题目描述

“又是早八...好痛苦 T_T”

Gahow 常常拖到最后一刻才从床上起来赶去上早八，他在路上不得不加快移动速度，因此他会在路上不断选择走一小段或者跑一小段。但是他的体力比较有限，所以他不能连续在两条路上进行奔跑。

现在我们形式化地将 Gahow 赶去上早八的路途抽象为一个**无向图**，其中有 n 个点标号为 $1, 2, \cdots, n$。Gahow 将从点 1 出发前往点 n。若两点间有一条距离为 s 的路径，则 Gahow 走过这条路所需的时间恰为 s，跑过这条路所需的时间为 $\lfloor \frac{s}{2} \rfloor$。

Gahow 担心上课要迟到了，想要以最短的时间到达教室，请你求出到达教室（点 n）所需的最小时间。若不存在一条路径可从点 1 到达点 n，那么 Gahow 将有充足的理由向老师请假，他将非常高兴，请你输出 `wow`。

注意：Gahow 不能连续在两条路径中选择奔跑。测试数据保证图中不存在重边和自环。

这个问题在最短路的基础上给出了新的条件：可以跑过一条路，不能连续跑两次。

考虑每个顶点（状态），对于每个邻接点v

如果上一条路不是跑过来的，我们可以选择走过去，也可以选择跑过去

否则，我们只能走过去

于是，我们可以把上一条边是否跑过加入状态里面，根据状态的转移方式建立一张新的图，将每个顶点u分为u_0（上一条边跑了）和u_1（上一条没跑）

对于原来的每条边(u, v, s)，在新图中加入的边如下

$(u_0, v_1, \lfloor \frac{s}{2} \rfloor)$

$(v_0, u_1, \lfloor \frac{s}{2} \rfloor)$（选择跑）

(u_0, v_0, s)

(v_0, u_0, s)（选择走）

(u_1, v_0, s)

(v_1, u_0, s)（只能选择走）

根据问题限制（不能连续跑两次），u_1与v_1之间无边

在新图中u_0到v_0或v_1的最短路就是问题所求，可以使用Dijkstra算法来计算。

代码：

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

using namespace std;

int n, m;
int t;

typedef long long ll;
typedef pair<int, ll> pill;
typedef pair<ll, int> plli;

const ll INF = 0x3f3f3f3f3f3f3f3f;

vector<vector<pill>> edge;
vector<ll> dist;

void dijkstra(int s) {
    dist.assign(n + 2, INF);
```

```
    dist[s] = 0;
    priority_queue<plli, vector<plli>, greater<plli>> pq;
    pq.emplace(0, s);

    while (!pq.empty()) {
        plli e = pq.top();
        ll d = e.first;
        int u = e.second;
        pq.pop();
        if (d > dist[u]) continue;

        for (pill e : edge[u]) {
            int v = e.first;
            ll w = e.second;
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.emplace(dist[v], v);
            }
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    cin >> t;
    while (t--) {
        cin >> n >> m;

        n *= 2;
        edge.assign(n + 2, vector<pill>());
        int u, v, w;
        for (int i = 0; i < m; ++i) {
            cin >> u >> v >> w;
            u <<= 1, v <<= 1;
            edge[u].emplace_back(v, w);
            edge[u].emplace_back(v + 1, w >> 1);
            edge[v].emplace_back(u, w);
            edge[v].emplace_back(u + 1, w >> 1);
            edge[u + 1].emplace_back(v, w);
            edge[v + 1].emplace_back(u, w);
        }

        dijkstra(2);

        ll ans = min(dist[n], dist[n + 1]);
        if (ans == INF) {
            cout << "wow" << endl;
        } else {
            cout << ans << endl;
        }
    }
    return 0;
}
```