트리(Tree)

▼ **트리(Tree)**

트리(Tree)의 개념

-트리는 **정점(Node)**과 **선분(branch)**를 이용하여 사이클을 이루지 않도록 구성한 그래프(Graph)의 특수한 형태이다.

- 가족이 계보(족보), 연산 수식, 회사 조직 구성도, **히프(Heep)** 등을 표현하기에 적합하다.

노드(Node)

- 트리의 기본 요소로서 자료 항목과 다른 항목에 대한 가지(Branch)를 합친 것
- Node = Branch

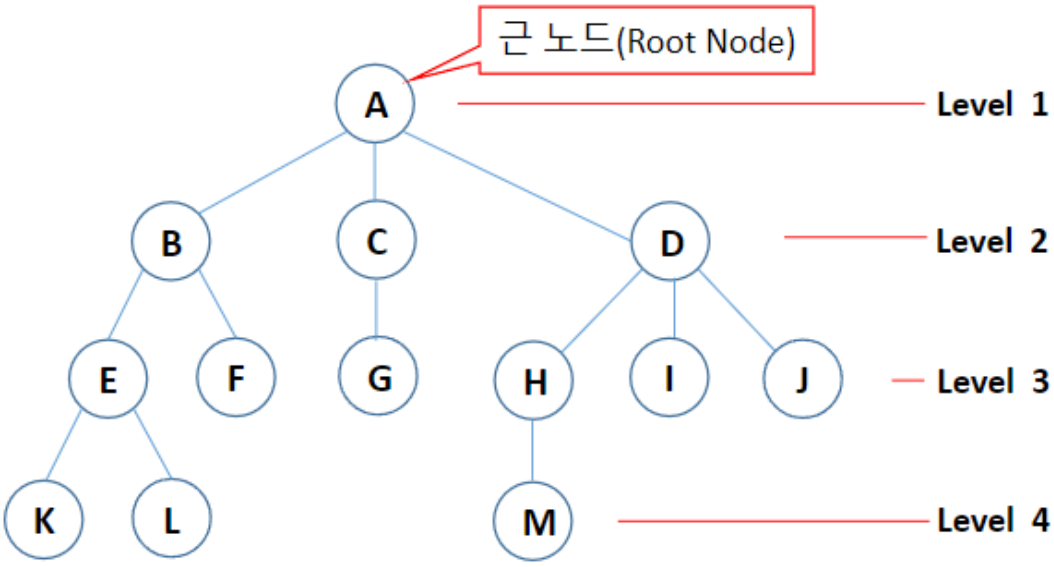
깊이(Depth,Height)

- 트리에서 노드가 가질 수 있는 최대의 레벨

트리 관련 용어+(이해 자료)

- **디그리(Degree)** : 차수로 각 노드에서 뻗어 나온 가지의 수
- **단말 노드(Terminal Node, = 잎(Leaf) 노드)** : 자식이 없는 노드 즉, Degree(차수)가 0인 노드
- **비단말 노드(Non-Terminal Node)**: 자식이 하나라도 있는 노드, Degree(차수)가 0이 아닌

  노드
- **조상 노드(Ancestors Node)** : 임의의 노드에서 근 노드에 이르는 경로상에 있는 노드들
- **자식 노드(Son Node)** : 어떤 노드에 연결된 다음 레벨의 노드들
- **부모 노드(Parent Node)** : 어떤 노드에 연결된 이전 레벨의 노드들
- **형제 노드(Brother Node, Sibling)** : 동일한 부모를 갖는 노드들



▼ 이진 트리(Binary Tree)

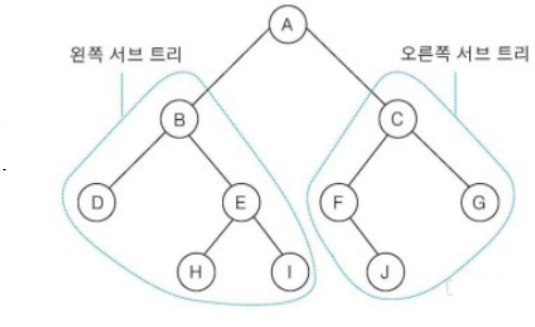
이진 트리(Binary Tree)의 개념

-모든 노드들의 자식 노드가 두 개 이하인 트리를 의미한다.

-이진 트리에서는 서브 트리가 두 개 이하기 때문에 서브 트리는

왼쪽 서브 트리와 오른쪽 서브 트리로 구분한다.

-이진 트리의 **전체 노드의 수 는: 2^k - 1 (k는 레벨)**



이진 트리(binary tree)의 순회

> 💡 이진 트리의 모든 노드를 특정한 순서대로 한 번씩 방문하는 것이다.
>
> 순회하는 방법에는 전위(preorder), 중위(inorder), 후위(postorder) 순회가 있다.

- **전위 순회(Preorder)**

노드(루트)를 먼저 방문하고 왼쪽 서브 트리, 오른쪽 서브 트리 순으로 방문

루트 방문 → 왼쪽 서브 트리 방문 → 오른쪽 서브 트리 방문 (Root -> Left -> Right)

- **중위 순회(Inorder)**

왼쪽 서브 트리, 루트, 오른쪽 서브 트리 순으로 방문

왼쪽 서브 트리 방문 → 루트 방문 → 오른쪽 서브 트리 방문 (Left -> Root -> Right)

- **후위 순회(PosTorder)**

왼쪽 서브 트리, 오른쪽 서브 트리, 루트 순으로 방문

왼쪽 서브 트리 방문 → 오른쪽 서브 트리 방문 → 루트 방문

(Left -> Right -> Root)

이진 트리 탐색(Binary Search Tree)

이진 탐색 트리(binary search tree)는 데이터의 삽입, 삭제, 탐색 등이 자주 발생하는

경우에 효율적인 구조로, 이진 트리이면서 같은 값을 갖는 노드가 없어야 한다.

왼쪽 서브 트리에 있는 모든 데이터는 현재 노드의 값보다 작고,

오른쪽 서브 트리에 있는 모든 노드의 데이터는 현재 노드의 값보다 크다.

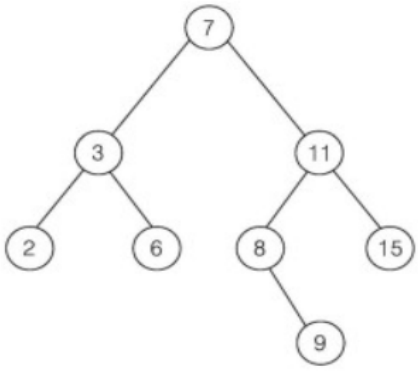
- **데이터 탐색은 루트에서부터 시작된다.**

- 루트 노드의 데이터와 찾으려는 데이터를 비교하여 **같으면 탐색은 성공 종료**
- **루트 노드가 작으면 루트 노드의 오른쪽**
- **루트 노드가 크면 루트 노드의 왼쪽**

- **데이터가 8인 노드를 탐색하는 과정을 살펴보면**

->7 > 11 > 8

-> 못 찾으면 실패

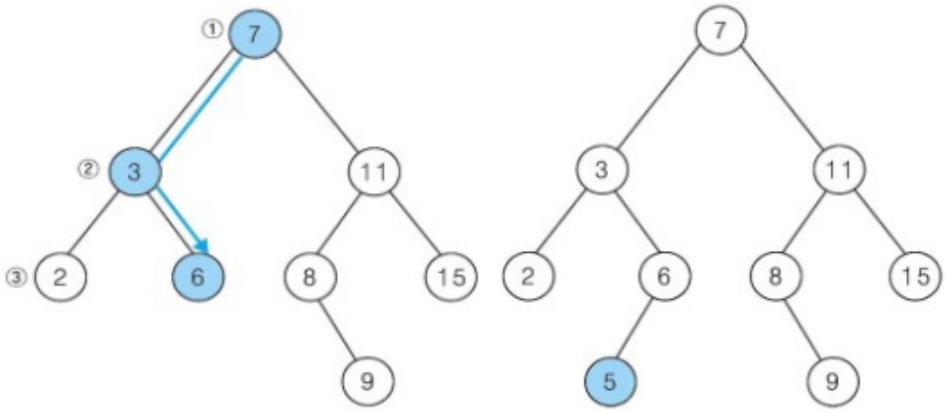


이진 트리 삽입

이진 탐색 트리에서의 **삽입은 탐색 동작을** 통해 이루어진다.

탐색에 성공하면 삽입은 실패하는데, 이는 이진 탐색 트리는 같은 데이터를 갖는

노드가 없어야 하기 때문이다.



이진 트리 삭제

💡 -이진 탐색 트리에서 노드를 삭제하는 동작은 삭제할 노드의 위치에 따라 세 가지로

구분된다.

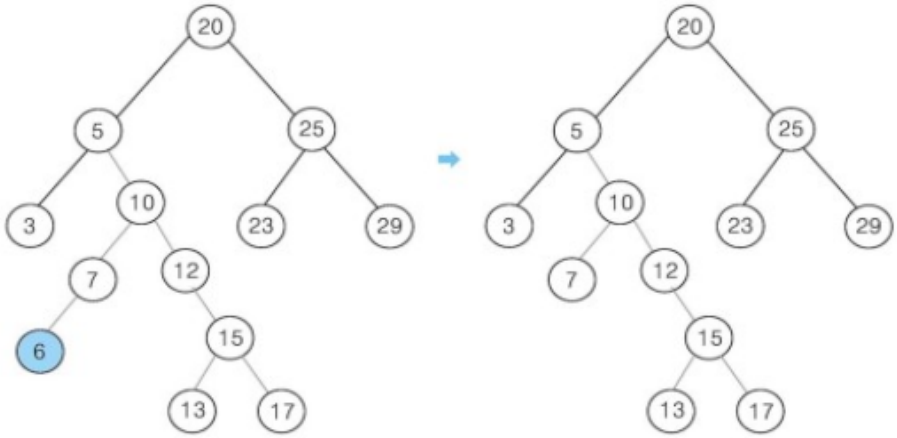
이진 트리의 노드에 대한 간단한 코드 구현

```
public class Node {
    private int data;
    private  Node left;
    private  Node right;


    public Node(int data, Node left, Node right) {
        this.data = data;
        this.left = left;
        this.right = right;

    }

    public int getData(){
        return data;
    }

    public void setData(int data){
        this.data = data;
    }
    public Node getLeft(){
        return left;
```

```
    }
    public void setLeft(Node left){
        this.left = left;
    }
    public Node getRight(){
        return right;
    }
    public void setRight(Node right){
        this.right = right;
    }

}
```

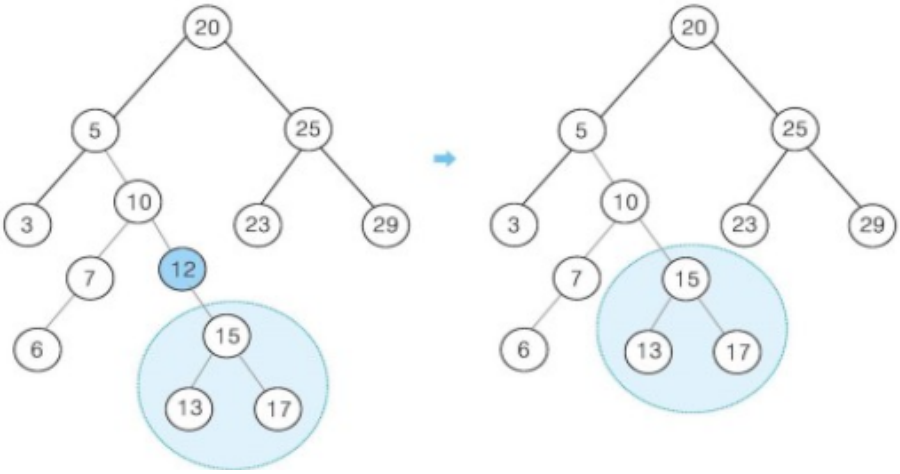
▼ 삭제할 노드가 단말 노드인 경우

부모 노드에서 삭제할 노드를 가리키는 링크를 제거하면 된다



▼ 삭제할 노드의 자식 노드가 하나인 경우

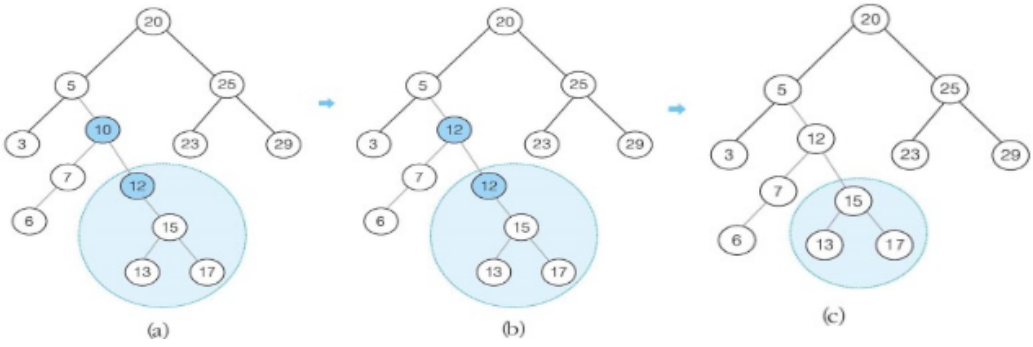
부모 노드에서 삭제할 노드를 가리키는 링크를 삭제할 노드의 자식 노드를 가리킨다.



▼ 삭제할 노드의 자식 노드가 두 개인 경우

-삭제할 노드의 자식 노드가 두 개인 경우는 조금 복잡하다.

-우선 삭제할 노드를 왼쪽 서브 트리에서 가장 큰 노드 또는 오른쪽 서브 트리에서 가장 작은 노드로 대체한다. 그리고 대체된 원래 노드를 삭제한다.



- 자식 노드가 두 개인 데이터 10 노드를 삭제하려면

  데이터 10 노드를 왼쪽 서브 트리에서 가장 큰 노드인 데이터 7 노드 또는 오른쪽

  서브 트리에서 가장 작은 노드인 데이터 12 노드로 대체한다.

- 만약 오른쪽 서브 트리에서 가장 작은 노드인 데이터 12 노드로 대체한다고 가정

  하면 (b)와 같이 나타난다. 그리고 대체된 원래의 데이터 12 노드를 삭제하면 (c)와 같은 결과가 나온다.

• **완전 이진 트리(complete binary tree)**

> 💡 단말 노드를 제외한 나머지 노드가 두 개의 자식 노드를 가지고 있는 트리.