图论-单源最短路的综合应用

一、AcWing 1135. 新年好（Dijkstra+搜索）

【题目描述】

重庆城里有n个车站，m条**双向**公路连接其中的某些车站。

每两个车站最多用一条公路连接，从任何一个车站出发都可以经过一条或者多条公路到达其他车站，但不同的路径需要花费的时间可能不同。

在一条路径上花费的时间等于路径上所有公路需要的时间之和。

佳佳的家在车站1，他有五个亲戚，分别住在车站a, b, c, d, e。

过年了，他需要从自己的家出发，拜访每个亲戚（顺序任意），给他们送去节日的祝福。

怎样走，才需要最少的时间？

【输入格式】

第一行：包含两个整数n, m，分别表示车站数目和公路数目。

第二行：包含五个整数a, b, c, d, e，分别表示五个亲戚所在车站编号。

以下m行，每行三个整数x, y, t，表示公路连接的两个车站编号和时间。

【输出格式】

输出仅一行，包含一个整数T，表示最少的总时间。

【数据范围】

$1 \le n \le 50000$

$1 \le m \le 10^5$

$1 < a, b, c, d, e \le n$

$1 \le x, y \le n$

$1 \le t \le 100$

【输入样例】

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

【输出样例】

```
21
```

【分析】

假设拜访亲戚的顺序为$1, a, b, c, d, e$，那么该拜访顺序所花费的最小时间为1到a的最小时间$+a$到b的最小时间$+\ldots$。

因此可以先预处理出起点以及每个亲戚到其他各点的最短距离，然后搜索出每种可能的访问顺序，对于某种访问顺序，查表求出该访问顺序所需要的最短距离即可，所有访问顺序中的最短距离即为答案。

【代码】

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

typedef pair<int, int> PII;
const int N = 50010, M = 200010;
int e[M], ne[M], d[M], h[N], idx;
int dis[6][N];//dis[i][]表示第i号亲戚到其他所有点的最短距离
int id[6];//id[i]表示第i号亲戚所在的车站号
bool st[N];
int n, m;
int res = 0x3f3f3f3f;

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

```
}

void dijkstra(int s)
{
    memset(st, false, sizeof st);
    priority_queue<PII, vector<PII>, greater<PII> > Q;
    Q.push({ 0, id[s] });//s号亲戚所在的车站为id[s]
    dis[s][id[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[s][t] + d[i] < dis[s][e[i]])
                dis[s][e[i]] = dis[s][t] + d[i], Q.push({ dis[s][e[i]], e[i] });
    }
}

//step表示当前搜了几个点，x表示搜到第几号亲戚，distance表示总共走过多少距离
void dfs(int step, int x, int distance)
{
    if (distance > res) return;
    if (step == 6) { res = min(res, distance); return; }
    for (int i = 1; i <= 5; i++)//枚举1~5号亲戚的每一种排列顺序
        if (!st[i])
        {
            st[i] = true;
            dfs(step + 1, i, distance + dis[x][id[i]]);
            st[i] = false;
        }
}

int main()
{
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    id[0] = 1;//0号亲戚表示自己家
    for (int i = 1; i <= 5; i++) scanf("%d", &id[i]);
    for (int i = 0; i < m; i++)

    {
```

```
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        add(a, b, c), add(b, a, c);
    }
    memset(dis, 0x3f, sizeof dis);//提前初始化所有的dis
    for (int i = 0; i <= 5; i++) dijkstra(i);//分别求出i号亲戚到其他各点的最短距离
    memset(st, false, sizeof st);//清空st数组，准备搜索
    dfs(1, 0, 0);//从第一个点，0号亲戚也就是自己家开始搜索
    printf("%d\n", res);
    return 0;
}
```

二、AcWing 340. 通信线路（双端队列BFS求最短路+二分）

【题目描述】

在郊区有N座通信基站，P条**双向**电缆，第i条电缆连接基站A_i和B_i。

特别地，1号基站是通信公司的总站，N号基站位于一座农场中。

现在，农场主希望对通信线路进行升级，其中升级第i条电缆需要花费L_i。

电话公司正在举行优惠活动。

农产主可以指定一条从1号基站到N号基站的路径，并指定路径上不超过K条电缆，由电话公司免费提供升级服务。

农场主只需要支付在该路径上剩余的电缆中，升级价格**最贵**的那条电缆的花费即可。

求至少用多少钱可以完成升级。

【输入格式】

第1行：三个整数N, P, K。

第$2 \sim P+1$行：第$i+1$行包含三个整数A_i, B_i, L_i。

【输出格式】

包含一个整数表示最少花费。

若1号基站与N号基站之间不存在路径，则输出`$-1$`。

【数据范围】

$0 \leq K < N \leq 1000$

$1 \leq P \leq 10000$

$1 \leq L_i \leq 1000000$

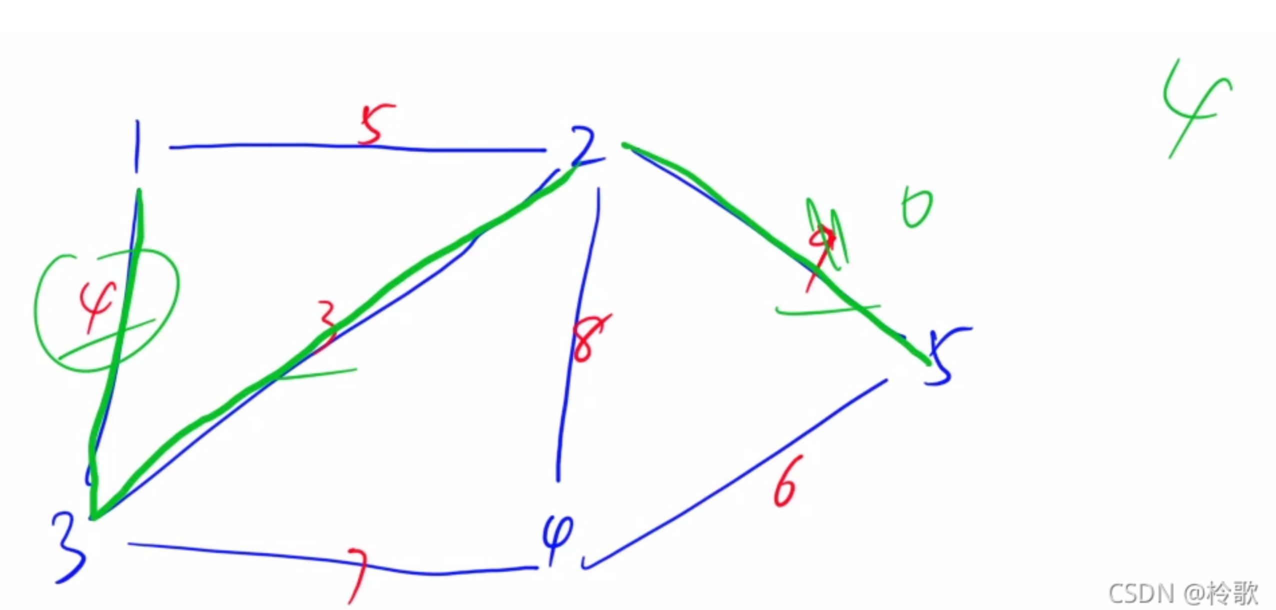
【输入样例】

```
5 7 1
1 2 5
3 1 4
2 4 8
3 2 3
5 2 9
3 4 7
4 5 6
```

【输出样例】

```
4
```

【分析】



首先题意是可以选择一条从1走到N的路径，选择不大于K条边将其边权变为0，因此最优选择一定是按边权**从大到小**选择，将其边权置为0。又因为某条路径的费用为该路径上权值最大的边的费用，因此将前K大的边都置为0后该路径的费用为第$K+1$大的边的权值。

那么我们就可以二分这个第$K+1$大的权值x，如果从1走到N走过的**边权大于x的边的数量的最小值**小于等于K，说明可以将这些边的权值都置为0，那么这条路径的费用就为x，求出满足条件的最小的x即为答案。

那么如何求出从1走到N走过的**边权大于x的边的数量的最小值**呢？我们可以将图看成由边权为$0,1$的边组成的，如果边权大于x，那么将其边权看成1，否则将其边权看成0，那么就可以用双端队列BFS求出1到N的最短路径长度，那么这个长度即表示从1到N走过的**边权大于x的边的数量的最小值**。

【代码】

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

const int N = 1010, M = 20010;
int e[M], ne[M], d[M], h[N], idx;
int dis[N];
bool st[N];
int n, m, k;

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

//判断从1走到n经过最少的边权大于mid的边的数量是否小于等于k
bool check(int mid)
{
    memset(st, false, sizeof st);
    memset(dis, 0x3f, sizeof dis);
    dis[1] = 0;
    deque<int> Q;
    Q.push_back(1);
    while (Q.size())
    {
        auto t = Q.front();
        Q.pop_front();
        if (st[t]) continue;
        st[t] = true;
        for (int i = h[t]; ~i; i = ne[i])
        {
            int v = d[i] > mid;//如果边权大于mid则记为1

            if (dis[t] + v < dis[e[i]])
```

```
            {
                dis[e[i]] = dis[t] + v;
                if (v) Q.push_back(e[i]);
                else Q.push_front(e[i]);
            }
        }
    }
    return dis[n] <= k;
}

int main()
{
    cin >> n >> m >> k;
    memset(h, -1, sizeof h);
    while (m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c), add(b, a, c);
    }
    int l = 0, r = 1e6 + 1;//r取1e6 + 1目的是区分无解与最大值解两种情况
    while (l < r)
    {
        int mid = l + r >> 1;
        if (check(mid)) r = mid;
        else l = mid + 1;
    }
    cout << (r == 1e6 + 1 ? -1 : r) << endl;
    return 0;
}
```

三、AcWing 342. 道路与航线（Dijkstra+拓扑排序）

【题目描述】

农夫约翰正在一个新的销售区域对他的牛奶销售方案进行调查。

他想把牛奶送到T个城镇，编号为$1 \sim T$。

这些城镇之间通过R条道路（编号为$1 \sim R$）和P条航线（编号为$1 \sim P$）连接。

每条道路i或者航线i连接城镇A_i到B_i，花费为C_i。

对于道路，$0 \le C_i \le 10,000$；然而航线的花费很神奇，花费C_i可能是负数$(-10,000 \le C_i \le 10,000)$。

道路是双向的，可以从A_i到B_i，也可以从B_i到A_i，花费都是C_i。

然而航线与之不同，只可以从A_i到B_i。

事实上，由于最近恐怖主义太嚣张，为了社会和谐，出台了一些政策：保证如果有一条航线可以从A_i到B_i，那么保证不可能通过一些道路和航线从B_i回到A_i。

由于约翰的奶牛世界公认十分给力，他需要运送奶牛到每一个城镇。

他想找到从发送中心城镇S把奶牛送到每个城镇的最便宜的方案。

【输入格式】

第一行包含四个整数T, R, P, S。

接下来R行，每行包含三个整数（表示一个道路）A_i, B_i, C_i。

接下来P行，每行包含三个整数（表示一条航线）A_i, B_i, C_i。

【输出格式】

第$1 \sim T$行：第i行输出从S到达城镇i的最小花费，如果不存在，则输出`NO PATH`。

【数据范围】

$1 \le T \le 25000$

$1 \le R, P \le 50000$

$1 \le A_i, B_i, S \le T$

【输入样例】

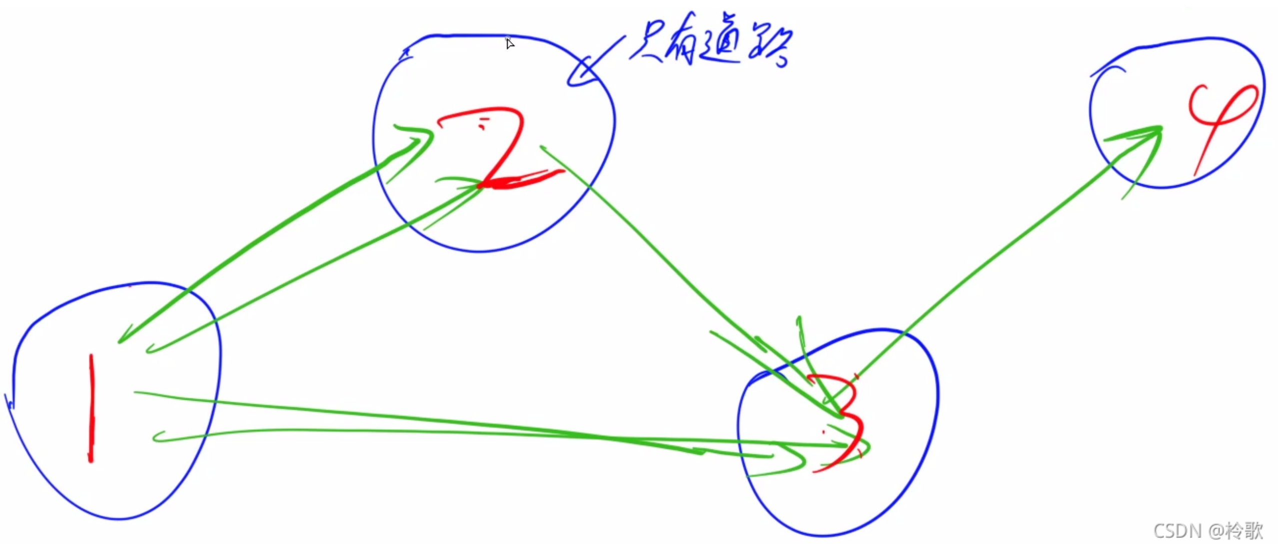
```
6 3 3 4
1 2 5
3 4 5
5 6 10
3 5 -100
4 6 -100
1 3 -10
```

【输出样例】

```
NO PATH
NO PATH
5
0
-95
-100
```

【分析】

首先说明：**本题SPFA会被卡！！！**



如上图所示，我们可将所有由道路（无向边）连接的连通块看成一个团，航线（有向边）将每个团连接起来，那么在这个**团的内部**由于各边都为道路，**边权都是非负的**，因此可以使用$Dijkstra$求最短距离；然后由于航线不会形成环，因此团与团之间一定存在**拓扑序列**，只要按照拓扑序列处理每个团，最后求出的结果一定也是最短距离。

时间复杂度分析：

- $Dijkstra$算法时间复杂度为$O(mlogn)$。
- 拓扑图不管边权是正是负均可按拓扑序扫描，时间复杂度是线性的。

具体求解思路：

1. 先输入所有双向道路，然后DFS出所有连通块，计算出两个数组：`id[]`存储每个点属于哪个连通块；`vector<int> block[]`存储每个连通块里有哪些点；
2. 输入所有航线，同时统计出每个连通块的入度；
3. 按照拓扑序依次处理每个连通块。先将所有入度为0的连通块的编号加入队列中；
4. 每次从队头取出一个连通块的编号`bid`；
5. 将`block[bid]`中的所有点加入优先队列中，然后对优先队列中的所有点跑一遍$Dijkstra$算法；
6. 即每次取出优先队列中的第一个点`ver`；

7. 遍历`ver`的所有邻点`j`，如果`id[ver] == id[j]`，那么如果`j`能被更新，则将`j`插入优先队列中；如果`id[ver] != id[j]`，则将`id[j]`所在连通块的入度`-1`，如果减为0了，则将该连通块插入拓扑排序的队列中。

【代码】

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

typedef pair<int, int> PII;
const int N = 25010, M = 150010;
int e[M], ne[M], d[M], h[N], idx;
int dis[N], id[N], in[N], bcnt;//id为每个点所在的连通块标号，in为每个连通块的入度
int t, r, p, s;
bool st[N];
vector<int> block[N];//记录每个连通块中的点
queue<int> Q;//拓扑排序队列

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

void dfs(int x, int bid)
{
    id[x] = bid;
    block[bid].push_back(x);
    for (int i = h[x]; ~i; i = ne[i])
        if (!id[e[i]]) dfs(e[i], bid);
}

void dijkstra(int bid)
{
    priority_queue<PII, vector<PII>, greater<PII> > PQ;
    for (auto x : block[bid]) PQ.push({ dis[x], x });//先将该连通块的所有点入队

    while (PQ.size())
```

```
    {
        auto t = PQ.top().second;
        PQ.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];
                if (id[e[i]] == id[t]) PQ.push({ dis[e[i]], e[i] });//如果两点在同一个连通块中则入队
            }
            if (id[e[i]] != id[t] && !--in[id[e[i]]]) Q.push(id[e[i]]);//如果不在同一个连通块中则所连接的那个连通块入度-1
        }
    }
}

void topSort()
{
    memset(dis, 0x3f, sizeof dis);
    dis[s] = 0;
    for (int i = 1; i <= bcnt; i++)
        if (!in[i]) Q.push(i);
    while (Q.size())//按拓扑序对每个连通块分别做dijkstra
    {
        auto t = Q.front();
        Q.pop();
        dijkstra(t);
    }
}

int main()
{
    ios::sync_with_stdio(false);
    cin >> t >> r >> p >> s;
    memset(h, -1, sizeof h);
    int a, b, c;
    while (r--)//读入道路
    {
        cin >> a >> b >> c;

        add(a, b, c), add(b, a, c);
```

```
    }
    for (int i = 1; i <= t; i++)//读入完道路后求出每个连通块
        if (!id[i]) dfs(i, ++bcnt);
    while (p--)//读入航线
    {
        cin >> a >> b >> c;
        add(a, b, c);
        in[id[b]]++;
    }
    topSort();
    for (int i = 1; i <= t; i++)
        if (dis[i] > 0x3f3f3f3f >> 1) cout << "NO PATH" << endl;
        else cout << dis[i] << endl;
    return 0;
}
```

四、AcWing 341. 最优贸易（SPFA+DP）

【题目描述】

C国有n个大城市和m条道路，每条道路连接这n个城市中的某两个城市。

任意两个城市之间最多只有一条道路直接相连。

这m条道路中有一部分为单向通行的道路，一部分为双向通行的道路，双向通行的道路在统计条数时也计为1条。

C国幅员辽阔，各地的资源分布情况各不相同，这就导致了同一种商品在不同城市的价格不一定相同。

但是，同一种商品在同一个城市的买入价和卖出价始终是相同的。

商人阿龙来到C国旅游。

当他得知"同一种商品在不同城市的价格可能会不同"这一信息之后，便决定在旅游的同时，利用商品在不同城市中的差价赚一点旅费。

设C国n个城市的标号从$1 \sim n$，阿龙决定从1号城市出发，并最终在n号城市结束自己的旅行。

在旅游的过程中，**任何城市可以被重复经过多次**，但不要求经过所有n个城市。

阿龙通过这样的贸易方式赚取旅费：他会选择一个经过的城市买入他最喜欢的商品——水晶球，并在之后经过的另一个城市卖出这个水晶球，用赚取的差价当做旅费。

因为阿龙主要是来C国旅游，他决定这个贸易只进行最多一次，当然，在赚不到差价的情况下他就无需进行贸易。

现在给出n个城市的水晶球价格，m条道路的信息（每条道路所连接的两个城市的编号以及该条道路的通行情况）。

请你告诉阿龙，他最多能赚取多少旅费。

注意：本题数据有加强。

【输入格式】

第一行包含2个正整数n和m，中间用一个空格隔开，分别表示城市的数目和道路的数目。

第二行n个正整数，每两个整数之间用一个空格隔开，按标号顺序分别表示这n个城市的商品价格。

接下来m行，每行有3个正整数，x, y, z，每两个整数之间用一个空格隔开。

如果$z = 1$，表示这条道路是城市x到城市y之间的单向道路；如果$z = 2$，表示这条道路为城市x和城市y之间的双向道路。

【输出格式】

一个整数，表示答案。

【数据范围】

$1 \leq n \leq 100000$

$1 \leq m \leq 500000$

$1 \leq$ 各城市水晶球价格 ≤ 100

【输入样例】

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

【输出样例】

```
5
```

【分析】

$SPFA$算法时间复杂度：$O(n+km)$。

先求出：

- 从1走到i的过程中，买入水晶球的最低价格`dmin[i]`；
- 从i走到n的过程中，卖出水晶球的最高价格`dmax[i]`。

然后枚举每个城市作为买卖的中间城市，求出`dmax[i] - dmin[i]`的最大值即可。

求`dmin[i]`和`dmax[i]`时，由于不是拓扑图，状态的更新**可能存在环**，因此不能使用动态规划，只能使用求最短路的方式。

另外，我们考虑能否使用$Dijkstra$算法，如果当前`dmin[i]`最小的点是5，那么有可能存在边`5->6, 6->7, 7->5`，假设当前`dmin[5] = 10`，则有可能存在6的价格是11，但7的价格是3，那么`dmin[5]`的值就应该被更新成3，因此当前最小值也不一定是最终最小值，**所以$Dijkstra$算法并不适用**，我们只能采用$SPFA$算法。

时间复杂度：

瓶颈是$SPFA$，$SPFA$算法的时间复杂度是$O(km)$，其中k一般情况下是个很小的常数，最坏情况下是n，n表示总点数，m表示总边数。因此总时间复杂度是$O(km)$。

【代码】

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

const int N = 100010, M = 2000010;
int h[N], rh[N], e[M], ne[M], idx;//rh为反图表头结点，便于求出各点到n的最大值
int dmin[N], dmax[N];//dmin[i]表示从1走到i能够经过的最小的点，dmax[i]表示从i走到n能经过的最大的点
int w[N];
bool st[N];
int n, m, res;

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

```
}

//s表示起点，flag为true时表示在正图上求dmin，反之表示在反图上求dmax
void spfa(int h[], int s, int dis[], bool flag)
{
    if (flag) memset(dis, 0x3f, sizeof dmin);//注意此处的size要取dmin的
    queue<int> Q;
    dis[s] = w[s];
    Q.push(s);
    st[s] = true;
    while (Q.size())
    {
        auto t = Q.front();
        Q.pop();
        st[t] = false;
        for (int i = h[t]; ~i; i = ne[i])
            if (flag && min(dis[t], w[e[i]]) < dis[e[i]] || !flag && max(dis[t], w[e[i]]) > dis[e[i]])
            {
                if (flag) dis[e[i]] = min(dis[t], w[e[i]]);
                else dis[e[i]] = max(dis[t], w[e[i]]);
                if (!st[e[i]]) Q.push(e[i]), st[e[i]] = true;
            }
    }
}

int main()
{
    ios::sync_with_stdio(false);
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> w[i];
    memset(h, -1, sizeof h);
    memset(rh, -1, sizeof rh);
    while (m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        add(h, a, b), add(rh, b, a);
        if (c == 2) add(h, b, a), add(rh, a, b);
    }
    spfa(h, 1, dmin, true);
    spfa(rh, n, dmax, false);

    for (int i = 1; i <= n; i++) res = max(res, dmax[i] - dmin[i]);
```

```
    cout << res << endl;
    return 0;
}
```