Document version: 1.1 (2015-11-15)

Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	ONG	Student ID:	19287368
Other name(s):	MING HANG		
Unit name:	Data Structures and Algorithms	Unit ID:	COMP1002
Lecturer / unit coordinator:	Valerie Maxville	Tutor:	Valerie Maxville
Date of submission:	21/05/2019	Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: ONG MING HANG Date of signature: 21/05/2019

(By submitting this form, you indicate that you agree with all the above text.)

Table of Contents

Documentation

TreeProfiler.java

This class allows to user to profile or interact any of the given trees. It starts with running the program with command-line arguments, when the user doesn't know how to run it properly and runs without any command-line arguments, it will display the usage details on how to run the program. Once the user reads the usage details, they can proceed to run the program normally. To run the profiling mode, the user must first enter "-p" as args[0] to choose the mode for profiling, then the user must enter the type of tree (e.g. BST, BT, TTFT) as args[1], data size as args[2] and input file as args[3]. If the number of command-line argument is correct, it will proceed to call the "profiler" submodule and prints out the tree statistic given by the type of tree that the user selected. The second mode, which enables the interactive environment of the program must be activated with "-i" as args[0]. If the input is correct, the program should display a list of options which allows the user to interact with one of the trees. When the user has made their choice, it would proceed to display a new menu depending on the choice they've selected, each of the tree has their own main menu method. The new menu covers a set of basic dynamic operations in which the user can select, it consists of an option to load new data into the tree, find a specific key in the tree, inserting a completely new node into the tree, deleting a specific node in the tree, display statistics of the tree and saving the tree as a text output, serialised file or binary export. The main menu method will keep looping until the user has chosen the quit option. This class is designed to be the main function of the entire program, built to allow the user to interact with the tree or display the profile of any of the given tree accompanied with statistics. Each of the tree main menu method may seem like a carbon copy of each other but some of the core functionality is different, such as how each tree class is constructed, how they load data from a file and how they save data into a file. This is due to the limitations of Serializable interface, when a file is serialized, loading the file back into the program requires it to be unique to itself (e.g. BlockTree can only load a serialized BlockTree file).

TreeIO.java

The main driver of the entire program, this method is responsible with the following function such as writing text output into a file, serializing a class into a file and loading a file back into the program. As seen in the code, each tree has their own writing, serializing or binary functions, except for load method which returns an object for any class but requires to be typecast and open binary returns a queue. This design can be further improved with the usage of inheritance but it requires the entire design to be revamped and cost time to do so. In the first place, the program was designed in a fashion that it would prioritize readability for the user, although it causes the code to be larger and may in fact lowers cohesion but due to some complexity issues, the usage of inheritance was scrapped.

UnitTestBTree.java|UnitTest234Tree.java|UnitTestBST.java

Unit test harnesses for each class and their methods. It test the constructor of the class alongside with the basic sets of dynamic operation that consists of testing insert method, find method, delete (for BST only) method, getSize method and getHeight method. It also test the average time it took in between each functions that interacts with the tree and gives a time estimation measured in nanoseconds. It will display the number of test it has passed out of the actual number of test depending on how to program is tested.

BlockTree.java

An implementation of advanced tree data structure that supports a limited range of basic set of dynamic operations including find, insert and delete. This class starts with a constructor to set a size for the B-Tree alongside with receiving a file input, in which the file input is then passed into a line parser in a while-loop that detects if it's the end-of-file and is then split into 8 variables, setting key as the same as ticker and the rest of the 7 variables is passed into StockClass to create a new stock with all of the information stored into one and will detect invalid lines, it is then automatically tossed into insert. Starting with insert function, which takes in a String key and Object value, in which value is actually a StockClass. Begins with the first condition, if the root of this B-Tree is null it signifies that the tree is indeed empty, thus it creates a new BlockNode representing the root node and both key and value is then inserted into the newly created node. The next condition checks if the root does not have childrens, it proceeds to a nested if-then-else statement, condition goes by if the root node's number of keys is lesser than maximum key size, it is then inserted normally, else if it's equal to maximum key size, it is then split and attempts to insert from the beginning again, also increments height because root is splitted upwards. Else it goes into a insert recursive method if there is an existing tree structure. Inside the recursive method, there are four cases to consider in which the base case is included, the base checks if the current node being passed in is a leaf node, if it is, insert (split when full), else iterate the tree until it finds the base case again. The other three case describe three condition, if the key to be inserted is lesser than the first index of the current node, it is then inserted recursively as the smallest key, else if the key is greater than the last index of the current node then it is inserted recursively as the greatest key. If both conditions doesn't match, it will search the entire node for a place to be inserted, setting "previous" as current index - 1 and next as "current" index, both of the variable is then used to traverse the current node, condition is if the key to be inserted is greater than "previous" and lesser or equals to "next", it is then inserted recursively. This all takes place in a while-loop which increments the index pointer throughout the current node and breaks when the condition is met. Next is the splitting method, it imports a node that is meant to be split, starts by grabbing the node's key size and using that to create a median index which is the value to be pushed upwards. Start by creating a left node and filling it up with values lesser than the median and it is then receiving all of the reference to any child that is lesser than median from the node-to-be-split. Same goes with the right node, instead it fills up

the right node with values greater than median and receiving reference to any child that is greater than median. After left and right node is fully initialised, it then check if the node-to-be-split has any parent, if there isn't, split upwards and made into a new root else if there are existing parent, just push up like so whilst checking if the parent is full, if it is just split. The find method works just the same as insert except it doesn't split and throws exception when a key isn't found, in the case which the node has no childrens, it keeps iterating until it reaches maximum key size, just means that there is nothing left to search. It also only returns a value which contains the information of StockClass. The deletion method isn't implemented due to its complexity, in the meantime it throws an exception.

TwoThreeFourTree.java

A specific type of B-Tree with an order of 4 in which the only difference is the splitting behaviour, every node with children (internal node) has either two, three, or four child nodes. Unlike B-Tree which only splits when it is required, 234-Tree splits when it encounters a full node. The rest of the method acts just like B-Tree (read above).

BinarySearchTree.java

A tree data structure implementation with at most 2 children, the left and right child. Each node contains a data, pointer to left child and pointer to right child, in which anything on the left is lesser than the root and anything on the right is greater than the root, this also applies to parent nodes (non-leaf nodes). It covers a set of basic operations, e.g. Find, Insert and Delete. It has both default and alternate constructor where default just makes an empty tree and alternate automatically inserts the values via file input. The find method recursively traverse the tree to find a specific key, in which returns a value when found. The insert function about the same with traversing, except it creates a new left node when it finds a correct spot to be inserted. The deletion method will consider all cases when it tries to delete, if the node is the root, a non-root node, a parent node and a leaf node. First it traverses the tree, find the correct key so it can be deleted, once it does find the key, set it to null. If the node to be deleted is the root, promotes the successor by getting the leftmost child of the right subtree. For the B-Tree and 234-Tree, getting the height would only involve splitting a node into a new root so it increments by one. For Binary Search Tree, it gets the longest pathway from root to leaf.

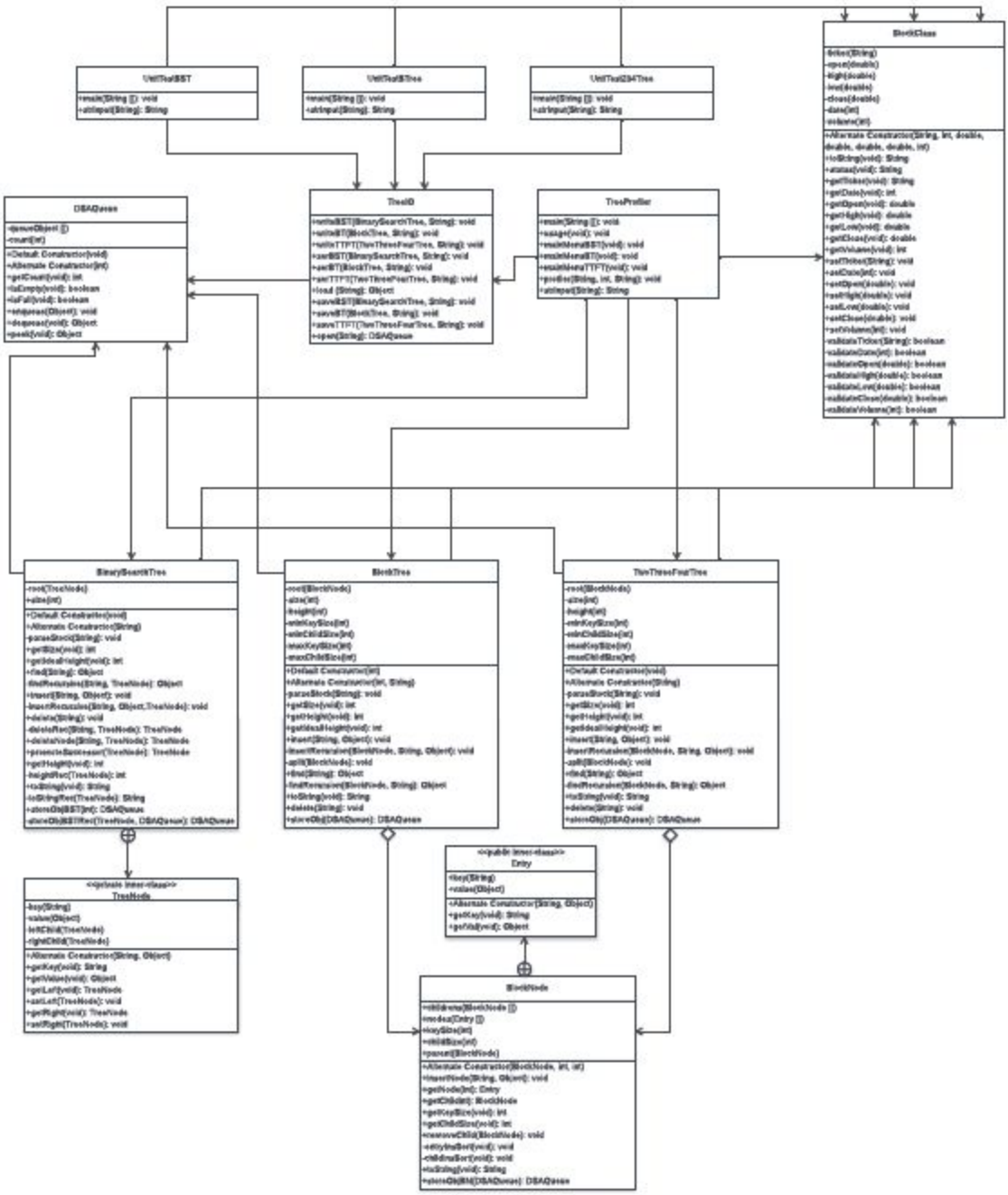
BlockNode.java

This class acts like the container class for both B-Tree and 234-Tree, it is responsible for adding node and child into the tree whilst sorting it in ascending order using insertion sort algorithm (It has the best time complexity for partially sorted data at $O(n)$). Inside the class exist an Entry class, which stores a reference to both String key and Object value, it is designed this way so the idea of String array and Object array would be scrapped and further optimizes readability and lessens complexity.

DSAQueue.java

An abstract data structure that stores value in FIFO (*first-in-first-out*) order so it is open on both ends. One end is always used to insert data which is enqueue and the other is used to remove data which is dequeue. Basically, a simple data structure that you can insert and remove data from it.

UML Diagram



Known bugs and issues

1. **Some keys does not appear when attempting to find it**
    a. How-to-replicate: Try it until you get an exception thrown by program.
2. **Stack overflow error**
    a. How-to-replicate: Serializing a degenerate binary tree.
3. **AIOB issues**
    a. How-to-replicate: Giving the B-Tree or 234-Tree a severely randomized sets of data.
4. **Some key isn't appearing as intended**
    a. How-to-replicate: For B-Tree, giving it a small order.
5. **The balance percentage giving weird symbols**
    a. How-to-replicate: When the order for B-Tree is too large, meaning when there is no height, it tries to get ideal height by using Math.log on 0.