

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

```
1  template <class ElemType>
2  void ChildSiblingTree<ElemType>::TreeLevel()
3  {
4      LinkQueue<ChildSiblingTreeNode<ElemType> *> q;
5      // 定义队列对象
6      ChildSiblingTreeNode<ElemType> *cur, *p;
7      root->level = 1;
8      if (root != NULL) q.Enqueue(root);
9      // 如果根非空,则根结点指针入队列
10     while (!q.IsEmpty()) {
11         q.Dequeue(cur);
12         // 队头结点出队为当前结点 cur
13         for (p = FirstChild(cur); p != NULL; p = NextSibling(p)) {
14             q.Enqueue(p);
15             // 依次将 cur 的孩子结点指针入队列
16             p->level = cur->level + 1;
17             // 每一结点所在层数等于其双亲结点的层数加一
18         }
19     }
20 }
```

统计树的宽度算法

```
1  template <class ElemType>
2  int ChildSiblingTree<ElemType>::TreeWidth()
3  {
4      int width[1000] = {0};
5      width[0] = -999;
6      int maxWidth = width[0];
7      LinkQueue<ChildSiblingTreeNode<ElemType> *> q;
8      // 定义队列对象
9      ChildSiblingTreeNode<ElemType> *cur, *p;
10     for (int i = 1; i <= Height(); i++) {
11         if (root != NULL) q.Enqueue(root);
12         // 如果根非空,则根结点指针入队列
13         while (!q.IsEmpty()) {
14             q.DeQueue(cur);
15             // 队头结点出队为当前结点 cur
16             for (p = FirstChild(cur); p != NULL; p = NextSibling(p)) {
17                 q.Enqueue(p);
18                 // 依次将 cur 的孩子结点指针入队列
19             }
20             if (cur->level == i) width[i]++;
21             // 当前层数的宽度
22         }
23         if (width[i] > maxWidth) maxWidth = width[i];
24     }
25     return maxWidth;
26 }
```

测试程序

```

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

```

Figure 1 (Example 1) shows a tree with root A and children B, C, D, E, F, G, H. The output statistics are: 1. 先根序遍历树 (1), 2. 后根序遍历树 (2), 3. 层次遍历树 (3), 4. 树的高 (4), 5. 树的度 (5), 6. 转换二叉树的高 (6), 7. 度数1结点数量 (7), 8. 树的宽度 (8), 9. 非递归后根遍历 (9). The final output is: 树的宽度为: 4.

Figure 2 (Example 2) shows a tree with root A and children B, C, D, E, F, G, H, I. The output statistics are: 1. 先根序遍历树 (1), 2. 后根序遍历树 (2), 3. 层次遍历树 (3), 4. 树的高 (4), 5. 树的度 (5), 6. 转换二叉树的高 (6), 7. 度数1结点数量 (7), 8. 树的宽度 (8), 9. 非递归后根遍历 (9). The final output is: 树的宽度为: 5.

Figure 3 (Example 3) shows a tree with root A and children B, C, D, E, F, G, H, I. The output statistics are: 1. 先根序遍历树 (1), 2. 后根序遍历树 (2), 3. 层次遍历树 (3), 4. 树的高 (4), 5. 树的度 (5), 6. 转换二叉树的高 (6), 7. 度数1结点数量 (7), 8. 树的宽度 (8), 9. 非递归后根遍历 (9). The final output is: 树的宽度为: 4.

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

```
1  template <class ElemType>
2  void ChildSiblingTree<ElemType>::NonRecursivePostRootOrder
3      (void (*Visit)(const ElemType &)) const
4  {
5      LinkStack<ChildSiblingTreeNode<ElemType> *> s;
6      // 定义栈对象
7      ChildSiblingTreeNode<ElemType> *cur;
8      if (root != NULL) s.Push(root);
9      // 如果根非空,则根结点指针入栈
10     cur = root;
11     while(!s.IsEmpty()) {
12         while (cur->firstChild != NULL && cur->firstChild->f == 0) {
13             s.Push(cur->firstChild);
14             cur = cur->firstChild;
15         }
16         s.Pop(cur);
17         (*Visit)(cur->data);
18         cur->f = 1;
19         if (cur->nextSibling != NULL) {
20             s.Push(cur->nextSibling);
21             cur = cur->nextSibling;
22         }
23     }
24 }
```

测试程序

```

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

```

Figure 4 (Example 1) shows a tree with root A and children B, C, D, E, F, G, H. The statistics are: 1. 先根序遍历树, 2. 后根序遍历树, 3. 层次遍历树, 4. 树的高, 5. 树的度, 6. 转换二叉树的高, 7. 度数1结点数量, 8. 树的宽度, 9. 非递归后根遍历. The traversal sequences are: 树的后根序遍历: E F B C G H D A, 非递归后根遍历: E F B C G H D A.

Figure 5 (Example 2) shows a tree with root A and children B, C, D, E, F, G, H, I. The statistics are: 1. 先根序遍历树, 2. 后根序遍历树, 3. 层次遍历树, 4. 树的高, 5. 树的度, 6. 转换二叉树的高, 7. 度数1结点数量, 8. 树的宽度, 9. 非递归后根遍历. The traversal sequences are: 树的后根序遍历: E F B G C H I D A, 非递归后根遍历: E F B G C H I D A.

Figure 6 (Example 3) shows a tree with root A and children B, C, D, E, F, G, H, I. The statistics are: 1. 先根序遍历树, 2. 后根序遍历树, 3. 层次遍历树, 4. 树的高, 5. 树的度, 6. 转换二叉树的高, 7. 度数1结点数量, 8. 树的宽度, 9. 非递归后根遍历. The traversal sequences are: 树的后根序遍历: F G B C H I D E A, 非递归后根遍历: F G B C H I D E A.

图 4: 样例一

图 5: 样例二

图 6: 样例三