UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS212: Practical 3

Release: Monday 12 April 2021, 18:00
Deadline: Friday 16 April 2021, 18:00

PLAGIARISM POLICY

UNIVERSITY OF PRETORIA

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to `http://www.library.up.ac.za/plagiarism/index.htm` (from the main page of the University of Pretoria site, follow the *Library* quick link, and then choose the *Plagiarism* option under the *Services* menu). If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding. Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

Objectives

The aim of this practical is to learn how to implement and use binary search trees.

Instructions

Complete the task below. Certain classes have been provided for you alongside this specification in the *Student files* folder. You have been given a main file which will test some code functionality, but it is by no means intended to provide extensive test coverage. You are encouraged to edit this file and test your code more thoroughly. Remember to test boundary cases. Submission instructions are given at the end of this document.

Task 1: Binary Search Trees [39]

A binary search tree is a variant of the binary tree that makes a binary search possible. For each node x of the tree, all values stored in its left subtree are less than the value of x, and all values stored in the right subtree are greater than the value of x. This allows a complexity between *O(log n)* and *O(n)* for search, insert and delete operations within a sequence of `n` elements. The shape of the tree determines the complexity, which can change when the tree is updated.

You have been given a partially implemented binary search tree class and a binary search tree node class to use. Your task is to implement the following methods in the binary search tree class according to the given specification:

```
boolean isEmpty()
```

This function should determine whether the binary search tree is empty or not. It returns true if the binary search tree is empty and false otherwise.

```
BSTNode<T> clone()
```

This function should create a **new** binary tree (with its own nodes) that is a clone of the original binary search tree. This function should return the root node of the new binary tree.

Note: Make sure you don't make any changes to the original tree.

```
BSTNode<T> mirror()
```

This function should create a **new** binary tree (with its own nodes) that is the mirror image across the vertical axis of the original binary search tree. This function should return the root node of the new binary tree.

Note: Make sure you don't make any changes to the original tree.

```
void insert(T element)
```

This function should insert the given `element` in the binary search tree. `element` should be inserted in the correct position in the binary search tree to maintain the sorted order required by a binary search tree. **Do not** insert duplicates into the tree.

`boolean deleteByMerge(T element)`

This function should delete the given `element` from the binary search tree. It returns true if `element` is removed successfully and false otherwise. This function should use the delete by merging strategy.

`boolean deleteByCopy(T element)`

This function should delete the given `element` from the binary search tree. It returns true if `element` is removed successfully and false otherwise. This function should use the delete by copying strategy.

`T search(T element)`

This function should find the given `element` in the binary search tree. If `element` is found in the binary search tree it should return the element and otherwise it should return `null`.

`T getPredecessor(T element)`

This function should find and return the element stored in the **predecessor** of the node containing the given `element` in the binary search tree. A predecessor of a node N is the node with the *largest* element which is *less* than the element stored in node N. Return `null` in the following scenarios:

- The tree is empty
- `element` is not found in the tree
- `element` is found in the tree but it does not have a predecessor (think of the scenarios where this is the case)

`T getSuccessor(T element)`

This function should find and return the element stored in the **successor** of the node containing the given `element` in the binary search tree. A successor of a node N is the node with the *smallest* element which is *greater* than the element stored in node N. Return `null` in the following scenarios:

- The tree is empty
- `element` is not found in the tree
- `element` is found in the tree but it does not have a successor (think of the scenarios where this is the case)

Implement the methods listed above. You can use both iterative and recursive approaches. You may use your own helper functions to assist in implementing the specification. However you may not modify any of the given method signatures. Also do not modify any of the other code that you were given for this task.

Submission

You need to submit your source files on the Fitch Fork website (https://ff.cs.up.ac.za/). All methods need to be implemented (or at least stubbed) before submission. Place your **BST.java** and **BSTNode.java** file in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number. There is no need to include any other files in your submission. You have 4 days to complete this practical, regardless of which practical session you attend. You have 5 submissions and your best mark will be your final mark. Upload your archive to the *Practical 3* slot on the Fitch Fork website. Submit your work before the deadline. **No late submissions will be accepted!**