# 数据结构实验报告——实验七

## 学号： 20201060330 姓名： 胡诚皓 得分：\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### 一、实验目的

1. 复习二叉树的逻辑结构、存储结构及基本操作；
2. 掌握使用顺序结构实现二叉树。

### 二、实验内容

阅读程序bitree.h和functions.cpp,完成以下题目：

1. 写出functions文件中每个函数对应的功能、输入要求，输出或返回内容；

2. 将functions文件的每个函数中你认为的关键代码罗列出来，并注释其作用；

3. 创建主程序，给出友好界面对functions文件的每个函数进行测试；

4. 根据所完成的主程序修改bitree头文件，使其没有缺失代码或多余代码。

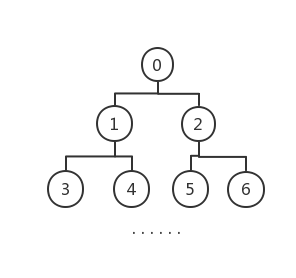
### 三、数据结构及算法描述

### 数据结构

宏定义和类型定义说明：

|  |  |
| --- | --- |
| **名称** | **说明** |
| TelemType | 即为char类型，作为二叉树中结点 |
| MAX\_TREE\_SIZE | 即为100，作为二叉树的最大结点数 |
| Status | 即为int类型，作为函数的返回值的类型，约定值为OK/ERROR |
| Boolean | 即为int类型，作为逻辑布尔值使用，约定值为TRUE/FALSE |
| TRUE、OK | 即为1，TRUE作为Boolean类型的约定值，OK作为Status类型的约定值 |
| FALSE、ERROR | 即为0，FALSE作为Boolean类型的约定值，ERROR作为Status类型的约定值 |
| SqBiTree | 即为char[100]，作为顺序存储二叉树的结构 |
| QElemType | 即为int，作为队列元素 |
| position | 结构体，包含两个int类型的变量level、order，level表示二叉树中的某一层，order表示level层中从左往右的序号。level与order都从1开始计。 |
| QueuePtr、LinkQueue | 定义与使用方法与之前的实验报告中相同 |

其中SqBiTree，用顺序存储方式存储二叉树，按层存储（下图中结点中数字为对应的顺序存储下标）



### 算法描述：

void InitBiTree(SqBiTree T)

使用for循环遍历顺序存储的二叉树T，将所有元素都先置空，即赋值为空字符

void CreateBiTree(SqBiTree T)

①初始化局部变量，将char数组s全部设为空字符

②调用InitBiTree将T置空

③先去除所有换行符，以避免读入上次输出/输入的缓冲区残留字符；再用gets函数读入剩下的字符。最后通过for循环将局部变量s中读入的各个字符赋值给T

Boolean BiTreeEmpty(const SqBiTree T)

通过判断根结点是否为空字符来确定该二叉树是否为空

int BiTreeDepth(SqBiTree const T)

①从后往前循环，找到顺序存储中二叉树的最后一个结点

②根据“每一层的最后一个结点下标为2n”来判断循环是否结束

TElemType Root(SqBiTree T)

调用BiTreeEmpty判断T是否为空，返回ERROR，否则直接返回根结点的值（即顺序存储下标为0对应的元素）

TElemType Value(SqBiTree T, position e)

①先计算level层以上的所有结点的个数，即level本层第一个结点对应的下标

②本层第一个结点的下标加上order再减1就是对应结点的下标，直接返回对应值

Status Assign(SqBiTree T, position e, TElemType value)

①与Value函数中同理可以求得位置e对应的下标

②对根结点的修改进行特殊处理

③排除两种无效情况：给无父结点的叶结点赋非空值、给有子结点的父节点赋空值

④将对应元素赋值为value

TElemType Parent(SqBiTree T, TElemType e)

①找到值为e的结点对应的下标

②根结点下标为0时，下标为i的结点的父结点为

TElemType LeftChild(SqBiTree T, TElemType e)

①找到值为e的结点对应的下标

②根结点下标为0时，下标为i的结点的左子结点为

TElemType RightChild(SqBiTree T, TElemType e)

①找到值为e的结点对应的下标

②根结点下标为0时，下标为i的结点的右子结点为

TElemType LeftSibling(SqBiTree T, TElemType e)

①找到值为e的右结点对应的下标

②根结点下标为0时，下标为i的结点的左兄弟结点为

TElemType RightSibling(SqBiTree T, TElemType e)

①找到值为e的左结点对应的下标

②根结点下标为0时，下标为i的结点的左兄弟结点为

void Move(SqBiTree Q, int j, SqBiTree T, int i)

①递归地将Q中j结点的左子树移动，作为T中i结点的左子树

②递归地将Q中j结点的右子树移动，作为T中i结点的右子树

③将Q的j结点本身移到T中i结点上，同时将Q的j结点置空

void InsertChild(SqBiTree T, TElemType p, int LR, SqBiTree C)

①找到值为p的结点对应的下标j

②根据LR计算出j的左/右子结点的下标

③若以k为根的子树不是空树，就把从T的k结点开始的子树移至k的右结点开始的子树

④把从C的i结点开始的子树移至T的k结点上

Status DeleteChild(SqBiTree T, position p, int LR)

①将位置p转换为对应下标i

②i的左/右结点不空则将其入队

③将i结点置空

④出队一个下标赋值给i。若队列为空，则flag被赋值为FALSE

void VisitFunc(TElemType e)

输出除了空格和空字符之外的字符

void PreTraverse(SqBiTree T, int e)

①访问当前结点

②若当前结点有左子树，递归访问当前结点的左子树

③若当前结点有右子树，递归访问当前结点的右子树

void PreOrderTraverse(SqBiTree T)

若二叉树T不为空，则调用PreTraverse先序遍历T

void InTraverse(SqBiTree T, int e)

①若当前结点有左子树，递归访问当前结点的左子树

②访问当前结点

③若当前结点有右子树，递归访问当前结点的右子树

void InOrderTraverse(SqBiTree T)

若二叉树T不为空，则调用InOrderTraverse中序遍历T

void PostTraverse(SqBiTree T, int e)

①若当前结点有左子树，递归访问当前结点的左子树

②若当前结点有右子树，递归访问当前结点的右子树

③访问当前结点

void PostOrderTraverse(SqBiTree T)

若二叉树T不为空，则调用PostOrderTraverse中序遍历T

void LevelOrderTraverse(SqBiTree T)

①从后往前循环，找到顺序存储中二叉树的最后一个结点

②由于顺序存储本来就是按层序存储二叉树的，因此从前往后遍历数组，在输出的时候跳过空结点即可

void Print(SqBiTree T)

①外层循环层数，调用BiTreeDepth得到T的层数

②内层从左往右循环第j层的各个结点，第j层有个结点。调用Value获取对应位置结点的值。同样地，在输出时跳过空结点

InitQueue、DestroyQueue、EnQueue、DeQueue与QueueEmpty函数在之前的实验报告中已经说明过，此处不再赘述。

### 详细设计

#include "bitree.h"

#include <iostream>

using namespace std;

int main() {

SqBiTree root, root2;

position pos;

char ch;

int LR;

InitBiTree(root);

InitBiTree(root2);

LinkQueue queue;

InitQueue(queue);

cout << "进行函数测试，有些函数在其他函数中已经有调用，不再单独测试" << endl;

cout << "按层序构造二叉树root" << endl;

CreateBiTree(root);

cout << "二叉树根结点的值为" << Root(root) << endl;

cout << "查询某个位置的结点" << endl;

cout << "请输入层号（根结点为第1层）" << endl;

cin >> pos.level;

cout << "请输入该层的某个位置（该层最左边的结点为第1个）" << endl;

cin >> pos.order;

cout << "第" << pos.level << "层" << "的第" << pos.order << "个结点的值为" << Value(root, pos) << endl;

cout << "修改某个位置的结点" << endl;

cout << "请输入层号（根结点为第1层）" << endl;

cin >> pos.level;

cout << "请输入该层的某个位置（该层最左边的结点为第1个）" << endl;

cin >> pos.order;

cout << "修改为（字符型）" << endl;

cin >> ch;

if (Assign(root, pos, ch) == ERROR)

cout << "修改失败" << endl;

else

cout << "修改成功" << endl;

cout << "按值查询结点" << endl;

cout << "查询父结点，请输入要查询的值（字符型）" << endl;

cin >> ch;

cout << ch << "的父结点的值为" << Parent(root, ch) << endl;

cout << "查询左子结点，请输入要查询的值（字符型）" << endl;

cin >> ch;

cout << ch << "的左子结点的值为" << LeftChild(root, ch) << endl;

cout << "查询右子结点，请输入要查询的值（字符型）" << endl;

cin >> ch;

cout << ch << "的右子结点的值为" << RightChild(root, ch) << endl;

cout << "查询左兄弟结点，请输入要查询的值（字符型）" << endl;

cin >> ch;

cout << ch << "的左兄弟结点的值为" << LeftSibling(root, ch) << endl;

cout << "查询右兄弟结点，请输入要查询的值（字符型）" << endl;

cin >> ch;

cout << ch << "的右兄弟结点的值为" << RightSibling(root, ch) << endl;

cout << "构造一棵只有右结点的树root2" << endl;

CreateBiTree(root2);

cout << "将root2移至root的一个结点上" << endl;

cout << "请输入目标结点" << endl;

cin >> ch;

cout << "移至" << ch << "的左（0）/右（1）结点，请输入0或1" << endl;

cin >> LR;

InsertChild(root, ch, LR, root2);

cout << "操作完成" << endl;

cout << "删除root中某一结点的左/右子树" << endl;

cout << "请输入目标层号（根结点为第1层）" << endl;

cin >> pos.level;

cout << "请输入目标层的某个位置（该层最左边的结点为第1个）" << endl;

cin >> pos.order;

cout << "删除" << Value(root, pos) << "结点的左（0）/右（1）子树，请输入0或1" << endl;

cin >> LR;

DeleteChild(root, pos, LR);

cout << "遍历二叉树root" << endl;

cout << "前序遍历：";

PreOrderTraverse(root);

cout << endl << "中序遍历：";

InOrderTraverse(root);

cout << endl << "后序遍历：";

PostOrderTraverse(root);

cout << "调用Print层序遍历：" << endl;

Print(root);

cout << "调用LevelOrderTraverse层序遍历：";

LevelOrderTraverse(root);

cout << "测试队列销毁DestroyQueue函数" << endl;

EnQueue(queue, 10);

if (DestroyQueue(queue))

cout << "测试通过" << endl;

else

cout << "测试失败" << endl;

return 0;

}

### 五、程序代码

原来文件中的代码已经全部经过优化（包括细节上的数组越界风险、整体格式等），并在相应语句上都加了详细注释。

### Question 1:

1. void InitBiTree(SqBiTree T)

功能：初始化顺序存储的二叉树，将T中的各元素置零

输入要求：SqBiTree类型的T

输出或返回内容：无输出，无返回值

1. void CreateBiTree(SqBiTree T)

功能：根据输入按层序创建一棵二叉树

输入要求：需保证T存在，按层序输入结点的值(字符)，空格表示空结点，结点数≤MAX\_TREE\_SIZE

输出或返回内容：无输出，无返回值

1. Status BiTreeEmpty(SqBiTree T)

功能：判断二叉树T是否为空

输入要求：需保证二叉树T存在

输出或返回内容：若T为空二叉树，返回TRUE，否则返回FALSE

1. int BiTreeDepth(SqBiTree T)

功能：得到二叉树T的深度

输入要求：需保证二叉树T存在

输出或返回内容：返回二叉树T的深度

1. TElemType Root(SqBiTree T)

功能：得到二叉树T根的值

输入要求：需保证二叉树T存在

输出或返回内容：若T为空，返回ERROR；若T不为空，返回二叉树T根结点的值

1. TElemType Value(SqBiTree T, position e)

功能：查询T中位置e对应的结点的值

输入要求：需保证二叉树T存在

输出或返回内容：返回T中e位置对应的结点值

1. Status Assign(SqBiTree T, position e, TElemType value)

功能：修改T中位置e对应结点的值

输入要求：需保证二叉树存在

输出或返回内容：修改成功返回OK，失败返回ERROR

1. TElemType Parent(SqBiTree T, TElemType e)

功能：查找值为e的结点并返回其父结点的值

输入要求：需保证二叉树存在

输出或返回内容：返回按层序找到的第一个值为e的结点（根结点除外）的父结点的值；若树为空或找不到，则返回“空”

1. TElemType LeftChild(SqBiTree T, TElemType e)

功能：查找值为e的结点并返回其左子结点的值

输入要求：需保证二叉树存在

输出或返回内容：返回按层序找到的第一个值为e的结点（根结点除外）的左子结点的值；若树为空或找不到，则返回“空”

1. TElemType RightChild(SqBiTree T, TElemType e)

功能：查找值为e的结点并返回其右子结点的值

输入要求：需保证二叉树存在

输出或返回内容：返回按层序找到的第一个值为e的结点（根结点除外）的右子结点的值；若树为空或找不到，则返回“空”

1. TElemType LeftSibling(SqBiTree T, TElemType e)

功能：查找值为e的右结点并返回其左兄弟结点的值

输入要求：需保证二叉树存在

输出或返回内容：返回按层序找到的第一个值为e的右结点（根结点除外）的左兄弟结点的值；若树为空或找不到，则返回“空”

1. TElemType RightSibling(SqBiTree T, TElemType e)

功能：查找值为e的左结点并返回其右兄弟结点的值

输入要求：需保证二叉树存在

输出或返回内容：返回按层序找到的第一个值为e的左结点（根结点除外）的右兄弟结点的值；若树为空或找不到，则返回“空”

1. void Move(SqBiTree Q, int j, SqBiTree T, int i)

功能：把二叉树Q以j为根结点的子树移至T的i结点上

输入要求：需保证二叉树Q与T都存在

输出或返回内容：无输出，无返回值

1. void InsertChild(SqBiTree T, TElemType p, int LR, SqBiTree C)

功能：将二叉树C根据LR插入为T的值为p的结点的左或右子树

输入要求：二叉树T存在，p是T中某个结点的值，LR为0或1，非空二叉树C与T不相交，C的右子树为空

输出或返回内容：无输出，无返回值

1. Status DeleteChild(SqBiTree T, position p, int LR)

功能：根据LR为0或1,删除T中位置p结点的左或右子树

输入要求：二叉树T存在，p指向T中某个结点，LR为1或0

输出或返回内容：删除成功返回OK，若T为空树，无法进行删除操作，返回ERROR

1. void VisitFunc(TElemType e)

功能：输出除了空格和空字符之外的字符

输入要求：TElemType（此处为char）类型的e

输出或返回内容：若e不为空格和空字符，则输出它

1. void PreTraverse(SqBiTree T, int e)

功能：作为先序遍历的递归调用函数

输入要求：e必须小于MAX\_TREE\_SIZE

输出或返回内容：每次都调用VisitFunc访问当前结点

1. void PreOrderTraverse(SqBiTree T)

功能：先序遍历二叉树T

输入要求：需要保证T存在

输出或返回内容：调用PreTraverse时会有对应输出

1. void InTraverse(SqBiTree T, int e)

功能：作为中序遍历的递归调用函数

输入要求：e必须小于MAX\_TREE\_SIZE

输出或返回内容：每次都调用VisitFunc访问当前结点

1. void InOrderTraverse(SqBiTree T)

功能：中序遍历二叉树T

输入要求：需要保证T存在

输出或返回内容：调用InTraverse时会有对应输出

1. void PostTraverse(SqBiTree T, int e)

功能：作为后序遍历的递归调用函数

输入要求：e必须小于MAX\_TREE\_SIZE

输出或返回内容：每次都调用VisitFunc访问当前结点

1. void PostOrderTraverse(SqBiTree T)

功能：后序遍历二叉树T

输入要求：需要保证T存在

输出或返回内容：调用PostTraverse时会有对应输出

1. void LevelOrderTraverse(SqBiTree T)

功能：层序遍历二叉树T

输入要求：需要保证T存在

输出或返回内容：对每个非空结点调用VisitFunc进行访问

1. void Print(SqBiTree T)

功能：层序递归遍历T，并逐层从左到右输出

输入要求：需要保证T存在

输出或返回内容：输出每一层的层号，每一层从左往右输出非空结点

1. Status InitQueue(LinkQueue &Q)

功能：队列初始化

输入要求：Q为LinkQueue类型

输出或返回内容：初始化成功返回OK，否则返回ERROR

1. Status DestroyQueue(LinkQueue &Q)

功能：销毁队列，释放各个元素所占空间

输入要求：需要保证Q存在

输出或返回内容：全部释放成功返回OK

1. Status EnQueue(LinkQueue &Q, QElemType e)

功能：元素e入列Q

输入要求：需要保证Q存在

输出或返回内容：操作成功返回OK，失败则返回ERROR

1. Status QueueEmpty(LinkQueue Q)

功能：判断队列Q是否为空

输入要求：需要保证Q存在

输出或返回内容：队列为空返回TRUE，队列非空返回FALSE

1. Status DeQueue(LinkQueue &Q, QElemType &e)

功能：从Q中出列一个元素存于e中

输入要求：需要保证Q、e存在

输出或返回内容：队列为空返回TRUE，队列非空返回FALSE

### Question 2:

1. /\* 初始化顺序存储的二叉树 \*/

void InitBiTree(SqBiTree T) {

for (int i = 0; i < MAX\_TREE\_SIZE; i++)

T[i] = '\0';//初值设为空字符，方便后期调试观察

}

2. /\* 按层序次序输入二叉树中结点的值(字符型), 构造顺序存储的二叉树T, 原数据将被置零 \*/

void CreateBiTree(SqBiTree T) {

int l;

char s[MAX\_TREE\_SIZE];

char ch;

memset(s, '\0', sizeof(s));

InitBiTree(T);

printf("请按层序输入结点的值(字符)，空格表示空结点，结点数≤%d:\n", MAX\_TREE\_SIZE);

while ((ch = (char) getchar()) == '\r' || ch == '\n');//吃掉换行符

gets(s);//读入字符串

//存入第一个字符

T[0] = ch;

l = (int) strlen(s);//求字符串的长度

//从输入的第二个字符开始循环

for (int i = 1; i <= l; i++)//将字符串赋值给T

T[i] = s[i - 1];

}

3. /\* 假设二叉树T存在。若T为空二叉树，则返回TRUE，否则FALSE \*/

Boolean BiTreeEmpty(const SqBiTree T) {

if (T[0] == '\0')//根结点为空字符，则树为空

return TRUE;

else

return FALSE;

}

4. /\* 假设二叉树T存在。返回T的深度 \*/

int BiTreeDepth(SqBiTree const T) {

int i, j = -1;

//找到最后一个结点

for (i = MAX\_TREE\_SIZE - 1; i >= 0; i--)

if (T[i] != NULL)

break;

i++;//便于计算

do

j++;

while (i >= pow(2, j));//使用do-while循环配合j=-1的初始化计算层数

return j;

}

5. /\* 假设二叉树T存在。当T不空，返回T的根，否则返回ERROR \*/

TElemType Root(SqBiTree T) {

if (BiTreeEmpty(T))//虽然保证T存在，但T仍有可能为空

return ERROR;

else

return T[0];

}

6. /\* 假设二叉树T存在。e是T中某个结点的位置，返回对应结点的值 \*/

TElemType Value(SqBiTree T, position e) {

//计算之前所有层的结点数

int pre\_node = (int) pow(2, e.level - 1) - 1;

return T[pre\_node + e.order - 1];

}

7. /\* 假设二叉树T存在。将T中e位置的结点赋新值value \*/

Status Assign(SqBiTree T, position e, TElemType value) {

//根据e计算T中对应下标

int i = (int) pow(2, e.level - 1) + e.order - 2;

//特判修改根结点值的操作

if (i == 0) {

if (T[i] == '\0')

return ERROR;

else {

T[i] = value;

return OK;

}

}

if (value != '\0' && T[(i + 1) / 2 - 1] == '\0')//不允许给无父结点的叶结点赋非空值

return ERROR;

if (value == '\0' && (T[i \* 2 + 1] != '\0' || T[i \* 2 + 2] != '\0'))//不允许给有子结点的父节点赋空值

return ERROR;

T[i] = value;

return OK;

}

8. /\* 假设二叉树T存在。按层序找到第一个值为e的结点（根结点除外）并返回其父结点的值；否则返回“空” \*/

TElemType Parent(SqBiTree T, TElemType e) {

int i;

if (T[0] == '\0')//空树直接返回“空”

return '\0';

for (i = 1; i <= MAX\_TREE\_SIZE - 1; i++)//直接从存在父结点的下标1开始

if (T[i] == e)//找到e，直接返回其父结点的值

return T[(i + 1) / 2 - 1];

return '\0';//未找到e，返回“空”

}

9. /\* 假设二叉树T存在。按层序找到第一个值为e的结点（根结点除外）并返回其左子结点的值；否则返回“空” \*/

TElemType LeftChild(SqBiTree T, TElemType e) {

int i;

if (T[0] == '\0')//空树直接返回“空”

return '\0';

for (i = 0; i <= MAX\_TREE\_SIZE - 1; i++)

if (T[i] == e)//找到e，直接返回其左子结点的值

return T[i \* 2 + 1];

return '\0';//未找到e，返回“空”

}

10. /\* 假设二叉树T存在。按层序找到第一个值为e的结点（根结点除外）并返回其右子结点的值；否则返回“空” \*/

TElemType RightChild(SqBiTree T, TElemType e) {

int i;

if (T[0] == '\0')//空树直接返回“空”

return '\0';

for (i = 0; i <= MAX\_TREE\_SIZE - 1; i++)

if (T[i] == e)//找到e，直接返回其右子结点的值

return T[i \* 2 + 2];

return '\0';//未找到e，返回“空”

}

11. /\* 假设二叉树T存在。按层序找到第一个值为e的右结点（根结点除外）并返回其左兄弟结点的值；否则返回“空” \*/

TElemType LeftSibling(SqBiTree T, TElemType e) {

int i;

if (T[0] == '\0')//空树直接返回“空”

return '\0';

for (i = 1; i <= MAX\_TREE\_SIZE - 1; i++)

if (T[i] == e && i % 2 == 0)//找到e，其序号为偶数，即右孩子

return T[i - 1];

return '\0';//未找到e，返回“空”

}

12. /\* 假设二叉树T存在。按层序找到第一个值为e的左结点（根结点除外）并返回其右兄弟结点的值；否则返回“空” \*/

TElemType RightSibling(SqBiTree T, TElemType e) {

int i;

if (T[0] == '\0')//空树直接返回“空”

return '\0';

for (i = 1; i <= MAX\_TREE\_SIZE - 1; i++)

if (T[i] == e && i % 2)//找到e，其序号为奇数，即左孩子

return T[i + 1];

return '\0';//未找到e，返回“空”

}

13. /\* 把从q的j结点开始（包括j结点本身）的子树移为从T的i结点开始的子树 \*/

void Move(SqBiTree Q, int j, SqBiTree T, int i) {

//若q的左子树不空，把q的j结点的左子树移为T的i结点的左子树

if (2 \* j + 1 < MAX\_TREE\_SIZE && Q[2 \* j + 1] != '\0')

Move(Q, (2 \* j + 1), T, (2 \* i + 1));

//若q的右子树不空，把q的j结点的右子树移为T的i结点的右子树

if (2 \* j + 2 < MAX\_TREE\_SIZE && Q[2 \* j + 2] != '\0')

Move(Q, (2 \* j + 2), T, (2 \* i + 2));

//把q的j结点移为T的i结点

T[i] = Q[j];

//把q的j结点置空

Q[j] = '\0';

}

14. /\* 假设二叉树T存在，p是T中某个结点的值，LR为0或1，非空二叉树C与T不相交，C的右子树为空。根据LR为0或1,将C作为T中p结点的左或右子树；若p结点的原来有左或右子树，则先将该子树作为C的右子树 \*/

void InsertChild(SqBiTree T, TElemType p, int LR, SqBiTree C) {

int j, k, i = 0;

//查找p的下标

for (j = 0; j < (int) pow(2, BiTreeDepth(T)) - 1; j++)

if (T[j] == p)//找到了p的下标j

break;

k = 2 \* j + 1 + LR;//T中将要被C替换掉的地方

if (T[k] != '\0') //T中以k为根的子树不是空树

Move(T, k, T, 2 \* k + 2);//把从T的k结点开始的子树移至k的右结点开始的子树

Move(C, i, T, k); //把从C的i结点开始的子树移至T的k结点上

}

15. /\* 假设二叉树T存在，p指向T中某个结点，LR为1或0。根据LR为0或1，删除T中p所指结点的左或右子树 \*/

Status DeleteChild(SqBiTree T, position p, int LR) {

int i;

Boolean flag = TRUE;//队列不空的标志

LinkQueue q;

InitQueue(q);//初始化队列，用于存放待删除的结点

i = (int) pow(2, p.level - 1) + p.order - 2;//将位置p转换为下标

if (T[i] == '\0')//T为空树

return ERROR;

i = i \* 2 + 1 + LR;//待删除子树的根结点的下标

while (flag) {

if (2 \* i + 1 < MAX\_TREE\_SIZE && T[2 \* i + 1] != '\0')//左结点不空

EnQueue(q, 2 \* i + 1);//入队左结点的下标

if (2 \* i + 2 < MAX\_TREE\_SIZE && T[2 \* i + 2] != '\0')//右结点不空

EnQueue(q, 2 \* i + 2);//入队右结点的下标

T[i] = '\0';//删除当前结点

flag = DeQueue(q, i);//获取队列中下一个结点下标并获取队列是否为空

}

return OK;

}

16. /\* 输出除了空格和空字符之外的字符 \*/

void VisitFunc(TElemType e) {

if (e != ' ' && e != '\0')

printf("%c", e);

}

17. /\* 先序遍历的递归调用函数 \*/

void PreTraverse(SqBiTree T, int e) {

//访问当前根结点

VisitFunc(T[e]);

//左子树不空，则遍历之

if (2 \* e + 1 < MAX\_TREE\_SIZE && T[2 \* e + 1] != '\0')

PreTraverse(T, 2 \* e + 1);

//右子树不空，则遍历之

if (2 \* e + 2 < MAX\_TREE\_SIZE && T[2 \* e + 2] != '\0')

PreTraverse(T, 2 \* e + 2);

}

18. /\* 先序遍历二叉树T \*/

void PreOrderTraverse(SqBiTree T) {

if (!BiTreeEmpty(T))//判断树不为空

PreTraverse(T, 0);

printf("\n");

}

19. /\* 中序遍历的递归调用函数 \*/

void InTraverse(SqBiTree T, int e) {

if (2 \* e + 1 < MAX\_TREE\_SIZE && T[2 \* e + 1] != '\0')//左子树不空，则遍历之

InTraverse(T, 2 \* e + 1);

//访问当前根结点

VisitFunc(T[e]);

if (2 \* e + 2 < MAX\_TREE\_SIZE && T[2 \* e + 2] != '\0')//右子树不空，则遍历之

InTraverse(T, 2 \* e + 2);

}

20. /\* 中序遍历二叉树T \*/

void InOrderTraverse(SqBiTree T) {

if (!BiTreeEmpty(T))//判断树不为空

InTraverse(T, 0);

printf("\n");

}

21. /\* 后序遍历的递归调用函数 \*/

void PostTraverse(SqBiTree T, int e) {

if (2 \* e + 1 < MAX\_TREE\_SIZE && T[2 \* e + 1] != '\0')//左子树不空，则遍历之

PostTraverse(T, 2 \* e + 1);

if (2 \* e + 2 < MAX\_TREE\_SIZE && T[2 \* e + 2] != '\0')//右子树不空，则遍历之

PostTraverse(T, 2 \* e + 2);

//访问当前根结点

VisitFunc(T[e]);

}

22. /\* 后序遍历二叉树T \*/

void PostOrderTraverse(SqBiTree T) {

if (!BiTreeEmpty(T))//判断树不为空

PostTraverse(T, 0);

printf("\n");

}

23. /\* 层序遍历二叉树 \*/

void LevelOrderTraverse(SqBiTree T) {

int i = MAX\_TREE\_SIZE - 1, j;

//找到最后一个非空结点的序号

while (T[i] == '\0')

i--;

//从根结点起，按层序遍历二叉树

for (j = 0; j <= i; j++)

//跳过空结点

if (T[j] != '\0')

VisitFunc(T[j]);

printf("\n");

}

24. /\* 逐层，每层从左到右输出二叉树 \*/

void Print(SqBiTree T) {

int j, k;

position p;

TElemType e;

//外层循环每一层

for (j = 1; j <= BiTreeDepth(T); j++) {

printf("第%d层: ", j);

//内层从左向右循环第j层

for (k = 1; k <= pow(2, j - 1); k++) {

p.level = j;

p.order = k;

//调用Value得到T中处于p位置结点的值

e = Value(T, p);

if (e != '\0')

printf("%c", e);

}

printf("\n");

}

}

25. /\* 队列初始化，即构造一个空队列\*/

Status InitQueue(LinkQueue &Q) {

//存储空间分配失败

if (!(Q.front = Q.rear = (QueuePtr) malloc(sizeof(QNode))))

return ERROR;

Q.front->next = NULL;

return OK;

}

26. /\* 销毁队列，同时释放各个元素所占空间 \*/

Status DestroyQueue(LinkQueue &Q) {

while (Q.front) {

//将队尾指针始终指向要删除的元素的下一个

Q.rear = Q.front->next;

free(Q.front);

Q.front = Q.rear;

}

return OK;

}

27. /\* 插入元素e为Q的新的队尾元素 \*/

Status EnQueue(LinkQueue &Q, QElemType e) {

QueuePtr p;

//存储空间分配失败

if (!(p = (QueuePtr) malloc(sizeof(QNode))))

return ERROR;

p->data = e;

//新元素的next应为NULL

p->next = NULL;

Q.rear->next = p;

//队尾变为新入队的元素

Q.rear = p;

return OK;

}

28. /\* 判断队列Q是否为空，空返回TRUE，非空返回FALSE \*/

Status QueueEmpty(LinkQueue Q) {

//带有头指针的队列判空方法

if (Q.front == Q.rear)

return TRUE;

else

return FALSE;

}

29. /\* 从Q中将出列一个元素，用e保存出列元素的值。出列失败，返回ERROR；出列成功，返回OK \*/

Status DeQueue(LinkQueue &Q, QElemType &e) {

QueuePtr p;

//先判断队列Q是否为空

if (QueueEmpty(Q) == TRUE)

return ERROR;

p = Q.front->next;

e = p->data;

Q.front->next = p->next;

//带头结点的队列变为空时的特殊处理

if (Q.rear == p)

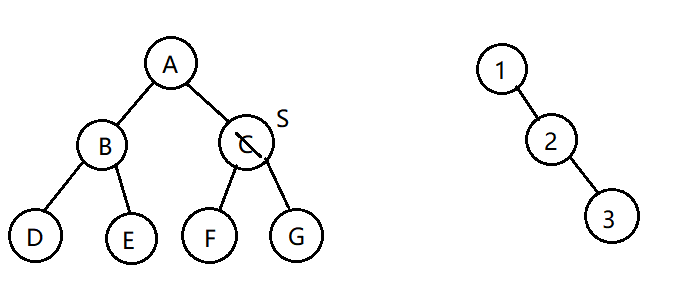
Q.rear = Q.front;

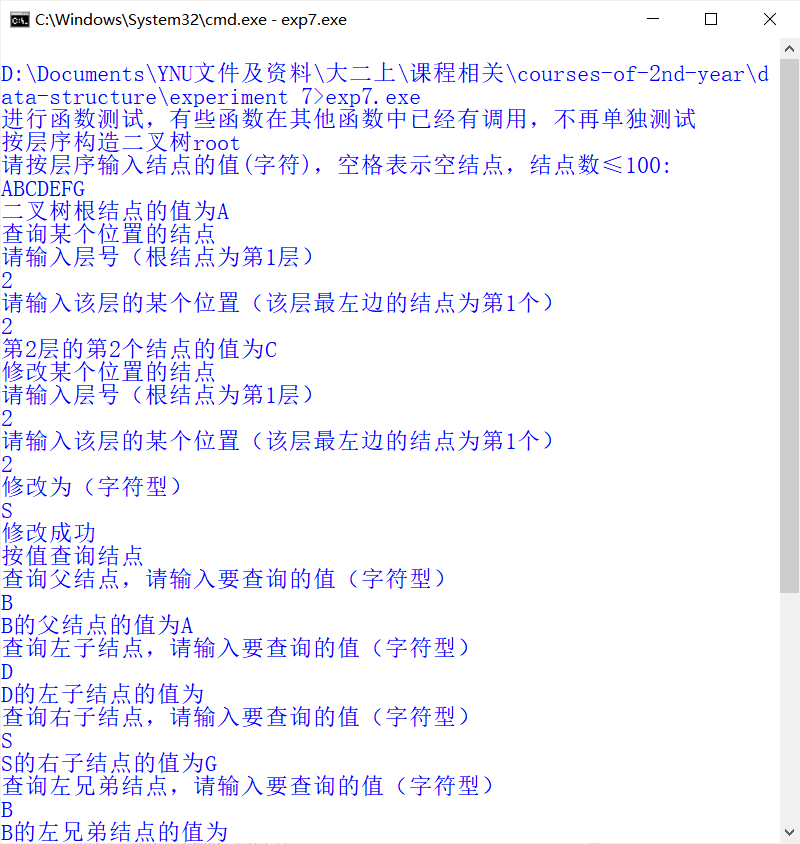
free(p);

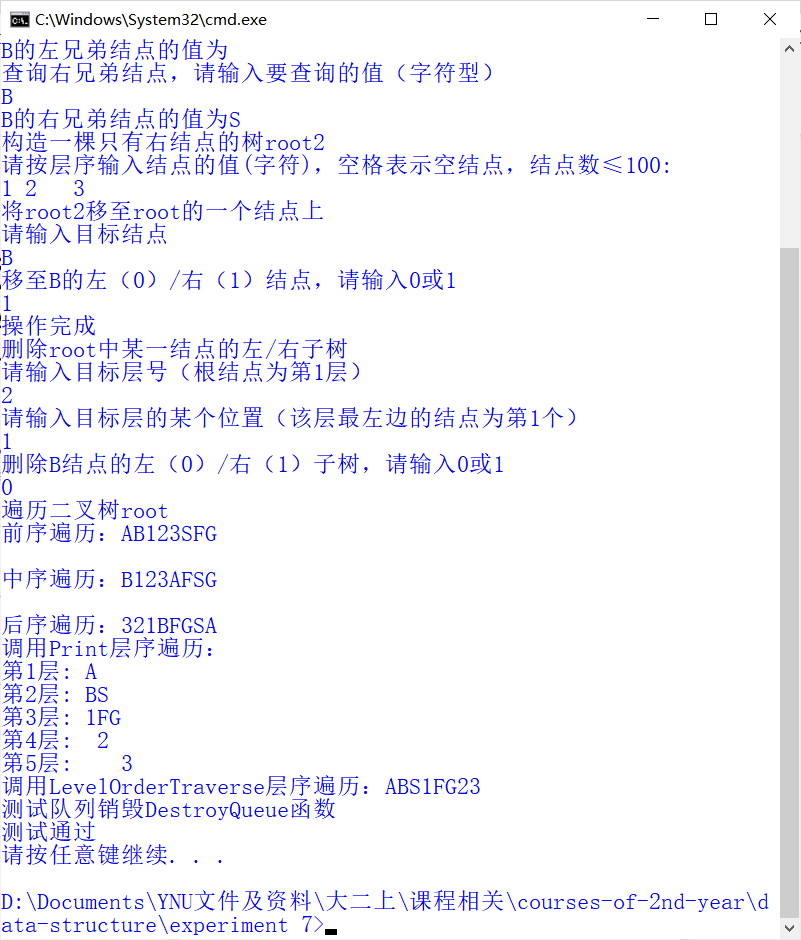
return OK;

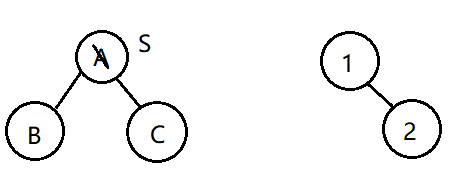
}

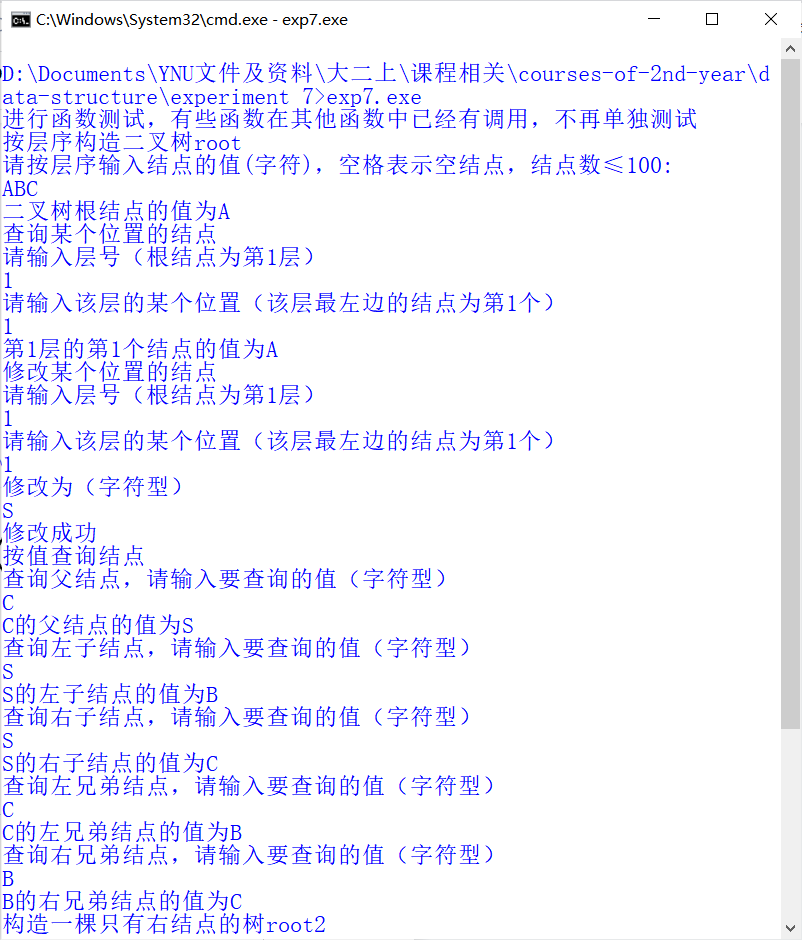
### 六、测试和结果

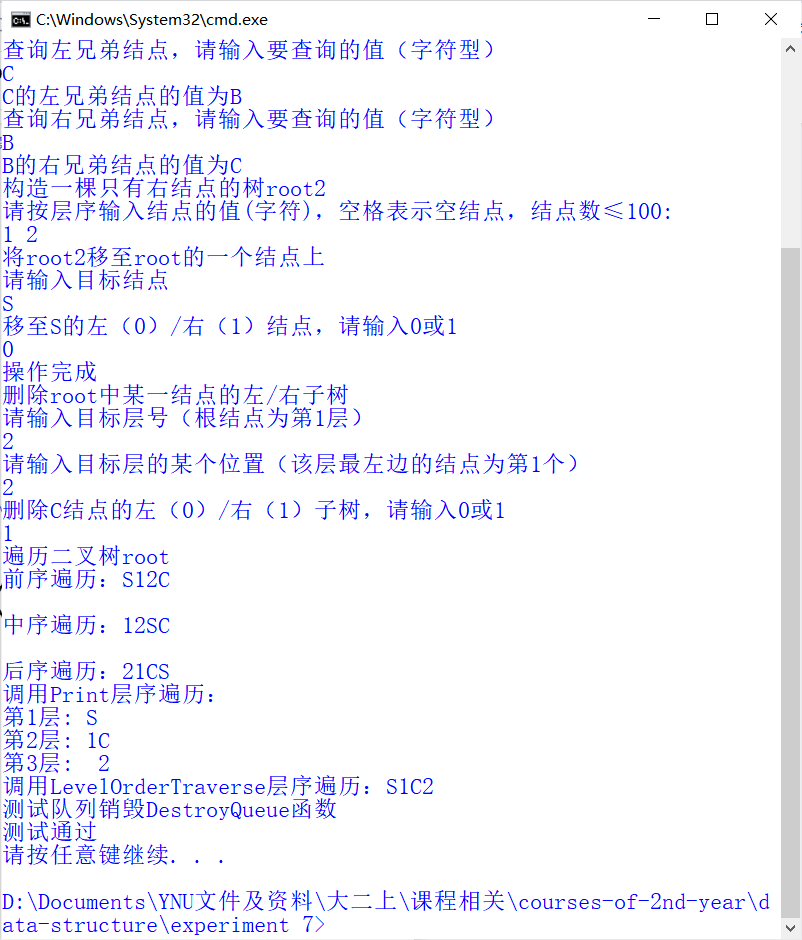












### 用户手册

输入的要查找的结点值、结点位置，都不能是空结点。**输出中的空（即对应位置没有内容）表示目标结点不存在。**