Muhammad Azhar Rohiman – RHMMUH005 – Assignment 1 Report.

Description of problem:

A telephone directory is given to be made into a Java application by using a Binary Search Tree as a data structure to store the information. The data we are given come in the following format: "address|phone_number|full_name". The full name must be used as the key for data manipulation.

The first application, PrintIt, has to take in all the data entries from a data file and store them in the Binary Search Tree. It then has to output the data in alphabetical order and this is done by doing an inorder traversal of the tree.

Another application, called SearchIt, must be created to search for specific data entries in the Binary Search Tree by reading a list of names from a query file. The data file, containing all the data entries must be loaded into the BST and the program will output the full entry for each query. Although the queries will only consist of the names, the application should output the data in its full format, which is: "address|phone_number|full_name".

The last application, SearchItLinear, does the same job as SearchIt, except that it makes use of an unsorted array rather than Binary Search Tree.

An experiment should be carried out to show the difference in execution speed when searching for items between a Binary Search Tree and an unsorted array.

Application Design:

1) PrintIt

The PrintIt program implements a Binary Search Tree as a data structure to store data entries. It also outputs the data alphabetically using an inorder traversal. Since the key being used here is the full name, a Person class has been made to separate the data entry into different instance variables: 'name', 'address', 'number'.

PrintIt has a variable to type Person so that the data entries can be broken down using String methods such as indexOf and substring to extract the names, address and numbers. The 'testdata' file is read by the program and a 'while' loop is used to go through all the entries and process each one by breaking them into names, addresses and numbers.

The constructor in the Person class is invoked taking the name, address and number as parameters. The instance variables in the Person class are initialized.

To store the data entries in the BST, a method (treeInsert) from the BinarySearchTree class is called taking a parameter of type Person. This means that the BST will store the data of type Person in its nodes. When the data entries are being stored in the BST using the treeInsert method, it checks the names using the compareTo method by calling the getter method for name from the Person class. If the

name is lower than the one in the parent node, the data entry linked to the former name is stored in the left node if there are no more left nodes, and if it is higher, then it is stored in the right node.

PrintIt has one more method (treeList) from the BinarySearchTree class to output all the data entries, using the names as key, in alphabetical order. This method is recursive as it searches for the last node on the left-side until there is a node with no child. Once that node is found, it is printed and the method returns to the previous node (which is the parent of the node which was recently printed). The latter is then printed, and a call to the function is made again for the right child. It checks for any left child recursively. If the node has no left children, it gets printed. Then checks for a right child. All the subtrees on the left-hand side of the root node are processed before printing the root node. After that, the same process gets repeated for all the subtrees on the right-hand side.

2) SearchIt

The SearchIt program implements a Binary Search Tree as a data structure to store data entries and serves to read queries from a text file to traverse the BST and check if there are entries with names related to the queries. It also has a variable of type Person to break down the data entry into name, address and number.

A while loop is used to process each data entry in the text file and is stored as a variable of type Person in the BST using treeInsert from the BinarySearchTree class.

When the insertion operation is over, the program asks a user to input the name of a query file with names. Another while loop is used for this operation. For every name being read from the text file, the method treeContains from the BinarySearchTree class is called to check if there is a match for the query name in the BST.

If the method treeContains returns null, it means the tree is empty and has no data entries to compare. Since objects of type Person are stored in the BST, and they have instance variables, name, address and number, the getter method for name is called from the Person class to access the name in the nodes. If the name matches with the root node, the data in that node is returned and printed in its full form. However, if the name is lower than the name in the root node, it recursively calls the treeContains method in the left subtree until it reaches the node which has the same name. If at some point, it reaches a node where the name is higher than the name in that node, it will call the function recursively in the right subtree from that node. And if the name is not found after reaching the last node in the BST, null is returned. If the query name is higher than the root node, the treeContains method will be recursively called in the right subtree of the BST until it finds the name or null is returned.

PrintIt first 20 lines of output:

03707 Botsford Fork, Lima|489-848-7299|Abbott Alec
44812 Wilderman Mountain, Vallejo|318.679.5603 x712|Abbott Alexandria
76400 Barton Fields #044, Cerritos|507.340.1186|Abbott Alia
02519 Zackery Village, San Mateo|602.992.4016|Abbott Brando

88126 Bruen Common, Beverly Hills|788.603.8604|Abbott Elwyn
51832 Bayer Pass, Simi Valley|1-035-079-0176 x61480|Abbott Hosea
87191 Suite Z, Selma|823.283.2198 x7192|Abbott Ima
27010 Sanford Center, Stanton|822.752.1004|Abbott Josh
17296 Elta Crossroad #362, Newport Beach|516-835-0116|Abbott Leann
18565 Suite B, Fountain Valley|1-117-789-3061|Abbott Meda
22345 Runte Garden, Steubenville|1-654-279-2374|Abbott Murray
32763 Langosh Route, San Diego|297-763-2822|Abbott Novella
90282 Haag Keys, Garden Grove|(681)856-6604 x642|Abbott Rahsaan
52000 Marques Loaf #288, Placentia|(961)238-9093|Abbott Sadye
78469 Renner Mill, Agoura Hills|1-515-459-1556|Abbott Santina
96179 Feil Tunnel #352, Canton|1-052-394-1236 x29668|Abernathy Amparo
98827 Gerlach Pike Apt. 743, Apple Valley|1-486-893-0367|Abernathy Austyn
14576 Harber Knolls, Riverside|1-331-934-0147|Abernathy Catalina
23694 Pier F, Tempe|(552)753-8320 x85031|Abernathy Chadd
36296 Batz Walk, San Francisco|(637)882-6835 x72457|Abernathy Cicero

SearchIt and SearchItLinear 20 entry query file & output:

1) Query file

Wolf Mariane
Hickle Leone
Moore Gilbert
Lehner Monica
Schaden Vernon
Stiedemann Johnathon
Little Elroy
Casper Jayce
Witting Marcellus
Eichmann Eliane
Pfeffer Kelsie
Herzog Ally
Langosh Rae
Gislason Kenna
Wolff Jaylin
Wilkinson Destiny
Zulauf Pamela
Ruecker Kenyon
Nolan Linnie
Little Marvin

2) Output

98289 Alexander Pine #571, Walnut|955.747.0624 x2156|Wolf Mariane
17386 Stephanie Parks, Palm Springs|018-594-2935 x716|Hickle Leone
97354 Queen Squares, Birmingham|(332)985-4036|Moore Gilbert

10548 McGlynn Rest #124, Richmond|1-311-460-2067|Lehner Monica
31370 Zula Freeway, Jeffersontown|1-417-882-5517 x5439|Schaden Vernon
30076 King Mews, Hayward|014-934-2377|Stiedemann Johnathon
98642 Ondricka Inlet Suite 390, Laguna Hills|1-622-051-1330 x0465|Little Elroy
78637 Florida Cliffs, Blythe|(234)229-3444|Casper Jayce
20865 Lang River, Whittier|(023)266-0111 x96257|Witting Marcellus
89174 Kristy Well, Temple City|720-419-4334|Eichmann Eliane
36649 Rippin Ports, Mentor|(069)989-8783 x644|Pfeffer Kelsie
23005 Morissette Fork Apt. 649, Florence|1-778-083-6571 x13579|Herzog Ally
15088 Pete Well, La Jolla|597.105.7768 x61822|Langosh Rae
51850 Kianna Squares, Terre Haute|552.531.3674|Gislason Kenna
49784 Schulist Ridge Suite 555, Temecula|583-988-4927 x940|Wolff Jaylin
02368 Bo Knoll, Diamond Bar|1-739-539-4411|Wilkinson Destiny
97809 Cayla Turnpike, Palmdale|327-095-5441 x0442|Zulauf Pamela
46517 Micheal Landing Suite 366, Sunnyvale|1-608-379-9357|Ruecker Kenyon
44317 Jacobi Corners #617, Pico Rivera|(390)231-4430 x7435|Nolan Linnie
67100 Floor 5, Barrow|649.774.7975 x34697|Little Marvin

Comparison Experiment:

To compare the execution time of the programs, SearchIt and SearchItLinear, an application named QueryFile is created to extract the names from testdata and store them in an arraylist. A for loop is then used to write the subsets of 'n' number of names, starting from n=1 up to n=10000, into a text file, named Query.

Using the "time" Unix application, multiple readings were taken for both SearchIt and SearchItLinear, for n=1, n=100, n=250, n=500, n=1000, n=2000,…, n=9000, n=10000, where n is the number of names.

Below are the tables and graphs given by the results of the experiments.

n	time/s
1	0.224
100	0.236
250	0.252
500	0.265
1000	0.284
2000	0.368
3000	0.372
4000	0.396
5000	0.448
6000	0.476
7000	0.496
8000	0.512
9000	0.528
10000	0.532

Figure 1: Table of values for SearchIt

n	time/s
1	0.236
100	0.216
250	0.232
500	0.27
1000	0.416
2000	0.568
3000	0.86
4000	1.24
5000	1.712
6000	2.328
7000	2.864
8000	3.536
9000	4.176
10000	5.24

Figure 2: Table of values for SearchItLinear

Figure 3: Graph for SearchIt program time execution.

Figure 4: Graph for SearchItLinear program time execution.

As the number of names increases, the SearchItLinear program takes more processing time. This is because one name must go through all the elements until it gets a match. If the data entry we are looking for is at the start of the array, then the number of searches decreases. However, if it is at the end, it has to go through most of the data entries.

With SearchIt, either the entire left subtree or the entire right subtree is considered when doing a search depending on whether the name is lower or higher than the root node. Therefore, even with a high number of names in the query file, the processing time is much faster using a BST, as shown on the graph.

We can conclude that searching for items in a Binary Search Tree is faster than in an unsorted array.