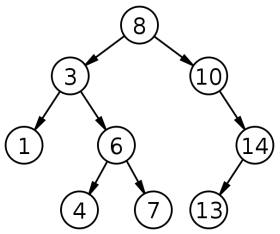
Anastasiia Halchenko

COMP232 – Data Structures & Problem Solving
Fall 2024

Homework #4
Binary Trees
[45 points]

Note: Answer #1 and #2 here. When you have completed all parts of this homework, save this document as PDF by creating a folder named "noncode-answers" in your git repository for this homework assignment, then push your code to GitHub. Complete the rest of the questions in Eclipse and push to GitHub.

1. Answer the questions below with respect to the following binary tree using the definitions from the text:



a. What value is at the root of the tree? 8
b. What values are in leaves of the tree? 1, 4, 7, 13
c. What values are in internal nodes of the tree? 8, 3, 6, 10, 14
d. What values are descendants of 3? 1, 6, 4, 7
e. What values are ancestors of 7? 6, 3, 8
f. What is the depth of node 6? 1
g. What is the height of the tree? 3 ?
h. What is the path length from 8 to 4? 3

i. List the node values in the order they would be visited by:

i. An in-order traversal. LRoR
1, 3, 4, 6, 7, 8, 10, 13, 14

ii. A pre-order traversal. RoLR
8, 3, 1, 6, 4, 7, 10, 14, 13

ii. A post-order traversal. LRRo
1, 4, 7, 6, 3, 13, 14, 10, 8

iii. A level-order traversal.
8, 3, 10, 1, 6, 14, 4, 7, 13

2. Assuming the variable `node` refers to a `BTNode` and that the tree is always sufficiently large, give a statement that identifies the `node's`:

a. leftmost grand child.

```
BTNode<K, V> leftmostGrandchild = node.left.left;
```

b. great grand parent.

```
BTNode<K, V> greatGrandParent = node.parent.parent.parent;
```

c. left child of right sibling, assuming `node` is a left child.

```
BTNode<K, V> leftChildRightSibling = node.parent.right.left;
```

3. Complete the implementation of the `MinKeyFinder` class in the `hw04` package. The `No3Tests` class contains tests that you can use to check your implementation of this class.

4. Implement the `main` method in the `MinKeyFinder` class in the `hw04` package so that it creates a tree with at least 10 nodes and uses a preorder traversal with your visitor to print the key associated with the minimum value in the tree.

5. Complete the implementation of the `visitInOrder` method in the `CS232LinkedBinaryTree` class in the `hw04` package. The `No5Tests` class contains tests that you can use to check your implementation of this method.

6. Complete the implementation of the three-argument constructor for the `CS232LinkedBinaryTree` class in your `hw04` package. The `No6Tests` class contains tests that you can use to check your implementations of this constructor.

7. Complete the implementation of the `contains` method in the `CS232LinkedBinaryTree` class in the `hw04` package. The `No7Tests` class contains tests that you can use to check your implementations of this method. For full credit, implement `contains` so that it does not rely on the `get` or `getNodeFromSubtree` methods. Instead, add a helper method `subtreeContains` that returns a `boolean` value. This will be similar to the `getNodeFromSubtree` method. Then call

`subtreeContains` from `contains`. Yes, this introduces repeated and unnecessary code, but the point here is not producing the most effective class, but for you to practice writing methods like this.

8. Complete the implementation of the `add` method in the `CS232LinkedBinaryTree` class in the `hw04` package. The `No8Tests` class contains tests that you can use to check your implementations of this method.

9. Complete the implementation of the `countLeafNodes` method in the `CS232LinkedBinaryTree` class in the `hw04` package. The `No9Tests` class contains tests that you can use to check your implementations of this method.