实验4二叉树实验

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |  |

4.1**实验目的**

1. 掌握二叉树的动态链表存储结构及表示。
2. 掌握二叉树的三种遍历算法（递归和非递归两类）。
3. 运用二叉树三种遍历的方法求解有关问题。

4.2 **实验要求**

1. 结构定义和算法实现放入库文件，如“BiTree.h”中；
2. 二叉树的测试数据用文本文件方式给出，例如测试数据名为bt151.btr的二叉树，可参考发来的二叉树形状和参考存储文件；
3. 二叉树创建方法可自行选择；
4. 可多次连续测试。

4.3 **实验任务**

设计二叉树的二叉链表存储结构，编写算法实现下列问题的求解。

1. 设计算法按中序次序输出二叉树中各结点的值及其所对应的层次数。

实验测试数据基本要求：

第一组数据： bt151.btr

第二组数据： bt21.btr

1. 求二叉树的叶子结点数和1度结点数。

实验测试数据基本要求：

第一组数据： bt261.btr

第二组数据： bt21.btr

1. 键盘输入一个元素x，求其父节点、兄弟结点、子结点的值，不存在时给出相应提示信息。对兄弟结点和孩子结点，存在时要明确指出是左兄弟、左孩子、右兄弟或右孩子。

实验测试数据基本要求：

第一组数据： bt31.btr

第二组数据： bt21.btr

1. 键盘输入一个元素x，求其在树中的层次。不存在时给出相应提示信息。

实验测试数据基本要求：

第一组数据： bt262.btr

第二组数据： bt21.btr

1. 将按顺序方式存储在数组中的二叉树转换为二叉链表形式。（数组中要扩展为完全二叉树）。

实验测试数据基本要求：

第一组数据： bt8.btr

第二组数据： bt14.btr

1. 输出二叉树从每个叶子结点到根结点的路径（经历的结点）。

实验测试数据基本要求：

第一组数据： bt261.btr

第二组数据： bt21.btr

1. 对二叉链表表示的二叉树，按从上到下，从左到右打印结点值，即按层次遍历序打印。（提示：需要使用队列）

实验测试数据基本要求：

第一组数据： bt261.btr

第二组数据： bt21.btr

**4.4数据结构设计**

typedef struct BiNode {

char data;

struct BiNode\* lChild, \* rChild;

}BiNode;

**4.5算法设计**

（除书上给出的基本运算（这部分不必给出设计思想），其它实验内容要给出算法设计思想）

从文件中读数据到二叉树中：

结点数据读入数组函数：因为是按先序次序递归创建二叉树，在创建过程中又要记录读取数据的行数，所以直接从文件中读取数据创建时处理不方便，因此可以先把文件中的数据读取到一个二维数组中，再从数组中读取数据来创建二叉树。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“BinaryTree”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。之后再一个字符的判断，是否第一个是’/’若是则说明是注释，跳到下一行。若不是则把一行的三个数据读入。最后关闭文件。在将数组读入二叉树中，结束。

1 【算法思想】改造先序遍历算法，增加一个结点层次参数，当递归遍历 T->lChild子树时，结点层次为 T 的层次加 1；之后递归遍历 T-> rChild子树时，结点层次为T + 1。输出形式：(结点值，层次)。初始调用 outMidOrder(T,1)。

【算法描述】

void outMidOrder(BiNode\* T, int level)

{

if (T) {

outMidOrder(T->lChild, level + 1); //T 的孩子层次为 T 的层次加 1

cout << "(" << T->data << "," << level << ") "; //访问根结点

outMidOrder(T->rChild, level + 1); //T 的兄弟层次与 T 相同

}

}

2 【算法思想】采用中序遍历，命名两个整型变量储存叶子节点数和1度节点数，首先递归调用左子树，当T->lChild == NULL && T->rChild == NULL 时ye++;当(T->lChild != NULL && T->rChild == NULL) || (T->lChild == NULL && T->rChild != NULL)时Du++，然后再递归调用右子树。

【算法描述】

void YeAnd1Du(BiNode\* T, int& Ye, int& Du) {

if (T) {

YeAnd1Du(T->lChild, Ye, Du);

if (T->lChild == NULL && T->rChild == NULL) {

Ye++;

}

if ((T->lChild != NULL && T->rChild == NULL) || (T->lChild == NULL && T->rChild != NULL)) {

Du++;

}

YeAnd1Du(T->rChild, Ye, Du);

}

}

3 【算法思想】采用先序遍历，三个形参，数结点，输入的变量，一个bool类型的数据用来标志是否查找到。首先是T->lChild != NULL && T->lChild->data == x，说明最多只有右兄弟，一种情况，再仔细判断是否有双亲结点，左孩子右孩子，最后把f标志为true。之后当T->rChild != NULL && T->rChild->data == x时说明最多只有左兄弟，一种情况，再仔细判断是否有双亲结点，左孩子右孩子，最后把f标志为true。做后是T->data == x，这说明是根节点，没有双亲结点和兄弟节点，只需判断是否有孩子节点即可。之后再分别递归调用左子树右子树。

【算法描述】

void find(BiNode\* T, char x, bool& f) {

if (T) {

if (T->lChild != NULL && T->lChild->data == x) {

cout << "x的双亲结点的值为：" << T->data << endl;

if (T->rChild != NULL) {

cout << "x的兄弟结点为右兄弟，其值为：" << T->rChild->data << endl;

}

if (T->lChild->lChild != NULL) {

cout << "x的左孩子结点的值为：" << T->lChild->lChild->data << endl;

}

else {

cout << "没有左孩子" << endl;

}

if (T->lChild->rChild != NULL) {

cout << "x的右孩子结点的值为：" << T->lChild->rChild->data << endl;

}

else {

cout << "没有右孩子" << endl;

}

f = true;

return;

}

else if (T->rChild != NULL && T->rChild->data == x) {

cout << "x的双亲结点的值为：" << T->data << endl;

if (T->lChild != NULL) {

cout << "x的兄弟结点为左兄弟，其值为：" << T->lChild->data << endl;

}

if (T->rChild->lChild != NULL) {

cout << "x的左孩子结点的值为：" << T->rChild->lChild->data << endl;

}

else {

cout << "没有左孩子" << endl;

}

if (T->rChild->rChild != NULL) {

cout << "x的右孩子结点的值为：" << T->rChild->rChild->data << endl;

}

else {

cout << "没有右孩子" << endl;

}

f = true;

return;

}

else if (T->data == x) {

cout << "x没有双亲结点" << endl;

cout << "x没有兄弟结点" << endl;

if (T->lChild != NULL) {

cout << "x的左孩子结点的值为：" << T->lChild->data << endl;

}

else {

cout << "没有左孩子" << endl;

}

if (T->rChild != NULL) {

cout << "x的右孩子结点的值为：" << T->rChild->data << endl;

}

else {

cout << "没有右孩子" << endl;

}

f = true;

return;

}

find(T->lChild, x, f);

find(T->rChild, x, f);

}

}

4 【算法思想】采用先序遍历，四个形参，数结点，高度， 输入的变量，一个bool类型的数据用来标志是否查找到。首先当树不空的时候，每次递归调用左子树或者右子树，都需要对高度进行加一，之后再每次递归调用之前判断是否查找到输入的元素，若查找到，进行输出当前传入形参的高度，再将标志标为true。最后递归调用左子树右子树。

【算法描述】

void findH(BiNode\* T, int level, char x, bool& f)

{

if (T) {

if (T->data == x) {

cout << T->data << "的层次为：" << level << endl; //访问根结点

f = true;

}

findH(T->lChild, level + 1, x, f); //T 的孩子层次为 T 的层次加 1

findH(T->rChild, level + 1, x, f);

}

}

5 【算法思想】造先序遍历实现。算法包括两个整型参数 i 和 num。i 为结点在完全二叉树上的编号，从 1 开始；num 为二叉树上最后有效结点编号。对每次i <= num && A[i] != '0'时，进行new 一个树节点，把数组数据传进去，再把该树的左子树右子树，全部赋值为NULL。最后在递归调用左右子树。注意，在顺序储存中，左子树和父节点的关系为2倍，右子树为父节点的二倍加1，当数组0下标为空时刚好下标与之对应。所以当调用左子树的时候，记得为2 \* i，调用右子树时为2 \* i + 1。

【算法描述】

void ArrayToBiTree(BiNode\*& T, char A[], int i, int num)

{

//i 为当前结点编号，从 1 开始

//num 为最后有效结点编号

if (i <= num && A[i] != '0') {

T = new BiNode;

T->data = A[i];

T->lChild = NULL;

T->rChild = NULL;

ArrayToBiTree(T->lChild, A, 2 \* i, num);

ArrayToBiTree(T->rChild, A, 2 \* i + 1, num);

}

}

6 【算法思想】从根结点开始遍历，用栈存储路径上每个结点的数值，有左子树往左走，没有左子树再往右子树走，如果都没有左右子树说明到了叶子结点了，那么路径就完整了，就可以打印栈的内容了，

1.当遍历到了叶子结点d，此时这条路径便已经完整了，而结点b还有另一个右孩子结点e，那么遍历到d的路径和遍历到e的路径只差了最后一个结点，那么只需要把结点d出栈，再把结点e入栈，遍历到e的路径就完整了吗，所以每次遍历到叶子结点将栈打印完后，只需将该叶子结点出栈。

2.当此时已经遍历完结点e并出栈了，此时b的左右孩子结点已经都遍历完了，不再被需要了，此时若想若遍历到c结点，只需要将结点b出栈，那么a的左子树已经遍历完，此时再往右子树走，也就是c点走，这样就能获取遍历到c结点的路径。

此算法实际操作时，为了操作方便，用数组来模拟栈。

【算法描述】

void Route(BiNode\* T, char\* path, int pathlen) { //输出二叉树中从每个叶子结点到根结点的路径

if (T) {

if (T->lChild == NULL && T->rChild == NULL) {

cout << "叶子结点 " << T->data << " 的路径为：";

for (int i = pathlen - 1; i >= 0; i--) {

cout << path[i] << " ";

}

cout << endl;

}

else {

path[pathlen++] = T->data;

Route(T->lChild, path, pathlen);

Route(T->rChild, path, pathlen);

}

}

}

7 【算法思想】二叉树的层次遍历，我们定义一个[队列](https://so.csdn.net/so/search?q=%E9%98%9F%E5%88%97&spm=1001.2101.3001.7020" \t "_blank)，先将根结点入队；当前结点是队头结点，将其出队并访问；若当前结点的左结点不为空将左结点入队；若当前结点的右结点不为空将其入队即可,最后队列为空即遍历完成 。

此算法实际操作时，为了操作方便，用数组来模拟队列。

【算法描述】

void LevelOrder(BiNode\* T) // 二叉树的层序遍历

{

BiNode\* Q[100]; // 数组模拟队列

int front = 0;

int rear = 0;

BiNode\* p;

Q[++rear] = T; // 根结点入队

while (front != rear) { // 若队列不为空

// 队头结点出队，并访问出队结点

p = Q[++front];

cout << p->data << " ";

// 出队结点的非空左右孩子依次入队

if (p->lChild != NULL) {

Q[++rear] = p->lChild;

}

if (p->rChild != NULL) {

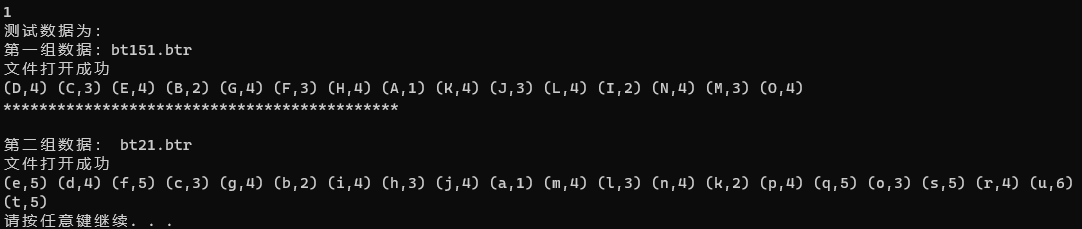
Q[++rear] = p->rChild;

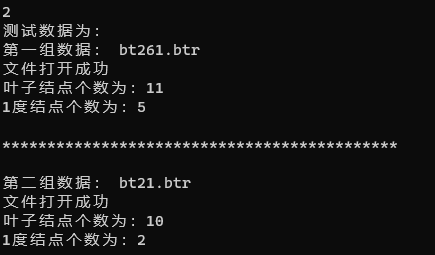
}

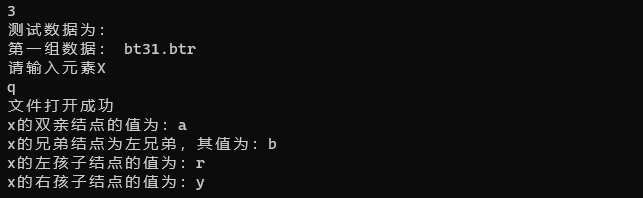
}

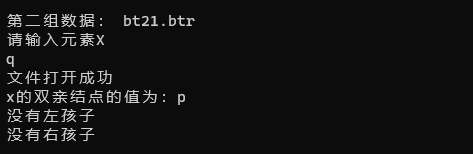
}

**4.6运行和测试**





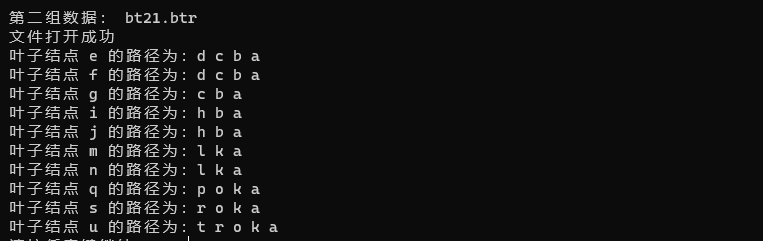














**4.7总结、心得和建议**

通过这个实验大概可以了解到，二叉树有两种储存方式：

一、顺序存储：

优点：读取某个指定的节点的时候效率比较高O(1)。  
缺点：会浪费空间(在非完全二叉树的时候)。  
二、链式存储：  
优点： 相对二叉树比较大的时候浪费空间较少

二叉树的顺序存储，寻找后代节点和祖先节点都非常方便，但对于普通的二叉树，顺序存储浪费大量的存储空间，同样也不利于节点的插入和删除。因此顺序存储一般用于存储完全二叉树。  
缺点： 读取某个指定节点的时候效率偏低O(nlogn)。  
链式存储相对顺序存储节省存储空间，插入删除节点时只需修改指针，但寻找指定节点时很不方便。不过普通的二叉树一般是用链式存储结构。

在本实验设计算法实现二叉树层次遍历，首先我觉得毫无思路，想了很久也还是一头雾水，但是后来在网上查阅相关资料，和同学之间进行交流，最后发现用队列可以轻松解决，我们定义一个[队列](https://so.csdn.net/so/search?q=%E9%98%9F%E5%88%97&spm=1001.2101.3001.7020" \t "_blank)，先将根结点入队；当前结点是队头结点，将其出队并访问；若当前结点的左结点不为空将左结点入队；若当前结点的右结点不为空将其入队即可,最后队列为空即遍历完成。由此可见，学了后面的不能忘了前面的，要多多复习，争取融会贯通。二叉树算法大量用到了递归，在课上感觉听起来简单，但是自己来做这次实验还是有不小的难度的，递归确实挺难理解的。

经过这次实验我对二叉树有了更加深刻的理解，了解到二叉树的大多算法都是改造遍历算法就可以实现了，熟练过后也觉得二叉树不是难么难了。总体来说本次实验花费时间较长，虽然最终还是没有取得非常好的结果但是对自己的能力有了更清楚的认识，也加深了对递归和二叉树知识的理解。希望在以后的学习中不断反思，提高能力。

**[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）

