

BOJ 16947: 서울 지하철 2호선

문제 풀이과정 정답

통계 수정 삭제

BFS DFS algolithms 백준

algolithms

▼ 목록 보기

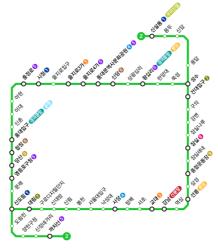
9/12 〈 →

서울 지하철 2호선

beberiche · 약 2시간 전

| 시간 제한 | 메모리 제한 | 제출 | 정답 | 맞힌 사람 | 정답 비율 |
|-------|--------|------|------|-------|---------|
| 2 초 | 512 MB | 5112 | 2483 | 1638 | 47.205% |
| | | | | | |

서울 지하철 2호선은 다음과 같이 생겼다.



지하철 2호선에는 51개의 역이 있고, 역과 역 사이를 연결하는 구간이 51개 있다. 즉, 정점이 51개이고, 양방향 간선이 51개인 그래프로 나타낼 수 있다. 2호선은 순환선 1개와 2개의 지 선으로 이루어져 있다. 한 역에서 출발해서 계속 가면 다시 출발한 역으로 들어올 수 있는 노선을 순환선이라고 한다. 지선은 순환선에 속하는 한 역에서 시작하는 트리 형태의 노선이다.

두 역(정점) 사이의 거리는 지나야 하는 구간(간선)의 개수이다. 역 A와 순환선 사이의 거리는 A와 순환선에 속하는 역 사이의 거리 중 최솟값이다.

지하철 2호선과 같은 형태의 노선도가 주어졌을 때, 각 역과 순환선 사이의 거리를 구해보자.

입력

첫째 줄에 역의 개수 N(3 ≤ N ≤ 3,000)이 주어진다. 돌째 줄부터 N개의 줄에는 역과 역을 연결하는 구간의 정보가 주어진다. 같은 구간이 여러 번 주어지는 경우는 없고, 역은 1번부터 N번까지 번호가 매겨져 있다. 임의의 두 역 사이에 경로가 항상 존재하는 노선만 입력으로 주어진다.

출력

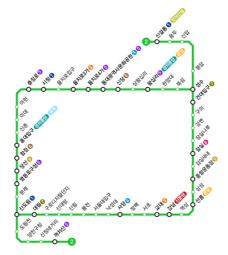
총 N개의 정수를 출력한다. 1번 역과 순환선 사이의 거리, 2번 역과 순환선 사이의 거리, ..., N번 역과 순환선 사이의 거리를 공백으로 구분해 출력한다.

문제

| 시간 제한 | 메모리 제한 | 제출 | 정답 | 맞힌 사람 | 정답 비율 |
|-------|--------|------|------|-------|---------|
| 2 초 | 512 MB | 5112 | 2483 | 1638 | 47.205% |

문제

서울 지하철 2호선은 다음과 같이 생겼다.



지하철 2호선에는 51개의 역이 있고, 역과 역 사이를 연결하는 구간이 51개 있다. 즉, 정점이 51개이고, 양방향 간선이 51개인 그래프로 나타낼 수 있다. 2호선은 순환선 1개와 2개의 지 선으로 이루어져 있다. 한 역에서 출발해서 계속 가면 다시 출발한 역으로 돌아올 수 있는 노선을 순환선이라고 한다. 지선은 순환선에 속하는 한 역에서 시작하는 트리 형태의 노선이다.

두 역(정점) 사이의 거리는 지나야 하는 구간(간선)의 개수이다. 역 A와 순환선 사이의 거리는 A와 순환선에 속하는 역 사이의 거리 중 최솟값이다.

지하철 2호선과 같은 형태의 노선도가 주어졌을 때, 각 역과 순환선 사이의 거리를 구해보자.

입력

첫째 줄에 역의 개수 N(3 ≤ N ≤ 3,000)이 주어진다. 둘째 졸부터 N개의 줄에는 역과 역을 연결하는 구간의 정보가 주어진다. 같은 구간이 여러 번 주어지는 경우는 없고, 역은 1번부터 N번까지 번호가 매겨져 있다. 임의의 두 역 사이에 경로가 항상 존재하는 노선만 입력으로 주어진다.

출력

총 N개의 정수를 출력한다. 1번 역과 순환선 사이의 거리, 2번 역과 순환선 사이의 거리, ..., N번 역과 순환선 사이의 거리를 공백으로 구분해 출력한다.

풀이과정

일단은 각 포인트에 대한 최단 경로를 탐색하는 문제.

사이클을 이루는 곳을 먼저 찾은 후, 해당 사이클 포인트마다 기준점을 찾는 문제.

나의 경우, 사이클을 찾는 것은 동일하나 이후 BFS 를 통해 풀이를 하였다.

(BFS의 경우, 너비 우선 탐색이기 때문에 가장 먼저 도달하는 곳이 곧 인접한 노드간의 최단 경로가 된다. BFS를 활용한 미로 문제를 풀어봤다면 금방 알 수 있다.)

사이클을 찾는 부분에서 생각보다 애를 먹었다. 나의 경우, 최소 depth 값과, 이중 flag 를 통해 확인하였다. 두개의 flag 중, 하나는 일반적인 visited 이고, 남은 하나는 그래서 현재 노드가 사이클 이 이루어졌는지 확인하는 용도이다. 사이클 의 경우, 현재 노드로 탐색할 수 있는 곳을 모두 탐색하고, 만약 자기 자신을 찾게 되는 경우, 사이클을 이룬다고 판단하도록 했다.

정답

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

public class Main {
    static List<Integer> adjList[];
    static boolean[] visited, isCycle;
    static int[] dist;
    static int N;
```

```
static Queue<Integer> q = new LinkedList<>();
   public static void main(String[] args) throws Exception {
       BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
       N = Integer.parseInt(br.readLine());
       adjList = new ArrayList[N + 1];
       for (int i = 1; i \le N; i++) {
           adjList[i] = new ArrayList<>();
       for (int i = 0; i < N; i++) {
           StringTokenizer st = new StringTokenizer(br.readLine());
           int curr = Integer.parseInt(st.nextToken());
           int next = Integer.parseInt(st.nextToken());
           adjList[curr].add(next);
           adjList[next].add(curr);
       }
       dist = new int[N + 1];
       isCycle = new boolean[N+1];
       // 순환선 찾기
       for(int i=1; i<=N; i++) {
           visited = new boolean[N + 1];
           findCycle(i, i,1);
       for(int i=1; i<=N; i++) {</pre>
           // 싸이클 아닌 애들 대표값에서 미리 분류
           if(!isCycle[i]) dist[i]= -1;
           else q.add(i);
       }
       // 찾은 순환선에서 떨어진 최소 루트 구하기
       // 다익스트라
       dijkstra();
       StringBuilder sb = new StringBuilder();
       for(int i=1; i<=N; i++) {
           sb.append(dist[i]).append(" ");
       }
       System.out.println(sb.toString());
   }
   private static void findCycle(int idx, int start ,int cnt) {
       visited[idx] = true;
       for (int next : adjList[idx]) {
           // 사이클인 경우
           // 1. 시작한 곳이 다시 나올것
           // 2. 최소 3개 이상의 노드로 구성될 것
                      // 2개의 경우에는 그냥 양방향으로 연결되어서 가는 경우도 있으므로 3개 이상이어야 함
           if(next == start && cnt >= 3) isCycle[next] = true;
           if(!visited[next]) findCycle(next, start,cnt+1);
       }
   }
   private static void dijkstra() {
       while(!q.isEmpty()) {
           int curr = q.poll();
           for(int next: adjList[curr]) {
               // 사이클이 아닌 노드라면
               if(dist[next] == -1) {
                   // BFS는 와이파이라고 했으니, 가장 먼저 도달하는 곳에서 +1 한 값이 최소 루트이겠지
                   dist[next] = dist[curr]+1;
                   q.add(next);
           }
      }
  }
}
```