CE204 ASSIGNMENT 1 2020

Set by: Mike Sanderson

Credit: 10% of total module mark

Deadline: 11.59.59, Tuesday 18 February

Introduction

You should refer to sections 5 and 7 of the Undergraduate Students' Handbook for details of the University policy regarding late submission and plagiarism; the work handed in must be entirely your own. Your programs will be checked by an intelligent plagiarism detection system that looks for similarities between the submitted programs.

The assignment comprises 2 exercises – the first may be written using either Java or Python 3; the second must be written in Java.

It is expected that marking of the assignments will be completed by the beginning of week 25.

Submissio

Copies of all source xt file containing a pasted copy of the output g d to FASER in a single `.zip` or `.7z` file; m in other formats.

Marking Criteria

Exercise 1 will contribute 35% of the total mar t (28% for the class and 7% for the code for testing it) and exercise 2 will contribute 65% (17.5% for the `nonleaves` method, 17.5% for the `depth` method and 30% for the `range` method).

The 7% for the testing of exercise 1 is allocated purely for the thoroughness of the testing. Of the remaining marks about three-quarters will be awarded for correct behaviour of the classes or methods, with the remaining quarter being awarded for efficiency and programming style.

A Java program that fails to compile or a Python program that is rejected by the Python syntax-checker will not earn more than half of the total available marks for the exercise.

Exercise 1

A double-ended queue (dequeue) is like a queue but items may be added to and removed from either end. The ends are known as left and right. The primitive operations are *createdeq*, *isempty*, *left*, *right*, *addleft*, *addright*, *removeleft* and *removeright.*

You are required to write in either Java or Python 3 a class that implements the dequeue ADT with methods for all of the primitive operations (other than *createdeq*, since a deque can be created using code such as `new Deque()`) The class should have a (Java) constructor or (Python) `__init__` method that ensures all newly-created queues are empty. The class should also have a (Java) `toString` or (Python) `__str__` method that generates a string of the form `<3,5,7>`. The data in the class must be private.

The *left*, *right*, *removeleft* and *removeright* method should throw/raise an exception when applied to an empty queue.

If you choose to use Java the class should be generic (i.e. declared as `Dequeue<T>`).

You should also write and submit code that tests the behaviour of all of the operations, generating output indicating what methods are being called and what results are returned and displaying the contents of the queue whenever changes are made. A fragment of the output might look like

```
Queue conten
Adding 10 to
Queue conten
Calling right: 7 returned
```

You should include code to test the behaviour of the ight, *removeleft* and *removeright* operations when applied to an em require four separate try-catch or try-except blocks.

The test code should ***not*** be interactive.

If you write the program in Java the test code must be written in a `main` method in a separate class in a separate file; if you choose to use Python you may place the test code in the same file, but it must be outside the class.

You may make use of most of the classes from the standard Java or Python libraries; however use of the `deque` class from the Python `collection` module or the `Deque` interface from the Java Collections Framework is ***not*** permitted, and you must not use any third-party classes. Additionally, since the use of classes such as `LinkedList` makes the coding much simpler, a class in which the data member is simply a Collections Framework object will not earn more than 75% of the marks available for correct behaviour. Code supplied for this module, such as the `QueueException` or `ListCell` classes, may be freely used without the need for acknowledgement.

Exercise 2

A `BST` class based on the class on slides 19-23 of part 3 of the lecture slides is provided online for this exercise. You are required to add three extra methods to the class. These ***must*** be declared as

```
public int nonleaves() { …… }
public int depth() { …… }
public int range(int min, int max) { …… }
```

The `nonleaves` method should return the number of non-leaf nodes in the tree.

The value returned by the `depth` method should the maximum depth of the tree (i.e. the number of nodes on the path from the root to the deepest leaf); a tree containing just a root will have a depth of 1.

Both of the above methods should return 0 if the tree is empty.

The `range` method should return the number of elements in the tree whose values are in the range `min` to `max` (inclusive); 0 should be returned if the tree is empty or contains no values in that range; an exception of type `IllegalArgumentException` should be thrown if the value of `max` is less than the value of `min`. (This exception class is part of the Java language; you do not have to write it.)

All of your code m ed file. You may if you wish add extra met ny new instance variables to this class nor mo ate support methods and instance variables to the `BST` class but must no xisting methods in this class.

It is strongly recommended that you write a se `main` method to test your new methods, but this class should ***not*** be submitted – I will use my class with a `main` method for testing of the program; this will assume that the three methods are declared exactly as shown above. If you complete just one or two of the methods you ***must*** provide a dummy version of the other(s) (with bodies that print a "not attempted" message and return 0), in order to allow the testing class to compile successfully.