逛公园

2021 年 5 月 21 日

逛公园

时间限制：C/C++ 3 秒，其他语言 6 秒空间限制：C/C++ 262144K，其他语言 524288K 64bit IO Format: %lld

题目描述

策策同学特别喜欢逛公园。公园可以看成一张 N 个点 M 条边构成的有向图，且没有自环和重边。其中 1 号点是公园的入口，N 号点是公园的出口，每条边有一个非负权值，代表策策经过这条边所要花的时间。策策每天都会去逛公园，他总是从 1 号点进去，从 N 号点出来。策策喜欢新鲜的事物，他不希望有两天逛公园的路线完全一样，同时策策还是一个特别热爱学习的好孩子，他不希望每天在逛公园这件事上花费太多的时间。如果 1 号点到 N 号点的最短路长为 d，那么策策只会喜欢长度不超过 d + K 的路线。策策同学想知道总共有多少条满足条件的路线，你能帮帮他吗？为避免输出过大，答案对 P 取模。如果有无穷多条合法的路线，请输出 −1。

输入描述:

第一行包含一个整数 T, 代表数据组数。接下来 T 组数据，对于每组数据：第一行包含四个整数 N,M,K,P，每两个整数之间用一个空格隔开。接下来 M 行，每行三个整数 ai,bi,ci，代表编号为 ai,bi 的点之间有一条权值为 ci 的有向边，每两个整数之间用一个空格隔开。

输出描述:

输出文件包含 T 行，每行一个整数代表答案。#### 示例 1 > 输入

```
2
5 7 2 10
1 2 1
2 4 0
4 5 2
2 3 2
3 4 1
3 5 2
1 5 3
2 2 0 10
1 2 0
2 1 0
```

输出

```
3
-1
```

说明

对于第一组数据，最短路为 3。1 - 5, 1 - 2 - 4 - 5, 1 - 2 - 3 - 5 为 3 条合法路径。
备注: 对于不同测试点，我们约定各种参数的规模不会超过如下：

思路

正图和反图各跑一边 Dijkstra，然后进行记忆化搜索即可。

$dp[i][j]$ 表示到达第 i 个点，比最短路最多多走 j 个距离的路径数。

代码

```
#include<bits/stdc++.h>

using namespace std;

typedef pair<long long, int> pli;
const int maxn = 2e5+1;

struct node {
```

测试点编号	T	N	M	K	是否有 0 边
1	5	5	10	0	否
2	5	1000	2000	0	否
3	5	1000	2000	50	否
4	5	1000	2000	50	否
5	5	1000	2000	50	否
6	5	1000	2000	50	是
7	5	100000	200000	0	否
8	3	100000	200000	50	否
9	3	100000	200000	50	是
10	3	100000	200000	50	是

图 1: img

```
    long long to, w;
};
long long n, m, k, p;
vector<node> g[maxn];
vector<node> rg[maxn];
long long dis[maxn], rdis[maxn], dp[maxn][51];
bool vis[maxn], rvis[maxn], flag[maxn][51];

void Dijkstra(int s) {
    for(int i = 1; i <= n; ++ i) {
        dis[i] = INT_MAX;
        vis[i] = false;
    }
    priority_queue<pli, vector<pli>, greater<> > que;
    que.push({dis[s] = 0, s});
    while(!que.empty()) {
        pli t = que.top();
        que.pop();
```

```
        if (vis[t.second]) continue;
        vis[t.second] = true;
        for(auto to: g[t.second]) {
            if (dis[to.to] > dis[t.second] + to.w) {
                que.push({dis[to.to] = dis[t.second] + to.w, to.to});
            }
        }
    }
}

void rDijkstra(int s) {
    for(int i = 1; i <= n; ++ i) {
        rdis[i] = INT_MAX;
        rvis[i] = false;
    }
    priority_queue<pli, vector<pli>, greater<> > que;
    que.push({rdis[s] = 0, s});
    while(!que.empty()) {
        pli t = que.top();
        que.pop();
        if (rvis[t.second]) continue;
        rvis[t.second] = true;
        for(auto to: rg[t.second]) {
            if (rdis[to.to] > rdis[t.second] + to.w) {
                que.push({rdis[to.to] = rdis[t.second] + to.w, to.to});
            }
        }
    }
}

long long dfs(int x, int y) {
    if (flag[x][y]) return -1;
    if (dp[x][y] != -1) return dp[x][y];
    flag[x][y] = true;
    long long ans = 0;
```

```
    if (x == n) ans = 1;
    for(auto to: g[x]) {
        if (rdis[to.to] == INT_MAX) continue;
        long long t = to.w + rdis[to.to] - rdis[x];
        if (t <= y) {
            long long son = dfs(to.to, y-t);
            if (son == -1) return -1;
            ans = (ans + son) % p;
        }
    }
    flag[x][y] = false;
    return dp[x][y] = ans%p;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int T;
    cin >> T;
    while(T --) {
        cin >> n >> m >> k >> p;
        for(int i = 1; i <= m; ++ i) {
            int u, v, w;
            cin >> u >> v >> w;
            g[u].push_back({v, w});
            rg[v].push_back({u, w});
        }
        Dijkstra(1);
        rDijkstra(n);
        for(int i = 1; i <= n; ++ i) {
            for(int j = 0; j <= 50; ++ j) {
                dp[i][j] = -1;
                flag[i][j] = false;
            }
        }
```

```
        cout << dfs(1, k) << "\n";
        for(int i = 1; i <= n; ++ i) {
            g[i].clear();
            rg[i].clear();
        }
    }
    return 0;
}
```