Assignment 2

COS 212

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

Department of Computer Science
Deadline: 13/03/2017 at 12:00pm

Objectives:

- Implementing a double threaded binary search tree.
- Implementing a double threaded AVL tree.

General instructions:

- This assignment should be completed individually, **no group effort** is allowed.
- Only the output of your Java program will be evaluated.
- Your code may be inspected to ensure that you've followed the instructions.
- Be ready to upload your assignment tasks well before the deadline, as no extensions will be granted.
- You are NOT allowed to import any Java packages. Doing so you will receive a mark of 0. You must provide your own implementation of any additional data structures that you require.
- If your code does not compile you will be awarded a mark of 0.
- You will be afforded three opportunities to upload your submissions for automarking.

Plagiarism:

The Department of Computer Science regards plagiarism as a serious offence. Your code will be subject to plagiarism checks and appropriate action will be taken against offending parties. You may also refer to the the Library's website at `www.library.up.ac.za/plagiarism/index.htm` for more information.

After completing this assignment:

Upon successful completion of this assignment you will have implemented your own double threaded binary search tree, as well as a double threaded AVL tree in Java.

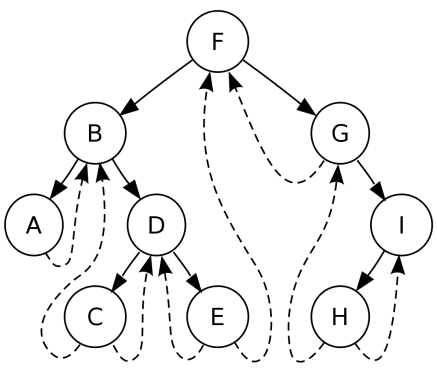
Task 1

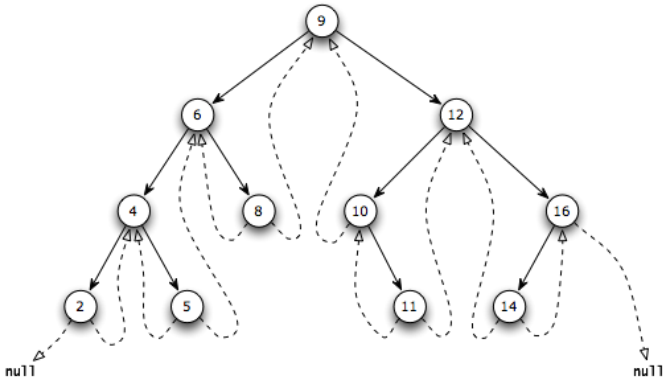
Download the archive `assignment2Code.tar.gz` and untar it in an appropriate directory. The archive contains the following files:

- DTNode.java
- DoubleThreadedBST.java
- DoubleThreadedAVL.java
- Tester.java

These files contain partial classes that you must complete. For Task 1, you will have to implement the `DTNode` and `DoubleThreadedBST`.

- Class `DTNode` describes a double threaded node. A double threaded node is a node that re-uses unused left and right data fields as shortcuts to the inorder predecessor and successor nodes, respectively, allowing for efficient **inorder** tree traversal that requires neither recursion nor a stack.
- Class `DoubleThreadedBST` describes a double threaded binary search tree, i.e. a binary search tree where every node is a double threaded node. Inserting into a double threaded BST is similar to inserting into a single threaded BST, with the difference that both the left and the right threads have to be maintained. Same applies to other operations such as deletion. Example double threaded BSTs:





The benefit of double threading is that efficient inorder traversal can be performed left to right (ascending order) or right to left (descending order).

Implement all the functions specified in the given classes. You are not allowed to modify method signatures or class names. You are allowed to add extra methods and/or member data fields as necessary.

Test your code thoroughly before proceeding to the next task.

Task 2

For this task, you will extend your double threaded tree into an AVL tree. An AVL tree re-balances itself every time a new node is added or deleted, thus ensuring the tree is efficient to search at all times. A double threaded AVL tree combines the benefit of no-stack bi-directional inorder traversal with efficient searching.

Class `DoubleThreadedAVL` describes a double threaded AVL tree that you have to implement. All methods of `DoubleThreadedBST` are inherited, but many will have to be overridden to suit the AVL needs. In particular, `DoubleThreadedBST` uses deletion by merging, while `DoubleThreadedAVL` must

use deletion by copying. And, of course, `DoubleThreadedAVL` must re-balance itself after every deletion/insertion. Corresponding threads will have to be **adjusted accordingly**. It is your task to devise an algorithm that will adjust the threads when the necessary rotations are performed.

HINT: Before jumping to the code, understand the process visually by drawing threaded 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, in which your code should be compiled with the following command:

```
javac *.java
```

Your makefile should also include a `run` rule which will execute your code if typed on the command line as in `make run`.

For your submission, 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/staff number. Separate upload links are provided for the two tasks. Each link grants 3 uploads. Every upload is a marking opportunity, do not use it to test your code. Test the code thoroughly, with multiple test cases, before submitting it for marking.

Good luck!