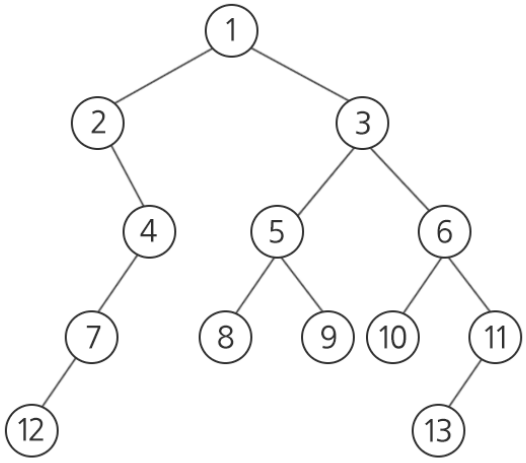
SWEA 공통조상

이진 트리에서 임의의 두 정점의 공통 조상 중 가장 가까운 것을 찾으려 한다.

예를 들어, 아래의 이진 트리에서 정점 8과 13의 공통 조상은 정점 3와 1 두 개가 있다.

이 중 8, 13에 가장 가까운 것은 정점 3이다.

정점 3을 루트로 하는 서브 트리의 크기(서브 트리에 포함된 정점의 수)는 8이다.



임의의 이진 트리가 주어지고, 두 정점이 명시될 때 이들의 공통 조상 중 이들에 가장 가까운 정점을 찾고, 그 정점을 루트로 하는 서브 트리의 크기를 알아내는 프로그램을 작성하라.

입력에서 주어지는 두 정점이 서로 조상과 자손 관계인 경우는 없다.

위의 트리에서 예를 든다면 두 정점이 "11과 3"과 같이 주어지는 경우는 없다.

[입력]

가장 첫줄은 전체 테스트케이스의 수이다.

10개의 테스트 케이스가 주어진다.

두 줄이 하나의 테스트 케이스가 되며, 따라서 전체 입력은 20줄로 이루어진다.

각 케이스의 첫줄에는 트리의 정점의 총 수 V와 간선의 총 수 E, 공통 조상을 찾는 두 개의 정점 번호가 주어진다 (정점의 수 V는 10 ≤ V ≤ 10000 이다).

그 다음 줄에는 E개 간선이 나열된다. 간선은 간선을 이루는 두 정점으로, 항상 "부모 자식" 순서로 표기된다.

위에서 예로 든 트리에서 정점 5와 8을 잇는 간선은 "5 8"로 표기되고, 절대로 "8 5"와 같이 표기되지는 않는다.

간선이 입력되는 순서는 정해져 있지 않다. 입력에서 이웃한 수는 모두 공백으로 구분된다.

정점의 번호는 1부터 V까지의 정수이며, 전체 트리에서 루트가 되는 정점은 항상 1번으로 표기된다.

부모 정점이 자식 정점보다 항상 번호가 작은 것은 아니다. 즉, 40번 정점이 20번 정점의 부모가 될 수도 있다.

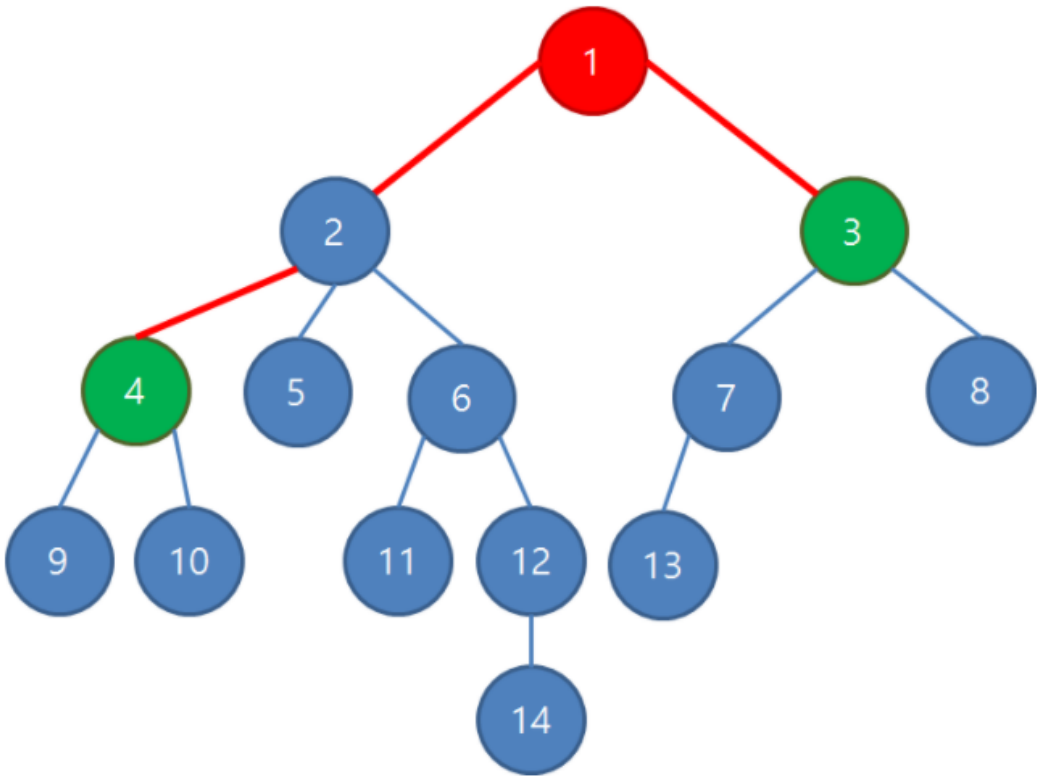
이 문제에서 자식 정점이 부모 정점의 왼쪽 자식인지 오른쪽 자식인지는 중요하지 않다.

[출력]

총 10줄에 10개의 테스트 케이스 각각에 대한 답을 출력한다.

각 줄은 테스트 케이스의 번호를 의미하는 '#x'로 시작하고 공백을 하나 둔 다음 답을 기록한다.

답은 공통조상 중 가장 가까운 것의 번호, 그것을 루트로 하는 서브 트리의 크기를 뜻하는 2개의 정수로 구성된다. 이 두 정수는 공백으로 구분한다.




```
u = 4 v = 3  => LCA = 1
```

공통조상을 찾는 방법 1

1. 두 정점 중 깊이가 더 깊은 정점에서 두 정점의 깊이가 같아질 때 까지 계속 부모로 이동합니다.

2. 두 정점이 만날 때 까지 동시에 부모로 이동시키며, 만나는 지점이 LCA 가 됩니다.

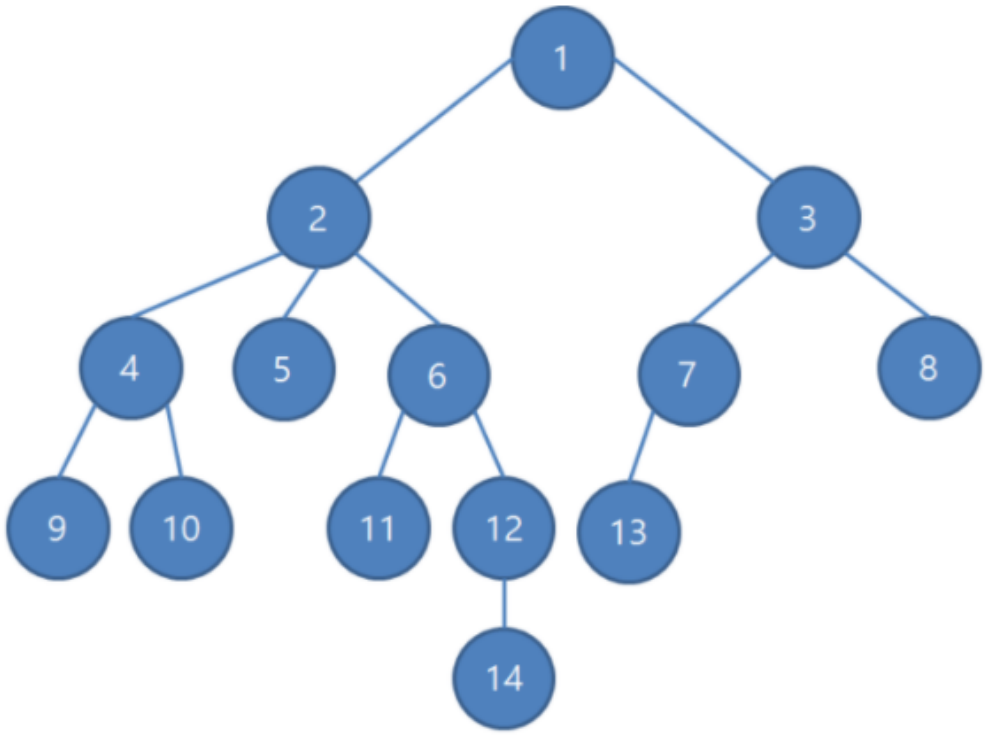
★ CheckPoint

하지만 최악의 경우 O(N)의 복잡도를 가지게 됩니다. (트리가 한쪽으로 치우친 경우)

```
                                root

                           1             2

                      3

                 4


             5

         ...


    500
```

만약 500 과 2의 LCA를 구해야 한다면, 같은 레벨로 올리기 위해 N만큼의 연산이 필요함

공통 조상을 찾는 방법2 (개선 된 버전)



k	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	-	1	1	2	2	2	3	3	4	4	6	6	7	12
1	-	-	-	1	1	1	1	1	2	2	2	2	3	6
2	-	-	-	-	-	-	-	-	-	-	-	-	-	1

기존의 부모 하나만을 parent 배열로 저장하던 것을 확장하여 2차원 배열 parent[u][k]로 생성

parent[4][1] => 4번 노드의 2 ^ k 번째 조상의 값 => 부모의 부모 값을 저장

공통조상 알고리즘의 기본 사이클

1. 높이를 갖게 맞춘다. (1개씩)

=> 개선된 알고리즘에서는 깊이가 많이 차이나는 것을 위의 테이블을 활용하여 개선합니다.

2 ^ k 번 조상의 높이를 다 계산하였기 때문에, 만약 둘의 깊이가 11이 차이가 난다면

11 = 1 + 2 + 8 이므로 parent[u][0] parent[u][1] parent[u][3] 의 형태로 따라간다면

3번만에 11번째 조상의 위치에 도달 하므로 개선할 수 있습니다.

2. 높이가 같을 때 각각 부모의 값으로 옮긴다.

=> 개선된 알고리즘에서는

만약 parent[u][k] != parent[v][k] 라고 하면,

두 정점의 LCA는 둘로 부터 2 ^ k 이상 떨어져 있는 것이 확실합니다.

근데 만약 parent[u][k + 1] == parent[v][k + 1] 이라면

LCA는 반드시 2 ^ k 번째 조상과 2 ^ k + 1 번째 조상 사이의 값에 존재하게 됩니다.

이제 k를 큰 값부터 시도하면서, parent[u][k] != parent[v][k] 조건이 만족하게 되면

두 정점의 높이를 동시의 2 ^ k 만큼 위로 올려보내는 것으로 단축할 수 있습니다.

코드(O(N))

```
package Algo_Study_SWEA;

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

public class Solution_공통조상 {
  static List<List<Integer>> list;
  static int[] depth, parent;
  static int size;

  private static void size_cnt(int node) {

    Queue<Integer> queue = new LinkedList<Integer>();

    queue.add(node);
    size++;
    while (!queue.isEmpty()) {
      int cur = queue.poll();

      if (list.get(cur).size() == 0) {
        continue;
      }
      for (int child : list.get(cur)) {
        if (depth[child] >= depth[cur]) {
          queue.add(child);
          size++;
        }
      }
    }
  }

  private static int solution(int A, int B) {
```

```
  while (depth[A] > depth[B]) {
    A = parent[A];
  }
  while (depth[A] < depth[B]) {
    B = parent[B];
  }
  while (A != B) {
    A = parent[A];
    B = parent[B];
  }

  return A;
}

private static void dfs(int node, int cnt) {
  depth[node] = cnt;

  for (int child : list.get(node)) {
    if (depth[child] == 0) {
      dfs(child, cnt + 1);
      parent[child] = node;
    }
  }
}

public static void main(String[] args) throws Exception {
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
  int T = Integer.parseInt(br.readLine());

  for (int tc = 1; tc <= T; tc++) {
    StringTokenizer str = new StringTokenizer(br.readLine());
    size = 0;
    int V = Integer.parseInt(str.nextToken());
    int E = Integer.parseInt(str.nextToken());

    int A = Integer.parseInt(str.nextToken());
    int B = Integer.parseInt(str.nextToken());

    list = new ArrayList<>();

    for (int i = 0; i <= V; i++) {
      list.add(new ArrayList<Integer>());
    }

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

    for (int i = 0; i < E; i++) {
      int a = Integer.parseInt(input.nextToken());
      int b = Integer.parseInt(input.nextToken());

      list.get(a).add(b);
      list.get(b).add(a);
    }

    depth = new int[V + 1];
    parent = new int[V + 1];
```

```
      dfs(1, 1);
      int result = solution(A, B);
      size_cnt(result);
      System.out.println("#" + tc + " " + solution(A, B) + " " + size);
    }
  }
}
```

개선된 코드(log(N))

```
추후 업로드..
```

출처) https://m.blog.naver.com/kks227/220820773477