Is It A Red-Black Tree Problem

2023-10-29

I.Introduction

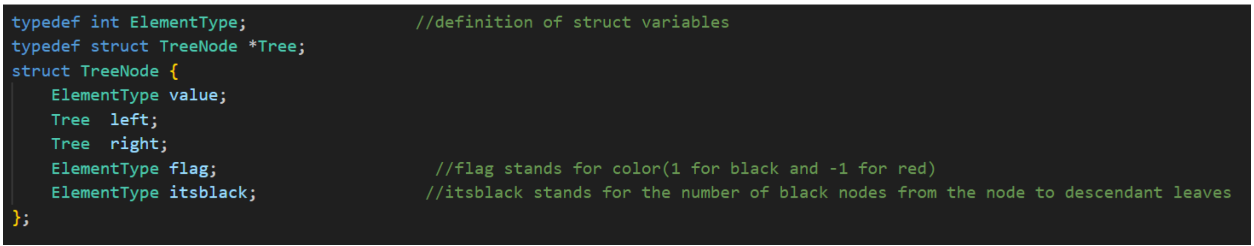
Is It A Red-Black Problem is a problem focusing on **judging whether a binary-search tree is a red-black tree** which is an advanced data structure, owing some features:

(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 contai the same number of black nodes.

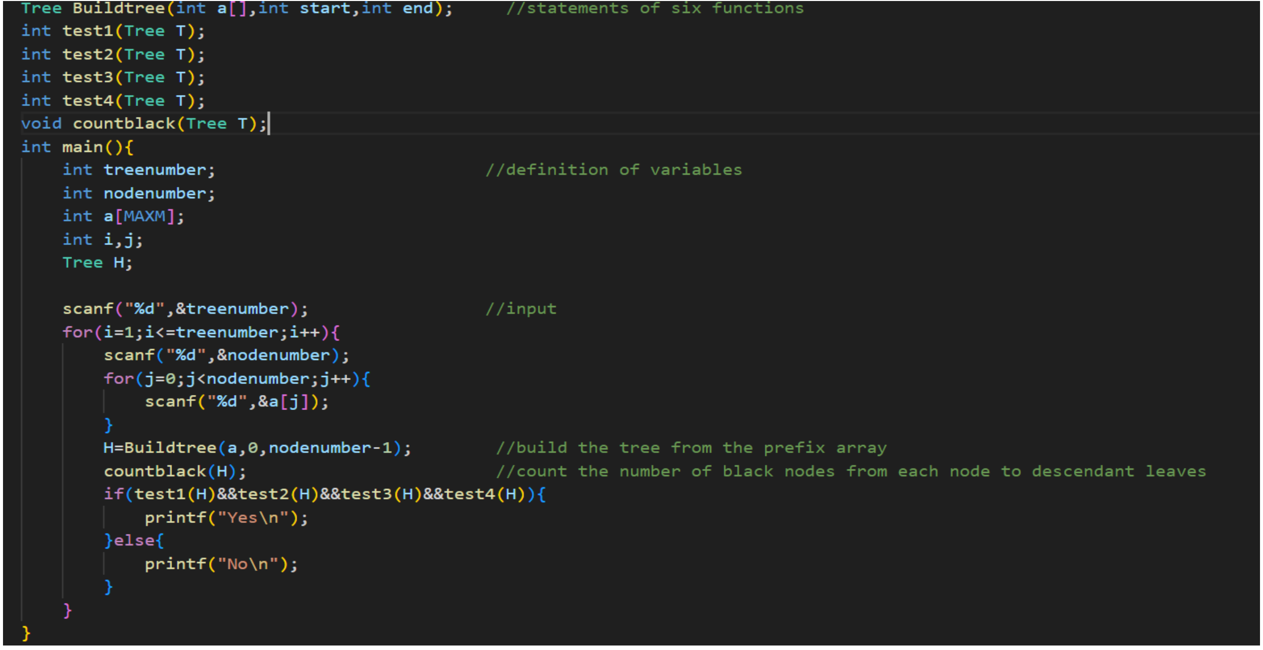
This project aims to find an efficient method to successfully tell whether a given prefix array can build a red-black tree.

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.



```c
typedef int ElementType;                //definition of struct variables
typedef struct TreeNode *Tree;
struct TreeNode {
    ElementType value;
    Tree  left;
    Tree  right;
    ElementType flag;                     //flag stands for color(1 for black and -1 for red)
    ElementType itsblack;                //itsblack stands for the number of black nodes from the node to descendant leaves
};
```

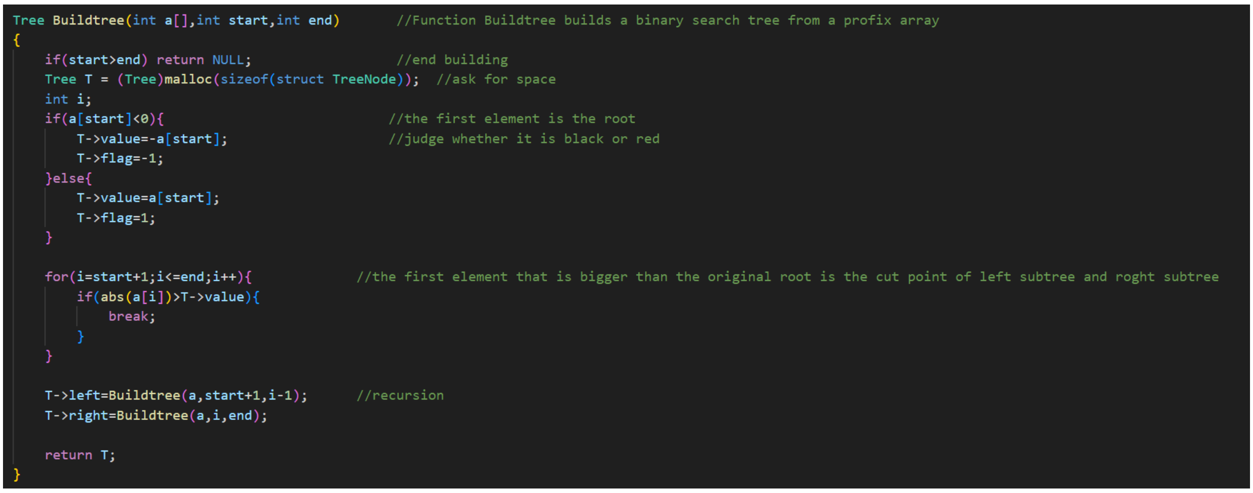


```c
Tree Buildtree(int a[],int start,int end);     //statements of six functions
int test1(Tree T);
int test2(Tree T);
int test3(Tree T);
int test4(Tree T);
void countblack(Tree T);
int main(){
    int treenumber;                          //definition of variables
    int nodenumber;
    int a[MAXM];
    int i,j;
    Tree H;

    scanf("%d",&treenumber);                 //input
    for(i=1;i<=treenumber;i++){
        scanf("%d",&nodenumber);
        for(j=0;j<nodenumber;j++){
            scanf("%d",&a[j]);
        }
        H=Buildtree(a,0,nodenumber-1);        //build the tree from the prefix array
        countblack(H);                        //count the number of black nodes from each node to descendant leaves
        if(test1(H)&&test2(H)&&test3(H)&&test4(H)){
            printf("Yes\n");
        }else{
            printf("No\n");
        }
    }
}
```

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

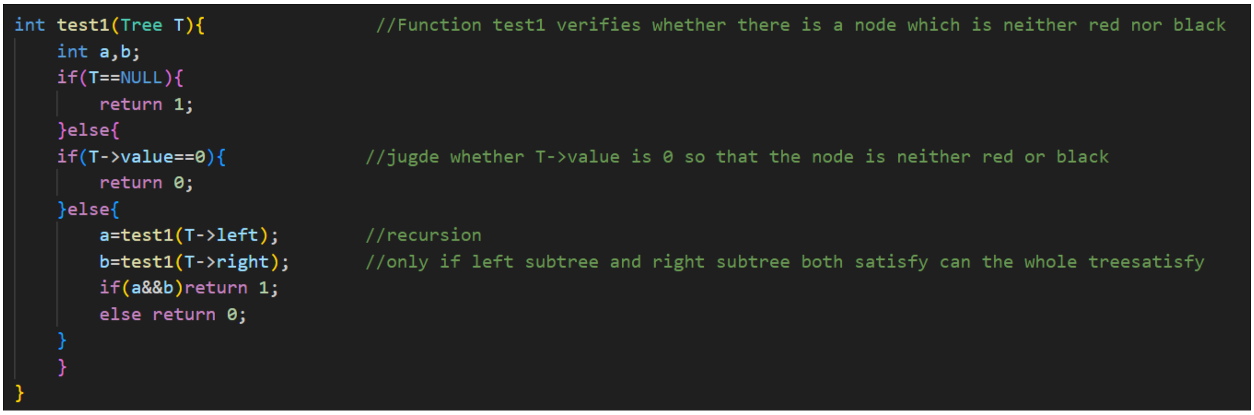
This function aims to build the binary-search tree from the given prefix array.Note that the root of the tree is the first element of array,and the cut point which differs the left subtree and the right tree is the first element that is bigger than the root.So we get two new arrays,one is the prefix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



```
Tree Buildtree(int a[],int start,int end)       //Function Buildtree builds a binary search tree from a profix array
{
    if(start>end) return NULL;                  //end building
    Tree T = (Tree)malloc(sizeof(struct TreeNode));  //ask for space
    int i;
    if(a[start]<0){                            //the first element is the root
        T->value=-a[start];                    //judge whether it is black or red
        T->flag=-1;
    }else{
        T->value=a[start];
        T->flag=1;
    }

    for(i=start+1;i<=end;i++){             //the first element that is bigger than the original root is the cut point of left subtree and roght subtree
        if(abs(a[i])>T->value){
            break;
        }
    }

    T->left=Buildtree(a,start+1,i-1);      //recursion
    T->right=Buildtree(a,i,end);

    return T;
}
```

Function 2: test1

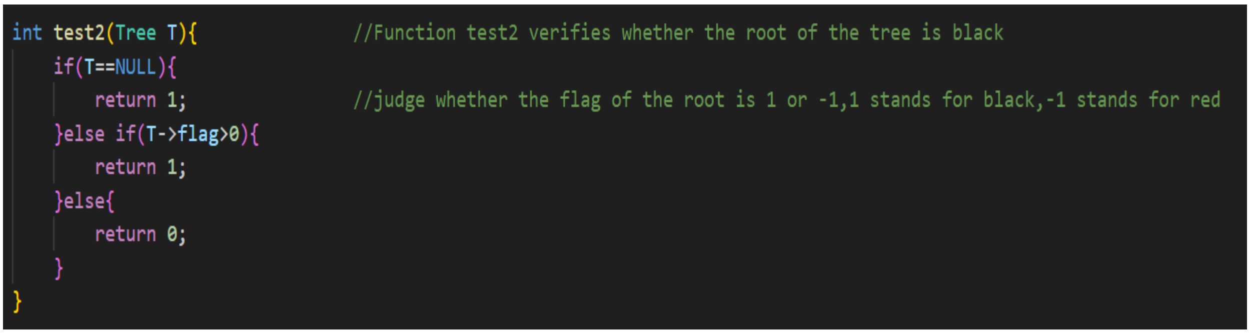
This function aims to test feature(1),verifing whether there is a node which is neither red or black.In fact,the problem is to find whether there is a node whose value is 0.The algorithm fist examines theroot,then separately examing the left subtree and the right subtree by recursion.



```
int test1(Tree T){                  //Function test1 verifies whether there is a node which is neither red nor black
    int a,b;
    if(T==NULL){
        return 1;
    }else{
    if(T->value==0){               //jugde whether T->value is 0 so that the node is neither red or black
        return 0;
    }else{
        a=test1(T->left);          //recursion
        b=test1(T->right);         //only if left subtree and right subtree both satisfy can the whole treesatisfy
        if(a&&b)return 1;
        else return 0;
    }
    }
}
```

Function 3: test2

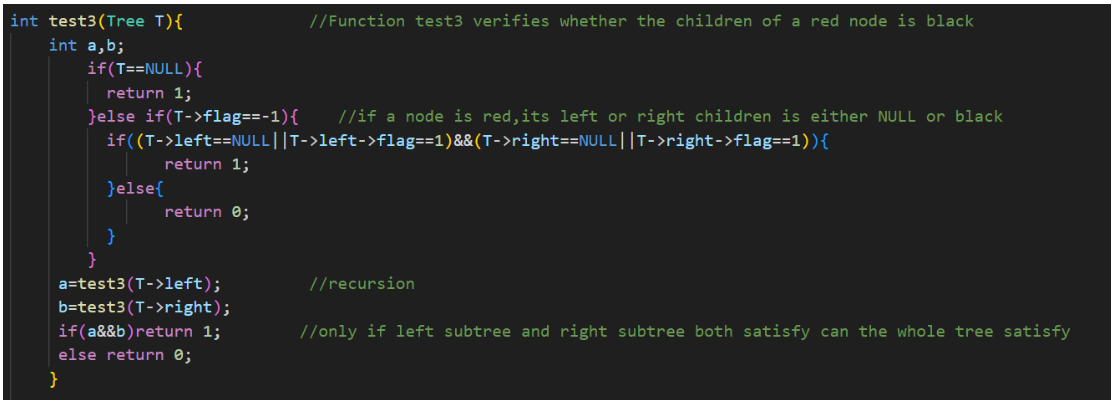
This function aims to test feature(2),verifying whether the root of the tree is black.The algorithm simply judges whether the flag of the root node is 1(which stands for black color.)



```c
int test2(Tree T){            //Function test2 verifies whether the root of the tree is black
    if(T==NULL){
        return 1;             //judge whether the flag of the root is 1 or -1,1 stands for black,-1 stands for red
    }else if(T->flag>0){
        return 1;
    }else{
        return 0;
    }
}
```

Function 4: test 3

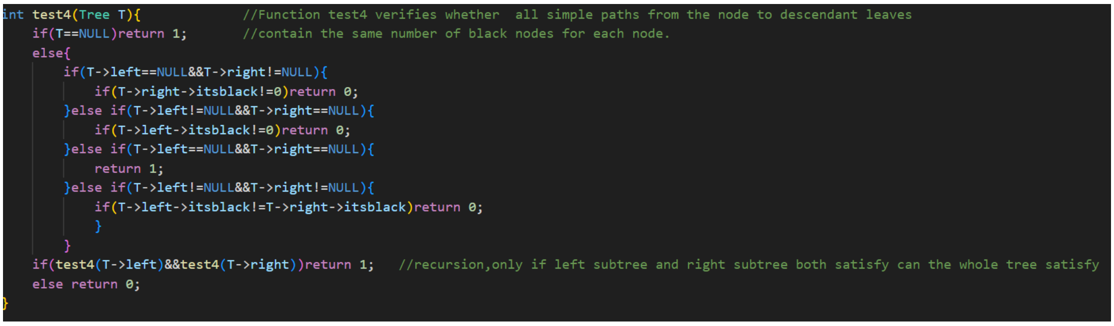
This function aims to verify whether the children of a red node is black.If a node is red,its left or right children is either NULL or black.The algorithm firstly examines the root,then examing the left subtree and right subtree by recursion.



```c
int test3(Tree T){              //Function test3 verifies whether the children of a red node is black
    int a,b;
        if(T==NULL){
          return 1;
        }else if(T->flag==-1){    //if a node is red,its left or right children is either NULL or black
          if((T->left==NULL||T->left->flag==1)&&(T->right==NULL||T->right->flag==1)){
                return 1;
          }else{
                return 0;
          }
        }
     a=test3(T->left);          //recursion
     b=test3(T->right);
     if(a&&b)return 1;         //only if left subtree and right subtree both satisfy can the whole tree satisfy
     else return 0;
    }
```

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

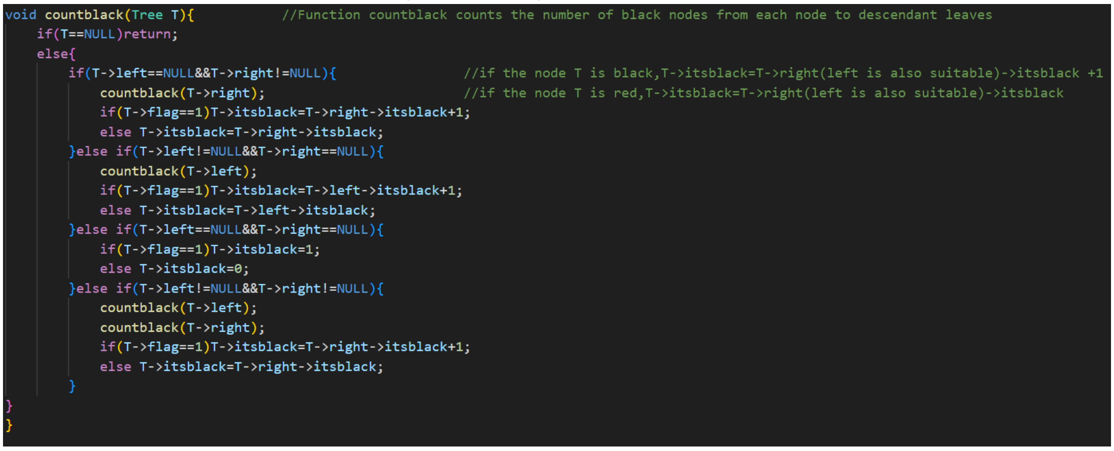


```c
int test4(Tree T){             //Function test4 verifies whether  all simple paths from the node to descendant leaves
    if(T==NULL)return 1;       //contain the same number of black nodes for each node.
    else{
        if(T->left==NULL&&T->right!=NULL){
            if(T->right->itsblack!=0)return 0;
        }else if(T->left!=NULL&&T->right==NULL){
            if(T->left->itsblack!=0)return 0;
        }else if(T->left==NULL&&T->right==NULL){
            return 1;
        }else if(T->left!=NULL&&T->right!=NULL){
            if(T->left->itsblack!=T->right->itsblack)return 0;
            }
        }
    if(test4(T->left)&&test4(T->right))return 1;   //recursion,only if left subtree and right subtree both satisfy can the whole tree satisfy
    else return 0;
}
```

Function 6: countblack

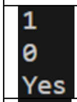
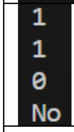
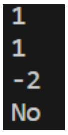
This function aims to count the number of black nodes to descendant leaves,and the number is stored in T->itsblack for each node T.When T is

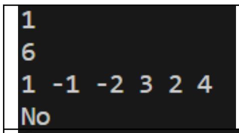
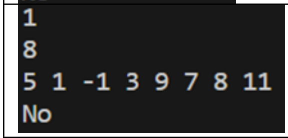
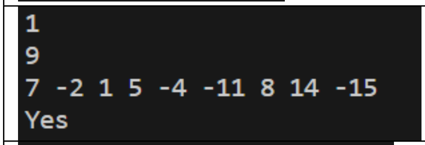
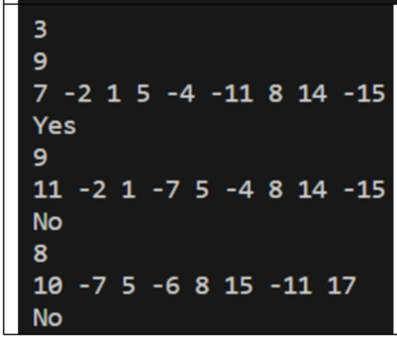
a black node,T->itsblack=T->left->itsblack+1(left or right actually don’t affect the test result because if left->itsblack==right->itsblack, there’s no diffenece.ifleft->itsblack!=right->itsblack,wecanfindthisbyrecursion,and the final result is also correct(not satisfying,return a “0”)).
When T is a red node,T->itsblack=T->left->itsblack.Then we can easily count the number for every node by recursion.



```
void countblack(Tree T){           //Function countblack counts the number of black nodes from each node to descendant leaves
    if(T==NULL)return;
    else{
        if(T->left==NULL&&T->right!=NULL){                //if the node T is black,T->itsblack=T->right(left is also suitable)->itsblack +1
            countblack(T->right);                         //if the node T is red,T->itsblack=T->right(left is also suitable)->itsblack
            if(T->flag==1)T->itsblack=T->right->itsblack+1;
            else T->itsblack=T->right->itsblack;
        }else if(T->left!=NULL&&T->right==NULL){
            countblack(T->left);
            if(T->flag==1)T->itsblack=T->left->itsblack+1;
            else T->itsblack=T->left->itsblack;
        }else if(T->left==NULL&&T->right==NULL){
            if(T->flag==1)T->itsblack=1;
            else T->itsblack=0;
        }else if(T->left!=NULL&&T->right!=NULL){
            countblack(T->left);
            countblack(T->right);
            if(T->flag==1)T->itsblack=T->right->itsblack+1;
            else T->itsblack=T->right->itsblack;
        }
}
}
```

III.Testing Results

Testing data and results	Test for what
 1 0 Yes	NULL tree
 1 1 0 No	not passing test 1:the node is neither red or black
 1 1 -2 No	not passing test 2:the root is not black.

1 6 1 -1 -2 3 2 4 No	not passing test 3:there are two consecutive red nodes on a path.
1 8 5 1 -1 3 9 7 8 11 No	not passing test 4: simple paths two nodes to descendant es contain different numbers of black nodes
1 9 7 -2 1 5 -4 -11 8 14 -15 Yes	Satisfying all condions
3 9 7 -2 1 5 -4 -11 8 14 -15 Yes 9 11 -2 1 -7 5 -4 8 14 -15 No 8 10 -7 5 -6 8 15 -11 17 No	Several arrays input at the same time

IV.Analysis and Comments

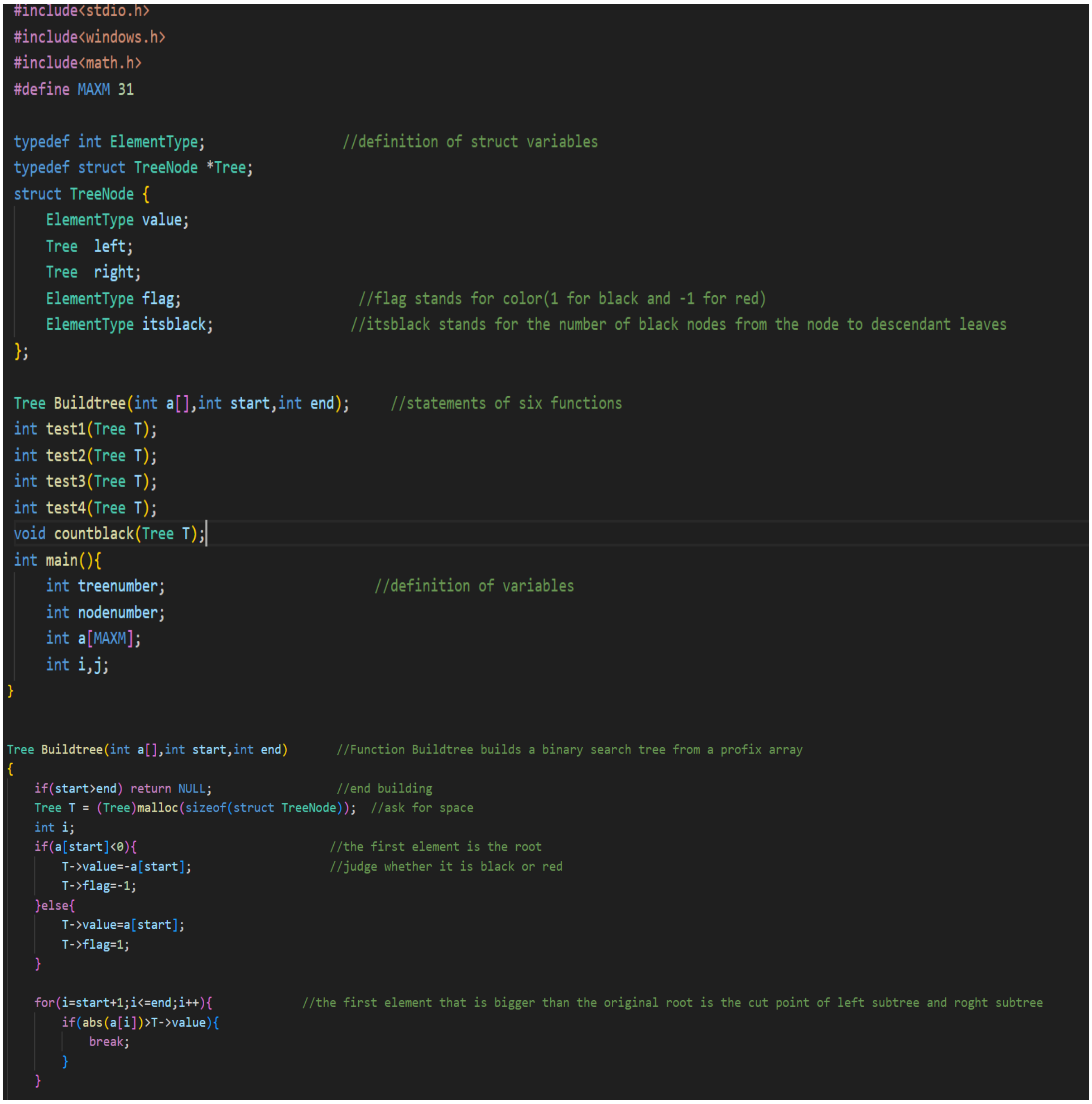
Let n be the the number of nodes of a tree.
Let d be the depth of the tree.

function	**Time complexity**	**Space complexity**
Buildtree	O(Log(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	O(n) the stack stores the

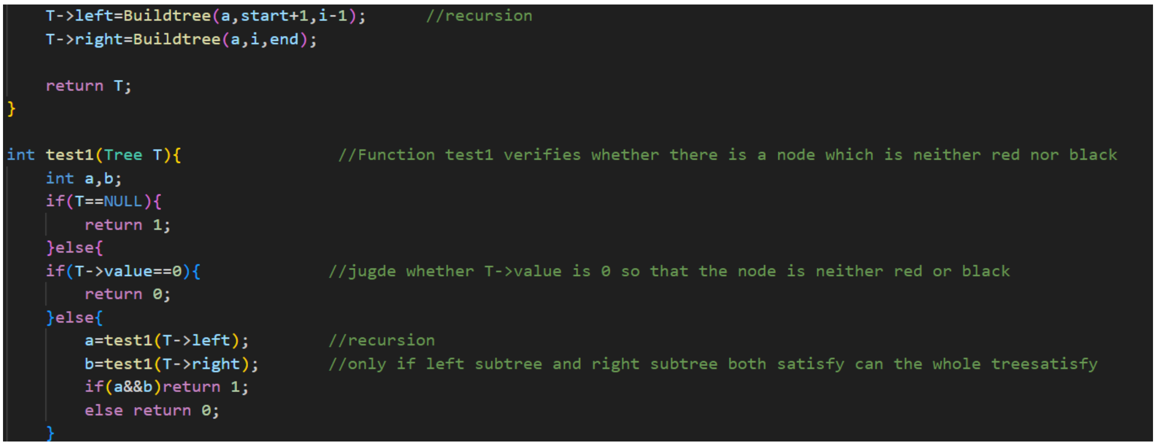
	the left children and the right children of each node. The operation is executed for n times.	results of all the nodes at most by recursion
countblack	O(N) The operation is executed for n times.	O(d) The stack stores the results of all the nodes of the longest path at most.

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 compar the two algorithms and choose the better one. Overall, I successfully finished my project, despite many difficulties. Hope that I can do even better next time!

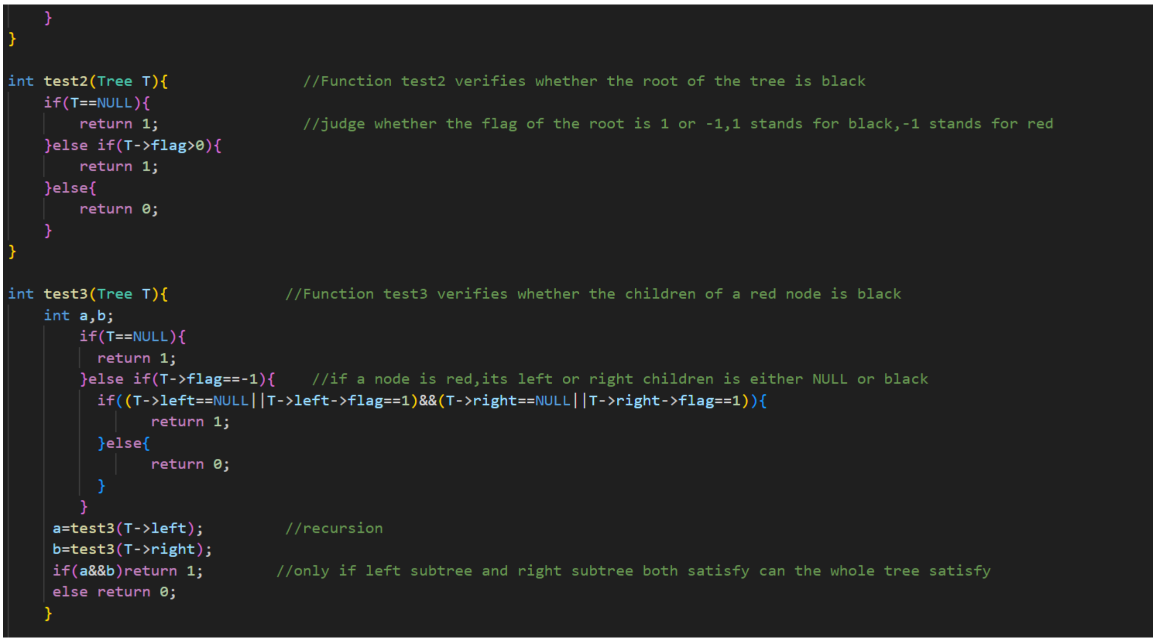
V.Appendix



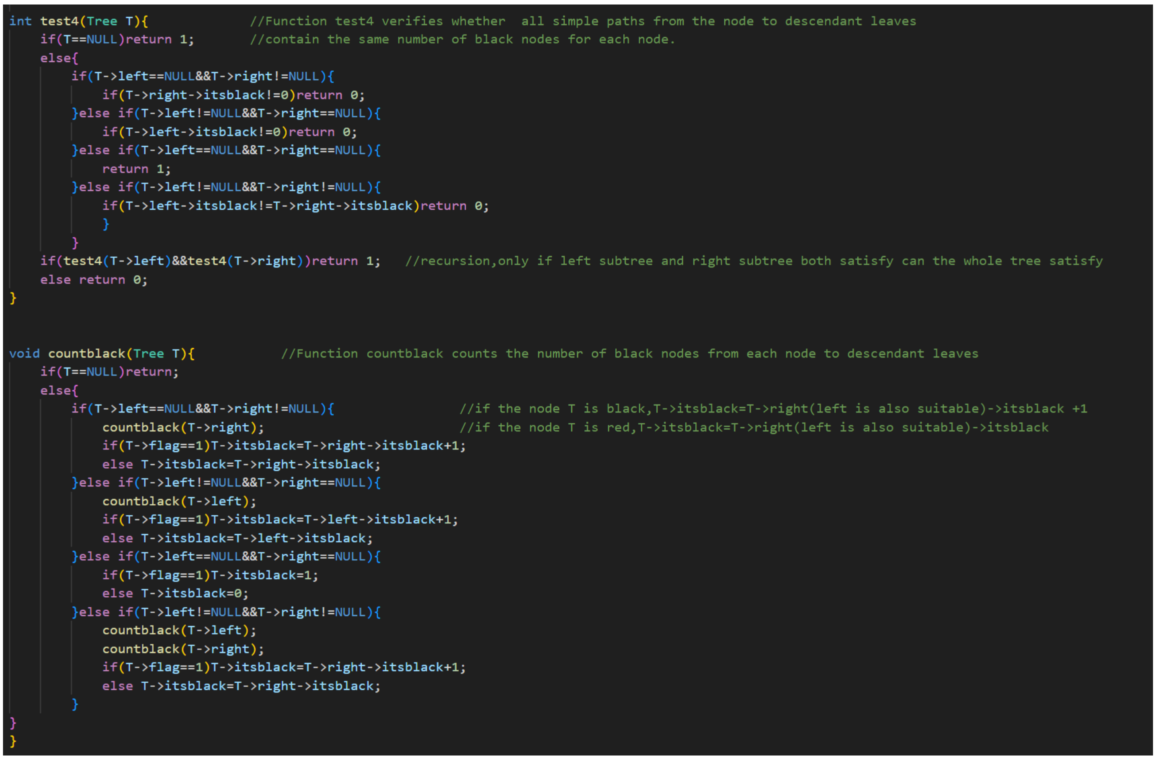
```
#include<stdio.h>
#include<windows.h>
#include<math.h>
#define MAXM 31

typedef int ElementType;                //definition of struct variables
typedef struct TreeNode *Tree;
struct TreeNode {
    ElementType value;
    Tree  left;
    Tree  right;
    ElementType flag;                       //flag stands for color(1 for black and -1 for red)
    ElementType itsblack;                  //itsblack stands for the number of black nodes from the node to descendant leaves
};

Tree Buildtree(int a[],int start,int end);     //statements of six functions
int test1(Tree T);
int test2(Tree T);
int test3(Tree T);
int test4(Tree T);
void countblack(Tree T);
int main(){
    int treenumber;                         //definition of variables
    int nodenumber;
    int a[MAXM];
    int i,j;
}


Tree Buildtree(int a[],int start,int end)       //Function Buildtree builds a binary search tree from a profix array
{
    if(start>end) return NULL;                  //end building
    Tree T = (Tree)malloc(sizeof(struct TreeNode));  //ask for space
    int i;
    if(a[start]<0){                            //the first element is the root
        T->value=-a[start];                    //judge whether it is black or red
        T->flag=-1;
    }else{
        T->value=a[start];
        T->flag=1;
    }

    for(i=start+1;i<=end;i++){             //the first element that is bigger than the original root is the cut point of left subtree and roght subtree
        if(abs(a[i])>T->value){
            break;
        }
    }
```



```c
    T->left=Buildtree(a,start+1,i-1);      //recursion
    T->right=Buildtree(a,i,end);

    return T;
}

int test1(Tree T){                  //Function test1 verifies whether there is a node which is neither red nor black
    int a,b;
    if(T==NULL){
        return 1;
    }else{
    if(T->value==0){                //jugde whether T->value is 0 so that the node is neither red or black
        return 0;
    }else{
        a=test1(T->left);           //recursion
        b=test1(T->right);          //only if left subtree and right subtree both satisfy can the whole treesatisfy
        if(a&&b)return 1;
        else return 0;
    }
```



```c
    }
}

int test2(Tree T){                  //Function test2 verifies whether the root of the tree is black
    if(T==NULL){
        return 1;                   //judge whether the flag of the root is 1 or -1,1 stands for black,-1 stands for red
    }else if(T->flag>0){
        return 1;
    }else{
        return 0;
    }
}

int test3(Tree T){                  //Function test3 verifies whether the children of a red node is black
    int a,b;
        if(T==NULL){
          return 1;
        }else if(T->flag==-1){     //if a node is red,its left or right children is either NULL or black
          if((T->left==NULL||T->left->flag==1)&&(T->right==NULL||T->right->flag==1)){
                return 1;
          }else{
                return 0;
          }
        }
    a=test3(T->left);               //recursion
    b=test3(T->right);
    if(a&&b)return 1;               //only if left subtree and right subtree both satisfy can the whole tree satisfy
    else return 0;
    }
```



```c
int test4(Tree T){              //Function test4 verifies whether  all simple paths from the node to descendant leaves
    if(T==NULL)return 1;        //contain the same number of black nodes for each node.
    else{
        if(T->left==NULL&&T->right!=NULL){
            if(T->right->itsblack!=0)return 0;
        }else if(T->left!=NULL&&T->right==NULL){
            if(T->left->itsblack!=0)return 0;
        }else if(T->left==NULL&&T->right==NULL){
            return 1;
        }else if(T->left!=NULL&&T->right!=NULL){
            if(T->left->itsblack!=T->right->itsblack)return 0;
            }
        }
    if(test4(T->left)&&test4(T->right))return 1;   //recursion,only if left subtree and right subtree both satisfy can the whole tree satisfy
    else return 0;
}


void countblack(Tree T){            //Function countblack counts the number of black nodes from each node to descendant leaves
    if(T==NULL)return;
    else{
        if(T->left==NULL&&T->right!=NULL){                //if the node T is black,T->itsblack=T->right(left is also suitable)->itsblack +1
            countblack(T->right);                          //if the node T is red,T->itsblack=T->right(left is also suitable)->itsblack
            if(T->flag==1)T->itsblack=T->right->itsblack+1;
            else T->itsblack=T->right->itsblack;
        }else if(T->left!=NULL&&T->right==NULL){
            countblack(T->left);
            if(T->flag==1)T->itsblack=T->left->itsblack+1;
            else T->itsblack=T->left->itsblack;
        }else if(T->left==NULL&&T->right==NULL){
            if(T->flag==1)T->itsblack=1;
            else T->itsblack=0;
        }else if(T->left!=NULL&&T->right!=NULL){
            countblack(T->left);
            countblack(T->right);
            if(T->flag==1)T->itsblack=T->right->itsblack+1;
            else T->itsblack=T->right->itsblack;
        }
}
}
```

VI. Declaration

I hereby declare that all the work done in this project is of my independen
effort.