

CSSE230 Exam 1 SRT Review Session

The exam will cover the following topics:

- Through day 21, HW7, and **EditorTrees** milestone 3.
- Binary trees, including BST, AVL, indexed (**EditorTrees**)
- Traversals and iterators, size vs. height including height-balanced trees, rank
- Recursive tree methods. One programming question will be a method that should only touch each node once for efficiency (like sum of heights and isHeightBalanced from HW5)
- Hash tables
- Heaps – basic concepts; not on programming part

Section 1. True or False:

Answer the following questions by circling either (T) rue or (F) alse. A statement is only considered true if it is **always** true, and a statement can be considered false if there is a **single counterexample**.

- | | | |
|----------|----------|--------------------------------------------------------------------------------------------------------------------------|
| T | F | The array in a binary min heap is always in ascending sorted order. |
| T | F | Fixing an imbalance in a height balanced tree, after inserting a value and an imbalance is found, is $O(1)$. |
| T | F | After insertion in an Editor Tree, you need to recalculate the height at some nodes to determine the new balance code |
| T | F | In a hash table, the purpose of quadratic probing is to solve the problem of clustering that occurs with linear probing. |
| T | F | Inserting a node in the right subtree of node N in an AVL tree may cause the rank of N to change. |
| T | F | A single right rotation at node N in an AVL tree may cause the rank of N to change. |

Section 2. Binary Trees:

Given a binary tree of **height** 4, answer the following questions:

What is the minimum number of nodes in the given tree: **5**

What is the maximum number of leaf nodes in the given tree: **16**

What is the minimum number of nodes given the tree is balanced: **12**

Section 3. Tree Methods:

Consider a tree, whose nodes have the following fields:

```
char element;  
Node left;  
Node right;  
int rank;  
Code balance;
```

Write a method that returns the total number of non-null left subtrees.

NOTE: Node p could be null or NULL_NODE

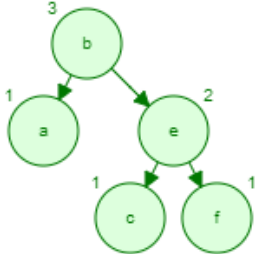
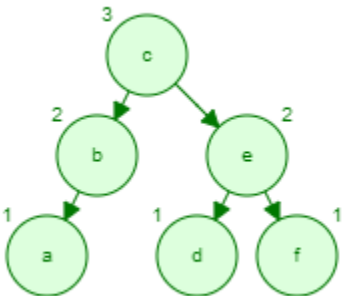
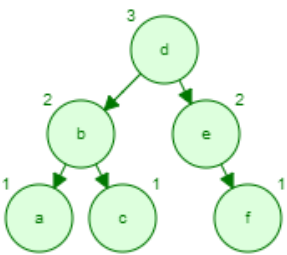
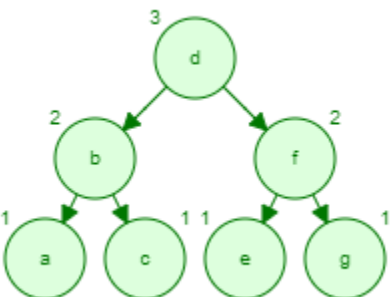
```
public int numLeftSubtrees(Node p) {  
    if (p == NULL_NODE || p == null)  
        return 0;  
    if (p.left == NULL_NODE)  
        return leftSubtrees(p.right);  
    return 1 + leftSubtrees(p.right) + leftSubtrees(p.left);  
}
```

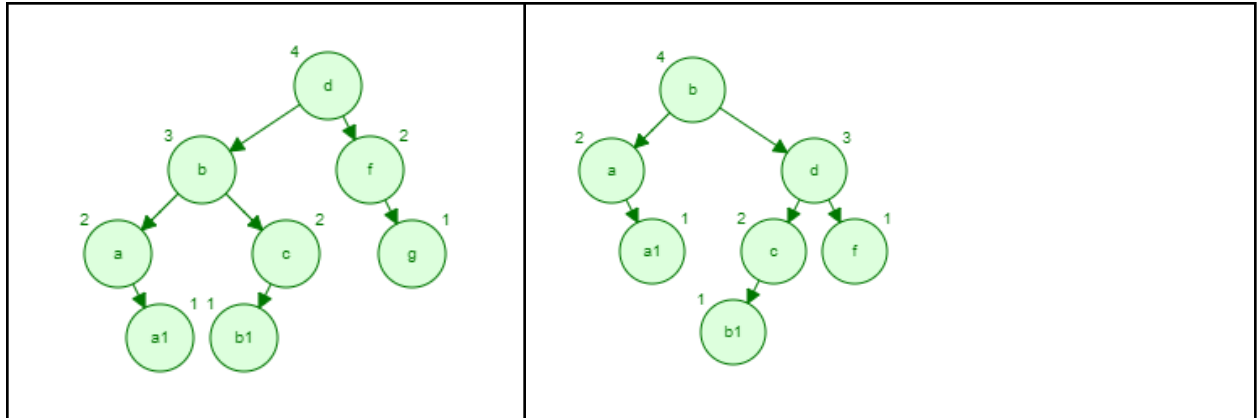
As an interesting aside, there is a glaring flaw with this approach (as in you would likely lose

points for it on homework). What is it? **This methods is recursively running outside the Node class!!!**

Section 4. AVL Tree Operations:

For each of the following trees + operations, draw what the tree should look like after the operation is complete.

On this tree, insert 'd' at position 3	Draw your result:
	
On this tree, insert g at position 6	Draw your result:
	
On this tree, delete 'g'	Draw your result:

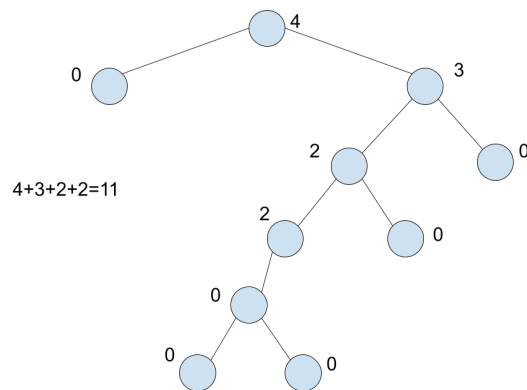


Section 5. Programming

On the provided git repo, you will find a java project that contains three programming problems for you complete, they are as follows:

Total Imbalance

Given a BST, find the total imbalance of the tree. Defined as the sum of the positive difference between every left and right subtree for every node in the tree.



Number of Collided Entries

In the hash package, implement the method `int numCollidedEntries()`. We simplified the implementation of the `StringHashSet` to use linear probing and to not grow the array and re-hash entries. You are to return the number of entries that must have collided with at least one other item when inserted. You can determine it by writing code to examine the internal hash array using knowledge of how linear probing works.

Note: do not re-run `add()` or `contains()` to determine this –changing either of those methods or adding fields to the class is not allowed. You may refer back to your implementation of the `StringHashSet` assignment as needed.

PreOrder + InOrder → Tree

In HW4, you were given the preorder and inorder traversals of a tree, and asked to determine the shape of the tree. In this problem you are to turn that informal process into actual code to automatically build a tree based on these two traversals. Complete this method in the `PreorderInorderBuild` class: