환승

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.

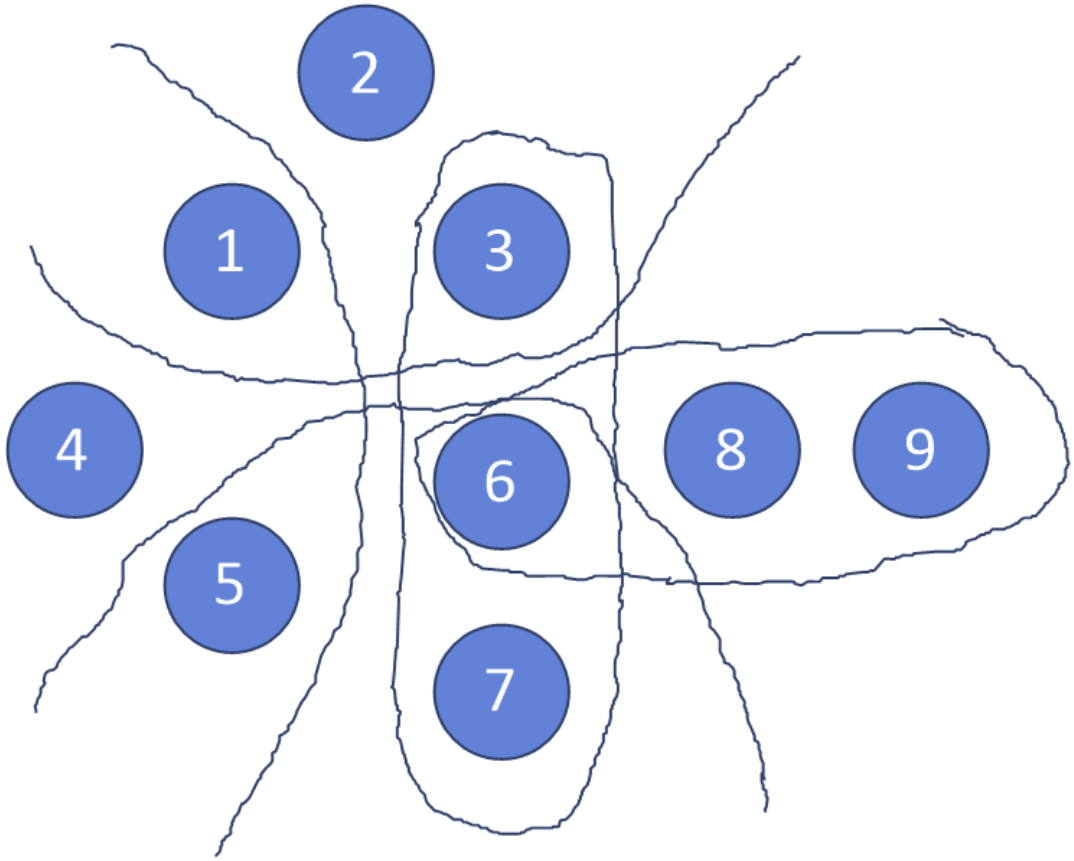
예제 입력 1 복사

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

예제 출력 1 복사

```
4
```

개념적으로 그린 조형도



첫 아이디어

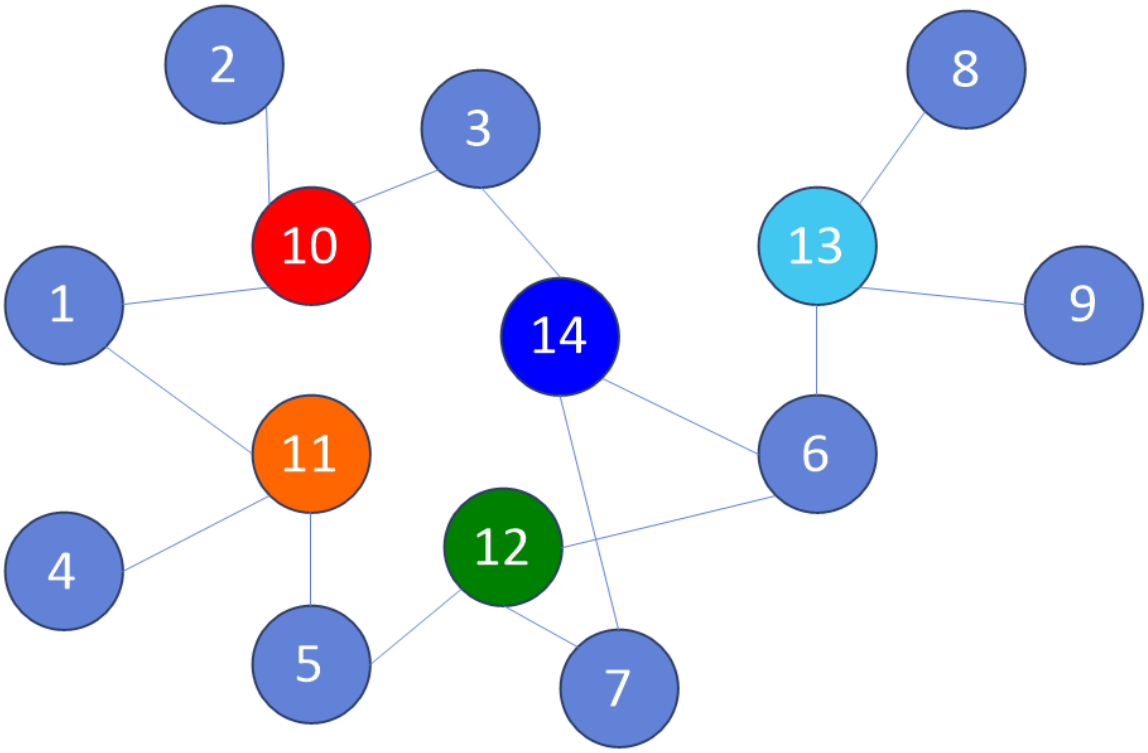
`Union-Find` 를 활용하여 시작점의 헤드와 도착점의 헤드가 일치할 때 까지 횟수를 세볼까?

`=>` `1` `-` `2` `-` `3` 한 그룹의 헤드가 한개로 특정 되지 않고

헤드를 바꾸게 된다면, 전체를 순회하며 바꿔줘야 하기 때문에 시간이 오래걸림.

하이퍼링크 `1`개를 하나의 노드로 변경하여

번호 `-` 하이퍼링크 `-` 번호 형태로 묶어 환승역 형태로 표현



그래프가 평범한 구조로 변경되었기 때문에, 단순한 BFS 형태로

시작점에서 시작하여 목표를 찾을 때 까지 탐색을 하면 된다.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
dist	1													
visit	T	F	F	F	F	F	F	F	F	F	F	F	F	F

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
dist	1									2	2			
visit	T	F	F	F	F	F	F	F	F	T	T	F	F	F

"

"

"

"

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
dist	1	3	3	3	3	5	5	7	7	2	2	4	6	4
visit	T	T	T	T	T	T	T	T	T	T	T	T	T	T

결론적으로 최종 답은 도착 할 때 까지 거쳐서 온 환승역의 수를 합하면 된다.

따라서, 환승역의 값을 `N + 1` 부터 설정해 주었으므로

BFS 과정에서 N보다 큰 경우 거리 값을 `+ 1` 해주는 형태로 최종 답을 도출.

코드

```java
import java.io.*;
import java.util.*;

public class Main {

  private static int stoi(String s) {
    return Integer.parseInt(s);
  }

  static List<List<Integer>> list;

  public static void main(String[] args) throws Exception {

    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    StringTokenizer str = new StringTokenizer(br.readLine());

    list = new ArrayList<>();

    int N = stoi(str.nextToken());
    int K = stoi(str.nextToken());
    int M = stoi(str.nextToken());
```

```
    int[] dist = new int[N + M + 1];
    boolean[] visit = new boolean[N + M + 1];

    for (int i = 1; i <= N + M + 1; i++) {
      list.add(new ArrayList<Integer>());
    }

    for (int i = N + 1; i <= N + M; i++) {
      str = new StringTokenizer(br.readLine());

      for (int j = 0; j < K; j++) {

        int X = stoi(str.nextToken());

        list.get(i).add(X);
        list.get(X).add(i);
      }
    }

    Queue<Integer> queue = new LinkedList<Integer>();

    queue.add(1);
    visit[1] = true;
       dist[1] = 1;

    while (!queue.isEmpty()) {
      int cur = queue.poll();

      if (cur == N) {
        break;
      }
      for (int node : list.get(cur)) {

        if (!visit[node]) {
          visit[node] = true;
          queue.add(node);
                    if(node > N) {
                        dist[node] = dist[cur] + 1;
                    }
          else {
                        dist[node] = dist[cur];
                    }
        }
      }
    }
    if (dist[N] == 0 && N != 1) {
      System.out.println(-1);
    } else {
      System.out.println(dist[N]);
    }
  }
}
```