최소 공통 조상(LCA)

수업 시작합니다!

CONTENTS

1교시

Lowest Common Ancestor

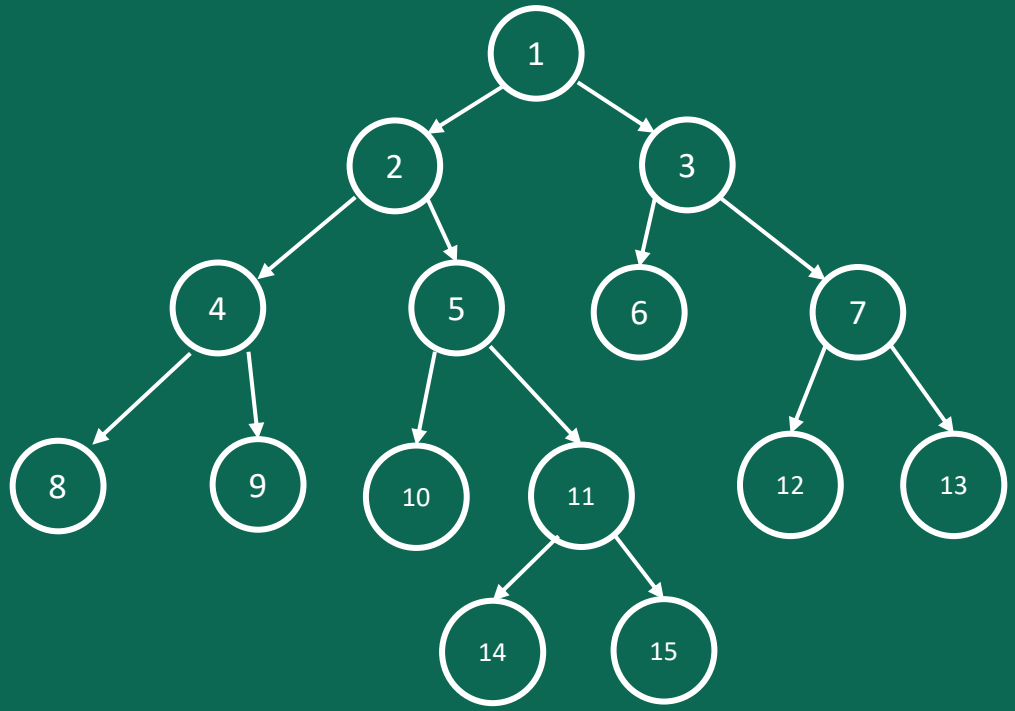
최소 공통 조상(Lowest Common Ancestor)

- 백준 – 11437 LCA (G3)

- N($2 \leq N \leq 50,000$)개의 정점으로 이루어진 트리가 주어진다. 트리의 각 정점은 1번부터 N번까지 번호가 매겨져 있으며, 루트는 1번이다.
- 두 노드의 쌍 M($1 \leq M \leq 10,000$)개가 주어졌을 때, 두 노드의 가장 가까운 공통 조상이 몇 번인지 출력한다.

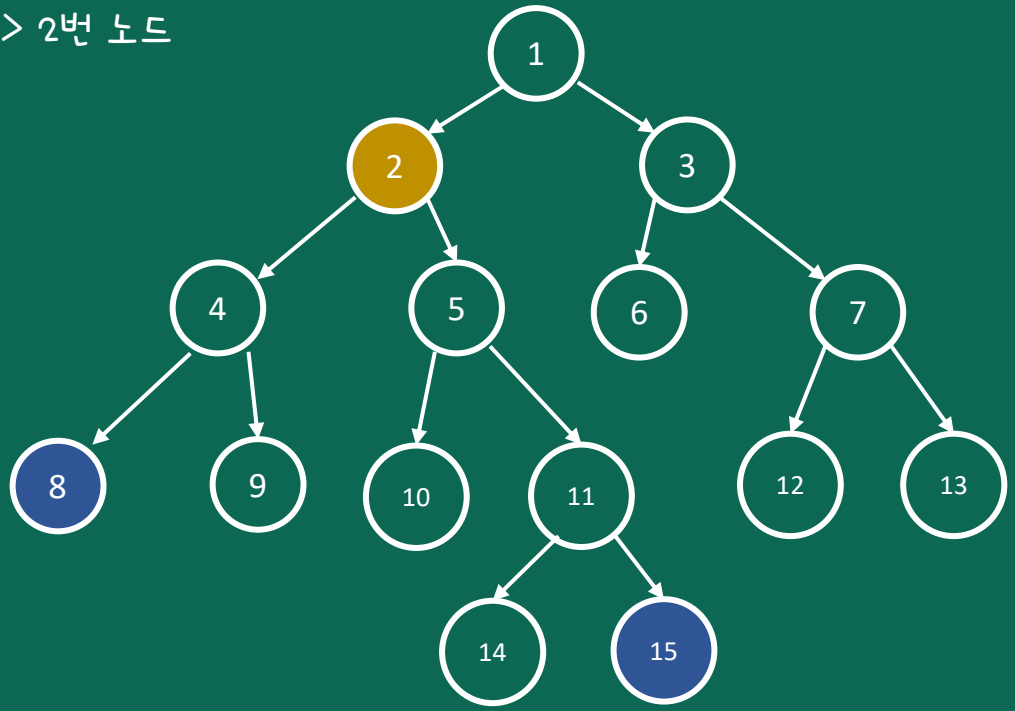
최소 공통 조상(Lowest Common Ancestor)

- 최소 공통 조상(LCA) 문제는 두 노드의 공통된 조상 중 가장 가까운 조상을 찾는 문제



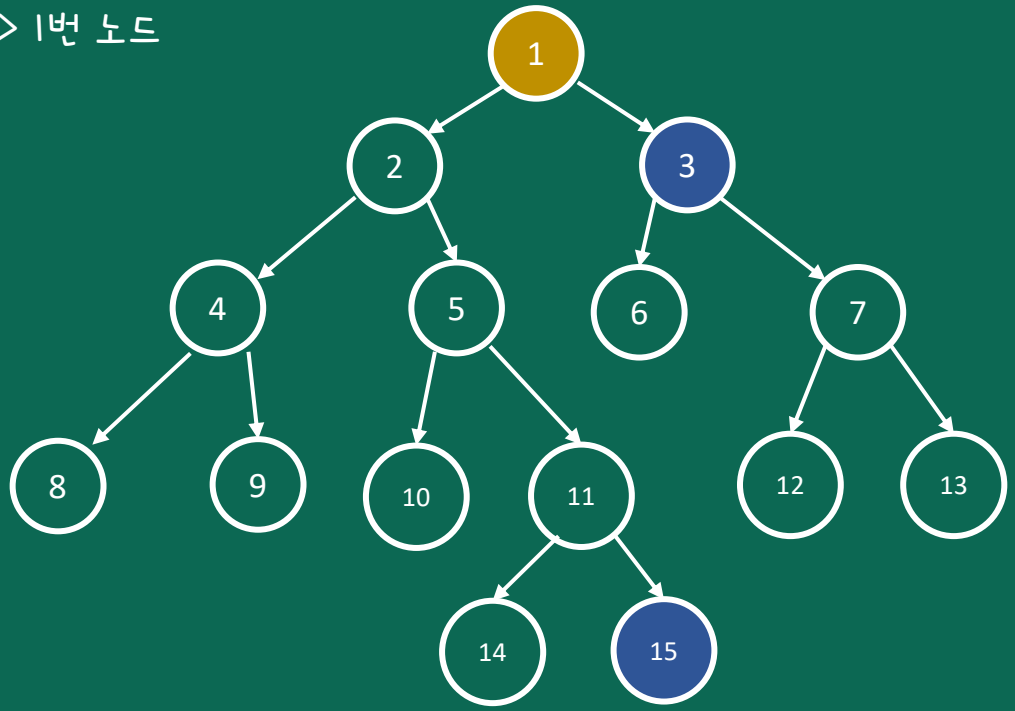
최소 공통 조상(Lowest Common Ancestor)

- 최소 공통 조상(LCA) 문제는 두 노드의 공통된 조상 중 가장 가까운 조상을 찾는 문제
- LCA(8번 노드, 15번 노드) -> 2번 노드



최소 공통 조상(Lowest Common Ancestor)

- 최소 공통 조상(LCA) 문제는 두 노드의 공통된 조상 중 가장 가까운 조상을 찾는 문제
- LCA(3번 노드, 15번 노드) -> 1번 노드



2교시

O(NM) 알고리즘

최소 공통 조상(Lowest Common Ancestor)

- 최소 공통 조상 찾기 알고리즘은 다음과 같습니다.

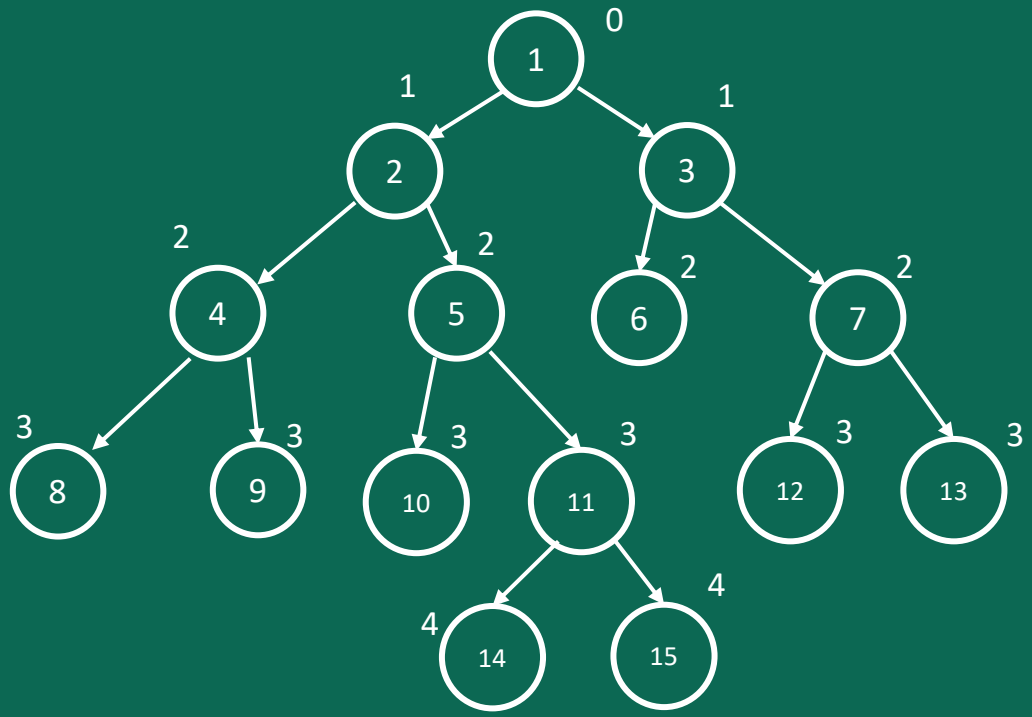
1. 모든 노드에 대한 깊이(depth)를 계산합니다

2. 최소 공통 조상을 찾을 두 노드를 확인합니다.
   2-1. 먼저 두 노드의 깊이(depth)가 동일하도록 거슬러 올라갑니다.
   2-2. 이후에 부모가 같아질 때까지 반복적으로 두 노드의 부모 방향으로 거슬러 올라갑니다.

3. 모든 LCA(a,b) 연산에 대하여 2번의 과정을 반복합니다.

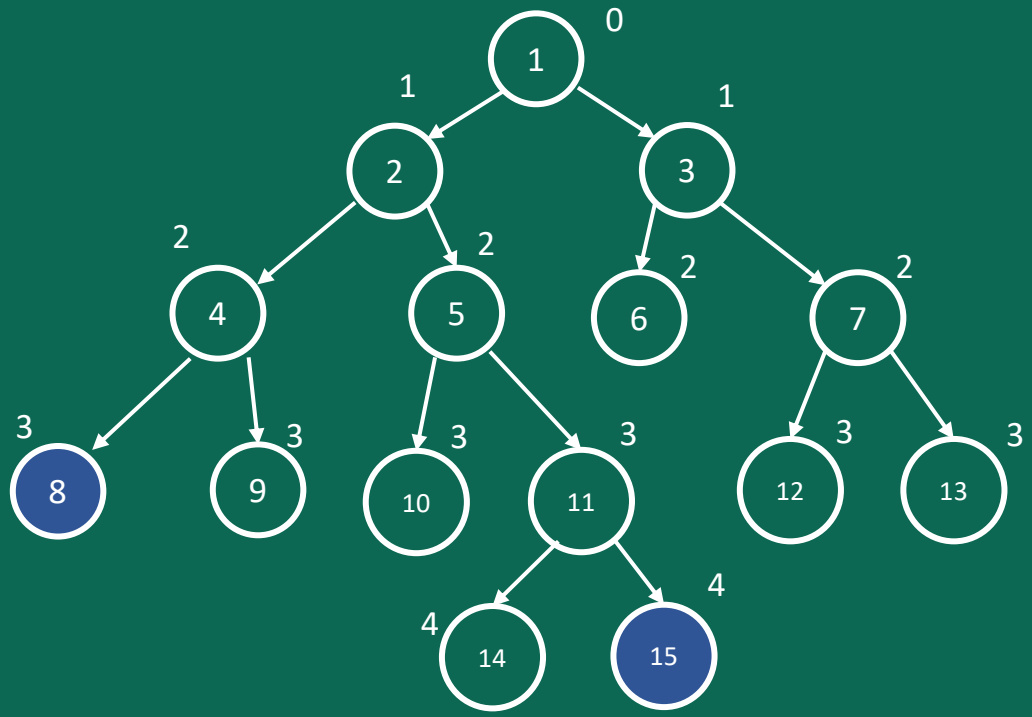
최소 공통 조상(Lowest Common Ancestor)

- DFS를 이용해 모든 노드에 대하여 깊이(depth)를 계산할 수 있습니다.



최소 공통 조상(Lowest Common Ancestor)

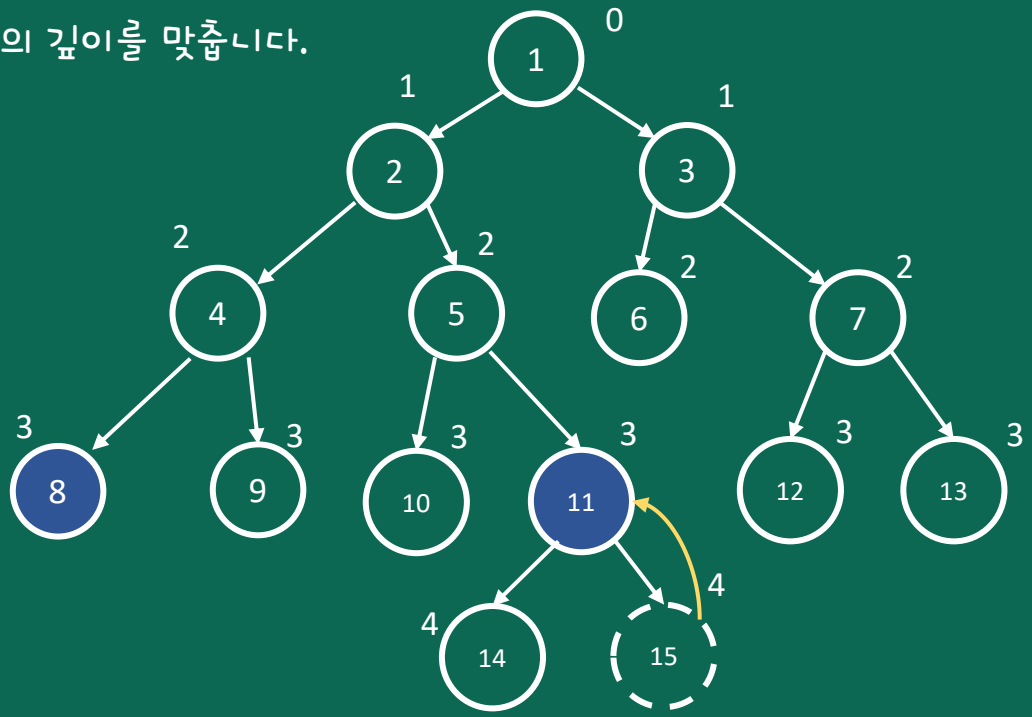
- LCA(8번 노드, 15번 노드)



최소 공통 조상(Lowest Common Ancestor)

- LCA(8번 노드, 15번 노드)

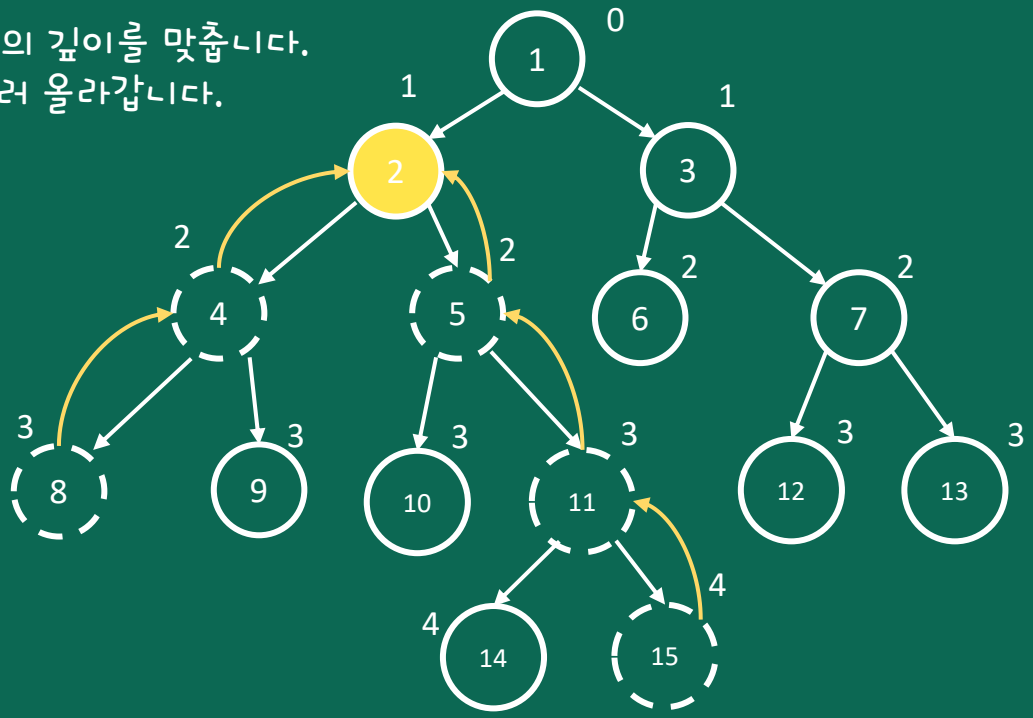
1. 먼저 두 노드의 깊이를 맞춥니다.



최소 공통 조상(Lowest Common Ancestor)

- LCA(8번 노드, 15번 노드)

1. 먼저 두 노드의 깊이를 맞춥니다.
2. 이후에 거슬러 올라갑니다.



최소 공통 조상(Lowest Common Ancestor)

● 백준 – 11437 LCA (G3)

```java
public class Main {

    /*
     * parent : 부모 노드 정보
     * depth : 각 노드까지의 깊이
     * check : 각 노드의 깊이가 계산되었는지 여부
     * graph : 그래프 정보
     */

    static int[] parent, depth;
    static boolean[] check;

    static List<List<Integer>> graph = new ArrayList<>();

    public static void main(String[] args) throws Exception {

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StringBuilder sb = new StringBuilder();

        int N = Integer.parseInt(br.readLine());

        parent = new int[N + 1];
        depth = new int[N + 1];
        check = new boolean[N + 1];

        for (int i = 0; i <= N; i++) {
            graph.add(new ArrayList<>());
        }

        for (int i = 0; i < N-1; i++) {
            StringTokenizer st = new StringTokenizer(br.readLine());
            int a = Integer.parseInt(st.nextToken());
            int b = Integer.parseInt(st.nextToken());

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

        //  루트 노드는 1번 노드
        DFS(1, 0);

        int M = Integer.parseInt(br.readLine());

        for (int i = 0; i < M; i++) {

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

            int a = Integer.parseInt(st.nextToken());
            int b = Integer.parseInt(st.nextToken());

            int ans = LCA(a, b);

            sb.append(ans + "\n");
        }

        System.out.println(sb.toString());

    }
```

최소 공통 조상(Lowest Common Ancestor)

```java
// 루트 노드부터 시작하여 깊이(depth)를 구하는 함수
static void DFS(int node, int d) {
    check[node] = true;
    depth[node] = d;

    for (int next : graph.get(node)) {

        // 이미 깊이를 구했다면 넘기기
        if (check[next]) {
            continue;
        }

        parent[next] = node;
        DFS(next, d + 1);
    }
}
```

```java
// A와 B의 공통 조상을 찾는 함수
static int LCA(int a, int b) {

    // 먼저 깊이(depth)가 동일하도록
    while (depth[a] != depth[b]) {

        if (depth[a] > depth[b]) {
            a = parent[a];
        } else {
            b = parent[b];
        }

    }

    // 노드가 같아지도록
    while (a != b) {
        a = parent[a];
        b = parent[b];
    }
    return a;
}
```

기본적인 최소 공통 조상 (LCA)알고리즘 : 시간 복잡도 분석

- 매 쿼리마다 부모 방향으로 거슬러 올라가기 위해 최악의 경우 O(N)의 시간 복잡도가 요구됩니다.
- 따라서 M개의 모든 쿼리를 처리할 때의 시간 복잡도는 O(NM)입니다.

3교시

O(MlogN) 알고리즘

최소 공통 조상(Lowest Common Ancestor)

- 백준 – 11438 LCA2 (P5)

- N(2 ≤ N ≤ 100,000)개의 정점으로 이루어진 트리가 주어진다. 트리의 각 정점은 1번부터 N번까지 번호가 매겨져 있으며, 루트는 1번이다. 두 노드의 쌍 M(1 ≤ M ≤ 100,000)개가 주어졌을 때, 두 노드의 가장 가까운 공통 조상이 몇 번인지 출력한다.

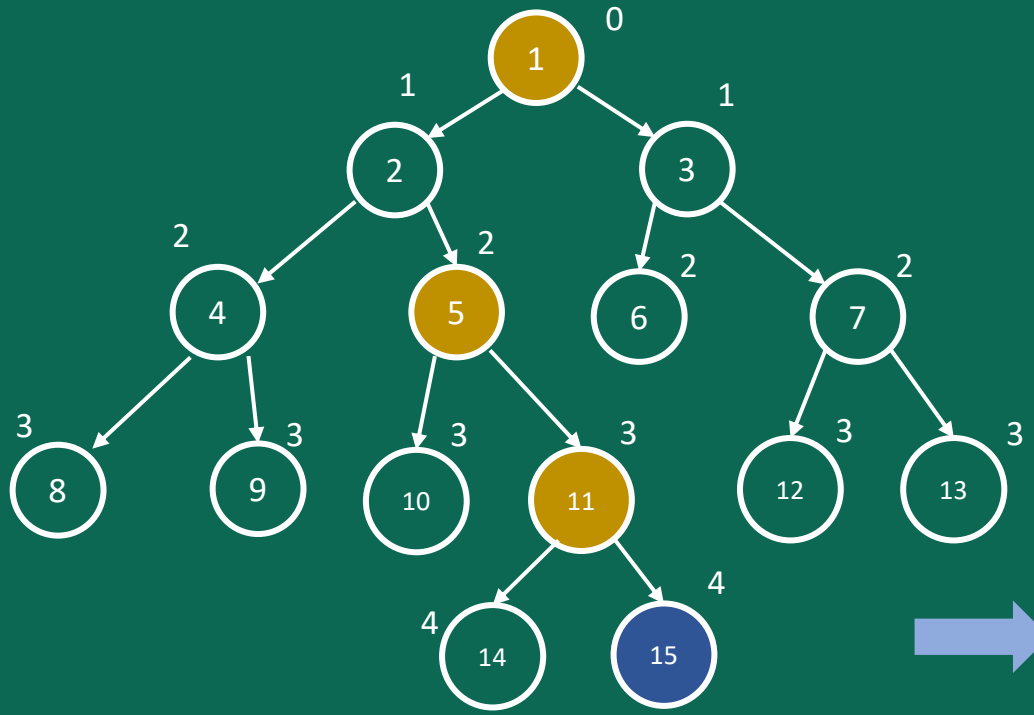
최소 공통 조상(Lowest Common Ancestor)

- 각 노드가 거슬러 올라가는 속도를 빠르게 만드는 방법에 대하여 고민해 봅시다.

- 만약 총 15칸 거슬러 올라가야 한다면?
  8칸 -> 4칸 -> 2칸 -> 1칸
- 2의 제곱 형태로 거슬러 올라가도록 하면 O(logN)의 시간 복잡도를 보장할 수 있습니다.
- 메모리를 조금 더 사용하여 각 노드에 대하여 2^i 번째 부모에 대한 정보를 기록합시다.

최소 공통 조상(Lowest Common Ancestor)

- 모든 노드에 대하여 깊이(depth)와 2^i 번째 부모에 대한 정보를 계산합니다.

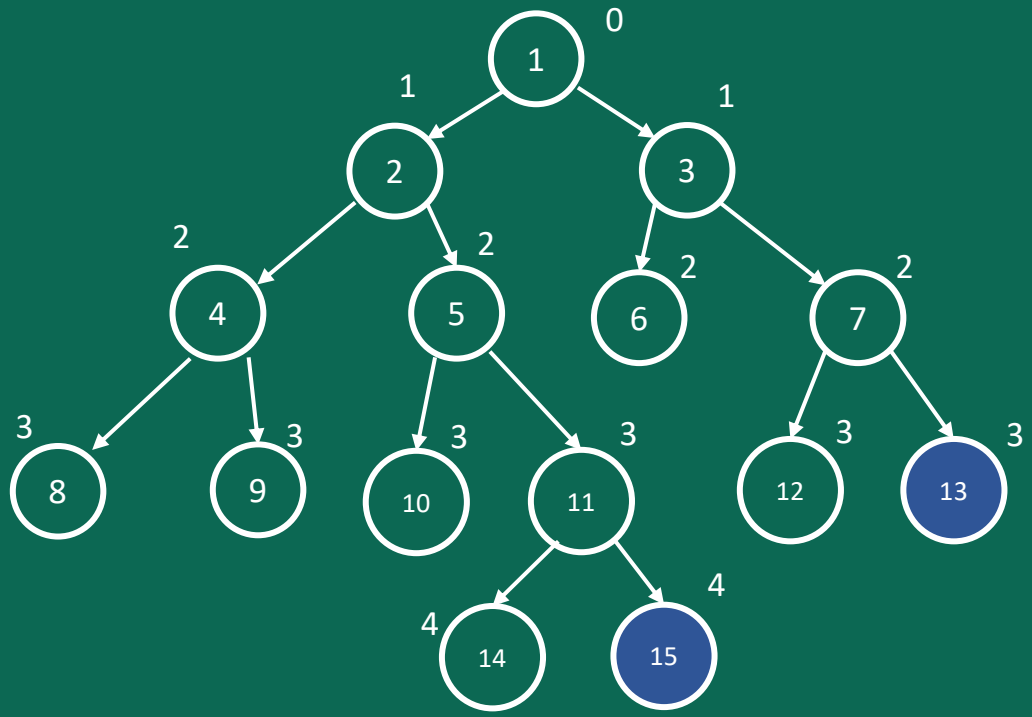


- 이때 노드의 개수가 N일 때 N*logN 만큼의 공간이 필요하다는 것을 알 수 있다.

i = 0	i = 1	i = 2
11	5	1

최소 공통 조상(Lowest Common Ancestor)

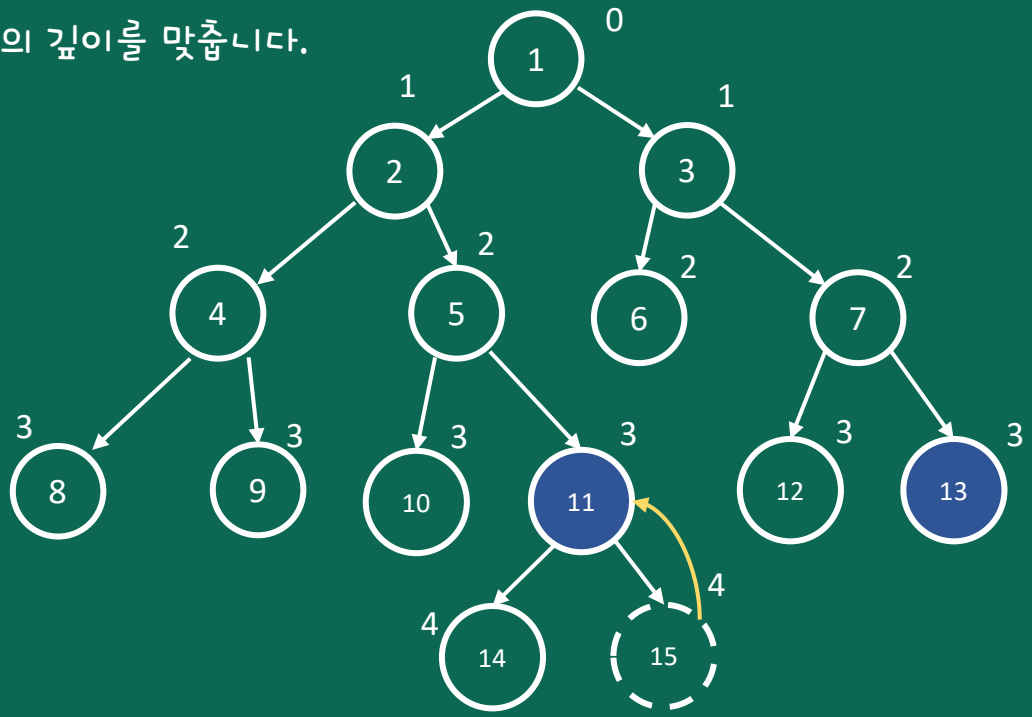
- LCA(13번 노드, 15번 노드)



최소 공통 조상(Lowest Common Ancestor)

- LCA(13번 노드, 15번 노드)

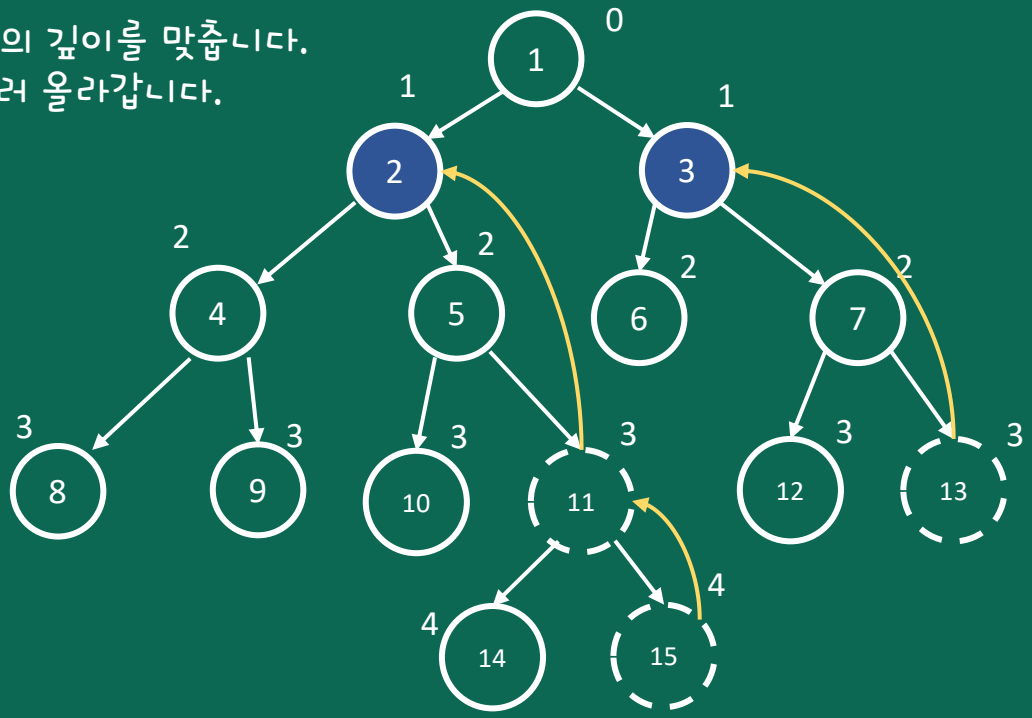
1. 먼저 두 노드의 깊이를 맞춥니다.



최소 공통 조상(Lowest Common Ancestor)

- LCA(13번 노드, 15번 노드)

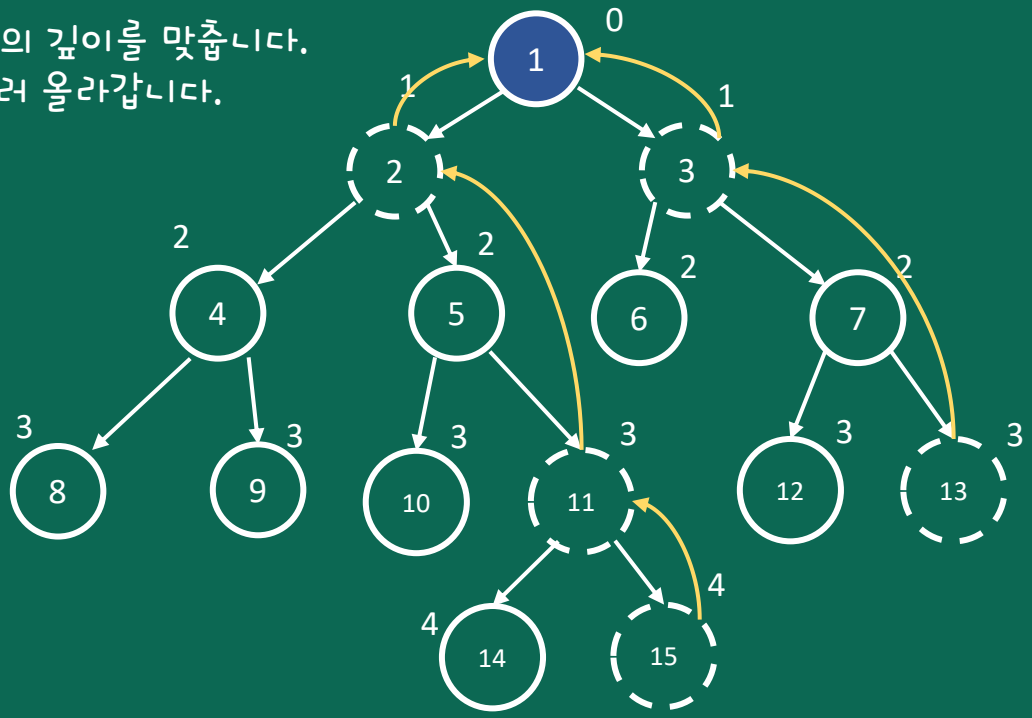
1. 먼저 두 노드의 깊이를 맞춥니다.
2. 이후에 거슬러 올라갑니다.



최소 공통 조상(Lowest Common Ancestor)

- LCA(13번 노드, 15번 노드)

1. 먼저 두 노드의 깊이를 맞춥니다.
2. 이후에 거슬러 올라갑니다.



기본적인 최소 공통 조상 (LCA)알고리즘 : 시간 복잡도 분석

- 다이나믹 프로그래밍을 이용해 시간 복잡도를 개선할 수 있습니다.
  - 세그먼트 트리를 이용하는 방법도 존재합니다.

- 매 쿼리마다 부모를 거슬러 올라가기 위해 O(logN)의 복잡도가 필요합니다.
  - 따라서 M개의 모든 쿼리를 처리할 때의 시간 복잡도는 O(MlogN)입니다.

최소 공통 조상(Lowest Common Ancestor)

● 백준 – 11438 LCA2 (P5)

```java
public class Main {
    /*
     * parent : 부모 노드 정보
     * depth : 각 노드까지의 깊이
     * check : 각 노드의 깊이가 계산되었는지 여부
     * graph : 그래프 정보
     */

    // 2^20 = 100_000, 최대 데이터 100만개 가정
    static int logN = 21, N;
    static int[] depth;
    static int[][] parent;
    static boolean[] check;

    static List<List<Integer>> graph = new ArrayList<>();

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

        N = Integer.parseInt(br.readLine());

        parent = new int[N + 1][logN];
        depth = new int[N + 1];
        check = new boolean[N + 1];

        for (int i = 0; i <= N; i++) {
            graph.add(new ArrayList<>());
        }

        for (int i = 0; i < N - 1; i++) {
            StringTokenizer st = new StringTokenizer(br.readLine());
            int a = Integer.parseInt(st.nextToken());
            int b = Integer.parseInt(st.nextToken());

            graph.get(a).add(b);
            graph.get(b).add(a);
        }
        setParent();

        int M = Integer.parseInt(br.readLine());

        for (int i = 0; i < M; i++) {

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

            int a = Integer.parseInt(st.nextToken());
            int b = Integer.parseInt(st.nextToken());

            int ans = LCA(a, b);

            sb.append(ans + "\n");
        }

        System.out.println(sb.toString());

    }
```

최소 공통 조상(Lowest Common Ancestor)

```java
//  루트 노드부터 시작하여 깊이(depth)를 구하는 함수
static void DFS(int node, int d) {
    check[node] = true;
    depth[node] = d;
    for (int next : graph.get(node)) {

        //  이미 깊이를 구했다면 넘기기
        if (check[next]) {
            continue;
        }
        parent[next][0] = node;
        DFS(next, d + 1);
    }
}

//  전체 부모 관계를 설정하는 함수
static void setParent() {

    //  루트 노드는 1번 노드
    DFS(1, 0);

    //  2^i 번째 부모를 저장하는 점화식
    for (int i = 1; i < logN; i++) {
        for (int j = 1; j <= N; j++) {
            parent[j][i] = parent[parent[j][i - 1]][i - 1];
        }
    }
}
```

```java
//  a와 b의 최소 공통 조상을 찾는 함수
static int LCA(int a, int b) {
    //  b가 더 깊도록 설정
    if (depth[a] > depth[b]) {
        int temp = a;
        a = b;
        b = temp;
    }

    //  먼저 깊이가 동일 하도록(2의 제곱수 만큼 이동)
    for (int i = logN - 1; i >= 0; i--) {
        if (depth[b] - depth[a] >= (1 << i)) {
            b = parent[b][i];
        }
    }

    //  높이만 조절 했는데 같아졌다 == a가 조상이었다
    if (a == b) {
        return a;
    }

    //  조상을 향해 거슬러 올라가기
    for (int i = logN - 1; i >= 0; i--) {
        if (parent[a][i] != parent[b][i]) {
            a = parent[a][i];
            b = parent[b][i];
        }
    }

    //  이후에 부모가 찾고자 하는 조상
    return parent[a][0];
}
```

$\frac{\text{ㄱㄱ}}{\text{ㅌ}}$