# 上海大学计算机工程与科学学院

# 数据结构 (2) 个人作业报告

作业：第九周上机练习

姓名：林艺珺

学号：18120189

日期：2020 年 3 月 20 日

# 第一题 树的宽度

在树的孩子兄弟链表示中，设计并实现相应函数，统计树的宽度（树的宽度是指各层中结点的最大数目）。

## 测试分析

题目所要求的统计树的宽度，即各层中结点的最大数目。而每一层结点的总数，可首先明确每一结点各在哪一层后进行统计。因此，在孩子兄弟表示树结点类添加一 int 类型数据成员 level，表示该结点在树中所在的层数，以根节点为例，其在第 1 层。

因为每一结点所在层数等于其双亲结点的层数加一，所以使用层次遍历来标记每一结点所在层数。

使用层次遍历统计相同层数结点个数以求得当前层数的宽度，最后找出所有层数中宽度最大的即为树的宽度。

## 源码程序

### 标记结点所在层数算法

```
template <class ElemType>
void ChildSiblingTree<ElemType>::TreeLevel()
{
    LinkQueue<ChildSiblingTreeNode<ElemType> *> q;
    // 定义队列对象
    ChildSiblingTreeNode<ElemType> *cur, *p;
    root->level = 1;
    if (root != NULL) q.EnQueue(root);
    // 如果根非空，则根结点指针入队列
    while (!q.IsEmpty())    {
        q.DelQueue(cur);
        // 队头结点出队为当前结点cur
        for (p = FirstChild(cur); p != NULL;  p = NextSibling(p)) {
            q.EnQueue(p);
            // 依次将cur的孩子结点指针入队列
            p->level = cur->level + 1;
            // 每一结点所在层数等于其双亲结点的层数加一
        }
    }
}
```

**统计树的宽度算法**

```
template <class ElemType>
int ChildSiblingTree<ElemType>::TreeWidth()
{
    int width[1000] = {0};
    width[0] = -999;
    int maxWidth = width[0];
    LinkQueue<ChildSiblingTreeNode<ElemType> *> q;
    // 定义队列对象
    ChildSiblingTreeNode<ElemType> *cur, *p;
    for (int i = 1; i <= Height(); i++) {
        if (root != NULL) q.EnQueue(root);
        // 如果根非空,则根结点指针入队列
        while (!q.IsEmpty())    {
            q.DelQueue(cur);
            // 队头结点出队为当前结点cur
            for (p = FirstChild(cur); p != NULL;  p = NextSibling(p)) {
                q.EnQueue(p);
                // 依次将cur的孩子结点指针入队列
            }
            if (cur->level == i) width[i]++;
            // 当前层数的宽度
        }
        if (width[i] > maxWidth) maxWidth = width[i];
    }
    return maxWidth;
}
```

测试程序

```
//样例一
char items[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'};
int parents[] = {-1, 0, 0, 0, 1, 1, 3, 3};
int r = 0, n = 8, cur;
//样例二
char items[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I'};
int parents[] = {-1, 0, 0, 0, 1, 1, 2, 3, 3};
int r = 0, n = 9, cur;
//样例三
char items[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I'};
int parents[] = {-1, 0, 0, 0, 0, 1, 1, 3, 3};
int r = 0, n = 9, cur;
//调用函数
cout << "树的宽度为：";
t.TreeLevel();
cout << t.TreeWidth() << endl;
```

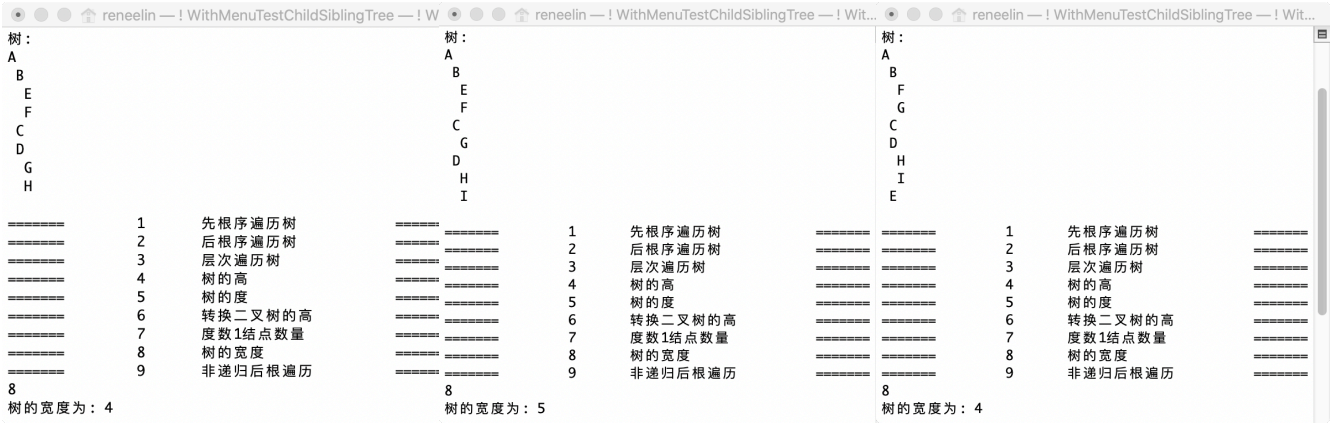


图 1: 样例一




图 2: 样例二




图 3: 样例三

# 第二题 非递归后根遍历

在树的孩子兄弟链表示中，设计非递归函数，实现树的后根遍历。

## 测试分析

非递归的树的后根遍历需要从最基础的逻辑判断：首先找到最左侧的结点，接下来判断其是否有兄弟结点，兄弟结点遍历完毕后返回其双亲结点，再继续判断其兄弟结点，直至返回至根节点。因此使用栈来辅助后根遍历。另外，在孩子兄弟表示树结点类添加一 bool 类型数据成员 f，初始值为 0，表示该结点是否已被遍历。

栈的使用中，先使根节点入栈，其次循环寻找第一个孩子结点直至没有，将栈顶元素出栈并标记遍历，再寻找其下一个兄弟结点并入栈，进入下一个循环对现在结点循环寻找第一个孩子结点..... 直至栈为空，遍历完毕。

## 源码程序

**非递归后根遍历算法**

```
template <class ElemType>
void ChildSiblingTree<ElemType>::NonRecursivePostRootOrder
                    (void (*Visit)(const ElemType &)) const
{
    LinkStack<ChildSiblingTreeNode<ElemType> *> s;
    // 定义栈对象
    ChildSiblingTreeNode<ElemType> *cur;
    if (root != NULL) s.Push(root);
    // 如果根非空，则根结点指针入栈
    cur = root;
    while(!s.IsEmpty()) {
        while (cur->firstChild != NULL && cur->firstChild->f == 0) {
            s.Push(cur->firstChild);
            cur = cur->firstChild;
        }
        s.Pop(cur);
        (*Visit)(cur->data);
        cur->f = 1;
        if (cur->nextSibling != NULL){
            s.Push(cur->nextSibling);
            cur = cur -> nextSibling;
        }
    }
}
```

测试程序

```
//样例一
char items[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'};
int parents[] = {-1, 0, 0, 0, 1, 1, 3, 3};
int r = 0, n = 8, cur;
//样例二
char items[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I'};
int parents[] = {-1, 0, 0, 0, 1, 1, 2, 3, 3};
int r = 0, n = 9, cur;
//样例三
char items[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I'};
int parents[] = {-1, 0, 0, 0, 0, 1, 1, 3, 3};
int r = 0, n = 9, cur;
//调用函数
cout << "树的后根序遍历：";
t.PostRootOrder(Write);
cout << endl;
cout << "非递归后根遍历：";
t.NonRecursivePostRootOrder(Write);
cout << endl;
```

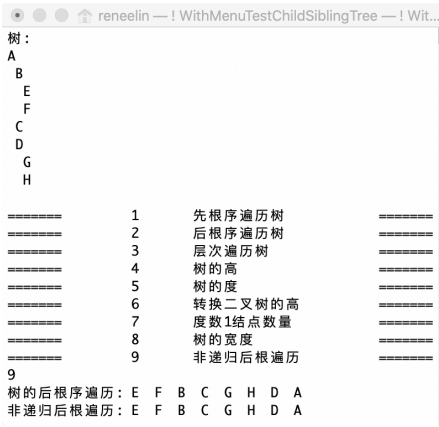


图 4: 样例一

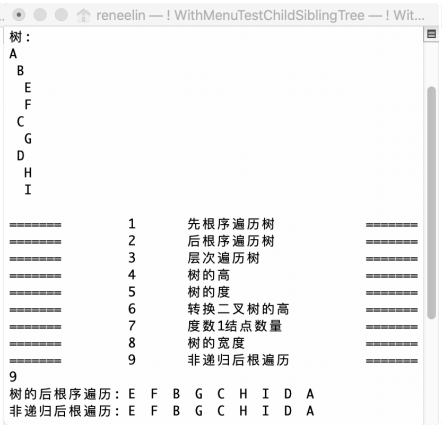


图 5: 样例二

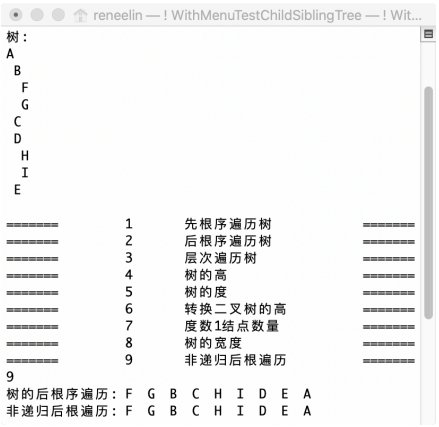


图 6: 样例三