数据结构实验报告

实验成绩：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 |  | 学号 |  | 专业班级 |  |
| 指导教师 | 张先宜 | 实验时间 |  | 实验地点 |  |

**实验名称： 二叉树实验**

1. **实验目标**
2. 掌握二叉树的动态链表存储结构及表示。
3. 掌握二叉树的三种遍历算法（递归和非递归两类）。
4. 运用二叉树三种遍历的方法求解有关问题。
5. **实验要求**
6. 结构定义和算法实现放入库文件，如“BiTree.h”中；
7. 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；
8. 二叉树创建方法可自行选择；

可多次连续测试

1. **实验内容**

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： bt26.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

1. **数据结构设计**

（所有存储结构的封装描述）

typedef struct LBNode

{

char data;

struct LBNode \*lchild,\*rchild;

}binode;

1. **算法设计**
2. 求节点数：利用递归遍历二叉树求得结点数

void nodehigh(BiNode\* T, int high) //1.输出二叉树中各结点的值及其所对应的层次数

{

if (T!=NULL) {

nodehigh(T->left,high+1);

cout << T->data<<" "<<high<<endl;

nodehigh(T->right,high+1);

}

}

2）求叶子节点数：利用递归，当节点无子节点时，返回+ 1

求度为1 的节点数：利用递归，当节点有1个子节点时，返回 + 1

int leaveNum(BiNode\* pBT) //2.求叶子结点数

{

if (pBT == NULL)

return 0;

if (pBT->left == NULL && pBT->right == NULL)

return 1;

return leaveNum(pBT->left) + leaveNum(pBT->right);

}

int fullNodeNum(BiNode\* pBT) //2.求度为1的结点数

{

if (pBT == NULL)

return 0;

if (pBT->left != NULL && pBT->right == NULL) {

return fullNodeNum(pBT->left) + 1;

}

else if (pBT->right != NULL && pBT->left == NULL) {

return fullNodeNum(pBT->right) + 1;

}

return fullNodeNum(pBT->left) + fullNodeNum(pBT->right);

}

3）求其父节点、兄弟结点、子结点的值：首先遍历树找到节点，返回节点指针，然后再对指针进行相关操作

BiNode\* Locate(BiNode\* pBT, char x) //找到x位置，方便对其父节点、兄弟结点、子结点进行分析

{

if (pBT == NULL)

return NULL;

if (pBT->data == x)

return pBT;

if (Locate(pBT->left, x) != NULL)

return Locate(pBT->left, x);

else

return Locate(pBT->right, x);

}

void getRelatedNode(BiNode \*root,BiNode\* pFound) {

if (pFound == NULL) {

cout << "该结点不存在!" << endl;

return;

}

if (Parent(root, pFound) != NULL) {

cout << "父节点值为:" << Parent(root, pFound)->data << endl;

if (Parent(root, pFound)->left == pFound) {

cout << "兄弟结点为右兄弟,其值为:" << Parent(root, pFound)->right->data << endl;

}

else if (Parent(root, pFound)->right == pFound) {

cout << "兄弟结点为左兄弟,其值为:" << Parent(root, pFound)->left->data << endl;

}

else cout << "不存在兄弟结点" << endl;

}

else {

cout << "不存在父节点!" << endl;

cout << "不存在兄弟结点!" << endl;

}

if (pFound->left != NULL || pFound->right != NULL) {

if (pFound->left != NULL) {

cout << "左孩子值为:" << pFound->left->data << endl;

}

if (pFound->right != NULL) {

cout << "右孩子值为:" << pFound->right->data << endl;

}

}

else cout << "不存在孩子结点!" << endl;

}

4）求层次数:遍历并记录层次数直到找到x，返回层次数,不存在时提示

void foundNode(BiNode\* pBT, int\* nRow, char x)

{

(\*nRow)++;

int nRowNext = \*nRow;

if (pBT != NULL)

{

foundNode(pBT->left, nRow, x);

\*nRow = nRowNext;

if (pBT->data == x)

cout << "层次数:" << nRowNext << endl;

foundNode(pBT->right, nRow, x);

}

return;

}

5）二叉树转换为二叉链表:先算出树的高度，利用高度算出完全二叉树总节点数，向数组中写入对应数量的‘\*’，然后再将二叉树按顺序写入数组

BiNode\* arrayToTree(char\* T, int sum, int i) //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

{

BiNode\* pBT;

if (i <= sum)

{

pBT = new BiNode;

pBT->data = T[i];

pBT->left = arrayToTree(T, sum, 2 \* i);

pBT->right = arrayToTree(T, sum, 2 \* i + 1);

}

else

{

return NULL;

}

return pBT;

}

6）输出二叉树从每个叶子结点到根结点的路径:先遍历求出叶子结点,叶子结点再遍历求父节点直到根节点为止.

void TreeRoad(BiNode\* pBT) {

static BiNode\* root = pBT;

if (pBT == NULL)

return;

if (pBT->left != NULL || pBT->right != NULL) {

TreeRoad(pBT->left);

TreeRoad(pBT->right);

}

else while (pBT != NULL) {

cout << pBT->data;

pBT = Parent(root,pBT);

}

cout << endl;

}

void treeToArray(BiNode\* pBT, char T[], int i) //将数存储在数组中

{

if (pBT != NULL)

{

T[i] = pBT->data;

treeToArray(pBT->left, T, 2 \* i);

treeToArray(pBT->right, T, 2 \* i + 1);

}

}

7）按层次遍历序打印: 设计一个队列,出队保存队头并访问, 将出队结点入队.循环至队列为空.

void BinaryTreeLevelOrder(BiNode\* root)

{

queue<BiNode\*> q;

if (root == NULL)

{

return;

}

q.push(root);

while (!q.empty())

{

BiNode\* front = q.front(); //出队保存队头并访问

cout << front->data;

q.pop();

if (front->left) { //将出队结点的左子树根入队

q.push(front->left);

}

if (front->right) { //将出队结点的右子树根入队

q.push(front->right);

}

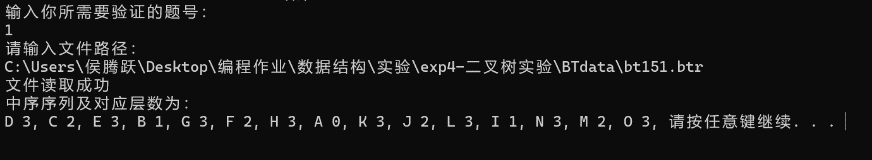
}

}

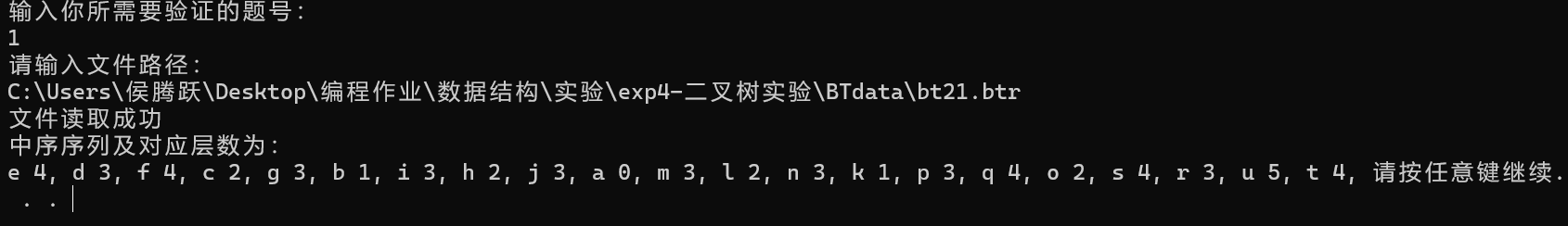
1. **运行和测试**

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。

第一组数据： bt151.btr

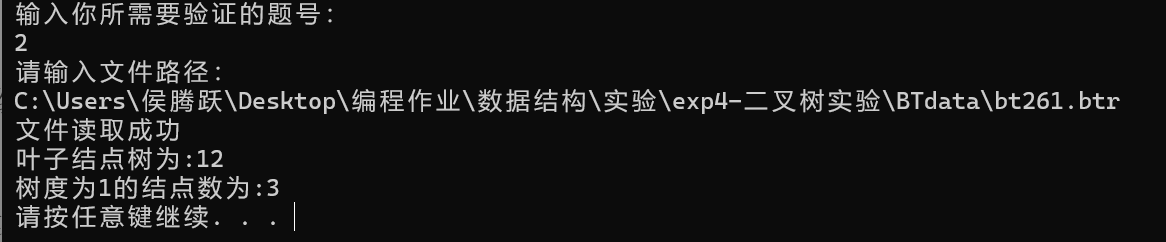


第二组数据： bt21.btr

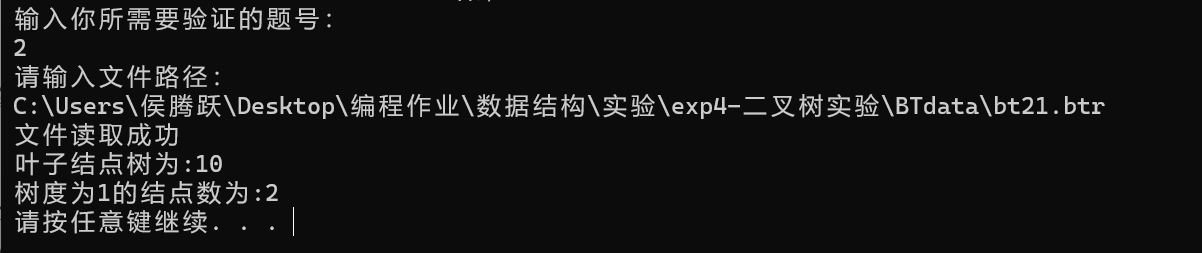


2）求二叉树的叶子结点数和1度结点数。

第一组数据： bt261.btr

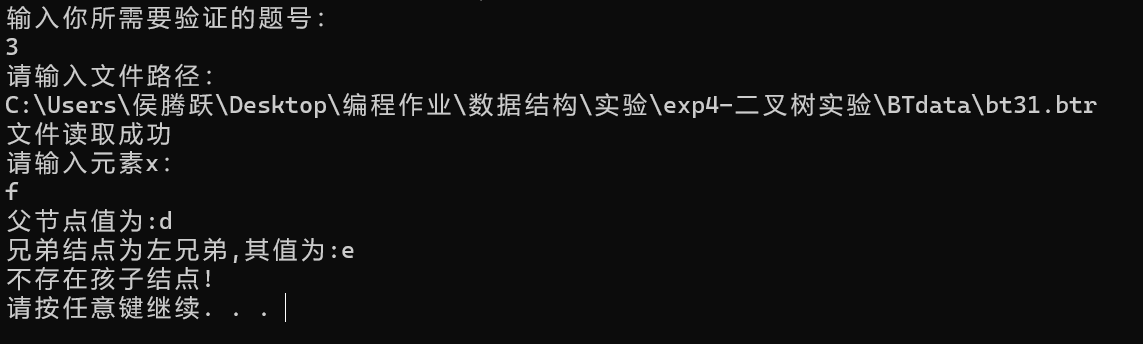


第二组数据： bt21.btr

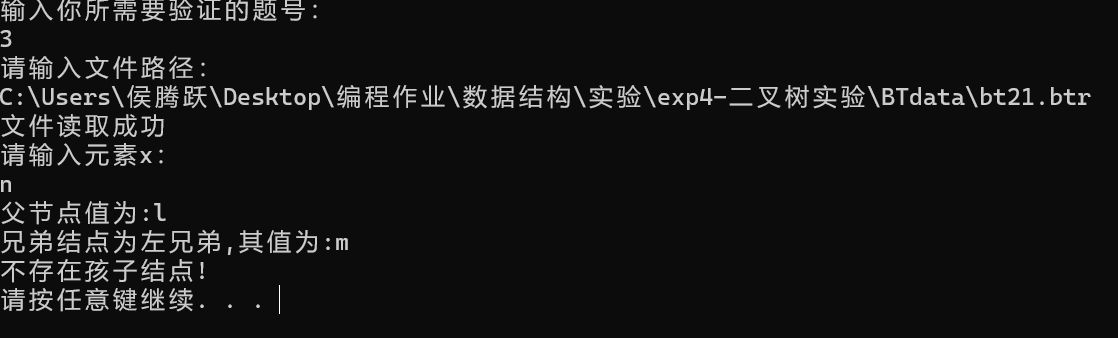


3）键盘输入一个元素x，求其父节点、兄弟结点、子结点的值，不存在时给出相应提示信息。对兄弟结点和孩子结点，存在时要明确指出是左兄弟、左孩子、右兄弟或右孩子。

第一组数据： bt31.btr

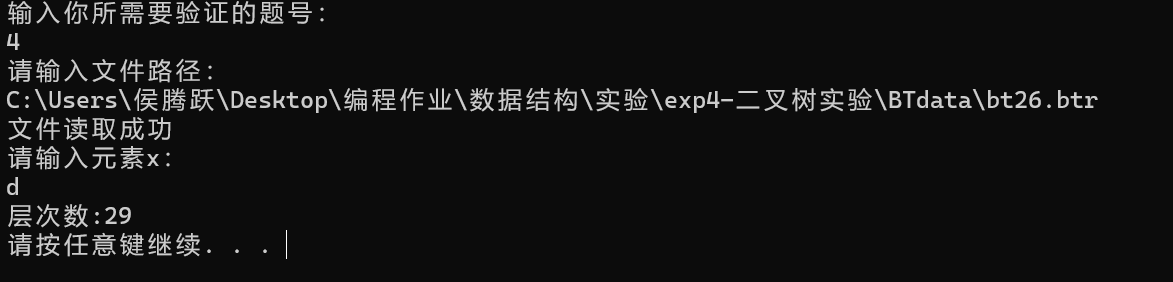


第二组数据： bt21.btr

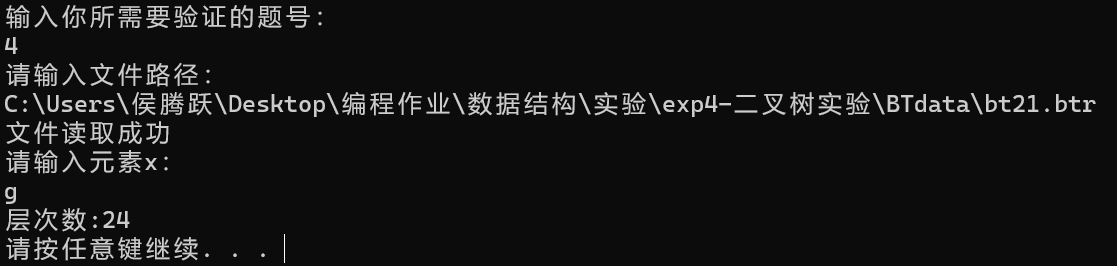


4）键盘输入一个元素x，求其在树中的层次。不存在时给出相应提示信息。

第一组数据： bt26.btr

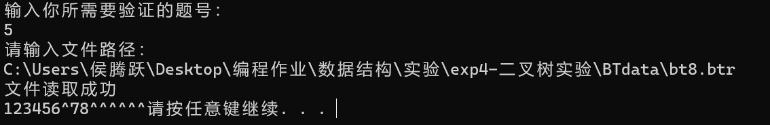


第二组数据： bt21.btr

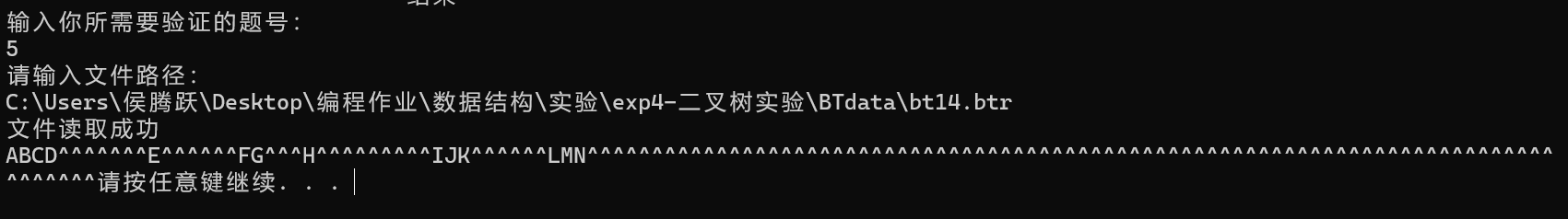


5）将按顺序方式存储在数组中的二叉树转换为二叉链表形式。。

第一组数据： bt8.btr



第二组数据： bt14.btr



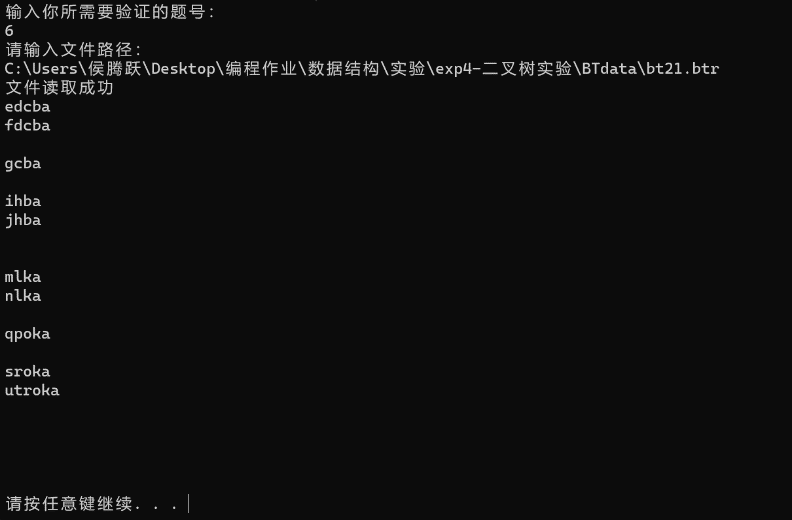
6）输出二叉树从每个叶子结点到根结点的路径（经历的结点）。

实验测试数据基本要求：

第一组数据： bt261.btr

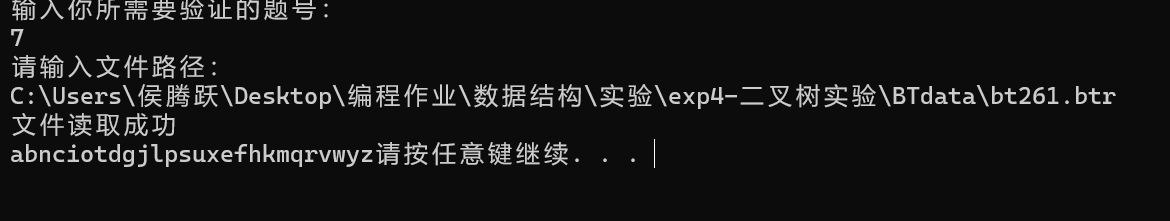


第二组数据： bt21.btr

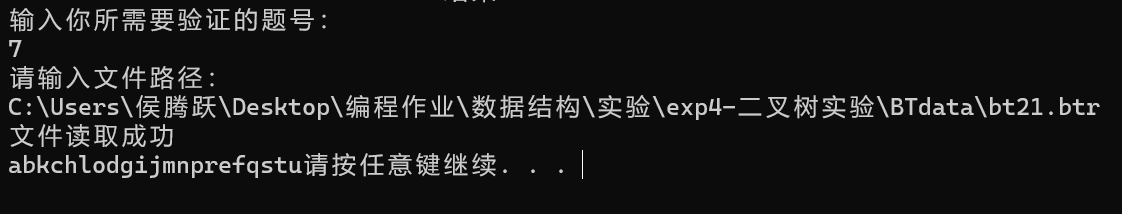


7）对二叉链表表示的二叉树，按从上到下，从左到右打印结点值，即按层次遍历序打印。

第一组数据： bt261.btr



第二组数据： bt21.btr



1. **总结、心得和建议**

经过这次实验，掌握了二叉树的动态链表存储结构及表示，学会了运用二叉树三种遍历的方法求解有关问题，我深刻理解了二叉树的储存以及操作方式，也加深我对递归算法的编写的理解，我学会了如何高效的使用递归算法，我深刻体会到二叉树这种非线性数据结构的巨大优势，并学会了如何利用他的优势来编写算法。