華中科技大學

课 程 实 验 报 告

课程名称： 数据结构实验

专业班级 CS2409

学　　号 U202414802

姓　　名 朱俊瑞

指导教师 李剑军

报告日期 2025 年 6 月 6 日

计算机科学与技术学院

目　　录

1　基于顺序存储结构的线性表实现 1
1.1　问题描述 1
1.1.1　实验目的 1
1.1.2　线性表运算定义 1
1.1.3　实验任务 2
1.2　系统设计 3
1.2.1　系统结构设计 3
1.2.2　数据结构设计 3
1.3　系统实现 4
1.4　系统测试 13
1.5　实验小结 17
2　基于二叉链表的二叉树实现 18
2.1　问题描述 18
2.1.1　实验目的 18
2.1.2　二叉树运算定义 18
2.1.3　实验任务 21
2.2　系统设计 21
2.2.1　系统结构设计 21
2.2.2　数据结构设计 22
2.3　系统实现 22
2.4　系统测试 35
2.5　实验小结 37
参考文献 38

1 基于顺序存储结构的线性表实现

1.1 问题描述

1.1.1 实验目的

通过实验达到：（1）加深对线性表的概念、基本运算的理解；（2）熟练掌握线性表的逻辑结构与物理结构的关系；（3）物理结构采用顺序表，熟练掌握顺序表基本运算的实现。

1.1.2 线性表运算定义

依据最小完备性和常用性相结合的原则，以函数形式定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等 12 种基本运算，具体运算功能定义如下：

（1）初始化表：函数名称是 InitList(L)；初始条件是线性表 L 不存在；操作结果是构造一个空的线性表；

（2）销毁表：函数名称是 DestroyList(L)；初始条件是线性表 L 已存在；操作结果是销毁线性表 L；

（3）清空表：函数名称是 ClearList(L)；初始条件是线性表 L 已存在；操作结果是将 L 重置为空表；

（4）判定空表：函数名称是 ListEmpty(L)；初始条件是线性表 L 已存在；操作结果是若 L 为空表则返回 TRUE, 否则返回 FALSE；

（5）求表长：函数名称是 ListLength(L)；初始条件是线性表已存在；操作结果是返回 L 中数据元素的个数；

（6）获得元素：函数名称是 GetElem(L,i,e)；初始条件是线性表已存在，1≤i≤ListLength(L)；操作结果是用 e 返回 L 中第 i 个数据元素的值；

（7）查找元素：函数名称是 LocateElem(L,e,compare())；初始条件是线性表已存在；操作结果是返回 L 中第 1 个与 e 满足关系 compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为 0；

（8）获得前驱：函数名称是 PriorElem(L,cur_e,pre_e)；初始条件是线性表 L 已存在；操作结果是若 cur_e 是 L 的数据元素，且不是第一个，则用 pre_e 返回它的前驱，否则操作失败，pre_e 无定义；

（9）获得后继：函数名称是 NextElem(L,cur_e,next_e)；初始条件是线性表 L 已存在；操作结果是若 cur_e 是 L 的数据元素，且不是最后一个，则用 next_e 返回它的后继，否则操作失败，next_e 无定义；

（10）插入元素：函数名称是 ListInsert(L,i,e)；初始条件是线性表 L 已存在，1≤i≤ListLength(L)+1；操作结果是在 L 的第 i 个位置之前插入新的数据元素 e。

（11）删除元素：函数名称是 ListDelete(L,i,e)；初始条件是线性表 L 已存在且非空，1≤i≤ListLength(L)；操作结果：删除 L 的第 i 个数据元素，用 e 返回的值；

（12）遍历表：函数名称是 ListTraverse(L,visit())，初始条件是线性表 L 已存在；操作结果是依次对 L 的每个数据元素调用函数 visit()。

附加功能：

（1）最大连续子数组和：函数名称是 MaxSubArray(L); 初始条件是线性表 L 已存在且非空，请找出一个具有最大和的连续子数组（子数组最少包含一个元素），操作结果是其最大和；

（2）和为 K 的子数组：函数名称是 SubArrayNum(L,k); 初始条件是线性表 L 已存在且非空, 操作结果是该数组中和为 k 的连续子数组的个数；

（3）顺序表排序：函数名称是 sortList(L)；初始条件是线性表 L 已存在；操作结果是将 L 由小到大排序；

（4）实现线性表的文件形式保存：其中，需要设计文件数据记录格式，以高效保存线性表数据逻辑结构 (D,R) 的完整信息；需要设计线性表文件保存和加载操作合理模式。附录 B 提供了文件存取的参考方法。

注：保存到文件后，可以由一个空线性表加载文件生成线性表。

（5）实现多个线性表管理：设计相应的数据结构管理多个线性表的查找、添加、移除等功能。

注：附加功能（5）实现多个线性表管理应能创建、添加、移除多个线性表，单线性表和多线性表的区别仅仅在于线性表管理的个数不同，多线性表管理应可自由切换到管理的每个表，并可单独对某线性表进行单线性表的所有操作。

1.1.3 实验任务

采用顺序表作为线性表的物理结构，实现 1.2 小节的运算。其中 ElemType 为数据元素的类型名，具体含义可自行定义，其它有关类型和常量的定义和引用

详见文献 [1] 的 p10。

构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。附录 A 提供了简易菜单的框架（供参考）。

1.2 系统设计

1.2.1 系统结构设计

采用顺序表作为线性表的物理结构，实现线性表的运算，并构造一个具有菜单的功能演示系统。共有 23 条操作指令：

1. InitList 初始化线性表
2. DestroyList 销毁线性表
3. ClearList 清空线性表
4. ListEmpty 线性表判空
5. ListLength 线性表长
6. GetElem 获取元素
7. LocateElem 获取元素位置
8. PriorElem 获取前驱元素
9. NextElem 获取后继元素
10. ListInsert 插入元素
11. ListDelete 删除元素
12. ListTraverse 遍历线性表
13. MaxSubArray 最大连续元素和
14. SubArrayNum 指定和的序列个数
15. SortList 排序
16. SaveList 保存线性表
17. LoadList 获取线性表
18. AddList 添加线性表
19. RemoveList 移除线性表
20. LocateList 获取线性表位置
21. ChangeList 切换线性表
22. ShowList 显示线性表

0. Exit 退出

其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。在 main 函数中使用 while 循环使输入操作可持续进行，根据菜单通过输入操作的编号来执行相应操作，并使用 switch 分支来处理相应编号的操作。整个程序使用一个指针指向当前线性表地址，实现对当前线性表的维护运行，从而利于切换线性表、从多线性表管理中选择单个线性表进行操作等。

1.2.2 数据结构设计

本实验使用顺序存储结构的线性表，并定义了线性表集合类型，相关定义如下：

```
typedef int status;
typedef int ElemType; //数据元素类型定义
#define LIST_INIT_SIZE 100
#define LISTINCREMENT  10
typedef struct {  //顺序表（顺序结构）的定义
    ElemType* elem;
    int length;
    int listsize;
}SqList;
typedef struct {  //线性表的管理表定义
    struct {
        char name[30];
        SqList L;
    } elem[10];
    int length;
    int listsize; //线性表集合的定义Lists
}LISTS;
```

1.3 系统实现

1.InitList(SqList*& L) 如果线性表 L 不存在，则构造一个空的线性表; 如果线性表 L 存在，则为指向当前线性表的指针重新分配空间，再构造一个空的线性表。

2.DestroyList(SqList& L) 如果线性表 L 存在，销毁线性表 L，释放数据元素的空间。

3.ClearList(SqList& L) 如果线性表 L 存在，删除线性表 L 中的所有元素。

4.ListEmpty(SqList L) 如果线性表 L 存在，判断线性表 L 是否为空，空就返回 TRUE，否则返回 FALSE。

5.ListLength(SqList L) 如果线性表 L 存在，返回线性表 L 的长度。

6.GetElem(SqList L, int i, ElemType& e) 如果线性表 L 存在，获取线性表 L 的第 i 个元素，保存在 e 中，返回 OK；如果 i 不合法（即 i<1 || i > L.length），返回 ERROR。

7.LocateElem(SqList L, ElemType e, int (*compare)(ElemType, ElemType)) 如

果线性表 L 存在，查找元素 e 在线性表 L 中的位置序号并返回该序号；如果 e 不存在，返回 0。

8.PriorElem(SqList L, ElemType e, ElemType& pre) 如果线性表 L 存在，获取线性表 L 中元素 e 的前驱，保存在 pre 中，返回 OK；如果没有前驱，返回 ERROR。采用线性遍历查找。

9.NextElem(SqList L, ElemType e, ElemType& next) 如果线性表 L 存在，获取线性表 L 元素 e 的后继，保存在 next 中，返回 OK；如果没有后继，返回 ERROR。采用线性遍历查找。

10.ListInsert(SqList& L, int i, ElemType e) 如果线性表 L 存在，将元素 e 插入到线性表 L 的第 i 个元素之前，返回 OK；当插入位置不正确时，返回 ERROR。判断 i 合不合法后直接插入，移动其它元素。

11.ListDelete(SqList& L, int i, ElemType& e) 如果线性表 L 存在，删除线性表 L 的第 i 个元素，并保存在 e 中，返回 OK；当删除位置不正确时，返回 ERROR。考虑是否是删除最后一个元素，如果是，只需 L.length- -。

12.ListTraverse(SqList L,void(*visit)(ElemType)) 如果线性表 L 存在，依次对 L 的每个数据元素调用函数 visit()，每个元素间空一格，返回 OK。

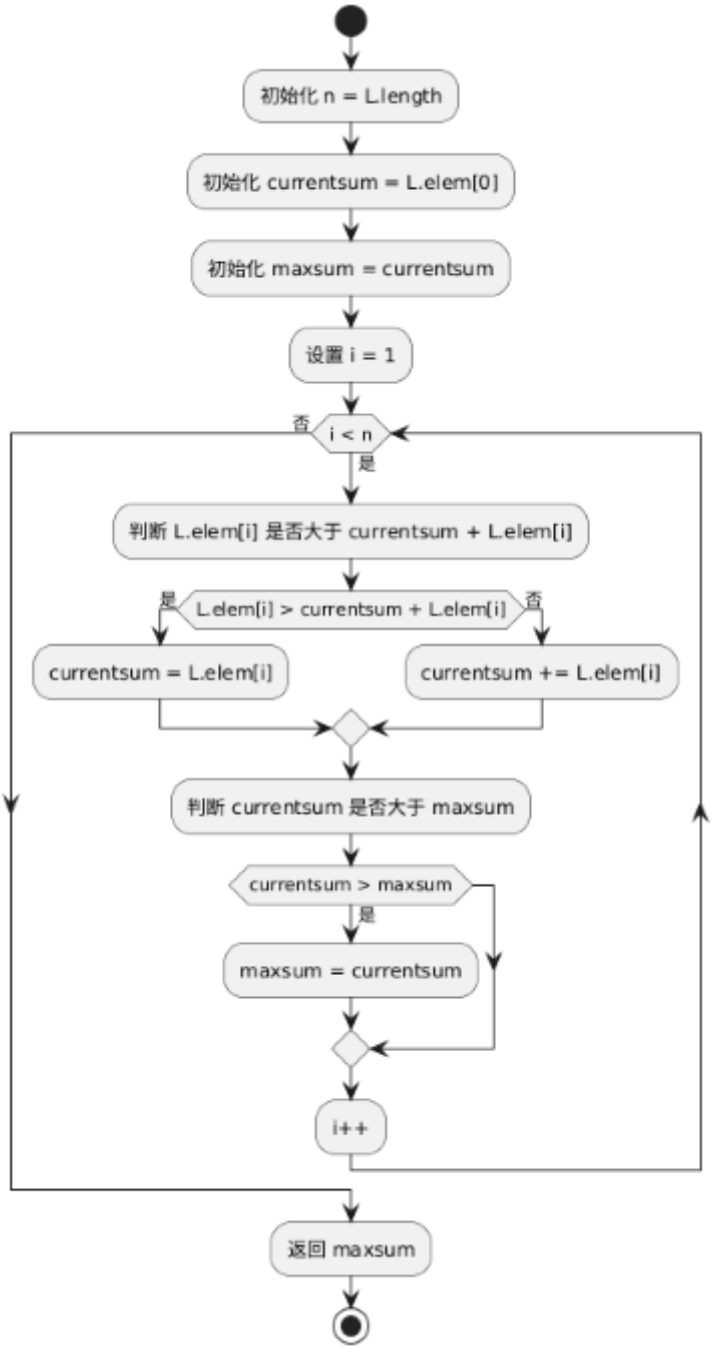

图 1-1　函数 13 的程序流程图

13.MaxSubArray(SqList L) 采用动态规划法找出一个具有最大和的连续子数组（子数组最少包含一个元素），操作结果是其最大和。

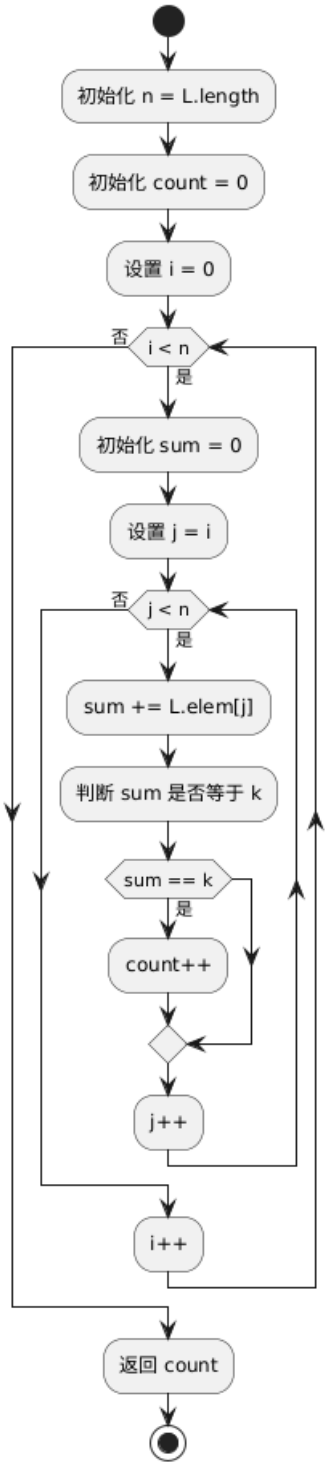

图 1-2　函数 14 的程序流程图

14.SubArrayNum(SqList L, ElemType k) 采用暴搜方法找该数组中和为 k 的连续子数组的个数。

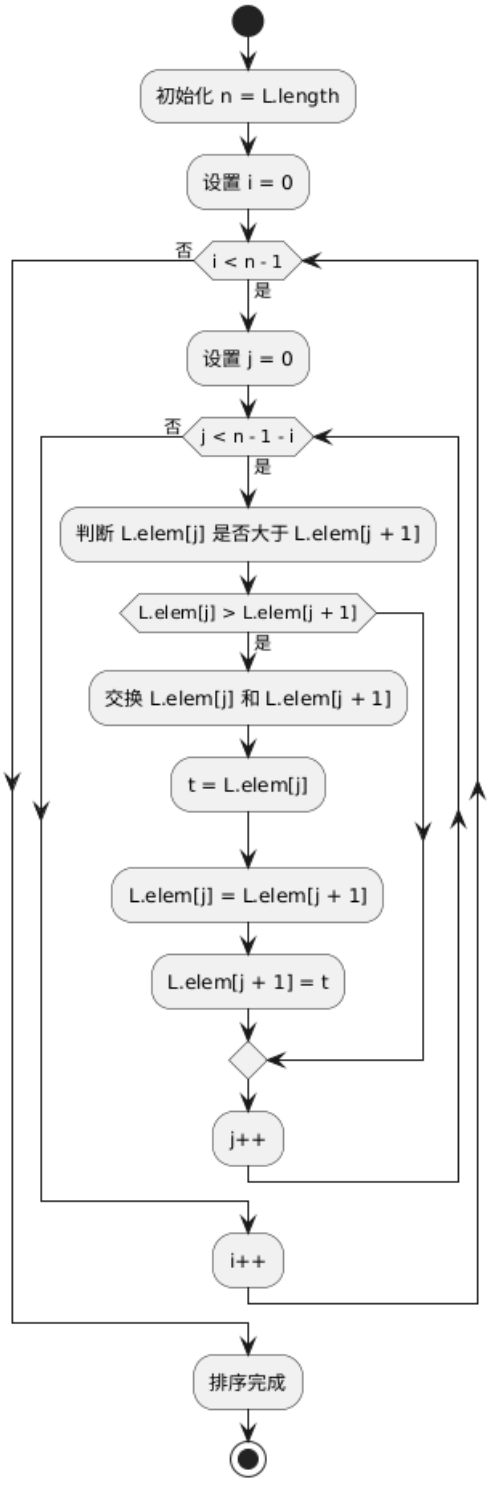

图 1-3　函数 15 的程序流程图

15.SortList(SqList L) 采用冒泡排序方法将 L 由小到大排序。

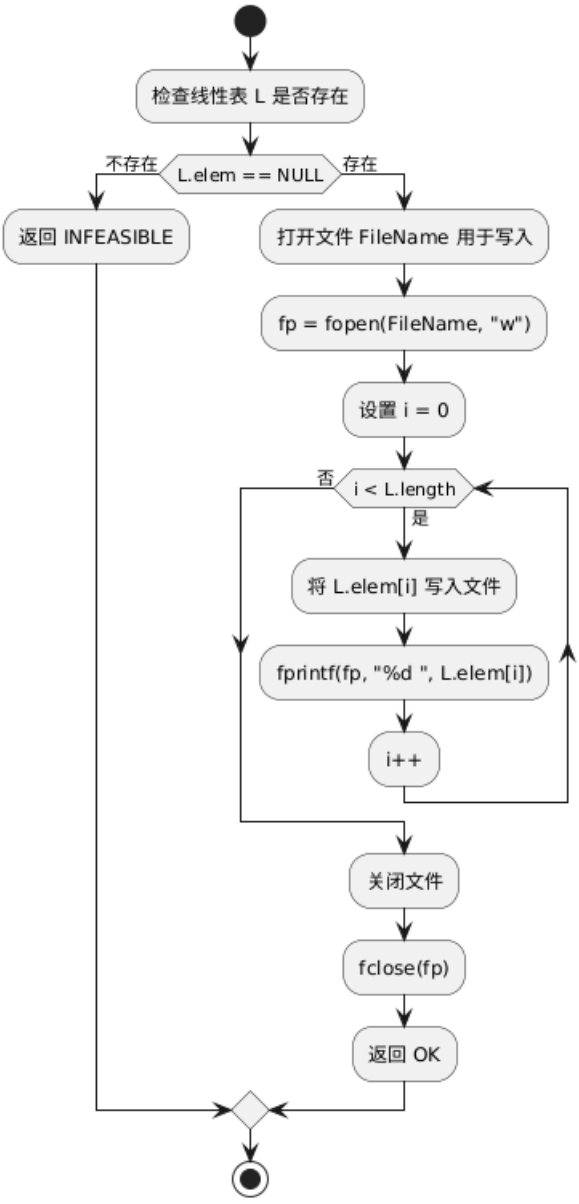

图 1-4　函数 16 的程序流程图

16.SaveList(SqList L, char FileName[]) 如果线性表 L 存在，将线性表 L 的的元素写到 FileName 文件中。

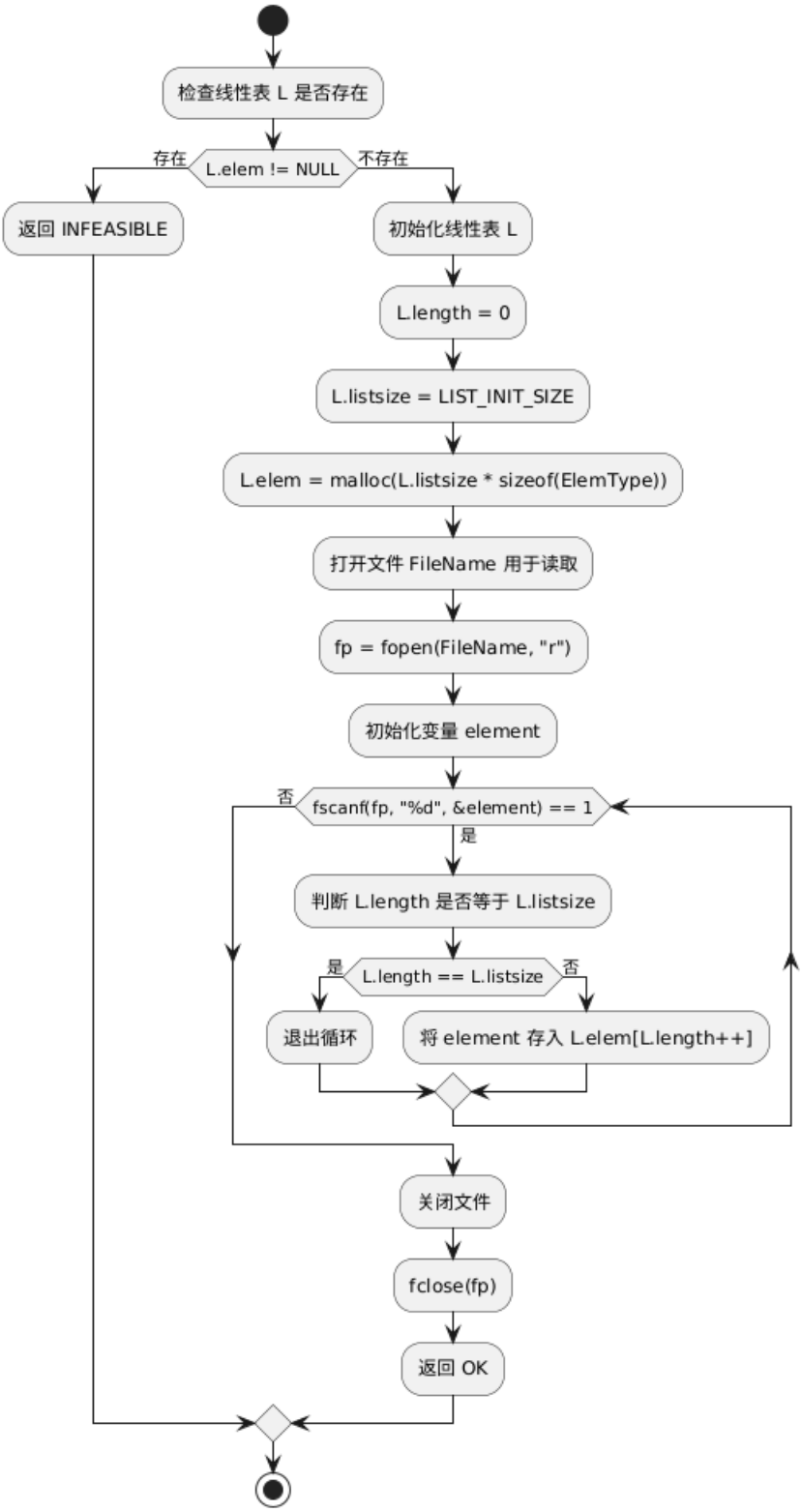

图 1-5　函数 17 的程序流程图

17.LoadList(SqList& L, char FileName[]) 如果线性表 L 不存在，将 FileName

文件中的数据读入到线性表 L 中。

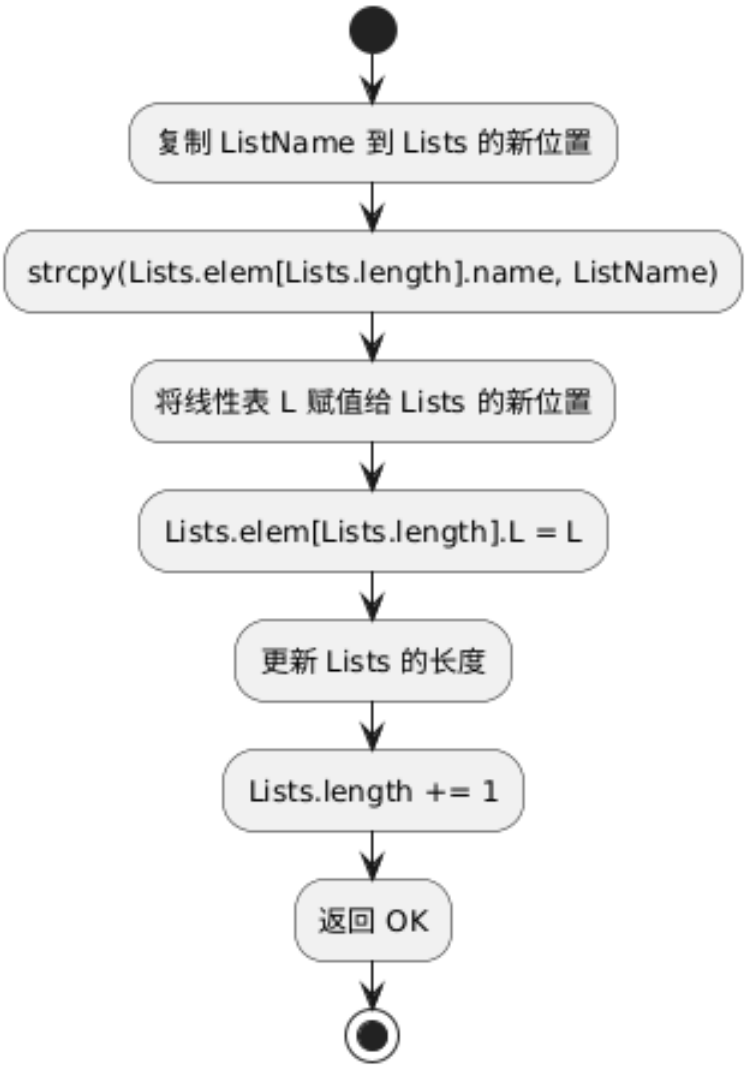

图 1-6　函数 18 的程序流程图

18.AddList(LISTS& Lists, SqList L,char ListName[]) 在多线性表管理 Lists 中增加一个名称为 ListName 的线性表。

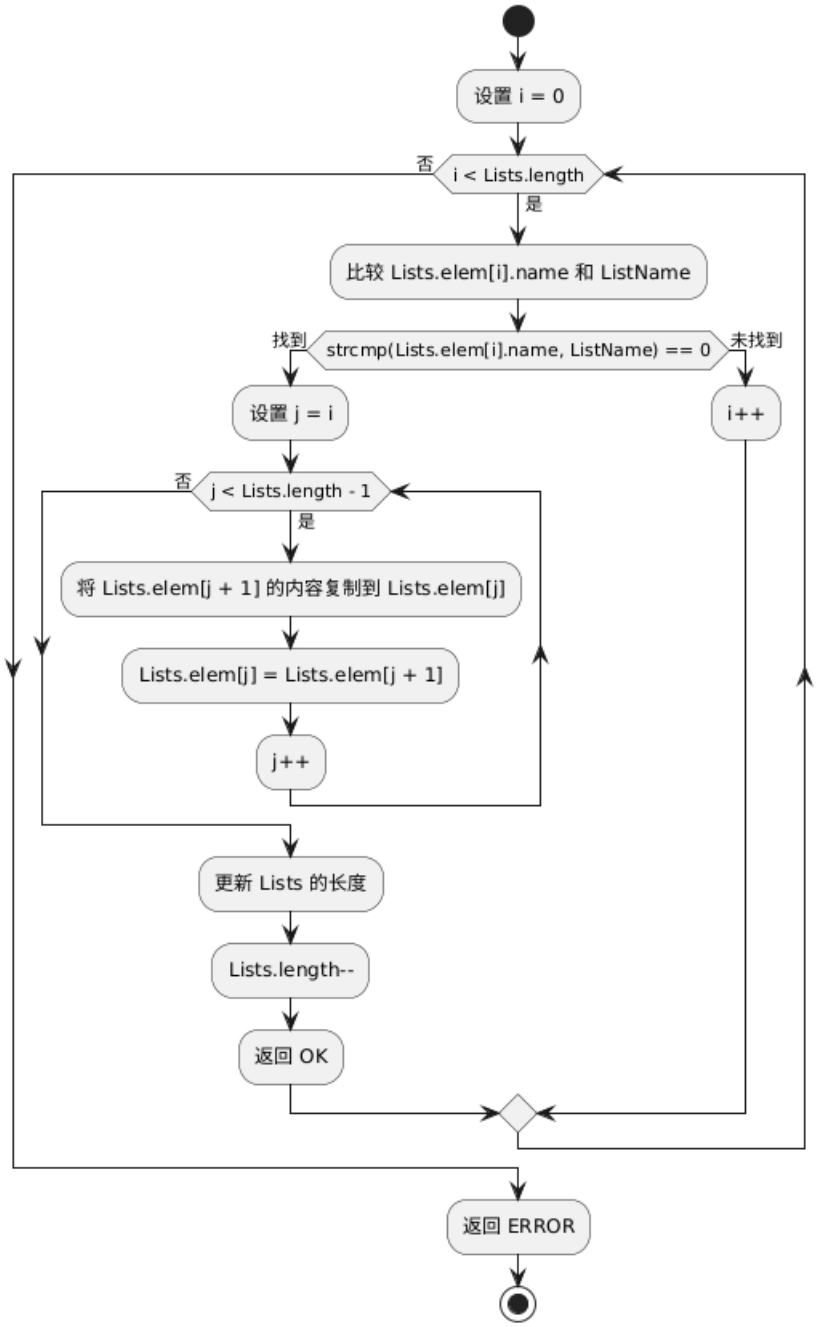

图 1-7　函数 19 的程序流程图

19.RemoveList(LISTS& Lists, char ListName[]) 在多线性表管理 Lists 中删除一个名称为 ListName 的线性表。

20.LocateList(LISTS Lists, char ListName[]) 在多线性表管理 Lists 中查找一个名称为 ListName 的线性表，成功返回逻辑序号，否则返回 0。

21.ShowList(LISTS Lists) 显示多线性表管理 Lists 中的所有线性表的名字。

1.4 系统测试

1.InitList, DestroyList, ClearList, GetElem, ListTraverse, SaveList, LoadList 成功返回 OK，否则返回 INFEASIBLE。

图 1-8　例：功能 2 的测试结果

2.ListEmpty 根据线性表 L 是否为空，空就返回 TRUE，否则返回 FALSE；如果线性表 L 不存在，返回 INFEASIBLE。

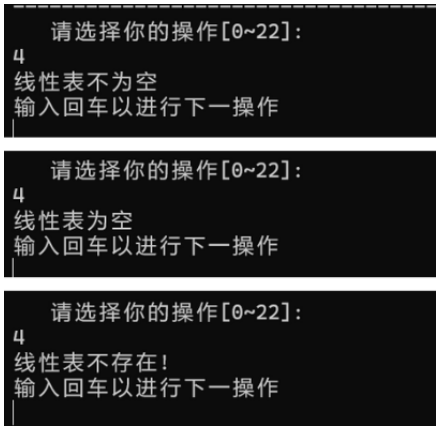

图 1-9　功能 4 的测试结果

3.ListLength 如果线性表 L 存在，返回线性表 L 的长度，否则返回 INFEASIBLE。

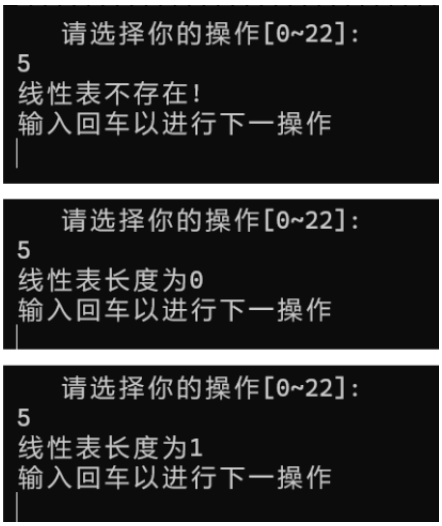

图 1-10　功能 5 的测试结果

4.LocateElem 如果线性表 L 存在，查找元素 e 在线性表 L 中的位置序号并返回该序号；如果 e 不存在，返回 0；当线性表 L 不存在时，返回 INFEASIBLE（即-1）。

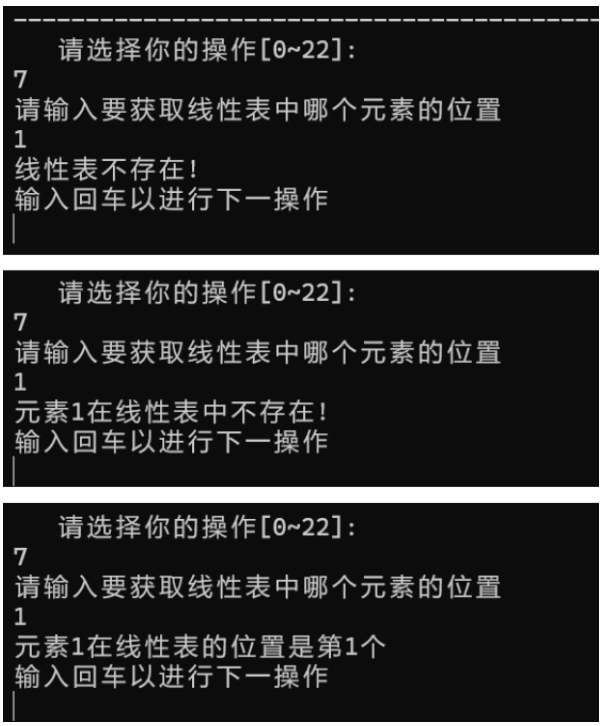

图 1-11　功能 7 的测试结果

5.PriorElem, NextElem 成功返回 OK，如果没有前驱/后继，返回 ERROR，否

则返回 INFEASIBLE。

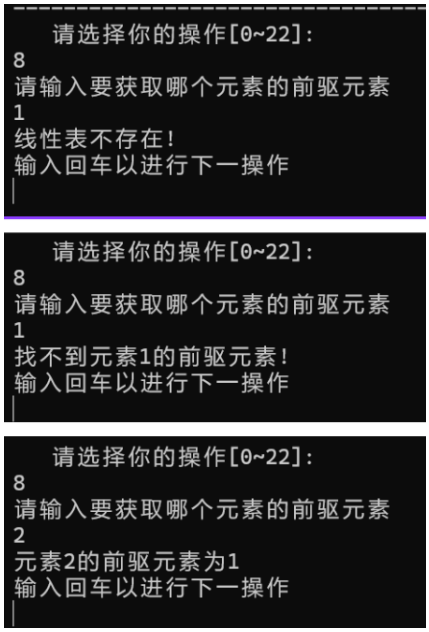

图 1-12　例：功能 8 的测试结果

6.ListInsert, ListDelete 成功返回 OK；当插入/删除位置不正确时，返回 ERROR，否则返回 INFEASIBLE。

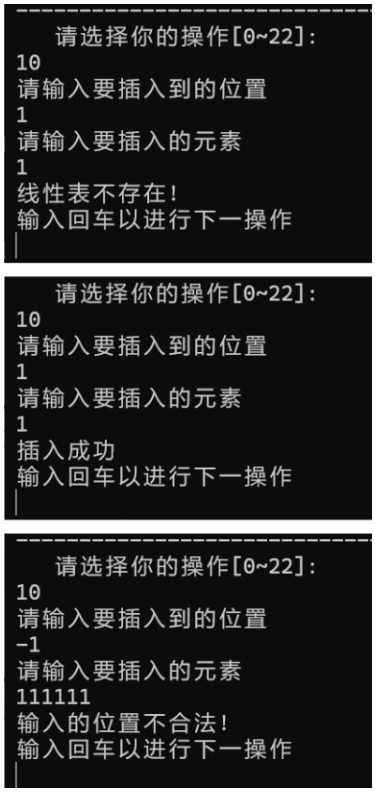

图 1-13　例：功能 10 的测试结果

7.MaxSubArray, SubArrayNum 直接返回结果。

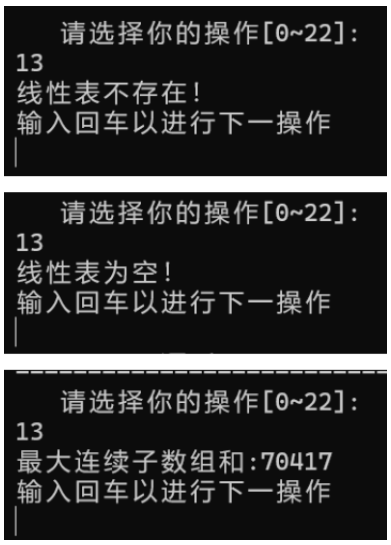

图 1-14　例：功能 13 的测试结果

8.RemoveList 成功返回 OK，否侧返回 ERROR。

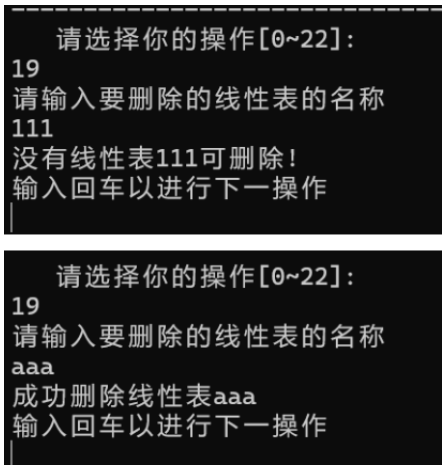

图 1-15　功能 19 的测试结果

9.LocateList 成功返回逻辑序号，否则返回 0。

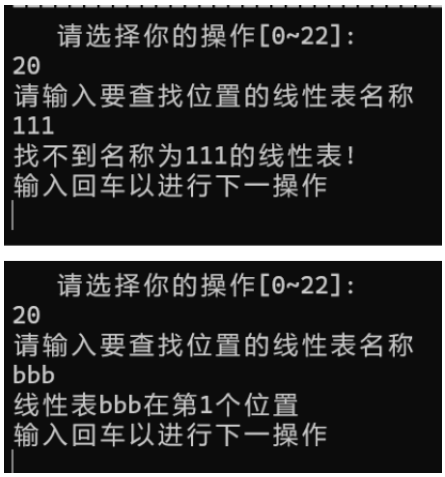

图 1-16　功能 20 的测试结果

1.5　实验小结

在本节数据结构实验中，我基于数组实现了线性表的顺序存储结构，完成了初始化、插入、删除及遍历等核心操作。通过数组下标与线性表逻辑位置的映射，深刻体会到顺序存储结构的存储特性与操作逻辑。

实验过程中，线性表初始化通过为数组分配空间并设置初始表长完成；插入操作需将插入位置后的元素依次后移，删除操作则要将删除位置后的元素前移，在插入和删除操作中对操作元素在边界的处理非常重要，需精准处理数组下标从 0 开始带来的边界差异，以及循环条件的处理和表长的更新。在遍历线性表时，表长作为循环终止条件，任何错误设置都可能导致数组越界。

操作实践中，我偶尔因为下标计算错误、边界条件遗漏导致程序异常。例如插入时未考虑数组满的情况，删除后未正确更新表长。通过绘制操作示意图、分步骤调试，逐步理清了逻辑关系，最终成功实现功能。

此次实验不仅巩固了线性表顺序存储的理论知识，更让我认识到编程细节对算法正确性的重要性。后续我计划探索动态数组扩容机制，并对比顺序存储与链式存储在不同场景下的性能差异，进一步深化对数据结构的理解与应用。

2 基于二叉链表的二叉树实现

二叉链表，别名左孩子右兄弟表示法，是树的存储结构，链表中结点的两个链域分别指向该结点的第一个孩子结点和下一个兄弟结点。

2.1 问题描述

2.1.1 实验目的

通过实验达到：（1）加深对二叉树的概念、基本运算的理解；（2）熟练掌握二叉树的逻辑结构与物理结构的关系；（3）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。

2.1.2 二叉树运算定义

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。具体运算功能定义和说明如下：

（1）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，如带空子树的二叉树前序遍历序列、或前序 + 中序、或后序 + 中序；操作结果是按 definition 构造二叉树 T；

注：要求 T 中各结点关键字具有唯一性。后面各操作的实现，也都要满足一棵二叉树中关键字的唯一性，不再赘述；CreateBiTree 中根据 definition 生成 T，不应在 CreateBiTree 中输入二叉树的定义。

（2）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；

注：销毁了则原先所有分配的空间都销毁。

（3）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；

注：清空只是将二叉树清空了，是空树，但是在内存中还存在空间，只是值不确定。

（4）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；

（5）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；

（6）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；

（7）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；

（8）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；

（9）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；

特殊情况，c 插入作为根结点？可以考虑 LR 为-1 时，作为根结点插入，原根结点作为 c 的右子树。

（10）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；

（11）前序遍历：函数名称是 PreOrderTraverse(T,Visit())；初始条件是二叉树 T 存在，Visit 是一个函数指针的形参（可使用该函数对结点操作）；操作结果：先序遍历，对每个结点调用函数 Visit 一次且一次，一旦调用失败，则操作失败。

注：前序、中序和后序三种遍历算法，要求至少一个用非递归算法实现。

（12）中序遍历：函数名称是 InOrderTraverse(T,Visit))；初始条件是二叉树 T 存在，Visit 是一个函数指针的形参（可使用该函数对结点操作）；操作结果是中序遍历 t，对每个结点调用函数 Visit 一次且一次，一旦调用失败，则操作失败；

（13）后序遍历：函数名称是 PostOrderTraverse(T,Visit))；初始条件是二叉树

T 存在，Visit 是一个函数指针的形参（可使用该函数对结点操作）；操作结果是后序遍历 t，对每个结点调用函数 Visit 一次且一次，一旦调用失败，则操作失败。

（14）按层遍历：函数名称是 LevelOrderTraverse(T,Visit))；初始条件是二叉树 T 存在，Visit 是对结点操作的应用函数；操作结果是层序遍历 t，对每个结点调用函数 Visit 一次且一次，一旦调用失败，则操作失败。

附加功能：

（1）最大路径和：函数名称是 MaxPathSum(T)，初始条件是二叉树 T 存在；操作结果是返回根节点到叶子结点的最大路径和；

注：从根节点开始。

（2）最近公共祖先：函数名称是 LowestCommonAncestor(T,e1,e2)；初始条件是二叉树 T 存在；操作结果是该二叉树中 e1 节点和 e2 节点的最近公共祖先；

注：最近公共祖先可以是 e1，e2 中的某个节点。

（3）翻转二叉树：函数名称是 InvertTree(T)，初始条件是线性表 L 已存在；操作结果是将 T 翻转，使其所有节点的左右节点互换；

（4）实现线性表的文件形式保存：其中，需要设计文件数据记录格式，以高效保存二叉树数据逻辑结构 (D,R) 的完整信息；需要设计二叉树文件保存和加载操作合理模式。附录 B 提供了文件存取的方法；

注：保存到文件后，可以由一个空二叉树加载文件生成二叉树。

（5）实现多个二叉树管理：可采用线性表的方式管理多个二叉树，线性表中的每个数据元素为一个二叉树的基本属性，至少应包含有二叉树的名称。基于顺序表实现的二叉树管理，其物理结构的参考设计如图 3-1 所示。

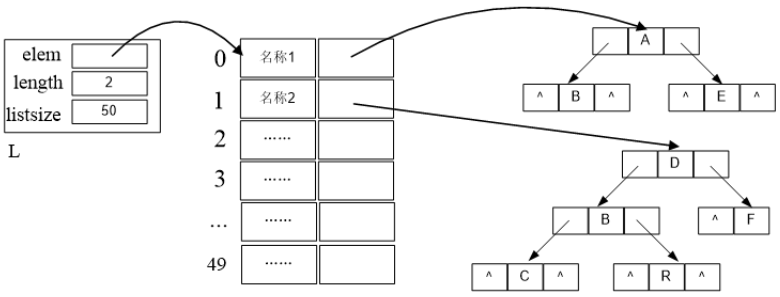

图 2-1　多二叉树管理的物理结构示意图

注：附加功能（5）实现多个二叉树管理应能创建、添加、移除多个二叉树，单二叉树和多二叉树的区别仅仅在于二叉树管理的个数不同，多二叉树管理应可自由切换到管理的每个二叉树，并可单独对某二叉树进行单二叉树的所有操

作。

2.1.3 实验任务

采用二叉链表表作为二叉树的物理结构，实现 3.2 小节的运算。其中 ElemType 为数据元素的类型名，具体含义可自行定义，但要求二叉树结点类型为结构型，至少包含二个部分，一个是能唯一标识一个结点的关键字（类似于学号或职工号），另一个是其它部分。其它有关类型和常量的定义和引用详见文献 [1] 的 p10。

构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示（供参考）。

2.2 系统设计

2.2.1 系统结构设计

采用二叉链表作为二叉树的物理结构，实现二叉树的运算，并构造一个具有菜单的功能演示系统。共有 25 条操作指令：

1. CreateBiTree 创建二叉树
2. DestroyBiTree 销毁二叉树
3. ClearBiTree 清空二叉树
4. BiTreeEmpty 二叉树判空
5. BiTreeDepth 二叉树深度
6. LocateNode 查找结点
7. Assign 结点赋值
8. GetSibling 获取兄弟结点
9. InsertNode 插入结点
10. DeleteNode 删除结点
11. PreOrderTraverse 前序遍历
12. InOrderTraverse 中序遍历
13. PostOrderTraverse
14. LevelOrderTraverse 层序遍历
15. MaxPathSum 最大路径和
16. LowestCommonAncestor 最近公共祖先
17. InvertTree 翻转二叉树
18. SaveTree 保存二叉树
19. LoadTree 获取二叉树
20. AddTree 添加二叉树
21. RemoveTree 移除二叉树
22. LocateTree 获取二叉树位置
23. ChangeTree 切换二叉树
24. ShowTree 显示二叉树

0. Exit 退出

其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。在 main 函数中使用 while 循环使输入操作可持续进行，根据菜单通过输入操作的编号来执行相应操作，并使用 switch 分支来处

理相应编号的操作。整个程序使用一个指针指向当前二叉树地址，实现对当前二叉树的维护运行，从而利于切换二叉树、从多二叉树管理中选择单个二叉树进行操作等。

2.2.2 数据结构设计

本实验使用基于二叉链表的二叉树，并定义了二叉树集合数据类型，相关定义如下:

```
typedef int status;
typedef int ElemType; //数据元素类型定义
typedef int KeyType;
#define LIST_INIT_SIZE 100
#define LISTINCREMENT  10
typedef struct {
    KeyType  key;
    char others[20];
} TElemType; //二叉树结点类型定义
typedef struct BiTNode {  //二叉链表结点的定义
    TElemType  data;
    struct BiTNode* lchild, * rchild;
} BiTNode, *BiTree;
typedef struct {  //二叉树的管理表定义
    struct {
        char name[30];
        BiTree L;
    } elem[10];
    int length;
    int listsize;
}LISTS;
```

2.3 系统实现

1.CreateBiTree(BiTree& T, TElemType definition[]) 根据带空枝的二叉树先根遍历序列 definition 构造一棵二叉树，将根节点指针赋值给 T 并返回 OK，如果有相同的关键字，返回 ERROR。构建 BuildTree(int& index, TElemType definition[])

函数递归地实现建造树。利用一个 array1 数组记录输入的关键字以检查是否输入相同关键字。

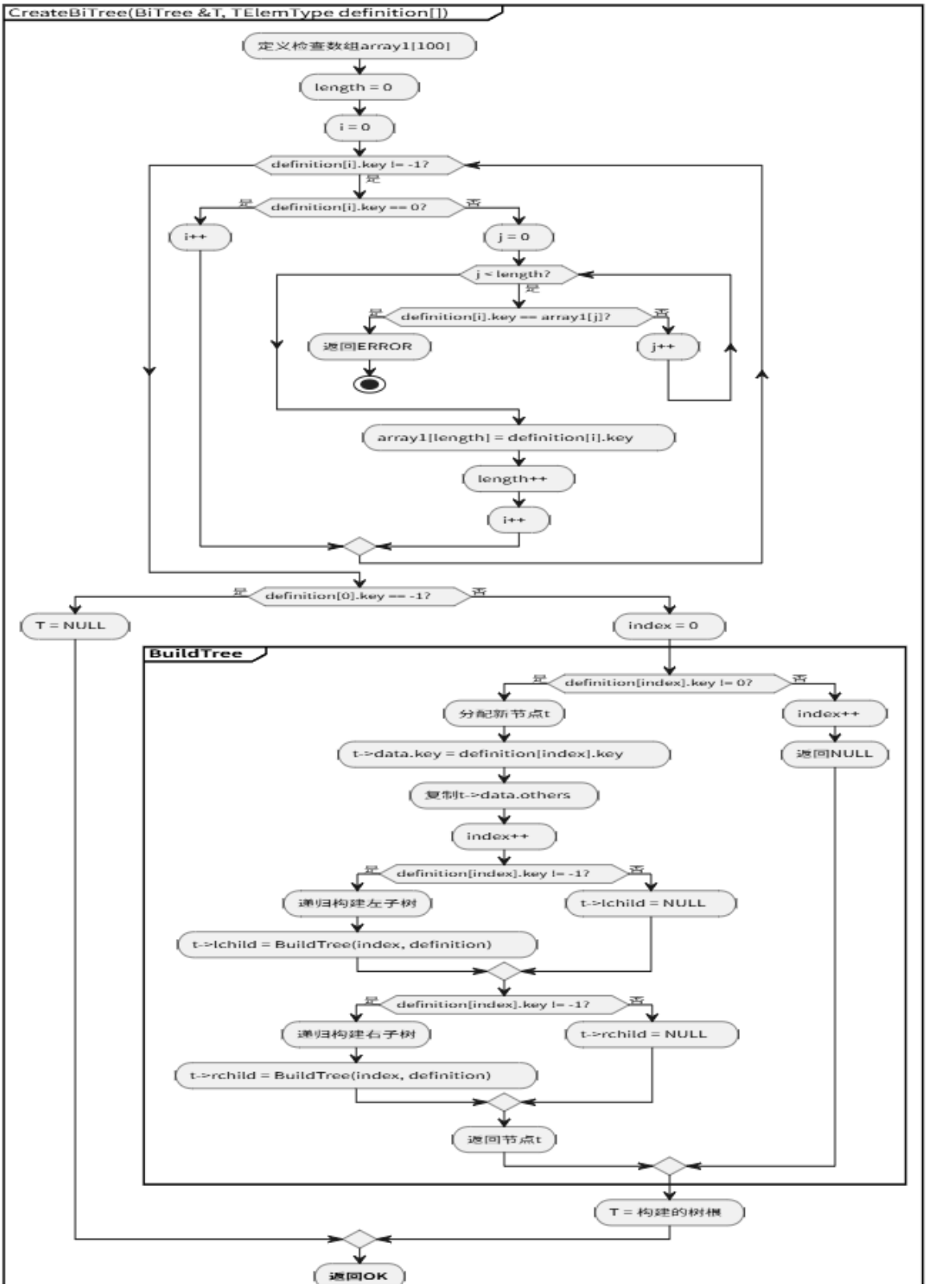

图 2-2　函数 1 的程序流程图

2.DestroyBiTree(BiTree& T) 将原先二叉树所有分配的空间都销毁。

3.ClearBiTree(BiTree& T) 将二叉树设置成空，并删除所有结点，释放结点空间。递归实现，将左右子树的指针域设置为 NULL。

4.BiTreeEmpty(BiTree T) 若 T 为空二叉树则返回 TRUE，否则返回 FALSE。

5.BiTreeDepth(BiTree T) 递归求二叉树 T 的深度。

6.LocateNode(BiTree T, KeyType e) 递归查找结点，先在左子树中找，随后在右子树中找。

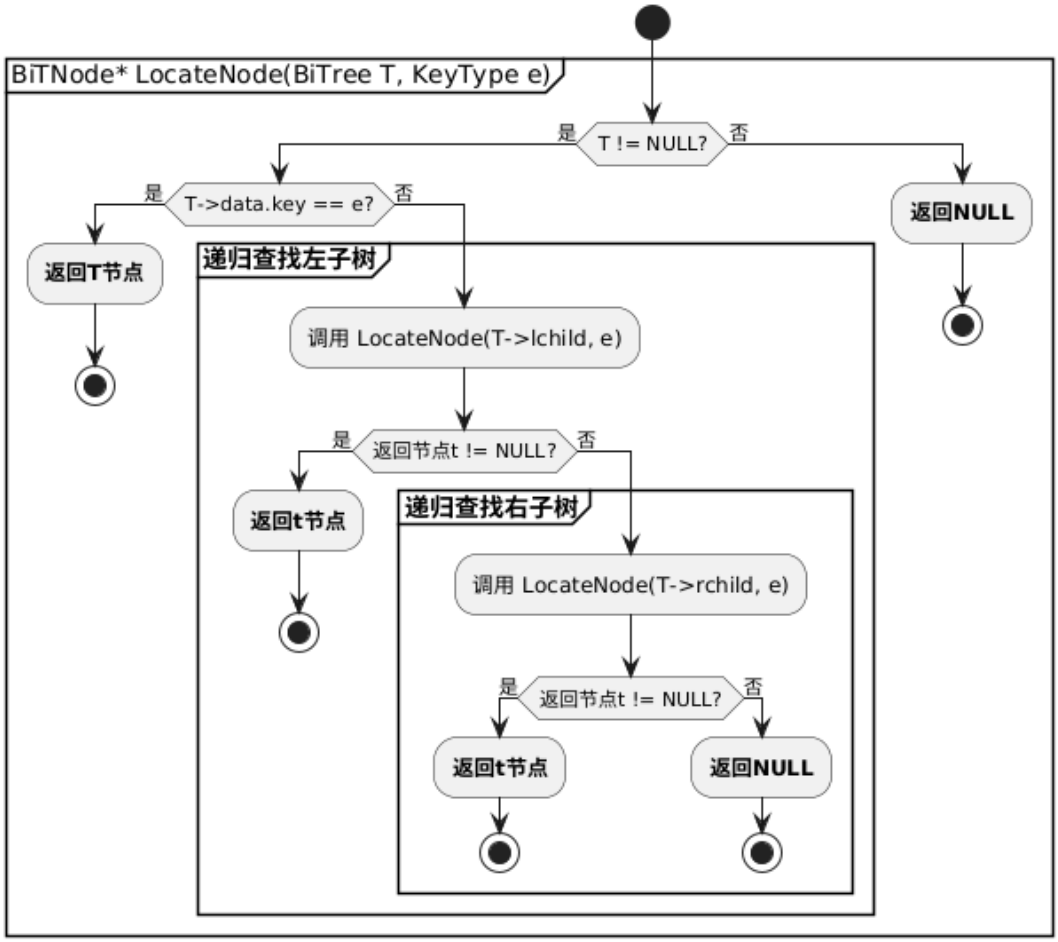

图 2-3　函数 6 的程序流程图

7.Assign(BiTree& T, KeyType e, TElemType value) 先利用 PreOrderTraverseKey(BiTree T, KeyType array1[], int& index) 函数记录已有的结点关键字，以便检查输入的关键字是否重复，进而实现结点赋值。

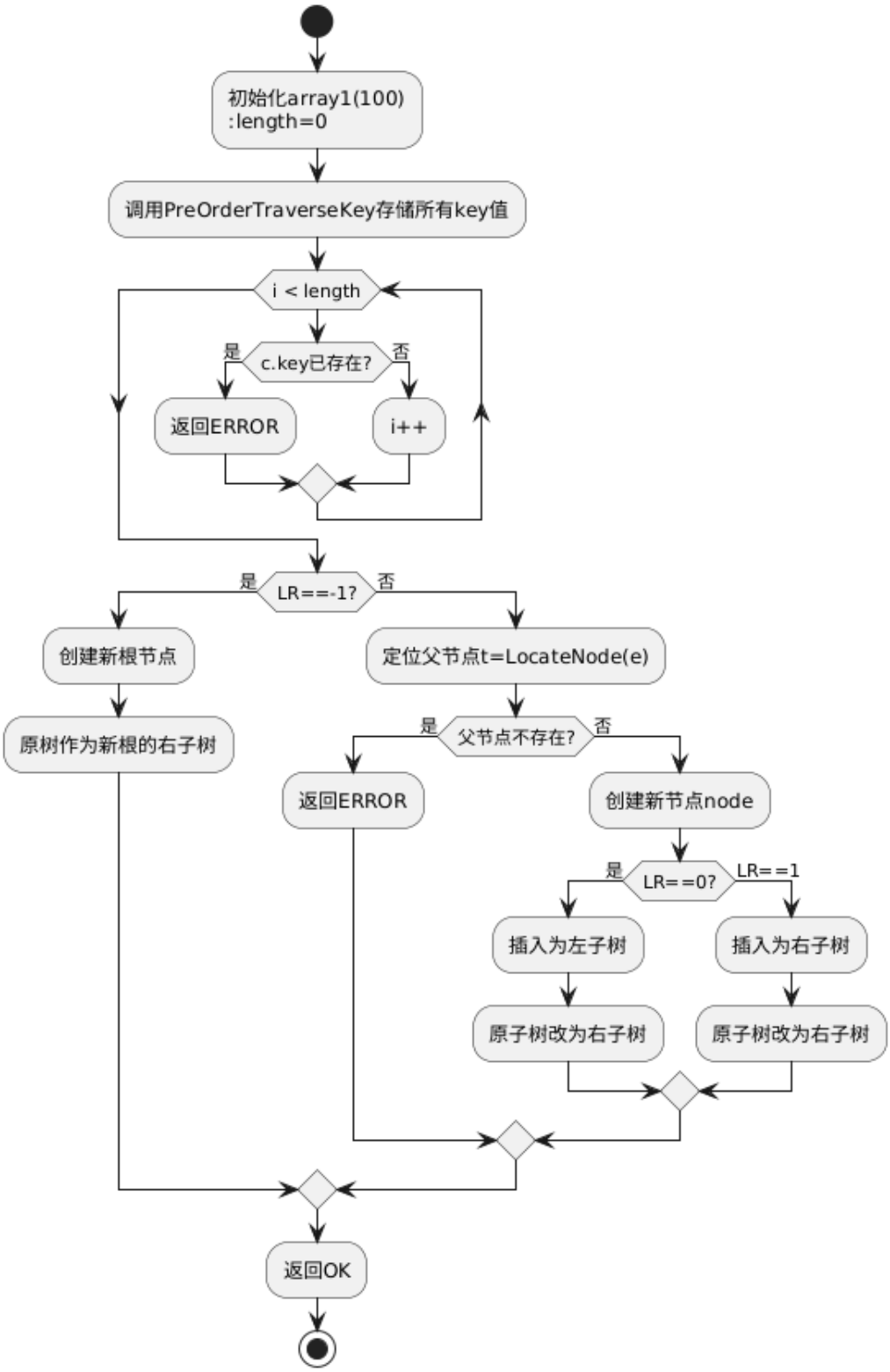

图 2-4　函数 7 的程序流程图

8.GetSibling(BiTree T, KeyType e) 先利用 LocateNodeFather(BiTree T, BiTree&

Father, KeyType e, int& flag) 函数找到所要求的结点的父结点，再获取兄弟结点。

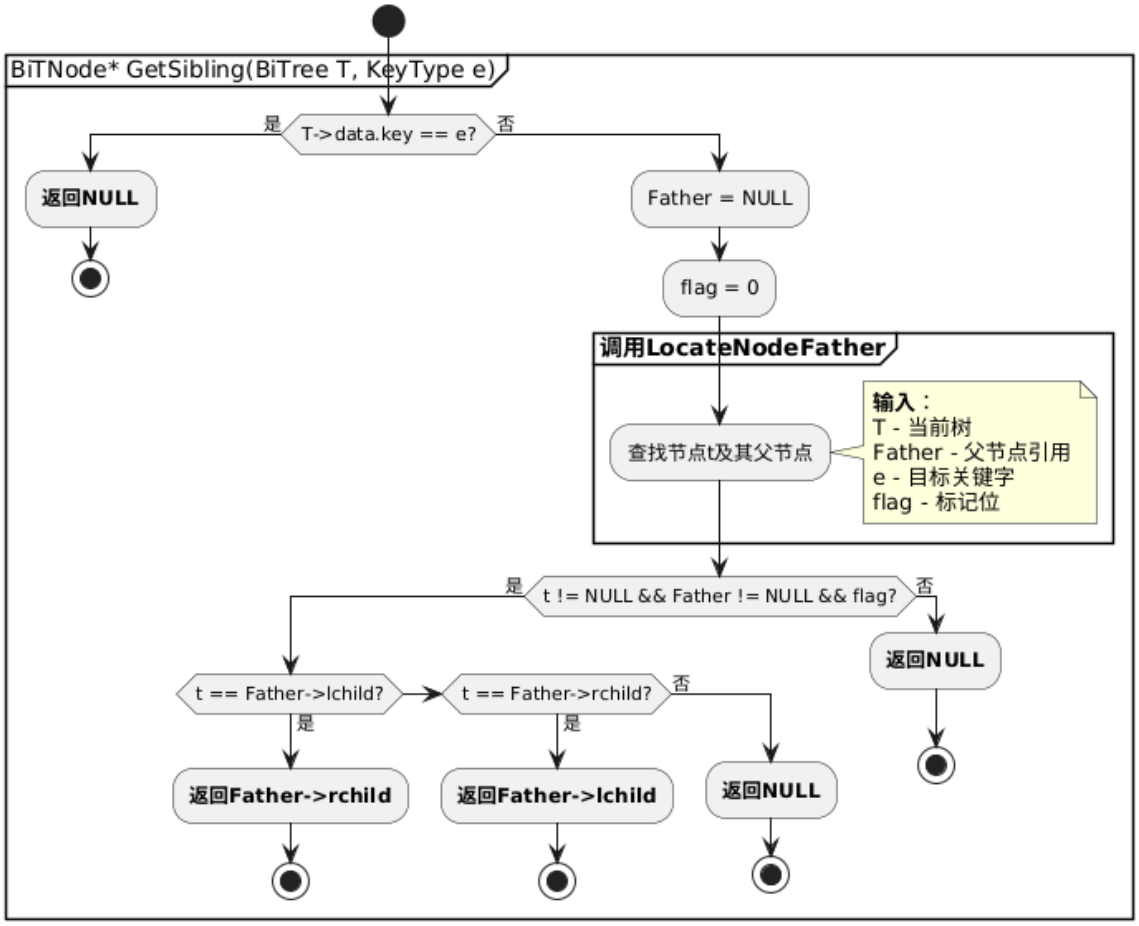

图 2-5　函数 8 的程序流程图

9.InsertNode(BiTree& T, KeyType e, int LR, TElemType c) 同样先利用 PreOrderTraverseKey(BiTree T, KeyType array1[], int& index) 函数记录已有的结点关键字，以便检查输入的关键字是否重复，再根据 LR 值调整插入结点位置。

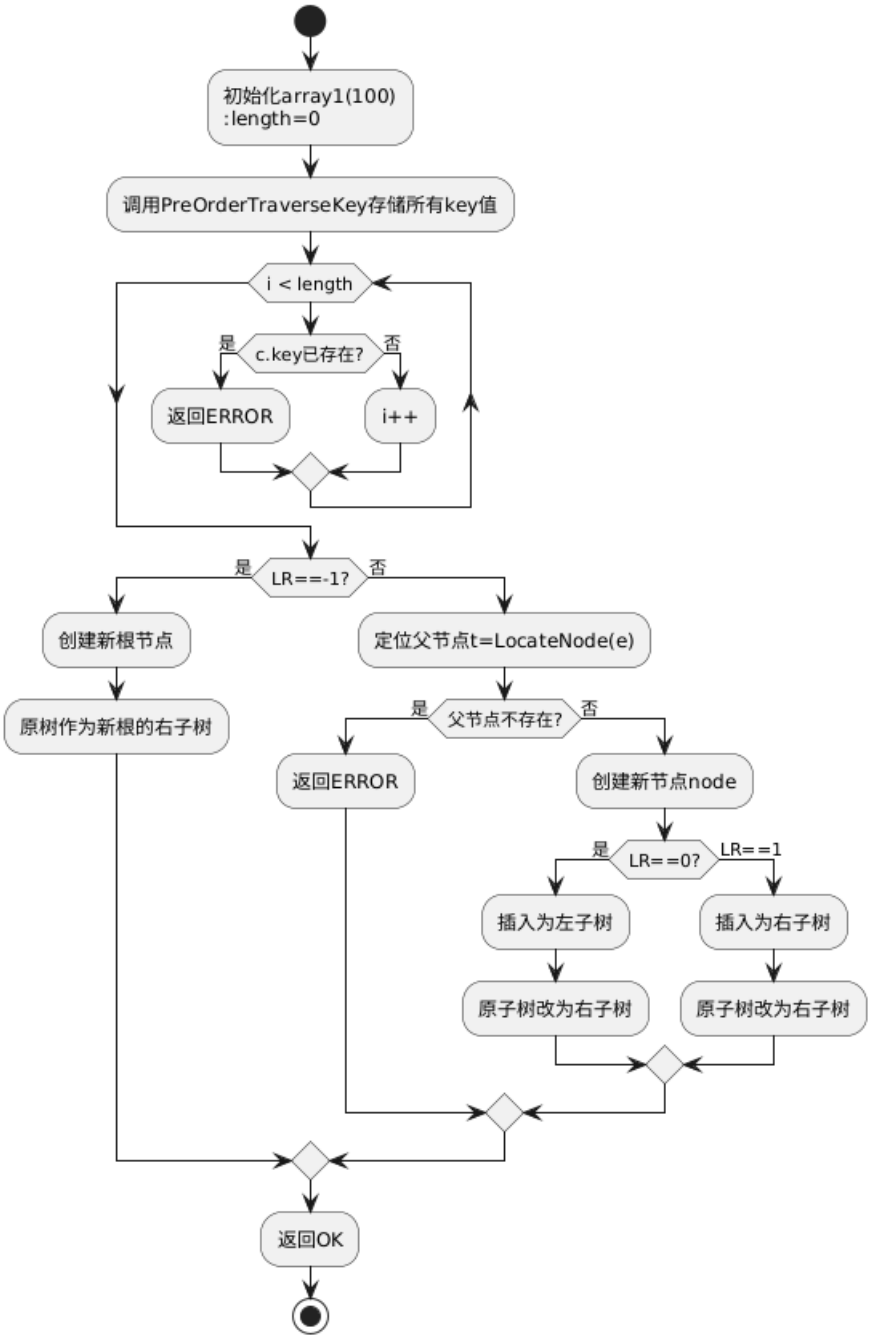

图 2-6　函数 9 的程序流程图

10.DeleteNode(BiTree& T, KeyType e) 先统计结点的结点度，再利用 LocateNodeFather(BiTree T, BiTree& Father, KeyType e, int& flag) 函数找到父结点，最后删除结点并调整剩余结点。

11.PreOrderTraverse(BiTree T, void (*visit)(BiTree)) 先序遍历二叉树。利用栈数据结构，每遍历一个结点就把它的左孩子和右孩子存进栈里（如果存在的话），

不断出栈的过程中实现先序遍历二叉树。

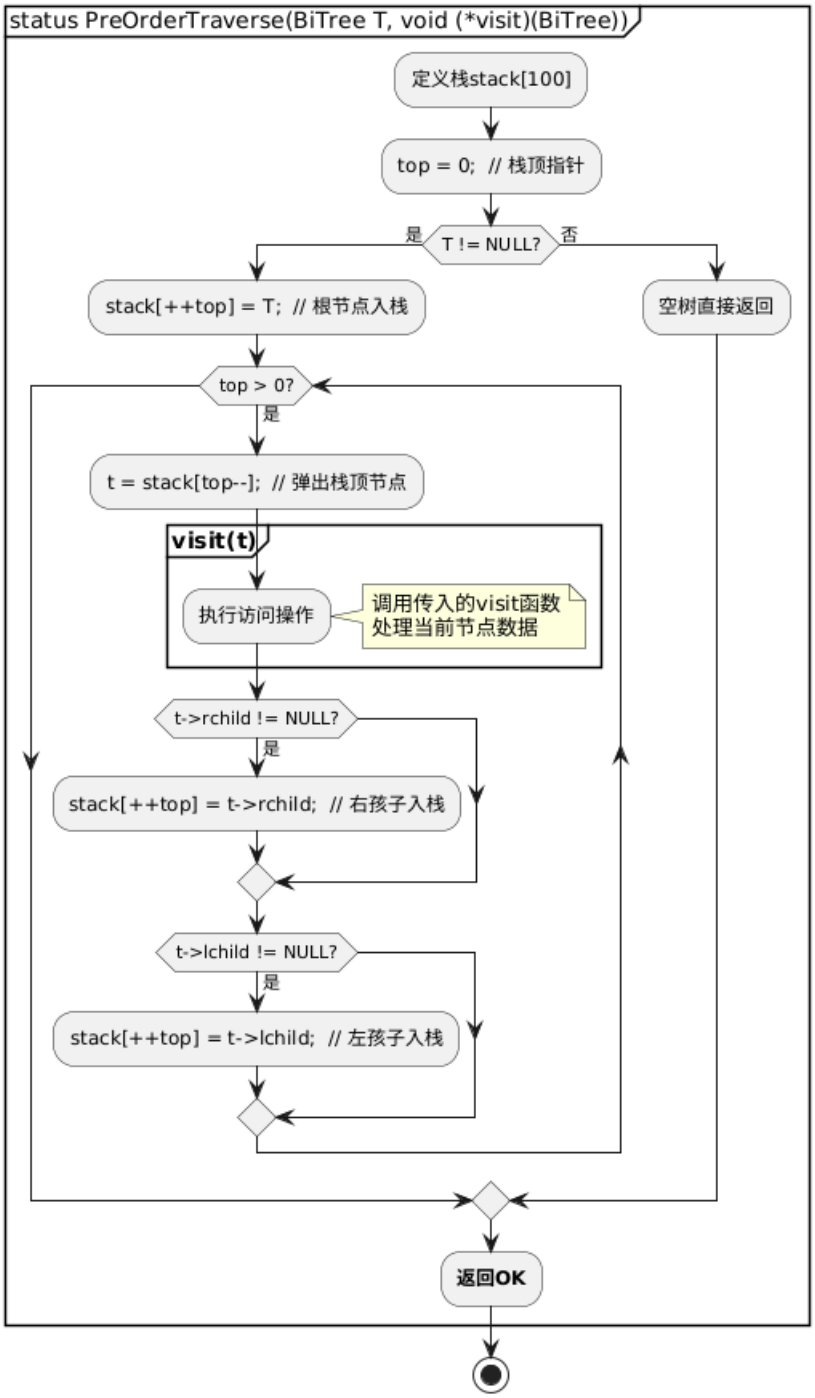

图 2-7　函数 11 的程序流程图

12.InOrderTraverse(BiTree T, void (*visit)(BiTree)) 中序遍历二叉树。利用栈数据结构，每遍历一个结点就把它的所有左孩子存进栈里，对出栈的结点的右孩子进行同样操作，最终实现中序遍历二叉树。

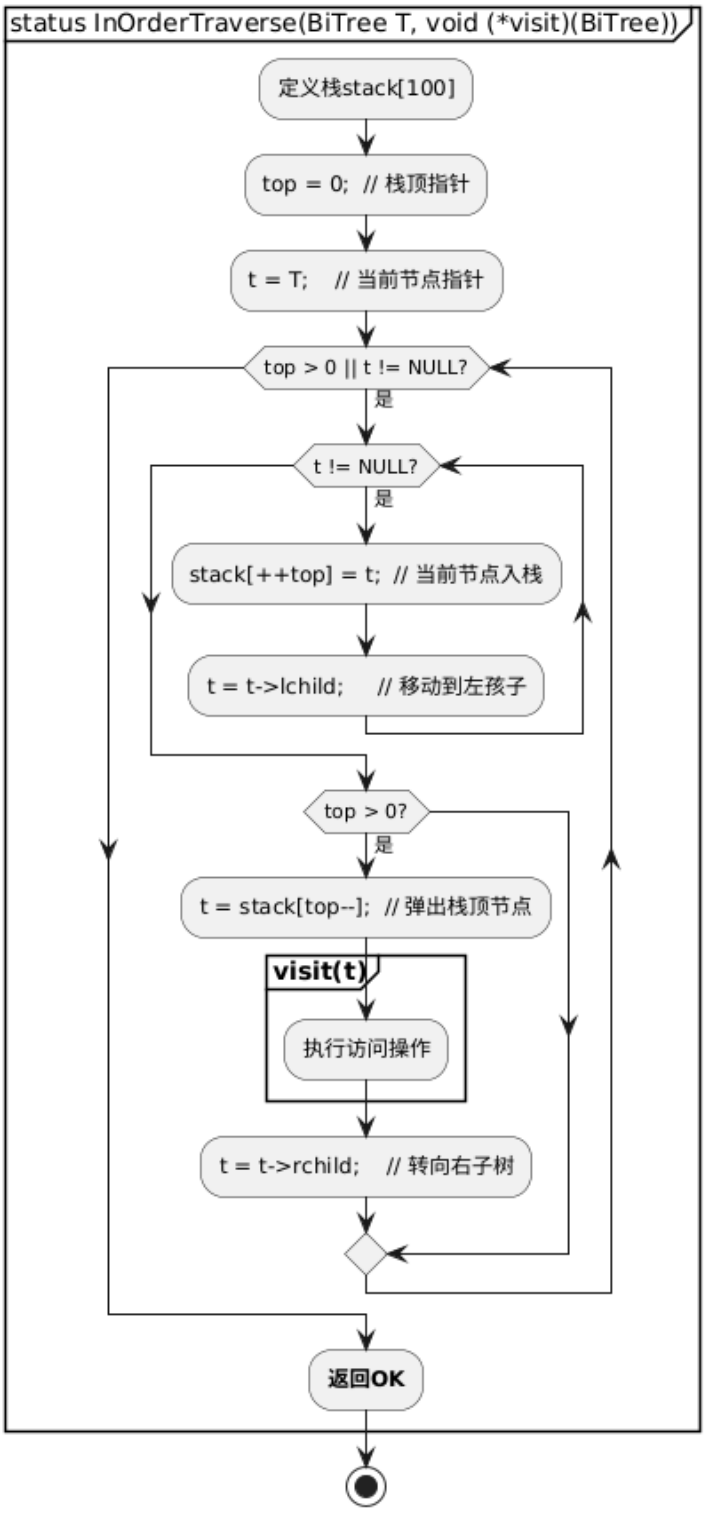

图 2-8　函数 12 的程序流程图

13.PostOrderTraverse(BiTree T, void (*visit)(BiTree)) 后序遍历二叉树。同样利用栈数据结构，每遍历一个结点就把它的所有左孩子存进栈里，同时记录上一个

遍历的结点，当栈顶的结点的右孩子不是上一个遍历的结点时（即出栈结点右孩子还未遍历），重复入栈操作，最终实现后序遍历二叉树。

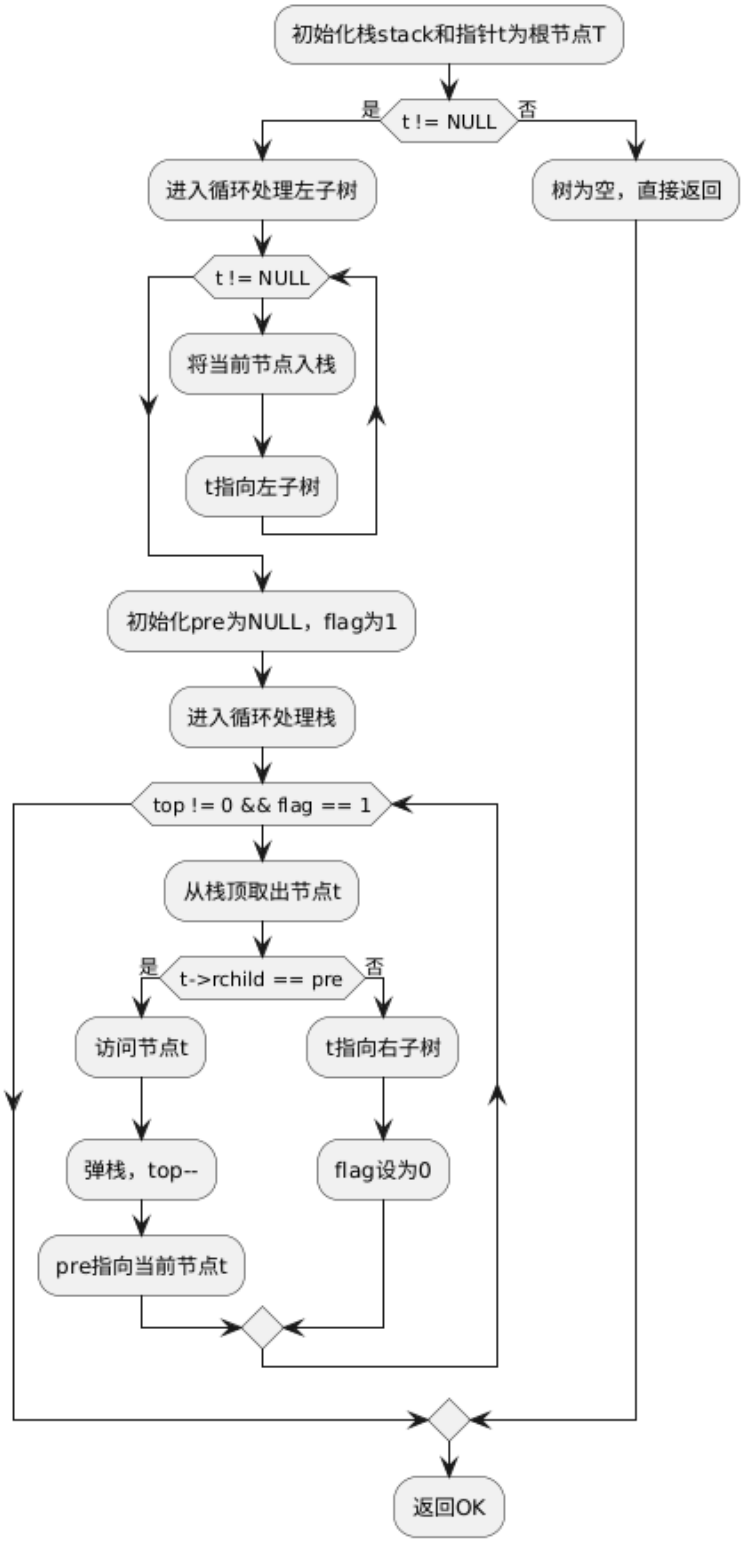

图 2-9　函数 13 的程序流程图

14.LevelOrderTraverse(BiTree T, void (*visit)(BiTree)) 按层遍历二叉树。使用一个二维数组记录每层的结点，遍历每一个结点的同时将其左右孩子存进下一层数组中，直到某一层没有结点则结束循环，实现按层遍历二叉树。

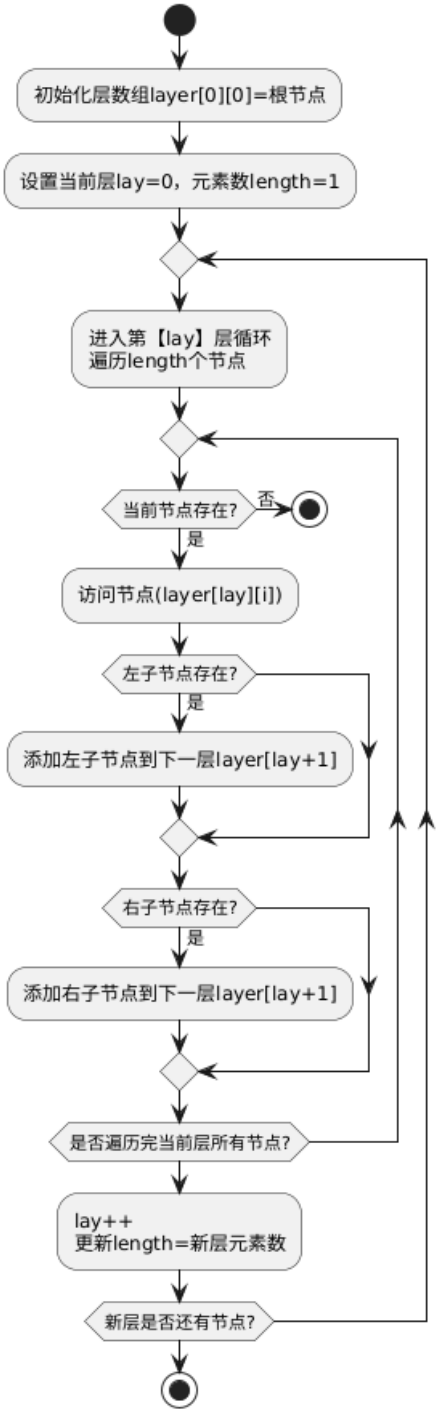

图 2-10　函数 14 的程序流程图

15.MaxPathSum(BiTree T) 通过递归的算法从左右孩子中比较取得最大路径

和，最终得到根节点到叶子结点的最大路径和。

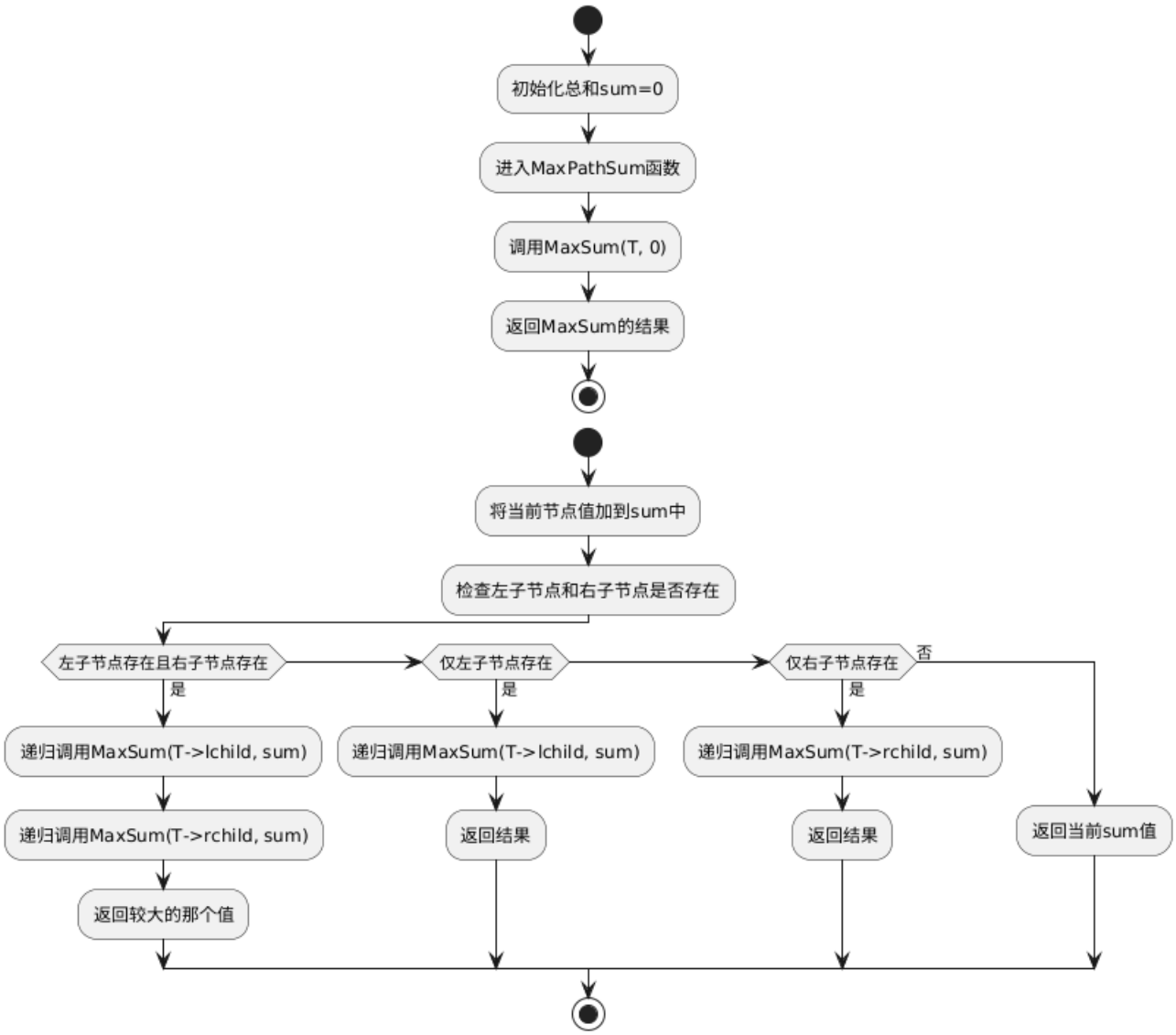

图 2-11　函数 15 的程序流程图

16.LowestCommonAncestor(BiTree T, KeyType e1, KeyType e2) 先通过 LocateNodeDepth(BiTree T, KeyType e, int depth) 函数得到两结点的深度，再由两个深度的大小关系进行父结点查找使其中一个深度降低，循环直到深度相同时且结点关键字一样则找到最近公共父结点。

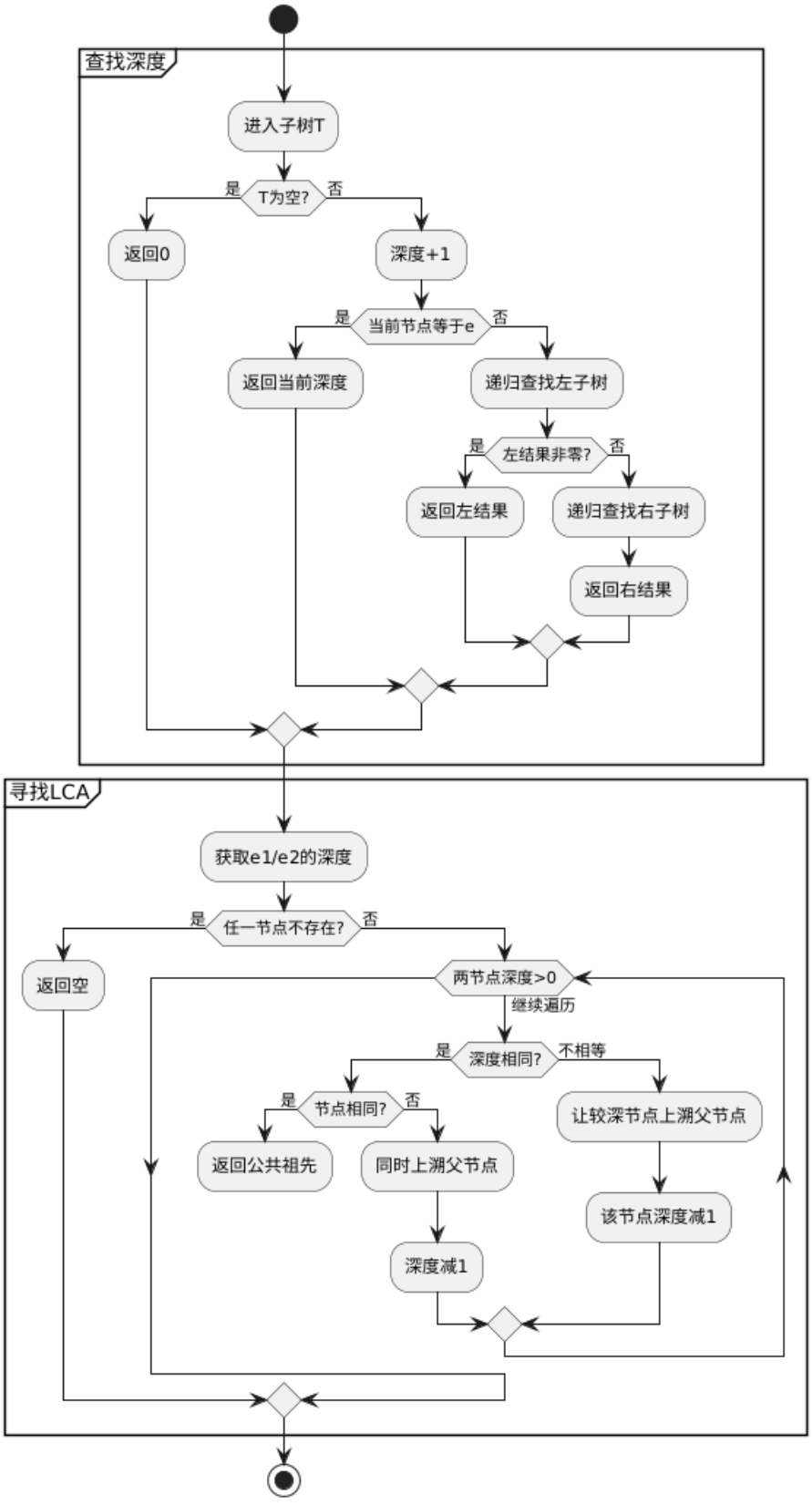

图 2-12　函数 16 的程序流程图

17.InvertTree(BiTree T) 通过递归交换左右子树结点实现翻转二叉树。

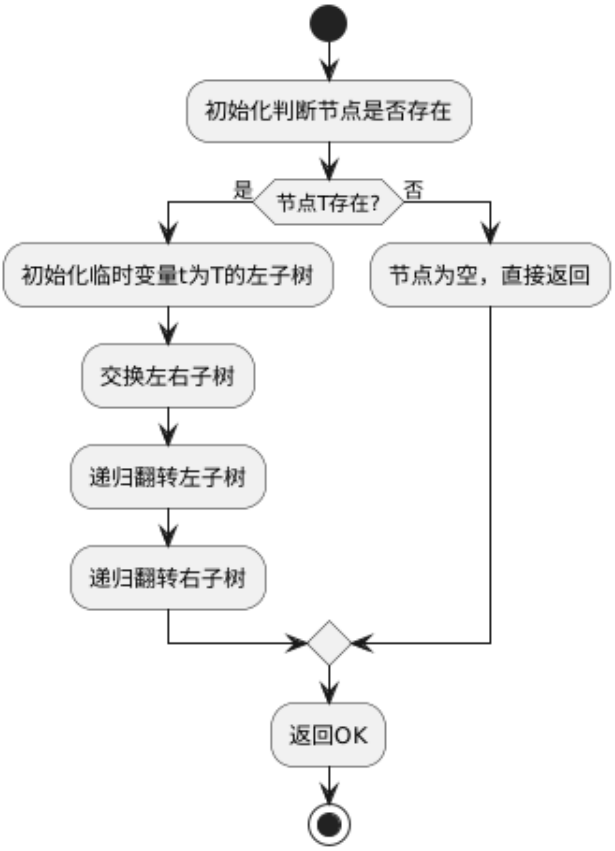

图 2-13　函数 17 的程序流程图

18.SaveBiTree(BiTree T, char FileName[]) 如果二叉树 T 存在，将二叉树 T 的的元素写到 FileName 文件中。

19.LoadBiTree(BiTree& T, char FileName[]) 如果二叉树 T 不存在，将 FileName 文件中的数据读入到二叉树 T 中。

20.AddList(LISTS& Lists, BiTree L, char ListName[]) 在多线性表管理 Lists 中增加一个名称为 ListName 的二叉树。

21.RemoveList(LISTS& Lists, char ListName[]) 在多线性表管理 Lists 中删除一个名称为 ListName 的二叉树。

22.LocateList(LISTS Lists, char ListName[]) 在多线性表管理 Lists 中查找一个名称为 ListName 的二叉树，成功返回逻辑序号，否则返回 0。

23.ShowList(LISTS Lists) 显示多线性表管理 Lists 中的所有二叉树的名字。

2.4 系统测试

1.CreateBiTree, Assign, InsertNode, DeleteNode, LowestCommonAncestor, SaveBiTree, LoadBiTree, RemoveTree 成功返回 OK，失败返回 ERROR。

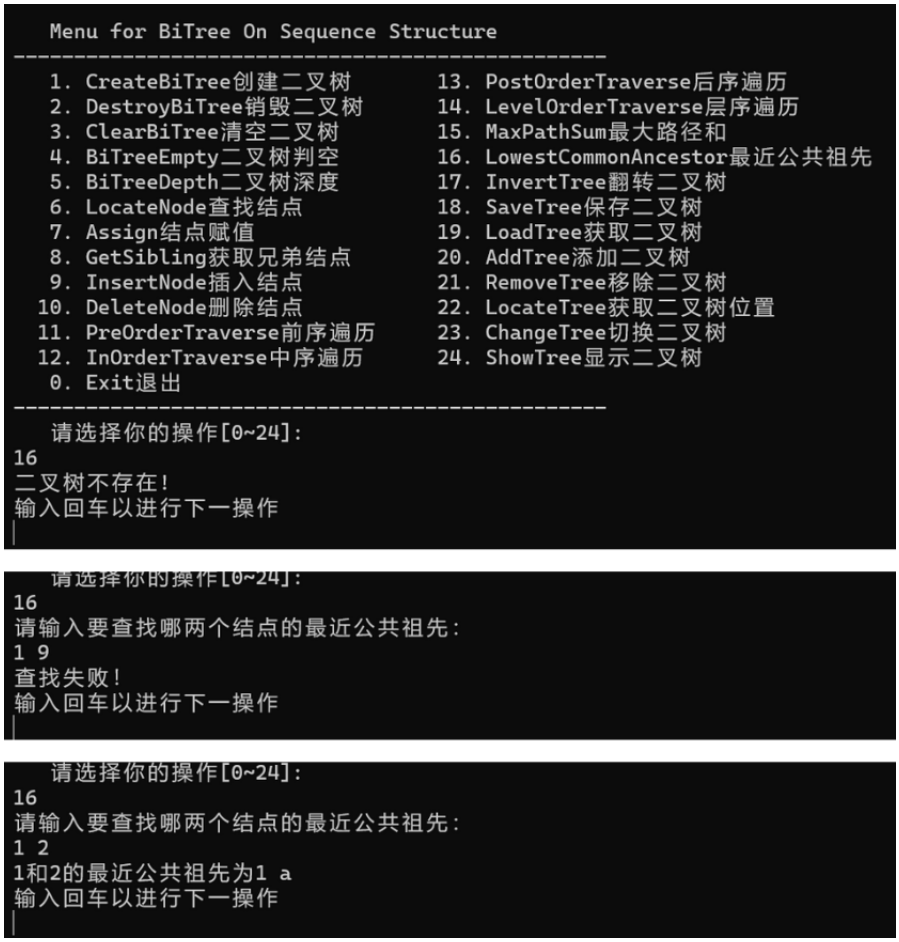

图 2-14　例：功能 16 的测试结果

2.BiTreeEmpty 如果二叉树为空返回 TRUE, 不为空返回 FALSE。

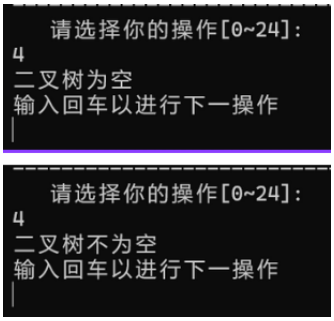

图 2-15　功能 4 的测试结果

3.BiTreeDepth 如果二叉树为空返回 0，否则返回二叉树的深度。

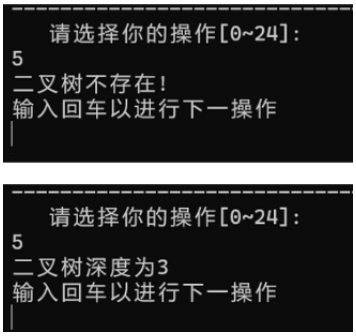

图 2-16　功能 5 的测试结果

4.LocateNode, GetSibling 查找到指定二叉树结点返回指向结点的指针，查找不到则返回 NULL。

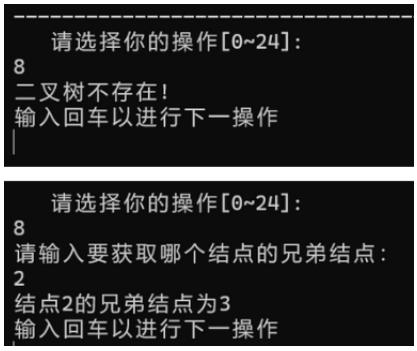

图 2-17　例：功能 8 的测试结果

5.MaxPathSum 返回二叉树的最大路径和。

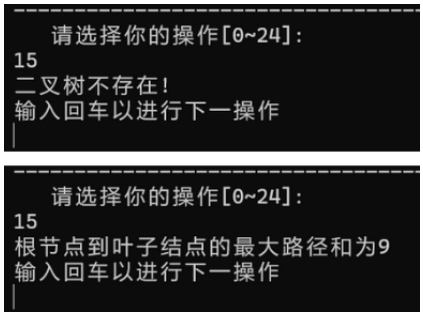

图 2-18　功能 15 的测试结果

6.LocateTree 成功返回逻辑序号，否则返回 0。

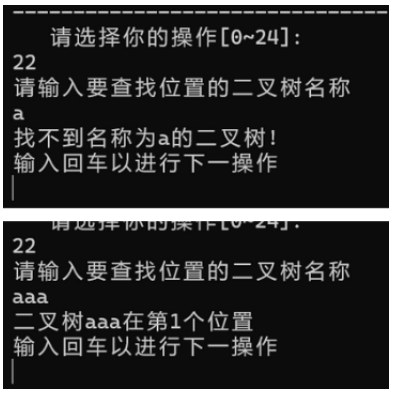

图 2-19　例：功能 22 的测试结果

2.5　实验小结

在本节数据结构实验中，我通过代码实践，成功完成了二叉树的创建、遍历（前序、中序、后序、层序）以及节点插入等核心功能。基于二叉链表的存储结构，每个节点包含数据域、左指针域和右指针域，这种结构能够灵活地表示二叉树的层次关系，有效避免了顺序存储可能产生的空间浪费问题。

在实验过程中，我深刻体会到递归算法在二叉树遍历中的简洁与高效，通过递归调用实现不同顺序的遍历，让二叉树的节点数据得以有序输出。然而，在实现复杂功能如节点插入时，遇到了指针指向混乱和二叉树结构破坏的问题。经过反复调试与查阅资料，我逐步掌握了如何正确处理指针的指向关系，确保二叉树的逻辑结构完整。

通过本次实验，我不但巩固了二叉树和链表的理论知识，而且提升了动手调试和解决实际问题的能力，深刻认识到数据结构设计对算法实现效率和准确性的重要性。后续，我计划进一步探索二叉树的更多应用场景，如哈夫曼树、二叉搜索树的实现，深化对数据结构的理解与应用。

参考文献

[1] 袁凌等. 数据结构—从概念到算法（C 语言版 | 微课版）[M]. 北京 : 人民邮电出版社, 2023.

[2] 严蔚敏, 吴伟民. 数据结构（C 语言版）: 清华大学计算机系列教材 [M]. 北京 : 清华大学出版社, 2007.

[3] 严蔚敏, 吴伟民, 米宁. 数据结构题集（C 语言版）: 清华大学计算机系列教材 [M]. 北京 : 清华大学出版社, 2007.