Is It A Red-Black Tree

汪珉凯

Date: 2023-11-14

Chapter 1: Introduction

Problem description:The problem is to judge whether a tree is a red black tree.A red black tree should have the following characters:

1. The root node is black.
2. Every node is either red or black.
3. Every child of a red node is black.
4. Every leaf(NULL) is black.
5. All simple paths from a node to descendant leaves contain the same number of black nodes.

and (if any) background of the algorithms.

Chapter 2: Algorithm Specification

Description of all algorithms:

1. Build Tree :

First build a NULL node as the root of the tree.Then begin to read in all the nodes one by one and add the to the tree at present.

Pseudo Code:

```
Build tree(number:integer):
    tree:=NULL
    For i:=1 to number do
        newdata:=integer
        if (tree==NULL)then
            newnode:*node
            newnode->data=newdata
            newnode->left=NULL
            newnode->right=NULL
        else
            if( abs(tree->data) > abs(newdata) )then
                tree->left=plus(tree->left,newdata)
            else
                tree->right=plus(tree->right,newdata)
```

2. Check1 :(judge if there are 2 continuous red nodes)

Return 1 means checked successfully;Return 0 means it is still possible to be a red-black tree;Return 0 means it's not a red-black tree.

Pseudo Code:

```
Check1 (root:node *):
    if(root==NULL)then
        return 1
    if(root->data<0)then
        if( (root->left && root->left->data<0) || (root->right && root->right->data<0) ) then
            return 0
    }
    return (check1(root->left) && check1(root->right))
```

3. Check2 :(judge if the number of black nodes are the same)

If the number of black nodes in every path ***from root to leaves*** are the same,then the numbers of black nodes in every path ***from a specific node to leaves*** are all the same.That means we only need to check all paths from root to leaves.

Pseudo Code:

```
Check2 (root:node* ,number:integer):
    if(root==NULL)then
        return 1
    if(root->data>0)then
        number++
    if((!root->left)&&(!root->right))then
        if(number==checker)then
            return 1
        else
            return 0
    else
        return ( check2(root->left,number) && check2(root->right,number) )
```

Main data structures:

1. Binary Search Tree
2. Red Black Tree

Sketch of the main program:

```
Main():
    x:integer
    for i:=1 to x do:
        number:integer
        flag:integer
        root:node*

        root=buildtree(number)
        if(root->data<0)then
            print(NO)
            break

        flag=check(root)
        if(flag!=0)then
            print(YES)
        else
            print(NO)
```

Chapter 3: Testing Results

Input	Purpose	Expected output	Actual result
1 9 7 -2 1 5 -4 -11 8 14 -15	Check if a red black tree can be successfully recognized.	Yes	Yes
1 9 11 -2 1 -7 5 -4 8 14 -15	Check the situation in which there are 2 constant red nodes.	No	No
1 8 10 -7 5 -6 8 15 -11 17	Check the situation in which the number of black nodes is different from a specific nodes to leaves through different paths.	No	No
1 9 -7 -2 1 5 -4 -11 8 14 -15	Check the situation in which the root is red.	No	No
1 1 1	Check the situation in which there's only a root in the tree.	Yes	Yes

Chapter 4: Analysis and Comments

Algorithm1:Build Tree

Time Complexity: Every insertion takes T(N)=O(d),where d is the depth of the binary tree.logN <= d <= N.There N insertions.So under the best conditions, the time complexity is ***O(N*logN)***; Under the worst conditions, the binary tree degenerates into a *linear* structure with a time complexity of ***O(N²)***.

Space Complexity: ***O(N),where N is the number of nodes in the tree.***

Every node takes O(1) space complexity.So the construction of the whole tree takes O(N) space complexity.

Algorithm2:Check1(constant red nodes?)

Time Complexity:***O(N),where N is the number of nodes in the tree.***The function recursively calls itself on the left and right subtrees. This means that for each node in the tree, the function is called twice (once for the left subtree and once for the right subtree).Thus the result is O(N).

Space Complexity:***O(N),where N is the number of nodes in the tree.***Every time the algorithm visit a never-visited node in the tree,it will apply O(1) memory.So the total algorithm takes O(N) space complexity.

Algorithm3:Check2(number of black nodes?)

Time Complexity:***O(N),where N is the number of nodes in the tree.***Similarly to Algorithm2,the function recursively calls itself on the left and right subtrees. Thus the result is also O(N).

Space Complexity:***O(N),where N is the number of nodes in the tree.***Every time the algorithm visit a never-visited node in the tree,it will apply O(1) memory.So the total algorithm takes O(N) space complexity.

Appendix: Source Code (in C)

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {                //definition of node
    int data;
    struct node* left;
    struct node* right;
}node;

node* root=NULL;

int abs(int x);
node* buildtree(int x);
node* plus(node* root,int newdata);
int check(node* root);
int check1(node* root);
int check2(node* root,int number);
//void print(node* root);
int checker;//used to check the number of black node in function 'check2'
int number;

int main(){
    int x;                  //x is the number of trees to be judged
    scanf("%d",&x);
    for (int i=0;i<x;i++){  //judge x times
        scanf("%d",&number);//'number' is the number of nodes in this tree
        int flag;           //flag=1 -> is a red-black tree
        checker=0;

        node* root;
        root=buildtree(number);//the entire tree has been constructed


        if(root->data<0){
            printf("No\n");
            continue;
        }//if the root is red -> not a red-black tree

        flag=check(root);//if is,flag=1;else flag=0.

        if(flag){
            printf("Yes\n");
        }
        else {
            printf("No\n");
        }
    }
}


int abs(int x){          //The function to fetch the absolute value of nodes
    if (x<0) x=-x;
    return x;
}


node* buildtree(int x){//The function to build a tree with x nodes

    node* root=NULL;//construction of root

    for(int i=0;i<x;i++){
        int newdata;
        scanf("%d",&newdata);
        root=plus(root,newdata);
    }//build others

    return root;
}
```

```c

node* plus(node* tree,int newdata){//The function to add a node into the tree
    if(tree==NULL){
        node* newnode=(node*)malloc(sizeof(node));
        newnode->data=newdata;
        newnode->left=NULL;
        newnode->right=NULL;
        return newnode;
    }//the construction of a new node
    else{
        if( abs(tree->data) > abs(newdata) ){
            tree->left=plus(tree->left,newdata);
        }
        else{
            tree->right=plus(tree->right,newdata);
        }
    }//determine the direction of insertion
    return tree;
}
```

```c
int check(node* root){//Check if it's a red-black tree,whose root has already been checked before.
    int result1=0,result2=0;

    result1 = check1(root);
    if(!result1) return 0;//check if there are 2 continuous red nodes.


    checker=0;
    number=0;
    node* _root=root;
    while(_root){
        if(_root->data>0)checker++;
        _root=_root->left;
    }
    result2 = check2(root,number);
    if(!result2) return 0;//check if the number of black nodes are the same

    return 1;
}

int check1(node* root){//check if there are 2 continuous red nodes.
    if(!root)return 1;//leaves

    if(root->data<0){
        node* L =root->left;
        node* R =root->right;
        if((L&&L->data<0)||(R&&R->data<0)) return 0;
    }//if it's a red node,then check if its children are red

    return (check1(root->left) && check1(root->right));//check both its children
}
```

```c
int check2(node* root,int number){//check if the number of black nodes are the same
    //If the number of black nodes in every path from root to leaves are the same,
    //then the number of black nodes is always the same in every path from a specific node to leaves
    if(!root)return 1;
    if(root->data>0)number++;
    if((!root->left)&&(!root->right)){
        if(number==checker)return 1;
        return 0;
    }
    return ( check2(root->left,number) && check2(root->right,number) );
}

```

Declaration

I hereby declare that all the work done in this project titled "Is It A Red-Black Tree " is of my independent effort.