实验12

练习1

题目

要求用孩子兄弟链表实现树。根据一棵树的先序和后序序列（即对应二叉树的先序和中序序列）构造该树。输出该树的叶子个数、规模、度和高度。

解析

树的规模可以用任意遍历方式计数，树的高度及叶子个数都可以用层序遍历简单实现。

下面简单说明一下计算度的方法：

1. 度数初始化为0。建立辅助队列queue，初始时根节点入队，然后使空结点入队作为分隔。
2. 重复下面几个步骤，直至队列为空。
3. 出队，若出队结点为空结点，使新的空结点入队，并直接进入下一个循环。
4. 使出队结点的孩子以及孩子的所有兄弟入队，统计这些结点的个数，若该值大于度数，将度数更新为该值。

代码

CST.java

```
// 使用Java泛型实现的二叉树，利用扩展二叉树的层序序列构建。
public class CST<Item> {
    private Node root;

    private class Node {
        Item data;
        Node child;
        Node sibling;

        public Node(Item data, Node child, Node sibling) {
            this.data = data;
            this.child = child;
            this.sibling = sibling;
        }
    }

    public CST() {
        this.root = null;
    }

    public CST(Item[] preOrder, Item[] postOrder) {
        this.root = buildTree(preOrder, postOrder, 0, 0, preOrder.length);
    }

    private Node buildTree(Item[] preOrder, Item[] postOrder, int preIndex, int
postIndex, int count)
    {
```

```
        if (count > 0) {
            Item r = preOrder[preIndex];
            int i = 0;
            for (; i < count; i++) {
                if (r == postOrder[i + postIndex]) {
                    break;
                }
            }
            Node pRoot = new Node(r, null, null);
            pRoot.child = buildTree(preOrder, postOrder, preIndex + 1,
postIndex, i);
            pRoot.sibling = buildTree(preOrder, postOrder, preIndex + i + 1,
                                      postIndex + i + 1, count - i - 1);
            return pRoot;
        }
        return null;
    }

    public int size() {
        int result = 0;
        if (this.root != null) {
            Queue<Node> queue = new Queue<Node>();
            queue.enqueue(this.root);
            while (!queue.isEmpty()) {
                Node p = queue.dequeue();
                result++;
                if (p.child != null) {
                    queue.enqueue(p.child);
                }
                if (p.sibling != null) {
                    queue.enqueue(p.sibling);
                }
            }
        }
        return result;
    }

    public int leaf() {
        int result = 0;
        if (this.root != null) {
            Queue<Node> queue = new Queue<Node>();
            queue.enqueue(this.root);
            while (!queue.isEmpty()) {
                Node p = queue.dequeue();
                if (p.child != null) {
                    queue.enqueue(p.child);
                } else {
                    result++;
                }
                if (p.sibling != null) {
                    queue.enqueue(p.sibling);
                }
            }
        }
        return result;
    }

    public int degree() {
```

```
        if (this.root == null) {
            return 0;
        }
        int result = 0;
        Queue<Node> queue = new Queue<Node>();
        queue.enqueue(this.root);
        queue.enqueue(new Node(null, null, null));
        while (queue.getHead() != null) {
            Node p = queue.dequeue();
            if (p.data == null && queue.getHead() != null) {
                queue.enqueue(new Node(null, null, null));
                continue;
            }
            p = p.child;
            int degree = 0;
            while (p != null) {
                degree++;
                p = p.sibling;
            }
            if (degree > result) {
                result = degree;
            }
        }
        return result;
    }

    public int height() {
        // 不包含根节点所在层
        if (this.root == null) {
            return 0;
        }
        int result = 0;
        Queue<Node> queue = new Queue<Node>();
        queue.enqueue(this.root);
        queue.enqueue(new Node(null, null, null));
        while (queue.getHead() != null) {
            Node p = queue.dequeue();
            if (p.data == null && queue.getHead() != null) {
                result++;
                queue.enqueue(new Node(null, null, null));
                continue;
            }
            p = p.child;
            while (p != null) {
                queue.enqueue(p);
                p = p.sibling;
            }
        }
        return result;
    }
}
```

Exercise_1.java

```
public class Exercise_1 {
    public static void main(String[] args) throws Exception {
        // read data and build tree
        String[] preOrder  = {"A", "B", "E", "K", "L", "C", "F", "G", "D", "H",
"I", "M", "J"};
        String[] postOrder = {"K", "L", "E", "B", "F", "G", "C", "H", "M", "I",
"J", "D", "A"};
        CST<String> a = new CST<String>(preOrder, postOrder);
        // output
        System.out.println("Leaf:   " + a.leaf());
        System.out.println("Size:   " + a.size());
        System.out.println("Degree: " + a.degree());
        System.out.println("Height: " + a.height());
    }
}
```

输入

```
无
```

输出

```
Leaf:   7
Size:   13
Degree: 3
Height: 3
```

心得体会

1.