**★★★★★**

**題組：Problem Set Archive** **with Online Judge**

**題號：12092: Paint the Roads**

**解題者：郭紫葳**

**解題日期：2025年5月1日**

**題意：**

在某個國家裡有 n 座城市，以 m 條單向道路互相連接。已知把一條長度為 d 的道路刷漆，需要 d 單位的cost，則題目要求選擇部分道路進行「刷漆」，使得被刷漆的道路集合可以被分割成若干互不重疊的有向環（disjoint cycles），並且每一個城市恰好出現在其中的 k 個有向環裡。在滿足上述條件下，要求總花費最小。若無法達成條件，輸出 -1。

**題意範例：**

4

4 8 1

0 1 1

1 0 2

2 3 1

3 2 2

0 2 5

2 0 6

1 3 5

3 1 6 🡪 6

3 4 1

0 1 5

1 0 6

0 2 7

2 0 8 🡪 -1  
**解法：**

該題可看成minimum cost maximum flow problem，如果符合題目條件，則圖之最小cost即為答案。

將每個城市拆成a, b兩個點 ( a點當作out-degree；b點當作in-degree)，並且外加兩個點作為source以及sink，其中source連到所有城市的a點，所有城市的b點連到sink，edge’s capacity設為k，確保該點最多用過k次，edge’s cost設為0。如果i城市和j城市之間有道路，則由i城市的a點連向j城市的b點，edge’s capacity設為1，確保該邊只能用一次，edge’s cost設為道路長度d。

若要滿足題目，則每個點的in-degree = out-degree = k，也代表每個城市恰好出現在k個disjoint cycle中，在新圖中則是代表最終總匯流量要為k\*n。

其中，解minimum cost maximum flow problem時，採用Shortest Path Faster Algorithm (SPFA)來找cost最小的增廣路徑。

**解法範例：**

假設題目為：

4 8 1

0 1 1

1 0 2

2 3 1

3 2 2

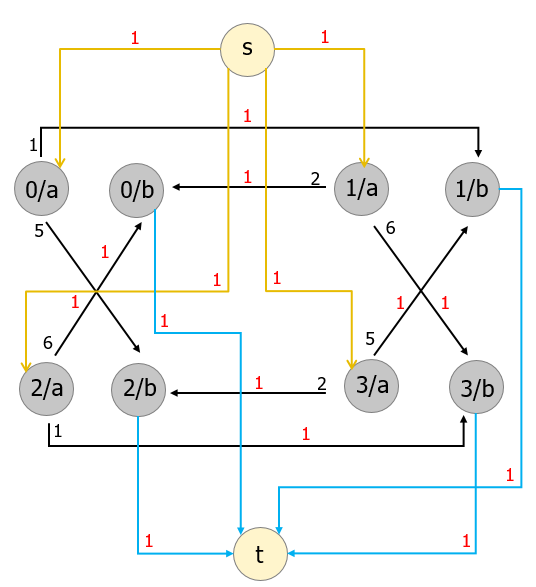
0 2 5

2 0 6

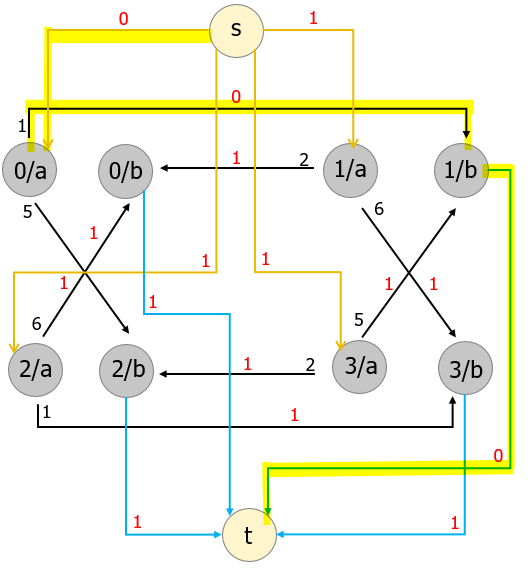
1 3 5

3 1 6

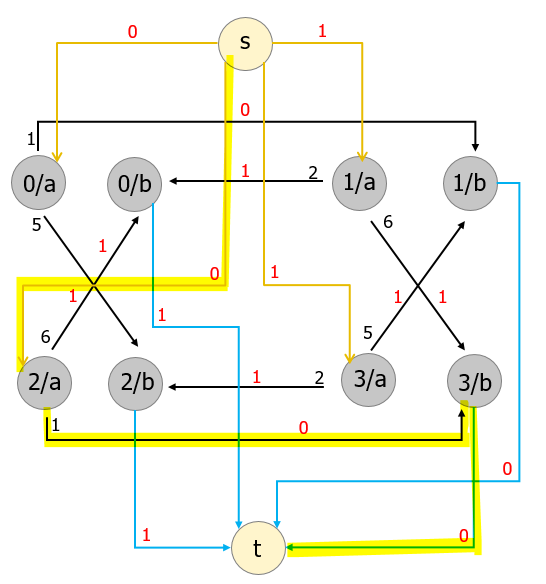
先建立新圖，並且根據上述演算法，依據選擇最短增廣路徑，直到無法無增廣路徑可以選擇。



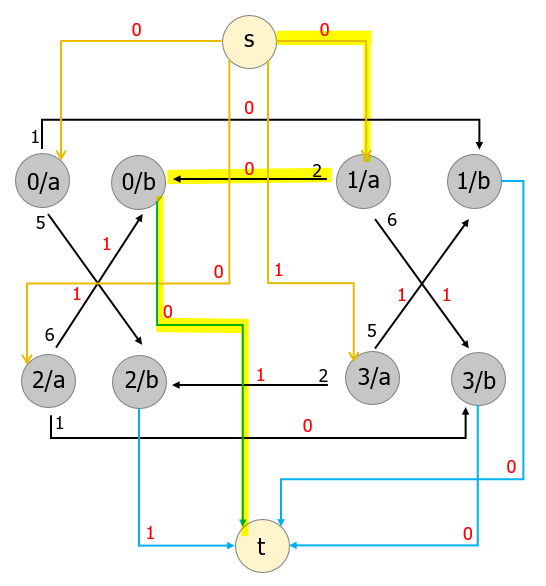
利用SPFA選擇第一條增廣路徑，此時cost為1。



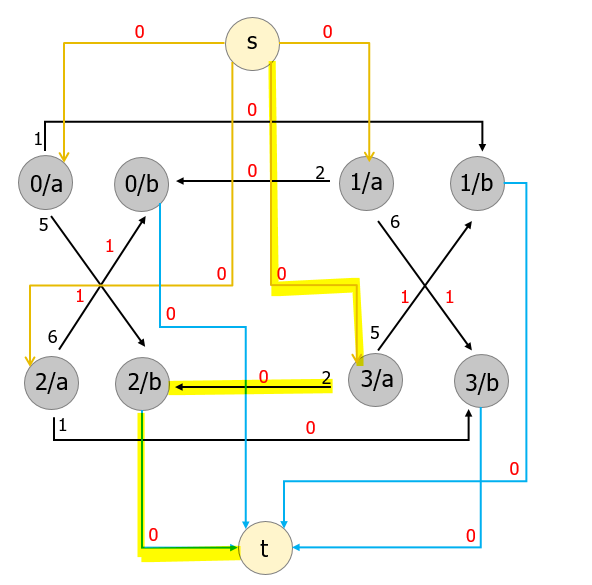
利用SPFA選擇第二條增廣路徑，此時cost為2。



利用SPFA選擇第三條增廣路徑，此時cost為4。

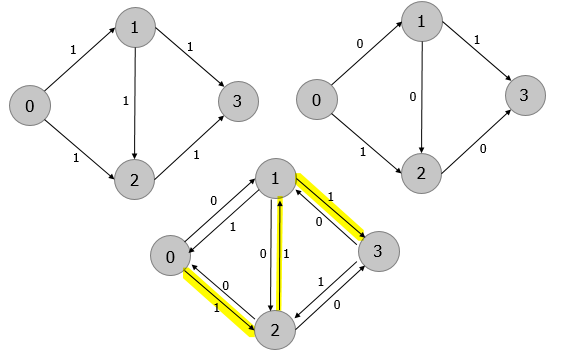


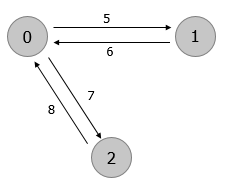
利用SPFA選擇第四條增廣路徑，此時cost為6。



此時，無增廣路徑可走，且已經為最大匯流，則6即為最小cost。

**討論：**

1. 為了避免過程中選錯路徑，因此需加入反向邊，在未來才能做修正路徑的動作。
2. 需要注意 disjoint cycle 定義，一個城市必須剛好在 k 個 cycle 中，也就是 simple cycle。  
     
   如下圖，可能會覺得 cycle 可以為 1 → 0 → 2 → 0 → 1，但事實上，這個並不是合法的 disjoint cycle，因為節點 0 在同一組 cycle 中被走了兩次，違反了 disjoint 的要求。



**程式：**

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N = 500;

const int M = 5000;

const int INF = 1e9;

int n, m, k, s, t;

int pre[N], inq[N];

ll temp\_flow[N], d[N];

struct Edge{

    int from, to;

    ll cap, flow;

    ll cos;

};

vector<Edge> edges;

vector<int> adj[M];

void addEdge(int from, int to, ll cap, ll co) {

    edges.push\_back((Edge){from, to, cap, 0, co});

    edges.push\_back((Edge){to, from, 0, 0, -co});

*//加入反向邊，以利之後做修正*

    int m = edges.size();

    adj[from].push\_back(m-2);

    adj[to].push\_back(m-1);

}

int mincost(int s, int t, ll& flow, ll& cost) {

*//SPFA找最短增廣路徑*

    queue<int> Q;

    memset(inq, 0, sizeof(inq));

    memset(temp\_flow, 0, sizeof(temp\_flow));

    memset(pre, 0, sizeof(pre));

    for (int i = 0; i < N; i++) d[i] = INF;

    d[s] = 0;

    temp\_flow[s] = INF;

    inq[s] = 1;

    pre[s] = 0;

    Q.push(s);

    while (!Q.empty()) {

        int u = Q.front();

        Q.pop();

        inq[u] = 0;

        for (int i = 0; i < adj[u].size(); i++) {

            Edge &e = edges[adj[u][i]];

            if (e.cap > e.flow && d[e.to] > d[u] + e.cos) {

                d[e.to] = d[u] + e.cos;

                temp\_flow[e.to] = min(temp\_flow[u], e.cap - e.flow);

                pre[e.to] = adj[u][i];

                if (!inq[e.to]) {

                    inq[e.to] = 1;

                    Q.push(e.to);

                }

            }

        }

    }

    if (d[t] == INF) return 0;

    flow += temp\_flow[t];

    cost += (ll)d[t] \* (ll)temp\_flow[t];

    for (int u = t; u != s; u = edges[pre[u]].from) {

        edges[pre[u]].flow += temp\_flow[t];

        edges[pre[u]^1].flow -= temp\_flow[t];

    }

    return 1;

}

int MCMF(int s, int t, ll& cost) {

    ll flow = 0;

    cost = 0;

    while ( mincost(s, t, flow, cost) );

    return flow;

}

int main() {

    int kase;

    cin >> kase;

    while ( kase-- ) {

        for (int i = 0; i <= m \* 2; i++) adj[i].clear();

        edges.clear();

        cin >> n >> m >> k;

        s = 2\*n + 1;

        t = 2\*n + 2;

        int u, v, d;

        for (int i = 0; i < n; i++) { *//拆成a與b兩點進行討論*

            addEdge(s, i, k, 0);

            addEdge(i+n, t, k, 0);

        }

        for (int i = 0; i < m; i++) { *//加入i城市到j城市的edge*

            cin >> u >> v >> d;

            addEdge(u, v+n, 1, d);

        }

        ll cost;

        int total = MCMF(s, t, cost);

        if (total < n\*k)  cout << "-1\n";

        else cout << cost << endl;

    }

    return 0;

}