Is It A Red-Black Tree

Problem

Author:陈硕

Number:3220106406

Date:2023-11-10

**I.Introduction**

**Is It A Red-Black Problem** is a problem focusing on **judging whether a binary-search tree is a red-black tree** which is an advanced data structure, owing some features:

(1) Every node is either red or black.

(2) The root is black.

(3) Every leaf (NULL) is black.

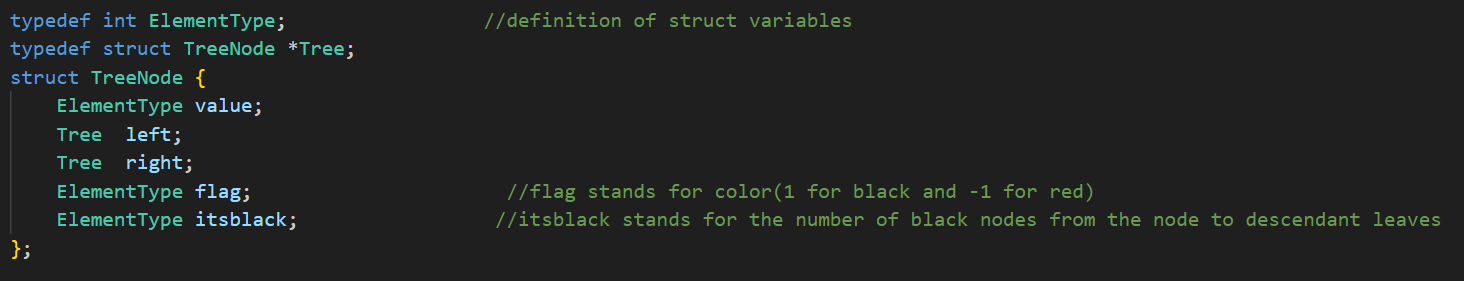
(4) If a node is red, then both its children are black.

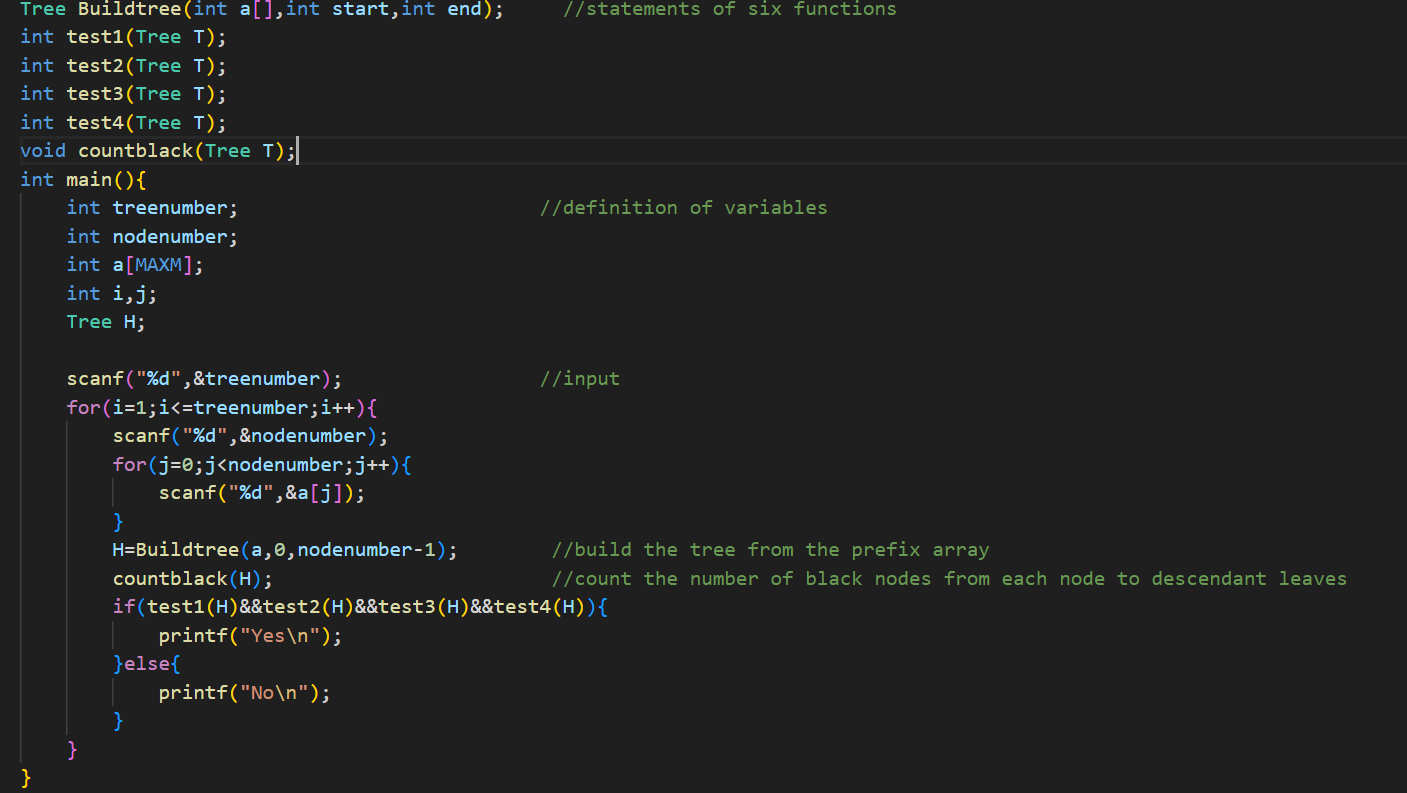
(5) For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

**This project** **aims to find an efficient method to successfully tell whether a given prefix array can build a red-black tree.**

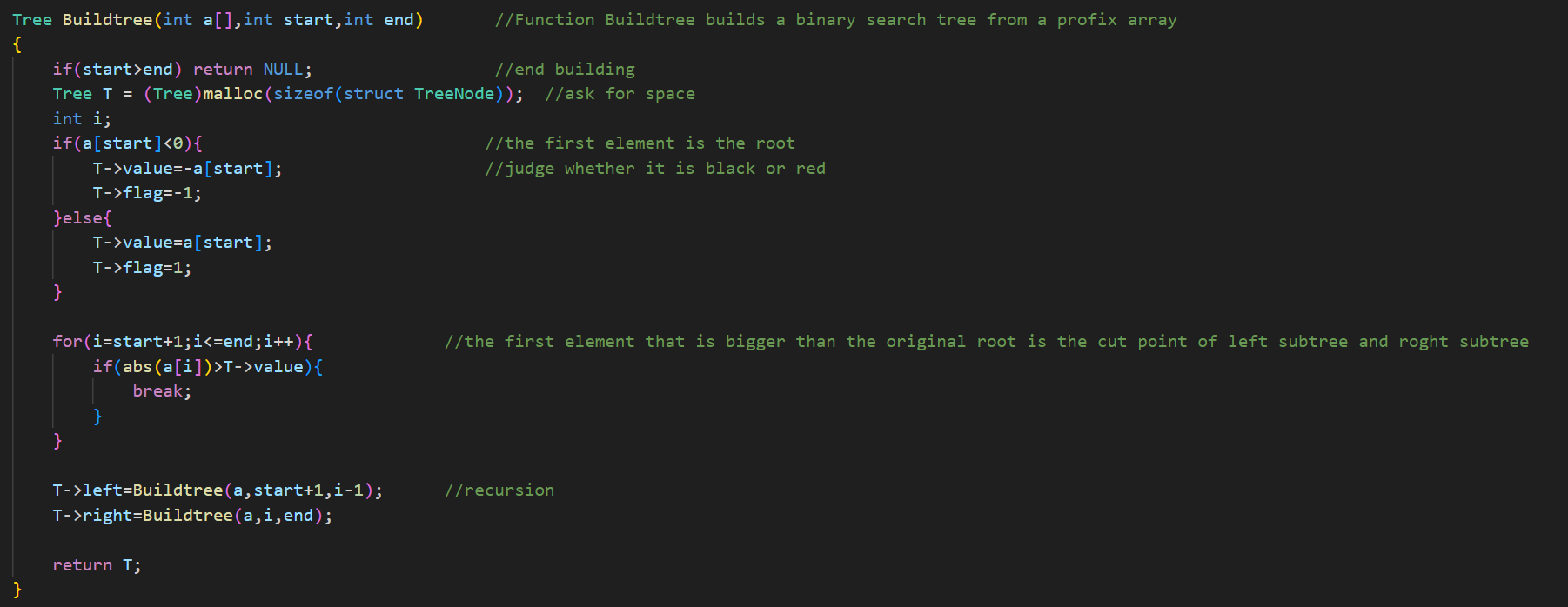
**II.Algorithm Specification**

Firstly,a struct variable “Tree” is defined.Besides normal elements like value, point of left and right node,I add flag to stand for color of node and add itsblack to stand for the number of black nodes from a code to descendant leaves.



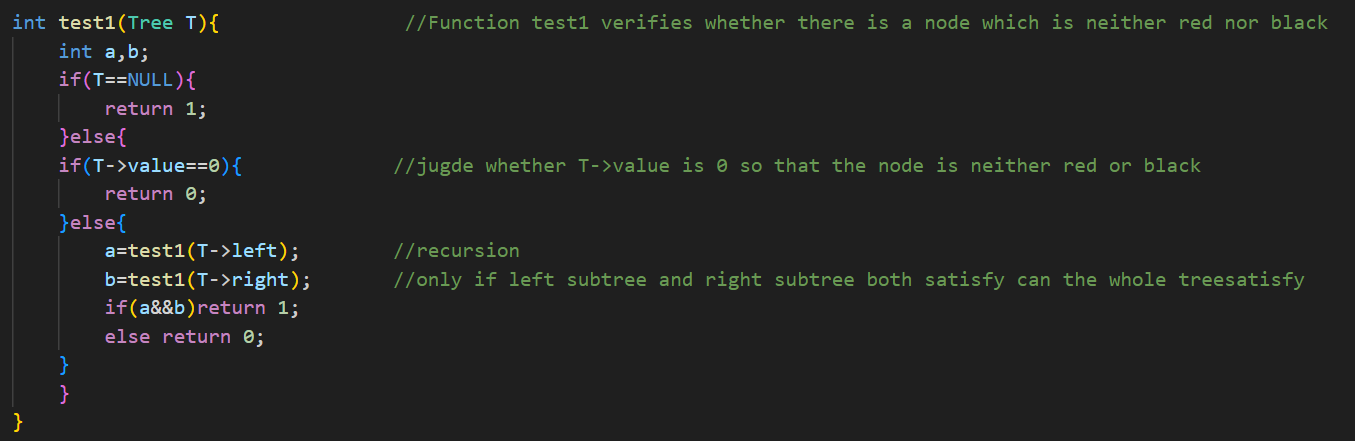
 Then,I design some functions to fulfill my object,including one to build the tree,one to count black nodes and four test functions to separately examine whether the tree satisfies the four properties above.(note that feature(3):every leaf(NULL) is black can be naturally satisfied by setting the node to NULL,so we needn’t take it into consideration.)

**Function 1: Buildtree**

This function aims to build the binary-search tree from the given prefix array.Note that the root of the tree is the first element of array,and the cut point which differs the left subtree and the right tree is the first element that is bigger than the root.So we get two new arrays,one is the prefix form for the left subtree and one for the right one.Then we can easily build the whole tree by recursion.

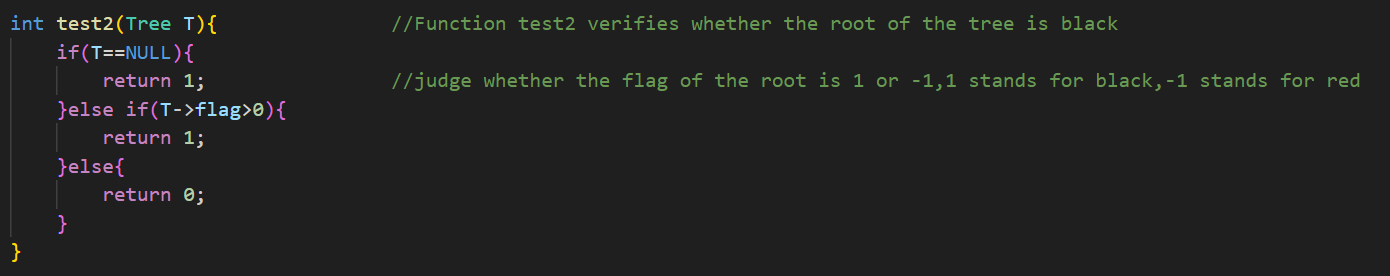
**Function 2: test1**

This function aims to test feature(1),verifing whether there is a node which is neither red or black.In fact,the problem is to find whether there is a node whose value is 0.The algorithm fist examines the root,then separately examing the left subtree and the right subtree by recursion.



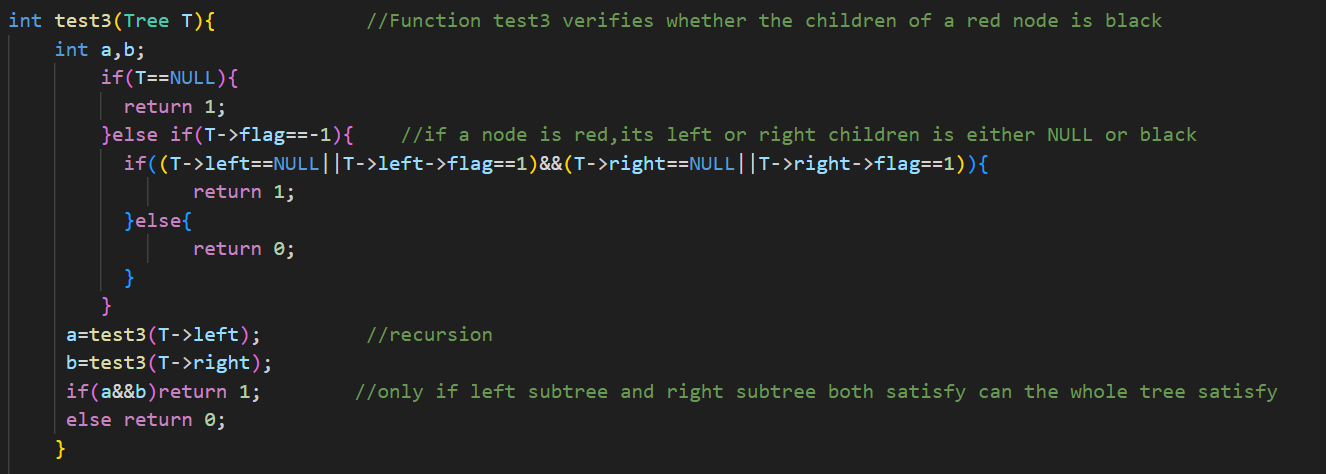
**Function 3: test2**

This function aims to test feature(2),verifying whether the root of the tree is black.The algorithm simply judges whether the flag of the root node is 1(which stands for black color.)



**Function 4: test 3**

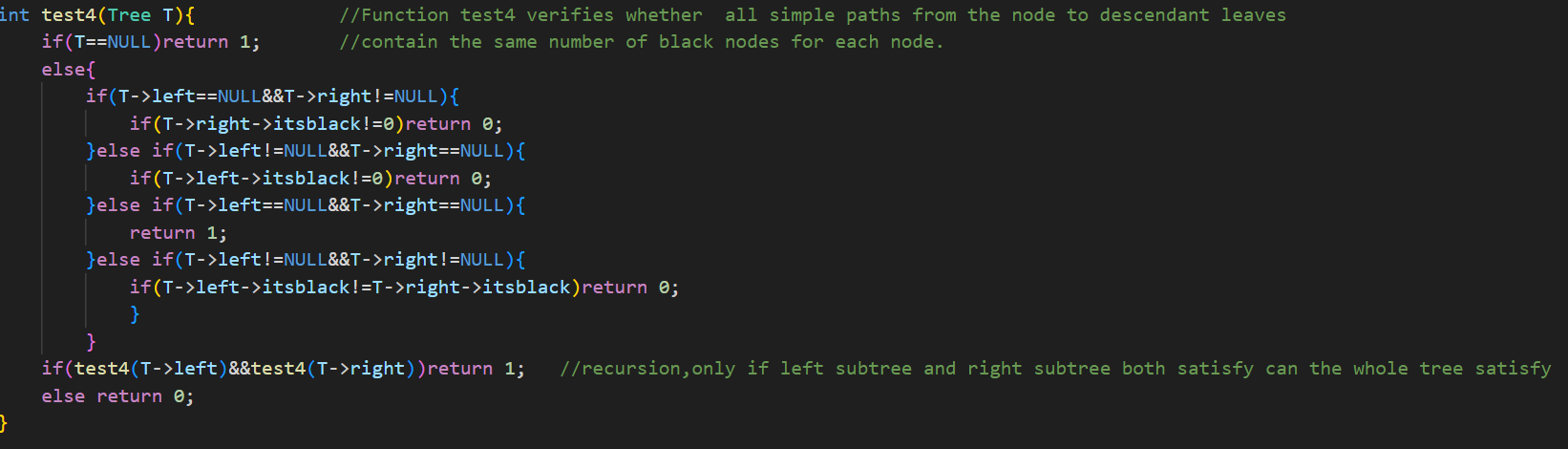
This function aims to verify whether the children of a red node is black.If a node is red,its left or right children is either NULL or black.The algorithm firstly examines the root,then examing the left subtree and right subtree by recursion.



**Function 5: test 4**

This function aims to verify whether all simple paths from the node to descendant leaves contain the same number of black nodes for each node.

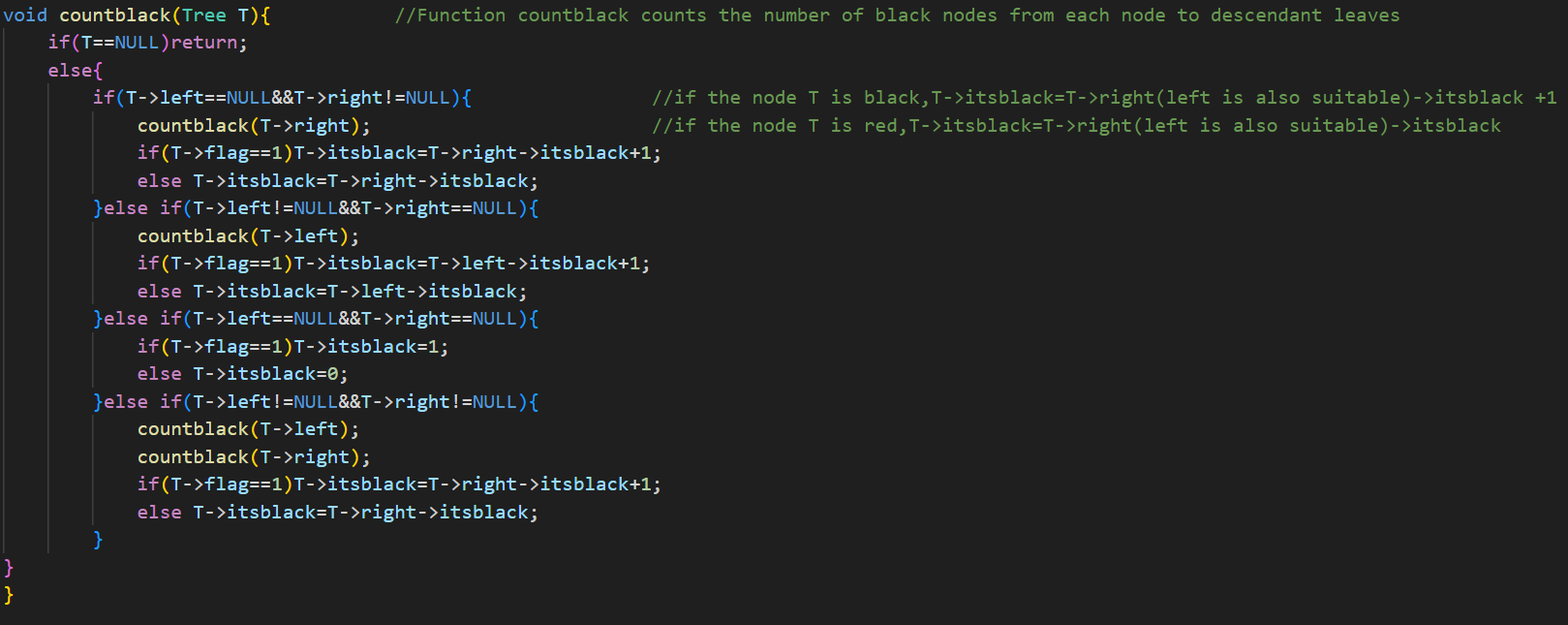
The algorithm divides four conditions of the existence of the left children and right children.Then after examing the root,we examine the left subtree and the right subtree by recursion.



**Function 6: countblack**

This function aims to count the number of black nodes to descendant leaves,and the number is stored in T->itsblack for each node T.When T is a black node,T->itsblack=T->left->itsblack+1(left or right actually don’t affect the test result because if left->itsblack==right->itsblack, there’s no diffenece.if left->itsblack!=right->itsblack,we can find this by recursion,and the final result is also correct(not satisfying,return a “0”)).

When T is a red node,T->itsblack=T->left->itsblack.Then we can easily count the number for every node by recursion.



**III.Testing Results**

|  |  |
| --- | --- |
| Testing data and results | Test for what |
|  | NULL tree |
|  | not passing test 1:the node is neither red or black |
|  | not passing test 2:the root is not black. |
|  | not passing test 3:there are two consecutive red nodes on a path. |
|  | not passing test 4: simple paths from two nodes to descendant leaves contain different numbers of black nodes |
|  | Satisfying all condions |
|  | Several arrays input at the same time |

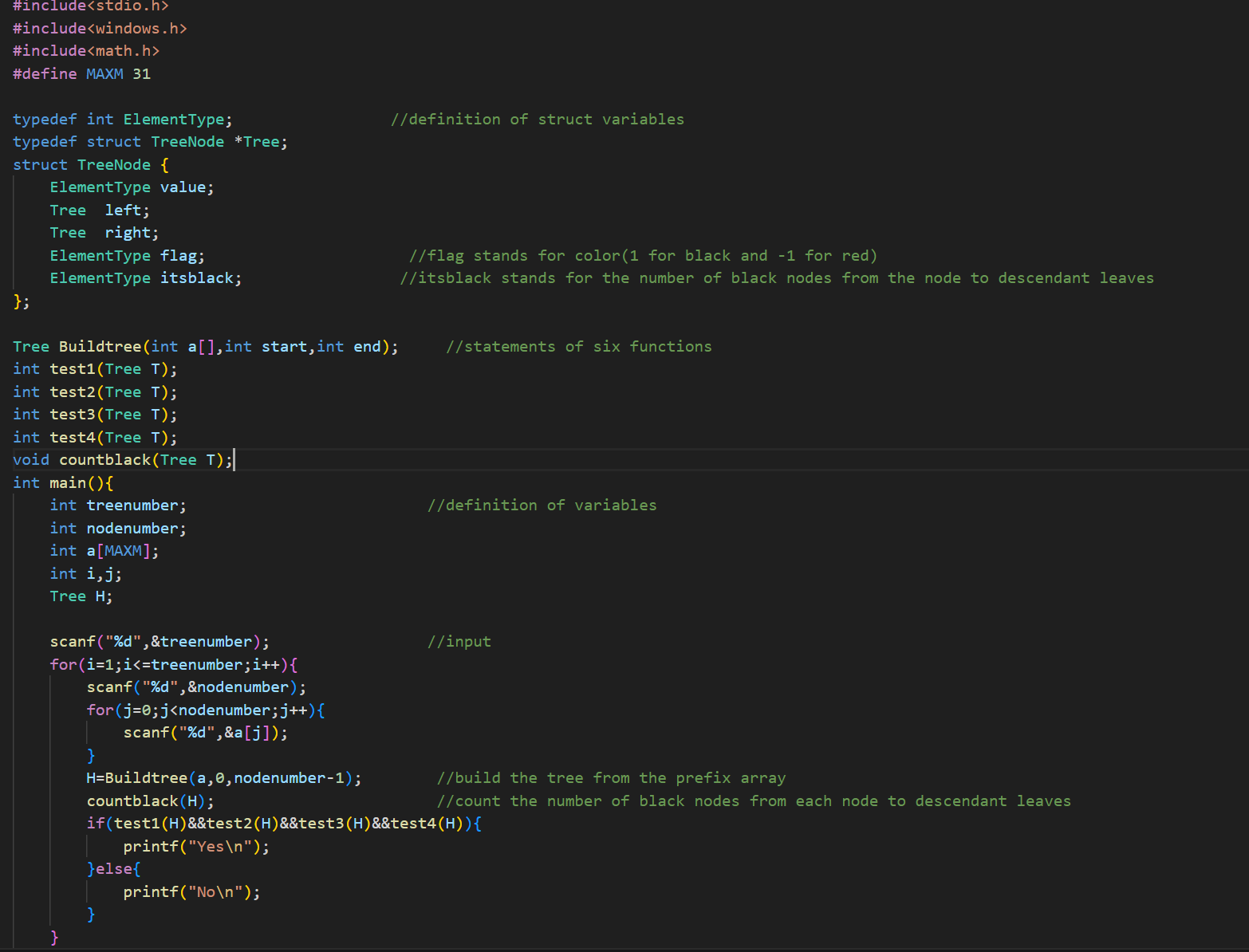
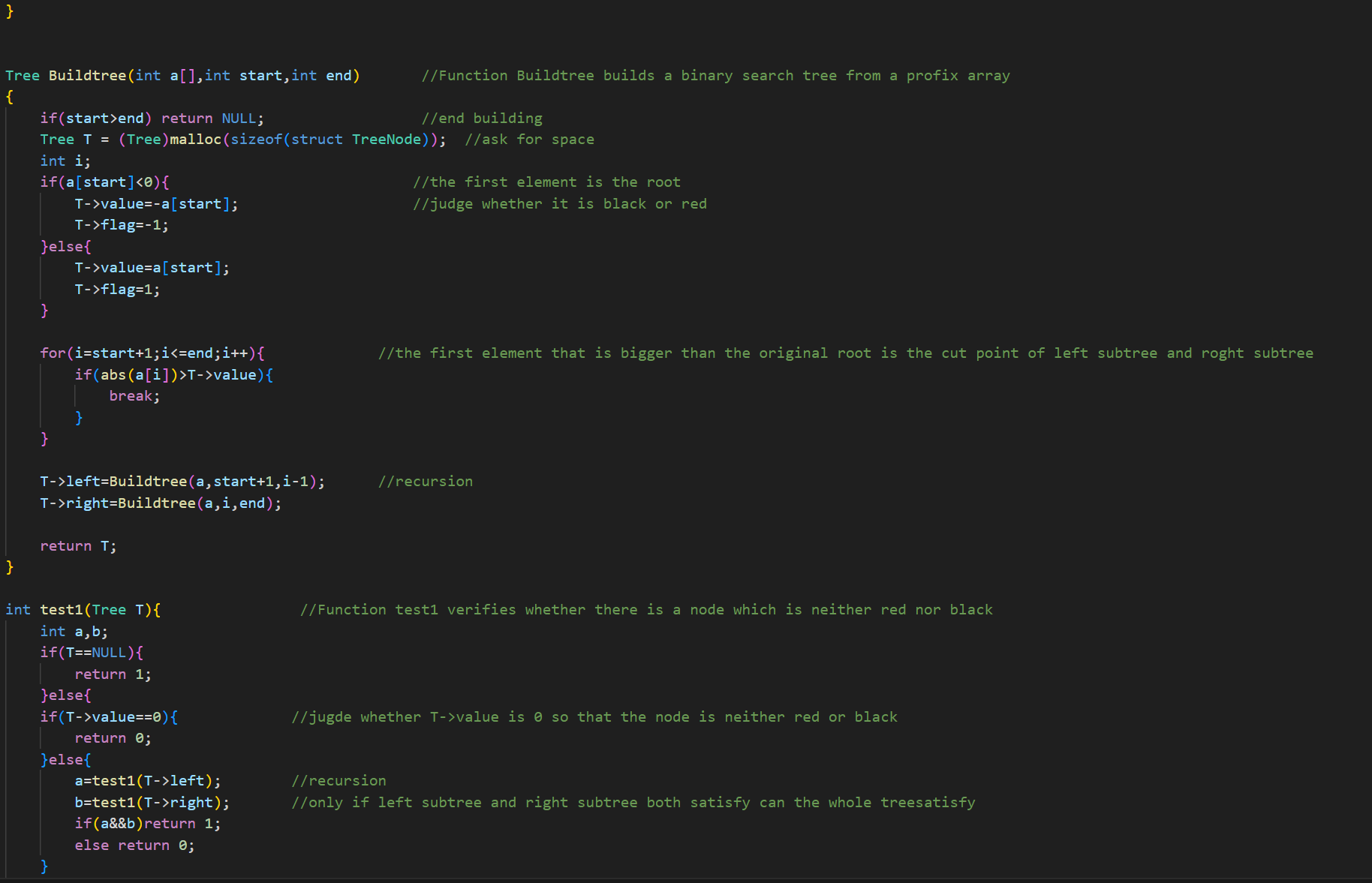
**IV.Analysis and Comments**

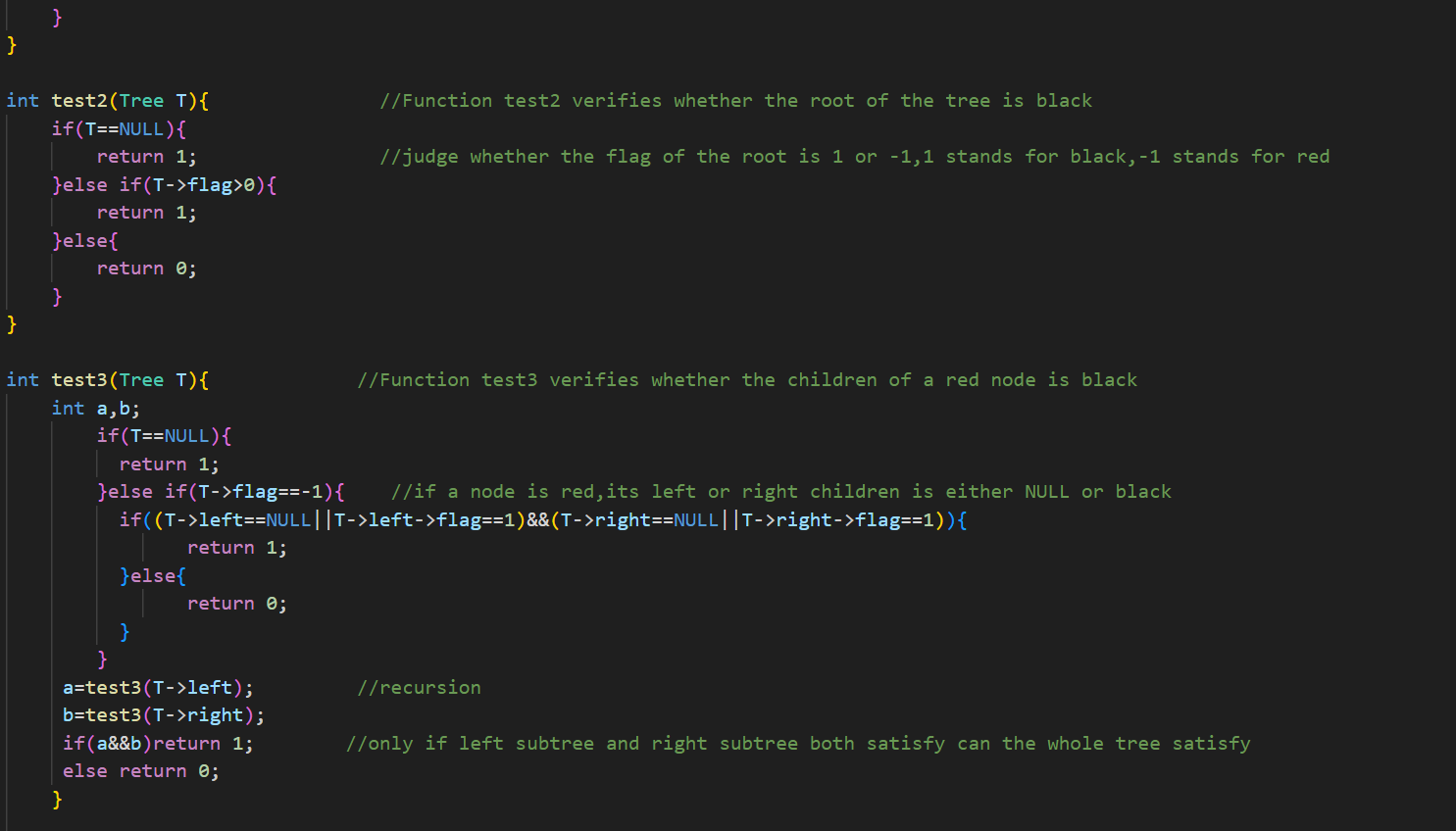
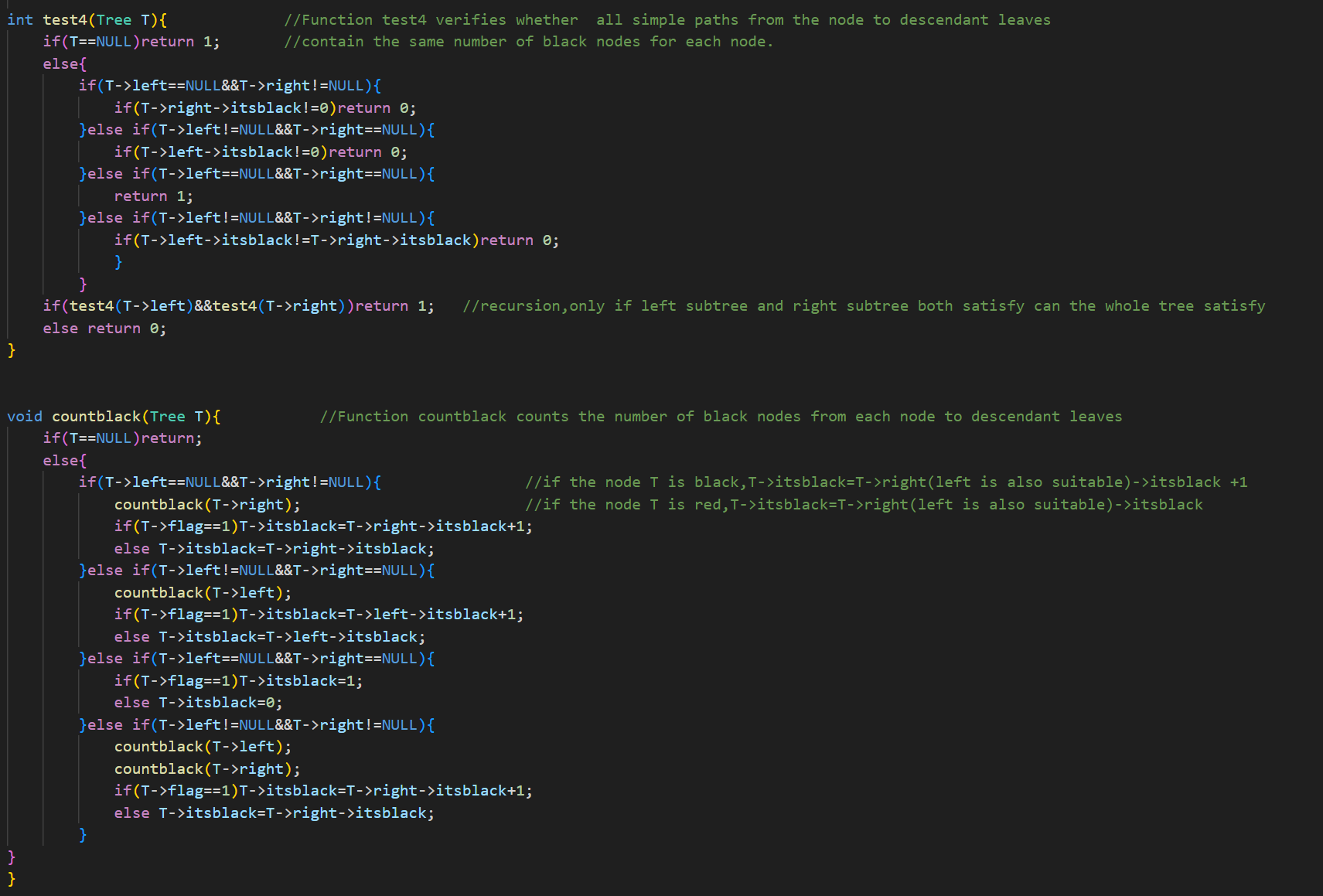
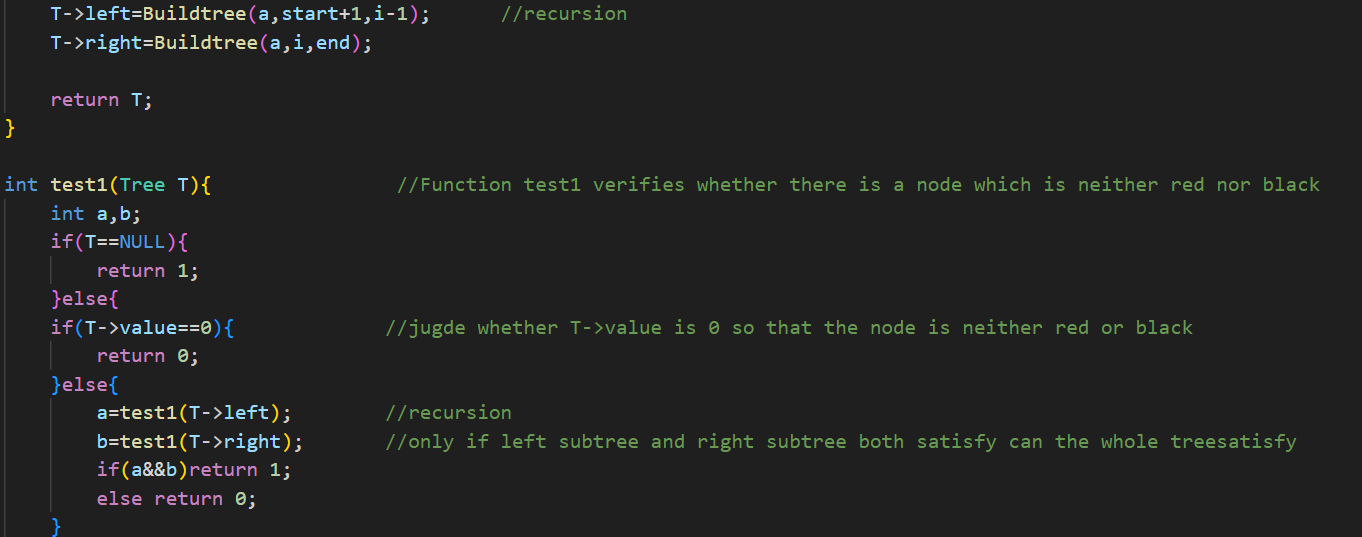
Let n be the the number of nodes of a tree.

Let d be the depth of the tree.

|  |  |  |
| --- | --- | --- |
| **function** | **Time complexity** | **Space complexity** |
| Buildtree | O(nLog(n))  We divide the tree into left subtree and right subtree every time | O(n)  the stack stores the results of all the nodes at most by recursion |
| Test 1 | O(n)  We actually ergodic all the nodes. | O(n)  the stack stores the results of all the nodes at most by recursion |
| Test 2 | O(1)  We only need to examine the root. | O(1)  The stack only stores one result. |
| Test 3 | O(n)  We need to examine all nodes. | O(n)  the stack stores the results of all the nodes at most by recursion |
| Test4 | O(n)  We need to compare the left children and the right children of each node.The operation is executed for n times. | O(n)  the stack stores the results of all the nodes at most by recursion |
| countblack | O(N)  The operation is executed for n times. | O(d)  The stack stores the results of all the nodes of the longest path at most. |

From the analysis of time and space complexity, I can feel that there’s still much space for my project and algorithm to improve.And I may use iterative methods instead of recursive methods in the future to compare the two algorithms and choose the better one.Overall, I successfully finished my project,despite many difficulties.Hope that I can do evenbetter next time!

**V.Appendix**

****

**VI. Declaration**

I hereby declare that all the work done in this project titled “陈硕” is of my independent effort.