American University of Armenia, CSE
CS121 Data Structures B
Spring 2022

Homework Assignment 5

Due Date: Sunday, April 10 by 23:59 electronically on Moodle

Please solve the programming tasks using Java, following good coding practices (details are on Moodle). Don't forget to submit the corresponding assignment report.

1. **(12 points)** Extend the `LinkedBinaryTree` class with a `ancestors(`p`)` method that returns an iterable collection containing the ancestors of position p. Test it in a program. What is the running time of your method? Briefly justify your answer.

2. **(15 points)** Write a recursive method that, given a non-empty Integer binary tree and its root's position, finds and returns the minimal value stored in that tree. Test it in a program using a `LinkedBinaryTree` object of size 7. What is the running time of your method? Briefly justify your answer.

3. **(23 points)** Write a program (potentially consisting of several generic methods), that given a non-empty binary tree, produces and returns its corresponding array representation. The absence of the positions in the tree should be denoted by null values in the corresponding cells of the array. Test it using a `LinkedBinaryTree` object of size at least 7. What is the running time of your program? Briefly justify your answer.

4. **(45 points)** Write a class `LinkedGeneralTree` that extends the `AbstractTree` and represents a tree where each node can have arbitrary number of children. For each Node, keep the child nodes in an array list. 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

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

   (a) a method for node creation

   (b) a method for node validation

   (c) a method `addRoot(e)` that places element e at the root of an empty tree and returns its new Position

   (d) a method `ithChild(p, i)` that returns the position of the `i-th` child of position p

(e) a method `addIth(p, e, i)` that creates a new *i-th* child of Position p storing element e and returns its Position.

(f) a method `addFirst(p, e)` that creates a new first child of Position p storing element e and returns its Position.

(g) a method `addLast(p, e)` that creates a new last child of Position p storing element e and returns its Position.

(h) a method `set(p, e)` that replaces the element at p with element e and returns the replaced element.

(i) a method `remove(p)` that removes the node at p and replaces it with its first child, if any. The method should throw and *IllegalArgumentException* if the node has more than `1` child.

(j) functionality for traversing the positions of the tree in preorder, postorder and breadth-first order traversals, i.e. `preorder(), postorder(), breadthfirst()` methods, all of which return an iterable collection of the positions of the tree.

(k) functionality for traversing the positions of the tree, i.e. a `positions()` method implementing breadth-first traversal.

(l) functionality for traversing the elements of the tree, i.e. an `iterator()` method.

Think carefully where you should add each of these methods; you may need to modify AbstractTree or LinkedGeneralTree classes.

Specify and justify the running time of each method in your implementation.