Computational Problem Solving **CSCI-603-03**
Deque and Binary Tree **Exam 2: Practical**

Full Name (printed): ____

Time Finished: ______________________

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!

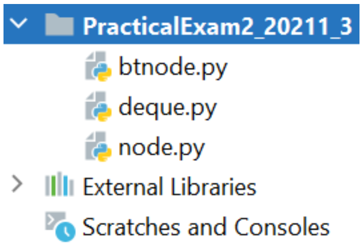
Failure to follow these instructions will result in automatic failure of the exam!

Getting started

For this programming assignment, you will be working on adding some functionality to a double-ended queue (referred to as a deque) and a binary tree node.

First, go to MyCourses, click on Assignments, and then Exam 2 Practical, Section 3. Download the starter code. Add the provided files to a new Pycharm project (if using Pycharm). Do not use any other versions of these files!

Your project structure must look as follows:



PracticalExam2_20211_3
btnode.py
deque.py
node.py
External Libraries
Scratches and Consoles

Here is a brief description of what is provided to you:

- `node` - represents a linkable node used in stacks, queues, and linked lists. **Do NOT modify this file.**
- `deque` - represents a double-ended queue, also called deque. This class is partially implemented.
- `btnode` - this is the representation of a binary tree node. This class is partially implemented.

The following questions will ask you to finish writing some of the methods required to complete the implementation of the `deque.py` and the `btnode.py` modules. **Unless instructed, you are NOT allowed to modify anything else in these 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 Problem 1: Double-ended Queue

A double-ended queue, also called deque, is similar to a queue, but it also supports operations to add elements to the front, and remove elements from the back.

1.1 Add a `size` field to `Deque` (10 points)

Add a new field, `size`, to `Deque` that keeps track of the number of items in the deque. Modify the `init` method to initialize it. You must also update the `size` accordantly in every method that adds and/or removes elements from the deque.

1.2 Deque.enqueueFront (20 points)

Implement the method `enqueueFront(self, newValue)` which inserts a new node with the value `newValue` at the front of the deque.

The time complexity of this method must be always $\mathcal{O}(1)$.

1.3 Deque.dequeueBack (20 points)

Implement the method `dequeueBack(self)` which removes the item at the back of the deque.

The time complexity of this method in the worst case scenario is $\mathcal{O}(n)$.

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

2 Problem 2: Binary Tree Node

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

- `value` - store the node's value
- `left` - references to the left child
- `right` - references to the right child

Find the `test` function at the bottom of the file. We have already created some trees: `parent`, `left`, `right`, and `tree`. `tree` is the tree depicted below. You will use it for testing the correctness of the functions you will implement later.

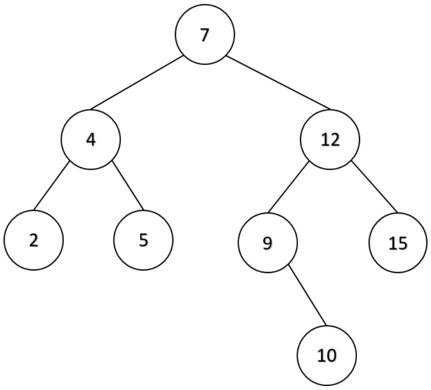


Figure 1: Binary Search Tree

2.1 BTNode.printDescendingOrder (20 points)

Complete the function `printDescendingOrder(root)` that prints in a single line the content of the given tree in descending order. We have seen different traversal methods for binary trees in class. A derivation of those methods can be used to print the tree's nodes value in descending order.

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

2.2 BTNode.pathToAncestor (25 points)

Implement pathToAncestor(node, desValue, ancValue) which returns a Python list of all of the nodes between the node with the value desValue and the node with the value ancValue (both included) in order from deepest to shallowest. You can assume that ancValue will always point to a node that is an ancestor of the node specified by desValue.

Consider the binary search tree tree from Figure 1.

- pathToAncestor(tree,10,7) returns [10, 9, 12, 7]
- pathToAncestor(tree,10,12) returns [10, 9, 12]
- pathToAncestor(tree,2,7) returns [2, 4, 7]
- pathToAncestor(tree,4,4) returns [4]

Note: You are not allowed to create any extra helper function to implement this functionality nor add more arguments to the function's signature. You are also not allowed to create any extra data structure besides the path list. The function must be recursive and fruitful. The path must be built recursively from the descendant node up to the ancestor node.

Submission

Submit only your deque.py and btnode.py modules to the MyCourses dropbox. Verify your submission by reloading and checking the dropbox.