자료 구조 스터디 05

2021-2 KCA

Binary Search / Binary Search Tree / AVL Tree 맛보기

본 ppt의 자료는 Pt.J님의 자료와 김성열 교수님의 강의 및 여러 블로그를 참고하였음을 밝힙니다.

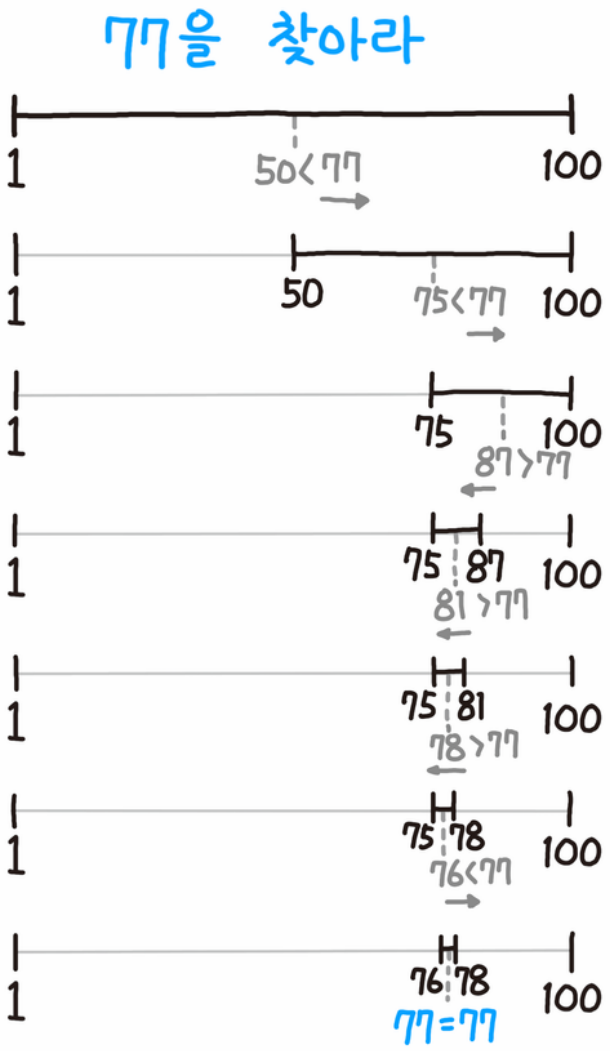
이진 탐색 (Binary Search)

의문 : 어떻게 특정 값을 (빠르게) 찾아낼 수 있을까?

- 이진 탐색 : 정렬된 List에서 특정 값을 찾는 방법

예) updown 게임에서 최소한의 횟수로 정답을 맞히려고 하는 경우를 생각해 보자!!

- 시간 복잡도 : O(logN) 매우매우 빠르다!
- 분할정복의 원리를 따른다.

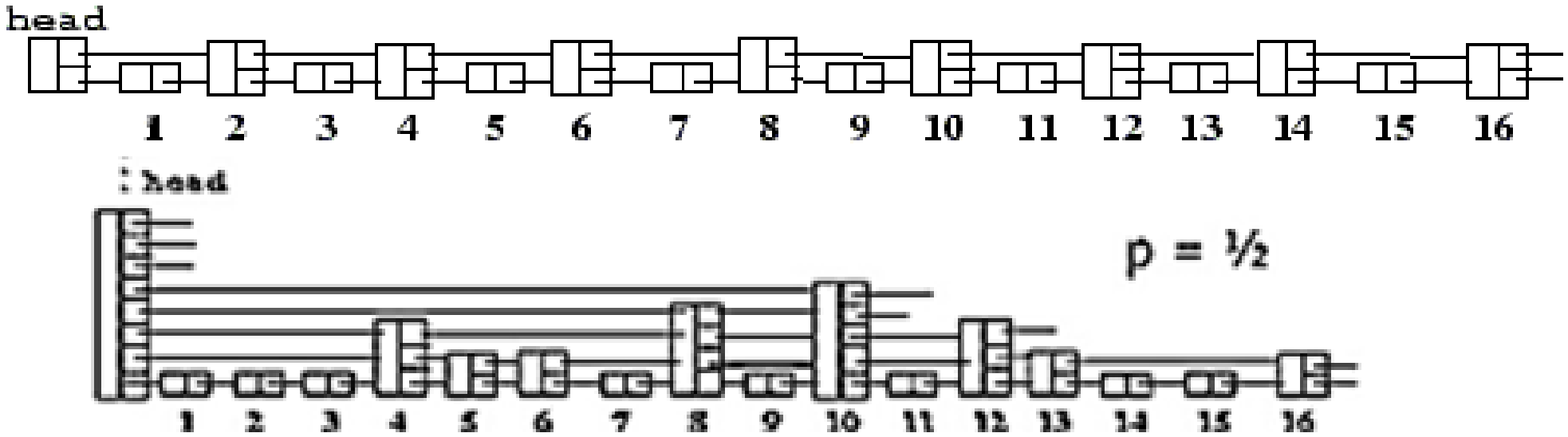


BST (Binary Search Tree) (이진 탐색 트리)

의문 : 연결리스트에서 이진 탐색을 사용할 수 있을까??

⇒ skip list 라는 것이 있음. (연결리스트의 노드가 포인터를 여러 개 가지는 형태이다.)

⇒시간복잡도 : 최상의 경우 logN, 최악의 경우 O(N)



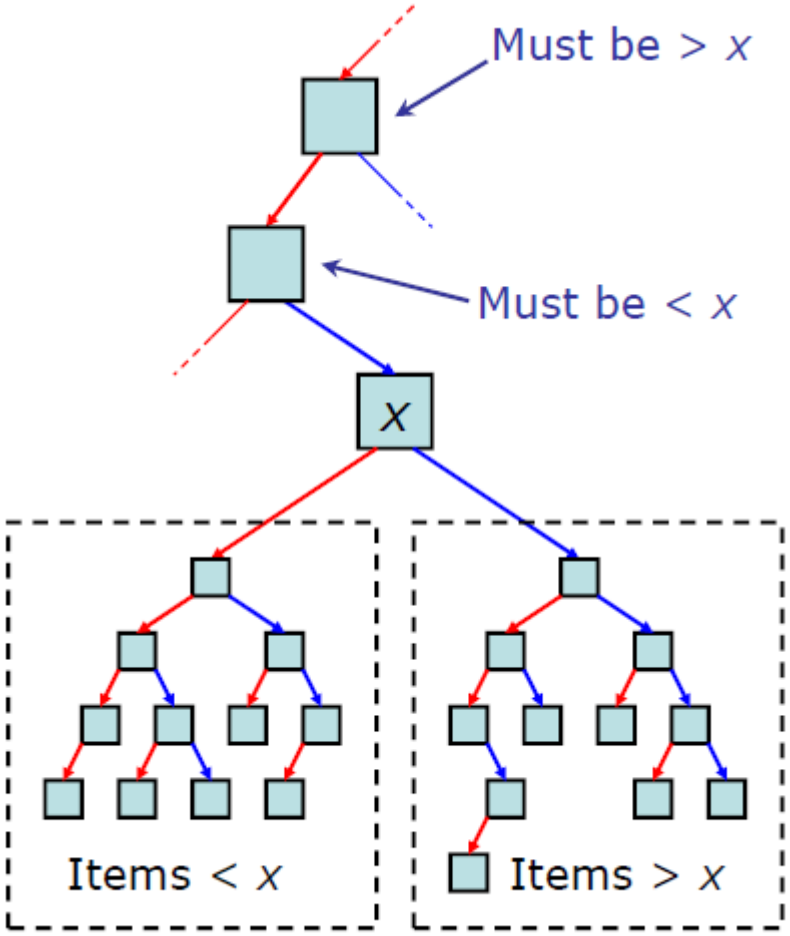
http://pigbrain.github.io/datastructure/2015/07/12/SkipList_on_DataStructure

BST (Binary Search Tree) (이진 탐색 트리)

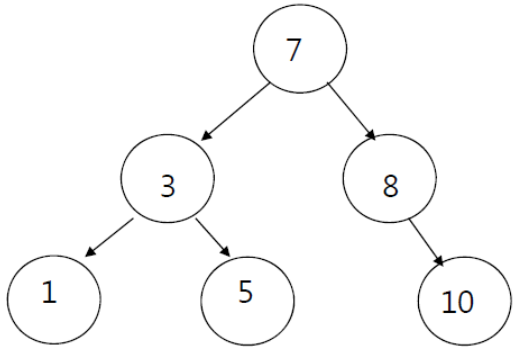
- Binary Search Tree : Tree를 이용해서 이진 탐색을 가능하게 한다.

<규칙 (BST 조건)>

- 각 노드의 왼쪽 서브 트리에는 해당 노드보다 작은 값만 있다.
- 각 노드의 오른쪽 서브 트리에는 해당 노드보다 큰 값만 있다.
- 중복된 값은 존재하지 않는다.



- 시간복잡도 : O(Height) (트리의 높이에 영향)



https://ratsgo.github.io/data%20structure&algorithm/2017/10/22/bst/

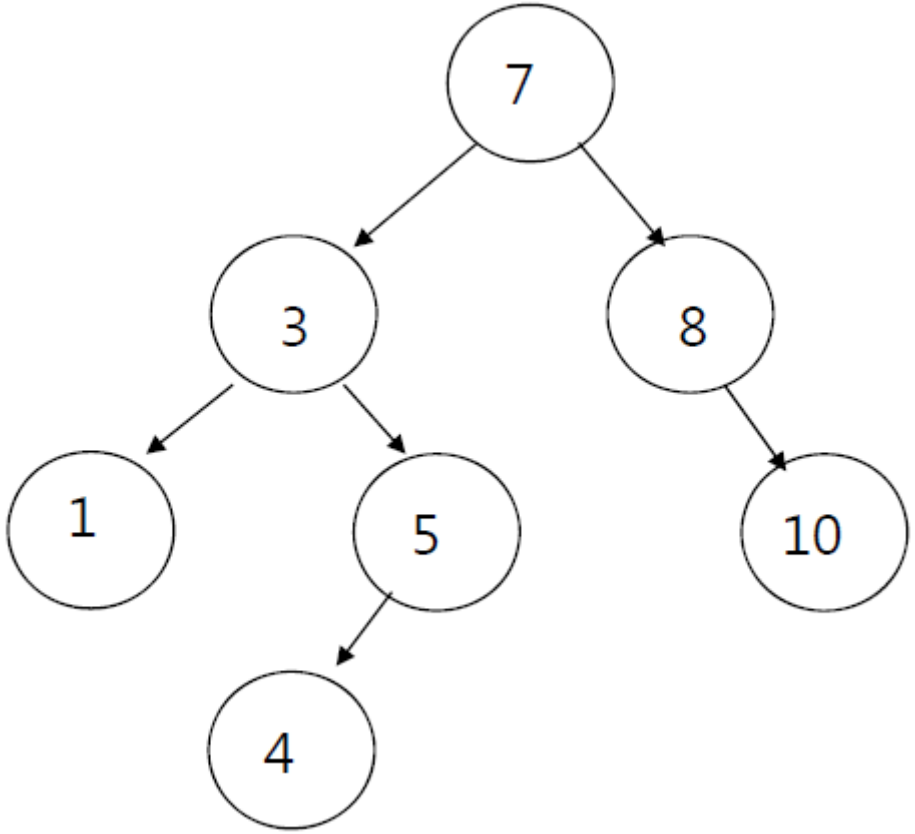
BST Search, Insert

- Search

예) search 10

7, 8, 10

비교하면서 아래로 내려간다.



- Insert

예) insert 4

7, 3, 5, insert

search 후 없다면 그 위치에 Insert.

단, leaf 노드에 insert 한다. (만약 중간 노드에 insert 할 경우, BST 조건을 다시 검사해 주어야 한다.)

- O(Height)

https://ratsgo.github.io/data%20structure&algorithm/2017/10/22/bst/

BST Delete

- Delete (BST 조건이 성립하는지 따져 주어야 한다)

case1. 삭제할 노드의 자식 노드가 없는 경우

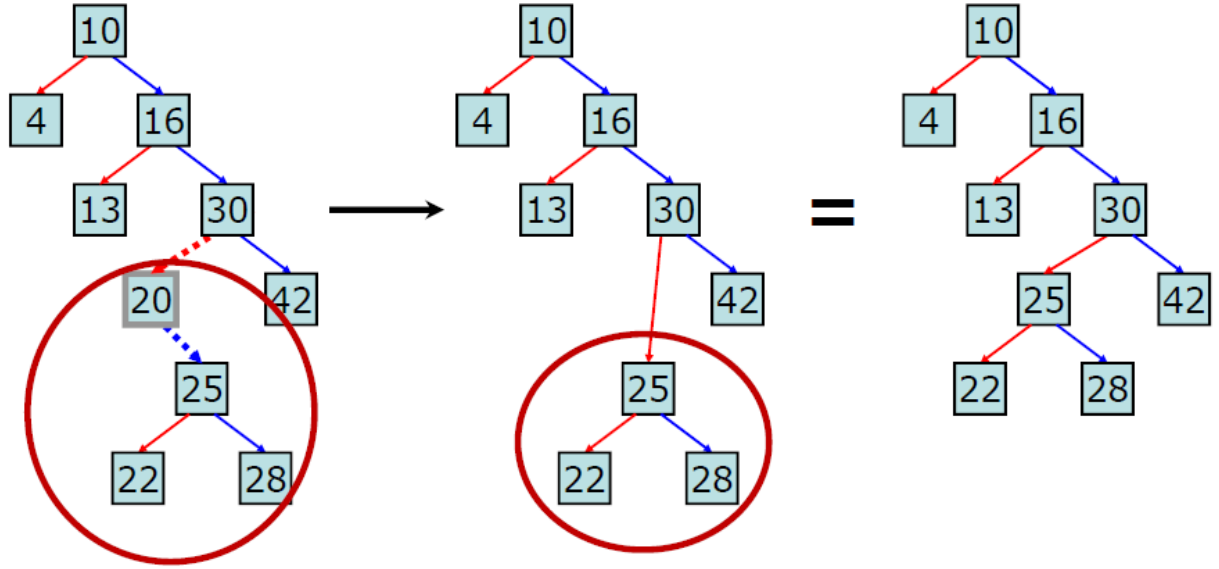
⇒ 바로 삭제 가능.

case2. 삭제할 노드의 자식 노드가 하나 있는 경우

⇒ 삭제하고,
자식 노드를 포함한 서브 트리를 삭제한 위치에 연결.

예) 오른쪽 그림에서, 20 삭제

20을 포함한 서브 트리는 모두 30보다 작기 때문에 BST 조건 성립!



https://ratsgo.github.io/data%20structure&algorithm/2017/10/22/bst/

BST Delete

case3. 삭제할 노드의 자식 노드가 두 개 있는 경우

예) 16 삭제.

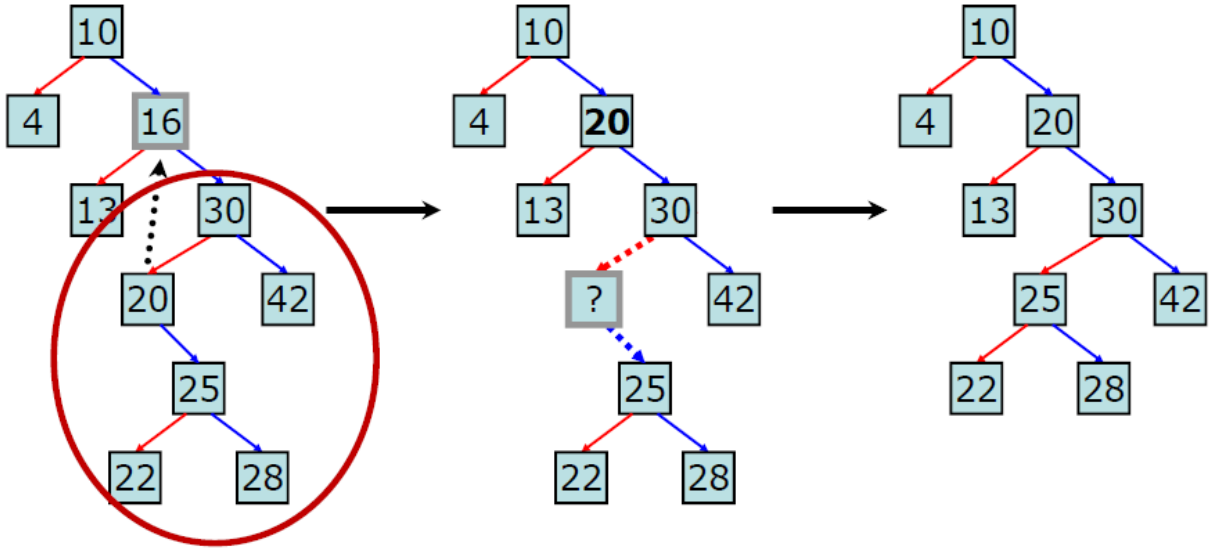
16과 20의 관계에 주목하자!!

16을 삭제하려면, 16의 위치에 어떤 노드를 가져와서 BST 조건을 성립하게 만들어야 한다.

30의 왼쪽 서브 트리의 노드를 x라 하면, 모든 x에 대해, 16 <x< 30 만족.

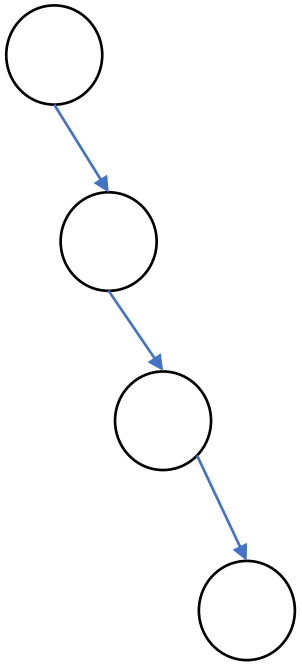
즉, 16을 삭제하고, x 중 하나를 가져오면 BST 성립!

(그 이후는 재귀적으로 구현)



https://ratsgo.github.io/data%20structure&algorithm/2017/10/22/bst/

AVL Tree (Adelson-Velsky and Landis) (발명자 이름)



- BST의 문제점 : 시간복잡도가 트리의 높이에만 영향을 받는데, 트리의 높이가 노드 수에 비해 클 때가 있다.

- 트리 높이를 줄인다 == 탐색 속도가 더 빨라진다.

- **AVL 트리** : 노드를 삽입/삭제 할 때, 트리 높이를 최대한 낮게 유지시키는 트리

(BST 조건을 따르는 BST 트리이다.)

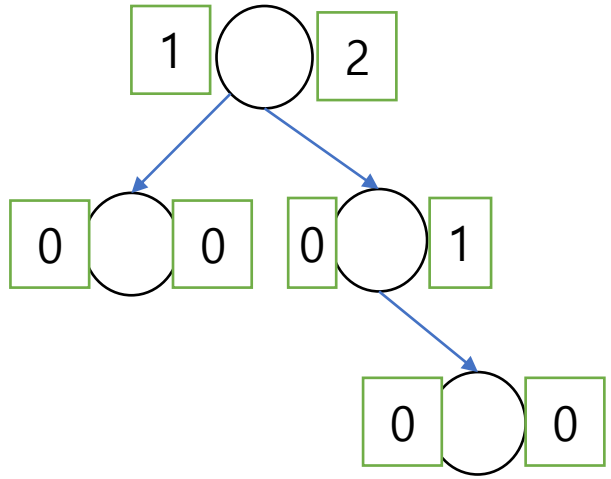
BST 트리를 균형을 맞추려면, root를 바꾸어야 한다. (이를 '보정' 이라 부름)

보정해야 하는 상황인지 어떻게 (빠르게) 판단 가능한가?

AVL Tree (Adelson-Velsky and Landis) (발명자 이름)

- 각 노드에 자신의 왼쪽, 오른쪽 서브트리의 높이를 나타내는 Label을 붙인다. (L, R 이라 명명)

(양쪽 높이의 차이를 나타내는 Label 하나만 붙일 수도 있음)



- 양 Label의 차이가 1이 초과되면 '보정' 한다.

<AVL Tree 조건>

- 모든 노드에서 |L-R| < 2 를 만족한다.

N개 노드의 AVL Tree 높이는 O(logN) 이다.

시간 복잡도 : O(logN)

구현해보면 좋을 것.

- Binary Search
- BST

- 각종 정렬
  삽입 정렬, 선택 정렬, 버블 정렬 / 머지소트, 퀵소트