

華中科技大學

課程實驗報告

課程名稱： _____ 數據結構實驗 _____

專業班級： _____

學 號： _____

姓 名： _____

指導教師： _____

報告日期： _____

計算機科學與技術學院

目 录

1 基于顺序存储结构的线性表实现	2
1.1 问题描述.....	2
1.2 系统设计.....	2
1.3 系统实现.....	6
1.4 实验小结.....	25
2 基于链式存储结构的线性表实现	27
2.1 问题描述.....	27
2.2 系统设计.....	27
2.3 系统实现.....	31
2.4 实验小结.....	52
3 基于二叉链表的二叉树实现	53
3.1 问题描述.....	53
3.2 系统设计.....	53
3.3 系统实现.....	55
3.4 实验小结.....	83
4 基于二叉链表的二叉树实现	85
4.1 问题描述.....	85
4.2 系统设计.....	85
4.3 系统实现.....	89
4.4 实验小结.....	109
参考文献	111
附录 A 基于顺序存储结构线性表实现的源程序	112
附录 B 基于链式存储结构线性表实现的源程序	122
附录 C 基于二叉链表二叉树实现的源程序	132
附录 D 基于邻接表图实现的源程序	147

1 基于顺序存储结构的线性表实现

1.1 问题描述

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,熟练掌握线性表的基本运算的实现。

1.1.1 线性表的顺序表实现

通过函数形式实现线性表的初始化、销毁、清空、判空、求表长、获得元素、获得前驱、获得后继、插入元素、删除元素、遍历总表 12 种操作。

1.1.2 构造具有菜单的功能演示系统

通过 switch 对输入的功能编号对线性表进行相应的操作或退出菜单,对非法输入予以提醒。

1.1.3 线性表的文件写入和读取

1. 输入文件名,将线性表写入相应的文件。
2. 输入文件名和线性表名 name,创建一个名为 name 的空线性表,将文件中的信息写入 name 中并输出。

1.1.4 多线性表管理

1. 线性表的新建,删除、根据输入的名字查找。
2. 能够对其中某个线性表进行 1.1.1 和 1.1.3 中共 14 种操作。

1.2 系统设计

1.2.1 线性表结构设计

LIST_SIZE=100

LISTINCREMENT=10

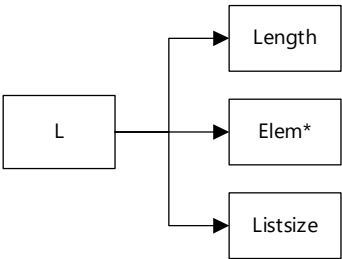


图 1-1 线性表 L 的数据结构设计

1.2.2 指针数组结构设计



图 1-2 指针数组 elem 的数据结构设计

1.2.3 多线性表管理结构设计

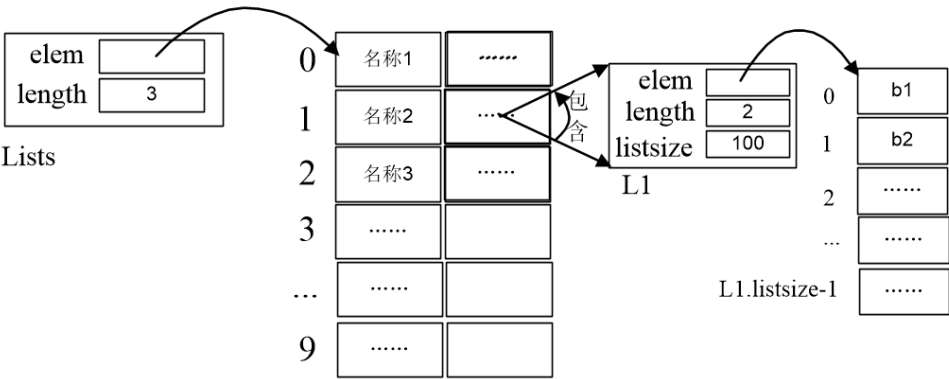


图 1-3 多线性表 LIST 的数据结构设计

1.2.4 菜单功能设计

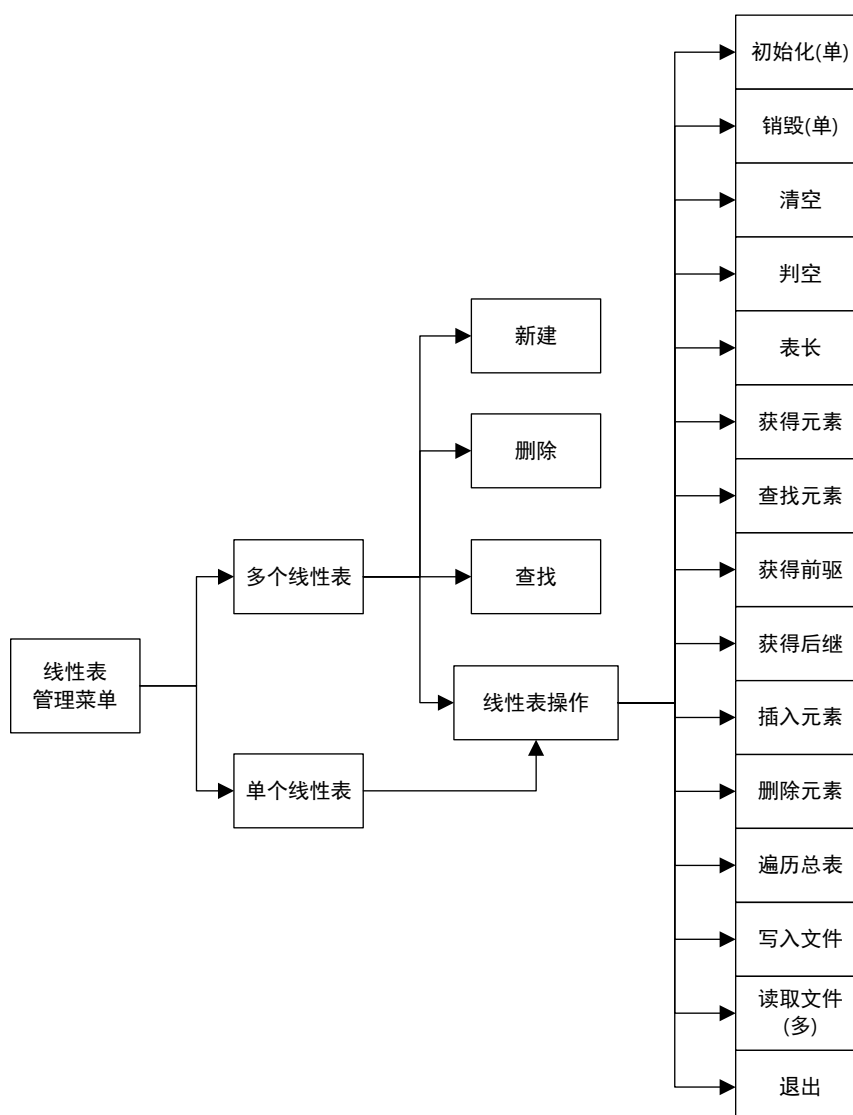


图 1-4 菜单的系统结构设计

1.2.5 文件格式设计

二进制格式(rb)写入，二进制格式(wb)写出。

1.2.6 函数设计

(1)初始化表：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：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：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: $\text{ListEmpty}(L)$, 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: $\text{ListLength}(L)$, 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: $\text{GetElem}(L, i, e)$, 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: $\text{LocateElem}(L, e)$, 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 相同的数据元素的位序, 若不存在, 则返回值为 0。

(8)获得前驱: $\text{PriorElem}(L, e, \text{pre})$, 初始条件是线性表 L 已存在; 操作结果是若 e 是 L 的数据元素, 且不是第一个, 则用 pre 返回它的前驱, 否则操作失败, pre 无定义。

(9)获得后继: $\text{NextElem}(L, e, \text{next})$, 初始条件是线性表 L 已存在; 操作结果是若 e 是 L 的数据元素, 且不是最后一个, 则用 next 返回它的后继, 否则操作失败, next 无定义。

(10)插入元素: $\text{ListInsert}(L, i, e)$, 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: $\text{ListDelete}(L, i, e)$, 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: $\text{ListTraverse}(L)$, 初始条件是线性表 L 已存在; 操作结果是依次输出 L 的每个数据元素。

(13)写入文件: $\text{SaveList}(L, \text{name})$, 初始条件是线性表 L 已存在; 操作结果是将数据元素写入文件。

(14)读取文件: $\text{LoadList}(L, \text{name})$, 初始条件是线性表 L 已存在; 操作结果是将数据元素写入文件。

(15)多线性表新建: $\text{AddList}(\text{Lists}, \text{name})$, 操作结果创建名为 name 的线性表。

(16)多线性表移除: $\text{RemoveList}(\text{Lists}, i)$, 操作结果是将第 $i+1$ 个线性表移除。

(17)多线性表查找: `LocateList(Lists, name)`, 操作结果是将查找名为 `name` 的线性表并返回逻辑序号。

(18)输入元素: `Input(L)`, 初始条件是线性表 `L` 已存在且为空; 操作结果是在空线性表 `L` 中输入元素。

1.3 系统实现

1.3.1 线性表的创建

1. 判断线性表是否存在, 若存在输出 `INFEASIBLE`, 结束功能
2. 否则 `malloc` 给 `elem` 分配 `LIST_INIT_SIZE` 大小的内存
3. `L.listsize=LIST_INIT_SIZE`
4. `L.length=0`

1.3.2 销毁线性表

1. 判断线性表是否存在, 不存在输出 `INFEASIBLE`, 结束功能
2. 否则释放 `elem` 的内存, 置为 `NULL`
3. `L.listsize=0, L.length=0`

1.3.3 清空线性表

1. 判断线性表是否存在, 不存在输出 `INFEASIBLE`, 结束功能
2. 否则释放 `elem` 的内存
3. `L.length=0`

1.3.4 线性表判空

1. 判断线性表是否存在, 不存在输出 `INFEASIBLE`, 结束功能
2. 否则判断 `L.length` 的值是否为 0, 输出 `FALSE` 或 `TRUE`

1.3.5 线性表长度

1. 判断线性表是否存在, 不存在输出 `INFEASIBLE`, 结束功能
2. 否则调用函数返回 `L.length` 的值

1.3.6 获取元素

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断 L.length 是否为 0，若为 0 输出空线性表
3. 否则若 $0 < i \leq L.length$ ，e 赋值为 elem[i-1]，否则输出 ERROR

1.3.7 查找元素

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断 L.length 是否为 0，若为 0 输出空线性表
3. 否则 for 循环遍历线性表，若 elem[i]=e，返回 i+1，否则输出 ERROR

i 用于 for 循环遍历数组

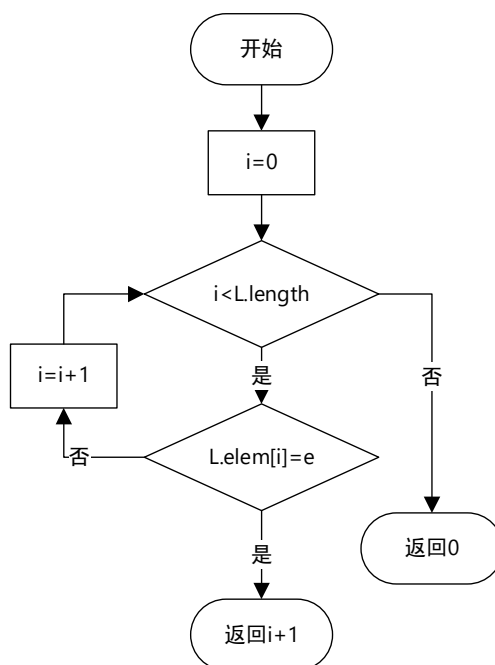


图 1-5 遍历查找流程图

1.3.8 获取前驱元素

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断 L.length 是否为 0，若为 0 输出空线性表
3. 否则调用 LocateElem(L, e)，若返回值为 i，若 i 大于 1，pre 赋值为 elem[i-2]，否则输出 ERROR

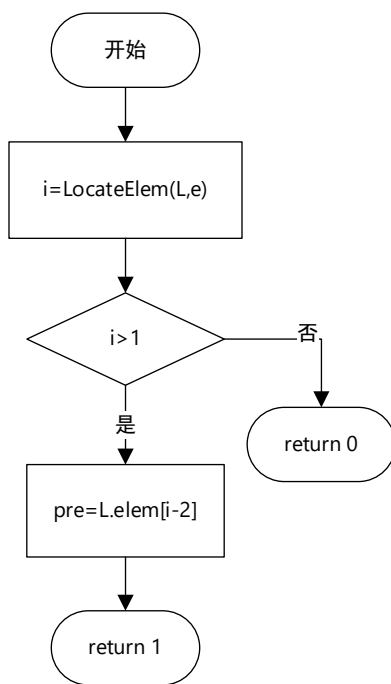


图 1-6 获取前驱流程图

1.3.9 获取后继元素

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断 L.length 是否为 0，若为 0 输出空线性表
3. 否则调用 LocateElem(L,e), 若返回值为 i，若 $0 < i < L.length$ ，next 赋值为 elem[i]，否则输出 ERROR

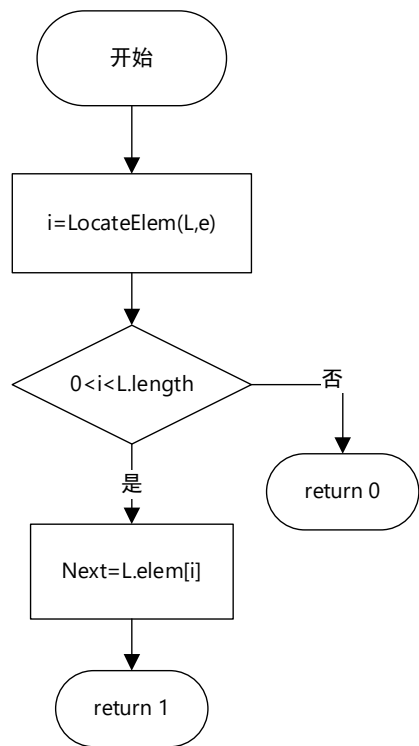


图 1-7 获取后继流程图

1.3.10 插入元素

1. 若 length+1 大于 LIST_INIT_SIZE，重新分配内存为 LIST_INIT_SIZE+LISTINCREMENT。

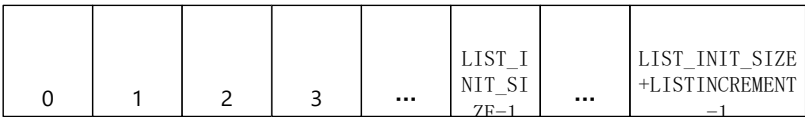


图 1-8 重新分配后线性表结构设计

- 2. 判断 i 是否合法，若不合法输出 ERROR，结束功能
- 3. 否则在第 i 个位置插入元素 e

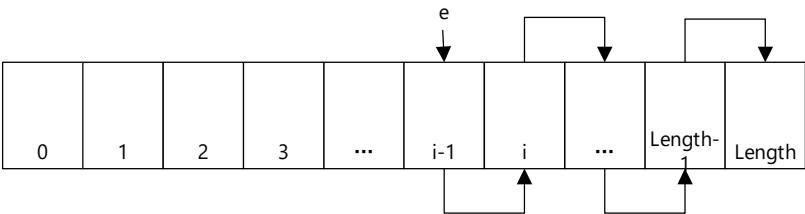


图 1-9 插入元素示意图

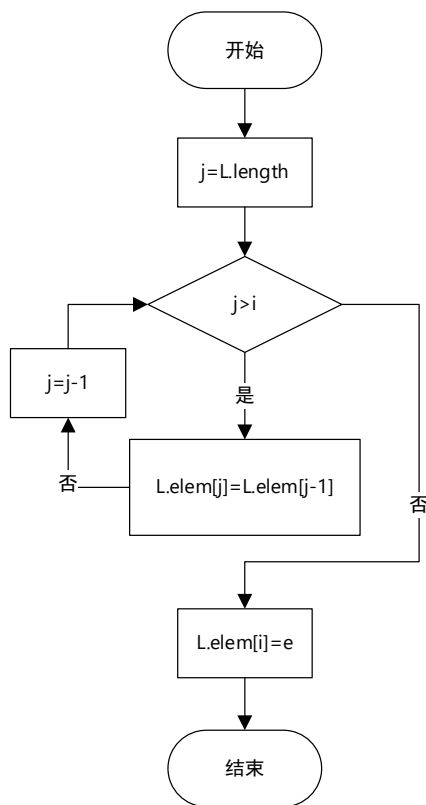


图 1-10 插入元素流程图

3. L.length++

1.3.11 删除元素

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断 L.length 是否为 0，若为 0 输出空线性表
3. 判断 i 是否合法，若不合法输出 ERROR，结束功能
4. 否则从第 i 个元素开始 for 循环删除元素（用后一个元素覆盖前一个）

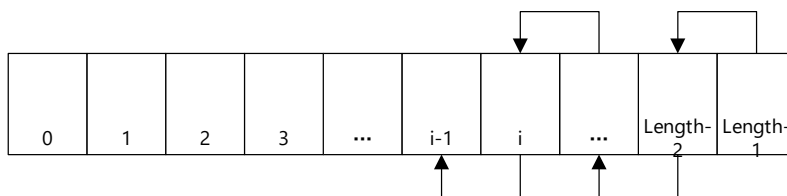


图 1-11 删除元素示意图

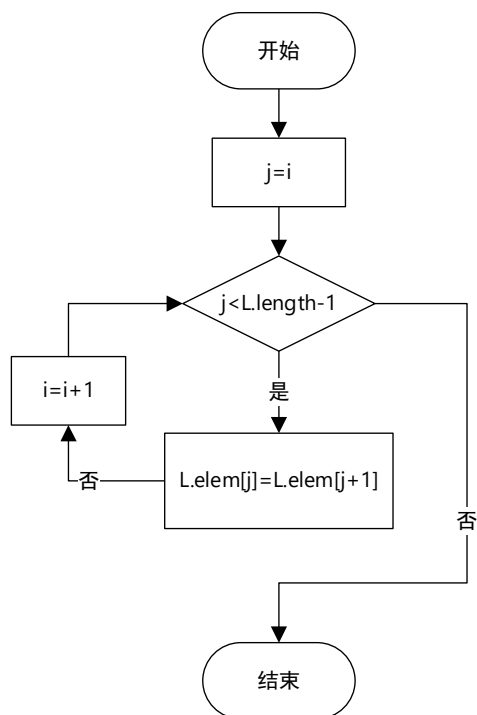


图 1-12 删除元素流程图

5. L.length—

1.3.12 遍历线性表

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断 L.length 是否为 0，若为 0 输出空线性表
3. 否则 for 循环遍历 elem 的元素，输出每个元素

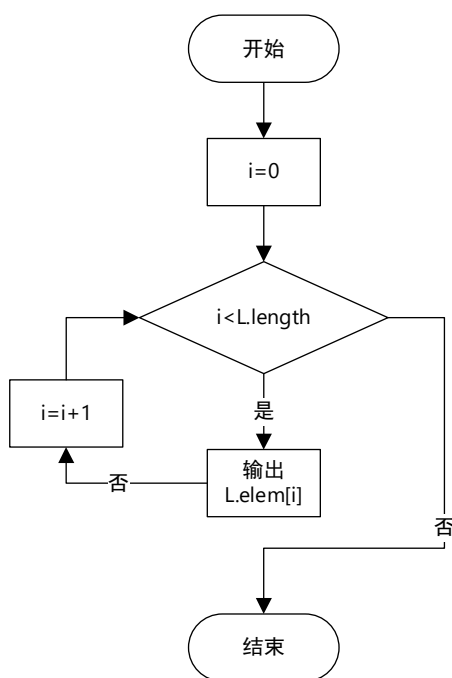


图 1-13 遍历线性表流程图

1.3.13 线性表写入文件

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断 L.length 是否为 0，若为 0 输出空线性表，结束功能
3. 否则定义 FILE 文件指针
2. 若文件名不合法，输出 ERROR，结束功能
3. 否则指针打开文件，fprintf 遍历 elem，将 elem 的数据元素写入 filename

1.3.14 线性表读取文件

1. 定义 FILE 文件指针
2. 若文件名不合法，输出打开文件失败，结束功能
3. 否则输入线性表名 name，若重名，重新输入
4. 创建名为 name 新线性表，加入多线性表管理中。指针打开文件，fscanf 遍历 filename，将 filename 中的元素写入新线性表 name 表中
5. 依次输出 name 中的数据元素

1.3.15 多线性表管理：增加新线性表

1. 若增加后线性表总数大于 10，输出 ERROR，结束功能
2. 若 name 与已存在的线性表重名，输出重名，重新输入
3. 否则 name 写入 Lists.elem[ListsLength-1].name
4. malloc 给 elem 分配 LIST_INIT_SIZE 大小的内存
5. 调用 Input(L)，输入线性表元素，创建线性表
6. Listname.length++

1.3.16 多线性表管理：移除线性表

1. 调用 LocateList，返回值为 i
2. 若 i!=0，删除该线性表（用后一个线性表覆盖前一个），否则输出 ERROR，结束功能

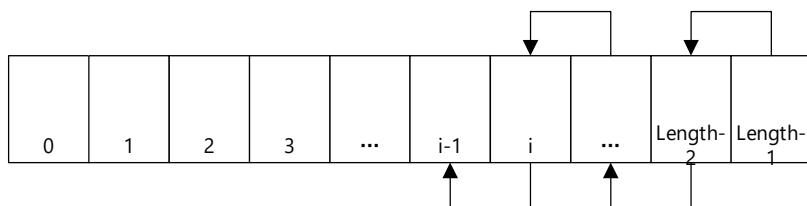


图 1-15 删除线性表示意图

3. length—

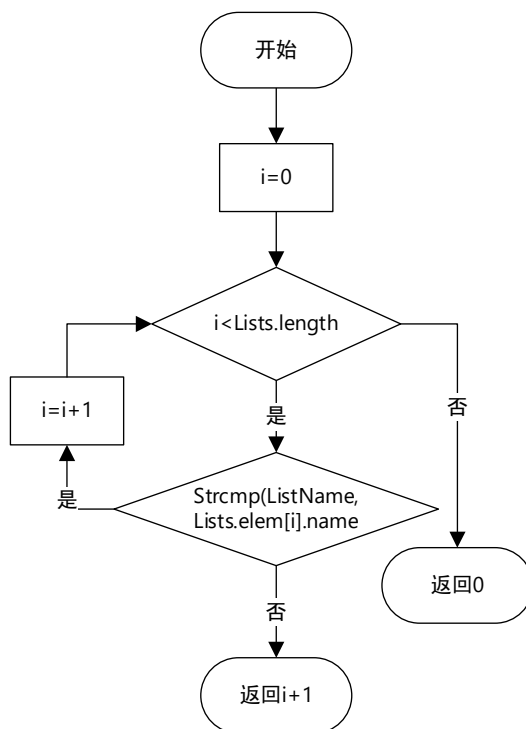


图 1-14 多线性表查找流程图

1.3.17 多线性表管理：查找线性表

1. 同上题遍历名字，查找对应线性表，返回 i
2. 若 $i \neq 0$ ，输出该线性表数据，否则输出 ERROR，结束功能

1.4 系统测试

1.4.1 菜单管理

功能 1、2 仅为单线性表操作，功能 14~17 仅为多线性表操作。若功能标号的选择与单或多选择不匹配，将不实现功能。

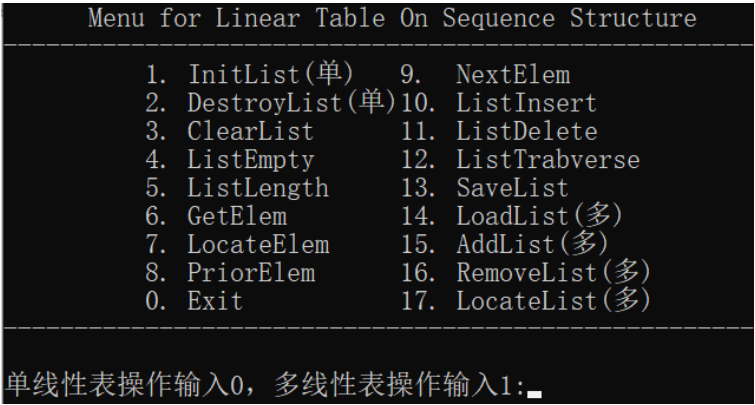


图 1-16 菜单

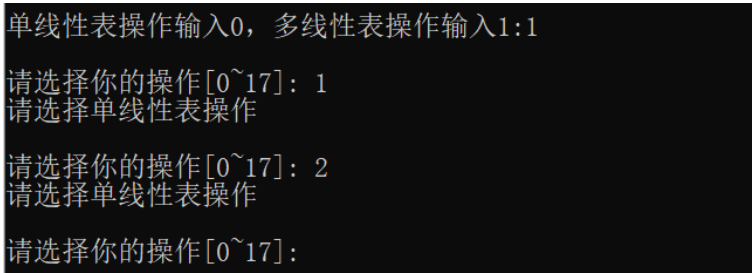


图 2-17 仅单线性表实现功能

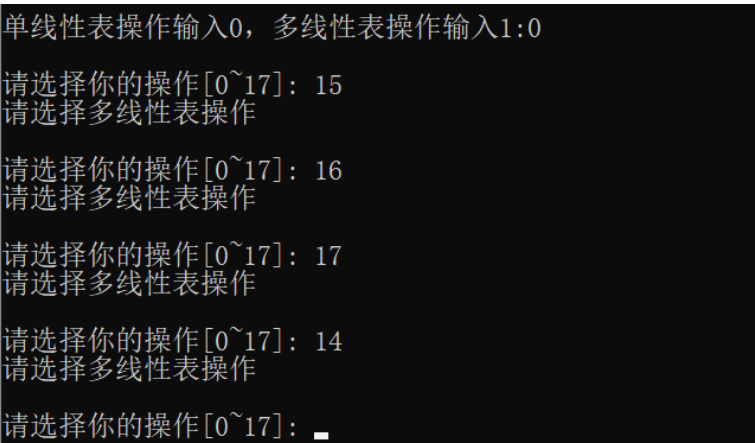


图 2-18 仅多线性表实现功能

1.4.2 输出结果及分析

系统测试输出结果及分析表

功能编号	输出结果	输出分析	功能编号	输出结果	输出分析
1	OK1	线性表创建成功	9	数字	后继
	ERROR	未正确分配内存		空线性表	空线性表
	INFEASIBLE	线性表已存在		ERROR	查找失败
				INFEASIBLE	线性表不存在

2	OK	线性表已销毁	10	数字串	线性表元素
	ERROR	未正确释放内存		ERROR	插入位置不合法
	INFEASIBLE	线性表不存在		INFEASIBLE	线性表不存在
3	OK	线性表已清空	11	数字串	线性表元素
	INFEASIBLE	线性表不存在		空线性表	空线性表
				ERROR	删除位置不合法
				INFEASIBLE	线性表不存在
4	TRUE	线性表为空	12	数字串	线性表元素
	FALSE	线性表不为空		空线性表	空线性表
	INFEASIBLE	线性表不存在		INFEASIBLE	线性表不存在
5	数字	线性表长度	13	OK	读入文件成功
	INFEASIBLE	线性表不存在		空线性表	空线性表
				ERROR	打开文件失败
				INFEASIBLE	线性表不存在
6	数字	数据元素的值	14	数字串	文件内容
	空线性表	空线性表		ERROR	打开文件失败
	ERROR	位置不合法		INFEASIBLE	线性表不存在
	INFEASIBLE	线性表不存在			
7	数字	元素的位置	15	OK	线性表创建成功
	空线性表	空线性表		ERROR	未正确分配内存
	ERROR	查找失败		重名	线性表重名
	INFEASIBLE	线性表不存在		OVERFLOW	溢出
8	数字	前驱	16	数字串	删除后的 线性表名+元素
	空线性表	空线性表			
	ERROR	查找失败		删除失败	线性表不存在
	INFEASIBLE	线性表不存在			
0	退出菜单		17	数字串	查找的 线性表名+元素
				查找失败	线性表不存在

表 1-1 系统测试输出说明

1.4.3 测试用例

1 2 3 4 5 0

11 12 13 14 15 0

1.4.4 测试截图

测试完用例后，依次实现 3 清空和 2 销毁功能，分别测试空线性表和线性表不存在的情况。

1 创建

```
请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
1 2 3 4 5 0

请选择你的操作[0~17]:1
INFEASIBLE
```

图 1-19 功能 1 的用例

2 销毁

```
请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
1 2 3 4 5 0

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:2
INFEASIBLE
```

图 1-20 功能 2 的用例

3 清空

```
请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
1 2 3 4 5 0

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:12
空线性表

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:2
INFEASIBLE
```

图 1-21 功能 3 的用例

4 判空

```
请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
1 2 3 4 5 0

请选择你的操作[0~17]:4
FALSE

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:4
TRUE

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:4
INFEASIBLE
```

图 1-22 功能 4 的用例

5 求表长

```

请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
1 2 3 4 5 0

请选择你的操作[0~17]:5
5

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:5
0

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:5
INFEASIBLE
    
```

图 1-23 功能 5 的用例

6 查找元素

```

请选择你的操作[0~17]: 1
输入元素, 结束输入0:
11 12 13 14 15 0
OK

请选择你的操作[0~17]: 6
输入要获取的元素的位置:3
13

请选择你的操作[0~17]: 6
输入要获取的元素的位置:6
ERROR

请选择你的操作[0~17]: 3
OK

请选择你的操作[0~17]: 6
空线性表

请选择你的操作[0~17]: 2
OK

请选择你的操作[0~17]: 6
INFEASIBLE
    
```

图 1-24 功能 6 的用例

7 获取位置

```

请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
11 12 13 14 15 0

请选择你的操作[0~17]:7
输入要查找位置的元素:12
2

请选择你的操作[0~17]:7
输入要查找位置的元素:18
ERROR

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:7
空线性表

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:7
INFEASIBLE
    
```

图 1-25 功能 7 的用例

8 获得前驱

```

请选择你的操作[0~17]:1
OK
输入元素, 结束输入0:
11 12 13 14 15 0

请选择你的操作[0~17]:8
输入要查找的元素: 13
12

请选择你的操作[0~17]:8
输入要查找的元素: 11
ERROR

请选择你的操作[0~17]:8
输入要查找的元素: 18
ERROR

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:8
空线性表

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:8
INFEASIBLE
    
```

图 1-26 功能 8 的用例

9 获得后继

```
请选择你的操作[0~17]:1
OK
输入元素, 结束输入0:
11 12 13 14 15 0

请选择你的操作[0~17]:9
输入要查找的元素: 13
14

请选择你的操作[0~17]:9
输入要查找的元素: 15
ERROR

请选择你的操作[0~17]:9
输入要查找的元素: 18
ERROR

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:9
空线性表

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:9
INFEASIBLE
```

图 1-27 功能 9 的用例

10 插入元素

```
请选择你的操作[0~17]: 1
输入元素, 结束输入0:
11 12 13 14 15 0
OK

请选择你的操作[0~17]: 10
输入要插入的位置和元素:1 10
10 11 12 13 14 15

请选择你的操作[0~17]: 10
输入要插入的位置和元素:7 16
10 11 12 13 14 15 16

请选择你的操作[0~17]: 10
输入要插入的位置和元素:10 19
ERROR

请选择你的操作[0~17]: 3
OK

请选择你的操作[0~17]: 10
输入要插入的位置和元素:1 10
10

请选择你的操作[0~17]: 2
OK

请选择你的操作[0~17]: 10
INFEASIBLE
```

图 1-28 功能 10 的用例

11 删除元素

```

请选择你的操作[0~17]: 1
输入元素, 结束输入0:
11 12 13 14 15 0
OK

请选择你的操作[0~17]: 11
输入要删除元素的位置:1
12 13 14 15

请选择你的操作[0~17]: 11
输入要删除元素的位置:4
12 13 14

请选择你的操作[0~17]: 11
输入要删除元素的位置:5
ERROR

请选择你的操作[0~17]: 3
OK

请选择你的操作[0~17]: 11
空线性表

请选择你的操作[0~17]: 2
OK

请选择你的操作[0~17]: 11
INFEASIBLE
    
```

图 1-29 功能 11 的用例

12 遍历表

```

请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
11 12 13 14 15 0

请选择你的操作[0~17]:12
11 12 13 14 15

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:12
空线性表

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:12
INFEASIBLE
    
```

图 1-30 功能 12 的用例

13 & 14 文件读取和写入

```

请选择你的操作[0~17]: 15
输入要增加的线性表的个数:1
输入线性表名字:1
输入元素, 结束输入0:
11 12 13 14 15 0
OK

请选择你的操作[0~17]: 13

输入操作的线性表的名字: 1
输入文件名:12345.txt
OK

请选择你的操作[0~17]: 14
输入文件名:12345.txt
输入线性表名:1
重名
输入线性表名:2
11 12 13 14 15

请选择你的操作[0~17]: 12

输入操作的线性表的名字: 2
11 12 13 14 15
    
```

图 1-31 功能 13、14 的用例

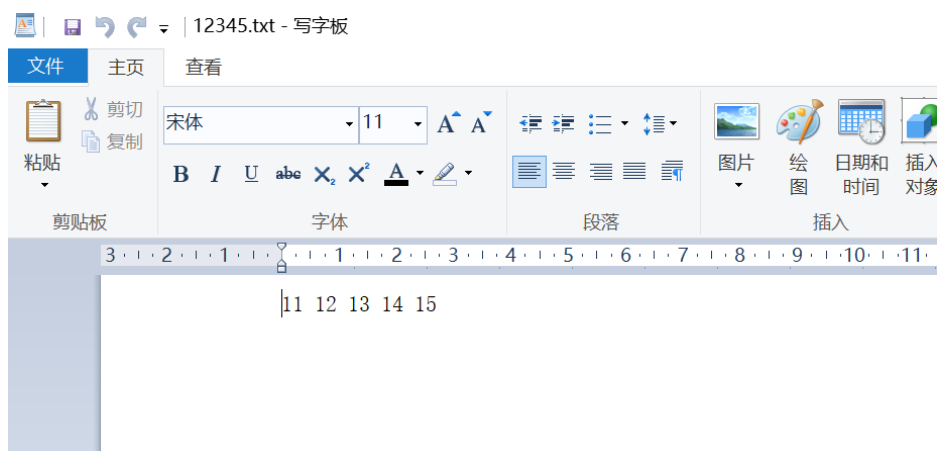


图 1-32 功能 13、14 的文件截图

15 新增线性表

```

请选择你的操作[0~17]: 15
输入要增加的线性表的个数:2
输入线性表名字:1
输入元素, 结束输入0:
11 12 13 14 15 0
OK
输入线性表名字:1
重名
输入线性表名字:2
输入元素, 结束输入0:
11 12 13 0
OK

请选择你的操作[0~17]: 15
输入要增加的线性表的个数:9
OVERFLOW
    
```

图 1-33 功能 15 的用例

16 移除线性表

```

请选择你的操作[0~17]:15
输入要增加的线性表的个数:3
输入线性表名字:1
OK
输入元素, 结束输入0:
11 12 13 14 15 0
输入线性表名字:2
OK
输入元素, 结束输入0:
11 12 13 0
输入线性表名字:3
OK
输入元素, 结束输入0:
11 0

请选择你的操作[0~17]:16
输入要删除的线性表的名字:2
1 11 12 13 14 15
3 11

请选择你的操作[0~17]:16
输入要删除的线性表的名字:4
删除失败
    
```

图 1-34 功能 16 的用例

17 查找线性表

```

请选择你的操作[0~17]:15
输入要增加的线性表的个数:3
输入线性表名字:1
OK
输入元素, 结束输入0:
11 12 13 14 15 0
输入线性表名字:2
OK
输入元素, 结束输入0:
11 12 13 0
输入线性表名字:3
OK
输入元素, 结束输入0:
11 0

请选择你的操作[0~17]:17
输入要查找的线性表的名字:2
2 11 12 13

请选择你的操作[0~17]:17
输入要查找的线性表的名字:4
查找失败
    
```

图 1-35 功能 17 的用例

0 退出

```

请选择你的操作[0~17]: 0

欢迎下次再使用本系统!

Process returned 0 (0x0)    execution time : 12.068 s
Press any key to continue.
    
```

图 1-36 功能 0

1.5 实验小结

1. 对线性表本身进行删或加操作后不能忘记改 L.length 的值。
2. 插入元素前先比较 length+1 和 listsize 的值，不够先重新分配内存。
3. 插入元素时要考虑插在最后一个元素后面的情况。
4. 销毁线性表时，释放 elem 空间后要置为 NULL，否则内存释放错误。
5. 多线性表管理时，对线性表进行删或加操作后不能忘记改 Lists.length 的值。
6. 写入文件时要注意空格怎么安排，读取文件时的空格和换行格式要和写入时一致。

7. 创建新线性表时要先遍历全表，检查是否有重名，否则查找和对线性表进行操作时会出错。

8. 要注意函数返回值是逻辑序号还是数组编号，逻辑序号=数组编号+1。

2 基于链式存储结构的线性表实现

2.1 问题描述

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)物理结构采用单链表，熟练掌握线性表的基本运算的实现。

2.1.1 线性表的顺序表实现

通过函数形式实现单链表的初始化、销毁、清空、判空、求表长、获得元素、获得前驱、获得后继、插入元素、删除元素、遍历总表 12 种操作。

2.1.2 构造具有菜单的功能演示系统

通过 switch 对输入的功能编号对单链表进行相应的操作或退出菜单，对非法输入予以提醒。

2.1.3 单链表的文件写入和读取

1. 输入文件名，将单链表写入相应的文件。
2. 输入文件名，将文件中的信息写入新建链表中并输出。

2.1.4 多单链表管理

1. 单链表的新建，删除、根据输入的名字查找。
2. 能够对其中某个单链表进行 2.1.1 和 2.1.3 中共 14 种操作。

2.2 系统设计

2.2.1 单链表结点结构设计

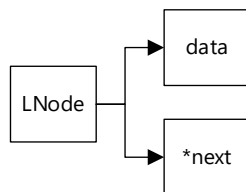


图 2-1 单链表结点 LNode 的数据结构设计

2.2.2 单链表结构设计

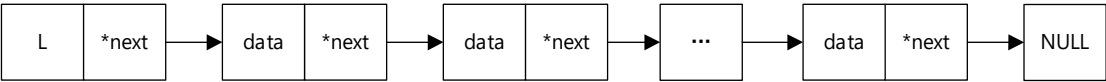


图 2-2 单链表结点 L 的数据结构设计

2.2.3 多线性表管理结构设计

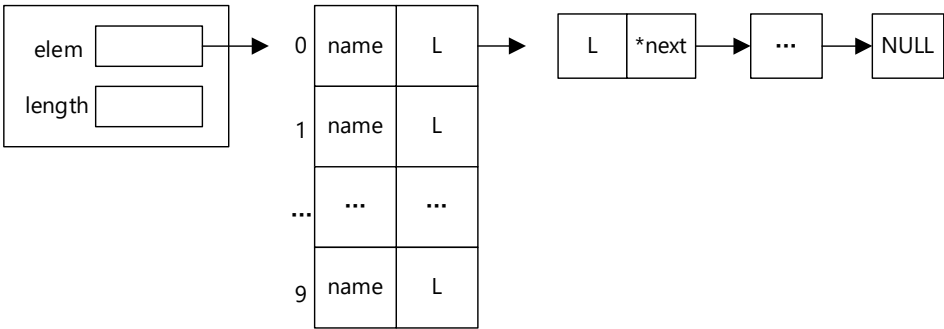


图 2-3 多线性表管理 LIST 的数据结构设计

2.2.4 菜单功能设计

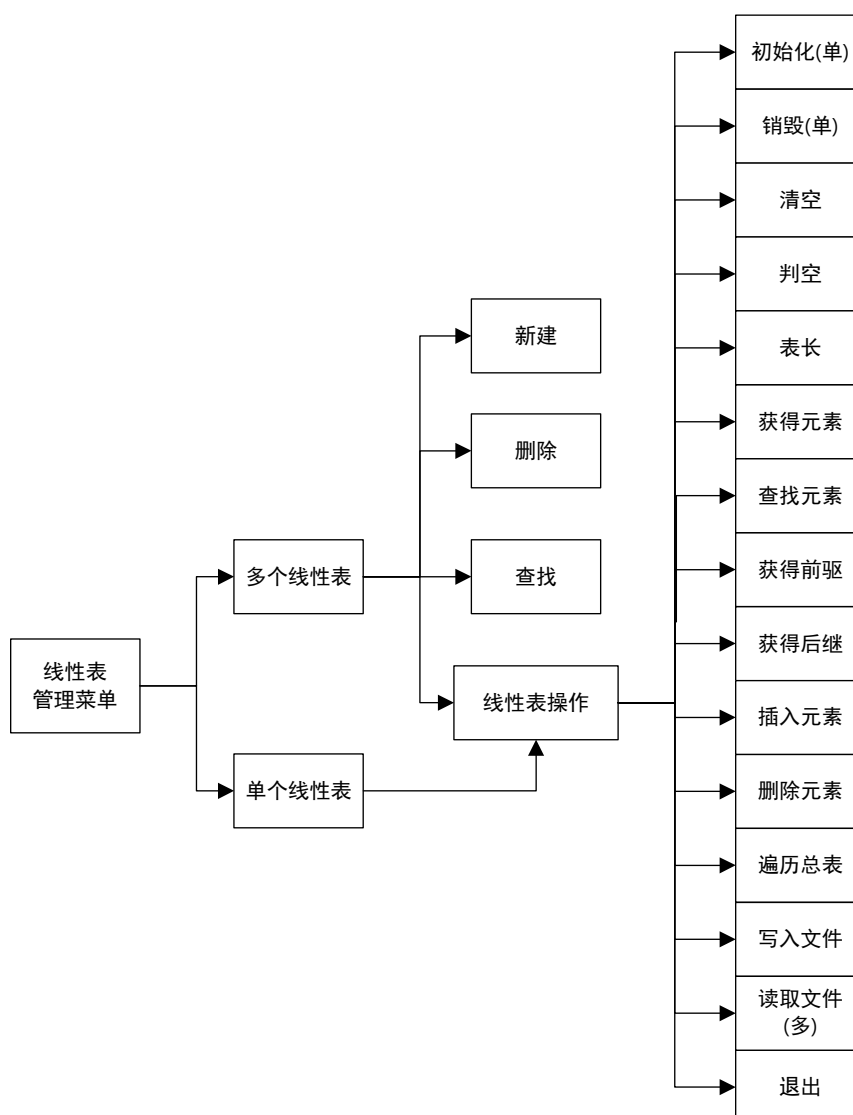


图 2-4 菜单的系统结构设计

2.2.5 文件格式设计

二进制格式(rb)写入，二进制格式(wb)写出。

2.2.6 函数设计

(1)初始化表：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：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：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: $\text{ListEmpty}(L)$; 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: $\text{ListLength}(L)$; 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: $\text{GetElem}(L, i, e)$; 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: $\text{LocateElem}(L, e)$; 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 相同的数据元素的位序, 若不存在, 则返回值为 0。

(8) 获得前驱: $\text{PriorElem}(L, e, \text{pre})$; 初始条件是线性表 L 已存在; 操作结果是若 e 是 L 的数据元素, 且不是第一个, 则用 pr 返回它的前驱, 否则操作失败, pre 无定义。

(9) 获得后继: $\text{NextElem}(L, e, \text{next})$; 初始条件是线性表 L 已存在; 操作结果是若 e 是 L 的数据元素, 且不是最后一个, 则用 next 返回它的后继, 否则操作失败, next 无定义。

(10) 插入元素: $\text{ListInsert}(L, i, e)$, 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: $\text{ListDelete}(L, i, e)$, 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: $\text{ListTraverse}(L)$, 初始条件是线性表 L 已存在; 操作结果是依次输出 L 的每个数据元素。

(13) 写入文件: $\text{SaveList}(L, \text{name})$, 初始条件是线性表 L 已存在; 操作结果是将数据元素写入文件。

(14) 读取文件: $\text{LoadList}(L, \text{name})$, 初始条件是线性表 L 已存在; 操作结果是将数据元素写入文件。

(15) 多线性表新建: $\text{AddList}(\text{Lists}, \text{name})$, 操作结果创建名为 name 的线性表。

(16) 多线性表移除: $\text{RemoveList}(\text{Lists}, i)$, 操作结果是将第 $i+1$ 个线性表移除。

(17) 多线性表查找: $\text{LocateList}(\text{Lists}, \text{name})$, 操作结果是将查找名为 name 的线性表并返回逻辑序号。

(18) 输入元素: $\text{Input}(L)$; 初始条件是线性表 L 已存在且为空; 操作结果是在空线性表 L 中输入元素。

2.3 系统实现

2.3.1 线性表的创建

1. 判断线性表是否存在, 若存在输出 INFEASIBLE, 结束功能
2. 否则 malloc 给头结点分配内存
2. 指针域为 NULL

2.3.2 销毁线性表

1. 判断线性表是否存在, 不存在输出 INFEASIBLE, 结束功能
2. 否则依次释放每个结点的内存, 置为 NULL
3. 释放表头内存, 置为 NULL

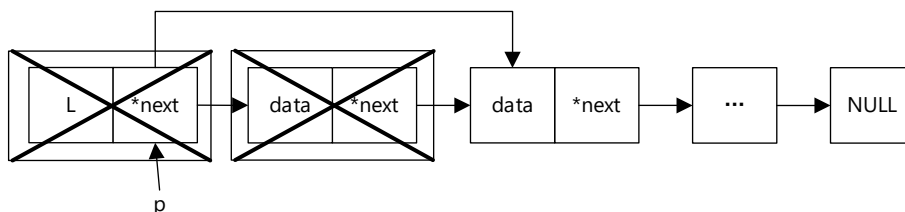


图 2-5 销毁线性表示意图

2.3.3 清空线性表

1. 判断线性表是否存在, 不存在输出 INFEASIBLE, 结束功能
2. 否则依次释放每个结点的内存, 置为 NULL

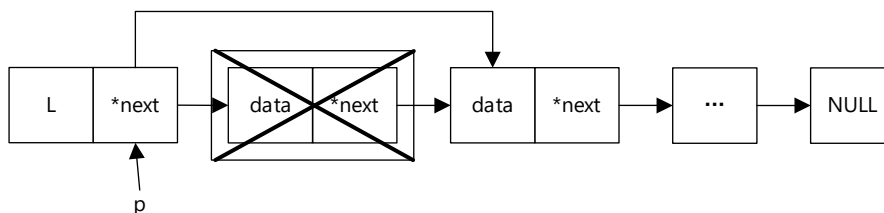


图 2-6 清空线性表示意图

2.3.4 线性表判空

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断头结点的指针域是否为空，若为空输出 TRUE，否则输出 FALSE

2.3.5 线性表长度

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则用指针 p 遍历链表，用 cnt 记录表长，直到最后一个结点的指针域为空

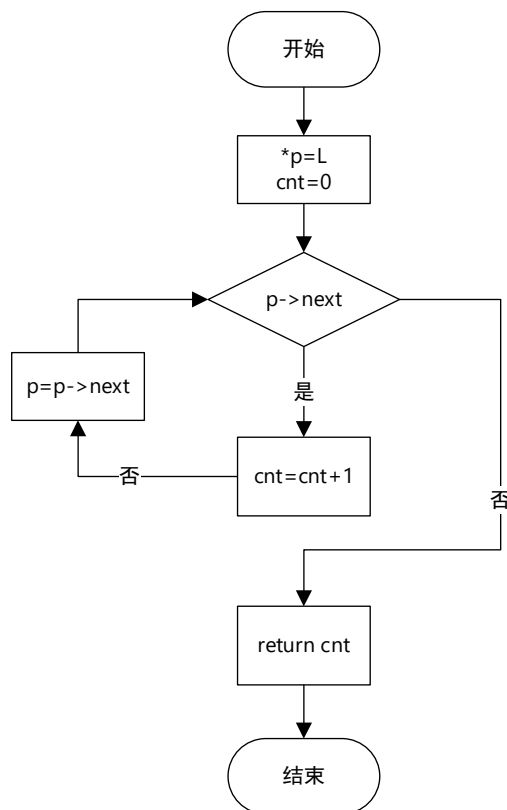


图 2-7 求表长流程图

2.3.6 获取元素

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断 i 是否合法，若不合法输出 ERROR，结束功能
3. 否则用指针 p 遍历线性表，e 赋值为第 i 个结点的数据域

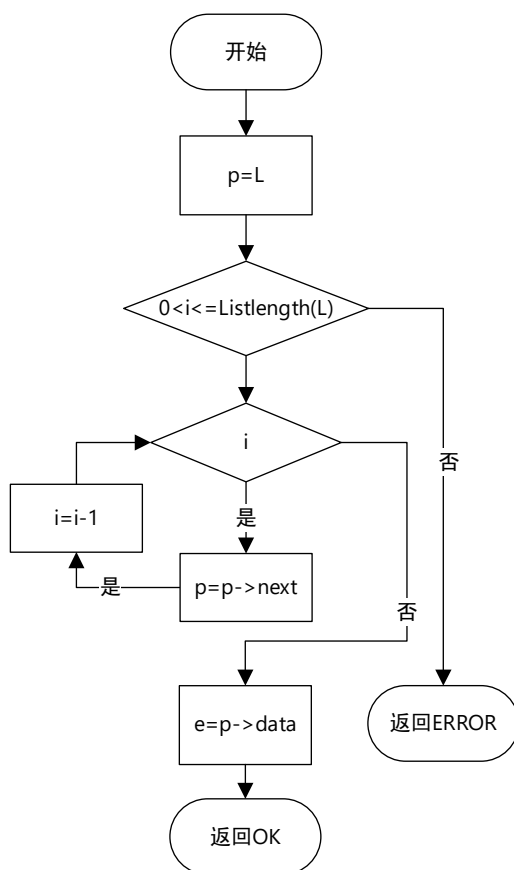


图 2-8 获取元素流程图

2.3.7 查找元素

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断 L.length 是否为 0，若为 0 输出空线性表
3. 否则用指针 p 循环遍历线性表

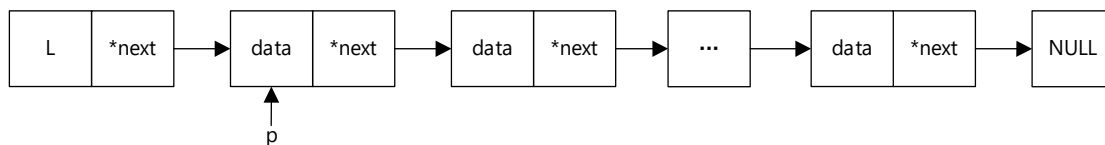


图 2-9 查找元素遍历示意图

4. 若 $\text{elem}[i]=e$, 返回 $i+1$, 否则返回 ERROR

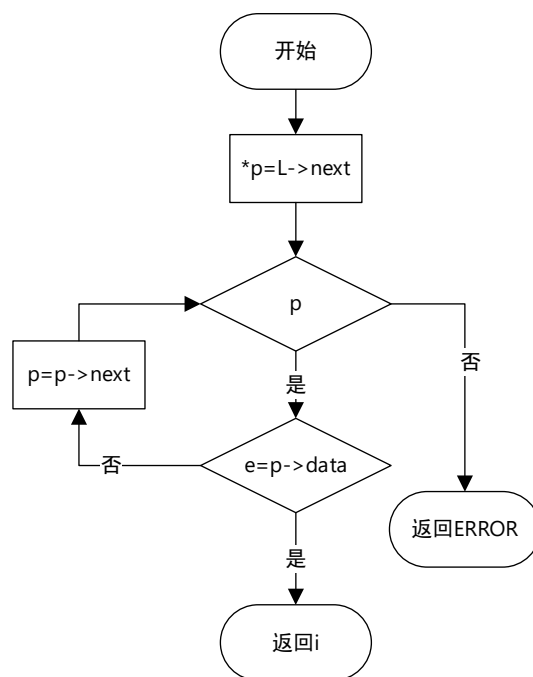


图 2-10 查找元素流程图

2.3.8 获取前驱元素

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断 L.length 是否为 0，若为 0 输出空线性表
3. 否则调用 LocateElem(L, e), 返回值为 i, i--

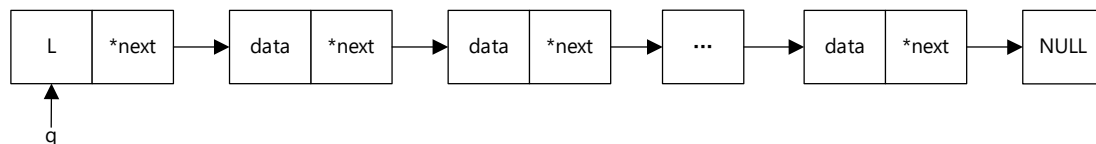


图 2-11 获取前驱元素示意图

4. 若 $i > 1$ ，指针 p 遍历至第 i 个结点，否则输出 ERROR，结束功能
5. pre 赋值为该结点的数据域

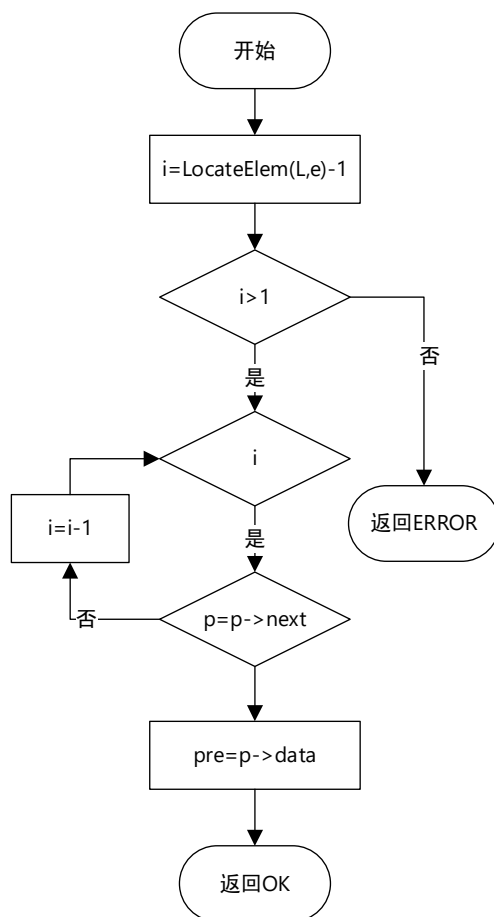


图 2-12 获取前驱流程图

2.3.9 获取后继元素

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断 L.length 是否为 0，若为 0 输出空线性表
3. 否则调用 LocateElem(L, e), 返回值为 i
4. 若 $1 < i < \text{ListLength}(L)$, 指针 p 遍历至第 i 个结点，否则输出 ERROR，结束功能
5. next 赋值为下一结点的数据域

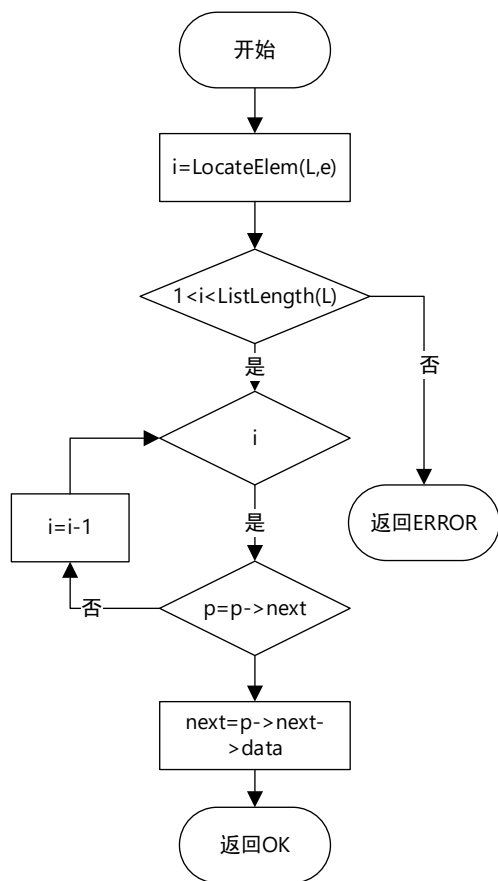


图 2-13 获取后继流程图

2.3.10 插入元素

- 1. 调用 LocateElem(L, e), 返回值为 i, 结束功能
- 2. 若 $0 < i \leq \text{ListLength}(L) + 1$, 指针 p 遍历至第 i 个结点, 指针 q 遍历至第 i-1 个结点。否则输出 ERROR, 结束功能
- 3. 给新结点 n 分配内存, $n \rightarrow \text{data} = e$
- 4. n 插入到 q 和 p 之间

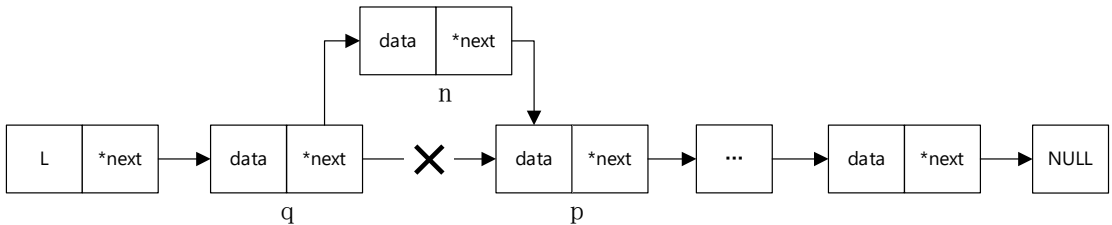


图 2-14 插入元素示意图

5. L.length++

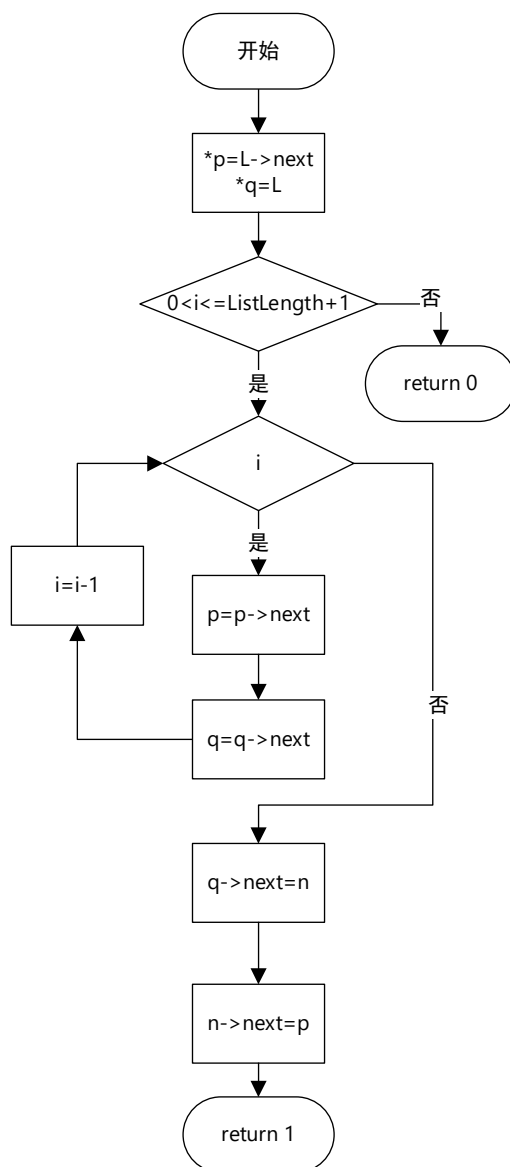


图 2-15 插入元素示意图

2.3.11 删除元素

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断 L.length 是否为 0，若为 0 输出空线性表，结束功能
3. 否则若 $0 < i \leq \text{ListLength}(L)$ ，指针 p 遍历至第 i 个结点，指针 q 遍历至第 i-1 个结点。否则输出 ERROR，结束功能
4. q 的指针域赋值为 p 的指针域，释放 p 的内存
5. L.length—

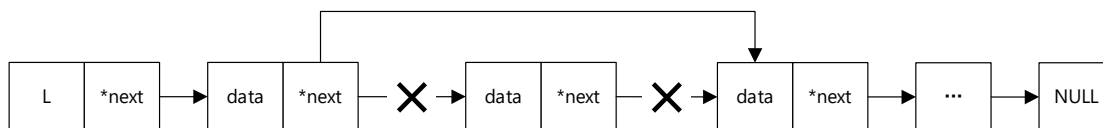


图 2-16 删除元素示意图

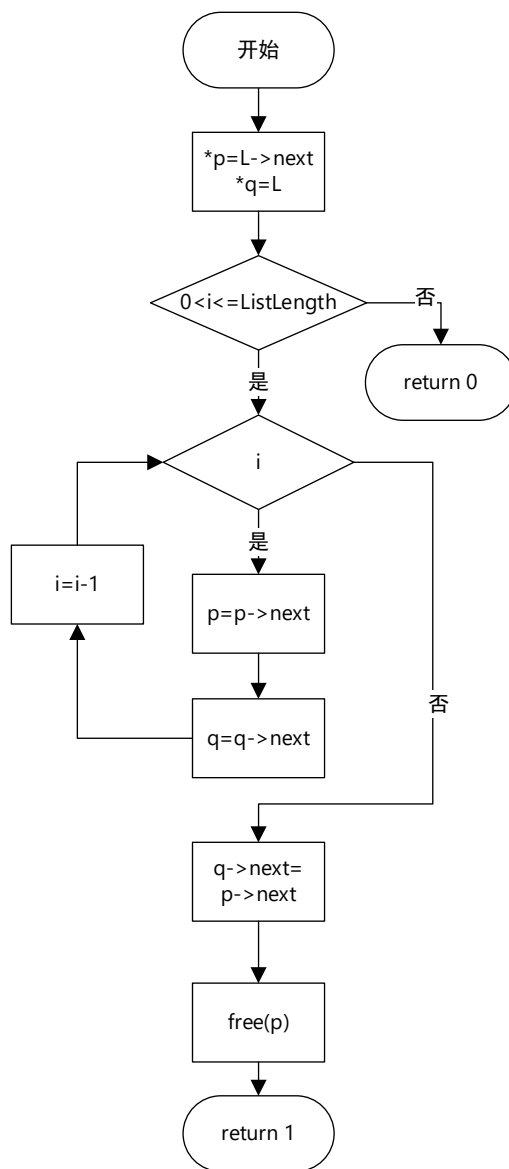


图 2-17 删除元素流程图

2.3.12 遍历线性表

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断 $L.length$ 是否为 0，若为 0 输出空线性表
3. 否则指针 p 遍历线性表，依次输出数据域的元素

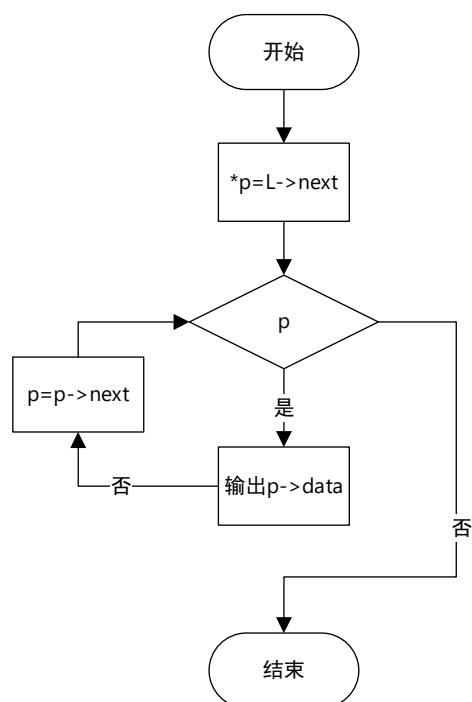


图 2-18 遍历表流程图

2.3.13 线性表写入文件

1. 判断线性表是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断 L.length 是否为 0，若为 0 输出空线性表
3. 否则定义 FILE 文件指针
4. 若文件名不合法，输出打开文件失败
5. 否则指针打开文件，fprintf 遍历链表 L，将 L 的数据元素写入 filename

2.3.14 线性表读取文件

1. 定义 FILE 文件指针
2. 若文件名不合法，输出打开文件失败，结束功能
3. 否则输入线性表名 name，若重名，重新输入
4. 否则创建名为 name 新线性表，加入多线性表管理中。指针打开文件，fscanf 遍历 filename，将 filename 中的元素写入新线性表 name 表中
5. 依次输出 name 中的数据元素

2.3.15 多线性表管理：增加新线性表

1. 若增加后线性表总数大于 10，输出 ERROR，结束功能

2. 若 name 与已存在的线性表重名，输出重名，重新输入
3. 否则将 name 写入 Lists.elem[ListsLength-1].name
4. malloc 给头结点分配内存
5. 调用 Input(L)，输入线性表元素，创建线性表
6. Listname.length++

2.3.16 多线性表管理：移除线性表

1. 调用 LocateList, 返回值为 i
2. 若 $i \neq 0$ ，删除该线性表（用后一个线性表覆盖前一个），否则输出 ERROR，结束功能

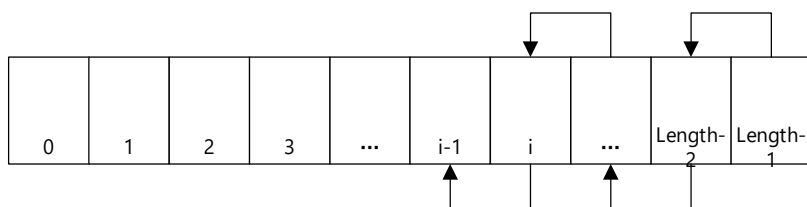


图 2-20 删除线性表示意图

3. length—

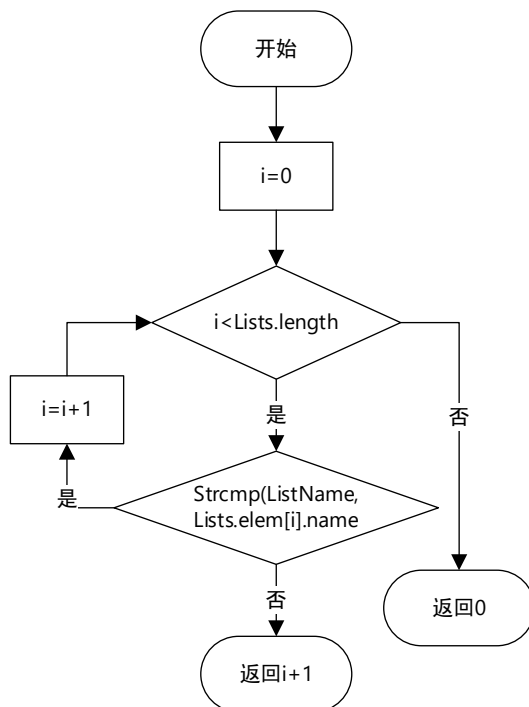


图 2-19 多线性表查找流程图

2.3.17 多线性表管理：查找线性表

1. 同上题遍历名字，查找对应线性表，返回 i
2. 若 $i \neq 0$ ，输出该线性表数据，否则输出 ERROR，结束功能

2.4 系统测试

2.4.1 菜单管理

功能 1、2 仅为单线性表操作，功能 14~17 仅为多线性表操作。若功能标号的选择与单或多选择不匹配，将不实现功能。

Menu for Linear Table On Sequence Structure	
1. InitList(单)	9. NextElem
2. DestroyList(单)	10. ListInsert
3. ClearList	11. ListDelete
4. ListEmpty	12. ListTraverse
5. ListLength	13. SaveList
6. GetElem	14. LoadList(多)
7. LocateElem	15. AddList(多)
8. PriorElem	16. RemoveList(多)
0. Exit	17. LocateList(多)

单线性表操作输入0，多线性表操作输入1: _

图 2-21 菜单

```
单线性表操作输入0，多线性表操作输入1:1
请选择你的操作[0~17]: 1
请选择单线性表操作

请选择你的操作[0~17]: 2
请选择单线性表操作

请选择你的操作[0~17]:
```

图 2-22 仅单线性表实现功能

```
单线性表操作输入0，多线性表操作输入1:0
请选择你的操作[0~17]: 15
请选择多线性表操作

请选择你的操作[0~17]: 16
请选择多线性表操作

请选择你的操作[0~17]: 17
请选择多线性表操作

请选择你的操作[0~17]: 14
请选择多线性表操作

请选择你的操作[0~17]: _
```

图 2-23 仅多线性表实现功能

2.4.2 输出结果及分析

系统测试输出结果及分析表

功能编号	输出结果	输出分析	功能编号	输出结果	输出分析
1	OK	线性表创建成功	9	数字	后继
	ERROR	未正确分配内存		空线性表	空线性表
	INFEASIBLE	线性表已存在		ERROR	查找失败
				INFEASIBLE	线性表不存在
2	OK	线性表已销毁	10	数字串	线性表元素
	ERROR	未正确释放内存		ERROR	插入位置不合法
	INFEASIBLE	线性表不存在		INFEASIBLE	线性表不存在
3	OK	线性表已清空	11	数字串	线性表元素
	INFEASIBLE	线性表不存在		空线性表	空线性表
				ERROR	删除位置不合法
				INFEASIBLE	线性表不存在
4	TRUE	线性表为空	12	数字串	线性表元素
	FALSE	线性表不为空		空线性表	空线性表
	INFEASIBLE	线性表不存在		INFEASIBLE	线性表不存在
5	数字	线性表长度	13	OK	读入文件成功
	INFEASIBLE	线性表不存在		空线性表	空线性表
				ERROR	打开文件失败
				INFEASIBLE	线性表不存在
6	数字	数据元素的值	14	数字串	文件内容
	空线性表	空线性表		ERROR	打开文件失败
	ERROR	位置不合法		INFEASIBLE	线性表不存在
	INFEASIBLE	线性表不存在			
7	数字	元素的位置	15	OK	线性表创建成功
	空线性表	空线性表		ERROR	未正确分配内存

	ERROR	查找失败		重名	线性表重名
	INFEASIBLE	线性表不存在		OVERFLOW	溢出
8	数字	前驱	16	数字串	删除后的
	空线性表	空线性表			线性表名+元素
	ERROR	查找失败		删除失败	线性表不存在
	INFEASIBLE	线性表不存在			
0	退出菜单		17	数字串	查找的
				查找失败	线性表不存在

表 2-1 系统测试输出说明

2.4.3 测试用例

1 2 3 4 5 0

11 12 13 14 15 0

2.4.4 测试截图

测试完用例后，依次实现 3 清空和 2 销毁功能，分别测试空线性表和线性表不存在的情况。

1 创建

```

请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
1 2 3 4 5 0

请选择你的操作[0~17]:1
INFEASIBLE
    
```

图 2-24 功能 1 的用例

2 销毁

```
请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
1 2 3 4 5 0

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:2
INFEASIBLE
```

图 2-25 功能 2 的用例

3 清空

```
请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
1 2 3 4 5 0

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:12
空线性表

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:2
INFEASIBLE
```

图 2-26 功能 3 的用例

4 判空

```
请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
1 2 3 4 5 0

请选择你的操作[0~17]:4
FALSE

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:4
TRUE

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:4
INFEASIBLE
```

图 2-27 功能 4 的用例

5 求表长

```
请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
1 2 3 4 5 0

请选择你的操作[0~17]:5
5

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:5
0

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:5
INFEASIBLE
```

图 2-28 功能 5 的用例

6 查找元素

```
请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
1 2 3 4 5 0

请选择你的操作[0~17]:6
输入要获取的元素的位置:3
OK
3

请选择你的操作[0~17]:6
输入要获取的元素的位置:7
ERROR

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:6
空线性表

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:6
INFEASIBLE
```

图 2-29 功能 6 的用例

7 获取位置

```

请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
11 12 13 14 15 0

请选择你的操作[0~17]:7
输入要查找位置的元素:12
2

请选择你的操作[0~17]:7
输入要查找位置的元素:18
ERROR

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:7
空线性表

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:7
INFEASIBLE
    
```

图 2-30 功能 7 的用例

8 获得前驱

```

请选择你的操作[0~17]:1
OK
输入元素, 结束输入0:
11 12 13 14 15 0

请选择你的操作[0~17]:8
输入要查找的元素: 13
12

请选择你的操作[0~17]:8
输入要查找的元素: 11
ERROR

请选择你的操作[0~17]:8
输入要查找的元素: 18
ERROR

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:8
空线性表

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:8
INFEASIBLE
    
```

图 2-31 功能 8 的用例

9 获得后继

```
请选择你的操作[0~17]:1
OK
输入元素, 结束输入0:
11 12 13 14 15 0

请选择你的操作[0~17]:9
输入要查找的元素: 13
14

请选择你的操作[0~17]:9
输入要查找的元素: 15
ERROR

请选择你的操作[0~17]:9
输入要查找的元素: 18
ERROR

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:9
空线性表

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:9
INFEASIBLE
```

图 2-32 功能 9 的用例

10 插入元素

```

请选择你的操作[0~17]: 1
输入元素, 结束输入0:
11 12 13 14 15 0
OK

请选择你的操作[0~17]: 10
输入要插入的位置和元素:1 10
10 11 12 13 14 15

请选择你的操作[0~17]: 10
输入要插入的位置和元素:7 16
10 11 12 13 14 15 16

请选择你的操作[0~17]: 10
输入要插入的位置和元素:10 19
ERROR

请选择你的操作[0~17]: 3
OK

请选择你的操作[0~17]: 10
输入要插入的位置和元素:1 10
10

请选择你的操作[0~17]: 2
OK

请选择你的操作[0~17]: 10
INFEASIBLE
    
```

图 2-33 功能 10 的用例

11 删除元素

```

请选择你的操作[0~17]: 1
输入元素, 结束输入0:
11 12 13 14 15 0
OK

请选择你的操作[0~17]: 11
输入要删除元素的位置:1
12 13 14 15

请选择你的操作[0~17]: 11
输入要删除元素的位置:4
12 13 14

请选择你的操作[0~17]: 11
输入要删除元素的位置:5
ERROR

请选择你的操作[0~17]: 3
OK

请选择你的操作[0~17]: 11
空线性表

请选择你的操作[0~17]: 2
OK

请选择你的操作[0~17]: 11
INFEASIBLE
    
```

图 2-34 功能 11 的用例

12 遍历

```

请选择你的操作[0~17]:1
OK

输入元素, 结束输入0:
11 12 13 14 15 0

请选择你的操作[0~17]:12
11 12 13 14 15

请选择你的操作[0~17]:3
OK

请选择你的操作[0~17]:12
空线性表

请选择你的操作[0~17]:2
OK

请选择你的操作[0~17]:12
INFEASIBLE
    
```

图 2-35 功能的用例

13 & 14 文件读取和写入

```

请选择你的操作[0~17]: 15
输入要增加的线性表的个数:1
输入线性表名字:1
输入元素, 结束输入0:
11 12 13 14 15 0
OK

请选择你的操作[0~17]: 13

输入操作的线性表的名字: 1
输入文件名:12345.txt
OK

请选择你的操作[0~17]: 14
输入文件名:12345.txt
输入线性表名:1
重名
输入线性表名:2
11 12 13 14 15

请选择你的操作[0~17]: 12

输入操作的线性表的名字: 2
11 12 13 14 15
    
```

图 2-36 功能 13、14 的用例

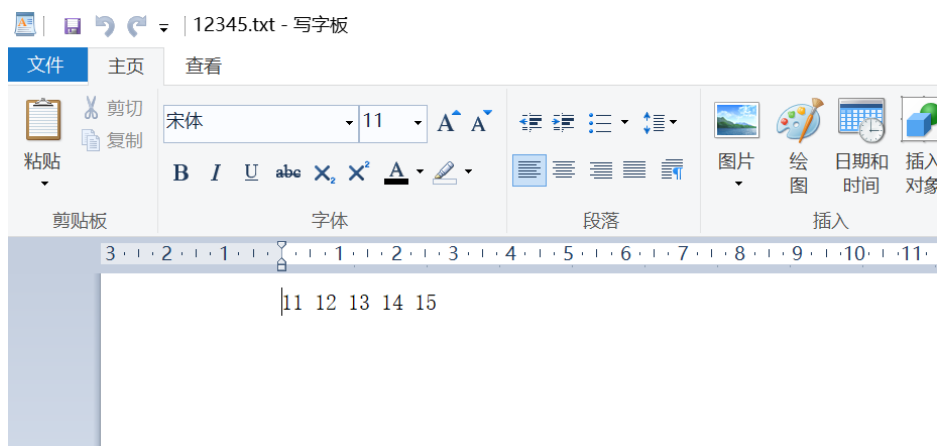


图 2-37 功能 13、14 的文件截图

15 新增线性表

```

请选择你的操作[0~17]: 15
输入要增加的线性表的个数:2
输入线性表名字:1
输入元素, 结束输入0:
11 12 13 14 15 0
OK
输入线性表名字:1
重名
输入线性表名字:2
输入元素, 结束输入0:
11 12 13 0
OK

请选择你的操作[0~17]: 15
输入要增加的线性表的个数:9
OVERFLOW
    
```

图 2-38 功能 15 的用例

16 移除线性表

```

请选择你的操作[0~17]:15
输入要增加的线性表的个数:3
输入线性表名字:1
OK
输入元素, 结束输入0:
11 12 13 14 15 0
输入线性表名字:2
OK
输入元素, 结束输入0:
11 12 13 0
输入线性表名字:3
OK
输入元素, 结束输入0:
11 0

请选择你的操作[0~17]:16
输入要删除的线性表的名字:2
1 11 12 13 14 15
3 11

请选择你的操作[0~17]:16
输入要删除的线性表的名字:4
删除失败
    
```

图 2-39 功能 16 的用例

17 查找线性表

```

请选择你的操作[0~17]:15
输入要增加的线性表的个数:3
输入线性表名字:1
OK
输入元素, 结束输入0:
11 12 13 14 15 0
输入线性表名字:2
OK
输入元素, 结束输入0:
11 12 13 0
输入线性表名字:3
OK
输入元素, 结束输入0:
11 0

请选择你的操作[0~17]:17
输入要查找的线性表的名字:2
2 11 12 13

请选择你的操作[0~17]:17
输入要查找的线性表的名字:4
查找失败
    
```

图 2-40 功能 17 的用例

0 退出

```

请选择你的操作[0~17]: 0

欢迎下次再使用本系统!

Process returned 0 (0x0)   execution time : 12.068 s
Press any key to continue.
    
```

图 2-41 功能 0

2.5 实验小结

1. 插入元素时要考虑插在最后一个元素后面的情况。
2. 销毁线性表时，释放空间后要将表头置为 NULL，否则内存释放错误。
3. 清空线性表时要保留头结点。
4. 多线性表管理时，对线性表进行删或加操作后不能忘记改 Lists.length 的值。
5. 删除元素后不能忘记释放该结点的内存。
6. 写入文件时要注意空格怎么安排，读取文件时的空格和换行格式要和写入时一致。

7. 创建新线性表时要先遍历全表，检查是否有重名，否则查找和对线性表进行操作时会出错。

8. 要注意函数返回值是逻辑序号还是数组编号，逻辑序号=数组编号+1。

3 基于二叉链表的二叉树实现

3.1 问题描述

通过实验达到(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。

3.1.1 二叉树的实现

通过函数形式实现二叉树的创建、销毁、清空、判空、求深度、查找结点、结点赋值、获得兄弟结点、插入结点、删除结点、前序遍历、中序遍历、后序遍历、按层遍历 14 种操作。

3.1.2 构造具有菜单的功能演示系统

通过 switch 对输入的功能编号对二叉树进行相应的操作或退出菜单，对非法输入予以提醒。

3.1.3 二叉树的文件写入和读取

1. 输入文件名，将二叉树写入相应的文件
2. 输入文件名，创建二叉树并将文件中的信息写入二叉树中并输出

3.1.4 多二叉树管理

1. 多二叉树的新建，删除、根据输入的名字查找
2. 能够对其中某个二叉树进行 3.1.2 和 3.1.3 中共 16 种操作

3.2 系统设计

3.2.1 二叉树结点结构设计

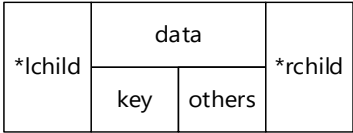


图 3-1 二叉树结点 BiTNode 的数据结构设计

3.2.2 二叉树结构设计

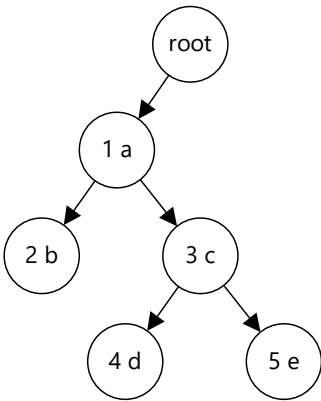


图 3-2 二叉树的数据结构设计

3.2.3 多二叉树管理结构设计

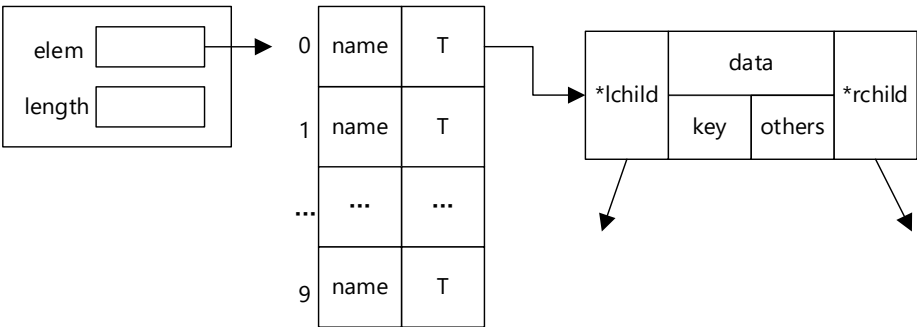


图 3-3 多二叉树管理 LIST 的数据结构设计

3.2.4 菜单功能设计

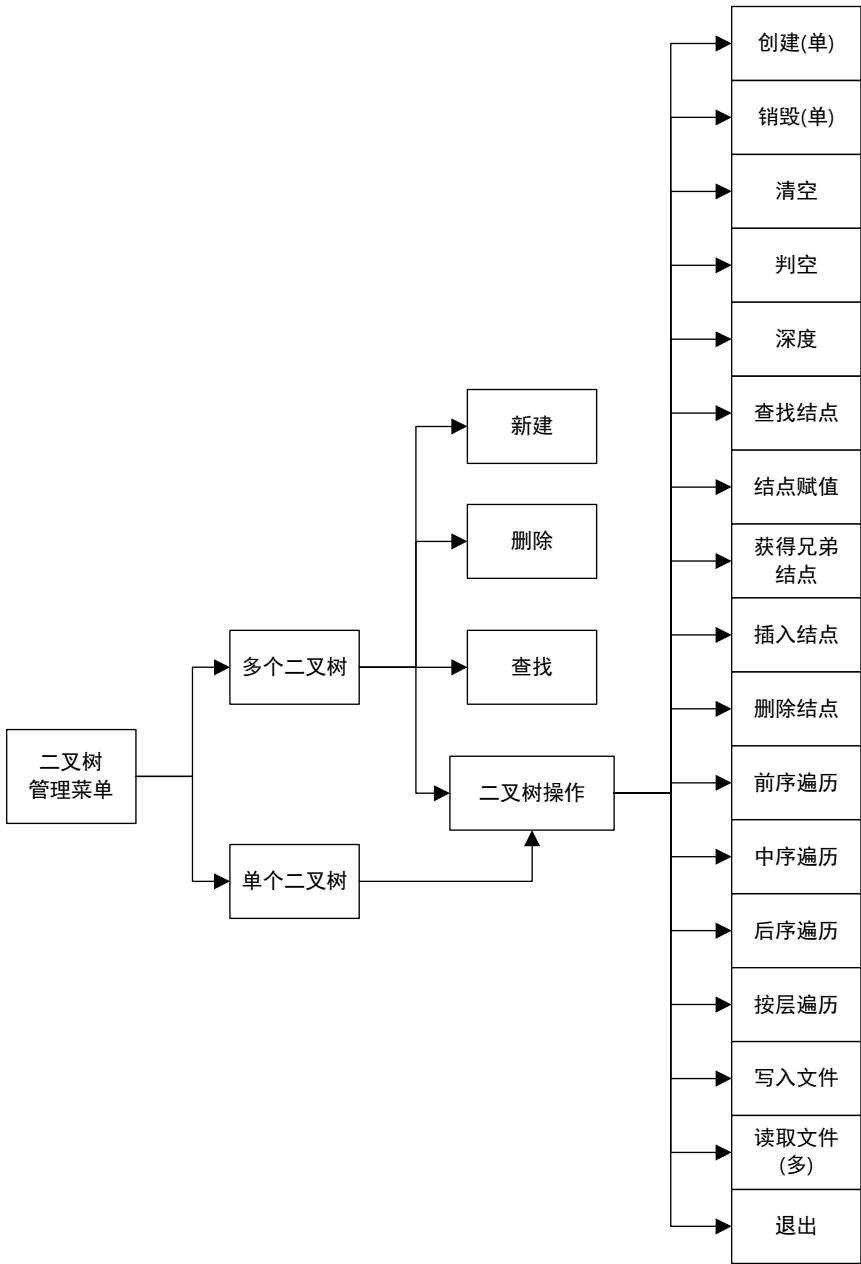


图 3-4 菜单的系统结构设计

3.2.5 文件格式设计

二进制格式(rb)写入，二进制格式(wb)写出。

3.2.6 函数设计

(1) 创建二叉树：CreateBiTree(T, definition)；初始条件是 T 不存在，definition 给出二叉树 T 的定义，操作结果是按 definition 构造二叉树 T。

(2) 销毁二叉树: 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: 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, 保留根节点。

(4) 判定空二叉树: 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: 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: 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: 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: 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: 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 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, 返回 OK。如果插入失败, 返回 ERROR。当 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: 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: PreOrderTraverse(T); 初始条件是二叉树 T 存在, 操作结果是先序遍历 T, 对每个结点调用函数 Visit 一次。

(12) 中序遍历: InOrderTraverse(T); 初始条件是二叉树 T 存在, 操作结果是中序遍历 T, 对每个结点调用函数 Visit 一次。

(13) 后序遍历: PostOrderTraverse(T); 初始条件是二叉树 T 存在, 操作结果是后序遍历 T, 对每个结点调用函数 Visit 一次。

(14) 按层遍历: LevelOrderTraverse(T); 初始条件是二叉树 T 存在, V 操作结果是层序遍历 T, 对每个结点调用函数 Visit 一次。

(15) 写入文件: SaveList(T, name), 初始条件是二叉树 T 已存在; 操作结果是将数据元素写入文件。

(16) 读取文件: LoadList(T, name), 初始条件是二叉树 T 已存在; 操作结果是将数据元素写入文件。

(17) 多二叉树新建: AddList(Lists, name), 操作结果创建名为 name 的二叉树。

(18) 多二叉树移除: RemoveList(Lists, i), 操作结果是将第 i+1 个二叉树移除。

(19) 多二叉树查找: LocateList(Lists, name), 操作结果是将查找名为 name 的二叉树 T 并返回逻辑序号。

(20) 二叉树先序+中序遍历: Traverse(T), 操作结果是输出二叉树 T 先序遍历和中序遍历的结果, 用于检查执行操作后的二叉树的正确性。

(21) 访问结点: Visit(T), 操作结果是输出结点的数据 key 和 others。

3.3 系统实现

3.3.1 二叉树的创建

1. 判断二叉树是否存在, 存在输出 INFEASIBLE, 结束功能
2. 否则用 malloc 给 root 分配内存
3. 输入结点的位序、关键值、字符串, 储存在数组 defination 中
4. 遍历 defination, 判断是否有结点的关键值不唯一, 有输出 ERROR
5. 否则按输入顺序给结点分配内存, 按位序储存在指针数组相应的位置中
6. 根据位序分配左右孩子的指针域
7. root 的左孩子 lchild 作为根结点, 位序为 1

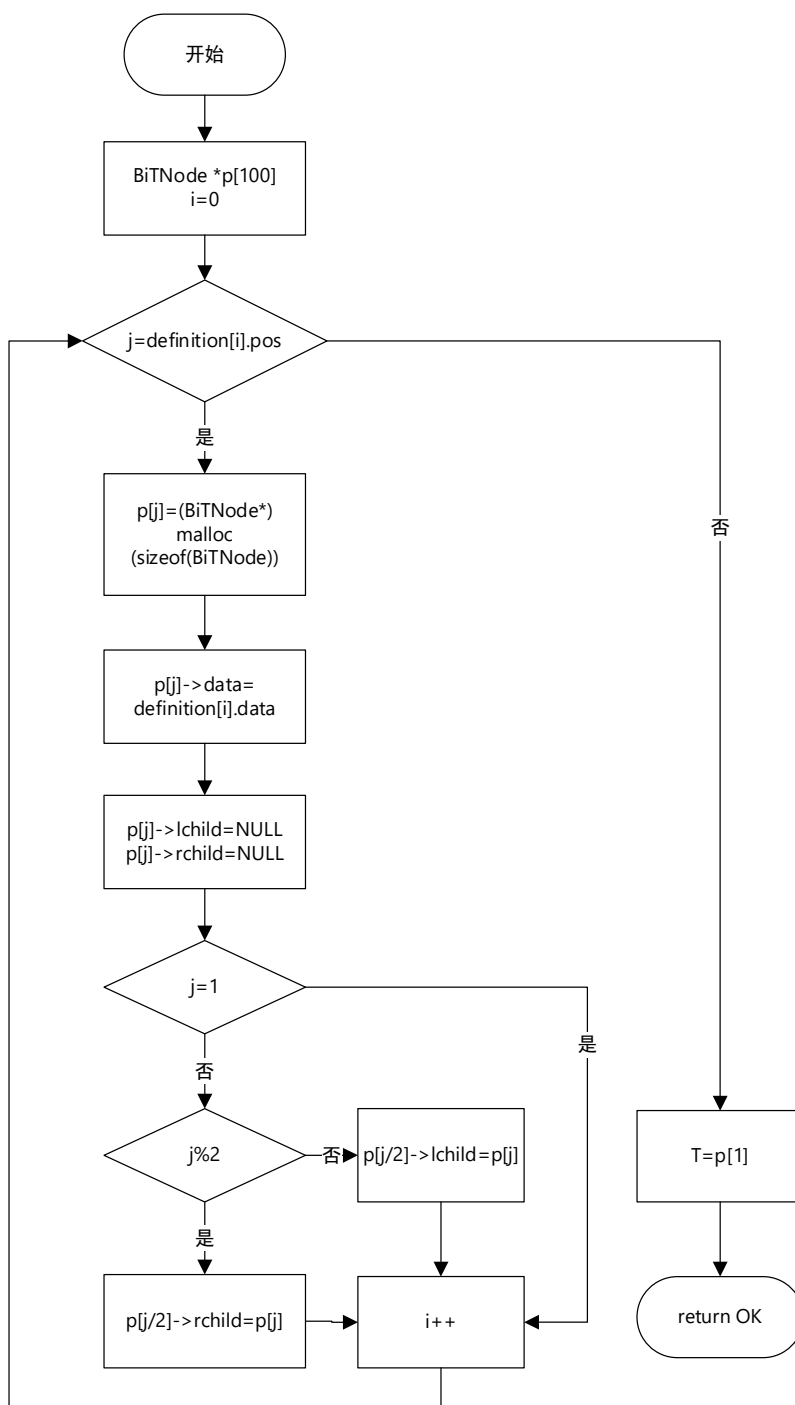


图 3-5 二叉树的创建函数流程图

3.3.2 销毁二叉树

1. 判断二叉树是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断根节点是否为空，若为空输出空二叉树
3. 否则从头结点开始依次释放各个结点的内存，置为 NULL
4. 释放 root 的内存，置为 NULL

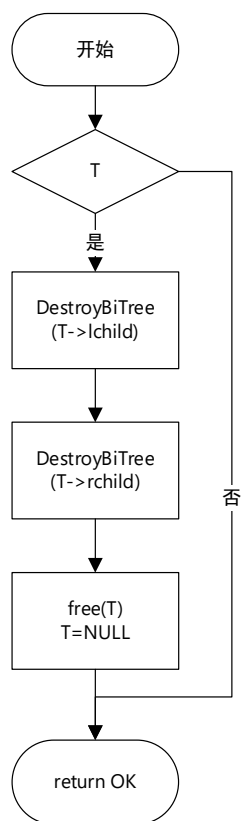


图 3-6 二叉树的销毁函数流程图

3.3.3 清空二叉树

1. 判断二叉树是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断根结点是否为空，若为空输出空二叉树
3. 否则从头结点依次释放各个结点的内存，置为 NULL
4. 保留 root

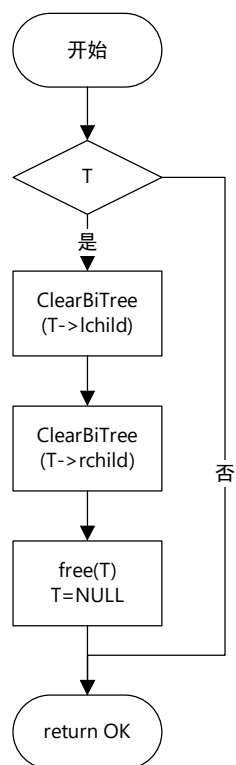


图 3-7 二叉树的清空函数流程图

3.3.4 二叉树判空

1. 判断二叉树是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断根结点是否为空，若是返回 TRUE，否则返回 FALSE

3.3.5 二叉树深度

1. 判断二叉树是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则先序递归遍历二叉树，以 a 和 b 记录左子树和右子树的深度，若结点为 NULL 返回 0，否则返回 a+1 和 b+1 中的较大值

3.3.6 查找结点

1. 判断二叉树是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断根节点是否为空，若为空输出空二叉树
3. 否则先序递归遍历二叉树，若结点关键字等于 e，返回指向该结点的指针。

调用函数前指针 q=NULL

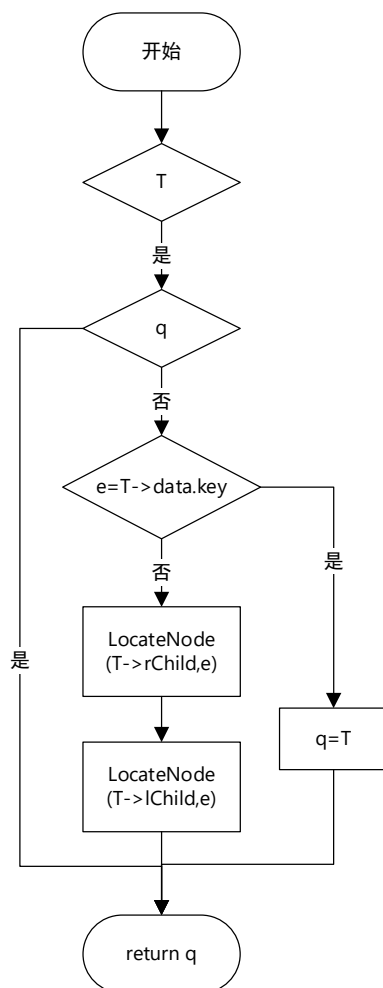


图 3-8 查找结点函数流程图

3.3.7 结点赋值

1. 判断二叉树是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断根结点是否为空，若为空输出空二叉树
3. 否则调用 LocateNode(T, value.key)，判断 value.key 是否与非目标结点的关键字相等，若相等输出 ERROR，结束功能
4. 否则调用 LocateNode(T, e)，将返回指针指向的结点的 data 改为 value

3.3.8 获得兄弟结点

1. 判断二叉树是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断根结点是否为空，若为空输出空二叉树
3. 否则先序递归遍历二叉树，若结点的左孩子关键值等于 e，返回指向右孩子的指针；若结点的右孩子关键值与 e 相同，返回指向左孩子的指针。输出指针

指向结点的 data。

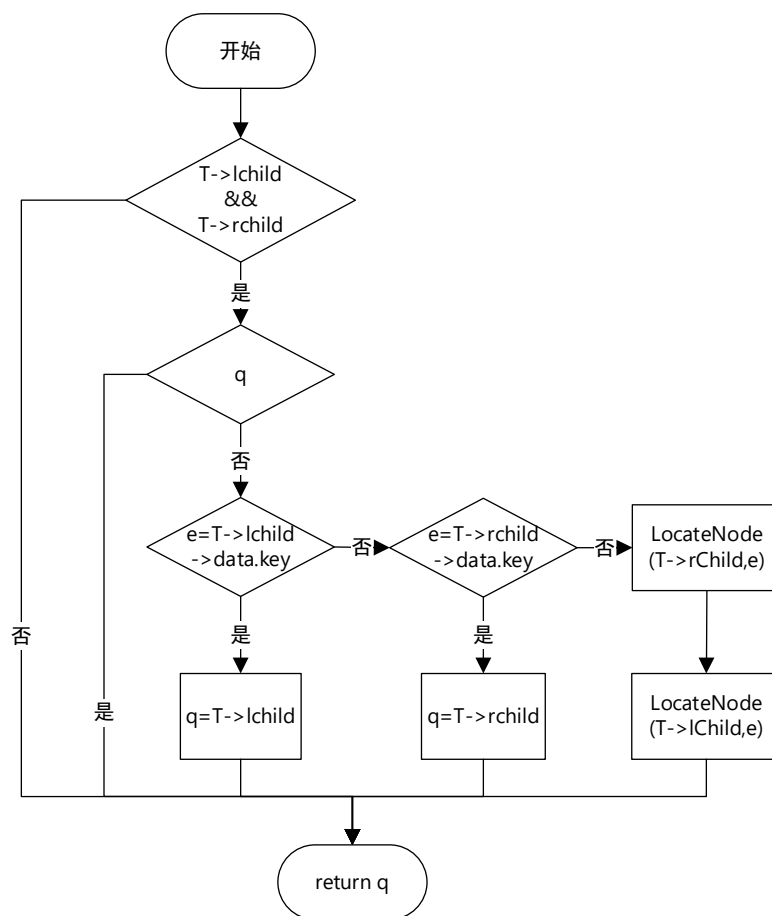


图 3-9 查找兄弟结点函数流程图

3.3.9 插入结点

1. 若二叉树存在，调用 LocateNode(T, value.key)，判断 value.key 是否与非目标结点的关键字相等，若相等输出 ERROR，结束功能

2. 否则创建新结点 newnode，data 赋值为 c。调用 LocateNode(T, e)，返回指向关键值为 e 的结点的指针。LR 为 0 或者 1，newnode 作为结点 e 的左或右孩子结点，结点 e 的原有左或右子树则为结点 c 的右子树，返回 OK。LR 为-1 时，newnode 作为根结点插入，作为 root 的左子树，原头结点作为 c 的右子树

3. 若二叉树为空，将输入的数据写入新结点，作为根结点，即 root 的左子树

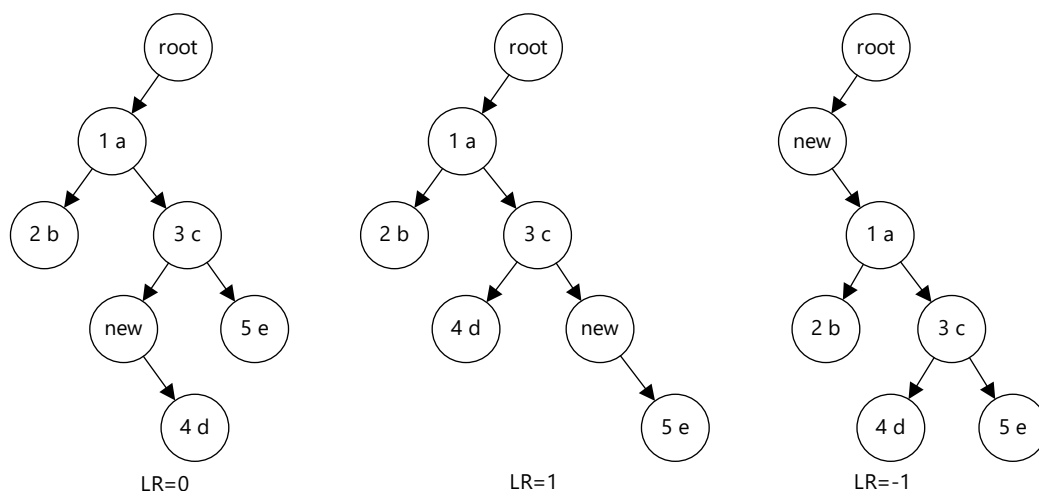


图 3-10 插入结点示意图

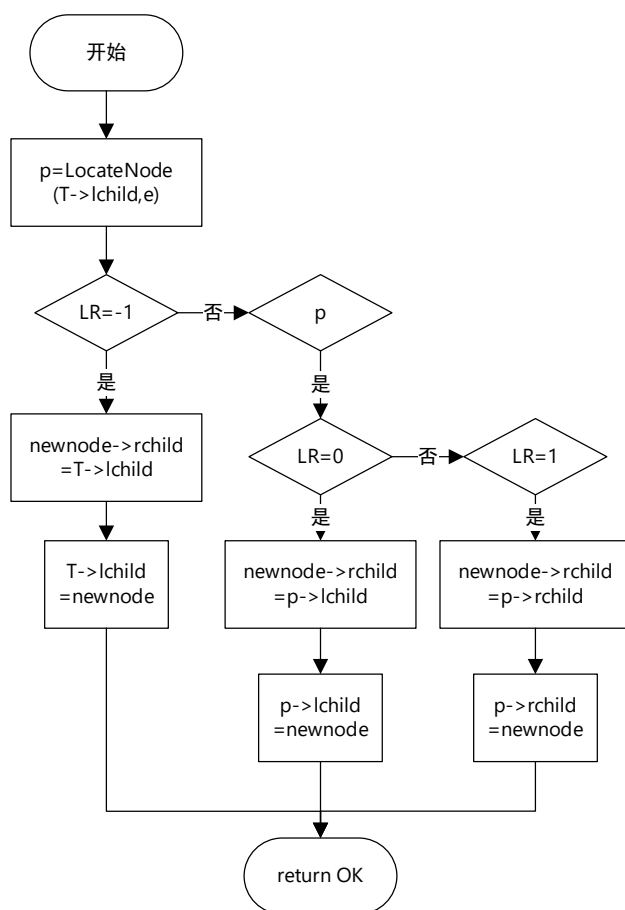


图 3-11 插入结点流程图

3.3.10 删除结点

1. 判断二叉树是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断根结点是否为空，若为空输出空二叉树
3. 否则调用 LocateNode(T, e)，返回关键值为 e 的结点的指针

4. 若指针为空输出 ERROR，结束功能

5. 否则删除该结点，释放内存。若结点 e 的结点度为 0，删除即可；若关键字为 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。

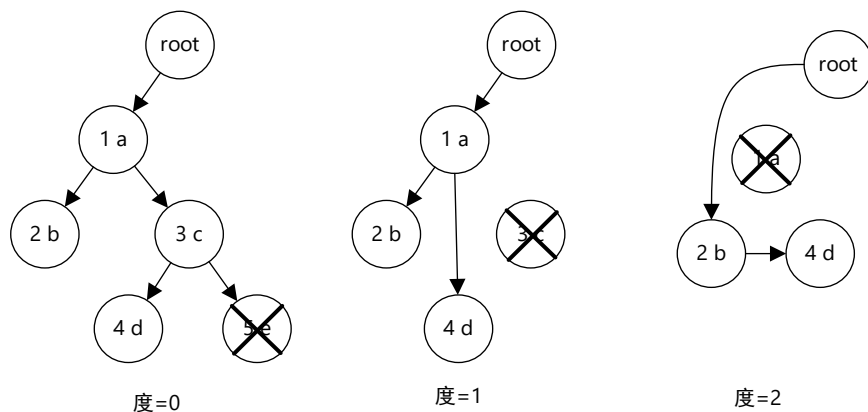


图 3-12 删除结点示意图

flag0 为外部变量，用于标记功能是否已实现成功

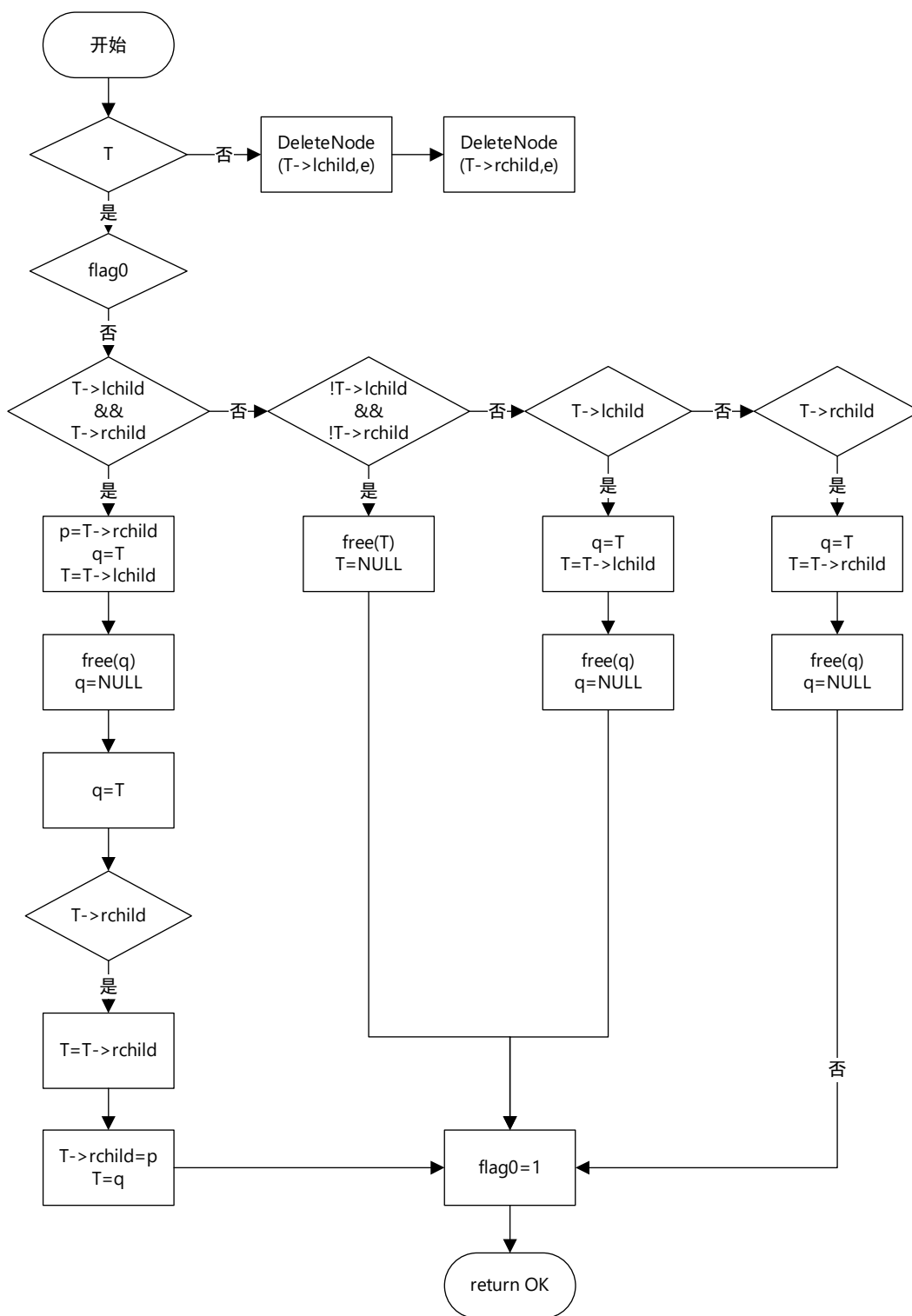


图 3-13 删除结点流程图

3.3.11 先序遍历

1. 判断二叉树是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断根结点是否为空，若为空输出空二叉树

3. 否则先序递归遍历二叉树：

若二叉树为空，则遍历结束

否则调用 Visit(T) (1) 访问根结点；(2) 遍历根的左子树；(3) 遍历根的右子树。

3.3.12 中序遍历

1. 判断二叉树是否存在，不存在输出 INFEASIBLE，结束功能

2. 否则判断根结点是否为空，若为空输出空二叉树

3. 否则中序递归遍历二叉树：

若二叉树为空，则遍历结束

否则调用 Visit(T) (1) 遍历根的左子树；(2) 访问根结点；(3) 遍历根的右子树。

3.3.13 后序遍历

1. 判断二叉树是否存在，不存在输出 INFEASIBLE，结束功能

2. 否则判断根结点是否为空，若为空输出空二叉树

3. 否则后序递归遍历二叉树：

若二叉树为空，则遍历结束

否则调用 Visit(T) (1) 遍历根的左子树；(2) 遍历根的右子树；(3) 访问根结点。

3.3.14 按层遍历

1. 判断二叉树是否存在，不存在输出 INFEASIBLE，结束功能

2. 否则判断根结点是否为空，若为空输出空二叉树

3. 否则将根结点存为指针数组的第一位，调用 Visit(T) 按顺序输出指针数组中的 data，同时将该结点的左孩子和右孩子按位序存入指针数组中。

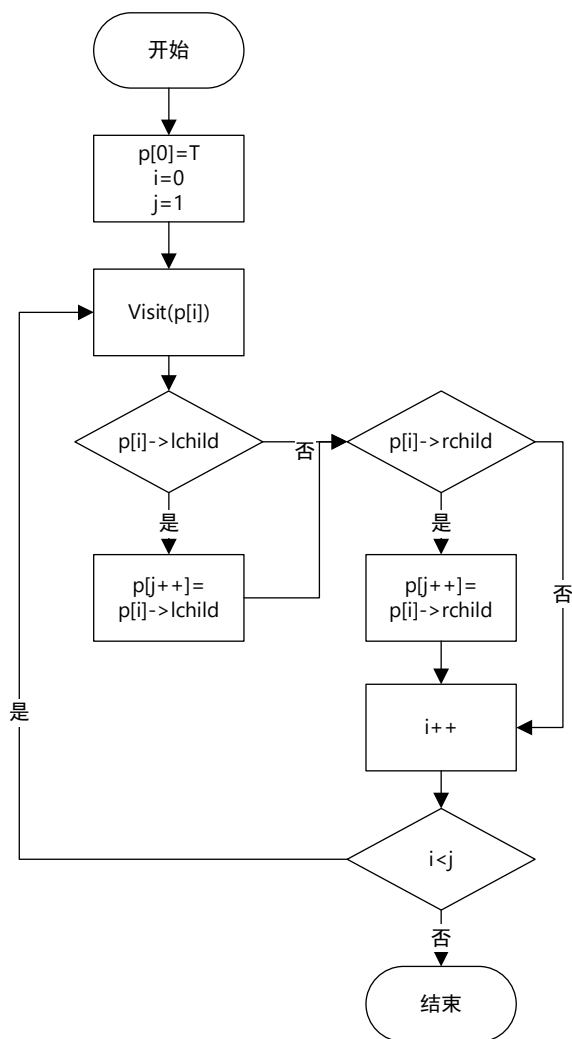


图 3-14 按层遍历流程图

3.3.15 二叉树写入文件

1. 判断二叉树是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则判断根结点是否为空，若为空输出空二叉树
3. 否则定义 FILE 文件指针
4. 若文件名不合法，输出 ERROR
5. 否则指针打开文件，将根结点存为指针数组的第一位，fprintf 遍历指针数组，将结点的数据元素写入 filename，同时将该同时将该结点的左孩子和右孩子按位序存入指针数组中。

3.3.16 二叉树读取文件

1. 定义 FILE 文件指针

2. 若文件名不合法，输出打开文件失败
3. 否则若 name 与已存在的二叉树重名，输出重名，重新输入
4. 创建名为 name 二叉树，加入多二叉树管理中。指针打开文件，fscanf 遍历 filename，将 filename 中的元素写入 DEF 数组中，调用 CreateBiTree(T,definition)创建新二叉树。
5. 输出新二叉树 name 的数据元素

3.3.17 多二叉树管理：增加一个新二叉树

1. 若增加后二叉树总数大于 10，输出 ERROR，结束功能
2. 若 name 与已存在的二叉树重名，输出重名，重新输入
3. 否则将 name 写入 Lists.elem[ListsLength-1].name
4. malloc 给 root 分配内存
5. 输入二叉树元素，调用 CreateBiTree(L,definition)创建二叉树
6. Listname.length++

3.3.18 多二叉树管理：移除一个二叉树

1. 调用 LocateList, 返回值为 i
2. 若 $i \neq 0$ ，删除该二叉树（用后一个二叉树覆盖前一个），否则输出 ERROR，结束功能

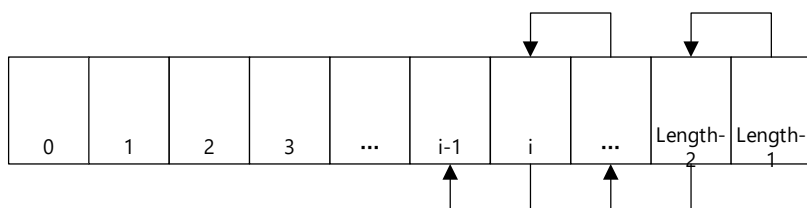


图 3-16 删除二叉树示意图

3. length--

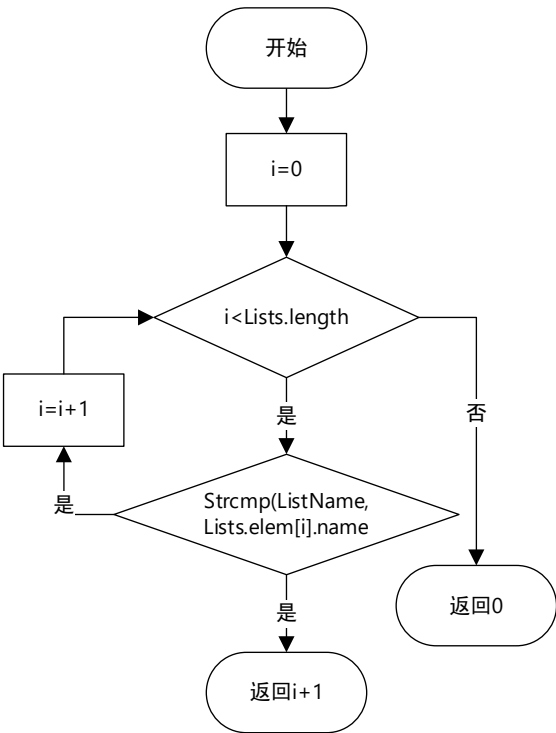


图 3-15 多二叉树查找流程图

3.3.19 多二叉树管理：查找二叉树

- 1. 同上题遍历名字，查找对应二叉树，返回 i
- 2. 若 i != 0，输出该二叉树数据，否则输出 ERROR，结束功能

3.4 系统测试

3.4.1 菜单管理

功能 1、2 仅为单二叉树操作，功能 16~19 仅为多二叉树操作。若功能标号的选择与单或多选择不匹配，将不实现功能。

Menu for Linear Table On Sequence Structure			
1. CreateBiTree(单)	10. DeleteNode		
2. DestroyBiTree(单)	11. PreOrderTraverse		
3. ClearBiTree	12. InOrderTraverse		
4. BiTreeEmpty	13. PostOrderTraverse		
5. BiTreeDepth	14. LevelOrderTraverse		
6. LocateNode	15. SaveBiTree		
7. Assign	16. LoadBiTree(多)		
8. GetSibling	17. AddList(多)		
9. InsertNode	18. RemoveList(多)		
0. exit	19. LocateList(多)		

单二叉树操作输入0，多二叉树操作输入1:

图 3-17 菜单

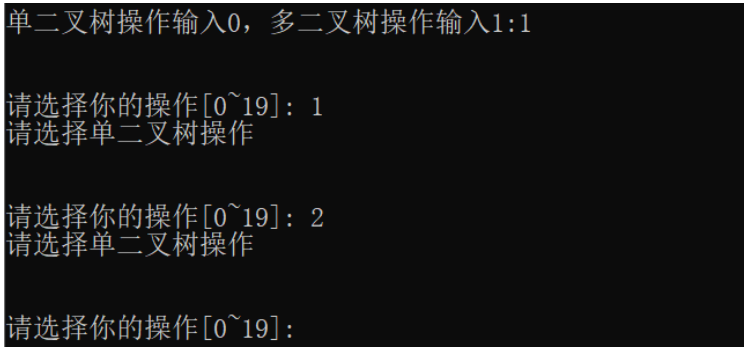


图 3-18 仅单二叉树功能

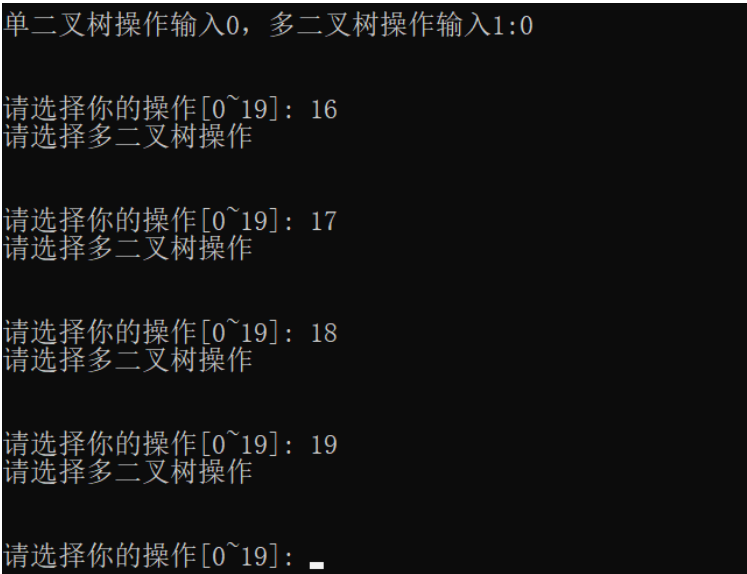


图 3-19 仅多二叉树功能

3.4.2 测试结果及分析

功能编号	输出结果	输出分析	功能编号	输出结果	输出分析
1	OK	二叉树创建成功	10	数字+字母	插入成功后的二叉树前序+中序
	ERROR	未正确分配内存		空二叉树	空二叉树
	INFEASIBLE	二叉树已存在		ERROR	未找到
2	OK	二叉树已销毁	11	INFEASIBLE	二叉树不存在
	ERROR	未正确释放内存		数字+字母	二叉树前序遍历
	INFEASIBLE	二叉树不存在		空二叉树	空二叉树
				INFEASIBLE	二叉树不存在

3	OK	二叉树已清空	12	数字+字母	二叉树中序遍历
	INFEASIBLE	二叉树不存在		空二叉树	空二叉树
	INFEASIBLE	二叉树不存在		INFEASIBLE	二叉树不存在
4	TRUE	二叉树为空	13	数字+字母	二叉树后序遍历
	FALSE	二叉树不为空		空二叉树	空二叉树
	INFEASIBLE	二叉树不存在		INFEASIBLE	二叉树不存在
5	数字	二叉树深度	14	数字+字母	二叉树按层遍历
	INFEASIBLE	二叉树不存在		空二叉树	空二叉树
	INFEASIBLE	二叉树不存在		INFEASIBLE	二叉树不存在
6	数字+字母	二叉树的数据	14	OK	读入文件成功
	空二叉树	空二叉树		空二叉树	空二叉树
	ERROR	未找到		ERROR	打开文件失败
	INFEASIBLE	二叉树不存在		INFEASIBLE	二叉树不存在
7	数字+字母 数字+字母	替换成功后的二 叉树前序+中序	16	数字+字母	根据文件创建的 二叉树的数据
	空二叉树	空二叉树		空二叉树	空二叉树
	ERROR	未找到或替换的 关键值不合法		ERROR	打开文件失败
	INFEASIBLE	二叉树不存在		INFEASIBLE	二叉树不存在
8	数字+字母	兄弟结点的数据	17	OK	二叉树创建成功
	空二叉树	空二叉树		ERROR	未正确分配内存
	ERROR	查找失败		重名	二叉树重名
	INFEASIBLE	二叉树不存在		OVERFLOW	溢出
9	数字+字母 数字+字母	插入成功后的二 叉树前序+中序	18	数字串	删除后的所有二 叉树名+元素
	ERROR	查找失败		删除失败	二叉树不存在
	INFEASIBLE	二叉树不存在			
0	退出菜单		19	数字+字母	查找的二叉树名 +数据

			查找失败	二叉树不存在
--	--	--	------	--------

表 3-1 测试结果及分析

3.4.3 测试用例

1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null

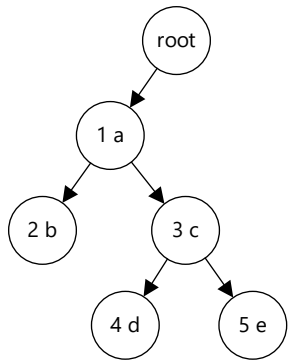


图 3-20 测试用例 1

1 1 a 2 2 b 3 3 c 4 4 d 7 5 e 7 6 f 0 0 null

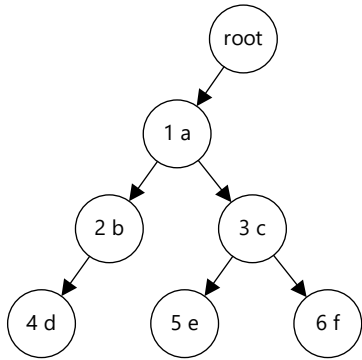


图 3-21 测试用例 2

3.4.4 测试截图

测试完用例后，依次实现 3 清空和 2 销毁功能，分别测试空二叉树和二叉树不存在的情况。

1 创建

```

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 1 c 0 0 null
ERROR

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 1
INFEASIBLE
    
```

图 3-22 功能 1 的用例

2 销毁

```

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 2
INFEASIBLE
    
```

图 3-23 功能 2 的用例

3 清空

```

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 4
FALSE

请选择你的操作[0~19]: 3
OK

请选择你的操作[0~19]: 4
TRUE

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 4
INFEASIBLE
    
```

图 3-24 功能 3 的用例

4 判空

```

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 4
FALSE

请选择你的操作[0~19]: 3
OK

请选择你的操作[0~19]: 4
TRUE

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 4
INFEASIBLE
    
```

图 3-25 功能 4 的用例

5 求深度

```

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 5
3

请选择你的操作[0~19]: 3
OK

请选择你的操作[0~19]: 5
0

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 5
INFEASIBLE
    
```

图 3-26 功能 5 的用例

6 查找结点

```

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 6
输入想要查找的节点的关键值: 1
1, a

请选择你的操作[0~19]: 6
输入想要查找的节点的关键值: 4
4, d

请选择你的操作[0~19]: 6
输入想要查找的节点的关键值: 7
ERROR

请选择你的操作[0~19]: 3
OK

请选择你的操作[0~19]: 6
空二叉树

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 6
INFEASIBLE
    
```

图 3-27 功能 6 的用例

7 替换结点

```

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 7
输入想要查找的节点的关键值: 5
输入想要替换的结点值: 5 z
1, a 2, b 3, c 4, d 5, z
2, b 1, a 4, d 3, c 5, z

请选择你的操作[0~19]: 7
输入想要查找的节点的关键值: 5
输入想要替换的结点值: 1 z
ERROR

请选择你的操作[0~19]: 7
输入想要查找的节点的关键值: 7
输入想要替换的结点值: 5 z
ERROR

请选择你的操作[0~19]: 3
OK

请选择你的操作[0~19]: 7
空二叉树

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 7
INFEASIBLE
    
```

图 3-28 功能 7 的用例

8 查找兄弟结点

```

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 8
输入想要查找的节点的关键值: 4
5, e

请选择你的操作[0~19]: 8
输入想要查找的节点的关键值: 1
ERROR

请选择你的操作[0~19]: 3
OK

请选择你的操作[0~19]: 8
空二叉树

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 8
INFEASIBLE
    
```

图 3-29 功能 8 的用例

9 插入结点

```

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 9
输入想要插入的节点的关键值: 0
L:0,R:1, 根节点:-1 :-1
输入想要插入的结点值: 6 f
6, f 1, a 2, b 3, c 4, d 5, e
6, f 2, b 1, a 4, d 3, c 5, e

请选择你的操作[0~19]: 9
输入想要插入的节点的关键值: 3
L:0,R:1, 根节点:-1 :1
输入想要插入的结点值: 7 g
6, f 1, a 2, b 3, c 4, d 7, g 5, e
6, f 2, b 1, a 4, d 3, c 7, g 5, e

请选择你的操作[0~19]: 9
输入想要插入的节点的关键值: 1
L:0,R:1, 根节点:-1 :0
输入想要插入的结点值: 8 h
6, f 1, a 8, h 2, b 3, c 4, d 7, g 5, e
6, f 8, h 2, b 1, a 4, d 3, c 7, g 5, e

请选择你的操作[0~19]: 3
OK

请选择你的操作[0~19]: 9
输入想要插入的结点值: 1 a
1, a
1, a

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 9
INFEASIBLE
    
```

图 3-30 功能 9 的用例

10 删除结点

```

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 10
输入想要删除的节点的关键值: 3
1, a 2, b 4, d 5, e
2, b 1, a 4, d 5, e

请选择你的操作[0~19]: 10
输入想要删除的节点的关键值: 4
1, a 2, b 5, e
2, b 1, a 5, e

请选择你的操作[0~19]: 10
输入想要删除的节点的关键值: 2
1, a 5, e
1, a 5, e

请选择你的操作[0~19]: 3
OK

请选择你的操作[0~19]: 10
空二叉树

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 10
INFEASIBLE
    
```

图 3-31 功能 10 的用例

11 前序遍历

```

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 11
1, a 2, b 3, c 4, d 5, e

请选择你的操作[0~19]: 3
OK

请选择你的操作[0~19]: 11
空二叉树

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 11
INFEASIBLE
    
```

图 3-32 功能 11 的用例

12 中序遍历

```

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 12
2, b 1, a 4, d 3, c 5, e

请选择你的操作[0~19]: 3
OK

请选择你的操作[0~19]: 12
空二叉树

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 12
INFEASIBLE
    
```

图 3-33 功能 12 的用例

13 后序遍历

```

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 13
2, b 4, d 5, e 3, c 1, a

请选择你的操作[0~19]: 3
OK

请选择你的操作[0~19]: 13
空二叉树

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 13
INFEASIBLE
    
```

图 3-34 功能 13 的用例

14 按层遍历

```

请选择你的操作[0~19]: 1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 14
1, a 2, b 3, c 4, d 5, e

请选择你的操作[0~19]: 3
OK

请选择你的操作[0~19]: 14
空二叉树

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 14
    
```

图 3-35 功能 14 的用例

15 & 16 文件

```

请选择你的操作[0~19]: 17
输入要增加的二叉树的个数:1
输入二叉树名字:1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK

请选择你的操作[0~19]: 15

输入操作的二叉树的名字: 1
输入文件名:1.txt
OK

请选择你的操作[0~19]: 16
输入文件名: 1.txt
输入二叉树名:1
重名
输入二叉树名:2
1, a 2, b 3, c 4, d 5, e
2, b 1, a 4, d 3, c 5, e
    
```

图 3-36 功能 15. 16 的用例

1.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

1 1 a 2 2 b 3 3 c 6 4 d 7 5 e

图 3-37 功能 15.16 的文件

17 新增二叉树

```
请选择你的操作[0~19]: 17
输入要增加的二叉树的个数:2
输入二叉树名字:1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK
输入二叉树名字:1
重名
输入二叉树名字:2
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 4 4 d 7 5 e 7 6 f 0 0 null
OK

请选择你的操作[0~19]: 17
输入要增加的二叉树的个数:9
OVERFLOW
```

图 3-38 功能 17 的用例

18 移除二叉树

```

请选择你的操作[0~19]: 17
输入要增加的二叉树的个数:3
输入二叉树名字:1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK
输入二叉树名字:2
请输入二叉树元素, 结束输入0 0 null: 0 0 null
OK
输入二叉树名字:3
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 4 4 d 6 5 e 7 6 f 0 0 null
OK

请选择你的操作[0~19]: 18
输入要删除的二叉树的名字:1
2
空线性表
3
1, a 2, b 4, d 3, c 5, e 6, f
4, d 2, b 1, a 5, e 3, c 6, f

请选择你的操作[0~19]: 18
输入要删除的二叉树的名字:2
3
1, a 2, b 4, d 3, c 5, e 6, f
4, d 2, b 1, a 5, e 3, c 6, f

请选择你的操作[0~19]: 18
输入要删除的二叉树的名字:4
删除失败
    
```

图 3-39 功能 18 的用例

19 查找二叉树

```

请选择你的操作[0~19]: 17
输入要增加的二叉树的个数:3
输入二叉树名字:1
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null
OK
输入二叉树名字:2
请输入二叉树元素, 结束输入0 0 null: 0 0 null
OK
输入二叉树名字:3
请输入二叉树元素, 结束输入0 0 null: 1 1 a 2 2 b 3 3 c 4 4 d 6 5 e 7 6 f 0 0 null
OK

请选择你的操作[0~19]: 19
输入要查找的二叉树的名字:1
1
1, a 2, b 3, c 4, d 5, e
2, b 1, a 4, d 3, c 5, e

请选择你的操作[0~19]: 19
输入要查找的二叉树的名字:2
2
空线性表

请选择你的操作[0~19]: 19
输入要查找的二叉树的名字:4
查找失败
    
```

图 3-40 功能 19 的用例

0 退出

```

请选择你的操作[0~17]: 0

欢迎下次再使用本系统!

Process returned 0 (0x0)    execution time : 12.068 s
Press any key to continue.
    
```

图 3-41 功能 0

3.5 实验小结

1. 调用对二叉树本身进行操作的函数时, 要注意传递过去的是二叉树本身的指针还是赋值的其他指针。
2. 销毁线性表时要销毁根节点, 清空线性表时要保留根节点。
3. 多二叉树管理时, 对二叉树进行删或加操作后不能忘记改 Lists.length 的值。
4. 调用插入结点函数时, 若在根节点插入结点, 在调用函数之前要将根节点

的指针保留，否则调用结束后指针 T 不再指向根节点。

5. 删除结点后不能忘记释放该结点的内存，同时将父亲结点指向该结点的指针域置为 NULL。

6. 写入文件时要注意空格怎么安排，读取文件时的空格和换行格式要和写入时一致。

7. 创建新二叉树时要先遍历全表，检查是否有重名，否则查找和对二叉树进行操作时会出错。

8. 要注意函数返回值是逻辑序号还是数组编号，逻辑序号=数组编号+1。

4 基于邻接表的图实现

4.1 问题描述

通过实验达(1)加深对图的概念、基本运算的理解;(2)熟练掌握图的逻辑结构与物理结构的关系;(3)以邻接表作为物理结构, 熟练掌握图基本运算的实现。

4.1.1 邻接表的实现

通过函数形式实现图的创建、销毁、查找顶点、顶点赋值、获得第一邻接点、获得下一邻接点、插入顶点、删除顶点、插入弧、删除弧、深度优先搜索遍历、广度优先搜索遍历 12 种操作。

4.1.2 构造具有菜单的功能演示系统

通过 switch 对输入的功能编号对二叉树进行相应的操作或退出菜单, 对非法输入予以提醒。

4.1.3 邻接表的文件写入和读取

1. 输入文件名, 将邻接表写入相应的文件
2. 输入文件名, 创建邻接表并将文件中的信息写入邻接表中并输出

4.1.4 多个图管理

1. 多个图的新建、删除、根据输入的名字查找
2. 能够对其中某个图进行 4.1.2 和 4.1.3 中共 14 种操作

4.2 系统设计

4.2.1 邻接表顶点结构设计

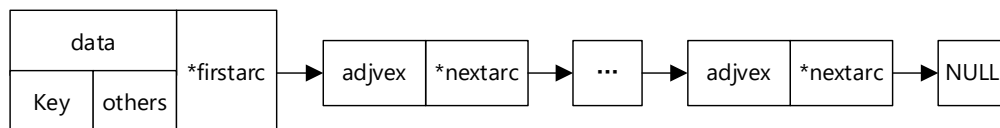


图 4-1 邻接表 G 的顶点数据结构设计

4.2.2 邻接表结构设计

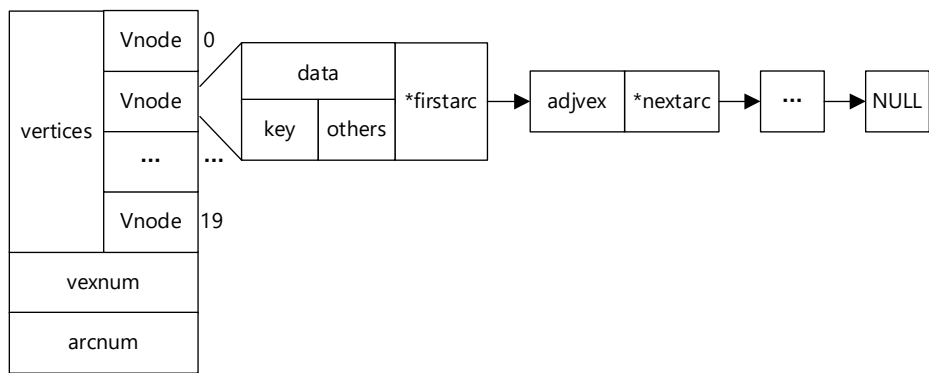


图 4-2 邻接表的数据结构设计

4.2.3 多邻接表管理结构设计

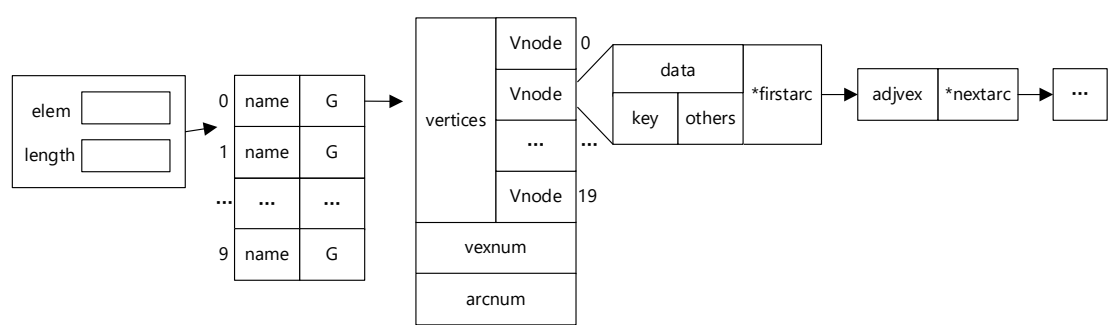


图 4-3 多图管理 LIST 的数据结构设计

4.2.4 菜单功能设计

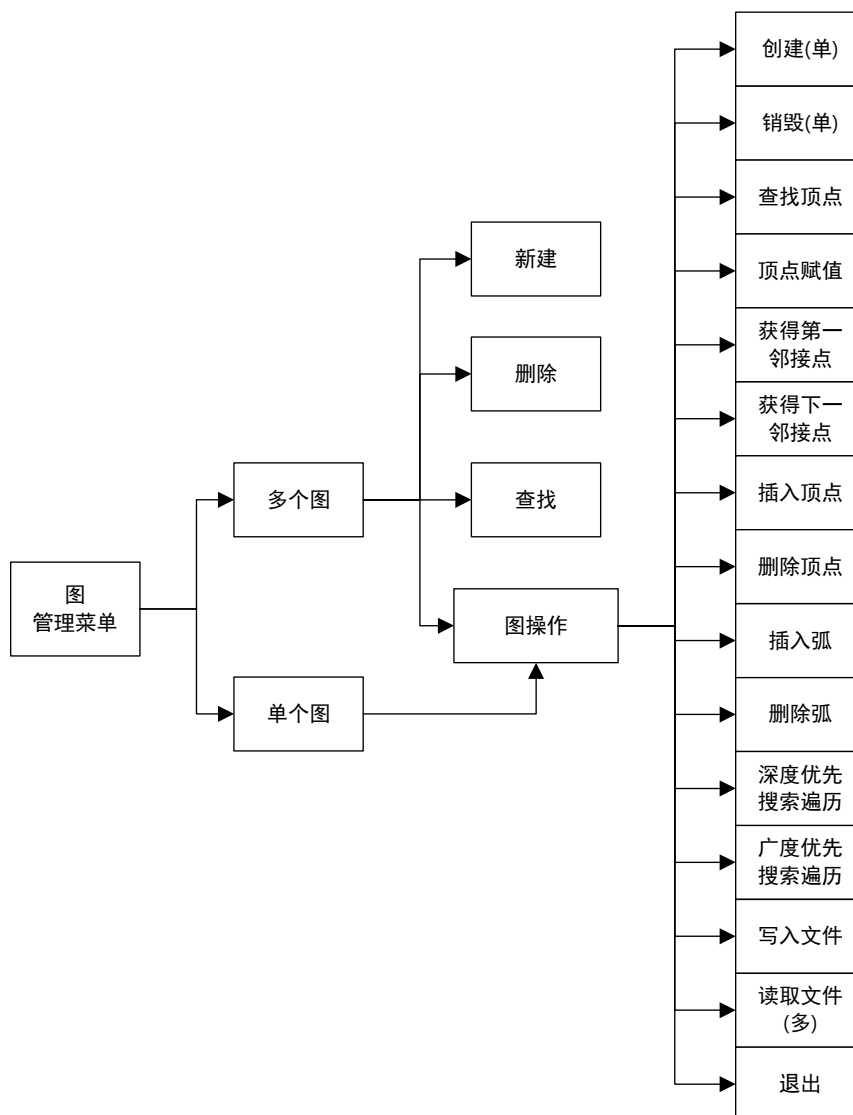


图 4-4 菜单的系统结构设计

4.2.5 文件格式设计

二进制格式(rb)写入，二进制格式(wb)写出。

4.2.6 函数设计

(1)创建图：CreateCraph(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。

(2)销毁图：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：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：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：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：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：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。

(8) 删除顶点：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：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 $\langle w, v \rangle$ 。

(10) 删除弧：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 $\langle v, w \rangle$ ，如果图 G 是无向图，还需要删除 $\langle w, v \rangle$ 。

(11) 深度优先搜索遍历：DFS Traverse (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：BFS Traverse (G)；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。

(13) 写入文件：SaveList (G, name)，初始条件是图 G 已存在；操作结果是将数据元素写入文件。

- (14) 读取文件: `LoadList(G, name)`; 操作结果是将数据元素写入文件。
- (15) 多图新建: `AddList(Lists, name)`, 操作结果创建名为 `name` 的图。
- (16) 多图移除: `RemoveList(Lists, i)`, 操作结果是将第 `i+1` 个图移除。
- (17) 多图查找: `LocateList(Lists, name)`, 操作结果是将查找名为 `name` 的图并返回逻辑序号。
- (18) 图遍历: `Traverse(G)`, 操作结果是输出图 `G` 的全部数据元素, 用于检查执行操作后的图的正确性。
- (19) 访问结点: `Visit(v)`, 操作结果是输出顶点的数据 `key` 和 `others`。

4.3 系统实现

4.3.1 创建图

1. 判断图是否存在, 若存在输出 `INFEASIBLE`, 结束功能
2. 否则输入顶点数据, 储存在数组 `V` 中
3. 判断顶点关键值是否唯一, 若不唯一, 输出 `ERROR`, 结束功能
4. 若唯一, 输入弧数据, 储存在二维数组 `VR` 中
5. 判断储存弧的数组中是否有不包含在顶点关键值中的数字, 若有, 输出 `ERROR`, 结束功能
6. 否则调用 `CreateGraph` 函数根据 `V` 和 `VR` 创建图 `G`, 先根据 `V` 给各个顶点依次赋值, 再根据 `VR` 先进后出创建单链表, 返回 `OK`

4.3.2 销毁图

1. 判断图是否存在, 不存在输出 `INFEASIBLE`, 结束功能
2. 否则一次释放每个顶点的内存, 将头指针置为 `NULL`
3. `G.vexnum=G.arcnum=0`

4.3.3 查找顶点

1. 判断图是否存在, 不存在输出 `INFEASIBLE`, 结束功能
2. 否则 `for` 循环查找关键值为 `u` 的顶点, 返回序号, 否则返回-1

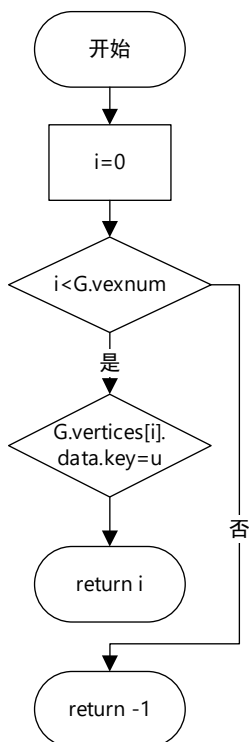


图 4-5 查找顶点流程图

4.3.4 顶点赋值

1. 判断图是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则调用 $\text{LocateVex}(G, u)$ ，返回值为 i
3. 若 $i=-1$ ，或赋值的关键值与非目标顶点相等，返回 ERROR，结束功能
4. 否则给目标顶点赋值为 value

4.3.5 获得第一邻接顶点

1. 判断图是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则调用 $\text{LocateVex}(G, u)$ ，返回值为 i
3. 若 $i=-1$ ，返回 ERROR，结束功能
4. 否则返回目标顶点的第一个结点的数据域

4.3.6 获得下一邻接顶点

1. 判断图是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则调用 $\text{LocateVex}(G, v)$ 和 $\text{LocateVex}(G, w)$ ，返回值为 i 和 j
3. 若 $i=-1$ 或 $j=-1$ ，返回 ERROR，结束功能

4. 否则指针 p 遍历该顶点链表，若 p 的数据域等于 j，返回下一结点的数据域

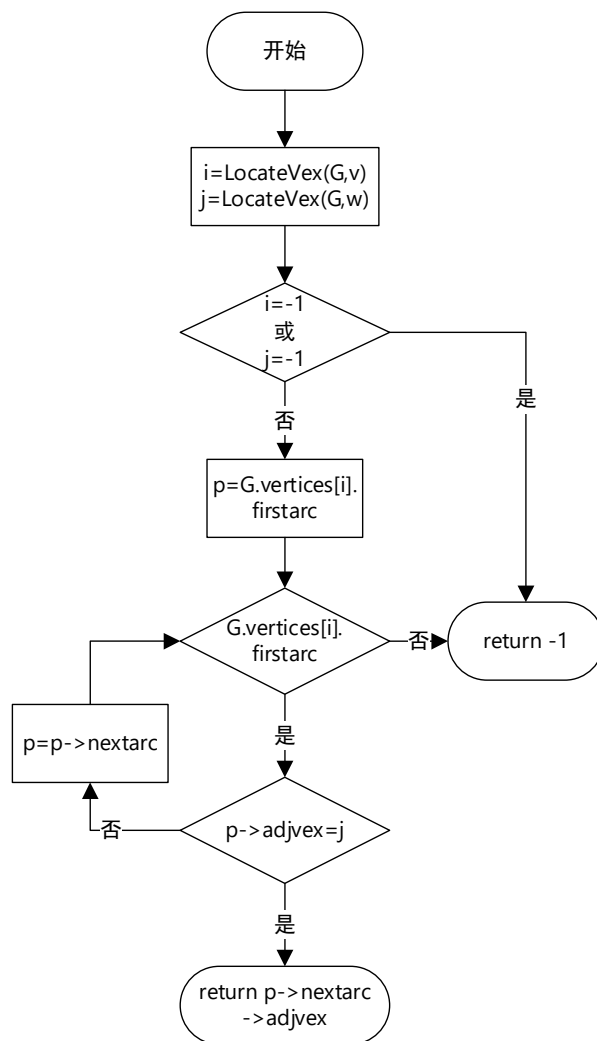


图 4-6 获得下一邻接顶点流程图

4.3.7 插入顶点

1. 判断图是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则调用 LocateVex(G, v)，返回值为 i
3. 若 i=-1 或 G.vexnum>19，返回 ERROR，结束功能
4. 否则给新顶点赋值，头结点的指针域置为 NULL，返回 OK
5. G.vexnum++

4.3.8 删除顶点

1. 判断图是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则调用 $\text{LocateVex}(G, v)$ ，返回值为 i
3. 若 $i=-1$ 或 $G.\text{vexnum}=1$ ，返回 ERROR，结束功能
4. 否则依次释放该顶点的 vertices 的结点的内存，将头结点的指针域置为 NULL
5. $G.\text{vexnum}--$ ，删除该顶点
6. 更新邻接表中邻接顶点的序号

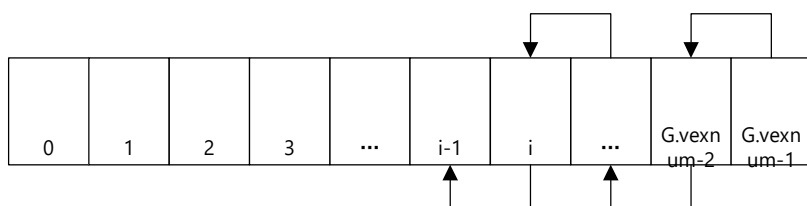


图 4-7 获得下一邻接顶点流程图

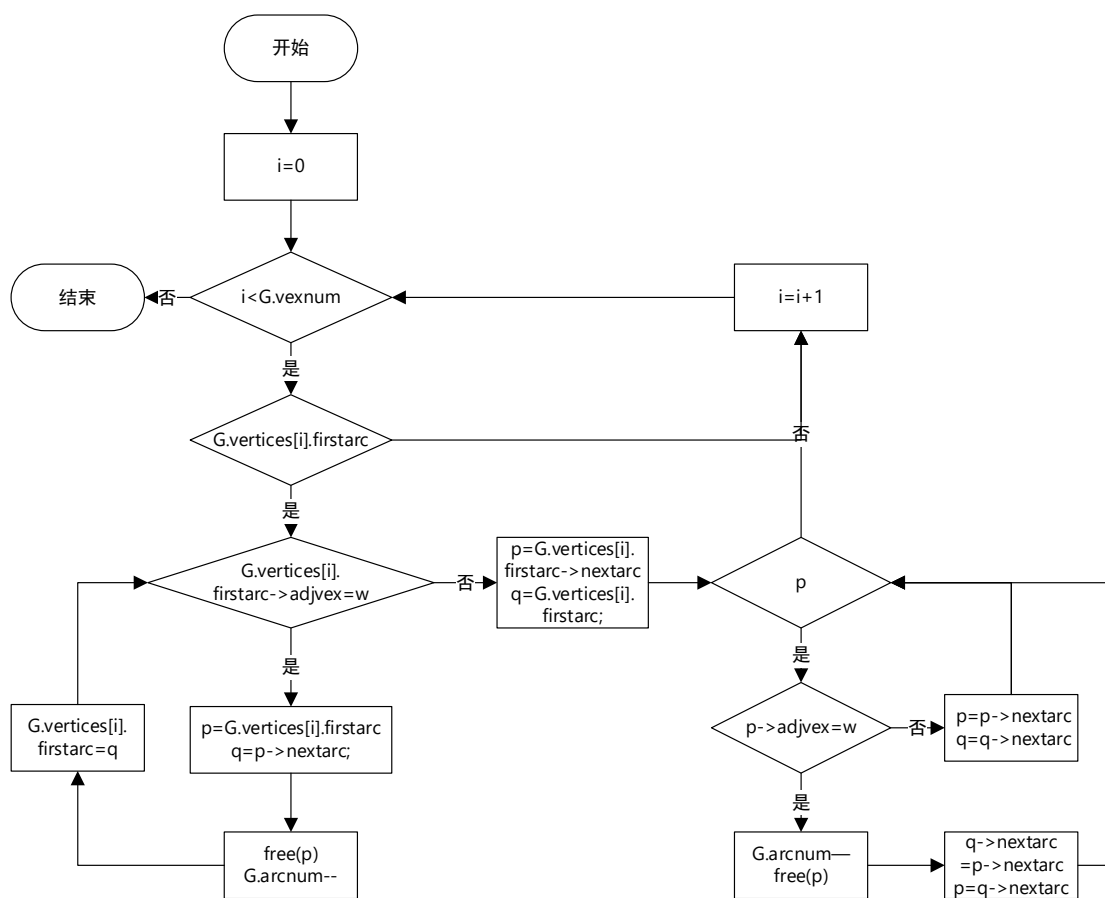


图 4-8 删除顶点流程图

4.3.9 插入弧

1. 判断图是否存在，不存在输出 INFEASIBLE，结束功能
2. 否则调用 $\text{LocateVex}(G, v)$ 和 $\text{LocateVex}(G, w)$ ，返回值为 i 和 j
3. 若 $i=-1$ 或 $j=-1$ ，返回 ERROR，结束功能
4. 否则判断弧是否已存在，若已存在，返回 ERROR，结束功能
5. 否则将目标顶点的新结点的数据域赋值为新连接顶点的序号，并插入到头结点和第一个结点之间
6. $G.\text{arcnum}++$

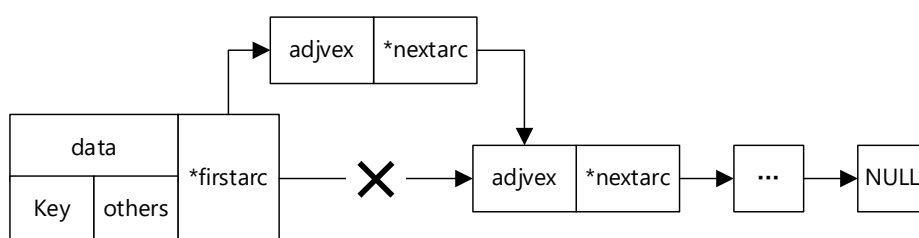


图 4-9 插入结点示意图

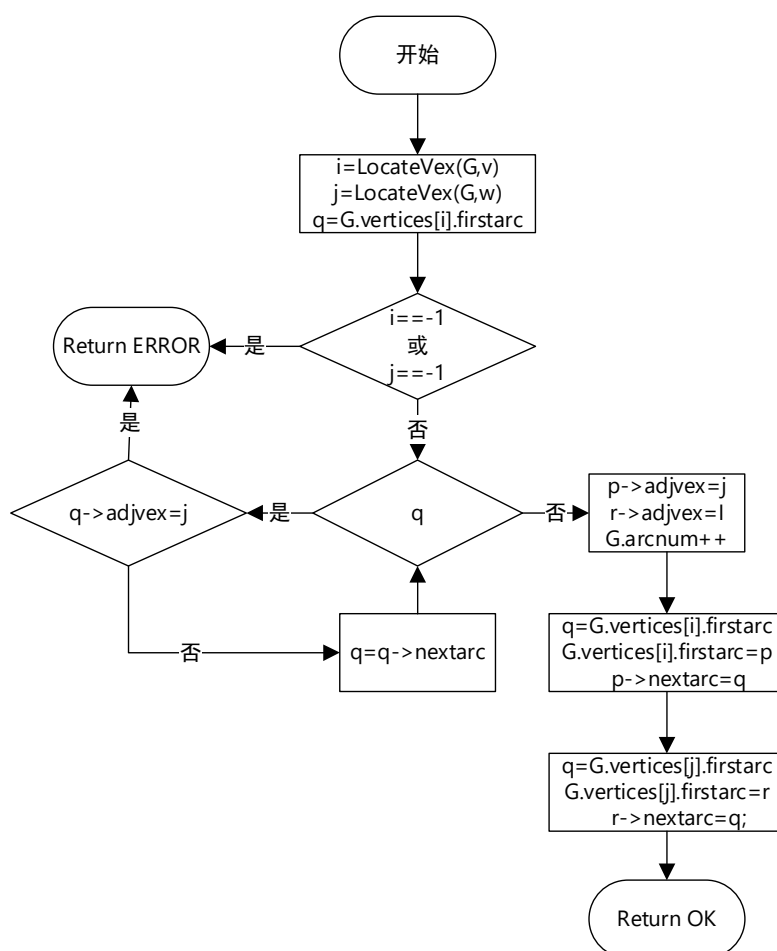


图 4-10 插入弧流程图

4.3.10 删除弧

1. 判断图是否存在，不存在输出 INFEASIBLE
2. 否则调用 LocateVex(G, v)，返回值为 i
3. 若 i=-1，返回 ERROR，结束功能
4. 否则判断弧是否存在，若不存在，返回 ERROR，结束功能
5. 否则将目标顶点的前结点的指针域赋值为后顶点，释放目标结点的内存
6. G. arcnum--

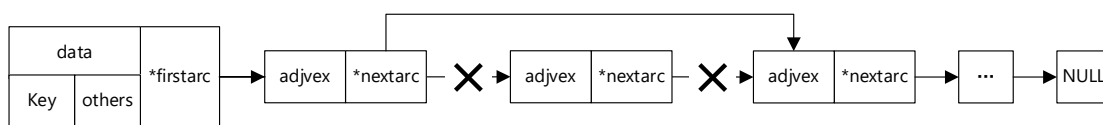


图 4-11 删除结点示意图

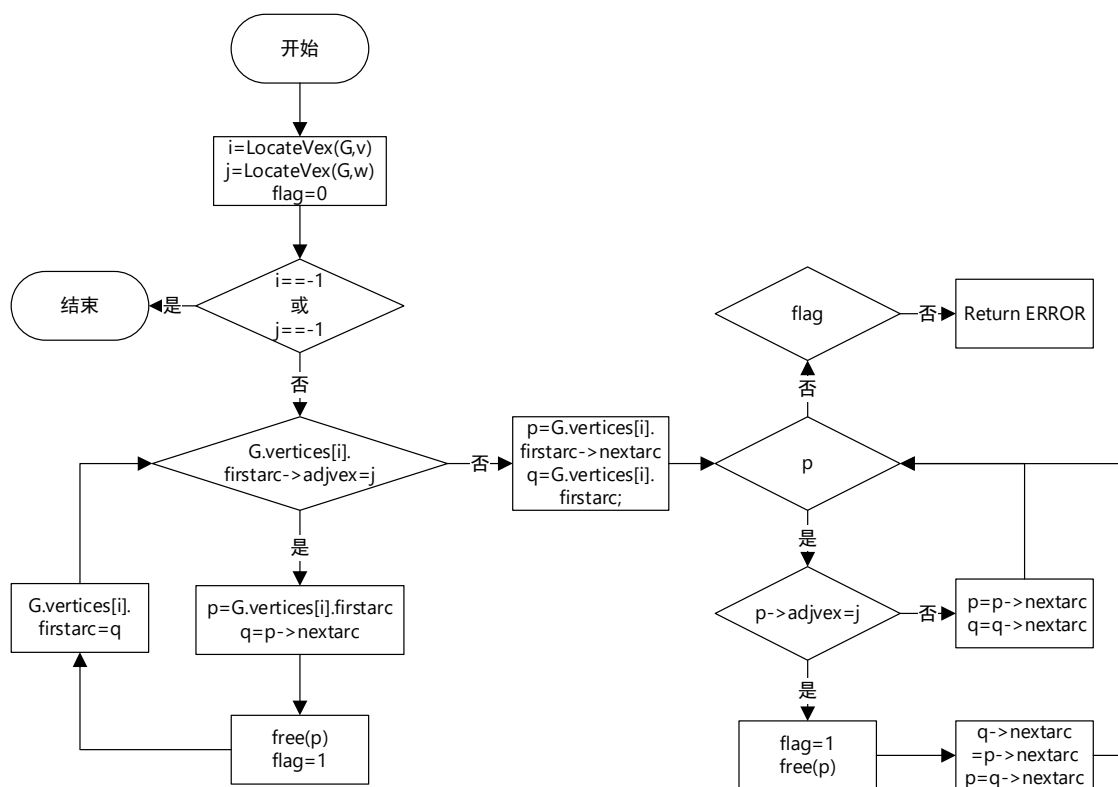


图 4-12 删除弧流程图 (其中一个顶点)

4.3.11 DFS 遍历

1. 判断图是否存在，不存在输出 INFEASIBLE，结束功能

2. 用 `visited` 数组储存每个顶点是否被访问过, `sum` 记录已访问过的顶点个数, `t` 记录当前访问的顶点
3. 调用 `Visit` 访问当前顶点, `sum++`
4. 指针 `p` 遍历当前顶点的连接点, 访问第一个未访问过的顶点, 从 3 开始递归
5. 若 `sum=G.vexnum`, 遍历结束
6. 否则继续查找下一个未访问过的顶点, 进行第二轮遍历

4.3.12 BFS 遍历

1. 判断图是否存在, 不存在输出 `INFEASIBLE`, 结束功能
2. 用 `visited` 数组储存每个顶点是否被访问过, `sum` 记录已访问过的顶点个数, `t` 记录当前访问的顶点, 队列 `q` 记录一轮是否遍历完成
3. 调用 `Visit` 访问当前顶点, `sum++`
4. 指针 `p` 遍历当前顶点的连接点, 访问所有未访问过的顶点, 储存入队列中
5. 访问队列中的下一个顶点
6. 若 `sum=G.vexnum`, 遍历结束
7. 否则继续查找下一个未访问过的顶点, 进行第二轮遍历

4.3.13 图写入文件

1. 判断图是否存在, 不存在输出 `INFEASIBLE`, 结束功能
2. 否则定义 `FILE` 文件指针
4. 若文件名不合法, 输出 `ERROR`
5. 否则指针打开文件, 写入图的顶点数和度数。
6. `fprint` 遍历图, 将顶点的数据及邻接表中的结点数据写入 `filename`

4.3.14 图读取文件

1. 定义 `FILE` 文件指针
2. 若文件名不合法, 输出打开文件失败, 结束功能
3. 否则输入图名 `name`, 若与已存在的图重名, 输出重名, 重新输入
4. 创建名为 `name` 新图, 加入多图管理中。指针打开文件, `fscanf` 遍历

filename, 写入先顶点数和度数, 再将 filename 中的顶点和邻接结点的元素写入数组 V 和二维数组 VR 中, 调用 CreateGraph(G, V, VR) 创建新图。

5. 输出新图 name 的数据元素

4.3.15 多图管理：增加新图

1. 若增加后图总数大于 10, 输出 ERROR, 结束功能
2. 若 name 与已存在的图重名, 输出重名, 结束功能
3. 否则将 name 写入 Lists.elem[Lists.length-1].name
4. 输入图的数据元素, 储存在 V 和 VR 中, 调用 CreateGraph(G, V, VR) 创建新图
5. List.length++

4.3.18 多图管理：移除图

1. 调用 LocateList, 返回值为 i
2. 若 $i \neq 0$, 删除该图 (用后一个图覆盖前一个), 否则输出 ERROR, 结束功能

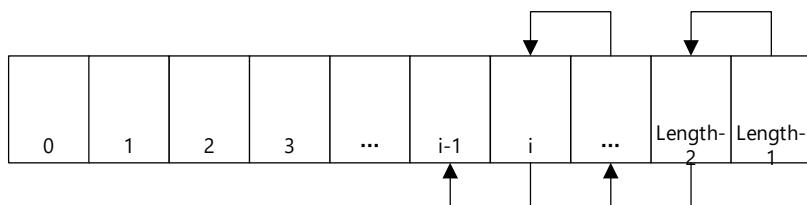


图 4-13 删除图示意图

3. List.length--

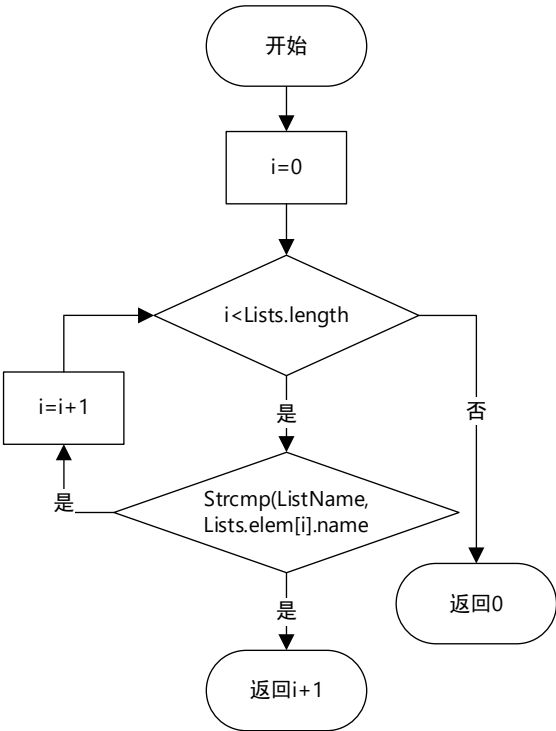


图 4-14 多图查找流程图

4.3.19 多图管理：查找图

- 1. 同上题遍历名字，查找对应图，返回 i
- 2. 若 i!=0，输出该图数据，否则输出 ERROR，结束功能

4.4 系统测试

4.4.1. 菜单管理

功能 1、2 仅为单图操作，功能 14~17 仅为多图操作。若功能标号的选择与单或多选择不匹配，将不实现功能。

Menu for Linear Table On Sequence Structure	
1. CreateCraph(单)	9. InsertArc
2. DestroyGraph(单)	10. DeleteArc
3. LocateVex	11. DFSTraverse
4. PutVex	12. BFSTraverse
5. FirstAdjVex	13. SaveGraph
6. NextAdjVex	14. LoadGraph(多)
7. InsertVex	15. AddList(多)
8. DeleteVex	16. RemoveList(多)
0. exit	17. LocateList(多)

单个图操作输入0，多个图操作输入1:

图 4-15 菜单

单个图操作输入0，多个图操作输入1:1

请选择你的操作[0~19]: 1
请选择单图操作

请选择你的操作[0~19]: 2
请选择单图操作

请选择你的操作[0~19]:

图 4-16 仅单图功能

单个图操作输入0，多个图操作输入1:0

请选择你的操作[0~19]: 14
请选择多图操作

请选择你的操作[0~19]: 15
请选择多图操作

请选择你的操作[0~19]: 16
请选择多图操作

请选择你的操作[0~19]: 17
请选择多图操作

请选择你的操作[0~19]: _

图 4-17 仅多图功能

4.4.2 测试结果及分析

功能编号	输出结果	输出分析	功能编号	输出结果	输出分析
1	OK	图创建成功	9	数字+字符串+数字串	成功后所有顶点和邻接点的数据
	ERROR	未正确分配内存		ERROR	插入失败
	INFEASIBLE	图已存在		INFEASIBLE	图不存在
2	OK	图销毁成功	10	数字+字符串+数字串	成功后所有顶点和邻接点的数据
	ERROR	未正确释放内存		ERROR	删除失败
	INFEASIBLE	图不存在		INFEASIBLE	图不存在
	数字+	查找的图数据		数字+	DFS 遍历的

3	字符串		11	字符串	顶点数据
	ERROR	查找失败		INFEASIBLE	图不存在
	INFEASIBLE	图不存在			
4	数字+	赋值后的	12	数字+	BFS 遍历的
	字符串	顶点数据		字符串	顶点数据
	ERROR	赋值失败		INFEASIBLE	图不存在
5	数字+	查找的第一	13	OK	读入文件成功
	字符串	临接点的数据		ERROR	打开文件失败
	ERROR	查找失败		INFEASIBLE	图不存在
6	数字+	查找的下一	14	数字+字符	根据文件创建
	字符串	临接点的数据		串+数字串	的图的数据
	ERROR	查找失败		ERROR	打开文件失败
7	数字+	成功后所有的	15	重名	重名
	字符串	顶点的数据		OK	图创建成功
	ERROR	插入失败		ERROR	未正确分配内存
8	数字+字符	成功后所有的	16	重名	二叉树重名
	串	顶点的数据		OVERFLOW	溢出
	ERROR	删除失败		数字+字符	删除后的
0	退出菜单		17	串+数字串	所有图名+数据
				删除失败	图不存在
				查找失败	图不存在

表 3-1 测试结果及分析

4.4.3. 测试用例

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

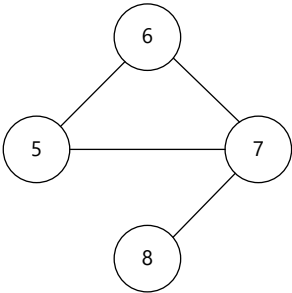


图 4-18 测试用例 1

1 G1 2 G2 3 G3 4 G4 5 G5 6 G6 7 G7 8 G8 9 G9 10 G10 11 G11 12 G12
-1 null
1 2 2 3 3 4 1 12 -1 -1

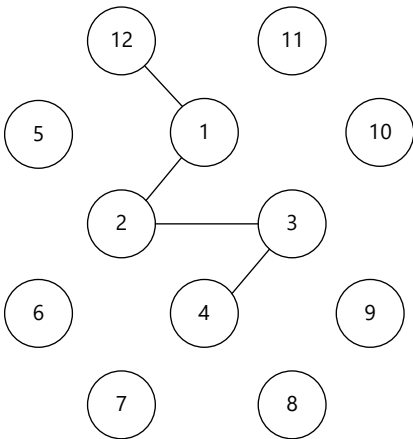


图 4-19 测试用例 2

4.4.4 测试截图

测试完用例后，实现 2 销毁功能，测试图不存在的情况。

1 创建

```

请选择你的操作[0~19]: 1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 7 无向图 -1 null
ERROR

请选择你的操作[0~19]: 1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 10 -1 -1
ERROR

请选择你的操作[0~19]: 1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK

请选择你的操作[0~19]: 1
INFEASIBLE
    
```

图 4-20 功能 1 的测试用例

2 销毁

```

请选择你的操作[0~19]: 1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 2
INFEASIBLE
    
```

图 4-21 功能 2 的测试用例

3 查找

```

请选择你的操作[0~19]: 1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK

请选择你的操作[0~19]: 3
请输入想查找的顶点关键字: 5
5 线性表

请选择你的操作[0~19]: 3
请输入想查找的顶点关键字: 4
ERROR

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 3
INFEASIBLE
    
```

图 4-22 功能 3 的测试用例

4 赋值

```

请选择你的操作[0~19]: 1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK

请选择你的操作[0~19]: 4
请输入想赋值的顶点关键字: 5
请输入顶点数值: 5 a
5 a 8 集合 7 二叉树 6 无向图

请选择你的操作[0~19]: 4
请输入想赋值的顶点关键字: 8
请输入顶点数值: 5 a
ERROR

请选择你的操作[0~19]: 4
请输入想赋值的顶点关键字: 6
请输入顶点数值: 9 有向图
5 a 8 集合 7 二叉树 9 有向图

请选择你的操作[0~19]: 4
请输入想赋值的顶点关键字: 6
请输入顶点数值: 9 有向图
ERROR

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 4
INFEASIBLE
    
```

图 4-23 功能 4 的测试用例

5 第一邻接顶点

```

请选择你的操作[0~19]: 1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK

请选择你的操作[0~19]: 5
请输入想查找的顶点关键字: 5
7 二叉树

请选择你的操作[0~19]: 5
请输入想查找的顶点关键字: 7
8 集合

请选择你的操作[0~19]: 5
请输入想查找的顶点关键字: 9
ERROR

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 5
INFEASIBLE
    
```

图 4-24 功能 5 的测试用例

6 下一邻接顶点

```

请选择你的操作[0~19]: 1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK

请选择你的操作[0~19]: 6
请输入想查找的顶点和邻接顶点关键字: 5 7
6 无向图

请选择你的操作[0~19]: 6
请输入想查找的顶点和邻接顶点关键字: 8 7
ERROR

请选择你的操作[0~19]: 6
请输入想查找的顶点和邻接顶点关键字: 5 8
ERROR

请选择你的操作[0~19]: 6
请输入想查找的顶点和邻接顶点关键字: 9 8
ERROR

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 6
INFEASIBLE
    
```

图 4-25 功能 6 的测试用例

7 插入顶点

```

请选择你的操作[0~19]: 1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK

请选择你的操作[0~19]: 7
请输入想插入的顶点的数据: 9 有向图
5 线性表 8 集合 7 二叉树 6 无向图 9 有向图

请选择你的操作[0~19]: 7
请输入想插入的顶点的数据: 7 有向图
ERROR

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 7
INFEASIBLE
    
```

图 4-26 功能 7 的测试用例

8 删除顶点

```

请选择你的操作[0~19]: 1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK

请选择你的操作[0~19]: 8
请输入想删除的顶点的键值: 6
5 线性表 2
8 集合 2
7 二叉树 1 0

请选择你的操作[0~19]: 8
请输入想删除的顶点的键值: 6
ERROR

请选择你的操作[0~19]: 8
请输入想删除的顶点的键值: 9
ERROR

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 8
INFEASIBLE

```

图 4-27 功能 8 的测试用例

9 插入弧

```

请选择你的操作[0~19]: 1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK

请选择你的操作[0~19]: 9
请输入想插入的弧: 5 8
5 线性表 1 2 3
8 集合 0 2
7 二叉树 1 3 0
6 无向图 2 0

请选择你的操作[0~19]: 9
请输入想插入的弧: 6 7
ERROR

请选择你的操作[0~19]: 9
请输入想插入的弧: 5 9
ERROR

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 9
INFEASIBLE

```

图 4-28 功能 9 的测试用例

10 删除弧

```

请选择你的操作[0~19]: 1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK

请选择你的操作[0~19]: 10
请输入想删除的弧: 6 7
5 线性表 2 3
8 集合 2
7 二叉树 1 0
6 无向图 0

请选择你的操作[0~19]: 10
请输入想删除的弧: 6 7
ERROR

请选择你的操作[0~19]: 10
请输入想删除的弧: 5 8
ERROR

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 10
INFEASIBLE
    
```

图 4-29 功能 10 的测试用例

11 DFS

```

请选择你的操作[0~19]: 1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK

请选择你的操作[0~19]: 11
5 线性表 7 二叉树 8 集合 6 无向图

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 11
INFEASIBLE
    
```

图 4-30 功能 11 的测试用例

12 BFS

```

请选择你的操作[0~19]: 1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK

请选择你的操作[0~19]: 12
5 线性表 7 二叉树 6 无向图 8 集合

请选择你的操作[0~19]: 2
OK

请选择你的操作[0~19]: 12
INFEASIBLE
    
```

图 4-31 功能 12 的测试用例

13 & 14 文件

```

请选择你的操作[0~19]: 15
输入要增加的图的个数:1
输入图名:1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK

请选择你的操作[0~19]: 13

输入操作的图的名字: 1
输入文件名:1.txt
OK

请选择你的操作[0~19]: 14
输入文件名: 1.txt
输入图名:1
重名
输入图名:2
5 线性表 2 3
8 集合 2
7 二叉树 1 3 0
6 无向图 2 0

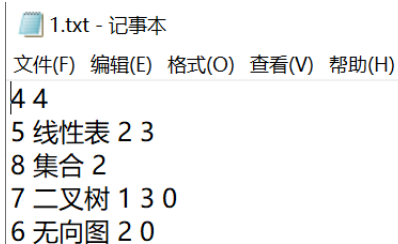
请选择你的操作[0~19]: 11

输入操作的图的名字: 2
5 线性表 7 二叉树 8 集合 6 无向图

请选择你的操作[0~19]: 12

输入操作的图的名字: 2
5 线性表 7 二叉树 6 无向图 8 集合
    
```

图 4-32 功能 13&14 的测试用例



```

1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
4 4
5 线性表 2 3
8 集合 2
7 二叉树 1 3 0
6 无向图 2 0
    
```

图 4-33 功能 13&14 的文件

15 新增图

```

请选择你的操作[0~19]: 15
输入要增加的图的个数:2
输入图名:1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK
输入图名:1
重名
输入图名:2
请输入顶点数据, 结束输入-1 null:
1 G1 2 G2 3 G3 4 G4 5 G5 6 G6 7 G7 8 G8 9 G9 10 G10 11 G11 12 G12 -1 null
请输入弧, 结束输入-1 -1: 1 2 2 3 3 4 1 12 -1 -1
OK

请选择你的操作[0~19]: 15
输入要增加的图的个数:9
OVERFLOW
    
```

图 4-34 功能 15 的测试用例

16 删除图

```

请选择你的操作[0~19]: 15
输入要增加的图的个数:3
输入图名:1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK
输入图名:2
请输入顶点数据, 结束输入-1 null:
1 G1 2 G2 3 G3 4 G4 5 G5 6 G6 7 G7 8 G8 9 G9 10 G10 11 G11 12 G12 -1 null
请输入弧, 结束输入-1 -1: 1 2 2 3 3 4 1 12 -1 -1
OK
输入图名:3
请输入顶点数据, 结束输入-1 null: 5 线性表 -1 null
请输入弧, 结束输入-1 -1: -1 -1
OK

请选择你的操作[0~19]: 16
输入要删除的图的名字:1
2
1 G1 11 1
2 G2 2 0
3 G3 3 1
4 G4 2
5 G5
6 G6
7 G7
8 G8
9 G9
10 G10
11 G11
12 G12 0
3
5 线性表

请选择你的操作[0~19]: 16
输入要删除的图的名字:4
删除失败
    
```

图 4-35 功能 16 的测试用例

17 查找图

```

请选择你的操作[0~19]: 15
输入要增加的图的个数:3
输入图名:1
请输入顶点数据, 结束输入-1 null: 5 线性表 8 集合 7 二叉树 6 无向图 -1 null
请输入弧, 结束输入-1 -1: 5 6 5 7 6 7 7 8 -1 -1
OK
输入图名:2
请输入顶点数据, 结束输入-1 null:
1 G1 2 G2 3 G3 4 G4 5 G5 6 G6 7 G7 8 G8 9 G9 10 G10 11 G11 12 G12 -1 null
请输入弧, 结束输入-1 -1: 1 2 2 3 3 4 1 12 -1 -1
OK
输入图名:3
请输入顶点数据, 结束输入-1 null: 5 线性表 -1 null
请输入弧, 结束输入-1 -1: -1 -1
OK

请选择你的操作[0~19]: 17
输入要查找的图的名字:1
1
5 线性表 2 3
8 集合 2
7 二叉树 1 3 0
6 无向图 2 0

请选择你的操作[0~19]: 17
输入要查找的图的名字:3
3
5 线性表

请选择你的操作[0~19]: 17
输入要查找的图的名字:4
查找失败
    
```

图 4-36 功能 17 的测试用例

0 退出

```

请选择你的操作[0~17]: 0

欢迎下次再使用本系统!

Process returned 0 (0x0)    execution time : 12.068 s
Press any key to continue.
    
```

图 4-37 功能 0

4.5 实验小结

1. 若图中只有一个顶点, 删除顶点功能会把图销毁, 所以当 $G.vexnum=1$ 时不能进行删除顶点操作。

2. 对图本身进行操作后不能忘记改 `G.vexnum` 和 `G.arcnum` 的值。
3. 多图操作后不能忘记改 `length` 的值。
4. 删除 `vertices` 中的邻接结点时，头结点和其他结点要分开操作。
5. DFS 和 BFS 遍历时，要考虑非联通图，遍历完一次后要比比较遍历顶点的总数与顶点总数是否相同，若不同，则该图是非联通图，需要寻找下一个未遍历过的顶点。
6. 写入文件时要注意空格怎么安排，读取文件时的空格和换行格式要和写入时一致。
7. 创建新图时要先遍历全表，检查是否有重名，否则查找和对图进行操作时会出错。
8. 要注意函数返回值是逻辑序号还是数组编号，逻辑序号=数组编号+1。

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

附录 A 基于顺序存储结构线性表实现的源程序

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define LIST_INIT_SIZE 100
#define LISTINCREMENT 10
typedef int status;
typedef int ElemType;
typedef struct{
    ElemType * elem;
    int length;
    int listsize;
}SqList;
typedef struct{
    struct { char name[30];
            SqList L;
        } elem[10];
    int length;
}LISTS;
status InitList(SqList& L);
status DestroyList(SqList& L);
status ClearList(SqList& L);
status ListEmpty(SqList L);
status ListLength(SqList L);
status GetElem(SqList L, int i, ElemType &e);
int LocateElem(SqList L, ElemType e);
status PriorElem(SqList L, ElemType e, ElemType &pre);
status NextElem(SqList L, ElemType e, ElemType &next);
status ListInsert(SqList &L, int i, ElemType e);
status ListDelete(SqList &L, int i, ElemType &e);
status ListTraverse(SqList L);
status AddList(LISTS &Lists, char ListName[]);
status RemoveList(LISTS &Lists, int i);
int LocateList(LISTS Lists, char ListName[]);
status Input(SqList &L);
status SaveList(SqList L, char FileName[]);
status LoadList(LISTS &Lists, char FileName[]);

int main() {
```

```

SqlList L;
L.elem=NULL;
LISTS Lists;
int n, i, j, e, pre, next, op, x, flag;
char FileName[30], name[30], Name[30];
Lists.length=0;
printf("      Menu for Linear Table On Sequence Structure \n");
printf("-----\n");
printf("      1. InitList(单)    9. NextElem\n");
printf("      2. DestroyList(单)10. ListInsert\n");
printf("      3. ClearList      11. ListDelete\n");
printf("      4. ListEmpty      12. ListTraverse\n");
printf("      5. ListLength     13. SaveList\n");
printf("      6. GetElem        14. LoadList(多)\n");
printf("      7. LocateElem     15. AddList(多)\n");
printf("      8. PriorElem      16. RemoveList(多)\n");
printf("      0. Exit           17. LocateList(多)\n");
printf("-----\n");
printf("      \n 单线性表操作输入 0, 多线性表操作输入 1:");
scanf("%d",&flag);          //flag 标记单或多操作
while(1) {
    printf("      \n 请选择你的操作[0~17]: ");
    scanf("%d",&op);          //选择的功能编号
    if(!flag&&op>13) {
        printf("请选择多线性表操作\n");
        continue;
    }
    if(flag&&op>0&&op<3) {
        printf("请选择单线性表操作\n");
        continue;
    }
    if(flag&&op<14&&op>0) {      //若对多线性表中的某个表进行操作
        printf("\n 输入操作的线性表的名字: ");
        scanf("%s", Name);
        x=LocateList(Lists, Name)-1;
        if(x>=0) L=Lists.elem[x].L;      //调用 LocateList 查找
        else {
            printf("INFEASIBLE\n");
            continue;
        }
    }
}
switch(op) {
    case 1:          //初始化
        if(L.elem) printf("INFEASIBLE\n");

```

```

else
    if(InitList(L)) {                                //创建线性表成功
        Input(L);                                    //输入线性表元素
        printf("OK\n");
    }
    else printf("ERROR\n");
break;
case 2:
    if(!L.elem) printf("INFEASIBLE\n");
    else{
        if(flag) j=DestroyList(Lists.elem[x].L);
        else j=DestroyList(L);
        if(j) printf("OK\n");
        else printf("ERROR\n");
    }
    break;
case 3:                                                //清空
    if(!L.elem) printf("INFEASIBLE\n");
    else{
        if(flag) j=ClearList(Lists.elem[x].L);
        else j=ClearList(L);
        if(j) printf("OK\n");
        else printf("ERROR\n");
    }
    break;
case 4:                                                //判空
    if(!L.elem) printf("INFEASIBLE\n");
    else
        if (ListEmpty(L)) printf("TRUE\n");
        else printf("FALSE\n");
    break;
case 5:                                                //求表长
    if(!L.elem) printf("INFEASIBLE\n");
    else printf("%d\n", ListLength(L));
    break;
case 6:                                                //获取
    if(!L.elem) printf("INFEASIBLE\n");
    else if(!L.length) printf("空线性表\n");
    else{
        printf("输入要获取的元素的位置:");
        scanf("%d", &i);
        if(GetElem(L, i, e)) printf("%d\n", e);
        else printf("ERROR\n");
    }
}

```

```

        break;
case 7:                                     //查找
    if(!L.elem) printf("INFEASIBLE\n");
    else if(!L.length) printf("空线性表\n");
    else{
        printf("输入要查找位置的元素:");
        scanf("%d",&e);
        if(LocateElem(L,e)) printf("%d\n",j);
        else printf("ERROR\n");
    }
    break;
case 8:                                     //前驱
    if(!L.elem) printf("INFEASIBLE\n");
    else if(!L.length) printf("空线性表\n");
    else{
        printf("输入要查找的元素:");
        scanf("%d",&e);
        if(PriorElem(L,e,pre)) printf("%d\n",pre);
        else printf("ERROR\n");
    }
    break;
case 9:                                     //后继
    if(!L.elem) printf("INFEASIBLE\n");
    else if(!L.length) printf("空线性表\n");
    else{
        printf("输入要查找的元素:");
        scanf("%d",&e);
        if(NextElem(L,e,next)) printf("%d\n",next);
        else printf("ERROR\n");
    }
    break;
case 10:                                    //插入
    if(!L.elem) printf("INFEASIBLE\n");
    else{
        printf("输入要插入的位置和元素:");
        scanf("%d%d",&i,&e);
        j=ListInsert(L,i,e);
        if(flag) Lists.elem[x].L.length++;
        else L.length++;
        if(j) ListTraverse(L);
        else printf("ERROR\n");
    }
    break;
case 11:                                    //删除

```

```

    if(!L.elem) printf("INFEASIBLE\n");
    else if(!L.length) printf("空线性表\n");
    else{
        printf("输入要删除元素的位置:");
        scanf("%d",&i);
        j=ListDelete(L,i,e);
        if(j){
            if(flag) Lists.elem[x].L.length--;
            else L.length--;
            ListTraverse(L);
        }else printf("ERROR\n");
    }
    break;
case 12:                                     //遍历
    if(!L.elem) printf("INFEASIBLE\n");
    else if(!L.length) printf("空线性表\n");
    else ListTraverse(L);
    break;
case 13:                                     //写入文件
    if(!L.elem) printf("INFEASIBLE\n");
    else{
        i=0;
        printf("输入文件名:");
        scanf("%s",FileName);
        if(SaveList(L,FileName))
            printf("OK\n");
        else printf("ERROR\n");
    }
    break;
case 14:                                     //读取文件
    printf("输入文件名:");
    scanf("%s",FileName);
    while(1){
        printf("输入线性表名:");
        scanf("%s",name);
        if(LocateList(Lists,name))
            printf("重名\n");           //重名,重新输入
        else break;
    }
    AddList(Lists,name);
    if(LoadList(Lists,FileName))
        ListTraverse(Lists.elem[Lists.length-1].L);
    else printf("ERROR\n");
    break;

```

```

case 15:                                     //添加新表
    printf("输入要增加的线性表的个数:");
    scanf("%d",&n);
    if(Lists.length+n>10){                  //表总数溢出
        printf("OVERFLOW\n");
        break;
    }
    while(n--){
        printf("输入线性表名字:");
        scanf("%s",name);
        if(LocateList(Lists,name)){        //判断是否有重名
            printf("重名\n");
            n++;
            continue;
        }
        else{
            if(AddList(Lists,name)){
                Input(Lists.elem[Lists.length-1].L);
                printf("OK\n");
            }else printf("ERROR\n");
        }
    }
    break;
case 16:                                     //删除表
    printf("输入要删除的线性表的名字:");
    scanf("%s",name);
    if(LocateList(Lists,name)){            //查找表
        if(RemoveList(Lists,LocateList(Lists,name)-1))
            for(n=0;n<Lists.length;n++){
                printf("%s ",Lists.elem[n].name);
                ListTraverse(Lists.elem[n].L);
            }
        }else printf("删除失败\n");
    break;
case 17:                                     //查找表
    printf("输入要查找的线性表的名字:");
    scanf("%s",name);
    if (n=LocateList(Lists,name)){
        printf("%s ",Lists.elem[n-1].name);
        ListTraverse(Lists.elem[n-1].L);
    }else printf("查找失败\n");
    break;
case 0:
    goto down;

```

```

    }
}
down:
printf("\n 欢迎下次再使用本系统! \n");
return 0;
}

status Input(SqlList &L) {
    int i;
    printf("输入元素, 结束输入 0:\n");
    scanf("%d",&i);
    while(i) {
        L.elem[L.length++]=i;
        scanf("%d",&i);
    }
}

status InitList(SqlList& L) {                                //初始化
    L.length=0;
    L.elem=(ElemType*)malloc(LIST_INIT_SIZE*sizeof(ElemType));
                                                                //分配内存
    L.listsize=LIST_INIT_SIZE;
    return OK;
}

status DestroyList(SqlList& L) {                             //销毁
    free(L.elem);                                             //释放内存
    L.length=0;
    L.elem=NULL;
    return OK;
}

status ClearList(SqlList& L) {                                //清空
    L.length=0;
    return OK;
}

status ListEmpty(SqlList L) {                                  //判空
    if(L.length) return FALSE;
    else return TRUE;
}

status ListLength(SqlList L) {                                  //求表长
    return L.length;
}

```

```

}

status GetElem(SqList L, int i, ElemType &e) {           //获取
    if(i<1||i>L.length) return ERROR;                 //位置不合法
    else{
        e=L.elem[i-1];
        return OK;
    }
}

int LocateElem(SqList L, ElemType e) {                 //查找
    int i;
    for(i=0;i<L.length;i++)
        if(L.elem[i]==e) return i+1;
    return 0;
}

status PriorElem(SqList L, ElemType e, ElemType &pre) {
    int i=LocateElem(L, e);                           //调用 LocateElem
    if(i>1) {                                           //判断 i 是否合法
        pre=L.elem[i-2];
        return OK;
    }
    else return ERROR;
}

status NextElem(SqList L, ElemType e, ElemType &next) {
    int i=LocateElem(L, e);                           //调用 LocateElem
    if(0<i<L.length) {                                //判断 i 是否合法
        next=L.elem[i];
        return OK;
    }
    else return ERROR;
}

status ListInsert(SqList &L, int i, ElemType e) {      //插入
    int j;
    if(i>L.length+1||i<1) return ERROR;              //位置不合法
    else if(!L.length) {
        L.elem[0]=e;
        return OK;
    }
    else{
        if(L.length+1>LIST_INIT_SIZE) { //如果插入后溢出，重新分配

```



```

        ElemType
    *newelem=(ElemType*)realloc (L. elem, sizeof (ElemType)*LIST_INIT_SIZE+LI
    STINCREMENT);
        L. elem=newelem;
    }
    for(j=L. length-1;j>=i-1;j--)
        L. elem[j+1]=L. elem[j];
    L. elem[i-1]=e;
    return OK;
}
}

status ListDelete(SqList &L, int i, ElemType &e) {           //删除
    int j;
    if(i<1||i>L. length) return ERROR;                       //位置不合法
    else{
        e=L. elem[i-1];
        for(j=i-1;j<L. length-1;j++)
            L. elem[j]=L. elem[j+1];
        return OK;
    }
}

status ListTraverse(SqList L) {                               //遍历
    int i;
    for(i=0;i<L. length;i++) {
        printf("%d", L. elem[i]);
        if(i!=L. length-1) printf(" ");
        else printf("\n");
    }
    return OK;
}

status SaveList(SqList L, char FileName[]) {                 //写入文件
    FILE *fp;
    int i=0;
    if(!(fp=fopen(FileName, "wb"))) return ERROR;           //打开文件失败
    else{
        while(i<L. length) {
            fprintf(fp, "%d ", L. elem[i]);                  //线性表写入文件
            i++;
        }
        fclose(fp);
        return OK;
    }
}

```

```

    }
}

status LoadList(LISTS &Lists, char FileName[]) {    //读出文件
    FILE *fp;
    int i=0;
    if(!(fp=fopen(FileName, "rb"))) return ERROR;    //打开文件失败
    else{
        while((fscanf(fp, "%d", &Lists.elem[Lists.length-1].L.elem[i]))!=EOF)    //读入线性表中
            if(!feof(fp)) i++;
        Lists.elem[Lists.length-1].L.length=i+1;    //表总数加一
        fclose(fp);
        return OK;
    }
}

status AddList(LISTS &Lists, char ListName[]) {    //增加新表
    strcpy(Lists.elem[Lists.length].name, ListName);

    Lists.elem[Lists.length].L.elem=(ElemType*)malloc(LIST_INIT_SIZE*sizeof(ElemType));
    Lists.elem[Lists.length].L.length=0;    //长度为 0
    Lists.length++;
    return OK;    //表总数加一
}

status RemoveList(LISTS &Lists, int i) {    //删除表
    free(Lists.elem[i].L.elem);    //释放内存
    for(int j=i; j<Lists.length-1; j++)
        Lists.elem[j]=Lists.elem[j+1];    //前一个覆盖
    Lists.length--;
    return OK;
}

int LocateList(LISTS Lists, char ListName[]) {    //查找表
    for(int i=0; i<Lists.length; i++)
        if(!strcmp(ListName, Lists.elem[i].name)) //名字相同
            return i+1;
    return 0;
}

```

附录 B 基于链式存储结构线性表实现的源程序

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
typedef int status;
typedef int ElemType;
typedef struct LNode{
    ElemType data;
    struct LNode *next;
}LNode,*LinkList;
typedef struct{
    struct { char name[30];
            LinkList L;
    }elem[10];
    int length;
}LISTS;
status InitList(LinkList& L);
status DestroyList(LinkList& L);
status ClearList(LinkList& L);
status ListEmpty(LinkList L);
status ListLength(LinkList L);
status GetElem(LinkList L, int i, ElemType &e);
int LocateElem(LinkList L, ElemType e);
status PriorElem(LinkList L, ElemType e, ElemType &pre);
status NextElem(LinkList L, ElemType e, ElemType &next);
status ListInsert(LinkList &L, int i, ElemType e);
status ListDelete(LinkList &L, int i, ElemType &e);
status ListTraverse(LinkList L);
status SaveList(LinkList L, char FileName[]);
status LoadList(LISTS &Lists, char FileName[]);
status AddList(LISTS &Lists, char ListName[]);
status RemoveList(LISTS &Lists, int i);
int LocateList(LISTS Lists, char ListName[]);
status Input(LinkList &L);

int main() {
    int op, i, j, e, pre, next, flag, x, n;
    LinkList L=NULL;
    LISTS Lists;
```

```

Lists.length=0;
char FileName[30], name[30], Name[30];
printf("      Menu for Linear Table On Sequence Structure \n");
printf("-----\n");
printf("      1. InitList(单)    9.  NextElem\n");
printf("      2. DestroyList(单)10. ListInsert\n");
printf("      3. ClearList      11. ListDelete\n");
printf("      4. ListEmpty      12. ListTraverse\n");
printf("      5. ListLength     13. SaveList\n");
printf("      6. GetElem        14. LoadList(多)\n");
printf("      7. LocateElem     15. AddList(多)\n");
printf("      8. PriorElem      16. RemoveList(多)\n");
printf("      0. Exit           17. LocateList(多)\n");
printf("-----\n");
printf("      单线性表操作输入 0，多线性表操作输入 1:");
scanf("%d",&flag);
while(1){
    printf("      \n 请选择你的操作[0~17]:");
    scanf("%d",&op);
    if(!flag&&op>13){
        printf("请选择多线性表操作\n");
        continue;
    }
    if(flag&&op>0&&op<3){
        printf("请选择单线性表操作\n");
        continue;
    }
    if(flag&&op<14&&op>0){
        printf("\n 输入操作的线性表的名字: ");
        scanf("%s",Name);
        x=LocateList(Lists,Name)-1;
        if(x>=0) L=Lists.elem[x].L;
        else{
            printf("INFEASIBLE\n");
            continue;
        }
    }
    switch(op){
        case 1:
            if(L) printf("INFEASIBLE\n");
            else
                if(InitList(L)){
                    Input(L);
                    printf("OK\n");
                }
    }
}

```

```

    }else printf("ERROR\n");
    break;
case 2:
    if(!L) printf("INFEASIBLE\n");
    else{
        if(flag) j=DestroyList(Lists.elem[x].L);
        else j=DestroyList(L);
        if(j) printf("OK\n");
        else printf("ERROR\n");
    }
    break;
case 3:                                     //清空
    if(!L) printf("INFEASIBLE\n");
    else
        if(ClearList(L)&&L&&!L->next) printf("OK\n");
        else printf("ERROR\n");
    break;
case 4:                                     //判空
    if(!L) printf("INFEASIBLE\n");
    else
        if(ListEmpty(L)) printf("TRUE\n");
        else printf("FALSE\n");
    break;
case 5:                                     //求表长
    if(!L) printf("INFEASIBLE\n");
    else printf("%d\n",ListLength(L));
    break;
case 6:                                     //获取
    if(!L) printf("INFEASIBLE\n");
    else if(!L->next) printf("空线性表\n");
    else{
        printf("输入要获取的元素的位置: ");
        scanf("%d",&i);
        if(GetElem(L,i,e)) printf("%d\n",e);
        else printf("ERROR\n");
    }
    break;
case 7:                                     //查找
    if(!L) printf("INFEASIBLE\n");
    else if(!L->next) printf("空线性表\n");
    else{
        printf("输入要获取位置的元素: ");
        scanf("%d",&e);
        if(LocateElem(L,e)) printf("%d\n",j);
    }

```

```

        else printf("ERROR");
    }
    break;
case 8:                                     //前驱
    if(!L) printf("INFEASIBLE\n");
    else if(!L->next) printf("空线性表\n");
    else{
        printf("输入要查找的元素: ");
        scanf("%d",&e);
        if(PriorElem(L,e,pre)) printf("%d\n",pre);
        else printf("ERROR\n");
    }
    break;
case 9:                                     //后继
    if(!L) printf("INFEASIBLE\n");
    else if(!L->next) printf("空线性表\n");
    else{
        printf("输入要查找的元素: ");
        scanf("%d",&e);
        if(NextElem(L,e,next)) printf("%d\n",next);
        else printf("ERROR\n");
    }
    break;
case 10:                                    //插入
    if(!L) printf("INFEASIBLE\n");
    else{
        printf("输入要插入的位置和元素: ");
        scanf("%d%d",&i,&e);
        if(ListInsert(L,i,e)) ListTraverse(L);
        else printf("ERROR");
    }
    break;
case 11:                                    //删除
    if(!L) printf("INFEASIBLE\n");
    else if(!L->next) printf("空线性表\n");
    else{
        printf("输入要删除的元素的位置: ");
        scanf("%d",&i);
        if(ListDelete(L,i,e)) ListTraverse(L);
        else printf("ERROR\n");
    }
    break;
case 12:                                    //遍历

```

```

        if(!L) printf("INFEASIBLE\n");
        else if(!L->next) printf("空线性表\n");
        else ListTraverse(L);
        break;
case 13:                                     //写入文件
        if(!L) printf("INFEASIBLE\n");
        else if(!L->next) printf("空线性表\n");
        else{
            printf("输入文件名: ");
            scanf("%s",FileName);
            if(SaveList(L,FileName))
                printf("OK\n");
            else printf("ERROR\n");
        }
        break;
case 14:
        printf("输入文件名:");
        scanf("%s",FileName);
        while(1){
            printf("输入线性表名:");
            scanf("%s",name);
            if(LocateList(Lists,name))
                printf("重名\n");           //重名,重新输入
            else break;
        }
        AddList(Lists,name);
        if(LoadList(Lists,FileName))
            ListTraverse(Lists.elem[Lists.length-1].L);
        else printf("ERROR\n");
        break;
case 15:                                     //增加新表
        printf("输入要增加的线性表的个数:");
        scanf("%d",&n);
        if(Lists.length+n>10){               //溢出
            printf("OVERFLOW\n");
            break;
        }
        while(n--){
            printf("输入线性表名字:");
            scanf("%s",name);
            if(LocateList(Lists,name)){       //判断是否有重名
                printf("重名\n");
                n++;
                continue;
            }
        }
    
```

```

        }
        else
            if(AddList(Lists, name))
                Input(Lists.elem[Lists.length-1].L);
            else printf("ERROR\n");
    }
    break;
case 16:                                     //删除表
    printf("输入要删除的线性表的名字:");
    scanf("%s", name);
    if(LocateList(Lists, name))             //查找表
        if(RemoveList(Lists, LocateList(Lists, name)-1))
            for(n=0; n<Lists.length; n++) {
                printf("%s ", Lists.elem[n].name);
                ListTraverse(Lists.elem[n].L);
            }
        else printf("删除失败\n");
    break;
case 17:                                     //查找表
    printf("输入要查找的线性表的名字:");
    scanf("%s", name);
    if (n=LocateList(Lists, name)) {
        printf("%s ", Lists.elem[n-1].name);
        ListTraverse(Lists.elem[n-1].L);
    }
    else printf("查找失败\n");
    break;
case 0:
    printf("\n 欢迎下次再使用本系统! \n");
    goto down;
}
}
down:
return 0;
}

status Input(LinkList &L) {                 //输入
    int i;
    LNode *s, *r=L;
    printf("输入元素, 结束输入 0: \n");
    scanf("%d", &i);
    while(i) {
        s=(LNode*) malloc(sizeof(LNode)); //新建结点
        s->data=i;                         //结点赋值
    }
}

```



```

        r->next=s;
        r=s;
        scanf("%d",&i);
    }
    r->next=NULL;
}

status InitList(LinkList &L) {                                //创建
    L=(struct LNode*)malloc(sizeof(ElemType));                //生成头结点
    L->next=NULL;
    return OK;
}

status DestroyList(LinkList &L) {                              //销毁
    struct LNode *p=L->next;
    while(p) {
        L->next=p->next;                                       //依次销毁头结点后的结点
        p=NULL;
        p=L->next;                                             //p 指向头结点下一结点
    }
    free(L);                                                    //释放内存
    L=NULL;                                                      //销毁表头
    return OK;
}

status ClearList(LinkList &L) {                                //清空
    struct LNode *p=L->next;
    while(p) {                                                  //依次销毁头结点后的结点
        L->next=p->next;
        free(p);                                              //p 指向头结点下一结点
        p=L->next;                                             //释放内存
    }
    return OK;
}

status ListEmpty(LinkList L) {                                  //判空
    if(!L->next) return TRUE;
    else return FALSE;
}

int ListLength(LinkList L) {                                    //求表长
    int cnt=0;
    struct LNode *p=L;
    while(p->next) {

```

```

        cnt++;
        p=p->next;
    }
    return cnt;
}

status GetElem(LinkList L, int i, ElemType &e) {
    if(i<1||i>ListLength(L)) return ERROR;
    struct LNode *p=L;
    while(i--) p=p->next;
    e=p->data;
    return OK;
}

status LocateElem(LinkList L, ElemType e) {
    int i=1;
    struct LNode *p=L->next;
    while(p) {
        if(e==p->data) return i;
        p=p->next;
        i++;
    }
    return ERROR;
}

status PriorElem(LinkList L, ElemType e, ElemType &pre) { //前驱
    struct LNode *p=L;
    int i=LocateElem(L, e)-1;
    if(!i||i==0) return ERROR;
    while(i--) p=p->next;
    pre=p->data;
    return OK;
}

status NextElem(LinkList L, ElemType e, ElemType &next) { //后继
    struct LNode *p=L;
    int i=LocateElem(L, e);
    if(!i||i==ListLength(L)) return ERROR;
    while(i++) p=p->next;
    next=p->next->data;
    return OK;
}

status ListInsert(LinkList &L, int i, ElemType e) { //插入

```

```

if(i<1||i>ListLength(L)+1) return ERROR;           //i 不合法
    i--;
    struct LNode *n=(struct LNode*)malloc(sizeof(ElemType)),
    *p=L->next,*q=L;
    while(i--){                                     //查找结点
        p=p->next;
        q=q->next;
    }
    q->next=n;
    n->data=e;                                       //结点赋值
    n->next=p;
    return OK;
}

status ListDelete(LinkList &L,int i,ElemType &e){    //删除
    if(i<1||i>ListLength(L)) return ERROR;         //i 不合法
    i--;
    struct LNode *p=L->next,*q=L;
    while(i--){
        p=p->next;
        q=q->next;
    }
    e=p->data;                                       //结点赋值
    q->next=p->next;                                 //删除结点
    free(p);
    return OK;
}

status ListTraverse(LinkList L){                    //遍历
    struct LNode *p=L->next;
    while(p){
        printf("%d",p->data);
        if(p->next) printf(" ");
        else printf("\n");
        p=p->next;
    }
    return OK;
}

status SaveList(LinkList L,char FileName[]){        //写入文件
    struct LNode *p=L->next;
    FILE *fp;
    if(!(fp=fopen(FileName,"wb"))) return ERROR;    //打开文件失败
    else{

```

```

        while(p) {
            fprintf(fp, "%d ", p->data);           //线性表写入文件
            p=p->next;
        }
        fclose(fp);
        return OK;
    }
}

status LoadList(LISTS &Lists, char FileName[]) {
    FILE *fp;
    struct LNode *p=Lists.elem[Lists.length-1].L;
    if((fp=fopen(FileName, "rb"))){
        Lists.elem[Lists.length-
1].L->next=(LinkedList)malloc(sizeof(LNode));           //生产新结点
        p=Lists.elem[Lists.length-1].L->next;
        p->next=NULL;
        while((fscanf(fp, "%d ", &p->data))!=EOF)           //读入线性表
            if(!feof(fp)) {
                p->next=(LinkedList)malloc(sizeof(LNode)); //生成新结点
                p=p->next;
            }
        p->next=NULL;
        fclose(fp);
        return OK;
    }else return ERROR;
}

status AddList(LISTS &Lists, char ListName[]) {           //创建新表
    strcpy(Lists.elem[Lists.length].name, ListName);
    Lists.elem[Lists.length].L=(LNode*)malloc(sizeof(LNode*));
    Lists.elem[Lists.length].L->next=NULL;
    Lists.length++;
    return OK;
}

status RemoveList(LISTS &Lists, int i) {           //删除表
    DestroyList(Lists.elem[i].L);
    for(int j=i; j<Lists.length-1; j++)
        Lists.elem[j]=Lists.elem[j+1];           //覆盖前一个
    Lists.length--;
    return OK;
}

```

```
int LocateList(LISTS Lists, char ListName[]) {           //查找表
    for(int i=0; i<Lists.length; i++)
        if(!strcmp(ListName, Lists.elem[i].name))
            return i+1;
    return 0;
}
```

附录 C 基于二叉链表二叉树实现的源程序

```
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
typedef int status;
typedef int KeyType;
typedef struct {
    KeyType key;
    char others[20];
} TElemType;
typedef struct BiTNode{
    TElemType data;
    struct BiTNode *lchild,*rchild;
}BiTNode, *BiTree;
typedef struct {
    int pos;
    TElemType data;
}DEF;
typedef struct{
    struct {
        char name[30];
        BiTree T;
    }elem[10];
    int length;
}LISTS;
void visit(BiTree T);
void Traverse(BiTree T);
status CreateBiTree(BiTree &T, DEF definition[]);
status DestroyBiTree(BiTree &T);
status ClearBiTree(BiTree &T);
status BiTreeEmpty(BiTree T);
int BiTreeDepth(BiTree T);
```

```

BiTNode* LocateNode(BiTree T,KeyType e);
status Assign(BiTree &T,KeyType e,TElemType value);
BiTNode* GetSibling(BiTree T,KeyType e);
status InsertNode(BiTree &T,KeyType e,int LR,TElemType c);
status DeleteNode(BiTree &T,KeyType e);
void PreOrderTraverse(BiTree T);
void InOrderTraverse(BiTree T);
void PostOrderTraverse(BiTree T);
void LevelOrderTraverse(BiTree T);
status SaveBiTree(BiTree T, char FileName[]);
status LoadBiTree(BiTree &T, char FileName[]);
status SaveBiTree(BiTree T, char FileName[]);
status LoadBiTree(BiTree &T, char FileName[]);
status AddList(LISTS &Lists,char ListName[]);
status RemoveList(LISTS &Lists,int i);
int LocateList(LISTS Lists,char ListName[]);
BiTNode* q=NULL;                //外部变量, 在某些函数中作返回值
int flag0;                      //外部变量, 在某些函数中作判断标识

int main() {
    BiTree T=NULL,p;
    LISTS Lists;
    TElemType c;
    BiTree s[100];
    DEF *definition;
    int op,flag,e,i=0,j,LR,x,n;
    Lists.length=0;
    char FileName[30],name[30],Name[30];
    printf("      Menu for Linear Table On Sequence Structure \n");
    printf("-----\n");
    printf("      1. CreateBiTree(单) 10. DeleteNode\n");
    printf("      2. DestroyBiTree(单) 11. PreOrderTraverse\n");
    printf("      3. ClearBiTree      12. InOrderTraverse\n");
    printf("      4. BiTreeEmpty      13. PostOrderTraverse\n");
    printf("      5. BiTreeDepth      14. LevelOrderTraverse\n");
    printf("      6. LocateNode       15. SaveBiTree\n");
    printf("      7. Assign           16. LoadBiTree(多)\n");
    printf("      8. GetSibling       17. AddList(多)\n");
    printf("      9. InsertNode       18. RemoveList(多)\n");
    printf("      0. exit             19. LocateList(多)\n");
    printf("-----\n");
    printf("      \n 单二叉树操作输入 0, 多二叉树操作输入 1:");
    scanf("%d",&flag);
    while(1) {

```

```

printf("\n\n 请选择你的操作[0~19]: ");
scanf("%d", &op);
if(!flag&&op>15) {
    printf("请选择多二叉树操作\n");
    continue;
}
if(flag&&op>0&&op<3) {
    printf("请选择单二叉树操作\n");
    continue;
}
if(flag&&op<16&&op>0) {
    printf("\n 输入操作的二叉树的名字: ");
    scanf("%s", Name);
    x=LocateList(Lists, Name)-1;
    if(x>=0) T=Lists.elem[x].T;
    else{
        printf("INFEASIBLE");
        continue;
    }
}
switch(op) {
    case 1:
        if(T) printf("INFEASIBLE");
        else{
            definition=(DEF*)malloc(100*sizeof(DEF));
            printf("请输入二叉树元素, 结束输入 0 0 null: ");
            do{

scanf("%d%d%s", &definition[i].pos, &definition[i].data.key, definition[
i].data.others);                                     //输入位序和结点数据
                }while(definition[i++].pos);
                for(i=0;definition[i+1].pos;i++) {
                    for(j=i+1;definition[j].pos;j++)

if(definition[i].data.key==definition[j].data.key) {
                        j=0;
                        break;
                    }
                }
                if(!j) break;
            }                                     //判断关键字是否唯一
            if(!j) printf("ERROR");
            else{
                T=(BiTree)malloc(sizeof(BiTNode));
                T->rchild=NULL;

```

```

        p=T;
        if(CreateBiTree(p->lchild,definition))
            printf("OK");
        else printf("ERROR");
        free(definition);
    }
}
break;
case 2:
    if(!T) printf("INFEASIBLE");
    else{
        if(flag) j=DestroyBiTree(Lists.elem[x].T);
        else j=DestroyBiTree(T);           //不保留 root
        if(j) printf("OK");
        else printf("ERROR");
    }
    break;
case 3:                                     //清空
    if(!T) printf("INFEASIBLE");
    else{
        if(flag) p=Lists.elem[x].T->lchild; //保留 root
        else p=T->lchild;
        if(ClearBiTree(p)){
            printf("OK");
            T->lchild=NULL;
        }
        else printf("ERROR");
    }
    break;
case 4:                                     //判空
    if(!T) printf("INFEASIBLE");
    else{
        if(BiTreeEmpty(T)) printf("TRUE");
        else printf("FALSE");
    }
    break;
case 5:                                     //求深度
    if(!T) printf("INFEASIBLE");
    else if(!T->lchild) printf("0");         //空二叉树
    else printf("%d",BiTreeDepth(T->lchild));
    break;
case 6:                                     //查找
    if(!T) printf("INFEASIBLE");
    else if(!T->lchild) printf("空二叉树");

```



```

else{
    printf("输入想要查找的节点的关键值: ");
    scanf("%d",&e);
    q=NULL;
    if(p=LocateNode(T->lchild,e))
        visit(p);
    else printf("ERROR");
}
break;
case 7:                                     //赋值
    if(!T) printf("INFEASIBLE");
    else if(!T->lchild) printf("空二叉树");
    else{
        if(flag) p=Lists.elem[x].T;
        else p=T;
        printf("输入想要查找的节点的关键值: ");
        scanf("%d",&e);
        printf("输入想要替换的结点值: ");
        scanf("%d%s",&c.key,c.others);
        if(Assign(p->lchild,e,c))
            Traverse(T->lchild);
        else printf("ERROR");
    }
    break;
case 8:                                     //获得兄弟
    if(!T) printf("INFEASIBLE");
    else if(!T->lchild) printf("空二叉树");
    else{
        printf("输入想要查找的节点的关键值: ");
        scanf("%d",&e);
        q=NULL;
        if(p=GetSibling(T->lchild,e))
            visit(p);
        else printf("ERROR");
    }
    break;
case 9:                                     //插入
    if(!T) printf("INFEASIBLE");
    else{
        if(flag) p=Lists.elem[x].T;
        else p=T;
        if(p->lchild){                       //非空二叉树
            printf("输入想要插入的节点的关键值: ");
            scanf("%d",&e);

```

```

        printf("L:0,R:1,根节点:-1 :");
        scanf("%d",&LR);
        printf("输入想要插入的结点值: ");
        scanf("%d%s",&c.key,c.others);
        q=NULL;
        if(LocateNode(p->lchild,c.key))
            printf("ERROR");
        else if(InsertNode(p,e,LR,c))
            Traverse(T->lchild);
        else printf("ERROR");
    }else{ //空二叉树
        printf("输入想要插入的结点值: ");
        scanf("%d%s",&c.key,c.others);
        BiTree
newnode=(BiTree)malloc(sizeof(BiTNode)); //生成新结点
        newnode->data.key=c.key; //将根节点赋值
        strcpy(newnode->data.others,c.others);
        newnode->lchild=NULL;
        newnode->rchild=NULL;
        p->lchild=newnode;
        Traverse(T->lchild);
    }
}
break;
case 10: //删除
    if(!T) printf("INFEASIBLE");
    else if(!T->lchild) printf("空二叉树");
    else{
        if(flag) p=Lists.elem[x].T;
        else p=T;
        printf("输入想要删除的节点的关键值: ");
        scanf("%d",&e);
        q=NULL;
        flag0=0;
        if(!LocateNode(T,e)) printf("ERROR");
        else if(DeleteNode(p->lchild,e))
            Traverse(T->lchild);
    }
    break;
case 11: //先序遍历
    if(!T) printf("INFEASIBLE");
    else if(!T->lchild) printf("空二叉树");
    else PreOrderTraverse(T->lchild);
    break;

```

```

case 12:                                     //中序遍历
    if(!T) printf("INFEASIBLE");
    else if(!T->lchild) printf("空二叉树");
    else InOrderTraverse(T->lchild);
    break;
case 13:                                     //后序遍历
    if(!T) printf("INFEASIBLE");
    else if(!T->lchild) printf("空二叉树");
    else PostOrderTraverse(T->lchild);
    break;
case 14:                                     //按层遍历
    if(!T) printf("INFEASIBLE");
    else if(!T->lchild) printf("空二叉树");
    else LevelOrderTraverse(T->lchild);
    break;
case 15:                                     //写入文件
    if(!T) printf("INFEASIBLE");
    else if(!T->lchild) printf("空二叉树");
    else{
        printf("输入文件名:");
        scanf("%s", FileName);
        SaveBiTree(T->lchild, FileName);
        printf("OK");
    }
    break;
case 16:
    printf("输入文件名: ");
    scanf("%s", FileName);
    while(1){
        printf("输入二叉树名:");
        scanf("%s", name);
        if(LocateList(Lists, name))
            printf("重名\n");          //重名, 重新输入
        else break;
    }
    AddList(Lists, name);
    p=Lists.elem[Lists.length-1].T;
    if(LoadBiTree(p, FileName))
        Traverse(Lists.elem[Lists.length-1].T->lchild);
    else printf("ERROR\n");
    break;
case 17:                                     //增加树
    printf("输入要增加的二叉树的个数:");
    scanf("%d", &n);

```

```

        if(Lists.length+n>10){
            printf("OVERFLOW\n");
            break;
        }
        definition=(DEF*)malloc(100*sizeof(DEF));
        while(n--){
            printf("输入二叉树名字:");
            scanf("%s",name);
            if(LocateList(Lists,name)){ //判断是否有重名
                printf("重名\n");
                n++;
                continue;
            }
            else
                if(AddList(Lists,name)){
                    i=0;
                    printf("请输入二叉树元素,结束输入 0 0 null:");
                    do{

scanf("%d%d%s",&definition[i].pos,&definition[i].data.key,definition[
i].data.others); //输入位序和结点数据
                    }while(definition[i++].pos);
                    i=0;
                    while(j=definition[i].pos){
                        s[j]=(BiTNode
*)malloc(sizeof(BiTNode));

                        s[j]->data=definition[i].data;
                        s[j]->lchild=NULL;
                        s[j]->rchild=NULL;
                        if(j!=1){
                            if(j%2) s[j/2]->rchild=s[j];
                            else s[j/2]->lchild=s[j];
                        }
                        i++;
                    }
                    Lists.elem[Lists.length-1].T->lchild=s[1];
                    printf("OK\n");
                    for(i=0;i<100;i++)
                        definition[i].pos=0;
                    for(i=0;i<100;i++)
                        s[i]=NULL;
                }else printf("ERROR\n");
            }
        }
    }

```

```

        break;
    case 18:                                     //删除树
        printf("输入要删除的二叉树的名字:");
        scanf("%s", name);
        if(LocateList(Lists, name)) {
            if(RemoveList(Lists, LocateList(Lists, name)-1))
                for(n=0; n<Lists.length; n++) {
                    printf("%s\n", Lists.elem[n].name);
                    if(!Lists.elem[n].T->lchild) printf("空二
叉树表\n");

                    else{
                        Traverse(Lists.elem[n].T->lchild);
                        printf("\n");
                    }
                }
            else printf("删除失败");
            break;
        case 19:                                     //查找树
            printf("输入要查找的二叉树的名字:");
            scanf("%s", name);
            if(n=LocateList(Lists, name)) {
                if(!Lists.elem[n-1].T) printf("INFEASIBLE");
                else{
                    printf("%s\n", Lists.elem[n-1].name);
                    if(!Lists.elem[n-1].T->lchild)
                        printf("空线性表\n");
                    else Traverse(Lists.elem[n-1].T->lchild);
                }
            }
            else printf("查找失败");
            break;
        case 0: goto down;
    }
}

down:
printf("\n 欢迎下次再使用本系统! \n");
return 0;
}

void Traverse(BiTree T) {
    PreOrderTraverse(T);
    printf("\n");
    InOrderTraverse(T);
}

```

```

}

status CreateBiTree(BiTree &T, DEF definition[]) {    //创建
    int i=0, j;
    BiTNode *p[100];                                //按位序储存结点
    while(j=definition[i].pos) {
        p[j]=(BiTree)malloc(sizeof(BiTNode));
        p[j]->data=definition[i].data;
        p[j]->lchild=NULL;
        p[j]->rchild=NULL;
        if(j!=1) {
            if(j%2) p[j/2]->rchild=p[j];            //按位序分配指针域
            else p[j/2]->lchild=p[j];
        }
        i++;
    }
    T=p[1];
    return OK;
}

status DestroyBiTree(BiTree &T) {                    //销毁
    if(T) {
        DestroyBiTree(T->lchild);                    //销毁左孩子
        DestroyBiTree(T->rchild);                    //销毁右孩子
        free(T);
        T=NULL;
    }
    return OK;
}

status ClearBiTree(BiTree &T) {                      //清空
    if(T) {
        ClearBiTree(T->lchild);
        ClearBiTree(T->rchild);
        free(T);
        T=NULL;
    }
    return OK;
}

status BiTreeEmpty(BiTree T) {                      //判空
    if(T->lchild) return FALSE;
    else return TRUE;
}

```

```

int BiTreeDepth(BiTree T) {                                //求深度
    if(!T) return 0;
    int a=BiTreeDepth(T->lchild);
    int b=BiTreeDepth(T->rchild);
    return (a>b)?(a+1):(b+1);                               //返回左右子树中更深的
}

BiTNode* LocateNode(BiTree T,KeyType e) {                  //查找
    if(T) {
        if(q) return q;
        if(e==T->data.key) {
            q=T;
            return q;
        }else{
            LocateNode(T->lchild,e);
            LocateNode(T->rchild,e);
        }
        return q;
    }
}

status Assign(BiTree &T,KeyType e,TElemType value) { //赋值
    if(value.key!=e) {
        q=NULL;
        if(LocateNode(T,value.key))                       //关键字不唯一
            return ERROR;
    }
    q=NULL;
    BiTree p=LocateNode(T,e);                               //调用 LocateNode
    if(p) {
        p->data.key=value.key;                               //结点赋值
        strcpy(p->data.others,value.others);
        return OK;
    }
    else return ERROR;
}

BiTNode* GetSibling(BiTree T,KeyType e) {                  //获得兄弟
    if(T->lchild&&T->rchild) {
        if(q) return q;
        if(T->lchild->data.key==e) {                         //等于左孩子
            q=T->rchild;
            return q;                                       //返回右孩子
        }
    }
}

```

```

        }else if(T->rchild->data.key==e) {           //等于右孩子
            q=T->lchild;
            return q;                                //返回左孩子
        }else{
            GetSibling(T->lchild,e);
            GetSibling(T->rchild,e);
        }
        return q;
    }
}

status InsertNode(BiTree &T,KeyType e,int LR,TElemType c){ //插入
    BiTree newnode=(BiTree)malloc(sizeof(BiTNode));      //生成新结点
    newnode->data.key=c.key;                                //结点赋值
    strcpy(newnode->data.others,c.others);
    newnode->lchild=NULL;
    newnode->rchild=NULL;
    if(LR==-1) {                                           //根结点插入
        newnode->rchild=T->lchild;
        T->lchild=newnode;
        return OK;
    }
    else{
        BiTree p=LocateNode(T->lchild,e);
        if(p) {
            if(!LR) {                                     //左结点插入
                newnode->rchild=p->lchild;
                p->lchild=newnode;
            }
            else if(LR==1) {                               //右结点插入
                newnode->rchild=p->rchild;
                p->rchild=newnode;
            }
            return OK;
        }
    }
    return ERROR;
}

status DeleteNode(BiTree &T,KeyType e) {                 //删除
    if(T) {
        if(flag0) return 1;
        if(e==T->data.key) {
            if(!T->lchild&&!T->rchild) {                  //没有孩子

```



```

        free(T); //释放内存
        T=NULL;
    }else if(T->lchild&&T->rchild){ //两个孩子
        BiTree p=T->rchild,q=T;
        T=T->lchild;
        free(q);
        q=NULL;
        q=T;
        while(T->rchild) T=T->rchild;
        T->rchild=p; //右孩子接入左孩子
        T=q;
    }else if(T->lchild){ //有左孩子
        BiTree q=T;
        T=T->lchild;
        free(q);
        q=NULL;
    }else if(T->rchild){ //有右孩子
        BiTree q=T;
        T=T->rchild;
        free(q);
        q=NULL;
    }
    flag0=1; //flag0用于标记是否完成删除
    return OK;
}
}

void visit(BiTree T){
    printf("%d,%s ",T->data.key,T->data.others);
}

void PreOrderTraverse(BiTree T){ //先序
    if(T){
        visit(T);
        PreOrderTraverse(T->lchild);
        PreOrderTraverse(T->rchild);
    }
}

void InOrderTraverse(BiTree T){ //中序

```

```

    if(T) {
        InOrderTraverse(T->lchild);
        visit(T);
        InOrderTraverse(T->rchild);
    }
}

void PostOrderTraverse(BiTree T) {                                //后序
    if(T) {
        PostOrderTraverse(T->lchild);
        PostOrderTraverse(T->rchild);
        visit(T);
    }
}

void LevelOrderTraverse(BiTree T) {                                //按层
    BiTree p[100]={0};                                           //数组用于储存位序
    p[0]=T;
    int i=0, j=1;
    do{
        visit(p[i]);
        if(p[i]->lchild)                                         //储存左孩子
            p[j++]=p[i]->lchild;
        if(p[i]->rchild)                                         //储存右孩子
            p[j++]=p[i]->rchild;
        i++;
    }while(i<j);
}

status SaveBiTree(BiTree T, char FileName[]) {                    //写入文件
    int i=1;
    FILE *fp;
    BiTree p[101]={0,};
    p[1]=T;
    if((fp=fopen(FileName, "wb"))){
        while(i<101){
            if(p[i]){
                fprintf(fp, "%d          %d          %s", i, p[i]->data.key, p[i]->data.others); //位序和结点数据写入文件
                if(p[i]->lchild)                               //左孩子存入数组
                    p[2*i]=p[i]->lchild;
                if(p[i]->rchild)                               //右孩子存入数组
                    p[2*i+1]=p[i]->rchild;
            }
        }
    }
}

```

```

        i++;
    }
    fclose(fp);
    return OK;
} else return ERROR;
}

status LoadBiTree(BiTree &T, char FileName[]) {
    int i=0, j;
    DEF p[100];
    for(j=0; j<100; j++)
        p[j].pos=0; //初始化数组
    FILE *fp;
    if(fp=fopen(FileName, "rb")) {
        while(fscanf(fp, "%d %d %s", &p[i].pos, &p[i].data.key, p[i].data.others) != EOF) //文件数据写入数组
            i++;
        CreateBiTree(T->lchild, p); //创建新二叉树
        fclose(fp);
        return OK;
    }
    else return ERROR;
}

status AddList(LISTS &Lists, char ListName[]) { //新增二叉树
    strcpy(Lists.elem[Lists.length].name, ListName);
    Lists.elem[Lists.length++].T=(BiTNode*)malloc(sizeof(BiTNode));
    return OK;
}

status RemoveList(LISTS &Lists, int i) { //移除二叉树
    free(Lists.elem[i].T);
    for(int j=i; j<Lists.length-1; j++)
        Lists.elem[j]=Lists.elem[j+1]; //覆盖前一个
    Lists.length--;
    return OK;
}

int LocateList(LISTS Lists, char ListName[]) { //查找二叉树
    for(int i=0; i<Lists.length; i++)
        if(!strcmp(ListName, Lists.elem[i].name))
            return i+1;
    return 0;
}

```

}

附录 D 基于邻接表图实现的源程序

```
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define MAX_VERTEX_NUM 20
typedef int status;
typedef int KeyType;
typedef struct{
    KeyType key;
    char others[20];
} VertexType;
typedef struct ArcNode{
    int adjvex;
    struct ArcNode *nextarc;
}ArcNode;
typedef struct VNode{
    VertexType data;
    ArcNode *firstarc;
}VNode, AdjList[MAX_VERTEX_NUM];
typedef struct{
    AdjList vertices;
    int vexnum, arcnum;
}ALGraph;
typedef struct{
    struct{
        char name[30];
        ALGraph G;
    }elem[10];
    int length;
}LISTS;
void Visit(VertexType v);
void Traverse(ALGraph G);
int locate(int key, VertexType V[]);
status CreateCraph(ALGraph &G, VertexType V[], KeyType VR[][2]);
status DestroyGraph(ALGraph &G);
int LocateVex(ALGraph G, KeyType u);
status PutVex(ALGraph &G, KeyType u, VertexType value);
```

```

int FirstAdjVex(ALGraph G,KeyType u);
int NextAdjVex(ALGraph G,KeyType v,KeyType w);
status InsertVex(ALGraph &G,VertexType v);
status DeleteVex(ALGraph &G,KeyType v);
status InsertArc(ALGraph &G,KeyType v,KeyType w);
status DeleteArc(ALGraph &G,KeyType v,KeyType w);
status DFSTraverse(ALGraph &G);
status BFSTraverse(ALGraph &G);
status SaveGraph(ALGraph G, char FileName[]);
status LoadGraph(ALGraph &G, char FileName[]);
status AddList(LISTS &Lists,char ListName[]);
status RemoveList(LISTS &Lists,int i);
int LocateList(LISTS Lists,char ListName[]);
int sum;                //外部变量,用于记录遍历时已遍历顶点的个数
int t;                  //外部变量,用于记录当前遍历的顶点
int visited[20];        //外部变量,用于记录顶点是否已遍历过

int main() {
    ALGraph G;
    G.vexnum=0;
    VertexType V[30],value;
    KeyType VR[100][2];
    int flag,op,i,j,u,v,w,n,x;
    LISTS Lists;
    Lists.length=0;
    char FileName[30],name[30],Name[30];
    printf("      Menu for Linear Table On Sequence Structure \n");
    printf("-----\n");
    printf("      1. CreateCraph(单)   9. InsertArc\n");
    printf("      2. DestroyGraph(单) 10. DeleteArc\n");
    printf("      3. LocateVex        11. DFSTraverse\n");
    printf("      4. PutVex           12. BFSTraverse\n");
    printf("      5. FirstAdjVex      13. SaveGraph\n");
    printf("      6. NextAdjVex       14. LoadGraph(多)\n");
    printf("      7. InsertVex        15. AddList(多)\n");
    printf("      8. DeleteVex        16. RemoveList(多)\n");
    printf("      0. exit             17. LocateList(多)\n");
    printf("-----\n");
    printf("      \n 单个图操作输入 0, 多个图操作输入 1:");
    scanf("%d",&flag);
    while(1) {
        printf("      \n\n 请选择你的操作[0~19]: ");
        scanf("%d",&op);
        if(!flag&&op>13) {

```

```

        printf("请选择多图操作\n");
        continue;
    }
    if(flag&&op>0&&op<3) {
        printf("请选择单图操作\n");
        continue;
    }
    if(flag&&op<14&&op>0) {
        printf("\n 输入操作的图的名字: ");
        scanf("%s",Name);
        x=LocateList(Lists,Name)-1;
        if(x>=0) G=Lists.elem[x].G;
        else{
            printf("INFEASIBLE");
            continue;
        }
    }
    switch(op) {
        case 1: //创建
            if(G.vexnum) printf("INFEASIBLE");
            else{
                printf("请输入顶点数据, 结束输入-1 null: ");
                i=0;
                do{
                    scanf("%d%s",&V[i].key,V[i].others);
                } while(V[i++].key!=-1); //输入顶点数据
                for(i=0;V[i+1].key!=-1;i++){
                    for(j=i+1;V[j].key!=-1;j++){
                        if(V[i].key==V[j].key){
                            j=0;
                            break;
                        }
                    }
                    if(!j) break;
                }
                if(!j){
                    printf("ERROR");
                    break;
                }
                else{
                    printf("请输入弧, 结束输入-1 -1: ");
                    i=0;
                    do{
                        scanf("%d%d",&VR[i][0],&VR[i][1]);

```

```

        }while (VR[i++][0]!=-1); //输入弧数据
        if(CreateCraph(G,V,VR))
            printf("OK");
        else printf("ERROR");
    }
}
break;
case 2: //销毁
    if(!G.vexnum) printf("INFEASIBLE");
    else{
        if(flag) j=DestroyGraph(Lists.elem[x].G);
        else j=DestroyGraph(G);
        if(j) printf("OK");
        else printf("ERROR");
    }
    break;
case 3: //清空
    if(!G.vexnum) printf("INFEASIBLE");
    else{
        printf("请输入想查找的顶点关键字: ");
        scanf("%d",&u);
        j=LocateVex(G,u);
        if(j!=-1) Visit(G.vertices[j].data);
        else printf("ERROR");
    }
    break;
case 4: //赋值
    if(!G.vexnum) printf("INFEASIBLE");
    else{
        printf("请输入想赋值的顶点关键字: ");
        scanf("%d",&u);
        printf("请输入顶点数值: ");
        scanf("%d%s",&value.key,value.others);
        if(flag)
            if(PutVex(Lists.elem[x].G,u,value))
                for(i=0;i<Lists.elem[x].G.vexnum;i++)
                    Visit(Lists.elem[x].G.vertices[i].data);
            else printf("ERROR");
        else
            if(PutVex(G,u,value))
                for(i=0;i<G.vexnum;i++)
                    Visit(G.vertices[i].data);
            else printf("ERROR");
    }

```

```

    }
    break;
case 5:                                     //获得第一邻接顶点
    if(!G.vexnum) printf("INFEASIBLE");
    else{
        printf("请输入想查找的顶点关键字: ");
        scanf("%d",&u);
        j=FirstAdjVex(G,u);
        if(j!=-1) Visit(G.vertices[j].data);
        else printf("ERROR");
    }
    break;
case 6:                                     //获得下一邻接顶点
    if(!G.vexnum) printf("INFEASIBLE");
    else{
        printf("请输入想查找的顶点和邻接顶点关键字: ");
        scanf("%d%d",&v,&w);
        j=NextAdjVex(G,v,w);
        if(j!=-1) Visit(G.vertices[j].data);
        else printf("ERROR");
    }
    break;
case 7:                                     //插入顶点
    if(!G.vexnum) printf("INFEASIBLE");
    else{
        printf("请输入想插入的顶点的数据: ");
        scanf("%d%s",&value.key,value.others);
        if(flag)
            if(InsertVex(Lists.elem[x].G,value))
                Traverse(Lists.elem[x].G);
            else printf("ERROR");
        else if(InsertVex(G,value))
            for(i=0;i<G.vexnum;i++)
                Visit(G.vertices[i].data);
            else printf("ERROR");
    }
    break;
case 8:                                     //删除顶点
    if(!G.vexnum) printf("INFEASIBLE");
    else{
        printf("请输入想删除的顶点的关键值: ");
        scanf("%d",&v);
        if(flag)
            if>DeleteVex(Lists.elem[x].G,v))

```



```

        Traverse(Lists.elem[x].G);
        else printf("ERROR");
        else if(DeleteVex(G,v))
            Traverse(G);
        else printf("ERROR");
    }
    break;
case 9:                                     //插入弧
    if(!G.vexnum) printf("INFEASIBLE");
    else{
        printf("请输入想插入的弧: ");
        scanf("%d%d",&v,&w);
        if(flag)
            if(InsertArc(Lists.elem[x].G,v,w))
                Traverse(Lists.elem[x].G);
            else printf("ERROR");
        else if(InsertArc(G,v,w))
            Traverse(G);
        else printf("ERROR");
    }
    break;
case 10:                                    //删除弧
    if(!G.vexnum) printf("INFEASIBLE");
    else{
        printf("请输入想删除的弧: ");
        scanf("%d%d",&v,&w);
        if(flag)
            if(DeleteArc(Lists.elem[x].G,v,w))
                Traverse(Lists.elem[x].G);
            else printf("ERROR");
        else if(DeleteArc(G,v,w))
            Traverse(G);
        else printf("ERROR");
    }
    break;
case 11:                                    //深搜
    if(!G.vexnum) printf("INFEASIBLE");
    else{
        t=sum=0;                            //搜索总数和当前搜索位置为0
        for(i=0;i<20;i++)
            visited[i]=0;                    //所有顶点未搜索
        DFSTraverse(G);
    }
    break;

```

```

case 12:                                     //广搜
    if(!G.vexnum) printf("INFEASIBLE");
    else{
        t=sum=0;                           //搜索总数和当前搜索位置为0
        for(i=0;i<20;i++)
            visited[i]=0;                   //所有顶点未搜索
        BFSTraverse(G);
    }
    break;
case 13:                                     //写入文件
    if(!G.vexnum) printf("INFEASIBLE");
    else{
        printf("输入文件名:");
        scanf("%s",FileName);
        if(SaveGraph(G,FileName));
        printf("OK");
    }
    break;
case 14:
    printf("输入文件名: ");
    scanf("%s",FileName);
    while(1){
        printf("输入图名:");
        scanf("%s",name);
        if(LocateList(Lists,name))
            printf("重名\n");              //重名,重新输入
        else break;
    }
    AddList(Lists,name);
    if(LoadGraph(Lists.elem[Lists.length-1].G,FileName))
        Traverse(Lists.elem[Lists.length-1].G);
    else printf("ERROR\n");
    break;
case 15:                                     //新增图
    printf("输入要增加的图的个数:");
    scanf("%d",&n);
    if(Lists.length+n>10){
        printf("OVERFLOW\n");
        break;
    }
    while(n--){
        printf("输入图名:");
        scanf("%s",name);
        if(LocateList(Lists,name)){        //判断是否有重名

```

```

        printf("重名\n");
        n++;
        continue;
    }
    else{
        if(AddList(Lists,name)) {
            printf("请输入顶点数据, 结束输入-1 null:
");
            i=0;
            do{
                scanf("%d%s",&V[i].key,V[i].others);
            }while(V[i++].key!=-1);
            printf("请输入弧, 结束输入-1 -1: ");
            i=0;
            do{
                scanf("%d%d",&VR[i][0],&VR[i][1]);
            }while(VR[i++][0]!=-1);
            if(CreateCraph(Lists.elem[List.length-
1].G,V,VR))

                printf("OK\n");
            }
            else printf("ERROR\n");
        }
        for(i=0;i<30;i++)           //初始化数组
            V[i].key=-1;
        for(i=0;i<100;i++)
            VR[i][0]=VR[i][1]=-1;
    }
    break;
case 16:           //删除图
    printf("输入要删除的图的名字:");
    scanf("%s",name);
    if(LocateList(Lists,name)) {           //查找图
        if(RemoveList(Lists,LocateList(Lists,name)-1)) {
            for(n=0;n<Lists.length;n++) {
                printf("%s\n",Lists.elem[n].name);
                if(!Lists.elem[n].G.vexnum) printf("空图
\n");

                else Traverse(Lists.elem[n].G);
            }
        }
    }
    else printf("删除失败");
    break;

```

```

        case 17:                                //查找图
            printf("输入要查找的图的名字:");
            scanf("%s", name);
            if(n=LocateList(Lists, name)) {
                if(!Lists.elem[n-1].G.vexnum)
printf("INFEASIBLE");
                else{
                    printf("%s\n", Lists.elem[n-1].name);
                    if(!Lists.elem[n-1].G.vexnum)
                        printf("空图\n");
                    else Traverse(Lists.elem[n-1].G);
                }
            }
            else printf("查找失败");
            break;
        case 0: goto down;
    }
}
down:
printf("\n 欢迎下次再使用本系统! \n");
return 0;
}

void Visit(VertexType v) {                    //输出顶点
    printf("%d %s ", v.key, v.others);
}

void Traverse(ALGraph G) {                    //遍历图
    for(int i=0; i<G.vexnum; i++) {
        ArcNode *p=G.vertices[i].firstarc;

printf("%d %s", G.vertices[i].data.key, G.vertices[i].data.others);
        while(p) {
            printf(" %d", p->adjvex);
            p=p->nextarc;
        }
        printf("\n");
    }
}

int locate(int key, VertexType V[]) {
    int i;
    for(i=0; V[i].key!=-1; i++)
        if(key==V[i].key)

```

```

        return i;
    }

status CreateCraph(ALGraph &G, VertexType V[], KeyType VR[][2]) { //创建
    int i, j, k, flag;      //flag 用于标记弧中的数据是否与顶点关键字匹配
    ArcNode *p;
    for(k=0; k<2; k++) {
        for(j=0; VR[j][k]!=-1; j++) {
            for(flag=i=0; V[i].key!=-1; i++)
                if(V[i].key==VR[j][k])
                    flag=1;
            if(!flag) return ERROR;
        }
        //判断弧中的数据是否与顶点关键字匹配
    }
    G.vexnum=G.arcnum=0;
    for(i=0; V[i].key!=-1; i++) {
        G.vexnum++;
        G.vertices[i].data.key=V[i].key;      //顶点赋值
        strcpy(G.vertices[i].data.others, V[i].others);
        G.vertices[i].firstarc=NULL;
        for(j=0; VR[j][0]!=-1; j++) {
            p=G.vertices[i].firstarc;
            if(VR[j][0]==V[i].key) {

G.vertices[i].firstarc=(ArcNode*)malloc(sizeof(ArcNode));
                G.vertices[i].firstarc->adjvex=locate(VR[j][1], V);
                G.vertices[i].firstarc->nextarc=p;
                G.arcnum++;
            }
            else if(VR[j][1]==V[i].key) {

G.vertices[i].firstarc=(ArcNode*)malloc(sizeof(ArcNode));
                G.vertices[i].firstarc->adjvex=locate(VR[j][0], V);
                G.vertices[i].firstarc->nextarc=p;
                G.arcnum++;
            }
        }
        //分别将弧的一组数据加到相应的顶点链表中
    }
    G.arcnum/=2;      //弧的总数需要除以 2
    return OK;
}

status DestroyGraph(ALGraph &G) {      //销毁
    int i;

```

```

    ArcNode *p,*q;
    for(i=0;i<G.vexnum;i++){
        while(G.vertices[i].firstarc){
            p=G.vertices[i].firstarc;
            q=p->nextarc;
            free(p);
            G.vertices[i].firstarc=q;
        }
        G.vertices[i].firstarc=NULL;
    }
    G.vexnum=G.arcnum=0;
    return OK;
}

int LocateVex(ALGraph G,KeyType u) { //查找
    int i;
    for(i=0;i<G.vexnum;i++){
        if(G.vertices[i].data.key==u)
            return i;
    }
    return -1;
}

status PutVex(ALGraph &G,KeyType u,VertexType value) { //赋值
    int i=LocateVex(G,u);
    if(i==-1) return ERROR; //判断顶点是否存在
    if(u!=value.key&&LocateVex(G,value.key)!=-1) //判断关键字是否唯一
        return ERROR;
    G.vertices[i].data.key=value.key;
    strcpy(G.vertices[i].data.others,value.others);
    return OK;
}

int FirstAdjVex(ALGraph G,KeyType u) { //获得第一邻接顶点
    int i=LocateVex(G,u);
    if(i==-1) return -1; //判断顶点是否存在
    else return G.vertices[i].firstarc->adjvex;
}

int NextAdjVex(ALGraph G,KeyType v,KeyType w) { //获得下一邻接顶点
    int i=LocateVex(G,v),j=LocateVex(G,w);
    if(i==-1||j==-1) return -1; //判断弧是否存在
    ArcNode *p=G.vertices[i].firstarc;
    while(p->nextarc){

```

```

        if(p->adjvex==j)
            return p->nextarc->adjvex;
        p=p->nextarc;
    }
    return -1;
}

status InsertVex(ALGraph &G, VertexType v) {           //插入顶点
    int i=LocateVex(G, v.key);
    if(i!=-1 || G.vexnum>19) return ERROR;           //判断关键字是否唯一
    G.vertices[G.vexnum].data.key=v.key;             //顶点赋值
    strcpy(G.vertices[G.vexnum].data.others, v.others);
    G.vertices[G.vexnum++].firstarc=NULL;
    return OK;
}

status DeleteVex(ALGraph &G, KeyType v) {             //删除顶点
    int i,w=LocateVex(G, v);
    if(w==-1 || G.vexnum==1) return ERROR;           //判断顶点是否存在
    ArcNode *p=NULL, *q=NULL;
    while(p=G.vertices[w].firstarc) {                 //释放链表内存
        q=p->nextarc;
        free(p);
        G.vertices[w].firstarc=q;
    }
    G.vexnum--;
    for(i=w; i<G.vexnum; i++)                          //后一顶点覆盖前一顶点
        G.vertices[i]=G.vertices[i+1];
    for(i=0; i<G.vexnum; i++) {                        //删除该顶点的弧
        if(!G.vertices[i].firstarc) continue;
        if(G.vertices[i].firstarc->adjvex==w) {        //为头结点下一结点
            G.arcnum--;
            p=G.vertices[i].firstarc;
            q=p->nextarc;
            free(p);
            G.vertices[i].firstarc=q;
        }
    }
    else {                                              //非头结点下一结点
        p=G.vertices[i].firstarc->nextarc;
        q=G.vertices[i].firstarc;
        while(p) {
            if(p->adjvex==w) {
                G.arcnum--;
                free(p);
            }
            p=q;
            q=q->nextarc;
        }
    }
}

```

```

        q->nextarc=p->nextarc;
        p=q->nextarc;
    }
    else{
        p=p->nextarc;
        q=q->nextarc;
    }
}
}
}
for(i=0;i<G.vexnum;i++){ //新的顶点位序调整
    p=G.vertices[i].firstarc;
    while(p){
        if(p->adjvex>w)
            p->adjvex--;
        p=p->nextarc;
    }
}
return OK;
}

status InsertArc(ALGraph &G,KeyType v,KeyType w){ //插入弧
    int i=LocateVex(G,v),j=LocateVex(G,w);
    if(i==-1||j==-1) return ERROR; //判断弧是否合法
    ArcNode
    *p=(ArcNode*)malloc(sizeof(ArcNode)),*q=G.vertices[i].firstarc,*r=(Ar
cNode*)malloc(sizeof(ArcNode)); //新结点分配内存
    while(q){
        if(q->adjvex==j) return ERROR; //判断弧是否已存
在
        q=q->nextarc;
    }
    p->adjvex=j; //新结点赋值
    r->adjvex=i;
    q=G.vertices[i].firstarc;
    G.vertices[i].firstarc=p;
    p->nextarc=q;
    q=G.vertices[j].firstarc;
    G.vertices[j].firstarc=r;
    r->nextarc=q;
    G.arcnum++;
    return OK;
}

```



```

status DeleteArc(ALGraph &G,KeyType v,KeyType w){    //删除弧
    int i=LocateVex(G,v),j=LocateVex(G,w),flag=0;
    if(i==-1||j==-1) return ERROR;                //判断弧是否存在
    ArcNode *p,*q;
    if(G.vertices[i].firstarc->adjvex==j){          //为头结点下一结点
        flag=1;
        p=G.vertices[i].firstarc;
        q=p->nextarc;
        free(p);
        G.vertices[i].firstarc=q;
    }
    else{                                             //为非头结点下一结点
        p=G.vertices[i].firstarc->nextarc;
        q=G.vertices[i].firstarc;
        while(p){
            if(p->adjvex==j){
                flag=1;
                q->nextarc=p->nextarc;                //删除结点
                free(p);                             //释放内存
                p=q->nextarc;                          //指向下一结点
            }
            else{
                p=p->nextarc;
                q=q->nextarc;
            }
        }
    }
    if(!flag) return ERROR;                          //flag用于标记是否已删除
    if(G.vertices[j].firstarc->adjvex==i){           //重复一次,删除另一个结点
        p=G.vertices[j].firstarc;
        q=p->nextarc;
        free(p);
        G.vertices[j].firstarc=q;
    }
    else{
        p=G.vertices[j].firstarc->nextarc;
        q=G.vertices[j].firstarc;
        while(p){
            if(p->adjvex==i){
                q->nextarc=p->nextarc;
                free(p);
                p=q->nextarc;
            }
            else{

```

```

        p=p->nextarc;
        q=q->nextarc;
    }
}
}
G. arcnum--;
return OK;
}

status DFSTraverse(ALGraph &G) { //深搜
    ArcNode *p=NULL;
    int w,i;
    Visit(G.vertices[t].data);
    visited[t]=1; //标记为已访问
    sum++; //访问数加 1
    p=G.vertices[t].firstarc;
    while(p) {
        w=p->adjvex;
        if(!visited[w]) {
            t=w;
            DFSTraverse(G); //访问下一未访问过的邻接顶点
        }
        p=p->nextarc;
    }
    if(sum!=G.vexnum) { //不连通图
        for(i=0;i<G.vexnum;i++) {
            if(!visited[i]) { //寻找下一未访问顶点
                t=i;
                break;
            }
        }
        DFSTraverse(G);
    }
}

status BFSTraverse(ALGraph &G) { //广搜
    ArcNode *p;
    int w,i;
    int q[G.vexnum], front=0, rear=0; //队列用于记录联通分支是否已访问完
    Visit(G.vertices[t].data);
    visited[t]=1;
    sum++;
    rear=(rear+1)%G.vexnum; //队尾指针加 1
    q[rear]=t;

```

```

while(front!=rear){
    front=(front+1)%G.vexnum;           //队首指针加 1
    w=q[front];
    p=G.vertices[w].firstarc;
    while(p){
        if(!visited[p->adjvex]){        //访问下一未访问过的顶点
            Visit(G.vertices[p->adjvex].data);
            sum++;
            visited[p->adjvex]=1;
            rear=(rear+1)%G.vexnum;    //队尾指针加 1
            q[rear]=p->adjvex;         //加入队列
        }
        p=p->nextarc;
    }
}
if(sum!=G.vexnum){                      //非连通图
    for(i=0;i<G.vexnum;i++){
        if(!visited[i]){               //寻找下一未访问过的顶点
            t=i;
            break;
        }
    }
    BFSTraverse(G);
}
}

status SaveGraph(ALGraph G, char FileName[]){ //写入文件
    int i;
    ArcNode *p;
    FILE *fp;
    if(fp=fopen(FileName,"wb")){
        fprintf(fp,"%d %d",G.vexnum,G.arcnum); //写入顶点数和弧数
        fputc('\n', fp);                       //写完后换行
        for(i=0;i<G.vexnum;i++){
            fprintf(fp,"%d %s",G.vertices[i].data.key,G.vertices[i].data.others);
            for(p=G.vertices[i].firstarc;p;p=p->nextarc)
                fprintf(fp," %d",p->adjvex);
            fputc('\n',fp);                     //写完一顶点后换行
        }
        fclose(fp);
        return OK;
    }else return ERROR;
}

```

```

status LoadGraph(ALGraph &G, char FileName[]) {    //读取文件
    ArcNode *p;
    FILE *fp;
    if(fp=fopen(FileName, "r")) {
        fscanf(fp, "%d %d", &G.vexnum, &G.arcnum);    //读取顶点数和弧数
        fgetc(fp);    //读取换行符
        int a=0, i;
        for(i=0; i<G.vexnum; i++) {
            fscanf(fp, "%d %s", &G.vertices[i].data.key, G.vertices[i].data.others);
            G.vertices[i].firstarc=NULL;
            if(fgetc(fp)!='\n') {
                fscanf(fp, "%d", &a);
                G.vertices[i].firstarc=(ArcNode
*)malloc(sizeof(ArcNode));    //生成头结点
                G.vertices[i].firstarc->adjvex=a;
                G.vertices[i].firstarc->nextarc=NULL;
                p=G.vertices[i].firstarc;
                while(fgetc(fp)!='\n') {    //换行符前为同一顶点数据
                    fscanf(fp, "%d", &a);
                    p->nextarc=(ArcNode *)malloc(sizeof(ArcNode));
                    p->nextarc->adjvex=a;
                    p->nextarc->nextarc=NULL;    //生产邻接结点
                    p=p->nextarc;
                }
            }
        }
        fclose(fp);
        return OK;
    }else return ERROR;
}

status AddList(LISTS &Lists, char ListName[]) {
    strcpy(Lists.elem[Lists.length++].name, ListName);
    return OK;
}

status RemoveList(LISTS &Lists, int i) {
    for(int j=i; j<Lists.length-1; j++)
        Lists.elem[j]=Lists.elem[j+1];
    Lists.length--;
    return OK;
}

```

```
int LocateList(LISTS Lists, char ListName[]) {  
    for(int i=0; i<Lists.length; i++)  
        if(!strcmp(ListName, Lists.elem[i].name))  
            return i+1;  
    return 0;  
}
```