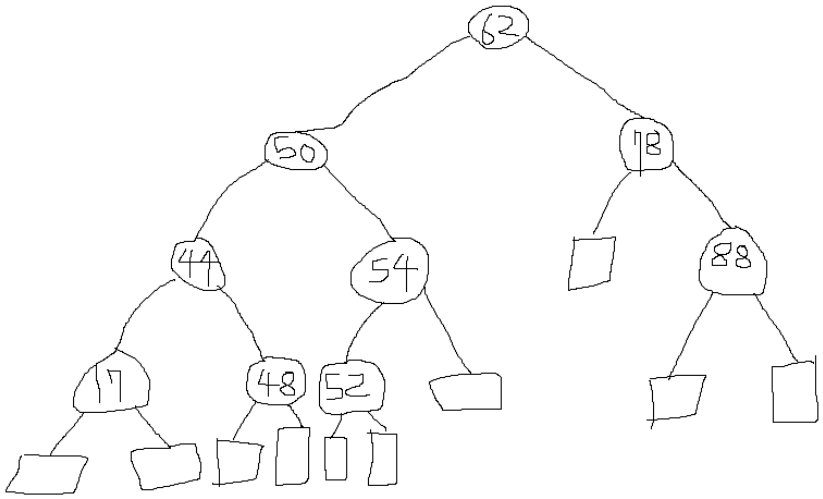
데이터 구조 Assn8

무은재학부
20180551 이준석
POVIS ID : ljs9904ljs

Problem 1 (R-10.8)

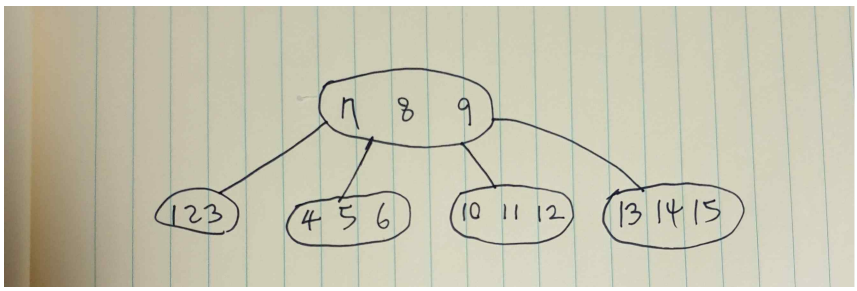


Problem 2 (R-10.15)

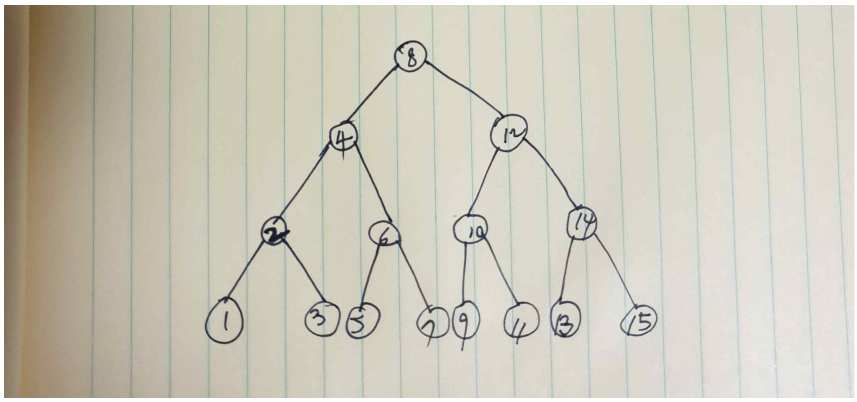
자식 node를 최소 2개부터 최대 4개까지 가질 수 있다.
자식 node의 수는 부모 node가 갖고 있는 key의 개수보다 하나 더 많으므로 부모 node의 key가 되도록 많아야 가장 적은 수의 node를 사용하여 만들 수 있을 것이다. 즉, 3개의 key를 갖는다.

반면에 가장 많은 수의 node를 갖게 하기 위해서는 key를 최대한 적게 갖도록 해야한다. 즉, 1개만 갖는다.

a. 가장 적은 수의 node



b. 가장 많은 수의 node

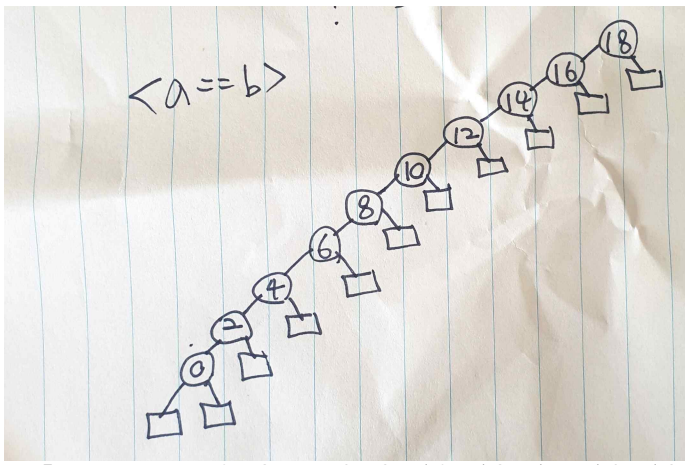


Problem 3 (R-10.20)

Perform the following sequence of operations in an initially empty splay tree and draw the tree after each set of operations.

after each set of operations 이므로 operation 단위가 아닌 operation set 단위로 그려야 한다. 즉, 0부터 18까지 insert한 최종 결과물과 1부터 19까지 search한 최종 결과물과 0부터 18까지 delete한 최종 결과물을 그리면 된다.

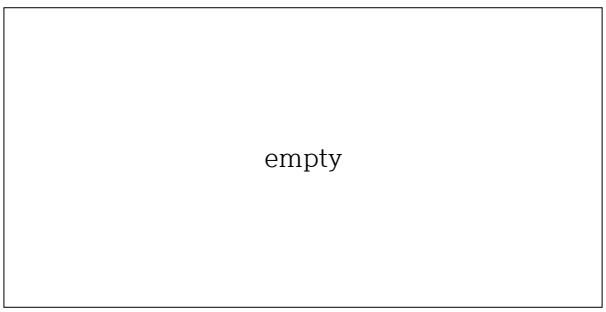
아래 사진은 a와 b 둘 다를 나타낸 것이다.



a. Insert keys 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, in this order.
b. Search for keys 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, in this order.

b에서 search 하는 것들이 전부 다 존재하지 않는 key이므로 18을 계속 splay한다. 그래서 a와 같은 모습을 갖는다.

c에서, 트리 안에 존재하는 모든 key 값들에 대한 delete가 이루어지므로 트리는 비게 된다.



Problem 4 (C-10.6)

dictionary는 같은 key를 갖는 여러 개의 entry를 허용한다.

-삽입에 대해서,

1. 새로 삽입되는 node가 기존의 node와 같은 key를 가질 때, 먼저 삽입된 것이 inorder traversal에서 우선적으로 탐색된다고 가정한다.
2. 서로 같은 key를 가지는 node들 중에서 가장 마지막에 삽입된 node를 v라고 하자.
3. 새로 삽입되면서 기존의 node와 같은 key를 가지는 node를 w라고 하자.

v의 오른쪽 자식 node가 없다면,
같은 key를 가지는 것은 그것과 같은 key를 가지는 node의 오른쪽 자식으로 추가하면 된다.

v의 오른쪽 자식 node(혹은 subtree)가 있다면,
v의 오른쪽 자식 node(혹은 subtree)의 부모를 w로 변경한다. (v의 오른쪽 자식 node(혹은 subtree)는 w의 오른쪽 자식 node(혹은 subtree)가 된다.)
v의 오른쪽 자식 node 자리에 w를 넣는다.

-제거에 대해서,

BST의 method 중에서, key를 이용해서 erase하는 것이 있다. 이 때, 그 key를 가지는 모든 node들을 제거하도록 변경하거나 key를 이용해서 제거할 수 없도록 변경해야 한다. 이런식으로 변경하지 않으면, (dictionary는 동일한 key를 가지는 여러 개의 node가 존재할 수 있기 때문에) 제거한 key를 가지는 node가 존재할 수도 있다.

-찾기에 대해서,

BST의 method 중에서, key를 이용해서 find하는 것이 있다. 단순히 그 key를 가지는 node 하나만을 찾는 것이 목적이라면 그대로 사용해도 무방하다. 하지만 그 key를 가지는 모든 node를 찾는 것이 목적이라면 그 key를 갖는 첫 node부터 마지막 node까지 다 찾아내도록 변경해야한다. 같은 key를 갖는다면 오른쪽 자식으로 추가되도록 삽입을 구성하였으므로 계속 오른쪽 자식을 탐색하면서 마지막 node까지 탐색할 수 있다. 현재 가리키고 있는 node와 그 node의 오른쪽 자식 node의 key값이 다를 때를 종료 조건으로 설정(현재 node와 오른쪽 자식 node의 key가 같은 동안에 true)하고 while문을 이용한다면 같은 key를 갖는 node들의 처음부터 마지막까지 탐색할 수 있다.

Problem 5 (C-10.20)

AVL tree의 balance 조건을 완화시켜서 만든 것이 바로 red-black tree이다. 그러므로 일단 AVL tree를 red-black tree로 바꾸는 것은 가능할 것으로 보인다.

혹은

AVL tree의 모든 원소를 하나씩 빼내서 red-black tree에 삽입하는 것도 방법이다.

1) 트리 t1과 t2가 있다고 하자.
2) t1은 AVL tree의 특성을 모두 만족한다고 하자.
3) t1의 원소들 중에서 임의의 원소 하나를 골라서 빼낸다. AVL tree의 특성을 만족시키도록 t1에서 restructuring 과정을 거친다.
4) t1에서 빼낸 원소를 t2에 저장한다. 그리고 red-black tree의 특성을 만족시키도록 t2에서 restructuring 혹은 recoloring 과정을 거친다.
5) t1이 완전히 빌 때까지 3번과 4번 과정을 반복하면 AVL tree가 red-black tree가 된다.

따라서,
AVL tree의 restructuring, red-black tree의 restructuring과 recoloring 3가지 모두가 어떤 경우에도 가능하다면, AVL tree는 red-black tree로 어떤 경우에도 변환 가능하다.