中 国 矿 业 大 学

2018 级《数据结构与算法分析》课程作业

学生姓名 王茂凯

学　　号 04181425

中国矿业大学信控学院

1. 在结点个数为 n（n > 1）的各棵树中，深度最小的树的深度是多少？它有多少叶结点？多少分支结点？深度最大的树的深度是多少？它有多少叶结点？多少分支结点？

深度最小为 2,n-1 个叶结点,1 个分支结点

深度最大为 n,1 个叶结点,n-1 个分支结点

2. 设一棵完全二叉树的第 k 层（根节点所处层次为 1）有 m 个叶节点。（$1 \le m < 2^{k-1}$）；

（1）该完全二叉树最少有多少节点？最多有多少节点？

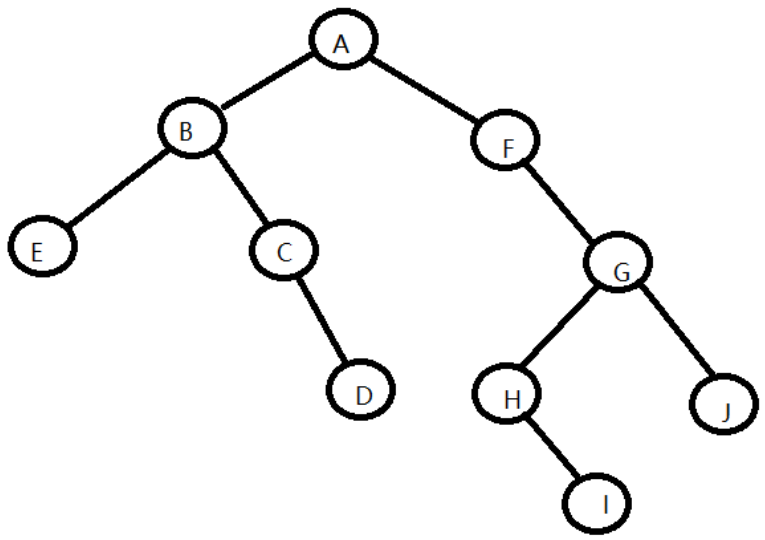
（2）该完全二叉树的深度可能是多少？

最少 2^{k-1} 个结点,最多 $2^{k+1}-3$ 个结点

深度可能是 k 或 k+1

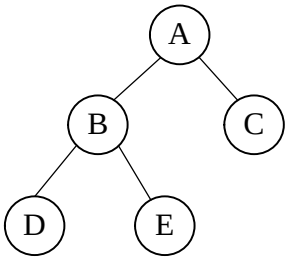
3. 已知一棵二叉树的前序遍历的结果是 ABECDFGHIJ，中序序列遍历的结果是 EBCDAFHIGJ，试画出这棵二叉树。

画图工具所画



4. 已知前序、中序序列（如上题所示），写出构造二叉树的算法?(递归或非递归实现) 说明: (1)程序要列出源代码; (2) 源代码要附上必要的注释; (3) 要附上必要的程序运行结果。

5. 设二叉树采用二叉链表表示，指针 root 指向根结点，试编写一个在二叉树中查找值为 x 的结点,并打印该结点所有祖先结点的算法。在此算法中，假设值为 x 的结点不多于一个。（可写伪代码）



运行结果

```
construct a binary tree according the preorder and inorder
test the tree by order:
preorder:
A B E C D F G H I J
inorder:
E B C D A F H I G J
postorder:
E D C B I H J G F A
levelorder:
A B F E C G D H J I
construct a binary tree and find a node and find its fathers
ABD**E**C**
preorder:
A B D E C
inorder:
D B E A C
find success
output its father
B A
find none
```

Main 函数部分

```
int main()
{
    cout << "construct a binary tree according the preorder and inorder
" << endl;
    Btree tree1 = pre_in_creatrtree("ABECDFGHIJ", "EBCDAFHIGJ", 10);
    cout << "test the tree by order:" << endl;
```

```
    cout << "preorder:" << endl;
    preorder(tree1); //ABECDFGHIJ
    cout << endl;
    cout << "inorder:" << endl;
    inorder(tree1); //EBCDAFHIGJ
    cout << endl;
    cout << "postorder:" << endl;
    postorder(tree1); //EDCBIHJGFA
    cout << endl;
    cout << "levelorder:" << endl;
    levelorder(tree1); //ABFECGDHJI
    cout << endl;
    cout << "construct a binary tree and find a node and find its fathe
rs" << endl;
    Btree tree2 = nullptr;
    Createtree(tree2);           //ABD**E**C**
    cout << "preorder:" << endl; //测试先序
    preorder(tree2);             //ABDEC
    cout << endl;
    cout << "inorder:" << endl; //测试中序
    inorder(tree2);             //DBEAC
    cout << endl;
    if (findnode(tree2, 'E')) //测试一种找到的情况
    {
        cout << "find success" << endl;
        cout << "output its father" << endl;
        findfather(tree2, 'E'); //BA
        cout << endl;
    }
    else
        cout << "find none" << endl;
    if (findnode(tree2, 'G')) //测试一种没有找到的情况
    {
        cout << "find success" << endl;
        cout << "output its father" << endl;
        findfather(tree2, 'G');
        cout << endl;
    }
    else
        cout << "find none" << endl;
    return 0;
}
```

完整代码如下

```
#include <iostream>
#include <queue>
#include <stack>
using namespace std;
typedef struct Bnode //二叉树存储结构
{
    char data;
    struct Bnode *lchild;
    struct Bnode *rchild;
} Bnode, *Btree;
void Createtree(Btree &T) //创建二叉树函数,补空法,将二叉树用*补为满二叉树
{
    //按先序次序输入二叉树结点的值(一个字符)
    char ch;
    cin >> ch;
    if (ch == '*') //*表示补的叶子
        return;
    else
    {
        T = new Bnode(); //生成新结点
        T->data = ch;
        Createtree(T->lchild); //递归创建左子树
        Createtree(T->rchild); //递归创建右子树
    }
}
void preorder(Btree T) //先序遍历
{
    if (T)
    {
        cout << T->data << " ";
        preorder(T->lchild);
        preorder(T->rchild);
    }
}
void inorder(Btree T) //中序遍历
{
    if (T)
    {
        inorder(T->lchild);
        cout << T->data << " ";
        inorder(T->rchild);
    }
```

```
}
void postorder(Btree T) //后序遍历
{
    if (T)
    {
        postorder(T->lchild);
        postorder(T->rchild);
        cout << T->data << " ";
    }
}
void levelorder(Btree T) //层次遍历
{
    Btree p;
    if (T != nullptr)
    {
        queue<Btree> que;
        que.push(T); //根结点入队
        while (!que.empty())
        {
            //根结点出队并将其左右孩子入队
            p = que.front();
            que.pop();
            cout << p->data << " ";
            if (p->lchild)
                que.push(p->lchild);
            if (p->rchild)
                que.push(p->rchild);
        }
    }
}
//前序中序还原树
Btree pre_in_creatrtree(const char *pre, const char *mid, int len)
{
    if (len == 0)
        return nullptr;
    char ch = pre[0]; //先序第一个结点为根结点
    int index = 0;
    while (mid[index] != ch) //在中序中找根结点
        index++;
    //构建根结点
    Btree T = new Bnode();
    T->data = ch;
    //递归构建左子树
    T->lchild = pre_in_creatrtree(pre + 1, mid, index);
```

```
    //递归构建右子树
    T->rchild = pre_in_creatrtree(pre + index + 1, mid + index + 1, len
 - index - 1);
    return T;
}
//后序中序还原树
Btree pos_in_creatrtree(const char *pos, const char *mid, int len)
{
    if (len == 0)
        return nullptr;
    char ch = pos[len - 1]; //后序最后一个结点为根结点
    int index = 0;
    while (mid[index] != ch) //在中序中找根结点
        index++;
    //构建根结点
    Btree T = new Bnode();
    T->data = ch;
    //递归构建左子树
    T->lchild = pos_in_creatrtree(pos, mid, index);
    //递归构建右子树
    T->rchild = pos_in_creatrtree(pos + index, mid + index + 1, len - i
ndex - 1);
    return T;
}
//寻早某结点
bool findnode(const Btree &T, const char &val)
{
    if (T != nullptr)
    {
        if (T->data == val) //找到返回 true
            return true;
        else if (findnode(T->lchild, val) || findnode(T->rchild, val))
        {
            return true;
        }
    }
    return false;
}
//打印结点所有祖先
bool findfather(const Btree &T, const char &val)
{
    if (T != nullptr)
    {
        if (T->data == val) //找到返回 true
```

```
            return true;
        //没有找到就寻早其子树,并打印经过的父结点的值
        else if (findfather(T->lchild, val) || findfather(T->rchild, va
l))
        {
            cout << T->data << " ";
            return true;
        }
    }
    return false;
}

int main()
{
    cout << "construct a binary tree according the preorder and inorder
" << endl;
    Btree tree1 = pre_in_creatrtree("ABECDFGHIJ", "EBCDAFHIGJ", 10);
    cout << "test the tree by order:" << endl;
    cout << "preorder:" << endl;
    preorder(tree1); //ABECDFGHIJ
    cout << endl;
    cout << "inorder:" << endl;
    inorder(tree1); //EBCDAFHIGJ
    cout << endl;
    cout << "postorder:" << endl;
    postorder(tree1); //EDCBIHJGFA
    cout << endl;
    cout << "levelorder:" << endl;
    levelorder(tree1); //ABFECGDHJI
    cout << endl;
    cout << "construct a binary tree and find a node and find its fathe
rs" << endl;
    Btree tree2 = nullptr;
    Createtree(tree2);           //ABD**E**C**
    cout << "preorder:" << endl; //测试先序
    preorder(tree2);             //ABDEC
    cout << endl;
    cout << "inorder:" << endl; //测试中序
    inorder(tree2);             //DBEAC
    cout << endl;
    if (findnode(tree2, 'E')) //测试一种找到的情况
    {
        cout << "find success" << endl;
        cout << "output its father" << endl;
```

```
        findfather(tree2, 'E'); //BA
        cout << endl;
    }
    else
        cout << "find none" << endl;
    if (findnode(tree2, 'G')) //测试一种没有找到的情况
    {
        cout << "find success" << endl;
        cout << "output its father" << endl;
        findfather(tree2, 'G');
        cout << endl;
    }
    else
        cout << "find none" << endl;
    return 0;
}
```