

湖南大学



题 目： 二叉树的实现

学生姓名 吴多智

学生学号 201626010520

专业班级 软件 1605

完成日期 2017-11-23

一、需求分析

1.问题描述

实现二叉树的基本操作。

- 1) 实现二叉树的创建。
- 2) 实现二叉树的前序遍历，中序遍历，后序遍历。
- 3) 实现在二叉树中查找一个元素。存在输出字符 The inputt exists, 不存在则输出 The input does not exist。
- 4) 计算二叉树的叶子结点统计。

2.输入数据

输入数据类型可以是整数型，浮点型，字符型。没有输入范围。以二叉树前序遍历的形式输入，空结点以#代替。

3.输出数据

输出数据类型与输入数据类型相同。以中序遍历，后序遍历输出二叉树，当查找元素时，存在输出字符 The input is exist, 不存在则输出 The input is not exist。

4.测试样例设计

- 1) 输入：#
没有输出
输入：A
输出：The input does not exist
测试目的：输入为空树，则没有输出。
- 2) 输入：A##
输出：
前序遍历：A
中序遍历：A
树的叶子结点数：1
输入：A
输出：The input exists
测试目的：当输入只有一个结点时。
- 3) 输入：A#B#C#D##
输出：
前序遍历：A B C D
中序遍历：A B C D
树的叶子结点数：1
输入：E
输出：The input does not exist。
测试目的：当输入仅存在右子树时，当查找的值不存在树中时。
- 4) 输入：ABCD#####
输出：
前序遍历：A B C D
中序遍历：D C B A

树的叶子结点数: 1

输入: D

输出: The input exists。

测试目的: 当输入仅为左子树时, 且查找为叶子结点时。

5) 输入: AB#D##CEG####FH##I##

输出:

前序遍历: A B D C E G F H I

中序遍历: B D A G E C H F I

后序遍历: D B G E H I F C A

树的叶子结点数: 4

输入: A

输出: The input exists。

测试目的: 此为正则,当查找为根结点时。

二、概要设计

1.抽象数据类型

1) 数据对象: D 具有相同特性的数据元素的集合。

2) 数据关系: 若 D 为空集, 则成空树。

否则:

在 D 中存在唯一的称为根的数据元素 root;

当 $n > 1$ 时, 其余结点可分为不相交的有限集 T_l, T_r , 其中每一个子集本身又是一个符合本定义的二叉树, 称为根 root 的子树。

3) 基本操作:

```
class binnode{
public:
    E data;
    binnode* left;
    binnode* right;
    binnode(){
        left = NULL;
        right = NULL;
    }
    binnode( E d){
        data = d;
        left = NULL;
        right = NULL;
    }

    void setleft( binnode* l){//给左赋值
        left = l;
    }
    binnode* Left(){//返回左
```

```

        return left;
    }

    void setright( binode* r ){//给右赋值
        right = r;
    }
    binode* Right(){//返回右
        return right;
    }

    bool isLeaf(){//return ture if the node is a leaf
        if( left == NULL && right == NULL)return true;
        else return false;
    }
};

```

2.算法的基本思想

1) 基于递归创建二叉树。

构建二叉树=储存根结点的信息+构建左子树+构建右子树，二叉树不断分解为根结点和左右子树；当输入为空信息时，返回空二叉树，子问题结束。

2) 递归实现二叉树。

前序遍历二叉树：若二叉树非空，则实现下列操作：访问根结点，前序周游左子树；前序周游右子树。

中序遍历二叉树：若二叉树非空，则实现下列操作：中序周游左子树，访问根结点，中序周游右子树。

后序遍历：若二叉树非空，则实现下列操作：后序周游左子树，后序周游右子树，访问根结点。

3) 基于递归查找元素。

遍历二叉树，在遍历的过程中查找某元素是否在二叉树中。

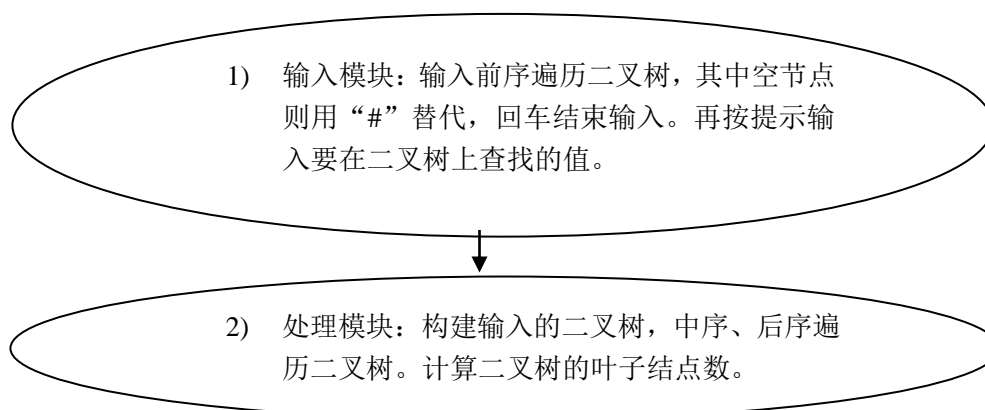
利用二叉树的遍历算法遍历二叉树，实现二叉树访问。若给定值等于结点的值，则查找成功。若遍历结束，仍没有查到与之相等的值，则查找不成功。

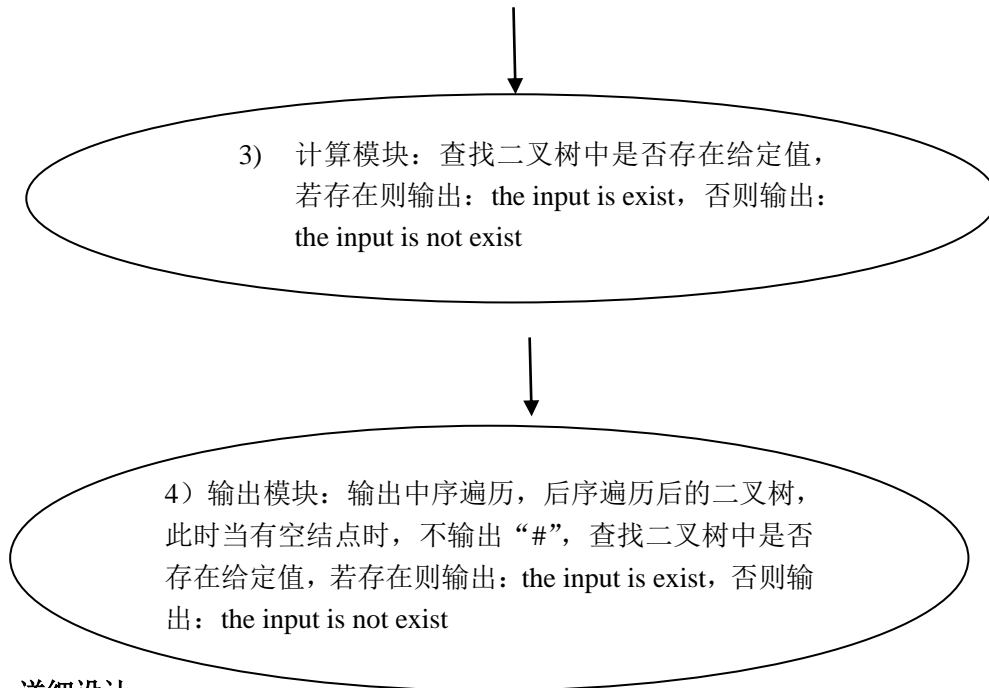
4) 基于遍历统计二叉树中叶子节点的个数。

算法动机：遍历二叉树，在遍历过程中查找叶子结点并计数。

利用二叉树的遍历算法遍历二叉树，实现访问二叉树。首先需要有一个变量来计数；访问每个结点时，判断是否为叶结点，并修改计数变量。

3.程序的流程





三、详细设计

1.物理数据类型

根据用户输入的数据类型定义相关的物理存储结构。如：当用户输入字符型时，二叉树的数据域物理储存类型同为字符型。

2.输入和输出的格式

输入格式：以前序遍历的方式输入树，空结点由#代替,例如：AB#D#CE##F##

输出格式：输出一串字符中间不含#。或者输出二叉树的叶子结点数。判断二叉树是否为空树，是则输出“the input is exist”，否则输出“the input is not exist”。

3.算法的具体步骤

树类：

```

class bintree{
private:
    binnode* root;
    void preorder( binnode* rt );
    void inorder(binnode* rt);
    void Count(binnode* rt);
    void posorder( binnode* rt );
    bool search(binnode* rt,char d);
public:
    bintree(){
        root = NULL;
    }
    bool search(char d);
}
    
```

```

void setroot( binnode* rt){
    root = rt;
}
void preorder();//前序遍历
void inorder();//中序遍历
void posorder();//后序遍历
void Count();//计算树的叶子结点数
binnode* CreatBiTree(){ // 先序递归创建二叉树
// 先按顺序驶入二叉树中节点的值(一个字符),空格字符代表空树
    char ch;
    if((ch=getchar()) == '#') // getchar() 为逐个读入标准库函数
        return NULL;
    else{
        binnode* T = new binnode(ch); // 产生新的子树
        T->setleft (CreatBiTree());
        // 递归创建左子树
        T->setright (CreatBiTree());
//      CreatBiTree(T->Right()); // 递归创建右子树
    }
}

```

};

输入，输出模块。

前序输出二叉树：

```

void bintree::preorder( binnode* rt ){//
    if ( rt == NULL ) return;
    cout << rt->data<<" ";
    preorder( rt->Left() );
    preorder( rt->Right() );
}

```

中序遍历输出二叉树：

```

void bintree::inorder(binnode* rt){
    if ( rt== NULL ) return;
    inorder( rt->Left() );
    cout << rt->data <<" ";
    inorder( rt->Right() );
}

```

后序遍历输出二叉树：

```

void bintree::posorder( binnode* rt ){
    if ( rt==NULL ) return;
    posorder( rt->Left() );
    posorder( rt->Right() );
    cout<< rt->data <<" ";
}

```

}

处理模块:

```

binnode* CreatBiTree(){ // 先序递归创建二叉树
    // 先按顺序驶入二叉树中节点的值(一个字符),空格字符代表空树
    char ch;
    if((ch=getchar()) == '#') {
        return NULL;
    }
    else{
        binnode* T = new binnode(ch); // 产生新的子树
        T->setleft(CreatBiTree());
        T->setright(CreatBiTree());
        return T;
    }
}

```

计算模块:

```

bool bintree::search(binnode* rt,char d){
    if(rt == NULL){

        return false;
    }
    if ( d==rt->data ){
        cout<<"The value  exists"<<endl;
        return true;
    }else{
        return (search(rt->Left(),d ) || search(rt->Right(),d ));
    }

}

void bintree::Count( binnode* rt){//计算叶子结点
    if ( rt == NULL ) return;
    if( rt->isLeaf() ){
        count++;
    }
    Count( rt->Left() );
    Count( rt->Right() );
}

T.Count();

char d;

```

```
cout<<"请输入要查找的值"<<endl;
cin>>d;
if ( !T.search(d) ){
    cout<< " The input does not exist"<<endl;
}
```

4.算法的时空分析

- 1) 创建二叉树: 时间复杂度 $\theta(n)$, 空间复杂度 $\theta(n)$ 因为在递归的过程中要存储一些信息。
 - 2) 遍历二叉树: 时间复杂度 $\theta(n)$, 空间复杂度 $\theta(n)$, 因为在递归的过程中要存储一些信息。
 - 3) 计算二叉树的深度: 时间复杂度 $\theta(n)$ 。
 - 4) 统计二叉树的叶子结点数: 基于递归时间复杂度 $\theta(n)$ 。
- 综上, 该程序的总的算法代价为 $\theta(n)$ 。

四、调试分析

1.调试方案设计

使用 demo 程序, 测序二叉树的各个基本操作的功能是否正常。

2.调试过程和结果, 及分析

在最终报告提交时, 还需要描述调试过程中发现的问题, 以及如何解决的。

五、测试结果

- 1) 输入: #
没有输出
输入: A
输出: The input does not exist

```
C:\Users\Administrator\Desktop\实验三.exe
创建一颗二叉树,其中a~z字符代表树的数据,用'#'表示空树:
#
preorder :
inorder :
posorder :
leapnode:
0
请输入要查找的值
A
The value non-exist

-----
Process exited after 4.939 seconds with return value 0
请按任意键继续. . .
```


2) 输入: A##

输出:

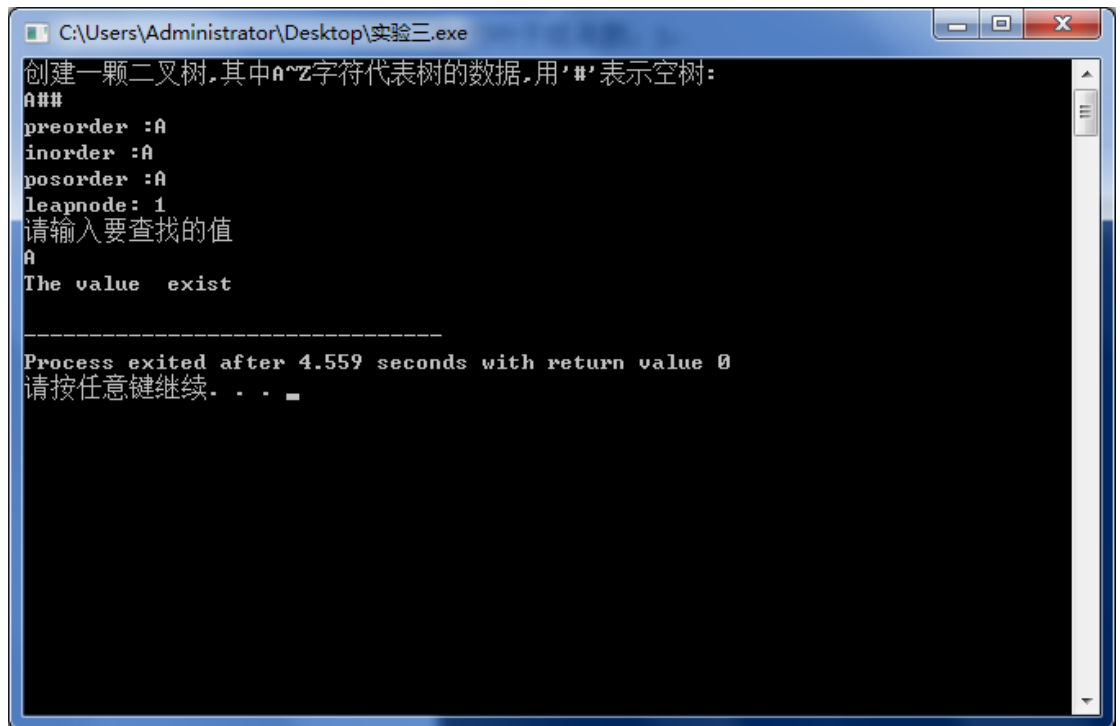
前序遍历: A

中序遍历: A

树的叶子结点数: 1

输入: A

输出: The input exists



```

C:\Users\Administrator\Desktop\实验三.exe
创建一颗二叉树,其中A~Z字符代表树的数据,用'#'表示空树:
A##
preorder :A
inorder :A
posorder :A
leafnode: 1
请输入要查找的值
A
The value exist

-----
Process exited after 4.559 seconds with return value 0
请按任意键继续. . .
    
```

3) 输入: A#B#C#D##

输出:

前序遍历: A B C D

中序遍历: A B C D

树的叶子结点数: 1

输入: E

输出: The input does not exist。



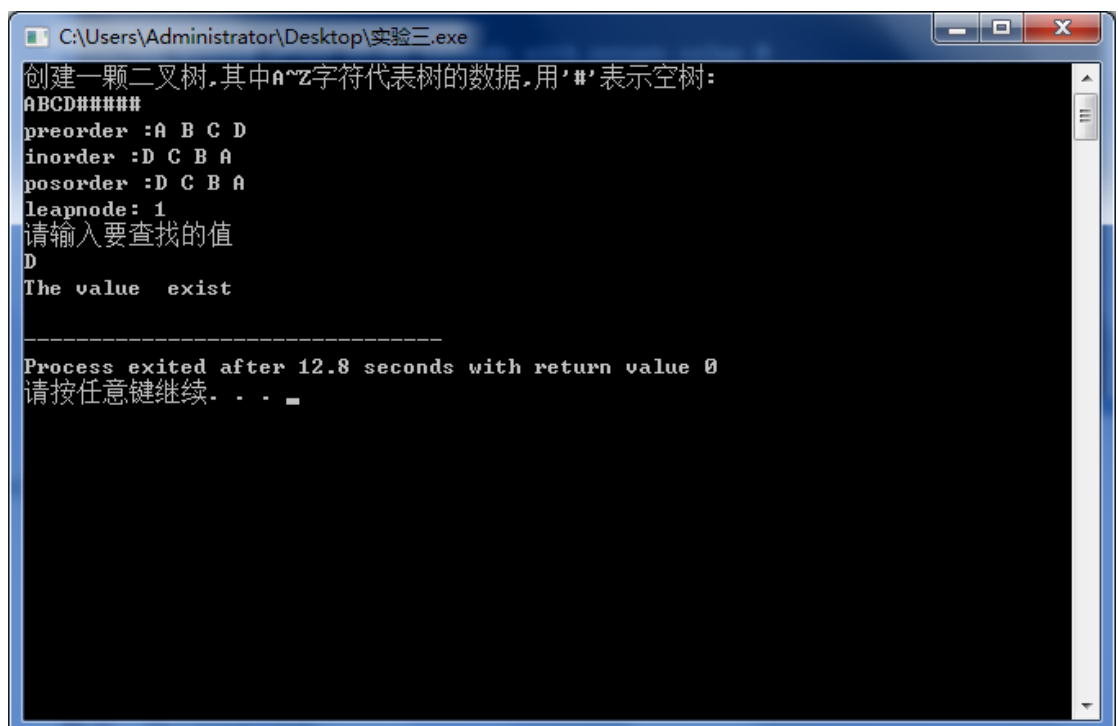
```

C:\Users\Administrator\Desktop\实验三.exe
创建一颗二叉树,其中a~z字符代表树的数据,用'#'表示空树:
A#B#C#D##
preorder :A B C D
inorder :A B C D
posorder :D C B A
leapnode: 1
请输入要查找的值
E
The value non-exist

-----
Process exited after 15.65 seconds with return value 0
请按任意键继续. . .

```

- 4) 输入: ABCD#####
 输出:
 前序遍历: A B C D
 中序遍历: D C B A
 树的叶子结点数: 1
 输入: D
 输出: The input exists。



```

C:\Users\Administrator\Desktop\实验三.exe
创建一颗二叉树,其中a~z字符代表树的数据,用'#'表示空树:
ABCD#####
preorder :A B C D
inorder :D C B A
posorder :D C B A
leapnode: 1
请输入要查找的值
D
The value exist

-----
Process exited after 12.8 seconds with return value 0
请按任意键继续. . .

```

5) 输入: AB#D##CEG###FH##I##

输出:

前序遍历: ABDCEGFHI

中序遍历: BDAGECHFI

后序遍历: DBG E H I F C A

树的叶子结点数: 4

输入: A

输出: The input exists。

```

C:\Users\Administrator\Desktop\实验三.exe
创建一颗二叉树,其中a~z字符代表树的数据,用'#'表示空树:
AB#D##CEG###FH##I##
preorder :A B D C E G F H I
inorder :B D A G E C H F I
posorder :D B G E H I F C A
leafnode: 4
请输入要查找的值
A
The value exist
-----
Process exited after 12.95 seconds with return value 0
请按任意键继续. . .
    
```

六、实验心得

由于二叉树的许多操作如构建、遍历都需要用到递归,在实现二叉树的过程中,加强了对递归的理解,在合适的时刻用递归,能简便我们的思路,让我们的代码可读性更强。同时,也对树这一重要的数据结构有了更深刻的理解。

七、用户使用说明(可选)

输入的仅为字符型数据,当输入缺少一个代表空格的字符#时,将会创建树不成功。