\*\*\*\*\*\*\*\*\*\*二叉树（基于数组）实现\*\*\*\*\*\*\*\*\*\*

1. **节点类(Anode.h)：**

typedef char E;

class ANode{

private:

E it;

int lc;// Index of left child

int rc;// Index of right child

int p;

public:

ANode() { lc = rc = 0; }

//返回节点元素

E element() { return it; }

//设置节点元素

void setElement(E e) { it = e; }

//返回左结点下标

int left() { return lc; }

//设置左孩子

void setLeft(int i) { lc = 2\*i+1; }

//返回右节点下标

int right() { return rc; }

//设置右孩子

void setRight(int i) { rc = 2\*i+2; }

};

1. **树类（BinTree.h）：**

#include <iostream>

#include <vector>

#include <queue>

#include <string.h>

using namespace std;

class ABinTree:public ANode{

private:

int root; // Root of the BinTree BinTree树的根结点

public:

vector<ANode> nodeArray; //用数组存储节点

int Root(){

return 0;

}

void setroot(int n){

root=n;

}

//是否为空树

bool BiTreeEmpty(int rt)

{

if(nodeArray[rt].element()=='#')

return true; //是一棵空树

else

return false;//不是空树

}

//前序遍历

void preorder(int rt)

{

if(nodeArray[rt].element()=='#')

return;

else

{

cout<<nodeArray[rt].element()<<" ";

preorder(nodeArray[rt].left());

preorder(nodeArray[rt].right());

}

}

//中序遍历

void Inorder(int rt)

{

if(nodeArray[rt].element()!='#')

{

Inorder(nodeArray[rt].left());

cout<<nodeArray[rt].element()<<" ";

Inorder(nodeArray[rt].right());

}

}

//后序遍历

void postorder(int rt)

{

if(nodeArray[rt].element()!='#')

{

postorder(nodeArray[rt].left());

postorder(nodeArray[rt].right());

cout<<nodeArray[rt].element()<<" ";

}

}

//层次遍历

void LevelOrderTraverse(int rt)

{

queue<int> q;

if(nodeArray[rt].element()!='#')

q.push(rt);

int temp;

while(!q.empty())

{

temp=q.front();

cout<<nodeArray[temp].element()<<' ';

q.pop();

int l=nodeArray[temp].left();

int r=nodeArray[temp].right();

//左右孩子不为空时则依次入队列

if(nodeArray[l].element()!='#')

q.push(l);

if(nodeArray[r].element()!='#')

q.push(r);

}

}

//树的深度

int BiTreeDepth(int rt)

{

int lh=0,rh=0 ;

if(nodeArray[rt].element()!='#')

{

lh=BiTreeDepth(nodeArray[rt].left());

rh=BiTreeDepth(nodeArray[rt].right());

return (lh>rh?lh:rh)+1;

}

else return 0 ;

}

//结点个数

int count(int rt)

{

if(nodeArray[rt].element()!='#') return count(nodeArray[rt].left())+count(nodeArray[rt].right())+1;

else return 0;

}

};

1. **测试类（ArrayBinTree.cpp）：**

#include <iostream>

#include<iomanip>

#include "ANode.h"

#include "ABinTree.h"

using namespace std;

typedef char E;

char c[50];

string s;

ANode a;

ABinTree T;

//建树操作 以层次遍历输入

int CreateBiTree(string s){

strcpy(c,s.c\_str());

for(int i=0;c[i]!='\0';i++){

a.setLeft(i);

a.setRight(i);

a.setElement(c[i]);

T.nodeArray.push\_back(a);

}

return 0;//返回根节点下标

}

int main()

{

int r;

int depth;

int num;

cout<<"提示："<<endl

<<"以字符串形式输入要构建的二叉树时，按层次遍历的顺序输入，所有的空节点以#代替"<<endl

<<"注意！要输入到最深层叶节点的下一层，并且该层元素为2^i个#符号！"<<endl

<<"例：ABC#DEF########"<<endl

<<"请输入：";

cin>>s;

//建树

r=CreateBiTree(s);

cout<<"-----------------------------------"<<endl;

cout<<"判断树是否是空树：";

if(T.BiTreeEmpty(r))

cout<<"输入为一棵空树"<<endl;

else cout<<"不是空树"<<endl;

cout<<"前序遍历：";

T.preorder(T.Root());

cout<<endl;

cout<<"中序遍历：";

T.Inorder(r);

cout<<endl;

cout<<"后序遍历：";

T.postorder(r);

cout<<endl;

cout<<"层次遍历：";

T.LevelOrderTraverse(r);

cout<<endl;

cout<<"树的深度：";

depth=T.BiTreeDepth(r);

cout<<depth<<endl;

cout<<"树的结点个数：";

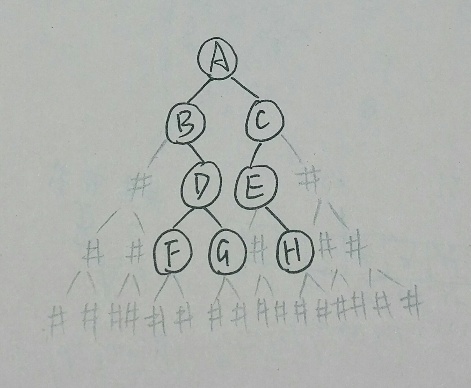
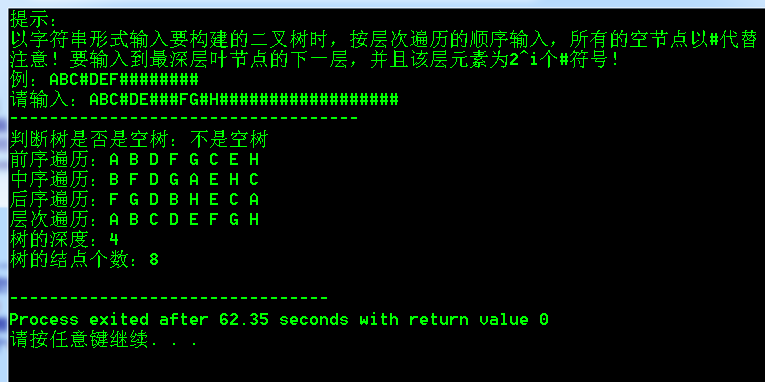
num=T.count(r);

cout<<num<<endl;

return 0;

}

**测试结果示例：**



**【附录】**

版本信息声明：

Dev-C++ 5.11

TDM-GCC 4.9.2 64-bit Release

资料整理人：  
物联网工程1402班 宁静仪

物联网工程1402班 吴彦彦