



트리 트리오 중간값

☑ 다시 풀어보기	☑
🔗 링크	https://programmers.co.kr/learn/courses/30/lessons/68937
🕒 생성일	@2020년 12월 21일 오후 11:01
# 소요시간(분)	
☰ 유형	BFS 트리
☰ 출처	Programmers

목차

[목차](#)

[문제 설명](#)

[접근 방법](#)

[처음 접근 방법](#)

[풀이 로직](#)

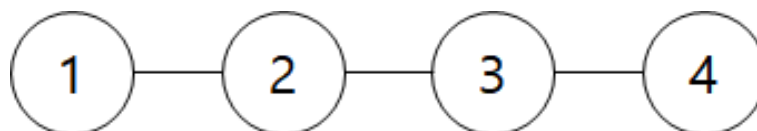
[상세 설명](#)

[소스 코드](#)

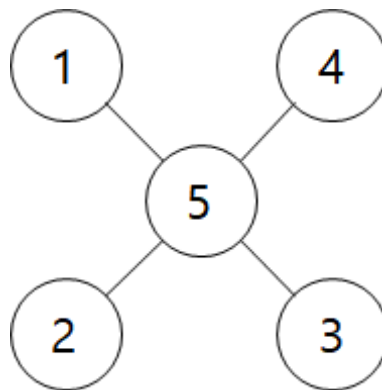
[마무리](#)

문제 설명

트리에서 나올 수 있는 모든 임의의 정점(a, b, c) 3개를 골라, 각 정점의 거리의 중간값 중 최대 값을 구하는 문제



a	b	c	a~b	b~c	c~a	f(a,b,c)
1	2	3	1	1	2	1
1	2	4	1	2	3	2
1	3	4	2	1	3	2
2	3	4	1	1	2	1



a	b	c	a~b	b~c	c~a	f(a,b,c)
1	5	3	1	1	2	1
1	2	3	2	2	2	2










접근 방법

처음 접근 방법



- 모든 정점들에 대해 3개를 선택하는 조합을 뽑아, 중간값의 최대값을 구한다 → n 의 범위가 $3 \leq n \leq 250,000$, 굉장히 비효율적..
- 그 외의 방법은 생각해내지 못함

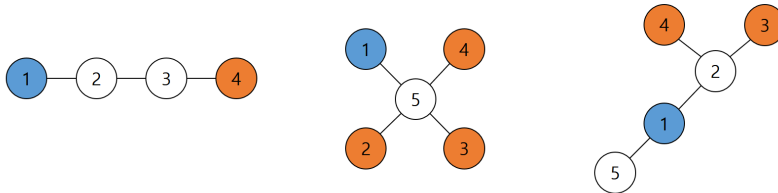
풀이 로직



• 트리의 지름을 이용한 방법

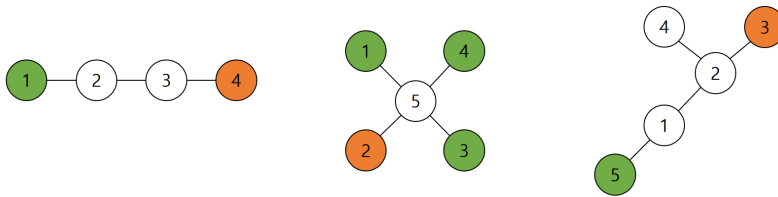
1. 임의의 노드(1: )에서 가장 거리가 먼 노드(X: )를 탐색한다. (한쪽 끝)
2. X()에서 한번 더 가장 먼 노드(Y: )를 탐색한다. (반대쪽 끝)
3. 만약 Y()의 개수가 2개 이상일 경우 트리의 지름이 중간값의 최대값이 된다
4. 만약 Y()의 개수가 1일 경우 Y에서 가장 먼 노드(Z: )를 탐색한다
5. 만약 Z()의 개수가 2개 이상일 경우 트리의 지름이 중간값의 최대값이 된다
6. 만약 Z()의 개수가 1개일 경우 트리 지름 - 1이 중간값의 최대값이 된다


상세 설명

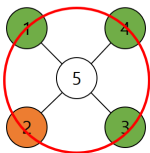
1. 임의의 노드(1: )에서 가장 거리가 먼 노드(X: )를 탐색한다. (한쪽 끝)





2. X()에서 한번 더 가장 먼 노드(Y: )를 탐색한다. (반대쪽 끝)



3. 만약 Y()의 개수가 2개 이상일 경우 트리의 지름이 중간값의 최대값이 된다

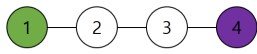


a	b	c	a~b	b~c	c~a	f(a,b,c)
1	5	3	1	1	2	1
2	1	4	2	2	2	2
2	3	4	2	2	2	2

4. 만약 Y()의 개수가 1일 경우 Y에서 가장 먼 노드(Z: )를 탐색한다

2024년 10월 10일

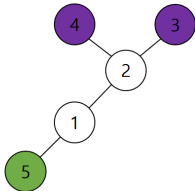
5. 만약 Z(●)의 개수가 1개일 경우 트리 지름 - 1이 중간값의 최대값이 된다
 이 경우 계속 반복하더라도 같은 노드를 선택하게 되는데 이때 중간값을 계산해 보면
 트리 지름의 -1이 중간값의 최댓값이 된다
 이유는 '중간값의 최댓값'을 구하기 위해선 가장 큰 값인 트리의 지름을 구성하는 노드
 2개를 우선 선택하는 것이 유리한데 다음의 예를 보자



a	b	c	a~b	b~c	c~a	f(a,b,c)
1	2	4	1	2	3	2
1	3	4	2	1	3	2

$$3(\text{트리 지름}) - 1 = 2$$

6. 만약 Z(●)의 개수가 2개 이상일 경우 트리의 지름이 중간값의 최대값이 된다



a	b	c	a~b	b~c	c~a	f(a,b,c)
1	3	5	2	3	3	3
1	4	5	2	3	1	2
2	3	5	1	3	2	2
2	4	5	1	3	2	2

소스 코드

```
// 소스 코드
package _12월3주차;

import java.util.*;

class 트리트리오중간값 {

    static ArrayList<Integer>[] nodes;
    static int N;

    public static int solution(int n, int[][] edges) {
        N = n;
        nodes = new ArrayList[N + 1];

        for (int i = 1; i < nodes.length; i++) {
            nodes[i] = new ArrayList<Integer>();
        }

        for (int[] edge : edges) {
            nodes[edge[0]].add(edge[1]);
            nodes[edge[1]].add(edge[0]);
        }

        // 임의의 정점 1에서 각 정점(들) x 구하기
        int start = 1;
        int[] distanceArr = bfs(start);

        // x는 임의의 정점 1에서 가장 먼 노드
        int X = start;
        for (int i = 1; i < distanceArr.length; i++) {
            if (distanceArr[X] <= distanceArr[i]) {
                X = i;
            }
        }
    }
}
```

```

// X로부터 가장 멀리 있는 Y 정점 구하기
distanceArr = bfs(X);
int Y = X;
for (int i = 1; i < distanceArr.length; i++) {
    if (distanceArr[Y] <= distanceArr[i]) {
        Y = i;
    }
}
// Y의 정점 개수 카운트
int cnt = 0;
for (int i = 1; i < distanceArr.length; i++) {
    if (distanceArr[Y] == distanceArr[i]) {
        cnt++;
    }
}
// 만약 Y 정점의 개수가 2개 이상이면 트리 지름이 정답
if (cnt >= 2) return distanceArr[Y];

// Y로 부터 가장 멀리있는 Z 정점 구하기
distanceArr = bfs(Y);
int Z = Y;
for (int i = 0; i < distanceArr.length; i++) {
    if (distanceArr[Z] <= distanceArr[i]) {
        Z = i;
    }
}
// Z의 정점 개수 카운트
cnt = 0;
for (int i = 1; i < distanceArr.length; i++) {
    if (distanceArr[Z] == distanceArr[i]) {
        cnt++;
    }
}

if (cnt >= 2) // Z가 2개 이상이면 트리 지름이 정답
    return distanceArr[Z];
else // Z가 1개면 트리 지름 -1이 정답
    return distanceArr[Z] - 1;
}

// 최대 거리의 있는 정점(들) 구하기
private static int[] bfs(int node) {
    Queue<Integer> queue = new LinkedList<>();
    boolean[] visited = new boolean[N + 1];

    queue.add(node);
    visited[node] = true;
    int[] dist = new int[N + 1];

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

        for (int linkedNode : nodes[curNode]) {
            if (visited[linkedNode]) continue;

```

```

        visited[linkedNode] = true;
        queue.add(linkedNode);
        dist[linkedNode] = dist[curNode] + 1;
    }

    }
    return dist;
}

public static void main(String[] args) {
    System.out.println(solution(4, new int[][]{{1, 2}, {2, 3}, {3, 4}}));
    System.out.println(solution(5, new int[][]{{1, 2}, {1, 3}, {2, 4}, {3, 5}}));
    System.out.println(solution(5, new int[][]{{1, 5}, {2, 5}, {3, 5}, {4, 5}}));
}
}

```

마무리

- 트리의 지름을 이용한 풀이의 이해가 어려웠음
- 해당 풀이에서는 BFS를 3번 돌려 풀었지만, BFS를 2번 사용하는 풀이로 개선할 수 있음