백준 5214번 - 환승

시간제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2초	256MB	7552	1928	1316	26.320%

문제

아주 먼 미래에 사람들이 가장 많이 사용하는 대중교통은 하이퍼튜브이다. 하이퍼튜브 하나는 역 K개를 서로 연결한다. 1번역에서 N번역으로 가는데 방문하는 최소 역의 수는 몇 개일까?

입력

첫째 줄에 역의 수 N과 한 하이퍼튜브가 서로 연결하는 역의 개수 K, 하이퍼튜브의 개수 M이 주어진다. (1 ≤ N ≤ 100,000, 1 ≤ K, M ≤ 1000)

다음 M개 줄에는 하이퍼튜브의 정보가 한 줄에 하나씩 주어진다. 총 K개 숫자가 주어지며, 이 숫자는 그 하이퍼튜브가 서로 연결하는 역의 번호이다.

출력

첫째 줄에 1번역에서 N번역으로 가는데 방문하는 역의 개수의 최솟값을 출력한다. 만약, 갈 수 없다면 -1을 출력한다.

예제 입력1

```
9 3 5
1 2 3
1 4 5
3 6 7
5 6 7
6 8 9
```

예제 출력1

```
4
```

출처

출처

알고리즘 분류

- 그래프 이론
- 그래프 탐색
- 너비 우선 탐색

접근 방법

초기 접근(MLE)

하나의 하이퍼튜브에 포함된 모든 노드들을 mesh-topology로 재구성하는 방법은 어떨까?

최악의 경우, N개의 노드를 Fully-connected-mesh topology로 구성하는 방법은 N(N-1)/2 의 공간 복잡도가 필요하므로 256MB내에는 해결이 불가능하다는 것을 알 수 있다.

개선책

문제의 요구 사항에 따라, 하나의 노드는 반드시 1개 이상의 Hypertube에 연결되는 것을 확인할 수 있다. 또한 하나의 노드는 다른 노드와 직접적으로 연결되지 못한다. 하나의 노드가 다른 노드와 연결되기 위해서는 반드시 1개 이상의 hypertube를 경유하여 도달해야함을 알 수 있다.

Hypertube를 노드들의 이동을 위한 중간 노드로 생각하여 bus-topology를 구성하면 어떨까?

이 경우에, Hypertube자체를 노드로 취급하여 인접리스트를 구성한다면 그래프의 노드는 최악의 경우 200,000개가 될 것이고 연결관계는 최대 1,000개 이므로 메모리 초과 없이 문제를 해결할 수 있다.

주의해야할점은, hypertube 자체는 문제 해결을 위해 임의로 도입한 노드이므로 `방문하는 역의 개수`에 포함 시키면 안된다는 것이다.

이 조건을 고려하기 위하여 Queue에는 `현재 노드가 hypertube인지` 파악할 수 있는 구분자가 필요하다.

현재 노드가 hypertube라면, 다음으로 방문할 노드가 station임이 보장되므로 이 경우에만 +1을 해준다음 최단 경로를 갱신한다.

소스코드

```cpp
#define FASTIO cin.tie(0)->sync_with_stdio(false), cout.tie(0)
////////////////////////////////////////////////////////////////
#include <bits/stdc++.h>
using namespace std;
int N, K, M;
int ans[100'001], tmp[100'001];
vector<int> adjHypertube[100'001];
vector<int> adjNodes[100'001];
int main(void){
    FASTIO;
////////////////////////////////////////////////////////////////
    // mesh topology를 구성하는 것은 메모리 초과 (최악의 경우, 10^12에 근사)
    cin >> N >> K >> M;
    // adjHypertube[i] := i번 역에서 연결되어 있는 하이퍼튜브의 vector
    // adjNodes[i] := i번 하이퍼튜브에서 갈 수 있는 정점들의 집합

    for (int i = 1; i <= M; i++) {
        for (int j = 0; j < K; j++) {
            int node;
            cin >> node;
```

```
            adjNodes[i].push_back(node);
            adjHypertube[node].push_back(i);
        }
    }

    // {passedBy, 현재 노드, 현재 노드가 hyperTube인가?}
    priority_queue<tuple<int, int, bool>, vector<tuple<int,int,bool>>, greater<>>
PQ;
    fill(ans, ans + 100'001, 1e9);
    fill(tmp, tmp + 100'001, 1e9);
    PQ.push({1, 1, false});
    ans[1] = 1;
    while (!PQ.empty()) {
        auto[passedBy, cur, isHypertube] = PQ.top(); PQ.pop();
        const vector<int>* adj;
        if (isHypertube) {
            adj = adjNodes;
        } else {
            adj = adjHypertube;
        }
        for (auto next: adj[cur]) {
            bool nxtIsHypertube = !isHypertube;
            if (!nxtIsHypertube && ans[next] <= passedBy + 1)
                continue;
            if (nxtIsHypertube && tmp[next] <= passedBy)
                continue;
            if (!nxtIsHypertube) {
                ans[next] = passedBy + 1;
            } else {
                tmp[next] = passedBy;
            }
            PQ.push({passedBy + !nxtIsHypertube, next, nxtIsHypertube});
        }
    }
    if (ans[N] == 1e9) ans[N] = -1;
    cout << ans[N];
    return 0;
}
```