CSCI – B505 Applied Algorithms

Programming Assignment – 5

Submitted by – Vishal Singh (singhvis)

Part 1

1. **Insert(key):** We traverse the graph left or right depending upon whether the new element is smaller or greater than the current node
2. **Contains(key)**: We traverse the graph left or right depending upon whether the element to be searched is greater or smaller than the current node
3. **Inorder()**: We keep going left in the tree until we obtain NULL and print the value at the node then for the sub-trees we print the right element
4. **Size()**: traverse through all sub-trees from to right
5. **Smallest()**: keeps going on the left branch till the last value and records it, by the property of BST, returns the smallest value
6. **Largest()**: keeps going on the right branch till the last value and records it, by the property of BST, returns the largest value
7. **Successor(key):** returns the successor by considering the following cases
    a. If the key has a right branch, then returns the smallest right branch value
    b. If the key does not have a right branch and key itself is a left branch (condition – key != parent.right) then returns the parent
    c. If the key does not have a right branch and key itself is a right branch, then it travels up to find a parent of which the key is a part of left subtree
8. **Predecessor(key):** return predecessor by considering the following cases
    a. If the key has a left branch, then returns the largest value of left branch
    b. If the key does not have a left branch and key itself is a right branch, then returns the parent
    c. If the key does not have a right branch and key itself is a right branch, then it travels up to find a parent of which the key is a part of left subtree

Function	Time Complexity	Space Complexity
Insert(key)	O(h) – traverse down left or right	O(n)
Contains(key)	O(h)	O(1)
Inorder()	O(n) – traverse through all the nodes	O(1)
Size()	O(n)	O(1)
Smallest() / Largest()	O(h) – traverse down left or right	O(1)
Successor()	O(h) – follow path up or down	O(1)

Predecessor()	O(h))	O(1)

Where n is the number of elements or nodes in the tree and h is the height of tree, which can be n in the worst case

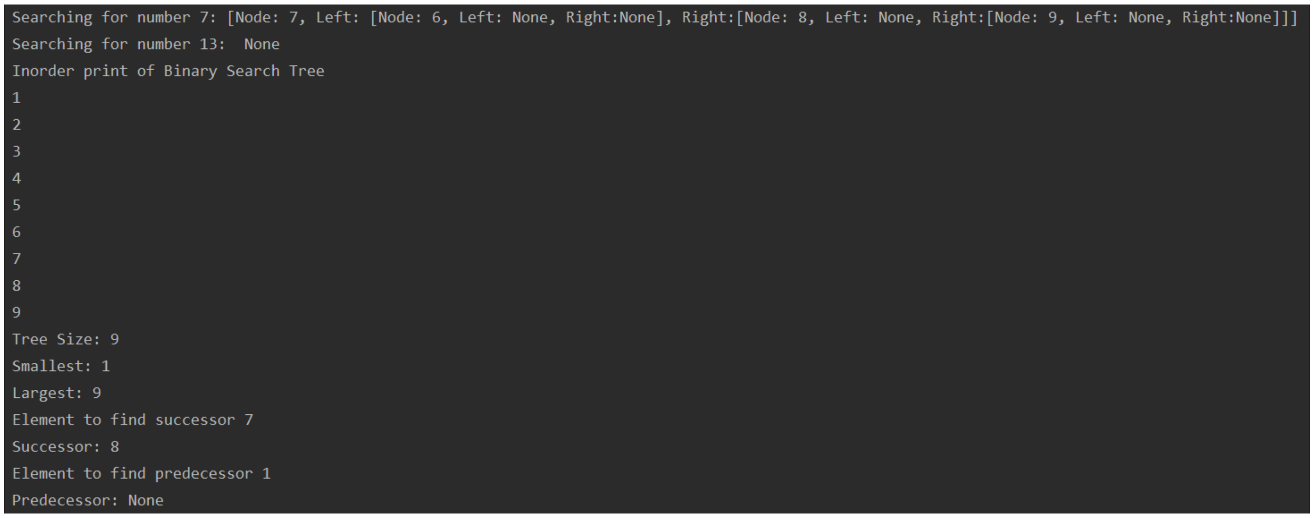
Part 2

The Greater Sum Tree is implemented in following steps-

1. We traverse the Binary Search Tree in the Depth First Search manner starting from right most element
2. When initialize a variable 'sum_element' which stores the sum of values of all the nodes visited
3. We replace each node value with the value of `sum_element` at that instant
4. Due to property of BST, we visit the nodes in decreasing order of their value, therefore, each value is replaced by the sum of the values of which are bigger than that value

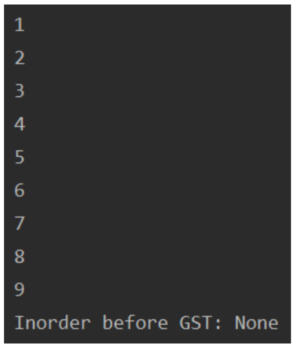
Output-

Part 1

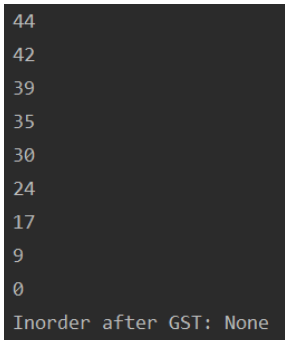


```
Searching for number 7: [Node: 7, Left: [Node: 6, Left: None, Right:None], Right:[Node: 8, Left: None, Right:[Node: 9, Left: None, Right:None]]]
Searching for number 13:  None
Inorder print of Binary Search Tree
1
2
3
4
5
6
7
8
9
Tree Size: 9
Smallest: 1
Largest: 9
Element to find successor 7
Successor: 8
Element to find predecessor 1
Predecessor: None
```

Part 2



```
1
2
3
4
5
6
7
8
9
Inorder before GST: None
```



```
44
42
39
35
30
24
17
9
0
Inorder after GST: None
```