Submitted By - Sanyam Rajpal
CSCI-B-505 Applied Algorithms
(FA19 - BL - 11503)
Programming Assignment - V

WRITE-UP

I used Python language for my code because it's easy to understand and reading the text files and storing it in an apt format is very simple. During my coding, I did make some syntax errors which I resolved and then ran the code perfectly.

I built a class with Binary Seach Tree Objects initialized pointers and using the value passed in the arguments.

Here n is the number of nodes and h is the height of the BST. Height can vary from Log2(n) to n that's why I have used h variable as is just for a better description of the complexities. The worst time and time complexity is for a BST with only the right(or left) children. Here the recursion stack memory is considered. **If we don't consider recursion stack space at** all the **worst-case space complexity will be O(1)** for all the functions stated above. If we consider recursion stack, the worst-case space complexity is as follows.

Function	Worst Time Complexity	Worst Space Complexity
Recursive Insert	O(h)	O(h)
Recursive Contains	O(h)	O(h)
Recursive Size	O(n)	O(h)
Iterative MinValue	O(h)	O(1)
Iterative MaxValue	O(h)	O(1)
Recursive Inorder	O(n)	O(h)
Iterative Predecessor	O(h)	O(1)
Iterative Successor	O(h)	O(1)
Iterative GreaterSumTree	O(n)	O(h)

Output -
Inorder BST Traversal
2
3
4
5
6
7
8
BST Contains 5
True
Size of BST
7
Minimum value of BST
2
Maximum value of BST
8
Predecessor of 5 in BST
4
Successor of 5 in BST
6
Calling GreaterSumTree for BST
Inorder BST Traversal after calling GreaterSumTree
33
30
26
21
15
8
0

Conclusion
As can be seen from the outputs of the inorder traversal before and after calling the GreaterSumTree function has changed and all the values of the tree are now replaced by there maximum values. The predecessor, successor, contains, minimum, maximum, size are returned correctly.

Discussion

The answers stated above are for the worst-case as stated above. I have written the average and worst-case answers below for reference, but the worst-case answers are same as above.

I built a recursive Insert function which inserts value maintaining the property of BST. The worst-case and average time complexity is O(h). The worst-case and the average space complexity for a recursion stack is O(h).

I built a recursive Contains function which used the BST properties to efficiently return the boolean answer. The worst-case and the average time complexity is O(h). The worst-case and the average space complexity for a recursion stack is O(h).

I built a recursive Size function which calculates the number of nodes in BST. The worst-case and the average time(both) complexity is O(n). The worst-case and the average space complexity for a recursion stack is O(h).

I built an iterative Min value function which efficiently calculates the minimum value in BST using its properties. The worst-case and average time complexity is O(h). The average and worst-case space complexity is O(1).

I built an iterative Max value function which efficiently calculates the minimum value in BST using its properties. The worst-case and the average time complexity is O(h). The average and worst-case space complexity is O(1).

I built a recursive Inorder traversal function which prints the node values in Inorder format. The worst-case and the average time complexity is O(n). The worst-case and the average space complexity for a recursion stack is O(h).

I built an iterative predecessor function which calculates the predecessor without having the need of a parent node. The worst-case and the average time complexity is O(h). The average and worst-case space complexity is O(1).

I built an iterative successor function which calculates the predecessor without having the need of a parent node. The worst-case and the average time complexity is O(h). The average and worst-case space complexity is O(1).

I built a recursive greaterSumTree function which calculates the sum of all the values greater than the current value in the BST and replaces the root value with that. So, we recur through the rightmost value and keep on updating that value which will be counted for all the lesser values in the left subtrees and thus becomes computationally and memory efficient. Because we use just 1 element array and just keep on updating that. We don't use

variable because otherwise, it may not use previously calculated values for the smaller values. The worst-case and average time complexity is O(n). The average and worst-case space complexity are O(h).

It was more difficult to code because we couldn't directly access a parent but it made it more general and useful. It was a great learning experience building these 2 functions without involving any parent node and exploiting the properties of the current node and its children.

I referred to the book, slides for referring to the pseudocode for most of the functions written above. And to calculate the predecessor and the successor of a BST without using its parent BST I referred to the following link for reference and conceptually understanding it.
https://cppsecrets.com/users/308114971101061051161146410411111610997105108469911110 9/Python-Program-to-find-inorder-predecessor-and-successor-of-a-key-input-by-the-user-in-a-Binary-Search-TreeBST-without-using-recursion-.php