CS566HW7-2024

October 23, 2024

0.1 Lab 7. Introduction to algorithms

This is seventh homework for CS 566.

0.2 Task 1. Solve the problem "Insert Into BST" from https://leetcode.com/problems/insert-into-a-binary-search-tree/description/ using Python3.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

[1]:
```python
from collections import deque
from typing import Optional, List
# Definition for a binary tree node.
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def insertIntoBST(self, root: Optional[TreeNode], val: int) -> Optional[TreeNode]:
        if not root:
            return TreeNode(val)
        if val>root.val:
            root.right = self.insertIntoBST(root.right,val)
        elif val<root.val:
            root.left = self.insertIntoBST(root.left,val)
        return root
```

0.2.1 Do not modify the testing code below. If you get message "Mistake in test case #", it means that you algorithm is incorrect.

[2]:
```python
# do not modify testing code
def preorder_step(node, result):
    if not node:
        return
```

```
    result.append(node.val)
    preorder_step(node.left, result)
    preorder_step(node.right, result)

def preorderTraversal(root: Optional[TreeNode]) -> List[int]:
    ls = []
    preorder_step(root, ls)
    return ls

#test_case_1
root = TreeNode(4, TreeNode(2, TreeNode(1), TreeNode(3)), TreeNode(7))
val = 5
expected = [TreeNode(4, TreeNode(2, TreeNode(1), TreeNode(3)), TreeNode(7,␣
 ↪TreeNode(5))),
            TreeNode(5, TreeNode(2, TreeNode(1), TreeNode(3)), TreeNode(7,␣
 ↪TreeNode(4)))]
actual = Solution().insertIntoBST(root, val)
pr_actual = preorderTraversal(actual)
pr_expected = [preorderTraversal(tree) for tree in expected]
assert pr_actual in pr_expected, "Mistake in test case 1"
print("OK")
```

```
OK
```

0.2.2 Write analysis of the Memory Complexity and Time Complexity using Aymptotic Notation O. (1 point)

Memory Analysis: O(h), where h is height of tree

Time Analysis: O(h), where h is height of tree

0.3 Task 2. Solve the problem "Delete in BST" from https://leetcode.com/problems/delete-node-in-a-bst/description/ using Python3.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

[3]:
```
from typing import Optional

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
```

```python
    def deleteNode(self, root: TreeNode, key: int) -> TreeNode:
        # base
        if not root:
            return root

        # search node to delete
        if key>root.val:
            root.right = self.deleteNode(root.right,key)
        elif key<root.val:
            root.left = self.deleteNode(root.left,key)
        # found node to delete
        else:
            # for 0 child or 1 right child, return right child
            if root.left is None:
                return root.right
            # for 1 left child, return left child
            elif root.right is None:
                return root.left
            # two children, find min in right subtree
            else:
                # go to leftmost node in rightsubtree
                curr = root.right
                while curr.left:
                    curr = curr.left
                # swap val with delete node val
                root.val = curr.val
                # update rightsubtree recursively, where we delete the swapped␣
 ↪value
                root.right = self.deleteNode(root.right,root.val)
        return root
```

[4]:
```python
# do not modify testing code
def preorder_step(node, result):
    if not node:
        return

    result.append(node.val)
    preorder_step(node.left, result)
    preorder_step(node.right, result)

def preorderTraversal(root: Optional[TreeNode]) -> List[int]:
    ls = []
    preorder_step(root, ls)
    return ls

#test_case_1
```

```
root = TreeNode(5, TreeNode(3, TreeNode(2), TreeNode(4)), TreeNode(6, None,␣
↪TreeNode(7)))
val = 3
expected = [TreeNode(5, TreeNode(4, TreeNode(2)), TreeNode(6, None,␣
↪TreeNode(7))),
            TreeNode(5, TreeNode(2, None, TreeNode(4))), TreeNode(6, None,␣
↪TreeNode(7))]
actual = Solution().deleteNode(root, val)
pr_actual = preorderTraversal(actual)
pr_expected = [preorderTraversal(tree) for tree in expected]
assert pr_actual in pr_expected, "Mistake in test case 1"
print("OK")
```

```
OK
```

0.3.1 Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(h), where h is height of tree

Time Analysis: O(h), where h is height of tree

0.3.2 Task 3. Theoretical Question.

As we have seen on lecture Binary Search Trees can be created differently for the same list of n keys. Some tress can be close to optimal (with the height close to log(n)), and some tress can be very suboptimal (with the height close to n/2), but still all these trees are valid Binary Search Trees for given list of n keys.

Derive mathematically how many valid Binary Search Trees can be created for the list of n distinct keys. Hint: you will come up with a very useful formula which is used in combinatorics. Write it on a list of paper and embed it here as a screenshot or write with markdown inside of colab notebook.

Good Luck!

Using n as number of distinct keys, we can use Catalan numbers to determine the number of distinct BSTs, which is given by:

$C_n = \frac{1}{n+1}\binom{2n}{n}$