Assignment 2

COS 212

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Objectives:

- Implementing an AVL tree with insertion and deletion operations.
- Combine a double threaded binary search tree with an AVL tree implementation.

General instructions:

- This assignment should be completed individually, and no group effort is allowed.
- As a result of the recent disruptions surrounding the COVID-19 pandemic, the assessment procedure for this assignment is to announced at a future date. In addition, the contribution of this assignment to the final course assessment, as well as the submission deadline is currently omitted for the same reason. Please check the course website regularly for relevant updates to these details. This specification will also be updated when such announcements are posted. Whatever decision is ultimately made regarding submission and mark contribution, an auto-marked assignment assignment submission will be made available in due course.
- You may not import any of Java's built-in data structures. Doing so will result in a mark of zero. If you require additional data structures, you will have to implement them yourself.
- If your code does not compile you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- Read the entire assignment before you start coding.

Plagiarism:

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.

After completing this assignment:

Upon successful completion of this assignment you will have implemented your own threaded AVL tree in Java, which supports efficient inorder and postorder traversals.

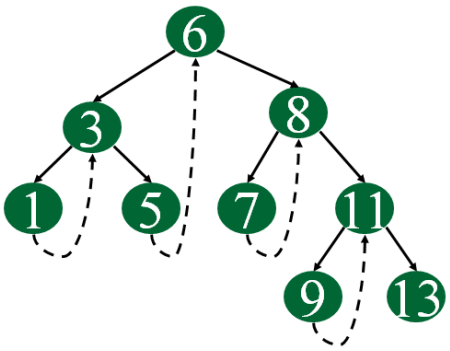
Your Task

Download the archive `assignment2Code.tar.gz` and untar it in an appropriate directory. The archive contains the following files:

- ThreadedAVLNode.java
- ThreadedAVLTree.java
- Tester.java

These files contain partial classes that you must complete.

- Class `ThreadedAVLNode` describes a threaded node for an AVL tree. A threaded node is a binary search tree node that re-uses unused right data fields as shortcuts to the inorder successor node, allowing for efficient inorder tree traversal that requires neither recursion nor a stack. It is also possible to implement non-recursive preorder and postorder traversals using the same threads. Threaded nodes for binary search trees are described in Section 6.4.3.1 of the textbook. The following is an example of a threaded binary search tree:



  The use of a threaded binary search tree to perform an inorder traversal was covered during lectures. For information on using a threaded binary search tree to perform a preorder traversal, refer to the last paragraph on page 240 of the textbook. A threaded AVL tree node is simply a threaded binary search tree node that additionally stores the balance factor that an AVL tree requires.

- Class `ThreadedAVLTree` describes a threaded AVL tree (i.e. an AVL tree where every node is a threaded node). AVL tree structure, as well as insertions and deletions for AVL trees, are described in Section 6.7.2 of the textbook. A threaded AVL tree is an AVL tree built out of threaded nodes. Crucially, the insertion and deletion operations must be augmented so that they correctly update the threads for nodes involved in tree rotations, where necessary. It is your task to determine when and how such threads need to be updated.

Implement all the methods specified in the provided classes. You are not allowed to modify provided method signatures, class names, or data fields. You are allowed to add additional methods and/or

data fields as necessary. Be sure to carefully follow all instructions provided in comments within the provided files.

Test your code very thoroughly. Test each method separately, and focus on edge cases (operations performed on empty trees, trees containing only one node, each of the AVL tree insertion and deletion cases, and so on). Make sure that all strings returned by the traversal methods conform to the specified format (check carefully for any deviations in format, including additional spaces at the end of the strings).

HINT: It is strongly advised that you start by focusing on the implementation of a standard AVL tree first, while ignoring threads. Only once you have a working AVL tree, shift your focus to adding support for threads and traversals. Before jumping to the code, understand the process visually by drawing threaded AVL trees on paper, performing various rotations on them, and observing how the existing threads must change. If you understand the process, you can write an algorithm in pseudocode to express it. If you can write the pseudocode, you can translate it to Java.

Submission Instructions:

You must create your own makefile, and include it in your submission. Your makefile will be overwritten during testing. The makefile used for testing will compile your code with the following command:

```
javac *.java
```

The makefile used for testing will also execute your program using the following command:

```
java Tester
```

Failure to include a makefile in your submission archive will result in your program failing to compile, and will result in a mark of zero.

Once you are satisfied that everything is working, you must tar your Java code together with the makefile, with no folders/subfolders. Your Java archive should be named sXXXass2.tar.gz, where XXX is your student number. Test your code thoroughly, with multiple test cases, before submitting it for marking.

Good luck!