Due : Oct. 13 (10/13), 13:00

Overview

This assignment consists of one part: implementing a generic k-ary tree.

General Notes

- Do not use Eclipse. We recommend Sublime Text (Linux/Mac/Windows), Atom (Linux/Mac/Windows), Notepad++ (Windows), TextWrangler (Mac), or VSCode (Linux/Mac/Windows).
  - Code that contains the line "package ..." at the beginning of the file breaks our autograder and Eclipse automatically adds that line.
- Do not change any method or class signatures. You should only edit inside of the functions. If your code changes any class or method names or signatures, you will receive an automatic 0. You should not implement any other functions or instance variables besides the ones that are provided, unless explicitly allowed.
- Make sure your code compiles. Non-compiling code will automatically receive a 0. If you have a problem that is causing you to not be able to compile, it may be better to just comment out the incorrect code and return a dummy value (something like null or -1) so the rest can compile.
- Make sure that your code does not print out anything (there should be no System.out.println in your code). You will receive an automatic 0 if your code outputs something to STDOUT during the tests.
- To ensure that your code will be accepted by the autograder, you should submit your code on YSCEC, download it again, unzip it, recompile it and check the provided test suite. This way, you know that the file you are submitting is the correct one.

Introduction

Trees are a generic data structure that show up in many applications. Later in the semester you will implement some special types of trees, but in this assignment you will simply be implementing a k-ary tree. As you learned in the lecture, a tree is made up of many nodes, one of which is specified as the *root* node. Each node has pointers to its children. In a k-ary tree, each node can have up to k children. A special type of k-ary trees are *binary trees*, where $k = 2$.

k-ary Tree

In this assignment, you will implement several operations for generic k-ary trees, and one operation that is specific to binary trees. You **are** allowed to add your own instance variables and methods, but you may not change existing ones.

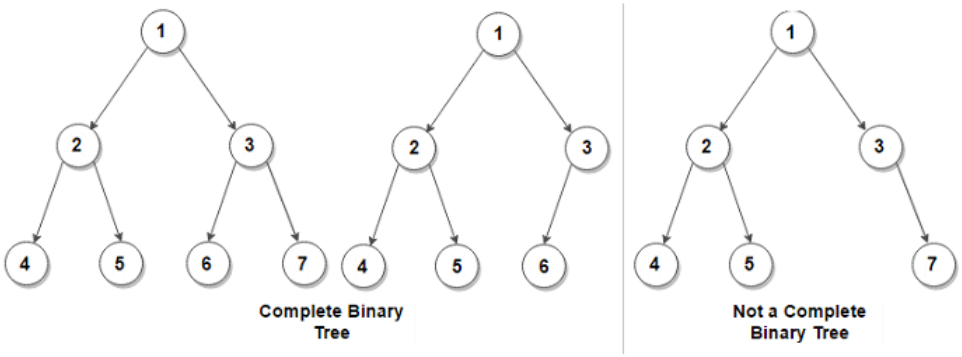
The methods you must implement are:

- `Tree()`
- `Tree(int k)`
- `void insert(T data)`
- `void remove()`
- `ArrayList<Node<T>> preorder()`
- `ArrayList<Node<T>> postorder()`
- `ArrayList<Node<T>> inorder()`
- `ArrayList<Node<T>> convertToArrayList()`
- `int getSize()`
- `void clear()`
- `boolean contains(T data)`
- `int getDepth()`
- `boolean isPerfect()`
- `Node<T> getLast()`
- `Node<T> getRoot()`

The descriptions of these methods can be found in the Tree.java file. You should pay close attention to the `Node<T>` class that we have implemented for you.

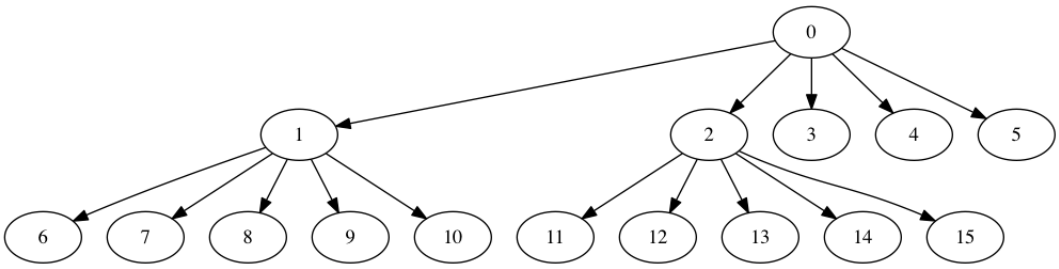
In this assignment, you will implement the tree so that is it always *complete*. In a complete tree, all levels of the tree must be completely full (meaning all nodes in that level have the maximum number of children) except

possibly the last level. If the last level of the tree is not full, the nodes in this level should all be as far to the left as possible. When adding to any complete tree, the nodes should be added from left to right.



As you can see in the picture above, the left two binary trees are complete, while the right one is not. In the first tree, calling `insert` would insert in the left most child slot for node 4. In the second tree, calling the `insert` method would insert in the right child of node 3. In the first tree, calling `remove` would remove node 7. In the second tree, calling `remove` would remove node 6.

The third tree is not complete, since the nodes in the last level are not all as far to the left as possible.



This tree is a 5-ary tree and is complete. Calling `insert` would add to the left most slot of node 3. Calling `insert` again would add to the second to left most slot of node 3. Once all of the child slots for nodes $3, 4$, and 5 have been filled, `insert` would add to the left most slot of node 6. Calling `remove` on the above tree would remove node 15.

General Directions

- Write your name and student ID number in the comment at the top of the java files.
- Implement all of the required functions in the java files.
- You should not import anything that is not already included in the file.
- Pay careful attention to the required return types and edge cases.

Submission Procedure

To submit your homework, simply drag and drop each file individually into the attachment box on the YSCEC assignment page. You should **NOT** zip them or place them in any folder. For this assignment, you should submit only the following files:

- Tree.java
- ⟨student_id⟩.txt

Inside of the ⟨student_id⟩.txt file, you should include the following text, and write your name at the bottom.

In completing this assignment, I pledge that I have not given nor received any unauthorized assistance.

If this file is missing, you will get a 0 on the assignment. It should be named *exactly* your student id, with no other text. For example, `2017123456.txt` is correct while something like `2017123456_hw1.txt` will receive a 0.

Testing

We have provided a small test suite for you to check your code. You can test your code by compiling and running the tester program. You will always get a notification if you fail any of the tests, but you may not get a notification if you passed.

Note that the test suite we use to grade your code will be much more rigorous than the one provided here (and not necessarily a superset of the provided tests). You should consider making your own test cases to check your code more thoroughly.