2023-여름방학-알고리즘

학번 : 2019775054

이름 : 전상훈

백준 1939번 : 중량제한

중량제한

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	128 MB	32618	8867	5428	25.603%

문제

N(2 ≤ N ≤ 10,000)개의 섬으로 이루어진 나라가 있다. 이들 중 몇 개의 섬 사이에는 다리가 설치되어 있어서 차들이 다닐 수 있다.

영식 중공업에서는 두 개의 섬에 공장을 세워 두고 물품을 생산하는 일을 하고 있다. 물품을 생산하다 보면 공장에서 다른 공장으로 생산 중이던 물품을 수송해야 할 일이 생기곤 한다. 그런데 각각의 다리마다 중량제한이 있기 때문에 무턱대고 물품을 옮길 순 없다. 만약 중량제한을 초과하는 양의 물품이 다리를 지나게 되면 다리가 무너지게 된다.

한 번의 이동에서 옮길 수 있는 물품들의 중량의 최댓값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N, M(1 ≤ M ≤ 100,000)이 주어진다. 다음 M개의 줄에는 다리에 대한 정보를 나타내는 세 정수 A, B(1 ≤ A, B ≤ N), C(1 ≤ C ≤ 1,000,000,000)가 주어진다. 이는 A번 섬과 B번 섬 사이에 중량제한이 C인 다리가 존재한다는 의미이다. 서로 같은 두 섬 사이에 여러 개의 다리가 있을 수도 있으며, 모든 다리는 양방향이다. 마지막 줄에는 공장이 위치해 있는 섬의 번호를 나타내는 서로 다른 두 정수가 주어진다. 공장이 있는 두 섬을 연결하는 경로는 항상 존재하는 데이터만 입력으로 주어진다.

백준 1939번 : 중량제한

예제 입력 1 복사

```
3 3
1 2 2
3 1 3
2 3 2
1 3
```

예제 출력 1 복사

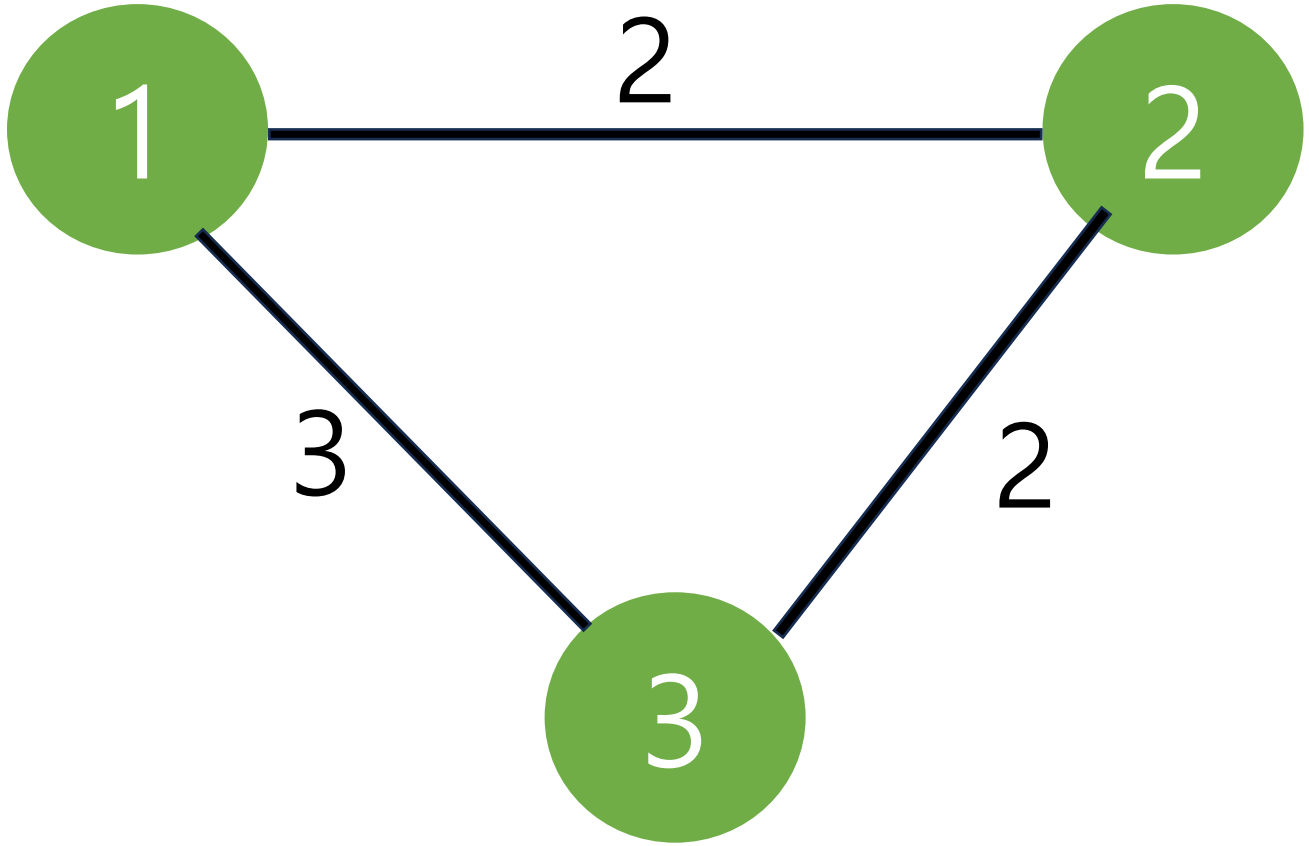
```
3
```

백준 1939번 : 중량제한

문제 : 섬이 있고 그 섬들 사이에 가중치가 있을 때
시작 섬부터 목표섬까지 옮길수있는 물품들의 중량의 최대값을 구하는 문제다

유형 : 한노드부터 목표 노드까지의 경로가 있는지부터 검사 해야하기때문에 bfs를 사용한다
가중치의 범위가 1 ~ 1,000,000,000 이다 빠르게 찾기위해 이분탐색을 사용하여 중량의 최대값을 구한다.
bfs진행중 이진탐색에서 사용된 mid(가중치값의미) 보다 큰지 확인하는 조건문을 넣는다.

백준 1939번 : 중량제한



백준 1939번 : 중량제한

값 입력받기

```java
public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    StringTokenizer st = new StringTokenizer(br.readLine());

    int N = Integer.parseInt(st.nextToken());
    int M = Integer.parseInt(st.nextToken());
    ArrayList<ArrayList<Node>> graph = new ArrayList<>(N + 1);

    for (int i = 0; i <= N; i++) {
        graph.add(new ArrayList<>());
    }

    int max = 0;
    for (int i = 0; i < M; i++) {
        st = new StringTokenizer(br.readLine());
        int a = Integer.parseInt(st.nextToken());
        int b = Integer.parseInt(st.nextToken());
        int c = Integer.parseInt(st.nextToken());

        graph.get(a).add(new Node(b, c));
        graph.get(b).add(new Node(a, c));
        max = Math.max(max, c);
    }
```

Node 클래스

```java
class Node {
    int to, weight;

    Node(int to, int weight) {
        this.to = to;
        this.weight = weight;
    }
}
```

입력예제

```
3 3
1 2 2
3 1 3
2 3 2
```

graph.get(1).add(new Node(2, 2));
graph.get(2).add(new Node(1, 2));

백준 1939번 : 중량제한

이진탐색

```
st = new StringTokenizer(br.readLine());
int start = Integer.parseInt(st.nextToken());
int end = Integer.parseInt(st.nextToken());

int result = 0;
int low = 1, high = max;

while (low <= high) {
    int mid = (low + high) / 2;

    if (bfs(graph, N, start, end, mid)) {
        result = mid;
        low = mid + 1;
    } else {
        high = mid - 1;
    }
}

System.out.println(result);
```

start : 시작노드
end : 목적지노드
result : 중량제한의 최대값
low : 중량제한범위의 최소값
high : 중량제한범위의 최대값
mid : 중량제한범위의 중앙값

If : bfs의 return이 true일 경우 mid(중량제한)값으로
시작노드부터 목적지 노드까지 이동 가능한 경로가 있다.

else : mid(중량)값으로 시작노드부터 목적지노드까지
이동가능한 경로가 없다

백준 1939번 : 중량제한

bfs

```java
private static boolean bfs(ArrayList<ArrayList<Node>> graph, int N, int start, int end, int mid) {
    Queue<Integer> queue = new LinkedList<>();
    boolean[] visited = new boolean[N + 1];

    queue.offer(start);
    visited[start] = true;

    while (!queue.isEmpty()) {
        int cur = queue.poll();
        if (cur == end) return true;

        ArrayList<Node> nodeList = graph.get(cur);
        for (Node node : nodeList) {
            if (!visited[node.to] && node.weight >= mid) {
                visited[node.to] = true;
                queue.offer(node.to);
            }
        }
    }

    return false;
}
```

방문하지않은 노드를 방문할때
중량이 mid 보다 크면 방문한다.

백준 1939번 : 중량제한

입력예제
3 3
1 2 2
3 1 3
2 3 2

Start = 1
End = 3

Low = 1
High = 3
Mid = 2

bfs
큐 : [1]
방문 : [F, T, F, F]

while
Cur = 1
큐 : [2, 3]
방문 : [F, T, T, T]

Cur = 2
큐 : [3]

Cur = 3 // cur=end
Return true;
Restult = 2

Low = 3
High = 3
Mid = 3

bfs
큐 : [1]
방문 : [F, T, F, F]

while
Cur = 1
큐 : [3]
방문 : [F, T, F, T]

Cur = 3 // cur=end
Return true
Result = 3

Low = 4
High = 3
종료

백준 1939번 : 중량제한

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
import java.util.StringTokenizer;

class Node {
    int to, weight;

    Node(int to, int weight) {
        this.to = to;
        this.weight = weight;
    }
}

public class Main {

    private static boolean bfs(ArrayList<ArrayList<Node>> graph, int N, int start, int end, int mid) {
        Queue<Integer> queue = new LinkedList<>();
        boolean[] visited = new boolean[N + 1];

        queue.offer(start);
        visited[start] = true;

        while (!queue.isEmpty()) {
            int cur = queue.poll();
            if (cur == end) return true;

            ArrayList<Node> nodeList = graph.get(cur);
            for (Node node : nodeList) {
                if (!visited[node.to] && node.weight >= mid) {
                    visited[node.to] = true;
                    queue.offer(node.to);
                }
            }
        }

        return false;
    }

    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StringTokenizer st = new StringTokenizer(br.readLine());

        int N = Integer.parseInt(st.nextToken());
        int M = Integer.parseInt(st.nextToken());
        ArrayList<ArrayList<Node>> graph = new ArrayList<>(N + 1);

        for (int i = 0; i <= N; i++) {
            graph.add(new ArrayList<>());
        }

        int max = 0;
        for (int i = 0; i < M; i++) {
            st = new StringTokenizer(br.readLine());
            int a = Integer.parseInt(st.nextToken());
            int b = Integer.parseInt(st.nextToken());
            int c = Integer.parseInt(st.nextToken());

            graph.get(a).add(new Node(b, c));
            graph.get(b).add(new Node(a, c));
            max = Math.max(max, c);
        }

        st = new StringTokenizer(br.readLine());
        int start = Integer.parseInt(st.nextToken());
        int end = Integer.parseInt(st.nextToken());

        int result = 0;
        int low = 1, high = max;

        while (low <= high) {
            int mid = (low + high) / 2;

            if (bfs(graph, N, start, end, mid)) {
                result = mid;
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }

        System.out.println(result);
    }
}
```