Comparison of Binary Search Tree with a traditional unsorted array data structure

Vusi Skosana

05 April 2021

CSC2001F

Design

Three classes have been created, the other five have been obtained from UCT CSC2001F recourses. The created classes are as follows:

Students

A simple class. It has two attributes, student number and full name. It has get methods for that student number and full name, a *compareTo* method to compare attributes of the calling object to the one in the argument and lastly a *toString* method to display the student number and full name in one.

AccessArrayApp

A class to process information from a file into an array data structure, and operations can be performed to retrieve specific data. It has two attributes, an integer counter variable, and an array of type Students with a fixed length of 5000 entries. It has *readFile* method that processes entries, in the form of student number and full name, from a file into a Student type object then place them into the array. A *printStudent* method that has one parameter, a student number, this method loops through the created array and compares all Student type objects in the array if any of them match the student number, if a match is found, the full name is displayed, if not "Access denied" is displayed. A *printAllStudents* method, without parameters, it displays student number and full name of all objects in the array data structure.

AccessBSTApp

A class to process information from a file into a Binary Tree data structure, and operations can be performed to retrieve specific data. It has one attribute, a BinarySearchTree object, which is one of the other classes, an explanation of the other classes leading to a BinarySearchTree:

- BinaryTreeNode class, a generic class, has three attributes, one is the generic type (data) which type Student, and two others of its type (left and right) which are BinaryTreeNode<Students> type. It has a single constructor to assign all the attributes to values. It has two methods, getLeft and getRight that return the attributes of that object calling them.
- BinaryTree class, a generic class, has one attribute, a BinaryTreeNode<Students> root. It has seven methods. The getHeight method, to obtain height of the tree using recursion, getSize method, to obtain the number of nodes in the tree using recursion, visit method, to display the Student object attribute in that node, preOrder, postOrder, inOrder, and

levelOrder to display all the data in each node differently. The inOrder method uses BTQueue and BTQueueNode classes that help order and display nodes properly.

- BinarySearchTree class, a generic class extending BinaryTree class, containing two static integer attributes (modification of the BinarySearchTree class), opCount and insertCounter. The class contains six methods, insert method, that adds nodes to a binary tree, a find method, transverses through all nodes int the tree comparing them to student object and calls the upCounter method (modified the BinarySearchTree class), a upCounter method, it is an incrementor method that increments the opCount attribute after every comparison is made in the find method. A delete method transvers through all nodes in the tree and deletes a node with a specific student number. A findMin returns the most minimum node in a tree, and the removeMin method removes the most minimum node in the tree.

The AccessBSTApp contains four methods. The main method calls the readFile, printStudent, and printAllStudents, and display operational counts of the BinarySearchTree attribute. The readFile method, processes entries, in the form of student number and full name, from a file into a Student type object then place the objects in a node of a tree. A *printStudent* method that has one parameter, a student number, this method transverses through the created tree and compares all Student type objects in the binary tree data structure, if a match is found, the full name is displayed, if not "Access denied" is displayed. A *printAllStudents* method, without parameters, it displays student number and full name of all objects in the binary tree data structure.

Description of Experimental Setup

The experiment was performed on a desktop workstation with 8 GB RAM and Core i3 Processor.

Trial Assessments

Programs were assessed using known valid and invalid student numbers, where the printStudent method is invoked, both programs produced the output as follows.

+ for valid student number

Student Number	*Output*
DLMLET017	Lethabo Dlamini
VNXWAR008	Warona Van
VSGALU001	Alunamda Visagie

+ for invalid student number

Student Number	*Output*
JFRMGD467	Access denied!

FGTNBY425	Access denied!
VJMYSE666	Access denied!

+ when assessed without student numbers, the printAllStudents method is invoked producing the following output of the first and last five lines.

AccessArrayApp	AccessBSTApp
MLLNOA014 Noah Maluleke WTBJAY001 Jayden Witbooi KHZOMA010 Omaatla Khoza MLTLUK019 Luke Malatji NKNTHA021 Thato Nkuna .... MSXROR015 Rorisang Mosia DNLAYA006 Ayabonga Daniels CHKOFE015 Ofentse Chauke MNGREA015 Reatlegile Moeng SHBCAL017 Caleb Shabangu	MLLNOA014 Noah Maluleke KHZOMA010 Omaatla Khoza WTBJAY001 Jayden Witbooi CHKONT018 Onthatile Chauke MGLLET011 Lethabo Mogale … … STHLEB017 Lebogang Sithole STHLET015 Lethabo Sithole STHLIA010 Liam Sithole STHLIN006 Lingomso Sithole MDSMEL022 Melokuhle Modise STHLIK021 Likuwe Sithole

Experiment Objective

The purpose of this experiment is to compare the Binary Search Tree with a traditional unsorted array data Structure, to demonstrate the speed difference for searching between BinarySearchTree and the traditional array, both implemented in java using a real-world application.

Experiment Execution

Firstly, Instrumentation was performed on both programs, a counter variable was positioned in the programs where comparison of student numbers was done, when student numbers are compared, the counter variable is incremented.

The experimental procedure was as follows:

- I created ten equally spaced subsets having entries randomly selected from the oklist.txt file.
- The subsets are 1 to 10 having, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, and 5000 entries that are randomly ordered inside.
- For every subset, insert all the subset entries into a the AccessBSTApp and AccessArrayApp programs.
- Use each entry of that subset, run the programs using a student number in that entry but also searching for that student number against the subset that it is it., i.e., java AccessArrayApp/AccessBSTApp student number, where the student number is going to be searched in its own subset.
- For a match found in the programs, that entry in the subset is replaced by the full name and the number of the counter variable, namely the comparisons the program made before it found that student number in that subset.

- This is performed for all entries in all subsets.

This takes a lot of time to perform, I created a python script that automated the procedure.

Data

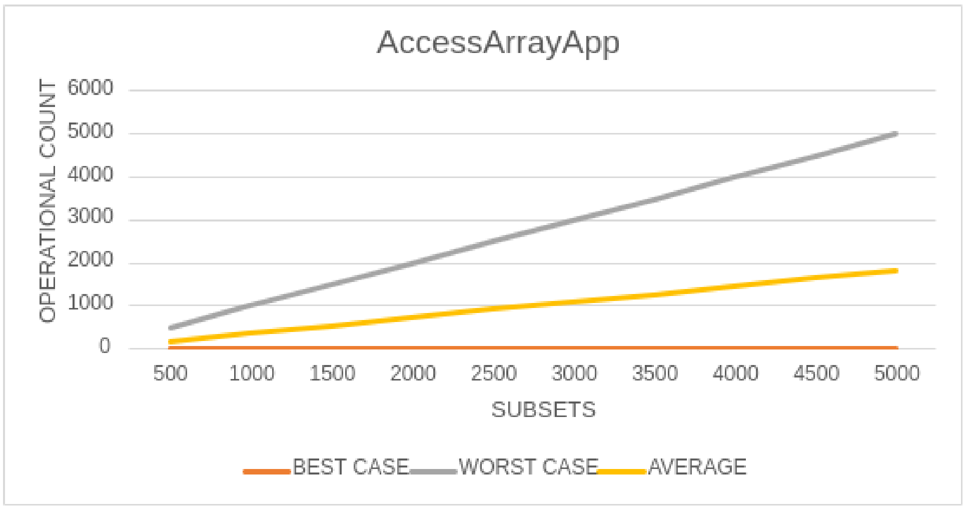
AccessArrayApp

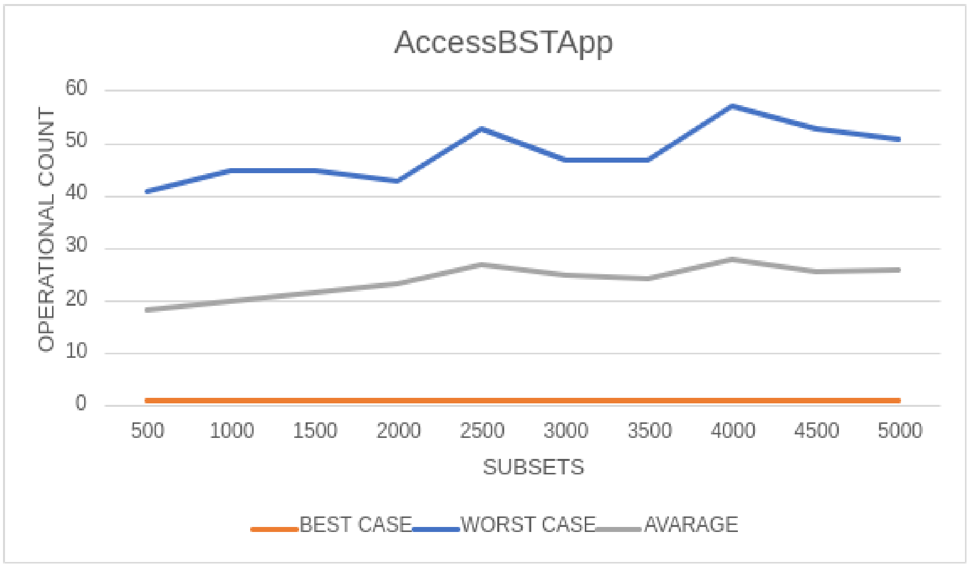
SUBSETS	BEST CASE	WORST CASE	AVARAGE
500	1	500	176.70
1000	1	1000	362.15
1500	1	1498	548.14
2000	1	2000	752.06
2500	1	2498	930.91
3000	1	2996	1087.88
3500	1	3497	1272.14
4000	1	4000	1474.75
4500	1	4500	1674.91
5000	1	4997	1832.75

AccessBSTApp

SUBSETS	BEST CASE	WORST CASE	AVARAGE
500	1	41	18.24
1000	1	45	19.88
1500	1	45	21.71
2000	1	43	23.24
2500	1	43	26.86
3000	1	47	24.98
3500	1	47	24.41
4000	1	57	28.10
4500	1	53	25.67
5000	1	51	25.99

Analysis of data





Discussion of results.

In both programs, the best case is O (1). As the number of elements in array data structure increase, the worst-case increases proportionally in order n, O(n) meaning more operations are performed to retrieve data. As the number of elements in the binary tree data structure increase, the worst-case increases in order logn, O(logn), meaning less operations are performed to retrieve data from a tree.

Conclusion.

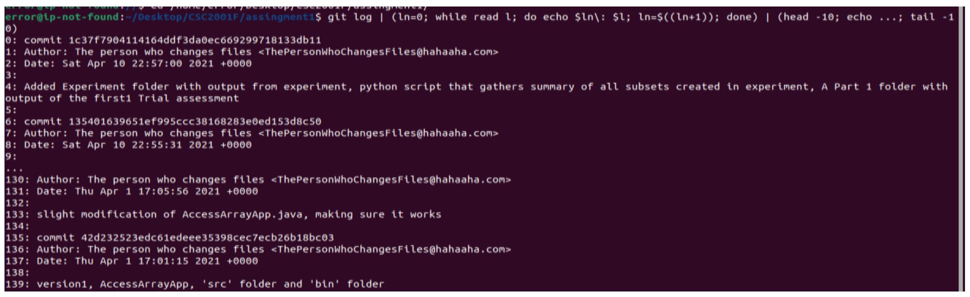
A Binary Tree data structure is more efficient in data retrieval than a linear array data structure.

Creativity

Instrumentation was performed in AccessBSTApp, in order to obtain how many numbers of comparisons that have been made in the binary tree data structure in order to find a node with a student number that is being searched. I modified or inserted a counter attribute in the BinarySearchTree class, but the counter attribute had to be incremented every time a comparison of student numbers is made.

- The find method with two parameters has two places where it compared student number.
    - if (d.compareTo (node.data) == 0)
    - return node;
    - else if (d.compareTo (node.data) < 0)
    - return (node.left == null) ? null : find (d, node.left);
    - else
    - return (node.right == null) ? null : find (d, node.right);
- In order to count the comparisons, I incremented the counter variable was before the first comparison.
    - opCount++;
    - if (d.compareTo (node.data) == 0)
    - return node;
- But for the second comparison, I couldn't increment the counter just before the second comparison as that would interfere with the if and else if statements, so came with a different idea to make a *upCounter* method that would increment the counter,
    - public static boolean upCouter(){
    - opCount++;
    - return true;
    - }
- but where would I put this method? This is where I put it.
    - else if (upCouter() & d.compareTo (node.data) < 0)
- The reason for this placement here is, I could not place it after the (d.compareTo (node.data) < 0) evaluation because if it is to be false then the *upCouter* method would not be executed. So, I put it before, so that it is true, then increment the opCount variable, followed by evaluating the second statement, meaning that, if the second statement is true, the code below the else if would run, if it is false, the code would not be executed.

Summary statistics of git usage



```
error@ip-not-found:~/Desktop/CSC2001F/assingment1$ git log | (ln=0; while read l; do echo $ln\: $l; ln=$((ln+1)); done) | (head -10; echo ...; tail -1
0)
0: commit 1c37f7904114164ddf3da0ec669299718133db11
1: Author: The person who changes files <ThePersonWhoChangesFiles@hahaaha.com>
2: Date: Sat Apr 10 22:57:00 2021 +0000
3:
4: Added Experiment folder with output from experiment, python script that gathers summary of all subsets created in experiment, A Part 1 folder with
output of the first1 Trial assessment
5:
6: commit 135401639651ef995ccc38168283e0ed153d8c50
7: Author: The person who changes files <ThePersonWhoChangesFiles@hahaaha.com>
8: Date: Sat Apr 10 22:55:31 2021 +0000
9:
...
130: Author: The person who changes files <ThePersonWhoChangesFiles@hahaaha.com>
131: Date: Thu Apr 1 17:05:56 2021 +0000
132:
133: slight modification of AccessArrayApp.java, making sure it works
134:
135: commit 42d232523edc61edeee35398cec7ecb26b18bc03
136: Author: The person who changes files <ThePersonWhoChangesFiles@hahaaha.com>
137: Date: Thu Apr 1 17:01:15 2021 +0000
138:
139: version1, AccessArrayApp, 'src' folder and 'bin' folder
```