AVL트리는 Binary Search Tree(이하 BST)에서 가장 초기에 Balanced를 제시한 트리이다. BST의 장점은 탐색속도가 빠르다는 것이지만 편향 트리일 경우 단방향 연결리스트의 탐색과 같은 속도를 내기 때문에 BST의 장점을 없애버린다. 그래서 AVL트리는 Balanced를 맞춰서 BST의 장점이 사라지지 않게 하는 것이다.

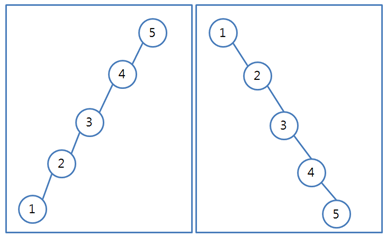
**AVL 트리란?**

AVL트리는 약자 같지만 Adelson-Velskii와 E.M. Landis가 논문을 발표했기 때문에 이름을 따서 AVL트리란 이름이 된 것이다. 각각의 노드마다 왼쪽 서브트리의 높이를 오른쪽 서브트리의 높이로 뺀 값인 균형치(balance factor)를 가지고 있으며, ±1 이하여야 한다. Height Balanced Tree(높이 균형 트리)라고도 한다.

삽입과 삭제를 할 때 트리의 높이가 달라지게 되는데 만약 어떠한 노드라도 균형치가 ±2일 경우 RR, LL, RL, LR의 네 가지 방식을 이용해 회전을 시킨다.

**AVL 트리 장점**

가장 큰 장점은 탐색속도가 빠르다는 것과 트리 전체를 재배열시키지 않아도 트리의 규형이 유지된다.



이진 탐색 트리의 경우 삽입이 무작위로 될 경우 한쪽으로 편향 트리가 될 가능성이 있다. 그럴 경우 탐색 시 최대 비교 횟수가 트리의 노드 개수로 서론에서 언급했듯이 단방향 연결리스트와 다를 바 없다. 하지만 AVL트리는 이미 균형이 이루어져 있기 때문에 탐색이 O(log n)을 넘지 않는다.

트리의 균형은 노드를 삽입이나 삭제할 때 깨질 수 있다. 삽입/삭제 연산을 할 경우 근접한 노드 뿐 아니라 루트까지의 경로에 있는 노드에도 영향을 줄 수 있다. 만약 그 영향으로 균형치가 ±2가 되는 노드가 생긴다면 그 노드의 서브트리들을 회전하여 트리의 균형을 잡는다.

**동작 구조**

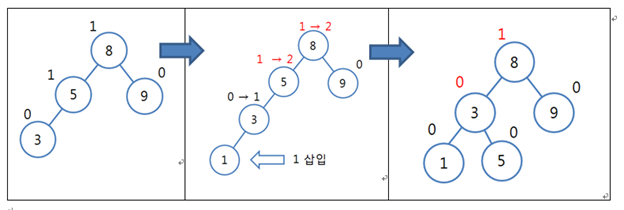
거의 모든 동작구조가 기본적인 이진 탐색 트리와 같다고 볼 수 있는데 노드의 삽입과 삭제는 노드들을 회전해야 하므로 노드를 재배열시키지 않는 이진 탐색 트리와는 차이가 있다.

회전방법에는 LL, LR, RR, RL의 네 가지 방법이 있으며 LL과 RR은 한번만 회전이 필요한 단순회전이고 LR과 RL은 두 번의 회전이 필요한 이중회전이다.

각각의 회전 방식은 새로 삽입된 노드 N으로부터 가장 가까우면서 균형치가 맞는 노드 A일 때

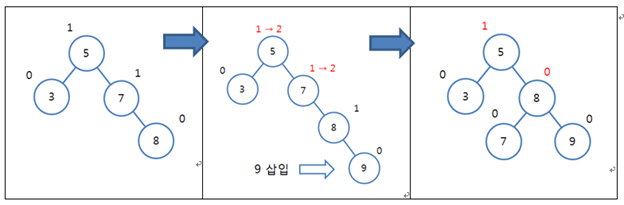
1. LL회전

- N이 A의 왼쪽 서브트리의 왼쪽 서브트리로 삽입되는 경우



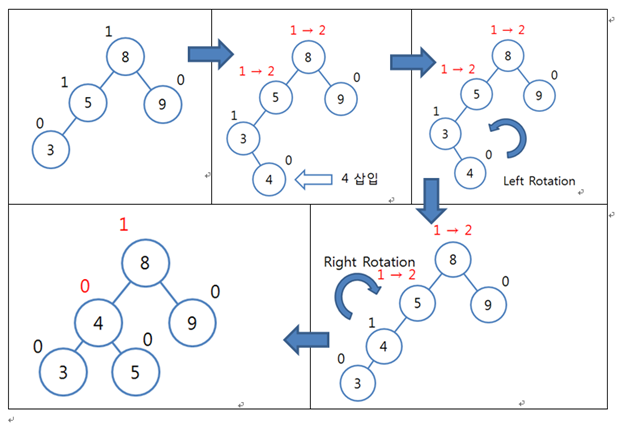
2. RR회전

- N이 A의 오른쪽 서브트리의 오른쪽 서브트리로 삽입되는 경우



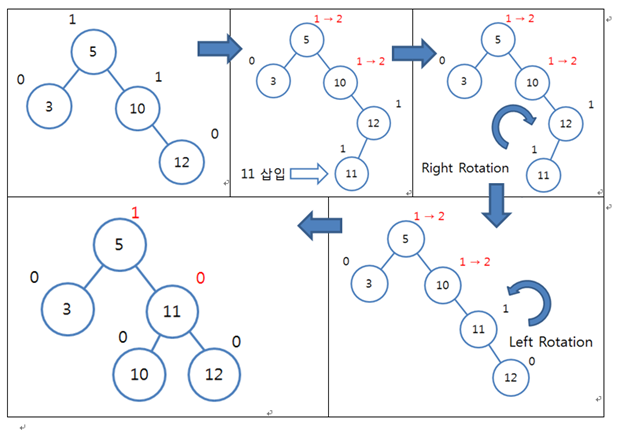
3. LR회전

- N이 A의 왼쪽 서브트리의 오른쪽 서브트리로 삽입되는 경우



4. RL회전

- N이 A의 오른쪽 서브트리의 왼쪽 서브트리로 삽입되는 경우



삭제할 때 역시 동일한 방법으로 회전을 하여 균형을 맞춘다.

**응용 분야**

이진 탐색 트리와 사용방식이나 구조가 같기 때문에 이진 탐색 트리를 사용할 수 있는 분야에선 모두 사용 가능하다. 컴퓨터의 디렉토리 구조나 족보 같은 계층구조를 나타내야 할 때 쉽게 표현할 수있다.

**용어 정리**

1. 균형치(Balance factor)

- 균형치 = 왼쪽 서브트리의 높이 – 오른쪽 서브트리의 높이

2. 트리의 높이(Height)

- 높이 혹은 깊이(Depth)라고 하며 그 트리에 속한 노드의 최대 레벨로 정해진다.

3. 레벨(Level)

- 루트를 레벨1로 정의하기도 하고 0으로 정의하기도 하지만 본 문서에선 1로 정의하도록 하겠다. 만약 한 노드의 레벨이 L이라면 그 자식의 레벨은 L+1이 된다.