골목 대장 호석 - 효율성 2 풀이

> 소싯적 호석이는 골목 대장의 삶을 살았다. 호석이가 살던 마을은 N 개의 교차로와 M 개의 골목이 있었다. 교차로의 번호는 1번부터 N 번까지로 표현한다. 골목은 서로 다른 두 교차로를 양방향으로 이어주며 임의의 두 교차로를 잇는 골목은 최대 한 개만 존재한다. 분신술을 쓰는 호석이는 모든 골목에 자신의 분신을 두었고, 골목마다 통과하는 사람에게 수금할 것이다. 수금하는 요금은 골목마다 다를 수 있다. 당신은 A 번 교차로에서 B 번 교차로까지 C 원을 가지고 가려고 한다. 호석이의 횡포를 보며 짜증은 나지만, 분신술을 이겨낼 방법이 없어서 돈을 내고 가려고 한다. 하지만 이왕 지나갈 거면, 최소한의 수치심을 받고 싶다. 당신이 받는 수치심은 경로 상에서 가장 많이 낸 돈에 비례하기 때문에, 결국 갈 수 있는 다양한 방법들 중에서 최소한의 수치심을 받으려고 한다. 즉, 한 골목에서 내야 하는 최대 요금을 최소화하는 것이다. 예를 들어, 위의 그림과 같이 5개의 교차로와 5개의 골목이 있으며, 당신이 1번 교차로에서 3번 교차로로 가고 싶은 상황이라고 하자. 만약 10원을 들고 출발한다면 2가지 경로로 갈 수 있다. 1번 -> 2번 -> 3번 교차로로 이동하게 되면 총 10원이 필요하고 이 과정에서 최대 수금액을 5원이었고, 1번 -> 4번 -> 5번 -> 3번 교차로로 이동하게 되면 총 8원이 필요하며 최대 수금액은 6원이 된다. 최소한의 수치심을 얻는 경로는 최대 수금액이 5인 경로이다. 하지만 만약 8원밖에 없다면, 전자의 경로는 갈 수 없기 때문에 최대 수금액이 6원인 경로로 가야 하는 것이 최선이다. 당신은 앞선 예제를 통해서, 수치심을 줄이고 싶을 수록 같거나 더 많은 돈이 필요하고, 수치심을 더 받는 것을 감수하면 같거나 더 적은 돈이 필요하게 된다는 것을 알게 되었다. 마을의 지도와 골목마다 존재하는 호석이가 수금하는 금액을 안다면, 당신이 한 골목에서 내야하는 최대 요금의 최솟값을 계산하자. 만약 지금 가진 돈으로는 절대로 목표 지점을 갈 수 없다면 -1을 출력하라.

그래프 G에서 정점 A에서 정점 B로 가는 최단 경로 중 최대 가중치를 최소화하는 문제입니다. Widest Path 문제로 알려져 있습니다.

그래프 G의 부분 그래프 G_w을 G의 모든 정점과 간선의 가중치가 w 이상인 간선으로 구성된 그래프라고 정의합니다. G_w에서 A에서 B로 가는 최단 거리가 C 이하인지 여부를 데이크스트라 알고리즘으로 판정할 수 있습니다.

$f(x) = G_x$ 에서 A에서 B로 가는 최단 거리가 C 이하인지 여부를 반환하는 함수로 정의합니다. 문제의 정답은 $f(1), f(2), \cdots, f(10^9)$ 중에서 $f(x)$가 참인 가장 작은 x입니다.

혹은 간선의 가중치를 작은 순서로 탐색하는 데이크스트라의 변형을 통해 이분 탐색 없이도 해결할 수 있습니다. 위키 백과 참고

소스 코드

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

using namespace std;

int main() {
    FAST();

    int n, m, a, b;
    long long c;
    cin >> n >> m >> a >> b >> c;

    vector<vector<pair<int, int>>> adj(n + 1);
    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
```

```
        adj[u].emplace_back(v, w);
        adj[v].emplace_back(u, w);
    }

    auto dijkstra = [&](int x) {
        priority_queue<pair<long long, int>, vector<pair<long long, int>>, greater<pair<long long, int>>>
pq;
        vector<long long> d(n + 1, 1e18);
        pq.emplace(0, a);
        d[a] = 0;

        while (!pq.empty()) {
            auto [cost, here] = pq.top();
            pq.pop();

            if (d[here] < cost) continue;

            for (auto [there, weight] : adj[here]) {
                if (weight > x) continue;

                long long there_cost = cost + weight;
                if (d[there] > there_cost) {
                    d[there] = there_cost;
                    pq.emplace(there_cost, there);
                }
```

```
            }
        }

        return d[b] <= c;
        };

    int lo = 0, hi = 1e9 + 1;
    while (lo + 1 < hi) {
        int mid = (lo + hi) / 2;
        if (dijkstra(mid)) {
            hi = mid;
        } else {
            lo = mid;
        }
    }

    cout << (hi == 1e9 + 1 ? -1 : hi) << '\n';
}
```