동아대학교

데이터 구조

10주차: Binary-Tree 구현, BST(Binary Search Tree)

박 경 훈

이진 트리 구현을 위한 클래스

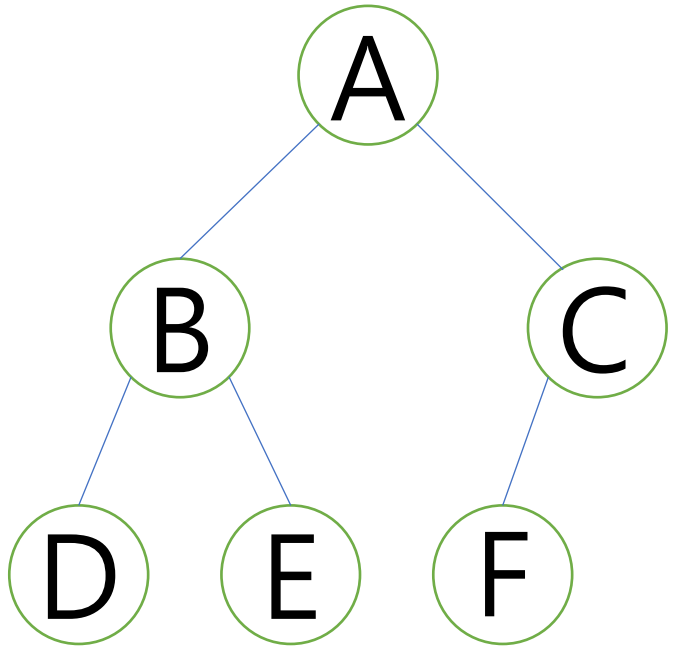

```
class TreeNode:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right
```

이진 트리 구현

※ 노드를 이용한 이진 트리 구현은 복잡
→ 주로 배열 자료형을 이용
→ 레벨이 증가할 수록 비교 해야 할 노드가 기하급수적으로 증가
→ 이전(앞) 노드에 대한 정보가 없으므로 매번 root 노드부터 다시 검색

※ 본 강의에서는 이진 트리 구현을 위한 별도 클래스 구현 하지 않고, Node 구조를 활용하여 직접 트리를 설계하여 이진 트리를 이해하려고 한다.

실습 예제



※ 구현방법
- 왼쪽 자손 노드 부터 생성
- 단말 노드부터 생성

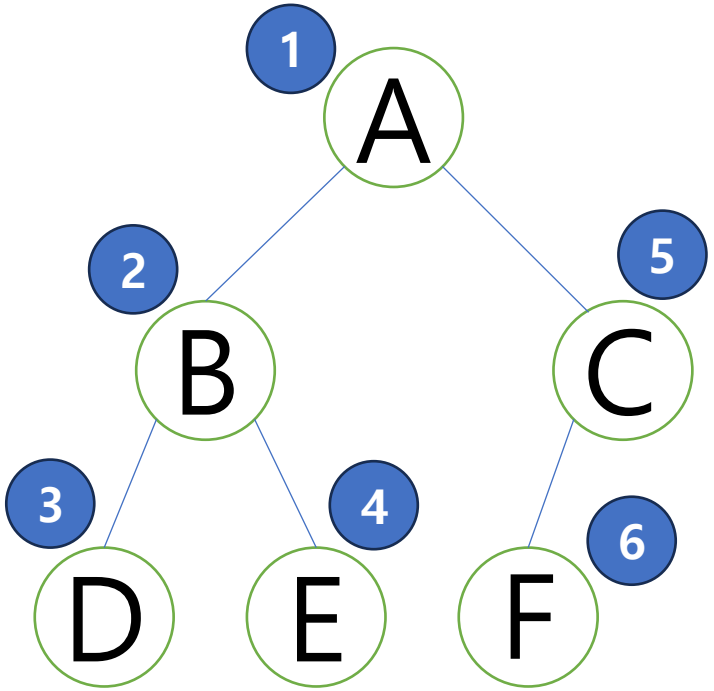
```
d = TreeNode('D')
e = TreeNode('E')
b = TreeNode('B', d, e)

f = TreeNode('F')
c = TreeNode('C', f)

root = TreeNode('A', b, c)
```

실습 예제 – 이진 트리 출력방법

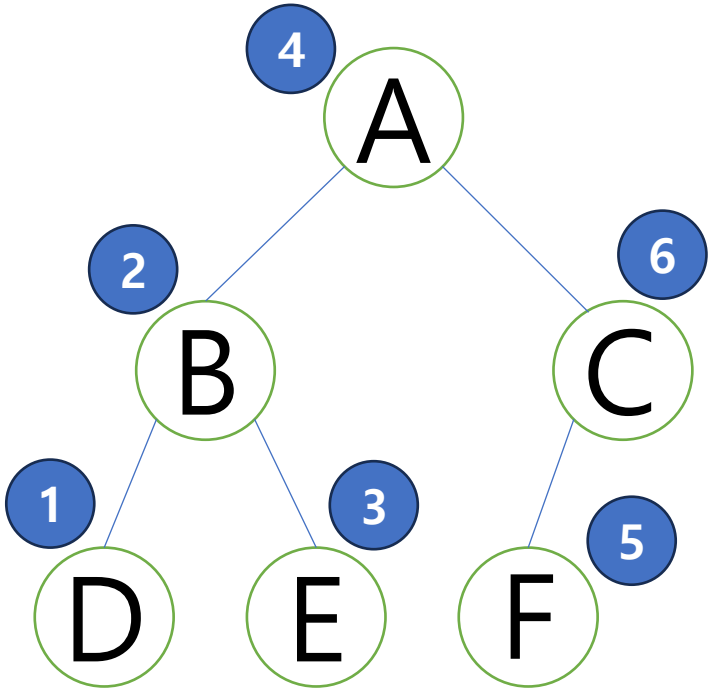
1) 전위순회 출력 (Preorder traversal) : V(root)L(left)R(right)




```
def preorder_recursive(node):
    if node is not None:
        print(node.value, end=' ')
        preorder_recursive(node.left)
        preorder_recursive(node.right)
```

실습 예제 – 이진 트리 출력방법

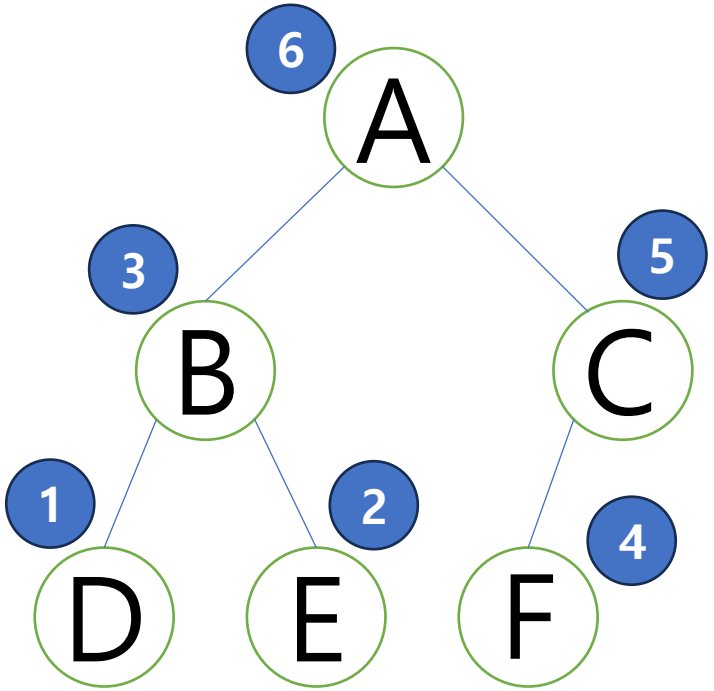
2) 중위순회 출력 (inorder traversal) : L(left)V(root)R(right)




```
def inorder_recursive(node):
    if node is not None:
# 직접 구현해 보세요.
```

실습 예제 – 이진 트리 출력방법

3) 후위순회 출력 (postorder traversal) : L(left)R(right)V(root)




```
def postorder_recursive(node):
    if node is not None:
# 직접 구현해 보세요.
```

실습 예제 – 전체 노드의 수 구하기

- 왼쪽 서브 트리의 노드 수와 오른쪽 서브 트리의 노드 수의 합

```
def count_node(node):
    if node is None:
        return 0
    else:
        return count_node(node.left) + count_node(node.right) + 1
```

실습 예제 – 이진 트리의 높이 구하기

- 좌우 트리의 높이 중에서 큰 값에 1을 더한 값

```
def calc_height(node):
    if node is None:
        return 0
    # 직접 구현해 보세요.
```

이진탐색트리(Binary Search Tree, BST)

● 이진 탐색 트리 속성

- 각 노드는 최대 두 개의 자식 노드를 가짐
- 각 노드의 왼쪽 서브트리에 있는 값은 해당 노드의 값보다 작음
- 각 노드의 오른쪽 서브 트리에 있는 값은 해당 노드의 값보다 큼
- 좌우 서브트리는 모두 위 속성들을 만족해야 함

※ 장점

- 정렬된 순서로 데이터를 정장하기 때문에, 이진 탐색 알고리즘을 사용하여 원하는 값을 빠르게 찾을 수 있음.
- 중위 순회를 수행하면 BST의 모든 노드를 정렬된 순서로 방문할 수 있습니다.
- 데이터의 삽입, 삭제, 검색 등의 동적연산에 대해 효율적 처리 가능
- 삽입 및 삭제 연산을 수행할 때 트리를 재조정할 필요가 없어 자료구조의 확장성 높음

이진탐색트리(Binary Search Tree, BST)

※ 단점

- 특정한 순서대로 데이터가 입력되는 경우에는 트리의 높이가 선형적으로 증가
- 데이터가 랜덤하게 입력되지 않거나, 특정한 순서로 입력될 경우 트리가 불균형
- 트리가 불균형 할 경우에는 균형을 유지하기 위해 추가적인 연산이 필요
- 일부 노드의 삽입 또는 삭제 연산은 트리의 구조를 재조정해야 하므로 복잡성 증가

이진탐색트리(Binary Search Tree, BST) 구현

문제. 이진 트리를 참고하여 이진탐색트리 클래스 구현

- 필요 메소드 : 삽입, 탐색 기능
- 부가기능 : 삽입 기능 구현을 위한 노드 순회, 원하는 데이터 찾기 위한 탐색기능 구현을 위한 순회