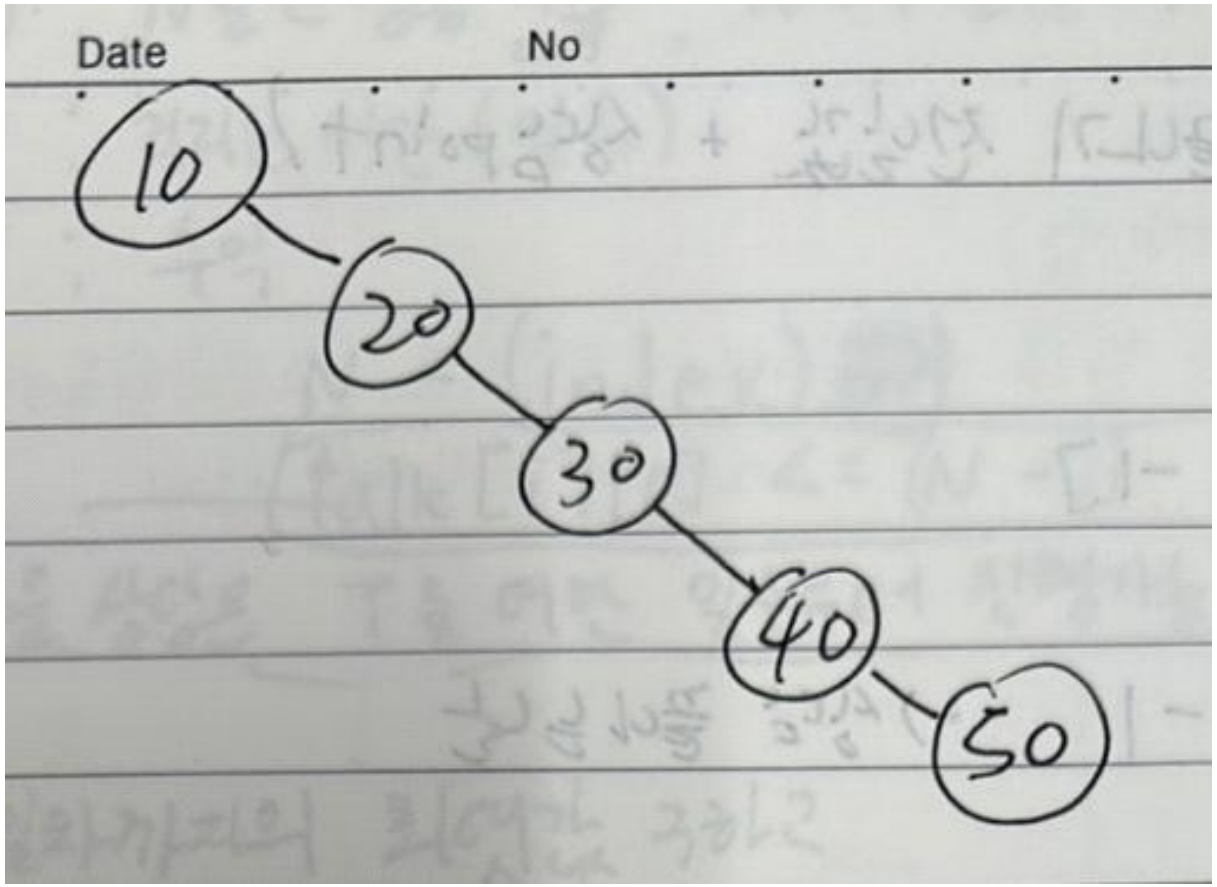
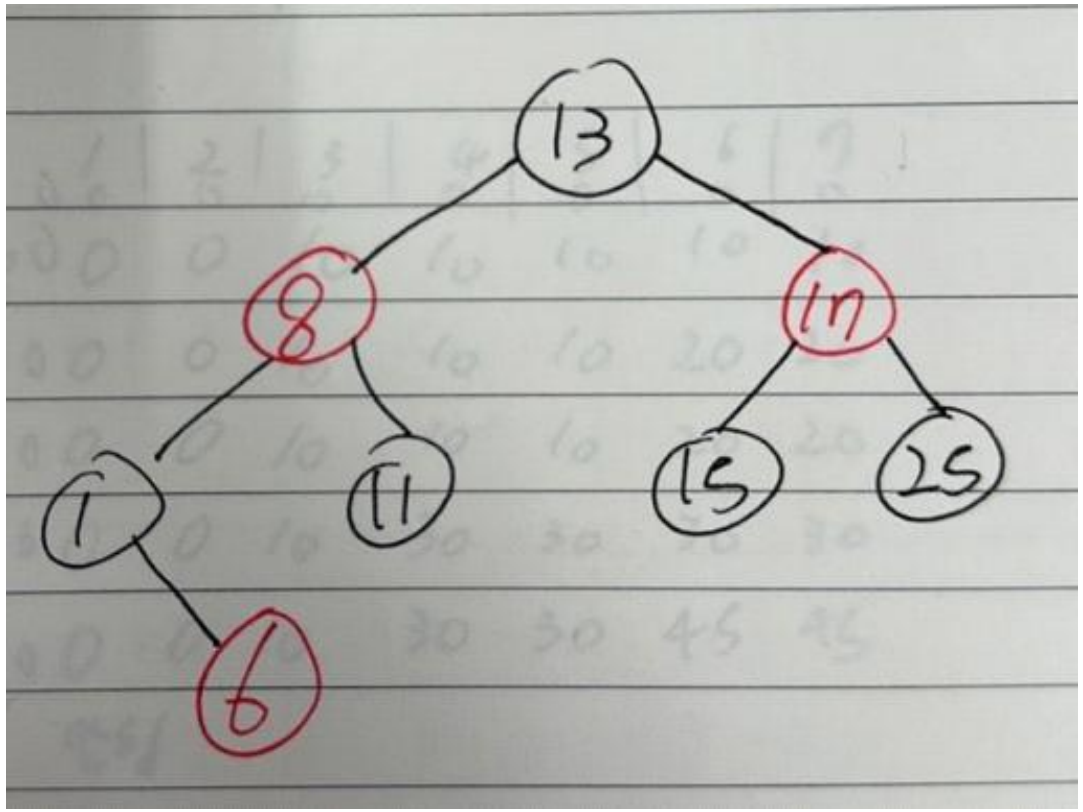
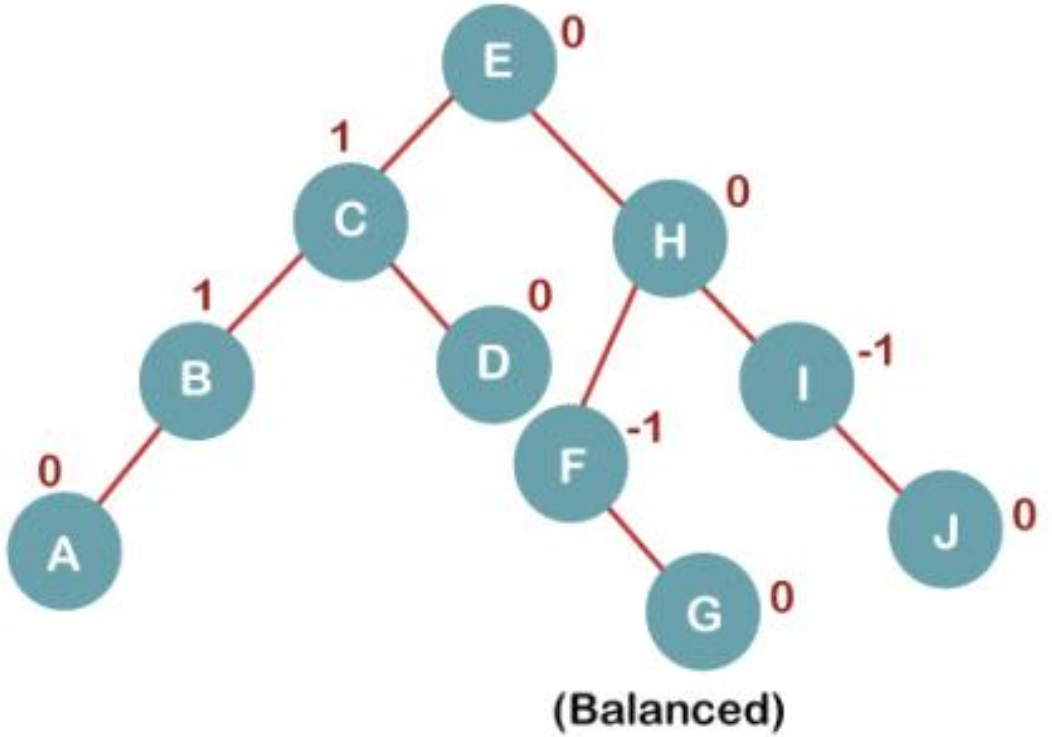


이진 탐색 트리
(Binary Search Tree)

- 임의 노드를 기준으로 left는 작은 값, right는 큰 값이 들어간다.
- 평균 시간복잡도는 O(logN)이다.
- 하지만 순차정렬 된 데이터가 들어올 경우 그림과 같이 편향 트리가 된다.
- 편향 트리일 경우 최악 시간복잡도는 O(N)까지 늘어날 수 있다.





Red Black Tree

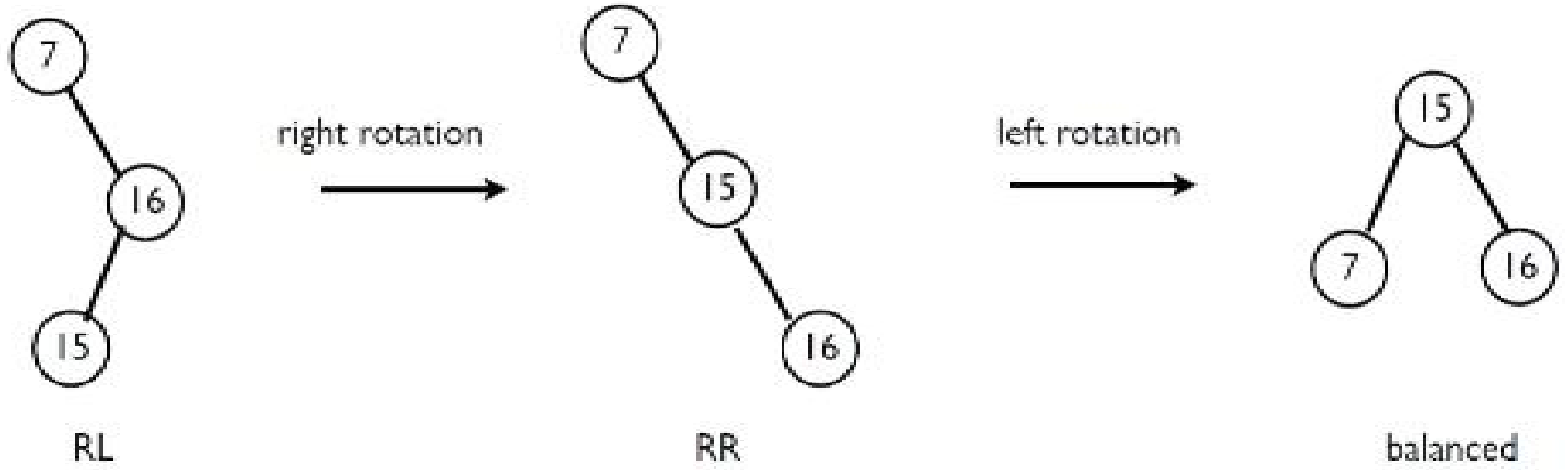
AVL Tree

두 트리 모두 BST의 단점을 개선하여, self balancing을 통해 균형을 유지한다.

왜?

RB Tree일까

최악 시간복잡도 : O(logN)으로 동일
+ AVL Tree는 RB Tree에 비해 훨씬 간단함



회전 연산

AVL Tree는 회전 연산의 빈도가 높기 때문

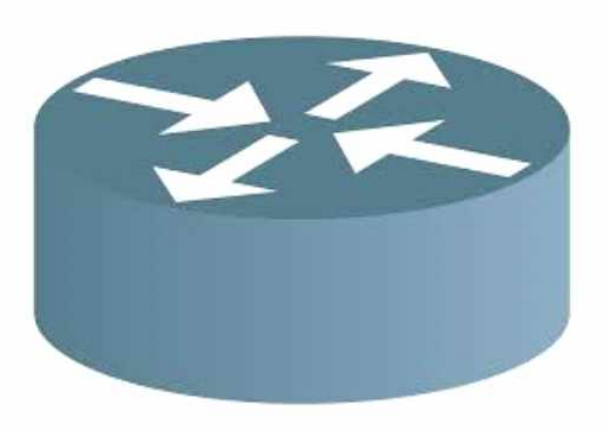
AVL Tree는 hight 차이를 1 이하로 유지해야 한다.

-> RB Tree에 비해 더 많은 회전 연산을 수행
-> 시간복잡도가 같더라도 연산의 빈도가 높아 결과적으로 더 많은 오버헤드를 초래한다.

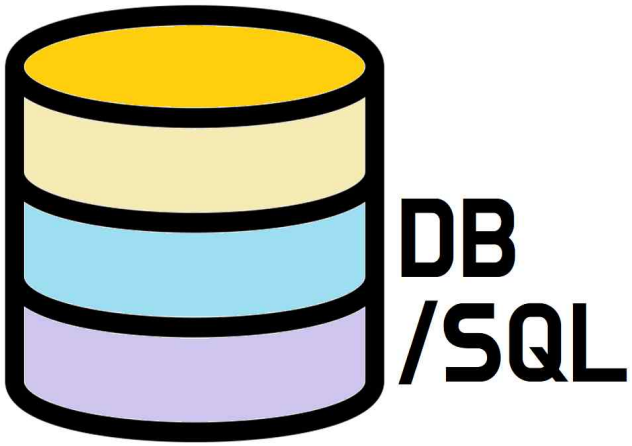
노드의 갯수가 동일하다는 가정 하에,
RB Tree는 AVL Tree보다 최대 2배 높은 height를 가질 수 있다.
(이론적으로)

- 만약 height 차이가 2배라고 가정하면, 삽입/삭제 연산에서
  RB Tree가 더 느릴 수 있다. (탐색 연산의 성능은 동일하다.)

- 그러나 이것은 최악의 경우이고, 일반적으로 AVL Tree와 비
  슷한 height를 가진다.

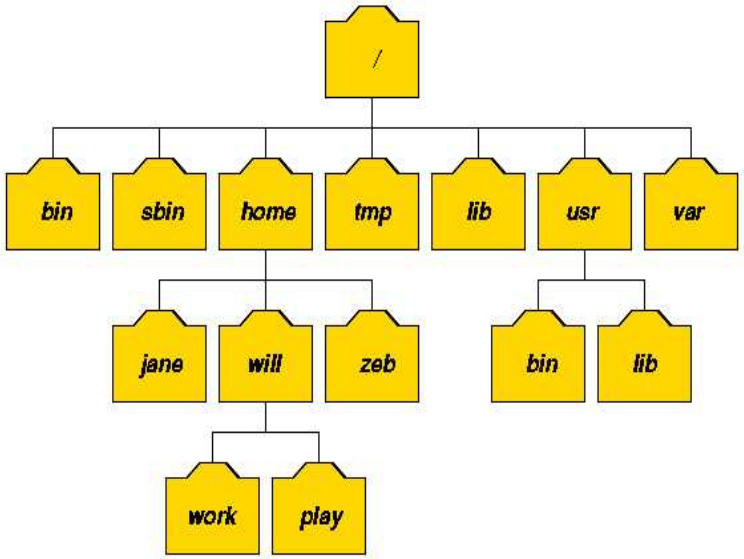


라우터



데이터베이스

RB Tree



리눅스 디렉토리 구조 (그림 출처 http://www.doc.ic.ac.uk/~wjk/UnixIntro/Lecture2.html)

파일 시스템

데이터를 정렬하고 검색하는 경우에 매우 유리하다.

RB Tree를 직접 구현해보며 느낀 것들...

- 서로 다른 자료구조간의 효율 차이가 어디서 나오는 것인지에 대한 이해
- 좋은 자료구조의 필요성에 대한 이해
- 포인터와 메모리 동적 할당/해제에 익숙해지며 컴퓨터를 자유롭게 다룬다는 느낌을 받았음