COS 225: Object-Oriented Design, Programming and Data Structures
Fall 2023

Homework Assignment 4: Trees
November 29, 2023
Due: December 14, 2023

Submission instructions:

Please submit your code in .java files. Do **not** submit .class files! If any part of the problems requires a written answer, please submit your answer in a Word (.doc or .docx) document, or in a .pdf document.

Total points: 100.

In this assignment, you will complete the programming tasks in Problems 1 and 2.

1. **Adding new methods to the binary tree class (80 points).**

In this problem, you will add a number of new methods to the `BinaryTree<E>` class, which we discussed in our lectures. For all parts, you may implement appropriate private helper methods to help you implement the public methods.

A base implementation of the `BinaryTree<E>` class and the `BinaryTreeNode<E>` class has been provided in `BinaryTree.java` and `BinaryTreeNode.java`, respectively. You should modify the provided code appropriately.

You may use appropriate data structures you implemented in Homework 3 to solve some of the tasks below. You should include implementations of any additional data structures you used in your submission.

(a) Implement a public method `height()` and a private recursive method `heightOfSubtree(BinaryTreeNode<E> node)` in the `BinaryTree<E>` class. The private method `heightOfSubtree(BinaryTreeNode<E> node)` should compute the height of the subtree whose root is the given `node` using recursion. The public method `height()` should call the `heightOfSubtree` method with an appropriate argument. What is the time complexity of computing the height of the tree? (5 points).

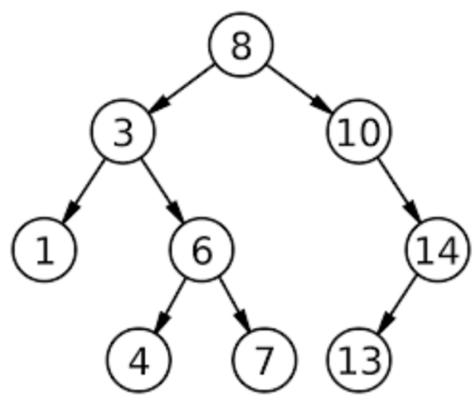
(b) In our lectures, we discussed a naïve implementation of the method `isBalanced()`, which checks whether the binary tree is balanced or not (see the lecture slides for Lecture 24: Trees (Part 2)). In particular, we discussed that the problem with the given implementation of `isBalanced()` is the repeated computation of the heights of the subtrees, especially for those whose roots are deeper in the tree. Show that the naïve implementation given in the lecture slides has a worst-case time complexity of $O(n^2)$. Reimplement `isBalanced()` and its recursive helper `isBalancedRecursive()` so that the time complexity of checking if the binary tree is balanced will be $O(n)$. (10 points).

(c) Implement a public method `insertIntoShorterSubtree(E new_data)`, which inserts `new_data` into the binary tree according to the following rules: (1) if the tree is empty, insert `new_data` at the root; (2) if the root does not have a left child, insert `new_data` at the left child of the root; (3) otherwise, if the root does not have a right child, insert `new_data` at the right child of the root; (4) if the root has both a left and a right child, insert `new_data` into the subtree with a smaller height (if the left subtree and the right subtree have the same height, then insert `new_data` into the left subtree). What is the time complexity of your implementation? (10 points).

(d) Implement a public method `insertIntoFirstAvailablePosition(E new_data)`, which inserts `new_data` into the binary tree at the first position available in breadth-first, level-by-level order (from left to right). What is the time complexity of your implementation? (10 points).

(e) Implement a public method `toString()` such that when the binary tree shown below is printed, we shall see the following on the computer screen:

```
(0) 8
    (1) 3
        (2) 1
        (2) 6
            (3) 4
            (3) 7
    (1) 10
        (2) null
        (2) 14
            (3) 13
            (3) null
```



The number in parentheses gives the level of the node. The amount of indentation is also related to the level of the node – e.g., level 0 has no indentation, level 1 has an indentation of 4 spaces, level 2 has an indentation of 8 spaces, etc. What is the time complexity of your implementation? (10 points).

(f) Implement a public method `deleteByPromotingInorderPredecessor(E data)`, which removes all occurrences of `data` from the binary tree, by promoting the in-order **predecessor** of any node containing `data` (if the node has two children). You may implement a private recursive helper method with the following method signature:

```
private BinaryTreeNode<E> deleteRecursiveByPromotingInorderPredecessor(E
data, BinaryTreeNode<E> node)
```

which removes all occurrences of data from the subtree whose root is given by node, by promoting the in-order <u>**predecessor**</u> of any node containing data (if the node has two children), and <u>**returns**</u> a new root node of the subtree with all occurrences of data removed. You should use this private recursive method to help you implement the public method. (20 points).

(g) Implement a public method `isBinarySearchTree()`, which returns `true` if the binary tree is a binary search tree, and `false` otherwise. What is the time complexity of your implementation? (10 points).

(h) In <u>another</u> class, implement a static `main(String[]  args)` method that tests the correctness of your implementations of all of the above public methods. In particular, you should first construct a binary tree shown in the figure below part (e) (*Hint*: you will need to use the `setLeft` and `setRight` methods defined in the `BinaryTreeNode<E>` class to construct a particular tree), and then do the following:

- Print the tree using `toString`;
- Compute its height using `height`;
- Check if it is balanced using `isBalanced`;
- Check if it is a binary search tree using `isBinarySearchTree`;
- Insert 0 into the tree using `insertIntoShorterSubtree`;
- Insert 9 into the tree using `insertIntoFirstAvailablePosition`;
- Delete 8 from the tree using `deleteByPromotingInorderPredecessor`;
- Print the tree again using `toString`.

Is the resulting tree a binary search tree? Why or why not? (5 points).

2. Implementing heap sort (20 points).

In this problem, you will implement the heap sort algorithm described in the lectures. You should implement the algorithm as a <u>generic</u> static method `heapSort(E[] arr)`, which sorts the array `arr` in-place using heap sort. The type parameter `E` should be declared with `Comparable<E>` as upper bound. You should also include a static `main(String[] args)` method, which sorts the array {33, 57, 21, 7, 45, 3, 27, 25, 40, 12} into ascending order.