American University of Armenia, CSE
CS121 Data Structures
Spring 2024

Homework Assignment 3

Due Date: Friday, March 29 by 23:59 electronically on Moodle

Solve the programming tasks using Java, following good coding practices and required format.

1. **(LinkedBinaryTree | 15 points)** Add two methods– public `removeAllNodes()` and private recursive `removeAllNodes(p)` inside the class `LinkedBinaryTree` that will be responsible for **efficiently** removing and printing all the elements in the tree. You should not use any existing traversal methods. You need to use the method `remove()`. What makes your approach efficient?

2. **(AliceInWonderland | 25 points)** Write a generic recursive method that given a binary tree, **efficiently** checks if the inorder traversals of the left and right subtrees of the root node are the reversed (mirrored) versions of each other. Your method may take additional arguments (up to four, including the tree).

   Test your method in a main method using an object of type `LinkedBinaryTree`.

3. **(LinkedTree | 60 points)** Write a class `LinkedTree` that extends the `AbstractTree` and represents a tree where each node can have an arbitrary number of children. For each Node, keep the child nodes in an `ArrayList`.
   Your inner class `Node` should be an appropriate modification of the code from the `LinkedBinaryTree` `Node` and in addition, support all of the following functionality:

   (a) a method for getting the *i-th* child of the node
   (b) a method for removing the *i-th* child of the node and returning the value stored within the node
   (c) a method for setting the *i-th* child of the node
   (d) a method for adding the *i-th* child of the node
   (e) a method for setting the element

   The `LinkedlTree` should override all the necessary methods and should also support the following functionality:

   (a) a method for position validation
   (b) a method `ithChild(p, i)` that returns the position of the `i-th` child of position p
   (c) a method `addIth(p, e, i)` that creates a new *i-th* child of Position p storing element e and returns its Position.
   (d) a method `addFirst(p, e)` that creates a new first child of Position p storing element e and returns its Position.
   (e) a method `addLast(p, e)` that creates a new last child of Position p storing element e and returns its Position.
   (f) a method `addRoot(e)` that places element e at the root of a tree and returns its new Position. If the tree is not empty, the current root becomes the first child of the newly added root.
   (g) a method `set(p, e)` that replaces the element at p with element e and returns the replaced element.
   (h) a method `remove(p)` that removes the node at p and replaces it with its first child, if any. The children of the promoted node (if any) precede the children of the removed node.
   (i) a method `void merge(Position<E> p, LinkedTree<E>... trees)` that adds the argument *trees* to the list of children of position p and empties all the *trees*.