Koc University

COMP202

Data Structures and Algorithms

Assignment 3

Instructor: Gozde Gul Sahin

Due Date: March 18 2020, 23:59

<u>Submission Through: Blackboard</u>

Introduction

This programming assignment will test your knowledge and your implementation abilities of what you have learned in the Trees, Binary Search Trees, and Priority Queues parts of the class.

This homework must be completed individually. Discussion about algorithms and data structures is allowed but group work is not. Coming up with the same algorithm and talking about the ways of implementation leads to very similar code which is treated as plagiarism! Discuss carefully. Any academic dishonesty, which includes taking someone else's code or having another person doing the assignment for you will not be tolerated. By submitting your assignment, you agree to abide by the Koc University codes of conduct.

Description

The assignment will have two parts. The first part requires you to implement a Binary Search Tree to be used as a map data structure with a few additional operations, implement an Array Based Heap and modify a BST. The last two will be used as a Priority Queue. The second part requires you to implement a student enrollment system as a table using your own binary search tree.

Part I: Implementing the Data Structures

You are going to implement a binary search tree that implements a given binary search tree interface and a given map interface. You are also given an adaptable priority queue interface. This is going to be implemented by an array-based heap and a binary search tree-based priority queue, which also extends the previous binary search tree. The details of these interfaces will not be given here due to the comments in the files. Feel free to use the discussion forum created for this assignment if anything is unclear. You are additionally given additional classes that will help with the implementation.

As mentioned above, the code has comments about the assignment and the desired functionality. Majority of what you need to do is actually in these comments so make sure you read them carefully. You are provided with a compressed file that contains following files. **Bold** ones are the ones you need to modify and they are placed under the code folder, with the rest under given.

- *iMap.java*: This file defines a simple map interface. Even though you do not need to modify this, make sure you read the comments in detail. It is pretty straightforward
- *iBinarySearchTree.java*: This file includes the iBinarySearchTree interface, which describes the desired binary search tree ADT. Even though you do not need to modify this, make sure you read the comments in detail
- *Entry.java*: This file includes the Entry class that describes a Key-Value pair. Certain useful methods are implemented for you. This will form the basis of your binary tree nodes and be used in the priority queues. You should take a look at its functions but do not modify it
- ***BinaryTreeNode.java***: This file includes the BinaryTreeNode class which is used in the binary tree as nodes. It extends the Entry class. You are free to design and implement it however you want
- ***BinarySearchTree.java***: This is where one of the main data structures of the assignment, the BinarySearchTree class, should be implemented. It extends both the iBinarySearchTree and the iMap interfaces. You should implement this class and the BinaryTreeNode class in conjunction
- *iAdaptablePriorityQueue.java*: This file defines the iAdaptablePriorityQueue interface, which describes an adaptable priority queue interface. As with the other interface files, make sure to go over the comments. There is one important point, its operations assume that both the keys and the values are unique. This is done to make the assignment

easier but it is not very realistic. If you ever need something like this in real life, keep this in mind! Another point is that the adaptations require you to find the right entries. This is slightly different than the Position concept we have seen, i.e., to change keys or values we find the relevant entry but do not store individual entries

- **BSTBasedPQ.java**: This file defines the binary search tree-based priority queue class BSTBasedPQ. It extends the BinarySearchTree class and implements the iAdaptablePriorityQueue interface. Do not try to create heap behavior but think about how to use a BST as a priority queue. Hint: There is a O(h) way to get to the minimum element!

- **ArrayBasedHeap.java**: This is where the other main data structure of the assignment, the ArrayBasedHeap class, should be implemented. It extends the iAdaptablePriorityQueue inter face. We suggest that you first implement the array based binary tree functionality (no interface given but you can get inspiration from the available interfaces/classes), then add heap functionality. Make sure that this actually behaves as a heap and not a sorted list!

- *AutograderMain.java*: This file has the auto-grading for your data structures implementations. Run this to see your grade and errors

- *TestPQ.java*: An implementation of an order book based on your priority queue implementations. This will be used as an "on the job" testing of your code. It simulates an order book for a single "item". It reads the orders under the exchange folders and updates the priority queues. You can directly call the main function of this code for debugging. The Exchange folder contains a small number of orders to help you debug. Your code will be tested on the other two exchange folders. Autograder code calls a method under this file to create output files (e.g. student heap Exchange2 output.txt) which are then compared to reference files, Exchange2 output.txt, and Exchange3 output.txt. Feel free to ask about its implementation in person but we are not going to delve into the details here

- *DefaultComparator.java*: This file implements the comparators that are used in the assignment. You do not need to touch this or even go over it unless you are curious

- *DataGenerator.java*: This file provides functionality to generate random data. You do not need to worry about it

- *Util.java*: This file includes utility functions. Do not worry about it, you will not need to use anything from this file in this assignment.

Part II: Student Enrollment Application

You are asked to implement a student enrollment application in the second part of the assignment. The student enrollment application stores student information and is efficiently searchable (In this context, efficient means not iterating over all the entries!) using two keys; student name and student number. In addition, it should be searchable by e-mail (not necessarily efficiently). You are only allowed to use your own tree implementation as a map in this second part!. My suggestion is for you to use two binary search trees, one for the student name and the other for the student number. All the rest are in the comments. The relevant files are:

- *StudenttInfo.java*: This file has the StudentInfo class that you should store as your values. You do not need to touch this but should go over it
- **StudentEnrollment.java**: This file includes the StudentEnrollment class which you need to complete based on the description given in this document and the comments in the file. To make things easier, you can assume that the name and numbers will be unique. (Hint: Multimaps). Other than only being allowed to use your own tree data structures, feel free to implement this class in any way you want
- *GradeStudentEnrollment.java*: This file contains the code for generating output using your student enrollment implementation. It uses the database.txt to fill your student enrollment and query.txt to perform operations on it. It generates a file named student-enrollment-output.txt to be compared with the provided enrollment-output.txt file. You can run this file separately as well. We are not going to delve into it how it works here but you can figure out how it works by looking at the code if needed. Feel free to create your own query and database files if the given ones are too big and complex to start with.

The input files for both testing the student enrollment and the priority queues are generated randomly, based on certain rules. We have used a mix of names and surname for the student enrollment and created random student numbers, e-mails, and addresses. Please do not take any of the resulting names seriously.

Grading

Your assignment will be graded through an autograder. Make sure to implement the code as instructed, using the same variable and method names. A version of the autograder has been

released to you. Our version will more or less be similar, with some additional checks. Run the main program in the AutograderMain.java to get the autograder output and your grade.

In case the autograder fails or gives you 0 when you think you should get more credit, do not panic. Let us know. We can go over everything even after your submission.

Submission

You are going to submit a compressed archive through the Blackboard site. The file should extract to a folder with your student ID without the leading zeros. This folder should only contain files that were in boldface in the previous section. Other files, which you should not have modified anyways, will be deleted and replaced with default versions.

We download all the submissions from Blackboard together and put them in a folder. We then run multiple scripts to extract, clean up and grade your codes. Your codes are compiled from the command line, no IDE is used. If you do not follow the instructions given below, then scripts might fail. This will lead you to get a lower grade than what the autograder suggests. More importantly, your code might not compile, which will result in a grade of 0. Make sure to follow these instructions:

Important: Make sure to download your submission to make sure it is not corrupted and it has your latest code. You are only going to be graded by your Blackboard submission.

Submission Instructions

- Make sure to remove all the unused imports and that you have the original package names. What compiles on your machine may not compile on ours due to simple import errors. Code that does not compile will receive a grade of 0.
- You are going to submit a compressed archive through the Blackboard site. The file can have zip, tar, rar, tar.gz or 7z format
- The compressed file should have all the files that you need to modify. If anything is missing, we automaticlly copy over the default versions
- The compressed file should not contain another compressed file
- One advice is after creating the compressed file, move it to your desktop and extract it. Then check if all the above criteria is met
- Once you are sure about your assignment and the compressed file, submit it through Blackboard. • After you submit your code, download it and check if it the one you intended to submit. • Do not submit code that does not terminate or that blows up the memory.