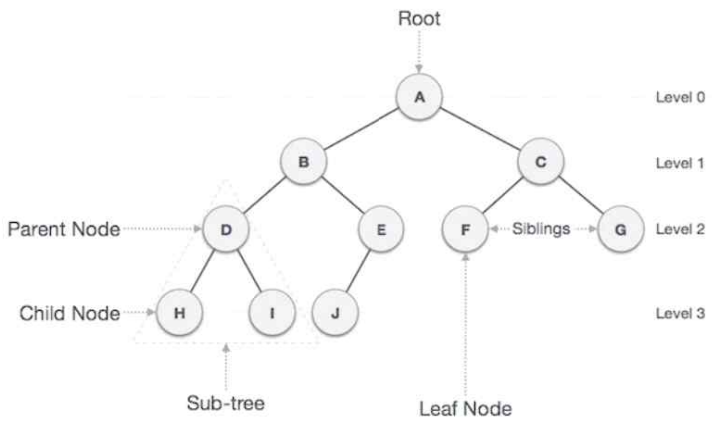
◉ Three

앞서 보인 큐(Queue), 스택(Stack) 은 자료구조에서 선형 구조라고 합니다.
선형 구조란 자료를 구성하고 있는 데이터들이 순차적으로 나열시킨 형태를 의미합니다.
트리는 바로 비선형 구조입니다.비선형 구조는
선형구조와는 다르게 데이터가 계층적 혹은 망으로 구성되어있습니다.

계층형구조

**Node**: 트리에서 데이터를 저장하는 기본 요소
**Root Node**: 트리 맨 위에 있는 노드
**Level**: 최상위 노드를 Level 0으로 하였을 때, 하위 Branch로 연결된 노드의 깊이를 나타냄
**Parent Node**: 어떤 노드의 상위 레벨에 연결된 노드
**Child Node**: 어떤 노드의 하위 레벨에 연결된 노드
**Leaf Node**(Terminal Node): Child Node가 하나도 없는 노드
**Sibling**: 동일한 Parent Node를 가진 노드
**Depth**: 트리에서 Node가 가질 수 있는 최대 Level




트리는 이진트리, 이진탐색트리, 균형트리, 이진힙 등 매우 다양한 트리가 있지만 이번 시간에는 이진트리와 완전이진트리에 대해서 학습합니다

이진트리의 특징은 바로 각 노드가 최대 두 개의 자식을 가진다
하위의 노두가 4~5개일 수 없다, 2개만 있어야 한다

```
   o      Level 0
 o o o    Level 1
o  o  o   Level 2 # 이진 트리(X)

   o      Level 0
 o   o    Level 1
o o o     Level 2 # 이진 트리(O)
```

👉 완전 이진 트리(Complete Binary Tree)의 특징은 바로
노드를 삽입할 때 최하단 왼쪽 노드부터 차례대로 삽입해야 한다는 것 입니다!

```
   o      Level 0
 o   o    Level 1
  o o     Level 2  # -> 이진 트리 O 완전 이진 트리 X

   o      Level 0
 o   o    Level 1
o o o     Level 2  # -> 이진 트리 O 완전 이진 트리 O
```

▼ 완전 이진 트리를 배열로 표현

👉 트리 구조를 표현하는 방법은
직접 클래스를 구현해서 사용하는 방법이 있고,
배열로 표현하는 방법이 있습니다.

엇, 트리 구조를 어떻게 배열에 저장할 수 있을까요?
바로 **완전 이진 트리**를 쓰는 경우에 사용할 수 있습니다!

완전 이진 트리는 왼쪽부터 데이터가 쌓이게 되는데,
이를 순서대로 배열에 쌓으면서 표현할 수 있습니다.

예를 들어 아래와 같은 완전 이진 트리를 배열에 표현한다면, 다음과 같습니다!

```
트리를 구현할 때는 편의성을 위해 0번째 인덱스는 사용되지 않습니다!
그래서 None 값을 배열에 넣고 시작합니다! [None]

     8      Level 0 -> [None, 8] 첫번째 레벨의 8을 넣고,
   6   3    Level 1 -> [None, 8, 6, 3] 다음 레벨인 6, 3을 넣고
  4 2 5     Level 2 -> [None, 8, 6, 3, 4, 2, 5] 다음 레벨인 4, 2, 5를 넣으면 됩니다!

자 그러면, [None, 8, 6, 3, 4, 2, 5] 라는 배열이 되는데
다시 역으로 이 배열을 활용해서 트리 구조를 분석해보겠습니다.
다음과 같은 방법으로 트리 구조를 파악할 수 있습니다.

1. 현재 인덱스 * 2 -> 왼쪽 자식의 인덱스
2. 현재 인덱스 * 2 + 1 -> 오른쪽 자식의 인덱스
3. 현재 인덱스 // 2 -> 부모의 인덱스

예를 들어서 1번째 인덱스인 8의 왼쪽 자식은 6, 오른쪽 자식은 3 입니다.
그러면 1 * 2 = 2번째 인덱스! 6!
그러면 1 * 2 + 1 = 3번째 인덱스! 3! 입니다!
부모를 찾아보면, 3 // 2 = 1번째 인덱스 8 이므로 부모를 찾을 수 있습니다.

이를 다시 생각해보면
[None, 8, 6, 3, 4, 2, 5] 는
8 밑에 6, 3 이 있고, 6, 3 밑에 4, 2, 5가 있는 완전 이진 트리구나! 생각할 수 있습니다.
```

아래 예시를 보면, 레벨을 k라고 한다면
각 레벨에 최대로 들어갈 수 있는 노드의 개수는
$2^k$ 개수 임을 알 수 있습니다.

```
     1            Level 0 -> 1개
   2   3          Level 1 -> 2개
  4 5 6 7         Level 2 -> 4개
8 9..... 14 15   Level 3 -> 8개
                 Level k -> 2^k 개
```

완전트리의 높이가 h라면  노드의 개수가 몇 개일까 ?( 꽉찬이진트리라면)
N = $2^{h+1}$ +1

**완전이진트리의 노드가 개수가 N 이라면 높이면 O( log N)이다.**

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    public TreeNode(int val) {
        this.val = val;
    }
}

public class BinaryTreeExample {
    public static void main(String[] args) {
        // 이진 트리 생성
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
        root.right.left = new TreeNode(6);
        root.right.right = new TreeNode(7);

        // 이진 트리 순회
        System.out.println("전위 순회 결과: ");
        preOrderTraversal(root);

        System.out.println("\n\n중위 순회 결과: ");
        inOrderTraversal(root);

        System.out.println("\n\n후위 순회 결과: ");
        postOrderTraversal(root);
    }

    // 전위 순회 (Pre-order Traversal)
    private static void preOrderTraversal(TreeNode root) {
        if (root != null) {
            System.out.print(root.val + " ");
            preOrderTraversal(root.left);
            preOrderTraversal(root.right);
        }
    }

    // 중위 순회 (In-order Traversal)
    private static void inOrderTraversal(TreeNode root) {
        if (root != null) {
            inOrderTraversal(root.left);
            System.out.print(root.val + " ");
            inOrderTraversal(root.right);
        }
    }

    // 후위 순회 (Post-order Traversal)
    private static void postOrderTraversal(TreeNode root) {
        if (root != null) {
            postOrderTraversal(root.left);
            postOrderTraversal(root.right);
            System.out.print(root.val + " ");
        }
    }
}
```