Balanced Binary Tree

- 균형 이진 탐색 트리 (Balanced Binary Search Tree)
- BST의 단점을 보완하기 위해서 만들어진 BST -> 최악의 경우에 O(N)이 될 수 있기 때문.
- 항상 균형을 유지하면서 삽입, 삭제가 진행되는 트리.
- 기본적으로 트리의 오른쪽 서브트리와 왼쪽 서브트리가 같은 개수의 노드를 갖도록.
- 최악의 경우에서도 삽입, 삭제, 탐색 동작에 대해서 모두 O(logN)에 동작.
- 높이 균형이 맞춰진 이진트리.
- 왼쪽과 오른쪽 트리의 높이 차이가 모두 1만큼 나는 트리.

1. AVL (Adelson-Velsky and Landis) Tree

- 스스로 균형을 잡는 이진 탐색 트리

1.1 AVL Tree의 특성

- AVL 트리에서 노드의 두 하위 트리(왼쪽, 오른쪽)의 높이 차이가 최대 1을 넘지 않는다.
- AVL 트리는 엄격하게 균형을 유지하기 때문에 Red-Black Tree 보다 더 빠른 성능을 가지지만 더 많은 작업을 수행해야만 한다.
- 특히 운영체제의 경우 이러한 자료구조에 의존한다.
- 대부분의 연산은 이진탐색트리와 동일하다.
- 삽입 연산의 경우 이진탐색트리와 동일하지만 모든 삽입연산은 트리가 균형을 유지하는지 확인을 해야한다.

2. Red-Black Tree

* Linked List, Binary Search Tree, Balanced Binary Tree

1. Linked List

- 구현이 쉽다.
- 많은 포인터(참조)를 저장한다.
- O(N)의 시간복잡도를 가진다.

2. Binary Search Tree

- 탐색의 O(N) 시간복잡도를 O(logN) 시간복잡도로 줄였다.
- Unbalanced Tree일 경우에 탐색의 속도가 최악의 경우 O(N)이 된다.

3. Balanced Binary Tree(AVL Tree, Red-Black Tree)

- 균형을 이루도록 보장한다.
- 항상 O(logN)의 시간복잡도를 보장한다.

참고