도망자 원숭이

도망자 원숭이 성공 분류

☆

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2 초	128 MB	1327	406	235	27.844%

문제

동물원에서 막 탈출한 원숭이 한 마리가 세상구경을 하고 있다. 그러나 그는 곧 동물원 직원에게 쫓기는 신세가 되었다. 원숭이와 동물원 직원사이에 쫓고 쫓기는 추격전을 살펴보자.

원숭이가 사는 나라는 여러 개의 도시와 도시들을 연결하는 길들로 구성되어 있다. 각 길들은 두 개의 도시를 양방향으로 연결한다. 또한, 각 길은 지나갈 때마다 일정한 시간이 걸린다. 원숭이는 시작도시에서 탈출하여 도착도시까지 최대한 빠른 시간에 가야한다.

그런데 원숭이의 오랜 숙적 멍멍이가 이를 갈며 원숭이를 기다리고 있었다. 멍멍이는 원숭이가 도망가는 경로 중 시작점과 도착점을 포함한 도시 중 한 군데에서 원숭이를 괴롭히기로 계획했다. 각 도시마다 구조가 다르기 때문에 멍멍이가 원숭이를 괴롭힐 수 있는 시간이 정해져있다.

그래서 멍멍이는 원숭이가 도망가는 경로 상에 있는 모든 도시들 중에서 가장 오랜 시간동안 괴롭힐 수 있는 도시에서 괴롭히기로 계획했다. 원숭이는 멍멍이를 피할 수 없다. 피할 수 없다면 즐겨라! 시작도시와 도착도시가 주어졌을 때, 원숭이가 최대한 빨리 도망갈 수 있는 시간을 구하는 프로그램을 작성하시오.

예를 들어, A, B, C, D 4개의 도시가 있고 원숭이는 A에서 도망쳐서 D로 가려고 한다고 하자. 이때, A-B와 B-D 간의 도로의 통행시간은 각각 50 이고 A-C 와 C-D 간의 도로의 통행시간은 각각 70 이면 A-B-D 의 경로가 더 이익이다. (각각 100 과 140 의 시간이 걸린다.)

그러나, 네 도시에서 멍멍이가 원숭이를 괴롭힐 수 있는 시간이 10, 80, 20, 10 이라면 A-C-D 를 통해 가는 것이 시간을 더 줄일 수 있는 방법이다. (A-B-D 의 경우 100+80 = 180 의 시간이 걸리고, A-C-D 의 경우 140+20 = 160 의 시간이 걸린다.)

입력

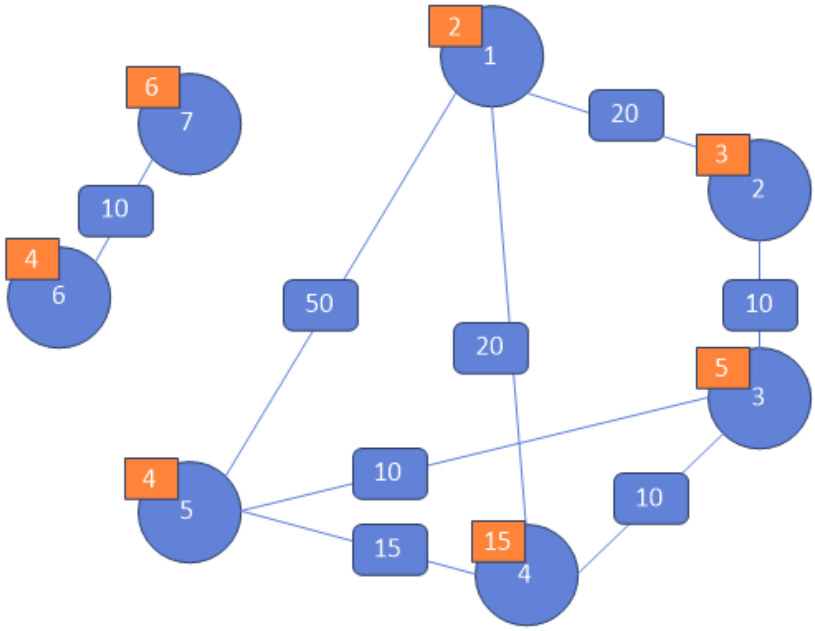
첫 번째 줄에는 도시의 개수 N (2 ≦ N ≦ 500) 과 도로의 개수 M (0 ≦ M ≦ 10,000), 그리고 질문의 개수 Q (0 ≦ Q ≦ 40,000) 가 주어진다.

그 다음 줄에, N개의 정수로 각 도시에서 멍멍이가 원숭이를 괴롭힐 수 있는 시간이 주어진다. 각 시간은 1이상 10,000이하의 정수이다. 그 후 M 줄에 각각 3개의 정수로, 해당 도로가 잇는 두 도시의 번호 a, b (1 <= a, b <= N) 와 해당 도로의 통행시간 d 가 주어진다. 통행시간은 1이상 10,000이하의 정수이다.

그 후 Q줄에 각각 2개의 정수로, 원숭이의 출발도시와 도착도시 S, T 가 주어진다.

출력

첫째 줄에 원숭이가 S 번 도시로부터 T 번 도시까지 도망가는 데 드는 최소시간을 출력한다. 만약 두 도시 간에 경로가 없을 경우, -1 을 출력한다.



A -> B 를 이동 할 때의 최단 거리를 구하는 문제

=> 플로이드 와샬, 다익스트라 알고리즘 이용

Try 1

각 노드를 지나칠 때 마다 원숭이의 방해 코스트가 증가한다고 생각

dp[i][j] = Math.min(dp[i][j] , dp[i][k] + dp[k][i] + dog[k]) 으로 계산

오류) 문제 상에서 멍멍이는 원숭이를 가장 오랜 시간 괴롭힐 수 있는

한 도시에서만 코스트를 더하게 한다.

Solution

최단 거리를 구하는 dp 와

방해 값이 포함된 dp 를 두개 사용하여 계산 해준다.

최단 거리 dp는 일반적인 플로이드 와샬로 구해주고,

방해 값이 포함한 dp는 이동한 경로 상에서 패널티 시간이 가장 큰 하나의 도시만 더해서 계산

이때, 순차적으로 거쳐가는 정점을 잡으면 안되고,

정점의 패널티가 적은 도시 순서대로 설정해줘야 됨.

방해 값이 포함된 dp
=> answer[i][j] 에는 [i] -> [j] 값 + Math.max(dog[i] , dog[j]) 값이 초기화

거쳐가는 k 정점에서 dog[k] 값이 dog[i], dog[j] 보다 크다면

갱신이 일어나게 되는데

적은 도시 순으로 순회를 하고 있기 때문에, k 정점을 지났는데 갱신이 된다면,

여태 껏 경로상에서 가장 큰 패널티를 가진 지점이 k가 된다.

코드

```
import java.io.*;
import java.util.Arrays;
import java.util.Comparator;
import java.util.StringTokenizer;

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

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

    int N = Integer.parseInt(str.nextToken());
    int M = Integer.parseInt(str.nextToken());
    int Q = Integer.parseInt(str.nextToken());
    final int INF = 987654321;

    int[][] dp = new int[N + 1][N + 1];
    int[][] answer = new int[N + 1][N + 1];
    int[] dog = new int[N + 1];
    Integer[] dogIdx = new Integer[N + 1];

    for(int i = 1; i <= N; i++) {
      for(int j = 1; j <= N; j++) {
        if(i == j) {
          dp[i][j] = 0;
          answer[i][j] = dog[i];
          continue;
        }
        dp[i][j] = INF;
        answer[i][j] = INF;
      }
    }

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

    for(int i = 1; i <= N; i++) {
      dog[i] = Integer.parseInt(str.nextToken());
      dogIdx[i] = i;
    }

    for(int i = 0; i < M; i++) {
      str = new StringTokenizer(br.readLine());

      int st = Integer.parseInt(str.nextToken());
      int ed = Integer.parseInt(str.nextToken());
      int cost = Integer.parseInt(str.nextToken());
```

```
      dp[st][ed] = cost;
      dp[ed][st] = cost;

      answer[st][ed] = cost + Math.max(dog[st], dog[ed]);
      answer[ed][st] = cost + Math.max(dog[ed], dog[st]);
    }

        Arrays.sort(dogIdx, 1, N + 1, new Comparator<Integer>() {

            @Override
            public int compare(Integer o1, Integer o2) {
                return dog[o1] - dog[o2];
            }

        });
    for(int k = 1; k <= N; k++) {

      int idx = dogIdx[k];
      for(int i = 1; i <= N; i++) {
        for(int j = 1; j <= N; j++) {
          dp[i][j] = Math.min(dp[i][j], dp[i][idx] + dp[idx][j]);
          answer[i][j] = Math.min(answer[i][j], dp[i][j] + Math.max(dog[i], Math.max(dog[idx], dog[j])));
        }
      }

    }
    for(int i = 0; i < Q; i++) {
      str = new StringTokenizer(br.readLine());

      int st = Integer.parseInt(str.nextToken());
      int ed = Integer.parseInt(str.nextToken());

      if(answer[st][ed] == INF) {
        System.out.println(-1);
        continue;
      }
      System.out.println(answer[st][ed]);
    }
  }
}
```