Computational Problem Solving CSCI-603-01 Exam 2: Practical

10/26/2020

Instructions

- You have 75 minutes to complete this practical and upload it to MyCourses.
- Your program should run using Python version 3.
- You are not required to comment the code you write for this exam.
- You may not communicate with anyone except for the proctor.
- You are not allowed to look at any other programs you have written prior to this exam.
- The only applications allowed are PyCharm (or any other IDE), and a web browser for accessing MyCourses. You are not allowed to use any other programs!

Failure to follow this instructions will result in automatic failure of the exam!

Getting started

For this programming assignment, you will be adding some functionality to a binary tree node.

Add the provided files to a new Pycharm project (if using Pycharm). Do not use any other versions of these files!

The following questions will ask you to finish writing some of the methods required to complete the implementation of the module `btnode.py`. **Unless instructed, you are NOT allowed to modify anything else in the files.**

Feel free to add any test code outside of the classes to check your methods.

Efficacy counts! For example, do not implement $\mathcal{O}(n)$ algorithm if a $\mathcal{O}(1)$ solution exists!

1 Part 1: Build test tree (10 points)

`btnode.py` is an implementation of a binary tree node. The class has three fields:

1. `val` stores the node's value
2. `left` references the left child
3. `right` references the right child

Find the `test()` function at the bottom of the file. We have already created some trees: `left`, `right`, and `parent`.

YOUR WORK: Uses those trees as a guide to complete `buildTestTree()` function. `buildTestTree()` must build and return the following tree depicted below. You will use this tree for testing the correctness of the functions you will implement later.

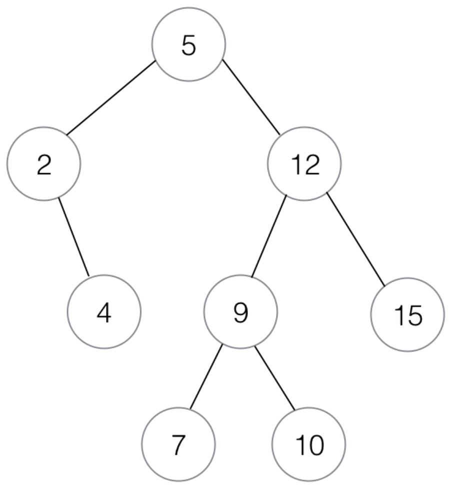

Figure 1: Binary Search Tree

2 Part 2: `depth(node, nodeValue)` (20 points)

YOUR WORK: Complete the `depth(node, nodeValue)` function such that it computes the depth of a node with the value nodeValue in a binary search tree.

Consider the binary search tree from Figure 1. The value 5 is at depth 0, the value 2 is at depth 1, the value 10 is at depth 3, and so on. If a value is not in the tree, you should return the depth that it would be, if present. So, for this tree, `depth(1)` should return 2, since 1 would be the left child of 2.

You can assume the tree is a correct binary search tree. To make sure that the function works correctly, you may want to add more test cases in the `test` function.

For full credit, you may not use any global variables.

3 Part 3: `printPretty(node)` (20 points)

<u>YOUR WORK</u>: Implement `printPretty(node)` which performs a depth order traversal, that is: first prints the root (depth 0), then, it prints the root's children from left to right (depth 1), then, it prints the root's grandchildren from left to right (depth 2), and so on.

Consider the binary search tree from Figure 1. `printPretty(t)` should print 5 2 12 4 9 15 7 10.

Note: This is not a recursive function. You need to use an auxiliary data structure.

4 Part 4: `numBetween(node, min, max)` (25 points)

<u>YOUR WORK</u>: Implement `numBetween(node, min, max)` which returns the number of elements in the tree with values between min and max, inclusive. For full credit, this must be calculated recursively, and cannot simply visit all elements in the tree and test them.

Consider the binary search tree from Figure 1. `numBetween(t, 6,10)` should be able to return 3 without ever visiting 2, 4, or 15.

5 Part 5: `hasPath(node, total)` (25 points)

<u>YOUR WORK</u>: Implement `hasPath(node, total)` which returns whether there is a path from root to leaf where the sum of the nodes' values is equal to the given total. The function will return false if no path can be found.

Consider the binary search tree `t` from Figure 1.

- `hasPath(t, 11)` returns `true`. Path 5 - 2 - 4.
- `hasPath(t, 32)` returns `true`. Path 5 - 12 - 15.
- `hasPath(t, 100)` returns `false`.

Note: The function must return a `boolean` result. You do not need to compute the path.

There are not more questions regarding modifications in `btnode.py`. Once you finish this question, you can submit your solution to myCourses.