CSC2001 ASSIGNMENT 1

DATA STRUCTURES: BINARY SEARCH TREE

Yaseen Hull
HLLYAS001

Contents

What is the problem?

The task aim is to explore the efficiency of the Binary Search Tree data structure. By comparing linear structures such as an array list we can identify the speed and memory used in executing tasks in either one. The task is therefore to write an electronic phonebook system in java by implementing a Binary Search Tree data structure to keep record of entries and perform various applications on the data held by the database. To test out the BST two main methods have been developed. The first exists to print all the data out chronologically using a specific String key within the entry. The second method serves the purpose of finding an entry based on a specific string key which is input into the data structure and returning the full value of the entry. Before either of applications can run the test data provided must be concatenated to just the 'name' of the entry. The name will therefore be used as a key and hence it is the main value in determining the output of the two methods. Furthermore, an array list containing all the entries will need to be created and similar search function must be formulated to identify entries in the list based on a key given.

Design

The design comprises of two main applications both of which traverses though the Binary Tree structure to find or order the information contained within nodes. The tree nodes in a BST are linked to the root/ head node and are connected to either right or left branch of the root depending on its value. The value is determined by the key and in the case of the electronic phone book the key is a string name of the contact entry. Hence the structure is formed with respect to the alphabetical order after the root node is inserted. If the entry after the root node is inserted has a key which is positioned alphabetically higher than the root key then it is inserted in the right branch. Else if it is positioned alphabetically lower then it is linked to the left branch. Furthermore if an entries key is higher chronologically than the root but lower than the parent in the right branch it is then positioned on the left branch of the right parent node. A similar conclusion can be drawn from an entry with a key value lower than the root key. It will then either be on the left or right subtree depending on its key on the left parent. The two methods designed for this application has been named printIt and searchIt and are discussed below.

PrintIt method

The printIt method prints every entry in the database in alphabetical order. If the key value was an integer the pattern produced would be ordered in terms of numbers, minimum to maximum. For the printIt method to operate the minimum value must be found, that is the name (key) beginning with the letter 'A'. The method therefore takes in the root node and checks if it is not null. Once this test is passed the left node is called using the 'getLeft' method from the Binary Tree Node class. The 'getLeft' method traverses to the left node from the root and returns the node value. But before returning the value another test must be passed. To find the minimum entry in the database the left nodes will be called up to the point where the node value equals null. This says that only when no more left nodes exists pass the node we currently on, can we return the value/ contact information of the node. The getleft method itself is passed through the printIt method to check if its null. A sort of loop is initiated to traverse right down the tree and then the last node in the left most branch is printed before going on to the 'getright' method which finds the right child. Once again the 'getright' method is also passed through the printIt method and traversed down to its minimum value before printing its entry.

The complete left hand side of the root node is done this way until all entries are printed after which the root is printed. The right side of the root is then traversed right down to its' minimum value to be printed and thereafter employs the same methods as before to print each nodes information until the node value is null.

The printIt method is generally known as the 'in order' traversal and from its results it is rightly named so.

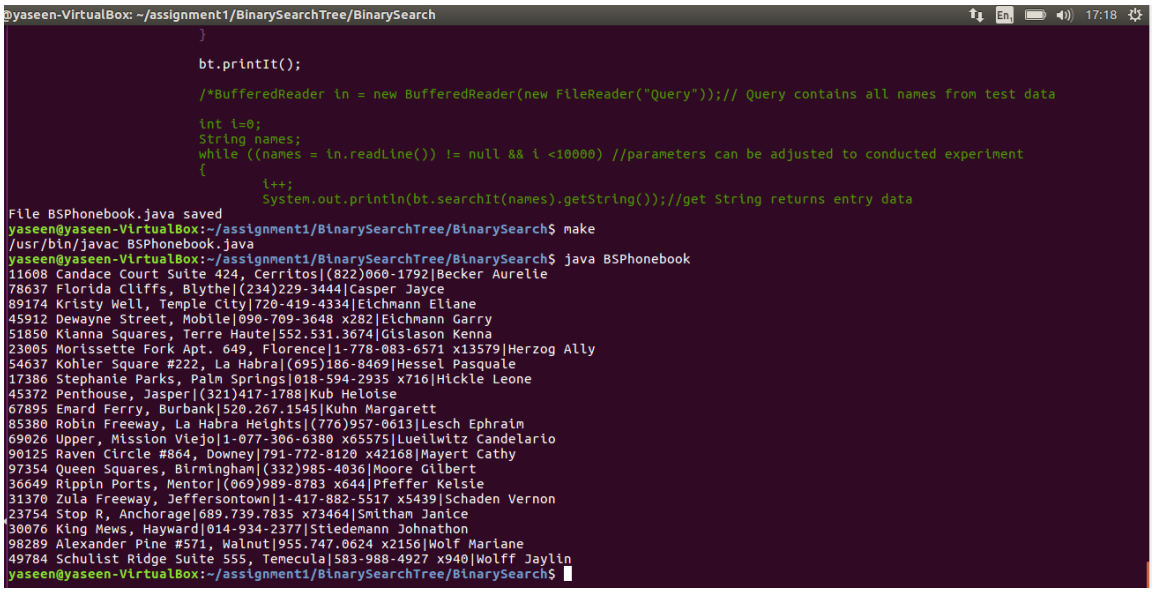
SearchIt method

Similarly to the printIt method, the 'SearchIt' method requires a database to exist to be operational. However, if the root is null the search method would just return null. The 'SearchIt' method requires an input value to compare with. In the case of the phonebook the key was set as the name of the contact which was also the input for the search method. A query file was created containing just the names of the contacts in the phonebook to use as a comparison. If the root was not null the string name and the node root were taken as arguments for the method. Three cases existed for this comparison. The first checked if the name input was the same as the key of the root. The second checked if the name input was chronologically lower than the root key and the last checked if it was chronologically higher than the root key.

The first case of the search method compares string values of the name and the key from the node v. If they're the same the node entry (contact information) is returned and printed to the screen. If this isn't the case the next condition is initiated. The second case finds which branch the name input lies in. Here the node argument of the method is used more extensively. If the input name is less than the key of the node argument then the returned value depends on a condition. If the left node connected to the node used as an argument is equal to null then null will be returned else the searchIt method will once again be invoked to find the left node which acts as argument in the search method. This process continues until the first case is satisfied. Similarly, if the name input given to the searchIt method was chronologically higher than node argument, the third case would initiate and instead of the left node condition the right node condition will appear.

Code input and outputs

PrintIt output



QueryFile input

SearchItLinear output

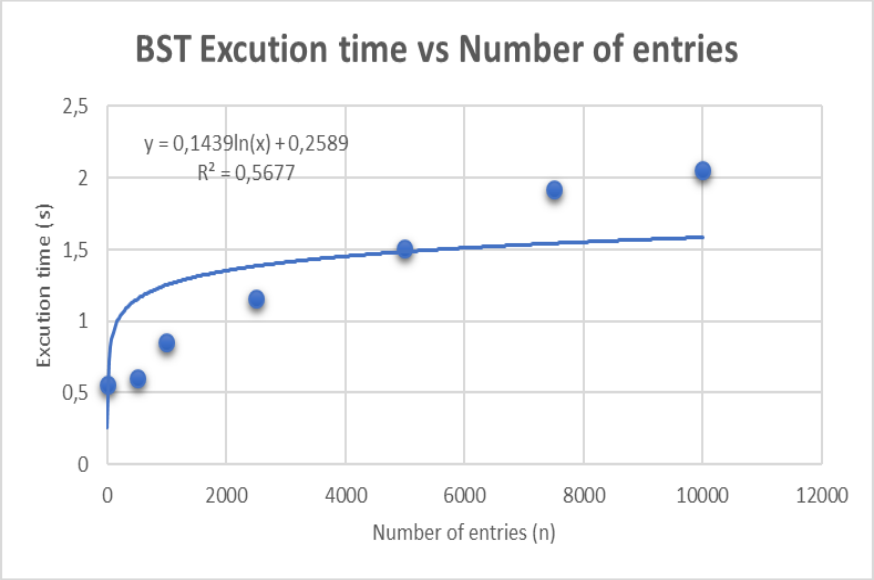
```
                    }
                    // System.out.println(entryList);

                    BufferedReader in = new BufferedReader(new FileReader("QueryFile"));
                    int j = 0;
                    String names;
                    while ((names = in.readLine()) != null) // && j <10000)
                    {       j++;
                            SearchItLinear(names,entryList);
                    }
                     return;

File LinearPhonebook.java saved
yaseen@yaseen-VirtualBox:~/assignment1/BinarySearchTree/Linear$ make
/usr/bin/javac LinearPhonebook.java
yaseen@yaseen-VirtualBox:~/assignment1/BinarySearchTree/Linear$ java LinearPhonebook
23754 Stop R, Anchorage|689.739.7835 x73464|Smitham Janice
67895 Emard Ferry, Burbank|520.267.1545|Kuhn Margarett
90125 Raven Circle #864, Downey|791-772-8120 x42168|Mayert Cathy
51850 Kianna Squares, Terre Haute|552.531.3674|Gislason Kenna
17386 Stephanie Parks, Palm Springs|018-594-2935 x716|Hickle Leone
97354 Queen Squares, Birmingham|(332)985-4036|Moore Gilbert
89174 Kristy Well, Temple City|720-419-4334|Eichmann Eliane
69026 Upper, Mission Viejo|1-077-306-6380 x65575|Lueilwitz Candelario
98289 Alexander Pine #571, Walnut|955.747.0624 x2156|Wolf Mariane
31370 Zula Freeway, Jeffersontown|1-417-882-5517 x5439|Schaden Vernon
54637 Kohler Square #222, La Habra|(695)186-8469|Hessel Pasquale
45912 Dewayne Street, Mobile|090-709-3648 x282|Eichmann Garry
36649 Rippin Ports, Mentor|(069)989-8783 x644|Pfeffer Kelsie
30076 King Mews, Hayward|014-934-2377|Stiedemann Johnathon
78637 Florida Cliffs, Blythe|(234)229-3444|Casper Jayce
11608 Candace Court Suite 424, Cerritos|(822)060-1792|Becker Aurelie
45372 Penthouse, Jasper|(321)417-1788|Kub Heloise
49784 Schulist Ridge Suite 555, Temecula|583-988-4927 x940|Wolff Jaylin
23005 Morissette Fork Apt. 649, Florence|1-778-083-6571 x13579|Herzog Ally
85380 Robin Freeway, La Habra Heights|(776)957-0613|Lesch Ephraim
yaseen@yaseen-VirtualBox:~/assignment1/BinarySearchTree/Linear$
```

SearchIt output

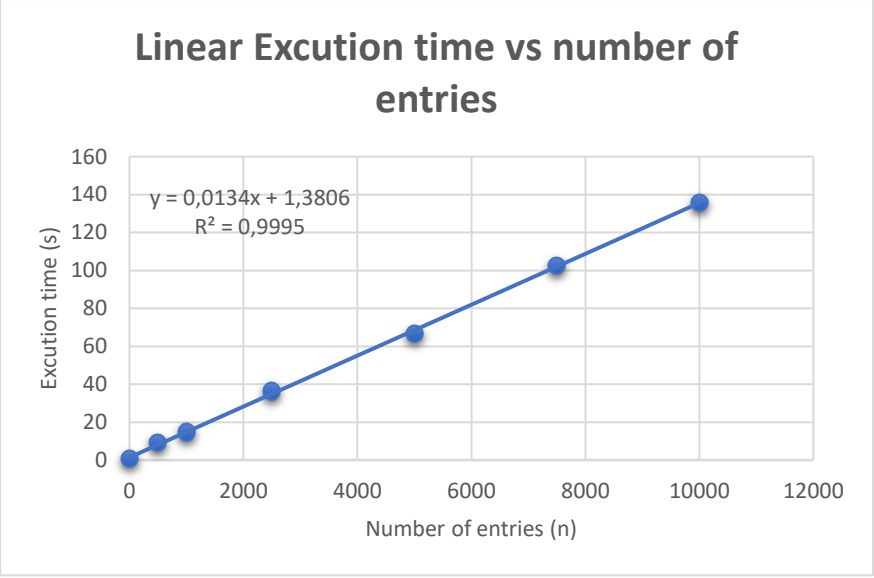
```
                {
                        System.out.println("File not found. ");
                }
                catch(IOException ioexception)
                {
                        System.out.println("File input error occured!");
                }
                catch(NullPointerException e)
                {
                        System.out.println("Not Found");
                }

File BSPhonebook.java saved
yaseen@yaseen-VirtualBox:~/assignment1/BinarySearchTree/BinarySearch$ make
/usr/bin/javac BSPhonebook.java
yaseen@yaseen-VirtualBox:~/assignment1/BinarySearchTree/BinarySearch$ java BSPhonebook
23754 Stop R, Anchorage|689.739.7835 x73464|Smitham Janice
67895 Emard Ferry, Burbank|520.267.1545|Kuhn Margarett
90125 Raven Circle #864, Downey|791-772-8120 x42168|Mayert Cathy
51850 Kianna Squares, Terre Haute|552.531.3674|Gislason Kenna
17386 Stephanie Parks, Palm Springs|018-594-2935 x716|Hickle Leone
97354 Queen Squares, Birmingham|(332)985-4036|Moore Gilbert
89174 Kristy Well, Temple City|720-419-4334|Eichmann Eliane
69026 Upper, Mission Viejo|1-077-306-6380 x65575|Lueilwitz Candelario
98289 Alexander Pine #571, Walnut|955.747.0624 x2156|Wolf Mariane
31370 Zula Freeway, Jeffersontown|1-417-882-5517 x5439|Schaden Vernon
54637 Kohler Square #222, La Habra|(695)186-8469|Hessel Pasquale
45912 Dewayne Street, Mobile|090-709-3648 x282|Eichmann Garry
36649 Rippin Ports, Mentor|(069)989-8783 x644|Pfeffer Kelsie
30076 King Mews, Hayward|014-934-2377|Stiedemann Johnathon
78637 Florida Cliffs, Blythe|(234)229-3444|Casper Jayce
11608 Candace Court Suite 424, Cerritos|(822)060-1792|Becker Aurelie
45372 Penthouse, Jasper|(321)417-1788|Kub Heloise
49784 Schulist Ridge Suite 555, Temecula|583-988-4927 x940|Wolff Jaylin
23005 Morissette Fork Apt. 649, Florence|1-778-083-6571 x13579|Herzog Ally
85380 Robin Freeway, La Habra Heights|(776)957-0613|Lesch Ephraim
yaseen@yaseen-VirtualBox:~/assignment1/BinarySearchTree/BinarySearch$
```

Comparison Experiment

Results



n	t
1	0,5565
500	0,6065
1000	0,8545
2500	1,16
5000	1,51
7500	1,922
10000	2,053
Average	1,2375



n	t
1	0,513
500	9,089
1000	14,673
2500	36,211
5000	66,462
7500	102,509
10000	135,932
Average	52,19843

Discussion

The graphs above display the correlation between the execution search time and the number of data entries in the book. The first graph displays the data using a Binary Search Tree (BST) data structure whereas the second uses an array list. The results claim that the correlation between the two variables are approximately 0.5 for the BST and 0.9 for the Linear method. It is evident that times are much faster in searching for entries in the BST data base. This is a direct result of a BST cutting the range of values to search when the value inserted in the searchIt function Is compared to the root. The value

is either the root or in the left or right branch. Thus, the function traverses down the right or left branch omitting the other branch completely from the search and hence reducing time taken to find a phonebook entry. The Linear search on the other hand goes through each entry in the list until the correct value is found.

Conclusion

Several conclusions can be drawn from this experiment. Firstly, the two search methods are close in terms of correlation coefficient although the BST has much better time frame. The BST is as much as 40% faster, this was calculated from the average time of both methods for all values of n. The BST also has a much better search functionality as it reduces the number of nodes to visit from the moment the search method is executed. From the linear graph, the number of entries is directly proportional to the time taken for the program to complete its search. The BST on the other hand follows a logarithmic curve which implies that more entries result in better times. The ratio of time execution to number of entries given therefore decreases. To increase the results accuracy a better performing pc could have been used and a shell script written to run the method several times.