实验5 树与森林实验

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 张渊 | 学号 | 2022210000 | 专业班级 | 计科22-0 |
| 指导教师 | 张先宜 | 实验时间 | 2023.6.11 | 实验地点 |  |

**5.1实验目的**

(1)掌握树和森林的孩子兄弟链表（二叉链表）表示方法。

(2)掌握树和二叉树的结构及算法之间的对应关系。

(3)掌握树的两种遍历算法及其应用。

5.2 **实验要求**

1. 结构定义和算法实现放入库文件，如“treeChildSibling.h”中；
2. 数和森林的测试数据用文本文件方式给出，例如测试数据名为tree10.tre的树或森林，可参考发来的树和森林形状和参考存储文件；
3. 数和森林创建方法可自行选择；
4. 可多次连续测试。

**5.3实验任务**

设计、实现算法求解下列问题：

1. 按先序、后序、层次遍历森林。

实验测试数据基本要求：

第一组数据： tree11.tre

第二组数据： f20.tre

1. 求森林的高度。

实验测试数据基本要求：

第一组数据： tree11.tre

第二组数据： f20.tre

1. 求森林叶子结点数。

实验测试数据基本要求：

第一组数据： tree11.tre

第二组数据： f20.tre

1. 求森林的度。

实验测试数据基本要求：

第一组数据： tree11.tre

第二组数据： f20.tre

1. 先序输出结点值及其层次号。

例对图7-1所示森林，输出为：(A,1) (B,2) (E,3) (K,4) (F,3) (G,3) (C,2) (H,3) (I,3) (D,2) (J,3) (L,1) (M,2) (N,2) (O,1) (P,2)

实验测试数据基本要求：

第一组数据： tree11.tre

第二组数据： f20.tre

1. 输出广义表表示的树。

例对图7-1所示森林，输出为：A( B(E(K),F,G),C(H,I),D(J)), L(M,N), O(P) )

实验测试数据基本要求：

第一组数据： tree11.tre

第二组数据： f20.tre

图7-1 一个森林示意图

A

B

D

F

I

E

G

J

K

H

C

N

M

L

O

P

**5.4数据结构设计**

typedef struct pNode

{

elementType data; //结点数据域

int parent; //父结点指针（下标）

}PTNode;

typedef struct csNode

{

elementType data;

struct csNode\* firstChild, \* nextSibling

;

}csNode, \* csTree;

**5.5算法设计**

从文件创建双亲表示的森林；从双亲表示的森林创建二叉链表表示的森林：

创建二叉链表表示的树（森林）分为2个步骤，第一步：读取文本文件，创建双亲表示的树（森林）；第二步：从双亲表示转换为二叉链表表示的树（森林）。

考虑到文件内容有点不一样。文件扩展名没有限制，程序只根据标识“Tree or Forest”判定是否二叉树数据文件。文件中可以有空行。文件可以有注释行，注释行必须以“//”开始，且注释行必须是单独的行，不能混杂在标识行、数据行中。因此读文件的时候需要自定义一个去除空格的函数，然后再一个循环中，判断，去掉空格的第一个和第二个数据是’/n’和’//’的时候跳过此次循环，当不为空格或者注释的时候，说明空格注释已经遍历过了，到了标识的时候。此时再进行判断是否含有BinaryTree，若有则正确，否则文件格式错误。之后在每次循环的开始首先调用去空格的函数，然后再判断去掉空格的第一个和第二个数据是’/n’和’//’的时候跳过此次循环，若不是则进行读数，将文件数据读入到以双亲存储的森林中。最后关闭文件。之后再利用递归函数将以双亲存储的森林写入以孩子兄弟储存的森林中。结束。

1 【算法思想】 先序遍历，后序遍历：递归函数遍历子树森林，在递归函数遍历兄弟森林。跟二叉树的先序后序遍历保持一致。

层次遍历森林算法思想：最外层循环处理森林中每一棵树，在二叉链表结构中，森林中的树通过根结点 T 的 nextSibling 指针找到。利用队列，下面算法定义队列元素包含 2 个分量，一 个是结点指针，另一个是结点层次。对每棵树，先将根结点入队。队列不空循环处理，队头 出队，找到队头的所有孩子结点入队，方法是通过 firstChild 找到第一个孩子，再通过此孩 子结点的 nextSibling 找到所有兄弟。孩子结点的层次在父结点（出队结点）层次上加 1。

为了操作方便，此次实验队列用数组模拟。

【算法描述】

void onexian(csTree s)

{

if (s)

{

cout << s->data << " ";

onexian(s->firstChild);

onexian(s->nextSibling);

}

}

void onehou(csTree s)

{

if (s)

{

onehou(s->firstChild);

cout << s->data << " ";

onehou(s->nextSibling);

}

}

void onecengci(csTree s)

{

//模拟队列

int front = 0, rear = 0;

csTree a[100];

csTree temp;

for (s; s; s = s->nextSibling)

{

a[++rear] = s;

}

while (front != rear)

{

temp = a[++front];

cout << temp->data << " ";

for (temp = temp->firstChild; temp; temp = temp->nextSibling)

{

a[++rear] = temp;

}

}

}

2 【算法思想】森林的高度算法思想：树（森林）以孩子兄弟链表表示。如果树不空，递归求第一个孩子结点为根的子树的高度，然后顺着此结点的右分支（兄弟结点），求每一棵子树的高度，取这些子树高度的最大值count，则树（森林）的高度即为count+1。

【算法描述】

void two(csTree s, int& n, int count)

{

if (s)

{

if (n < count)

{

n = count;

}

two(s->firstChild, n, count + 1);

two(s->nextSibling, n, count);

}

}

3 【算法思想】求森林叶子结点数：改造前序遍历算法，增设一个引用参数n为计数器，计算结点数，每次访问根节点时，如果T->firstChild==NULL时，n++。

【算法描述】

void three(csTree s, int& n)

{

if (s)

{

if (s->firstChild == NULL)

{

n++;

}

three(s->firstChild, n);

three(s->nextSibling, n);

}

}

4 【算法思想】求森林的度：改造前序遍历算法，增设一个引用参数count为计数器，计算结点数，每次访问根节点时，如果T->firstChild==NULL时，count++。

【算法描述】

void four(csTree s, int& n, int count)

{

if (s)

{

if (n < count)

{

n = count;

}

four(s->firstChild, n, 1);

four(s->nextSibling, n, count + 1);

}

}

5 【算法思想】先序输出结点值及其层次号：改造先序遍历算法，增加一个结点层次参数，当递归遍历 T->firstChild 子树时，结点层次为 T 的层次加 1；递归遍历 T->nextSibling 子树时，结点层次与T相同。输出形式：(结点值，层次)。初始调用 outPreOrder(T,1)。

【算法描述】

void five(csTree s, int count)

{

if (s)

{

cout << "( " << s->data << ", " << count << " )" << " ";

five(s->firstChild, count + 1);

five(s->nextSibling, count);

}

}

6 【算法思想】输出广义表表示的树：改造先序遍历算法完成，访问结点为打印结点值，判断如果 firstChild 不为空，说明结点有子树递归遍历，但在遍历之前先要打印“(”；判断 nextSibling 是否为空， 不空时，要打印“,”，为空时打印“)”表示子树遍历结束。这个算法有个缺点：根结点层 次最后会多出一个“)”，需要手工在调用函数前打印一个“(”，表示整个树（森林）。

【算法描述】

void six1(csTree s, csTree& head)

{

if (s)

{

cout << s->data;

if (s->firstChild)

{

cout << "(";

}

six1(s->firstChild, head);

if (s->nextSibling)

{

cout << ",";

}

if (s->nextSibling == NULL)

{

cout << ")";

}

if (s != head)

six1(s->nextSibling, head);

}

}

void six(csTree s)

{

for (; s; s = s->nextSibling)

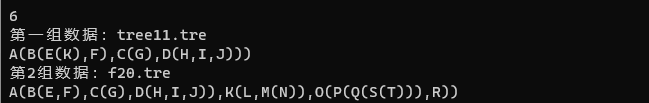
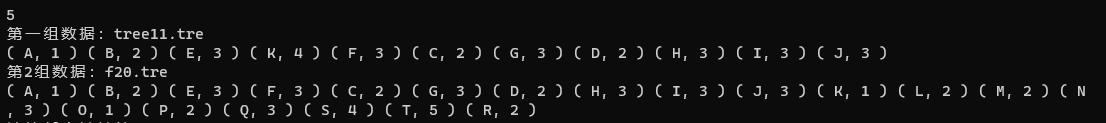
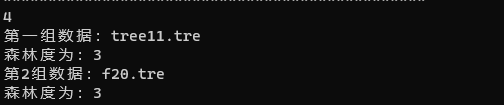
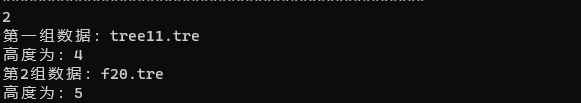
{

six1(s, s);

}

}

**5.6运行和测试**



**5.7总结、心得和建议**

在本次数据结构实验中，我深入学习了森林数据结构，并着重理解了森林的孩子兄弟表示方法。在森林中，一个节点可以有零个或多个子节点，而这些子节点之间没有任何关系。显然，节点之间的联系可以通过父节点来定义。因此，我们可以使用各种数据结构来表示森林的孩子结点。对于森林的孩子兄弟表示法来说，它更加符合实际场景中的需求，将若干棵子树串在一起，避免了通过指针指向进行访问的方式，大幅提升了效率。

我采用了链式存储表示法，将每个节点看作一个结构体，其中包含该节点的数据信息和指向其第一个孩子结点的指针。每个孩子结点在该节点的链表中通过指针串联起来，使得每个节点的孩子结点可以方便地插入和删除。同时，孩子结点在链表中的顺序也很重要，因为它们决定了该节点的访问方式。在我的代码实现中，我使用了递归的方式来访问每个节点及其孩子节点。

通过本次实验，我进一步掌握了数据结构的基本原理和实现方法，并深刻认识到合适的数据结构可以大大提高程序的效率和运行速度。同时，对森林的孩子兄弟表示方法有了更深刻的理解，也为我今后在算法设计和数据结构实现中提供了重要的思路和方法。

**[8.附录]**

（源代码清单。纸质报告不做要求。电子版报告，可直接附源文件，删除编译生成的所有文件）

****