图论-单源最短路的扩展应用

一、AcWing 1137. 选择最佳线路（虚拟源点）

【题目描述】

有一天，琪琪想乘坐公交车去拜访她的一位朋友。

由于琪琪非常容易晕车，所以她想尽快到达朋友家。

现在给定你一张城市交通路线图，上面包含城市的公交站台以及公交线路的具体分布。

已知城市中共包含n个车站（编号$1 \sim n$）以及m条公交线路。

每条公交线路都是**单向的**，从一个车站出发直接到达另一个车站，两个车站之间可能存在多条公交线路。

琪琪的朋友住在s号车站附近。

琪琪可以在任何车站选择换乘其它公共汽车。

请找出琪琪到达她的朋友家（附近的公交车站）需要花费的最少时间。

【输入格式】

输入包含多组测试数据。

每组测试数据第一行包含三个整数n, m, s，分别表示车站数量，公交线路数量以及朋友家附近车站的编号。

接下来m行，每行包含三个整数p, q, t，表示存在一条线路从车站p到达车站q，用时为t。

接下来一行，包含一个整数w，表示琪琪家附近共有w个车站，她可以在这w个车站中选择一个车站作为始发站。

再一行，包含w个整数，表示琪琪家附近的w个车站的编号。

【输出格式】

每个测试数据输出一个整数作为结果，表示所需花费的最少时间。

如果无法达到朋友家的车站，则输出-1。

每个结果占一行。

【数据范围】

$n \leq 1000, m \leq 20000$

$1 \leq s \leq n$

$0 < w < n$

$0 < t \leq 1000$

【输入样例】

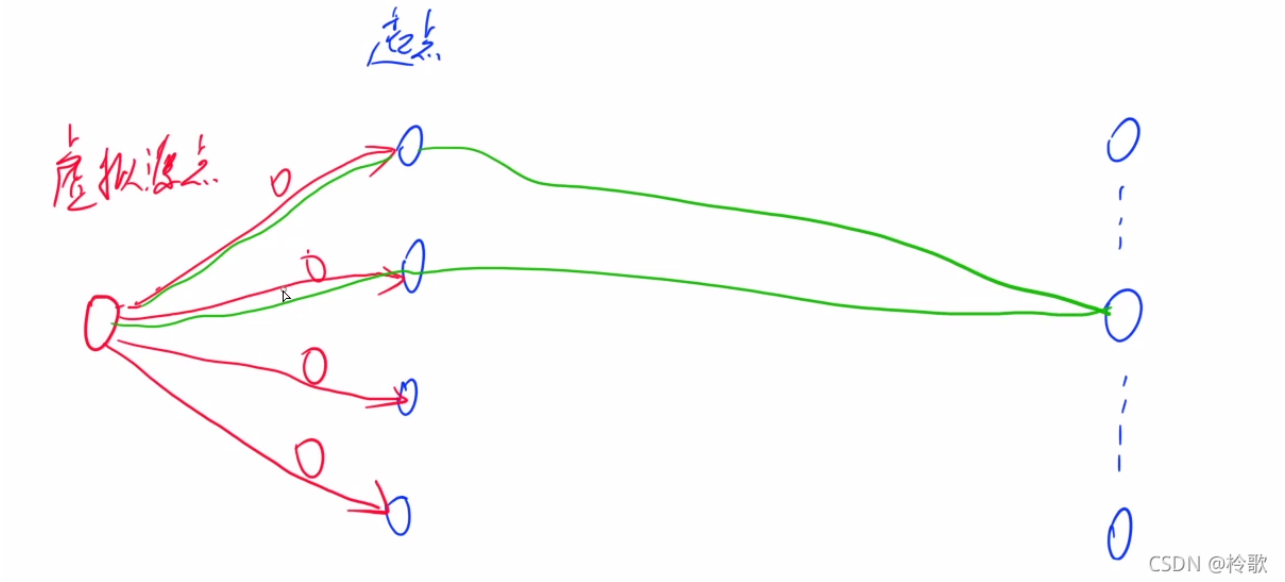
```
5 8 5
1 2 2
1 5 3
1 3 4
2 4 7
2 5 6
2 3 5
3 5 1
4 5 1
2
2 3
4 3 4
1 2 3
1 3 4
2 3 2
1
1
```

【输出样例】

```
1
-1
```

【分析】

本题的题意为从给定的多个起点中**任选一个**作为起点求出到某个终点的最短路，那么其中一个思路是建立反图，以终点为起点跑一遍最短路，求出终点到每个起点的最短路，取最小值即为答案，但是这种方法不适用于**终点也有多个**的情况，因此我们采用建立**虚拟源点**的方式求解本题。



如上图所示，我们建立一个虚拟源点，虚拟源点到每个原起点的距离为**0**，那么就从原问题：从每个起点出发，到达终点的所有路线的最短距离，转变成新问题：从虚拟源点出发，到达终点的所有路线的最短距离。也就是只需要以虚拟源点为起点，跑一遍最短路即可。

【代码】

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>
#include <queue>
using namespace std;

typedef pair<int, int> PII;
const int N = 1010, M = 21010;
int e[M], ne[M], d[M], h[N], idx;
int dis[N];
bool st[N];
int n, m, s, w;

void add(int u, int v, int w)
{
    e[idx] = v, ne[idx] = h[u], d[idx] = w, h[u] = idx++;
}

int dijkstra()
{
    memset(st, false, sizeof st);

    memset(dis, 0x3f, sizeof dis);
```

```
    priority_queue<PII, vector<PII>, greater<PII> > Q;
    dis[0] = 0;
    Q.push({ 0, 0 });
    while (Q.size())
    {
        auto t = Q.top().second;
        Q.pop();
        if (st[t]) continue;
        st[t] = true;
        for (int i = h[t]; ~i; i = ne[i])
            if (dis[t] + d[i] < dis[e[i]])
                dis[e[i]] = dis[t] + d[i], Q.push({ dis[e[i]], e[i] });
    }
    return dis[s] == 0x3f3f3f3f ? -1 : dis[s];
}

int main()
{
    ios::sync_with_stdio(false);
    while (cin >> n >> m >> s)
    {
        memset(h, -1, sizeof h), idx = 0;//多组样例记得初始化
        int a, b, c;
        while (m--) cin >> a >> b >> c, add(a, b, c);
        cin >> w;
        while (w--) cin >> a, add(0, a, 0);//从虚拟源点向每个起点连一条权值为0的边
        cout << dijkstra() << endl;
    }
    return 0;
}
```

二、AcWing 1131. 拯救大兵瑞恩（分层图，拆点）

【题目描述】

1944年，特种兵麦克接到国防部的命令，要求立即赶赴太平洋上的一个孤岛，营救被敌军俘虏的大兵瑞恩。

瑞恩被关押在一个迷宫里，迷宫地形复杂，但幸好麦克得到了迷宫的地形图。

迷宫的外形是一个长方形，其南北方向被划分为N行，东西方向被划分为M列，于是整个迷宫被划分为$N \times M$个单元。

每一个单元的位置可用一个有序数对（单元的行号，单元的列号）来表示。

南北或东西方向相邻的2个单元之间可能互通，也可能有一扇锁着的门，或者是一堵不可逾越的墙。

注意：门可以从两个方向穿过，即可以看成一条无向边。

迷宫中有一些单元存放着钥匙，同一个单元可能存放**多把钥匙**，并且所有的门被分成P类，打开同一类的门的钥匙相同，不同类门的钥匙不同。

大兵瑞恩被关押在迷宫的东南角，即(N,M)单元里，并已经昏迷。

迷宫只有一个入口，在西北角。

也就是说，麦克可以直接进入$(1,1)$单元。

另外，麦克从一个单元移动到另一个相邻单元的时间为1，拿取所在单元的钥匙的时间以及用钥匙开门的时间可忽略不计。

试设计一个算法，帮助麦克以最快的方式到达瑞恩所在单元，营救大兵瑞恩。

【输入格式】

第一行有三个整数，分别表示N,M,P的值。

第二行是一个整数k，表示迷宫中门和墙的总数。

接下来k行，每行包含五个整数，$X_{i1},Y_{i1},X_{i2},Y_{i2},G_i$：当$G_i \geq 1$时，表示$(X_{i1},Y_{i1})$单元与$(X_{i2},Y_{i2})$单元之间有一扇第$G_i$类的门，当$G_i = 0$时，表示 (X_{i1},Y_{i1})单元与(X_{i2},Y_{i2})单元之间有一面不可逾越的墙。

接下来一行，包含一个整数S，表示迷宫中存放的钥匙的总数。

接下来S行，每行包含三个整数X_{i1},Y_{i1},Q_i，表示(X_{i1},Y_{i1})单元里存在一个能开启第Q_i类门的钥匙。

【输出格式】

输出麦克营救到大兵瑞恩的最短时间。

如果问题无解，则输出-1。

【数据范围】

$|X_{i1} - X_{i2}| + |Y_{i1} - Y_{i2}| = 1$

$0 \leq G_i \leq P$

$1 \leq Q_i \leq P$

$1 \leq N, M, P \leq 10$

$1 \leq k \leq 150$

【输入样例】

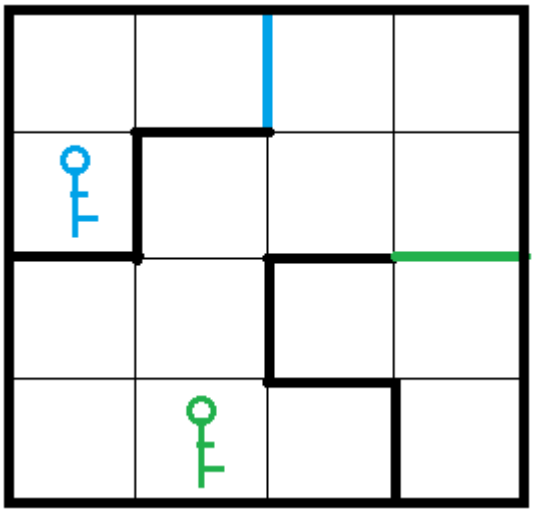
```
4 4 9
9
1 2 1 3 2
1 2 2 2 0
2 1 2 2 0
2 1 3 1 0
2 3 3 3 0
2 4 3 4 1
3 2 3 3 0
3 3 4 3 0
4 3 4 4 0
2
2 1 2
4 2 1
```

【输出样例】

```
14
```

【样例解释】

迷宫如下所示：



【分析】

状态表示：

试想下如果本题没有钥匙和门的条件，只要求从左上角走到右下角的最小步数，就是简单的迷宫问题了，可以使用BFS解决。加上钥匙和门的的条件，便是类似于八数码问题了。实际上BFS解决的最短路问题都可以看作求从初始状态到结束状态需要的最小转移次数，普通迷宫问题的状态就是当前所在的坐标，八数码问题的状态就是当前棋盘的局面。本题在迷宫问题上加上了钥匙和门的条件，显然，**处在同一个坐标下，持有钥匙和不持有钥匙就不是同一个状态了**，为了能够清楚的表示每个状态，除了当前坐标外还需要加上当前获得的钥匙信息，即`f[x][y][st]`表示当前处在(x,y)位置下持有钥匙状态为st，将二维坐标压缩成一维就得到`f[id][st]`这样的状态表示了，或者说，id是格子的编号，从上到下，从左而右的编号依次为$0\sim n\times m-1$。由于钥匙和门的种类最多只有十种，因此可以**用st的二进制表示中的第i位来表示是否拥有i号钥匙**，若第i位为1说明有第i号的钥匙，例如当st为`0011`时，表示持有第$1,2$类钥匙，这里注意我在表示状态时抛弃了最右边的一位，因为钥匙编号是$1\sim 11$，为了减少状态数，将其看成$0\sim 10$，我想确定是否持有第i类钥匙时，只需要判断`st >> i & 1`是不是等于1即可。

状态转移：

如果两个相邻格子间有墙，就不能转移；有门，持有该类门钥匙就能转移，没有钥匙就不能转移；没有障碍，正常转移。

首先不管有没有钥匙，先转移到这个格子再说，需要将步数加一，若转移到有钥匙的格子，则拿起钥匙不移动位置进入另一种状态，步数不变。这种转移方式步数有加一与不变两种情况，因此可以使用双端队列BFS、Dijkstra算法求解。

【双端队列BFS代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#include <deque>
#include <set>
#define X first
#define Y second
using namespace std;

typedef pair<int, int> PII;
const int N = 110, M = 400, P = 1 << 10;//P表示钥匙的最大状态数(用二进制的每一位表示)
int e[M], ne[M], d[M], h[N], idx;
int dis[N][P];
bool st[N][P];
int key[N];//每个点的钥匙状态
```

```cpp
int n, m, p, k, s;
set<PII> edges;//记录路径上是门或墙的边
int dx[4] = { -1, 0, 1, 0 }, dy[4] = { 0, 1, 0, -1 };

void add(int u, int v, int w)
{
    e[idx] = v, d[idx] = w, ne[idx] = h[u], h[u] = idx++;
}

//枚举每个点，然后枚举四个移动方向，如果两点间既没有门也没有墙则连一条权值为0的边
void build()
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            for (int k = 0; k < 4; k++)
            {
                int nx = i + dx[k], ny = j + dy[k];
                if (nx >= 0 && nx < n && ny >= 0 && ny < m)
                {
                    int a = i * m + j, b = nx * m + ny;
                    if (!edges.count({ a, b })) add(a, b, 0);
                }
            }
}

int bfs()
{
    memset(dis, 0x3f, sizeof dis);
    deque<PII> Q;
    Q.push_back({ 0, 0 });//起点为0，钥匙状态也为0
    dis[0][0] = 0;
    while (Q.size())
    {
        auto t = Q.front();
        Q.pop_front();
        if (st[t.X][t.Y]) continue;
        st[t.X][t.Y] = true;
        if (t.X == n * m - 1) return dis[t.X][t.Y];//终点为n * m - 1
        if (key[t.X])//如果当前点有钥匙
        {
            int state = t.Y | key[t.X];//拿起钥匙后的状态

            if (dis[t.X][t.Y] < dis[t.X][state])
```

```
                dis[t.X][state] = dis[t.X][t.Y], Q.push_front({ t.X, state });
        }
        for (int i = h[t.X]; ~i; i = ne[i])
        {
            if (d[i] && !(t.Y >> d[i] - 1 & 1)) continue;//如果当前路径上有门而当前的状态中没有所需的钥匙则无法转移
            if (dis[t.X][t.Y] < dis[e[i]][t.Y])
                dis[e[i]][t.Y] = dis[t.X][t.Y] + 1, Q.push_back({ e[i], t.Y });
        }
    }
    return -1;
}

int main()
{
    memset(h, -1, sizeof h);
    cin >> n >> m >> p >> k;
    while (k--)
    {
        int x, y, u, v, c;
        cin >> x >> y >> u >> v >> c;
        int a = (x - 1) * m + y - 1, b = (u - 1) * m + v - 1;
        if (c) add(a, b, c), add(b, a, c);
        edges.insert({ a, b }), edges.insert({ b, a });
    }
    build();//建立剩余的既不是门也不是墙的边
    cin >> s;
    while (s--)
    {
        int x, y, id;
        cin >> x >> y >> id;
        key[(x - 1) * m + y - 1] |= 1 << id - 1;//题中钥匙编号从1开始，将其减一即可映射成0~10
    }
    cout << bfs() << endl;
    return 0;
}
```

【Dijkstra代码】

```
#include <iostream>
```

```cpp
#include <cstring>
#include <algorithm>
#include <queue>
#include <set>
#define X first
#define Y second
using namespace std;

typedef pair<int, int> PII;
typedef pair<PII, int> PIII;
const int N = 110, M = 400, P = 1 << 10;//P表示钥匙的最大状态数(用二进制的每一位表示)
int e[M], ne[M], d[M], h[N], idx;
int dis[N][P];
bool st[N][P];
int key[N];//每个点的钥匙状态
int n, m, p, k, s;
set<PII> edges;//记录路径上是门或墙的边
int dx[4] = { -1, 0, 1, 0 }, dy[4] = { 0, 1, 0, -1 };

void add(int u, int v, int w)
{
    e[idx] = v, d[idx] = w, ne[idx] = h[u], h[u] = idx++;
}

//枚举每个点，然后枚举四个移动方向，如果两点间既没有门也没有墙则连一条权值为0的边
void build()
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            for (int k = 0; k < 4; k++)
            {
                int nx = i + dx[k], ny = j + dy[k];
                if (nx >= 0 && nx < n && ny >= 0 && ny < m)
                {
                    int a = i * m + j, b = nx * m + ny;
                    if (!edges.count({ a, b })) add(a, b, 0);
                }
            }
}


int dijkstra()
```

```
{
    memset(dis, 0x3f, sizeof dis);
    priority_queue<PIII, vector<PIII>, greater<PIII> > Q;
    Q.push({ { 0, 0 }, 0 });//起点为0，钥匙状态也为0
    dis[0][0] = 0;
    while (Q.size())
    {
        auto t = Q.top();
        Q.pop();
        if (st[t.X.Y][t.Y]) continue;
        st[t.X.Y][t.Y] = true;
        if (t.X.Y == n * m - 1) return dis[t.X.Y][t.Y];//终点为n * m - 1
        if (key[t.X.Y])//如果当前点有钥匙
        {
            int state = t.Y | key[t.X.Y];//拿起钥匙后的状态
            if (dis[t.X.Y][t.Y] < dis[t.X.Y][state])
                dis[t.X.Y][state] = dis[t.X.Y][t.Y], Q.push({ { dis[t.X.Y][state], t.X.Y }, state });
        }
        for (int i = h[t.X.Y]; ~i; i = ne[i])
        {
            if (d[i] && !(t.Y >> d[i] - 1 & 1)) continue;//如果当前路径上有门而当前的状态中没有所需的钥匙则无法转移
            if (dis[t.X.Y][t.Y] < dis[e[i]][t.Y])
                dis[e[i]][t.Y] = dis[t.X.Y][t.Y] + 1, Q.push({ { dis[e[i]][t.Y], e[i] }, t.Y });
        }
    }
    return -1;
}

int main()
{
    memset(h, -1, sizeof h);
    cin >> n >> m >> p >> k;
    while (k--)
    {
        int x, y, u, v, c;
        cin >> x >> y >> u >> v >> c;
        int a = (x - 1) * m + y - 1, b = (u - 1) * m + v - 1;
        if (c) add(a, b, c), add(b, a, c);
        edges.insert({ a, b }), edges.insert({ b, a });

    }
```

```
    build();//建立剩余的既不是门也不是墙的边
    cin >> s;
    while (s--)
    {
        int x, y, id;
        cin >> x >> y >> id;
        key[(x - 1) * m + y - 1] |= 1 << id - 1;//题中钥匙编号从1开始，将其减一即可映射成0~10
    }
    cout << dijkstra() << endl;
    return 0;
}
```

三、AcWing 1134. 最短路计数（方案数）

【题目描述】

给出一个N个顶点M条边的无向无权图，顶点编号为1到N。

问从顶点1开始，到其他每个点的最短路有几条。

【输入格式】

第一行包含2个正整数N,M，为图的顶点数与边数。

接下来M行，每行两个正整数x,y，表示有一条顶点x连向顶点y的边，请注意可能有自环与重边。

【输出格式】

输出N行，每行一个非负整数，第i行输出从顶点1到顶点i有多少条不同的最短路，由于答案有可能会很大，你只需要输出对100003取模后的结果即可。

如果无法到达顶点i则输出0。

【数据范围】

$1 \leq N \leq 10^5$

$1 \leq M \leq 2 \times 10^5$

【输入样例】

```
5 7
1 2
1 3
2 4
3 4
2 3
4 5
4 5
```

【输出样例】

```
1
1
1
2
4
```

【分析】

要求最短路计数首先满足条件是不能存在值为0的环，因为存在的话那么被更新的点的条数就为INF了。

要把图抽象成一种最短路树（拓扑图）。

求最短的算法有以下几种：

1. BFS：每个点只入队一次，出队一次。可以抽象成拓扑图，因为它可以保证被更新点的父节点一定已经是最短距离了，并且这个点的条数已经被完全更新过了。这个性质是核心性质。
2. Dijkstra：每个点只出队一次。也可以抽象成拓扑图，同理由于每一个出队的点一定已经是最短距离，并且它出队的时候是队列中距离最小的点，这就代表他的最短距离条数已经被完全更新了，所以构成拓扑性质。
3. Bellman_ford：SPFA本身不具备拓扑序，因为更新它的点不一定是最短距离，所以会出错。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
using namespace std;
```

```cpp

const int N = 100010, M = 400010, MOD = 100003;
int e[M], ne[M], h[N], idx;
int dis[N], cnt[N];
int n, m;

void add(int u, int v)
{
    e[idx] = v, ne[idx] = h[u], h[u] = idx++;
}

void bfs()
{
    memset(dis, 0x3f, sizeof dis);
    dis[1] = 0, cnt[1] = 1;
    queue<int> Q;
    Q.push(1);
    while (Q.size())
    {
        auto t = Q.front();
        Q.pop();
        for (int i = h[t]; ~i; i = ne[i])
            if (dis[t] + 1 < dis[e[i]])
                dis[e[i]] = dis[t] + 1, cnt[e[i]] = cnt[t], Q.push(e[i]);
            else if (dis[t] + 1 == dis[e[i]]) cnt[e[i]] = (cnt[e[i]] +
cnt[t]) % MOD;
    }
}

int main()
{
    ios::sync_with_stdio(false);
    cin >> n >> m;
    memset(h, -1, sizeof h);
    while (m--)
    {
        int a, b;
        cin >> a >> b;
        add(a, b), add(b, a);
    }
    bfs();
    for (int i = 1; i <= n; i++) cout << cnt[i] << endl;

    return 0;
```

```
48  }
```

四、AcWing 383. 观光（拆点，次短路条数）

【题目描述】

“您的个人假期”旅行社组织了一次比荷卢经济联盟的巴士之旅。

比荷卢经济联盟有很多公交线路。

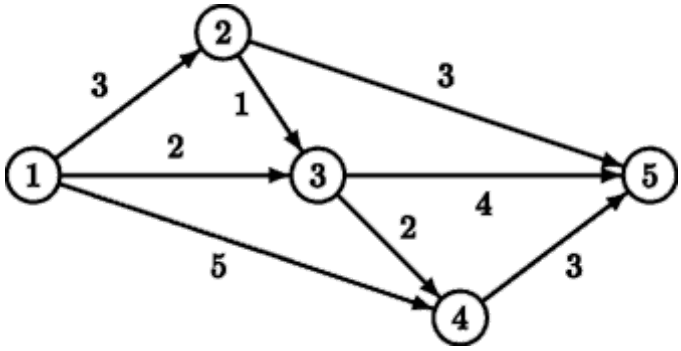
每天公共汽车都会从一座城市开往另一座城市。

沿途汽车可能会在一些城市（零或更多）停靠。

旅行社计划旅途从S城市出发，到F城市结束。

由于不同旅客的景点偏好不同，所以为了迎合更多旅客，旅行社将为客户提供多种不同线路。

游客可以选择的行进路线有所限制，要么满足所选路线总路程为S到F的**最小路程**，要么满足所选路线总路程**仅比最小路程多一个单位长度**。



如上图所示，如果$S=1,F=5$，则这里有两条最短路线$1\to2\to5,1\to3\to5$，长度为6；有一条比最短路程多一个单位长度的路线$1\to3\to4\to5$，长度为7。

现在给定比荷卢经济联盟的公交路线图以及两个城市S和F，请你求出旅行社最多可以为旅客提供多少种不同的满足限制条件的线路。

【输入格式】

第一行包含整数T，表示共有T组测试数据。

每组数据第一行包含两个整数N和M，分别表示总城市数量和道路数量。

接下来M行，每行包含三个整数A,B,L，表示有一条线路从城市A通往城市B，长度为L。

需注意，线路是**单向的**，存在从A到B的线路不代表一定存在从B到A的线路，另外从城市A到城市B可能存在多个不同的线路。

接下来一行，包含两个整数S和F，数据保证S和F不同，并且S,F之间至少存在一条线路。

【输出格式】

每组数据输出一个结果，每个结果占一行。

数据保证结果不超过10^9。

【数据范围】

$2 \leq N \leq 1000$

$1 \leq M \leq 10000$

$1 \leq L \leq 1000$

$1 \leq A, B, S, F \leq N$

【输入样例】

```
2
5 8
1 2 3
1 3 2
1 4 5
2 3 1
2 5 3
3 4 2
3 5 4
4 5 3
1 5
5 6
2 3 1
3 2 1
3 1 10
4 5 2
5 2 7
5 2 7
4 1
```

【输出样例】

```
3
2
```

【分析】

设状态$dist[i][0,1]$分别表示初始城市S到城市i的最短距离和次短距离，$cnt[i][0,1]$表示城市S到城市i的最短路径和次短路经的条数，$d[i]$表示每条路的长度。

初始时，$dist[S][0]$为0，$cnt[S][0]$为1。（其余都初始化成正无穷，初始时S不存在次短路）

Dijkstra算法枚举城市ver可通往的城市j时，有四种情况：

1. $dist[j][0] > dist[ver][type] + d[i]$：当前最短路变成次短路，更新最短路，将最短路和次短路加入优先队列；
2. $dist[j][0] = dist[ver][type] + d[i]$：找到一条新的最短路，更新最短路条数；
3. $dist[j][1] > dist[ver][type] + d[i]$：找到一条更短的次短路，覆盖掉当前次短路，加入优先队列；
4. $dist[j][1] = dist[ver][type] + d[i]$：找到一条新的次短路，更新次短路条数。

如果到F城市的次短路径长度比最短路径长度恰好多1，则结果为次短路径与最短路径条数之和，否则结果为最短路径条数。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
#define X first
#define Y second
using namespace std;

typedef pair<int, int> PII;
typedef pair<PII, int> PIII;
const int N = 1010, M = 10010;
int e[M], ne[M], d[M], h[N], idx;
int dis[N][2], cnt[N][2];//第二维为0表示最短路，为1表示次短路
bool st[N][2];
int n, m, s, f;

void add(int u, int v, int w)
{
    e[idx] = v, d[idx] = w, ne[idx] = h[u], h[u] = idx++;
}

int dijkstra()
{

    memset(dis, 0x3f, sizeof dis);
```

```
    memset(st, false, sizeof st);
    memset(cnt, 0, sizeof cnt);
    dis[s][0] = 0, cnt[s][0] = 1;//起点没有次短路
    priority_queue<PIII, vector<PIII>, greater<PIII> > Q;
    Q.push({ { 0, s }, 0 });//第一维表示距离，第二维表示点的序号，第三维表示路线类型
    while (Q.size())
    {
        auto t = Q.top();
        Q.pop();
        int ver = t.X.Y, w = t.X.X, type = t.Y;
        if (st[ver][type]) continue;
        st[ver][type] = true;
        for (int i = h[ver]; ~i; i = ne[i])
            if (dis[e[i]][0] > w + d[i])//最短路需要被更新的情况
            {
                dis[e[i]][1] = dis[e[i]][0], cnt[e[i]][1] = cnt[e[i]][0];//最短路沦为次短路
                Q.push({ { dis[e[i]][1], e[i] }, 1 });
                dis[e[i]][0] = w + d[i], cnt[e[i]][0] = cnt[ver][type];//更新最短路
                Q.push({ { dis[e[i]][0], e[i] }, 0 });
            }
            else if (dis[e[i]][0] == w + d[i]) cnt[e[i]][0] += cnt[ver][type];//最短路条数增加的情况
            else if (dis[e[i]][1] > w + d[i])//次短路需要被更新的情况
            {
                dis[e[i]][1] = w + d[i], cnt[e[i]][1] = cnt[ver][type];
                Q.push({ { dis[e[i]][1], e[i] }, 1 });
            }
            else if (dis[e[i]][1] == w + d[i]) cnt[e[i]][1] += cnt[ver][type];//次短路条数增加的情况
    }
    if (dis[f][0] == dis[f][1] - 1) return cnt[f][0] + cnt[f][1];//判断次短路的距离是不是比次短路多1
    return cnt[f][0];
}

int main()
{
    ios::sync_with_stdio(false);
    int T;

    cin >> T;
```

```
    while (T--)
    {
        cin >> n >> m;
        memset(h, -1, sizeof h);
        idx = 0;
        while (m--)
        {
            int a, b, c;
            cin >> a >> b >> c;
            add(a, b, c);
        }
        cin >> s >> f;
        cout << dijkstra() << endl;
    }
    return 0;
}
```