UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS212: Assignment 2

Deadline: Friday 29 April 2022, 11:59

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

This assignment aims to learn how to implement different types of a Threaded AVL binary tree and to traverse them without using a stack or a recursion.

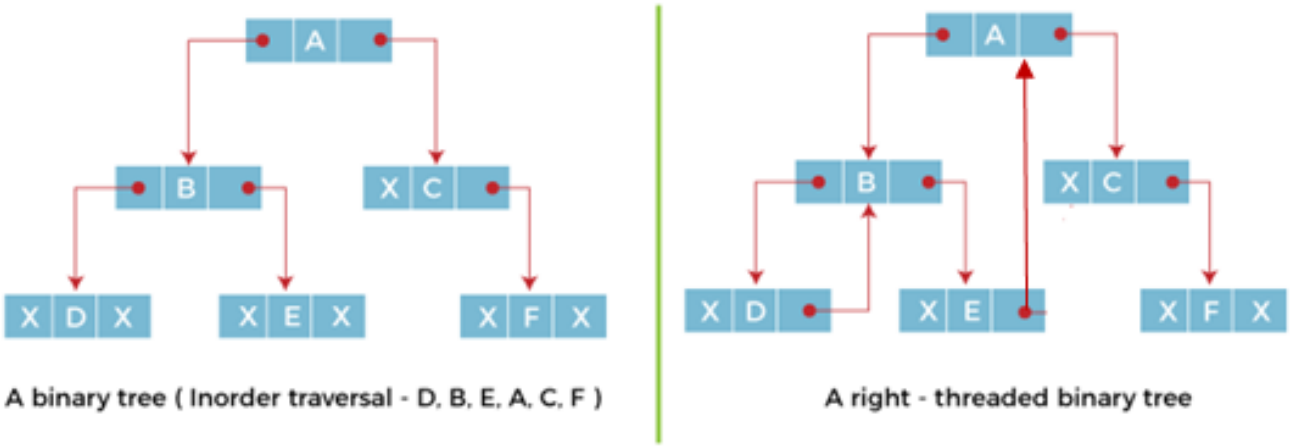
Background

An AVL tree is one in which the height of the left and right subtrees of every node differs by at most one. In other words, AVL Tree is a height-balanced binary tree. Each node is associated with a balanced factor which is calculated as the difference between the height of its left subtree and the right subtree. AVL is most applicable in scenarios where search operation is more frequent compared to insertion and deletion. Detailed background and examples of AVL can be found on pages 261 to 267 of the textbook.

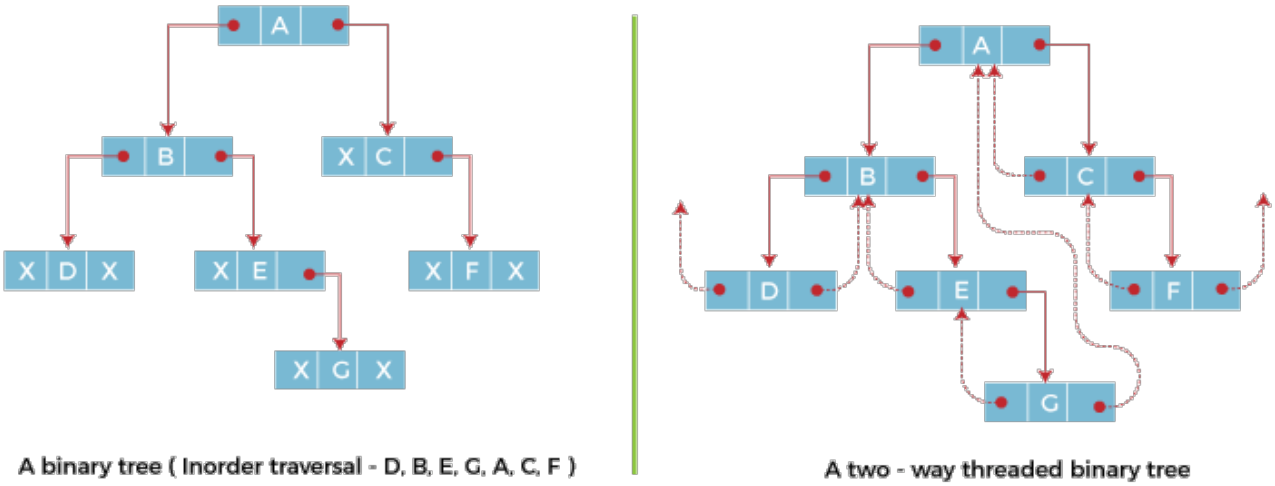
In this assignment, a threaded AVL will be implemented. What is Threaded AVL? In the linked representation of binary trees, more than one-half of the link fields contain NULL values which results in wastage of storage space. If a binary tree consists of "n" nodes then "n+1" link fields contain NULL values. So, in order to effectively manage the space, a method was devised by Perlis and Thornton in which the NULL links are replaced with special links known as threads. Such binary trees with threads are known as threaded binary trees. A binary tree is made threaded by making all right child pointers that would normally be a NULL point to the in-order successor of the node (if it exists) for a single threaded, or both left and right NULL pointers are made to point to in-order predecessor and in-order successor respectively.

Moreover, the idea of threaded binary trees is to make in-order traversal faster and do it without stack and without recursion.

The below figure shows the in-order traversal of this binary tree yields D, B, E, A, C, F. When this tree is represented as a right threaded binary tree, the right link field of leaf node D which contains a NULL value is replaced with a thread that points to node B which is the in-order successor of a node D. In the same way other nodes containing values in the right link field will contain NULL value.



The below figure shows the in-order traversal of this binary tree yields D, B, E, G, A, C, F. If we consider the two-way threaded Binary tree, the node E whose left field contains NULL is replaced by a thread pointing to its in-order predecessor i.e. node B. Similarly, for node G whose right and left linked fields contain NULL values are replaced by threads such that right link field points to its in-order successor and left link field points to its in-order predecessor. In the same way, other nodes containing NULL values in their link fields are filled with threads.



Example

You can use the AVL Tree visualiser (https://www.cs.usfca.edu/ galles/visualization/AVLtree.html) to get a visual demonstration as to how the insertion and deletion strategies should work.

Instructions

Complete the tasks below. Certain classes have been provided for you in the *files* zip archive of the assignment. You have also 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. Upload **only** the given source files with your changes in a compressed archive before the deadline. Please comment your name **and** student number in at the top of each file.

Task 1: AvlTree [40]

Your task is to implement an AVL Tree that can be used to store data and re-balance it to achieve O(log N) time complexity.

You should use your own helper functions to assist in implementing the AVL tree.

AvlTree.java

`Node<T> insert(Node<T> node, T data)`

Insert the given element `data` into the tree and return the root node. Duplicates are not allowed. Return the given node if is a duplicate. Re-balance the tree after insertion.

`Node<T> removeNode(Node<T> root, T data)`

Delete the given element `data` from the tree and return the root node. Re-balance the tree after deletion. If the `data` is not in the tree, return the given node / root.

`Remember to set Node.height when you removeNode or insert.`

Task 2: Threaded AvlTree [60]

Your task is to convert task 1 to a Threaded AVL Tree. You should use your own helper functions to assist you in the conversion process.

ThreadedAvlTree.java

`void convertAVLtoThreaded(Node<T> node)`

Implement this function to convert a given AVL tree to a threaded AVL tree. Implement A right threaded.

`Node<T> insert(Node<T> node, T data)`

Insert the given element `data` into the tree and return the root node. Duplicates are not allowed. Return the given node if is a duplicate. Re-balance the tree after insertion.

`Node<T> removeNode(Node<T> root, T data)`

Delete the given element `data` from the tree and return the root node. Re-balance the tree after deletion. If the `data` is not in the tree, return the given node / root.

`Remember to set Node.height when you removeNode or insert.`

Only implement the methods listed above. You may use your own helper functions, such as methods for moving up or moving down an element, 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 Assignment website `https://ff.cs.up.ac.za/`. All tasks need to be implemented (or at least stubbed) before submission. Place **AvlTree.java** file in a zip or tar/gzip archive for task 1, and place **AvlTree.java and ThreadedAvlTree.java** files in a zip or tar/gzip archive for task 2 (you need to compress your tar archive) named uXXXXXXXX.zip or uXXXXXXXX.tar.gz where XXXXXXXX is your student number. Upload your archive to the *Assignment 2 - Task 1 and 2* slots respectively on the Assignment website. Submit your work before the deadline. No late submissions will be accepted.