Name: Georges Hatem

CS 590 Homework 3: Binary Search Trees Creativity Exercises

Due Date: February 13, 2022

Problem 3.6.17:

In this exercise, we are searching through the Binary Search Tree (BST) T to locate all numbers with k = key(v), where v is a node of a Binary Search Tree (BST) T. The differences between this exercise and the Algorithm TreeSearch(k,v) in Figure 3.2.4 is that the Algorithm TreeSearch(k,v) in Figure 3.2.4 searches for the value k equal to the value of key(v) and return the node v where the value of k is equal to the value of key(v). However, in this exercise, we will have duplicate numbers. Therefore, we need to return all nodes where k is equal to key(v). The way to do that is by adding a collection a that has all the v nodes where k is equal to key(v). The Algorithm TreeSearch(k,v) in Figure 3.2.4 return v when k = key(v). Since we have duplicate numbers in this exercise, and we need to find all numbers, then we need to add statement as follows when k = key(v):

First, we need to add the node v to the collection a, we do that as follows:

$$a.addLast(v)$$

Second, we need to continue searching the right child of this node v that we added to the collection (since same keys are usually allowed on the right side), and this should keep searching to find if there is any duplicate, we do this as follows:

$$return\ findAllElements(k, T.right(v), a)$$

This program ends when we reach an external node (when v is an external node)

The pseudocode for this Algorithm is as follows:

Algorithm findAllElements(k, v, a):
Input: The search key k, a node of the Binary Search Tree (BST) v, and a collection (a) of all nodes v when k = key(v)
Output: A collection (a) of all nodes v when k = key(v)

```
if v is an external node then
    return a
if k = key(v) then
    a.addLast(v)
    return findAllElements(k, T.right(v), a)
else if k < key(v) then
    return findAllElements(k, T.left(v), a)
else
    return findAllElements(k, T.right(v), a)
```

Now, let's show that this method runs in time O(h + s) where h is the height of T and s is the number of items returned:

We are calling the function recursively until we get to an external node. So, basically, we are starting from the Tree root and keep moving right or left until we get to an external node. So, from that we can see that in the worst case scenario, we are traversing the height of the Binary Search Tree (BST). Also, when k = key(v), we are also calling the Algorithm method again through the return call. From that, we can see that the return call in k = key(v) gets called everytime k = key(v) (for every duplicate item where k = key(v)). This means that the comparison to check whether k = key(v) and the return call in k = key(v) are called s times where s is the number of items returned.

Summarizing this with what I wrote about the height above, we obtain the following:

The running time of the Algorithm in this exercise is O(h + s), where h is the height of T, and s is the number of items returned.

Problem 4.7.25:

As indicated in Section 4.3 and in Table 4.3.1, the structural changes to the find method are none. This means that we use the same Algorithm to search whether we are using a Binary Search Tree or an AVL Tree.

In this exercise, we are searching through the balanced tree T to locate all numbers with k = key(v), where v is a node of the balanced tree T. The differences between this exercise and the Algorithm TreeSearch(k,v) in Figure 3.2.4 is that the Algorithm TreeSearch(k,v) in Figure 3.2.4 searches for the value k equal to the value of key(v) and return the node v where the value of k is equal to the value of key(v). However, in this exercise, we will have duplicate numbers.

Therefore, we need to return all nodes where k is equal to key(v). The way to do that is by adding a collection a that has all the v nodes where k is equal to key(v). The Algorithm TreeSearch(k,v) in Figure 3.2.4 return v when k = key(v). Since we have duplicate numbers in this exercise, and we need to find all numbers, then we need to add statement as follows when k = key(v):

First, we need to add the node v to the collection a, we do that as follows:

$$a.addLast(v)$$

Second, we need to continue searching the right child of this node v that we added to the collection (since same keys are usually allowed on the right side), and this should keep searching to find if there is any duplicate, we do this as follows:

$return\ findAllElements(k, T.right(v), a)$

This program ends when we reach an external node (when v is an external node)

The pseudocode for this Algorithm is as follows:

Algorithm findAllElements(k, v, a):
Input: The search key k, a node of the Balanced Tree v, and a collection (a) of all nodes v when k = key(v)
Output: A collection (a) of all nodes v when k = key(v)

```
if v is an external node then
    return a
if k = key(v) then
    a.addLast(v)
    return findAllElements(k, T.right(v), a)
else if k < key(v) then
    return findAllElements(k, T.left(v), a)
```

else

return findAllElements(k, T.right(v), a)

Now, let's show that this method runs in time O(log(n) + s) where n is the number of elements stored in the tree and s is the number of items returned:

We are calling the function recursively until we get to an external node. So, basically, we are starting from the Tree root and keep moving right or left until we get to an external node. So, from that we can see that in the worst case scenario, we are traversing the height of the balanced tree. Also, when k = key(v), we are also calling the Algorithm method again through the return call. From that, we can see that the return call in k = key(v) gets called everytime k = key(v) (for every duplicate item where k = key(v)). This means that the comparison to check whether k = key(v) and

the return call in k = key(v) are called s times where s is the number of items returned.

Summarizing this with what I wrote about the height above, we obtain the following:

The running time of the Algorithm in this exercise is O(h + s), where h is the height of T, and s is the number of items returned.

From Theorem 4.3.1 in Section 4.3, the height (h) of an AVL Tree T storing n items is O(log(n)). And since the running time of the Algorithm in this exercise is O(h + s), this means that the running time of the Algorithm in this exercise is O(log(n) + s), where n is the number of elements stored in the tree and s is the number of items returned.

Therefore, the running time of the Algorithm in this exercise is O(log(n) + s), where n is the

number of elements stored in the tree and s is the number of items returned.