Data Structures II - HW5

Instructions:

1. Download the hw5-files.zip file from the COL submission folder for HW5. Unzip the file into the src folder that is inside the workspace you created for this class (as you have done on previous homeworks). If you are unsure where that is, inside Eclipse, right click the src folder and select properties (at the bottom).

2. Start up Eclipse. You will probably need to refresh the project before you see the hw5 folder. (Right click on the src folder in the Project Explorer and select refresh.)

3. Open the hw5 package and you will see two files:
    - HW5Test – the JUnit test harness for the assignment
    - TwoThreeTree – The java source file with the methods you must implement.

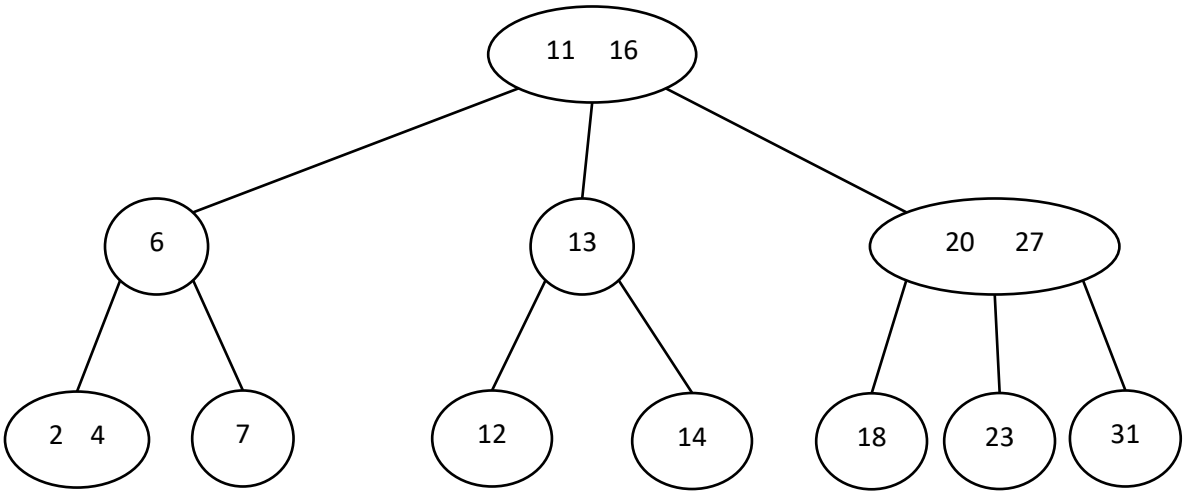
4. Your task is to implement 2-3 trees as explained in the book and in class. The methods you must implement are:
    - boolean contains(int k) – Returns true if k is in the tree and false otherwise
    - void add(int k) – Adds k to the tree. If k is already in the tree, it does nothing
    - height() – Returns the height of the tree
    - nodeCount() – Returns the number of nodes in the tree

5. The methods contains, height, and nodeCount should be fairly straightforward. They should work very similarly to their BinaryTree versions. You just have to take into account that some nodes are 2-nodes while others are 3-nodes. Note that a tree will never have any 4-nodes when this method is called so your implementation does not have to handle the 4-node case.

6. The add method will be significantly more complicated than the other methods. Here are some tips:
    - Since you almost always add a new key to a pre-existing leaf, it's probably best/easiest if you make a leaf the base case rather than null.
    - The only time you add doesn't put the key in a pre-existing leaf is when the entire tree is empty. Handle this case in the public method. (In other words, check if the tree is empty in the public method and handle that case directly in the public method. Only call the helper method if root is not null.)
    - When you add a key to a pre-existing leaf, that may turn the leaf into a 4-node. However, the leaf cannot fix the problem. You can only split a 4-node when you are at the parent of a 4-node. SO…
        - Allow your recursive helper method to return a 2-3 tree that possibly has a 4-node at the root (but no other 4-nodes)
        - This means that whenever your code calls the recursive helper method, it must check if the answer (temp) is a 4-node. If it is, then you are the correct place to do the split.

- **Draw pictures!** Doing the split is a bit tedious and it is easy to make a mistake. So draw before and after pictures of the split. These should look very much like the figures "Insert into a single 3-node", "Insert into a 3-node whose parent is a 2-node", "Insert into a 3-node whose parent is a 3-node", and "Splitting the root" in https://algs4.cs.princeton.edu/33balanced/
  Note that when you insert into a 3-node whose parent is a 3-node, you don't have to worry about splitting the parent. The parent will temporarily be a 4-node, and it is OK to return a tree whose root is a 4-node! When that 4-node is returned to its parent, its parent will split it. This is probably the most confusing part of the assignment, so rewatch the class discussion on it and ask clarifying questions if you are unsure.

7. The TwoThreeTree.java file you are given contains some methods to help with testing. The method identical checks of two trees are identical (have the same keys int the same location). There is also a constructor that takes a String that is a level order traversal of the tree you want to construct. A comma separates nodes and a space separates two keys that are in the same node. For example, suppose you want to construct the tree:



   You can use:
   `TwoThreeTree t = new TwoThreeTree("11 16,6,13,20 27,2 4,7,12,14,18,23,31");`

   This means you can also write tests for the easy methods even if you don't have the add method finished yet!

8. Give yourself plenty of time, especially for the add method. It will take you a while to get something coded up and even longer to debug it.

9. When you are finished, submit your TwoThreeTree.java file as well as a screenshot of what happens when you run HW5Test.java. This assignment does not have any written questions.

Coding Restrictions:

You may not make use of any classes provide by Java or by the book in your solution. You may add additional methods to both the TwoThreeTree class as well as the Node class nested inside the TwoThreeThree class. For example, you might add a "isLeaf" method to the Node class since your add method will probably be asking if a node is a leaf. However, you are not required to do this if you don't find it helpful. You might also find it helpful to have a toString method so that you can "see" the nodes / trees more easily when you run in the debugger.

Submission:

Your submission should include the following files:

- TwoThreeTree.java
- A screenshot of what happens when you run HW5Test.

Grading:

Each test in the HW5Test file has a number indicating how much that test is worth. Your score on HW5 is the sum of the points for all the tests that passed.
Note that the tests for add count for 40 points. You definitely need to work on it to get a good grade. But please take my advice, and finish the other methods first. They are significantly easier.

A Final Plea:

This assignment is definitely challenging. I suspect the majority of students will struggle with it. Get started early, don't give up, and keep asking questions. It is crucial that you get started early so that you can see where you are stuck and can ask questions in class. It will be extremely difficult for me to help you on this assignment via email. You want to be able to ask questions in class and you won't know what to ask until you start working on the assignment.