J - To the Paradise

题目描述

给出 n 个结点 m 条边的**有向无环带权图**，结点依次标号 1 至 n，第 i 条边由结点 a_i 连向 b_i，长度为 w_i。

给出 q 次询问，第 i 次询问会选择三个结点 u_i，v_i 与 s_i，你需要求出在**不经过** s_i 的条件下从 u_i 到 v_i 的最短距离。

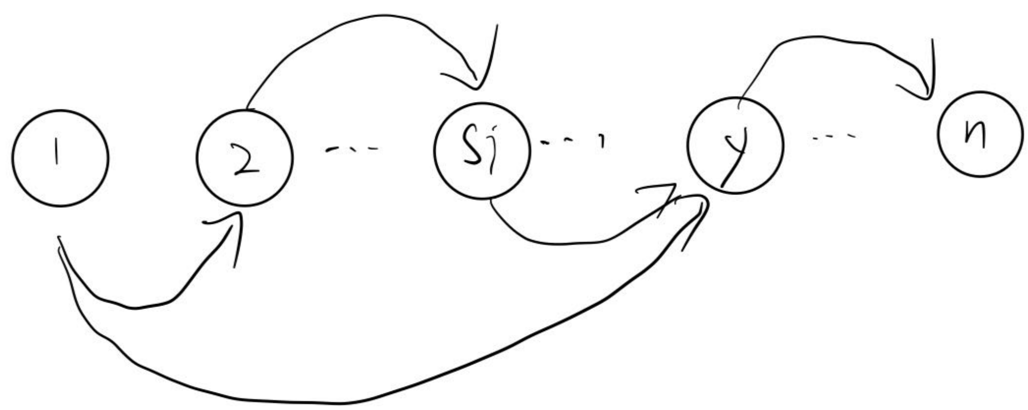
如果在**不经过** s_i 的条件下**不能**从 u_i 到达 v_i，我们认为答案为 0。

每次询问后需要你立刻求出答案，因此你在读取 u_i, v_i, s_i 后需要对这三个值进行以下操作以得到正确的输入：

$$\begin{aligned} u_i &= (u_i + \text{last}) \bmod n + 1 \\ v_i &= (v_i + \text{last}) \bmod n + 1 \\ s_i &= (s_i + \text{last}) \bmod n + 1 \end{aligned}$$

其中 last 表示上一次询问的答案。如果之前没有询问，那么 last 为 0。

按照顶点在原图的拓扑序重新标号1...n，建图。



在新图中，一定不会存在一条边(u,v)使得$u > v$

根据拓扑序的性质，任意一条u到v的路径，$s_1 s_2 \dots s_n$，有$s_1 < s_2 < \dots < s_n$

因此对于所有不经过s_i的$u \to v$的路径：

- $s_i < u$或$v < s_i$时，等价于$u \to v$的最短路径
- $s_i = u$或$s_i = v$时，不存在这样的路径
- $u < s_i < v$时，假设路径中第一个编号$> s_i$的点是y:

  路径由两部分组成，

  $u \to y$（只经过编号$< s_i$的点），$y \to v$

可以考虑使用Floyd算法计算出$d(i,j,k)$（即$i \to j$的只经过编号$\le k$的点的最短路径）

因此对于所有不经过s_i的$u \to v$的路径：

- $s_i < u$或$v < s_i$时，答案为$d(u,v,n)$
- $s_i = u$或$s_i = v$时，不存在这样的路径

- $u < s_i < v$时，
  - 答案为$max_{y>s_i}\{d(i,y,s_i-1)+d(y,j,n)\}$

在进行答案查询时需要将点的编号转化为新图中的编号（拓扑序）

时间复杂度为$O(n^3+qn)$

代码

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

using namespace std;

typedef pair<int, int> ii;

int n, m, q;
int id[301];
vector<ii> edge[301];
int indeg[301];
int dist[301][301][301];
const int INF = 0x3f3f3f3f;

void toposort() {
    queue<int> q;
    for (int u = 1; u <= n; ++u) {
        if (indeg[u] == 0) {
            q.push(u);
        }
    }

    int idx = 0;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (auto [w, v] : edge[u]) {
            --indeg[v];
            if (indeg[v] > 0) continue;
            q.push(v);
        }
        id[u] = ++idx;
    }
}

void floyd() {
    for (int k = 1; k <= n; ++k) {
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j) {
                dist[i][j][k] = dist[i][j][k - 1];
                if (dist[i][k][k - 1] + dist[k][j][k - 1] < dist[i][j][k - 1]) {
                    dist[i][j][k] = dist[i][k][k - 1] + dist[k][j][k - 1];
                }
            }
        }
    }
}
```

```cpp
int shortestPath(int st, int ed, int ban) {
    if (ban < st || ban > ed) {
        return dist[st][ed][n];
    } else if (ban == st || ban == ed) {
        return INF;
    }

    int ans = INF;
    for (int u = ban + 1; u <= ed; ++u) {
        ans = min(ans, dist[st][u][ban - 1] + dist[u][ed][n]);
    }
    return ans;
}

int main() {
    cin.tie(NULL);
    ios::sync_with_stdio(false);
    int u, v, w, s;
    cin >> n >> m >> q;
    for (int i = 0; i < m; ++i) {
        cin >> u >> v >> w;
        edge[u].emplace_back(w, v);
        ++indeg[v];
    }

    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) {
            for (int k = 0; k <= n; ++k) {
                dist[i][j][k] = INF;
            }
        }
    }

    toposort();

    for (int i = 1; i <= n; ++i) {
        dist[i][i][0] = 0;

        for (auto [w, v] : edge[i]) {
            dist[id[i]][id[v]][0] = min(w, dist[id[i]][id[v]][0]);
        }
    }

    floyd();

    /*for (int i = 1; i <= n; ++i) {
        cout << id[i] << " \n"[i == n];
    }*/

    int ans = 0;
    for (int i = 0; i < q; ++i) {
        cin >> u >> v >> s;
        u = (u + ans) % n + 1;
        v = (v + ans) % n + 1;
        s = (s + ans) % n + 1;
```

```
        //cout << u << " " << v << " " << s << " " << endl;

        ans = shortestPath(id[u], id[v], id[s]);

        if (ans == INF) ans = 0;
        cout << ans << endl;
    }

    return 0;
}
```