BOJ 1504 : 특정한 최단 경로

beberiche · 약 2시간 전

1504 algolithms 다익스트라 백준

0

algolithms

1. BOJ 1593 : 문자해독
2. BOJ 11051 : 이항계수2
3. BOJ 2482 : 색상환
4. Programmers : 귤 고르기
5. BOJ 7569 : 토마토
6. BOJ 14500 : 테트로미오
7. **BOJ 1504 : 특정한 최단 경로**
8. Programmers : n^2 배열자르기

숨기기 7/8

문제

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	59179	14981	10112	24.565%

문제

방향성이 없는 그래프가 주어진다. 세준이는 1번 정점에서 N번 정점으로 최단 거리로 이동하려고 한다. 또한 세준이는 두 가지 조건을 만족하면서 이동하는 특정한 최단 경로를 구하고 싶은데, 그것은 바로 임의로 주어진 두 정점은 반드시 통과해야 한다는 것이다.

세준이는 한번 이동했던 정점은 물론, 한번 이동했던 간선도 다시 이동할 수 있다. 하지만 반드시 최단 경로로 이동해야 한다는 사실에 주의하라. 1번 정점에서 N번 정점으로 이동할 때, 주어진 두 정점을 반드시 거치면서 최단 경로로 이동하는 프로그램을 작성하시오.

입력

첫째 줄에 정점의 개수 N과 간선의 개수 E가 주어진다. ($2 \le N \le 800$, $0 \le E \le 200,000$) 둘째 줄부터 E개의 줄에 걸쳐서 세 개의 정수 a, b, c가 주어지는데, a번 정점에서 b번 정점까지 양방향 길이 존재하며, 그 거리가 c라는 뜻이다. ($1 \le c \le 1,000$) 다음 줄에는 반드시 거쳐야 하는 두 개의 서로 다른 정점 번호 v_1과 v_2가 주어진다. ($v_1 \neq v_2$, $v_1 \neq N$, $v_2 \neq 1$) 임의의 두 정점 u와 v사이에는 간선이 최대 1개 존재한다.

출력

첫째 줄에 두 개의 정점을 지나는 최단 경로의 길이를 출력한다. 그러한 경로가 없을 때에는 -1을 출력한다.

예제 입력 1 복사

```
4 6
1 2 3
2 3 3
3 4 1
1 3 5
2 4 5
1 4 4
2 3
```

예제 출력 1 복사

```
7
```

풀이과정

문제에도 나와있듯이, 최단경로 알고리즘을 활용하는 문제.
단, 기존의 최단경로와는 다르게 반드시 두개의 포인트를 지나는 최단 경로여야 한다.

시작점 부터, 특정 포인트 A,B 를 지나 정점까지 도달하는 최단 경로의 식은 다음과 같다.

> **루트 A** : (시작점 -> a) 의 최단경로 + (a -> b) 의 최단경로 + (b -> 정점) 의 최단경로
> **루트 B** : (시작점 -> b) 의 최단경로 + (b -> a) 의 최단경로 + (a -> 끝점) 의 최단경로
> **루트 A** 와 **루트 B** 가운데 더 짧은 경로

정답

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

public class Main {
    static class Node {
        // 진행하는 경로
        int next;
        // 비용
        int cost;

        public Node(int next, int cost) {
            this.next = next;
            this.cost = cost;
        }
```

```
    }

    static List<Node>[] map;
    static final int INF = (int) 1e9;
    static int N;
    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StringTokenizer st = new StringTokenizer(br.readLine());
        N = Integer.parseInt(st.nextToken());
        int E = Integer.parseInt(st.nextToken());

        map = new ArrayList[N + 1];

        for(int i=1; i<=N; i++) map[i] = new ArrayList<>();

        while(--E>=0) {
            st = new StringTokenizer(br.readLine());
            int prev = Integer.parseInt(st.nextToken());
            int next = Integer.parseInt(st.nextToken());
            int cost = Integer.parseInt(st.nextToken());

            map[prev].add(new Node(next, cost));
            map[next].add(new Node(prev, cost));
        }

        st = new StringTokenizer(br.readLine());
        int nodeA = Integer.parseInt(st.nextToken());
        int nodeB = Integer.parseInt(st.nextToken());



        int case1 = dijkstra(1, nodeA) + dijkstra(nodeA, nodeB) + dijkstra(nodeB, N);
        int case2 = dijkstra(1, nodeB) + dijkstra(nodeB, nodeA) + dijkstra(nodeA, N);

        if (case1 < 0 || case2 < 0 || case1 >= INF || case2 >= INF) System.out.println(-1)
        else System.out.println(Math.min(case1, case2));
    }

    private static int dijkstra(int st, int ed) {
        PriorityQueue<Node> pq = new PriorityQueue<>((n1, n2) -> n1.cost - n2.cost);
        boolean[] visited = new boolean[N+1];
        int[] dist = new int[N+1];
        Arrays.fill(dist, INF);

        dist[st] = 0;
        pq.add(new Node(st, 0));

        while (!pq.isEmpty()) {
            Node curr = pq.poll();
            if (visited[curr.next]) continue;
            visited[curr.next] = true;

            for (Node nNode : map[curr.next]) {
                if (dist[nNode.next] > nNode.cost + curr.cost) {
                    dist[nNode.next] = nNode.cost + curr.cost;
                    pq.add(new Node(nNode.next, dist[nNode.next]));
                }
            }
        }

        return dist[ed];
    }
```