图论-单源最短路的建图方式

一、AcWing 1129. 热浪

【题目描述】

John已经研究过可以把牛奶从威斯康星运送到德克萨斯州的路线。

这些路线包括起始点和终点一共有T个城镇，为了方便标号为1到T。

除了起点和终点外的每个城镇都由双向道路连向至少两个其它的城镇。

每条道路有一个通过费用。

给定一个地图，包含C条直接连接2个城镇的道路。

每条道路由道路的起点R_s，终点R_e和花费C_i组成。

求从起始的城镇T_s到终点的城镇T_e最小的总费用。

【输入格式】

第一行：4个由空格隔开的整数：T, C, T_s, T_e；

第2到第$C+1$行：第$i+1$行描述第i条道路，包含3个由空格隔开的整数：R_s, R_e, C_i。

【输出格式】

一个单独的整数表示从T_s到T_e的最小总费用。

数据保证至少存在一条道路。

【数据范围】

$1 \leq T \leq 2500$

$1 \leq C \leq 6200$

$1 \leq T_s, T_e, R_s, R_e \leq T$

$1 \leq C_i \leq 1000$

【输入样例】

```
7 11 5 4
2 4 2
1 4 3
7 2 2
3 4 3
5 7 5
7 3 3
6 1 1
6 3 4
2 4 3
5 6 3
7 2 1
```

【输出样例】

```
7
```

【分析】

最裸的单源最短路问题，作为复习模板和热身，以下给出两种写法。

【SPFA代码】

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

const int N = 2510, M = 12410;
int e[M], ne[M], d[M], h[N], idx;
int dis[N];
bool st[N];
int n, m, s, t;

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


int spfa()
```

```
{
    memset(dis, 0x3f, sizeof dis);
    queue<int> Q;
    Q.push(s);
    dis[s] = 0;
    st[s] = true;
    while (Q.size())
    {
        int t = Q.front();
        Q.pop();
        st[t] = false;
        for (int i = h[t]; ~i; i = ne[i])
            if (dis[t] + d[i] < dis[e[i]])
            {
                dis[e[i]] = dis[t] + d[i];
                if (!st[e[i]]) Q.push(e[i]), st[e[i]] = true;
            }
    }
    return dis[t];
}

int main()
{
    cin >> n >> m >> s >> t;
    memset(h, -1, sizeof h);
    while (m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c), add(b, a, c);
    }
    cout << spfa() << endl;
    return 0;
}
```

【堆优化Dijkstra代码】

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

```

```cpp
typedef pair<int, int> PII;
const int N = 2510, M = 12410;
int e[M], ne[M], d[M], h[N], idx;
int dis[N];
bool st[N];
int n, m, s, t;

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);
    priority_queue<PII, vector<PII>, greater<PII> > Q;
    dis[s] = 0;
    Q.push({ 0, s });
    while (Q.size())
    {
        int 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[t];
}

int main()
{
    cin >> n >> m >> s >> t;
    memset(h, -1, sizeof h);
    while (m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c), add(b, a, c);
    }
    cout << dijkstra() << endl;

    return 0;
```

```
50 }
```

二、AcWing 1128. 信使

【题目描述】

战争时期，前线有n个哨所，每个哨所可能会与其他若干个哨所之间有通信联系。

信使负责在哨所之间传递信息，当然，这是要花费一定时间的（以天为单位）。

指挥部设在第一个哨所。

当指挥部下达一个命令后，指挥部就派出若干个信使向与指挥部相连的哨所送信。

当一个哨所接到信后，这个哨所内的信使们也以同样的方式向其他哨所送信。信在一个哨所内停留的时间可以忽略不计。

直至所有n个哨所全部接到命令后，送信才算成功。

因为准备充足，**每个哨所内都安排了足够的信使**（如果一个哨所与其他k个哨所有通信联系的话，这个哨所内至少会配备k个信使）。

现在总指挥请你编一个程序，计算出完成整个送信过程最短需要多少时间。

【输入格式】

第1行有两个整数n和m，中间用1个空格隔开，分别表示有n个哨所和m条通信线路。

第2至$m+1$行：每行三个整数i、j、k，中间用1个空格隔开，表示第i个和第j个哨所之间存在双向通信线路，且这条线路要花费k天。

【输出格式】

一个整数，表示完成整个送信过程的最短时间。

如果不是所有的哨所都能收到信，就输出-1。

【数据范围】

$1 \leq n \leq 100$

$1 \leq m \leq 200$

$1 \leq k \leq 1000$

【输入样例】

```
4 4
1 2 4
2 3 7
2 4 1
3 4 6
```

【输出样例】

```
11
```

【分析】

分析题目可知当**最晚**的一个点收到信，说明送完了整个流程。

因此我们只需要找到**最晚**收到信的点所需要的**最短时间**就行，因此可以求出每个点到指挥部（**1**号点）的最短距离，其中最大的那个距离即为最终答案。

【代码】

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

const int N = 110;
int g[N][N];
int n, m, res;

int main()
{
    cin >> n >> m;
    memset(g, 0x3f, sizeof g);
    while (m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        g[a][b] = g[b][a] = min(g[a][b], c);
    }
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
```

```
    for (int i = 2; i <= n; i++)
        res = max(res, g[1][i]);
    if (res == 0x3f3f3f3f) cout << -1 << endl;
    else cout << res << endl;
    return 0;
}
```

三、AcWing 1127. 香甜的黄油

【题目描述】

农夫John发现了做出全威斯康辛州最甜的黄油的方法：糖。

把糖放在一片牧场上，他知道N只奶牛会过来舔它，这样就能做出能卖好价钱的超甜黄油。

当然，他将付出额外的费用在奶牛上。

农夫John很狡猾，就像以前的巴甫洛夫，他知道他可以训练这些奶牛，让它们在听到铃声时去一个特定的牧场。

他打算将糖放在那里然后下午发出铃声，以至他可以在晚上挤奶。

农夫John知道每只奶牛都在各自喜欢的牧场（一个牧场不一定只有一头牛）。

给出各头牛在的牧场和牧场间的路线，找出使所有牛到达的路程和最短的牧场（他将把糖放在那）。

数据保证至少存在一个牧场和所有牛所在的牧场连通。

【输入格式】

第一行三个数：奶牛数N，牧场数P，牧场间道路数C。

第二行到第$N+1$行：1到N号奶牛所在的牧场号。

第$N+2$行到第$N+C+1$行每行有三个数：相连的牧场A、B，两牧场间距D，当然，连接是双向的。

【输出格式】

共一行，输出奶牛必须行走的最小的距离和。

【数据范围】

$1 \leq N \leq 500$

$2 \leq P \leq 800$

$1 \leq C \leq 1450$

$1 \leq D \leq 255$

【输入样例】

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

【输出样例】

```
8
```

【分析】

根据题意，我们需要找到一个距离所有牛最短的牧场。那么只需要枚举以所有牧场为起点，求此时所有其他有牛的牧场到它的最短路之和即可。

【代码】

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

typedef pair<int, int> PII;
const int N = 810, M = 2910;
const int INF = 0x3f3f3f3f;
int e[M], ne[M], d[M], h[N], idx;
int dis[N], id[N];
bool st[N];
int n, p, m;

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

```cpp
}

int dijkstra(int s)
{
    memset(dis, 0x3f, sizeof dis);
    memset(st, false, sizeof st);
    priority_queue<PII, vector<PII>, greater<PII> > Q;
    Q.push({ 0, s });
    dis[s] = 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] });
    }
    int res = 0;
    for (int i = 0; i < n; i++)
        if (dis[id[i]] == INF) return INF;
        else res += dis[id[i]];
    return res;
}

int main()
{
    cin >> n >> p >> m;
    for (int i = 0; i < n; i++) cin >> id[i];
    memset(h, -1, sizeof h);
    for (int i = 0; i < m; i++)
    {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c), add(b, a, c);
    }
    int res = INF;
    for (int i = 1; i <= p; i++) res = min(res, dijkstra(i));
    cout << res << endl;
    return 0;
}
```

四、AcWing 1126. 最小花费

【题目描述】

在n个人中，某些人的银行账号之间可以互相转账。

这些人之间转账的手续费各不相同。

给定这些人之间转账时需要从转账金额里扣除百分之几的手续费，请问A最少需要多少钱使得转账后B收到100元。

【输入格式】

第一行输入两个正整数n,m，分别表示总人数和可以互相转账的人的对数。

以下m行每行输入三个正整数x,y,z，表示标号为x的人和标号为y的人之间互相转账需要扣除$z\%$的手续费$(z<100)$。

最后一行输入两个正整数A,B。

数据保证A与B之间可以直接或间接地转账。

【输出格式】

输出A使得B到账100元最少需要的总费用。

精确到小数点后8位。

【数据范围】

$1\le n\le 2000$

$m\le 10^5$

【输入样例】

```
3 3
1 2 1
2 3 2
1 3 3
1 3
```

【输出样例】

```
103.07153164
```

【分析】

假设T转给B的手续费为$w\%$，那么当B收到100元时，T手上有$100/(1-w\%)$元。那么将B作为起点，A作为终点，若要使A的钱尽可能少，那么就要使得B走过的每条路的手续费都是最低的，即$(1-w\%)$最大，$x/(1-w\%)$最小，因此用$x/(1-w\%)$来更新每个点到B的最短距离即可。

【代码】

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

typedef pair<double, int> PDI;
const int N = 2010, M = 200010;
int e[M], ne[M], h[N], idx;
double d[M], dis[N];
bool st[N];
int n, m, A, B;

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

double dijkstra()
{
    fill(dis + 1, dis + n + 1, 1e8);
    dis[B] = 100;
    priority_queue<PDI, vector<PDI>, greater<PDI> > Q;
    Q.push({ 100, B });
    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] / (1 - d[i]) < dis[e[i]])
                dis[e[i]] = dis[t] / (1 - d[i]), Q.push({ dis[e[i]], e[i]
});

    }
```

```
    return dis[A];
}

int main()
{
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    for (int i = 0; i < m; i++)
    {
        int a, b; double c;
        scanf("%d%d%lf", &a, &b, &c);
        add(a, b, c / 100), add(b, a, c / 100);
    }
    scanf("%d%d", &A, &B);
    printf("%.8lf\n", dijkstra());
    return 0;
}
```

五、AcWing 920. 最优乘车

【题目描述】

H城是一个旅游胜地，每年都有成千上万的人前来观光。

为方便游客，巴士公司在各个旅游景点及宾馆，饭店等地都设置了巴士站并开通了一些**单程**巴士线路。

每条单程巴士线路从某个巴士站出发，依次途经若干个巴士站，最终到达终点巴士站。

一名旅客最近到H城旅游，他很想去S公园游玩，但如果从他所在的饭店没有一路巴士可以直接到达S公园，则他可能要先乘某一路巴士坐几站，再下来换乘同一站台的另一路巴士，这样换乘几次后到达S公园。

现在用整数$1,2,\ldots,N$给H城的所有的巴士站编号，约定这名旅客所在饭店的巴士站编号为1，S公园巴士站的编号为N。

写一个程序，帮助这名旅客寻找一个最优乘车方案，使他在从饭店乘车到S公园的过程中**换乘的次数最少**。

【输入格式】

第一行有两个数字M和N，表示开通了M条单程巴士线路，总共有N个车站。

从第二行到第$M+1$行依次给出了第1条到第M条巴士线路的信息，其中第$i+1$行给出的是第i条巴士线路的信息，从左至右按运行顺序依次给出了该线路上的所有站号，相邻两个站号之间用一个空格隔开。

【输出格式】

共一行，如果无法乘巴士从饭店到达S公园，则输出`NO`，否则输出最少换乘次数，换乘次数为0表示不需换车即可到达。

【数据范围】

$1 \leq M \leq 100$

$2 \leq N \leq 500$

【输入样例】

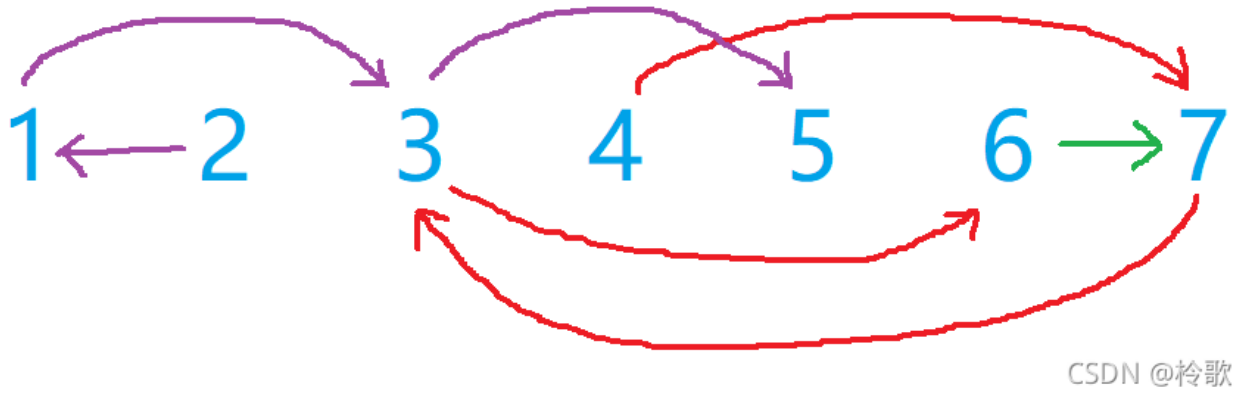
```
3 7
6 7
4 7 3 6
2 1 3 5
```

【输出样例】

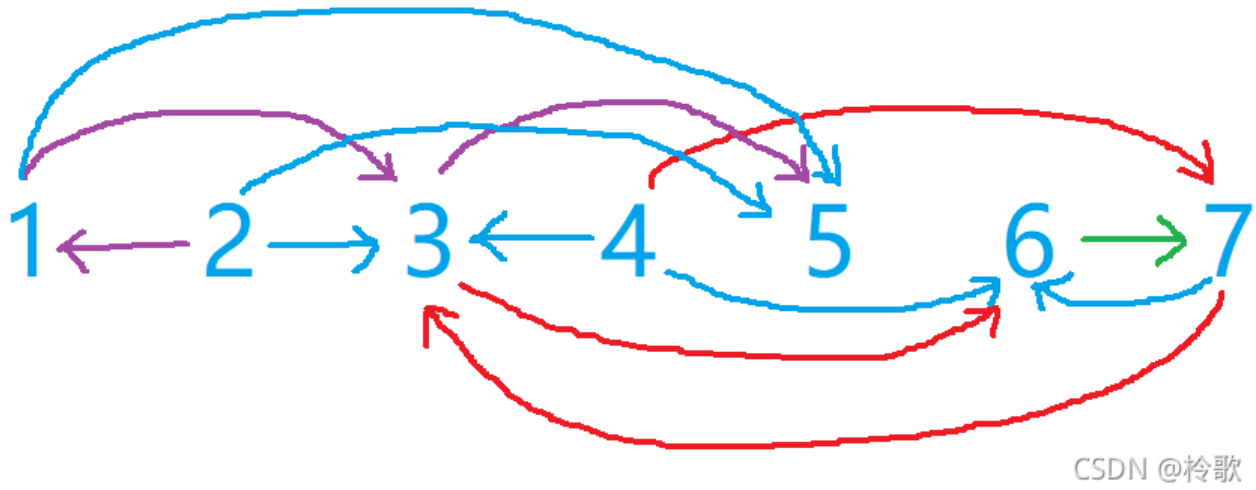
```
2
```

【分析】

以输入样例为例，可画出示意图如下（其中不同颜色的路线表示不同的巴士线路）：



因此如果从1号点上车，在这次乘车机会中可以到达3号点和5号点，即花费乘车次数都为1；同理如果在4号点上车，那么花费一次乘车次数即可到达$7,3,6$中任意一个点，所以可以将图重新建为以下的形式：



其中每条边的权值都为1，将某个点能到达的点都连上边，比如1经过两站能到达5，因此从1号点连一条权值为1的边到5号点，意为在1号点只需要花费一次乘车次数即可到5号点，其它边同理。

最后，由于在起点上车不属于转车，因此求出从起点到终点所需的最少乘车次数−1即为最少转车次数。

【代码】

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

const int N = 510, M = 50010;
bool g[N][N];
int dis[N];
int n, m;

void bfs()
{
    memset(dis, 0x3f, sizeof dis);
    queue<int> Q;
    Q.push(1);
    dis[1] = 0;
    while (Q.size())
    {
        auto t = Q.front();
        Q.pop();
```

```
        if (t == n) break;
        for (int i = 1; i <= n; i++)
            if (g[t][i] && dis[t] + 1 < dis[i])
                dis[i] = dis[t] + 1, Q.push(i);
    }
}

int main()
{
    cin >> m >> n;
    string str;
    getline(cin, str);
    while (m--)
    {
        getline(cin, str);
        stringstream ssin(str);
        int cnt = 0, t;
        int stop[N];
        while (ssin >> t) stop[cnt++] = t;
        for (int i = 0; i < cnt - 1; i++)
            for (int j = i + 1; j < cnt; j++)
                g[stop[i]][stop[j]] = true;
    }
    bfs();
    if (dis[n] == 0x3f3f3f3f) puts("NO");
    else cout << dis[n] - 1 << endl;
    return 0;
}
```

六、AcWing 903. 昂贵的聘礼

【题目描述】

年轻的探险家来到了一个印第安部落里。

在那里他和酋长的女儿相爱了，于是便向酋长去求亲。

酋长要他用10000个金币作为聘礼才答应把女儿嫁给他。

探险家拿不出这么多金币，便请求酋长降低要求。

酋长说：“嗯，如果你能够替我弄到大祭司的皮袄，我可以只要8000金币。如果你能够弄来他的水晶球，那么只要5000金币就行了。”

探险家就跑到大祭司那里，向他要求皮袄或水晶球，大祭司要他用金币来换，或者替他弄来其他的东西，他可以降低价格。

探险家于是又跑到其他地方，其他人也提出了类似的要求，或者直接用金币换，或者找到其他东西就可以降低价格。

不过探险家没必要用多样东西去换一样东西，因为不会得到更低的价格。

探险家现在很需要你的帮忙，让他用最少的金币娶到自己的心上人。

另外他要告诉你的是，在这个部落里，等级观念十分森严。

地位差距超过一定限制的两个人之间不会进行任何形式的直接接触，包括交易。

他是一个外来人，所以可以不受这些限制。

但是如果他和某个地位较低的人进行了交易，地位较高的的人不会再和他交易，他们认为这样等于是间接接触，反过来也一样。

因此你需要在考虑所有的情况以后给他提供一个最好的方案。

为了方便起见，我们把所有的物品从1开始进行编号，酋长的允诺也看作一个物品，并且编号总是1。

每个物品都有对应的价格P，主人的地位等级L，以及一系列的替代品T_i和该替代品所对应的“优惠价格”V_i。

如果两人地位等级差距超过了M，就不能“间接交易”。

你必须根据这些数据来计算出探险家最少需要多少金币才能娶到酋长的女儿。

【输入格式】

输入第一行是两个整数M, N，依次表示地位等级差距限制和物品的总数。

接下来按照编号从小到大依次给出了N个物品的描述。

每个物品的描述开头是三个非负整数P、L、X，依次表示该物品的价格、主人的地位等级和替代品总数。

接下来X行每行包括两个整数T和V，分别表示替代品的编号和“优惠价格”。

【输出格式】

输出最少需要的金币数。

【数据范围】

$1 \leq N \leq 100$

$1 \leq P \leq 10000$

$1 \leq L, M \leq N$

$0 \leq X < N$

【输入样例】

```
1 4
10000 3 2
2 8000
3 5000
1000 2 1
4 200
3000 2 1
4 200
50 2 0
```

【输出样例】

```
5250
```

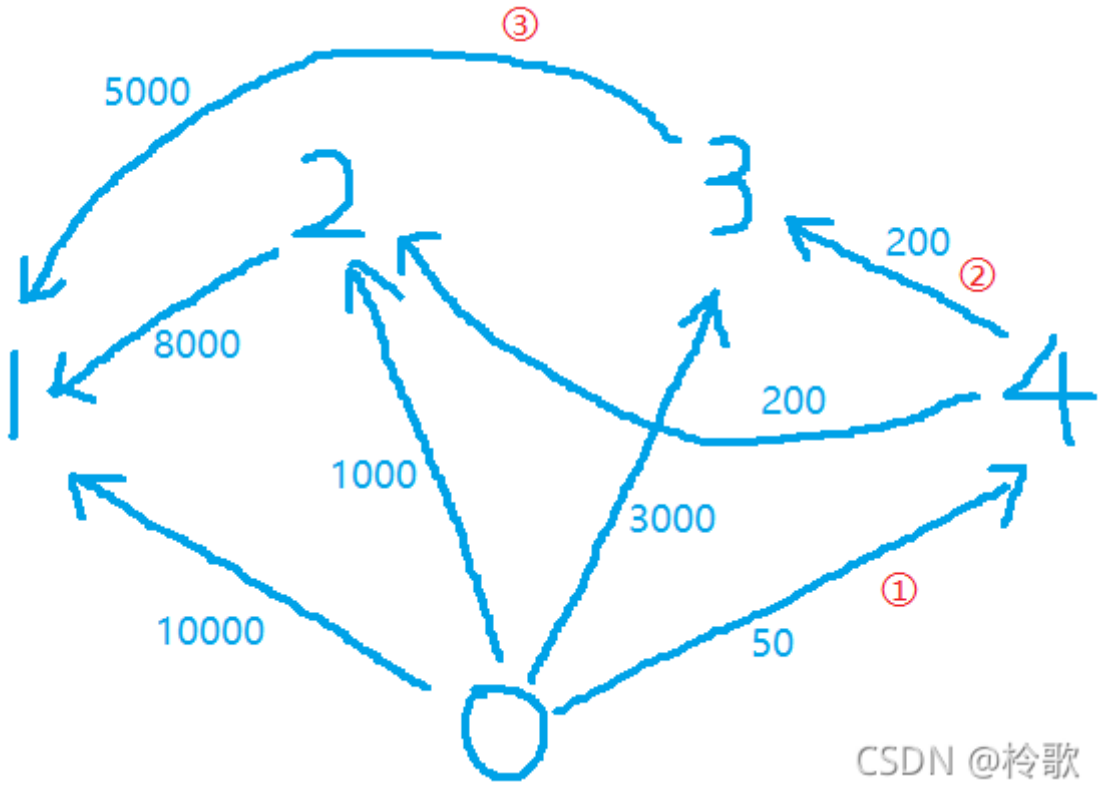
【分析】

---

1. 不考虑等级制度时如何建图？

   根据题意可知，我们要使取得物品**1**所花费的费用最少，因此可将**1**设为终点；由于每个物品i都可以花费$price[i]$元直接购买，因此我们可以设置一个**虚拟源点**（假设编号为**0**），从该源点走到点i的费用为$price[i]$；当取得物品j时物品i可用$value$的优惠价格购买i时，代表从j走到i的费用为$value$。

   以输入样例为例，可建图如下：



因此从虚拟源点到终点1的最短路径如图中红色标号所示，花费为5250。

2. 如何考虑等级制度的影响？

   首先，访问的点的等级差距的最大值不能超过M，而最终目标是取得1号物品，因此1号点的等级$lv[1]$一定要在等级差距区间$[low, high]$中。所以我们可以枚举所有等级区间$[lv[1]-M, lv[1]] \sim [lv[1], lv[1]+M]$，对于每个区间跑一遍$Dijkstra$，只访问等级在区间内的点即可。

【朴素版Dijkstra代码】

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

const int N = 110;
int g[N][N], lv[N];
int dis[N];
bool st[N];
int n, m;

//假设虚拟源点为0号点，返回从虚拟源点到1号点且只走等级在区间[low, high]中的点的最短路径
int dijkstra(int low, int high)
{
```

```
    memset(dis, 0x3f, sizeof dis);
    memset(st, false, sizeof st);
    dis[0] = 0;//从虚拟源点开始走
    for (int i = 0; i <= n; i++)
    {
        int t = -1;
        for (int j = 0; j <= n; j++)
            if (!st[j] && (t == -1 || dis[j] < dis[t]))
                t = j;
        st[t] = true;
        for (int j = 1; j <= n; j++)
            if (lv[j] >= low && lv[j] <= high)
                dis[j] = min(dis[j], dis[t] + g[t][j]);
    }
    return dis[1];
}

int main()
{
    cin >> m >> n;
    memset(g, 0x3f, sizeof g);
    for (int i = 1; i <= n; i++)
    {
        int cnt;
        cin >> g[0][i] >> lv[i] >> cnt;//每个点的价值即为虚拟源点直接走到该点的费用
        for (int j = 0; j < cnt; j++)
        {
            int t, v;
            cin >> t >> v;
            g[t][i] = min(g[t][i], v);
        }
    }
    //枚举所有等级区间(1号点必须在区间内)
    int res = 0x3f3f3f3f;
    for (int i = lv[1] - m; i <= lv[1]; i++) res = min(res, dijkstra(i, i + m));
    cout << res << endl;
    return 0;
}
```

【堆优化版Dijkstra代码】

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

typedef pair<int, int> PII;
const int N = 110, M = 10010;
int e[M], ne[M], d[M], h[N], idx;
int lv[N], dis[N];
bool st[N];
int n, m;

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

int dijkstra(int low, int high)
{
    memset(dis, 0x3f, sizeof dis);
    memset(st, false, sizeof st);
    priority_queue<PII, vector<PII>, greater<PII> > Q;
    Q.push({ 0, 0 });
    dis[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 (lv[e[i]] >= low && lv[e[i]] <= high && dis[t] + d[i] < dis[e[i]])
                dis[e[i]] = dis[t] + d[i], Q.push({ dis[e[i]], e[i] });
    }
    return dis[1];
}

int main()
{
    cin >> m >> n;

    memset(h, -1, sizeof h);
```

```
    for (int i = 1; i <= n; i++)
    {
        int price, cnt;
        cin >> price >> lv[i] >> cnt;
        add(0, i, price);
        for (int j = 0; j < cnt; j++)
        {
            int t, v;
            cin >> t >> v;
            add(t, i, v);
        }
    }
    int res = 0x3f3f3f3f;
    for (int i = lv[1] - m; i <= lv[1]; i++) res = min(res, dijkstra(i, i + m));
    cout << res << endl;
    return 0;
}
```