## 3基于二叉链表的二叉树实现

## 3.1 问题描述

通过实验达到⑴加深对二叉树的概念、基本运算的理解；⑵熟练掌握二叉树的逻辑结构与物理结构的关系；⑶以二叉链表作为物理结构，熟练掌握二叉树基本运算的实现。

## 3.1.1 实现基本操作

依据最小完备性和常用性相结合的原则，以函数形式定义了二叉树的创建二叉树、销毁二叉树、清空二叉树、判定空二叉树和求二叉树深度等14种基本运算。

## 3.1.2 实现数据存储与读取

设计函数使得能够从文件中读取或者存储顺序存储二叉树

## 3.1.3 实现多个二叉树管理

能管理多个二叉树。

## 3.1.4 实现树的显示

能用图形方式展示树的结构

## 3.2 系统设计

## 3.2.1 程序总体框架

该系统包括用户交互模块，操作管理模块，多二叉树管理模块，文件I/O模块，单个二叉树操作模块。其中文件I/O模块包括写入与读取功能；单个二叉树操作模块包括创建二叉树，销毁二叉树，清空二叉树，判定空二叉树，求二叉树深度，查找节点，节点赋值，获得兄弟节点，插入节点，删除节点，前序遍历，中序遍历（非递归），后序遍历，按层遍历，图形展示，如图2-1所示。

操作管理模块

多二叉树管理模块

用户交互模块

单个二叉树操作模块

文件I/O模块

二叉树元素定位

二叉树初始化

……

二叉树销毁

二叉树清空

图3-1系统总体框架

## 3.2.2 系统环境

系统：macOS Catalina

IDE：clion

## 3.2.3 自定义部分

using namespace std;

#include <stdio.h>

#include <stdlib.h>

#include <stack>

#include <vector>

#include <queue>

/\*\*\*初始化定义\*\*\*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

//数据为0表示空节点

#define EMPTY 0

//默认队列等的大小

#define QUEUESIZE 200

//多二叉树管理数组的大小

#define LLIST\_SIZE 10

typedef int status;

typedef int ElemType; //数据元素类型定义

//文件绝对路径

char FILE\_NAME[100] =

"/Users/akunda/Downloads/DS+/DS3/output.txt";

//前序遍历带空子树的数组，用来初始化树

ElemType init\_btree\_array[] = {1, 2, 4, 0, 5, 0, 0, 0, 3, 6,

0, 7, 0, 0, 8, 0, 0};

//元素结构

typedef struct elem {

int id;

ElemType data;

} Elem;

//树结构

typedef struct BTreeNode {

Elem elem;

struct BTreeNode \*lchild;

struct BTreeNode \*rchild;

} BTreeNode, \*BTree;

## 3.3 系统实现

## 3.3.1 菜单栏

菜单栏操作步骤如图2-2所示：

开始

选择操作

选择二叉树

退出

单个二叉树操作模块

文件I/O模块

存储二叉树

读取二叉树

……

节点赋值

定位元素

初始化二叉树

图3-2菜单栏示意图

## 3.3.2 函数实现与分析

⑴创建二叉树：函数名称是CreateBiTree(T,definition)；初始条件是definition 给出二叉树T的定义，本实验给出的定义是二叉树的带空子树前序遍历的数组。

时间复杂度：O(n)

实现：分配头结点，递归造子树，调用了PreOrderCreateBiTree(BTree &root, ElemType array[], int &index\_array, int &index\_tree)。

流程图：

分配头结点存储空间

并初始化

开始

二叉树存在

是

否

返回error

递归前序遍历创建二叉树

返回OK

图3-3创建二叉树流程图

设置两个index分别服务树节点（id）和数组

是

否

对左右子树递归前序遍历创建二叉树

递归前序遍历创建二叉树

元素为空，说明节点不存在（空节点）

调整树的index（index\_tree--）

返回

图3-4递归创建二叉树函数流程图

⑵销毁二叉树：函数名称是DestroyBiTree(T)；初始条件是二叉树T已存在；操作结果是销毁二叉树T。

时间复杂度：O(n)

实现：递归销毁左右子树，最后销毁头结点。

⑶清空二叉树：函数名称是ClearBiTree (T)；初始条件是二叉树T存在；操作结果是将二叉树T清空。

时间复杂度：O(n)

实现：递归销毁头结点的左右子树。

⑷判定空二叉树：函数名称是BiTreeEmpty(T)；初始条件是二叉树T存在；操作结果是若T为空二叉树则返回TRUE，否则返回FALSE。

时间复杂度：O(1)

实现：如果头指针为空或者头指针的左子树为空（即内容树为空），则认为是空树返回TRUE，否则返回FALSE。

⑸求二叉树深度：函数名称是BiTreeDepth(T)；初始条件是二叉树T存在；操作结果是返回T的深度。

时间复杂度：O(n)

实现：调用了SearchDepth(BTree T)；SearchDepth函数返回左右子树的最大值再加1，递归求深度

⑹查找结点：函数名称是LocateNode(T,e)；初始条件是二叉树T已存在，e是和T中结点关键字类型相同的给定值；操作结果是返回查找到的结点指针，如无关键字为e的结点，返回NULL。

时间复杂度：O(n)

实现：递归遍历左右子树。

⑺结点赋值：函数名称是Assign(T,e,value)；初始条件是二叉树T已存在，e是和T中结点关键字类型相同的给定值；操作结果是关键字为e的结点赋值为value。

时间复杂度：O(n)

实现：先定位目标节点，调用LocateNode(T,e)，再赋值。

⑻获得兄弟结点：函数名称是GetSibling(T,e)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值；操作结果是返回关键字为e的结点的（左或右）兄弟结点指针。若关键字为e的结点无兄弟，则返回NULL。

时间复杂度：O(n)

实现：先调用函数LocateFatherNode(BTree T, int index)，递归查找父节点，然后再返回目标节点的兄弟节点。其中LocateFatherNode在判断是否为父节点时，要先判断左(右)子树是否存在，再判断左(右)节点id是否与给定id相等，如果直接判断子树id是否相等会有非法访问。

流程图：

否

开始

递归定位父节点

找到了父节点

返回目标节点的兄弟节点

返回NULL

是

图3-5获得兄弟节点流程图

⑼插入结点：函数名称是InsertNode(T,e,LR,c)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值，LR为0或1，c是待插入结点；操作结果是根据LR为0或者1，插入结点c到T中，作为关键字为e的结点的左或右孩子结点，结点e的原有左子树或右子树则为结点c的右子树。

时间复杂度：O(n)

实现：先定位目标节点，调用LocateNode(T,e)，再根据LR插入节点。

⑽删除结点：函数名称是DeleteNode(T,e)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值。操作结果是删除T中关键字为e的结点；同时，如果关键字为e的结点度为0，删除即可；如关键字为e的结点度为1，用关键字为e的结点孩子代替被删除的e位置；如关键字为e的结点度为2，用e的左孩子代替被删除的e位置，e的右子树作为e的左子树中最右结点的右子树。

时间复杂度：O(n)

流程图：

开始

递归定位父节点

找到了父节点

返回error

是

否

明确删除节点和删除节点的位置（左或右）

分4种情况讨论

1.度为0；2.度为1且左节点为空；3.度为1且右节点为空；4.度为2

返回OK

图3-6删除节点流程图

⑾前序遍历：函数名称是PreOrderTraverse(T)；初始条件是二叉树T存在，操作结果：先序遍历，一旦调用失败，则操作失败。

时间复杂度：O(n)

实现：先访问节点，再递归左子树，再递归右子树。

⑿中序遍历：函数名称是InOrderTraverse(T)；初始条件是二叉树T存在，操作结果：中序遍历，一旦调用失败，则操作失败。

时间复杂度：O(n)

实现：使用了栈，1. 先向左遍历并依次入栈直到遇到空节点，3. 再访问栈的首元素即上一个访问的节点并让当前指针指向右子树，同时栈首元素出栈，3.重复1，2直到栈空。

⒀后序遍历：函数名称是PostOrderTraverse(T)；初始条件是二叉树T存在，操作结果：后序遍历，一旦调用失败，则操作失败。

时间复杂度：O(n)

实现：先递归左子树，再递归右子树，再访问节点。

⒁按层遍历：函数名称是LevelOrderTraverse(T)；初始条件是二叉树T存在，操作结果：中序遍历，一旦调用失败，则操作失败。

时间复杂度：O(n)

实现：使用了队列，每访问一个节点就把他的左右节点从队尾压进队列，然后访问首节点并将首节点出队，直到队列为空。

⒂图形展示：函数名称是Display(root, ident)；当ident为0时，输出整个树的图形化表示

时间复杂度：O(n)

实现：递归遍历左右子树，根据数组vec\_left来输出空格和连线。

⒃存储二叉树：f\_write(BTree T, char \*file\_name)，操作结果是将T的数据存到file\_name文件里。

时间复杂度：O(n)

实现：判断树是否存在，打开文件，判断文件是否打开，写入数据，调用FilePreOrderWrite(T, file)函数，递归把节点内容写入文件并且用\t分隔，关闭文件。

⒄读取二叉树：f\_read(BTree &T, char \*file\_name)，操作结果是将file\_name的数据读入到T内。

时间复杂度：O(n)

意外情况：1.树已存在，返回error；2、文件未打开，输出File open error，返回error。

实现：判断表是否已存在，打开文件，判断文件是否打开，写入数据，使用了一个read\_buffer数组来将文件中的数据读入数组里，然后调用 CreateBiTree(T, read\_buffer)函数实现创建，关闭文件。

**3.4 系统测试**

## 3.4.1 准备

用初始数组init\_btree\_array创建一个二叉树方便测试

## 数组内容为：{1, 2, 4, 0, 5, 0, 0, 0, 3, 6, 0, 7, 0, 0, 8, 0, 0}

## 3.4.2 测试

说明：

在选择操作前需要选择操作的二叉树，所以实际上已经实现了对多个二叉树进行管理，测试用函数的使用如下：

CreateBiTree(LList[0], init\_btree\_array);

菜单栏如下：

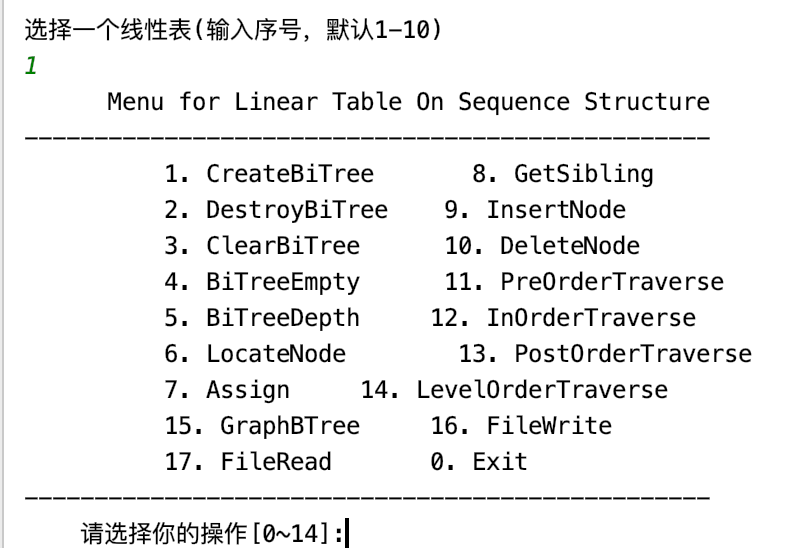


图3-7菜单栏

为了方便展示，在每一个操作后都加了一个图形展示

⑴创建二叉树

操作结果：

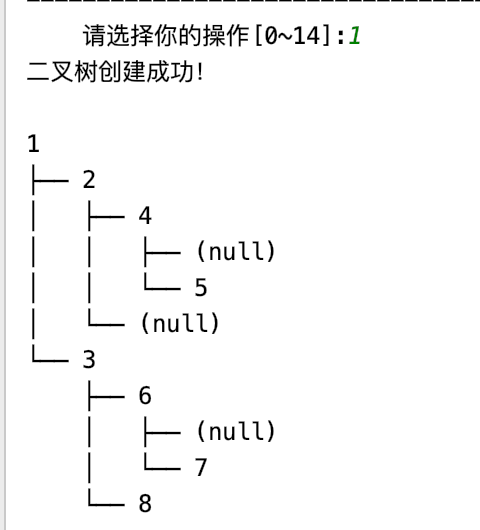


图3-8创建二叉树结果

⑵销毁二叉树

操作结果：

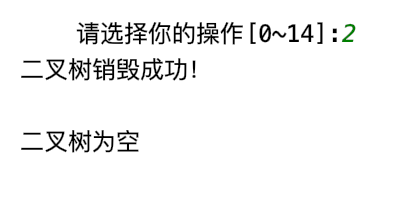


图3-9销毁二叉树结果

⑶清空二叉树

操作结果：

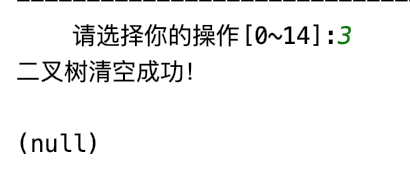


图3-10清空二叉树结果

⑷判定空二叉树

操作结果：

表 3-1判定空表结果

|  |  |
| --- | --- |
| 操作情况 | 操作结果 |
| 二叉树非空 | 截屏2019-12-01下午9.34.03  图3-11-a判定空二叉树结果（内容存在） |
| 二叉树空 | 截屏2019-12-01下午9.32.43  图3-11-b判定空二叉树结果(内容不存在) |

⑸求二叉树深度

操作结果：

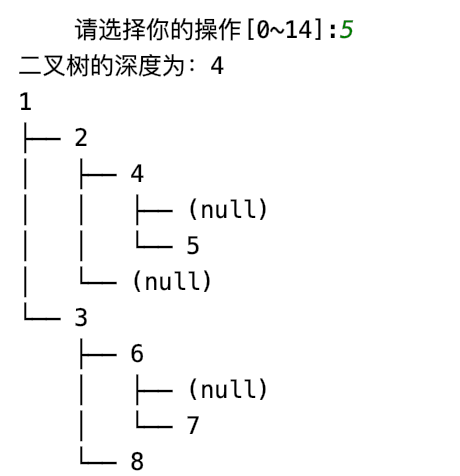


图3-12求二叉树深度结果

⑹查找节点

操作结果：

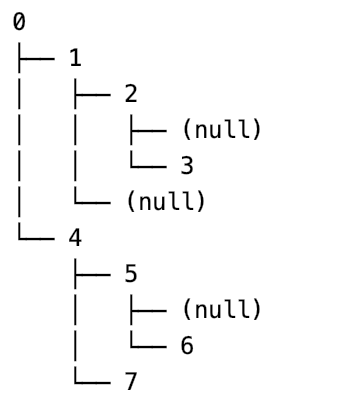


图3-13-a节点内容示意图

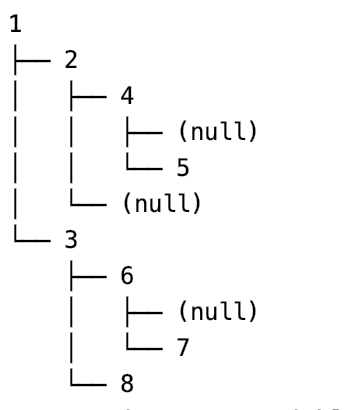


图3-13-b节点id示意图

表 3-2查找元素结果

|  |  |
| --- | --- |
| 操作情况 | 操作结果 |
| 正常操作 | 截屏2019-12-01下午10.59.48  图3-14-a查找节点结果（正常） |
| i不在二叉树范围内 | 截屏2019-12-01下午9.39.16  图3-14-b查找节点结果（非正常） |

⑺节点赋值

操作结果：

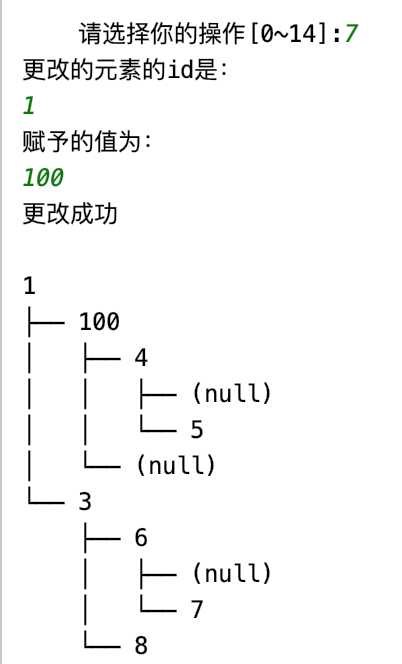


图3-15节点赋值结果

⑻获得兄弟节点

操作结果：

表 3-3获得前驱结果

|  |  |
| --- | --- |
| 操作情况 | 操作结果 |
| 正常操作 | 截屏2019-12-01下午9.41.31  图3-16-a获得兄弟节点结果（正常） |
| 目标元素没有兄弟节点 | 截屏2019-12-01下午9.43.21  图3-16-b获得兄弟节点结果（非正常） |

⑼插入节点

操作结果：

表 3-4获得后继结果

|  |  |
| --- | --- |
| 操作情况 | 操作结果 |
| 插到右节点 | 截屏2019-12-01下午9.44.06  图3-17-a插入节点结果（右） |
| 插到左节点 | 截屏2019-12-01下午9.44.32  图3-17-b插入节点结果（左） |

⑽删除节点

操作结果：

表 3-5插入元素结果

|  |  |
| --- | --- |
| 操作情况 | 操作结果 |
| 删除非根节点 | 截屏2019-12-01下午9.45.20  图3-18-a删除节点结果（非根节点） |
| 删除根节点 | 截屏2019-12-01下午9.45.50  图3-18-b删除节点结果（根节点） |

遍历树：

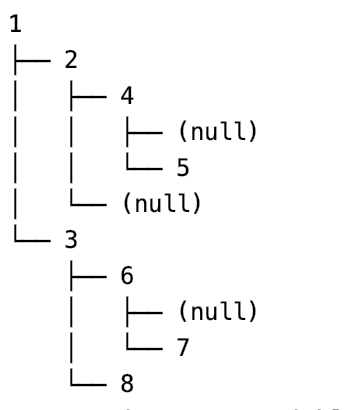


图3-19-a遍历树样例

⑾前序遍历

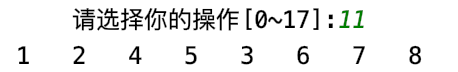


图3-19-b遍历树结果（前序）

⑿中序遍历

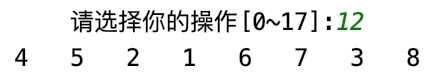


图3-19-c遍历树结果（中序）

⒀后序遍历

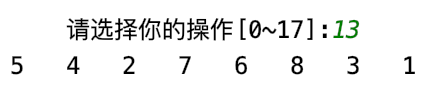


图3-19-d遍历树结果（后序）

⒁按层遍历

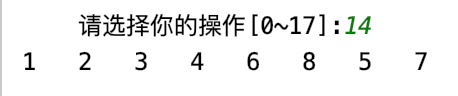


图3-19-e遍历树结果（按层）

⒂图形展示

操作结果：

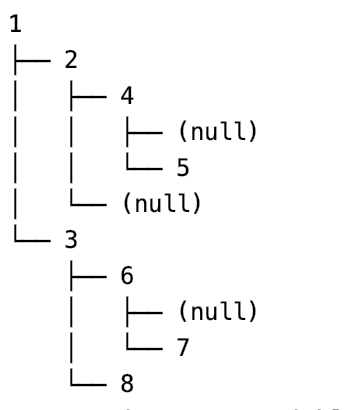
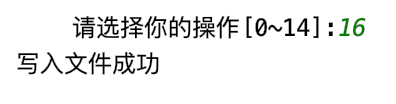


图3-20图形展示结果

⒃存储二叉树

操作结果：



截屏2019-12-01下午9.58.39

图3-21写入文件结果

⒄读取二叉树

操作结果：

截屏2019-12-01下午9.58.39

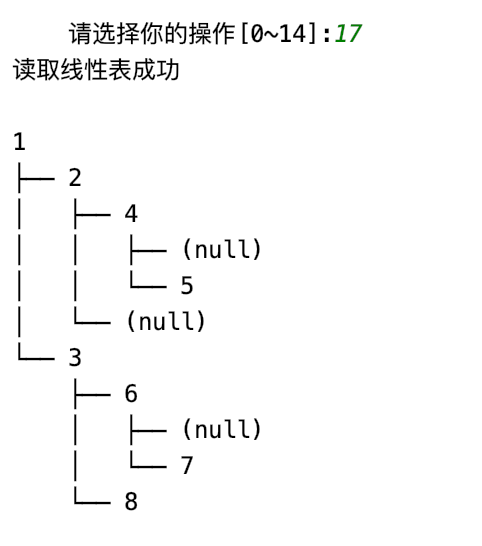


图3-22读取文件结果

## 3.5 实验小结

1. 掌握了对链式存储二叉树的使用，加深了理解。
2. 进一步规范注释的使用，与第二次实验相比注释内容增多
3. 增添了多个辅助函数，一种是简单的运算比如取最大值，还有一种是递归函数，由于递归之前可能需要加一些辅助判断（比如判断空树等），需要将递归部分单独分离出来。而且将函数分离出来可以加强程序的复用性。
4. 涉及到了栈和队列的使用，树的遍历顺序与栈的先进后出和队列的先进先出很好的匹配。
5. 多次使用了递归的算法，注意：1.递归的base line，2.递归返回值
6. 注意指向空节点的指针必须赋值为null，否则很容易出现非法访问。
7. 在访问未知节点之前必须要检查节点是否存在避免非法访问，可以巧妙利用&&运算符的短路评估。