

华中科技大学

课程实验报告

课程名称： 数据结构实验

专业班级： 计算机科学与技术 202011 班

学 号： U202015084

姓 名： 张文浩

指导教师： 许贵平

报告日期： 2021 年 6 月 17 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.....	3
1.1 问题描述.....	3
1.1.1 实验目的.....	3
1.1.2 实验要求.....	3
1.2 系统设计.....	3
1.2.1 整体系统结构设计.....	3
1.2.2 数据结构设计.....	4
1.2.3 有关类型和常量定义.....	6
1.3 系统实现.....	7
1.4 系统测试.....	14
1.5 实验小结.....	24
2 基于顺序链式结构的线性表实现.....	25
2.1 问题描述.....	25
2.1.1 实验目的.....	25
2.1.2 实验要求.....	25
2.2 系统设计.....	25
2.2.1 整体系统结构设计.....	25
2.2.2 数据结构设计.....	27
2.2.3 有关类型和常量定义.....	28
2.3 系统实现.....	29
2.4 系统测试.....	37
2.5 实验小结.....	46
3 基于二叉链表的二叉树实现.....	47
3.1 问题描述.....	47
3.1.1 实验目的.....	47
3.1.2 实验要求.....	47
3.2.2 系统设计.....	47
3.2.1 整体系统结构设计.....	47

3.2.2 数据结构设计.....	49
3.2.3 有关类型和常量定义.....	51
3.3 系统实现.....	51
3.4 系统测试.....	66
3.5 实验小结.....	73
4 基于邻接表的图实现.....	74
4.1 问题描述.....	74
4.1.1 实验目的.....	74
4.1.2 实验要求.....	74
4.2.2 系统设计.....	74
4.2.1 整体系统结构设计.....	74
4.2.2 数据结构设计.....	76
4.2.3 有关类型和常量定义.....	78
4.3 系统实现.....	78
4.4 系统测试.....	87
4.5 实验小结.....	93
参考文献.....	94

1 基于顺序储存结构的线性表实现

1.1 问题描述

构造顺序表，呈现一个简易菜单的功能演示系统，该演示系统可选择实现多个线性表管理以及单线性表的操作。

1.1.1 实验目的

- (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, 熟练掌握线性表的基本运算的实现。

1.1.2 实验要求

需要在主程序中完成函数调用以及所需实参值和函数执行结果的输出。定义线性表及线性表集合的初始化、销毁、清空、判空、求表长和获得元素等函数，并给出适当的操作提示，并且可选择以文件的形式进行存储和加载。

1.2 系统设计

1.2.1 整体系统结构设计

本演示系统可以实现通过操作菜单对多线性表进行管理，并对单个线性表进行操作，也支持通过文件的写入与读取。

多线性表管理菜单（线性表集合菜单/一级菜单）可供选择的操作有：初始化线性表集合、添加一个线性表、删除特定名称的线性表、查找特定名称的线性表、清空线性表集合、线性表集合判空、求线性表集合中线性表的个数、打印所有线性表的序号及名称、操作特定序号的线性表。

多线性表管理菜单中的“操作特定序号的线性表”可以调出单线性表操作菜单（二级菜单），可供选择的操作有：初始化线性表、销毁线性表、清空线性表、线性表判空、求线性表长度、取出特定位置的元素、定位某元素、求某元素的前驱、求某元素的后继、在特定位置插入元素、删除特定位置的元素、打印线性表中的所有元素、将线性表的数据写入文件、通过键盘输入或者读取文件数据向线性表中一次性输入若干元素。整体系统结构设计示意图如下：

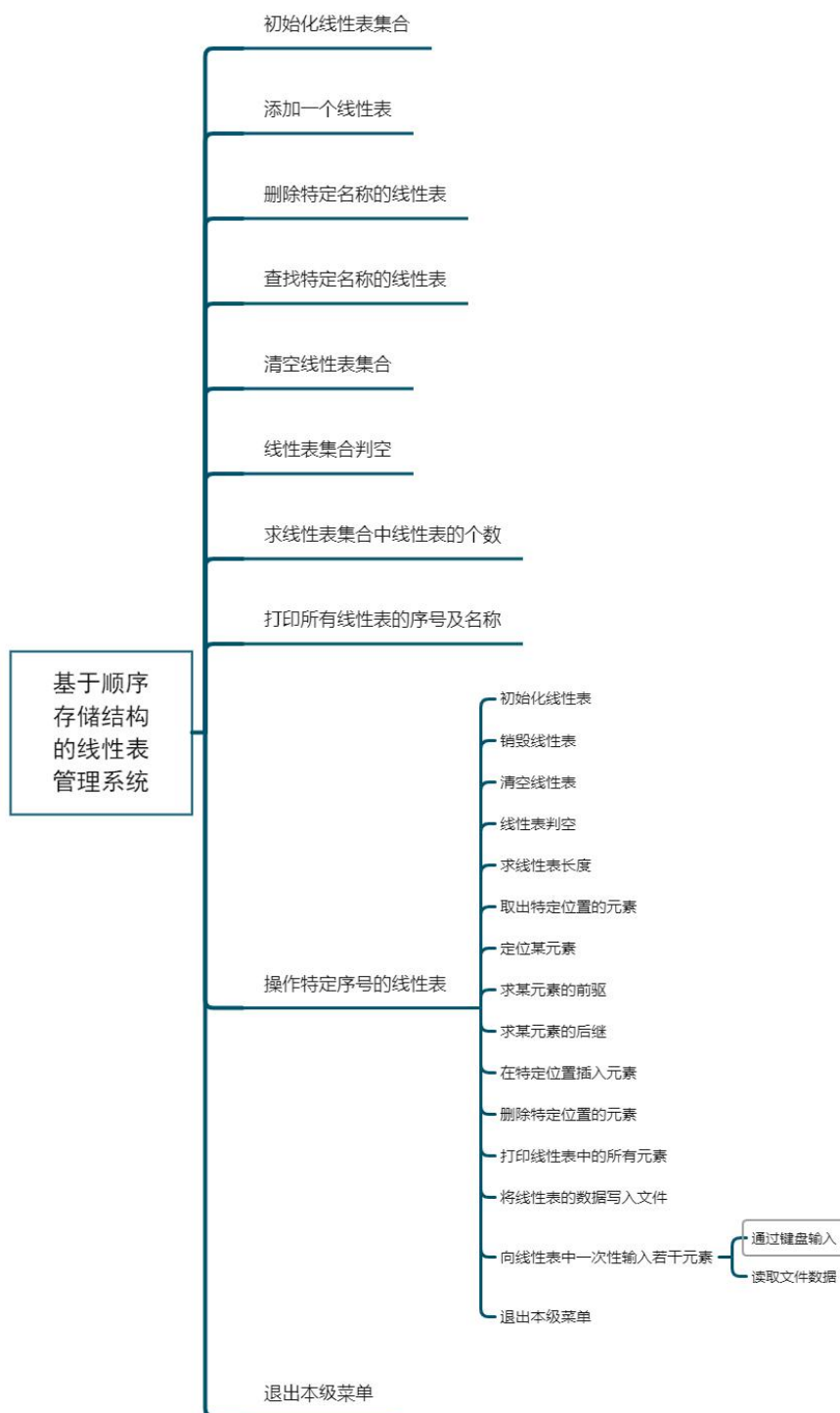


图 1-1 整体系统结构设计

1.2.2 数据结构设计

本演示系统设计了两个物理结构，分别是线性表和线性表集合，两者均以结构形式定义。

线性表结构中包括数据数组指针、线性表长度、当前线性表最大长度。线性表集合结构中包括线性表数组（每个数组元素都包括线性表以及其名称）和线性表集合长度。

结构定义如下：

```
typedef struct { //顺序表（顺序结构）的定义
    ElemType* elem;
    int length;
    int listsize;
} SqList;

typedef struct { //线性表的集合类型定义
    struct {
        char name[30];
        SqList L;
    } elem[10];
    int length;
} LISTS;
```

图 1-2 物理结构设计

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， $1 \leq i \leq \text{ListLength}(L)$ ；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置，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， $1 \leq i \leq \text{ListLength}(L)+1$ ；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， $1 \leq i \leq \text{ListLength}(L)$ ；操作结果：删除 `L` 的第 `i` 个数据元素，用 `e` 返回其值。

(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()`。

1.2.3 有关常量和类型定义

数据元素类型的定义：

```
typedef int status;
```

```
typedef int ElemType;
```

有关常量的定义：

```
#define TRUE 1
```

```
#define FALSE 0
```

```
#define OK 1
```

```
#define ERROR 0
```

```
#define INFEASTABLE -1
```

```
#define OVERFLOW -2
```

```
#define LIST_INIT_SIZE 100
```

```
#define LISTINCREMENT 10
```

1.3 系统实现

本演示系统在 Windows 环境下，使用 Visual Studio 2019 完成。演示系统涉及的部分主要函数如下，其中（1）-（4）为对线性表集合的操作，（5）-（12）为对单线性表的操作。

（1）向线性表集合中添加一个空线性表（这里的实现与头歌的通关代码不同，这里将对添加的线性表的内存分配，长度和最大长度的初始化均移入对单线性表的操作中进行）

函数名称：status AddList(LISTS *Lists, char ListName[])

输入：线性表集合指针以及需要添加的线性表的名称

输出：函数执行状态

思想：在 Lists 中加入一个名称为 ListName 的线性表，成功则返回 OK

操作：

① 将 ListName 数组拷贝到线性表集合中线性表数组的最后一个元素之后的元素的 name 成员变量中

② 将新添加的线性表的 elem 指针置空

③ 线性表集合长度加一

④ 结束，返回 OK

时间复杂度：O(1)

空间复杂度：O(1)

（2）删除线性表集合中指定名称的线性表

函数名称：status RemoveList(LISTS *Lists, char ListName[])

输入：线性表集合指针以及需要删除的线性表名称

输出：函数执行状态

思想：删除指定名称的线性表，删除成功则返回 OK，否则返回 ERROR

操作：

① 定义 $i=0$ ，当 i 小于线性表集合长度时，执行下列循环：

a. 比较第 i 个线性表的名称和所给名称

b. 当两者相同时，将第 i 个线性表的名称指针的首元素置为 ' $\backslash 0$ '，并使 $elem$ 指针分配的空间用 $free$ 函数收回，并使 $elem$ 指针指向 $NULL$ ，将线性表数组中该线性表后面的线性表依次前移，线性表集合的长度减一，返回 OK

② 若循环结束后函数没有返回值，说明集合中没有该名称的线性表，返回 $ERROR$

时间复杂度： $O(n)$ （设线性表集合长度为 n ）

空间复杂度： $O(1)$

(3) 查找指定名称的线性表在集合中的位置

函数名称：`status LocateList(LISTS Lists, char ListName[])`

输入：线性表集合以及需要查找的线性表名称

输出：该线性表的序号或者函数执行状态

思想：在集合中寻找指定名称的线性表，成功则返回其逻辑序号，否则返回 $ERROR$

操作：

① 定义 $i=0$ ，当 i 小于线性表集合长度时，执行下列循环：

a. 比较第 i 个线性表的名称与所给名称

b. 如果相同，返回 $i+1$ ，否则继续循环

② 如果循环执行完后函数没有返回值，说明集合中没有该名称的线性表，返回 $ERROR$

时间复杂度： $O(n)$ （设线性表长度为 n ，此处为最坏情况）

空间复杂度： $O(1)$

(4) 清空线性表集合

函数名称：`status ClearLists(LISTS *Lists)`

输入：线性表集合指针

输出：函数执行状态

思想：将线性表集合的长度变成 0，返回 OK

操作：将 Lists 的 length 赋值为 0，返回 OK

时间复杂度：O(1)

空间复杂度：O(1)

(5) 销毁线性表

函数名称：status DestroyList(SqList *L)

输入：线性表指针

输出：函数执行状态

思想：如果线性表已经初始化，销毁线性表，释放数据元素的空间，返回 OK，否则返回 INFEASIBLE

操作：如果线性表未初始化（elem 指向 NULL），返回 INFEASIBLE，否则释放 elem 的空间，将 elem 指向 NULL，返回 OK

时间复杂度：O(1)

空间复杂度：O(1)

(6) 清空线性表

函数名称：status ClearList(SqList *L)

输入：线性表指针

输出：函数执行状态

思想：如果线性表已经初始化，将其长度赋值为 0，以此来达到删除所有元素的目的

操作：如果线性表已经初始化，将其长度赋值为 0，返回 OK，否则返回 INFEASIBLE

时间复杂度：O(1)

空间复杂度：O(1)

(7) 获取线性表特定位置的元素

函数名称：status GetElem(SqList L, int i, ElemType *e)

输入：线性表，需要获取的元素的位置，存放该元素的变量指针

输出：函数执行状态

思想：如果线性表已经初始化，获取线性表的第 i 个元素，保存在 e 中，返回 OK，如果 i 不合理，返回 ERROR；如果线性表未初始化，返回 INFEASIBLE

操作：如果线性表已经初始化，将线性表的第 i 个元素，保存在 e 中，并返回 OK，如果 i 不合理（ $i < 1$ 或者 $i > \text{线性表长度}$ ），返回 ERROR；如果线性表未初始化（ elem 指向 NULL），返回 INFEASIBLE

时间复杂度： $O(1)$

空间复杂度： $O(1)$

(8) 确定某元素在线性表中的位置

函数名称：status LocateElem(Sqlist L, ElemType e)

输入：线性表，某元素

输出：元素位置或者函数执行状态

思想：若线性表已经初始化，查找元素位置并返回序号，查找不成功时返回相应的函数执行状态

操作：

①如果线性表未初始化，返回 INFEASIBLE

②线性表已经初始化，定义 $i=0$ ，执行下列循环：

当第 i 个元素不等于 e 且 i 小于线性表长度时， i 自增。

③如果循环结束后 i 等于线性表长度，说明线性表中没有该元素，返回 ERROR，否则返回 $i+1$

时间复杂度： $O(n)$

空间复杂度： $O(1)$

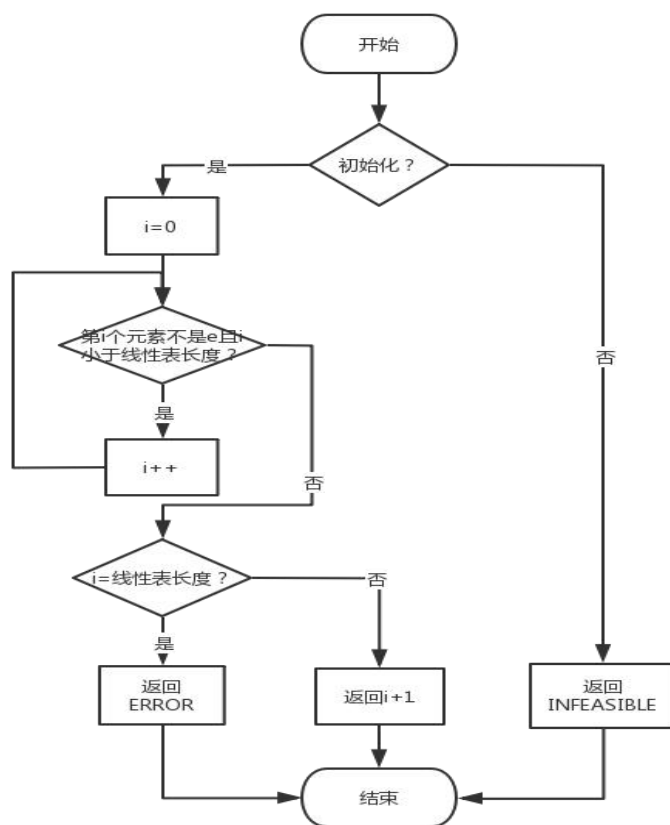


图 1-3 LocateElem 流程图

(9) 在某位置插入某元素

函数名称: `status ListInsert (SqList *L, int i, ElemType e)`

输入: 线性表指针, 插入位置, 插入元素

输出: 函数执行状态

思想: 若线性表已经初始化, 且插入位置正确, 则插入元素, 返回 OK, 否则返回相应的函数执行状态

操作:

①如果线性表未初始化, 返回 INFEASIBLE

②线性表已经初始化:

a. 如果插入位置不正确, 返回 ERROR

b. 如果线性表长度大于等于最大长度, 使用 `realloc` 扩容; 如果长度大于 0, 将插入位置及之后的元素后移一位, 将插入元素放在 `i` 位置, 长度自增, 返

回 OK

时间复杂度： $O(n)$

空间复杂度： $O(1)$

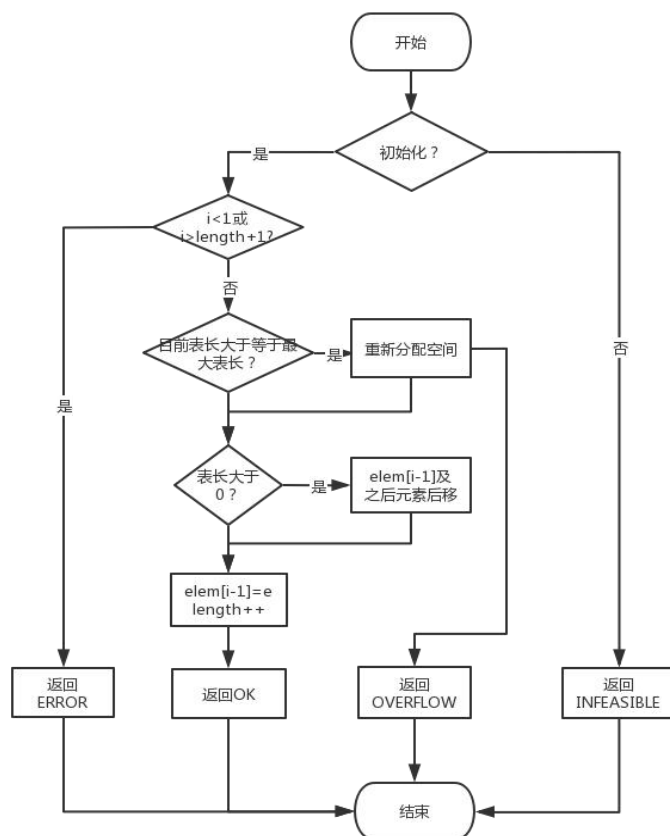


图 1-4 ListInsert 流程图

(10) 删除某位置元素

函数名称：status ListDelete (SqList *L, int i, ElemType *e)

输入：线性表指针，删除位置，储存删除元素的变量指针

输出：函数执行状态

思想：若线性表已经初始化且删除位置正确，则删除对应位置元素，返回 OK，否则返回相应的函数执行状态

操作：

①如果线性表未初始化，返回 INFEASIBLE

②线性表未初始化：

a. 若删除位置不正确，返回 ERROR

b. 否则用 e 保存该位置元素，后面元素前移，长度减一，返回 OK

时间复杂度：O(n)

空间复杂度：O(1)

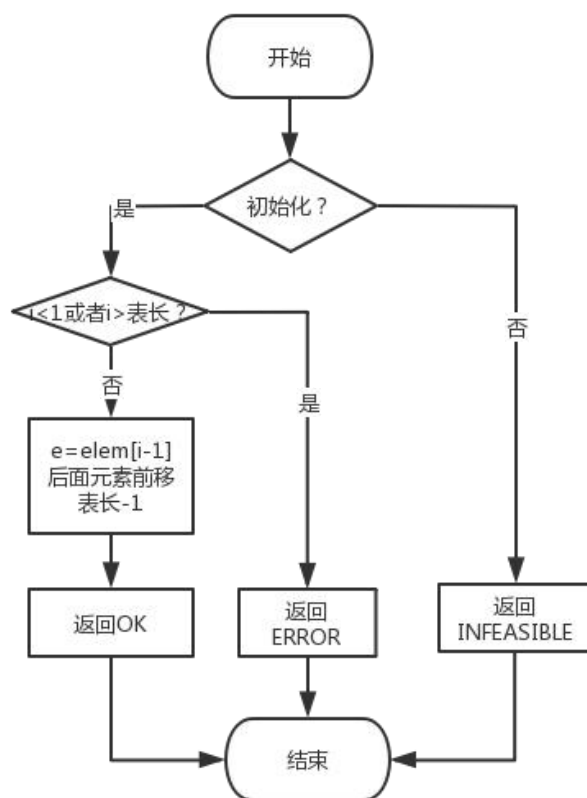


图 1-5 ListDelete 流程图

(11)将线性表的元素写入某文件

函数名称：status SaveList (SqList L, char FileName[])

输入：线性表，文件名称

输出：函数执行状态

思路：如果线性表已经初始化，将元素写入文件，否则返回对应的函数执行状态

操作：

①如果线性表未初始化，返回 INFEASIBLE

②线性表已经初始化，定义文件指针并指向打开文件

a. 若打开失败，返回 ERROR

b. 打开成功，将元素依次写入，并关闭文件指针，返回 OK

时间复杂度： $O(n)$

空间复杂度： $O(1)$

(12) 读取某文件中的数据到一空线性表

函数名称：status LoadList (SqList *L, char FileName[])

输入：线性表指针，文件名称

输出：函数执行状态

思路：如果线性表已经初始化，将文件中的数据读入线性表，否则返回相应的函数执行状态

操作：

①如果线性表未初始化，返回 INFEASIBLE

②线性表已经初始化，定义文件指针并指向打开文件

a. 若打开失败，返回 ERROR

b. 打开成功，将元素依次读取，并关闭文件指针，返回 OK

时间复杂度： $O(n)$

空间复杂度： $O(1)$

1.4 系统测试

程序采用较为简单的界面，一级菜单和二级菜单分别如图 1-6, 1-7 所示（在一级菜单中可通过操作 9 进入二级菜单）。本次系统测试选取了针对线性表集合操作的 AddList、RemoveList、LocateList 函数以及针对单线性表操作的 ClearList、ListEmpty、GetElem、LocateElem、PriorElem、NextElem、ListInsert、ListDelete、ListTraverse、SaveList、LoadList 函数。

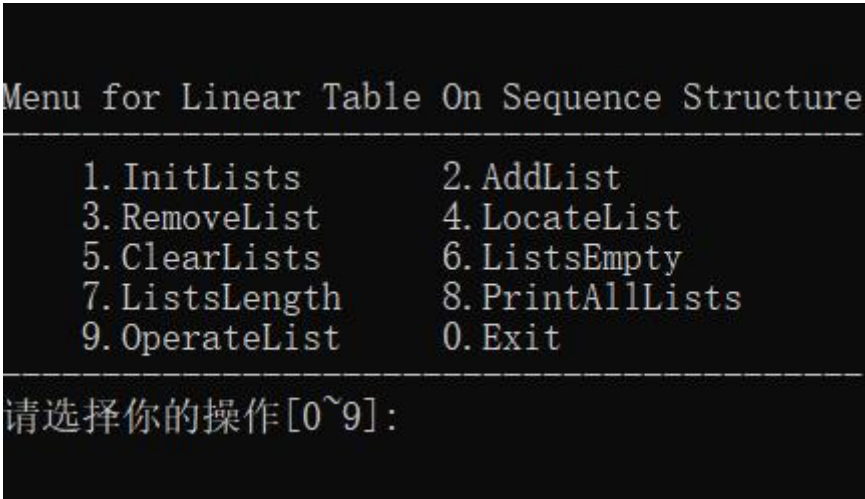


图 1-6 一级菜单示意图

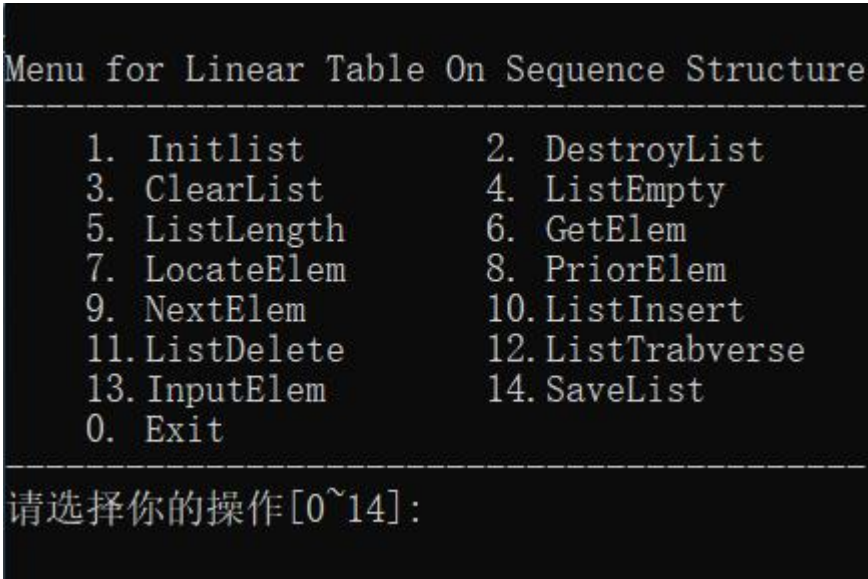


图 1-7 二级菜单示意图

下面是函数测试：

(1) 添加线性表的测试：

测试用例及结果如表 1-1 所示

表 1-1 添加线性表测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (集合中已有 aaa 线性表)	名称: aaa	名称有误！请重新 输入！	<div>请选择你的操作[0~9]:2 请输入需要添加的线性表的名称: aaa 名称重复！请重新输入！</div> <div>请选择你的操作[0~9]:8 所有线性表依次为: 1 aaa</div>

用例 2 (空集合)	名称: bbb	添加成功!	<div> 请选择你的操作[0~9]:2 请输入需要添加的线性表的名称: bbb 添加成功! </div> <div> 请选择你的操作[0~9]:8 所有线性表依次为: 1 bbb </div>
------------	---------	-------	---

综合上述测试,添加线性表功能对于输出名称重复时以及未重复时都可以正确处理,所以添加线性表功能符合实验要求。

(2) 删除线性表的测试:

测试用例及结果如表 1-2 所示

表 1-2 删除线性表测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (集合中已有 aaa 线性表)	名称: aaa	删除成功!	<div> 请选择你的操作[0~9]:3 请输入需要删除的线性表的名称: aaa 删除成功! </div> <div> 请选择你的操作[0~9]:8 该线性表集合中没有元素! </div>
用例 2 (集合中已有 aaa 线性表)	名称: bbb	线性表集合中没有该线性表!	<div> 请选择你的操作[0~9]:3 请输入需要删除的线性表的名称: bbb 线性表集合中没有该线性表! </div> <div> 请选择你的操作[0~9]:8 所有线性表依次为: 1 aaa </div>
用例 3 (空集合)	名称: aaa	线性表集合中没有该线性表!	<div> 请选择你的操作[0~9]:3 请输入需要删除的线性表的名称: aaa 线性表集合中没有该线性表! </div> <div> 请选择你的操作[0~9]:8 该线性表集合中没有元素! </div>

综合上述测试,删除线性表功能对于线性表集合中有或无对应名称的线性表时都可以正确处理,所以删除线性表功能符合实验要求。

(3) 定位线性表的测试

测试用例及结果如表 1-3 所示

表 1-3 定位线性表测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (空集合)	名称: aaa	线性表集合中没有该线性表!	请选择你的操作[0~9]:4 请输入需要定位的线性表的名称: aaa 线性表集合中没有该线性表!
用例 2 (集合中已有 aaa, bbb 线性表)	名称: bbb	该线性表位置是 2	请选择你的操作[0~9]:4 请输入需要定位的线性表的名称: bbb 该线性表的位置为2 请选择你的操作[0~9]:8 所有线性表依次为: 1 aaa 2 bbb
用例 3 (集合中已有 aaa, bbb 线性表)	名称: ccc	线性表集合中没有该线性表!	请选择你的操作[0~9]:8 所有线性表依次为: 1 aaa 2 bbb 请选择你的操作[0~9]:4 请输入需要定位的线性表的名称: ccc 线性表集合中没有该线性表!

综合上述测试,定位线性表功能对于线性表集合中有或无对应名称的线性表时都可以正确处理,所以定位线性表功能符合实验要求

(4) 清空线性表的测试

测试用例及结果如表 1-4 所示

表 1-4 清空线性表测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (空线性表)	空线性表	线性表为空!	请选择你的操作[0~14]:3 线性表为空!
用例 2 (未初始化的线性表)	未初始化的线性表	线性表未初始化!	请选择你的操作[0~14]:3 线性表未初始化!
用例 3 (s={1, 2, 3})	s={1, 2, 3}	清空成功!	请选择你的操作[0~14]:3 线性表清空成功! 请选择你的操作[0~14]:12 线性表是空表!

综合上述测试，清空线性表功能对于线性表是否初始化、是否是空表都可以正确处理，所以清空线性表功能符合实验要求

(5) 线性表判空的测试

测试用例及结果如表 1-5 所示

表 1-5 线性表判空测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1（空线性表）	空线性表	线性表为空！	请选择你的操作[0~14]:4 线性表为空！
用例 2(未初始化的线性表)	未初始化的线性表	线性表未初始化！	请选择你的操作[0~14]:4 线性表未初始化！
用例 3($s=\{1, 2, 3\}$)	$s=\{1, 2, 3\}$	线性表不为空！	请选择你的操作[0~14]:4 线性表不为空！

综合上述测试，线性表判空功能对于线性表是否初始化、是否是空表都可以正确处理，所以线性表判空功能符合实验要求

(6) 获取元素的测试

测试用例及结果如表 1-6 所示

表 1-6 获取元素测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 ($s=\{1, 2, 3\}$)	2	第 2 号元素为 2！	请选择你的操作[0~14]:6 请输入需要获取的元素的位置: 2 第2号元素为2。
用例 2(未初始化的线性表)	未初始化的线性表	线性表未初始化！	请选择你的操作[0~14]:6 请输入需要获取的元素的位置: 1 线性表未初始化！

用例 3 ($s=\{1, 2, 3\}$)	4	输入位置有误!	
--------------------------	---	---------	---

综合上述测试，获取元素功能对于线性表是否初始化、输入位置是否正确都可以正确处理，所以获取元素功能符合实验要求

(7) 查找元素的测试

测试用例及结果如表 1-7 所示

表 1-7 查找元素测试及结果表

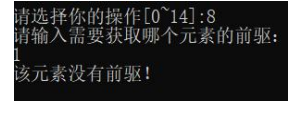
测试用例	程序输入	理论结果	运行结果
用例 1 ($s=\{1, 2, 3\}$)	4	线性表中没有该元素!	
用例 2 (未初始化的线性表)	未初始化的线性表	线性表未初始化!	
用例 3 ($s=\{1, 2, 3\}$)	3	该元素在线性表中位置为 3!	

综合上述测试，查找元素功能对于线性表是否初始化、输入元素是否在线性表中都可以正确处理，所以查找元素功能符合实验要求

(8) 获取某元素的前驱的测试

测试用例及结果如表 1-8 所示

表 1-8 获取某元素的前驱测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 ($s=\{1, 2, 3\}$)	1	该元素没有前驱!	

用例 2 (未初始化的线性表)	未初始化的线性表	线性表未初始化!	<pre> 请选择你的操作[0~14]:8 请输入需要获取哪个元素的前驱: 1 线性表未初始化! </pre>
用例 3 (s={1, 2, 3})	3	该元素的前驱为 2!	<pre> 请选择你的操作[0~14]:8 请输入需要获取哪个元素的前驱: 3 该元素的前驱为2 </pre>
用例 4 (s={1, 2, 3})	6	该元素没有前驱!	<pre> 请选择你的操作[0~14]:8 请输入需要获取哪个元素的前驱: 6 该元素没有前驱! </pre>

综合上述测试，获取某元素前驱功能对于线性表是否初始化、输入不同位置的元素都可以正确处理，所以获取某元素前驱功能符合实验要求

(8) 获取某元素的后继的测试

测试用例及结果如表 1-9 所示

表 1-9 获取某元素的后继测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (s={1, 2, 3})	1	该元素的后继为 2!	<pre> 请选择你的操作[0~14]:9 请输入需要获取哪个元素的后继: 1 该元素的后继为2 </pre>
用例 2 (未初始化的线性表)	未初始化的线性表	线性表未初始化!	<pre> 请选择你的操作[0~14]:9 请输入需要获取哪个元素的后继: 1 线性表未初始化! </pre>
用例 3 (s={1, 2, 3})	3	该元素没有后继!	<pre> 请选择你的操作[0~14]:9 请输入需要获取哪个元素的后继: 3 该元素没有后继! </pre>
用例 4 (s={1, 2, 3})	6	该元素没有后继!	<pre> 请选择你的操作[0~14]:9 请输入需要获取哪个元素的后继: 3 该元素没有后继! </pre>

综合上述测试，获取某元素后继功能对于线性表是否初始化、输入不同位置的元素都可以正确处理，所以获取某元素后继功能符合实验要求

(10) 插入元素的测试

测试用例及结果如表 1-10 所示

表 1-10 插入元素测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 ($s=\{1, 2, 3\}$)	位置: 1 元素: 4	插入成功!	
用例 2 (未初始化的线性表)	未初始化的线性表	线性表未初始化!	
用例 3 ($s=\{1, 2, 3\}$)	位置: 5	插入位置有误!	

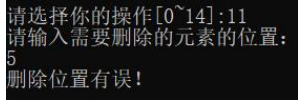
综合上述测试, 插入元素功能对于线性表是否初始化、插入位置是否正确都可以正确处理, 所以插入元素功能符合实验要求

(11) 删除元素的测试

测试用例及结果如表 1-11 所示

表 1-11 删除元素测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 ($s=\{1, 2, 3\}$)	位置: 1	删除成功, 删除的元素是 1!	
用例 2 (未初始化的线性表)	未初始化的线性表	线性表未初始化!	

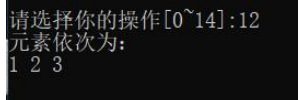

用例 3 ($s=\{1, 2, 3\}$)	位置: 5	删除位置有误!	
--------------------------	-------	---------	---

综合上述测试, 删除元素功能对于线性表是否初始化、删除位置是否正确都可以正确处理, 所以删除元素功能符合实验要求

(12) 打印元素的测试

测试用例及结果如表 1-12 所示

表 1-12 打印元素测试及结果表

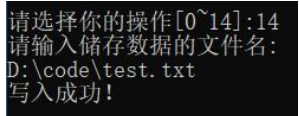
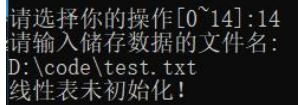
测试用例	程序输入	理论结果	运行结果
用例 1 ($s=\{1, 2, 3\}$)	非空线性表	元素依次为: 1 2 3	
用例 2 (未初始化的线性表)	未初始化的线性表	线性表未初始化!	

综合上述测试, 打印元素功能对于线性表是否初始化都可以正确处理, 所以打印元素功能符合实验要求

(13) 写入文件的测试

测试用例及结果如表 1-13 所示

表 1-13 写入文件测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 ($s=\{1, 2, 3\}$)	D:\code\test.txt	写入成功	
用例 2 (未初始化的线性表)	未初始化的线性表	线性表未初始化!	

综合上述测试, 写入文件功能对于线性表是否初始化都可以正确处理, 所以

写入文件功能符合实验要求

(14) 读取文件的测试

测试用例及结果如表 1-14 所示

表 1-14 读取文件测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1(空表)	D:\code\test.txt	读入成功	<pre>请选择你的操作[0~14]:13 请选择通过键盘对线性表中元素赋值 (1) 或者通过读入文件赋值 (2): 2 请输入文件名: D:\code\test.txt 读入成功!</pre>
用例 2(未初始化的线性表)	未初始化的线性表	线性表未初始化!	<pre>请选择你的操作[0~14]:13 请选择通过键盘对线性表中元素赋值 (1) 或者通过读入文件赋值 (2): 2 请输入文件名: D:\code\test.txt 线性表未初始化!</pre>
用例 3	11	读入失败	<pre>请选择你的操作[0~14]:13 请选择通过键盘对线性表中元素赋值 (1) 或者通过读入文件赋值 (2): 2 请输入文件名: 11 读入失败!</pre>
用例 4(非空表)	D:\code\test.txt	线性表中已经有元素	<pre>请选择你的操作[0~14]:13 请选择通过键盘对线性表中元素赋值 (1) 或者通过读入文件赋值 (2): 2 请输入文件名: D:\code\test.txt 线性表中已经有元素!</pre>

综合上述测试，读入文件功能对于线性表是否初始化，文件名是否正确，线性表中是否有元素都可以正确处理，所以读入文件功能符合实验要求

总结：综合上述所有函数的测试，测试结果都未发现问题，符合本次实验的要求，也较完整较正确地完成了题目要求。

1.5 实验小结

本次实验框架在提供的资料里面已经写好，只需要我们自己将所有的函数补充完整，这样不仅有针对性地练习了线性表的相关操作，也训练了我们的代码集成能力。

同时，在调试自己代码的时候，也出现了很多的问题：

①从前在使用释放内存的 free 函数之后，就不再处理原来的指针，本次实验必须要将释放内存后的指针指向 NULL，这样也让指针的使用更加安全。

②我们写的函数里面，往往有多个返回值。第一次集成代码的时候，我将函数都放到了 if, else if 后的括号里面，这样就导致，当 if 判断完之后，相当于函数执行了一次，但当进入 else if 之后，就会再执行一次，这样执行多次，就会出现错误。最后改进方法是在进入分支语句之前，用变量来储存函数返回值，之后就只需要判断该变量的值就可以了。

③对文件的处理不太熟悉，导致刚开始无法将数据从文件中读取出来，最后又学习了一下相关函数，解决了相关问题。

④有一些储存执行状态的变量（如 flag, k 等）在执行一次之后没能恢复到原来的值，导致如果进行第二次操作的时候就会出现错误。

通过这次实验让我意识到自己还有很多地方需要改进，比如不够细心、不够严谨、考虑问题不够全面等，我会更加努力地改进与提高。

最后很感谢老师与助教对我的问题的及时解答，帮助我比较顺利地完成了这次实验。

2 基于链式储存结构的线性表实现

2.1 问题描述

构造链表，呈现一个简易菜单的功能演示系统，该演示系统可选择实现多个线性表管理以及单线性表的操作。

2.1.1 实验目的

- (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) 物理结构采用链表, 熟练掌握线性表的基本运算的实现。

2.1.2 实验要求

需要在主程序中完成函数调用以及所需实参值和函数执行结果的输出。定义线性表及线性表集合的初始化、销毁、清空、判空、求表长和获得元素等函数，并给出适当的操作提示，并且可选择以文件的形式进行存储和加载。

2.2 系统设计

2.2.1 整体系统结构设计

本演示系统可以实现通过操作菜单对多线性表进行管理，并对单个线性表进行操作，也支持通过文件的写入与读取。

多线性表管理菜单（线性表集合菜单/一级菜单）可供选择的操作有：初始化线性表集合、添加一个线性表、删除特定名称的线性表、查找特定名称的线性表、清空线性表集合、线性表集合判空、求线性表集合中线性表的个数、打印所有线性表的序号及名称、操作特定序号的线性表。

多线性表管理菜单中的“操作特定序号的线性表”可以调出单线性表操作菜单（二级菜单），可供选择的操作有：初始化线性表、销毁线性表、清空线性表、线性表判空、求线性表长度、取出特定位置的元素、定位某元素、求某元素的前驱、求某元素的后继、在特定位置插入元素、删除特定位置的元素、打印线性表中的所有元素、将线性表的数据写入文件、通过键盘输入或者读取文件数据向线性表中一次性输入若干元素。整体系统结构设计示意图如下：

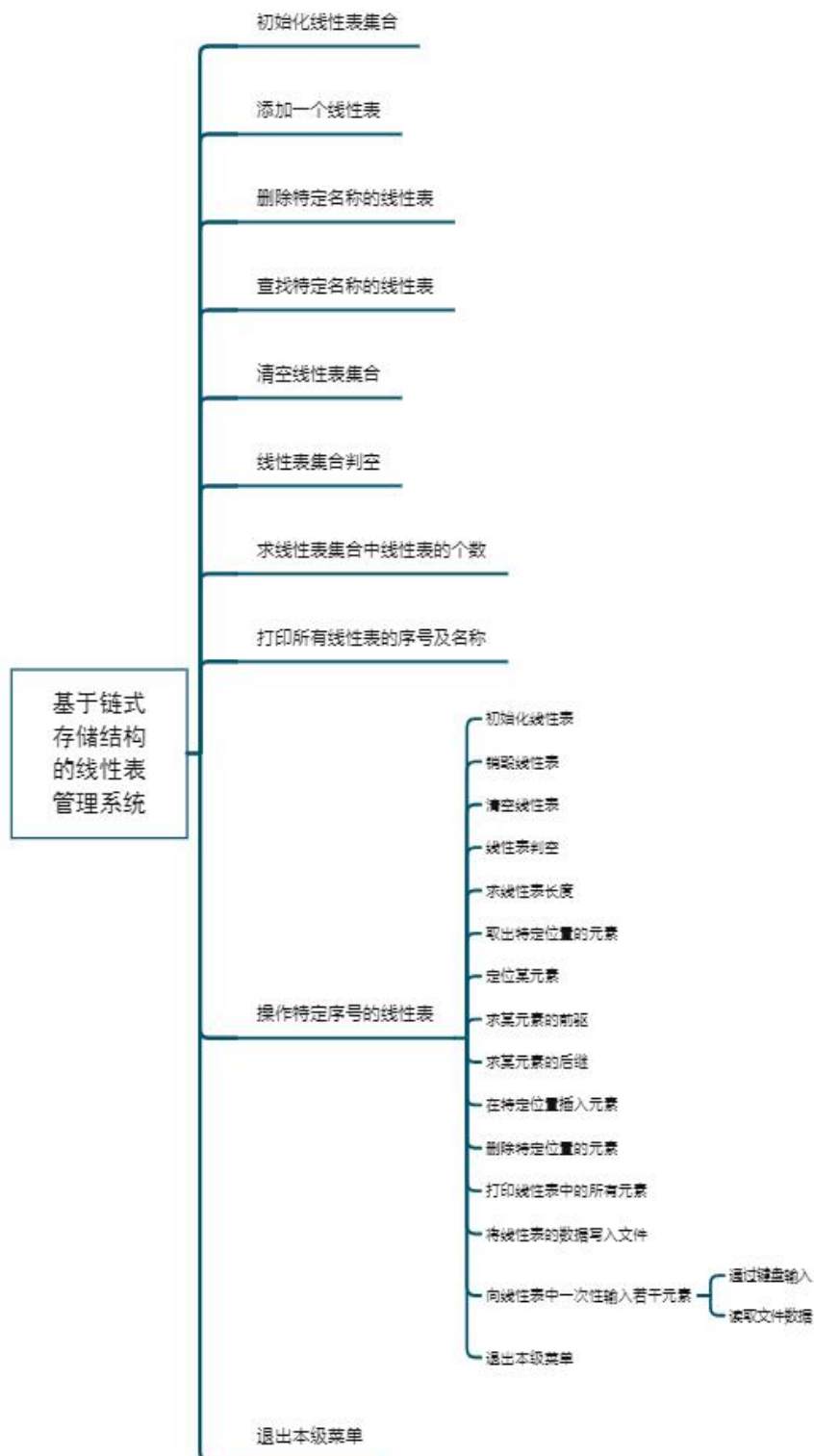


图 2-1 整体系统结构设计

2.2.2 数据结构设计

本演示系统设计了两个物理结构，分别是线性表和线性表集合，两者均以结构形式定义。

线性表结构中包括数据与指向下一个结点的指针。线性表集合结构中包括线性表数组（每个数组元素都包括线性表以及其名称）和线性表集合长度。

结构定义如下：

```
typedef struct LNode { //单链表（链式结构）结点的定义
    ElemType data;
    struct LNode* next;
}LNode, * LinkList;

typedef struct { //链表的集合类型定义
    struct {
        char name[30]; //链表名称
        LinkList L; //链表
    } elem[10]; //链表数组
    int length; //链表集合长度
} LISTS;
```

图 2-2 物理结构设计

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， $1 \leq i \leq \text{ListLength}(L)$ ；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置，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， $1 \leq i \leq \text{ListLength}(L)+1$ ；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， $1 \leq i \leq \text{ListLength}(L)$ ；操作结果：删除 `L` 的第 `i` 个数据元素，用 `e` 返回其值。

(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()`。

2.2.3 有关常量和类型定义

数据元素类型的定义：

```
typedef int status;
```

```
typedef int ElemType;
```

有关常量的定义：

```
#define TRUE 1
```

```
#define FALSE 0
```

```
#define OK 1
```

```
#define ERROR 0
```

```
#define INFEASTABLE -1
```

```
#define OVERFLOW -2
```

2.3 系统实现

本演示系统在 Windows 环境下，使用 Visual Studio 2019 完成。演示系统涉及的部分主要函数如下，其中 (1) - (4) 为对线性表集合的操作，(5) - (12) 为对单线性表的操作。

(1) 向线性表集合中添加一个空线性表

函数名称: `status AddList(LISTS *Lists, char ListName[])`

输入: 线性表集合指针以及需要添加的线性表的名称

输出: 函数执行状态

思想: 在 Lists 中加入一个名称为 ListName 的线性表，成功则返回 OK

操作:

将 ListName 数组拷贝到线性表集合中线性表数组的最后一个元素之后的元素的 name 成员变量中；将新添加的线性表置空；线性表集合长度加一；结束，返回 OK。

时间复杂度: $O(1)$

空间复杂度: $O(1)$

(2) 删除线性表集合中指定名称的线性表

函数名称: `status RemoveList(LISTS *Lists, char ListName[])`

输入: 线性表集合指针以及需要删除的线性表名称

输出: 函数执行状态

思想: 删除指定名称的线性表，删除成功则返回 OK，否则返回 ERROR

操作:

定义 $i=0$ ，当 i 小于线性表集合长度时，执行下列循环：

a. 比较第 i 个线性表的名称和所给名称

b. 当两者相同时，将第 i 个线性表的名称指针的首元素置为 `'\0'`，并使链表指针分配的空间用 `free` 函数收回，并使链表指针指向 `NULL`，将线性表数组中该线性表后面的线性表依次前移，线性表集合的长度减一，返回 OK

若循环结束后函数没有返回值，说明集合中没有该名称的线性表，返回 ERROR

时间复杂度： $O(n)$ （设线性表集合长度为 n ）

空间复杂度： $O(1)$

(3) 查找指定名称的线性表在集合中的位置

函数名称：status LocateList(LISTS Lists, char ListName[])

输入：线性表集合以及需要查找的线性表名称

输出：该线性表的序号或者函数执行状态

思想：在集合中寻找指定名称的线性表，成功则返回其逻辑序号，否则返回 ERROR

操作：

① 定义 $i=0$ ，当 i 小于线性表集合长度时，执行下列循环：

a. 比较第 i 个线性表的名称与所给名称

b. 如果相同，返回 $i+1$ ，否则继续循环

② 如果循环执行完后函数没有返回值，说明集合中没有该名称的线性表，返回 ERROR

时间复杂度： $O(n)$ （设线性表长度为 n ，此处为最坏情况）

空间复杂度： $O(1)$

(4) 清空线性表集合

函数名称：status ClearLists(LISTS *Lists)

输入：线性表集合指针

输出：函数执行状态

思想：将线性表集合的长度变成 0，返回 OK

操作：将 Lists 的 length 赋值为 0，返回 OK

时间复杂度： $O(1)$

空间复杂度： $O(1)$

(5) 销毁线性表

函数名称：status DestroyList(LinkList *L)

输入：线性表指针

输出：函数执行状态

思想：如果线性表已经初始化，销毁线性表，释放每一个结点分配的空间，返回 OK，否则返回 INFEASIBLE

操作：如果线性表未初始化（L 指向 NULL），返回 INFEASIBLE，定义两个指针 q, p 分别指向头结点和首结点。之后循环，当 p 不为空时，让 q 指向 p 的下一个结点并释放 p，把 q 赋值给 p。循环结束后释放头结点 L，将 L 指向 NULL，返回 OK。

时间复杂度：O (n)

空间复杂度：O (1)

(6)清空线性表

函数名称：status ClearList (LinkList *L)

输入：线性表指针

输出：函数执行状态

思想：如果线性表已经初始化，释放除头结点之外的所有结点，以此来达到删除所有元素的目的。

操作：如果线性表未初始化（L 指向 NULL），返回 INFEASIBLE，定义两个指针 q, p 分别指向头结点和首结点。之后循环，当 p 不为空时，让 q 指向 p 的下一个结点并释放 p，把 q 赋值给 p。循环结束后将首结点指向 NULL，返回 OK。

时间复杂度：O (n)

空间复杂度：O (1)

(7)获取线性表特定位置的元素

函数名称：status GetElem (LinkList L, int i, ElemType *e)

输入：线性表，需要获取的元素的位置，存放该元素的变量指针

输出：函数执行状态

思想：如果线性表已经初始化，获取线性表的第 i 个元素，保存在 e 中，返回 OK，如果 i 不合理，返回 ERROR；如果线性表未初始化，返回 INFEASIBLE

操作：如果线性表已经初始化，定义一指针指向首结点，当未到达所需位置时且 p 不指向 NULL 时，将指针后移，循环以上操作。循环结束后将元素保存在 e 中，并返回 OK，如果 i 不合理（ $i < 1$ 或者 $i > \text{线性表长度}$ ），返回 ERROR；如果线性表未初始化（L 指向 NULL），返回 INFEASIBLE

时间复杂度： $O(n)$

空间复杂度： $O(1)$

(8) 确定某元素在线性表中的位置

函数名称：status LocateElem(LinkList L, ElemType e)

输入：线性表，某元素

输出：元素位置或者函数执行状态

思想：若线性表已经初始化，查找元素位置并返回序号，查找不成功时返回相应的函数执行状态

操作：

①如果线性表未初始化，返回 INFEASIBLE

②线性表已经初始化，定义 $i=1$ 和指针 p 指向首结点，执行下列循环：

当 p 不指向 NULL 且 p 的数据域不为 e 时，i 自增。

③如果循环结束后 p 指向 NULL，说明线性表中没有该元素，返回 ERROR，否则返回 i

时间复杂度： $O(n)$

空间复杂度： $O(1)$

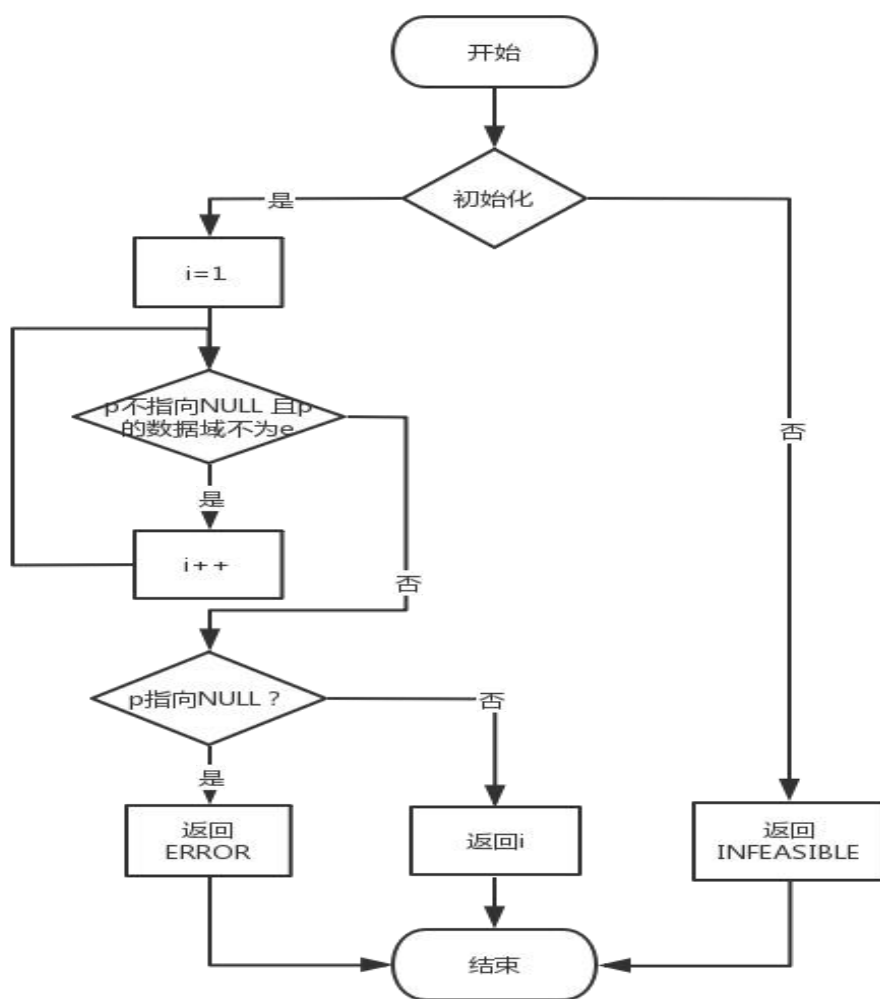


图 2-3 LocateElem 流程图

(9)在某位置插入某元素

函数名称: `status ListInsert (LinkList *L, int i, ElemType e)`

输入: 线性表指针, 插入位置, 插入元素

输出: 函数执行状态

思想: 若线性表已经初始化, 且插入位置正确, 则插入元素, 返回 OK, 否则返回相应的函数执行状态

操作:

①如果线性表未初始化, 返回 INFEASIBLE

②线性表已经初始化:

- a. 如果插入位置不正确 ($i < 1$ 或 $i > \text{线性表长度}+1$), 返回 ERROR
- b. 插入位置正确时, 定义 $k=0$, 指针 p 指向首结点, 指针 q 指向头结点, 当 k 小于 $i-1$ 且 p 不为空时, 执行循环: 将 p 赋值给 q , 将 p 的 next 赋值给 p , 让 $k++$ 。循环结束后创建新结点 s , 将 e 赋值给 s 的数据域, 将 s 赋值给 q 的 next , 将 p 赋值给 s 的 next , 返回 OK。

时间复杂度: $O(n)$

空间复杂度: $O(1)$

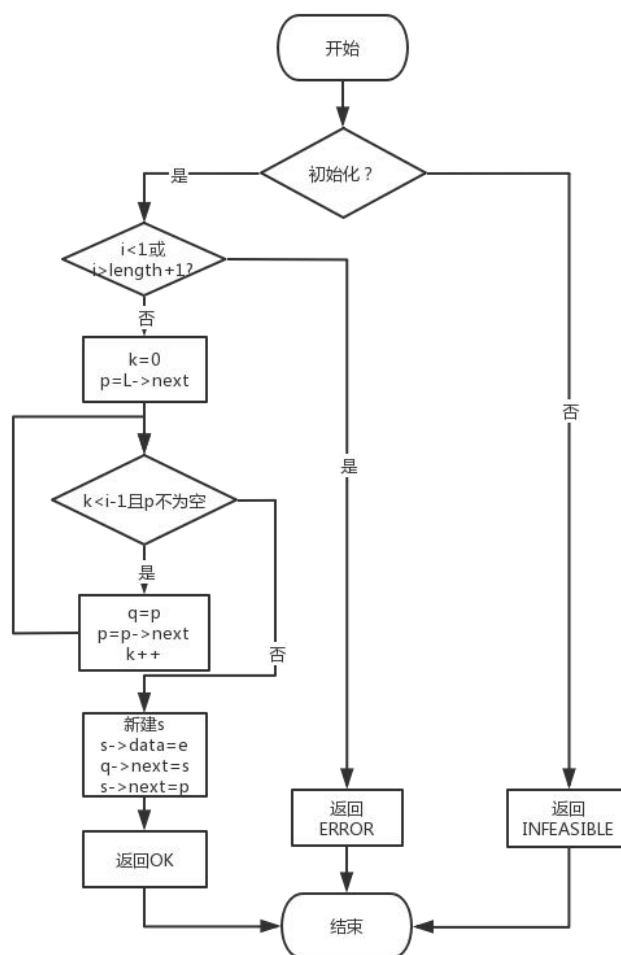


图 2-4 ListInsert 流程图

(10)删除某位置元素

函数名称: `status ListDelete (LinkList *L, int i, ElemType *e)`

输入：线性表指针，删除位置，储存删除元素的变量指针

输出：函数执行状态

思想：若线性表已经初始化且删除位置正确，则删除对应位置元素，返 OK，否则返回相应的函数执行状态

操作：

①如果线性表未初始化，返回 INFEASIBLE

②线性表未初始化：

a. 如果插入位置不正确 ($i < 1$ 或 $i > \text{线性表长度}$)，返回 ERROR

b. 插入位置正确时，定义 $k=0$ ，指针 p 指向首结点，指针 q 指向头结点，当 k 小于 $i-1$ 且 p 不为空时，执行循环：将 p 赋值给 q ，将 p 的 next 赋值给 p ，让 $k++$ 。循环结束后将 p 的 data 赋值给 e ，将 p 的 next 赋值给 q 的 next ，释放 p ，返回 OK。

时间复杂度： $O(n)$

空间复杂度： $O(1)$

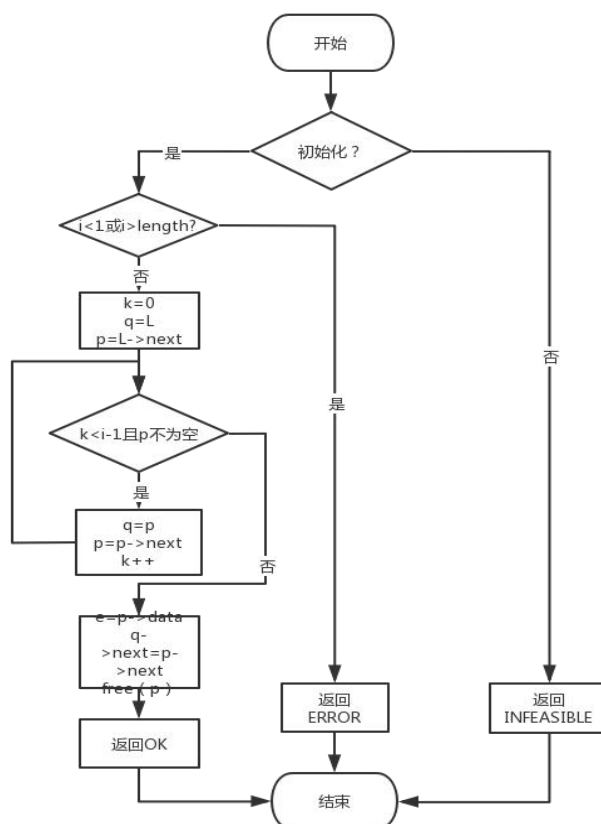


图 2-5 ListDelete 流程图

(11)将线性表的元素写入某文件

函数名称: `status SaveList (LinkList L, char FileName[])`

输入: 线性表, 文件名称

输出: 函数执行状态

思路: 如果线性表已经初始化, 将元素写入文件, 否则返回对应的函数执行状态

操作:

①如果线性表未初始化, 返回 INFEASIBLE

②线性表已经初始化, 定义文件指针并指向打开文件

a. 若打开失败, 返回 ERROR

b. 打开成功, 将文件依次写入, 并关闭文件指针, 返回 OK

时间复杂度: $O(n)$

空间复杂度: $O(1)$

(12)读取某文件中的数据到一空线性表

函数名称: `status LoadList (LinkList *L, char FileName[])`

输入: 线性表指针, 文件名称

输出: 函数执行状态

思路: 如果线性表已经初始化, 将文件中的数据读入线性表, 否则返回相应的函数执行状态

操作:

①如果线性表未初始化, 返回 INFEASIBLE

②线性表已经初始化, 定义文件指针并指向打开文件

a. 若打开失败, 返回 ERROR

b. 打开成功, 将元素依次读取, 并关闭文件指针, 返回 OK

时间复杂度: $O(n)$

空间复杂度: $O(1)$

2.4 系统测试

程序采用较为简单的界面，一级菜单和二级菜单分别如图 2-6，2-7 所示（在一级菜单中可通过操作 9 进入二级菜单）。本次系统测试选取了针对线性表集合操作的 AddList、RemoveList、LocateList 函数以及针对单线性表操作的 ClearList、ListEmpty、GetElem、LocateElem、PriorElem、NextElem、ListInsert、ListDelete、ListTraverse、SaveList、LoadList 函数。

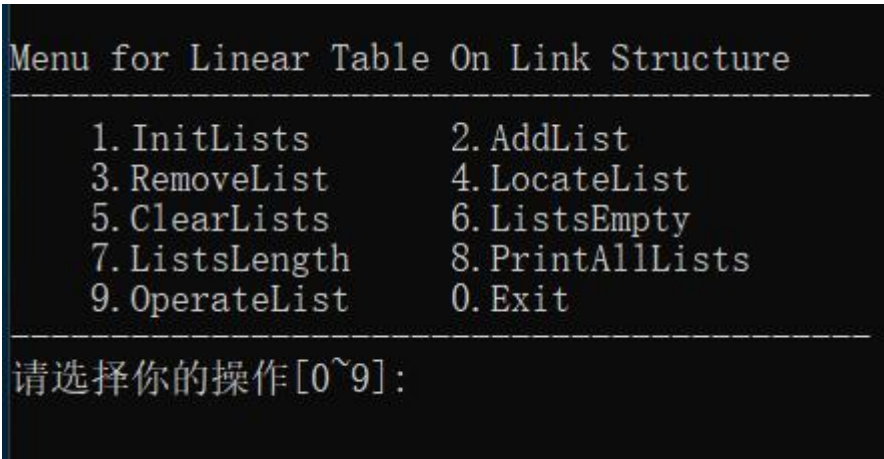


图 2-6 一级菜单示意图

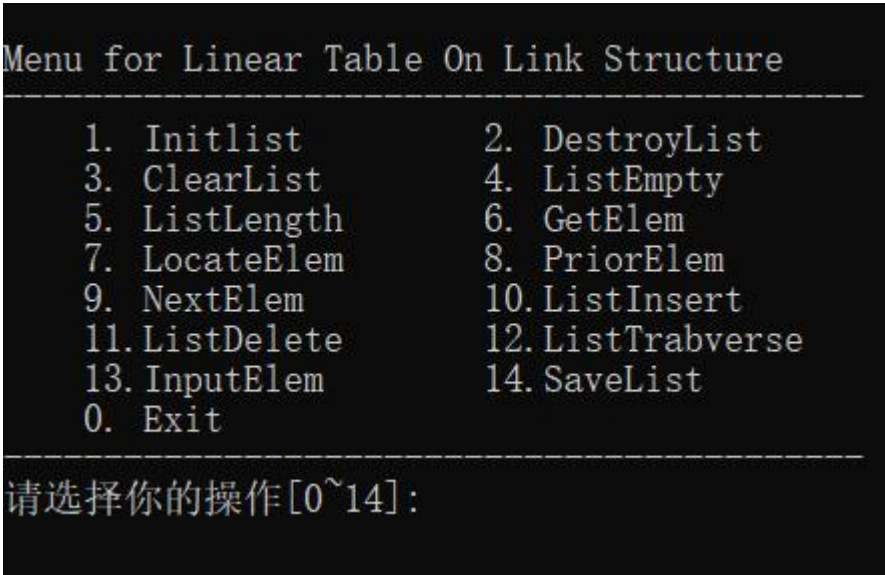


图 2-7 二级菜单示意图

下面是函数测试：

(1) 添加线性表的测试：

测试用例及结果如表 2-1 所示

表 2-1 添加线性表测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (集合中已有 aaa 线性表)	名称: aaa	名称有误! 请重新 输入!	<pre> 请选择你的操作[0~9]:2 请输入需要添加的线性表的名称: aaa 名称重复! 请重新输入! 请选择你的操作[0~9]:8 所有线性表依次为: 1 aaa </pre>
用例 2 (空集合)	名称: bbb	添加成功!	<pre> 请选择你的操作[0~9]:2 请输入需要添加的线性表的名称: bbb 添加成功! 请选择你的操作[0~9]:8 所有线性表依次为: 1 bbb </pre>

综合上述测试, 添加线性表功能对于输出名称重复时以及未重复时都可以正确处理, 所以添加线性表功能符合实验要求。

(2) 删除线性表的测试:

测试用例及结果如表 2-2 所示

表 2-2 删除线性表测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (集合中已有 aaa 线性表)	名称: aaa	删除成功!	<pre> 请选择你的操作[0~9]:3 请输入需要删除的线性表的名称: aaa 删除成功! 请选择你的操作[0~9]:8 该线性表集合中没有元素! </pre>
用例 2 (集合中已有 aaa 线性表)	名称: bbb	线性表集合中没 有该线性表!	<pre> 请选择你的操作[0~9]:3 请输入需要删除的线性表的名称: bbb 线性表集合中没有该线性表! 请选择你的操作[0~9]:8 所有线性表依次为: 1 aaa </pre>
用例 3 (空集合)	名称: aaa	线性表集合中没 有该线性表!	<pre> 请选择你的操作[0~9]:3 请输入需要删除的线性表的名称: aaa 线性表集合中没有该线性表! 请选择你的操作[0~9]:8 该线性表集合中没有元素! </pre>

综合上述测试, 删除线性表功能对于线性表集合中有或无对应名称的线性表

时都可以正确处理，所以删除线性表功能符合实验要求。

(3) 定位线性表的测试

测试用例及结果如表 2-3 所示

表 2-3 定位线性表测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (空集合)	名称: aaa	线性表集合中没有该线性表!	
用例 2 (集合中已有 aaa, bbb 线性表)	名称: bbb	该线性表位置是 2	
用例 3 (集合中已有 aaa, bbb 线性表)	名称: ccc	线性表集合中没有该线性表!	

综合上述测试，定位线性表功能对于线性表集合中有或无对应名称的线性表时都可以正确处理，所以定位线性表功能符合实验要求

(4) 清空线性表的测试

测试用例及结果如表 2-4 所示

表 2-4 清空线性表测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (空线性表)	空线性表	线性表为空!	

用例 2 ($s=\{1, 2, 3\}$)	$s=\{1, 2, 3\}$	清空成功!	<div>请选择你的操作[0~14]:3 线性表清空成功!</div> <div>请选择你的操作[0~14]:12 线性表是空表!</div>
用例 3 (未初始化的线性表)	未初始化的线性表	线性表未初始化!	<div>请选择你的操作[0~14]:3 线性表未初始化!</div>

综合上述测试，清空线性表功能对于线性表是否初始化、是否是空表都可以正确处理，所以清空线性表功能符合实验要求

(5) 线性表判空的测试

测试用例及结果如表 2-5 所示

表 2-5 线性表判空测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (空线性表)	空线性表	线性表为空!	<div>请选择你的操作[0~14]:4 线性表为空!</div>
用例 2 ($s=\{1, 2, 3\}$)	$s=\{1, 2, 3\}$	线性表不为空!	<div>请选择你的操作[0~14]:4 线性表不为空!</div>
用例 3 (未初始化的线性表)	未初始化的线性表	线性表未初始化!	<div>请选择你的操作[0~14]:4 线性表未初始化!</div>

综合上述测试，线性表判空功能对于线性表是否初始化、是否是空表都可以正确处理，所以线性表判空功能符合实验要求

(6) 获取元素的测试

测试用例及结果如表 2-6 所示

表 2-6 获取元素测试及结果表

测试用例	程序输入	理论结果	运行结果
------	------	------	------

用例 1 ($s=\{1, 2, 3\}$)	2	第 2 号元素为 2!	请选择你的操作[0~14]:6 请输入需要获取的元素的位置: 2 第2号元素为2。
用例 2 ($s=\{1, 2, 3\}$)	4	输入位置有误!	请选择你的操作[0~14]:6 请输入需要获取的元素的位置: 4 输入位置有误!
用例 3(未初始化的 线性表)	未初始化的线性 表	线性表未初始化!	请选择你的操作[0~14]:6 请输入需要获取的元素的位置: 1 线性表未初始化!

综合上述测试，获取元素功能对于线性表是否初始化、输入位置是否正确都可以正确处理，所以获取元素功能符合实验要求

(7) 查找元素的测试

测试用例及结果如表 2-7 所示

表 2-7 查找元素测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 ($s=\{1, 2, 3\}$)	4	线性表中没有该 元素!	请选择你的操作[0~14]:7 请输入需要查找的元素: 4 线性表中没有该元素!
用例 2 ($s=\{1, 2, 3\}$)	3	该元素在线性表 中位置为 3!	请选择你的操作[0~14]:7 请输入需要查找的元素: 3 该元素在线性表中的位置为3
用例 3(未初始化的 线性表)	未初始化的线性 表	线性表未初始化!	请选择你的操作[0~14]:7 请输入需要查找的元素: 1 线性表未初始化!

综合上述测试，查找元素功能对于线性表是否初始化、输入元素是否在线性表中都可以正确处理，所以查找元素功能符合实验要求

(8) 获取某元素的前驱的测试

测试用例及结果如表 2-8 所示

表 2-8 获取某元素的前驱测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 ($s=\{1, 2, 3\}$)	1	该元素没有前驱!	请选择你的操作[0~14]:8 请输入需要获取哪个元素的前驱: 1 该元素没有前驱!
用例 2 ($s=\{1, 2, 3\}$)	3	该元素的前驱为 2!	请选择你的操作[0~14]:8 请输入需要获取哪个元素的前驱: 3 该元素的前驱为2
用例 3 ($s=\{1, 2, 3\}$)	6	该元素没有前驱!	请选择你的操作[0~14]:8 请输入需要获取哪个元素的前驱: 6 该元素没有前驱!
用例 4 (未初始化的 线性表)	未初始化的线性 表	线性表未初始化!	请选择你的操作[0~14]:8 请输入需要获取哪个元素的前驱: 1 线性表未初始化!

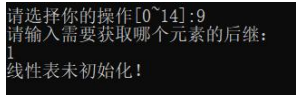
综合上述测试，获取某元素前驱功能对于线性表是否初始化、输入不同位置的元素都可以正确处理，所以获取某元素前驱功能符合实验要求

(8) 获取某元素的后继的测试

测试用例及结果如表 2-9 所示

表 2-9 获取某元素的后继测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 ($s=\{1, 2, 3\}$)	1	该元素的后继为 2!	请选择你的操作[0~14]:9 请输入需要获取哪个元素的后继: 1 该元素的后继为2
用例 2 ($s=\{1, 2, 3\}$)	3	该元素没有后继!	请选择你的操作[0~14]:9 请输入需要获取哪个元素的后继: 3 该元素没有后继!
用例 3 ($s=\{1, 2, 3\}$)	6	该元素没有后继!	请选择你的操作[0~14]:9 请输入需要获取哪个元素的后继: 6 该元素没有后继!

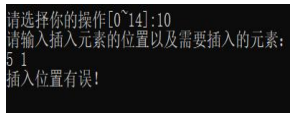
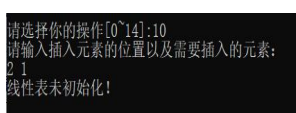
用例 4(未初始化的线性表)	未初始化的线性表	线性表未初始化!	
----------------	----------	----------	---

综合上述测试，获取某元素后继功能对于线性表是否初始化、输入不同位置的元素都可以正确处理，所以获取某元素后继功能符合实验要求

(10) 插入元素的测试

测试用例及结果如表 2-10 所示

表 2-10 插入元素测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 ($s=\{1, 2, 3\}$)	位置: 1 元素: 4	插入成功!	 
用例 2 ($s=\{1, 2, 3\}$)	位置: 5	插入位置有误!	
用例 3(未初始化的线性表)	未初始化的线性表	线性表未初始化!	

综合上述测试，插入元素功能对于线性表是否初始化、插入位置是否正确都可以正确处理，所以插入元素功能符合实验要求

(11) 删除元素的测试

测试用例及结果如表 2-11 所示

表 2-11 删除元素测试及结果表

测试用例	程序输入	理论结果	运行结果
------	------	------	------

用例 1 ($s=\{1, 2, 3\}$)	位置: 1	删除成功, 删除的元素是 1!	请选择你的操作[0~14]:11 请输入需要删除的元素的位置: 1 删除成功, 删除的元素为1
用例 2 ($s=\{1, 2, 3\}$)	位置: 5	删除位置有误!	请选择你的操作[0~14]:11 请输入需要删除的元素的位置: 5 删除位置有误!
用例 3 (未初始化的线性表)	未初始化的线性表	线性表未初始化!	请选择你的操作[0~14]:11 请输入需要删除的元素的位置: 1 线性表未初始化!

综合上述测试, 删除元素功能对于线性表是否初始化、删除位置是否正确都可以正确处理, 所以删除元素功能符合实验要求

(12) 打印元素的测试

测试用例及结果如表 2-12 所示

表 2-12 打印元素测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 ($s=\{1, 2, 3\}$)	非空线性表	元素依次为: 1 2 3	请选择你的操作[0~14]:12 元素依次为: 1 2 3
用例 2 (未初始化的线性表)	未初始化的线性表	线性表未初始化!	请选择你的操作[0~14]:12 线性表未初始化!

综合上述测试, 打印元素功能对于线性表是否初始化都可以正确处理, 所以打印元素功能符合实验要求

(13) 写入文件的测试

测试用例及结果如表 2-13 所示

表 2-13 写入文件测试及结果表

测试用例	程序输入	理论结果	运行结果
------	------	------	------

用例 1($s=\{1, 2, 3\}$)	D:\code\test.txt	写入成功	请选择你的操作[0~14]:14 请输入储存数据的文件名: D:\code\test.txt 写入成功!
用例 2(未初始化的线性表)	未初始化的线性表	线性表未初始化!	请选择你的操作[0~14]:14 请输入储存数据的文件名: D:\code\test.txt 线性表未初始化!

综合上述测试，写入文件功能对于线性表是否初始化都可以正确处理，所以写入文件功能符合实验要求

(14) 读取文件的测试

测试用例及结果如表 2-14 所示

表 2-14 读取文件测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1(空表)	D:\code\test.txt	读入成功	请选择你的操作[0~14]:13 请选择通过键盘对线性表中元素赋值(1)或者通过读入文件赋值(2): 2 请输入文件名: D:\code\test.txt 读入成功!
用例 2	11	读入失败	请选择你的操作[0~14]:13 请选择通过键盘对线性表中元素赋值(1)或者通过读入文件赋值(2): 2 请输入文件名: 11 读入失败!
用例 3(非空表)	D:\code\test.txt	线性表中已经有元素	请选择你的操作[0~14]:13 请选择通过键盘对线性表中元素赋值(1)或者通过读入文件赋值(2): 2 请输入文件名: D:\code\test.txt 线性表中已经有元素!
用例 4(未初始化的线性表)	未初始化的线性表	线性表未初始化!	请选择你的操作[0~14]:13 请选择通过键盘对线性表中元素赋值(1)或者通过读入文件赋值(2): 2 请输入文件名: D:\code\test.txt 线性表未初始化!

综合上述测试，读入文件功能对于线性表是否初始化，文件名是否正确，线

性表中是否有元素都可以正确处理，所以读入文件功能符合实验要求

总结：综合上述所有函数的测试，测试结果都未发现问题，符合本次实验的要求，也较完整较正确地完成了题目要求。

2.5 实验小结

本次实验框架在提供的资料里面已经写好，只需要我们自己将所有的函数补充完整，这样不仅有针对性地练习了线性表的相关操作，也训练了我们的代码集成能力。

由于这次的实验与上次的实验相差不大，故这次的实验也比较顺利。

但写代码的时候仍然出现了一些问题：

①编写文件读取函数时，由于循环的问题，在最后实际上多建立了一个空结点，这样在遍历生成的线性表的时候最后总会多出一个 0，解决方案是，另添加了一个指向结点的指针 r ，使这个指针总是指向在循环中新生成的结点。最后在循环结束后，指针 p 指向的是多出来的空结点，指针 r 指向的是它的前一个结点，这时只需将 p 释放掉，并让 $r \rightarrow next$ 指向 NULL 即可。

②有些函数传入的参数是 $LinkList *L$ ，由于 $LinkList$ 定义的是指针类型，故传入的是二级指针，如果要对单链表进行操作，需要先解引用 ($*L$)，再进行单链表的操作。

最后很感谢老师与助教对我的问题的及时解答，帮助我比较顺利地完成了这次实验。

3 基于二叉链表的二叉树实现

3.1 问题描述

构造二叉树，呈现一个简易菜单的功能演示系统，该演示系统可选择实现多个二叉树管理以及单二叉树的操作。

3.1.1 实验目的

- (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) 物理结构采用二叉链表, 熟练掌握二叉树的基本运算的实现。

3.1.2 实验要求

需要在主程序中完成函数调用以及所需实参值和函数执行结果的输出。定义二叉树的创建、清空、销毁、求深度、定位结点、结点赋值、获得兄弟结点、插入结点、删除结点、遍历等函数及集合的初始化、销毁、清空、判空、求表长和获得元素等函数，并给出适当的操作提示，并且可选择以文件的形式进行存储和加载。

3.2 系统设计

3.2.1 整体系统结构设计

本演示系统可以实现通过操作菜单对多二叉树进行管理，并对单个二叉树进行操作，也支持通过文件的写入与读取。

多二叉树管理菜单（二叉树集合菜单/一级菜单）可供选择的操作有：初始化二叉树集合、添加一个二叉树、删除特定名称的二叉树、查找特定名称的二叉树、清空二叉树集合、二叉树集合判空、求二叉树集合中二叉树的个数、打印所有二叉树的序号及名称、操作特定序号的二叉树。

多二叉树管理菜单中的“操作特定序号的二叉树”可以调出单二叉树操作菜单（二级菜单），可供选择的操作有：创建二叉树、销毁二叉树、清空二叉树、求二叉树深度、定位某结点、结点赋值、求某结点的兄弟结点、在特定位置插入结点、删除特定位置的结点、遍历二叉树、将二叉树的数据写入文件、通过键盘

输入或者读取文件数据向二叉树中一次性输入若干元素。整体系统结构设计示意图如下：

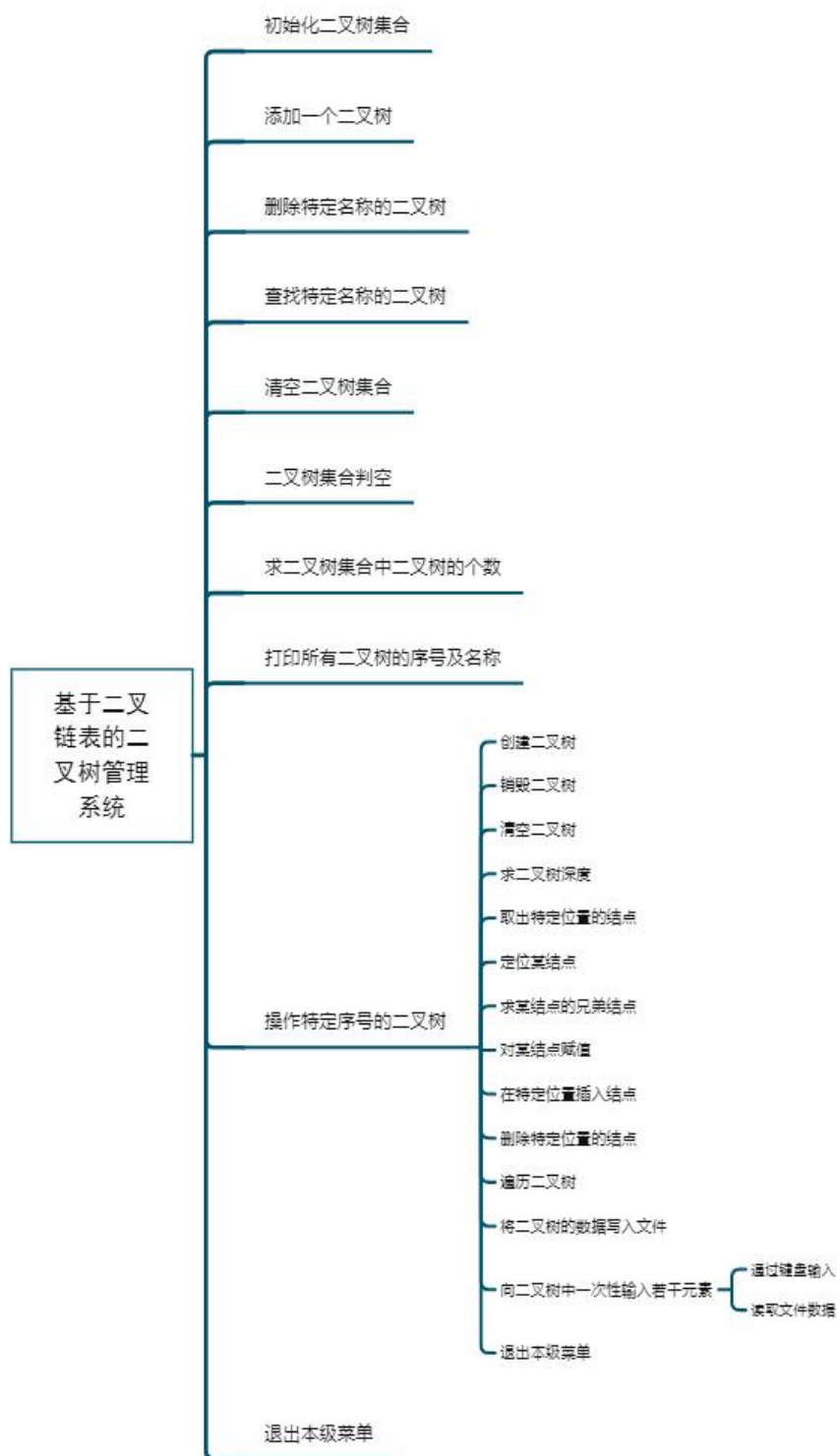


图 3-1 整体系统结构设计

3.2.2 数据结构设计

本演示系统设计了两个物理结构，分别是二叉树和二叉树集合，两者均以结构形式定义。

二叉树结构中包括数据与指向左孩子和右孩子的指针。二叉树集合结构中包括二叉树数组（每个数组元素都包括二叉树以及其名称）和二叉树集合长度。

结构定义如下：

```
typedef struct BiTNode { //二叉树结点的定义
    TElemType data; //数据域
    struct BiTNode* lchild, * rchild; //左右孩子指针
} BiTNode, * BiTree;
typedef struct { //二叉树的集合类型定义
    struct {
        char name[30]; //二叉树名称
        BiTree T; //二叉树
    } elem[10]; //二叉树数组
    int length; //二叉树集合长度
} BITREES;
```

图 3-2 数据结构设计

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。

(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。

(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。

(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。

(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。

(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。

3.2.3 有关常量和类型定义

数据元素类型的定义：

```
typedef int status;

typedef int KeyType;

typedef struct {
    KeyType key;
    char others[20];
} TElemType; //二叉树数据类型定义
```

有关常量的定义：

```
#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2
```

3.3 系统实现

本演示系统在 Windows 环境下，使用 Visual Studio 2019 完成。演示系统涉及的部分主要函数如下，其中（1）-（4）为对二叉树集合的操作，（5）-（15）为对单二叉树的操作。

(1)向二叉树集合中添加一个空二叉树

函数名称：status AddBiTree(BITREES* BiTrees, char BiTreeName[])

输入：二叉树集合指针以及需要添加的二叉树的名称

输出：函数执行状态

思想：在 Lists 中加入一个名称为 ListName 的二叉树，成功则返回 OK

操作：

将 ListName 数组拷贝到二叉树集合中二叉树数组的最后一个元素之后

的元素的 name 成员变量中；将新添加的二叉树置空；二叉树集合长度加一；结束，返回 OK。

时间复杂度：O(1)

空间复杂度：O(1)

(2) 删除二叉树集合中指定名称的二叉树

函数名称：status RemoveBiTree(BITREES* BiTrees, char BiTreeName[])

输入：二叉树集合指针以及需要删除的二叉树名称

输出：函数执行状态

思想：删除指定名称的二叉树，删除成功则返回 OK，否则返回 ERROR

操作：

定义 i=0，当 i 小于二叉树集合长度时，执行下列循环：

a. 比较第 i 个二叉树的名称和所给名称

b. 当两者相同时，将第 i 个二叉树的名称指针的首元素置为 '\0'，并使二叉树指针分配的空间用 free 函数收回，并使二叉树指针指向 NULL，将二叉树数组中该二叉树后面的二叉树依次前移，二叉树集合的长度减一，返回 OK

若循环结束后函数没有返回值，说明集合中没有该名称的二叉树表，返回 ERROR

时间复杂度：O(n) (设二叉树集合长度为 n)

空间复杂度：O(1)

(3) 查找指定名称的二叉树在集合中的位置

函数名称：status LocateBiTree(BITREES BiTrees, char BiTreeName[])

输入：二叉树集合以及需要查找的二叉树名称

输出：该二叉树的序号或者函数执行状态

思想：在集合中寻找指定名称的二叉树，成功则返回其逻辑序号，否则返回 ERROR

操作：

① 定义 i=0，当 i 小于二叉树集合长度时，执行下列循环：

a. 比较第 i 个二叉树的名称与所给名称

b. 如果相同，返回 $i+1$ ，否则继续循环

② 如果循环执行完后函数没有返回值，说明集合中没有该名称的二叉树，返回 ERROR

时间复杂度： $O(n)$ （设二叉树长度为 n ，此处为最坏情况）

空间复杂度： $O(1)$

(4) 清空二叉树集合

函数名称：status ClearBiTrees(BITREES* BiTrees)

输入：二叉树集合指针

输出：函数执行状态

思想：将二叉树集合的长度变成 0，返回 OK

操作：将 Lists 的 length 赋值为 0，返回 OK

时间复杂度： $O(1)$

空间复杂度： $O(1)$

(5) 创建二叉树

函数名称：status CreateBiTree(BiTree& T, TElemType definition[])

输入：二叉树、数据数组

输出：函数执行状态

思想：采用递归思想，根据数据数组中数据的数据按先序创建二叉树

操作：

0. 用一指针 p 指向数据数组

1. 判断数据数组中是否存在关键字重复的情况，若重复，则返回 ERROR

2. 若此时 $*p$ 的关键字为 -1，说明创建完成，返回 -1

3. 如果 $*p$ 的关键字为 0，说明此处为空，把 T 置空；如果不为 0，将 $*p$ 赋值给 T 的数据域， p 后移，再判断关键字是否为 -1，是则返回 OK，不是则创建左子树，结束后再判断是否为 -1，不是则 p 后移，再判断 p 关键字是否为 -1，不是则创建右子树。

时间复杂度: $O(n)$

空间复杂度: $O(1)$

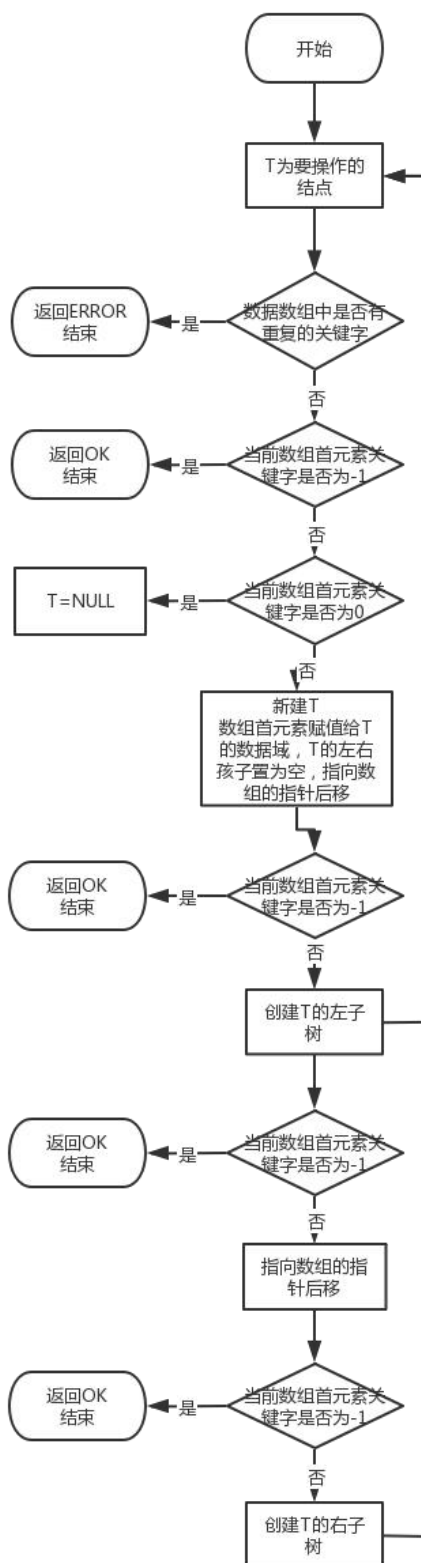


图 3-3 创建二叉树流程图

(6)清空二叉树

函数名称: `status ClearBiTree(BiTree& T)`

输入: 二叉树

输出: 函数执行状态

思想: 采用递归思想, 依次清除每一个结点

操作: 如果 T 为空, 返回 OK, 否则删除其左孩子, 完成后删除右孩子, 完成后 `free(T)`, 把 T 置空, 返回 OK

时间复杂度: $O(n)$

空间复杂度: $O(1)$

(7)求二叉树深度

函数名称: `int BiTreeDepth(BiTree T)`

输入: 二叉树

输出: 树的深度

思想: 采用递归思想, 对于每个结点, 求左右子树的深度, 返回最大值加一

操作: 如果 T 为空, 返回 0, 否则依次求左子树深度和右子树深度并储存在变量中, 返回左右子树深度最大值加一之后的数字

时间复杂度: $O(n)$

空间复杂度: $O(n)$

(8)结点赋值

函数名称: `status Assign(BiTree& T, KeyType e, TElemType value)`

输入: 二叉树、待赋值的结点的关键字、赋值数据

输出: 函数执行状态

思想: 采用递归思想, 遍历寻找关键字为 e 的结点, 若已找到但 `value.key < e`, 则会造成关键字重复, 不合题意, 否则进行赋值; 若没有找到, 就依次在结点的左右子树寻找。

操作: 如果 T 为空, 返回 ERROR。若 T 不为空, 且 `T->data.key == e`, 如果 `value.key < e`, 返回 ERROR, 否则 `T->data = value`, 返回 OK; 若 `T->data.key != e`,

将该结点左子树传入函数的返回值赋值给 temp, 如果 temp==OK, 返回 OK, 如果等于 ERROR, 将该结点右子树传入函数的返回值赋值给 temp, 如果 temp==OK, 返回 OK, 否则返回 ERROR。

时间复杂度: $O(n)$

空间复杂度: $O(n)$

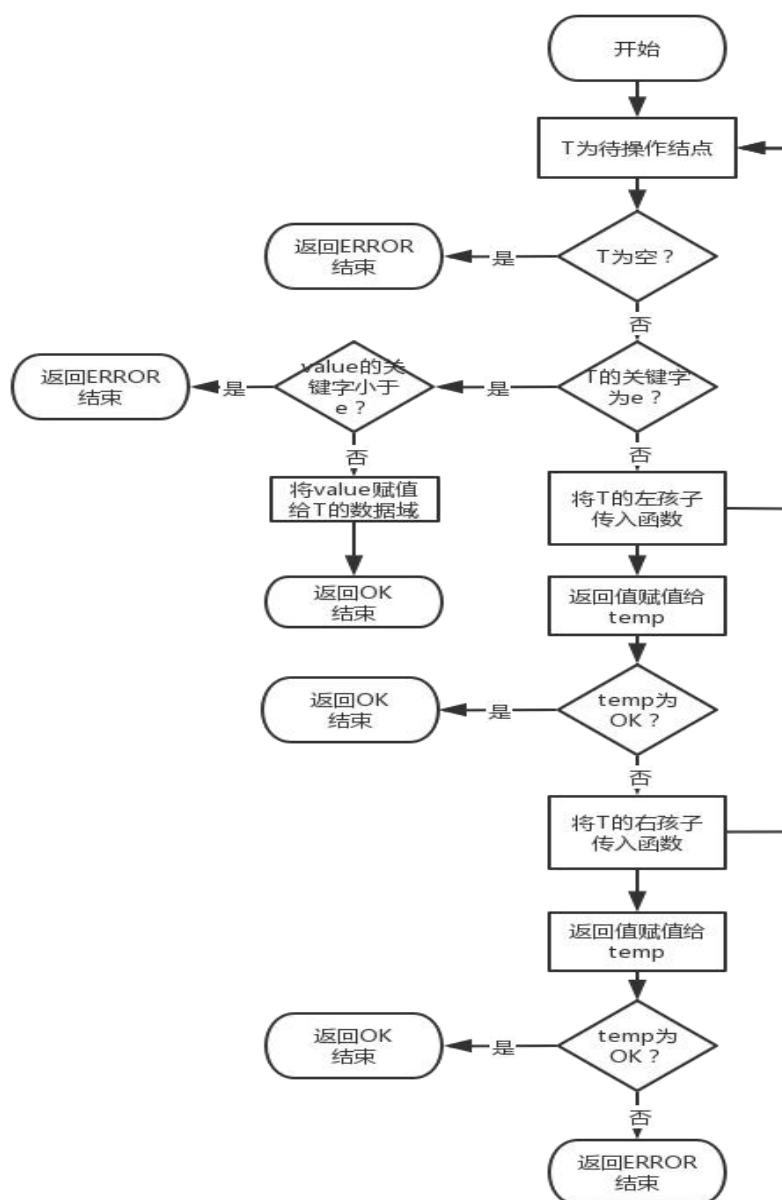


图 3-4 结点赋值流程图

(9) 获得兄弟结点

函数名称: `BiTNode* GetSibling(BiTree T, KeyType e)`

输入: 二叉树, 需要获取兄弟结点的结点的关键字

输出: 结点指针

思想: 采用递归思想, 找到关键字等于 e 的结点的父结点, 返回该父结点的另一个孩子。

操作: 如果 T 为空, 返回 `NULL`。如果 T 有一个孩子为空, 就返回 `NULL`; 如果 T 的左孩子的关键字为 e , 就返回 T 的右孩子; 如果 T 的右孩子的关键字为 e , 就返回 e 的左孩子; 否则将该结点的左孩子传入函数的返回值赋值给 `temp1`, 如果 `temp1` 不为空, 就返回 `temp1`, 否则将该结点的右孩子传入函数的返回值赋值给 `temp1`, 如果 `temp1` 不为空, 返回 `temp1`, 否则返回 `NULL`

时间复杂度: $O(n)$

空间复杂度: $O(n)$

流程图见下页

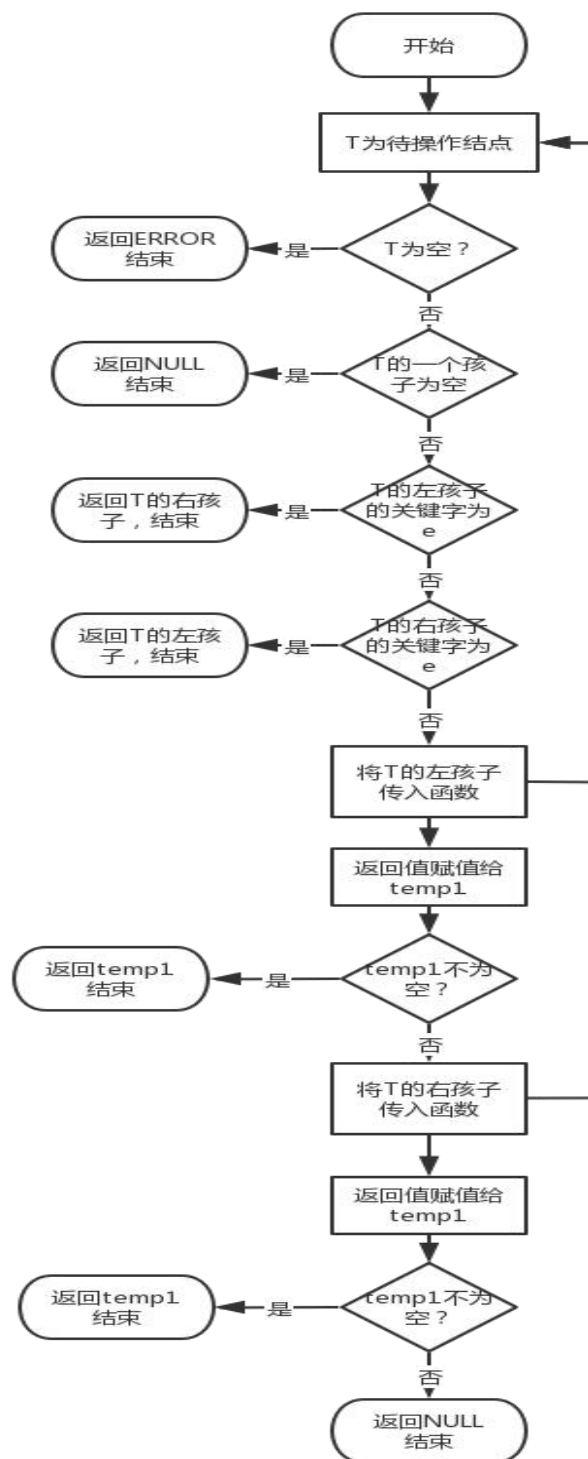


图 3-5 获得兄弟结点流程图

(10)插入结点

函数名称: `status InsertNode(BiTree& T,KeyType e, int LR, TElemType c)`

输入: 二叉树, 插入位置的结点的关键字, 插入方式, 插入数据

输出: 函数执行状态

思想: 采用递归思想, 找到关键字为 `e` 的结点, 根据传入的数据进行相应方式的插入。

操作: 如果 `T` 为空, 返回 `ERROR`; 如果 `LR=-1` 且树中存在关键字为 `e` 的结点 (用其他函数判断), 新建结点, 将原根结点作为新建结点的右孩子, 返回 `OK`; 否则如果 `c.key<=e`, 说明关键字重复, 返回 `ERROR`, 如果 `LR=0` 且 `T->data.key=e`, 新建结点, 将新结点作为 `T` 的左孩子插入, 返回 `OK`, 如果 `LR=1` 且 `T->data.key=e`, 新建结点, 将新结点作为 `T` 的右孩子插入, 返回 `OK`; 如果 `T->data.key!=e`, 将该结点左子树传入函数的返回值赋值给 `temp`, 如果 `temp==OK`, 返回 `OK`, 如果等于 `ERROR`, 将该结点右子树传入函数的返回值赋值给 `temp`, 如果 `temp==OK`, 返回 `OK`, 否则返回 `ERROR`。

时间复杂度: $O(n)$

空间复杂度: $O(n)$

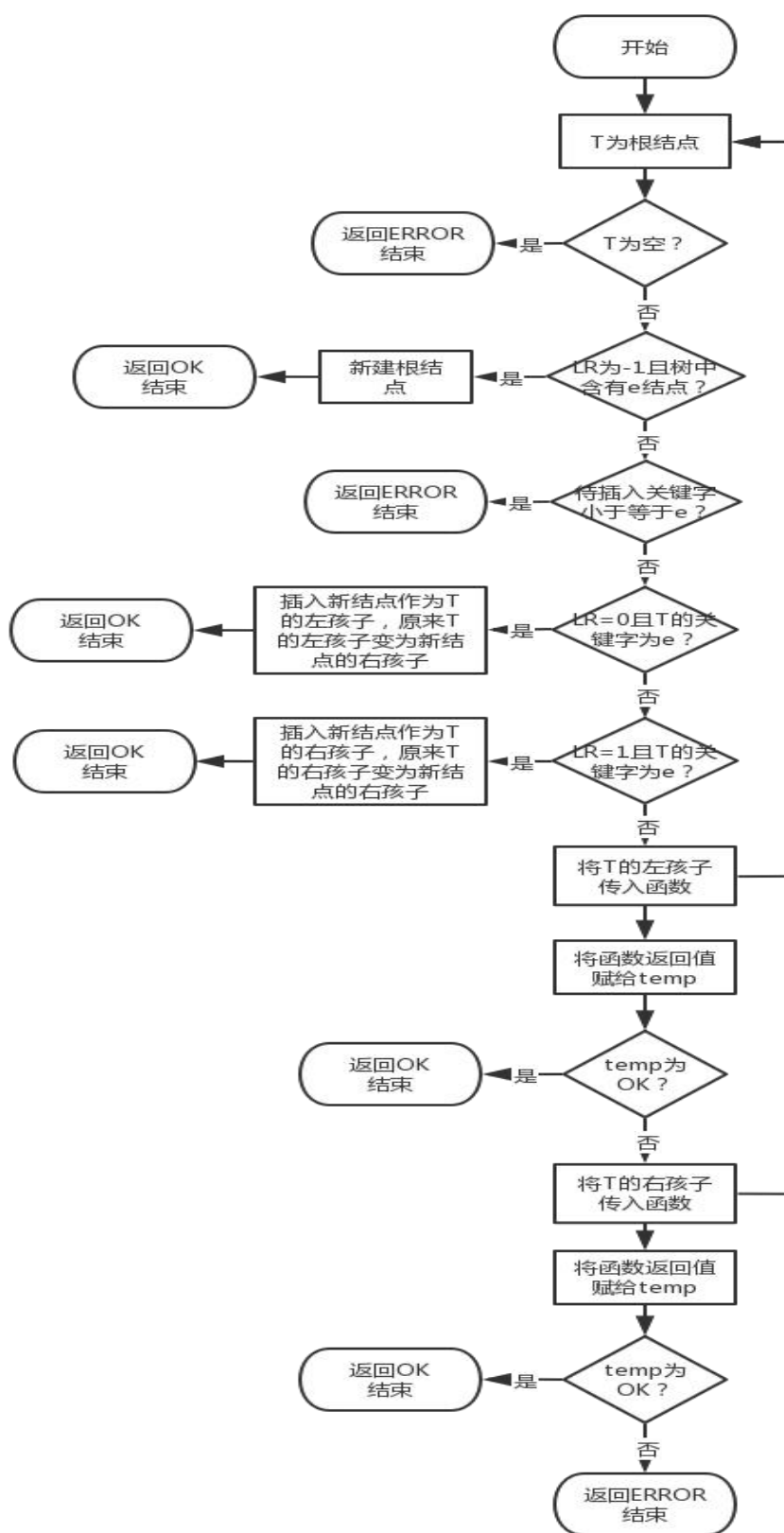


图 3-6 插入结点流程图

(11)删除结点

函数名称: `status DeleteNode(BiTree& T, KeyType e)`

输入: 二叉树, 要删除的结点的关键字

输出: 函数执行状态

思想: 如果根结点为要删除的结点, 删除即可, 否则找到关键字为 e 的结点的父结点 (用其他函数完成), 根据要删除的结点的度, 进行不同的删除操作。

操作: 如果关键字为 e 的结点为根结点, 直接对根结点进行操作, 否则找到关键字为 e 的结点的父结点, 如果没有该结点, 返回 `ERROR`, 否则对该父结点的关键字为 e 的孩子进行操作, 操作为: 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。 成功删除结点后返回 `OK`, 否则返回 `ERROR`。

时间复杂度: $O(1)$ 或 $O(\text{depth})$ 或 $O(n)$ 或 $O(n+\text{depth})$

空间复杂度: $O(1)$

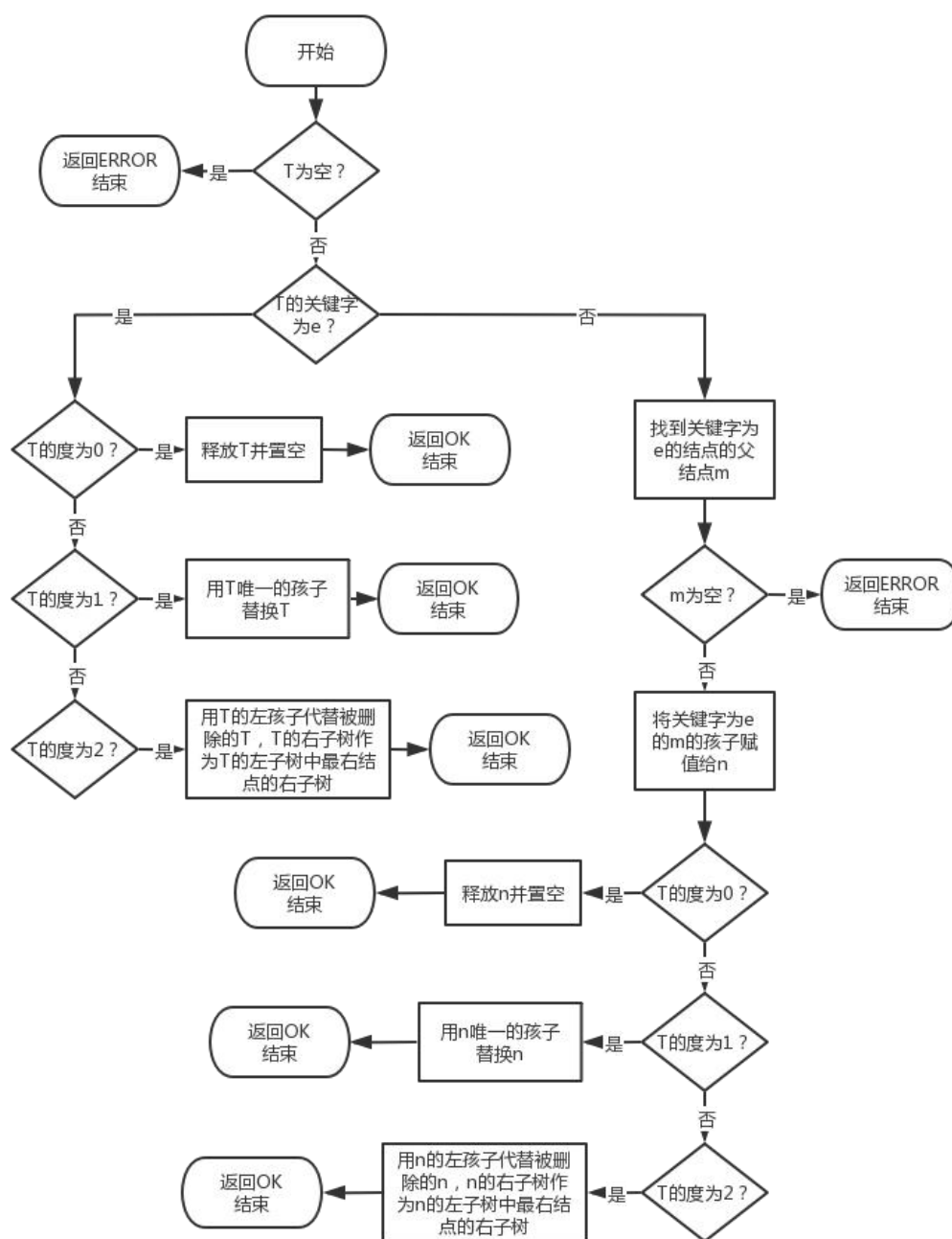


图 3-7 删除结点流程图

(12) 先序遍历

函数名称: `status PreOrderTraverse(BiTree T, void (*visit)(BiTree))`

输入: 二叉树, 操作函数指针

输出: 函数执行状态

思想: 采用递归思想, 依次访问根结点, 左子树, 右子树

操作: 如果 T 不为空, 将 T 传入操作函数, 然后访问其左子树, 完成后访问

其右子树，完成后返回 OK

时间复杂度： $O(n)$

空间复杂度： $O(1)$

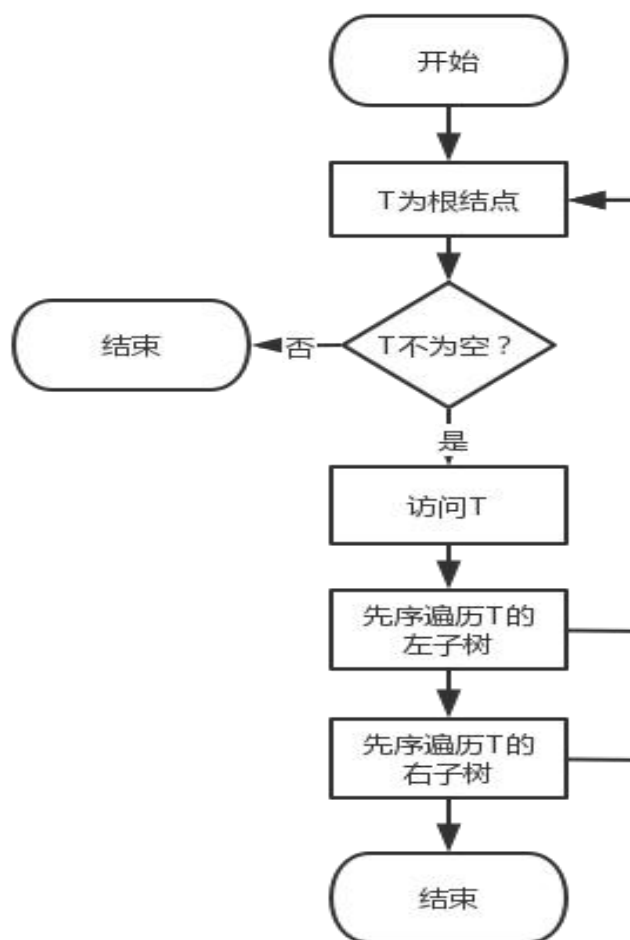


图 3-8 先序遍历流程图

(13) 中序遍历

函数名称：status InOrderTraverse(BiTree T, void (*visit)(BiTree))

输入：二叉树，操作函数指针

输出：函数执行状态

思想：设置一个栈存放所经过的根结点的信息；第一次访问到根结点时并不访问，而是入栈；中序遍历左子树，左子树遍历结束后，第一次遇到根结点，就将根结点退栈并访问，然后中序遍历它的右子树；当需要退栈时，如果栈为空就结束。

操作：定义一结点指针栈并置空。当 T 不为空时，T 入栈并让 T 指向其左子树，循环结束后，如果栈非空，就弹出栈顶的根结点 T 并访问，之后让 T 指向其右子树，当栈非空或者 T 不为空时，继续进行以上循环。循环结束后返回 OK

时间复杂度： $O(n)$

空间复杂度： $O(1)$

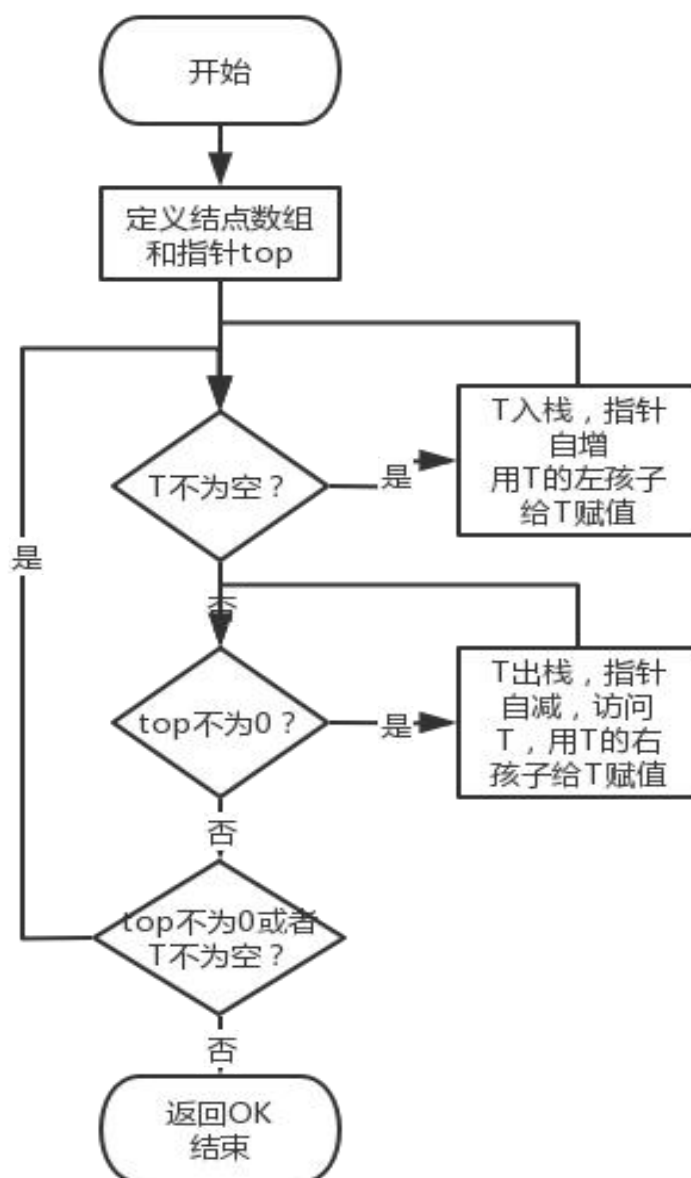


图 3-9 中序遍历流程图

(14) 后序遍历

函数名称: `status PostOrderTraverse(BiTree T, void (*visit) (BiTree))`

输入: 二叉树, 操作函数指针

输出: 函数执行状态

思想: 采用递归思想, 依次访问根结点, 左子树, 右子树

操作: 如果 T 不为空, 访问其左子树, 完成后访问其右子树, 完成后将 T 传入操作函数, 返回 OK

时间复杂度: $O(n)$

空间复杂度: $O(1)$

(15) 层序遍历

函数名称: `status LevelOrderTraverse(BiTree T, void (*visit) (BiTree))`

输入: 二叉树, 操作函数指针

输出: 函数执行状态

思想: 设置一个结点数组, 利用快慢指针, 快指针遍历所有结点并存入数组, 慢指针用于输出当前结点

操作: 设置一个结点数组 `pt`, 设置快指针 `in` 和慢指针 `out`, 初始化为 0, 将根结点赋值给 `pt[in++]`, 当 `in > out` 时执行下列循环: 当 `pt[out]` 不为空时, 访问 `pt[out]`, 将 `pt[out]` 的左孩子赋值给 `pt[in++]`, 将 `pt[out]` 的右孩子赋值给 `pt[in++]`; 之后 `out++`, 执行下一次循环。循环结束后返回 OK

时间复杂度: $O(n)$

空间复杂度: $O(1)$

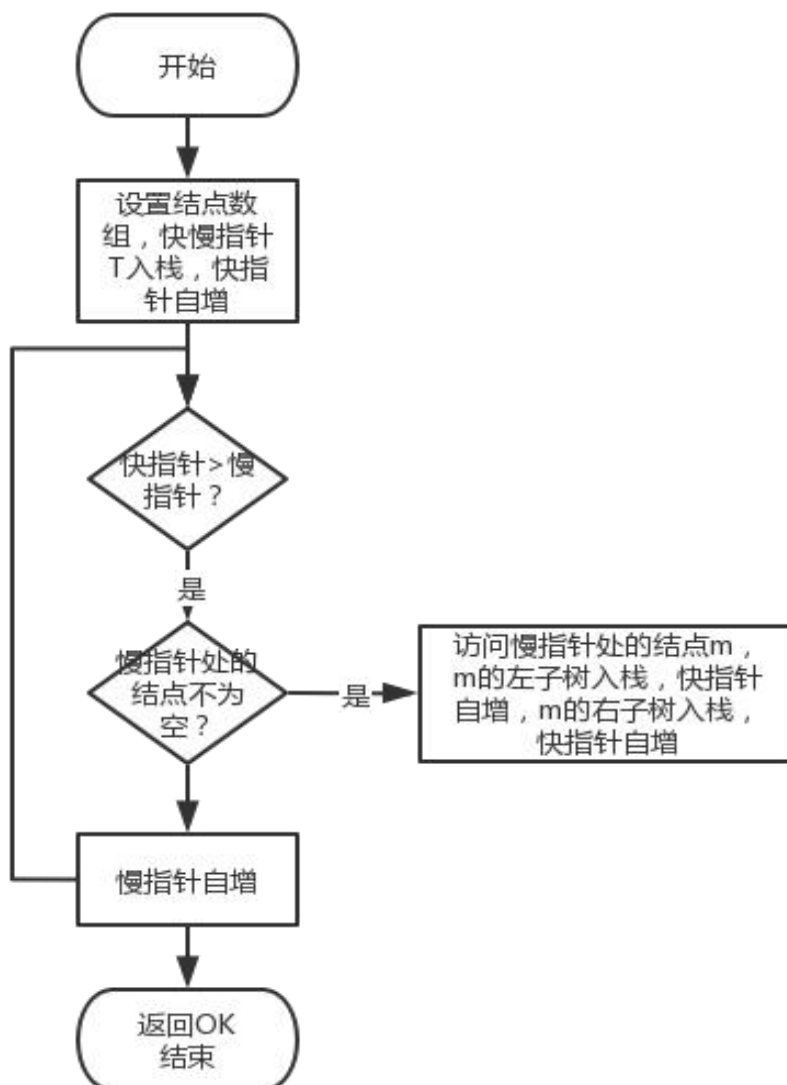


图 3-10 层序遍历流程图

3.4 系统测试

程序采用较为简单的界面，一级菜单和二级菜单分别如图 3-11，3-12 所示（在一级菜单中可通过操作 9 进入二级菜单）。本次系统测试选取了针对二叉树集合操作的 AddBiTree、RemoveBiTree、LocateBiTree 函数以及针对单二叉树操作的 LocateNode、Assign、GetSibling、InsertNode、DeleteNode、Traverse 函数。

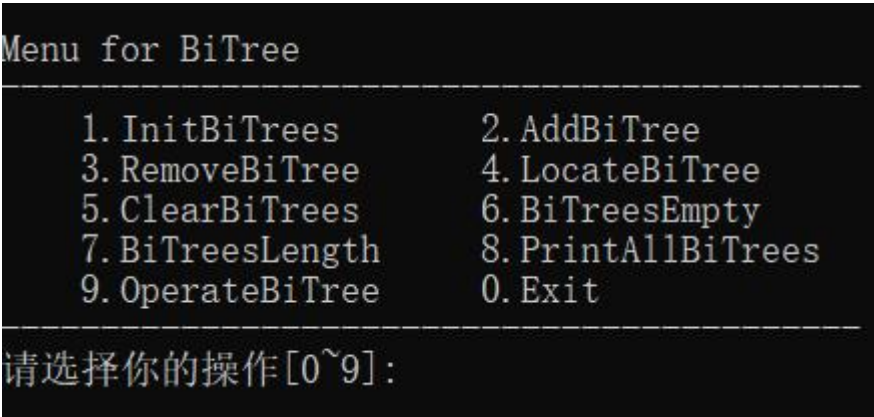


图 3-11 一级菜单示意图

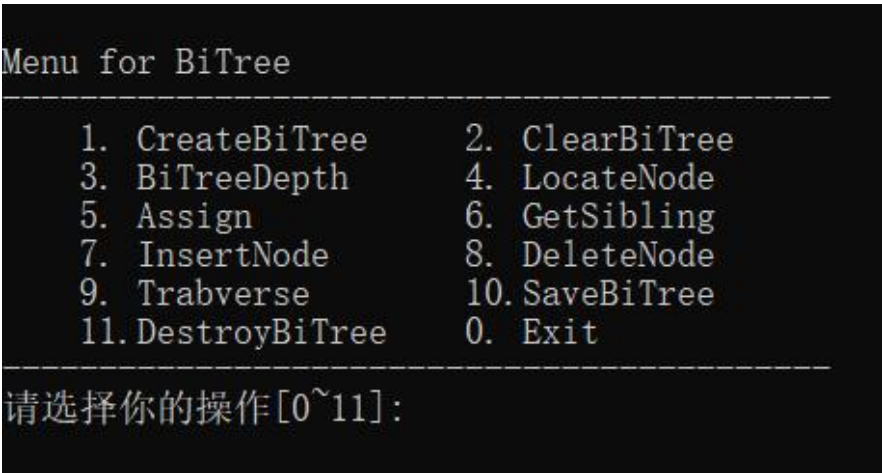


图 3-12 二级菜单示意图

下面是函数测试：

函数测试中使用的非空单二叉树 T 的先序遍历数据为

1 a 2 b 0 null 0 null 3 c 4 d 0 null 0 null 5 e 0
null 0 null -1 null

(1)添加二叉树的测试：

测试用例及结果如表 3-1 所示

表 3-1 添加二叉树测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1(集合中已有 11 二叉树)	名称: 11	名称有误！请重新 输入！	

用例 2（空集合）	名称：11	添加成功！	<pre> 请选择你的操作[0~9]:2 请输入需要添加的二叉树的名称: 11 添加成功! </pre>
-----------	-------	-------	--

综合上述测试，添加二叉树功能对于输出名称重复时以及未重复时都可以正确处理，所以添加二叉树功能符合实验要求。

(2) 删除二叉树的测试：

测试用例及结果如表 3-2 所示

表 3-2 删除二叉树测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (集合中已有 11 二叉树)	名称：11	删除成功！	<pre> 请选择你的操作[0~9]:3 请输入需要删除的二叉树的名称: 11 删除成功! </pre>
用例 2 (集合中已有 22 二叉树)	名称：11	二叉树集合中没有该二叉树！	<pre> 请选择你的操作[0~9]:3 请输入需要删除的二叉树的名称: 11 二叉树集合中没有该二叉树! </pre>
用例 3 (空集合)	名称：11	二叉树集合中没有该二叉树！	<pre> 请选择你的操作[0~9]:3 请输入需要删除的二叉树的名称: 11 二叉树集合中没有该二叉树! </pre>

综合上述测试，删除二叉树功能对于二叉树集合中有或无对应名称的二叉树时都可以正确处理，所以删除二叉树功能符合实验要求。

(3) 定位二叉树的测试

测试用例及结果如表 3-3 所示

表 3-3 定位二叉树测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1（空集合）	名称：11	二叉树集合中没有该二叉树！	<pre> 请选择你的操作[0~9]:4 请输入需要定位的二叉树的名称: 11 二叉树集合中没有该二叉树! </pre>

用例 2(集合中已有 11, 22 二叉树)	名称: 22	该二叉树位置是 2	<pre> 请选择你的操作[0~9]:4 请输入需要定位的二叉树的名称: 22 该二叉树的位置为2 </pre>
用例 3(集合中已有 11, 22 二叉树)	名称: 33	二叉树集合中没有该二叉树!	<pre> 请选择你的操作[0~9]:4 请输入需要定位的二叉树的名称: 33 二叉树集合中没有该二叉树! </pre>

综合上述测试, 定位二叉树功能对于二叉树集合中有或无对应名称的二叉树时都可以正确处理, 所以定位二叉树功能符合实验要求.

(4) 定位结点的测试

测试用例及结果如表 3-4 所示

表 3-4 定位结点测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (空二叉树)	关键字: 1	二叉树中没有该元素	<pre> 请选择你的操作[0~11]:4 请输入需要查找的元素的数值: 1 二叉树中没有该元素 </pre>
用例 2(二叉树 T)	关键字: 6	二叉树中没有该元素	<pre> 请选择你的操作[0~11]:4 请输入需要查找的元素的数值: 6 二叉树中没有该元素 </pre>
用例 3(二叉树 T)	关键字: 3	该结点数据为 3 c	<pre> 请选择你的操作[0~11]:4 请输入需要查找的元素的数值: 3 该结点数据为3 c </pre>

综合上述测试, 定位结点功能对于空二叉树, 二叉树中有无该元素时都能正确处理, 所以定位结点功能符合实验要求。

(5) 结点赋值的测试

测试用例及结果如表 3-5 所示

表 3-5 结点赋值测试及结果表

测试用例	程序输入	理论结果	运行结果
------	------	------	------

用例 1 (空二叉树)	关键字: 1 数据: 2 1	赋值失败	请选择你的操作[0~11]:5 请输入需要赋值的结点的数值: 1 请输入新的数值和串: 2 1 赋值失败
用例 2(二叉树 T)	关键字: 9 数据: 1 1	赋值失败	请选择你的操作[0~11]:5 请输入需要赋值的结点的数值: 9 请输入新的数值和串: 1 1 赋值失败
用例 3(二叉树 T)	关键字: 4 数据: 6 1	赋值成功	请选择你的操作[0~11]:5 请输入需要赋值的结点的数值: 4 请输入新的数值和串: 6 1 赋值成功
用例 4(二叉树 T)	关键字: 2 数据: 1 1	赋值失败	请选择你的操作[0~11]:5 请输入需要赋值的结点的数值: 2 请输入新的数值和串: 1 1 赋值失败

综合上述测试, 结点赋值功能对于空二叉树, 二叉树中有无该元素, 新关键字是否重复时都能正确处理, 所以结点赋值功能符合实验要求。

(6) 获得兄弟结点的测试

测试用例及结果如表 3-6 所示

表 3-6 获得兄弟结点测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (空二叉树)	关键字: 1	获取失败	请选择你的操作[0~11]:6 请输入需要获取其兄弟结点的结点数值: 1 获取失败
用例 2(二叉树 T)	关键字: 6	获取失败	请选择你的操作[0~11]:6 请输入需要获取其兄弟结点的结点数值: 6 获取失败
用例 3(二叉树 T)	关键字: 1	获取失败	请选择你的操作[0~11]:6 请输入需要获取其兄弟结点的结点数值: 1 获取失败
用例 4(二叉树 T)	关键字: 2	兄弟结点的数据 为: 3 c	请选择你的操作[0~11]:6 请输入需要获取其兄弟结点的结点数值: 2 兄弟结点的数据为: 3 c

综合上述测试，获取兄弟结点功能对于空二叉树，二叉树中有无该元素，该结点是否有兄弟结点时都能正确处理，所以获取兄弟结点功能符合实验要求。

（7）插入结点的测试

测试用例及结果如表 3-7 所示

表 3-7 插入结点测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1（空二叉树）	1 1 1 1	插入失败	<pre> 请选择你的操作[0~11]:7 请输入插入位置的数据、方式、待插入结点的数值和串: 1 1 1 1 插入失败 </pre>
用例 2(二叉树 T)	2 -1 2 w	插入成功	<pre> 请选择你的操作[0~11]:7 请输入插入位置的数据、方式、待插入结点的数值和串: 2 -1 2 w 插入成功 </pre>
用例 3(二叉树 T)	3 1 9 e	插入成功	<pre> 请选择你的操作[0~11]:7 请输入插入位置的数据、方式、待插入结点的数值和串: 3 1 9 e 插入成功 </pre>
用例 4(二叉树 T)	6 1 2 v	插入失败	<pre> 请选择你的操作[0~11]:7 请输入插入位置的数据、方式、待插入结点的数值和串: 6 1 2 v 插入失败 </pre>
用例 5（二叉树 T）	4 0 8 m	插入成功	<pre> 请选择你的操作[0~11]:7 请输入插入位置的数据、方式、待插入结点的数值和串: 4 0 8 m 插入成功 </pre>

综合上述测试，插入结点功能对于空二叉树，二叉树中有无该元素，输入不同插入方式时都能正确处理，所以获取插入结点功能符合实验要求。

（8）删除结点的测试

测试用例及结果如表 3-8 所示

表 3-8 删除结点测试及结果表

测试用例	程序输入	理论结果	运行结果
------	------	------	------

用例 1 (空二叉树)	关键字: 1	删除失败	请选择你的操作[0~11]:8 请输入需要删除的结点的数值: 1 删除失败
用例 2(二叉树 T)	关键字: 9	删除失败	请选择你的操作[0~11]:8 请输入需要删除的结点的数值: 9 删除失败
用例 3(二叉树 T)	关键字: 4	删除成功	请选择你的操作[0~11]:8 请输入需要删除的结点的数值: 4 删除成功
用例 4 (只有一个结点 1)	关键字: 1	删除成功	请选择你的操作[0~11]:8 请输入需要删除的结点的数值: 1 删除成功

综合上述测试, 删除结点功能对于空二叉树, 二叉树中有无该元素, 二叉树中只有根结点时都能正确处理, 所以删除结点功能符合实验要求。

(9) 遍历二叉树的测试

测试用例及结果如表 3-9 所示

表 3-9 遍历二叉树测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (空二叉树)	无	二叉树中没有元素!	请选择你的操作[0~11]:9 二叉树中没有元素!

用例 2(二叉树 T)	无	输出四种遍历的结果	<pre> 请选择你的操作[0~11]:9 前序: 1 a 2 b 3 c 4 d 5 e 中序: 2 b 1 a d c e 4 c 3 e 5 e 后序: 2 b d e c a 4 c 5 e 1 a 层序: 1 a 2 b 3 c 4 d 5 e </pre>
-------------	---	-----------	---

综合上述测试，遍历二叉树功能对于空二叉树，非空二叉树都能正确处理，所以遍历二叉树功能符合实验要求。

总结：综合上述所有函数的测试，测试结果都未发现问题，符合本次实验的要求，也较完整较正确地完成了题目要求。

3.5 实验小结

本次实验与前两次有很大的不同，而且也有一定的难度。但让我更加了解了递归函数的写法。

同时自己也存在很多问题。

在写创建二叉树的函数时，由于不清楚如何在递归函数中传入数组，导致出现了很多问题。最后使用了一个指向数组的全局指针，通过改变指针的指向来实现数组的依次访问。同时，自己在写删除结点的函数时，没有考虑到只有根结点的情况，导致在验收时出现了问题。而且自己很多次也没有考虑到指针判空的问题，导致调用结点的左孩子或者右孩子但指向空时出现了问题。而且我的文件操作也不太熟练，在同学的指导之下完成了本次实验的文件操作。

最后感谢老师和同学的指导，让我顺利完成本次实验。

4 基于邻接表的图实现

4.1 问题描述

构造无向图，呈现一个简易菜单的功能演示系统，该演示系统可选择实现多个无向图管理以及单无向图的操作。

4.1.1 实验目的

- (1) 加深对无向图的概念、基本运算的理解；
- (2) 熟练掌握无向图的逻辑结构与物理结构的关系；
- (3) 物理结构采用邻接表, 熟练掌握无向图的基本运算的实现。

4.1.2 实验要求

需要在主程序中完成函数调用以及所需实参值和函数执行结果的输出。定义无向图的创建、销毁、定位顶点、顶点赋值、获得第一邻接结点、获得第二邻接顶点、插入顶点、删除顶点、插入弧、删除弧、深度优先搜索、广度优先搜索等函数及集合的初始化、销毁、清空、判空、求表长和获得元素等函数，并给出适当的操作提示，并且可选择以文件的形式进行存储和加载。

4.2 系统设计

4.2.1 整体系统结构设计

本演示系统可以实现通过操作菜单对多无向图进行管理，并对单个无向图进行操作，也支持通过文件的写入与读取。

多无向图管理菜单（无向图集合菜单/一级菜单）可供选择的操作有：初始化无向图集合、添加一个无向图、删除特定名称的无向图、查找特定名称的无向图、清空无向图集合、无向图集合判空、求无向图集合中无向图的个数、打印所有无向图的序号及名称、操作特定序号的无向图。

多无向图管理菜单中的“操作特定序号的无向图”可以调出单无向图操作菜单（二级菜单），可供选择的操作有：创建无向图、销毁无向图、定位某顶点、顶点赋值、求某顶点的第一、第二邻接顶点、插入顶点或弧、删除顶点或弧、深度优先搜索、广度优先搜索、将无向图的数据写入文件、通过键盘输入或者读取

文件数据向无向图中一次性输入若干元素。整体系统结构设计示意图如下：

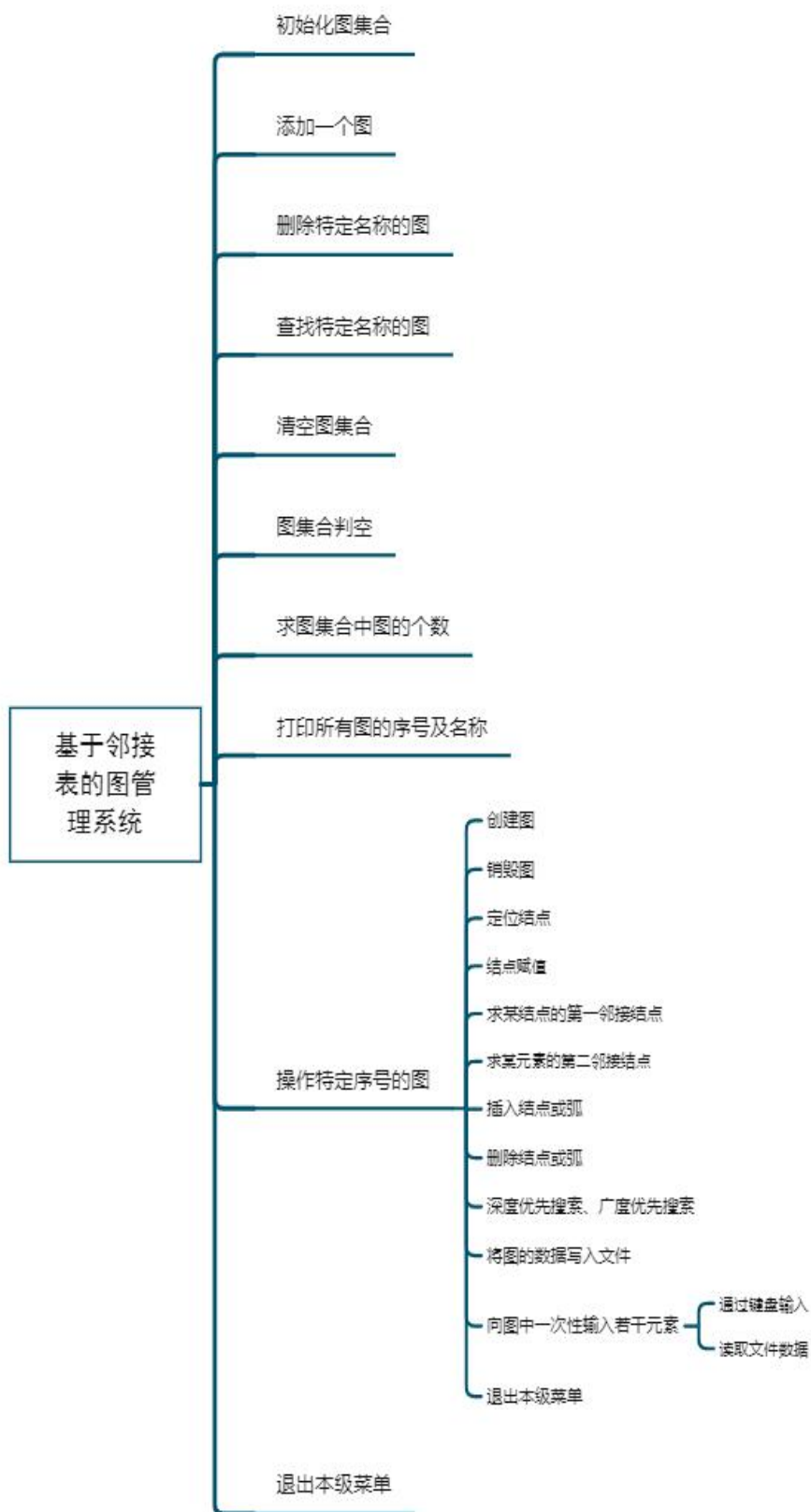


图 4-1 整体系统结构设计

4.2.2 数据结构设计

本演示系统设计了五个物理结构，分别是顶点、表结点、头结点、邻接表和图集合，均以结构形式定义。

顶点结构中包括关键字与字符串；表结点结构中包括顶点位置编号和下一个表结点指针；头结点结构中包括顶点数据和指向第一条弧的表结点指针；邻接表结构中包括头结点数组、顶点数、弧数、图的类型；图集合结构中包括图数组（每个数组元素都包括图以及其名称）和图集合长度。

结构定义如下：

```
typedef struct {
    KeyType key;
    char others[20];
} VertexType; //顶点类型定义
typedef struct ArcNode { //表结点类型定义
    int adjvex; //顶点位置编号
    struct ArcNode *nextarc; //下一个表结点指针
} ArcNode;
typedef struct VNode { //头结点及其数组类型定义
    VertexType data; //顶点信息
    ArcNode *firstarc; //指向第一条弧
} VNode, AdjList[MAX_VERTEX_NUM];
typedef struct { //邻接表的类型定义
    AdjList vertices; //头结点数组
    int vexnum;
    int arcnum; //顶点数、弧数
    GraphKind kind; //图的类型
} ALGraph;
typedef struct { //图的集合类型定义
    struct {
        char name[30]; //图名称
        ALGraph G; //图
    } elem[10]; //图数组
    int length; //图集合长度
} ALGraphs;
```

图 4-2 数据结构设计

依据最小完备性和常用性相结合的原则，以函数形式定义了创建图、销毁图、

查找顶点、获得顶点值和顶点赋值等 12 种基本运算。具体运算功能定义和说明如下。具体运算功能定义如下。

(1) 创建图：函数名称是 `CreateGraph(G,V,VR)`；初始条件是 V 是图的顶点集， VR 是图的关系集；操作结果是按 V 和 VR 的定义构造图 G 。

注：①要求图 G 中顶点关键字具有唯一性。后面各操作的实现，也都要满足一个图中关键字的唯一性，不再赘述；② V 和 VR 对应的是图的逻辑定义形式，比如 V 为顶点序列， VR 为关键字对的序列。不能将邻接矩阵等物理结构来代替 V 和 VR 。

(2) 销毁图：函数名称是 `DestroyGraph(G)`；初始条件图 G 已存在；操作结果是销毁图 G 。

(3) 查找顶点：函数名称是 `LocateVex(G,u)`；初始条件是图 G 存在， u 是和 G 中顶点关键字类型相同的给定值；操作结果是若 u 在图 G 中存在，返回关键字为 u 的顶点位置序号（简称位序），否则返回其它表示“不存在”的信息。

(4) 顶点赋值：函数名称是 `PutVex (G,u,value)`；初始条件是图 G 存在， u 是和 G 中顶点关键字类型相同的给定值；操作结果是对关键字为 u 的顶点赋值 $value$ 。

(5) 获得第一邻接点：函数名称是 `FirstAdjVex(G, u)`；初始条件是图 G 存在， u 是 G 中顶点的位序；操作结果是返回 u 对应顶点的第一个邻接顶点位序，如果 u 的顶点没有邻接顶点，否则返回其它表示“不存在”的信息。

(6) 获得下一邻接点：函数名称是 `NextAdjVex(G, v, w)`；初始条件是图 G 存在， v 和 w 是 G 中两个顶点的位序， v 对应 G 的一个顶点， w 对应 v 的邻接顶点；操作结果是返回 v 的（相对于 w ）下一个邻接顶点的位序，如果 w 是最后一个邻接顶点，返回其它表示“不存在”的信息。

(7) 插入顶点：函数名称是 `InsertVex(G,v)`；初始条件是图 G 存在， v 和 G 中的顶点具有相同特征；操作结果是在图 G 中增加新顶点 v 。（在这里也保持顶点关键字的唯一性）

(8) 删除顶点：函数名称是 `DeleteVex(G,v)`；初始条件是图 G 存在， v 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中删除关键字 v 对应的顶点以及相关的弧。

(9) 插入弧：函数名称是 `InsertArc(G,v,w)`；初始条件是图 G 存在， v 、 w 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中增加弧 $\langle v,w \rangle$ ，如果图

G 是无向图，还需要增加<w,v>。

(10)删除弧：函数名称是 DeleteArc(G,v,w)；初始条件是图 G 存在，v、w 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中删除弧<v,w>，如果图 G 是无向图，还需要删除<w,v>。

(11)深度优先搜索遍历：函数名称是 DFSTraverse(G,visit())；初始条件是图 G 存在；操作结果是图 G 进行深度优先搜索遍历，依次对图中的每一个顶点使用函数 visit 访问一次，且仅访问一次。

(12)广深度优先搜索遍历：函数名称是 BFSTraverse(G,visit())；初始条件是图 G 存在；操作结果是图 G 进行广度优先搜索遍历，依次对图中的每一个顶点使用函数 visit 访问一次，且仅访问一次。

4.2.3 有关常量和类型定义

数据元素类型的定义：

```
typedef int status;
typedef int KeyType;
typedef enum { DG, DN, UDG, UDN } GraphKind;
typedef struct {
    KeyType key;
    char others[20];
} VertexType; //顶点类型定义
```

有关常量的定义：

```
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASTABLE -1
#define OVERFLOW -2
#define MAX_VERTEX_NUM 20
```

4.3 系统实现

本演示系统在 Windows 环境下，使用 Visual Studio 2019 完成。演示系统

涉及的部分主要函数如下，其中（1）-（4）为对图集合的操作，（5）-（15）为对单无向图的操作。

(1) 向无向图集合中添加一个空无向图

函数名称：status AddALGraph(ALGraphs* ALGraphs, char ALGraphName[])

输入：无向图集合指针以及需要添加的无向图的名称

输出：函数执行状态

思想：在 Lists 中加入一个名称为 ListName 无向图，成功则返回 OK

操作：

将 ListName 数组拷贝到无向图集合中无向图数组的最后一个元素之后的元素的 name 成员变量中；无向图集合长度加一；结束，返回 OK。

时间复杂度：O(1)

空间复杂度：O(1)

(2) 删除无向图集合中指定名称的无向图

函数名称：status RemoveALGraph(ALGraphs* ALGraphs, char ALGraphName[])

输入：无向图集合指针以及需要删除的无向图名称

输出：函数执行状态

思想：删除指定名称的无向图，删除成功则返回 OK，否则返回 ERROR

操作：

定义 i=0，当 i 小于无向图集合长度时，执行下列循环：

a. 比较第 i 个无向图的名称和所给名称

b. 当两者相同时，将第 i 个无向图的名称指针的首元素置为 '\0'，将无向图数组中该无向图后面的无向图依次前移，无向图集合的长度减一，返回 OK

若循环结束后函数没有返回值，说明集合中没有该名称的无向图，返回 ERROR

时间复杂度：O(n)（设无向图集合长度为 n）

空间复杂度：O(1)

(3) 查找指定名称的无向图在集合中的位置

函数名称: `status LocateALGraph(ALGraphs ALGraphs, char ALGraphName[])`

输入: 无向图集合以及需要查找的无向图名称

输出: 该无向图的序号或者函数执行状态

思想: 在集合中寻找指定名称的无向图, 成功则返回其逻辑序号, 否则返回 ERROR

操作:

① 定义 $i=0$, 当 i 小于无向图集合长度时, 执行下列循环:

a. 比较第 i 个无向图名称与所给名称

b. 如果相同, 返回 $i+1$, 否则继续循环

② 如果循环执行完后函数没有返回值, 说明集合中没有该名称的无向图, 返回 ERROR

时间复杂度: $O(n)$ (设无向图长度为 n , 此处为最坏情况)

空间复杂度: $O(1)$

(4) 清空无向图集合

函数名称: `status ClearALGraphs(ALGraphs* ALGraphs)`

输入: 无向图集合指针

输出: 函数执行状态

思想: 将无向图集合的长度变成 0, 返回 OK

操作: 将 Lists 的 length 赋值为 0, 返回 OK

时间复杂度: $O(1)$

空间复杂度: $O(1)$

(5) 创建无向图

函数名称: `status CreateGraph(ALGraph &G, VertexType V[], KeyType VR[][2])`

输入：无向图、顶点数据数组、弧数据数组

输出：函数执行状态

操作：

首先输入顶点数据，当输入的关键字不为-1 且没有重复时，执行循环，将数据放入顶点数组并将对应顶点的首弧指针置空，顶点数增加。之后输入表结点（弧）数据，当输入不为-1 时，在每条边涉及的两个顶点对应的链表上头插对应结点，并保存对应节点的位序，弧数增加，返回 OK，其他情况表明输入数据有误，返回 ERROR。

时间复杂度： $O(n)$

空间复杂度： $O(1)$

(6) 顶点赋值

函数名称：status PutVex(ALGraph &G, KeyType u, VertexType value)

输入：图、目标顶点的关键字、待赋值的数据

输出：函数执行状态

思想：当输入的目标关键字和数据关键字符合要求时，遍历找到顶点并赋值。

操作：当输入的顶点不存在或者待赋值数据的关键字重复时，返回 ERROR，否则遍历顶点数组，找到关键字等于 u 的结点并赋值，返回 OK。

时间复杂度： $O(n)$

空间复杂度： $O(1)$

(7) 获得第一邻接顶点

函数名称：int FirstAdjVex(ALGraph G, KeyType u)

输入：图，需要获取第一邻接顶点的结点的关键字

输出：第一邻接顶点位序

思想：遍历顶点数组，找到对应顶点后输出第一邻接顶点。

操作：遍历顶点数组，如果当前结点的关键字等于 u，如果该顶点的首弧为 NULL，返回-1，否则返回首弧指向顶点的位置；如果循环结束，说明数组中没有关键字为 u 的顶点，返回-1。

时间复杂度: $O(n)$

空间复杂度: $O(1)$

(8) 获得第二邻接结点

函数名称: `int NextAdjVex(ALGraph G, KeyType v, KeyType w)`

输入: 图、目标顶点的关键字, 相对顶点的关键字

输出: 第二邻接顶点位序

思想: 遍历找到目标顶点, 遍历对应链表找到关键字为 w 的结点。返回其下一个顶点的位序。

操作: 遍历顶点数组, 找到关键字为 w 的顶点的位序, 并同时判断图中是否有关键字为 v 的顶点, 没有则返回-1。否则遍历关键字为 v 的顶点的表结点链表, 当当前顶点的位序等于 w 的位序时, 如果它的下一个表结点为 NULL, 返回-1, 否则返回下一个表结点的位序, 返回 OK。

时间复杂度: $O(n)$

空间复杂度: $O(1)$

(9) 插入顶点

函数名称: `status InsertVex(ALGraph &G, VertexType v)`

输入: 图、插入顶点的数据

输出: 函数执行状态

思想: 当插入数据的关键字符合要求时, 放入顶点数组。

操作: 当插入数据的关键字重复或者顶点数组已满时, 返回 ERROR; 否则将 v 的 key 赋值给顶点数组第一个空结点的关键字, 将 v 的 others 拷贝给空结点的字符串, 顶点的首弧置为 NULL, 图的顶点数加一, 返回 OK。

时间复杂度: $O(n)$

空间复杂度: $O(1)$

(10) 删除顶点

函数名称: `status DeleteVex(ALGraph &G, KeyType v)`

输入：图，要删除的顶点的关键字

输出：函数执行状态

思想：删除关键字为 v 的结点和与之相关的边。

操作：如果要删除的结点是图中唯一一个顶点，返回 ERROR；遍历顶点数组，如果没有关键字为 v 的顶点，返回 ERROR；对每个顶点后面的表结点链表，清空即可（简单的链表操作，不做赘述）在清空过程中，每删除一个结点，弧数减一，然后顶点数组中让删除的顶点的后面的顶点前移；顶点数减一，之后再次遍历顶点数组，将表结点位置数据等于被删除顶点的位序的结点删除，将表结点位置数据大于被删除顶点的位序的结点的位置数据减一，返回 OK。

时间复杂度： $O(m*n)$

空间复杂度： $O(1)$

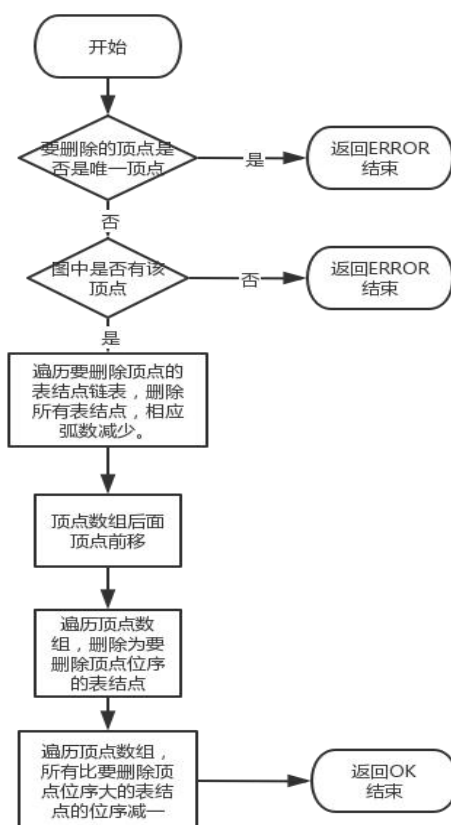


图 4-3 删除顶点流程图

(11) 插入弧

函数名称: `status InsertArc(ALGraph &G,KeyType v,KeyType w)`

输入: 图、新添加弧的两个顶点的关键字

输出: 函数执行状态

思想: 当输入的关键字符符合要求时, 在两个顶点后头插对应含对应顶点位序的结点。

操作: 遍历顶点数组, 记下关键字为 v , w 的顶点的位序 i , j ; 如果数组中没有顶点 v 或者 w , 返回 ERROR; 遍历对应顶点的表结点链表, 如果已有要插入的弧, 返回 ERROR; 否则在两个顶点后头插对应顶点。返回 OK。

时间复杂度: $O(n)$

空间复杂度: $O(1)$

(12) 删除弧

函数名称: `status DeleteArc(ALGraph &G,KeyType v,KeyType w)`

输入: 图、需要删除弧的两个顶点 v , w

输出: 函数执行状态

思想: 当输入关键字符合要求时, 遍历目标顶点链表, 删除对应表结点。

操作: 遍历顶点数组, 记下关键字为 v , w 的顶点的位序 i , j ; 如果数组中没有顶点 v 或者 w , 返回 ERROR; 遍历对应顶点的表结点链表, 如果没有要删除的弧, 返回 ERROR; 否则遍历两个顶点的表结点链表, 删除对应的表结点, 弧数减一, 返回 OK。

时间复杂度: $O(m*n)$

空间复杂度: $O(1)$

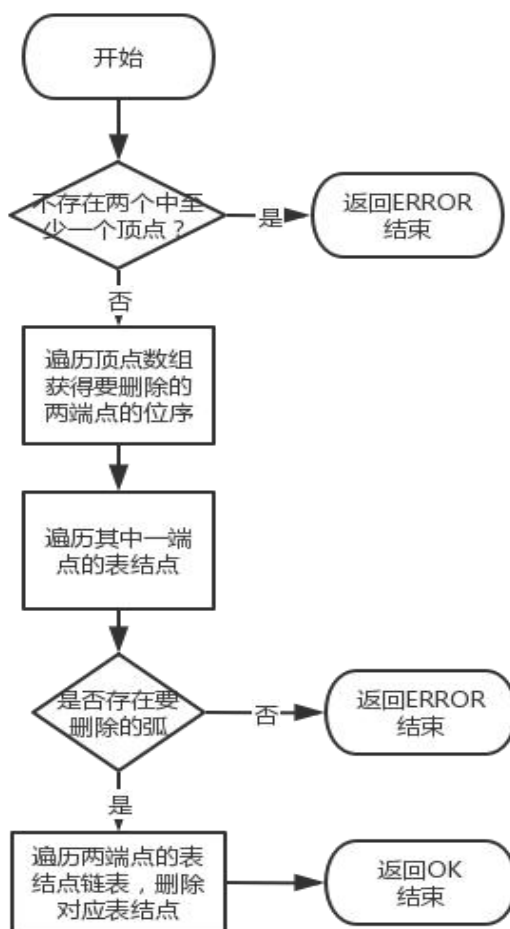


图 4-4 删除弧流程图

(13) 深度优先搜索

函数名称: `status DFSTraverse(ALGraph &G, void (*visit) (VertexType))`

`void DFS(ALGraph &G, int v)`

输入: 图、访问函数指针、操作顶点

输出: 函数执行状态

思想: 从选定顶点开始, 任意选取其一邻居节点, 沿一条路径直到无路可走, 就回退到上一个顶点, 寻找另一条可行路径, 直到全部顶点均被访问一次之后结束。

操作: 设置一个访问标记数组, 设置循环遍历顶点数组, 从第一个顶点开始, 如果第一个顶点未被访问, 就对其进行深度优先搜索, 并标记, 设置循环遍历其邻接结点, 如果未被访问, 就对其进行深度优先搜索。所有循环都结束之后返回

OK。

时间复杂度： $O(n+m)$

空间复杂度： $O(d)$

(14) 广度优先搜索

函数名称：status BFSTraverse(ALGraph &G, void (*visit)(VertexType))

输入：图、访问指针函数

思想：从选定顶点开始，遍历其邻接结点，对于每一个邻接结点，再次遍历它的邻接结点，直到所有结点都被访问完毕。

操作：设置一个队列和标记数组。循环遍历顶点数组，如果当前结点未被访问，就访问该结点并标记，同时该结点入队，当队列不为空时，队首结点出队，依次遍历该结点的邻接结点，如果邻接结点未被访问，就访问邻接结点并标记，之后将节点入栈。直到循环结束，返回 OK。

时间复杂度： $O(n+m)$

空间复杂度： $O(1)$

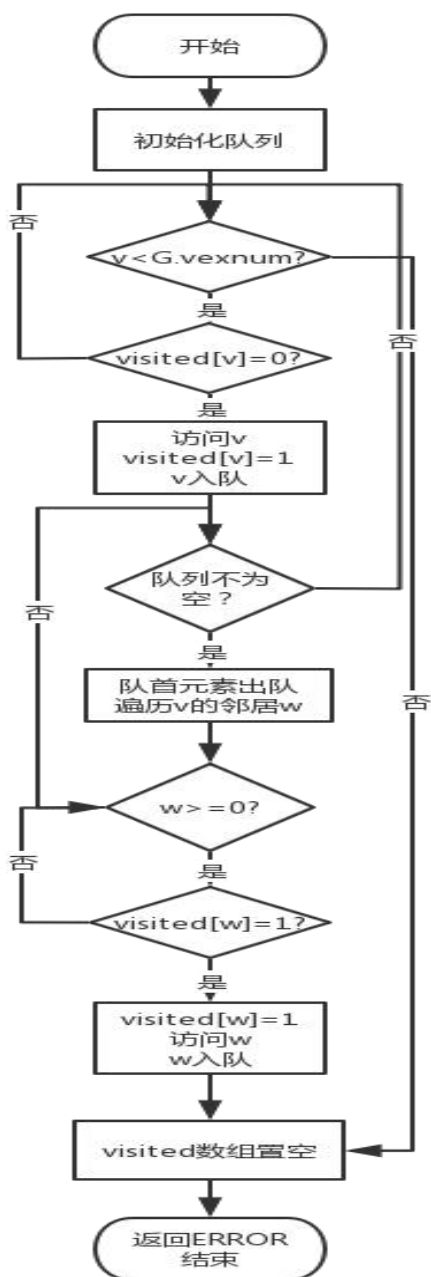


图 4-5 BFS 流程图

4.4 系统测试

程序采用较为简单的界面，一级菜单和二级菜单分别如图 4-6，4-7 所示（在一级菜单中可通过操作 9 进入二级菜单）。本次系统测试选取了针对无向图集合操作的 AddALGraph、RemoveALGraph、LocateALGraph 函数以及针对单无向图操作的 PutVex、NextAdjVex、InsertVex、DeleteArc、DFSTraverse、BFSTraverse

函数。

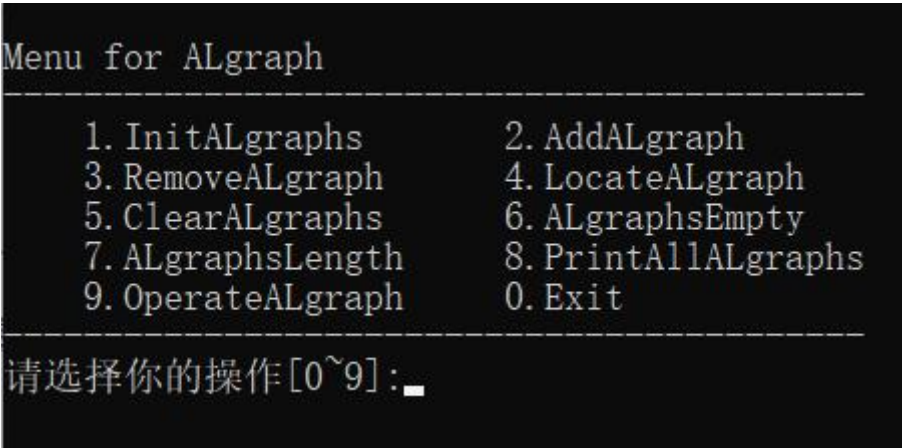


图 4-6 一级菜单示意图

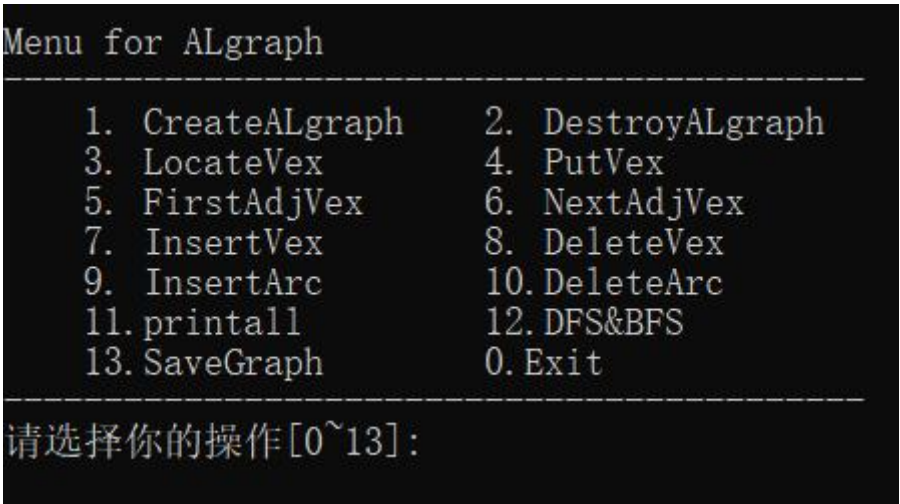


图 4-7 二级菜单示意图

下面是函数测试：

函数测试中使用的非空图 G 的数据为

5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1

(1)添加无向图的测试：

测试用例及结果如表 4-1 所示

表 4-1 添加无向图测试及结果表

测试用例	程序输入	理论结果	运行结果
------	------	------	------

用例 1(集合中已有 11 无向图)	名称: 11	名称重复! 请重新 输入!	请选择你的操作[0~9]:2 请输入需要添加的图的名称: 11 名称重复! 请重新输入!
用例 2(空集合)	名称: 11	添加成功!	请选择你的操作[0~9]:2 请输入需要添加的图的名称: 11 添加成功!

综合上述测试,添加无向图功能对于输出名称重复时以及未重复时都可以正确处理,所以添加无向图功能符合实验要求。

(2) 删除无向图的测试:

测试用例及结果如表 4-2 所示

表 4-2 删除无向图测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1(集合中已有 11 无向图)	名称: 11	删除成功!	请选择你的操作[0~9]:3 请输入需要删除的图的名称: 11 删除成功!
用例 2(集合中已有 22 无向图)	名称: 11	图集合中没有该 图!	请选择你的操作[0~9]:3 请输入需要删除的图的名称: 11 图集合中没有该图!
用例 3(空集合)	名称: 11	图集合中没有该 图!	请选择你的操作[0~9]:4 请输入需要定位的图的名称: 11 图集合中没有该图!

综合上述测试,删除无向图功能对于无向图集合中有或无对应名称的无向图时都可以正确处理,所以删除无向图功能符合实验要求。

(3) 定位无向图的测试

测试用例及结果如表 4-3 所示

表 4-3 定位无向图测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (空集合)	名称: 11	无向图集合中没有该无向图!	请选择你的操作[0~9]:4 请输入需要定位的图的名称: 11 图集中没有该图!
用例 2(集合中已有 11, 22 无向图)	名称: 22	该无向图位置是 2	请选择你的操作[0~9]:4 请输入需要定位的图的名称: 22 该图的位置为2
用例 3(集合中已有 11, 22 无向图)	名称: 33	无向图集合中没有该无向图!	请选择你的操作[0~9]:4 请输入需要定位的图的名称: 33 图集中没有该图!

综合上述测试, 定位无向图功能对于无向图集合中有或无对应名称的无向图时都可以正确处理, 所以定位无向图功能符合实验要求.

(4) 顶点赋值的测试

测试用例及结果如表 4-4 所示

表 4-4 结点赋值测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (空图)	无	图为空	请选择你的操作[0~13]:4 图为空!
用例 2(图 G)	关键字: 5 数据: 1 a	赋值成功	请选择你的操作[0~13]:4 请输入需要赋值的结点的数值: 5 请输入新的数值和串: 1 a 赋值成功
用例 3(图 G)	关键字: 5 数据: 5 a	赋值成功	请选择你的操作[0~13]:4 请输入需要赋值的结点的数值: 5 请输入新的数值和串: 5 a 赋值成功
用例 4(图 G)	关键字: 5 数据: 6 a	赋值失败	请选择你的操作[0~13]:4 请输入需要赋值的结点的数值: 5 请输入新的数值和串: 6 a 赋值失败

用例 4(图 G)	关键字: 9 数据: 1 1	赋值失败	请选择你的操作[0~13]:4 请输入需要赋值的结点的数值: 9 请输入新的数值和串: 1 1 赋值失败
-----------	-------------------	------	---

综合上述测试, 顶点赋值功能对于空图, 图中有无该元素, 新关键字是否重复时都能正确处理, 所以结点赋值功能符合实验要求。

(5) 获得第二邻接结点的测试

测试用例及结果如表 4-5 所示

表 4-5 获得第二邻接结点测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (空图)	无	图为空	请选择你的操作[0~13]:6 图为空!
用例 2(图 G)	关键字: 5 7	该结点位序为 3 数据为 6 无向图	请选择你的操作[0~13]:6 请输入查找顶点和相对顶点: 5 7 该结点位序为3, 该结点数据为 6 无向图
用例 3(图 G)	关键字: 5 6	获取失败	请选择你的操作[0~13]:6 请输入查找顶点和相对顶点: 5 6 获取失败
用例 4(图 G)	关键字: 9 5	获取失败	请选择你的操作[0~13]:6 请输入查找顶点和相对顶点: 9 5 获取失败

综合上述测试, 获取兄弟结点功能对于空图, 图中有无该元素, 该结点是否有第二邻接结点时都能正确处理, 所以获取第二邻接结点功能符合实验要求。

(6) 插入顶点的测试

测试用例及结果如表 4-6 所示

表 4-6 插入顶点测试及结果表

测试用例	程序输入	理论结果	运行结果
------	------	------	------

用例 1（空图）	无	图为空	请选择你的操作[0~13]:7 图为空!
用例 4(图 G)	5 a	插入失败	请选择你的操作[0~13]:7 请输入待插入结点的数值和串: 5 a 插入失败
用例 5（图 G）	1 a	插入成功	请选择你的操作[0~13]:7 请输入待插入结点的数值和串: 1 a 插入成功

综合上述测试，插入顶点功能对于空图，图中有无该元素时都能正确处理，所以获取插入顶点功能符合实验要求。

（7）删除弧的测试

测试用例及结果如表 4-7 所示

表 4-7 删除弧测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1（空图）	无	图为空	请选择你的操作[0~13]:10 图为空!
用例 2(图 G+顶点 1 a)	1 5	删除失败	请选择你的操作[0~13]:10 请输入需要删除的弧的两个顶点: 1 5 删除失败
用例 3(图 G)	5 8	删除失败	请选择你的操作[0~13]:10 请输入需要删除的弧的两个顶点: 5 8 删除失败
用例 4（图 G）	5 6	删除成功	请选择你的操作[0~13]:10 请输入需要删除的弧的两个顶点: 5 6 删除成功

综合上述测试，删除弧功能对于空图，图中有无该弧时都能正确处理，所以删除弧功能符合实验要求。

(8) DFS&BFS 的测试

测试用例及结果如表 4-8 所示

表 4-8DFS&BFS 测试及结果表

测试用例	程序输入	理论结果	运行结果
用例 1 (空图)	无	图为空	
用例 2(图 G)	无	DFS&BFS 的结果	

综合上述测试, DFS&BFS 功能对于空图, 非空图都能正确处理, 所以 DFS&BFS 功能符合实验要求。

总结: 综合上述所有函数的测试, 测试结果都未发现问题, 符合本次实验的要求, 也较完整较正确地完成了题目要求。

4.5 实验小结

本次实验较为顺利, 没有出现太大的错误。

同时自己也存在很多问题。

由于本次实验的邻接表中也涉及到了链表, 但链表的删除结点等功能不是太熟练, 导致经常出现指针乱用的错误。涉及到删除结点和弧等的函数里面, 总是忘记把图中弧的个数减少, 导致头歌好几次提交不通过。还有很多函数的一些特殊情况没有考虑周全, 导致验收的时候没有通过特殊样例。

这次实验让我再次认识到了算法健壮性的重要, 以及写程序时一定要细心和考虑周全, 这样才能完美通过。

最后感谢老师和同学的指导, 让我顺利完成本次实验。

参考文献

- [1] 严蔚敏等.数据结构(C语言版).清华大学出版社
- [2] Larry Nyhoff. ADTs, Data Structures, and Problem Solving with C++.Second Edition, Calvin College, 2005
- [3] 殷立峰. Qt C++跨平台图形界面程序设计基础. 清华大学出版社,2014:192~197
- [4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社