + 1));

        }

    }

}

int main(void)

{

    cin >> n >> m >> t;

    for (int i = 0; i < t; i++)

    {

        int r, c, a, b;

        cin >> r >> c >> a >> b;

        for (int j = a; j <= b; j++)//设置危险区域

            visited[r][c][j] = true;

    }

    cout << BFS(1, 1, n, m) << endl;

    return 0;

}

## 3.15炉石传说(游戏模拟)

|  |  |
| --- | --- |
| 试题编号： | 201609-3 |
| 试题名称： | 炉石传说 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　《炉石传说：魔兽英雄传》（Hearthstone: Heroes of Warcraft，简称炉石传说）是暴雪娱乐开发的一款集换式卡牌游戏（如下图所示）。游戏在一个战斗棋盘上进行，由两名玩家轮流进行操作，本题所使用的炉石传说游戏的简化规则如下： http://118.190.20.162/RequireFile.do?fid=RT2fdRLm 　　\* 玩家会控制一些**角色**，每个角色有自己的**生命值**和**攻击力**。当生命值小于等于 0 时，该角色**死亡**。角色分为**英雄**和**随从**。 　　\* 玩家各控制一个英雄，游戏开始时，英雄的生命值为 30，攻击力为 0。当英雄死亡时，游戏结束，英雄未死亡的一方获胜。 　　\* 玩家可在游戏过程中召唤随从。棋盘上每方都有 7 个可用于放置随从的空位，从左到右一字排开，被称为**战场**。当随从死亡时，它将被从战场上移除。 　　\* 游戏开始后，两位玩家轮流进行操作，每个玩家的连续一组操作称为一个**回合**。 　　\* 每个回合中，当前玩家可进行零个或者多个以下操作： 　　1) **召唤随从**：玩家召唤一个随从进入战场，随从具有指定的生命值和攻击力。 　　2) **随从攻击**：玩家控制自己的某个随从攻击对手的英雄或者某个随从。 　　3) **结束回合**：玩家声明自己的当前回合结束，游戏将进入对手的回合。该操作一定是一个回合的最后一个操作。 　　\* 当随从攻击时，攻击方和被攻击方会同时对彼此造成等同于自己攻击力的**伤害**。受到伤害的角色的生命值将会减少，数值等同于受到的伤害。例如，随从 *X* 的生命值为 *HX*、攻击力为 *AX*，随从 *Y* 的生命值为 *HY*、攻击力为 *AY*，如果随从 *X* 攻击随从 *Y*，则攻击发生后随从 *X* 的生命值变为 *HX* - *AY*，随从 *Y* 的生命值变为 *HY* - *AX*。攻击发生后，角色的生命值可以为负数。 　　本题将给出一个游戏的过程，要求编写程序模拟该游戏过程并输出最后的局面。  **输入格式**  　　输入第一行是一个整数 *n*，表示操作的个数。接下来 *n* 行，每行描述一个操作，格式如下： 　　<action> <arg1> <arg2> ... 　　其中<action>表示操作类型，是一个字符串，共有 3 种：summon表示召唤随从，attack表示随从攻击，end表示结束回合。这 3 种操作的具体格式如下： 　　\* summon <position> <attack> <health>：当前玩家在位置<position>召唤一个生命值为<health>、攻击力为<attack>的随从。其中<position>是一个 1 到 7 的整数，表示召唤的随从出现在战场上的位置，原来该位置及右边的随从都将顺次向右移动一位。 　　\* attack <attacker> <defender>：当前玩家的角色<attacker>攻击对方的角色 <defender>。<attacker>是 1 到 7 的整数，表示发起攻击的本方随从编号，<defender>是 0 到 7 的整数，表示被攻击的对方角色，0 表示攻击对方英雄，1 到 7 表示攻击对方随从的编号。 　　\* end：当前玩家结束本回合。 　　注意：随从的编号会随着游戏的进程发生变化，当召唤一个随从时，玩家指定召唤该随从放入战场的位置，此时，原来该位置及右边的所有随从编号都会增加 1。而当一个随从死亡时，它右边的所有随从编号都会减少 1。任意时刻，战场上的随从总是从1开始连续编号。  **输出格式**  　　输出共 5 行。 　　第 1 行包含一个整数，表示这 *n* 次操作后（以下称为 *T* 时刻）游戏的胜负结果，1 表示先手玩家获胜，-1 表示后手玩家获胜，0 表示游戏尚未结束，还没有人获胜。 　　第 2 行包含一个整数，表示 *T* 时刻先手玩家的英雄的生命值。 　　第 3 行包含若干个整数，第一个整数 *p* 表示 *T* 时刻先手玩家在战场上存活的随从个数，之后 *p* 个整数，分别表示这些随从在 *T* 时刻的生命值（按照从左往右的顺序）。 　　第 4 行和第 5 行与第 2 行和第 3 行类似，只是将玩家从先手玩家换为后手玩家。  **样例输入**  8 summon 1 3 6 summon 2 4 2 end summon 1 4 5 summon 1 2 1 attack 1 2 end attack 1 1  **样例输出**  0 30 1 2 30 1 2  **样例说明**  　　按照样例输入从第 2 行开始逐行的解释如下： 　　1. 先手玩家在位置 1 召唤一个生命值为 6、攻击力为 3 的随从 A，是本方战场上唯一的随从。 　　2. 先手玩家在位置 2 召唤一个生命值为 2、攻击力为 4 的随从 B，出现在随从 A 的右边。 　　3. 先手玩家回合结束。 　　4. 后手玩家在位置 1 召唤一个生命值为 5、攻击力为 4 的随从 C，是本方战场上唯一的随从。 　　5. 后手玩家在位置 1 召唤一个生命值为 1、攻击力为 2 的随从 D，出现在随从 C 的左边。 　　6. 随从 D 攻击随从 B，双方均死亡。 　　7. 后手玩家回合结束。 　　8. 随从 A 攻击随从 C，双方的生命值都降低至 2。  **评测用例规模与约定**  　　\* 操作的个数0 ≤ *n* ≤ 1000。 　　\* 随从的初始生命值为 1 到 100 的整数，攻击力为 0 到 100 的整数。 　　\* 保证所有操作均合法，包括但不限于： 　　1) 召唤随从的位置一定是合法的，即如果当前本方战场上有 *m* 个随从，则召唤随从的位置一定在 1 到 *m* + 1 之间，其中 1 表示战场最左边的位置，*m* + 1 表示战场最右边的位置。 　　2) 当本方战场有 7 个随从时，不会再召唤新的随从。 　　3) 发起攻击和被攻击的角色一定存在，发起攻击的角色攻击力大于 0。 　　4) 一方英雄如果死亡，就不再会有后续操作。 　　\* 数据约定： 　　前 20% 的评测用例召唤随从的位置都是战场的最右边。 　　前 40% 的评测用例没有 attack 操作。 　　前 60% 的评测用例不会出现随从死亡的情况。 |

#include<iostream>

#include<string>

#include<vector>

using namespace std;

struct Role//角色

{

    int health;//生命值

    int attack;//攻击力

    Role() {};

    Role(int health1, int attack1) :health(health1), attack(attack1) {};

};

struct Player//玩家

{

    vector<Role> role;

    Player()

    {

        role.push\_back(Role(30, 0));//添加英雄

    }

};

int main(void)

{

    Player player[2];//玩家

    int index = 0;//0号玩家为先手玩家，1号玩家为后手玩家

    int n;//n次操作

    cin >> n;

    while (n--)

    {

        string op;

        cin >> op;

        if (op == "summon")

        {

            int a, b, c;

            cin >> a >> b >> c;//在位置a召唤一个攻击力为b，生命值为c的随从

            player[index].role.insert(player[index].role.begin() + a, Role(c, b));

        }

        else if (op == "attack")

        {

            int attacker, defender;

            cin >> attacker >> defender;

            player[!index].role[defender].health -= player[index].role[attacker].attack;

            player[index].role[attacker].health -= player[!index].role[defender].attack;

            if (player[!index].role[defender].health <= 0)//防御者的随从死亡

            {

                if (defender != 0)

                    player[!index].role.erase(player[!index].role.begin() + defender);

                else break;//英雄死亡

            }

            if (player[index].role[attacker].health <= 0)//进攻者的随从死亡

            {

                player[index].role.erase(player[index].role.begin() + attacker);

            }

        }

        else if (op == "end")

        {

            index = !index;//切换玩家

        }

    }

    /\*输出胜负\*/

    if (player[0].role[0].health <= 0) cout << -1 << endl;

    else if (player[1].role[0].health <= 0) cout << 1 << endl;

    else cout << 0 << endl;

    /\*先手玩家\*/

    cout << player[0].role[0].health << endl;

    cout << player[0].role.size() - 1;

    for (int i = 1; i < player[0].role.size(); i++) cout << ' ' << player[0].role[i].health;

    cout << endl;

    /\*后手玩家\*/

    cout << player[1].role[0].health << endl;

    cout << player[1].role.size() - 1;

    for (int i = 1; i < player[1].role.size(); i++) cout << ' ' << player[1].role[i].health;

}

## 3.16交通规划(变形最短路径Dijkstra)

|  |  |
| --- | --- |
| 试题编号： | 201609-4 |
| 试题名称： | 交通规划 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　G国国王来中国参观后，被中国的高速铁路深深的震撼，决定为自己的国家也建设一个高速铁路系统。 　　建设高速铁路投入非常大，为了节约建设成本，G国国王决定不新建铁路，而是将已有的铁路改造成高速铁路。现在，请你为G国国王提供一个方案，将现有的一部分铁路改造成高速铁路，使得任何两个城市间都可以通过高速铁路到达，而且从所有城市乘坐高速铁路到首都的最短路程和原来一样长。请你告诉G国国王在这些条件下最少要改造多长的铁路。  **输入格式**  　　输入的第一行包含两个整数*n*, *m*，分别表示G国城市的数量和城市间铁路的数量。所有的城市由1到*n*编号，首都为1号。 　　接下来*m*行，每行三个整数*a*, *b*, *c*，表示城市*a*和城市*b*之间有一条长度为*c*的双向铁路。这条铁路不会经过*a*和*b*以外的城市。  **输出格式**  　　输出一行，表示在满足条件的情况下最少要改造的铁路长度。  **样例输入**  4 5 1 2 4 1 3 5 2 3 2 2 4 3 3 4 2  **样例输出**  11  **评测用例规模与约定**  　　对于20%的评测用例，1 ≤ *n* ≤ 10，1 ≤ *m* ≤ 50； 　　对于50%的评测用例，1 ≤ *n* ≤ 100，1 ≤ *m* ≤ 5000； 　　对于80%的评测用例，1 ≤ *n* ≤ 1000，1 ≤ *m* ≤ 50000； 　　对于100%的评测用例，1 ≤ *n* ≤ 10000，1 ≤ *m* ≤ 100000，1 ≤ *a*, *b* ≤ n，1 ≤ *c* ≤ 1000。输入保证每个城市都可以通过铁路达到首都。 |

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

struct Node

{

    int v;

    int length;

    Node() {};

    Node(int v1, int length1) :v(v1), length(length1) {};

};

struct QNode

{

    int v;

    int len;

    int cost;

    QNode() {};

    QNode(int v1, int len1, int cost1) :v(v1), len(len1), cost(cost1) {};

};

bool operator<(const QNode &node1, const QNode &node2)

{

    if (node1.len == node2.len)

        return node1.cost > node2.cost;

    return node1.len > node2.len;

}

int n, m;//城市数量，铁路数量

int totalcost;//总花费

vector<int> lenth(10001, 0x3f3f3f3f);

vector<bool> visited(10001, 0);//标记结点是否被访问

void Dijkstra(vector<vector<Node> > &G, int src)

{

    lenth[src] = 0;

    priority\_queue<QNode> q;

    q.push(QNode(src, 0, 0));//源点入队

    while (!q.empty())

    {

        QNode front = q.top(); q.pop();

        if (!visited[front.v])

        {

            visited[front.v] = true; totalcost += front.cost;

            for (int i = 0; i < G[front.v].size(); i++)

            {

                if (!visited[G[front.v][i].v])

                {

                    if (lenth[front.v] + G[front.v][i].length <= lenth[G[front.v][i].v])

                    {

                        lenth[G[front.v][i].v] = lenth[front.v] + G[front.v][i].length;

                        q.push(QNode(G[front.v][i].v, lenth[G[front.v][i].v], G[front.v][i].length));

                    }

                }

            }

        }

    }

}

int main(void)

{

    cin >> n >> m;

    vector<vector<Node> > G(n + 1);

    for (int i = 0; i < m; i++)

    {

        int a, b, c;

        scanf("%d %d %d", &a, &b, &c);

        G[a].push\_back(Node(b, c));

        G[b].push\_back(Node(a, c));

    }

    Dijkstra(G, 1);

    cout << totalcost << endl;

}

## 3.17压缩编码(类似矩阵链乘+动态规划)

|  |  |
| --- | --- |
| 试题编号： | 201612-4 |
| 试题名称： | 压缩编码 |
| 时间限制： | 3.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　给定一段文字，已知单词*a*1, *a*2, …, *an*出现的频率分别*t*1, *t*2, …, *tn*。可以用01串给这些单词编码，即将每个单词与一个01串对应，使得任何一个单词的编码（对应的01串）不是另一个单词编码的前缀，这种编码称为前缀码。 　　使用前缀码编码一段文字是指将这段文字中的每个单词依次对应到其编码。一段文字经过前缀编码后的长度为： 　　*L*=*a*1的编码长度×*t*1+*a*2的编码长度×*t*2+…+ *an*的编码长度×*tn*。 　　定义一个前缀编码为字典序编码，指对于1 ≤ *i*< *n*，*ai*的编码（对应的01串）的字典序在*ai*+1编码之前，即*a*1, *a*2, …, *an*的编码是按字典序升序排列的。 　　例如，文字E A E C D E B C C E C B D B E中， 5个单词A、B、C、D、E出现的频率分别为1, 3, 4, 2, 5，则一种可行的编码方案是A:000, B:001, C:01, D:10, E:11，对应的编码后的01串为1100011011011001010111010011000111，对应的长度*L*为3×1+3×3+2×4+2×2+2×5=34。 　　在这个例子中，如果使用哈夫曼(Huffman)编码，对应的编码方案是A:000, B:01, C:10, D:001, E:11，虽然最终文字编码后的总长度只有33，但是这个编码不满足字典序编码的性质，比如C的编码的字典序不在D的编码之前。 　　在这个例子中，有些人可能会想的另一个字典序编码是A:000, B:001, C:010, D:011, E:1，编码后的文字长度为35。 　　请找出一个字典序编码，使得文字经过编码后的长度*L*最小。在输出时，你只需要输出最小的长度*L*，而不需要输出具体的方案。在上面的例子中，最小的长度*L*为34。  **输入格式**  　　输入的第一行包含一个整数*n*，表示单词的数量。 　　第二行包含*n*个整数，用空格分隔，分别表示*a*1, *a*2, …, *an*出现的频率，即*t*1, *t*2, …, *tn*。请注意*a*1, *a*2, …, *an*具体是什么单词并不影响本题的解，所以没有输入*a*1, *a*2, …, *an*。  **输出格式**  　　输出一个整数，表示文字经过编码后的长度*L*的最小值。  **样例输入**  5 1 3 4 2 5  **样例输出**  34  **样例说明**  　　这个样例就是问题描述中的例子。如果你得到了35，说明你算得有问题，请自行检查自己的算法而不要怀疑是样例输出写错了。  **评测用例规模与约定**  　　对于30%的评测用例，1 ≤ *n* ≤ 10，1 ≤ *ti* ≤ 20； 　　对于60%的评测用例，1 ≤ *n* ≤ 100，1 ≤ *ti* ≤ 100； 　　对于100%的评测用例，1 ≤ *n* ≤ 1000，1 ≤ *ti* ≤ 10000。 |

#include<iostream>

#include<vector>

using namespace std;

int n;

int w[1001][1001];//w[i][j]表示包含单词ai-aj的子树的频率和

int c[1001][1001];//c[i][j]表示包含单词ai-aj的总长度

int main(void)

{

    cin >> n;

    vector<int> frequence(n+1);

    for (int i = 1; i <= n; i++)

        cin >> frequence[i];

    for (int i = 1; i <= n; i++)

    {

        c[i][i] = 0;

        w[i][i] = frequence[i];

    }

    for (int L = 2; L <= n; L++)//包含的单词数从2增长到n

    {

        for (int i = 1; i <= n - L + 1; i++)

        {

            int j = i + L - 1;//子树包含关键字ai-aj

            w[i][j] = w[i][i] + w[i + 1][j];//计算包含关键字ai-aj的子树的频率和

            int min\_cost = INT\_MAX;

            for (int k = i; k < j; k++)//将子树分为ai-ak和ak+1-aj两部分

            {

                int curcost = c[i][k] + c[k + 1][j] + w[i][j];

                if (curcost < min\_cost)

                    min\_cost = curcost;

            }

            c[i][j] = min\_cost;

        }

    }

    cout << c[1][n] << endl;

    return 0;

}

## 3.18地铁修建(Kruskal最小生成树)

|  |  |
| --- | --- |
| 试题编号： | 201703-4 |
| 试题名称： | 地铁修建 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　A市有n个交通枢纽，其中1号和n号非常重要，为了加强运输能力，A市决定在1号到n号枢纽间修建一条地铁。 　　地铁由很多段隧道组成，每段隧道连接两个交通枢纽。经过勘探，有m段隧道作为候选，两个交通枢纽之间最多只有一条候选的隧道，没有隧道两端连接着同一个交通枢纽。 　　现在有n家隧道施工的公司，每段候选的隧道只能由一个公司施工，每家公司施工需要的天数一致。而每家公司最多只能修建一条候选隧道。所有公司同时开始施工。 　　作为项目负责人，你获得了候选隧道的信息，现在你可以按自己的想法选择一部分隧道进行施工，请问修建整条地铁最少需要多少天。  **输入格式**  　　输入的第一行包含两个整数*n*, *m*，用一个空格分隔，分别表示交通枢纽的数量和候选隧道的数量。 　　第2行到第*m*+1行，每行包含三个整数*a*, *b*, *c*，表示枢纽*a*和枢纽*b*之间可以修建一条隧道，需要的时间为*c*天。  **输出格式**  　　输出一个整数，修建整条地铁线路最少需要的天数。  **样例输入**  6 6 1 2 4 2 3 4 3 6 7 1 4 2 4 5 5 5 6 6  **样例输出**  6  **样例说明**  　　可以修建的线路有两种。 　　第一种经过的枢纽依次为1, 2, 3, 6，所需要的时间分别是4, 4, 7，则整条地铁线需要7天修完； 　　第二种经过的枢纽依次为1, 4, 5, 6，所需要的时间分别是2, 5, 6，则整条地铁线需要6天修完。 　　第二种方案所用的天数更少。  **评测用例规模与约定**  　　对于20%的评测用例，1 ≤ *n* ≤ 10，1 ≤ *m* ≤ 20； 　　对于40%的评测用例，1 ≤ *n* ≤ 100，1 ≤ *m* ≤ 1000； 　　对于60%的评测用例，1 ≤ *n* ≤ 1000，1 ≤ *m* ≤ 10000，1 ≤ *c* ≤ 1000； 　　对于80%的评测用例，1 ≤ *n* ≤ 10000，1 ≤ *m* ≤ 100000； 　　对于100%的评测用例，1 ≤ *n* ≤ 100000，1 ≤ *m* ≤ 200000，1 ≤ *a*, *b* ≤ *n*，1 ≤ *c* ≤ 1000000。  　　所有评测用例保证在所有候选隧道都修通时1号枢纽可以通过隧道到达其他所有枢纽。 |

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

struct Node

{

    int a, b, c;

    Node() {};

    Node(int a1, int b1, int c1) :a(a1), b(b1), c(c1) {};

    bool operator<(const Node &node)

    {

        return c < node.c;

    }

};

vector<int> root(100004, -1);

vector<Node> v;//保存所有的路段

int find(int x)

{

    if (root[x] == -1) return x;

    else return root[x] = find(root[x]);

}

int n, m;

int Kruskal()

{

    for (int i = 0; i < v.size(); i++)

    {

        //cout << v[i].a << ' ' << v[i].b << ' ' << v[i].c << endl;

        int roota, rootb;

        roota = find(v[i].a); rootb = find(v[i].b);

        if (roota == rootb)

        {

            continue;

        }

        else

        {

            root[roota] = rootb;

        }

        if (find(1) == find(n))

            return v[i].c;

    }

}

int main(void)

{

    cin >> n >> m;

    for (int i = 0; i < m; i++)

    {

        int a, b, c;

        scanf("%d %d %d", &a, &b, &c);

        v.push\_back(Node(a, b, c));

    }

    sort(v.begin(), v.end());

    cout << Kruskal();

    return 0;

}

## 3.19通信网络(多次DFS)

|  |  |
| --- | --- |
| 试题编号： | 201709-4 |
| 试题名称： | 通信网络 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　某国的军队由*N*个部门组成，为了提高安全性，部门之间建立了*M*条通路，每条通路只能单向传递信息，即一条从部门*a*到部门*b*的通路只能由*a*向*b*传递信息。信息可以通过中转的方式进行传递，即如果*a*能将信息传递到*b*，*b*又能将信息传递到*c*，则*a*能将信息传递到*c*。一条信息可能通过多次中转最终到达目的地。 　　由于保密工作做得很好，并不是所有部门之间都互相知道彼此的存在。只有当两个部门之间可以直接或间接传递信息时，他们才彼此知道对方的存在。部门之间不会把自己知道哪些部门告诉其他部门。 http://118.190.20.162/RequireFile.do?fid=yHg9gf9q 　　上图中给了一个4个部门的例子，图中的单向边表示通路。部门1可以将消息发送给所有部门，部门4可以接收所有部门的消息，所以部门1和部门4知道所有其他部门的存在。部门2和部门3之间没有任何方式可以发送消息，所以部门2和部门3互相不知道彼此的存在。 　　现在请问，有多少个部门知道所有*N*个部门的存在。或者说，有多少个部门所知道的部门数量（包括自己）正好是*N*。  **输入格式**  　　输入的第一行包含两个整数*N*, *M*，分别表示部门的数量和单向通路的数量。所有部门从1到*N*标号。 　　接下来*M*行，每行两个整数*a*, *b*，表示部门*a*到部门*b*有一条单向通路。  **输出格式**  　　输出一行，包含一个整数，表示答案。  **样例输入**  4 4 1 2 1 3 2 4 3 4  **样例输出**  2  **样例说明**  　　部门1和部门4知道所有其他部门的存在。  **评测用例规模与约定**  　　对于30%的评测用例，1 ≤ *N* ≤ 10，1 ≤ *M* ≤ 20； 　　对于60%的评测用例，1 ≤ *N* ≤ 100，1 ≤ *M* ≤ 1000； 　　对于100%的评测用例，1 ≤ *N* ≤ 1000，1 ≤ *M* ≤ 10000。 |

#include<iostream>

#include<vector>

#include<memory.h>

using namespace std;

int visited[1004];

bool know[1004][1004];//know[i][j]表示部门i知道部门j的存在

vector<int> G[1004];//邻接表表示图

int n, m;

void DFS(int s,int root)

{

    visited[s] = true;

    for (int i = 0; i < G[s].size(); i++)

    {

        int adjv = G[s][i];//s的第i个邻接点

        if (!visited[adjv])

        {

            know[root][adjv] = know[adjv][root] = true;

            DFS(adjv, root);

        }

    }

}

int main(void)

{

    cin >> n >> m;

    for (int i = 0; i < m; i++)

    {

        int a, b;

        cin >> a >> b;

        G[a].push\_back(b);

    }

    for (int i = 1; i <= n; i++)

    {

        memset(visited, 0, sizeof(visited));

        DFS(i, i);

    }

    int know\_others = 1;

    int ans = 0;

    for (int i = 1; i <= n; i++)

    {

        for (int j = 1; j <= n; j++)

        {

            if (i == j) continue;

            if (!know[i][j]) {

                know\_others = 0;

                break;

            }

        }

        if (know\_others == 1) ans++;

        know\_others = 1;

    }

    cout << ans << endl;

    return 0;

}

## 3.20行车路线(SPFA+FLOYD)

试题编号：201712-4试题名称：行车路线时间限制：1.0s内存限制：256.0MB问题描述：

**问题描述**

　　小明和小芳出去乡村玩，小明负责开车，小芳来导航。  
　　小芳将可能的道路分为大道和小道。大道比较好走，每走1公里小明会增加1的疲劳度。小道不好走，如果连续走小道，小明的疲劳值会快速增加，连续走*s*公里小明会增加*s*2的疲劳度。  
　　例如：有5个路口，1号路口到2号路口为小道，2号路口到3号路口为小道，3号路口到4号路口为大道，4号路口到5号路口为小道，相邻路口之间的距离都是2公里。如果小明从1号路口到5号路口，则总疲劳值为(2+2)2+2+22=16+2+4=22。  
　　现在小芳拿到了地图，请帮助她规划一个开车的路线，使得按这个路线开车小明的疲劳度最小。

**输入格式**

　　输入的第一行包含两个整数*n*, *m*，分别表示路口的数量和道路的数量。路口由1至*n*编号，小明需要开车从1号路口到*n*号路口。  
　　接下来*m*行描述道路，每行包含四个整数*t*, *a*, *b*, *c*，表示一条类型为*t*，连接*a*与*b*两个路口，长度为*c*公里的双向道路。其中*t*为0表示大道，*t*为1表示小道。保证1号路口和*n*号路口是连通的。

**输出格式**

　　输出一个整数，表示最优路线下小明的疲劳度。

**样例输入**

6 7  
1 1 2 3  
1 2 3 2  
0 1 3 30  
0 3 4 20  
0 4 5 30  
1 3 5 6  
1 5 6 1

**样例输出**

76

**样例说明**

　　从1走小道到2，再走小道到3，疲劳度为52=25；然后从3走大道经过4到达5，疲劳度为20+30=50；最后从5走小道到6，疲劳度为1。总共为76。

**数据规模和约定**

　　对于30%的评测用例，1 ≤ *n* ≤ 8，1 ≤ *m* ≤ 10；  
　　对于另外20%的评测用例，不存在小道；  
　　对于另外20%的评测用例，所有的小道不相交；  
　　对于所有评测用例，1 ≤ *n* ≤ 500，1 ≤ *m* ≤ 105，1 ≤ *a*, *b* ≤ *n*，*t*是0或1，*c*≤ 105。保证答案不超过106。

#include<iostream>

#include<queue>

using namespace std;

#define MAX\_VEX 501

#define MAX 0x3f3f3f3f3f3f3f3f

int vex\_num, arc\_num;

long long G[MAX\_VEX][MAX\_VEX], G1[MAX\_VEX][MAX\_VEX];//大路、小路

long long dis[MAX\_VEX], dis1[MAX\_VEX];//大路疲惫值，小路疲惫值

long long in\_queue[MAX\_VEX];//标记在队列中的顶点

void SPFA(int source);

int main()

{

    scanf("%d %d", &vex\_num, &arc\_num);

    int i, j;//循环

    //初始化图

    for(i=1;i<=vex\_num;i++)

        for (j = 1; j <= vex\_num; j++)

        {

            if (i == j) G[i][j] = G1[i][j] = 0;

            else G[i][j] = G1[i][j] = MAX;

        }

    //创建图

    int t, a, b, c;//边

    while (arc\_num--)

    {

        scanf("%d %d %d %d",&t,&a,&b,&c);

        if (t)//小路

            G1[a][b] = G1[b][a] = c;

        else G[a][b] = G[b][a] = c;

    }

    //FORD算法计算只走小路时两两点间距离

    int u, v, w;

    for(u=1;u<=vex\_num;u++)

        for(v=1;v<=vex\_num;v++)

            for (w = 1; w <= vex\_num; w++)

            {

                if (G1[v][u] + G1[u][w] < G1[v][w])

                    G1[v][w] = G1[v][u] + G1[u][w];

            }

    SPFA(1);//计算1号路口到各路口所需最小疲劳度

    printf("%d", min(dis[vex\_num], dis1[vex\_num]));

    //system("pause");

    return 0;

}

void SPFA(int source)

{

    queue<int> Q;

    int i;

    for (i = 1; i <= vex\_num; i++)

    {

        dis[i] = dis1[i] = MAX;

        in\_queue[i] = 0;

    }

    dis[source] = dis1[source] = 0;

    Q.push(source);

    in\_queue[source] = 1;

    while (!Q.empty())

    {

        int u, v;

        u = Q.front();

        Q.pop();

        in\_queue[u] = 0;

        for (v = 1; v <= vex\_num; v++)//松弛操作

        {

            if (u == v) continue;

            if (dis[u] + G[u][v] < dis[v])//大路+大路

            {

                dis[v] = dis[u] + G[u][v];

                if (!in\_queue[v])

                {

                    Q.push(v);

                    in\_queue[v] = 1;

                }

            }

            if(G1[u][v]!=MAX)//若存在u到v的小路

                    if (dis[u] + G1[u][v]\*G1[u][v] < dis1[v])//大路+小路

                    {

                        dis1[v] = G1[u][v] \* G1[u][v] + dis[u];

                        if (!in\_queue[v])

                        {

                            Q.push(v);

                            in\_queue[v] = 1;

                        }

                    }

            if (dis1[u] + G[u][v] < dis[v])//小路+大路

            {

                dis[v] = dis1[u] + G[u][v];

                if (!in\_queue[v])

                {

                    Q.push(v);

                    in\_queue[v] = 1;

                }

            }

        }

    }

}

## 3.21棋局评估(暴力DFS+博弈搜索)

|  |  |
| --- | --- |
| 试题编号： | 201803-4 |
| 试题名称： | 棋局评估 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　Alice和Bob正在玩井字棋游戏。 　　井字棋游戏的规则很简单：两人轮流往3\*3的棋盘中放棋子，Alice放的是“X”，Bob放的是“O”，Alice执先。当同一种棋子占据一行、一列或一条对角线的三个格子时，游戏结束，该种棋子的持有者获胜。当棋盘被填满的时候，游戏结束，双方平手。 　　Alice设计了一种对棋局评分的方法： 　　- 对于Alice已经获胜的局面，评估得分为(棋盘上的空格子数+1)； 　　- 对于Bob已经获胜的局面，评估得分为 -(棋盘上的空格子数+1)； 　　- 对于平局的局面，评估得分为0；  http://118.190.20.162/RequireFile.do?fid=yrNHga4H 　　例如上图中的局面，Alice已经获胜，同时棋盘上有2个空格，所以局面得分为2+1=3。 　　由于Alice并不喜欢计算，所以他请教擅长编程的你，如果两人都以最优策略行棋，那么当前局面的最终得分会是多少？  **输入格式**  　　输入的第一行包含一个正整数*T*，表示数据的组数。 　　每组数据输入有3行，每行有3个整数，用空格分隔，分别表示棋盘每个格子的状态。0表示格子为空，1表示格子中为“X”，2表示格子中为“O”。保证不会出现其他状态。 　　保证输入的局面合法。(即保证输入的局面可以通过行棋到达，且保证没有双方同时获胜的情况) 　　保证输入的局面轮到Alice行棋。  **输出格式**  　　对于每组数据，输出一行一个整数，表示当前局面的得分。  **样例输入**  3 1 2 1 2 1 2 0 0 0 2 1 1 0 2 1 0 0 2 0 0 0 0 0 0 0 0 0  **样例输出**  3 -4 0  **样例说明**  　　第一组数据： 　　Alice将棋子放在左下角(或右下角)后，可以到达问题描述中的局面，得分为3。 　　3为Alice行棋后能到达的局面中得分的最大值。 　　第二组数据： http://118.190.20.162/RequireFile.do?fid=fEF6hem5  　　Bob已经获胜(如图)，此局面得分为-(3+1)=-4。 　　第三组数据： 　　井字棋中若双方都采用最优策略，游戏平局，最终得分为0。  **数据规模和约定**  　　对于所有评测用例，1 ≤ *T* ≤ 5。 |

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int matrix[4][4];//表示3\*3的棋盘,0为空，1为Alice，2为Bob

bool win()//判断当前棋局下是否胜利

{

    for (int i = 1; i <= 3; i++)

    {

        if (matrix[i][1] != 0 && matrix[i][1] == matrix[i][2] && matrix[i][2] == matrix[i][3]) return true;//同一行

        if (matrix[1][i] != 0 && matrix[1][i] == matrix[2][i] && matrix[2][i] == matrix[3][i]) return true;//同一列

    }

    if (matrix[1][1] != 0 && matrix[1][1] == matrix[2][2] && matrix[2][2] == matrix[3][3]) return true;

    if (matrix[1][3] != 0 && matrix[1][3] == matrix[2][2] && matrix[2][2] == matrix[3][1]) return true;

    return false;

}

int space()//统计棋盘上的空格数

{

    int cnt = 0;

    for (int i = 1; i <= 3; i++)

        for (int j = 1; j <= 3; j++)

            if (matrix[i][j] == 0) cnt++;

    return cnt;

}

int DFS(int role)//role=1,Alice执棋，role=2，Bob执棋

{

    int score = space();//统计空格数

    if (win())//当前棋局下已经分出胜负

    {

        score++;

        if (role == 1)

            return -score;//轮到Alice执棋，但是Bob已经胜利

        else return score;//轮到Bob执棋，但是Alice已经胜利

    }

    else if (score == 0) return 0;//

    int minScore = INT\_MAX, maxScore = INT\_MIN;

    for (int i = 1; i <= 3; i++)//搜索所有可以下棋的位置

        for (int j = 1; j <= 3; j++)

        {

            if (matrix[i][j] == 0)

            {

                matrix[i][j] = role;

                if (role == 1)

                {

                    maxScore = max(maxScore, DFS(2));

                }

                else

                {

                    minScore = min(minScore, DFS(1));

                }

                matrix[i][j] = 0;

            }

        }

    if (role == 1) return maxScore;

    else return minScore;

}

int main(void)

{

    int T;

    cin >> T;

    while (T--)

    {

        for (int i = 1; i <= 3; i++)

            for (int j = 1; j <= 3; j++)

                cin >> matrix[i][j];

        cout << DFS(1) << endl;

    }

    return 0;

}

## 3.22再卖菜(回溯法)

|  |  |
| --- | --- |
| 试题编号： | 201809-4 |
| 试题名称： | 再卖菜 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　在一条街上有n个卖菜的商店，按1至n的顺序排成一排，这些商店都卖一种蔬菜。 　　第一天，每个商店都自己定了一个正整数的价格。店主们希望自己的菜价和其他商店的一致，第二天，每一家商店都会根据他自己和相邻商店的价格调整自己的价格。具体的，每家商店都会将第二天的菜价设置为自己和相邻商店第一天菜价的平均值（用去尾法取整）。 　　注意，编号为1的商店只有一个相邻的商店2，编号为n的商店只有一个相邻的商店n-1，其他编号为i的商店有两个相邻的商店i-1和i+1。 　　给定第二天各个商店的菜价，可能存在不同的符合要求的第一天的菜价，请找到符合要求的第一天菜价中字典序最小的一种。 　　字典序大小的定义：对于两个不同的价格序列(a1, a2, ..., an)和(b1, b2, b3, ..., bn)，若存在i (i>=1), 使得ai<bi，且对于所有j<i，aj=bj，则认为第一个序列的字典序小于第二个序列。  **输入格式**  　　输入的第一行包含一个整数n，表示商店的数量。 　　第二行包含n个正整数，依次表示每个商店第二天的菜价。  **输出格式**  　　输出一行，包含n个正整数，依次表示每个商店第一天的菜价。  **样例输入**  8 2 2 1 3 4 9 10 13  **样例输出**  2 2 2 1 6 5 16 10  **数据规模和约定**  　　对于30%的评测用例，2<=n<=5，第二天每个商店的菜价为不超过10的正整数； 　　对于60%的评测用例，2<=n<=20，第二天每个商店的菜价为不超过100的正整数； 　　对于所有评测用例，2<=n<=300，第二天每个商店的菜价为不超过100的正整数。 　　请注意，以上都是给的第二天菜价的范围，第一天菜价可能会超过此范围。 |

#include<iostream>

#include<vector>

using namespace std;

int visited[301][301][301];

int n;

vector<int> First, Second;//第一天的菜价、第二天的菜价

bool findAns;

bool persuit(int i)//求解第i个店铺的菜价

{

    if (i == n - 1)//是最后一家店铺

    {

        if (n == 2)//只有两家店铺

        {

            int base = 2 \* Second[i-1] - First[i-1];

            for (int k = 0; k < 2; k++)

            {

                if ((base + k + First[i-1]) / 2 == Second[i])

                {

                    findAns = true;

                    First.push\_back(base + k);

                    return true;

                }

            }

            return false;

        }

        else

        {

            int base = 3 \* Second[i - 1] - First[i - 1] - First[i - 2];

            for (int k = 0; k < 3; k++)

            {

                if ((base + k + First[i - 1]) / 2 == Second[i])

                {

                    findAns = true;

                    First.push\_back(base + k);

                    return true;

                }

            }

            return false;

        }

    }

    if (i == 1)//是第二家店铺

    {

        int price;

        for (price = 2 \* Second[0] - First[0]; price <= 2 \* Second[0] - First[0] + 1; price++)

        {

            if (price <= 0) continue;//非法菜价

            First.push\_back(price);

            if (persuit(i + 1)) return true;

            First.pop\_back();

        }

    }

    else

    {

        if (visited[i][First[i - 1]][First[i - 2]])

            return false;

        visited[i][First[i - 1]][First[i - 2]] = true;

        int base = 3 \* Second[i - 1] - First[i - 1] - First[i - 2];

        for (int k = 0; k < 3; k++)

        {

            if (base + k <= 0) continue;

            First.push\_back(base + k);

            if (persuit(i + 1)) return true;

            First.pop\_back();

        }

        return false;

    }

    return false;

}

int main(void)

{

    cin >> n;

    Second.resize(n);//第二天的菜价

    for (int i = 0; i < n; i++)

        cin >> Second[i];

    for (int i = 1; i < 2 \* Second[0]; i++)//第一天的菜价范围是1-2\*Second[0]

    {

        First.push\_back(i);

        if (persuit(1)) break;

        First.pop\_back();

    }

    for (int i = 0; i < n; i++)

        cout << First[i] << ' ';

    cout << endl;

    return 0;

}

## 3.23 A1003 Emergency

### Dijkstra

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

#define MAX\_NODE 510

struct ArcNode{

    int v;//邻接点

    int len;//邻接边长度

    ArcNode(int v1, int len1):v(v1), len(len1){};

};

struct QNode{

    int vex;

    int len;

    QNode(int vex1, int len1):vex(vex1), len(len1){};

};

bool operator<(const QNode &n1, const QNode &n2){

        return n1.len>n2.len;

}

int N, M, C1, C2;

vector<vector<ArcNode> > G(MAX\_NODE);

vector<int> cityRescues(MAX\_NODE);

vector<int> routeNum(MAX\_NODE, 0);//从源点到i的最短路径的条数

vector<int> visited(MAX\_NODE, 0);//标识节点是否被访问

vector<int> len(MAX\_NODE, 0x3f3f3f3f);//源点到i的最短距离

vector<int> num(MAX\_NODE, 0);//从源点到i的营救人员的个数

void dijkstra(){

    priority\_queue<QNode> q;

    routeNum[C1] = 1;

    len[C1] = 0;

    num[C1] = cityRescues[C1];

    q.push(QNode(C1, 0));

    while(!q.empty()){

        QNode top = q.top();

        q.pop();

        if(!visited[top.vex]){

            visited[top.vex] = 1;

            for(int i=0;i<G[top.vex].size();i++){

                int adjVex = G[top.vex][i].v;

                int adjLen = G[top.vex][i].len;

                if(!visited[adjVex]){

                    int newLen = len[top.vex] + adjLen;

                    int newRescues = num[top.vex] + cityRescues[adjVex];

                    if(newLen < len[adjVex]){

                        len[adjVex] = newLen;

                        routeNum[adjVex] = routeNum[top.vex];

                        num[adjVex] = newRescues;

                        q.push(QNode(adjVex, newLen));

                    }else if(newLen == len[adjVex]){

                        routeNum[adjVex] += routeNum[top.vex];

                        if(newRescues > num[adjVex])

                            num[adjVex] = newRescues;

                    }

                }

            }

        }

    }

}

int main(void){

    cin >>N >>M>>C1>>C2;

    //每个城市的营救人员个数

    for(int i=0;i<N;i++){

        cin>>cityRescues[i];

    }

    //道路信息

    for(int i=0;i<M;i++){

        int c1, c2, L;

        cin>>c1>>c2>>L;

        G[c1].push\_back(ArcNode(c2, L));

        G[c2].push\_back(ArcNode(c1, L));

    }

    //求解最短路径

    dijkstra();

    cout << routeNum[C2] <<' '<<num[C2]<<endl;

}

### BellmanFord

#include <queue>

#include <set>

#include <vector>

#include <iostream>

using namespace std;

#define MAX\_NODE 510

struct ArcNode {

    int v; //邻接点

    int len; //邻接边长度

    ArcNode(int v1, int len1)

        : v(v1)

        , len(len1){};

};

int N, M, C1, C2;

vector<vector<ArcNode> > G(MAX\_NODE);

vector<int> cityRescues(MAX\_NODE); //城市i的营救人员个数

vector<int> routeNum(MAX\_NODE, 0); //从源点到i的最短路径的条数

vector<int> len(MAX\_NODE, 0x3f3f3f3f); //源点到i的最短距离

vector<int> totalRescues(MAX\_NODE, 0); //从源点到i的营救人员的个数

set<int> preNode[MAX\_NODE];

void relax(int a, int b, int c)

{

    int newLen = len[a] + c;//源点经由a到达b的距离

    int newRescues = totalRescues[a] + cityRescues[b];//源点经由a到达b的营救人员的个数

    if (newLen < len[b]) {

        //更新距离

        len[b] = newLen;

        //更新路径数

        preNode[b].clear();

        preNode[b].insert(a);

        routeNum[b] = routeNum[a];

        //更新营救人员个数

        totalRescues[b] = newRescues;

    }else if(newLen == len[b]){

        //更新路径数

        preNode[b].insert(a);

        routeNum[b] = 0;

        for(set<int>::iterator it = preNode[b].begin();it!=preNode[b].end();it++){

            routeNum[b] += routeNum[\*it];

        }

        //更新营救人员的个数

        if(newRescues > totalRescues[b]){

            totalRescues[b] = newRescues;

        }

    }

}

void bellmanFord()

{

    //初始化

    len[C1] = 0;

    routeNum[C1] = 1;

    totalRescues[C1] = cityRescues[C1];

    //对每条边执行N-1次松弛操作

    for (int i = 0; i < N - 1; i++) {

        for (int v = 0; v < N; v++) { //对于所有的边执行松弛操作

            for (int k = 0; k < G[v].size(); k++) {

                relax(v, G[v][k].v, G[v][k].len);

            }

        }

    }

}

int main(void)

{

    cin >> N >> M >> C1 >> C2;

    //每个城市的营救人员个数

    for (int i = 0; i < N; i++) {

        cin >> cityRescues[i];

    }

    //道路信息

    for (int i = 0; i < M; i++) {

        int c1, c2, L;

        cin >> c1 >> c2 >> L;

        G[c1].push\_back(ArcNode(c2, L));

        G[c2].push\_back(ArcNode(c1, L));

    }

    //求解最短路径

    bellmanFord();

    cout << routeNum[C2] << ' ' << totalRescues[C2] << endl;

}

### SPFA

#include <iostream>

#include <queue>

#include <set>

#include <vector>

using namespace std;

const int MAX\_NODE = 510;

struct ArcNode {

    int v; //邻接点

    int len; //邻接边长度

    ArcNode(int v1, int len1)

        : v(v1)

        , len(len1){};

};

int N, M, C1, C2;

vector<vector<ArcNode> > G(MAX\_NODE);

vector<int> cityRescues(MAX\_NODE); //城市i的营救人员个数

vector<int> routeNum(MAX\_NODE, 0); //从源点到i的最短路径的条数

vector<int> len(MAX\_NODE, 0x3f3f3f3f); //源点到i的最短距离

vector<int> totalRescues(MAX\_NODE, 0); //从源点到i的营救人员的个数

set<int> preNode[MAX\_NODE];

vector<int> visited(MAX\_NODE, 0); //标识节点是否在队列中

void relax(int v, int u, int arcLen, queue<int>& q)

{

    int newLen = len[v] + arcLen; //源点经由a到达b的距离

    int newRescues = totalRescues[v] + cityRescues[u]; //源点经由a到达b的营救人员的个数

    if (len[u] > len[v] + arcLen) {

        //更新距离

        len[u] = len[v] + arcLen;

        //更新路径数

        preNode[u].clear();

        preNode[u].insert(v);

        routeNum[u] = routeNum[v];

        //更新营救人员个数

        totalRescues[u] = cityRescues[u] + totalRescues[v];

        //入队

        if (!visited[u]) {

            q.push(u);

            visited[u] = 1;

        }

    } else if (len[u] == arcLen + len[v]) {

        //更新路径数

        preNode[u].insert(v);

        routeNum[u] = 0;

        for (set<int>::iterator it = preNode[u].begin(); it != preNode[u].end(); it++) {

            routeNum[u] += routeNum[\*it];

        }

        //更新营救人员的个数

        if (totalRescues[u] < totalRescues[v] + cityRescues[u]) {

            totalRescues[u] = totalRescues[v] + cityRescues[u];

        }

        //入队

        if (!visited[u]) {

            q.push(u);

            visited[u] = true;

        }

    }

}

void SPFA()

{

    //初始化

    totalRescues[C1] = cityRescues[C1];

    len[C1] = 0;

    routeNum[C1] = 1;

    queue<int> q;

    q.push(C1);

    visited[C1] = true;

    while (!q.empty()) {

        int v = q.front();

        q.pop();

        visited[v] = false;

        for (int i = 0; i < G[v].size(); i++) {

           relax(v, G[v][i].v, G[v][i].len, q);

        }

    }

}

int main(void)

{

    cin >> N >> M >> C1 >> C2;

    //每个城市的营救人员个数

    for (int i = 0; i < N; i++) {

        cin >> cityRescues[i];

    }

    //道路信息

    for (int i = 0; i < M; i++) {

        int c1, c2, L;

        cin >> c1 >> c2 >> L;

        G[c1].push\_back(ArcNode(c2, L));

        G[c2].push\_back(ArcNode(c1, L));

    }

    //求解最短路径

    SPFA();

    cout << routeNum[C2] << ' ' << totalRescues[C2] << endl;

}

## 3.24 A1030 Travel Plan

### Dijkstra

#include <iostream>

#include <queue>

#include <vector>

#include <stack>

using namespace std;

#define MAX\_NODE 510

struct ArcNode {

    int vex;

    int len;

    int cost;

    ArcNode(int vex1, int len1, int cost1)

        : vex(vex1)

        , len(len1)

        , cost(cost1){};

};

struct QNode {

    int vex;

    int len;

    QNode(int vex1, int len1)

        : vex(vex1)

        , len(len1){};

};

bool operator<(const QNode& n1, const QNode& n2)

{

    return n1.len > n2.len;

}

int N, M, S, D;

vector<vector<ArcNode> > G(MAX\_NODE);

vector<int> visited(MAX\_NODE, 0);

vector<int> len(MAX\_NODE, 0x3f3f3f3f);

vector<int> cost(MAX\_NODE, 0x3f3f3f3f);

vector<int> preNode(MAX\_NODE, -1);

int totalCost;

void dijkstra()

{

    priority\_queue<QNode> q;

    len[S] = 0;

    cost[S] = 0;

    q.push(QNode(S, 0));

    while (!q.empty()) {

        QNode top = q.top();

        q.pop();

        if (!visited[top.vex]) {

            visited[top.vex] = true;

            for (int i = 0; i < G[top.vex].size(); i++) {

                int adjVex = G[top.vex][i].vex;

                int adjLen = G[top.vex][i].len;

                int adjCost = G[top.vex][i].cost;

                if (!visited[adjVex]) {

                    int newLen = len[top.vex] + adjLen;

                    int newCost = cost[top.vex] + adjCost;

                    if (newLen < len[adjVex]) {

                        len[adjVex] = newLen;

                        cost[adjVex] = newCost;

                        preNode[adjVex] = top.vex;

                        q.push(QNode(adjVex, len[adjVex]));

                    } else if (newLen == len[adjVex]) {

                        if (newCost < cost[adjVex]) {

                            cost[adjVex] = newCost;

                            preNode[adjVex] = top.vex;

                        }

                    }

                }

            }

        }

    }

}

int main(void)

{

    //读取输入

    cin >> N >> M >> S >> D;

    for (int i = 0; i < M; i++) {

        int c1, c2, dis, cost;

        cin >> c1 >> c2 >> dis >> cost;

        G[c1].push\_back(ArcNode(c2, dis, cost));

        G[c2].push\_back(ArcNode(c1, dis, cost));

    }

    //求解最短路径

    dijkstra();

    //输出结果

    stack<int> route;

    int i = D;

    while(i != -1){

        route.push(i);

        i = preNode[i];

    }

    while(!route.empty()){

        cout<<route.top()<<' ';

        route.pop();

    }

    cout << len[D] << ' ' << cost[D] << endl;

}

### BellmanFord

#include <iostream>

#include <queue>

#include <vector>

#include <stack>

using namespace std;

#define MAX\_NODE 510

struct ArcNode {

    int vex;

    int len;

    int cost;

    ArcNode(int vex1, int len1, int cost1)

        : vex(vex1)

        , len(len1)

        , cost(cost1){};

};

int N, M, S, D;

vector<vector<ArcNode> > G(MAX\_NODE);

vector<int> len(MAX\_NODE, 0x3f3f3f3f);

vector<int> cost(MAX\_NODE, 0x3f3f3f3f);

vector<int> preNode(MAX\_NODE, -1);

int totalCost;

void relax(int a, int b, int arcLen, int arcCost){

    int newLen = len[a] + arcLen;

    int newCost = cost[a] + arcCost;

    if(newLen < len[b]){

        //更新长度

        len[b] = newLen;

        //更新花费

        cost[b] = newCost;

        //更新路线

        preNode[b] = a;

    } else if(newLen == len[b]){

        if(newCost < cost[b]){

            //更新花费

            cost[b] = newCost;

            //更新路线

            preNode[b] = a;

        }

    }

}

void bellmanFord(){

    //初始化

    len[S] = 0;

    cost[S] = 0;

    //N-1次松弛操作

    for(int i=0;i<N-1;i++){

        for(int v = 0;v<N;v++){

            for(int k=0;k<G[v].size();k++){

                relax(v, G[v][k].vex, G[v][k].len, G[v][k].cost);

            }

        }

    }

}

int main(void)

{

    //读取输入

    cin >> N >> M >> S >> D;

    for (int i = 0; i < M; i++) {

        int c1, c2, dis, cost;

        cin >> c1 >> c2 >> dis >> cost;

        G[c1].push\_back(ArcNode(c2, dis, cost));

        G[c2].push\_back(ArcNode(c1, dis, cost));

    }

    //求解最短路径

    bellmanFord();

    //输出结果

    stack<int> route;

    int i = D;

    while(i != -1){

        route.push(i);

        i = preNode[i];

    }

    while(!route.empty()){

        cout<<route.top()<<' ';

        route.pop();

    }

    cout << len[D] << ' ' << cost[D] << endl;

}

### SPFA

#include <iostream>

#include <queue>

#include <stack>

#include <vector>

using namespace std;

#define MAX\_NODE 510

struct ArcNode {

    int vex;

    int len;

    int cost;

    ArcNode(int vex1, int len1, int cost1)

        : vex(vex1)

        , len(len1)

        , cost(cost1){};

};

int N, M, S, D;

vector<vector<ArcNode> > G(MAX\_NODE);

vector<int> len(MAX\_NODE, 0x3f3f3f3f);

vector<int> cost(MAX\_NODE, 0x3f3f3f3f);

vector<int> preNode(MAX\_NODE, -1);

vector<int> visited(MAX\_NODE);

int totalCost;

void relax(int a, int b, int arcLen, int arcCost, queue<int>& q)

{

    int newLen = len[a] + arcLen;

    int newCost = cost[a] + arcCost;

    if (newLen < len[b]) {

        //更新长度

        len[b] = newLen;

        //更新花费

        cost[b] = newCost;

        //更新路线

        preNode[b] = a;

        //入队

        if (!visited[b]) {

            q.push(b);

            visited[b] = true;

        }

    } else if (newLen == len[b]) {

        if (newCost < cost[b]) {

            //更新花费

            cost[b] = newCost;

            //更新路线

            preNode[b] = a;

            //入队

            if (!visited[b]) {

                q.push(b);

                visited[b] = true;

            }

        }

    }

}

void SPFA()

{

    //初始化

    len[S] = 0;

    cost[S] = 0;

    //求解

    queue<int> q;

    q.push(S);

    visited[S] = true;

    while (!q.empty()) {

        int v = q.front();

        q.pop();

        visited[v] = false;

        for (int i = 0; i < G[v].size(); i++) {

            relax(v, G[v][i].vex, G[v][i].len, G[v][i].cost, q);

        }

    }

}

int main(void)

{

    //读取输入

    cin >> N >> M >> S >> D;

    for (int i = 0; i < M; i++) {

        int c1, c2, dis, cost;

        cin >> c1 >> c2 >> dis >> cost;

        G[c1].push\_back(ArcNode(c2, dis, cost));

        G[c2].push\_back(ArcNode(c1, dis, cost));

    }

    //求解最短路径

    SPFA();

    //输出结果

    stack<int> route;

    int i = D;

    while (i != -1) {

        route.push(i);

        i = preNode[i];

    }

    while (!route.empty()) {

        cout << route.top() << ' ';

        route.pop();

    }

    cout << len[D] << ' ' << cost[D] << endl;

}