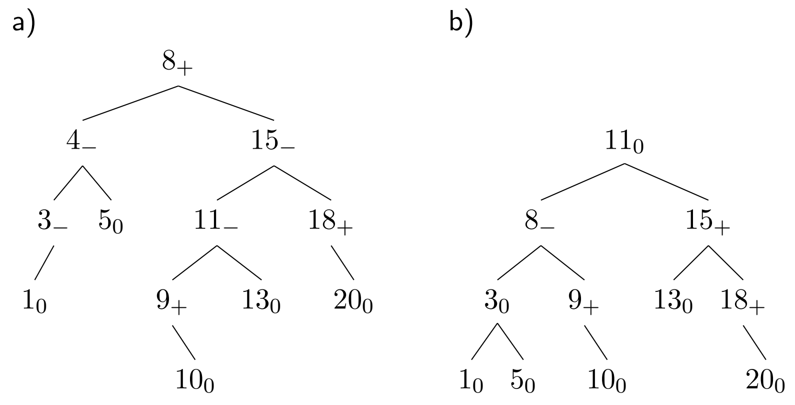
CSC263 - Assignment 3

Cristyn Howard

Tuesday, Februrary 13, 2018

Question 1



.

Question 2

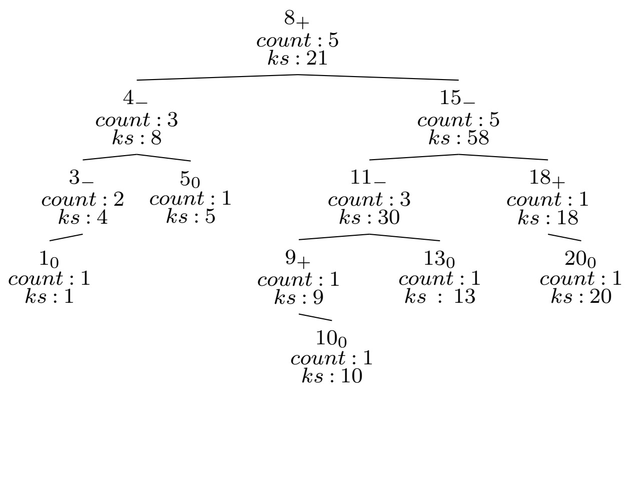
a) Let each integer $i \in S$ serve as the key for a node in an AVL tree.

Each node x additionally stores:

- x.count : number of nodes in the subtree rooted at x whose keys are less than or equal to x.key (x included)
  $|R|$, *where* $[i \in R \subset S] \leftrightarrow [(i \in x.subtree) \wedge (i.key \leq x.key)]$
- x.keysum : sum of keys of nodes in subtree rooted at x that are less than or equal to x.key (x included)
  $\sum i.key \;\; \forall i \in x.subtree \subset S \mid i.key \leq x.key$

Note that for a node z with no children, we have z.count = 1 and z.keysum = z.key.

Here is an example, taken from question 1a:



b) ADD(i)

1. create node x with key i, then insert x as in normal AVL tree (BST insert, rebalance)
   - Whenever rebalancing means that a subtree $y.subtree$ (i.e. subtree rooted at y) becomes the new **left** child of node z, z and all ancestor nodes of z with keys is greater than or equal to $z.key$ must be amended so that: $zOrAncestorOfZ.keysum+ = y.keysum, zOrAncestorOfZ.count+ = y.count$
2. go up ancestor nodes of x in it's final position until root is reached, and for each ancestor node whose key is greater than or equal to x.key, add x.key to the keysum of the ancestor and increment the ancestors count attribute by 1.

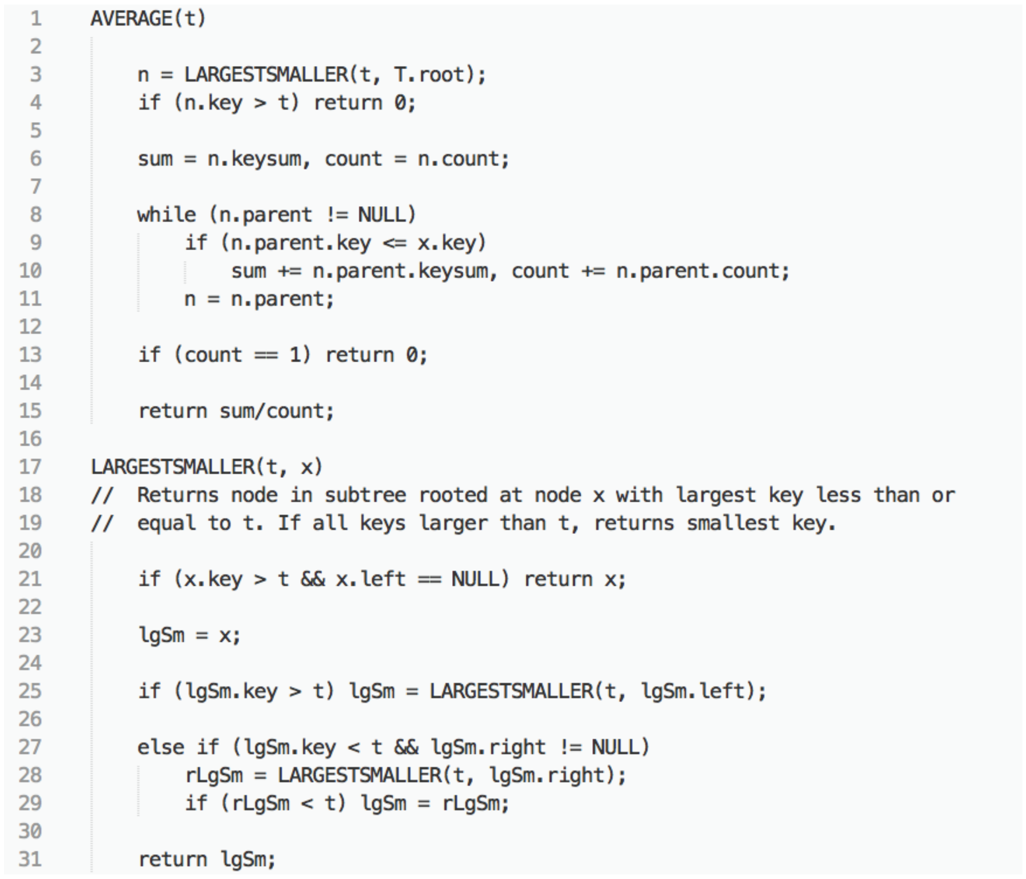
- ADD $\in O(logn)$ because the number of constant time operations (adjusting counts and keysums of ancestors after rebalancing or adding node) is limited by the height of the tree, which for an AVL tree with n nodes is $\lfloor logn \rfloor$.

AVERAGE(t)

1. find the node x in the AVL BST whose key is the largest integer in S that is less than or equal to t (return 0 if none in $S < t$)
2. start with x's keysum and count
3. go up ancestor nodes until root is reached, and for each ancestor node whose key is less than or equal to x.key, add that ancestor's keysum and count to the working total keysum and count
4. when root is reached, if count > 1 (i.e. if there is at least one node in T with key <= x.key) divide keysum by count and return the result as the average

- AVERAGE $\in O(logn)$ because the number of constant time operations is limited by the number of nodes between x and the root of the tree, which for an AVL tree with n nodes is at most $\lfloor logn \rfloor$.

Here is some psuedocode for AVERAGE:



```
AVERAGE(t)

    n = LARGESTSMALLER(t, T.root);
    if (n.key > t) return 0;

    sum = n.keysum, count = n.count;

    while (n.parent != NULL)
        if (n.parent.key <= x.key)
            sum += n.parent.keysum, count += n.parent.count;
        n = n.parent;

    if (count == 1) return 0;

    return sum/count;

LARGESTSMALLER(t, x)
//  Returns node in subtree rooted at node x with largest key less than or
//  equal to t. If all keys larger than t, returns smallest key.

    if (x.key > t && x.left == NULL) return x;

    lgSm = x;

    if (lgSm.key > t) lgSm = LARGESTSMALLER(t, lgSm.left);

    else if (lgSm.key < t && lgSm.right != NULL)
        rLgSm = LARGESTSMALLER(t, lgSm.right);
        if (rLgSm < t) lgSm = rLgSm;

    return lgSm;
```