**COMP20003 Assignment 1 Report**

**Sid:1025379**

**User:luuh**

**Introduction:**

For this assignment, I construct a BST to contain the data from the input file. A BST is a binary tree where nodes are ordered in the following way:

* each node has one key
* the keys in the left subtree are smaller than its parent node’s key
* the keys in the right subtree are greater than its parent node’s key

For insertion, I will begin at the first node (the root) and recursively go down to find a location to insert a new node. If the key is already in the tree, I will add the data to the linked list so that one key can hold many data then continue recursively to the end of the input file.

For doing the dict1, I am required to search for keys in a BST so I look at the root first and compare the keys stored at the root with the searching keys. If they don’t match up, I recursive either to the left if the comparison is negative and go to right child otherwise until I find the matching keys.

So that searching in a BST has O(h) worst-case runtime complexity, where h is the height of the tree and for the best case, it is O(log(n)) where n is the number of nodes.

To verify this time complexity theory, I will create 10 input files which different sizes to insert the nodes for the BST. The first input file will contain only 1 variable. The second one will contain 5 random variables include the duplicated one. The third one will have 10 variables and so on. Then I will have 10 input files which have these different size {1,5,10,50,100,500,1000,5000,10000,50000). For the searching keys, I will generate 10 different values to get the number of comparisons to find the searching keys and then take the average numbers of these comparisons.

For doing dict2 which is required to code a function to take an instance in the data from the same BST as an input and return the all the keys from the records which match the input, using in-order tree traversal. An in-order tree traversal will go all the nodes in the tree in this order: visit the left child, then the parent and the right child and when it finds the matching value, it will print to the output file.

So that searching using the in-order tree traversal in a BST has a constant O(n) time complexity which n is the total node in the tree or in other words, the total numbers of inputs.

To verify this time complexity theory, I will do the same thing as above, create 10 input files which contain {1,5,10,50,100,500,1000,5000,10000,50000) variables respectively and create the searching keys as requirement to test the numbers of comparisons.

**Stage 1:**

For stage 1, thanks to the method as mentioned above, I have this table:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Number of elements in the input | 1 | 5 | 10 | 50 | 100 | 500 | 1000 | 5000 | 10000 | 50000 |
| Average number of comparisons | 1 | 2.2 | 3.2 | 5.4 | 6.5 | 8.7 | 9.8 | 12.1 | 13.0 | 15.4 |
| Log(n) of number of inputs | 0 | 2.32 | 3.32 | 5.64 | 6.64 | 8.96 | 9.96 | 12.29 | 13.29 | 15.6 |

Using these values, I have this graph to show the relationship between the growth of inputs and number of comparisons:

**A close up of a flower

Description automatically generated**

According to the graph above, the time complexity of searching in BST tree is almost as same as the O(log(n)) so the result is verified with the theory.

**Stage 2:**

For stage 1, thanks to the method as mentioned above, I have this table:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Number of elements in the input | 1 | 5 | 10 | 50 | 100 | 500 | 1000 | 5000 | 10000 | 50000 |
| Average number of comparisons | 1 | 5 | 10 | 50 | 100 | 500 | 1000 | 5000 | 10000 | 50000 |
| N of number of inputs | 1 | 5 | 10 | 50 | 100 | 500 | 1000 | 5000 | 10000 | 50000 |

Using these values, I have this graph to show the relationship between the growth of inputs and number of comparisons:

**A picture containing wall, text

Description automatically generated**

According to the graph above, the time complexity of searching in-order traversal tree is the same as O(n) so the result is verified with the theory.

**Discussion:**

With these experiments, I can verify that the average time complexity for searching in BST is approximate O(log(n)). However, the numbers of comparisons usually smaller than O(log(n)) though the differences are tiny. This maybe because for this assignment, I can have duplicated keys but different data for each node. Therefore, the numbers of comparisons can reduce due to these duplicated keys.

In the other hand, for the in-order traversal tree, I have to check the data from every keys include all the data of duplicated keys to find the matching values so I have to travel all the nodes and every data contains in each node. Therefore, the number of comparisons will be the numbers of inputs in each input file and the time complexity for this will be O(n).