E4-H题

题目描述

在这循环往复的季节里，
对你的倾诉仍未说出口。
「我，任何改变都没有吗？」

给定 n 个点 m 条边的**带权有向图** G。G 上的点依次标号 1,2,…,n。第 i 条边 $u_i \to v_i$ 的起点为 u_i，终点为 v_i，边权为 w_i。

对于 G 上的环 $p_1 \to p_2 \to \cdots \to p_k \to p_1$，定义环的权值为环上所有边的边权平均值。

请你求出 G 上环的权值的最小值。如果 G 上没有环，则认为答案为 0。

输入

本题测试点包含多组数据。

第一行，一个正整数 T（1≤T≤10），表示数据组数。

对于每组数据：

第一行，两个正整数 n,m（$1 \le n \le 10^3$，$1 \le m \le 10^3$），表示 G 的点数与边数。

接下来 m 行，每行三个正整数 u_i, v_i, w_i（$1 \le u_i \le n$，$1 \le v_i \le n$，$1 \le w_i \le 10^6$，$u_i \neq v_i$），表示有向边的起点、终点与边权。

输出

对于每组数据：

输出一行，一个实数，表示答案。答案保留小数点后 4 位。

输入样例

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

输出样例

```
1.0000
0.0000
```

解题思路：二分法+SPFA判断负环

解题步骤

对于带权有向图G：

1. 初始时，以0为左边界 L，边权最大值为右边界 R；
2. 求二者中点$m = \frac{L+R}{2}$；
3. 将图G每条边的权值都减去m，得到图G'；
4. 判断图G'中是否存在负环（使用SPFA），更新 L 或 R：
    - 如果存在负环，则说明m偏大，$R = m$；
    - 如果不存在负环，则m偏小，$L = m$；
5. 恢复图G，即将图G'每条边的权值都加上第一步中得到的m；
6. 重复2~5步，随着二分次数的增加将不断逼近需要的答案，直到$R - L < EPS$。

EPS的选取

要求保留4位小数，故二分时的精度应该保留到5位，以保证答案的准确性；

本题可以取 $EPS = 1e - 5$；

SPFA判断负环

SPFA的过程

每次都拿一个点到起点的距离来松弛其他的点到起点的距离

SPFA判负环

负环是一个边权值和等于负数的环，如果SPFA遇到了负环，就会一直松弛下去，因为每次出现的负数都可以让目前最短的边变得更短。
所以可以根据这一点，得到如下步骤：

1. 开一个数组用来记录这是这一条链上第几个入队的数；
2. 每次松弛的时候都把到达的点入队的数标为前面这个点入队的次数+1（因为这是一个顺序的过程）；
3. 如果出现了负环就会一直松弛下去，然后这个负环上的点入队的数就会不断变大；
4. 当某个点的入队数大于n，那证明存在负环。

    原因：考虑极端情况，n个数连成一个点，那么入队数最大才只能是n，所以只要某个点的入队数大于n，那说明遇到了负环，将会一直松弛下去。

参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int SIZEN = 1e3 + 5;

struct Edge {
    int s, t;
    double w;
};

int n;
vector<Edge> edges;  //存边
vector<int> G[SIZEN];  // 存图
bool sign[SIZEN];  // 标记是否在队内
double dis[SIZEN];  // 记录点到起点的距离
int cot[SIZEN];  // 记录入队次序

void initSPFA() {
    edges.clear();
    for (int i = 0; i < n; ++i)
        G[i].clear();
}

void addEdge(int s, int t, double w) {
    edges.push_back((Edge) {s, t, w});
    int id = edges.size() - 1;
    G[s].push_back(id);
}

bool hasNegativeCycle() {
    queue<int> q;

    memset(sign, false, sizeof(sign));
    memset(cot, 0, sizeof(cot));
    for (int i = 0; i < n; ++i) {
        dis[i] = 0;
        q.push(i);
    }
    sign[0] = true;

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        sign[u] = false;
        for (int i = 0; i < G[u].size(); ++i) {
            Edge e = edges[G[u][i]];
            if (dis[e.t] > dis[u] + e.w) {
                dis[e.t] = dis[u] + e.w;
                if (!sign[e.t]) {
                    q.push(e.t);
                    sign[e.t] = true;
                    if (++cot[e.t] > n) // 入队数大于n，存在负环
                        return true;
```

```
                }
            }
        }
    }
    return false;
}


bool judge(double x) {
    for (int i = 0; i < edges.size(); ++i)
        edges[i].w -= x;
    bool flag = hasNegativeCycle();
    for (int i = 0; i < edges.size(); ++i)
        edges[i].w += x;
    return flag;
}


int main() {
    int tt, m, x, y;
    cin >> tt;
    while (tt--) {
        scanf("%d %d", &n, &m);
        initSPFA();
        double z, L = 0, R = -1;
        for (int i = 1; i <= m; i++) {
            scanf("%d %d %lf", &x, &y, &z);
            x--, y--;
            addEdge(x, y, z);
            R = max(R, z);
        }
        if (!judge(R + 1)) R = 0.0; //判断图中是否存在环
        else {
            // 二分法
            while (R - L > 1e-5) {
                double mm = (L + R) / 2;
                if (judge(mm)) R = mm;
                else L = mm;
            }
        }
        printf("%.4lf\n", R);
    }
    return 0;
}
```