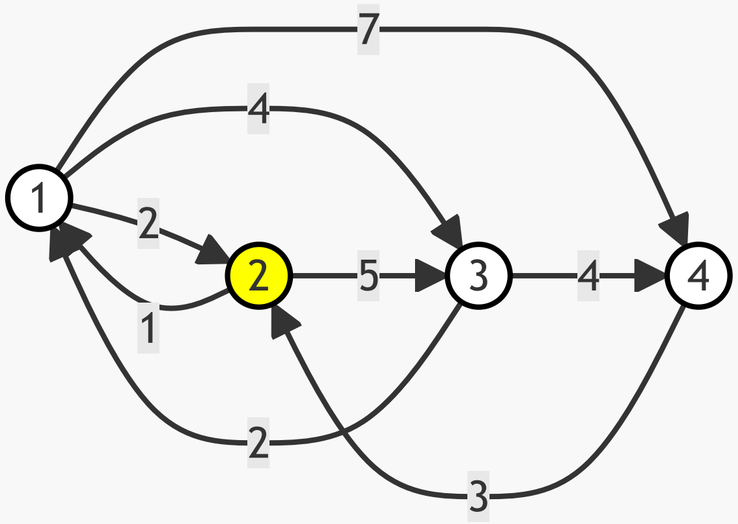
파티 풀이

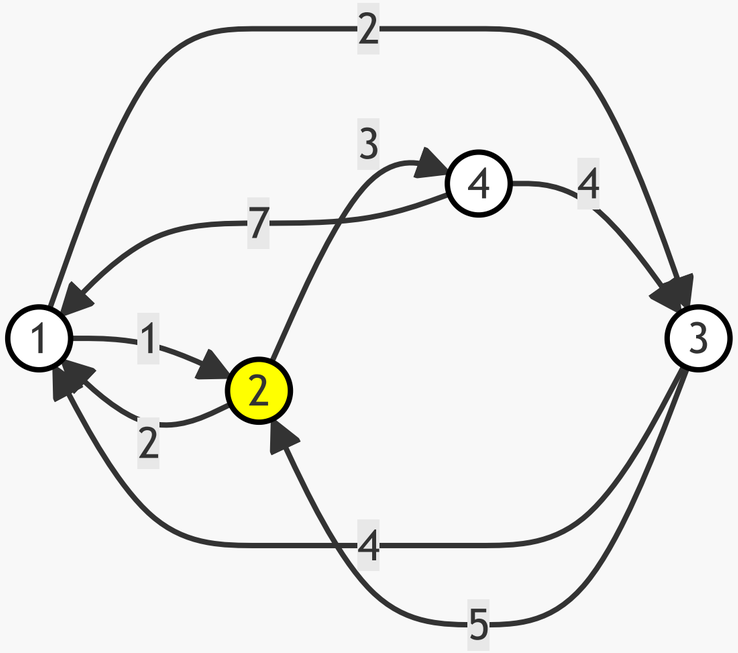
> N개의 숫자로 구분된 각각의 마을에 한 명의 학생이 살고 있다. 어느 날 이 N명의 학생이 X (1 ≤ X ≤ N)번 마을에 모여서 파티를 벌이기로 했다. 이 마을 사이에는 총 M개의 단방향 도로들이 있고 i번째 길을 지나는데 Ti(1 ≤ Ti ≤ 100)의 시간을 소비한다. 각각의 학생들은 파티에 참석하기 위해 걸어가서 다시 그들의 마을로 돌아와야 한다. 하지만 이 학생들은 워낙 게을러서 최단 시간에 오고 가기를 원한다. 이 도로들은 단방향이기 때문에 아마 그들이 오고 가는 길이 다를지도 모른다. N명의 학생들 중 오고 가는데 가장 많은 시간을 소비하는 학생은 누구일지 구하여라.



방향 그래프 G가 주어지고, $\text{dist}(x, y)$를 정점 x와 y의 최단 거리라고 할 때 $\max_{1 \le y \le N}(\text{dist}(y, x) + \text{dist}(x, y))$를 구하는 문제입니다.

데이크스트라 알고리즘은 한 정점에서 다른 모든 정점까지의 최단 거리를 구하는 알고리즘이기 때문에 $\text{dist}(x, y)$는 x 정점에서 시작하는 데이크스트라 한 번으로 구할 수 있습니다.

$\text{dist}(y, x)$를 구하기 위해 $y = 1, 2, \cdots, N$에 대해서 모두 데이크스트라 알고리즘을 수행할 수 있습니다. 정점의 개수 $N = 1,000$이고 데이크스트라 알고리즘의 시간 복잡도는 $\mathcal{O}(N \log N)$이므로 이 방법으로 문제를 해결할 수 있습니다.



혹은 모든 간선의 방향을 뒤집은 그래프 G^T를 만들어서 데이크스트라 알고리즘을 한 번만 수행할 수 있습니다. 방향 그래프 G에서 u와 v의 최단 경로는 그래프 G^T에서 v와 u의 최단 경로와 같기 때문입니다.

소스 코드

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

using namespace std;

int main() {
    int n, m, x;
    cin >> n >> m >> x;

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

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

    auto dijkstra = [&](int s, vector<vector<pair<int, int>>> adj) {
```

```
        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
        vector<int> d(n + 1, 1e9);

        pq.emplace(0, s);
        d[s] = 0;

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

            if (d[here] < cost) continue;

            for (auto [there, weight] : adj[here]) {
                auto there_cost = cost + weight;
                if (d[there] > there_cost) {
                    d[there] = there_cost;
                    pq.emplace(there_cost, there);
                }
            }
        }

        return d;
        };

    auto d = dijkstra(x, adj), d_t = dijkstra(x, adj_t);
```

```
    int max = 0;
    for (int i = 1; i <= n; i++) {
        max = ::max(max, d[i] + d_t[i]);
    }

    cout << max << '\n';
}
```