

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 校际交流1601班**

**学 号： U201610504**

**姓 名： 刘逸帆**

**指导教师： 魏巍**

**报告日期： 2018年1月11日**

**计算机科学与技术学院**

**目 录**

[1基于顺序存储结构的线性表实现 1](#_Toc503181733)

[1.1 问题描述 1](#_Toc503181734)

[1.2 系统设计 3](#_Toc503181735)

[1.3 系统实现 15](#_Toc503181736)

[1.4 实验小结 33](#_Toc503181737)

[2基于链式存储结构的线性表实现 36](#_Toc503181738)

[2.1 问题描述 36](#_Toc503181739)

[2.2 系统设计 38](#_Toc503181740)

[2.3 系统实现 50](#_Toc503181741)

[2.4 实验小结 68](#_Toc503181742)

[3基于链式存储结构的二叉树实现 70](#_Toc503181743)

[3.1 问题描述 70](#_Toc503181744)

[3.2 系统设计 73](#_Toc503181745)

[3.3 系统实现 93](#_Toc503181746)

[3.4 实验小结 121](#_Toc503181747)

[4基于邻接表存储结构的图(无向网)实现 124](#_Toc503181748)

[4.1 问题描述 124](#_Toc503181749)

[4.2 系统设计 126](#_Toc503181750)

[4.3 系统实现 144](#_Toc503181751)

[4.4 实验小结 173](#_Toc503181752)

[参考文献 177](#_Toc503181753)

[附录A 基于顺序存储结构的线性表实现的源程序 179](#_Toc503181754)

[附录B 基于链式存储结构的线性表实现的源程序 191](#_Toc503181757)

[附录C 基于链式存储结构的二叉树实现的源程序 206](#_Toc503181760)

[附录D 基于邻接表存储结构的图实现的源程序 232](#_Toc503181763)

# 1基于顺序存储结构的线性表实现

## 1.1 问题描述

为实现基于顺序存储结构的线性表及以此为基础的多表管理、文件读写功能，按如下形式设计数据的逻辑结构及基本运算。

### 1.1.1 线性表抽象数据类型

依据最小完备性和常用性相结合的原则，设计了线性表的数据对象和数据关系，并定义了线性表的初始化、销毁、置空、求长度、获取数据元素等12种基本运算，具体数据和运算功能定义如下。

1. 数据结构

数据对象：D = { ai|ai ∈ Elem }

数据关系：R = { <ai-1,ai>|ai-1, ai∈D }

1. 基本运算

初始化表：函数名称是InitaList(\*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。

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

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

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

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

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

查找元素：函数名称是LocateElem(L,e,\*i)；初始条件是线性表已存在；操作结果是用i返回L中第1个数据元素e的位序，若这样的数据元素不存在，则返回值为0。

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

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

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

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

遍历表：函数名称是ListTraverse(L)，初始条件是线性表L已存在；操作结果是依次输出展示L的每个数据元素。

### 1.1.2 多线性表抽象数据类型

基于1.1.1中已经实现的线性表数据类型与基本运算功能，设计多线性表的数据对象、数据关系。具体数据和多表间切换功能定义如下。

1. 数据结构

数据对象：D = { ei|ei ∈ SqList }

数据关系：R = { <ei-1,ei>|ei-1,ei∈D }

1. 基本运算

定义Sqlist类型静态数组MulList[MAX]，其中MAX即多顺序表个数；定义Sqlist类型指针\*L，通过其指向数组的不同元素实现对多顺序表的基本操作。

### 1.1.3 演示系统与数据文件

构造一个具有文本菜单的功能演示系统，用于演示对顺序表以及多表进行的各项操作。其中，在主函数中完成子函数调用所需的参数输入、结果显示以及对应的文字操作提示。

在设计了相应数据存储格式后，演示系统可进行线性表数据的文件保存或读取，增加系统合理性。

演示系统还可以根据需要实现多线性表的管理操作。

## 1.2 系统设计

对数据类型与基本操作进行相应分析后，从物理结构，具体算法等方面合理设计系统，具体设计分析如下。

### 1.2.1 数据物理结构

1. 线性表数据的存储结构

typedef struct{ //顺序表（线性表/顺序结构）的定义

ElemType \*elem; //指向线性表数据的线性存储区

int length; //表中元素个数（长度）

int listsize; //存储容量

}SqList;

1. 多线性表数据的存储结构

SqList MulList[MAX] //通过数组形式构建多表结构

多线性表数据的物理存储结构如图1-1所示。

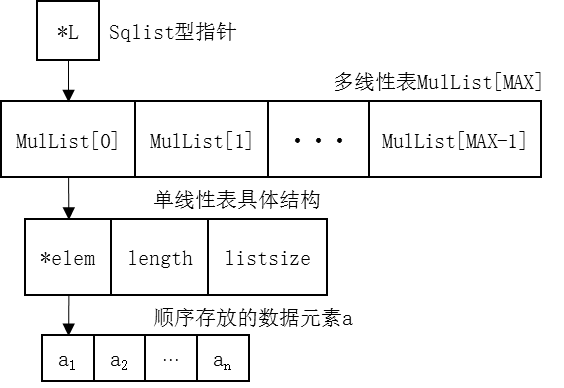


图1-1 多线性表数据的物理存储结构

### 1.2.2 演示系统

演示系统使用基于标准输入输出库（stdio.h）实现的文本界面，由功能菜单界面和用户操作（交互）界面组成。

功能菜单界面以文本形式输出功能菜单，展示当前操作线性表序号以及可选的线性表操作项，供用户参考选择调用需要的功能。

用户操作（交互）界面供用户进行功能的选择、数据的输入等，将用户输入的相关信息传递给对应线性表操作函数，输出调用函数后的操作结果或错误提示等信息。

演示系统具体模块结构如图1-2所示。

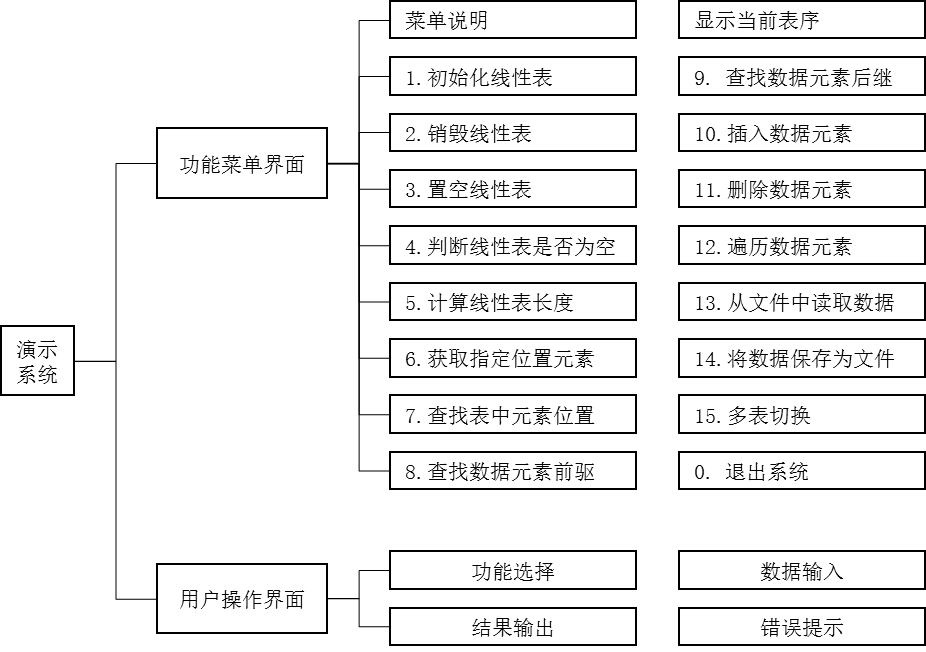


图1-2 演示系统模块结构图

### 1.2.3 数据文件存储格式

考虑到存储的数据文件需具有简洁高效的直观实用的性能，按如下方式设计数据文件存储格式。

文件格式：txt，便于直接查看和修改等。

存放方式：通过fprintf函数顺序存放数据值的十进制字符形式，每两个元素间添加一个空格符以分开。

数据文件存储格式如图1-3所示。

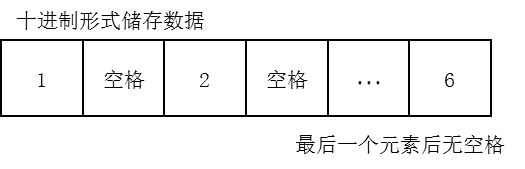


图1-3 数据文件存储格式示例

### 线性表运算算法实现

线性表运算算法及设计如下：

1. 初始化表函数InitaList(\*L)

**算法分析：**首先判断L->elem是否为空（判断线性表创建与否），若不为空则返回ERROR提示线性表已被初始化，不执行函数功能；否则为L->elem分配LIST\_INIT\_SIZE个ElemType类型大小的存储空间，若分配失败则返回ERROR提示溢出，否则初始化表长为0且记录存储容量为LIST\_INIT\_SIZE，返回OK。

**操作结果：**构造一个空的线性表。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 销毁表函数DestroyList(\*L)

**算法分析：**首先判断L->elem是否为空（判断线性表创建与否），若为空则返回ERROR提示线性表未创建，不执行函数功能；若不为空，释放L->elem所占用的内存空间，并将L->elem指针置空，销毁线性表，返回OK。

**操作结果：**销毁线性表。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 清空表函数ClearList(\*L)

**算法分析：**首先判断L->elem是否为空（判断线性表创建与否），若为空则返回ERROR提示线性表未创建，不执行函数功能；若不为空，使L->elem指针置空并重新初始化线性表以分配存储空间，从而实现置空线性表功能，返回OK。

**操作结果：**置空线性表。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 判定空表函数ListEmpty(L)

**算法分析：**首先判断L->elem是否为空（判断线性表创建与否），若为空则返回ERROR提示线性表未创建，不执行函数功能；若不为空，读取L.length表长数据，若为0，返回OK提示是空表，否则返回error提示不是空表，。

**操作结果：**判断线性表是否为空。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 求表长函数ListLength(\*L)

**算法分析：**首先判断L->elem是否为空（判断线性表创建与否），若为空则返回ERROR提示线性表未创建，不执行函数功能；若不为空，读取L->length表长数据，返回表长度。

**操作结果：**计算线性表长度。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 获得元素函数GetElem(L,i,\*e)

**算法分析：**首先判断L->elem是否为空（判断线性表创建与否），若为空则返回ERROR提示线性表未创建，不执行函数功能；若不为空，判断i值是否在线性表合法范围1≤i≤ListLength(L)内，i不合法则返回ERROR提示操作越界，否则用e传递第i个数据元素的值，返回OK。

**操作结果：**用e返回线性表L中第i个数据元素的值。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 查找元素函数LocateElem(L,e,\*i)

**算法分析：**首先判断L->elem是否为空（判断线性表创建与否），若为空则返回ERROR提示线性表未创建，不执行函数功能；若不为空，遍历线性表查找数据元素e第一次出现的位置，若查询到，则用i传递该位置的位序并返回OK，若不存在这样的数据元素e，则返回ERROR提示未查询到对应的数据元素。

**操作结果：**查找线性表L中数据元素e的位序。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**算法流程图：**如图1-4所示。

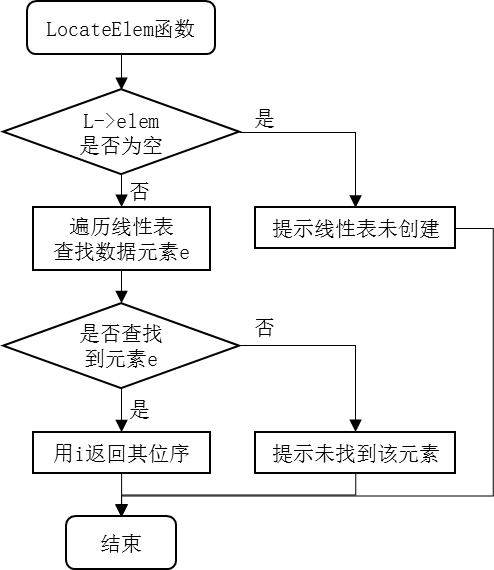


图1-4 查找元素函数LocateElem算法流程图

1. 获得前驱函数PriorElem(L,cur,\*pre\_e)

**算法分析：**首先判断L->elem是否为空（判断线性表创建与否），若为空则返回ERROR提示线性表未创建，不执行函数功能；若不为空，遍历线性表查找数据元素cur第一次出现的位置，若找到且其位序不为1，则用pre\_e传递该数据元素的前驱元素并返回OK，若不存在这样的数据元素e，则返回ERROR提示未查询到对应的数据元素或对应的数据元素无前驱。

**操作结果：**查找线性表L中数据元素cur的前驱元素pre\_e。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**算法流程图：**如图1-5所示。

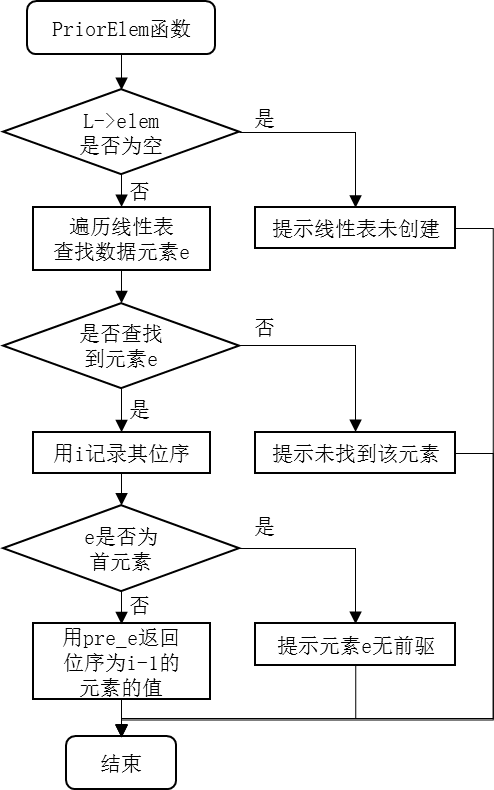


图1-5 获得前驱函数PriotElem算法流程图

1. 获得后继函数NextElem(L,cur,\*next\_e)

**算法分析：**首先判断L->elem是否为空（判断线性表创建与否），若为空则返回ERROR提示线性表未创建，不执行函数功能；否则，遍历线性表查找数据元素cur第一次出现的位置，若找到且其位序不为length，则用next\_e传递该数据元素的后继元素并返回OK，若不存在这样的数据元素e，则返回ERROR提示未查询到对应的数据元素或对应的数据元素无后继。

**操作结果：**查找线性表L中数据元素cur的后继元素next\_e。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**算法流程图：**如图1-6所示。

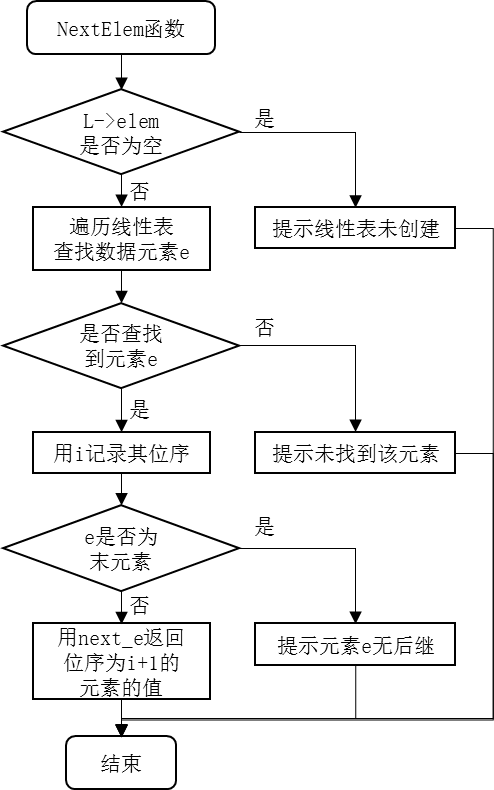


图1-6 获得后继函数NextElem算法流程图

1. 插入元素函数ListInsert(\*L,i,e)

**算法分析：**首先判断L->elem是否为空（判断线性表创建与否），若为空则返回ERROR提示线性表未创建，不执行函数功能；若不为空，判断插入位序i是否属于合法范围1≤i≤ListLength(L)+1，若插入位序不在合法范围内，返回ERROR并提示操作越界，否则先判断线性表存储空间是否足够，不足则申请一个新的ElemType类型大小的空间，申请失败则返回ERROR提示分配空间失败，申请成功或线性表存储空间足够则在位序i处插入新的数据元素e，返回OK。

**操作结果：**在线性表L的第i个位置之前插入新的数据元素e。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图1-7所示。

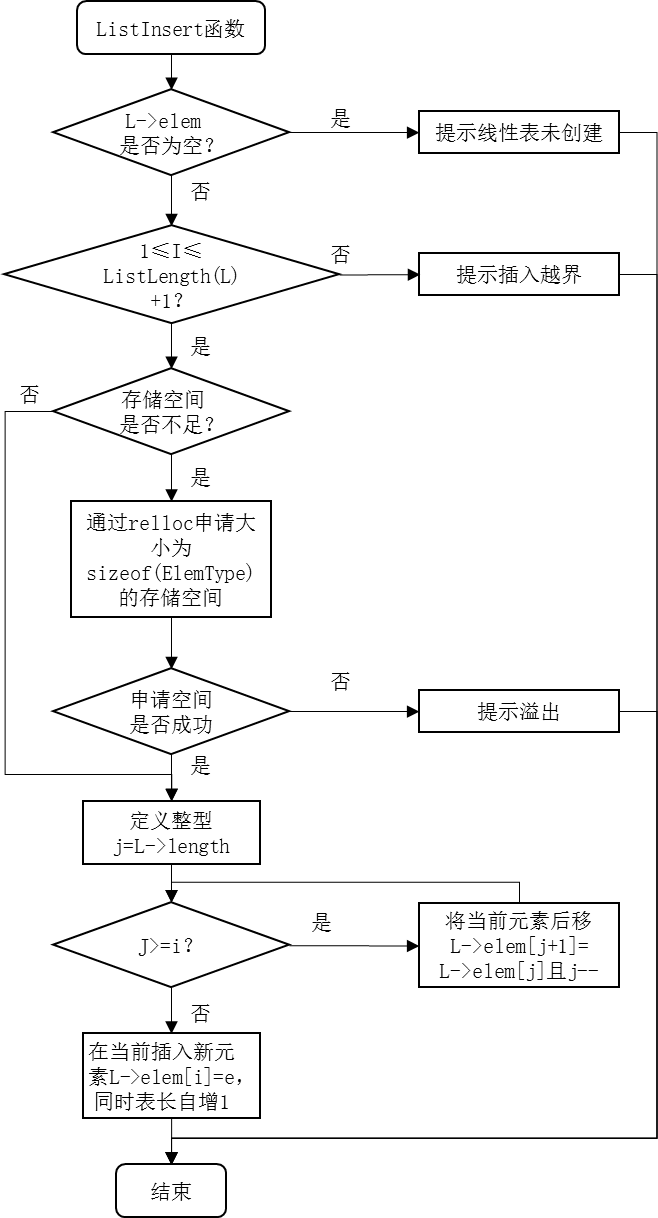


图1-7 插入元素函数ListInsert算法流程图

1. 删除元素函数ListDelete(\*L,i,\*e)

**算法分析：**首先判断L->elem是否为空（判断线性表创建与否），若为空则返回ERROR提示线性表未创建，不执行函数功能；若不为空， 判断待删除位序i是否属于合法范围1≤i≤ListLength(L)，若位序不在合法范围内，返回ERROR并提示操作越界，否则用e传递将被删除的数据元素的值，并从第i个数据元素开始，将其后继元素前移一位，线性表长度L->length--，返回OK。

**操作结果：**将线性表第i个数据元素删除并用e返回其值。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图1-8所示。

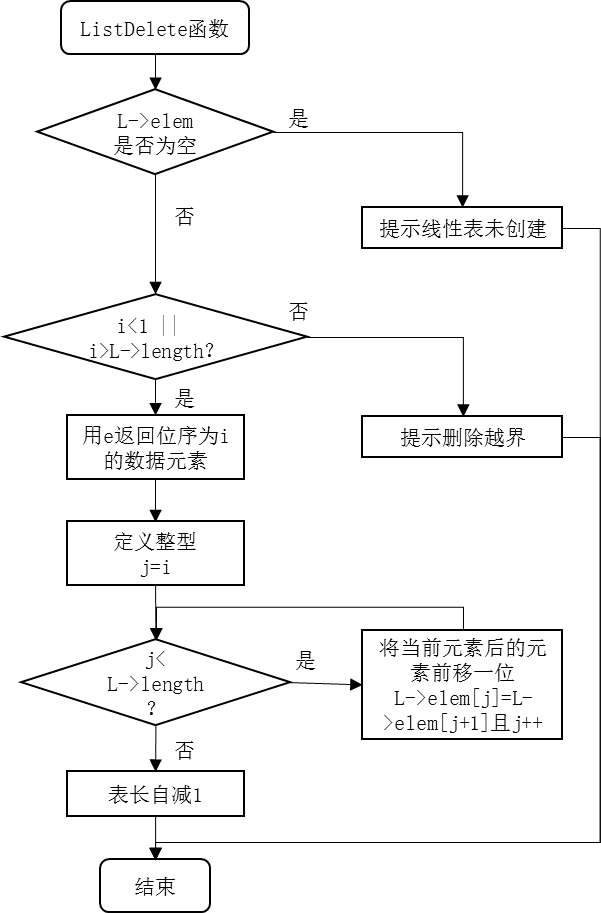


图1-8 删除元素函数ListDelete算法流程图

1. 遍历表函数ListTraverse(L)

**算法分析：**首先判断L->elem是否为空（判断线性表创建与否），若为空则返回ERROR提示线性表未创建，不执行函数功能；若不为空，通过printf绘制分界线，遍历线性表输出展示所有数据元素，每两个元素间间隔一个空格符，返回线性表长度。

**操作结果：**遍历线性表并输出L的每个数据元素。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

1. case13中的文件读取算法

**算法分析：**首先判断L->elem是否为空（判断线性表创建与否），若为空则返回ERROR提示线性表未创建，不执行文件读取功能；若不为空，输出相应提示信息，提示输入需要读取的文件名称，通过只读方式打开文件，利用fscanf函数从文件中读取规范格式的十进制数据并将其通过ListInsert函数插入当前操作表首部，直至读取到文件尾。同时读取时提示输出读入的数据信息，读取完成后输出表长并关闭文件。

**操作结果：**将指定文件中十进制数据读入系统并添加在当前操作表表首。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图1-9所示。

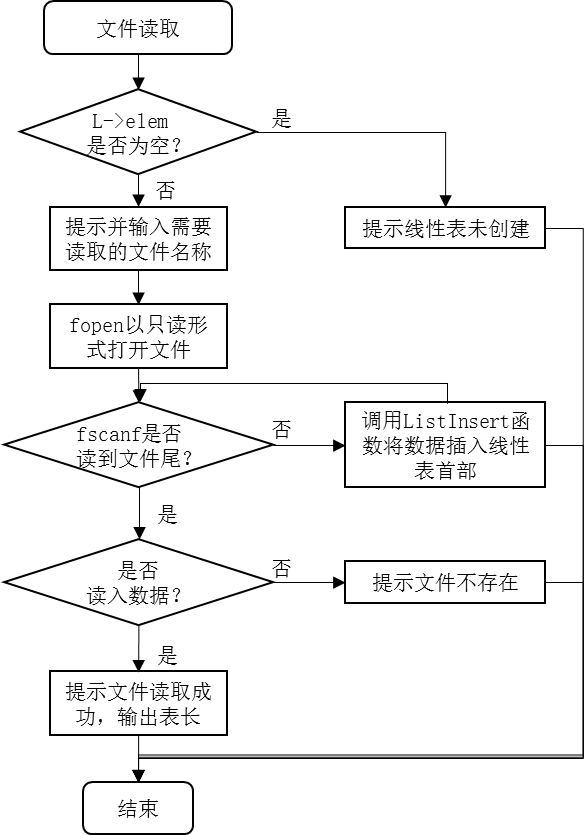


图1-9 文件读取算法流程图

1. case14中的文件输出算法

**算法分析：**首先判断L->elem是否为空（判断线性表创建与否），若为空则返回ERROR提示线性表未创建，不执行文件输出功能；若不为空，输出相应提示信息，提示输入将写入的文件名称，通过wb+参数以新建或覆盖的方式打开文件，利用fprintf函数向文件中写入规范格式的十进制数据并输出提示当前写入数据的信息，直至写入所有线性表中的数据，提示写入完毕并关闭文件。

**操作结果：**向指定文件中写入十进制数据。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图1-10所示。

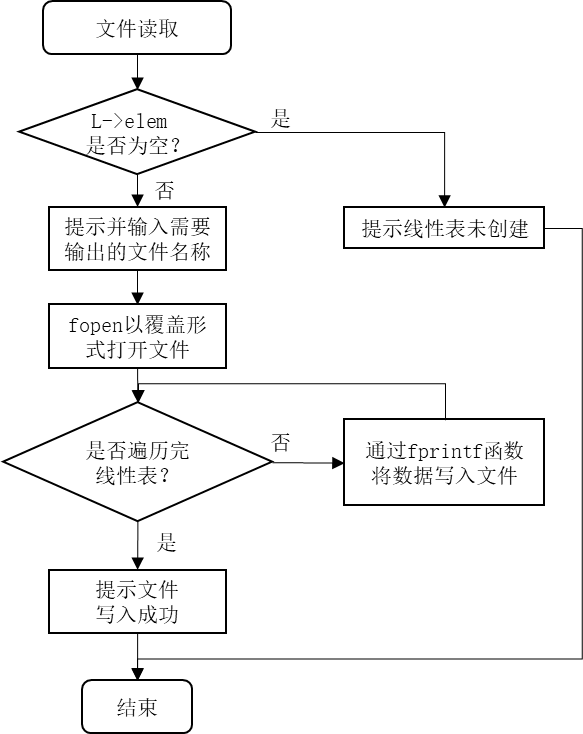


图1-10 文件输出算法流程图

### 1.2.5 多线性表切换算法实现

case15中多表切换算法

**算法分析：**数据结构设计中，MulList[MAX]为Sqlist类型的静态数组，\*L为Sqlist类型的指针。提示输入表序号list\_num，判断序号是否在合法范围内1≤i≤MAX，非法则输出错误信息并切换至表1，否则通过L=MulList+list\_num切换当前操作的线性表。

**操作结果：**通过指定序号切换当前操作的线性表。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

## 1.3 系统实现

具体系统实现结果如下。

### 1.3.1 实验环境

实验代码使用code::block编写与编译。

### 1.3.2 演示系统操作

演示系统采用键盘操作，控制台输出操作提示与结果。

文本界面显示可选的功能菜单（共有0~15项功能），其中第二行显示当前操作表的序号（默认为1），输入对应操作指令可使用相应功能，输入不在菜单中的操作指令会刷新界面并重新等待输入。演示系统界面如图1-11所示。



图1-11 启动程序后的操作菜单

输入操作指令0，程序退出，如图1-12所示。



图1-12 程序退出

### 1.3.3 测试计划

对系统的线性表操作功能按以下计划测试，其中单表运算操作测试计划如表1-1所示，多表操作功能测试计划如表1-2所示。

表1-1 线性表操作功能测试

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 待测试功能 | 测试顺序 | 测试输入 | 预计输出 | 线性表状态 |
| 1.IntiaList | 1 | 无 | 初始化线性表成功 | 创建了空表，表长为0 |
| 2.DestroyList | 15 | 无 | 销毁线性表成功 | 线性表不存在 |
| 3.ClearList | 9 | 无 | 置空线性表成功 | 线性表为空表 |
| 4.ListEmpty | 6 | 无 | 线性表不为空 | 表中元素为1、5  表长为2 |
| 5.ListLenggth | 5 | 无 | 线性表长度为2 | 表中元素为1、5  表长为2 |
| 6.GetElem | 8 | 分别输入  位序2、  位序1 | 第一次查找越界提示错误信息、  第二次查找得到数据1 | 表中元素为1  表长为1 |
| 7.LocateElem | 4 | 分别输入  数据5、  数据6 | 第一次查找得到位序2、  第二次查找提示错误信息 | 表中元素为1、5  表长为2 |
| 8.PriorElem | 3 | 分别输入  数据1、  数据5 | 第一次查找提示错误信息、  第二次查找前驱数据为1 | 表中元素为1、5  表长为2 |
| 9.NextElem | 13 | 分别输入  数据50、  数据60 | 第一次查找得到后继数据为60、  第二次查找失败提示越界 | 表中元素为10、20、50、60  表长为4 |
| 10.ListInsert | 2 | 分别输入  位序2数据1、  位序1数据1、  位序2数据5 | 第一次插入失败提示越界、  第二个数据插入成功、  第三个数据插入成功 | 表中元素为1、5  表长为2 |
| 11.ListDelete | 7 | 分别输入  位序2、  位序2 | 第一次删除成功返回数据5、  第二次删除失败提示越界 | 表中元素为1  表长为1 |
| 12.ListTraverse | 10、12 | 无 | 输出表中所有数据  第一次调用输出1、  第二次调用输出10、20、50、60 | 第一次调用时  表中元素为1  表长为1、  第二次调用时表中元素为10、20、50、60  表长为4 |
| 13.OpenFile | 11 | 打开准备好的文件file001，其中的元素为10、20、50、60 | 文件读取成功 | 表中元素10、20、50、60，表长为4 |
| 14.SaveAsFile | 14 | 保存为文件file002 | 文件保存成功  文件file002中为十进制数据10 20 50 60 | 表中元素10、20、50、60，表长为4 |
| 0.Exit | 16 | 无 | 退出系统 | 线性表不存在 |

表1-2 多线性表操作功能测试

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试顺序 | 测试功能 | 测试输入 | 预计输出 | 线性表状态 |
| 1 | IntiaList | 无 | 初始化线性表1成功 | 创建了空表1，表长为0 |
| 2 | ListInsert | 位序1、数据1 | 插入数据进入表1成功 | 表1元素为1，长度为1 |
| 3 | ListTraverse | 无 | 输出表1所有数据 | 表1元素为1，长度为1 |
| 4 | MulList | 输入2 | 切换表成功，当前操作表序为2 | 表1元素为1，长度为1 |
| 5 | IntiaList | 无 | 初始化线性表2成功 | 表1元素为1，长度为1  创建了空表2，表长为0 |
| 6 | OpenFile | 打开准备好的文件file001，其中的元素为10、20、50、60 | 文件读取成功 | 表1元素为1，长度为1  表2元素为10、20、50、60，表长为4 |
| 7 | ListTraverse | 无 | 输出表2所有数据 | 表1元素为1，长度为1  表2元素为10、20、50、60，表长为4 |
| 8 | DestroyList | 无 | 销毁表2成功 | 表1元素为1，长度为1  表2未创建 |
| 9 | ClearList | 无 | 置空表2失败，表2未创建 | 表1元素为1，长度为1  表2未创建 |
| 10 | MulList | 输入1 | 切换表成功，当前操作表序号为1 | 表1元素为1，长度为1  表2未创建 |
| 11 | ListTraverse | 无 | 输出表1所有数据 | 表1元素为1  长度为1 |
| 12 | DestroyList | 无 | 销毁表1 | 线性表不存在 |
| 13 | Exit | 无 | 退出系统 | 线性表不存在 |

### 1.3.4 测试结果

1. 单表操作的测试结果
2. 执行第一步测试，初始化线性表，提示操作成功（图1-13）。

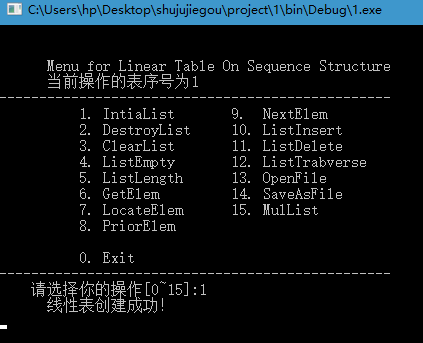


图1-13 初始化表成功

1. 执行第二步测试，向线性表中插入数据（图1-14、15）。

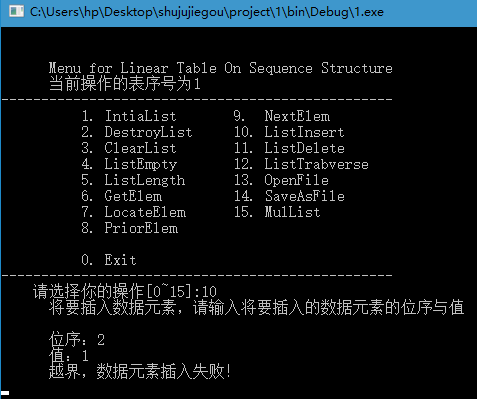


图1-14 插入操作失败，访问越界



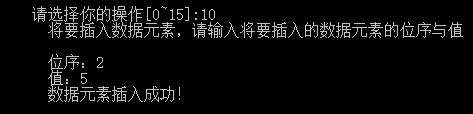


图1-15 插入数据成功

1. 执行第三步测试，分别求数据的前驱（图1-16、17）。

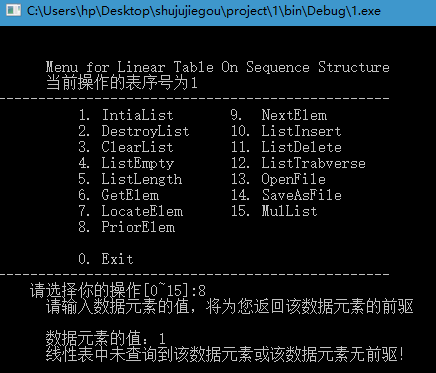


图1-16 数据元素无前驱

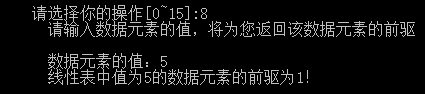


图1-17 成功查询数据前驱

1. 执行第四步测试，分别求数据5、6的位序（图1-18、19）。

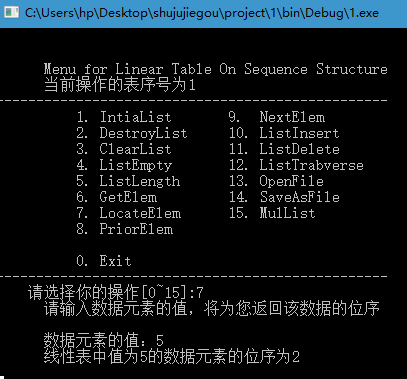


图1-18 成功查询数据位序

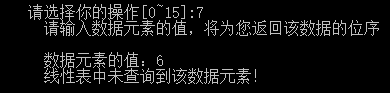


图1-19 未查询到相应数据元素

1. 执行第五步测试，求线性表长度（图1-20）。

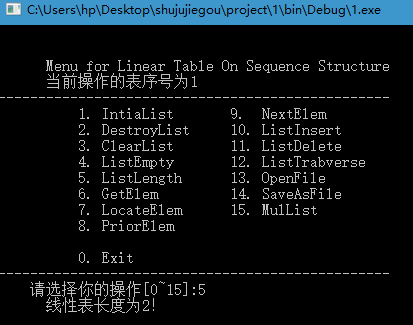


图1-20 求得线性表长为2

1. 执行第六步测试，判断线性表是否为空（图1-21）。



图1-21 判断线性表不为空

1. 执行第七步测试，删除线性表中位序位2的元素（图1-22、23）。

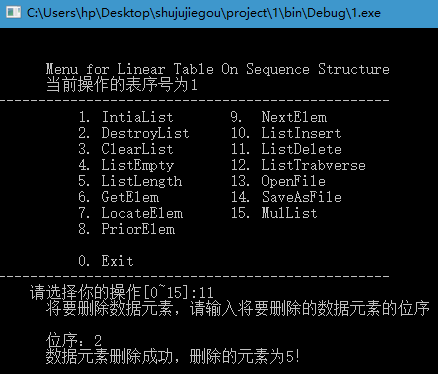


图1-22 删除线性表中元素成功

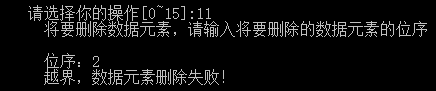


图1-23 删除线性表中元素失败

1. 执行第八步测试，分别获取线性表中位序为2、1的元素（图1-24、25）。

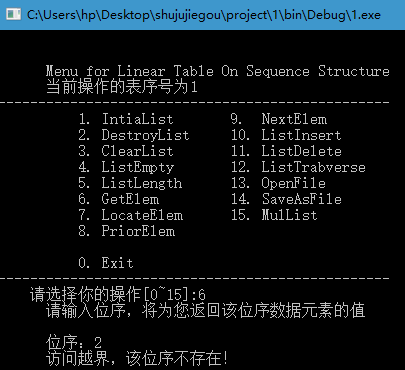


图1-24 访问越界，获取元素失败

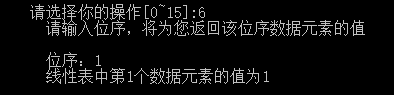


图1-25 成功获取线性表中位序为1的数据值

1. 执行第九步测试，置空线性表（图1-26）。

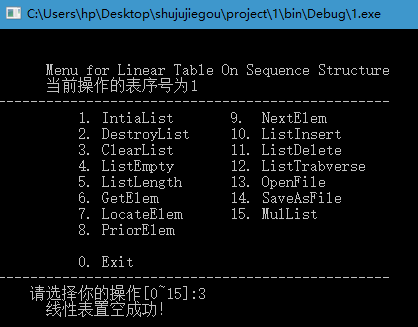


图1-26 线性表置空成功

1. 执行第十步测试，输出线性表中所有数据，表中无数据，提示为空表（图1-27）。



图1-27 输出表中所有数据失败

1. 执行第十一步测试，从文件中读取数据进入线性表（图1-28）。

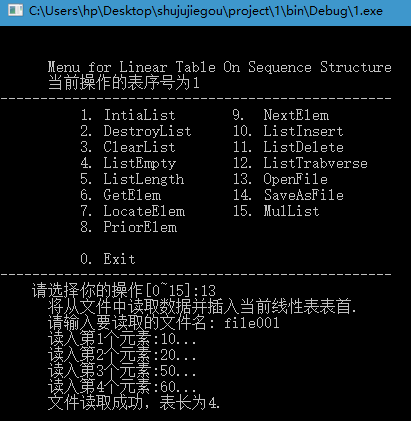


图1-28 从文件中读取数据进入线性表

1. 执行第十二步测试，输出线性表中所有数据（图1-29）。



图1-29 输出线性表中所有数据

1. 执行第十三步测试，分别求数据元素50、60的后继元素（图1-30、31）。

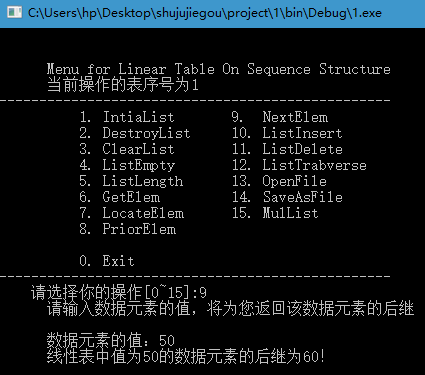


图1-30 求得数据元素后继成功

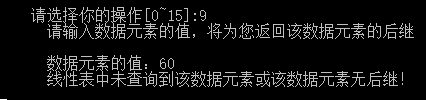


图1-31 求得数据元素后继失败

1. 执行第十四步测试，线性表中数据存储至文件file002中（图1-32、33）。

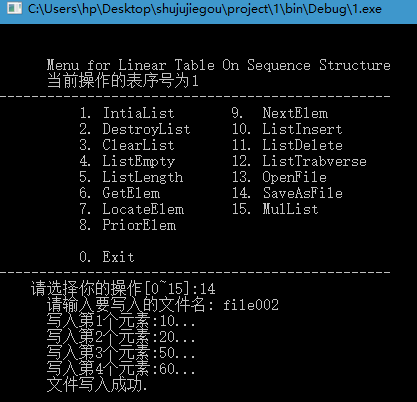


图1-32 将数据存储至文件中

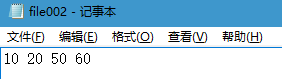


图1-33系统目录下file002内容

1. 执行第十五步测试，销毁线性表（图1-34）。

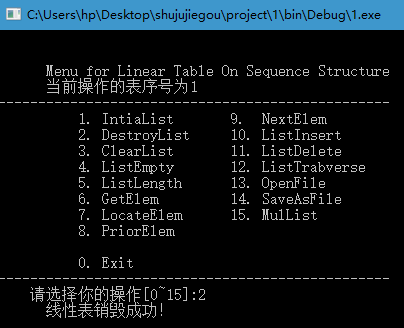


图1-34 线性表销毁成功

1. 执行第十六步测试，退出系统（图1-35）。

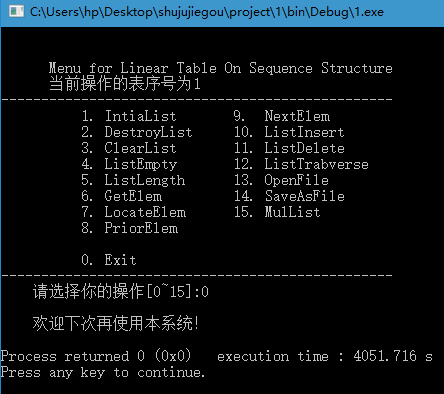


图1-35 退出系统

测试所得程序运行结果均与预计输出相匹配，验证了所编写的程序单线性表操作的正确性。

1. 多表切换操作的测试结果
2. 执行多表测试步骤1、2、3，初始化表1并向表1中插入数据，展示结果如图1-36所示。



图1-36 表1中的所有数据元素

1. 执行多表测试步骤4，切换结果如图1-37所示。



图1-37 切换至表2

1. 执行多表测试步骤5、6、7，初始化表2并向其中导入file001中的数据，展示结果如图1-38所示。

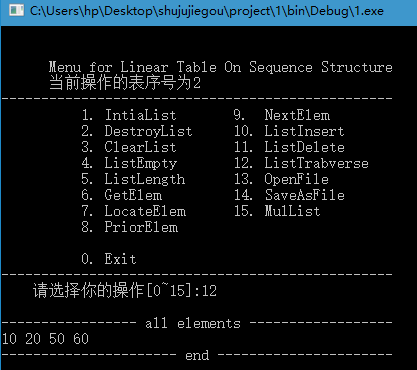


图1-38 表2中的所有数据元素

1. 执行多表测试步骤8、9，销毁表2后置空表2，无法置空证明表2已被销毁，如图1-39所示。

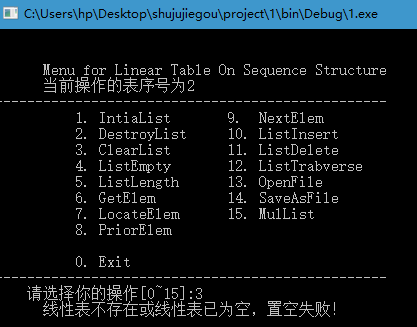


图1-39 表2已被销毁，置空失败

1. 执行多表测试步骤10、11，表1元素仍然可被输出，证明两表操作具有独立性，如图1-40所示。



图1-40 表1数据仍可被输出

1. 执行多表测试步骤12、13，销毁表1并退出系统。

多表测试所得程序运行结果与预计结果相匹配，证明了多表之间的数据插入、文件导入、表销毁等操作不会相互影响，即多表操作具有独立性且切换功能可以正常使用，体现了程序功能的正确性。

## 1.4 实验小结

通过本次的实验我大致遇到了如下的问题并有了以下一些体会。

**在实验中主要遇到的问题有：**

1. 在涉及到多表操作与函数形参设置时，指针的使用是一个比较困难的问题。如在设计多表的结构时，需要使用Sqlist类型的指针实现对静态数组MulList[max]的切换操作，从而能每次仅对其中的某个单线性表进行操作而又能在换表后保留对前一个线性表的操作结果。在此处一开始并没有将L定义为指针，从而产生的操作结果在换表后不能够保存，后在检查语句逻辑时发现了问题并修改；又如在编写各种单线性表操作功能函数时需要根据每个功能的不同需求确定形参的格式，同时需要对函数调用时的实参格式多加注意，才能保证各种数据的传递、取地址、间访等关系。
2. 刚开始编写函数时对struct成员访问的操作比较生疏，编译时产生了如图1-41的报错信息。

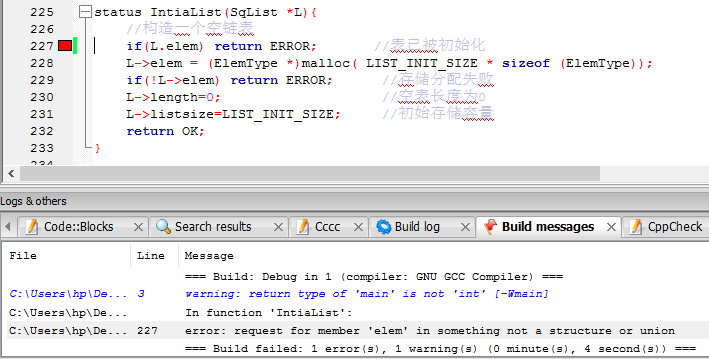


图1-41 对struct成员引用书写错误

分析过后发现两种写法虽然效果近似但是书写格式是不同的，对指针\*L进行操作L->elem与(\*L).elem是等价的，从形式简练的角度出发多采用L->elem的写法，但对于非指针类型的变量L使用->是非法的，仅可以使用L.elem，这个问题也在对C语言的知识的回顾中得到解决。

1. 在后期对程序进行调试时，常出现未初始化时对线性表进行基本操作导致系统死机，如图1-42所示。

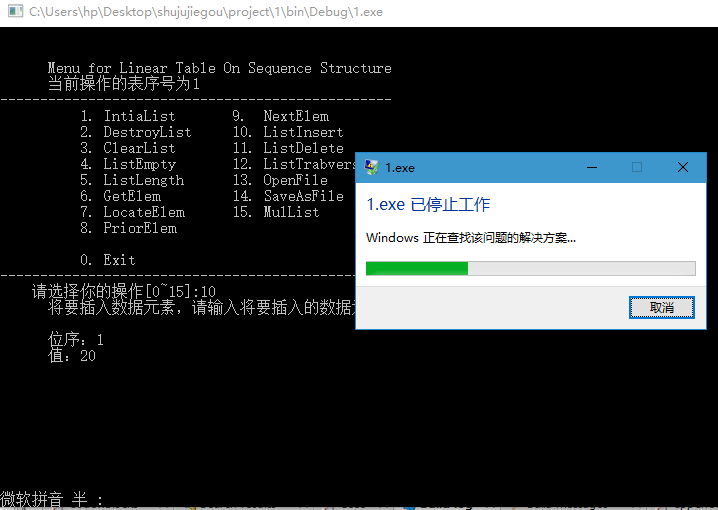


图1-42 未添加判断表是否初始化时直接调用功能，导致死机

分析后得出结论是由于未初始化时L->elem指针为野指针，对其进行操作容易引起不可预料的后果，故需要在子函数内或主函数内添加相应的判断语句，在线性表未初始化时提示相应的错误信息，从而保证系统的健壮性。

**本次实验过后我的体会：**

1. 就C语言而言，指针是其中的精华，也是比较难以掌握的一个知识点，而C语言的灵活性大都通过指针表达出来。就数据结构这一门课程而言，灵活的编程代表着能够通过更多样的方式完成一项功能，从而使其时空复杂度在不同程度上达到更高的要求。从而能够熟练地使用指针的相关操作、理解多重指针的意义并善于使用、联系各种类型的函数时能够正确访问指针并传递需要的值等等便也是在使用C语言理解数据结构时的重中之重。而在此次实验过后我发现自己的编程能力还是十分有限的，需要更多的尝试与锻炼来夯实自己的编程基础。
2. 在本次实验对系统的测试时第一次使用了测试计划表。本以为会增加自己测试系统的工作量，尝试过后发现将理论计划与实际测试分步进行能够更有效的完成系统测试。撰写测试计划表时纯粹的脑力活动逻辑更加严谨；在按表进行测试时也大大降低了各种小失误所带来的影响，同时经过对测试计划的编写与实践，测试计划的正确性也能得到多次确认，最终能够使检测系统的效率大大提升。
3. 在本次实验中首次关注代码的时空复杂度。对于轻量级的程序系统而言代码是否复杂其实影响并不大，但若能熟练对代码复杂度进行分析，在日后完成更复杂的程序或工程时便有了实践之外的另一种检测代码正确性的手段。与此同时程序对内存的使用也是另一个必须关注的方向，善于为程序节省内存，免去不必要的内存占用，及时清理废弃的已用内存，对内存使用有良好的把握是提高程序健壮性重要的一环。

**2基于链式存储结构的线性表实现**

**2.1 问题描述**

为实现基于链式存储结构的线性表及以此为基础的多表管理、文件读写功能，按如下形式设计数据的逻辑结构及基本运算。

**2.1.1 线性表抽象数据类型**

依据最小完备性和常用性相结合的原则，设计了线性表的数据对象和数据关系，并定义了线性表的初始化、销毁、置空、求长度、获取数据元素等14种基本运算，具体数据和运算功能定义如下。

1. 数据结构

数据对象：D = { ai|ai ∈ Elem }

数据关系：R = { <ai-1,ai>|ai-1, ai∈D }

1. 基本运算

初始化表：函数名称是InitaList(\*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。

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

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

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

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

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

查找元素：函数名称是LocateElem(L,e,\*i)；初始条件是线性表已存在；操作结果是用i返回L中第1个数据元素e的位序，若这样的数据元素不存在，则返回值为0。

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

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

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

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

遍历表：函数名称是ListTraverse(L)，初始条件是线性表L已存在；操作结果是依次输出展示L的每个数据元素。

读取文件：函数名称是LoadFile(L,filename)，初始条件是线性表L已存在；操作结果是将名为filename文件中的制定结构数据全部插入L的表首。

写文件：函数名称是SaveAsFile(L,filename)，初始条件是线性表L已存在；操作结果是将L中的数据全部以制定结构存入名为filename的文件。

**2.1.2 多线性表抽象数据类型**

基于2.1.1中已经实现的线性表数据类型与基本运算功能，设计多线性表的数据对象、数据关系。具体数据和多表切换功能定义如下。

1. 数据结构

数据对象：D = { ei|ei ∈ SqList }

数据关系：R = { <ei-1,ei>|ei-1,ei∈D }

1. 基本运算

定义一个名为MulNode的结构体，其成员为表序、单表头指针、next指针。切换操作表时先存储当前操作表，再读取目标操作表，若对应操作表不存在则新建，从而实现多表切换。

**2.1.3 演示系统与数据文件**

构造一个具有文本菜单的功能演示系统，用于演示对顺序表以及多表进行的各项操作。其中，在主函数中完成子函数调用所需的参数输入、结果显示以及对应的文字操作提示。

在设计了相应数据存储格式后，演示系统可进行线性表数据的文件保存或读取，增加了系统合理性。此外系统还可根据需要实现多线性表的切换操作。

**2.2 系统设计**

对数据类型与基本操作进行相应分析后，从物理结构，具体算法等方面合理设计系统，具体设计分析如下。

**2.2.1 数据物理结构**

1. 线性表数据的存储结构

typedef struct LNode{ //顺序表（单链表存储结构）的定义

ElemType data; //线性表数据

struct LNode \*next; //指向下一个线性表数据结点的指针

}LNode, \*LinkList;

1. 多线性表数据的存储结构

typedef struct MulNode{ //顺序表多表结构的定义

int list\_num; //操作表序号

LinkList list\_head; //操作表头指针

struct MulNode \*next; //指向下一个表头指针结点的指针

}MulNode;

多线性表数据的物理存储结构如图2-1所示。

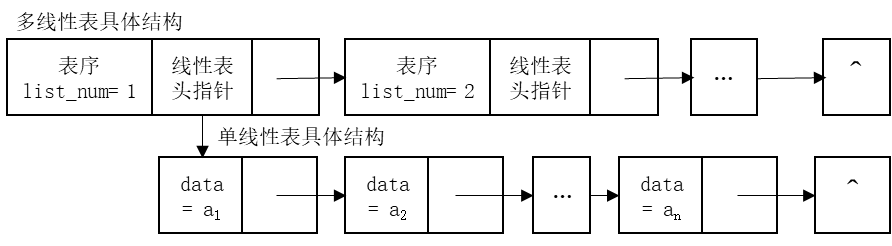


图2-1 多线性表数据的物理存储结构

**2.2.2 演示系统**

演示系统使用基于标准输入输出库（stdio.h）实现的文本界面，由功能菜单界面和用户操作（交互）界面组成。

功能菜单界面以文本形式输出功能菜单，展示当前操作线性表序号以及可选的线性表操作项，供用户参考选择调用需要的功能。

用户操作（交互）界面供用户进行功能的选择、数据的输入等，将用户输入的相关信息传递给对应线性表操作函数，输出调用函数后的操作结果或错误提示等信息。

演示系统具体模块结构如图2-2所示。

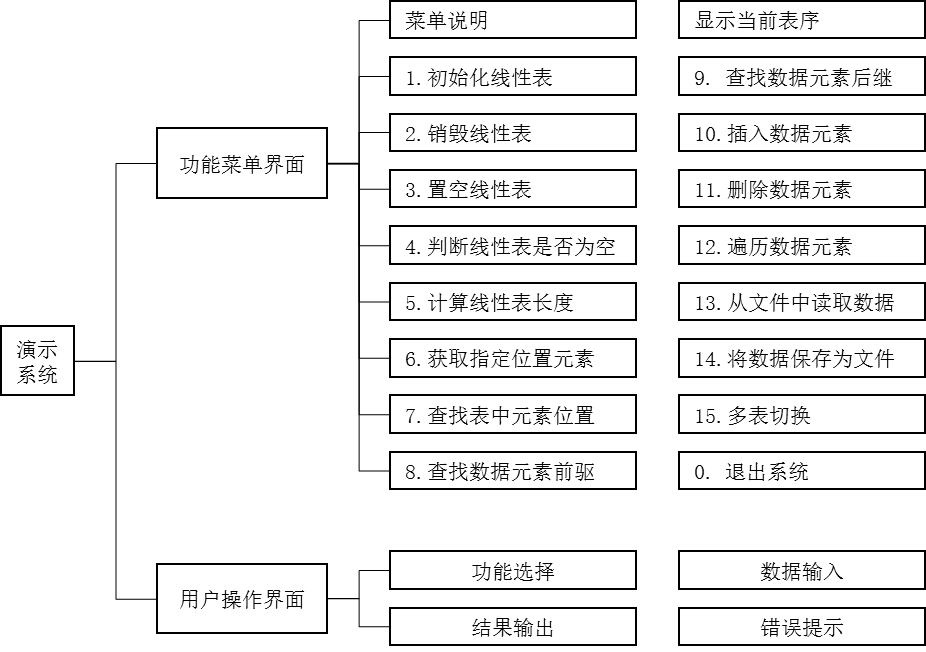


图2-2 演示系统模块结构图

**2.2.3 数据文件存储格式**

考虑到存储的数据文件需具有简洁高效的直观实用的性能，按如下方式设计数据文件存储格式。

文件格式：txt，便于直接查看和修改等。

存放方式：通过fprintf函数顺序存放数据值的十进制字符形式，每两个元素间添加一个空格符以分开。

数据文件存储格式如图2-3所示。

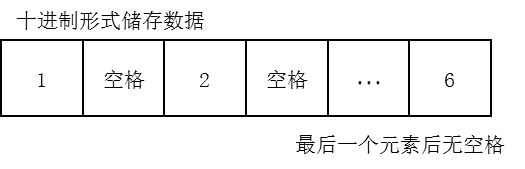


图2-3 数据文件存储格式示例

**2.2.4 线性表运算算法实现**

线性表运算算法及设计如下：

1. 初始化表函数InitaList(\*L)

**算法分析：**首先在主函数中判断L是否为空（判断线性表创建与否），若不为空则提示线性表已被初始化，不执行函数功能；否则为L分配LNode类型大小的存储空间，若分配失败则返回ERROR提示溢出，否则初始化头指针指向的存放数据的头结点为NULL，返回OK。

**操作结果：**构造一个空的线性表。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 销毁表函数DestroyList(\*L)

**算法分析：**首先在主函数中判断L是否为空（判断线性表创建与否），若为空则提示线性表未初始化，不执行函数功能；若不为空，释放包括头指针L在内所有结点所占用的内存空间，并将L置空，销毁线性表，返回OK。

**操作结果：**销毁线性表。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 清空表函数ClearList(\*L)

**算法分析：**首先在主函数中判断L是否为空（判断线性表创建与否），若为空则提示线性表未初始化，不执行函数功能；若不为空，释放从存放数据的头结点L->next开始的所有结点所占用的内存空间并将L->next置空，从而实现置空线性表功能，返回OK。

**操作结果：**置空线性表。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 判定空表函数ListEmpty(L)

**算法分析：**首先在主函数中判断L是否为空（判断线性表创建与否），若为空则提示线性表未初始化，不执行函数功能；若不为空，判断头结点L->next是否为空，若不为空返回OK提示线性表不为空，否则返回ERROR提示是空表。

**操作结果：**判断线性表是否为空。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 求表长函数ListLength(L)

**算法分析：**首先在主函数中判断L是否为空（判断线性表创建与否），若为空则提示线性表未初始化，不执行函数功能；若不为空，通过for循环for(length=0;L->next!=NULL;L=L->next){length++;}遍历数据结点，其中length为计数器，通过length返回表长度。

**操作结果：**计算线性表长度。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

1. 获得元素函数GetElem(L,i,\*e)

**算法分析：**首先在主函数中判断L是否为空（判断线性表创建与否），若为空则提示线性表未初始化，不执行函数功能；若不为空，判断i值是否在线性表合法范围1≤i≤ListLength(L)内，i不合法则返回ERROR提示操作越界，否则通过for循环遍历线性表i次寻找第i个数据结点，用e传递其数据值，返回OK。

**操作结果：**用e返回线性表L中第i个数据元素的值。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

1. 查找元素函数LocateElem(L,e,\*i)

**算法分析：**首先在主函数中判断L是否为空（判断线性表创建与否），若为空则提示线性表未初始化，不执行函数功能；若不为空，通过for循环遍历线性表查找数据元素e第一次出现的位置，若查询到，则用i传递该位置的位序并返回OK，若不存在这样的数据元素e，则返回ERROR提示未查询到对应的数据元素。

**操作结果：**查找线性表L中数据元素e的位序。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**算法流程图：**如图2-4所示。

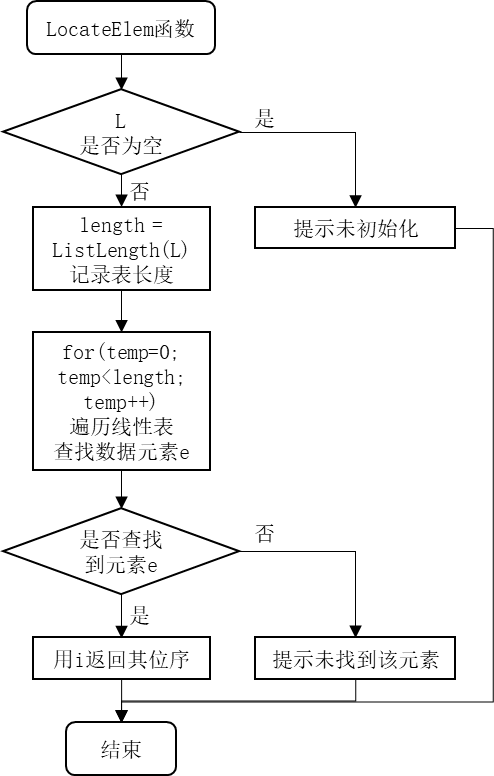


图2-4 查找元素函数LocateElem算法流程图

1. 获得前驱函数PriorElem(L,cur,\*pre\_e)

**算法分析：**首先在主函数中判断L是否为空（判断线性表创建与否），若为空则提示线性表未初始化，不执行函数功能；若不为空，通过LocateElem函数遍历线性表查找数据元素cur第一次出现的位置，若找到且其位序不为1，则通过GetElem函数用pre\_e传递位序为pre\_e-1的该数据元素的前驱元素并返回OK，若不存在这样的数据元素e，则返回ERROR提示未查询到对应的数据元素或对应的数据元素无前驱。

**操作结果：**查找线性表L中数据元素cur的前驱元素pre\_e。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**算法流程图：**如图2-5所示。

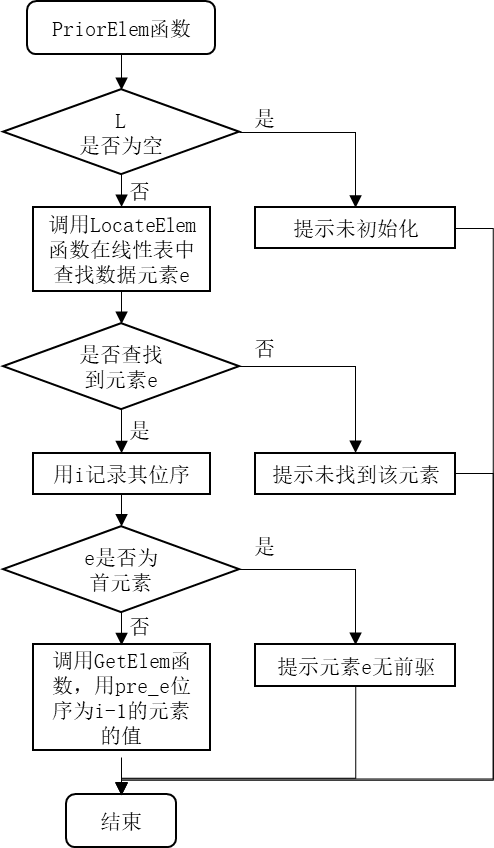


图2-5 获得前驱函数PriotElem算法流程图

1. 获得后继函数NextElem(L,cur,\*next\_e)

**算法分析：**首先在主函数中判断L是否为空（判断线性表创建与否），若为空则提示线性表未初始化，不执行函数功能；否则，通过LocateElem函数遍历线性表查找数据元素cur第一次出现的位置，若找到且其位序不为ListLength(L)，则通过GetElem函数用next\_e传递该数据元素的后继元素并返回OK，若不存在这样的数据元素e，则返回ERROR提示未查询到对应的数据元素或对应的数据元素无后继。

**操作结果：**查找线性表L中数据元素cur的后继元素next\_e。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**算法流程图：**如图2-6所示。

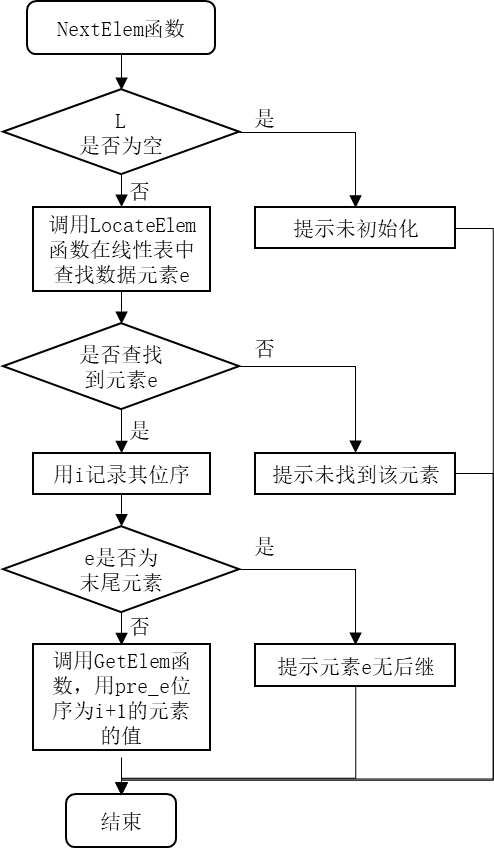


图2-6 获得后继函数NextElem算法流程图

1. 插入元素函数ListInsert(\*L,i,e)

**算法分析：**首先在主函数中判断L是否为空（判断线性表创建与否），若为空则提示线性表未初始化，不执行函数功能；若不为空，判断插入位序i是否属于合法范围1≤i≤ListLength(\*L)+1，若插入位序不在合法范围内，返回ERROR并提示操作越界，否则为待插入结点申请一个LNode类型大小的空间，申请失败则返回OVERFLOW提示分配空间失败，申请成功则用temp\_L指针遍历链表至第i-1个结点，在其后方即位序i处插入新的带有数据元素e的结点，返回OK。

**操作结果：**在线性表L的第i个位置之前插入新的数据元素e。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**算法流程图：**如图2-7所示。

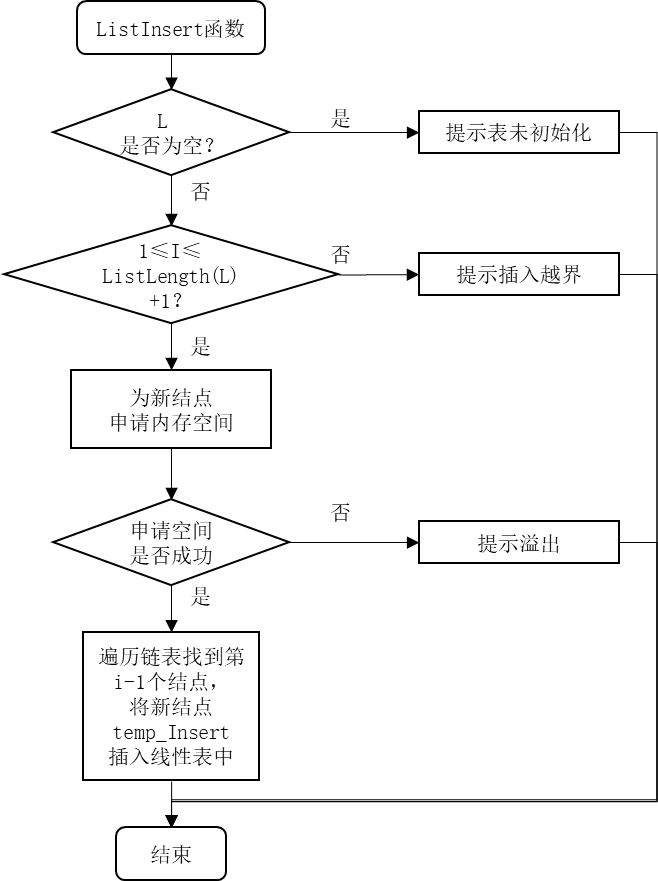


图2-7 插入元素函数ListInsert算法流程图

1. 删除元素函数ListDelete(\*L,i,\*e)

**算法分析：**首先在主函数中判断L是否为空（判断线性表创建与否），若为空则提示线性表未初始化，不执行函数功能；若不为空， 判断待删除位序i是否属于合法范围1≤i≤ListLength(L)，若位序不在合法范围内，返回ERROR并提示操作越界，否则用temp\_L指针遍历线性表找到第i-1个结点，用e传递将被删除的第i个节点上数据元素的值，并将第i个结点从线性表中删除，随后释放删除结点占用的内存，返回OK。

**操作结果：**将线性表第i个数据元素删除并用e返回其值。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**算法流程图：**如图2-8所示。

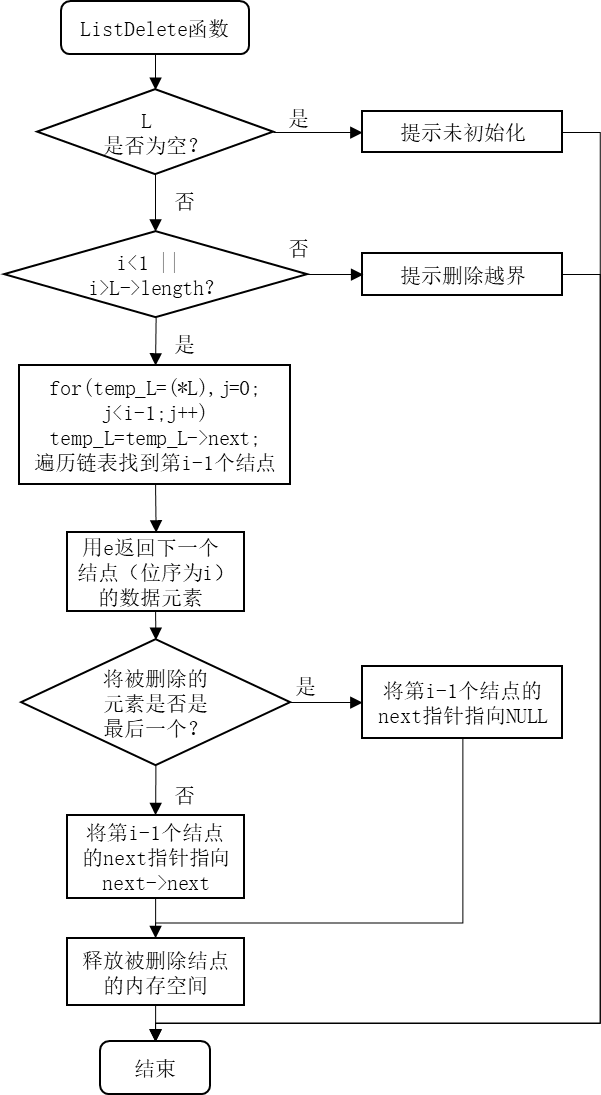


图2-8 删除元素函数ListDelete算法流程图

1. 遍历表函数ListTraverse(L)

**算法分析：**首先在主函数中判断L是否为空（判断线性表创建与否），若为空则提示线性表未初始化，不执行函数功能；若不为空，通过printf绘制分界线，遍历线性表输出展示所有数据元素，每两个元素间间隔一个空格符，返回OK。

**操作结果：**遍历线性表并输出L的每个数据元素。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

1. 读取文件函数LoadFile(L,filename)

**算法分析：**首先在主函数中判断L是否为空（判断线性表创建与否），若为空则提示线性表未初始化，不执行函数功能；若不为空，输出相应信息提示输入文件名称，通过只读方式打开文件，利用fscanf函数从文件中读取规范格式的十进制数据并将其通过ListInsert函数插入当前操作表首部，直至读取到文件尾。读取文件时用i记录读取数据的个数，并同时输出读入的数据信息，读取结束后，关闭文件。若记录读取数据的个数为0，则返回ERROR提示读取失败，否则返回OK提示读取成功。

**操作结果：**将指定文件中十进制数据读入系统并添加在当前操作表表首。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**算法流程图：**如图2-9所示。

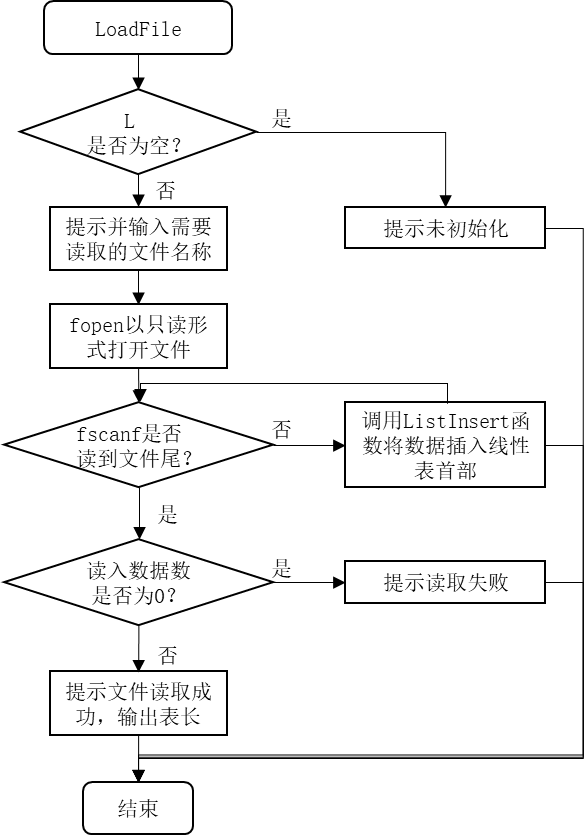


图2-9 文件读取算法流程图

1. 保存文件函数SaveAsFile(L,filename)

**算法分析：**首先在主函数中判断L是否为空（判断线性表创建与否），若为空则提示线性表未初始化，不执行函数功能；若不为空，输出相应信息提示输入文件名称，通过wb+参数以新建或覆盖的方式打开文件，利用fprintf函数向文件中写入规范格式的十进制数据并输出提示当前写入数据的信息，直至写入所有线性表中的数据，关闭文件，返回OK提示写入完毕。

**操作结果：**向指定文件中写入十进制数据。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**算法流程图：**如图2-10所示。

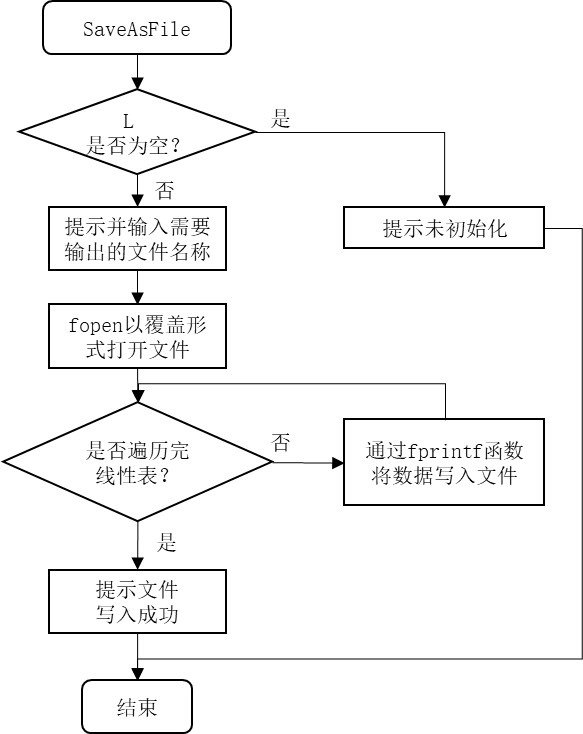


图2-10 文件保存算法流程图

**2.2.5 多线性表切换算法实现**

多表切换函数MulList(\*L,\*M,list\_num\_cur,list\_num)

**算法分析：**数据结构设计中，M为多表结构的头指针，函数参数中list\_num\_cur为当前表序list\_num为将要切换到的表序。在主函数中提示输入要切换到的表序list\_num，先存储当前表的数据：遍历多表结构M中的每个单表头指针，将当前表头指针L赋值给多表结构M中对应表序的头指针；随后切换当前操作表为目标表：再次遍历多表结构M中的每个单表头指针，若查询到对应表序的单表头指针在多表结构M中已经存在，则将其赋值给当前操作表的头指针L，若目标表不存在，则在多表结构M中新建大小为MulNode类型结点，为新结点申请空间失败则返回OVERFLOW提示溢出，否则为新结点赋值，随后用新结点为当前操作表头指针L赋值。

**操作结果：**通过指定序号切换当前操作的线性表。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**2.3 系统实现**

具体系统实现结果如下。

**2.3.1 实验环境**

实验代码使用code::block编写与编译。

**2.3.2 演示系统操作**

演示系统采用键盘操作，控制台输出操作提示与结果。

文本界面显示可选的功能菜单（共有0~15项功能），其中第二行显示当前操作表的序号（默认为1），输入对应操作指令可使用相应功能，输入不在菜单中的操作指令会刷新界面并重新等待输入。演示系统界面如图2-11所示。



图2-11 启动程序后的操作菜单

输入操作指令0，程序退出，如图1-12所示。



图2-12 程序退出

**2.3.3 测试计划**

对系统的线性表操作功能按以下计划测试，其中单表运算操作测试计划如表2-1所示，多表操作功能测试计划如表2-2所示。

表2-1 线性表操作功能测试

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 待测试功能 | 测试顺序 | 测试输入 | 预计输出 | 线性表状态 |
| 1.IntiaList | 1 | 无 | 初始化线性表成功 | 创建了空表，表长为0 |
| 2.DestroyList | 15 | 无 | 销毁线性表成功 | 线性表不存在 |
| 3.ClearList | 9 | 无 | 置空线性表成功 | 线性表为空表 |
| 4.ListEmpty | 6 | 无 | 线性表不为空 | 表中元素为1、5  表长为2 |
| 5.ListLenggth | 5 | 无 | 线性表长度为2 | 表中元素为1、5  表长为2 |
| 6.GetElem | 8 | 分别输入  位序2、  位序1 | 第一次查找越界提示错误信息、  第二次查找得到数据1 | 表中元素为1  表长为1 |
| 7.LocateElem | 4 | 分别输入  数据5、  数据6 | 第一次查找得到位序2、  第二次查找提示错误信息 | 表中元素为1、5  表长为2 |
| 8.PriorElem | 3 | 分别输入  数据1、  数据5 | 第一次查找提示错误信息、  第二次查找前驱数据为1 | 表中元素为1、5  表长为2 |
| 9.NextElem | 13 | 分别输入  数据50、  数据60 | 第一次查找得到后继数据为60、  第二次查找失败提示越界 | 表中元素为10、20、50、60  表长为4 |
| 10.ListInsert | 2 | 分别输入  位序2数据1、  位序1数据1、  位序2数据5 | 第一次插入失败提示越界、  第二个数据插入成功、  第三个数据插入成功 | 表中元素为1、5  表长为2 |
| 11.ListDelete | 7 | 分别输入  位序2、  位序2 | 第一次删除成功返回数据5、  第二次删除失败提示越界 | 表中元素为1  表长为1 |
| 12.ListTraverse | 10、12 | 无 | 输出表中所有数据  第一次调用输出1、  第二次调用输出10、20、50、60 | 第一次调用时  表中元素为1  表长为1、  第二次调用时表中元素为10、20、50、60  表长为4 |
| 13.OpenFile | 11 | 打开准备好的文件file001，其中的元素为10、20、50、60 | 文件读取成功 | 表中元素10、20、50、60，表长为4 |
| 14.SaveAsFile | 14 | 保存为文件file002 | 文件保存成功  文件file002中为十进制数据10 20 50 60 | 表中元素10、20、50、60，表长为4 |
| 0.Exit | 16 | 无 | 退出系统 | 线性表不存在 |

表1-2 多线性表操作功能测试

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试顺序 | 测试功能 | 测试输入 | 预计输出 | 线性表状态 |
| 1 | IntiaList | 无 | 初始化线性表1成功 | 创建了空表1，表长为0 |
| 2 | ListInsert | 位序1、数据1 | 插入数据进入表1成功 | 表1元素为1，长度为1 |
| 3 | ListTraverse | 无 | 输出表1所有数据 | 表1元素为1，长度为1 |
| 4 | MulList | 输入2 | 新建并切换表成功，当前操作表序号为2 | 表1元素为1，长度为1，  表2未初始化 |
| 5 | IntiaList | 无 | 初始化线性表2成功 | 表1元素为1，长度为1  创建了空表2，表长为0 |
| 6 | OpenFile | 打开准备好的文件file001，其中的元素为10、20、50、60 | 文件读取成功 | 表1元素为1，长度为1  表2元素为10、20、50、60，表长为4 |
| 7 | ListTraverse | 无 | 输出表2所有数据 | 表1元素为1，长度为1  表2元素为10、20、50、60，表长为4 |
| 8 | DestroyList | 无 | 销毁表2成功 | 表1元素为1，长度为1  表2未创建 |
| 9 | ClearList | 无 | 置空表2失败，表2未创建 | 表1元素为1，长度为1  表2未创建 |
| 10 | MulList | 输入1 | 切换表成功，当前操作表序号为1 | 表1元素为1，长度为1  表2未创建 |
| 11 | ListTraverse | 无 | 输出表1所有数据 | 表1元素为1  长度为1 |
| 12 | DestroyList | 无 | 销毁表1 | 线性表不存在 |
| 13 | Exit | 无 | 退出系统 | 线性表不存在 |

**2.3.4 测试结果**

1. 单表操作的测试结果
2. 执行第一步测试，初始化线性表，提示操作成功（图2-13）。



图2-13 初始化表成功

1. 执行第二步测试，向线性表中插入数据（图2-14、15）。



图2-14 插入操作失败，访问越界



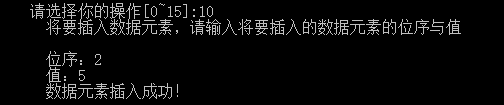


图2-15 插入数据成功

1. 执行第三步测试，分别求数据的前驱（图2-16、17）。



图2-16 数据元素无前驱

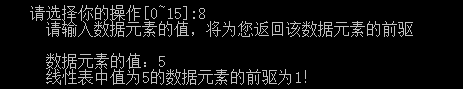


图2-17 成功查询数据前驱

1. 执行第四步测试，分别求数据5、6的位序（图2-18、19）。



图2-18 成功查询数据位序

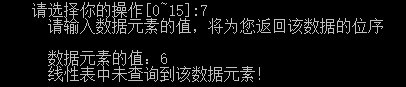


图2-19 未查询到相应数据元素

1. 执行第五步测试，求线性表长度（图2-20）。



图2-20 求得线性表长为2

1. 执行第六步测试，判断线性表是否为空（图2-21）。



图2-21 判断线性表不为空

1. 执行第七步测试，删除线性表中位序位2的元素（图2-22、23）。



图2-22 删除线性表中元素成功

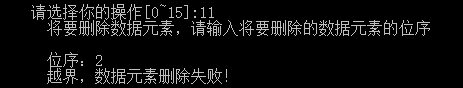


图2-23 删除线性表中元素失败

1. 执行第八步测试，分别获取线性表中位序为2、1的元素（图2-24、25）。



图2-24 访问越界，获取元素失败

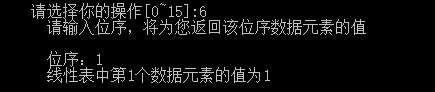


图2-25 成功获取线性表中位序为1的数据值

1. 执行第九步测试，置空线性表（图2-26）。

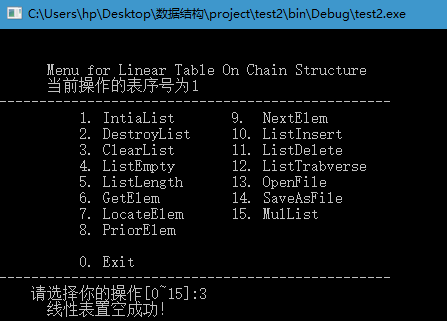


图2-26 线性表置空成功

1. 执行第十步测试，输出线性表中所有数据，表中无数据，提示为空表（图2-27）。



图2-27 输出表中所有数据失败

1. 执行第十一步测试，从文件中读取数据进入线性表（图2-28）。



图2-28 从文件中读取数据进入线性表

1. 执行第十二步测试，输出线性表中所有数据（图2-29）。



图2-29 输出线性表中所有数据

1. 执行第十三步测试，分别求数据元素50、60的后继元素（图2-30、31）。



图2-30 求得数据元素后继成功

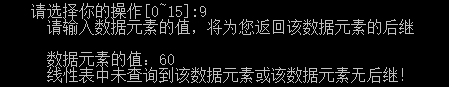


图2-31 求得数据元素后继失败

1. 执行第十四步测试，线性表中数据存储至文件file002中（图2-32、33）。



图2-32 将数据存储至文件中

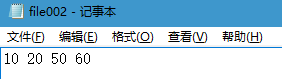


图2-33系统目录下file002内容

1. 执行第十五步测试，销毁线性表（图2-34）。

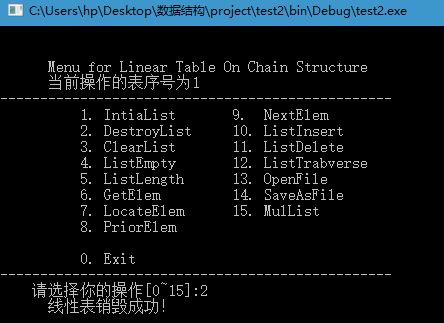


图2-34 线性表销毁成功

1. 执行第十六步测试，退出系统（图2-35）。



图2-35 退出系统

测试所得程序运行结果均与预计输出相匹配，验证了所编写的程序单线性表操作的正确性。

1. 多表切换操作的测试结果
2. 执行多表测试步骤1、2、3，初始化表1并向表1中插入数据，展示结果如图2-36所示。



图2-36 表1中的所有数据元素

1. 执行多表测试步骤4，切换结果如图2-37所示。



图2-37 切换至表2

1. 执行多表测试步骤5、6、7，初始化表2并向其中导入file001中的数据，展示结果如图2-38所示。



图2-38 表2中的所有数据元素

1. 执行多表测试步骤8、9，销毁表2后置空表2，无法置空证明表2已被销毁，如图2-39所示。



图2-39 表2已被销毁，置空失败

1. 执行多表测试步骤10、11，表1元素仍然可被输出，证明两表操作具有独立性，如图2-40所示。



图2-40 表1数据仍可被输出

1. 执行多表测试步骤12、13，销毁表1并退出系统。

多表测试所得程序运行结果与预计结果相匹配，证明了多表之间的数据插入、文件导入、表销毁等操作不会相互影响，即多表操作具有独立性且切换功能可以正常使用，体现了程序功能的正确性。

**2.4 实验小结**

通过本次的实验我大致遇到了如下的问题并有了以下一些体会。

**在实验中主要遇到的问题有：**

1. 在实现多表结构时，由于数据结构类型的定义与实验一中的顺序表有一定的差异，在使用原来的构建方法上遇到了困难：在新的链式存储结构中使用的定义为typedef struct LNode{}\*LinkList，对应的编写单表功能时声明对应头指针时使用的语句为LinkList L。若使用曾使用过的静态数组构建的多表结构，在为静态数组赋值时应赋以当前操作表头指针的地址，即需要更换存储定义为typedef struct LNode{}LNode，同时声明头指针的语句替换为LNode \*L，再对其余语句进行对应的更改。实验过程中为了避免麻烦的修改，也为了尝试其他的实现方式，最后编写了另一种实现多表结构的方式。

经过分析，可在单表操作的基础上添加十字链表结构搭建多表结构，即新建一个多表结构定义typedef struct MulNode{…}MulNode，其结构成员有表序号、对应序号表的头指针、next指针。这样的多表结构能够协助建立十字链表，并且能够方便的实现动态分配多表所占用的内存空间。

1. 编写Locate函数时起初遇到了一些小问题，因为希望代码更加简洁，直接使用了L指针进行链表的遍历，对应的在每经过一个循环后求表长函数ListLength(L)的值会发生变化，所以不能够直接将其作为for循环的判断条件，后使用了int型变量length记录表L的长度，依然使用L指针进行遍历。

解决问题后发现在编写代码的过程中对变量的定义不需要过分吝啬，重复使用同一个变量或符号反而容易导致细节上的小错误。需要学会适当的通过规定变量的名称、增添注释等方式来减少代码出错的可能性。

**本次实验过后我的体会：**

1. 本次实验大体的框架与第一次的实验并无太大区别，系统整体的框架没有进行太大的改动，需要检测的功能与实现的原理大体是相同的，所以使用的测试计划等也没有进行变动。总之遇到的问题比较少，除了多表结构的实现上与之前的实验有较大的区别以外，在其他功能上进行的工作大多是对顺序存储结构的修改与调整，使对应的功能能够基于链表结构运行。

在进行改动的过程中，通过对顺序表与链式表两种线性表的比对，我也从实际操作的层面上更加体会到了两者的共性与差异：对于顺序存储结构的线性表而言，由于其内存连续、支持随机读取，故在需要获得某个结点的前驱与后继的时候能够十分便捷地达到目的；但对应的，由于内存连续，在插入或删除元素时可能需要对多个数据甚至所有数据元素进行移动，时间耗费较高。对于链式存储的线性表而言，首先由于指针本身会占用一定的空间，其对空间的利用效率可能略低于顺序存储，但基于链式存储的特性，其能够更有效的利用零碎的空间。同时链式存储结构可在需要使用内存空间时随时申请，避免了顺序结构中申请过多空间造成浪费的弊端。在完成对应功能上，链式存储的线性表在需要进行结点的增加或删除时十分方便，更改指针的指向即可；但在进行求结点前驱和后继等操作时，由于不可进行随机读取，需要对链表进行遍历，时间耗费较高。此外，在链式存储结构的链表头指针数据域中可存储诸如链表长度、序号等信息，提高对其进行操作的效率。

经过对比可以发现，不同的存储结构各有自己的长处与短处，故而在具体的实际应用中还需要通过对实际问题的分析选择合适的结构进行设计，对问题也要“因地制宜”才是最好的解决方案。

1. 再次使用同一个框架时，对编写过程中的一些细节进行了修改优化，比如在这次的实验代码中，统一将包括文件读写、多表操作在内的所有功能函数的运算部分写入了对应的函数，主函数中仅负责对初始化的判断、对函数所需参数的提示与读取。

经过格式化的修改后，代码的结构更加清晰，可读性得到了提升，同时这种用函数对功能进行分块的写法能够在编写代码遇到问题时更加方便的进行问题的查找与解决，同时每个模块之间的影响较小，在大型工程中也能实现多人合作的效果。

**3基于链式存储结构的二叉树实现**

**3.1 问题描述**

为实现基于链式存储结构的二叉树及以此为基础的森林管理、文件读写功能，按如下形式设计数据的逻辑结构及基本运算。

**3.1.1 二叉树抽象数据类型**

依据最小完备性和常用性相结合的原则，设计了二叉树的数据对象和数据关系，并定义了线性表的初始化、销毁、置空、求二叉树深度、获取数据元素等22种基本运算，具体数据和运算功能定义如下。

1. 数据结构

数据对象：D是具有相同特性的数据元素的集合

数据关系：若D ＝ Φ，则R = Φ，称BinaryTree为空二叉树；

若D ≠ Φ，则R = {H}，H是如下二元关系：

1. 在D中存在唯一的称为根的数据元素root，他在关系H下无前驱；
2. 若D-{root}≠Φ，则存在D-{root}={D1,Dr},且D1∩Dr=Φ；
3. 若D1≠Φ，则D1中存在唯一的元素x1，<root，x1>∈H，且存在D1上的关系H1包含于H；若Dr≠Φ，则Dr中存在唯一的元素xr，<root，xr>∈H，且存在Dr上的关系Hr包含于H；H={<root，x1>，<root，xr>，H1，Hr}；
4. (D1,{H1})是一棵符合本定义的二叉树，称为根的左子树，(Dr，{Hr})是一棵符合本定义的二叉树，成为根的右子树。
5. 基本运算

初始化二叉树：函数名称是InitBiTree(\*T)；初始条件是二叉树T不存在；操作结果是构造空二叉树T。

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

创建二叉树：函数名称是CreateBiTree(\*T,definition)；初始条件是definition 给出二叉树T的定义；操作结果是按definition构造二叉树T。

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

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

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

获得根结点：函数名称是Root(T,e,elem)；初始条件是二叉树T已存在；操作结果是用e和elem返回T的根的key和值。

获得结点：函数名称是Value(T,e,elem)；初始条件是二叉树T已存在，e是T中的某个结点；操作结果是用elem返回e的值。

结点赋值：函数名称是Assign(\*T,&e,value)；初始条件是二叉树T已存在，e是T中的某个结点；操作结果是结点e赋值为value。

获得双亲结点：函数名称是Parent(T,e) ；初始条件是二叉树T已存在，e是T中的某个结点；操作结果是若e是T的非根结点，则返回它的双亲结点指针，否则返回NULL。

获得左孩子结点：函数名称是LeftChild(T,e)；初始条件是二叉树T存在，e是T中某个节点；操作结果是返回e的左孩子结点指针。若e无左孩子，则返回NULL。

获得右孩子结点：函数名称是RightChild(T,e)；初始条件是二叉树T已存在，e是T中某个结点；操作结果是返回e的右孩子结点指针。若e无右孩子，则返回NULL。

获得左兄弟结点：函数名称是LeftSibling(T,e)；初始条件是二叉树T存在，e是T中某个结点；操作结果是返回e的左兄弟结点指针。若e是T的左孩子或者无左兄弟，则返回NULL。

获得右兄弟结点：函数名称是RightSibling(T,e)；初始条件是二叉树T已存在，e是T中某个结点；操作结果是返回e的右兄弟结点指针。若e是T的右孩子或者无有兄弟，则返回NULL。

插入子树：函数名称是InsertChild(\*T,e,LR)；初始条件是二叉树T存在，e是T中的某个结点，LR为0或1；操作结果是根据LR为0或者1，创建一个与T不相交且右子树为空的非空子树，将其插入为T中结点e的左或右子树，e结点的原有左子树或右子树则为c的右子树。

删除子树：函数名称是DeleteChild(\*T,e,LR)；初始条件是二叉树T存在，e是T中的某个结点，LR为0或1；操作结果是根据LR为0或者1，删除T中e结点的左或右子树。

前序遍历：函数名称是PreOrderTraverse(T)；初始条件是二叉树T存；操作结果是先序遍历t，对每个结点输出结点数据，一旦输出失败，则操作失败。

中序遍历：函数名称是InOrderTraverse(T)；初始条件是二叉树T存在；操作结果是中序遍历t，对每个结点输出结点数据，一旦输出失败，则操作失败。

后序遍历：函数名称是PostOrderTraverse(T)；初始条件是二叉树T存在；操作结果是后序遍历t，对每个结点输出结点数据，一旦输出失败，则操作失败。

按层遍历：函数名称是LevelOrderTraverse(T)；初始条件是二叉树T存在；操作结果是层序遍历t，对每个结点输出结点数据，一旦输出失败，则操作失败。

读取文件：函数名称是LoadFile(\*T,filename)，初始条件是二叉树T已存在；操作结果是将名为filename文件中的指定结构数据覆盖至当前操作树。

写文件：函数名称是SaveAsFile(T,filename)，初始条件是二叉树T已存在；操作结果是将当前操作树T中的数据全部以指定结构存入名为filename的文件。

**3.1.2 森林抽象数据类型**

基于3.1.1中已经实现的二叉树数据类型与基本运算功能，设计多树(森林)的数据对象、数据关系。具体数据和森林切换功能定义如下。

1. 数据结构

数据对象：D = { ei|ei ∈ SqList }

数据关系：R = { <ei-1,ei>|ei-1,ei∈D }

1. 基本运算

定义一个名为MulNode的结构体，其成员为树序号、单个二叉树头指针、next指针。切换操作树时先存储当前二叉树，再读取目标二叉树，若对应二叉树不存在则新建，从而实现森林内树的切换。

**3.1.3 演示系统与数据文件**

构造一个具有文本菜单的功能演示系统，用于演示对二叉树以及森林进行的各项操作。其中，在主函数中完成子函数调用所需的参数输入、结果显示以及对应的文字操作提示。

在设计了相应数据存储格式后，演示系统可进行二叉树数据的文件保存或读取，增加了系统合理性。此外系统还可根据需要实现森林内二叉树的切换操作。

**3.2 系统设计**

对数据类型与基本操作进行相应分析后，从物理结构，具体算法等方面合理设计系统，具体设计分析如下。

**3.2.1 数据物理结构**

1. 二叉树数据的存储结构

typedef struct BiTNode{ //二叉树链式存储结构的定义

int key; //结点的唯一标识

ElemType data; //结点的数据

struct BiTNode \*lchild, \*rchild; //指向结点左右孩子的指针

}BiTNode, \*BiTree;

1. 用于层次遍历二叉树的队列存储结构

typedef struct seqqueue{//队列定义

BiTree data[MAX]; //队列大小

int front; //队列头

int rear; //队列尾

}seqqueue;

1. 森林数据的存储结构

typedef struct MulNode{ //二叉树森林结构的定义

int list\_num; //操作表序号

BiTree list\_head; //操作表头指针

struct MulNode \*next; //指向下一个表头指针结点的指针

}MulNode;

森林数据的物理存储结构如图3-1所示。

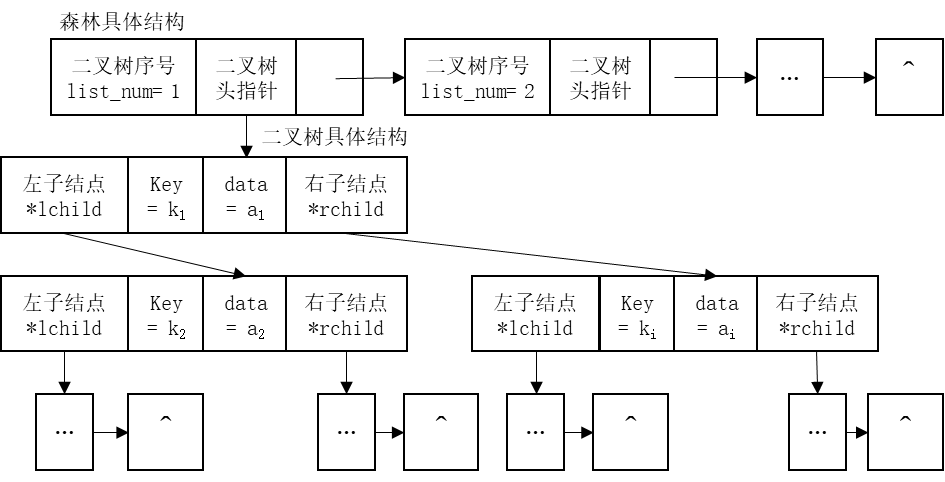


图3-1 森林数据的物理存储结构

**3.2.2 演示系统**

演示系统使用基于标准输入输出库（stdio.h）实现的文本界面，由功能菜单界面和用户操作（交互）界面组成。

功能菜单界面以文本形式输出功能菜单，展示当前操作树序号以及可选的树操作项，供用户参考选择调用需要的功能。

用户操作（交互）界面供用户进行功能的选择、数据的输入等，将用户输入的相关信息传递给对应树操作函数，输出调用函数后的操作结果或错误提示等信息。

其中演示系统的可选功能有：

初始化、摧毁、创建、置空二叉树；判断二叉树是否为空；求取二叉树深度、根结点信息、任意结点信息；二叉树结点赋值；求取结点的双亲结点、左右孩子结点、左右兄弟结点；插入、删除子树；先序、中序、后序、层次遍历二叉树；读取、存储文件。

演示系统具体模块结构如图3-2所示。

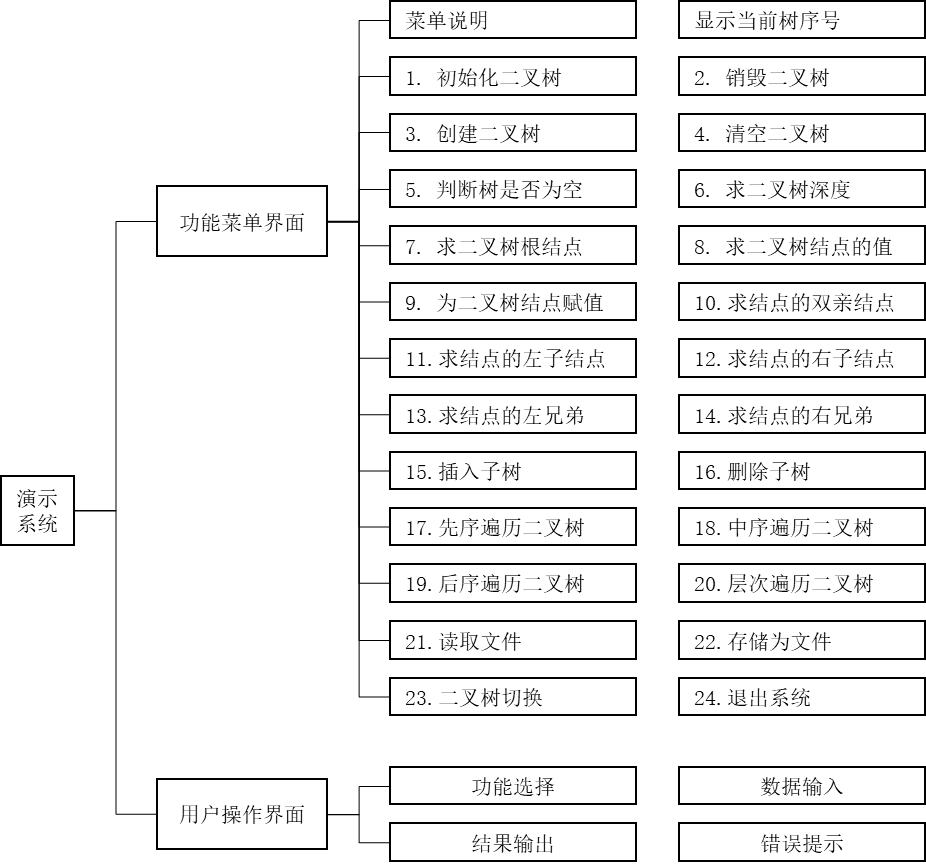


图3-2 演示系统模块结构图

**3.2.3 数据文件存储格式**

考虑到存储的数据文件需具有简洁高效的直观实用的性能，按如下方式设计数据文件存储格式。

文件格式：txt，便于直接查看和修改等。

存放方式：通过fprintf函数将二叉树还原为definition序列存入文件。

数据文件存储格式如图3-3所示。

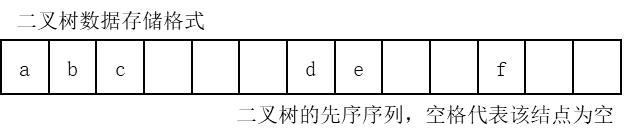


图3-3 数据文件存储格式示例

**3.2.4 二叉树运算算法实现**

二叉树运算算法及设计如下：

1. 初始化树函数InitBiTree(\*T)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若不为空则提示二叉树已被初始化，不执行函数功能；否则为T分配BiTNode类型大小的存储空间，若分配失败则返回ERROR提示溢出，否则初始化头指针指向的存放数据的头结点为NULL，返回OK。

**操作结果：**构造一个空的二叉树。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 销毁树函数DestroyBiTree(\*T)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，释放包括头指针T在内所有结点所占用的内存空间，并将T置空，销毁二叉树，返回OK。

**操作结果：**销毁二叉树。

**时间复杂度分析：**常量阶O(n)。

**空间复杂度分析：**常量阶O(1)。

1. 创建树函数CreatBiTree(\*T)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，系统输出相关信息，提示输入definition序列用于创建二叉树，若创建成功，返回OK；否则创建失败输出相应错误提示信息并返回ERROR。

**操作结果：**创建二叉树。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 置空树函数ClearBiTree(\*T)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，释放从存放数据的头结点T->lchild开始的所有结点所占用的内存空间并将T->lchild置空，从而实现置空二叉树功能，返回OK。

**操作结果：**置空二叉树。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 判定空树函数BiTreeEmpty(T)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，判断头结点T->lchild是否为空，若不为空返回OK提示二叉树不为空，否则返回ERROR提示是空树。

**操作结果：**判断二叉树是否为空。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 求二叉树最大深度函数BiTreeDepth(T)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，通过递归比较返回二叉树最大深度。

**操作结果：**计算二叉树最大深度。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

1. 获得二叉树根节点函数Root(T,\*key,\*elem)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，头结点T-lchild是否为空，若头结点为空，提示二叉树为空、当前二叉树无根结点；否则通过用key和elem分别传递根结点的key和data，返回OK。

**操作结果：**用key和elem返回二叉树T中根节点的数据。

**时间复杂度分析：**线性阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 查找结点函数Value(T,key,\*elem)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，通过递归遍历二叉树查找关键字为key的结点，若查询到，则用elem传递该结点的数据并返回OK，若不存在这样的结点，则返回ERROR提示未查询到对应的关键字为key的结点。

**操作结果：**查找二叉树T中关键字为key的结点的数据。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图3-4所示。

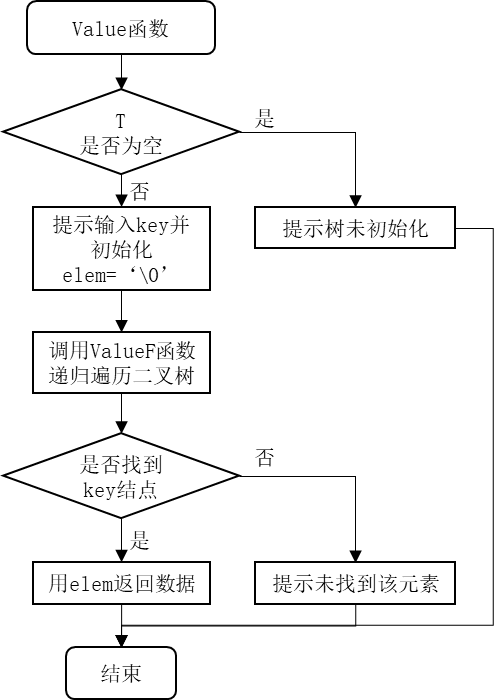


图3-4 查找结点函数Value算法流程图

1. 赋值结点函数Assign(\*T,key,elem)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，通过递归遍历二叉树查找关键字为key的结点，若查询到，则将该结点的数据赋值为elem并返回OK，若不存在这样的结点，则返回ERROR提示未查询到对应的关键字为key的结点。

**操作结果：**对二叉树T中关键字为key的结点的数据域进行赋值。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：** 如图3-5所示。

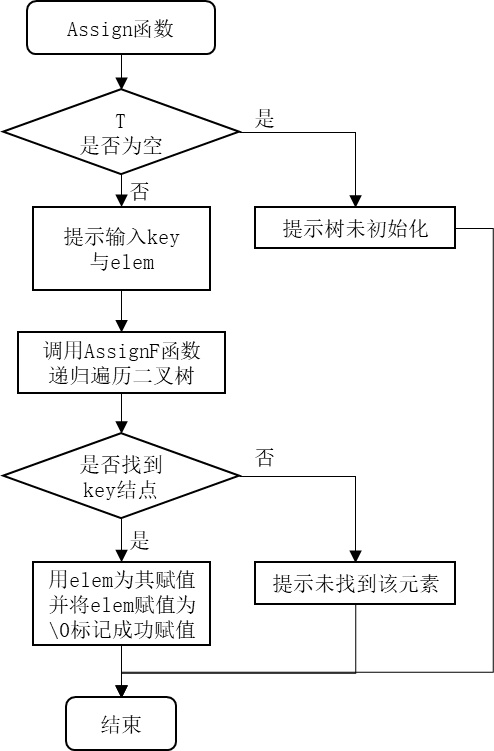


图3-5 赋值结点函数Assign算法流程图

1. 获得双亲结点函数Parent(T,head\_key,key)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，通过递归遍历二叉树，查找有左或右孩子，且其左或右孩子的关键字key为所查找key的结点，若找到，则返回指向这个双亲结点的指针；若不存在这样的双亲结点，则返回NULL提示未查询到对应的结点或关键字为key的结点无双亲结点。

**操作结果：**查找二叉树T中关键字为key的结点的双亲结点。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图3-6所示。

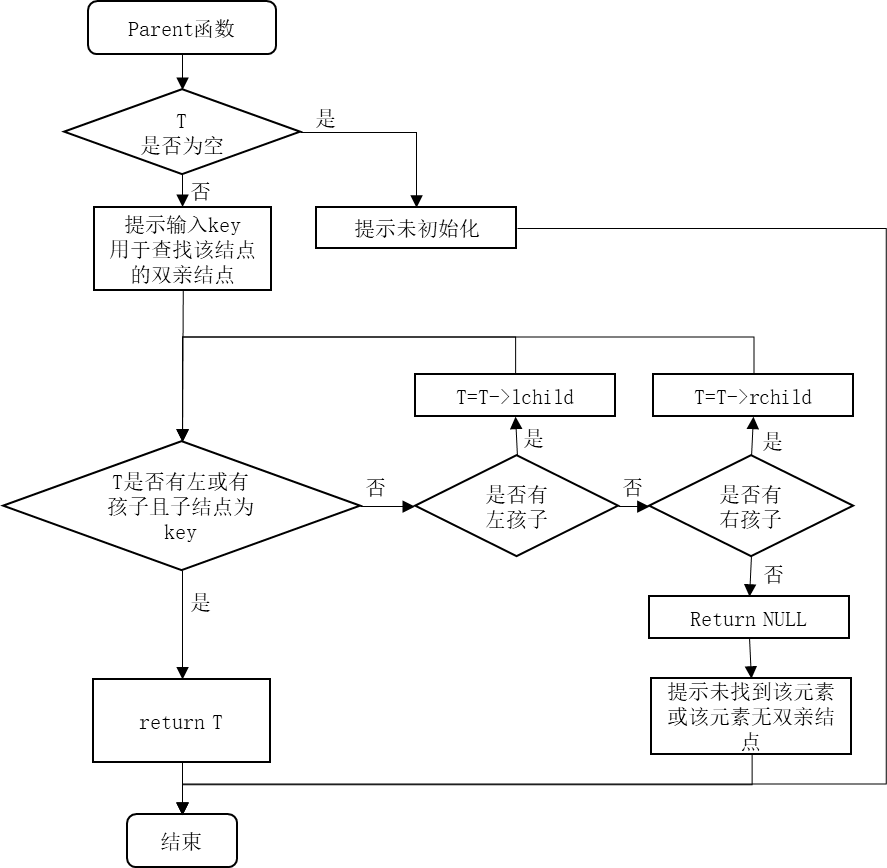


图3-6 获得双亲结点函数Parent算法流程图

1. 获得左孩子结点函数LeftChild(T,key)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，通过递归遍历二叉树，查找关键字为key的结点，若找到，则返回其左孩子指针（若左孩子指针为空，返回NULL并提示左孩子结点为空）；若不存在这样的结点，则返回NULL提示未查询到对应的结点。

**操作结果：**查找二叉树T中关键字为key的结点的左孩子结点。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图3-7所示。

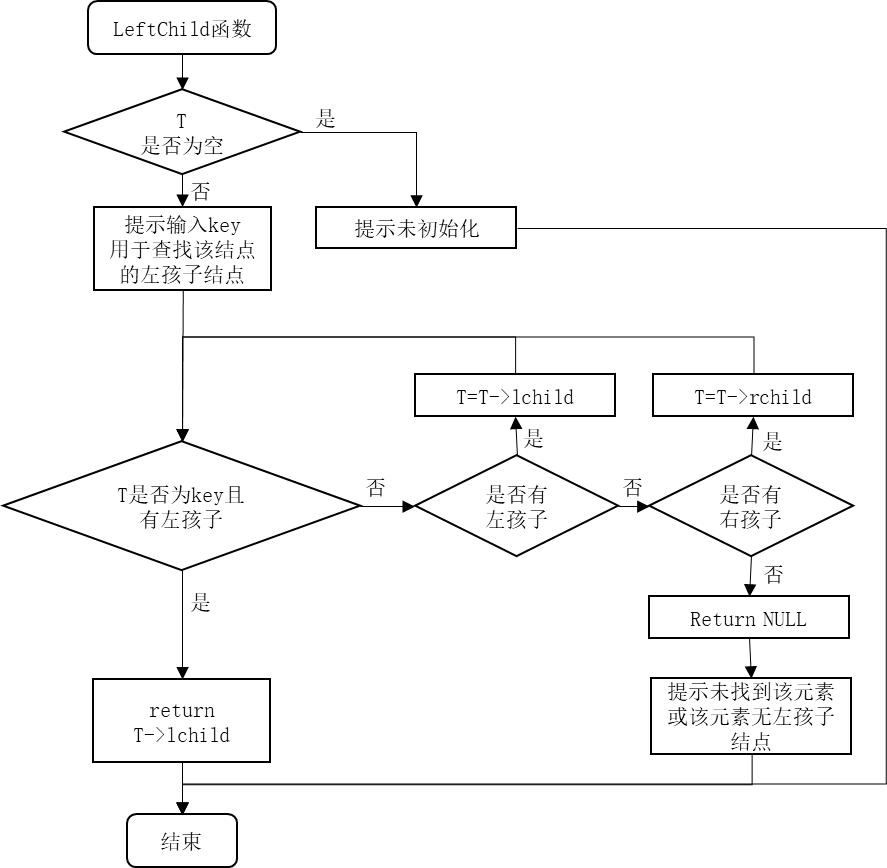


图3-7 获得左孩子结点函数LeftChild算法流程图

1. 获得右孩子结点函数RightChild(T,key)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，通过递归遍历二叉树，查找关键字为key的结点，若找到，则返回其右孩子指针（若右孩子指针为空，返回NULL并提示右孩子结点为空）；若不存在这样的结点，则返回NULL提示未查询到对应的结点。

**操作结果：**查找二叉树T中关键字为key的结点的右孩子结点。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图3-8所示。

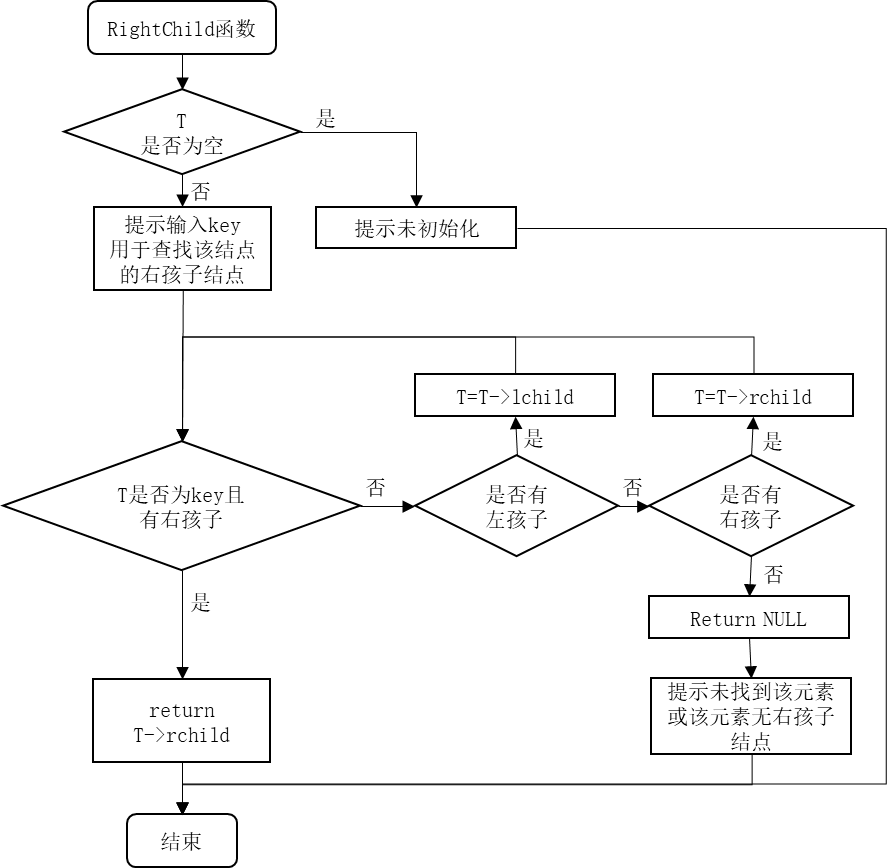


图3-8 获得右孩子结点函数RightChild算法流程图

1. 获得左兄弟结点函数LeftSibling(T,head\_key,key)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，通过递归遍历二叉树，查找右孩子的关键字key为所查找key的结点，若找到，则返回指向这个结点的左孩子指针（若这个结点的左孩子指针为空，返回NULL并提示所查找结点的左兄弟结点为空）；若不存在这样的结点，则返回NULL提示未查询到对应的结点。

**操作结果：**查找二叉树T中关键字为key的结点的左兄弟结点。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图3-9所示。

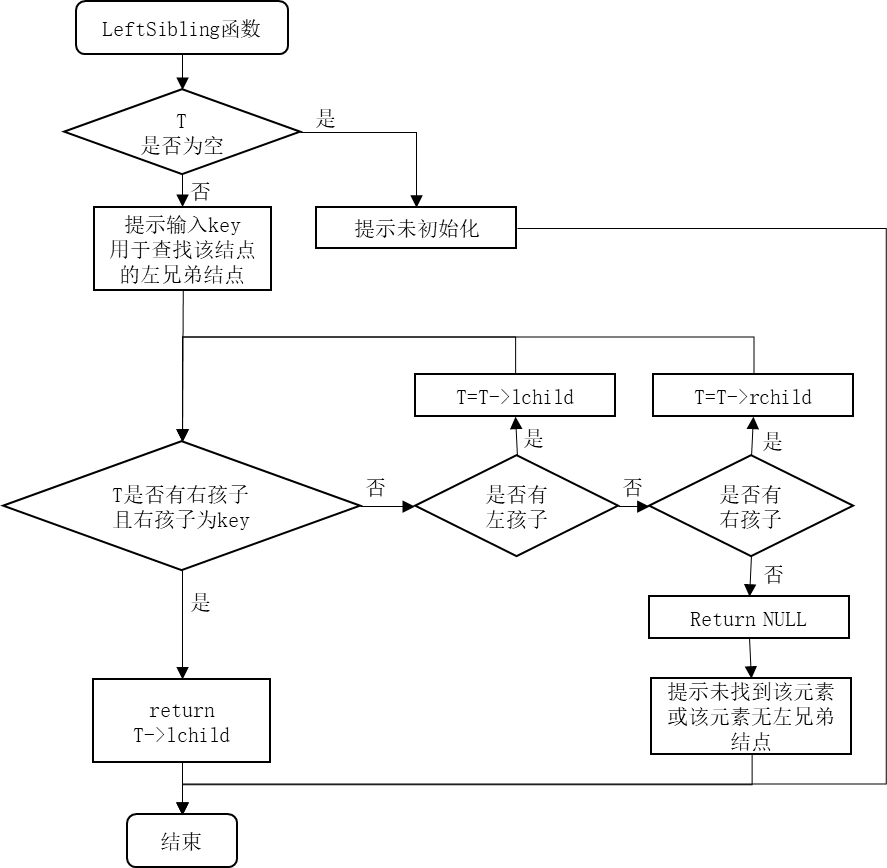


图3-9 获得左兄弟结点函数LeftSibling算法流程图

1. 获得右兄弟结点函数RightSibling(T,head\_key,key)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，通过递归遍历二叉树，查找左孩子的关键字key为所查找key的结点，若找到，则返回指向这个结点的右孩子指针（若这个结点的右孩子指针为空，返回NULL并提示所查找结点的右兄弟结点为空）；若不存在这样的结点，则返回NULL提示未查询到对应的结点。

**操作结果：**查找二叉树T中关键字为key的结点的右兄弟结点。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图3-10所示。

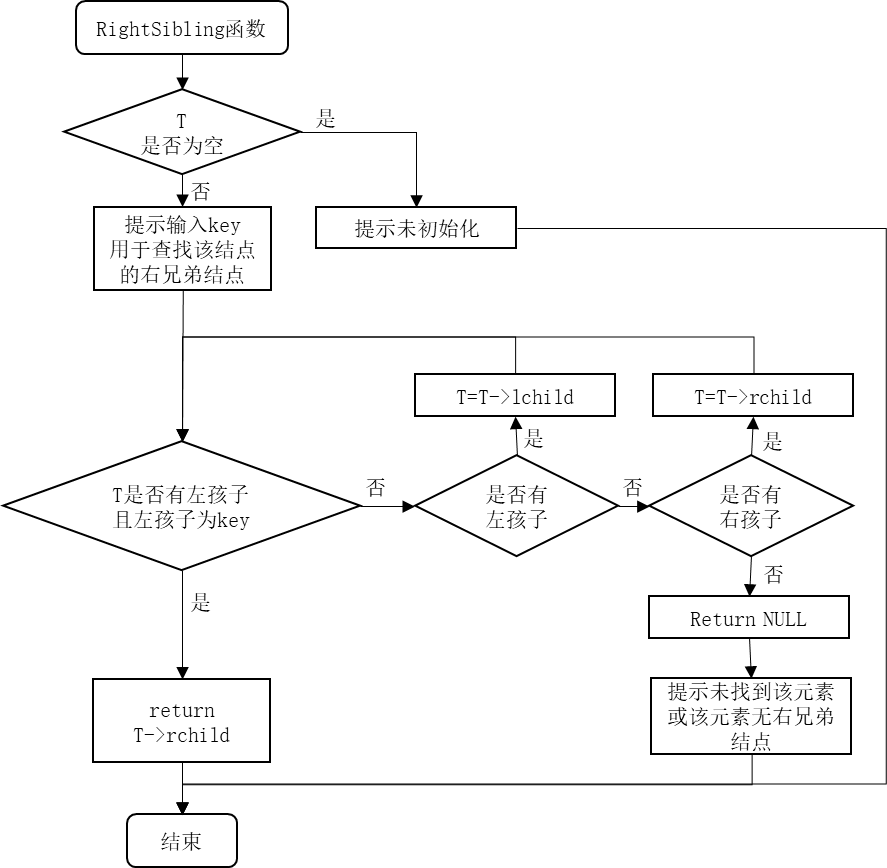


图3-10 获得右兄弟结点函数RightSibling算法流程图

1. 插入子树函数InsertChild(\*T,key,LR)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；同时在主函数中判断LR是否为0或1，若不为0或1则提示输入LR有误，不执行函数。若不为空，首先提示输入用于创建子树的definition序列：若输入序列为空，提示创建失败并返回ERROR；若创建子树根结点含右结点，提示插入子树格式有误并返回ERROR。若创建子树成功，随后通过NODE函数遍历二叉树寻找关键字为key的结点，若未查询到对应节点，提示未找到关键字为kye的结点并返回ERROR；若找到对应节点，根据LR的值为0或1将子树按照指定规则插入为左子树或右子树，返回OK。

**操作结果：**在二叉树T中关键字为key的结点下插入左或右子树。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图3-11所示。

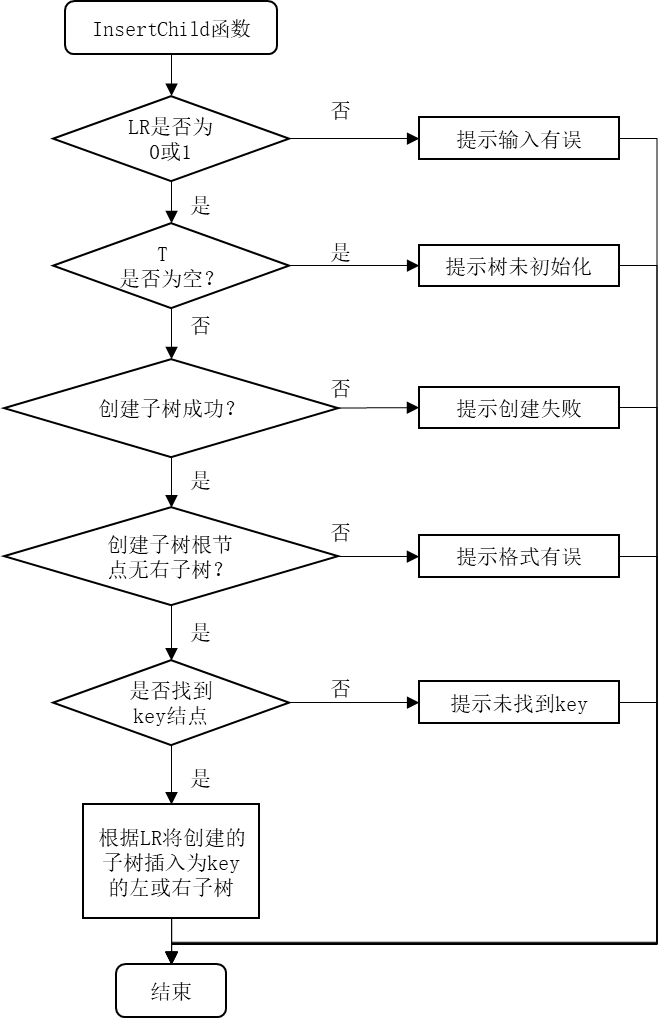


图3-11 插入子树函数InsertChild算法流程图

1. 删除子树函数DeleteChild(\*T,key,LR)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；同时在主函数中判断LR是否为0或1，若不为0或1则提示输入LR有误，不执行函数。若不为空，提示输入要删除的子树的双亲结点的关键字key，随后输入LR选择删除其双亲结点的左子树还是右子树。若未查询到关键字为key的结点，提示未找到结点的错误信息并返回ERROR；否则找到双亲结点，根据LR的值尝试删除其左或右子树，若左或右子树已为空，提示对应子树已为空并返回ERROR，否则删除子树、释放子树内存并返回OK。

**操作结果：**将二叉树T中关键字为key的结点的左或右子树删除。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图3-12所示。

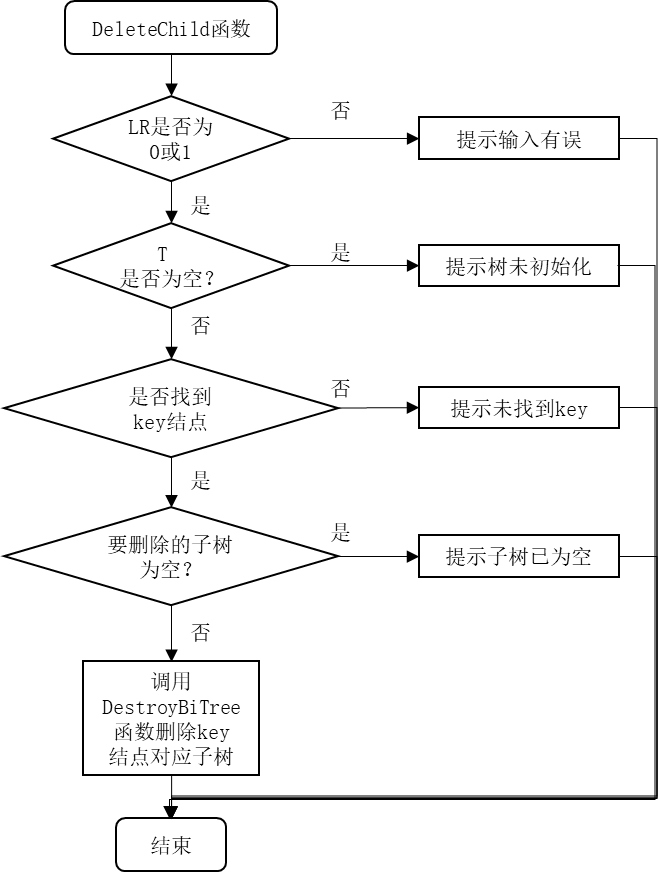


图3-12 删除子树函数DeleteChild算法流程图

1. 先序遍历树函数PreOrderTraverse(T)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，通过printf绘制分界线，递归先序遍历二叉树输出展示所有结点的数据，一行输出一个结点的key与data，输出结束后返回OK。

**操作结果：**先序遍历并输出二叉树的每个结点的数据。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

1. 中序遍历树函数InOrderTraverse(T)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，通过printf绘制分界线，递归中序遍历二叉树输出展示所有结点的数据，一行输出一个结点的key与data，输出结束后返回OK。

**操作结果：**中序遍历并输出二叉树的每个结点的数据。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

1. 后序遍历树函数PostOrderTraverse(T)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，通过printf绘制分界线，递归后序遍历二叉树输出展示所有结点的数据，一行输出一个结点的key与data，输出结束后返回OK。

**操作结果：**后序遍历并输出二叉树的每个结点的数据。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

1. 层次遍历树函数LevelOrderTraverse(T)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，通过printf绘制分界线，定义队列结构，由头结点开始将子结点存入队列，结点出队列时输出其中数据并将其子结点存入队列，所有节点进出队列并输出完成后返回OK。

**操作结果：**层次遍历并输出二叉树的每个结点的数据。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图3-13所示。

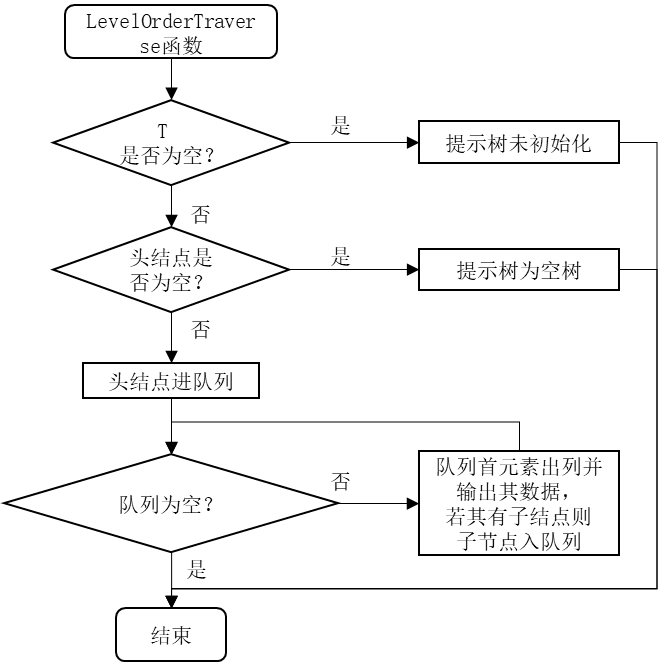


图3-13 层次遍历树函数LevelOrderTraverse算法流程图

1. 读取文件函数LoadFile(\*T,filename)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，输出相应信息提示输入文件名称，通过只读方式打开文件，利用fscanf函数从文件中读取规范格式的definition序列并通过CreatBiTree函数将其覆盖至当前操作表。读取完毕后关闭文件。若definition序列不能够创建二叉树则返回ERROR提示读取失败，否则返回OK提示读取成功。

**操作结果：**将指定文件中字符序列读入系统并利用其创建二叉树覆盖当前已有二叉树。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图3-14所示。

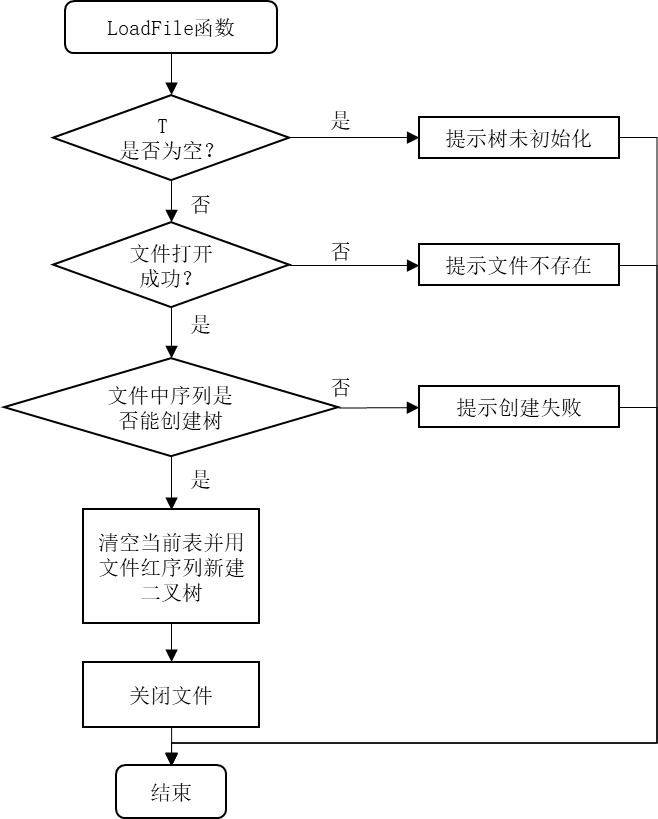


图3-14 读取文件函数LoadFile算法流程图

1. 保存文件函数SaveAsFile(T,filename)

**算法分析：**首先在主函数中判断T是否为空（判断二叉树创建与否），若为空则提示二叉树未初始化，不执行函数功能；若不为空，输出相应信息提示输入文件名称，通过wb+参数以新建或覆盖的方式打开文件，调用Save\_PreOrderTraverse函数利用二叉树生成definition序列并存入字符串save\_definition，生成后利用fprintf函数向文件中写入规范格式的字符串，关闭文件，返回OK提示写入完毕。

**操作结果：**向指定文件中写入definition字符序列。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图3-15所示。

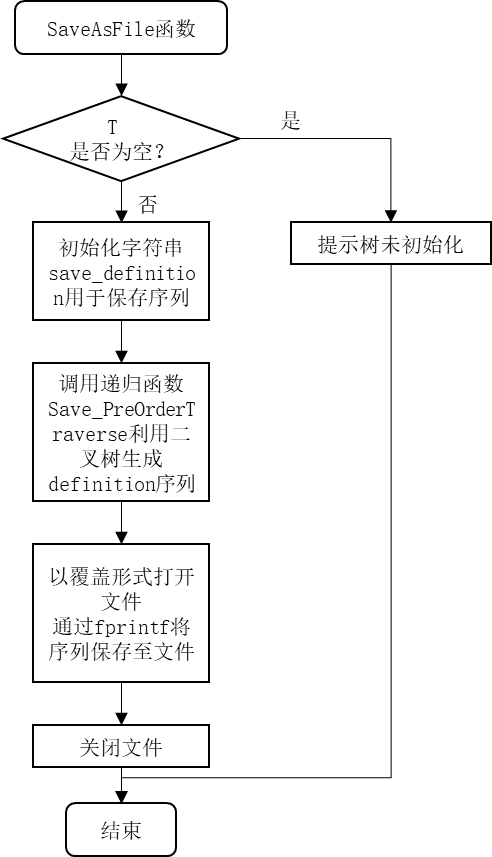


图3-15 保存文件函数SaveAsFile算法流程图

**3.2.5 森林切换算法实现**

森林切换函数MulList(\*T,\*M,list\_num\_cur,list\_num)

**算法分析：**数据结构设计中，M为森林结构的头指针，函数参数中list\_num\_cur为当前树序号、list\_num为将要切换到的树序号。在主函数中提示输入要切换到的树序号list\_num，先存储当前树的数据：遍历森林结构M中的每个二叉树头指针，将当前二叉树头指针T赋值给森林结构M中对应树序号的头指针；随后切换当前操作树为目标树：再次遍历森林结构M中的每个二叉树头指针，若查询到对应树序号的单表头指针在森林结构M中已经存在，则将其赋值给当前操作树的头指针T，若目标树不存在，则在森林结构M中新建大小为MulNode类型结点，为新结点申请空间，失败则返回OVERFLOW提示溢出，否则为新结点赋值，随后用新结点为当前操作树头指针T赋值。

**操作结果：**通过指定序号切换当前操作的二叉树。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**3.3 系统实现**

具体系统实现结果如下。

**3.3.1 实验环境**

实验代码使用code::block编写与编译。

**3.3.2 演示系统操作**

演示系统采用键盘操作，控制台输出操作提示与结果。

文本界面显示可选的功能菜单（共有0~23项功能），其中第二行显示当前操作二叉树的序号（默认为1），输入对应操作指令可使用相应功能，输入不在菜单中的操作指令会刷新界面并重新等待输入。演示系统界面如图3-16所示。

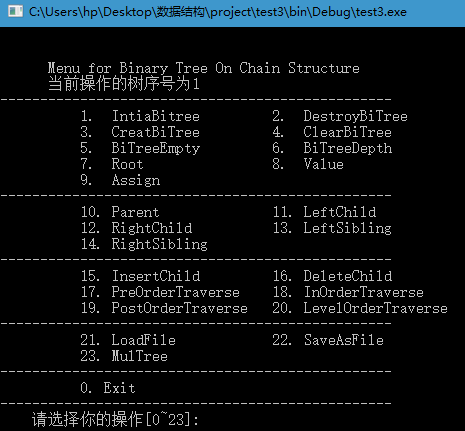


图3-16 启动程序后的操作菜单

输入操作指令0，程序退出，如图3-17所示。

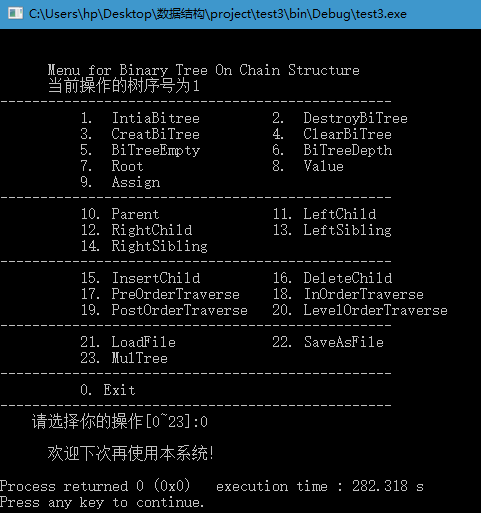


图3-17 程序退出

**3.3.3 测试计划**

对系统的二叉树操作功能按以下计划测试，其中二叉树运算操作测试计划如表3-1所示，森林操作功能测试计划如表3-2所示。

表3-1 二叉树操作功能测试

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 待测试功能 | 测试顺序 | 测试输入 | 预计输出 | 二叉树状态 |
| 1.IntiaBiTree | 1 | 无 | 初始化二叉树成功 | 创建了空树，深度为0 |
| 2.DestroyBiTree | 22 | 无 | 销毁二叉树成功 | 二叉树不存在 |
| 3.CreatBiTree | 2 | 第一次输入“a a”、  第二次输入“abc de f ” | 第一次创建二叉树失败、  第二次创建二叉树成功 | 第一次二叉树为空树；  第二次创建二叉树，且先序序列为“abc de f ” |
| 4.ClearBiTree | 16 | 无 | 清空二叉树 | 二叉树为空树 |
| 5.BiTreeEmpty | 3 | 无 | 二叉树不为空 | 二叉树先序序列为“abc de f ” |
| 6.BiTreeDepth | 4 | 无 | 二叉树最大深度为3 | 二叉树先序序列为“abc de f ” |
| 7.Root | 5 | 无 | 二叉树根key为0，data为a | 二叉树先序序列为“abc de f ” |
| 8.Value | 6 | 分别输入  key值10、  key值4 | 第一次查找提示错误信息、  第二次查找data为e | 二叉树先序序列为“abc de f ” |
| 9.Assign | 7 | 分别输入  key为7，data为o、  key为4，data为q | 第一次赋值失败、  第二次赋值成功 | 第二次赋值后二叉树先序序列为“abc dq f ” |
| 10.Parent | 8 | 分别输入  key为0、1 | 第一次查找双亲失败、  第二次查找双亲key为0、data为a | 二叉树先序序列为“abc dq f ” |
| 11.LeftChild | 9 | 分别输入  key为2、0 | 第一次查找左孩子失败、  第二次查找左孩子key为1、data为b | 二叉树先序序列为“abc dq f ” |
| 12.RightChild | 10 | 分别输入  key为5、3 | 第一次查找右孩子失败、  第二次查找右孩子key为5、data为f | 二叉树先序序列为“abc dq f ” |
| 13.LeftSibling | 11 | 分别输入key为1、3 | 第一次查找左兄弟失败、  第二次查找左兄弟key为1、data为b | 二叉树先序序列为“abc dq f ” |
| 14.RightSibling | 12 | 分别输入key为2、1 | 第一次查找右兄弟失败、  第二次查找右兄弟key为3、data为d | 二叉树先序序列为“abc dq f ” |
| 15.InsertChild | 14 | key为5、  LR=1，  创建序列分别为  “abc de f “、  “ab “ | 第一次插入失败、  第二次将先序序列为“ab ”的子树插入为key为5的结点的右子树 | 第二次插入后  二叉树先序序列为“abc dq f ab ” |
| 16.DeleteChild | 19 | 输入key为0，LR为1 | 删除key为0结点的右子树 | 二叉树先序序列为“abc ” |
| 17.PreOrderTraverse | 13 | 无 | 先序遍历输出二叉树，输出数据依次为abcdqf | 二叉树先序序列为“abc dq f ” |
| 18.InOrderTraverse | 18 | 无 | 中序遍历输出二叉树，输出数据依次为cbaedf | 二叉树先序序列为“abc de f ” |
| 19.PostOrderTraverse | 20 | 无 | 后序遍历输出二叉树，输出数据依次为cba | 二叉树先序序列为“abc ” |
| 20.LevelOrderTraverse | 15 | 无 | 层次遍历输出二叉树，输出数据依次为abdcqfab | 二叉树先序序列为“abc dq f ab ” |
| 21.LoadFile | 17 | 打开准备好的文件file001，其中的序列为“abc de f ” | 利用file001中的字符串成功创建二叉树 | 二叉树先序序列为“abc de f ” |
| 22.SaveAsFile | 21 | 输入文件名  file002 | 将字符串“abc ”存入文件file002 | 二叉树先序序列为“abc ” |
| 0.exit | 23 | 无 | 退出系统 | 二叉树不存在 |

表3-2 森林操作功能测试

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试顺序 | 测试功能 | 测试输入 | 预计输出 | 二叉树状态 |
| 1 | IntiaBiTree | 无 | 初始化二叉树1成功 | 创建了二叉树1，为空树 |
| 2 | CreatBiTree | 序列“a ” | 创建二叉树1成功 | 二叉树1前序序列为“a ” |
| 3 | PreOrderTraverse | 无 | 先序遍历输出二叉树1 | 二叉树1前序序列为“a ” |
| 4 | MulList | 输入2 | 新建并切换树成功，当前操作树序号为2 | 二叉树1前序序列为“a ”  树2未初始化 |
| 5 | IntiaBiTree | 无 | 初始化二叉树2成功 | 二叉树1前序序列为“a ”  树2为空树 |
| 6 | LoadFile | 打开已有文件file001，其中的序列为“abc de f ” | 文件读取成功 | 二叉树1前序序列为“a ”  二叉树2前序序列为“abc de f ” |
| 7 | PreOrderTraverse | 无 | 先序输出二叉树2所有数据 | 二叉树1前序序列为“a ”  二叉树2前序序列为“abc de f ” |
| 8 | DestroyBiTree | 无 | 销毁树2成功 | 二叉树1前序序列为“a ”  二叉树2未初始化 |
| 9 | ClearBiTree | 无 | 置空树2失败，树2未创建 | 二叉树1前序序列为“a ”  二叉树2未初始化 |
| 10 | MulTree | 输入1 | 切换树成功，当前操作树序号为1 | 二叉树1前序序列为“a ”  二叉树2未初始化 |
| 11 | InsertChild | key=0、LR=1、序列为“bc ” | 子树插入成功 | 二叉树1前序序列为  “a bc ” |
| 12 | PreOrderTraverse | 无 | 先序输出树1所有数据 | 二叉树1前序序列为  “a bc ” |
| 13 | DestroyBiTree | 无 | 销毁树1 | 树未初始化 |
| 14 | Exit | 无 | 退出系统 | 树未初始化 |

**3.3.4 测试结果**

1. 二叉树操作的测试结果
2. 执行第一步测试，初始化二叉树，提示操作成功（图3-18）。

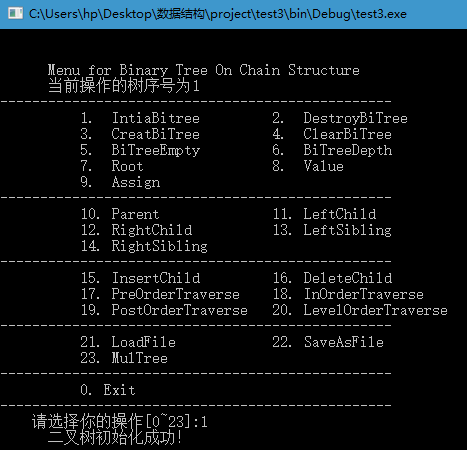


图3-18 初始化二叉树成功

1. 执行第二步测试，创建二叉树（图3-19、20）。

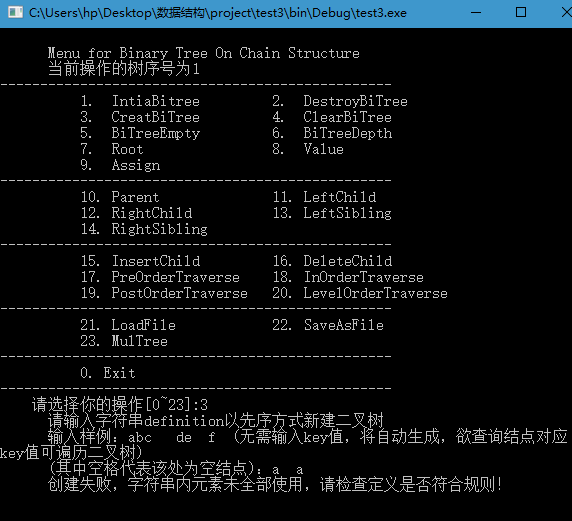


图3-19 创建二叉树失败

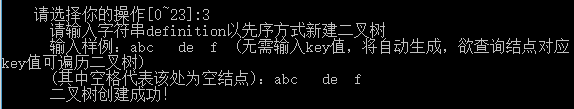


图3-20 创建二叉树成功

1. 执行第三步测试，判断二叉树是否为空（图3-21）。

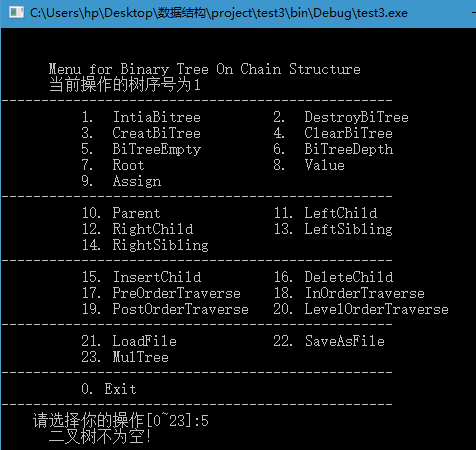


图3-21 判断二叉树是否为空

1. 执行第四步测试，求二叉树的深度（图3-22）。

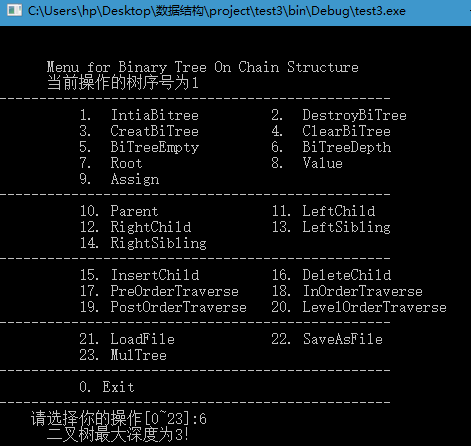


图3-22 求取二叉树的最大深度

1. 执行第五步测试，求二叉树根的值（图3-23）。

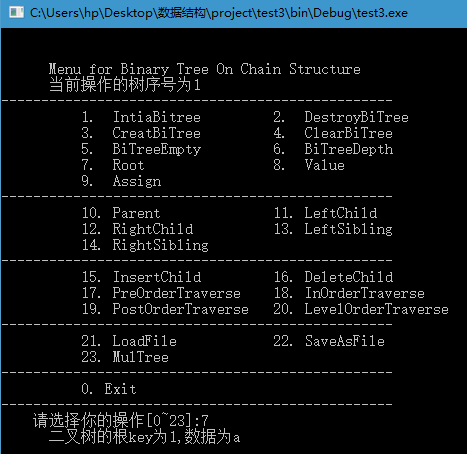


图3-23 求得二叉树根的值

1. 执行第六步测试，查找二叉树中key分别为10、4的结点的数据（图3-24、25）。



图3-24 查找二叉树中结点失败

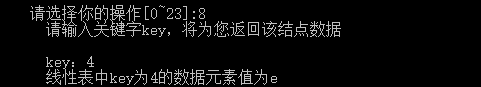


图3-25 查找二叉树中的结点成功

1. 执行第七步测试，对二叉树中key为7，4的结点分别赋值o、q（图3-26、27）。

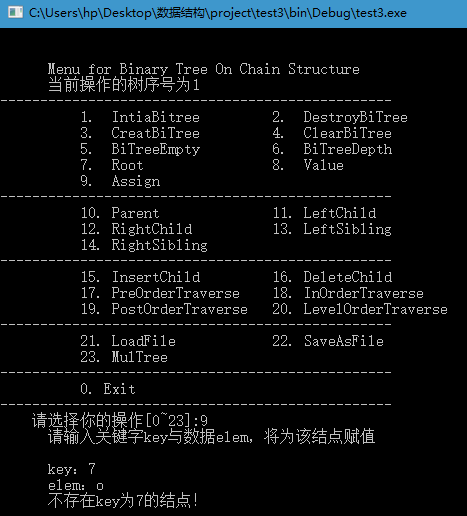


图3-26 二叉树结点赋值失败

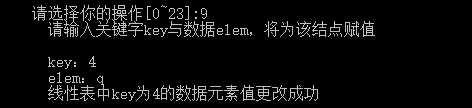


图3-27 二叉树结点赋值成功

1. 执行第八步测试，分别获取二叉树中key为0、1的结点的双亲结点（图3-28、29）。

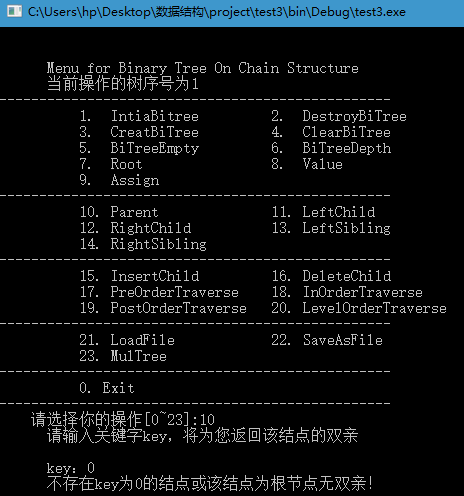


图3-28 查找双亲结点失败

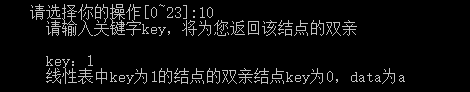


图3-29 查找双亲结点成功

1. 执行第九步测试，分别获取二叉树中key为2、0的结点的左孩子结点（图3-30、31）。



图3-30 查找左子结点失败

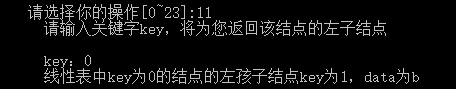


图3-31 查找左子结点成功

1. 执行第十步测试，查找二叉树中key为5、3的结点的右子结点（图3-32、33）。



图3-32 查找右子结点失败

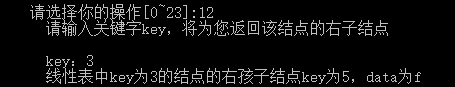


图3-33 查找右子结点成功

1. 执行第十一步测试，查找二叉树中key为1、3的结点的左兄弟结点（图3-34、35）。

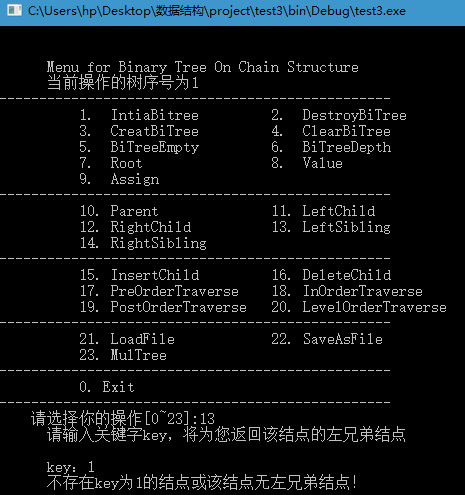


图3-34 查找左兄弟结点失败

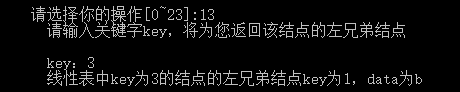


图3-35 查找左兄弟结点成功

1. 执行第十二步测试，查找二叉树中key为2、1的结点的右兄弟结点（图3-36、37）。



图3-36 查找右兄弟结点失败

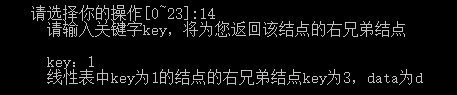


图3-37 查找右兄弟结点成功

1. 执行第十三步测试，先序输出二叉树（图3-38）。

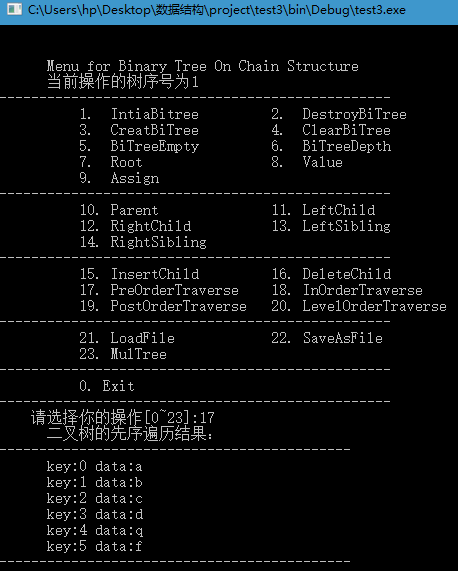


图3-38 先序遍历输出二叉树

1. 执行第十四步测试，向key为5的结点分别插入序列为“abc de f ”、“ab ”的子树（图3-39、40）。

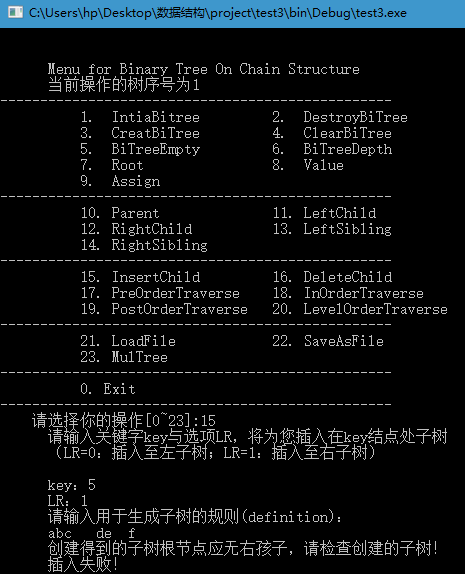


图3-39 子树结构错误，插入子树失败

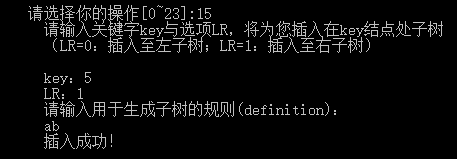


图3-40插入子树成功

1. 执行第十五步测试，层次遍历输出二叉树（图3-41）。

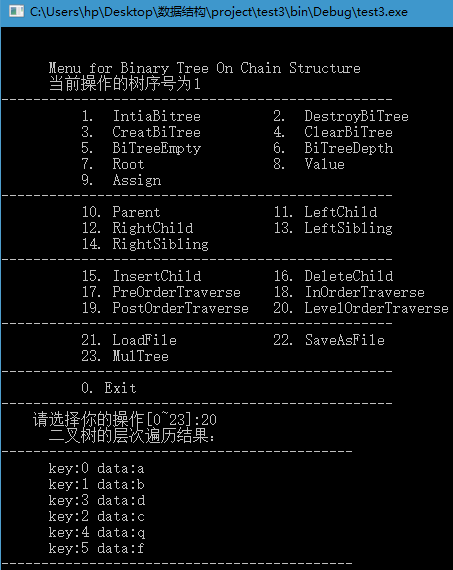


图3-41 层次遍历输出二叉树

1. 执行第十六步测试，清空二叉树（图3-42）。



图3-42 清空二叉树

1. 执行第十七步测试，从文件file001中读入二叉树（图3-43）。



图3-43 读取文件

1. 执行第十八步测试，中序遍历输出二叉树（图3-44）。

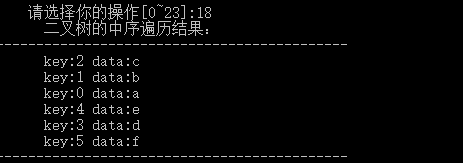


图3-44 中序遍历输出二叉树

1. 执行第十九步测试，重复删除根节点的右子树（图3-45、3-46）。

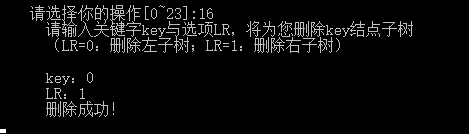


图3-45 删除子树成功

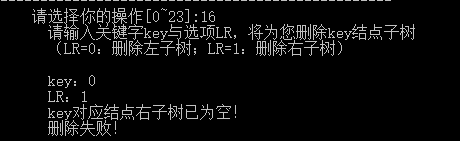


图3-46 重复删除子树失败

1. 执行第二十步测试，后序遍历输出二叉树（图3-47）。

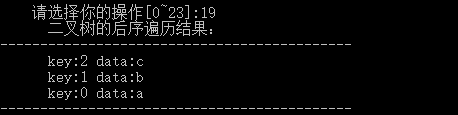


图3-47 后序遍历输出二叉树

1. 执行第二十一步测试，将二叉树中数据存储至文件file002中（图3-48、49）。

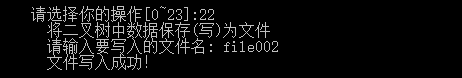


图3-48 存储文件

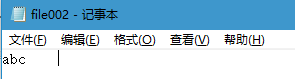


图3-49 file002文件中的数据

1. 执行第二十二步测试，销毁线性表（图3-50）。



图3-50 线性表销毁成功

1. 执行第二十三步测试，退出系统（图3-51）。

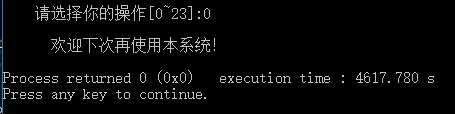


图3-51 退出系统

测试所得程序运行结果均与预计输出相匹配，验证了所编写的程序二叉树操作的正确性。

1. 森林相关操作的测试结果
2. 执行森林测试步骤1、2、3，初始化二叉树1并创建二叉树1，展示结果如图3-52所示。



图3-52 二叉树1中的所有数据元素

1. 执行森林测试步骤4，切换结果如图3-53所示。

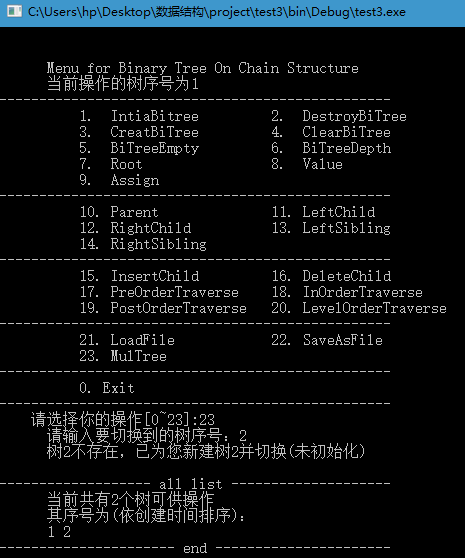


图3-53 切换至二叉树2

1. 执行森林测试步骤5、6、7，初始化二叉树2并向其中导入file001中的数据，展示结果如图3-54所示。

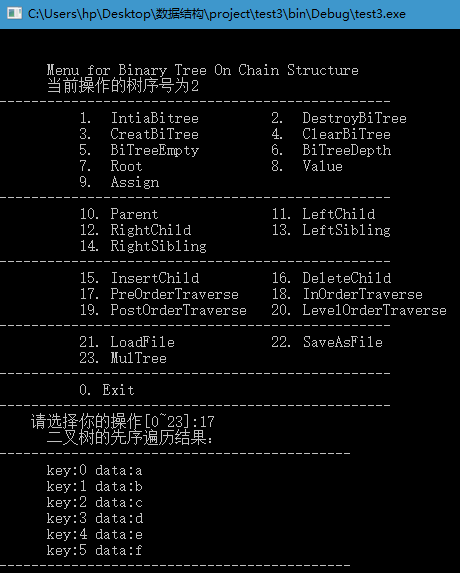


图3-54 二叉树2中的所有数据元素

1. 执行森林测试步骤8、9，销毁二叉树2后置二叉树2，无法置空证明二叉树2已被销毁，如图3-55所示。

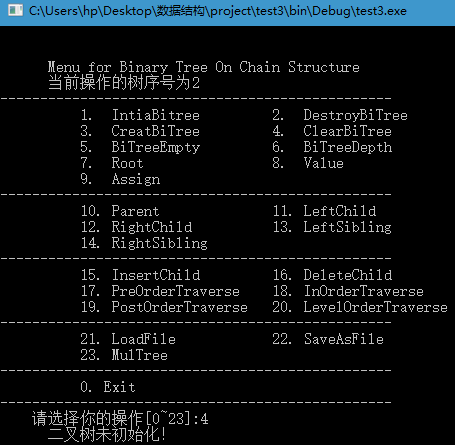


图3-55 二叉树2已被销毁，置空失败

1. 执行森林测试步骤10、11、12，二叉树1元素仍然可被输出，且自动生成的key值不受影响，证明两树操作具有独立性，如图3-56所示。

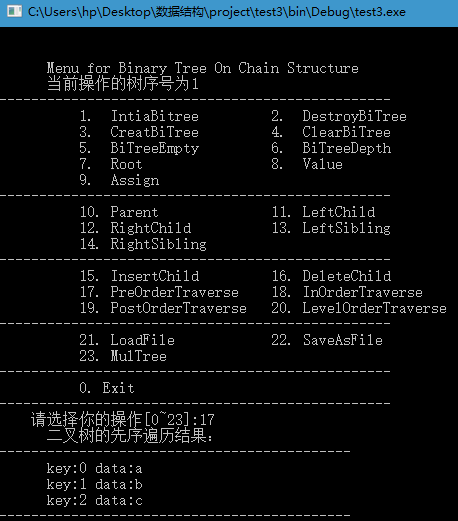


图3-56 二叉树1数据仍可被输出

1. 执行森林测试步骤13、14，销毁二叉树1并退出系统。

森林测试所得程序运行结果与预计结果相匹配，证明了森林中不同二叉树之间的数据插入、文件导入、树销毁等操作不会相互影响，即森林操作具有独立性且切换功能可以正常使用，体现了程序功能的正确性。

**3.4 实验小结**

通过本次的实验我大致遇到了如下的问题并有了以下一些体会。

**在实验中主要遇到的问题有：**

1. 调试时，系统执行函数的时候常因为接口不够完善产生各种各样的bug。比如当结点为空时对其进行操作，输入definition序列为空时仍利用其系统初始化的序列直接进行创建二叉树，轻则导致输出数据有误，重则导致死机。

经过分析，在每个可能出现类似问题的地方都加上了判断语句，确保输入符合一定要求后才能将其传入函数，防止因为输入数据格式不符的原因导致函数执行出错。通过这样的方式可以保证程序内各种接口的健康性。

1. 编写递归代码时由于代码不够直观，调试时常由于访问了未分配内存的空指针导致系统死机。分析后发现不应使用如function(l->lchild)这样的函数执行语句，将其更改为if(t->lchild)function(l->child)，添加一层判断语句会减少很多出现错误的可能性。
2. 编写递归代码实现遍历二叉树寻找结点时最开始使用了如下写法，没有能够遍历完整个树：

function(temp,key)

{

if(temp->key==key)return temp;

if(temp->lchild)return temp->lchild;

if(temp->rchild)return temp->rchild;

}

这样的函数会由于逻辑问题只向左子树挖掘到底端而不能够对右子树进行判断，分析后为其添加了judge指针作为中转，从而使递归逻辑能够全部执行，代码如下：

function2(temp,key)

{

BiTree \*judge = NULL;

if(temp->key==key)return temp;

if(temp->lchild)judge = temp->lchild;

if(judge) return judge;

if(temp->rchild)judge= temp->rchild;

if(judge) return judge;

return NULL;

}

1. 在编写创建树函数时，需要在递归函数内对字符串definition序列进行遍历。为记录读取到的位置，分析后使用了二重指针传递字符串。如char \*\*d = definition，此时\*\*d即为字符串中第一个字符，而\*((\*d)+1)即第二个字符，同时在递归函数中这样的指针能够实现类似静态变量的更改，达到了目的。
2. 层次遍历起初有一定的困难无处下手。分析后引入了新的队列结构(顺序表实现)，通过进出队列的方式能够实现树的层次遍历，但同时这样的解决方案对二叉树的大小也有了限制，还需要添加相应的错误判断，或更改为链式队列，但会增加内存使用。
3. 编写森林(多树切换)功能时，因为系统中的key设置为了自动生成且初始化、清空、销毁表时刷新，多树之间的生成key会相互影响，可能导致一个树中出现两个key相同的结点。

分析后使用了一个类型为全局变量的数组存储每个二叉树中生成key的状态，从而保证了每个二叉树中生成或刷新key不会相互影响，增强了程序的正确性。

**本次实验过后我的体会：**

1. 这次实验中由于涉及到对二叉树的操作，较多的使用了递归函数。体会到了递归写法的特性：方便，逻辑简单，将大问题拆分为一个个小的子问题；但容易出错，因为涉及到函数的反复使用，单步调试观察变量时也比较困难；且递归函数实现功能的效率较低。

但总的来说，在解决问题是从递归的写法入手还是一种比较简单的方式，编写递归函数时需要注意考虑到所有可能传入函数的参数并对其做出相应的判断；同时应该注意逻辑的正确性，保证函数能够完成目的。在这次实验过后编写递归函数的能力有了一定的提升。

1. 函数参数的选择十分重要，如本次实验的创建树函数中的definition序列需要频繁的传入与传出函数，为保证读取位置的稳定性，使用了双重指针的方式记录读取位置。

在编写函数时需要注意函数的参数形式，根据目的的不同适当挑选能够传递变量更改或不能够传递变量更改的数据格式，保证函数能够达到编写者的目的。

1. 编写代码时对错误输入需要有一定的判断，如本次实验中对definition序列为空的判断，或调用头指针、头结点时对其是否为空的判断。

程序运行过程中由于输入错误信息导致崩溃的情况有很多，而在大型系统中这也是可能成为整个系统中的一个漏洞，所以对错误输入信息进行必须的阻挡显得尤为重要，在优化了类似这样的接口处后系统的正确性得到了提升。

**4基于邻接表存储结构的图(无向网)实现**

**4.1 问题描述**

为实现基于邻接表存储结构的图及以此为基础的多图管理、文件读写功能，按如下形式设计数据的逻辑结构及基本运算。

**4.1.1 图抽象数据类型**

依据最小完备性和常用性相结合的原则，设计了图的数据对象和数据关系，并定义了图的创建、销毁、获取数据元素、遍历等15种基本运算，具体数据和运算功能定义如下。

1. 数据结构

数据对象：V是具有相同特性的数据元素的集合，称为顶点集。

数据关系：R ＝ {VR}

VR = {<v，w>|v，w∈V且P(v，w)，<v，w>表示从v到w的弧，谓词P(v，w)定义了弧<v，w>的意义或信息}

1. 基本运算

创建图：函数名称是CreateCraph(\*G,Vertex,VR)；初始条件是图G不存在，Vertex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。

销毁图：函数名称是DestroyBiTree(\*G)；初始条件图G已存在；操作结果是销毁图G。

查找顶点：函数名称是LocateVex(G,\*num,data)；初始条件是图G存在，欲查找的顶点u和G中的顶点具有相同数据特征(data)；操作结果是若u在图G中存在，返回顶点u的位置信息，否则返回其它信息。

获得顶点值：函数名称是GetVex (G,num,\*data)；初始条件是图G存在，位置为num的顶点v是G中的某个顶点；操作结果是返回v的值。

顶点赋值：函数名称是PutVex (G,num,value)；初始条件是图G存在，位置为num的顶点v是G中的某个顶点；操作结果是对v赋值value。

获得第一邻接点：函数名称是FirstAdjVex(G, num)；初始条件是图G存在，位置为num的顶点v是G的一个顶点；操作结果是返回v的第一个邻接顶点，如果v没有邻接顶点，返回“空”。

获得下一邻接点：函数名称是NextAdjVex(G, num\_v, num\_w)；初始条件是图G存在，位置为num\_v的顶点v是G的一个顶点, 位置为num\_w的顶点w是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，如果w是最后一个邻接顶点，返回“空”。

插入顶点：函数名称是InsertVex(\*G,data,\*VR)；初始条件是图G存在，值为data的顶点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，并根据VR关系集插入与v邻接的边。

删除顶点：函数名称是DeleteVex(\*G,num)；初始条件是图G存在，位置为num的顶点v是G的一个顶点；操作结果是在图G中删除顶点v和与v相关的弧。

插入弧：函数名称是InsertArc(\*G,\*VR)；初始条件是图G存在，输入边关系集，根据VR关系集在G中插入对应的边。

删除弧：函数名称是DeleteArc(\*G,num\_v,num\_w)；初始条件是图G存在，位置为num\_v、num\_w的顶点v、w是G的顶点；操作结果是在图G中删除弧<v,w>与<w,v>。

深度优先搜索遍历：函数名称是DFSTraverse(G)；初始条件是图G存在；操作结果是对图G进行深度优先搜索遍历，依次对图中的每一个顶点访问顶点数据。

广深度优先搜索遍历：函数名称是BFSTraverse(G)；初始条件是图G存在；操作结果是对图G进行广度优先搜索遍历，依次对图中的每一个顶点访问顶点数据。

读取文件：函数名称是LoadFile(\*T,filename)，初始条件是图G已存在；操作结果是将名为filename文件中的指定结构数据覆盖至当前操作图。

写文件：函数名称是SaveAsFile(T,filename)，初始条件是图G已存在；操作结果是将当前操作图G中的数据全部以指定结构存入名为filename的文件。

**4.1.2 多图抽象数据类型**

基于4.1.1中已经实现的图数据类型与基本运算功能，设计多图的数据对象、数据关系。具体数据和多图切换功能定义如下。

1. 数据结构

数据对象：D = { ei|ei ∈ SqList }

数据关系：R = { <ei-1,ei>|ei-1,ei∈D }

1. 基本运算

定义一个名为MulNode的结构体，其成员为图序号、单个图的头指针、next指针。切换操作图时先存储当前二叉树，再读取目标图，若对应图不存在则新建，从而实现多图结构内图的切换。

**4.1.3 演示系统与数据文件**

构造一个具有文本菜单的功能演示系统，用于演示对图以及多图结构进行的各项操作。其中，在主函数中完成子函数调用所需的参数输入、结果显示以及对应的文字操作提示。

在设计了相应数据存储格式后，演示系统可进行图数据的文件保存或读取，增加了系统合理性。此外系统还可根据需要实现多图结构内图的切换操作。

**4.2 系统设计**

对数据类型与基本操作进行相应分析后，从物理结构，具体算法等方面合理设计系统，具体设计分析如下。

**4.2.1 数据物理结构**

1. 图(邻接表结构)中边关系结点的存储结构

typedef struct ArcNode{ //图(邻接表式存储结构)边关系的定义

int adjvex; //该弧所指向的顶点的位置

struct ArcNode \*nextarc; //指向下一条弧的指针

int info; //该弧相关信息的指针

}ArcNode;

1. 图(邻接表结构)中顶点的数据存储结构

typedef struct VNode{//图顶点结构的定义

VertexType data; //顶点信息

ArcNode \*firstarc; //指向第一条依附该顶点的弧的指针

}VNode,AdjList[MAX\_VERTEX\_NUM];

1. 用于存储边关系集VR的链表存储结构

typedef struct Arc\_Relation{//存放输入的弧关系

char tail; //弧尾数据

char head; //弧头数据

int weight; //弧的权

struct Arc\_Relation \*next; //指向下一个关系

}Relation;

1. 多图数据的存储结构

typedef struct MulNode{ //多图结构的定义

int list\_num; //操作图序号

BiTree list\_head; //操作图头指针

struct MulNode \*next; //指向下一个图头指针结点的指针

}MulNode;

森林数据的物理存储结构如图4-1所示。

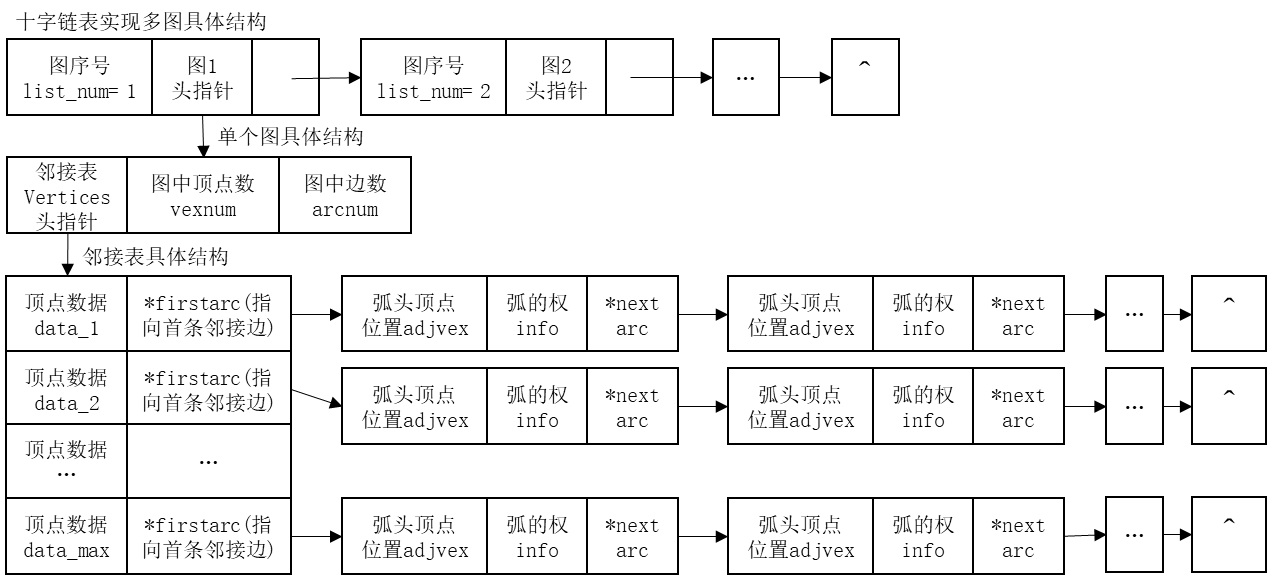


图4-1 多图数据的物理存储结构

**4.2.2 演示系统**

演示系统使用基于标准输入输出库（stdio.h）实现的文本界面，由功能菜单界面和用户操作（交互）界面组成。

功能菜单界面以文本形式输出功能菜单，展示当前操作图序号以及可选的图操作项，供用户参考选择调用需要的功能。

用户操作（交互）界面供用户进行功能的选择、数据的输入等，将用户输入的相关信息传递给对应图操作函数，输出调用函数后的操作结果或错误提示等信息。

其中演示系统的可选功能有：

创建、摧毁图；求取、修改图中任意顶点信息；求取图中顶点的邻接点信息、相对邻接点信息；添加、删除图中顶点与边；DFS、BFS遍历图；读取、存储文件。

演示系统具体模块结构如图4-2所示。

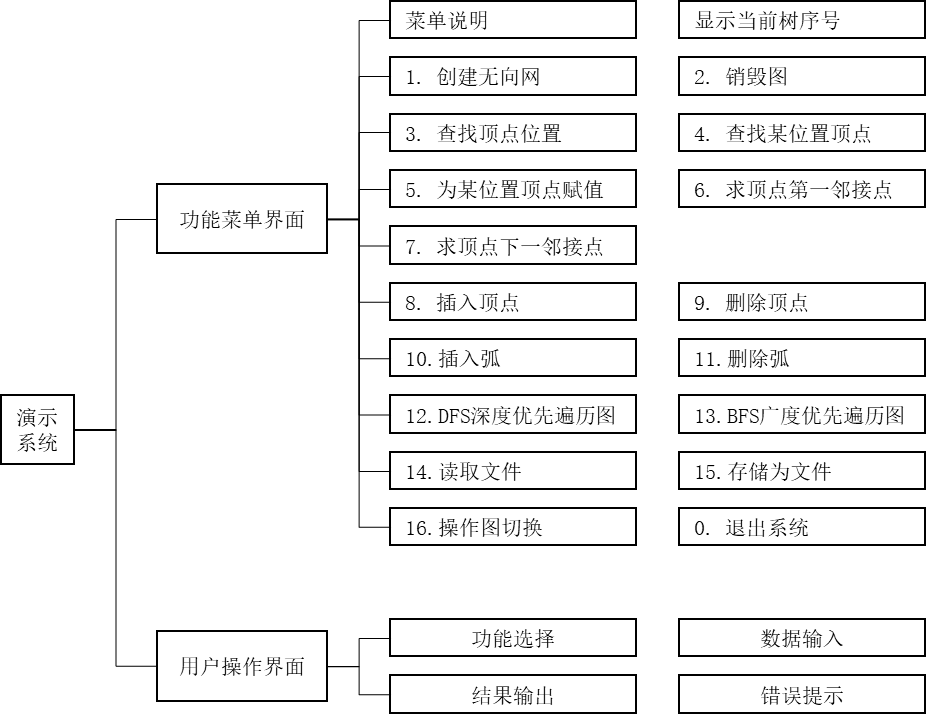


图4-2 演示系统模块结构图

**4.2.3 数据文件存储格式**

考虑到存储的数据文件需具有简洁高效的直观实用的性能，按如下方式设计数据文件存储格式。

文件格式：txt，便于直接查看和修改等。

存放方式：通过fprintf函数将图还原为顶点序列与边关系集存入文件。

数据文件存储格式如图4-3所示。

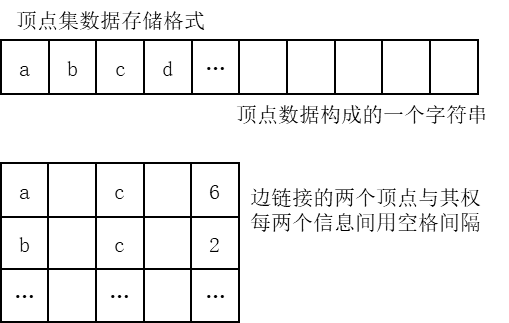


图4-3 数据文件存储格式示例

**4.2.4 图运算算法实现**

图运算算法及设计如下：

1. 创建图函数CreateGraph(\*G,\*Vertex,\*VR)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若不为空则提示图已被创建，不执行函数功能；否则为G分配ALNode类型大小的存储空间，若分配失败则返回ERROR提示溢出。若初始化图头结点成功，则初始化邻接表的每个顶点，并读入Vertex顶点集与VR边关系集。若传入的顶点集无重复顶点且格式正确，则根据其为邻接表中顶点赋值，随后根据传入的边关系集VR向图中插入边，返回OK。

**操作结果：**创建一个无向网G。

**时间复杂度分析：**指数阶O(n2)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图4-4所示

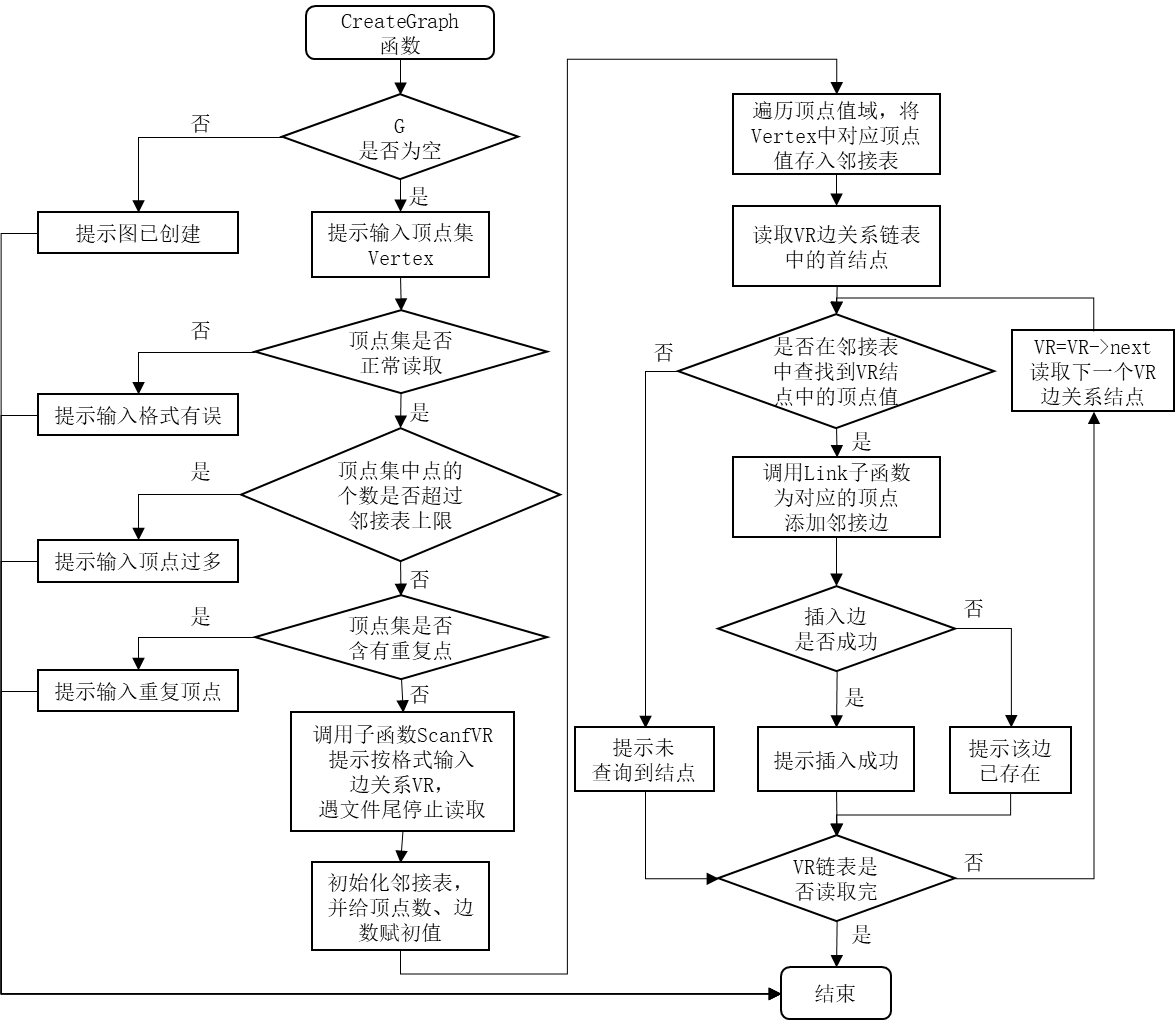


图4-4 创建图函数CreateGraph算法流程图

1. 销毁图函数DestroyGraph(\*G)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若为空则提示图未创建，不执行函数功能；若不为空，释放包括头结点G在内所有结点所占用的内存空间，并将G置空，销毁图，返回OK。

**操作结果：**销毁图G。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

1. 查找顶点函数LocateVex(G,\*num,data)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若为空则提示图未创建，不执行函数功能；若不为空，遍历邻接表中顶点集查找值为data的顶点，若查询到，则用num传递该顶点位置并返回OK；否则返回ERROR并提示在邻接表中未查询到值为data的顶点。

**操作结果：**用num返回图G中值为data的顶点的数据。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

1. 获得顶点函数GetVex(G,num,\*data)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若为空则提示图未创建，不执行函数功能；若不为空，判断输入的位置num是否处于合法范围num<0||num>=G->vexnum内，若不在该范围内，返回ERROR并提示输入num有误，访问越界；否则用data传递位置为num的顶点的数据并返回OK。

**操作结果：**查找图G中位置为num的顶点的数据。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 赋值顶点函数PutVex(\*G,num,value)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若为空则提示图未创建，不执行函数功能；若不为空，判断输入的位置num是否处于合法范围0=<num<G->vexnum内，若不在该范围内，返回ERROR并提示输入num有误，访问越界；否则修改位置为num的顶点的值为value。

**操作结果：**对图G中位置为num的顶点进行赋值。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

1. 获得首邻接点函数FirstAdjVex(G,num)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若为空则提示图未创建，不执行函数功能；若不为空，首先判断输入的num是否在合法范围0=<num<G->vexnum内，若不在则提示访问越界并返回空指针NULL，否则继续判断在位置num上的顶点是否有邻接点，若无则提示位置num上的结点无第一邻接点并返回空指针NULL，否则返回第一邻接点。

**操作结果：**查找图G中位置num上的顶点的第一邻接点。

**时间复杂度分析：**常量阶O(1)。

**空间复杂度分析：**常量阶O(1)。

**算法流程图：**如图4-5所示

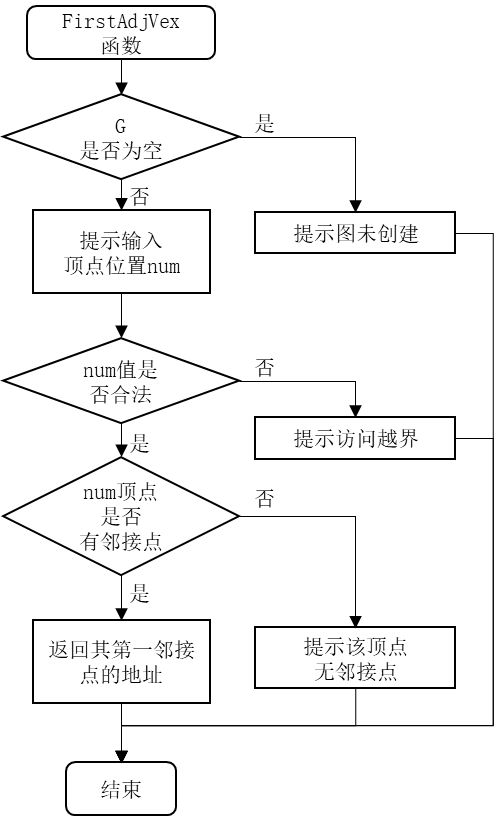


图4-5 获得首邻接点函数FirstAdjVex算法流程图

1. 获得下一邻接点函数NextAdjVex(G,num,num\_w)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若为空则提示图未创建，不执行函数功能；若不为空，首先判断输入的两个顶点的位置是否在合法范围0=<num<G->vexnum内，若不在则提示访问越界，对应顶点不存在并返回NULL，否则继续判断位置为num的顶点是否有邻接点，若无则返回NULL提示位置为num的顶点无临界点，若有在其邻接点中寻找位置为num\_w的顶点；若在num顶点的邻接点中未找到num\_w，则返回NULL并提示两顶点不邻接，若找到num\_w且其为最后一个邻接点则返回NULL并提示位置为num\_w的顶点已经是位置为num的顶点的最后一个邻接点，否则若找到num\_w且其不为最后一个邻接点则返回其下一邻接点。

**操作结果：**查找图G中位置num上的顶点相对位置为num\_w的顶点的下一邻接点。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**算法流程图：**如图4-6所示

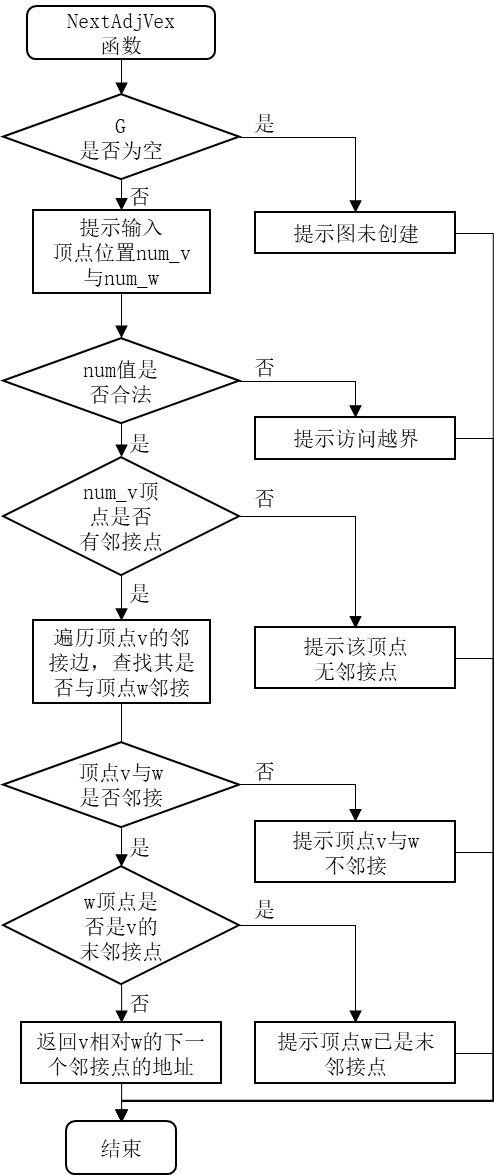


图4-6 获得下一邻接点函数NextAdjVex算法流程图

1. 插入顶点函数InsertVex(\*G,data,\*VR)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若为空则提示图未创建，不执行函数功能；若不为空，则首先在主函数中提示输入data及与新顶点相关的边关系集VR。将对应参数传入函数后，首先判断邻接表内是否仍有剩余空间，若无则提示邻接表已满并返回ERROR，否则继续判断插入的顶点是否已在邻接表内存在；若顶点已存在，则提示顶点在邻接表中已经存在并返回ERROR，否则在顶点集最后新增该顶点，记录图的顶点数+1并根据边关系集在新点与对应顶点之间插入边，图的边数对应增加。

**操作结果：**在图G中插入新顶点及与其相关的边。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图4-7所示

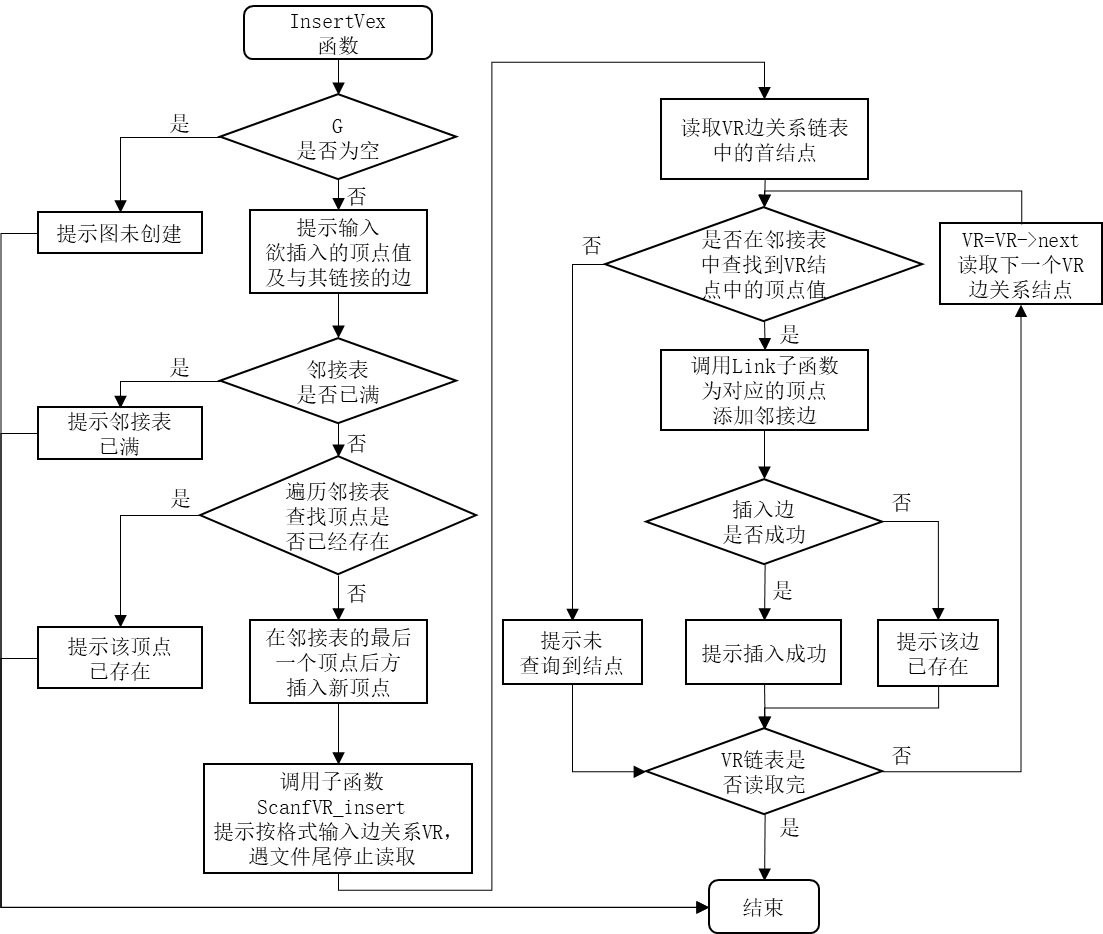


图4-7 插入顶点函数InsertVex算法流程图

1. 删除顶点函数DeleteVex(\*G,num)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若为空则提示图未创建，不执行函数功能；若G已经创建，首先在主函数中提示输入数据data，通过LocateVex函数查询该顶点的位置num，若查找失败则提示邻接表中无对应结点，不执行函数功能；若查找成功则向函数中传入待删除顶点的位序num，首先释放num位置上的顶点的相关边，随后遍历邻接表删除其他顶点与num顶点相关的边，删除相关边结束后删除该顶点，将顶点集中后续顶点向前移动，将邻接表原末顶点空间释放并初始化，记录图的顶点数-1。

**操作结果：**将图G中位置num上的顶点删除。

**时间复杂度分析：**指数阶O(n2)。

**空间复杂度分析：**线性阶O(1)。

**算法流程图：**如图4-8所示

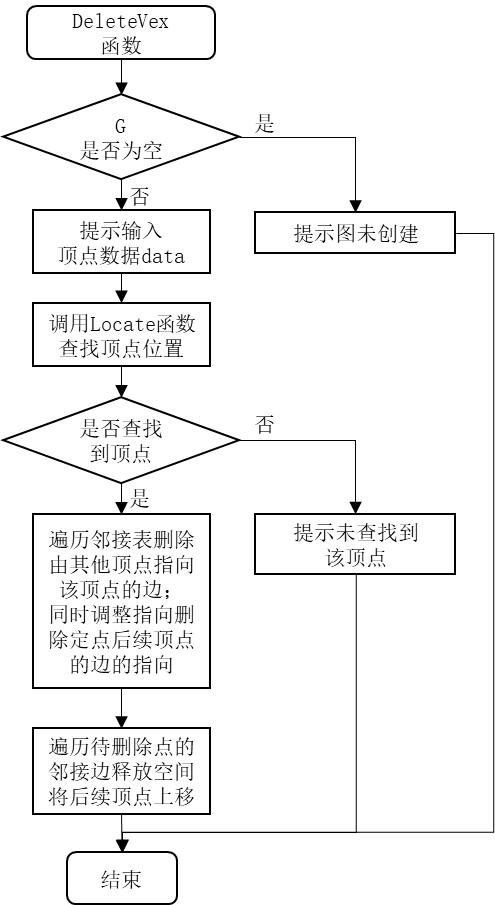


图4-8 删除顶点函数DeleteVex算法流程图

1. 插入弧函数InsertArc(\*G,\*VR)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若为空则提示图未创建，不执行函数功能；若不为空，则首先在主函数中提示输入边关系集VR并传入函数，根据VR边关系链表，遍历邻接表寻找对应的弧尾与弧头；若查找失败则提示未找到对应顶点并尝试构造下一条弧，若查找成功则判断边是否已经存在；若已经存在则提示对应边已存在并尝试构造下一条弧，若对应邻接边尚不存在则调用Link函数添加对应的两条边，并记录图中边数+1，VR边关系读取完毕后，返回OK并提示插入弧完成。

**操作结果：**在图G中批量插入弧。

**时间复杂度分析：**指数阶O(n2)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图4-9所示

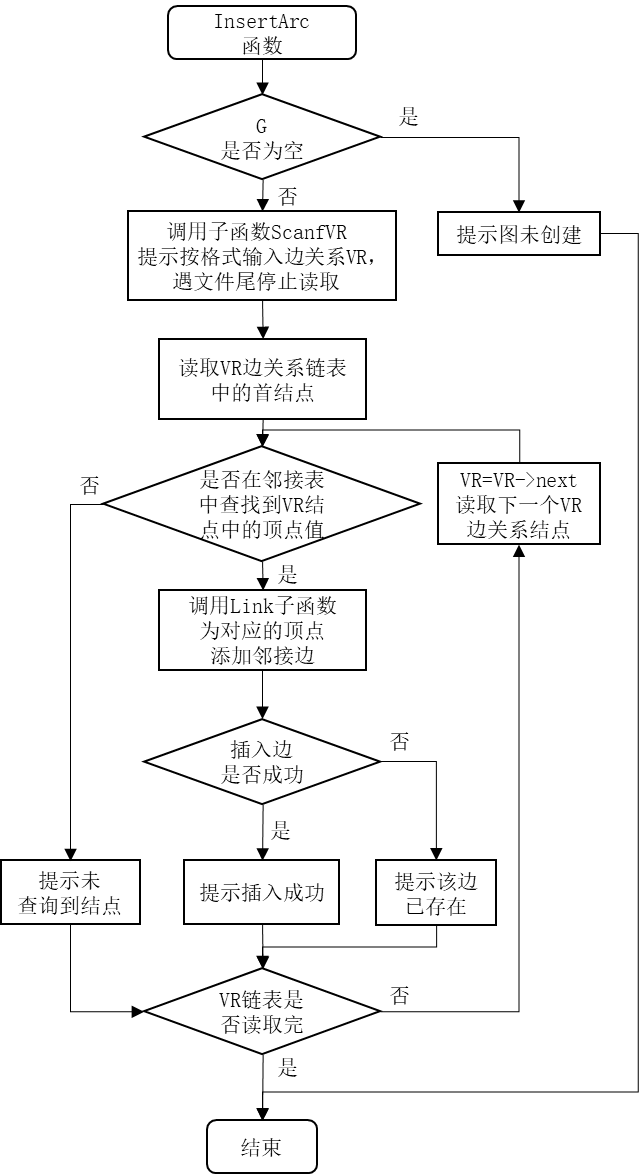


图4-9 插入弧函数InsertArc算法流程图

1. 删除弧函数DeleteArc(\*G,num\_v,num\_w)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若为空则提示图未创建，不执行函数功能；若不为空，首先判断num\_v与num\_w是否在合法范围0=<num<G->vexnum内，若不在合法范围内则提示对应结点访问越界并返回ERROR；否则判断顶点v与w是否有邻接边，若无则提示顶点v与w不邻接并返回ERROR，若顶点有邻接边则遍历相应顶点查找其与另一顶点的邻接边；若查找到则删除其边，记录图中的边数-1并返回OK，提示删除对应边成功，否则未查找到则提示顶点v与w不邻接并返回ERROR。

**操作结果：**将图G中位置num\_v与num\_w上的顶点之间的边删除。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**算法流程图：**如图4-10所示

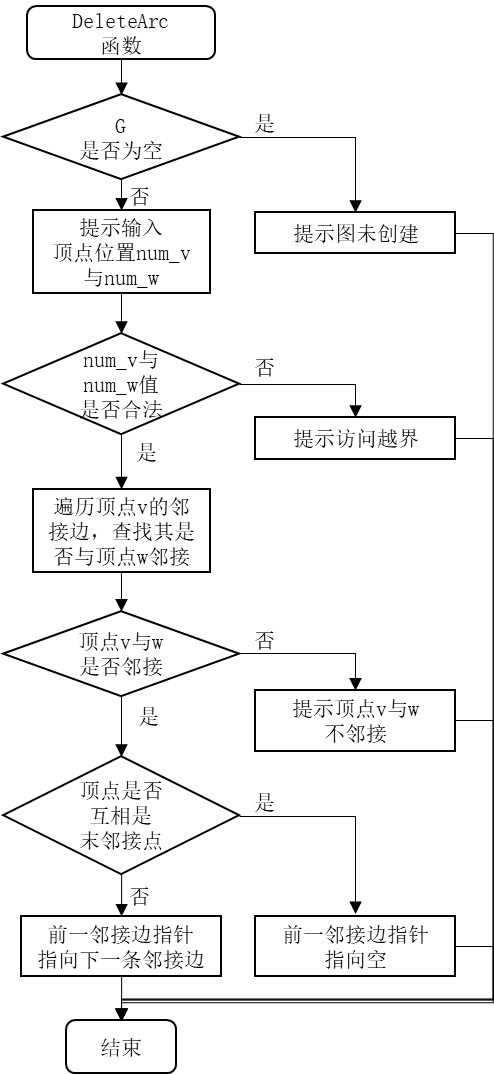


图4-10 删除弧函数DeleteArc算法流程图

1. 深度优先遍历图函数DFSTraverse(G)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若为空则提示图未创建，不执行函数功能；若不为空，通过printf绘制分界线，在函数中构造一个与顶点对应的数组用于标记顶点是否已被访问，通过递归以深度优先的方式遍历图并输出展示所有结点的数据，每两个顶点间的数据用空格间隔，输出结束后返回OK提示遍历结束。

**操作结果：**深度优先遍历并输出图的每个顶点的数据。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n) =O(v+e) (其中v、e分别为顶点、边数)。

**算法流程图：**如图4-11所示

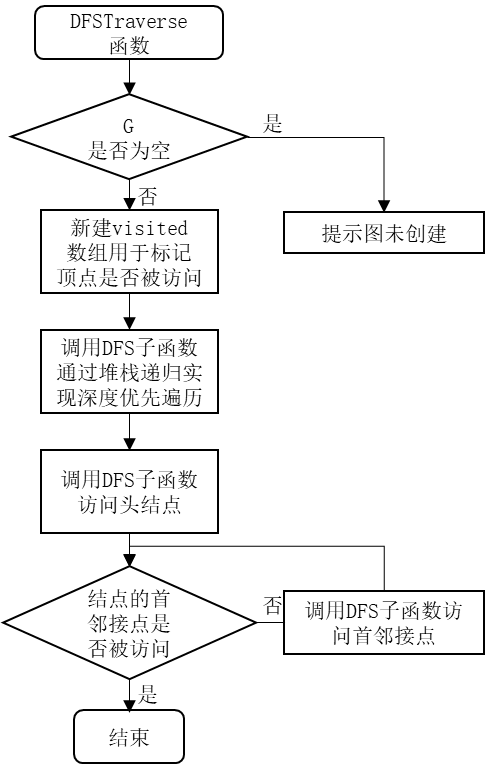


图4-11 深度优先遍历函数DFSTraverse算法流程图

1. 广度优先遍历图函数BFSTraverse(G)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若为空则提示图未创建，不执行函数功能；若不为空，通过printf绘制分界线，在函数中构造一个辅助队列用于广度优先遍历图，并构造一个与顶点对应的数组用于标记顶点是否已经被访问过，类似树的层次遍历，访问并标记首顶点，将其存入顶点，随后从队列中取出顶点，访问、标记并将其邻接顶点存入队列，直至队列中无顶点，输出完毕，随后依次对每个顶点进行判断，若未访问则进行广度优先遍历，对所有顶点判断结束后返回OK提示遍历结束。

**操作结果：**广度优先遍历并输出图的每个顶点的数据。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n)=O(v+e) (其中v、e分别为顶点、边数)。

**算法流程图：**如图4-12所示

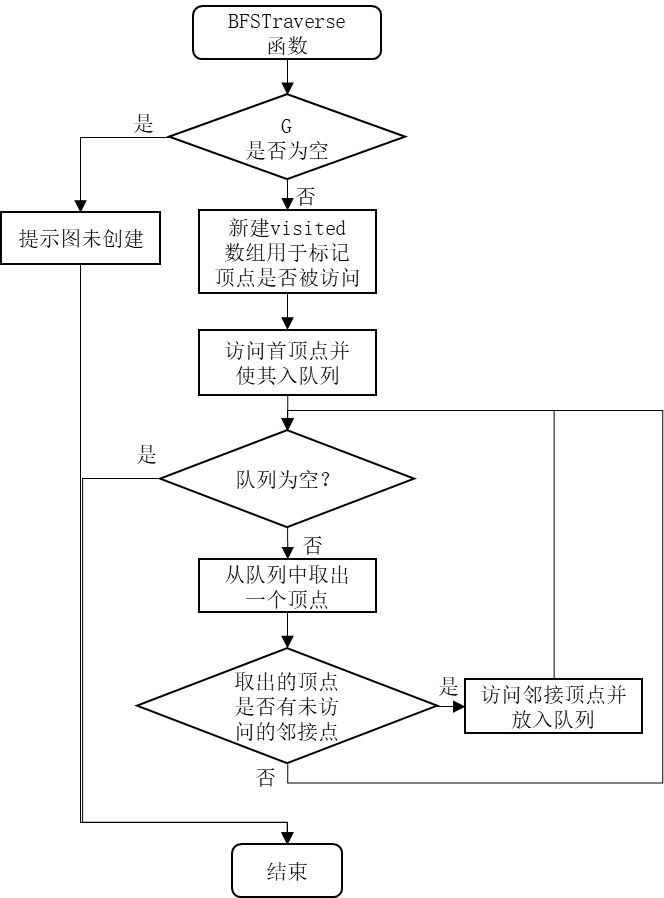


图4-12 广度优先遍历函数BFSTraverse算法流程图

1. 读取文件函数LoadFile(\*G,filename)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若为空则提示图未创建，不执行函数功能；若不为空，输出相应信息提示输入文件名称，通过只读方式打开文件，利用fscanf函数从文件中读取规范格式的顶点集序列与VR边关系集链表，读取完毕后关闭文件，随后尝试通过CreateGraph函数尝试创建图，若创建成功则将其覆盖至当前操作图并返回OK，否则提示创建失败，文件中数据有误并返回ERROR。

**操作结果：**将指定文件中字符序列读入系统，利用其创建图并覆盖当前已图。

**时间复杂度分析：**指数阶O(n2)。

**空间复杂度分析：**线性阶O(n)。

**算法流程图：**如图4-13所示。

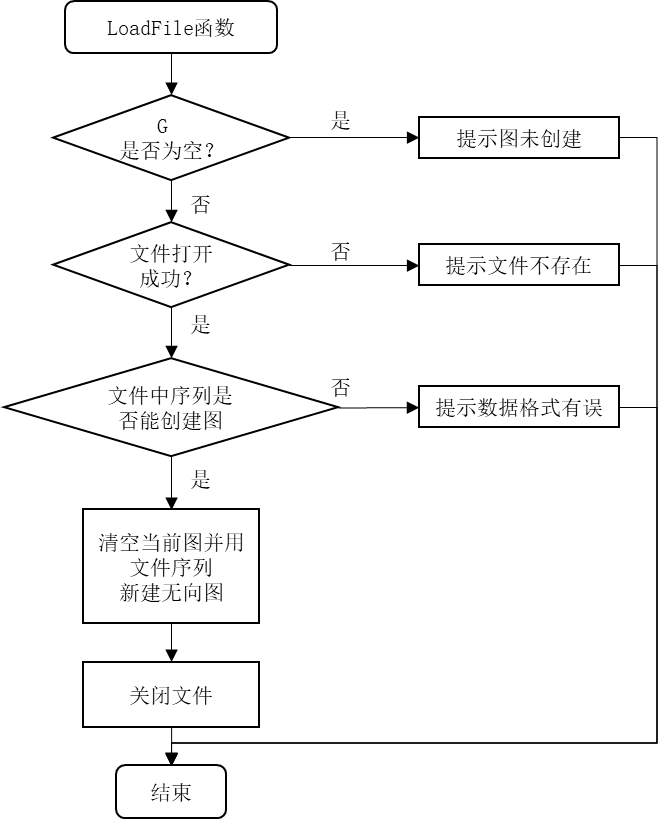


图4-13 读取文件函数LoadFile算法流程图

1. 保存文件函数SaveAsFile(G,filename)

**算法分析：**首先在主函数中判断G是否为空（判断图创建与否），若为空则提示图未创建，不执行函数功能；若不为空，输出相应信息提示输入文件名称，通过wb+参数以新建或覆盖的方式打开文件，首先将顶点集复原为字符串并利用fprintf函数将字符串写入文件；随后使用广度优先遍历的方式将图中边关系提取为对应顶点数据与边的权重，利用fprintf函数向文件中每行写入一条边关系，将图转换为指定格式的数据并写入文件后关闭文件，返回OK提示写入完毕。

**操作结果：**向指定文件中写入顶点集序列与边关系序列。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**线性阶O(n) =O(v+e) (其中v、e分别为顶点、边数)。

**算法流程图：**如图4-14所示。

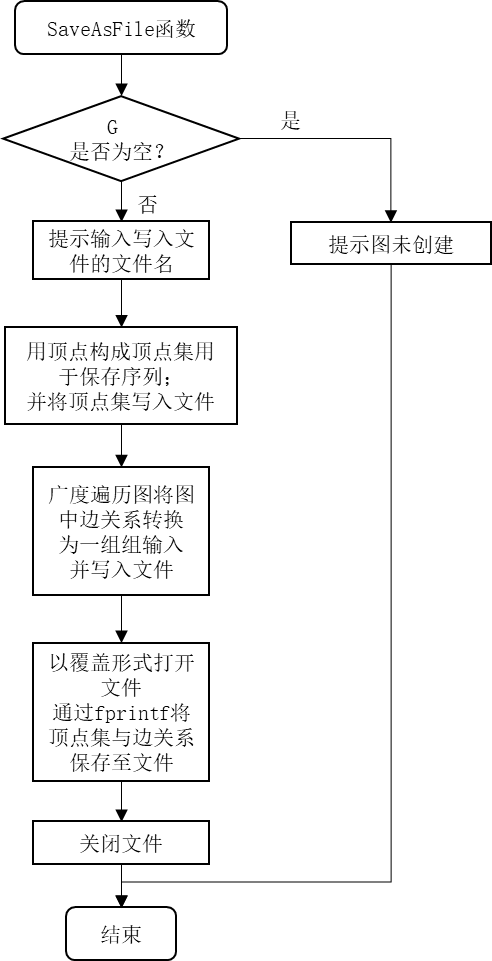


图4-14 保存文件函数SaveAsFile算法流程图

**4.2.5 多图切换算法实现**

多图切换函数MulList(\*G,\*M,list\_num\_cur,list\_num)

**算法分析：**数据结构设计中，M为多图结构的头指针，函数参数中list\_num\_cur为当前图序号、list\_num为将要切换到的图序号。在主函数中提示输入要切换到的图序号list\_num，先存储当前图的数据：遍历多图结构M中的每个图的头指针，将当前图的头指针G赋值给多图结构M中对应图序号的头指针；随后切换当前操作图为目标图：再次遍历多图结构M中的每个图的头指针，若查询到对应序号的图的头指针在多图结构M中已经存在，则将其赋值给当前操作图的头指针G，若目标图不存在，则在多图结构M中新建大小为MulNode类型结点，为新结点申请空间，失败则返回OVERFLOW提示溢出，否则为新结点赋值，随后用新结点为当前操作图头指针G赋值。

**操作结果：**通过指定序号切换当前操作的图。

**时间复杂度分析：**线性阶O(n)。

**空间复杂度分析：**常量阶O(1)。

**算法流程图：**如图4-15所示。

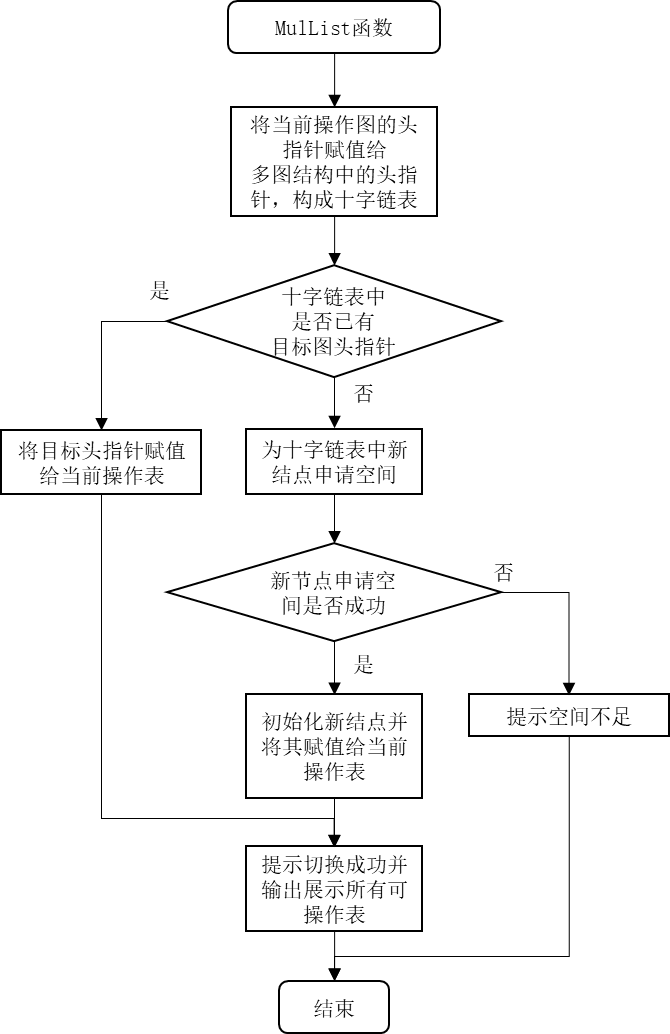


图4-15 多图切换函数MulList算法流程图

**4.3 系统实现**

具体系统实现结果如下。

**4.3.1 实验环境**

实验代码使用code::block编写与编译。

**4.3.2 演示系统操作**

演示系统采用键盘操作，控制台输出操作提示与结果。

文本界面显示可选的功能菜单（共有0~16项功能），其中第二行显示当前操作图的序号（默认为1），输入对应操作指令可使用相应功能，输入不在菜单中的操作指令会刷新界面并重新等待输入。演示系统界面如图4-16所示。

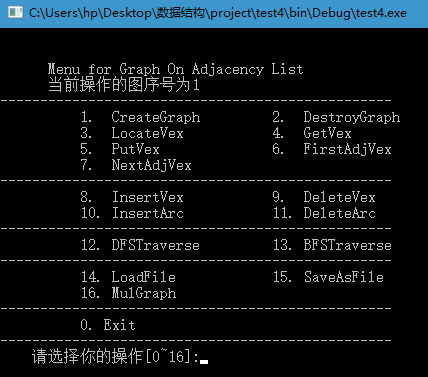


图4-16 启动程序后的操作菜单

输入操作指令0，程序退出，如图4-17所示。



图4-17 程序退出

**4.3.3 测试计划**

对系统的图操作功能按以下计划测试，其中图运算操作测试计划如表4-1所示，多图操作功能测试计划如表4-2所示。

表4-1 图操作功能测试

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 待测试功能 | 测试顺序 | 测试输入 | 预计输出 | 图状态 |
| 1. CreateGraph | 1 | 顶点集  第一次输入“abca”、  第二次输入“abcde”  边关系依次输入  abcd、a a 6、a f 2、a b 6、b c 7、c d 8、a d 3、d a 3 | 第一次输入顶点集创建失败、第二次创建成功；  边关系前两组输入提示数据错误，第三组提示未查找到顶点、随后四组输入成功创建、最后一组输入提示重复 | 第一次创建失败，图未创建；  第二次创建顶点集为abcde，边关系为abcd四个顶点构成环 |
| 2. DestroyGraph | 15 | 无 | 销毁图成功 | 图不存在 |
| 3. LocateVex | 2 | 第一次输入q、第二次输入a | 第一次获得位置失败、  第二次获得位置成功 | 图顶点集为abcde，边关系为abcd四个顶点构成环 |
| 4. GetVex | 4 | 第一次输入5、第二次输入0 | 第一次查找顶点失败、第二次查找顶点成功为q | 图顶点集为qbcde，边关系为qbcd四个顶点构成环 |
| 5. PutVex | 3 | 第一次输入5 q、第二次输入0 q | 第一次修改顶点值失败、  第二次修改顶点值成功 | 第一次图顶点集为abcde，边关系为abcd四点构成环；第二次操作后图顶点集为qbcde，边关系为qbcd四个顶点构成环 |
| 6. FirstAdjVex | 11 | 分别输入0、3、5 | 第一次输入提示第一邻接点num为4、值为p；第二次输入提示无邻接点；第三次输入提示访问越界，未找到num顶点 | 图顶点集为qcdep，边关系为q与p相连，c与d相连 |
| 7. NextAdjVex | 6 | 分别输入num\_v及num\_w为0 2、0 5、0 1 | 第一次输入提示v与w不邻接；第二次输入提示w已是v的末邻接点；第三次输入提示下一邻接点为d | 图顶点集为qbcdep，边关系为qbcd四个顶点构成环且pq相连 |
| 8. InsertVex | 5 | 分别输入  data为q、p，输入p时随后输入q 10使其链接结点q | 第一次插入失败提示顶点已经存在、第二次插入顶点与相关边成功 | 第一次图顶点集为qbcde，边关系为qbcd四点构成环；第二次操作后图顶点集为qbcdep，边关系为qbcd四个顶点构成环且pq相连 |
| 9. DeleteVex | 7 | 分别输入  data为a、  b | 第一次删除失败、  第二次删除成功 | 第二次操作后图顶点集为qcdep，边关系为q与d、p相连，c与d相连 |
| 10. InsertArc | 13 | 分别输入a b 1、a f 1、a e 1 | 第一次插入边提示边已经存在、第二次插入边提示未查找到结点、第三次插入边成功 | 顶点集为abcde，边关系为abcd四个顶点构成环、ae相互连接 |
| 11. DeleteArc | 9 | 分别输入  0 5与0 2 | 第一次访问越界删除失败、  第二次删除成功 | 图顶点集为qcdep，边关系为q与p相连，c与d相连 |
| 12. DFSTraverse | 8 | 无 | 深度优先遍历输出序列为q、d、c、p、e | 图顶点集为qcdep，边关系为q与d、p相连，c与d相连 |
| 13. BFSTraverse | 10 | 无 | 广度优先遍历输出序列为q、p、c、d、e | 图顶点集为qcdep，边关系为q与p相连，c与d相连 |
| 14. LoadFile | 12 | 从file001中读入数据 | 利用file001中的数据覆盖当前图 | 顶点集为abcde，边关系为abcd四个顶点构成环 |
| 15. SaveAsFile | 14 | 存储为文件file002 | 文件中顶点集合为abcde、边为a b 6、a d 3、a e 1、b c 7、d c 8 | 顶点集为abcde，边关系为abcd四个顶点构成环、ae相互连接 |
| 0.exit | 16 | 无 | 退出系统 | 图未创建 |

表4-2 多图操作功能测试

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试顺序 | 测试功能 | 测试输入 | 预计输出 | 图状态 |
| 1 | CreateGraph | 输入顶点集为abc、输入文件尾不创建边 | 创建图1成功 | 创建了图1，顶点集为abc、无边 |
| 2 | DFSTraverse | 无 | 深度优先输出图1，序列为abc | 创建了图1，顶点集为abc、无边 |
| 3 | MulList | 输入2 | 新建并切换图成功，当前操作图序号为2 | 图1顶点集为abc、无边；  图2未创建 |
| 4 | CreateGraph | 输入顶点集为def、输入文件尾不创建边 | 创建图2成功 | 图1顶点集为abc、无边；  图2顶点集为def、无边 |
| 5 | LoadFile | 打开已有文件file001，其中的顶点集为“abcde”、边关系为abcd成环 | 用文件中数据覆盖图2 | 图1顶点集为abc、无边；  图2顶点集为“abcde”、边关系为abcd成环 |
| 6 | BFSTraverse | 无 | 广度优先输出图2，序列为abdce | 图1顶点集为abc、无边；  图2顶点集为“abcde”、边关系为abcd成环 |
| 7 | DestroyGraph | 无 | 销毁图2成功 | 图1顶点集为abc、无边；  图2未创建 |
| 8 | DestroyGraph | 无 | 销毁图2失败，图2未创建 | 图1顶点集为abc、无边；  图2未创建 |
| 9 | MulGraph | 输入1 | 切换图成功，当前操作图序号为1 | 图1顶点集为abc、无边；  图2未创建 |
| 10 | InsertVex | 输入d | 插入新顶点d成功 | 图1顶点集为abcd、无边；  图2未创建 |
| 11 | DFSTraverse | 无 | 深度优先输出图1，序列为abcd | 图1顶点集为abcd、无边；  图2未创建 |
| 12 | DestroyGraph | 无 | 销毁图1 | 图未创建 |
| 13 | Exit | 无 | 退出系统 | 图未创建 |

**4.3.4 测试结果**

1. 图操作的测试结果
2. 执行第一步测试，根据输入序列创建图，第一次创建图失败；第二次提示操作成功并对输入的每条边进行判断后插入（图4-18、19）。

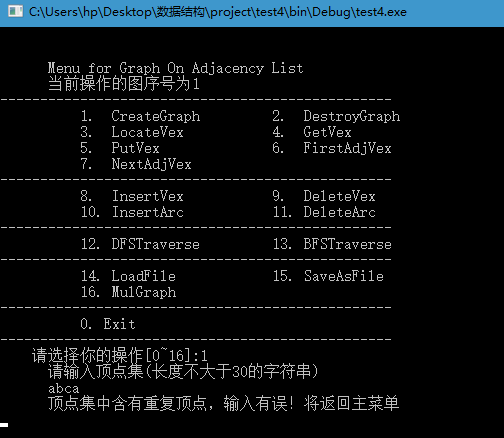


图4-18 创建图失败

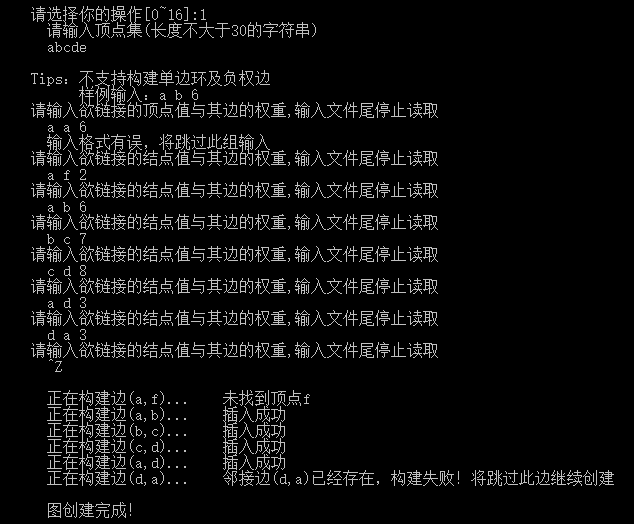


图4-19 创建图(无向网)成功

1. 执行第二步测试，分别取得顶点q、a的位置（图4-20、21）。

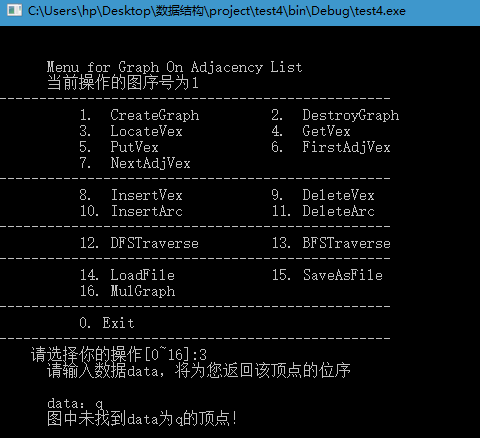


图4-20 取得q顶点位置失败

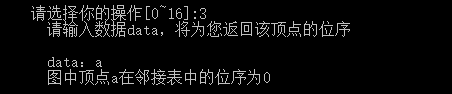


图4-21 取得a顶点位置成功

1. 执行第三步测试，分别修改位置为5、0的顶点的值为q（图4-22、23）。

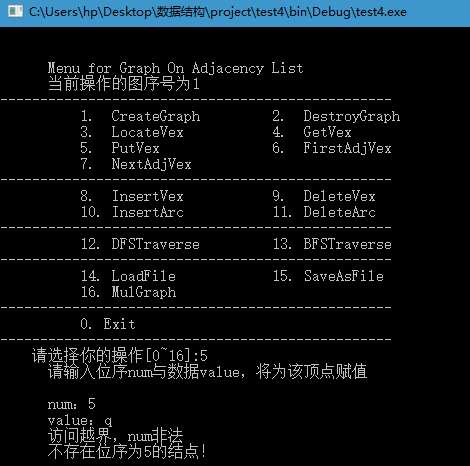


图4-22 修改顶点值失败

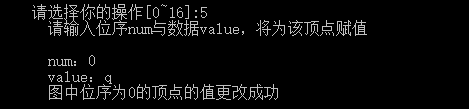


图4-23 修改顶点值成功

1. 执行第四步测试，分别求位置5、0上顶点的值（图4-24、25）。

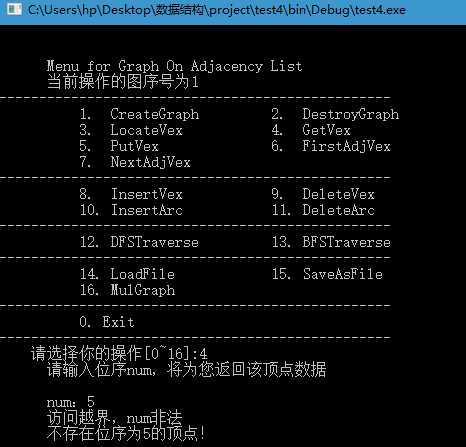


图4-24 获得图顶点失败

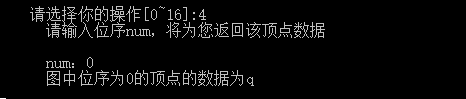


图4-24 获得图顶点成功

1. 执行第五步测试，分别向图中插入q、p顶点（图4-25、26）。



图4-25 插入顶点失败

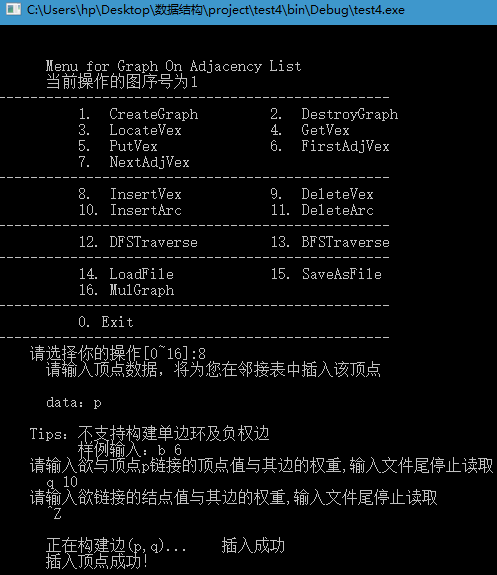
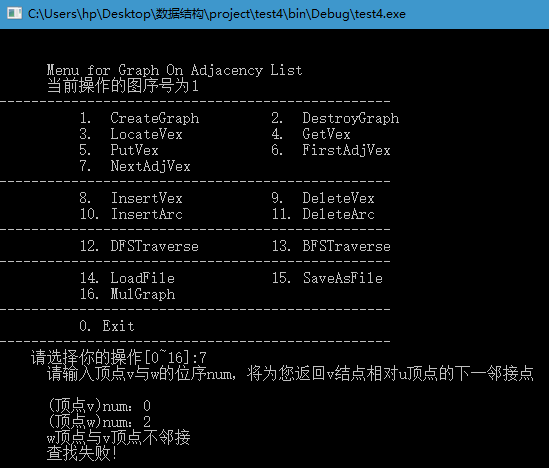


图4-26 插入顶点成功

1. 执行第六步测试，分别查找图中顶点0的邻接点中相对顶点2的下一个顶点、顶点0的邻接点中相对顶点5的下一个顶点、顶点0中相对顶点1的下一个顶点（图4-27、28）。



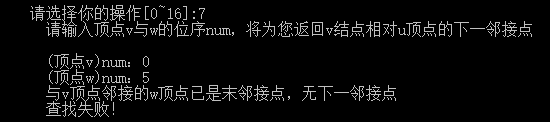


图4-27 查找下一邻接点失败

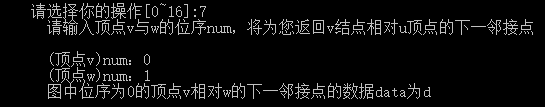


图4-28 查找下一邻接点成功

1. 执行第七步测试，分别删除图中值为a、b的顶点（图4-29、30）。

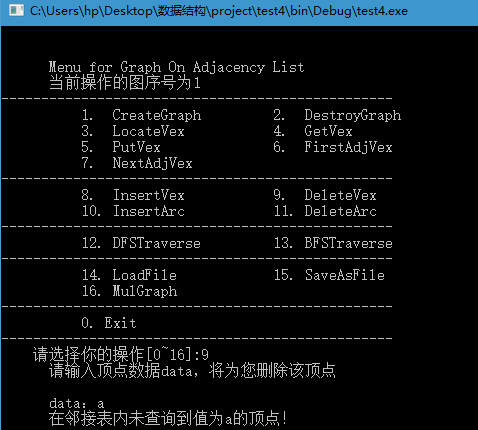


图4-29 删除顶点失败

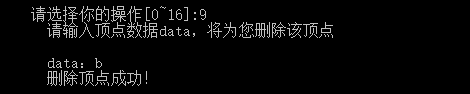


图4-30 删除顶点成功

1. 执行第八步测试，深度优先遍历输出图顶点（图4-31）。

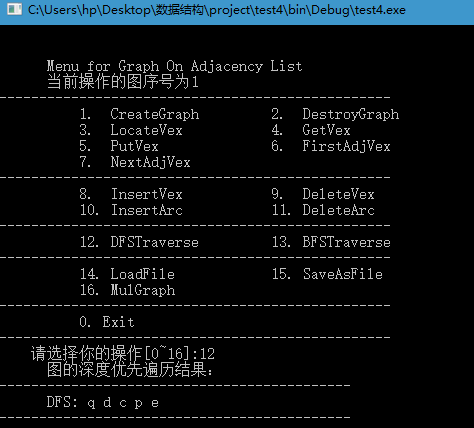


图4-31 深度优先遍历图

1. 执行第九步测试，分别删除图中顶点位置0、5之间的边、顶点位置0、2之间的边（图4-32、4-33）。

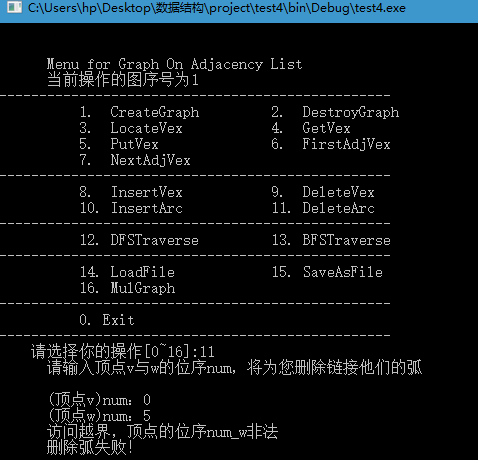


图4-32 删除边失败

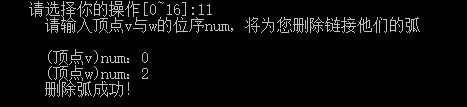


图4-32 删除边成功

1. 执行第十步测试，广度优先遍历图输出顶点（图4-33）。

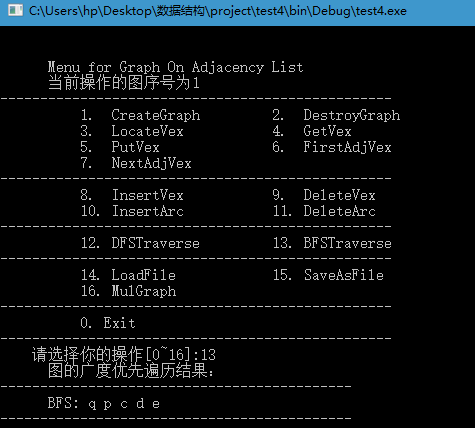


图4-33 广度优先遍历图

1. 执行第十一步测试，分别查找图中位置为0、3、5的顶点的首邻接点（图4-34、4-35）。

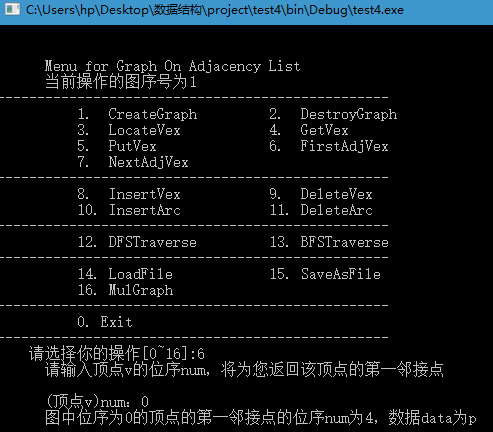
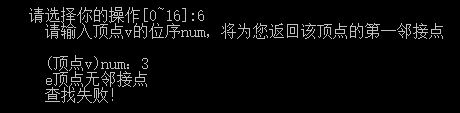


图4-34 查找首邻接点成功



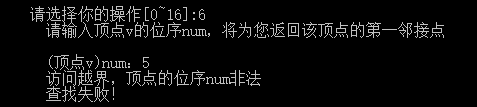


图3-35 查找首邻接点失败

1. 执行第十二步测试，从准备好的文件file001中读取数据覆盖当前图（图4-36）。

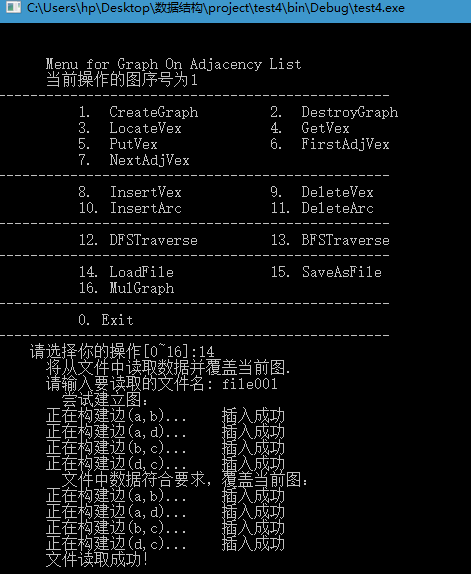


图4-36 从文件中读取数据建立图成功

1. 执行第十三步测试，分别插入边a b 1、a f 1、a e 1（图4-37）。



图4-37 向图中插入边成功

1. 执行第十四步测试，将图中数据写入文件file002进行保存（图4-38、39）。



图4-38 保存图中数据成功

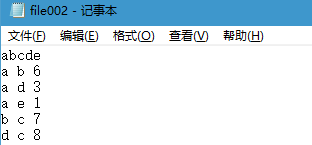


图4-39 文件file002中的数据

1. 执行第十五步测试，销毁图（图4-40）。



图4-40 销毁图成功

1. 执行第十六步测试，退出系统（图4-41）。

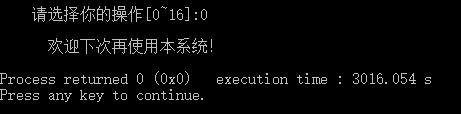


图4-41 退出系统

测试所得程序运行结果均与预计输出相匹配，验证了所编写的程序二叉树操作的正确性。

1. 多图相关操作的测试结果
2. 执行多图测试步骤1、2，创建图1并深度优先遍历，输出结果如图4-42所示。

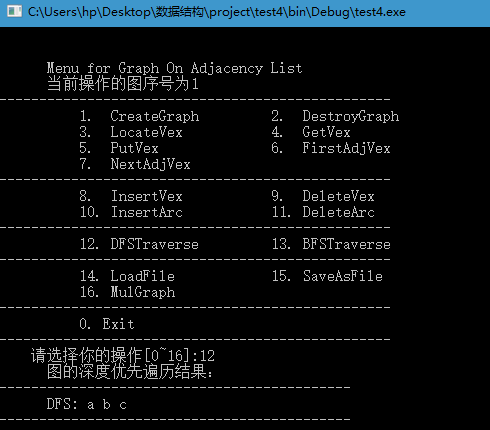


图4-42 图1的深度优先遍历

1. 执行多图测试步骤3，切换结果如图4-43所示。

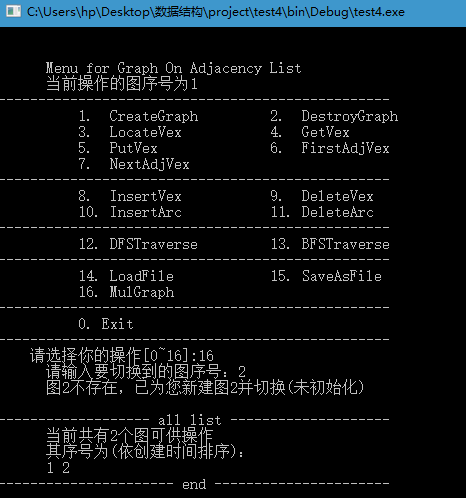


图4-43 切换至图2

1. 执行多图测试步骤4、5、6，创建二叉树2然后向其中导入file001中的数据并广度优先遍历，展示结果如图4-44所示。

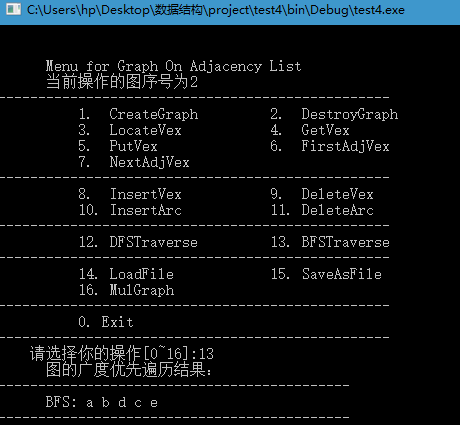


图4-44 图2的广度优先遍历

1. 执行多图测试步骤7、8，重复销毁图2，第二次无法销毁证明图2已被销毁，如图4-45所示。



图4-45 图2已被销毁，销毁失败

1. 执行多图测试步骤9、10、11，图1元素仍然可被修改并输出，证明两图操作具有独立性，如图4-46所示。

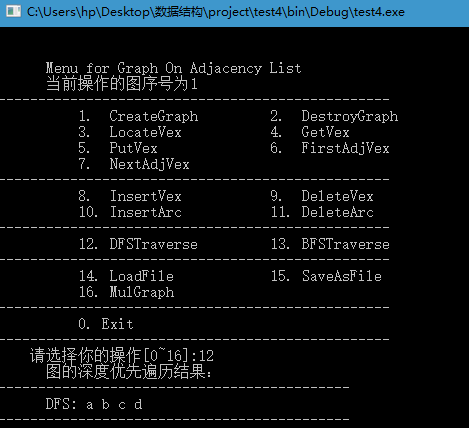


图4-46 图1数据仍可被输出

1. 执行多图测试步骤12、13，销毁图1并退出系统。

多图测试所得程序运行结果与预计结果相匹配，证明了多图结构中不同图之间的数据插入、文件读取、图销毁等操作不会相互影响，即多图结构具有独立性且切换功能可以正常使用，体现了程序功能的正确性。

**4.4 实验小结**

通过本次的实验我大致遇到了如下的问题并有了以下一些体会。

**在实验中主要遇到的问题有：**

1. 编写Create函数的过程中，在VR边关系存储格式的选定以及输入数据格式(Vertex顶点集的格式、VR边关系集的格式)是否规范的判断、以及输入数据组数的限定上遇到了一定的问题。

Create函数的边关系集参数VR若使用字符串的形式，在构建网时边的权值传递会遇到困难。分析后采用了传递链表的形式，VR为边关系链表的头结点，每个后续结点都是一组边关系。

在输入数据的格式判断上，主要分为两个方面，一是对Vertex顶点集的判断，只需要循环遍历字符串判断是否有重复顶点，并且对字符串做好初始化标记来判断是否读到了数据；另一个是对边关系的读取，因为数据量较多，若输入不正确容易读入空格或换行符，最后采用了读取完后对读取到的信息进行判断，若读到了空格、回车或没有正确读入权值，则放弃此组数据并输出错误提示，避免了边关系的混乱。

在输入组数上，因为觉得人工多次调用函数或输入组数过于麻烦，通过while(scanf()!=EOF)的语句使数据持续输入，直至输入文件尾时停止。

同时因为多处使用了scanf函数，容易在系统的输入流中残留换行符或其他数据，需要多次调试，使用getchar()或win系统特有的fflush函数来对输入流做好控制，保持程序的健壮性。

1. 编写DeleteVex函数时，起初遗漏了一些必要的处理，使得函数执行后，程序存储的数据产生了一些错误：对顶点进行删除后，若使用邻接表后续顶点前移的处理方法，需要注意弧的存储结构中的弧头是用邻接表下标记录的，若弧头记录的下标大于删除的顶点下标，均需要-1。

同时编写DeleteVex函数时要注意到各种边界情况，需要对首邻接点、末邻接点、一般邻接点分别进行考虑；在执行删除弧的语句后、继续遍历以修改其余弧时，需要对指针指向是否为空进行考虑，避免读取野指针造成死机。

由于涉及的情形较多，分析后设计了对应的邻接表数据并删除其中顶点以检查函数的合理性，数据中包含有：顶点在待删除顶点前且首邻接点是待删除点、顶点在待删除顶点前且末邻接点是待删除点、顶点在待删除顶点后且首邻接点是待删除点、顶点在待删除顶点前且末邻接点是待删除点、仅有一邻接点且其为待删除点。

测试输入及输入后的观测面板如下：

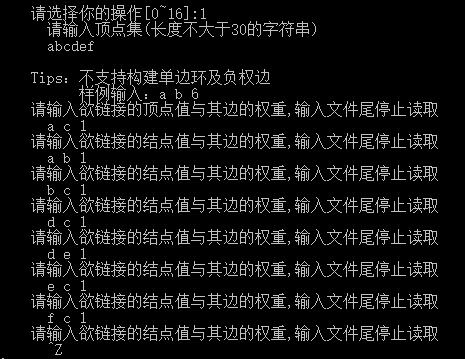


图4-47 检验DeleteVex函数的测试数据

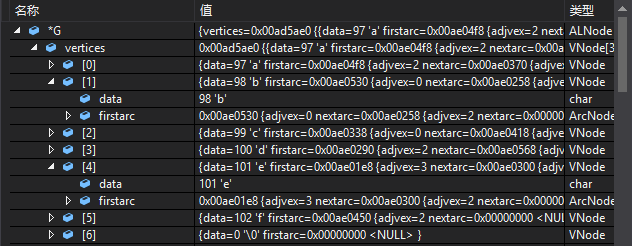


图4-48 测试数据输入后的邻接表

删除顶点c后的观测面板如下：

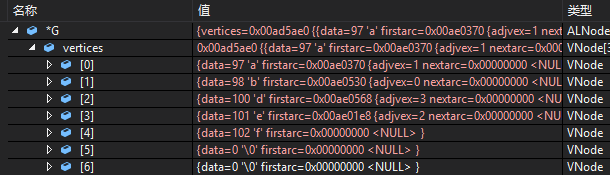


图4-49 删除顶点c的邻接表

删除顶点后由调试面板可以看出邻接表数据正常，验证了函数的合理性。

1. 编写SaveAsFile函数时发现使用fprintf向文件中输出一组数据+\n的情形下，使用记事本打开文件无法显示换行。

网上查找后发现是由于windows系统的特性会将换行拆分为\r\n，可以使用其他的文本管理器打开文件或在写入时加上回车符\r，最后得到了合理的文件。

1. 编写DFS、BFS遍历函数时考虑到无向图可能不连通，可能无法通过一次遍历输出所有顶点的数据。

分析后在DFS与BFS遍历的外层嵌套了一层for循环，对所有顶点依次执行，能够得到合理输出。

**本次实验过后我的体会：**

1. 图是比较常用的存储结构，比如在构筑社交关系网、地图求取最短路径等方面都会使用到各种不同的图结构，故而熟悉图的存储结构与对图的系列操作是十分必要的。

本次实验主要接触了无向带权图的邻接表存储结构，通过编写对其的查找、修改、遍历等操作，对其有了一定的掌握。

1. 在进行本次实验时考虑到的输入格式、边界情形时都比较充分，在构造对应测试数据时也能够达到目的。

在Create函数中设计的对输入格式的判断较为充分，能够辨别是否成功读取、读取的数据是否符合要求，系统能够对其做出相应的反应并提示错误信息。在对Delete函数进行编写时，开始考虑到的各种边界情况已经比较充分，编写也比较迅速，在发现功能存在缺陷时也很快的设计出对应的测试输入以检验函数正确性。

关于自身的能力，在本学期四次实验过后，我认为自己对于数据读写存储修改的掌握有了一定的进步，编写实现目的的代码时更加得心应手，调试程序也有了更多的途径。

1. 在本次实验之外，我尝试了一些能够在线测评代码的树、图习题，其中有哈夫曼树的构建、判断多顶点(1M)的无向图是否连通等。

通过对不同的树与图进行创建与操作，发现对于目的不同的数据结构，其存储结构可能会出现比较大的变化。如哈夫曼树的建立中，使用顺序存储辅助会比较适宜对可操作树的遍历、数值比较与链接，同样各式各样的图存储结构也会根据目的有所不同，但总的说来还是一通百通，在熟练掌握了其中一种结构后对其他的稍有变化的结构也会更加敏感。另外在处理顶点数比较庞大的图时，邻接表的存储结构已经体现出了不足，广度遍历已经无法申请出辅助队列，深度遍历也会因为时间复杂度过高被oj系统报错，这种时候便需要其他更巧妙的方法来达到自己的目的。

总的来说在本学期的课程结束后我的能力有了提升，但也发现了自身更多的不足之处、发现了更多可以提升的空间。

# 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

**指导教师评定意见**

一、对实验报告的评语

|  |
| --- |
|  |

二、对实验报告评分

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 评分项目  (分值) | 程序内容  (36.8分) | 程序规范  (9.2分) | 报告内容  (36.8分) | 报告规范  (9.2分) | 考勤  （8分） | 逾期扣分 | 合 计  (100分) |
| 得分 |  |  |  |  |  |  |  |

# 附录A 基于顺序存储结构的线性表实现的源程序

## 1.“LinearTable.h”头文件

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#define OK 1

#define ERROR -1

#define TRUE 1

#define FALSE -2

#define OVERFLOW -2

typedef int status;

typedef int ElemType; //数据元素类型定义

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

#define MAX 20

typedef struct{ //顺序表（顺序结构）的定义

ElemType \*elem;

int length;

int listsize;

}SqList;

status IntiaList(SqList \*L);

status DestroyList(SqList \*L);

status ClearList(SqList \*L);

status ListEmpty(SqList L);

int ListLength(SqList \*L);

status GetElem(SqList L,int i,ElemType \*e);

status LocateElem(SqList L,ElemType e,ElemType \*i); //简化过

status PriorElem(SqList L,ElemType cur,ElemType \*pre\_e);

status NextElem(SqList L,ElemType cur,ElemType \*next\_e);

status ListInsert(SqList \*L,int i,ElemType e);

status ListDelete(SqList \*L,int i,ElemType \*e);

status ListTrabverse(SqList L); //简化过

## 2.“main.c”文件

#include "LinearTable.h"

void main(void){

SqList MulList[MAX];//多表

int temp;

for(temp=0;temp<MAX;temp++){//初始化elem为null

MulList[temp].elem=NULL;

}

SqList \*L;//当前表

int list\_num=0;

L=MulList+list\_num;//指向表1

int op=1;

int length;//线性表长度

int i;//位序

int e;//返回位序i的值e

int cur;//需要寻找前驱或后继的数据

int pre\_e;//返回前驱

int next\_e;//返回后继

ElemType elem\_read;//读取文件缓存

FILE \*fp;

char filename[20];

while(op){

fflush(stdin);

system("cls"); printf("\n\n");

printf(" Menu for Linear Table On Sequence Structure \n");

printf(" 当前操作的表序号为%d\n",list\_num+1);

printf("-------------------------------------------------\n");

printf(" 1. IntiaList 9. NextElem\n");

printf(" 2. DestroyList 10. ListInsert\n");

printf(" 3. ClearList 11. ListDelete \n");

printf(" 4. ListEmpty 12. ListTrabverse\n");

printf(" 5. ListLength 13. OpenFile\n");

printf(" 6. GetElem 14. SaveAsFile\n");

printf(" 7. LocateElem 15. MulList\n");

printf(" 8. PriorElem \n\n");

printf(" 0. Exit\n");

printf("-------------------------------------------------\n");

printf(" 请选择你的操作[0~15]:");

scanf("%d",&op);

switch(op){

case 1:

//printf("\n----IntiaList功能待实现！\n");

if(IntiaList(L)==OK) printf(" 线性表创建成功！\n");

else printf(" 线性表已存在或存储空间不足，创建失败！\n");

getchar();getchar();

break;

case 2:

//printf("\n----DestroyList功能待实现！\n");

if(DestroyList(L)==OK) printf(" 线性表销毁成功！\n");

else printf(" 线性表不存在，销毁失败！\n");

getchar();getchar();

break;

case 3:

//printf("\n----ClearList功能待实现！\n");

if(ClearList(L)==OK) printf(" 线性表置空成功！\n");

else printf(" 线性表不存在或线性表已为空，置空失败！\n");

getchar();getchar();

break;

case 4:

//printf("\n----ListEmpty功能待实现！\n");

if(!L->elem) printf(" 线性表未创建！\n");//表不存在

else if(ListEmpty(\*L)==OK) printf(" 线性表为空！\n");

else printf(" 线性表不为空！\n");

getchar();getchar();

break;

case 5:

//printf("\n----ListLength功能待实现！\n");

if(!L->elem) printf(" 线性表未创建！\n");//表不存在

else{

length=ListLength(L);

printf(" 线性表长度为%d！\n",length);

}

getchar();getchar();

break;

case 6:

//printf("\n----GetElem功能待实现！\n");

if(!L->elem) printf(" 线性表未创建！\n");//表不存在

else{

printf(" 请输入位序，将为您返回该位序数据元素的值\n\n");

printf(" 位序：");

scanf("%d",&i);

if(GetElem(\*L,i,&e)==OK) printf(" 线性表中第%d个数据元素的值为%d\n",i,e);

else printf(" 访问越界，该位序不存在！\n");

}

getchar();getchar();

break;

case 7:

//printf("\n----LocateElem功能待实现！\n");

if(!L->elem) printf(" 线性表未创建！\n");//表不存在

else{

printf(" 请输入数据元素的值，将为您返回该数据的位序\n\n");

printf(" 数据元素的值：");

scanf("%d",&e);

if(LocateElem(\*L,e,&i)==OK) printf(" 线性表中值为%d的数据元素的位序为%d\n",e,i);

else printf(" 线性表中未查询到该数据元素！\n");

}

getchar();getchar();

break;

case 8:

//printf("\n----PriorElem功能待实现！\n");

if(!L->elem) printf(" 线性表未创建！\n");//表不存在

else{

printf(" 请输入数据元素的值，将为您返回该数据元素的前驱\n\n");

printf(" 数据元素的值：");

scanf("%d",&cur);

if(PriorElem(\*L,cur, &pre\_e)==OK) printf(" 线性表中值为%d的数据元素的前驱为%d！\n",cur,pre\_e);

else printf(" 线性表中未查询到该数据元素或该数据元素无前驱！\n");

}

getchar();getchar();

break;

case 9:

//printf("\n----NextElem功能待实现！\n");

if(!L->elem) printf(" 线性表未创建！\n");//表不存在

else{

printf(" 请输入数据元素的值，将为您返回该数据元素的后继\n\n");

printf(" 数据元素的值：");

scanf("%d",&cur);

if(NextElem(\*L,cur, &next\_e)==OK) printf(" 线性表中值为%d的数据元素的后继为%d！\n",cur,next\_e);

else printf(" 线性表中未查询到该数据元素或该数据元素无后继！\n");

}

getchar();getchar();

break;

case 10:

//printf("\n----ListInsert功能待实现！\n");

if(!L->elem) printf(" 线性表未创建！\n");//表不存在

else{

printf(" 将要插入数据元素，请输入将要插入的数据元素的位序与值\n\n");

printf(" 位序：");

scanf("%d",&i);

printf(" 值：");

scanf("%d",&e);

if(ListInsert(L,i,e)==OK) printf(" 数据元素插入成功！\n");

else printf(" 越界，数据元素插入失败！\n");

}

getchar();getchar();

break;

case 11:

//printf("\n----ListDelete功能待实现！\n");

if(!L->elem) printf(" 线性表未创建！\n");//表不存在

else{

printf(" 将要删除数据元素，请输入将要删除的数据元素的位序\n\n");

printf(" 位序：");

scanf("%d",&i);

if(ListDelete(L,i,&e)==OK) printf(" 数据元素删除成功，删除的元素为%d！\n",e);

else printf(" 越界，数据元素删除失败！\n");

}

getchar();getchar();

break;

case 12:

//printf("\n----ListTrabverse功能待实现！\n");

if(!L->elem) printf(" 线性表未创建！\n");//表不存在

else if(!ListTrabverse(\*L)) printf(" 线性表是空表！\n");

getchar();getchar();

break;

case 13:

//printf("\n----读取文件功能待实现！\n");

if(!L->elem){

printf(" 线性表未创建！\n");//表不存在

getchar();getchar();

}

else{

printf(" 将从文件中读取数据并插入当前线性表表首.\n 请输入要读取的文件名: ");

getchar();

gets(filename);

fp = fopen(filename, "r");

i=0;

/\*将文件中十进制字符形式元素依次按读入系统\*/

while(fscanf(fp,"%d ",&elem\_read)!=EOF){

printf(" 读入第%d个元素:%d...\n", ++i , elem\_read);

ListInsert(L,i,elem\_read);//插入读到的数据

}

if(!i) printf(" 文件读取失败，文件不存在或其中无数据\n");

else printf(" 文件读取成功，表长为%d.\n",L->length);

fclose(fp);getchar();

}

break;

case 14:

//printf("\n----输出文件功能待实现！\n");

if(!L->elem){

printf(" 线性表未创建！\n");//表不存在

getchar();getchar();

}

else{

printf(" 请输入要写入的文件名: ");

getchar();

gets(filename);

fp = fopen(filename, "wb+");

/\*将表中元素依次按十进制字符形式写入文件\*/

for (i = 1; i <= L->length; i++){

printf(" 写入第%d个元素:%d...\n", i , L->elem[i]);

fprintf(fp, "%d ", L->elem[i]);

}

printf(" 文件写入成功.\n");

fclose(fp);getchar();

}

break;

case 15:

//printf("\n----多表处理功能待实现！\n");

printf(" 请输入要切换到的表序号[1~%d]：",MAX);

scanf("%d",&list\_num);

if(--list\_num<0 || list\_num>MAX-1){

printf("\n 对应表不存在，切换为表1\n");

list\_num=0;

getchar();getchar();

break;

}

else printf("\n 多表操作切换成功！\n");

L=MulList+list\_num;

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("\n 欢迎下次再使用本系统！\n");

}//end of main()

/\*\*

\* 函数名称：InitiaList

\* 函数参数：线性表L的地址

\* 函数功能：构造一个空的线性表

\* 返回值：成功构造返回OK，否则返回ERROR

\*\*/

status IntiaList(SqList \*L){

//构造一个空链表

if(L->elem) return ERROR; //表已被初始化

L->elem = (ElemType \*)malloc( LIST\_INIT\_SIZE \* sizeof (ElemType));

if(!L->elem) return ERROR; //存储分配失败

L->length=0; //空表长度为0

L->listsize=LIST\_INIT\_SIZE; //初始存储容量

return OK;

}

/\*\*

\* 函数名称：DestroyList

\* 函数参数：线性表L的地址

\* 函数功能：删除线性表

\* 返回值：成功销毁返回OK，否则返回ERROR

\*\*/

status DestroyList(SqList \*L){

//销毁线性表

if(!L->elem) return ERROR;//表不存在

free(L->elem);

L->elem=NULL;

return OK;

}

/\*\*

\* 函数名称：ClearList

\* 函数参数：线性表L的地址

\* 函数功能：重置线性表

\* 返回值：成功置空返回OK，否则返回ERROR

\*\*/

status ClearList(SqList \*L){

//将L重置为空表

if(!L->elem) return ERROR;//表不存在

if(L->length==0) return ERROR;//线性表已为空

L->elem=NULL;

IntiaList(L);

return OK;

}

/\*\*

\* 函数名称：ListEmpty

\* 函数参数：线性表L

\* 函数功能：判断线性表是否为空

\* 返回值：若L为空表则返回OK，否则返回ERROR

\*\*/

status ListEmpty(SqList L){

//判断L是否为空表，是则返回OK，否则返回ERROR

if(L.length == 0) return OK;

else return ERROR;

}

/\*\*

\* 函数名称：ListLength

\* 函数参数：线性表L的地址

\* 函数功能：计算线性表L中数据元素个数；

\* 返回值：成功返回L中数据元素个数，失败返回ERROR

\*\*/

int ListLength(SqList \*L){

//计算顺序线性表长度

if(L) return L->length;

return ERROR;

}

/\*\*

\* 函数名称：GetElem

\* 函数参数：线性表L，第i个数据，e为第i个数据；

\* 函数功能：查找并用e返回L中第i个元素的值；

\* 返回值：成功则返回OK，否则返回ERROR

\*\*/

status GetElem(SqList L,int i,ElemType \*e){

//用e返回L中第i个数据元素的值

if(i<1 || i>L.length) return ERROR;//越界

else (\*e)=L.elem[i];

return OK;

}

/\*\*

\* 函数名称：LocateElem

\* 函数参数：线性表L，第i个数据，e为第i个数据；

\* 函数功能：查找L中与e相同数据所在的位序；用i返回该位序

\* 返回值：成功则返回OK，不存在则返回ERROR

\*\*/

status LocateElem(SqList L,ElemType e,ElemType \*i){//简化过

//线性表L已存在，用i返回L中第一个e的位序。若这样的e不存在，返回ERROR

int temp;

for(temp=1;temp<=L.length;temp++){

if(L.elem[temp]==e){

(\*i)=temp;

return OK;

}

}

return ERROR;//未找到

}

/\*\*

\* 函数名称：PriorElem

\* 函数参数：线性表L，查找的数据cur,前驱pre\_e；

\* 函数功能：查找L中与cur相同的第一个数据，并用pre\_e返回其前驱

\* 返回值：成功则返回OK；若无前驱以及其他情况则返回ERROR

\*\*/

status PriorElem(SqList L,ElemType cur,ElemType \*pre\_e){

//若线性表L已存在

//若cur是L的数据元素且不是第一个，则用pre\_e返回它的前驱，否则操作失败，pre.e无定义

int cur\_e;//位序

if(LocateElem(L,cur,&cur\_e)==OK){

if(cur\_e!=1){

(\*pre\_e) = L.elem[cur\_e-1];

return OK;

}

}

return ERROR;

}

/\*\*

\* 函数名称：NextElem

\* 函数参数：线性表L，查找的数据cur,后继next\_e；

\* 函数功能：查找L中与cur相同的第一个数据，并用next\_e返回其后继

\* 返回值：成功则返回OK；若无后继以及其他情况则返回ERROR

\*\*/

status NextElem(SqList L,ElemType cur,ElemType \*next\_e){

//若线性表L已存在

//若cur是L的数据元素且不是最后一个，则用next\_e返回它的后继，否则操作失败，next.e无定义

int cur\_e;//位序

if(LocateElem(L,cur,&cur\_e)){

if(cur\_e!=L.length){

(\*next\_e) = L.elem[cur\_e+1];

return OK;

}

}

return ERROR;

}

/\*\*

\* 函数名称：ListInsert

\* 函数参数：线性表L的地址，插入的位置i，插入的数据元素e。

\* 函数功能：在L的第i个位置之前插入新的数据元素e；若已插满，则按照分配增量，分配更大的空间。

\* 返回值：成功插入返回OK，否则返回ERROR

\*\*/

status ListInsert(SqList \*L,int i,ElemType e){

//在顺序线性表L中第i个位置之前插入新的元素e

int j;

//i的合法值为1≤i≤ListLength+1

if (i<1 || i>L->length+1) return ERROR; //i不合法

if (L->length >= L->listsize){ //溢出，扩充存储空间

ElemType \*newbase = (ElemType\*)realloc(L->elem,(L->listsize+LISTINCREMENT)\*sizeof(ElemType));

if (newbase==NULL) return OVERFLOW; //分配空间失败

L->elem = newbase;

L->listsize += LISTINCREMENT; // 增加固定大小存储容量

}

//向后移动元素，空出第i个元素的分量elem[i-1]

for(j=L->length;j>=i;j--)

L->elem[j+1]=L->elem[j];

L->elem[i]=e; /\*新元素插入\*/ //j+1

L->length++; /\*线性表长度加1\*/

return OK;

}

/\*\*

\* 函数名称：ListDelete

\* 函数参数：线性表L的地址，删除的位序i，指针e的地址。

\* 函数功能：删除L的第i个数据元素，用e返回其值。

\* 返回值：成功删除返回OK，否则返回ERROR

\*\*/

status ListDelete(SqList \*L,int i,ElemType \*e)

{

//删除L的第i个数据元素，用e返回其值

int j;

if (i<1 || i>L->length) return ERROR;

else {

(\*e)=L->elem[i];

for(j=i;j<L->length;j++)

L->elem[j]=L->elem[j+1];

L->length--;

}

return OK;

}

/\*\*

\* 函数名称：ListTrabverse

\* 函数参数：线性表L

\* 函数功能：依次显示线性表中的每个元素。

\* 返回值：成功遍历返回线性表的长度，否则返回ERROR

\*\*/

status ListTrabverse(SqList L){

int i;

printf("\n----------------- all elements ------------------\n");

for(i=1;i<=L.length;i++) printf("%d ",L.elem[i]);

printf("\n---------------------- end ----------------------\n");

return L.length;

}

# 附录B 基于链式存储结构的线性表实现的源程序

## 1.“LinearTable.h”头文件

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

/\*---------page 10 on textbook ---------\*/

#define OK 1

#define ERROR -1

#define TRUE 1

#define FALSE -2

#define OVERFLOW -2

typedef int status;

typedef int ElemType; //数据元素类型定义

/\*-------page 22 on textbook -------\*/

typedef struct LNode{ //顺序表（单链表存储结构）的定义

ElemType data; //线性表数据

struct LNode \*next; //指向下一个线性表数据结点的指针

}LNode, \*LinkList;

typedef struct MulNode{ //顺序表多表结构的定义

int list\_num; //操作表序号

LinkList list\_head; //操作表头指针

struct MulNode \*next; //指向下一个表头指针结点的指针

}MulNode;

/\*-----page 19 on textbook ---------\*/

status IntiaList(LinkList \*L);

status DestroyList(LinkList \*L);

status ClearList(LinkList \*L);

status ListEmpty(LinkList L);

int ListLength(LinkList L);

status GetElem(LinkList L,int i,ElemType \*e);

status LocateElem(LinkList L,ElemType e,ElemType \*i); //简化过

status PriorElem(LinkList L,ElemType cur,ElemType \*pre\_e);

status NextElem(LinkList L,ElemType cur,ElemType \*next\_e);

status ListInsert(LinkList \*L,int i,ElemType e);

status ListDelete(LinkList \*L,int i,ElemType \*e);

status ListTrabverse(LinkList L); //简化过

status LoadFile(LinkList L,char filename[20]);

status SaveAsFile(LinkList L,char filename[20]);

status MulList(LinkList \*L,MulNode \*M,int list\_num\_cur,int list\_num);

## 2.“main.c”头文件

#include "LinearTable.h"

void main(void){

MulNode \*MulList\_head=NULL;//多表表头,其data域存储多表个数

MulNode \*MulList\_temp=NULL;//用于遍历多表结点输出所有表序

//初始化多表结构

MulList\_head = (MulNode\*)malloc(sizeof(MulNode));

MulList\_head->list\_num= 1;//表头data域存储多表个数

MulList\_head->list\_head=NULL;

MulList\_head->next=(MulNode\*)malloc(sizeof(MulNode));//新建表1

MulList\_head->next->list\_num=1;//表序为1

MulList\_head->next->list\_head=NULL;

MulList\_head->next->next=NULL;

LinkList L=NULL;//当前表表头

int list\_num=1;//表序号

int list\_num\_cur=0;//记录当前表序号

int op=1;

int i;//位序

int e;//返回位序i的值e

int cur;//需要寻找前驱或后继的数据

int pre\_e;//返回前驱

int next\_e;//返回后继

int flag;//多表操作中记录函数返回值

char filename[20];

while(op){

fflush(stdin);

system("cls"); printf("\n\n");

printf(" Menu for Linear Table On Chain Structure \n");

printf(" 当前操作的表序号为%d\n",list\_num);

printf("-------------------------------------------------\n");

printf(" 1. IntiaList 9. NextElem\n");

printf(" 2. DestroyList 10. ListInsert\n");

printf(" 3. ClearList 11. ListDelete \n");

printf(" 4. ListEmpty 12. ListTrabverse\n");

printf(" 5. ListLength 13. OpenFile\n");

printf(" 6. GetElem 14. SaveAsFile\n");

printf(" 7. LocateElem 15. MulList\n");

printf(" 8. PriorElem \n\n");

printf(" 0. Exit\n");

printf("-------------------------------------------------\n");

printf(" 请选择你的操作[0~15]:");

scanf("%d",&op);

switch(op){

case 1:

//printf("\n----IntiaList功能待实现！\n");

if(L) printf(" 线性表已初始化，初始化失败！\n");

else{

if(IntiaList(&L)==OK) printf(" 线性表创建成功！\n");

else printf(" 线性表已存在或存储空间不足，创建失败！\n");

}

getchar();getchar();

break;

case 2:

//printf("\n----DestroyList功能待实现！\n");

if(!L) printf(" 线性表未初始化！\n");

else{

if(DestroyList(&L)==OK) printf(" 线性表销毁成功！\n");

}

getchar();getchar();

break;

case 3:

//printf("\n----ClearList功能待实现！\n");

if(!L) printf(" 线性表未初始化！\n");

else{

if(ClearList(&L)==OK) printf(" 线性表置空成功！\n");

else printf(" 线性表已为空，置空失败！\n");

}

getchar();getchar();

break;

case 4:

//printf("\n----ListEmpty功能待实现！\n");

if(!L) printf(" 线性表未初始化！\n");//表不存在

else if(ListEmpty(L)==OK) printf(" 线性表为空！\n");

else printf(" 线性表不为空！\n");

getchar();getchar();

break;

case 5:

//printf("\n----ListLength功能待实现！\n");

if(!L) printf(" 线性表未初始化！\n");//表不存在

else{

printf(" 线性表长度为%d！\n",ListLength(L));

}

getchar();getchar();

break;

case 6:

//printf("\n----GetElem功能待实现！\n");

if(!L) printf(" 线性表未初始化！\n");//表不存在

else{

printf(" 请输入位序，将为您返回该位序数据元素的值\n\n");

printf(" 位序：");

scanf("%d",&i);

if(GetElem(L,i,&e)==OK) printf(" 线性表中第%d个数据元素的值为%d\n",i,e);

else printf(" 访问越界，该位序不存在！\n");

}

getchar();getchar();

break;

case 7:

//printf("\n----LocateElem功能待实现！\n");

if(!L) printf(" 线性表未初始化！\n");//表不存在

else{

printf(" 请输入数据元素的值，将为您返回该数据的位序\n\n");

printf(" 数据元素的值：");

scanf("%d",&e);

if(LocateElem(L,e,&i)==OK) printf(" 线性表中值为%d的数据元素的位序为%d\n",e,i);

else printf(" 线性表中未查询到该数据元素！\n");

}

getchar();getchar();

break;

case 8:

//printf("\n----PriorElem功能待实现！\n");

if(!L) printf(" 线性表未初始化！\n");//表不存在

else{

printf(" 请输入数据元素的值，将为您返回该数据元素的前驱\n\n");

printf(" 数据元素的值：");

scanf("%d",&cur);

if(PriorElem(L,cur, &pre\_e)==OK) printf(" 线性表中值为%d的数据元素的前驱为%d！\n",cur,pre\_e);

else printf(" 线性表中未查询到该数据元素或该数据元素无前驱！\n");

}

getchar();getchar();

break;

case 9:

//printf("\n----NextElem功能待实现！\n");

if(!L) printf(" 线性表未初始化！\n");//表不存在

else{

printf(" 请输入数据元素的值，将为您返回该数据元素的后继\n\n");

printf(" 数据元素的值：");

scanf("%d",&cur);

if(NextElem(L,cur, &next\_e)==OK) printf(" 线性表中值为%d的数据元素的后继为%d！\n",cur,next\_e);

else printf(" 线性表中未查询到该数据元素或该数据元素无后继！\n");

}

getchar();getchar();

break;

case 10:

//printf("\n----ListInsert功能待实现！\n");

if(!L) printf(" 线性表未初始化！\n");//表不存在

else{

printf(" 将要插入数据元素，请输入将要插入的数据元素的位序与值\n\n");

printf(" 位序：");

scanf("%d",&i);

printf(" 值：");

scanf("%d",&e);

if(ListInsert(&L,i,e)==OK) printf(" 数据元素插入成功！\n");

else printf(" 越界，数据元素插入失败！\n");

}

getchar();getchar();

break;

case 11:

//printf("\n----ListDelete功能待实现！\n");

if(!L) printf(" 线性表未初始化！\n");//表不存在

else{

printf(" 将要删除数据元素，请输入将要删除的数据元素的位序\n\n");

printf(" 位序：");

scanf("%d",&i);

if(ListDelete(&L,i,&e)==OK) printf(" 数据元素删除成功，删除的元素为%d！\n",e);

else printf(" 越界，数据元素删除失败！\n");

}

getchar();getchar();

break;

case 12:

//printf("\n----ListTrabverse功能待实现！\n");

if(!L) printf(" 线性表未初始化！\n");//表不存在

else{

ListTrabverse(L);

if(!L->next) printf(" 线性表是空表！\n");

}

getchar();getchar();

break;

case 13:

//printf("\n----读取文件功能待实现！\n");

if(!L){

printf(" 线性表未初始化！\n");//表不存在

getchar();

}

else{

printf(" 将从文件中读取数据并插入当前线性表表首.\n 请输入要读取的文件名: ");getchar();

gets(filename);

if(LoadFile(L,filename)==OK) printf(" 文件读取成功，表长为%d.\n",ListLength(L));

else printf(" 文件读取失败，文件不存在或其中无数据\n");

}

getchar();

break;

case 14:

//printf("\n----输出文件功能待实现！\n");

if(!L){

printf(" 线性表未初始化！\n");//表不存在

getchar();

}

else{

printf(" 将线性表中数据保存(写)为文件\n 请输入要写入的文件名: ");getchar();

gets(filename);

if(SaveAsFile(L,filename)==OK) printf(" 文件写入成功.\n");

}

getchar();

break;

case 15:

//printf("\n----多表处理功能待实现！\n");

list\_num\_cur=list\_num;//记录当前表序以存储数据

printf(" 请输入要切换到的表序号：");

scanf("%d",&list\_num);

flag=MulList(&L,MulList\_head,list\_num\_cur,list\_num);

if(flag==OK) printf(" 切换成功，当前操作表为表%d\n",list\_num);

else if(flag==ERROR) printf(" 表%d不存在，已为您新建表%d并切换(未初始化)\n",list\_num,list\_num);

else if(flag==OVERFLOW) {//新表创建失败

printf(" 空间不足，线性表创建失败！\n");

list\_num=list\_num\_cur;//当前表序回退至换表前

}

//输出所有表序

printf("\n------------------- all list --------------------\n");

printf(" 当前共有%d个线性表可供操作\n 其表序为(依创建时间排序)：\n ",MulList\_head->list\_num);

for(MulList\_temp=MulList\_head;MulList\_temp->next!=NULL;MulList\_temp=MulList\_temp->next){//输出所有表序

printf("%d ",MulList\_temp->next->list\_num);

}

printf("\n---------------------- end ----------------------\n");

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("\n 欢迎下次再使用本系统！\n");

}//end of main()

/\*\*

\* 函数名称：InitiaList

\* 函数参数：线性表L的地址

\* 函数功能：构造一个空的线性表

\* 返回值：成功构造返回OK，否则返回ERROR

\*\*/

status IntiaList(LinkList \*L){

//构造一个空线性表

\*L = (LinkList)malloc(sizeof (LNode));

if(!(\*L)) return ERROR; //存储分配失败

(\*L) -> next = NULL; //next指针指空-线性表置空

return OK;

}

/\*\*

\* 函数名称：DestroyList

\* 函数参数：线性表L的地址

\* 函数功能：删除线性表

\* 返回值：成功销毁返回OK，否则返回ERROR

\*\*/

status DestroyList(LinkList \*L){

//销毁线性表

LinkList temp;

LinkList tail;

for(tail=\*L;tail->next!=NULL;){//释放结点空间

temp=tail;

tail=tail->next;

free(temp);

}

free(tail);

(\*L)=NULL;

return OK;

}

/\*\*

\* 函数名称：ClearList

\* 函数参数：线性表L的地址

\* 函数功能：重置线性表

\* 返回值：成功置空返回OK，否则返回ERROR

\*\*/

status ClearList(LinkList \*L){

//将L重置为空表

if((\*L)->next==NULL) return ERROR;//线性表已为空

LinkList temp;

LinkList tail;

for(tail=(\*L)->next;tail->next!=NULL;){//释放数据结点空间

temp=tail;

tail=tail->next;

free(temp);

}

free(tail);

(\*L)->next=NULL;//重置为空表

return OK;

}

/\*\*

\* 函数名称：ListEmpty

\* 函数参数：线性表L

\* 函数功能：判断线性表是否为空

\* 返回值：若L为空表则返回OK，否则返回ERROR

\*\*/

status ListEmpty(LinkList L){

//判断L是否为空表，是则返回OK，否则返回ERROR

if(!L->next) return OK;

return ERROR;

}

/\*\*

\* 函数名称：ListLength

\* 函数参数：线性表L的

\* 函数功能：计算线性表L中数据元素个数；

\* 返回值：成功返回L中数据元素个数，失败返回ERROR

\*\*/

int ListLength(LinkList L){

//计算顺序线性表长度

int length;

for(length=0;L->next!=NULL;L=L->next){

length++;

}

return length;

}

/\*\*

\* 函数名称：GetElem

\* 函数参数：线性表L，第i个数据，e为第i个数据；

\* 函数功能：查找并用e返回L中第i个元素的值；

\* 返回值：成功则返回OK，否则返回ERROR

\*\*/

status GetElem(LinkList L,int i,ElemType \*e){

//用e返回L中第i个数据元素的值

if(i<1 || i>ListLength(L)) return ERROR;//越界

for(;i>0;i--){

L=L->next;

}

(\*e)=L->data;

return OK;

}

/\*\*

\* 函数名称：LocateElem

\* 函数参数：线性表L，第i个数据，e为第i个数据；

\* 函数功能：查找L中与e相同数据所在的位序；用i返回该位序

\* 返回值：成功则返回OK，不存在则返回ERROR

\*\*/

status LocateElem(LinkList L,ElemType e,ElemType \*i){//简化过

//线性表L已存在，用i返回L中第一个e的位序。若这样的e不存在，返回ERROR

int temp;

int length = ListLength(L);//直接使用L遍历，故LISTLENGTH的值会变化，使用length存储线性表长度

for(temp=0;temp<length;temp++){

L=L->next;

if(L->data==e){

(\*i)=temp+1;

return OK;

}

}

return ERROR;//未找到

}

/\*\*

\* 函数名称：PriorElem

\* 函数参数：线性表L，查找的数据cur,前驱pre\_e；

\* 函数功能：查找L中与cur相同的第一个数据，并用pre\_e返回其前驱

\* 返回值：成功则返回OK；若无前驱以及其他情况则返回ERROR

\*\*/

status PriorElem(LinkList L,ElemType cur,ElemType \*pre\_e){

//若cur是L的数据元素且不是第一个，则用pre\_e返回它的前驱，否则操作失败，pre.e无定义

int cur\_e;//cur元素的位序

if(LocateElem(L,cur,&cur\_e)==OK){//若元素存在

if(cur\_e!=1){//不是第一个元素

GetElem(L,cur\_e-1,pre\_e);//用pre\_e返回前驱元素

return OK;

}

}

return ERROR;//元素不存在或其无前驱

}

/\*\*

\* 函数名称：NextElem

\* 函数参数：线性表L，查找的数据cur,后继next\_e；

\* 函数功能：查找L中与cur相同的第一个数据，并用next\_e返回其后继

\* 返回值：成功则返回OK；若无后继以及其他情况则返回ERROR

\*\*/

status NextElem(LinkList L,ElemType cur,ElemType \*next\_e){

//若cur是L的数据元素且不是最后一个，则用next\_e返回它的后继，否则操作失败，next.e无定义

int cur\_e;//cur元素的位序

if(LocateElem(L,cur,&cur\_e)==OK){//若元素存在

if(cur\_e!=ListLength(L)){//不是最后一个元素

GetElem(L,cur\_e+1,next\_e);//用next\_e返回后继元素

return OK;

}

}

return ERROR;

}

/\*\*

\* 函数名称：ListInsert

\* 函数参数：线性表L的地址，插入的位置i，插入的数据元素e。

\* 函数功能：在L的第i个位置之前插入新的数据元素e；若已插满，则按照分配增量，分配更大的空间。

\* 返回值：成功插入返回OK，否则返回ERROR

\*\*/

status ListInsert(LinkList \*L,int i,ElemType e){

//在顺序线性表L中第i个位置之前插入新的元素e

//i的合法值为1≤i≤ListLength+1

int j;//计数器

LinkList temp\_L=NULL;//临时指针，用于指向第i-1个元素

LinkList temp\_Insert=NULL;//待插入的结点

if (i<1 || i>ListLength(\*L)+1) return ERROR; //i不合法

for(temp\_L=(\*L),j=0;j<i-1;j++){//找到第i-1个结点

temp\_L=temp\_L->next;

}

temp\_Insert = (LinkList)malloc(sizeof(LNode));

if (temp\_Insert==NULL) return OVERFLOW; //分配空间失败

temp\_Insert -> data = e;

temp\_Insert -> next = temp\_L -> next;

temp\_L -> next = temp\_Insert;

return OK;

}

/\*\*

\* 函数名称：ListDelete

\* 函数参数：线性表L的地址，删除的位序i，指针e的地址。

\* 函数功能：删除L的第i个数据元素，用e返回其值。

\* 返回值：成功删除返回OK，否则返回ERROR

\*\*/

status ListDelete(LinkList \*L,int i,ElemType \*e)

{

//删除L的第i个数据元素，用e返回其值

int j;//计数器

LinkList temp\_L=NULL;//临时指针，用于指向第i-1个元素

LinkList temp\_Delete=NULL;//临时指针，用于指向待删除元素

if (i<1 || i>ListLength(\*L)) return ERROR; //i不合法

for(temp\_L=(\*L),j=0;j<i-1;j++){//找到第i-1个结点

temp\_L=temp\_L->next;

}

temp\_Delete = temp\_L->next;

(\*e)=temp\_Delete->data;

if (temp\_L -> next -> next){//如果被删除元素不是最后一个

temp\_L -> next = temp\_L -> next -> next;

}

else temp\_L -> next = NULL;//被删除元素是最后一个，则前一个结点指空

free(temp\_Delete);

return OK;

}

/\*\*

\* 函数名称：ListTrabverse

\* 函数参数：线性表L

\* 函数功能：依次显示线性表中的每个元素。

\* 返回值：成功遍历返回线性表的长度，否则返回ERROR

\*\*/

status ListTrabverse(LinkList L){

int i;//计数器

printf("\n----------------- all elements ------------------\n");

for(i=0;i<ListLength(L);) {

L=L->next;

printf("%d ",L->data);

}

printf("\n---------------------- end ----------------------\n");

return OK;

}

/\*\*

\* 函数名称：LoadFile

\* 函数参数：线性表L，文件名filename

\* 函数功能：将文件中数据读入，插入当前线性表。

\* 返回值：成功读取返回OK，否则返回ERROR

\*\*/

status LoadFile(LinkList L,char filename[20]){

ElemType elem\_read;//读取文件缓存

int i=0;//计数器

FILE \*fp;

fp = fopen(filename, "r");

/\*将文件中十进制字符形式元素依次按读入系统\*/

while(fscanf(fp,"%d ",&elem\_read)!=EOF){

printf(" 读入第%d个元素:%d...\n", ++i , elem\_read);

ListInsert(&L,i,elem\_read);//插入读到的数据

}

fclose(fp);

if(!i) return ERROR;

else return OK;

}

/\*\*

\* 函数名称：SaveAsFile

\* 函数参数：线性表L，文件名filename

\* 函数功能：将线性表中数据存入文件。

\* 返回值：若成功写入则返回OK，否则返回ERROR

\*\*/

status SaveAsFile(LinkList L,char filename[20]){

ElemType elem\_read;//读取文件缓存

int i;//计数器

FILE \*fp;

fp = fopen(filename, "wb+");

/\*将表中元素依次按十进制字符形式写入文件\*/

LinkList temp\_L=L;

for (i = 1; i <= ListLength(L); i++){

temp\_L = temp\_L->next;

printf(" 写入第%d个元素:%d...\n", i , temp\_L->data);

fprintf(fp, "%d ", temp\_L->data);

}

fclose(fp);

return OK;

}

/\*\*

\* 函数名称：MulList

\* 函数参数：线性表L的地址，多表表头M的地址，当前表序号list\_num\_cur，欲切换到的表序号list\_num

\* 函数功能：切换操作表。

\* 返回值：成功切换返回OK，否则返回ERROR

\*\*/

status MulList(LinkList \*L,MulNode \*M,int list\_num\_cur,int list\_num){

//切换操作表，若目标表不存在则新建

MulNode \*mul\_temp=NULL;//用于遍历多表结构

MulNode \*mul\_new=NULL;//用于新建多表结点

//存储表

for(mul\_temp=M->next;mul\_temp!=NULL;mul\_temp=mul\_temp->next){//给多表结构中的对应节点赋值，保存数据

if(mul\_temp->list\_num==list\_num\_cur){

mul\_temp->list\_head=(\*L);

break;

}

}

//切换表（为表头L赋值）

for(mul\_temp=M;mul\_temp->next!=NULL;mul\_temp=mul\_temp->next){//查询多表结构中的对应节点，切换操作表

if(mul\_temp->next->list\_num==list\_num){//若查询成功，读入表

(\*L) = mul\_temp->next->list\_head;

return OK;

}

}

//若查询失败，在多表结构中新建表，此时mul\_temp->next指针已经指向链表尾部NULL

mul\_new = (MulNode \*)malloc(sizeof(MulNode));

if (mul\_new==NULL) return OVERFLOW; //分配空间失败

mul\_new -> list\_num = list\_num;//赋值表序

mul\_new -> list\_head = NULL;//新表头节点置空

mul\_new -> next = NULL;//新节点next置空

mul\_temp -> next = mul\_new;//为新节点赋值

M->list\_num++;//记录多表个数+1

(\*L) = mul\_new -> list\_head;

return ERROR;

}

# 附录C 基于链式存储结构的二叉树实现的源程序

## 1.“bitree.h”头文件

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#define OK 1

#define ERROR -1

#define TRUE 1

#define FALSE -2

#define OVERFLOW -2

#define MAX 1000

typedef int status;

typedef char ElemType;//数据元素类型定义

typedef struct BiTNode{ //二叉树链式存储结构的定义

int key; //结点的唯一标识

ElemType data; //结点的数据

struct BiTNode \*lchild, \*rchild; //指向结点左右孩子的指针

}BiTNode, \*BiTree;

typedef struct seqqueue{//队列定义

BiTree data[MAX]; //队列大小

int front; //队列头

int rear; //队列尾

}seqqueue;

typedef struct MulNode{ //二叉树森林结构的定义

int list\_num; //操作表序号

BiTree list\_head; //操作表头指针

struct MulNode \*next; //指向下一个表头指针结点的指针

}MulNode;

status InitBiTree(BiTree \*T);//初始化

status DestroyBiTree(BiTree \*T);//销毁

status CreatBiTree(BiTree \*T,char \*definition);//创建char\*\*definition

status ClearBiTree(BiTree \*T);//清空

status BiTreeEmpty(BiTree T);//判定是否为空

int BiTreeDepth(BiTree T);//返回树T深度

status Root(BiTree T, int \*e, ElemType \*elem);//用elem返回T的根

status Value(BiTree T,int e, ElemType \*elem);//e是T中某个结点 返回e的值

status Assign(BiTree \*T,int e,ElemType elem);//结点赋值

BiTree Parent(BiTree T,int h\_e,int e);//获得双亲结点

BiTree LeftChild(BiTree T,int e);//获得左孩子

BiTree RightChild(BiTree T,int e);//获得右孩子

BiTree LeftSibling(BiTree T,int h\_e,int e);//获得左兄弟

BiTree RightSibling(BiTree T,int h\_e,int e);//获得右兄弟

status InsertChild(BiTree \*T,int e,int LR);//插入子树

status DeleteChild(BiTree \*T,int e,int LR);//删除子树

status PreOrderTraverse(BiTree T);//前序遍历

status InOrderTraverse(BiTree T);//中序遍历

status PostOrderTraverse(BiTree T);//后序遍历

status LevelOrderTraverse(BiTree T);//层次遍历

status LoadFile(BiTree \*T,char \*filename);//读取文件

status SaveAsFile(BiTree T,char \*filename);//存储文件

status MulList(BiTree \*T,MulNode \*M,int list\_num\_cur,int list\_num);//多表切换

## 2.“main.c”文件

#include"bitree.h"

int key\_static[MAX];//生成所有结点的key

int list\_num=1;//表序号

void main(void){

MulNode \*MulList\_head=NULL;//多表表头,其data域存储多表个数

MulNode \*MulList\_temp=NULL;//用于遍历多表结点输出所有表序

//初始化多表结构

MulList\_head = (MulNode\*)malloc(sizeof(MulNode));

MulList\_head->list\_num= 1;//表头data域存储多表个数

MulList\_head->list\_head=NULL;

MulList\_head->next=(MulNode\*)malloc(sizeof(MulNode));//新建表1

MulList\_head->next->list\_num=1;//表序为1

MulList\_head->next->list\_head=NULL;

MulList\_head->next->next=NULL;

int list\_num\_cur=0;//记录当前表序号

int flag;//多表操作中记录函数返回值

BiTree T=NULL;//当前二叉树头指针 lchild指向根节点

BiTree temp=NULL;//用于暂时存放查询到的结点

ElemType elem;//用于传递数据元素值

int key;//用于传递key值

int LR;//用于选择对左右子树进行操作

int tree\_num=0;//树编号（名字）

char definition[MAX];//用于creat树

char filename[MAX];

int op=1;

while(op){

fflush(stdin);

system("cls"); printf("\n\n");

printf(" Menu for Binary Tree On Chain Structure \n");

printf(" 当前操作的树序号为%d\n",list\_num);

printf("-------------------------------------------------\n");

printf(" 1. IntiaBitree 2. DestroyBiTree\n");

printf(" 3. CreatBiTree 4. ClearBiTree\n");

printf(" 5. BiTreeEmpty 6. BiTreeDepth\n");

printf(" 7. Root 8. Value\n");

printf(" 9. Assign\n");

printf("-------------------------------------------------\n");

printf(" 10. Parent 11. LeftChild\n");

printf(" 12. RightChild 13. LeftSibling\n");

printf(" 14. RightSibling\n");

printf("-------------------------------------------------\n");

printf(" 15. InsertChild 16. DeleteChild\n");

printf(" 17. PreOrderTraverse 18. InOrderTraverse\n");

printf(" 19. PostOrderTraverse 20. LevelOrderTraverse\n");

printf("-------------------------------------------------\n");

printf(" 21. LoadFile 22. SaveAsFile\n");

printf(" 23. MulTree \n");

printf("-------------------------------------------------\n");

printf(" 0. Exit\n");

printf("-------------------------------------------------\n");

printf(" 请选择你的操作[0~23]:");

scanf("%d",&op);

switch(op){

case 1:

//初始化

if(T) printf(" 二叉树已初始化，初始化失败！\n");

else{

if(InitBiTree(&T)==OK) printf(" 二叉树初始化成功！\n");

else printf(" 存储空间不足，初始化失败！\n");

}

getchar();getchar();

break;

case 2:

//销毁

if(!T) printf(" 二叉树未初始化！\n");

else{

if(DestroyBiTree(&T)==OK) printf(" 二叉树销毁成功！\n");

key\_static[list\_num]=0;//key重新生成

}

getchar();getchar();

break;

case 3:

//创建

if(!T) printf(" 二叉树未初始化！\n");

else{

printf(" 请输入字符串definition以先序方式新建二叉树\n 输入样例：abc de f (无需输入key值，将自动生成，欲查询结点对应key值可遍历二叉树)\n (其中空格代表该处为空结点)：");

definition[0]='\0';

getchar();

scanf("%[^\n]",definition);

if(definition[0]=='\0'){

printf(" 输入definition序列不能为空！\n");

}

else if(CreatBiTree(&T,&definition)==OK) printf(" 二叉树创建成功！\n");

else printf(" 创建失败，字符串内元素未全部使用，请检查定义是否符合规则！\n");

}

getchar();getchar();

break;

case 4:

//清空

if(!T) printf(" 二叉树未初始化！\n");

else{

if(ClearBiTree(&T)==OK) printf(" 二叉树置空成功！\n");

else printf(" 二叉树已为空，置空失败！\n");

key\_static[list\_num]=0;//key重新生成

}

getchar();getchar();

break;

case 5:

//判断是否为空

if(!T) printf(" 二叉树未初始化！\n");//表不存在

else if(BiTreeEmpty(T)==OK) printf(" 二叉树为空！\n");

else printf(" 二叉树不为空！\n");

getchar();getchar();

break;

case 6:

//求二叉树最大深度

if(!T) printf(" 二叉树未初始化！\n");//表不存在

else{

printf(" 二叉树最大深度为%d！\n",BiTreeDepth(T));

}

getchar();getchar();

break;

case 7:

//根的值

if(!T) printf(" 二叉树未初始化！\n");//表不存在

else{

if(Root(T,&key,&elem)==OK) printf(" 二叉树的根key为%d,数据为%c",key,elem);

else printf(" 该二叉树为空树，无根！\n");

}

getchar();getchar();

break;

case 8:

//关键字为key的结点的数据

if(!T) printf(" 二叉树未初始化！\n");//树不存在

else{

printf(" 请输入关键字key，将为您返回该结点数据\n\n");

printf(" key：");

scanf("%d",&key);

if(Value(T,key,&elem)==OK) printf(" 线性表中key为%d的数据元素值为%c\n",key,elem);

else printf(" 不存在key为%d的结点！\n",key);

}

getchar();getchar();

break;

case 9:

//给关键字为key的结点赋值

if(!T) printf(" 二叉树未初始化！\n");//树不存在

else{

printf(" 请输入关键字key与数据elem，将为该结点赋值\n\n");

printf(" key：");

scanf("%d",&key);

getchar();

printf(" elem：");

scanf("%c",&elem);

if(Assign(&T,key,elem)==OK) printf(" 线性表中key为%d的数据元素值更改成功\n",key);

else printf(" 不存在key为%d的结点！\n",key);

}

getchar();getchar();

break;

case 10:

//查找Parent结点

if(!T) printf(" 二叉树未初始化！\n");//树不存在

else{

printf(" 请输入关键字key，将为您返回该结点的双亲\n\n");

printf(" key：");

scanf("%d",&key);

temp=Parent(T->lchild,T->lchild->key,key);

if(temp!=NULL) printf(" 线性表中key为%d的结点的双亲结点key为%d，data为%c\n",key,temp->key,temp->data);

else printf(" 不存在key为%d的结点或该结点为根节点无双亲！\n",key);

}

temp=NULL;//置空暂存结点

getchar();getchar();

break;

case 11:

//查找LeftChild结点

if(!T) printf(" 二叉树未初始化！\n");//树不存在

else{

printf(" 请输入关键字key，将为您返回该结点的左子结点\n\n");

printf(" key：");

scanf("%d",&key);

temp=LeftChild(T->lchild,key);

if(temp!=NULL) printf(" 线性表中key为%d的结点的左孩子结点key为%d，data为%c\n",key,temp->key,temp->data);

else printf(" 不存在key为%d的结点或该结点无左子结点！\n",key);

}

temp=NULL;

getchar();getchar();

break;

case 12:

//查找RightChild结点

if(!T) printf(" 二叉树未初始化！\n");//树不存在

else{

printf(" 请输入关键字key，将为您返回该结点的右子结点\n\n");

printf(" key：");

scanf("%d",&key);

temp=RightChild(T->lchild,key);

if(temp!=NULL) printf(" 线性表中key为%d的结点的右孩子结点key为%d，data为%c\n",key,temp->key,temp->data);

else printf(" 不存在key为%d的结点或该结点无右子结点！\n",key);

}

temp=NULL;

getchar();getchar();

break;

case 13:

//查找LeftSiBling结点

if(!T) printf(" 二叉树未初始化！\n");//树不存在

else{

printf(" 请输入关键字key，将为您返回该结点的左兄弟结点\n\n");

printf(" key：");

scanf("%d",&key);

temp=LeftSibling(T->lchild,T->lchild->key,key);

if(temp!=NULL) printf(" 线性表中key为%d的结点的左兄弟结点key为%d，data为%c\n",key,temp->key,temp->data);

else printf(" 不存在key为%d的结点或该结点无左兄弟结点！\n",key);

}

temp=NULL;

getchar();getchar();

break;

case 14:

//查找RightSiBling结点

if(!T) printf(" 二叉树未初始化！\n");//树不存在

else{

printf(" 请输入关键字key，将为您返回该结点的右兄弟结点\n\n");

printf(" key：");

scanf("%d",&key);

temp=RightSibling(T->lchild,T->lchild->key,key);

if(temp!=NULL) printf(" 线性表中key为%d的结点的右兄弟结点key为%d，data为%c\n",key,temp->key,temp->data);

else printf(" 不存在key为%d的结点或该结点无右兄弟结点！\n",key);

}

temp=NULL;

getchar();getchar();

break;

case 15:

//插入子树

if(!T) printf(" 二叉树未初始化！\n");//树不存在

else{

printf(" 请输入关键字key与选项LR，将为您插入在key结点处子树\n （LR=0：插入至左子树；LR=1：插入至右子树）\n\n");

printf(" key：");

scanf("%d",&key);

getchar();

printf(" LR：");

scanf("%d",&LR);

if(LR!=0&&LR!=1) printf(" 输入LR有误，返回主菜单！\n");

else if(InsertChild(&T,key,LR)==OK) printf(" 插入成功！\n");

else printf(" 插入失败！\n");

}

getchar();getchar();

break;

case 16:

//删除子树

if(!T) printf(" 二叉树未初始化！\n");//树不存在

else{

printf(" 请输入关键字key与选项LR，将为您删除key结点子树\n （LR=0：删除左子树；LR=1：删除右子树）\n\n");

printf(" key：");

scanf("%d",&key);

getchar();

printf(" LR：");

scanf("%d",&LR);

if(LR!=0&&LR!=1) printf(" 输入LR有误，返回主菜单！\n");

else if(DeleteChild(&T,key,LR)==OK) printf(" 删除成功！\n");

else printf(" 删除失败！\n");

}

getchar();getchar();

break;

case 17:

//先序遍历输出二叉树中数据

if(!T) printf(" 二叉树未初始化！\n");//树不存在

else if(!T->lchild) {//是否为空树

printf(" 二叉树为空！\n");

}

else{

printf(" 二叉树的先序遍历结果：\n");

printf("--------------------------------------------\n");

PreOrderTraverse(T->lchild);

printf("--------------------------------------------\n");

}

getchar();getchar();

break;

case 18:

//中序遍历输出二叉树中数据

if(!T) printf(" 二叉树未初始化！\n");//树不存在

else if(!T->lchild) {//是否为空树

printf(" 二叉树为空！\n");

}

else{

printf(" 二叉树的中序遍历结果：\n");

printf("--------------------------------------------\n");

InOrderTraverse(T->lchild);

printf("--------------------------------------------\n");

}

getchar();getchar();

break;

case 19:

//后序遍历输出二叉树中数据

if(!T) printf(" 二叉树未初始化！\n");//树不存在

else if(!T->lchild) {//是否为空树

printf(" 二叉树为空！\n");

}

else{

printf(" 二叉树的后序遍历结果：\n");

printf("--------------------------------------------\n");

PostOrderTraverse(T->lchild);

printf("--------------------------------------------\n");

}

getchar();getchar();

break;

case 20:

//层次遍历输出二叉树中数据

if(!T) printf(" 二叉树未初始化！\n");//树不存在

else if(!T->lchild) {//是否为空树

printf(" 二叉树为空！\n");

}

else{

printf(" 二叉树的层次遍历结果：\n");

printf("--------------------------------------------\n");

LevelOrderTraverse(T->lchild);

printf("--------------------------------------------\n");

}

getchar();getchar();

break;

case 21:

//读取文件

if(!T){

printf(" 二叉树未初始化！\n");//树不存在

getchar();

}

else{

printf(" 将从文件中读取数据并覆盖当前二叉树.\n 请输入要读取的文件名: ");getchar();

gets(filename);

if(LoadFile(&T,filename)==OK) printf(" 文件读取成功！\n");

else printf(" 文件读取失败！\n");

}

getchar();

break;

case 22:

//存储文件

if(!T){

printf(" 二叉树未初始化！\n");//树不存在

getchar();

}

else if(!T->lchild){

printf(" 二叉树为空！无法存储\n");

getchar();

}

else{

printf(" 将二叉树中数据保存(写)为文件\n 请输入要写入的文件名: ");getchar();

gets(filename);

if(SaveAsFile(T->lchild,filename)==OK) printf(" 文件写入成功！\n");

}

getchar();

break;

case 23:

list\_num\_cur=list\_num;//记录当前表序以存储数据

printf(" 请输入要切换到的树序号：");

scanf("%d",&list\_num);

flag=MulList(&T,MulList\_head,list\_num\_cur,list\_num);

if(flag==OK) printf(" 切换成功，当前操作树为树%d\n",list\_num);

else if(flag==ERROR) printf(" 树%d不存在，已为您新建树%d并切换(未初始化)\n",list\_num,list\_num);

else if(flag==OVERFLOW) {//新树创建失败

printf(" 空间不足，树创建失败！\n");

list\_num=list\_num\_cur;//当前表序回退至换表前

}

//输出所有树序号

printf("\n------------------- all list --------------------\n");

printf(" 当前共有%d个树可供操作\n 其序号为(依创建时间排序)：\n ",MulList\_head->list\_num);

for(MulList\_temp=MulList\_head;MulList\_temp->next!=NULL;MulList\_temp=MulList\_temp->next){//输出所有表序

printf("%d ",MulList\_temp->next->list\_num);

}

printf("\n---------------------- end ----------------------\n");

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("\n 欢迎下次再使用本系统！\n");

}//end of main()

/\*\*

\* 函数名称：InitBiTree

\* 函数参数：二叉树T的地址

\* 函数功能：构造一个空的二叉树

\* 返回值：成功构造返回OK，否则返回ERROR

\*\*/

status InitBiTree(BiTree \*T)

{

//构造一个二叉树

\*T = (BiTree)malloc(sizeof(BiTNode));

if(!(\*T)) return ERROR; //存储分配失败

(\*T)->lchild=NULL; //二叉树头结点置空

(\*T)->rchild=NULL; //右子结点置空

return OK;

}

/\*\*

\* DestroyBiTree的子函数

\*\*/

void DestroyF(BiTree \*T)

{

//递归释放结点空间

if((\*T)->lchild) DestroyF(&((\*T)->lchild));

if((\*T)->rchild) DestroyF(&((\*T)->rchild));

free(\*T);

}

/\*\*

\* 函数名称：DestroyBiTree

\* 函数参数：二叉树T的地址

\* 函数功能：删除二叉树

\* 返回值：成功删除返回OK，否则返回ERROR

\*\*/

status DestroyBiTree(BiTree \*T)

{

//销毁二叉树

DestroyF(T);

(\*T)=NULL;

return OK;

}

/\*\*

\* CreatBiTree的子函数

\*\*/

void CreatF(BiTree \*T,char \*\*d)

{

//递归赋值

if (\*\*d ==' ') {

(\*T)=NULL;

(\*d)++;

}

else

{

(\*T)=(BiTree)malloc(sizeof(BiTNode));

(\*T)->key =key\_static[list\_num]++;

(\*T)->data=\*((\*d)++); //生成根结点

(\*T)->lchild = NULL;

(\*T)->rchild = NULL;

if(\*\*d!='\0')CreatF(&(\*T)->lchild,d); //递归构造左子树

if(\*\*d!='\0')CreatF(&(\*T)->rchild,d); //递归构造右子树

}

}

/\*\*

\* 函数名称：CreatBiTree

\* 函数参数：二叉树T的地址，先序序列definition

\* 函数功能：根据先序序列definition创建二叉树

\* 返回值：成功创建返回OK，否则返回ERROR

\*\*/

status CreatBiTree(BiTree \*T,char \*definition)

{

//以definition规则 先序创建一个二叉树

CreatF(&(\*T)->lchild,&definition);//从根结点创建

if(\*definition!='\0'){

ClearBiTree(T);

return ERROR;

}

return OK;

}

/\*\*

\* 函数名称：ClearBiTree

\* 函数参数：二叉树T的地址

\* 函数功能：清空二叉树

\* 返回值：成功清空返回OK，否则返回ERROR

\*\*/

status ClearBiTree(BiTree \*T)

{

//清空二叉树

if(!(\*T)->lchild) return ERROR;//树已为空

else{

DestroyF(&((\*T)->lchild)); //销毁数据域

((\*T)->lchild)=NULL; //头结点置空

return OK;

}

}

/\*\*

\* 函数名称：BiTreeEmpty

\* 函数参数：二叉树T

\* 函数功能：判断二叉树是否为空

\* 返回值：二叉树为空返回OK，否则返回ERROR

\*\*/

status BiTreeEmpty(BiTree T)

{

if(T->lchild==NULL) return OK;

else return ERROR;

}

/\*\*

\* 函数名称：BiTreeDepth

\* 函数参数：二叉树T

\* 函数功能：求取二叉树深度

\* 返回值：返回二叉树深度，求取失败返回ERROR

\*\*/

int BiTreeDepth(BiTree T)

{

//返回二叉树最大深度

if(T->lchild&&T->rchild)

return 1+(BiTreeDepth(T->lchild)>BiTreeDepth(T->rchild)?BiTreeDepth(T->lchild):BiTreeDepth(T->rchild));

if(T->lchild)

return 1+BiTreeDepth(T->lchild);

if(T->rchild)

return 1+BiTreeDepth(T->rchild);

return 0;

}

/\*\*

\* 函数名称：Root

\* 函数参数：二叉树T，结点位序key，结点数据elem

\* 函数功能：用key与elem返回二叉树的头结点

\* 返回值：返回二叉树头结点成功则返回OK，失败则返回ERROR

\*\*/

status Root(BiTree T, int \*key, ElemType \*elem)

{

//返回根的值

if(T->lchild){//返回根值

(\*key)=T->lchild->key;

(\*elem)=T->lchild->data;

return OK;

}

return ERROR;//空树

}

/\*\*

\* Value的子函数

\*\*/

void ValueF(BiTree T,int key, ElemType \*elem)

{

//递归搜索关键字为key的结点

if(T->key==key) (\*elem)=T->data;

if(T->lchild) ValueF(T->lchild,key,elem);

if(T->rchild) ValueF(T->rchild,key,elem);

}

/\*\*

\* 函数名称：Value

\* 函数参数：二叉树T，结点位序key，结点数据elem

\* 函数功能：用elem返回位序为key的结点的数据

\* 返回值：返回二叉树结点数据成功则返回OK，失败则返回ERROR

\*\*/

status Value(BiTree T,int key, ElemType \*elem)

{

//返回关键字为key的结点的数据

(\*elem)='\0';

ValueF(T->lchild,key,elem);

if((\*elem)=='\0') return ERROR;//未找到

return OK;

}

/\*\*

\* Assign的子函数

\*\*/

void AssignF(BiTree \*T,int key, ElemType \*elem)

{

//递归更改关键字为key的结点

if((\*T)->key==key) {

(\*T)->data=(\*elem);

(\*elem)='\0';

}

if((\*T)->lchild) AssignF(&(\*T)->lchild,key,elem);

if((\*T)->rchild) AssignF(&(\*T)->rchild,key,elem);

}

/\*\*

\* 函数名称：Assign

\* 函数参数：二叉树T的地址，结点位序key，结点数据elem

\* 函数功能：用elem为位序为key的结点赋值

\* 返回值：为二叉树结点赋值成功返回OK，失败则返回ERROR

\*\*/

status Assign(BiTree \*T,int key,ElemType elem)

{

//结点赋值

AssignF(T,key,&elem);

if (elem=='\0') return OK;

return ERROR;//赋值失败

}

/\*\*

\* 函数名称：Parent

\* 函数参数：二叉树T，结点双亲位序key，结点位序key

\* 函数功能：查找结点的双亲

\* 返回值：若查找成功返回双亲结点地址，否则返回NULL

\*\*/

BiTree Parent(BiTree T,int head\_key,int key)

{

if(key==head\_key) return NULL;//头结点无双亲返回NULL

if((T->lchild&&T->lchild->key==key)||(T->rchild)&&(T->rchild->key==key)){

return T;

}

//当前未找到 则递归搜索子节点关键字为key的结点

BiTree temp=NULL;

if(T->lchild) temp = Parent(T->lchild,head\_key,key);

if(temp) return temp;

if(T->rchild) temp = Parent(T->rchild,head\_key,key);

if(temp) return temp;

//未找到

return NULL;

}

/\*\*

\* 函数名称：LeftChild

\* 函数参数：二叉树T，结点位序key

\* 函数功能：查找结点的左孩子

\* 返回值：若查找成功返回左孩子结点地址，否则返回NULL

\*\*/

BiTree LeftChild(BiTree T,int key)

{

if(T->key==key){//节点为所寻key 返回左孩子

return T->lchild;//左孩子可能为空

}

//当前未找到 则递归搜索节点关键字为key的结点

BiTree temp=NULL;

if(T->lchild) temp = LeftChild(T->lchild,key);

if(temp) return temp;

if(T->rchild) temp = LeftChild(T->rchild,key);

if(temp) return temp;

return NULL;

}

/\*\*

\* 函数名称：RightChild

\* 函数参数：二叉树T，结点位序key

\* 函数功能：查找结点的右孩子

\* 返回值：若查找成功返回右孩子结点地址，否则返回NULL

\*\*/

BiTree RightChild(BiTree T,int key)

{

if(T->key==key){//节点为所寻key 返回右孩子

return T->rchild;//右孩子可能为空

}

//当前未找到 则递归搜索节点关键字为key的结点

BiTree temp=NULL;

if(T->lchild) temp = RightChild(T->lchild,key);

if(temp) return temp;

if(T->rchild) temp = RightChild(T->rchild,key);

if(temp) return temp;

return NULL;

}

/\*\*

\* 函数名称：LeftSibling

\* 函数参数：二叉树T，结点双亲位序，结点位序key

\* 函数功能：查找结点的左兄弟

\* 返回值：若查找成功返回左兄弟结点地址，否则返回NULL

\*\*/

BiTree LeftSibling(BiTree T,int head\_key,int key)

{

//寻找左兄弟

if(key==head\_key) return NULL;//头结点无兄弟结点

if(T->rchild&&T->rchild->key==key){//右子节点为所寻key 返回左子结点

return T->lchild;//左兄弟可能为空

}

//当前未找到 则递归搜索右子节点关键字为key的结点

BiTree temp=NULL;

if(T->lchild) temp = LeftSibling(T->lchild,head\_key,key);

if(temp) return temp;

if(T->rchild) temp = LeftSibling(T->rchild,head\_key,key);

if(temp) return temp;

return NULL;

}

/\*\*

\* 函数名称：RightSibling

\* 函数参数：二叉树T，结点双亲位序，结点位序key

\* 函数功能：查找结点的右兄弟

\* 返回值：若查找成功返回右兄弟结点地址，否则返回NULL

\*\*/

BiTree RightSibling(BiTree T,int head\_key,int key)

{

//寻找右兄弟

if(key==head\_key) return NULL;//头结点无兄弟结点

if(T->lchild&&T->lchild->key==key){//左子节点为所寻key 返回右子结点

return T->rchild;//右兄弟可能为空

}

//当前未找到 则递归搜索左子节点关键字为key的结点

BiTree temp=NULL;

if(T->lchild) temp = RightSibling(T->lchild,head\_key,key);

if(temp) return temp;

if(T->rchild) temp = RightSibling(T->rchild,head\_key,key);

if(temp) return temp;

return NULL;

}

/\*\*

\* InsertChild的子函数

\*\*/

BiTree Node(BiTree T, int key)

{

//寻找结点

if (T&&T->key==key) return T;

BiTree temp = NULL;

if (T->lchild) temp = Node(T->lchild, key);

if (temp) return temp;

if (T->rchild) temp = Node(T->rchild, key);

if (temp) return temp;

return NULL;

}

/\*\*

\* InsertChild的子函数

\*\*/

status CreatInsertTree(BiTree \*T\_Insert,char \*definition)

{

//以definition规则 先序创建一个二叉树

CreatF(T\_Insert,&definition);//从根结点创建

if(\*definition!='\0'){

ClearBiTree(T\_Insert);

return ERROR;

}

return OK;

}

/\*\*

\* 函数名称：InsertChild

\* 函数参数：二叉树T的地址，插入结点位序key，左或右子树判断LR

\* 函数功能：在key结点下根据LR的取值插入左或右子树

\* 返回值：若插入子树成功返回OK，否则返回ERROR

\*\*/

status InsertChild(BiTree \*T,int key,int LR)//插入子树

{

BiTree T\_Insert = NULL;//待插入子树的头结点（无头指针）

char definition\_insert[MAX];

printf(" 请输入用于生成子树的规则(definition)：\n ");

getchar();

definition\_insert[0]='\0';

scanf("%[^\n]",definition\_insert);

if(\*definition\_insert=='\0'){

printf(" 输入definition序列不能为空！");

return ERROR;

}

if(CreatInsertTree(&T\_Insert,definition\_insert)==OK){//创建待插入子树

if(T\_Insert->rchild) {

printf(" 创建得到的子树根节点应无右孩子，请检查创建的子树！\n");

return ERROR;

}

if (Node(\*T,key)==NULL) {

printf(" 未找到key对应结点！\n");

return ERROR;

}

if(LR==0){//插入为左子树

T\_Insert->rchild = Node(\*T,key)->lchild;

Node(\*T,key)->lchild = T\_Insert;

}

if(LR==1){//插入右子树

T\_Insert->rchild = Node(\*T,key)->rchild;

Node(\*T,key)->rchild = T\_Insert;

}

return OK;

}

else{

printf(" 创建子树失败，请检查输入序列！\n");

return ERROR;

}

}

/\*\*

\* 函数名称：DeleteChild

\* 函数参数：二叉树T的地址，删除结点位序key，左或右子树判断LR

\* 函数功能：在key结点下根据LR的取值删除左或右子树

\* 返回值：若删除子树成功返回OK，否则返回ERROR

\*\*/

status DeleteChild(BiTree \*T, int key, int LR)//删除子树

{

BiTree temp = Node(\*T, key);

if (temp == NULL) {

printf(" 未找到对应结点！\n");

return ERROR;

}

if (LR == 0) {//删除左子树

if (temp->lchild == NULL) {

printf(" key对应结点左子树已为空！\n");

return ERROR;

}

DestroyBiTree(&temp->lchild);

}

if (LR == 1) {//删除右子树

if (temp->rchild == NULL) {

printf(" key对应结点右子树已为空！\n");

return ERROR;

}

DestroyBiTree(&temp->rchild);

}

return OK;

}

/\*\*

\* 函数名称：PreOrderTraverse

\* 函数参数：二叉树T

\* 函数功能：前序遍历输出二叉树结点值

\* 返回值：若输出成功返回OK，否则返回ERROR

\*\*/

status PreOrderTraverse(BiTree T)//前序遍历

{

printf(" key:%d data:%c\n",T->key,T->data);

if(T->lchild) PreOrderTraverse(T->lchild);

if(T->rchild) PreOrderTraverse(T->rchild);

return OK;

}

/\*\*

\* 函数名称：InOrderTraverse

\* 函数参数：二叉树T

\* 函数功能：中序遍历输出二叉树结点值

\* 返回值：若输出成功返回OK，否则返回ERROR

\*\*/

status InOrderTraverse(BiTree T)//中序遍历

{

if(T->lchild) InOrderTraverse(T->lchild);

printf(" key:%d data:%c\n",T->key,T->data);

if(T->rchild) InOrderTraverse(T->rchild);

return OK;

}

/\*\*

\* 函数名称：PostOrderTraverse

\* 函数参数：二叉树T

\* 函数功能：后序遍历输出二叉树结点值

\* 返回值：若输出成功返回OK，否则返回ERROR

\*\*/

status PostOrderTraverse(BiTree T)//后序遍历

{

if(T->lchild) PostOrderTraverse(T->lchild);

if(T->rchild) PostOrderTraverse(T->rchild);

printf(" key:%d data:%c\n",T->key,T->data);

return OK;

}

/\*\*

\* LevelOrderTraverse的子函数

\*\*/

void enter\_seqqu(seqqueue \*Q,BiTree T)

{

//进队列

if(Q->rear==MAX){

printf("队列已满，二叉树过大\n");

}

else{

Q->data[Q->rear]=T;

Q->rear++;

}

}

/\*\*

\* LevelOrderTraverse的子函数

\*\*/

BiTree exit\_seqqu(seqqueue \*Q)

{

//出队列

if(Q->front == Q->rear){//队列已为空

return NULL;

}

else{

Q->front++;

return Q->data[Q->front-1];

}

}

/\*\*

\* 函数名称：LevelOrderTraverse

\* 函数参数：二叉树T

\* 函数功能：层次遍历输出二叉树结点值

\* 返回值：若输出成功返回OK，否则返回ERROR

\*\*/

status LevelOrderTraverse(BiTree T)//传入头结点

{//层次遍历

seqqueue Q;

BiTree temp;

Q.front=0;

Q.rear=0;

enter\_seqqu(&Q,T);//树头结点进队列

while(Q.front!=Q.rear){

temp=exit\_seqqu(&Q);//出队列并保存其值

printf(" key:%d data:%c\n",temp->key,temp->data);

if(temp->lchild){//左子结点进队列

enter\_seqqu(&Q,temp->lchild);

}

if(temp->rchild){//右子结点进队列

enter\_seqqu(&Q,temp->rchild);

}

}

return OK;

}

/\*\*

\* 函数名称：LoadList

\* 函数参数：二叉树T的地址，文件名filename

\* 函数功能：数据加载，将文件中数据读入，覆盖当前二叉树

\* 返回值：加载成功返回OK，加载失败返回ERROR

\*\*/

status LoadFile(BiTree \*T,char \*filename)

{

BiTree temp=(BiTree)malloc(sizeof(BiTNode));

temp -> lchild = NULL;

char definition\_load[MAX];//读取缓冲区

definition\_load[0]='\0';//初始化

FILE \*fp;

fp = fopen(filename, "r");

if(fp==NULL) {

printf("文件不存在！\n");

return ERROR;

}

/\*将文件中数据读入系统\*/

fscanf(fp,"%[^\n]",definition\_load);//读取文件

if(CreatBiTree(&temp,definition\_load)==OK){//覆盖当前树

key\_static[list\_num]=0;

ClearBiTree(T);

CreatBiTree(T,definition\_load);

fclose(fp);

return OK;

}

else{

printf(" 序列definition中数据未全部使用！\n");

}

fclose(fp);

return ERROR;

}

/\*\*

\* SaveAsFile的子函数

\*\*/

void Save\_PreOrderTraverse(BiTree T,char \*\*d)//前序遍历将树转换为字符串序列

{

\*((\*d)++) = T->data;

if(!T->lchild) \*((\*d)++) = ' ';

if(T->lchild) Save\_PreOrderTraverse(T->lchild,d);

if(!T->rchild) \*((\*d)++) = ' ';

if(T->rchild) Save\_PreOrderTraverse(T->rchild,d);

}

/\*\*

\* 函数名称：SaveAsFile

\* 函数参数：二叉树T的地址，文件名filename

\* 函数功能：数据保存，将二叉树中数据保存至文件

\* 返回值：保存成功返回OK，加载失败返回ERROR

\*\*/

status SaveAsFile(BiTree T,char \*filename)

{

char \*save\_definition=(char\*)malloc(1000\*sizeof(char));

\*save\_definition='\0';

char \*\*d=save\_definition;

Save\_PreOrderTraverse(T,&d);

\*d='\0';

FILE \*fp;

fp = fopen(filename, "wb+");

/\*将树中元素依次写入文件\*/

fprintf(fp, "%s", save\_definition);

fclose(fp);

return OK;

}

/\*\*

\* 函数名称：MulList

\* 函数参数：二叉树T的地址，森林表头M的地址，当前树序号list\_num\_cur，欲切换到的树序号list\_num

\* 函数功能：切换操作树。

\* 返回值：成功切换返回OK，否则返回ERROR

\*\*/

status MulList(BiTree \*T,MulNode \*M,int list\_num\_cur,int list\_num){

//切换操作树，若目标表不存在则新建

MulNode \*mul\_temp=NULL;//用于遍历多表结构

MulNode \*mul\_new=NULL;//用于新建多表结点

//存储表

for(mul\_temp=M->next;mul\_temp!=NULL;mul\_temp=mul\_temp->next){//给多表结构中的对应节点赋值，保存数据

if(mul\_temp->list\_num==list\_num\_cur){

mul\_temp->list\_head=(\*T);

break;

}

}

//切换表（为表头T赋值）

for(mul\_temp=M;mul\_temp->next!=NULL;mul\_temp=mul\_temp->next){//查询多表结构中的对应节点，切换操作表

if(mul\_temp->next->list\_num==list\_num){//若查询成功，读入表

(\*T) = mul\_temp->next->list\_head;

return OK;

}

}

//若查询失败，在多表结构中新建表，此时mul\_temp->next指针已经指向链表尾部NULL

mul\_new = (MulNode \*)malloc(sizeof(MulNode));

if (mul\_new==NULL) return OVERFLOW; //分配空间失败

mul\_new -> list\_num = list\_num;//赋值表序

mul\_new -> list\_head = NULL;//新表头节点置空

mul\_new -> next = NULL;//新节点next置空

mul\_temp -> next = mul\_new;//为新节点赋值

M->list\_num++;//记录多表个数+1

(\*T) = mul\_new -> list\_head;

return ERROR;

}

# 附录D 基于邻接表存储结构的图实现的源程序

## 1.“graph.h”头文件

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#include<string.h>

#define OK 1

#define ERROR -1

#define TRUE 1

#define FALSE -2

#define OVERFLOW -3

#define MAX 1000

#define MAX\_VERTEX\_NUM 30

typedef int status;

typedef char VertexType;//数据元素类型定义

typedef struct ArcNode{ //图邻接表式存储结构的定义

int adjvex; //该弧所指向的顶点的位置

struct ArcNode \*nextarc; //指向下一条弧的指针

int info; //该弧相关信息的指针

}ArcNode;

typedef struct VNode{//图顶点

VertexType data; //顶点信息

ArcNode \*firstarc; //指向第一条依附该顶点的弧的指针

}VNode,AdjList[MAX\_VERTEX\_NUM];

typedef struct{//图头结点

AdjList vertices;

int vexnum,arcnum; //图的当前顶点数和弧数

//int kind; //图的种类标志

}\*ALGraph,ALNode;

typedef struct Arc\_Relation{//弧关系

char tail; //弧尾数据

char head; //弧头数据

int weight; //弧的权

struct Arc\_Relation \*next; //指向下一个关系

}Relation;

typedef struct MulNode{ //多图结构的定义

int list\_num; //操作图序号

ALGraph list\_head; //操作图头指针

struct MulNode \*next; //指向下一个图头指针结点的指针

}MulNode;

void ScanfVR(Relation \*\*VR);//读取VR 构建(链式)边关系

void ScanfVR\_insert(Relation \*\*VR,char tail);//插入时读取VR 构建(链式)边关系

status CreateGraph(ALGraph \*G, char \*Vertex, Relation \*VR);//创建

status DestroyGraph(ALGraph \*G);//销毁

status LocateVex(ALGraph G,int \*num,VertexType data);//查找图G中U(data)的位置信息

status GetVex(ALGraph G,int num,VertexType \*data);//获得(位置为num的)顶点值

status PutVex(ALGraph \*G,int num,VertexType value);//对v(num)赋值value

VNode\* FirstAdjVex(ALGraph G,int num);//获得第一邻接点

VNode\* NextAdjVex(ALGraph G,int num,int num\_w);//获得num相对num\_v的下一邻接点

status InsertVex(ALGraph \*G,VertexType data,Relation \*VR);//插入顶点

status DeleteVex(ALGraph \*G,int num);//删除顶点

status InsertArc(ALGraph \*G,Relation \*VR);//插入弧

status DeleteArc(ALGraph \*G,int num\_v,int num\_w);//删除弧

status DFSTraverse(ALGraph G);//深度优先搜索遍历

status BFSTraverse(ALGraph G);//广度优先搜索遍历

status LoadFile(ALGraph \*G,char \*filename);//读取文件

status SaveAsFile(ALGraph G,char \*filename);//存储文件

status MulList(ALGraph \*T,MulNode \*M,int list\_num\_cur,int list\_num);//多图切换

## 2.“main.c”文件

#include"graph.h"

void main(void){

MulNode \*MulList\_head=NULL;//多表表头,其data域存储多表个数

MulNode \*MulList\_temp=NULL;//用于遍历多表结点输出所有表序

//初始化多表结构

MulList\_head = (MulNode\*)malloc(sizeof(MulNode));

MulList\_head->list\_num= 1;//表头data域存储多表个数

MulList\_head->list\_head=NULL;

MulList\_head->next=(MulNode\*)malloc(sizeof(MulNode));//新建表1

MulList\_head->next->list\_num=1;//表序为1

MulList\_head->next->list\_head=NULL;

MulList\_head->next->next=NULL;

int list\_num=1;//图编号（名字）

int list\_num\_cur=0;//记录当前图序号

int flag;//多表操作中记录函数返回值

ALGraph G=NULL;//当前图头结点

char Vertex[MAX];//顶点集合

Vertex[0]='\0';//初始化

Relation \*VR=NULL;//VR边关系的头结点

int i,j,k,flag\_c;//计数器，用于检验vertex顶点集是否重复

int num;//顶点v的位序num

int num\_w;//顶点w的位序num

char filename[MAX];//文件名称

VertexType data;//返回数据

VertexType value;//用于赋值的顶点值

VNode \*temp=NULL;//返回查找到的顶点

int op=1;

while(op){

fflush(stdin);

system("cls"); printf("\n\n");

printf(" Menu for Graph On Adjacency List \n");

printf(" 当前操作的图序号为%d\n",list\_num);

printf("-------------------------------------------------\n");

printf(" 1. CreateGraph 2. DestroyGraph\n");

printf(" 3. LocateVex 4. GetVex\n");

printf(" 5. PutVex 6. FirstAdjVex\n");

printf(" 7. NextAdjVex\n");

printf("-------------------------------------------------\n");

printf(" 8. InsertVex 9. DeleteVex\n");

printf(" 10. InsertArc 11. DeleteArc\n");

printf("-------------------------------------------------\n");

printf(" 12. DFSTraverse 13. BFSTraverse\n");

printf("-------------------------------------------------\n");

printf(" 14. LoadFile 15. SaveAsFile\n");

printf(" 16. MulGraph\n");

printf("-------------------------------------------------\n");

printf(" 0. Exit\n");

printf("-------------------------------------------------\n");

printf(" 请选择你的操作[0~16]:");

fflush(stdin);

scanf("%d",&op);

switch(op){

case 1:

//创建图

if(G) {

printf(" 图已创建，创建失败！\n");

getchar();

}

else{

//读入顶点集

printf(" 请输入顶点集(长度不大于%d的字符串)\n ",MAX\_VERTEX\_NUM);

scanf("%s",Vertex);

getchar();

fflush(stdin);

//Vertex序列中检查顶点个数

k=strlen(Vertex);//顶点个数

if(k>MAX\_VERTEX\_NUM){//若输入顶点过多

printf(" 输入顶点超出上限！将返回主菜单\n");

Vertex[0]='\0';

getchar();

break;

}

if(Vertex[0]=='\0'){//顶点集格式有误

printf(" 顶点集输入格式有误！将返回主菜单\n");

getchar();

break;

}

//Vertex序列中检查重复顶点

flag\_c=0;//标记是否检查到重复顶点

for(i=0;i<k-1;i++){//双重循环检查重复顶点

for(j=i+1;j<k;j++){

if(Vertex[i]==Vertex[j]){//检查到重复顶点

flag\_c=1;

break;

}

}

if(flag\_c) break;

}

if(flag\_c){//若顶点集中有重复顶点

printf(" 顶点集中含有重复顶点，输入有误！将返回主菜单\n");

Vertex[0]='\0';

}

else{//继续读取边关系

ScanfVR(&VR);

if(CreateGraph(&G,Vertex,VR->next)==OK) printf("\n 图创建完成！\n");

else printf(" 存储空间不足，创建失败！\n");

}

}

getchar();

break;

case 2:

//销毁

if(!G) printf(" 图未创建！\n");

else{

if(DestroyGraph(&G)==OK) printf(" 图销毁成功！\n");

}

getchar();getchar();

break;

case 3:

//查找data的位置信息

if(!G) printf(" 图未创建！\n");//图不存在

else{

printf(" 请输入数据data，将为您返回该顶点的位序\n\n");

getchar();

printf(" data：");

scanf("%c",&data);

if(LocateVex(G,&num,data)==OK) printf(" 图中顶点%c在邻接表中的位序为%d",data,num);

else printf(" 图中未找到data为%c的顶点！\n",data);

}

getchar();getchar();

break;

case 4:

//查找对应位置的data

if(!G) printf(" 图未创建！\n");//图不存在

else{

printf(" 请输入位序num，将为您返回该顶点数据\n\n");

printf(" num：");

scanf("%d",&num);

if(GetVex(G,num,&data)==OK) printf(" 图中位序为%d的顶点的数据为%c\n",num,data);

else printf(" 不存在位序为%d的顶点！\n",num);

}

getchar();getchar();

break;

case 5:

//对应位置顶点赋值为value

if(!G) printf(" 图未创建！\n");//图不存在

else{

printf(" 请输入位序num与数据value，将为该顶点赋值\n\n");

printf(" num：");

scanf("%d",&num);

getchar();

printf(" value：");

scanf("%c",&value);

if(PutVex(&G,num,value)==OK) printf(" 图中位序为%d的顶点的值更改成功\n",num);

else printf(" 不存在位序为%d的结点！\n",num);

}

getchar();getchar();

break;

case 6:

//获得num顶点的第一邻接点

if(!G) printf(" 图未创建！\n");//图不存在

else{

printf(" 请输入顶点v的位序num，将为您返回该顶点的第一邻接点\n\n");

printf(" (顶点v)num：");

scanf("%d",&num);

temp=FirstAdjVex(G,num);

if(temp!=NULL) printf(" 图中位序为%d的顶点的第一邻接点的位序num为%d，数据data为%c\n",num,G->vertices[num].firstarc->adjvex,temp->data);

else printf(" 查找失败！\n");

}

temp=NULL;//置空暂存结点

getchar();getchar();

break;

case 7:

//获得num顶点相对于num\_w的下一邻接点

if(!G) printf(" 图未创建！\n");//图不存在

else{

printf(" 请输入顶点v与w的位序num，将为您返回v结点相对u顶点的下一邻接点\n\n");

printf(" (顶点v)num：");

scanf("%d",&num);

printf(" (顶点w)num：");

scanf("%d",&num\_w);

temp=NextAdjVex(G,num,num\_w);

if(temp!=NULL) printf(" 图中位序为%d的顶点v相对w的下一邻接点的数据data为%c\n",num,temp->data);

else printf(" 查找失败！\n");

}

temp=NULL;//置空暂存结点

getchar();getchar();

break;

case 8:

//插入顶点

if(!G) printf(" 图未创建！\n");//图不存在

else{

printf(" 请输入顶点数据，将为您在邻接表中插入该顶点\n\n");

printf(" data：");

getchar();

scanf("%c",&data);

getchar();

ScanfVR\_insert(&VR,data);//读取边关系

if(InsertVex(&G,data,VR->next)==OK) printf(" 插入顶点成功！\n");

else printf(" 插入顶点失败！\n");

}

getchar();

break;

case 9:

//删除顶点

if(!G) printf(" 图未创建！\n");//图不存在

else{

printf(" 请输入顶点数据data，将为您删除该顶点\n\n");

printf(" data：");

getchar();

scanf("%c",&data);

if(LocateVex(G,&num,data)==OK){

if(DeleteVex(&G,num)==OK) printf(" 删除顶点成功！\n");

else printf(" 删除顶点失败！\n");

}

else{

printf(" 在邻接表内未查询到值为%c的顶点！\n",data);

}

}

getchar();getchar();

break;

case 10:

//插入弧

if(!G) printf(" 图未创建！\n");//图不存在

else{

getchar();

ScanfVR(&VR);

if(InsertArc(&G,VR->next)==OK) printf(" 插入完成！\n");

else printf(" 插入失败！\n");

}

getchar();

break;

case 11:

//删除弧

if(!G) printf(" 图未创建！\n");//图不存在

else{

printf(" 请输入顶点v与w的位序num，将为您删除链接他们的弧\n\n");

printf(" (顶点v)num：");

scanf("%d",&num);

getchar();

printf(" (顶点w)num：");

scanf("%d",&num\_w);

if(DeleteArc(&G,num,num\_w)==OK) printf(" 删除弧成功！\n");

else printf(" 删除弧失败！\n");

}

getchar();getchar();

break;

case 12:

//DFS

if(!G) printf(" 图未创建！\n");//图不存在

else{

printf(" 图的深度优先遍历结果：\n");

printf("--------------------------------------------\n");

DFSTraverse(G);

printf("--------------------------------------------\n");

}

getchar();getchar();

break;

case 13:

//BFS

if(!G) printf(" 图未创建！\n");//图不存在

else{

printf(" 图的广度优先遍历结果：\n");

printf("--------------------------------------------\n");

BFSTraverse(G);

printf("--------------------------------------------\n");

}

getchar();getchar();

break;

case 14:

//读取文件

if(!G){

printf(" 图未创建！\n");//图不存在

getchar();

}

else{

printf(" 将从文件中读取数据并覆盖当前图.\n 请输入要读取的文件名: ");getchar();

gets(filename);

if(LoadFile(&G,filename)==OK) printf(" 文件读取成功！\n");

else printf(" 文件读取失败！\n");

}

getchar();

break;

case 15:

//存储文件

if(!G){

printf(" 图未创建！\n");//图不存在

getchar();

}

else{

printf(" 将图中数据保存(写)为文件\n 请输入要写入的文件名: ");getchar();

gets(filename);

if(SaveAsFile(G,filename)==OK) printf(" 文件写入成功！\n");

}

getchar();

break;

case 16:

list\_num\_cur=list\_num;//记录当前图序以存储数据

printf(" 请输入要切换到的图序号：");

scanf("%d",&list\_num);

flag=MulList(&G,MulList\_head,list\_num\_cur,list\_num);

if(flag==OK) printf(" 切换成功，当前操作图为图%d\n",list\_num);

else if(flag==ERROR) printf(" 图%d不存在，已为您新建图%d并切换(未初始化)\n",list\_num,list\_num);

else if(flag==OVERFLOW) {//新图创建失败

printf(" 空间不足，图创建失败！\n");

list\_num=list\_num\_cur;//当前图序回退至换表前

}

//输出所有图序号

printf("\n------------------- all list --------------------\n");

printf(" 当前共有%d个图可供操作\n 其序号为(依创建时间排序)：\n ",MulList\_head->list\_num);

for(MulList\_temp=MulList\_head;MulList\_temp->next!=NULL;MulList\_temp=MulList\_temp->next){//输出所有表序

printf("%d ",MulList\_temp->next->list\_num);

}

printf("\n---------------------- end ----------------------\n");

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("\n 欢迎下次再使用本系统！\n");

}//end of main()

/\*\*

\* 函数名称：ScanfVR

\* 函数功能：用于读取VR边关系集的子函数

\*\*/

void ScanfVR(Relation \*\*VR)

{

//读取边关系VR

char tail,head;

int weight=-1;

\*VR=(Relation\*)malloc(sizeof(Relation));

(\*VR)->next=NULL;

Relation \*temp=\*VR;//用于遍历

Relation \*insert=NULL;//待插入结点

printf("\n Tips：不支持构建单边环及负权边\n");

printf(" 样例输入：a b 6\n");

printf(" 请输入欲链接的顶点值与其边的权重,输入文件尾停止读取\n ");

while(scanf("%c %c %d",&tail,&head,&weight)!=EOF){

if(tail=='\n'||tail==' '||head=='\n'||head==' '||weight<=0||tail==head){//若读取格式有误

printf(" 输入格式有误，将跳过此组输入\n");

printf(" 请输入欲链接的结点值与其边的权重,输入文件尾停止读取\n ");

fflush(stdin);

continue;

}

getchar();

//新结点赋值

insert=(Relation\*)malloc(sizeof(Relation));

insert->tail=tail;

insert->head=head;

insert->weight=weight;

insert->next=NULL;

//链接

temp->next=insert;

temp=temp->next;

printf(" 请输入欲链接的结点值与其边的权重,输入文件尾停止读取\n ");

fflush(stdin);//清空输入流

weight=-1;//重置权

}

printf("\n");

}

/\*\*

\* 函数名称：ScanfVR\_insert

\* 函数功能：用于读取VR边关系集的子函数

\*\*/

void ScanfVR\_insert(Relation \*\*VR,char tail)

{

//读取边关系VR

char head;

int weight=-1;

\*VR=(Relation\*)malloc(sizeof(Relation));

(\*VR)->next=NULL;

Relation \*temp=\*VR;//用于遍历

Relation \*insert=NULL;//待插入结点

printf("\n Tips：不支持构建单边环及负权边\n");

printf(" 样例输入：b 6\n");

printf(" 请输入欲与顶点%c链接的顶点值与其边的权重,输入文件尾停止读取\n ",tail);

while(scanf("%c %d",&head,&weight)!=EOF){

if(head=='\n'||head==' '||weight<=0||tail==head){//若读取格式有误

printf(" 输入格式有误，将跳过此组输入\n");

printf(" 请输入欲链接的结点值与其边的权重,输入文件尾停止读取\n ");

fflush(stdin);

continue;

}

getchar();

//新结点赋值

insert=(Relation\*)malloc(sizeof(Relation));

insert->tail=tail;

insert->head=head;

insert->weight=weight;

insert->next=NULL;

//链接

temp->next=insert;

temp=temp->next;

printf(" 请输入欲链接的结点值与其边的权重,输入文件尾停止读取\n ");

fflush(stdin);//清空输入流

weight=-1;//重置权

}

printf("\n");

}

/\*\*

\* 函数名称：Link

\* 函数功能：用于根据位序链接邻接表中两个顶点

\* 返回值：成功链接返回OK，否则返回ERROR

\*\*/

status Link(ALGraph \*G, ArcNode \*insert\_arc, int tail\_vex, int head\_vex)

{

//在邻接表中用弧insert\_arc链接(tail\_vex,head\_vex)

ArcNode \*temp = NULL;//用于遍历邻接弧

if (!(\*G)->vertices[tail\_vex].firstarc) {//若弧尾无邻接弧

(\*G)->vertices[tail\_vex].firstarc = (ArcNode\*)malloc(sizeof(ArcNode));

\*(\*G)->vertices[tail\_vex].firstarc = \*insert\_arc;

return OK;

}

//否则(若该边已有邻接弧)

for (temp = (\*G)->vertices[tail\_vex].firstarc;

temp != NULL;temp = temp->nextarc) {//遍历邻接边

if (temp->adjvex == head\_vex) {//如果邻边已经存在

return ERROR;

}

if (temp->nextarc == NULL) {//插入到最后一个邻接弧

temp->nextarc = (ArcNode\*)malloc(sizeof(ArcNode));

\*temp->nextarc = \*insert\_arc;

return OK;

}

}

return ERROR;

}

/\*\*

\* 函数名称：CreateGraph

\* 函数参数：图G的地址，Vertex顶点集，VR边关系集

\* 函数功能：根据顶点集与边关系集构造一个图

\* 返回值：成功构造返回OK，否则返回ERROR

\*\*/

status CreateGraph(ALGraph \*G, char \*Vertex, Relation \*VR)

{

//结点集为Vertex 以VR边关系 创建一个图

int i,j,k;//计数器

int flag\_1=0,flag\_2=0;//标志是否找到结点

ArcNode \*insert\_arc=NULL;//代插入弧

if(\*G){//若图已经存在

return ERROR;

}

//否则创建图

(\*G)=(ALGraph)malloc(sizeof(ALNode));//创建图头结点

(\*G)->vexnum = strlen(Vertex);

(\*G)->arcnum = 0;

//初始化邻接表

for(i=0;i<MAX\_VERTEX\_NUM;i++){

(\*G)->vertices[i].data='\0';

(\*G)->vertices[i].firstarc=NULL;//初始化第一条邻接弧

}

for(i=0;i<(\*G)->vexnum;i++){//顶点赋值

(\*G)->vertices[i].data = \*Vertex++;

}

for(;VR!=NULL;VR=VR->next){//边赋值

flag\_1=0;

flag\_2=0;

printf(" 正在构建边(%c,%c)...",VR->tail,VR->head);

for(j=0;j<(\*G)->vexnum;j++){//找弧尾

if((\*G)->vertices[j].data==VR->tail){//若找到弧尾

for(k=0;k<(\*G)->vexnum;k++){//找弧头

if((\*G)->vertices[k].data==VR->head){//且若找到弧头

//创建新结点

insert\_arc = (ArcNode \*)malloc(sizeof(ArcNode));

insert\_arc->adjvex = k;

insert\_arc->nextarc = NULL;

insert\_arc->info = VR->weight;

//插入

if(Link(G,insert\_arc,j,k)==OK){//若成功插入边

insert\_arc -> adjvex = j;

Link(G,insert\_arc,k,j);

(\*G)->arcnum++;

printf(" 插入成功\n");

}

else{//若插入失败

printf(" 邻接边(%c,%c)已经存在，构建失败！将跳过此边继续创建\n",VR->tail,VR->head);

}

free(insert\_arc);

flag\_2=1;

break;

}

if(k==(\*G)->vexnum-1&&flag\_2==0){//遍历完成未找到弧头

printf(" 未找到顶点%c\n",VR->head);

}

}

flag\_1=1;

break;

}

if(j==(\*G)->vexnum-1&&flag\_1==0){//遍历完成未找到弧尾

printf(" 未找到顶点%c\n",VR->tail);

}

}

}

return OK;

}

/\*\*

\* 函数名称：DestroyGraph

\* 函数参数：图G的地址

\* 函数功能：删除图

\* 返回值：成功删除返回OK，否则返回ERROR

\*\*/

status DestroyGraph(ALGraph \*G)

{

//销毁图

ArcNode \*temp1=NULL;

ArcNode \*temp2=NULL;

int i;//计数器

for(i=0;i<(\*G)->vexnum;i++){//遍历顶点

temp1=(\*G)->vertices[i].firstarc;

while(temp1){

temp2=temp1->nextarc;

free(temp1);

temp1=temp2;

}

}

free(\*G);

(\*G)=NULL;

return OK;

}

/\*\*

\* 函数名称：LocateVex

\* 函数参数：图G，顶点位序num，顶点数据data

\* 函数功能：用num返回数据为data的顶点的位序号

\* 返回值：成功查到则返回OK，否则返回ERROR

\*\*/

status LocateVex(ALGraph G, int \*num, VertexType data)

{

//返回数据为data的顶点的位置num

int i;//计数器

for(i=0;i<G->vexnum;i++){//遍历顶点集

if(data==G->vertices[i].data){//若找到

(\*num)=i;

return OK;

}

}

//否则未找到

return ERROR;

}

/\*\*

\* 函数名称：LocateVex

\* 函数参数：图G，顶点位序num，顶点数据data

\* 函数功能：用data返回位序为num的顶点的数据

\* 返回值：成功查到则返回OK，否则返回ERROR

\*\*/

status GetVex(ALGraph G, int num, VertexType \*data)

{

//返回位序为num的结点的数据data

//判断num是否合法//num ∈ 0 ~ max-1

if(num<0||num>=G->vexnum) {

printf(" 访问越界，num非法\n");

return ERROR;

}

//传出data

(\*data) = G->vertices[num].data;

return OK;

}

/\*\*

\* 函数名称：LocateVex

\* 函数参数：图G，顶点位序num，顶点数据value

\* 函数功能：用value为位序为num的顶点赋值

\* 返回值：成功赋值则返回OK，否则返回ERROR

\*\*/

status PutVex(ALGraph \*G, int num, VertexType value)

{

//对位序为num的结点赋值value

//判断num是否合法//num ∈ 0 ~ max-1

if(num<0||num>=(\*G)->vexnum) {

printf(" 访问越界，num非法\n");

return ERROR;

}

//赋值

(\*G)->vertices[num].data=value;

return OK;

}

/\*\*

\* 函数名称：FirstAdjVex

\* 函数参数：图G，顶点位序num

\* 函数功能：查找位序为num的顶点的第一邻接点

\* 返回值：成功查到则返回第一邻接点的地址，否则返回NULL

\*\*/

VNode\* FirstAdjVex(ALGraph G,int num)

{

//返回位序为num的顶点的第一邻接点

//判断num是否合法//num ∈ 0 ~ max-1

if(num<0||num>=G->vexnum) {

printf(" 访问越界，顶点的位序num非法\n");

return NULL;

}

//若num顶点无第一邻接点

if(!G->vertices[num].firstarc){

printf(" %c顶点无邻接点\n",G->vertices[num].data);

return NULL;

}

//返回num顶点的第一邻接点

return &(G->vertices[G->vertices[num].firstarc->adjvex]);

}

/\*\*

\* 函数名称：NextAdjVex

\* 函数参数：图G，v顶点位序num，w顶点位序num\_w

\* 函数功能：查找位序为num的顶点v相对顶点w的下一邻接点

\* 返回值：成功查到则返回下一邻接点的地址，否则返回NULL

\*\*/

VNode\* NextAdjVex(ALGraph G,int num,int num\_w)

{

//返回位序为num的相对w的下一个邻接点

//判断num与num\_w是否合法//num ∈ 0 ~ max-1

if(num<0||num>=G->vexnum) {

printf(" 访问越界，v顶点的位序num非法\n");

return NULL;

}

if(num\_w<0||num\_w>=G->vexnum) {

printf(" 访问越界，w顶点的位序num非法\n");

return NULL;

}

//若num顶点无邻接点

if(!G->vertices[num].firstarc){

printf(" num顶点无邻接点\n");

return NULL;

}

ArcNode \*temp=NULL;//用于遍历邻接边

//若num顶点与w顶点不邻接

for(temp=G->vertices[num].firstarc;temp!=NULL;temp=temp->nextarc){//遍历邻接边

if(temp->adjvex==num\_w){//若找到w顶点

if(temp->nextarc) return &(G->vertices[temp->nextarc->adjvex]);//若有下一邻接点，则返回

printf(" 与v顶点邻接的w顶点已是末邻接点，无下一邻接点\n");

return NULL;

}

}

//未找到邻接点w

printf(" w顶点与v顶点不邻接\n");

return NULL;

}

/\*\*

\* 函数名称：InsertVex

\* 函数参数：图G的地址，顶点数据data，边关集VR

\* 函数功能：插入数据为data的点进入邻接表并链接与其相关的顶点

\* 返回值：成功插入则返回OK，否则返回ERROR

\*\*/

status InsertVex(ALGraph \*G,VertexType data,Relation \*VR)

{

//插入顶点

//若无空间

if((\*G)->vexnum==MAX\_VERTEX\_NUM){

printf(" 邻接表已满\n");

return ERROR;

}

int i;//计数器

int num=(\*G)->vexnum;

for(i=0;i<num;i++){//循环检查顶点是否重复

if((\*G)->vertices[i].data==data){

printf(" 顶点已在邻接表中存在，插入失败！\n");

return ERROR;

}

}

//赋值顶点

(\*G)->vertices[num].data=data;

//链接与新顶点相关的边

int flag;//记录是否找到顶点

ArcNode \*insert\_arc=NULL;//待插入的新边

for(;VR!=NULL;VR=VR->next){//边赋值

flag=0;

printf(" 正在构建边(%c,%c)...",VR->tail,VR->head);

for(i=0;i<(\*G)->vexnum;i++){//找弧头

if((\*G)->vertices[i].data==VR->head){//若找到弧头

//创建新结点

insert\_arc = (ArcNode \*)malloc(sizeof(ArcNode));

insert\_arc->adjvex = i;

insert\_arc->nextarc = NULL;

insert\_arc->info = VR->weight;

//插入

if(Link(G,insert\_arc,num,i)==OK){//若成功插入边

insert\_arc -> adjvex = num;

Link(G,insert\_arc,i,num);

(\*G)->arcnum++;

printf(" 插入成功\n");

}

else{//若插入失败

printf(" 邻接边(%c,%c)已经存在，构建失败！将跳过此边继续创建\n",VR->tail,VR->head);

}

free(insert\_arc);

flag=1;

break;

}

if(i==(\*G)->vexnum-1&&flag==0){//遍历完成未找到弧头

printf(" 未找到顶点%c\n",VR->head);

}

}

}

(\*G)->vexnum++;

return OK;

}

/\*\*

\* 函数名称：DeleteVex

\* 函数参数：图G的地址，顶点位置num

\* 函数功能：删除位置为num的顶点

\* 返回值：成功删除则返回OK，否则返回ERROR

\*\*/

status DeleteVex(ALGraph \*G,int num)

{

//删除顶点

ArcNode \*temp1=NULL;

ArcNode \*temp2=NULL;//用于释放边结点空间

//释放空间(\*G)->vertices[num].firstarc

temp1=(\*G)->vertices[num].firstarc;

while(temp1){

temp2=temp1->nextarc;

free(temp1);

temp1=temp2;

}

(\*G)->vertices[num].firstarc=NULL;

int i;//计数器

ArcNode \*temp=NULL;//用于遍历邻接边

ArcNode \*temp\_del=NULL;//用于存放待删除结点

//删除与之相关的边

for(i=0;i<(\*G)->vexnum;i++){//遍历结点

if((\*G)->vertices[i].firstarc){//若有邻接点

temp=(\*G)->vertices[i].firstarc;

if (temp->adjvex == num) {//单独处理首邻接边

(\*G)->vertices[i].firstarc = temp->nextarc;

temp = (\*G)->vertices[i].firstarc;

}

if (temp) {//（temp可能进行删除后为空）若删除后仍有邻接边

if (temp->adjvex > num) temp->adjvex--;//修改后续边对应指向

for (;temp && temp->nextarc;temp = temp->nextarc) {//遍历邻接边

if (temp->nextarc->adjvex == num) {//若temp->next邻接num

temp\_del = temp->nextarc;

temp->nextarc = temp->nextarc->nextarc;

free(temp\_del);

}

if (temp->nextarc) {//若处理后仍有下一邻接点，修改后续边对应指向

if (temp->nextarc->adjvex > num) {

temp->nextarc->adjvex--;

}

}

}

}

}

}

//删除点

for(i=num;i<(\*G)->vexnum;i++){

(\*G)->vertices[i]=(\*G)->vertices[i+1];

}

//释放空间(\*G)->vertices[i].firstarc

temp1=(\*G)->vertices[i].firstarc;

while(temp1){

temp2=temp1->nextarc;

free(temp1);

temp1=temp2;

}

//置空

(\*G)->vertices[i].data='\0';

(\*G)->vertices[i].firstarc=NULL;

(\*G)->vexnum--;

return OK;

}

/\*\*

\* 函数名称：InsertArc

\* 函数参数：图G的地址，边关系集VR

\* 函数功能：按照边关系集VR在邻接表中插入边

\* 返回值：成功插入则返回OK，否则返回ERROR

\*\*/

status InsertArc(ALGraph \*G,Relation \*VR)

{

//插入弧

//以VR边关系链接边

int i,j;//计数器

int flag\_1=0,flag\_2=0;//标志是否找到结点

ArcNode \*insert\_arc=NULL;//待插入弧结点

for(;VR!=NULL;VR=VR->next){//边赋值

flag\_1=0;

flag\_2=0;

printf(" 正在构建边(%c,%c)...",VR->tail,VR->head);

for(i=0;i<(\*G)->vexnum;i++){//找弧尾

if((\*G)->vertices[i].data==VR->tail){//若找到弧尾

for(j=0;j<(\*G)->vexnum;j++){//找弧头

if((\*G)->vertices[j].data==VR->head){//且若找到弧头

//创建新结点

insert\_arc = (ArcNode \*)malloc(sizeof(ArcNode));

insert\_arc->adjvex = j;

insert\_arc->nextarc = NULL;

insert\_arc->info = VR->weight;

//插入

if(Link(G,insert\_arc,i,j)==OK){//若成功插入边

insert\_arc -> adjvex = i;

Link(G,insert\_arc,j,i);

(\*G)->arcnum++;

printf(" 插入成功\n");

}

else{//若插入失败

printf(" 邻接边(%c,%c)已经存在，构建失败！将跳过此边继续创建\n",VR->tail,VR->head);

}

free(insert\_arc);

flag\_2=1;

break;

}

if(j==(\*G)->vexnum-1&&flag\_2==0){//遍历完成未找到弧头

printf(" 未找到顶点%c\n",VR->head);

}

}

flag\_1=1;

break;

}

if(i==(\*G)->vexnum-1&&flag\_1==0){//遍历完成未找到弧尾

printf(" 未找到顶点%c\n",VR->tail);

}

}

}

return OK;

}

/\*\*

\* 函数名称：DeleteVex

\* 函数参数：图G的地址，v顶点位置num\_v，w顶点位置num\_w

\* 函数功能：删除v，w顶点之间的邻接边

\* 返回值：成功删除则返回OK，否则返回ERROR

\*\*/

status DeleteArc(ALGraph \*G,int num\_v,int num\_w)

{

//删除弧

//判断num\_v与num\_w是否合法

if(num\_v<0||num\_v>=(\*G)->vexnum) {

printf(" 访问越界，顶点的位序num\_v非法\n");

return ERROR;

}

if(num\_w<0||num\_w>=(\*G)->vexnum) {

printf(" 访问越界，顶点的位序num\_w非法\n");

return ERROR;

}

//删除边

int i;//计数器

ArcNode \*temp=NULL;//用于遍历邻接边

ArcNode \*temp\_del=NULL;//用于存放待删除结点

//删除v->w

if((\*G)->vertices[num\_v].firstarc){//若有邻接点

temp=(\*G)->vertices[num\_v].firstarc;

if(temp->adjvex==num\_w){//若首邻接点为num

(\*G)->vertices[num\_v].firstarc=temp->nextarc;

free(temp);

}

else{//否则遍历邻接边

for(;temp->nextarc!=NULL;temp=temp->nextarc){//遍历邻接边

if(temp->nextarc->adjvex==num\_w && temp->nextarc->nextarc!=NULL){//若temp->next邻接num且不为末邻接点

temp\_del=temp->nextarc;

temp->nextarc=temp->nextarc->nextarc;

free(temp\_del);

break;

}

if(temp->nextarc->adjvex==num\_w && temp->nextarc->nextarc==NULL){//若temp->next邻接num且为末邻接点

free(temp->nextarc);

temp->nextarc=NULL;

break;

}

}

}

}

//删除w->v

if((\*G)->vertices[num\_w].firstarc){//若有邻接点

temp=(\*G)->vertices[num\_w].firstarc;

if(temp->adjvex==num\_v){//若首邻接点为num

(\*G)->vertices[num\_w].firstarc=temp->nextarc;

free(temp);

(\*G)->arcnum--;

return OK;

}

else{//否则遍历邻接边

for(;temp->nextarc!=NULL;temp=temp->nextarc){//遍历邻接边

if(temp->nextarc->adjvex==num\_v && temp->nextarc->nextarc!=NULL){//若temp->next邻接num且不为末邻接点

temp\_del=temp->nextarc;

temp->nextarc=temp->nextarc->nextarc;

free(temp\_del);

(\*G)->arcnum--;

return OK;

}

if(temp->nextarc->adjvex==num\_v && temp->nextarc->nextarc==NULL){//若temp->next邻接num且为末邻接点

free(temp->nextarc);

temp->nextarc=NULL;

(\*G)->arcnum--;

return OK;

}

}

}

}

//若删除失败

printf(" 顶点v与w之间无邻接边\n");

return ERROR;

}

/\*\*

\* DFSTraverse的子函数

\*\*/

void DFS(ALGraph G, int i, int \*visited)

{

//深度优先搜索遍历图的递归实现

ArcNode \*temp=NULL;

visited[i] = 1;

printf("%c ", G->vertices[i].data);

temp = G->vertices[i].firstarc;

while (temp != NULL)

{

if (!visited[temp->adjvex]){

DFS(G, temp->adjvex, visited);

}

temp = temp->nextarc;

}

}

/\*\*

\* 函数名称：DFSTraverse

\* 函数参数：图G

\* 函数功能：深度优先遍历图

\* 返回值：成功遍历则返回OK，否则返回ERROR

\*\*/

status DFSTraverse(ALGraph G)

{

//深度优先搜索遍历图

int i; //计数器

int visited[MAX\_VERTEX\_NUM]; //顶点访问标记

for (i = 0; i < G->vexnum; i++){//初始化访问标记

visited[i] = 0;

}

printf(" DFS: ");

for (i = 0; i < G->vexnum; i++){

if (!visited[i]){

DFS(G, i, visited);

}

}

printf("\n");

return OK;

}

/\*\*

\* 函数名称：BFSTraverse

\* 函数参数：图G

\* 函数功能：广度优先遍历图

\* 返回值：成功遍历则返回OK，否则返回ERROR

\*\*/

status BFSTraverse(ALGraph G)

{

//广度优先搜索遍历

int head = 0;

int rear = 0;

int queue[MAX\_VERTEX\_NUM]; //辅组队列

int visited[MAX\_VERTEX\_NUM]; //顶点访问标记

int i, j, k; //计数器

ArcNode \*temp=NULL;

for (i = 0; i < G->vexnum; i++){//初始化访问标记

visited[i] = 0;

}

printf(" BFS: ");

for (i = 0; i < G->vexnum; i++){

if (!visited[i]){

visited[i] = 1;

printf("%c ", G->vertices[i].data);

queue[rear++] = i; //头结点入队列

}

while (head != rear) {

j = queue[head++]; //出队列

temp = G->vertices[j].firstarc;

while (temp != NULL){

k = temp->adjvex;

if (!visited[k]){

visited[k] = 1;

printf("%c ", G->vertices[k].data);

queue[rear++] = k; //未访问的周边结点入队列

}

temp = temp->nextarc;

}

}

}

printf("\n");

return OK;

}

/\*\*

\* 函数名称：LoadList

\* 函数参数：图G的地址，文件名filename

\* 函数功能：数据加载，将文件中数据读入，覆盖当前图

\* 返回值：加载成功返回OK，加载失败返回ERROR

\*\*/

status LoadFile(ALGraph \*G,char \*filename)

{

ALGraph G\_temp=NULL;

char Vertex[MAX];//顶点集缓冲区

Relation \*VR;//边关系缓冲区

Vertex[0]='\0';//初始化

VR=(Relation\*)malloc(sizeof(Relation));

VR->next=NULL;//初始化

FILE \*fp;//文件指针

fp = fopen(filename, "r");

if(fp==NULL) {

printf("文件不存在！\n");

return ERROR;

}

//将文件中数据读入系统

fscanf(fp,"%s\r\n",Vertex);//读取vertex

//构建VR

char tail,head;

int weight;

Relation \*temp=VR;//用于遍历

Relation \*insert=NULL;//待插入结点

while(fscanf(fp,"%c %c %d\r\n",&tail,&head,&weight)!=EOF){

//新结点赋值

insert=(Relation\*)malloc(sizeof(Relation));

insert->tail=tail;

insert->head=head;

insert->weight=weight;

insert->next=NULL;

//链接

temp->next=insert;

temp=temp->next;

}

printf(" 尝试建立图：\n");

if(CreateGraph(&G\_temp,Vertex,VR->next)==OK){//若文件内数据能够建图覆盖当前图

printf(" 文件中数据符合要求，覆盖当前图：\n");

DestroyGraph(G);

CreateGraph(G,Vertex,VR->next);

fclose(fp);

return OK;

}

else{

printf(" 文件中数据格式有误，读取失败！\n");

}

fclose(fp);

return ERROR;

}

/\*\*

\* 函数名称：SaveAsFile

\* 函数参数：图G的地址，文件名filename

\* 函数功能：数据保存，将图中数据保存至文件

\* 返回值：保存成功返回OK，加载失败返回ERROR

\*\*/

status SaveAsFile(ALGraph G,char \*filename)

{

int head = 0;

int rear = 0;

int queue[MAX\_VERTEX\_NUM]; //辅组队列

int visited[MAX\_VERTEX\_NUM]; //顶点访问标记

int i, j, k; //计数器

char VR[MAX\_VERTEX\_NUM];//顶点集

ArcNode \*temp=NULL;

FILE \*fp;

fp = fopen(filename, "wb+");

//顶点集存入文件

for(i=0;i<MAX\_VERTEX\_NUM;i++){

VR[i]=G->vertices[i].data;

}

fprintf(fp, "%s\r\n", VR);

//广度遍历将图中元素转换为输入格式并存入文件

for (i = 0; i < G->vexnum; i++){//初始化访问标记

visited[i] = 0;

}

for (i = 0; i < G->vexnum; i++){

if (!visited[i]){

queue[rear++] = i; //头结点入队列

}

while (head != rear) {

j = queue[head++]; //出队列

visited[j] = 1;

temp = G->vertices[j].firstarc;

while (temp != NULL){

k = temp->adjvex;

if (!visited[k]){

fprintf(fp,"%c %c %d\r\n",G->vertices[j].data,G->vertices[k].data,temp->info);

queue[rear++] = k; //未访问的周边结点入队列

}

temp = temp->nextarc;

}

}

}

fclose(fp);

return OK;

}

/\*\*

\* 函数名称：MulList

\* 函数参数：图G的地址，多图表头M的地址，当前图序号list\_num\_cur，欲切换到的图序号list\_num

\* 函数功能：切换操作图。

\* 返回值：成功切换返回OK，否则返回ERROR

\*\*/

status MulList(ALGraph \*G,MulNode \*M,int list\_num\_cur,int list\_num){

//切换操作图，若目标图不存在则新建

MulNode \*mul\_temp=NULL;//用于遍历多图结构

MulNode \*mul\_new=NULL;//用于新建多图结点

//存储图

for(mul\_temp=M->next;mul\_temp!=NULL;mul\_temp=mul\_temp->next){//给多图结构中的对应节点赋值，保存数据

if(mul\_temp->list\_num==list\_num\_cur){

mul\_temp->list\_head=(\*G);

break;

}

}

//切换图（为图头指针G赋值）

for(mul\_temp=M;mul\_temp->next!=NULL;mul\_temp=mul\_temp->next){//查询多图结构中的对应节点，切换操作表

if(mul\_temp->next->list\_num==list\_num){//若查询成功，读入图

(\*G) = mul\_temp->next->list\_head;

return OK;

}

}

//若查询失败，在多图结构中新建图，此时mul\_temp->next指针已经指向链表尾部NULL

mul\_new = (MulNode \*)malloc(sizeof(MulNode));

if (mul\_new==NULL) return OVERFLOW; //分配空间失败

mul\_new -> list\_num = list\_num;//赋值表序

mul\_new -> list\_head = NULL;//新表头节点置空

mul\_new -> next = NULL;//新节点next置空

mul\_temp -> next = mul\_new;//为新节点赋值

M->list\_num++;//记录多表个数+1

(\*G) = mul\_new -> list\_head;

return ERROR;

}