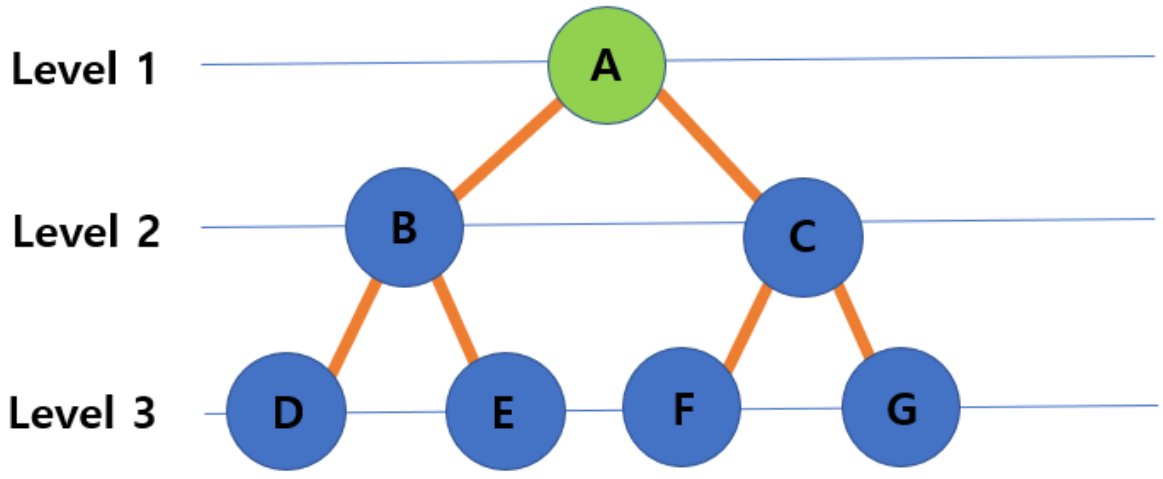
17주차 : 이진트리 / 이진탐색트리

※ 이진 트리란, 각 노드가 최대 두 개의 자식 노드를 가지는 트리 자료구조이다.

- 각각의 노드가 최대 2개의 자식 노드를 가질 수 있는 트리이다.
- 정렬과 검색 알고리즘을 위한 하나의 도구
  - 이진 트리의 모양에 따라 알고리즘의 성능에 차이가 있다.
  - 트리의 형태는 레벨과 노드 수에 따라서 결정된다.

<이진트리 종류>

1. Perfect binary tree 포화 이진 트리



포화 이진 트리

1. 포화 이진 트리는 모든 레벨의 노드가 가득 차있는 트리이다.
   - 노드가 2개의 자식을 가지고 있다.

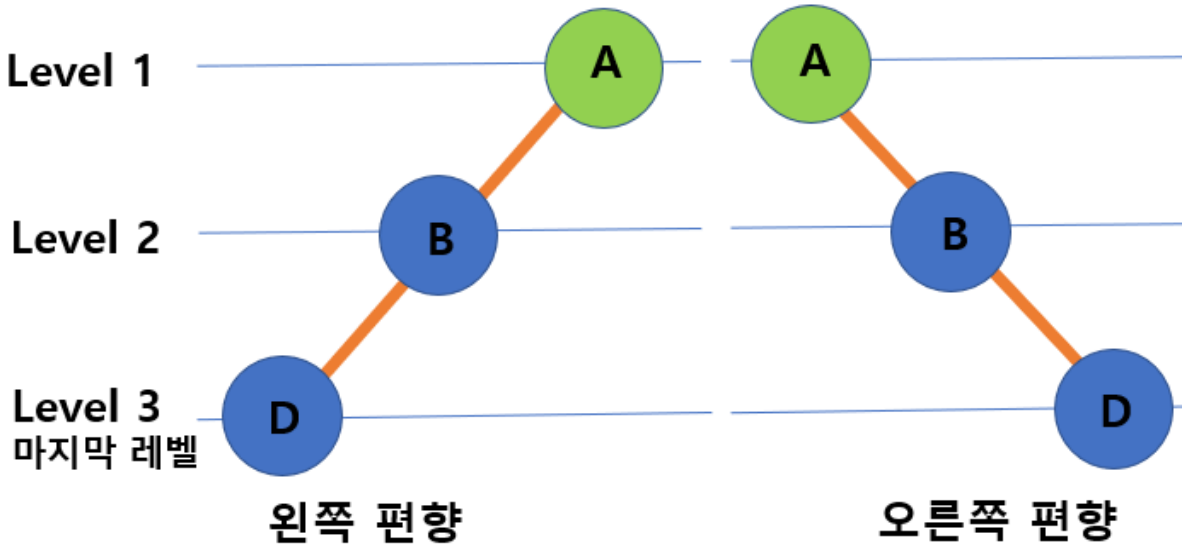
- 차 수(Degree) 가 2 이다.

2. 모든 노드가 가득 차있어 단말 노드부터 루트 노드까지 높이가 같다.
3. 노드의 개수 n = 2^h - 1, h 는 높이

2. Complete binary tree 완전 이진 트리

1. 완전 이진 트리는 마지막 레벨 바로 전까지는 꽉 차있고, 마지막 레벨에서 왼쪽부터 차례대로 채워져 있는 트리이다.
2. 완전 이진 트리의 개념은 힙(heap)과 관련이 있다.
3. 노드의 개수 n < 2^h -1, h 는 높이

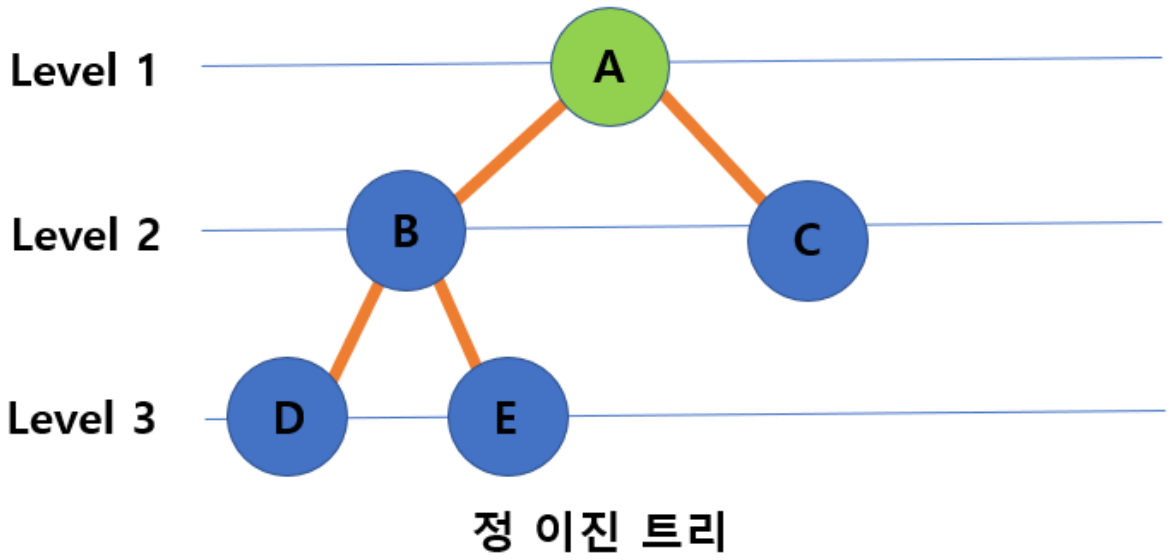
3. Skewed binary tree 편향 이진 트리



1. 왼쪽 또는 오른쪽으로 편향되게 채워져있는 트리이다.

2. 각각의 높이에서 1개의 노드만 있다.
    - 따라서 왼쪽 혹은 오른쪽으로만 편향되게 된다.
3. h <= 노드의 개수 n <= 2^h - 1, h 는 높이
    a. 편향 이진 트리에서 노드의 개수 n은 위 식의 최솟값 h와 같다.
    b. 최댓값은 포화 이진 트리와 같다.

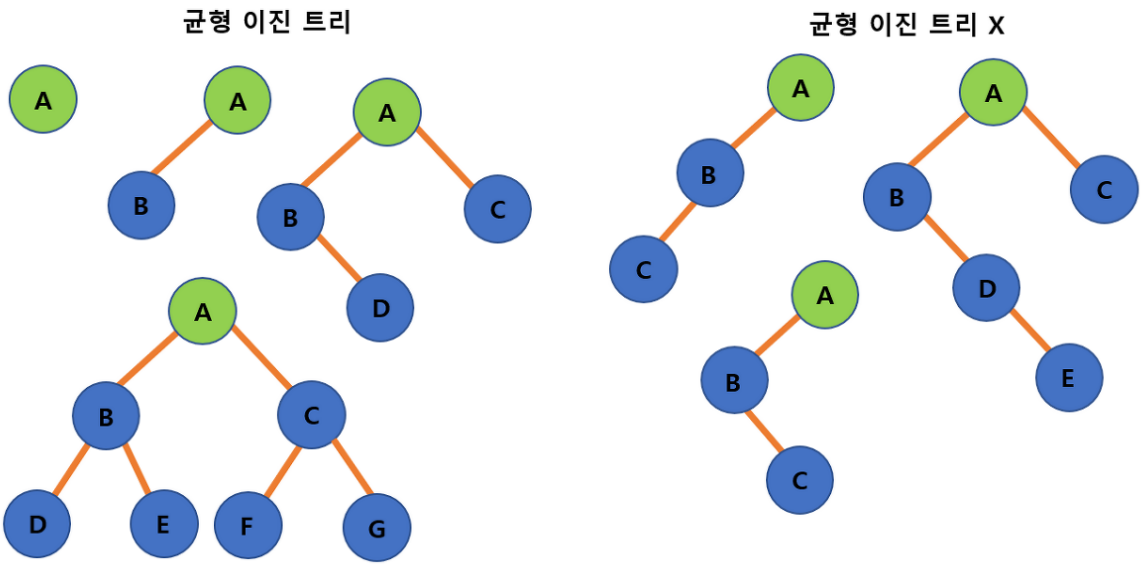
4. Full binary tree 정 이진 트리



정 이진 트리

1. 정 이진 트리는 모든 노드가 0 개 또는 2개의 자식 노드만을 갖는 트리이다.
2. 2 * height + 1 <= 노드의 개수 n <= 2^(height+1) - 1

5. Balanced binary tree 균형 이진 트리



1. 균형 이진 트리는 모든 노드의 왼쪽과 오른쪽 서브 트리의 높이가 2 이상 차이가 나지 않는 트리이다.

<이진 트리 탐색 방법>

1. 전위 순회(Preorder)

전위 순회는 **루트 노드** → **왼쪽 노드** → **오른쪽 노드** 순서로 트리의 노드들을 방문하는 방법.

2. 중위 순회(Inorder)

중위 순회는 **왼쪽 노드** → **루트 노드** → **오른쪽 노드** 순서로 트리의 노드들을 방문하는 방법.

1. 후위 순회(Postorder)

후위 순회는 **왼쪽 노드** → **오른쪽 노드** → **루트 노드** 순서로 트리의 노드들을 방문하는 방법.

※ 이진 탐색 트리란, "왼쪽 자식 노드는 부모 노드 보다 작고, 오른쪽 자식 노드는 부모 노드 보다 큰 이진 트리" 이다.

- 이진 탐색 트리를 중위순회(Inorder Traversal)하면 모든 키를 정렬된 순서로 가져올 수 있다.
- 균형잡힌 트리에서는 O(logN)의 시간복잡도를 갖지만 최악의 경우 O(n)의 시간복잡도를 갖는다.

- 삽입/삭제가 일어나면 시간복잡도를 유지하기 어려워 보통 O(h)라고 통칭한다.(h는 트리의 깊이)
- 자료의 중복을 허용하지 않는다.

> why?
>
> - 검색 목적 자료구조인데, 중복이 많은 경우에 트리를 사용하여 검색 속도를 느리게 할 필요가 없기 때문
> - 트리에 삽입하는 것보다, 노드에 count값을 가지게 하여 처리하는 것이 훨씬 효율적

▼ 이진 탐색 트리 파이썬 코드

```
# 노드 생성과 삽입
class Node(object):
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data


class BinarySearchTree(object):
    def __init__(self):
        self.root = None

    # 노드 삽입
    def insert(self, data):
        self.root = self._insert_value(self.root, data)
        return self.root is not None

    def _insert_value(self, node, data):
        if node is None:
            node = Node(data)
        else:
            if data <= node.data:
                node.left = self._insert_value(node.left, data)
            else:
                node.right = self._insert_value(node.right, data)
        return node

    # 노드 탐색
    def find(self, key):
        return self._find_value(self.root, key)

    def _find_value(self, root, key):
        if root is None or root.data == key:
            return root is not None
        elif key < root.data:
            return self._find_value(root.left, key)
```

```
        else:
            return self._find_value(root.right, key)

    # 노드 삭제
    def delete(self, key):
        self.root, deleted = self._delete_value(self.root, key)
        return deleted

    def _delete_value(self, node, key):
        if node is None:
            return node, False

        deleted = False
        # 해당 노드가 삭제할 노드일 경우
        if key == node.data:
            deleted = True
            # 삭제할 노드가 자식이 두개일 경우
            if node.left and node.right:
                # 오른쪽 서브 트리에서 가장 왼쪽에 있는 노드를 찾고 교체
                parent, child = node, node.right
                while child.left is not None:
                    parent, child = child, child.left
                child.left = node.left
                if parent != node:
                    parent.left = child.right
                    child.right = node.right
                node = child
            # 자식 노드가 하나일 경우 해당 노드와 교체
            elif node.left or node.right:
                node = node.left or node.right
            # 자식 노드가 없을 경우 그냥 삭제
            else:
                node = None
        elif key < node.data:
            node.left, deleted = self._delete_value(node.left, key)
        else:
            node.right, deleted = self._delete_value(node.right, key)
        return node, deleted


# array = [40, 4, 34, 45, 14, 55, 48, 13, 15, 49, 47]
array = [5, 2, 4, 22, 10, 12, 15, 60, 44, 9]
bst = BinarySearchTree()
for x in array:
    bst.insert(x)

print(bst.find(22))  # True
print(bst.find(61))  # False
print(bst.find(60))  # True
print(bst.delete(60))  # True
print(bst.find(60))  # False
print(bst.delete(22))  # True
print(bst.delete(44))  # True
print(bst.find(22))  # False
print(bst.find(44))  # False
```

오늘의 문제

- 균형 (Gold5) : https://www.acmicpc.net/problem/22968

출처

- https://velog.io/@lky9303/이진-탐색-트리
- https://yoongrammer.tistory.com/71
- https://hsc-tech.tistory.com/7