CE204 ASSIGNMENT 1 2019

Set by: Mike Sanderson

Credit: 10% of total module mark

Deadline: 11.59.59, Tuesday 19 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.

The assignment comprises 2 equally-weighted 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 end of week 24.

Submission

Copies of all source code and all Java `.class` files and also a text file containing a pasted copy of the output generated from exercise 1 must be submitted to the online submission system in a single `.zip` or `.7z` file; marks will be deducted for any files submitted in other formats.

Lab Demonstration

You may be required to demonstrate your solution to exercise 2 in the lab in week 22 or week 23; you will be informed by the end of week 21 whether such demonstrations will be required. If this requirement is imposed, 10% will be deducted from the mark for that exercise for any assignment that is not demonstrated in the lab (unless there are genuine extenuating circumstances that prevent attendance at both labs).

Marking Criteria

Exercise 1 will contribute 50% of the total mark for this assignment (about 40% for the class and 10% for the code for testing it) and exercise 2 will contribute 50% (about 20% for the `nonleaves` method and 30% for the `range` method).

The 10% for the testing for 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 40% of the total available marks for the exercise.

Exercise 1

A priority queue is like a queue but when an item is added it is given a priority and it will advance ahead of any items in the queue with lower priority (while remaining behind items with the same priority or with higher priority). The primitive operations are *createpq*, *isempty*, *front* (which should return the string from the front item in the queue without any information about its priority), *deletefront*, *frontpri* (which should return the priority of the front item), and *addtopq* (which should take two arguments, a string and a priority).

You are required to write in either Java or Python 3 a class that implements a priority queue of strings with methods for all of the primitive operations (other than *createpq*, which is not needed since the constructor (or Python `__init__` method) should ensure that any new priority queue object is empty). The class should also have a (Java) `toString` or (Python) `__str__` method that generates a string of the form `<"hello":7,"hi":2>`. The data in the class must be private. If you choose to use Java you are not allowed to make use of the `PriorityQueue` class from the Collections Framework.

For this exercise priorities are to be represented by integers in the range 1 to 20, with 1 denoting the highest priority and 20 the lowest. The *front*, *frontpri* and *deletefront* methods should throw/raise an exception when applied to an empty queue; the *addtopq* method should throw/raise an exception if the priority is not in the range 1 to 20. If you choose to use Python all methods should also raise an exception if an argument of the wrong type is supplied.

You may make use of any classes supplied for this module such as `ListCell` and `QueueException`.

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 contents: <"hello":1,"hi":5>
Adding "ho" with priority 3
Queue contents: <"hello":1,"ho":3,"hi":5>
Calling frontpri: 1 returned
Calling front: "hello" returned
```

You should include code to test the behaviour of the *front*, *frontpri*, and *removefront* operations when applied to an empty queue – this will require three separate try-catch or try-except blocks. You should also test the behaviour of the *addtopq* method when supplied with an out-of-range priority argument.

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 any classes from the standard Java or Python libraries (other than `PriorityQueue`), but must not use any third-party classes. 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 on Moodle for this exercise. You are required to add two extra methods to the class. These ***must*** be declared as

```
public int greater(int n) { …… }
public int nth(int i) { …… }
```

The value returned by the `greater` method should be how many numbers with value greater than `n` appear in the tree; the method should return 0 if the tree is empty.

The `nth` method should return the element that would be found in location `i` if the contents of the tree were stored in ascending order in an array. An exception of type `NoSuchElementException` should be thrown if such an element would not exist (i.e. if `i` is not in the range 0 to `count`-1, where `count` is the number of elements in the tree); you will need to import this exception class from the `java.util` package. Note that to gain maximum marks you should not actually generate the array – there are more efficient ways to obtain the result.

All of your code (apart from import statements at the beginning of the file) must be written inside the classes in the supplied file. You may if you wish add extra methods to the `BTNode` class, but must not add any new instance variables to this class nor modify its constructor. You may add extra private support methods and instance variables to the `BST` class but must not modify any of the existing methods in this class.

It is strongly recommended that you write a separate class with a `main` method to test your new methods, but this class should ***not*** be submitted – a class with a `main` method will be supplied (after the deadline) if the lab demonstration of the program takes place; this will assume that the two methods are declared exactly as shown above. If you complete just one of the methods you ***must*** provide a dummy version of the other (with bodies that print a "not attempted" message and return 0), in order to allow the testing class to compile successfully.