二叉树和树

一 目的

1、掌握二叉树的存储结构；

2、掌握二叉树的递归遍历算法；

3、掌握二叉树的各种遍历算法及应用；

4、训练和培养良好的程序设计能力和综合编程能力

二　内容

1、对于任意二叉树，实现二叉树的二叉链表存储结构。

2、在二叉链表存储的基础上，通过递归方式实现二叉树的前、中、后序遍历算法。

3、求二叉链表存储的二叉树的深度。

4、编写递归算法，计算二叉树中叶子结点的数目。

5、编写算法实现二叉树在二叉链表存储结构上的层次遍历，并输出层次遍历的结果。

6*、通过非递归遍历的方式实现二叉树的中序遍历，并实现前序和后序遍历算法。

7*、实现字符串的最长前缀匹配问题。

三 设计说明

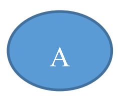
首先设计 TreeNode 类实现二叉树链式存储，对节点进行定义，对元素进行赋值存储，在二叉链表的基础上通过递归方式实现二叉树的前、中、后序遍历算法。

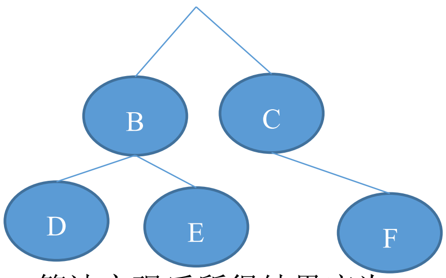
四 功能说明

Height()求树的深度，Size()求树的节点数，`PreOrder(TreeNode subTree)`构造先序遍历方法，`public void InOrder(TreeNode subTree)`构造中序遍历方法，`public void PostOrder(TreeNode subTree)`构造后序遍历方法。

五 调试分析

设计初创建原始二叉树如下图所示





算法实现后所得结果应为：

层次遍历结果：ABCDEF

先序遍历结果：ABDECF

中序遍历结果：DBEACF

后序遍历结果：DEBFCA

上机过程中原计划实现层次遍历的非递归算法，由于队列类引用失败未能实现。

六 测试结果

```
the size of the tree is: 6
深度: 3
*******先根(前序)[ABDECF]遍历*****************
key:1--name:rootNode()
key:2--name:B
key:4--name:D
key:5--name:E
key:3--name:C
key:6--name:F
*******中根(中序)[DBEACF]遍历*****************
key:4--name:D
key:2--name:B
key:5--name:E
key:1--name:rootNode()
key:3--name:C
key:6--name:F
*******后根(后序)[DEBFCA]遍历*****************
key:4--name:D
key:5--name:E
key:2--name:B
key:6--name:F
key:3--name:C
key:1--name:rootNode()
```

七 带注释的源代码

```java
package sj;
public class TreeNode {
```

```java
private int key = 0;
private String data = null;
private boolean isVisted = false;
private TreeNode leftChild = null;
private TreeNode rightChild = null;
public TreeNode(){
}
public TreeNode(int key, String data){
this.key = key;
this.data = data;
this.leftChild = null;
this.rightChild = null;
}
public int getKey() {
return key;
}
public void setKey(int key) {
this.key = key;
}
public String getData() {
return data;
}
public void setData(String data) {
this.data = data;
}
public TreeNode getLeftChild() {
return leftChild;
}
public void setLeftChild(TreeNode leftChild) {
this.leftChild = leftChild;
}
public TreeNode getRightChild() {
return rightChild;
}
public void setRightChild(TreeNode rightChild) {
this.rightChild = rightChild;
}
public boolean isVisted() {
return isVisted;
}
public void setVisted(boolean isVisted) {
this.isVisted = isVisted;
}
public static class BinaryTree {
```

```java
private TreeNode root = null;
public BinaryTree() {
root = new TreeNode(1, "rootNode()");
}
public void createBinTree(TreeNode root){
TreeNode newNodeB = new TreeNode(2,"B");
TreeNode newNodeC = new TreeNode(3,"C");
TreeNode newNodeD = new TreeNode(4,"D");
TreeNode newNodeE = new TreeNode(5,"E");
TreeNode newNodeF = new TreeNode(6,"F");
root.setLeftChild(newNodeB);
root.setRightChild(newNodeC);
root.getLeftChild().setLeftChild(newNodeD);
root.getLeftChild().setRightChild(newNodeE);
root.getRightChild().setRightChild(newNodeF);
}
public boolean IsEmpty() {
// 判二叉树空否
return root == null;
}
public int Height() {
// 求树高度
return Height(root);
}
public int Height(TreeNode subTree) {
if (subTree == null)
return 0; //递归结束：空树高度为 0
else {
int i = Height(subTree.getLeftChild());
int j = Height(subTree.getRightChild());
return (i < j) ? j + 1 : i + 1;
}
}
public int Size() {
// 求结点数
return Size(root);
}
public int Size(TreeNode subTree) {
if (subTree == null)
return 0;
```

```
else {
return 1 + Size(subTree.getLeftChild())
+ Size(subTree.getRightChild());
}
}
public TreeNode Parent(TreeNode element) {

//返回双亲结点

return (root == null || root == element) ? null : Parent(root, element);
}
public TreeNode Parent(TreeNode subTree, TreeNode element) {
if (subTree == null)
return null;
if (subTree.getLeftChild() == element|| subTree.getRightChild() == element)

//找到，返回父结点地址

return subTree;
TreeNode p;

//先在左子树中找，如果左子树中没有找到，才到右子树去找

if ((p = Parent(subTree.getLeftChild(), element)) != null)

//递归在左子树中搜索

return p;
else

//递归在左子树中搜索

return Parent(subTree.getRightChild(), element);
}
public TreeNode LeftChild(TreeNode element) {

//返回左子树

return (element != null) ? element.getLeftChild() : null;
}
public TreeNode RightChild(TreeNode element) {

//返回右子树

return (element != null) ? element.getRightChild() : null;
}
public TreeNode getRoot() {

//取得根结点

return root;
}
```

```java
public void destroy(TreeNode subTree) {

//私有函数：删除根为 subTree 的子树

if (subTree != null) {

destroy(subTree.getLeftChild()); //删除左子树

destroy(subTree.getRightChild()); //删除右子树

//delete subTree; //删除根结点

subTree = null;
}
}
public void Traverse(TreeNode subTree) {
System.out.println("key:" + subTree.getKey() + "--name:"
+ subTree.getData());
Traverse(subTree.getLeftChild());
Traverse(subTree.getRightChild());
}
public void PreOrder(TreeNode subTree) {

//先根

if (subTree != null) {
visted(subTree);
PreOrder(subTree.getLeftChild());
PreOrder(subTree.getRightChild());
}
}
public void InOrder(TreeNode subTree) {

//中根

if (subTree != null) {
InOrder(subTree.getLeftChild());
visted(subTree);
InOrder(subTree.getRightChild());
}
}
public void PostOrder(TreeNode subTree) {

//后根

if (subTree != null) {
PostOrder(subTree.getLeftChild());
PostOrder(subTree.getRightChild());
visted(subTree);
```

```java
}
}

public void LevelOrder(TreeNode subTree) {
//水平遍边
}
public boolean Insert(TreeNode element){
//插入
return true;
}
public boolean Find(TreeNode element){
//查找
return true;
}
public void visted(TreeNode subTree) {
subTree.setVisted(true);
System.out.println("key:" + subTree.getKey() + "--name:"
+ subTree.getData());
}
public static void main(String[] args) {
BinaryTree bt = new BinaryTree();
bt.createBinTree(bt.root);
System.out.println("the size of the tree is: " + bt.Size());
System.out.println("深度: " + bt.Height());

System.out.println("*******先根(前序)[ABDECF]遍历****************");
bt.PreOrder(bt.root);
System.out.println("*******中根(中序)[DBEACF]遍历****************");
bt.InOrder(bt.root);
System.out.println("*******后根(后序)[DEBFCA]遍历****************");
bt.PostOrder(bt.root);
}
}
    }
```