

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 计算机科学与技术201904**

**学 号： U201915050**

**姓 名： 高博文**

**指导教师： 袁凌**

**报告日期： 2020年 10月 27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 2](#_Toc458159879)

[1.1 问题描述 2](#_Toc458159880)

[1.2 系统设计 6](#_Toc458159882)

[1.3 系统实现 12](#_Toc458159883)

[1.4 实验小结 19](#_Toc458159884)

[2 基于链式存储结构的线性表实现 21](#_Toc458159885)

[2.1 问题描述 21](#_Toc458159886)

[2.2 系统设计 25](#_Toc458159887)

[2.3 系统实现 29](#_Toc458159888)

[2.4 实验小结 36](#_Toc458159889)

[3 基于二叉链表的二叉树实现 38](#_Toc458159890)

[3.1 问题描述 38](#_Toc458159891)

[3.2 系统设计 42](#_Toc458159892)

[3.3 系统实现 45](#_Toc458159893)

[3.4 实验小结 51](#_Toc458159894)

[4 基于二叉链表的二叉树实现 52](#_Toc458159895)

[4.1 问题描述 52](#_Toc458159896)

[4.2 系统设计 56](#_Toc458159897)

[4.3 系统实现 58](#_Toc458159898)

[4.4 实验小结 64](#_Toc458159899)

[参考文献 65](#_Toc458159900)

[附录A 基于顺序存储结构线性表实现的源程序 66](#_Toc458159901)

[附录B 基于链式存储结构线性表实现的源程序 90](#_Toc458159902)

[附录C 基于二叉链表二叉树实现的源程序 119](#_Toc458159903)

[附录D 基于邻接表图实现的源程序 152](#_Toc458159904)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

线性表是一个非常重要的数据结构，贯穿数据结构课程的整个学习过程中，本实验以顺序表作为线性表的物理结构，通过实现线性表的基本操作，掌握线性表的基本概念。同时需要构建一个菜单系统，能够对线性表实施控制管理。

## 1.2 系统设计

## 1.2.1 设计方案

设计一个多线性表管理系统，其中包括4个功能：增加线性表、删除线性表、查找线性表以及指定线性表进入管理。

设计一个单线性表管理系统，其中包括13个函数，创建线性表、销毁线性表、清空线性表、判空线性表、求线性表表长、获取其中数据元素、查找表中数据元素、获取指定元素前驱、获取指定元素后继、插入数据元素、删除数据元素、显示表中元素、线性表保存文件、读取文件进入线性表增加线性表、删除线性表、查找线性表。

整体框架：在多线性表管理系统中对多个线性表整体进行操作，再选择线性表进入单线性表管理系统，进行其中的数据管理等操作。用两个大循环结构中的switch函数执行不同的函数指令，单个线性表循环退出后回到多线性表管理系统选择其它线性表再次进行操作。

## 1.2.2 相关定义

定义不同函数的返回值以及执行状态，定义相关线性表的常量，定义相关数据类型

其中：

TRUE正确

FALSE错误

OK执行成功

ERROR执行错误

INFEASIBLE为不可执行

OVERFLOW为溢出

LIST\_INIT\_SIZE为线性表初始长度

LISTINCREMENT为线性表增加长度

status为函数返回值类型

ElemType为参数类型

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef int status;

typedef int ElemType;

其它：

顺序结构的定义

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

多线性表类型定义

typedef struct{ //线性表的集合类型定义

struct { char name[30];

SqList L;

} elem[10];

int length;

int listsize;

}LISTS;

线性表集合：

LISTS Lists;

## 1.2.3 构造函数

（1）InitList(SqList&L)

线性表L不存在，构造一个空的线性表，返回OK，否则返回INFEASIBLE。

（2）DestroyList(SqList &L)

如果线性表L存在，销毁线性表L，释放数据元素的空间，返回OK，否则返回INFEASIBLE。

（3）ClearList (SqList &L)

如果线性表L存在，删除线性表L中的所有元素，返回OK，否则返回INFEASIBLE。

（4）ListEmpty(SqList L)

如果线性表L存在，判断线性表L是否为空，空就返回TRUE，否则返回FALSE；如果线性表L不存在，返回INFEASIBLE。

（5）ListLength(SqList L)

如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

（6）GetElem(SqList L,int i,ElemType &e)

如果线性表L存在，获取线性表L的第i个元素，保存在e中，返回OK；如果i不合法，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

（7）LocateElem(SqList L,ElemType e)

如果线性表L存在，查找元素e在线性表L中的位置序号并返回OK；如果e不存在，返回ERROR；当线性表L不存在时，返回INFEASIBLE。

（8）PriorElem（SqList L,ElemType e,ElemType &pre）

如果线性表L存在，获取线性表L中元素e的前驱，保存在pre中，返回OK；如果没有前驱，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

（9）NextElem（SqList L,ElemType e,ElemType &next）

如果线性表L存在，获取线性表L元素e的后继，保存在next中，返回OK；如果没有后继，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

（10）ListInsert(SqList &L,int i,ElemType e)

如果线性表L存在，将元素e插入到线性表L的第i个元素之前，返回OK；当插入位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

（11）ListDelete(SqList &L,int i,ElemType &e)

如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回OK；当删除位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

（12）ListTraverse(SqList L)

如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，返回OK；如果线性表L不存在，返回INFEASIBLE。

（13）SaveList(SqList L,char FileName[]);

如果线性表L存在，将线性表L的的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。

（14）LoadList(SqList &L,char FileName[]);

如果线性表L不存在，将FileName文件中的数据读入到线性表L中，返回OK，否则返回INFEASIBLE。

（15）AddList(LISTS &Lists,char ListName[])

只需要在Lists中增加一个名称为ListName的空线性表，线性表数据又后台测试程序插入。

（16）RemoveList(LISTS &Lists,char ListName[])

Lists中删除一个名称为ListName的线性表。

（17）LocateList(LISTS Lists,char ListName[])

在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，否则返回0。

## 1.2.4 复杂性分析

创建、销毁、清空、求长、获取元素这些操作既不需要移动元素，也不需要遍历，时间复杂度均为O(1)；其余的函数需要循环遍历线性表等，时间复杂度为O(n).

表1-1时间复杂度分析表

|  |  |
| --- | --- |
| InitList | O(1) |
| DestroyList | O(1) |
| ClearList | O(n) |
| ListEmpty | O(1) |
| ListLength | O(1) |
| GetElem | O(1) |
| LocateElem | O(n) |
| PriorElem | O(n) |
| NextElem | O(n) |
| ListInsert | O(n) |
| ListDelete | O(n) |
| ListTraverse | O(n) |
| SaveList | O(n) |
| LoadList | O(n) |
| AddList | O(n) |
| RemoveList | O(n) |
| LocateList | O(n) |

## 1.3 系统实现

（1）InitList(SqList&L)

为L分配空间，令L.length为0，L.listsize为100。 后附流程图1-1。

（2）DestroyList(SqList &L)

判断L.elem如果为空，返回INFEASIBLE；如果非空，释放L.elem的空间，令L.elem为空。

（3）ClearList (SqList &L)

判断L.elem如果为空，返回INFEASIBLE；如果非空，循环遍历线性表令L.elem为空，最后将L.length置为0。

（4）ListEmpty(SqList L)

判断如果L.elem为空，返回INFEASIBLE；如果非空，判断L.length是否为0：如果为0，返回TRUE；如果不为0，返回FALSE。

（5）ListLength(SqList L)

判断如果L.elem为空，返回INFEASIBLE；不为空则返回L .length。

（6）GetElem(SqList L,int i,ElemType &e)

判断如果L.elem为空，返回INFEASIBLE；不为空则判断i是否在1到L.length之间：如果是将L.elem[i]赋给e；如果不是返回ERROR。

（7）LocateElem(SqList L,ElemType e)

判断如果L.elem为空，返回INFEASIBLE；不为空则循环遍历线性表，如果找到e和L.elem[i]相等，返回i+1；没找到返回ERROR。后附流程图1-3。

（8）PriorElem（SqList L,ElemType e,ElemType &pre）

判断如果L.elem为空，返回INFEASIBLE；不为空则循环遍历线性表，如果找到e和L.elem[i]相等且i不为0，则将L.elem[i-1]赋给e；没找到或者i等于0返回ERROR。

（9）NextElem（SqList L,ElemType e,ElemType &next）

判断如果L.elem为空，返回INFEASIBLE；不为空则循环遍历线性表，如果找到e和L.elem[i]相等且i不为L.length-1,则将L.elem[i-1]赋给e；没找到或者i等于L.length-1返回ERROR。

（10）ListInsert(SqList &L,int i,ElemType e)

判断如果L.elem为空，返回INFEASIBLE;不为空则判断i是否在1到L.length之间：如果不是返回ERROR；如果是，给L.elem重新分配空间，大小比原来多一个数据元素大小；再从最后一个数据开始向前循环到i，将i以及后面的数据向后移一位，将e赋给L.elem[i-1],最后让L.length自增。后附流程图1-2。

（11）ListDelete(SqList &L,int i,ElemType &e)

判断如果L.elem为空，返回INFEASIBLE;不为空则判断i是否在1到L.length之间：如果不是返回ERROR；如果是，从第i个数据开始向后循环到最后，将i后面的数据向前移一位覆盖掉前一位元素,最后让L.length自减。

（12）ListTraverse(SqList L)

判断如果L.elem为空，返回INFEASIBLE;不为空则循环遍历线性表，每次输出一个元素一直到最后。

（13）SaveList(SqList L,char FileName[]);

新建文件指针，如果打开失败返回ERROR；如果打开成功，fwrite写入L.length长度的线性表，关闭文件。

（14）LoadList(SqList &L,char FileName[]);

新建文件指针，如果打开失败返回ERROR；如果打开成功，循环fread读取文件内容，同时L.length自增一次，关闭文件。

（15）AddList(LISTS &Lists,char ListName[])

循环将ListName 赋给Lists.elem[Lists.length].name，再为新建L分配空间，令L.length为0，L.listsize为100，令Lists.elem[Lists.length].L为L，同时Lists.length自增。

（16）RemoveList(LISTS &Lists,char ListName[])

循环遍历所有线性表名Lists.elem[i].name中循环判断Lists.elem[i].name是否为ListName;如果没找到返回ERROR;如果找到，从i开始循环，用i后面的线性表覆盖前一个，达到将第i个删除的效果，最后Lists.length自减。

（17）LocateList(LISTS Lists,char ListName[])

循环遍历所有线性表名Lists.elem[i].name中循环判断Lists.elem[i].name是否为ListName;找到返回i+1；没找到返回0。

status InitList(SqList& L)

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

L.elem? Y

N

L.length=0;

L.listsize=100;

return OK

结

束

return INFEASIBLE

图 1-1 InitList(&L) 流程图

status ListInsert(SqList &L,int i,ElemType e)

L.elem? Y

N

Y

return ERROR ((i>L.length+1)||(i<=0))?

N

ElemType \* newbase Y

=(ElemType \* )realloc L.length>=L.listsize?

(L.elem,(L.listsize

+LISTINCREMENT) N

\*sizeof(ElemType));

L.elem=newbase;

L.listsize+=LISTIN for(k=L.length;k>=i;k--)

CREMENT; L.elem[k]=L.elem[k-1];

L.elem[i-1]=e;

L.length++;

return OK

return INFEASIBLE

结束

图 1-2 ListInsert(SqList &L,int i,ElemType e) 流程图

status LocateElem(SqList L,ElemType e)

L.elem? Y

N

int i=0

Y

return i+1 L.elem[i]==e?

N

i++

Y

i<=L.length-1?

N

return ERROR

return INFEASIBLE

结束

图 1-3 LocateElem(SqList L,ElemType e) 流程图

**1.4 系统测试**

测试用一个从创建到销毁的整体循环来测试系统的完整性与正确性。

表1-2测试步骤表

|  |  |  |
| --- | --- | --- |
| 操作 | 理论结果 | 步骤 |
| 输入“1 Enter L1 Enter” | 增加线性表成功 | 1 |
| 输入“1 Enter L2 Enter” | 增加线性表成功 | 2 |
| 输入“1 Enter L3 Enter” | 增加线性表成功 | 3 |
| 输入“2 Enter L2 Enter” | 删除线性表成功 | 4 |
| 输入“3 Enter L1 Enter” | 查找的线性表序号为1 | 5 |
| 输入“4 Enter 1 Enter” | 进入单线性表操作系统 | 6 |
| 输入“4 Enter” | 线性表为空 | 7 |
| 输入“10 Enter 1 Enter 1 Enter” | 插入成功，显示线性表中元素：1 | 8 |
| 输入“10 Enter 2 Enter 2 Enter” | 插入成功，显示线性表中元素：1 2 | 9 |
| 输入“10 Enter 3 Enter 3 Enter” | 插入成功，显示线性表中元素：1 2 3 | 10 |
| 输入“10 Enter 4 Enter 4 Enter” | 插入成功，显示线性表中元素：1 2 3 4 | 11 |
| 输入“4 Enter” | 线性表非空 | 12 |
| 输入“5 Enter” | 该线性表的长度为4 | 13 |
| 输入“6 Enter 1 Enter” | 第1个数的值为1 | 14 |
| 输入“7 Enter 2 Enter” | 元素2的序号为2 | 15 |
| 输入“8 Enter 3 Enter” | 元素3的前驱为2 | 16 |
| 输入“8 Enter 1 Enter” | 查找前驱失败 | 17 |
| 输入“9 Enter 3 Enter” | 元素3的后继为4 | 18 |
| 输入“9 Enter 4 Enter” | 查找后继失败 | 19 |
| 输入“11 Enter 2 Enter” | 删除成功，显示线性表中元素：1 3 4 | 20 |
| 输入“11 Enter 5 Enter” | 删除失败 | 21 |
| 输入“12 Enter” | 1 3 4 | 22 |
| 输入“13 Enter file1.dat Enter” | 保存成功 | 23 |
| 输入“3 Enter” | 清空线性表成功 | 24 |
| 输入“12 Enter” | /无显示/ | 25 |
| 输入“14 Enter file1.dat Enter” | 文件读取成功 | 26 |
| 输入“12 Enter” | 1 3 4 | 27 |
| 输入“2 Enter” | 销毁线性表成功 | 28 |
| 输入“0 Enter” | 返回多线性表管理系统 | 29 |
| 输入“0 Enter” | 程序结束 | 30 |

以下为程序运行结果图，其中多线性表菜单：

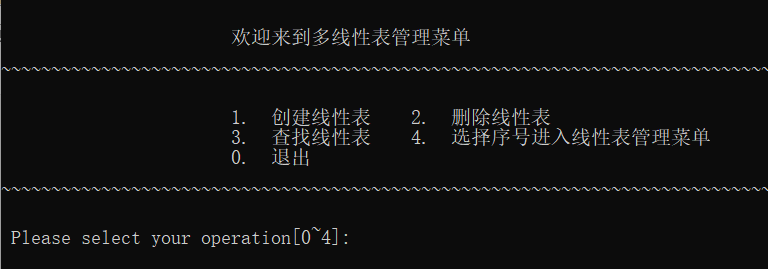


表1-3实验结果图

|  |  |
| --- | --- |
| 步骤 | 程序运行图 |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 |  |
| 20 |  |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 |  |
| 25 |  |
| 26 |  |
| 27 |  |
| 28 |  |
| 29 |  |
| 30 |  |

## 1.5 实验小结

数据结构第一次实验是一次对线性表全章节内容的总结与实践，不仅巩固了学习内容，也在设计过程中能够查漏补缺、推陈出新。总体设计框架的逻辑结构用循环嵌套式，过程中发生过一些错误，例如getchar函数的添加，表L作用范围的改变，在多线性表系统进入单线性表系统时操作的表需要赋为线性表集合中的指定表。同时C语言的语法也得到巩固，例如switch结构、循环结构、函数调用、文件操作等，在写函数过程中需要不断回忆甚至再次学习C语言知识。另外，在细节方面，线性表有许多需要细致考虑的情况，例如表不存在时的创建，表满时重新追加空间，前驱结点考虑第一个元素，后继结点考虑最后一个元素，插入元素时移动表中大量元素时的始末位置等等。总之，作为新学期第一次实验，我重新恢复了设计程序系统的熟练度，也复习了已学知识，对线性表结构能够熟练掌握。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

本实验以链表作为线性表的物理结构，通过实现线性表的基本操作，掌握线性表的基本概念。同时需要构建一个菜单系统，能够对线性表实施控制管理。

## 2.2 系统设计

## 2.2.1 设计方案

本实验与第一个实验相似，旨在实现一个多线性表管理系统，其中包括4个功能：增加线性表、删除线性表、查找线性表以及指定线性表进入管理。

同时完成一个单线性表管理系统，其中包括13个函数，创建线性表、销毁线性表、清空线性表、判空线性表、求线性表表长、获取其中数据元素、查找表中数据元素、获取指定元素前驱、获取指定元素后继、插入数据元素、删除数据元素、显示表中元素、线性表保存文件、读取文件进入线性表增加线性表、删除线性表、查找线性表。

整体框架：在多线性表管理系统中对多个线性表整体进行操作，再选择线性表进入单线性表管理系统，进行其中的数据管理等操作。用两个大循环结构中的switch函数执行不同的函数指令，单个线性表循环退出后回到多线性表管理系统选择其它线性表再次进行操作。

## 2.2.2 相关定义

定义不同函数的返回值以及执行状态，定义相关线性表的常量，定义相关数据类型

其中：

TRUE正确

FALSE错误

OK执行成功

ERROR执行错误

INFEASIBLE为不可执行

OVERFLOW为溢出

LIST\_INIT\_SIZE为线性表初始长度

LISTINCREMENT为线性表增加长度

status为函数返回值类型

ElemType为参数类型

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef int status;

typedef int ElemType;

其它：

顺序结构的定义

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

链表结构的定义

typedef struct LNode

{

ElemType data;

struct LNode \*next;

}LNode, \*LinkList;

多线性表类型定义

typedef struct{ //线性表的集合类型定义

struct { char name[30];

LinkList L;

} elem[10];

int length;

int listsize;

}LISTS;

线性表集合：

LISTS Lists;

## 2.2.3 构造函数

（1）InitList(LinkList&L)

线性表L不存在，构造一个空的线性表，返回OK，否则返回INFEASIBLE。

（2）DestroyList(LinkList &L)

如果线性表L存在，销毁线性表L，释放数据元素的空间，返回OK，否则返回INFEASIBLE。

（3）ClearList (LinkList &L)

如果线性表L存在，删除线性表L中的所有元素，返回OK，否则返回INFEASIBLE。

（4）ListEmpty(LinkList L)

如果线性表L存在，判断线性表L是否为空，空就返回TRUE，否则返回FALSE；如果线性表L不存在，返回INFEASIBLE。

（5）ListLength(LinkList L)

如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

（6）GetElem(LinkList L,int i,ElemType &e)

如果线性表L存在，获取线性表L的第i个元素，保存在e中，返回OK；如果i不合法，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

（7）LocateElem(LinkList L,ElemType e)

如果线性表L存在，查找元素e在线性表L中的位置序号并返回OK；如果e不存在，返回ERROR；当线性表L不存在时，返回INFEASIBLE。

（8）PriorElem（LinkList L,ElemType e,ElemType &pre）

如果线性表L存在，获取线性表L中元素e的前驱，保存在pre中，返回OK；如果没有前驱，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

（9）NextElem（LinkList L,ElemType e,ElemType &next）

如果线性表L存在，获取线性表L元素e的后继，保存在next中，返回OK；如果没有后继，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

（10）ListInsert(LinkList &L,int i,ElemType e)

如果线性表L存在，将元素e插入到线性表L的第i个元素之前，返回OK；当插入位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

（11）ListDelete(LinkList &L,int i,ElemType &e)

如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回OK；当删除位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

（12）ListTraverse(LinkList L)

如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，返回OK；如果线性表L不存在，返回INFEASIBLE。

（13）SaveList(LinkList L,char FileName[]);

如果线性表L存在，将线性表L的的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。

（14）LoadList(LinkList &L,char FileName[]);

如果线性表L不存在，将FileName文件中的数据读入到线性表L中，返回OK，否则返回INFEASIBLE。

（15）AddList(LISTS &Lists,char ListName[])

在Lists中增加一个名称为ListName的空线性表。

（16）RemoveList(LISTS &Lists,char ListName[])

Lists中删除一个名称为ListName的线性表。

（17）LocateList(LISTS Lists,char ListName[])

在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，否则返回0。

## 2.2.4 复杂性分析

链表在操作过程中往往需要顺着指针遍历，因此类似查找元素这一类的函数时间复杂度就会升为n。

表2-1时间复杂度分析表

|  |  |
| --- | --- |
| InitList | O(1) |
| DestroyList | O(n) |
| ClearList | O(n) |
| ListEmpty | O(1) |
| ListLength | O(n) |
| GetElem | O(n) |
| LocateElem | O(n) |
| PriorElem | O(n) |
| NextElem | O(n) |
| ListInsert | O(n) |
| ListDelete | O(n) |
| ListTraverse | O(n) |
| SaveList | O(n) |
| LoadList | O(n) |
| AddList | O(n) |
| RemoveList | O(n) |
| LocateList | O(n) |

## 2.3 系统实现

（1）InitList(LinkList&L)

判断L不为空，则返回INFEASIBLE；否则为L分配新空间，将L的next指针指向NULL，返回OK。

（2）DestroyList(LinkList &L)

判断如果L为空，返回INFEASIBLE；否则用p指针循环遍历所有元素结点，释放元素空间，最后将L指向NULL，返回OK。

（3）ClearList (LinkList &L)

判断如果L为空，返回INFEASIBLE；否则保留头结点，用p指针遍历后面所有结点清空元素，将L的next指针指向NULL，返回OK。

（4）ListEmpty(LinkList L)

判断如果L为空，返回INFEASIBLE；否则判断L的next指针是否为空：是返回TRUE；否返回FALSE。

（5）ListLength(LinkList L)

判断如果L为空，返回INFEASIBLE；否则用p指针遍历头结点后面所有结点，用i计数，返回i。

（6）GetElem(LinkList L,int i,ElemType &e)

判断如果L为空，返回INFEASIBLE；否则判断i是否合法：不合法返回ERROR；合法用p遍历链表，找到第i个元素赋给e，返回OK。

（7）LocateElem(LinkList L,ElemType e)

判断如果L为空，返回INFEASIBLE；否则用p遍历链表，用i计数，找到e返回i；未找到返回ERROR。

（8）PriorElem（LinkList L,ElemType e,ElemType &pre）

判断如果L为空，返回INFEASIBLE；否则用p遍历链表，用q跟进作为p的前驱；找到e将q赋给pre，返回OK；未找到返回ERROR。

（9）NextElem（LinkList L,ElemType e,ElemType &next）

判断如果L为空，返回INFEASIBLE；否则用p遍历链表，找到e将p->next赋给next，返回OK；未找到返回ERROR。

（10）ListInsert(LinkList &L,int i,ElemType e)

判断如果L为空，返回INFEASIBLE；否则用p遍历链表，用q跟进作为p的前驱；找到e后创建data为e的新结点i，将q->next赋给i->next，再将q->next指向i插入链表，返回OK；未找到返回ERROR。

（11）ListDelete(LinkList &L,int i,ElemType &e)

判断如果L为空，返回INFEASIBLE；再判断i是否合法：不合法返回ERROR；合法则用p循环到第i个元素，用q跟进作为p的前驱；将p->data赋给e，p->next赋给q->next，将p->next指向空，释放p结点，返回OK。

L==NULL？ return INFEASIBLE

N

LinkList p,q,l; 结束

p=L;

int k=0;

k<=i-1? N

Y

k++;

q=p;

p=p->next;

N

p==NULL?

Y

return ERROR;

结束 e=p->data;

q->next=p->next;

p->next=NULL;

free(p);

return OK；

（12）ListTraverse(LinkList L)

判断如果L为空，返回INFEASIBLE；再判断L->next是否为空：为空则输出空线性表；不为空则依次输出所有结点元素，返回OK。

Y

L==NULL？ return INFEASIBLE

N

LinkList p,q,l; 结束

p=L->next;

p==NULL? Y

N

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

p=p->next;

p==NULL?

N

return OK; printf(" %d",p->data);

p=p->next;

结束 printf("空线性表");

（13）SaveList(LinkList L,char FileName[]);

判断如果L为空，返回INFEASIBLE；否则用p指针循环fwrite函数写到文件中，关闭文件，返回OK。

（14）LoadList(LinkList &L,char FileName[]);

用循环条件为fread函数读取文件到temp中成功时，将temp赋给p，p指向下一个结点。循环结束后将p的next指向空，关闭文件，返回OK。

（15）AddList(LISTS &Lists,char ListName[])

循环将ListName 赋给Lists.elem[Lists.length].name，再为新建L分配空间，令L.length为0，L.listsize为100，令Lists.elem[Lists.length].L为L，同时Lists.length自增。

（16）RemoveList(LISTS &Lists,char ListName[])

循环遍历所有线性表名Lists.elem[i].name中循环判断Lists.elem[i].name是否为ListName;如果没找到返回ERROR;如果找到，从i开始循环，用i后面的线性表覆盖前一个，达到将第i个删除的效果，最后Lists.length自减。

（17）LocateList(LISTS Lists,char ListName[])

循环遍历所有线性表名Lists.elem[i].name中循环判断Lists.elem[i].name是否为ListName;找到返回i+1；没找到返回0。

## 2.4 系统测试

表2-2测试步骤表

|  |  |  |
| --- | --- | --- |
| 操作 | 理论结果 | 步骤 |
| 输入“1 Enter L1 Enter” | 增加线性表成功 | 1 |
| 输入“1 Enter L2 Enter” | 增加线性表成功 | 2 |
| 输入“1 Enter L3 Enter” | 增加线性表成功 | 3 |
| 输入“2 Enter L2 Enter” | 删除线性表成功 | 4 |
| 输入“3 Enter L1 Enter” | 查找的线性表序号为1 | 5 |
| 输入“4 Enter 1 Enter” | 进入单线性表操作系统 | 6 |
| 输入“4 Enter” | 线性表为空 | 7 |
| 输入“10 Enter 1 Enter 1 Enter” | 插入成功，显示线性表中元素：1 | 8 |
| 输入“10 Enter 2 Enter 2 Enter” | 插入成功，显示线性表中元素：1 2 | 9 |
| 输入“10 Enter 3 Enter 3 Enter” | 插入成功，显示线性表中元素：1 2 3 | 10 |
| 输入“10 Enter 4 Enter 4 Enter” | 插入成功，显示线性表中元素：1 2 3 4 | 11 |
| 输入“4 Enter” | 线性表非空 | 12 |
| 输入“5 Enter” | 该线性表的长度为4 | 13 |
| 输入“6 Enter 1 Enter” | 第1个数的值为1 | 14 |
| 输入“7 Enter 2 Enter” | 元素2的序号为2 | 15 |
| 输入“8 Enter 3 Enter” | 元素3的前驱为2 | 16 |
| 输入“8 Enter 1 Enter” | 查找前驱失败 | 17 |
| 输入“9 Enter 3 Enter” | 元素3的后继为4 | 18 |
| 输入“9 Enter 4 Enter” | 查找后继失败 | 19 |
| 输入“11 Enter 2 Enter” | 删除成功，显示线性表中元素：1 3 4 | 20 |
| 输入“11 Enter 5 Enter” | 删除失败 | 21 |
| 输入“12 Enter” | 1 3 4 | 22 |
| 输入“13 Enter file1.dat Enter” | 保存成功 | 23 |
| 输入“3 Enter” | 清空线性表成功 | 24 |
| 输入“12 Enter” | /无显示/ | 25 |
| 输入“14 Enter file1.dat Enter” | 文件读取成功 | 26 |
| 输入“12 Enter” | 1 3 4 | 27 |
| 输入“2 Enter” | 销毁线性表成功 | 28 |
| 输入“0 Enter” | 返回多线性表管理系统 | 29 |
| 输入“0 Enter” | 程序结束 | 30 |

表2-3实验结果图

|  |  |
| --- | --- |
| 步骤 | 程序运行图 |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 |  |
| 20 |  |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 |  |
| 25 |  |
| 26 |  |
| 27 |  |
| 28 |  |
| 29 |  |
| 30 |  |

## 2.5 实验小结

本次实验总体和第一次实验比较相像，总体结构已经构思完成，但是具体函数设计思路又很不同，链表的存储结构不能实现随意访问，访问结点需要顺着指针结点遍历，因此时间复杂度会增加。但是在类似插入元素的函数中，不需要在插入后像第一次那样将所有元素整体移动，只需要在目标指针处进行指针指向的转变，简化了操作。

本次实验总体是一个函数设计思路的转变，让我对链表的运用更加熟练，加深了理解。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

二叉树是一个非常重要的数据结构，本实验以二叉链表作为二叉树的物理结构，通过实现二叉树的基本操作，掌握二叉树的相关知识。

## 3.2 系统设计

## 3.2.1 设计方案

本实验需要设计一个以森林为整体框架的二叉树管理系统。其实和之前以多表管理系统下的单表管理系统很像，森林的功能有添加树、查找树、删除树、选择树进入二叉树管理系统；二叉树管理系统的功能包括创建二叉树、销毁二叉树、清空二叉树、求二叉树的深度、结点查找、结点赋值、查找兄弟结点、插入结点、删除结点、前序遍历、中序遍历、后序遍历、按层遍历、文件存储、文件读取以及打印二叉树。

程序设计和之前一样用循环结构不断实现各种函数的调用来对二叉树进行操作。

## 3.2.2 相关定义

定义不同函数的返回值以及执行状态，定义相关线性表的常量，定义相关数据类型

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status;

typedef int KeyType;

二叉树结点类型定义：

typedef struct

{

KeyType key;

char others[20];

} TElemType; //二叉树结点类型定义

DEF结构定义（为了更好管理输入数组）

typedef struct

{

int pos;

TElemType data;

} DEF;

二叉链表结点定义：

typedef struct BiTNode

{ //二叉链表结点的定义

TElemType data;

struct BiTNode \*lchild, \*rchild;

} BiTNode, \*BiTree;

森林结构定义：

typedef struct Forest

{ //森林定义

BiTree tree;

char name[30];

struct Forest \*next;

} Forest, \*Fp;

## 3.2.3 构造函数

（1）status CreateBiTree(BiTree &T,TElemType definition[])

根据带空枝的二叉树先根遍历序列definition构造一棵二叉树，将根节点指针赋值给T并返回OK，如果有相同的关键字，返回ERROR。

（2）status ClearBiTree(BiTree &T)

将二叉树设置成空，并删除所有结点，释放结点空间，返回OK。

（3）int BiTreeDepth(BiTree T)

求二叉树T的深度，成功返回深度，失败返回INFEASIBLE。

（4）BiTNode\* LocateNode(BiTree T,KeyType e)

查找关键字为e的结点，查找成功返回结点，失败返回NULL。

（5）status Assign(BiTree &T,KeyType e,TElemType value)

将value的关键字赋给关键字为e的结点；成功返回OK。

（6）BiTNode\* GetSibling(BiTree T,KeyType e)

获得关键字为e的结点的兄弟结点，返回兄弟结点，失败返回NULL。

（7）status InsertNode(BiTree &T,KeyType e,int LR,TElemType c)

将结点c插入为关键字为e的结点的左孩子或右孩子结点，结点e的原左子树或右子树作为c的右子树，返回OK，失败返回ERROR。

（8）status DeleteNode(BiTree &T,KeyType e)

删除T中关键字为e的结点；同时，如果关键字为e的结点度为0，删除即可；如关键字为e的结点度为1，用关键字为e的结点孩子代替被删除的e位置；如关键字为e的结点度为2，用e的左孩子代替被删除的e位置，e的右子树作为e的左子树中最右结点的右子树。 成功删除结点后返回OK，否则返回ERROR。

（9）status PreOrderTraverse(BiTree T,void (\*visit)(BiTree))

先序遍历二叉树T，用visit函数访问每一个结点，返回OK。

（10）status InOrderTraverse(BiTree T,void (\*visit)(BiTree))

中序遍历二叉树T，用visit函数访问每一个结点，返回OK。

（11）status PostOrderTraverse(BiTree T,void (\*visit)(BiTree))

后序遍历二叉树T，用visit函数访问每一个结点，返回OK。

（12）status LevelOrderTraverse(BiTree T,void (\*visit)(BiTree))

层序遍历二叉树T，用visit函数访问每一个结点，返回OK。

（13）status SaveBiTree(BiTree T, char FileName[])

将二叉树的结点数据写入到文件FileName中，成功返回OK，失败返回ERROR。

（14）status LoadBiTree(BiTree &T, char FileName[])

读入文件FileName的结点数据，创建二叉树，成功返回OK，失败返回ERROR。

（15）Fp AddTree(Fp &F, char \*name)

在森林中添加一棵二叉树，返回森林指针。

（16）Fp LocateTree(Fp F, char \*name)

在森林中查找一棵二叉树，查找成功返回二叉树的森林指针；若失败返回NULL。

（17）Fp deleteTree(Fp &F, char \*name)

在森林中删除一棵二叉树，删除成功返回森立指针，删除失败返回NULL。

（18）status switchList(Fp F, char \*name, BiTree &T)

在森林中选择一棵二叉树进入单个二叉树的管理表，成功返回OK;失败返回ERROR。

（19）void printTree(BiTree T, int h = 0)

打印正在操作的二叉树。

## 3.2.4 复杂性分析

二叉树的操作与线性表有很大不同，很多函数需要运用到递归。如果是遍历类型的递归，则每个结点访问一次时间复杂度为O(n)。一般的递归过程的函数例如插入结点或是查找兄弟结点等，最多也只是相当于遍历二叉树，因此时间复杂度为O(n)。

表31时间复杂度分析表

|  |  |
| --- | --- |
| CreateBiTree | O(n) |
| ClearBiTree | O(n) |
| BiTreeDepth | O(n) |
| LocateNode | O(n) |
| Assign | O(n) |
| GetSibling | O(n) |
| InsertNode | O(n) |
| DeleteNode | O(n) |
| PreOrderTraverse | O(n) |
| InOrderTraverse | O(n) |
| PostOrderTraverse | O(n) |
| LevelOrderTraverse | O(n) |
| SaveBiTree | O(n) |
| LoadBiTree | O(n) |
| AddTree | O(n) |
| LocateTree | O(n) |
| deleteTree | O(n) |
| switchList | O(n) |
| printTree | O(n) |

## 3.3 系统实现

（1）status CreateBiTree(BiTree &T,TElemType definition[])

首先循环遍历数组definition，用新建数组记录每个结点的关键字，找到一个关键字新建数组对应的元素+1。循环结束后，再循环新建数组所有元素，若存在大于或等于2的，返回ERROR。创建一个数组，循环为每个元素分配新空间后构建结点，当元素序号j为奇数，将序号为j/2的元素的右孩子结点赋为p[j]，否则将左孩子结点赋为p[j]，最后将p[1]赋给二叉树头指针T，返回OK。

（2）status ClearBiTree(BiTree &T)

用递归结构，若结点左孩子存在则调用函数清除左孩子，若右孩子存在清楚右孩子，最后将T置空，返回OK。

（3）int BiTreeDepth(BiTree T)

用递归结构，在T指针存在的循环条件下，判断i是否大于计数的count，如果是将i的值赋给count。先调用函数返回左子树深度，再调用函数返回右子树深度，最后将i值自减。循环结束标志递归结束，返回count值。若一开始T不存在，返回INFEASIBLE。

（4）BiTNode\* LocateNode(BiTree T,KeyType e)

用递归结构，在T存在的情况下，判断结点关键字是否等于e，如果是返回T，如果左孩子存在则调用函数判断左孩子，右孩子存在则调用函数判断右孩子，找到返回T，递归结束后没找到或是T不存在返回NULL。

（5）status Assign(BiTree &T,KeyType e,TElemType value)

调用4函数查找关键字为e的结点，找到将value赋给结点Node，没找到或是关键字重复返回ERROR。

（6）BiTNode\* GetSibling(BiTree T,KeyType e)

将关键字为e的结点的双亲结点赋给parent，判断如果双亲的左孩子等于e，返回右孩子；如果右孩子等于e，返回左孩子；若左孩子或右孩子不存在，返回NULL。

（7）status InsertNode(BiTree &T,KeyType e,int LR,TElemType c)

首先判断结点c的关键字是否重复，关键字e是否合法，如果重复或是不合法返回ERROR。判断LR是否为0，如果是，则新建一个结点Node，将c的数据赋给Node，右孩子等于原来根节点的左孩子，左孩子置空，最后将根节点的左孩子变为Node。如果LR是1，则新建一个结点Node，将c的数据赋给Node，右孩子等于原来根节点的右孩子，左孩子置空，最后将根节点的右孩子变为Node。返回OK。

（8）status DeleteNode(BiTree &T,KeyType e)

将e的双亲结点赋为parent，判断e是左孩子还是右孩子结点，左孩子记3，右孩子记2，root记为关键字为e的结点。如果结点左孩子存在，则左子树的最右结点赋为root的右孩子，再将root的左孩子赋给temp；如果右孩子存在，将右孩子赋给temp;若都不存在temp为NULL。判断e是右孩子结点则parent的右孩子赋为temp；是左孩子结点则parent的左孩子赋为temp,返回OK。

（9）status PreOrderTraverse(BiTree T,void (\*visit)(BiTree))

用递归结构，访问结点T，调用函数访问T的左子树，调用函数访问T的右子树，返回OK。

（10）status InOrderTraverse(BiTree T,void (\*visit)(BiTree))

用递归结构，调用函数访问T的左子树，访问结点T，调用函数访问T的右子树，返回OK。

（11）status PostOrderTraverse(BiTree T,void (\*visit)(BiTree))

用递归结构，调用函数访问T的左子树，调用函数访问T的右子树，访问结点T，返回OK。

（12）status LevelOrderTraverse(BiTree T,void (\*visit)(BiTree))

创建数组，若T存在将T赋给s[0]，int k=1，i=0，m=0，循环当k不为0时，将s[m]赋给p后，访问p，如果p的左孩子存在，那么k自增，i自增，且s[i]赋为p的左孩子结点；如果p的右孩子存在，那么ki自增，右孩子进栈，这样循环访问知道栈为空，也就是k为0。最后返回OK。

（13）status SaveBiTree(BiTree T, char FileName[])

创建文件指针，将二叉树写入数组list，结点写入Tlist后，用fwrite函数将树以及结点写入文件，返回OK。

（14）status LoadBiTree(BiTree &T, char FileName[])

读入文件FileName的整形数据至tail，树结构数据至list，结点数据至tp，循环创建二叉树，关闭文件，返回OK。

（15）Fp AddTree(Fp &F, char \*name)

在森林中添加一棵二叉树，用strcpy函数赋名，指针置空，调用创建二叉树函数创建二叉树。

（16）Fp LocateTree(Fp F, char \*name)

在森林中查找一棵二叉树，用strcmp函数判断是否有该名称的二叉树，如果有返回指针，没有返回NULL。

（17）Fp deleteTree(Fp &F, char \*name)

在森林中循环遍历，判断如果出现名称为name的二叉树，就将pre指针指向该树后面的树，释放树的空间。没找到返回NULL。

（18）status switchList(Fp F, char \*name, BiTree &T)

调用函数查找是否有该名称的二叉树，如果有将T赋为该二叉树，返回OK。没有返回ERROR。

（19）void printTree(BiTree T, int h = 0)

用递归结构，打印右子树，打印左子树。

## 3.4 系统测试

表3-2测试步骤表

|  |  |  |
| --- | --- | --- |
| 操作 | 理论结果 | 步骤 |
| 输入“15 2 a 1 1 1 2 1 3 0 0 0  b 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 6 6 7 7 7 8 8 8 0 0 0” | 二叉树创建失败  二叉树创建成功 | 1 |
| 输入“16 b” | 1,1 2,2 4,4 8,8 5,5 3,3 6,6 7,7 | 2 |
| 输入“18 b” | 切换成功 | 3 |
| 输入“3” | 二叉树的深度为4 | 4 |
| 输入“4 2” | 该key值对应的结点的内容为2 | 5 |
| 输入“4 9” | 查找失败 | 6 |
| 输入“5 1 9 9” | 结点赋值成功 | 7 |
| 输入“19” | 打印二叉树 | 8 |
| 输入“5 9 1 1” | 结点赋值成功 | 9 |
| 输入“6 2” | 结点的兄弟结点的key值为3,others值为3 | 10 |
| 输入“7 1 0 9 9” | 插入成功 | 11 |
| 输入“19” | 打印二叉树 | 12 |
| 输入“8 9” | 删除成功 | 13 |
| 输入“19” | 打印二叉树 | 14 |
| 输入“9” | 1,1 2,2 4,4 8,8 5,5 3,3 6,6 7,7 | 15 |
| 输入“10” | 8,8 4,4 2,2 5,5 1,1 6,6 3,3 7,7 | 16 |
| 输入“11” | 8,8 4,4 5,5 2,2 6,6 7,7 3,3 1,1 | 17 |
| 输入“12” | 1,1 2,2 3,3 4,4 5,5 6,6 7,7 8,8 | 18 |
| 输入“13” | 二叉树保存成功 | 19 |
| 输入“2” | 二叉树清空成功 | 20 |
| 输入“19” | 二叉树不存在 | 21 |
| 输入“14” | 二叉树读取成功 | 22 |
| 输入“19” | 打印二叉树 | 23 |
| 输入“0” | 欢迎下次再使用本系统！ | 24 |

表3-3实验结果图

|  |  |
| --- | --- |
| 步骤 | 程序运行图 |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 |  |
| 20 |  |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 |  |

## 3.5 实验小结

在本次实验中，对二叉树的基本操作有了一个整体的理解。二叉树的管理函数与之前最大的不同点便是多次用到递归，并且存储顺序对遍历很不友好。这次实验训练对递归写法的训练很有帮助，也有一些函数例如创建二叉树我先用非递归写了3个小时后再用递归写了一遍，虽然非递归难度很大，但对我也起到了很大的启发作用。这对我的逻辑结构例如栈方法又加强了构造能力。

最能体现逻辑思维的是四种遍历方式，在先序遍历中序遍历和后续遍历的递归函数中访问子树的顺序十分经典。而层序遍历的写法又用到了栈方法，由于课堂上讲到的是递归写法，将递归转化为栈又是一个思考方向，用栈来记录二叉树达到层层入栈层层出栈的算法十分精妙。同样在前三种遍历方式中也可以用栈来代替递归，这是一种很固定的转化方法。

## 4 基于邻接表的图实现

## 4.1 问题描述

图是一个非常重要、且最复杂的一种数据结构，图可分为无向图、有向图、无向网和有向网4种，每一种又有多种形式的物理结构。本实验以邻接表作为无向图的物理结构，通过实现图的基本操作，掌握图的相关知识。

## 4.2 系统设计

## 4.2.1 设计方案

本实验需要设计一个以多图管理为整体框架的图管理系统。其实和之前以多表管理系统下的单表管理系统很像，多图菜单的功能有创建无向图、查找无向图、删除无向图、选择无向图；无向图管理系统的功能包括创建无向图、销毁无向图、查找顶点、顶点赋值、获得第一邻接点、获得下一邻接点、插入顶点、 删除顶点、插入弧、删除弧、深度优先搜索遍历、广度优先搜索遍历、文件存储、文件读取、 打印图。

程序设计和之前一样用循环结构不断实现各种函数的调用来对无向图进行操作。

## 4.2.2 相关定义

定义不同函数的返回值以及执行状态，定义相关线性表的常量，定义相关数据类型

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define MAX\_VERTEX\_NUM 20

#define MAXLENGTH 20

typedef int status;

typedef int KeyType;

typedef enum {DG,DN,UDG,UDN} GraphKind;

typedef struct {

KeyType key;

char others[20];

} VertexType; //顶点类型定义

typedef struct ArcNode { //表结点类型定义

int adjvex; //顶点位置编号

struct ArcNode \*nextarc; //下一个表结点指针

} ArcNode;

typedef struct VNode{ //头结点及其数组类型定义

VertexType data; //顶点信息

ArcNode \*firstarc; //指向第一条弧

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

typedef struct { //邻接表的类型定义

AdjList vertices; //头结点数组

int vexnum,arcnum; //顶点数、弧数

GraphKind kind; //图的类型

} ALGraph;

typedef struct{ //多图类型定义

struct { char name[30];

ALGraph G;

} elem[10];

int length;

int listsize;

}LISTS;

## 4.2.3 构造函数

（1）status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2])

根据V和VR构造图T并返回OK，如果V和VR不正确，返回ERROR

如果有相同的关键字，返回ERROR。

（2）status DestroyGraph(ALGraph &G)

销毁无向图，释放空间，返回OK。

（3）int LocateVex(ALGraph G,KeyType u)

根据u在图G中查找顶点，查找成功返回位序，否则返回-1。

（4）status PutVex(ALGraph &G,KeyType u,VertexType value)

根据u在图G中查找顶点，查找成功将该顶点值修改成value，返回OK；

如果查找失败或关键字不唯一，返回ERROR。

（5）int FirstAdjVex(ALGraph G,KeyType u)

根据u在图G中查找顶点，查找成功返回顶点u的第一邻接顶点位序，否则返回-1。

（6）int NextAdjVex(ALGraph G,KeyType v,KeyType w)

根据u在图G中查找顶点，查找成功返回顶点v的邻接顶点相对于w的下一邻接顶点的位序，查找失败返回-1。

（7）status InsertVex(ALGraph &G,VertexType v)

在图G中插入顶点v，成功返回OK,否则返回ERROR。

（8）status DeleteVex(ALGraph &G,KeyType v)

在图G中删除关键字v对应的顶点以及相关的弧，成功返回OK,否则返回ERROR。

（9）status InsertArc(ALGraph &G,KeyType v,KeyType w)

在图G中增加弧<v,w>，成功返回OK,否则返回ERROR。

（10）status DeleteArc(ALGraph &G,KeyType v,KeyType w)

在图G中删除弧<v,w>，成功返回OK,否则返回ERROR。

（11）status DFSTraverse(ALGraph &G,void (\*visit)(VertexType))

对图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。

（12）status BFSTraverse(ALGraph &G,void (\*visit)(VertexType))

对图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。

（13）status SaveGraph(ALGraph G, char FileName[])

将图的数据写入到文件FileName中，成功返回OK，失败返回ERROR。

（14）status LoadGraph(ALGraph& G, char FileName[])

读入文件FileName的图数据，创建图的邻接表，成功返回OK，失败返回ERROR。

（15）status AddList(LISTS &Lists,char ListName[])

只需要在Lists中增加一个名称为ListName的无向图。

（16）status RemoveList(LISTS &Lists,char ListName[])

Lists中删除一个名称为ListName的无向图。

（17）int LocateList(LISTS Lists,char ListName[])

在Lists中查找一个名称为ListName的无向图，成功返回逻辑序号，否则返回0。

## 4.2.4 复杂性分析

无向图的操作与线性表十分类似，存储结构的大的框架也是线性的。很多函数类似查找结点、删除结点之类的普遍用到循环遍历，因此时间复杂度大多是O(n)。如果含有对弧的操作最多也只是再次遍历弧，因此时间复杂度还是O(n)。

表4-1时间复杂度分析表

|  |  |
| --- | --- |
| CreateGraph | O(n) |
| DestroyGraph | O(n) |
| LocateVex | O(n) |
| PutVex | O(n) |
| FirstAdjVex | O(n) |
| NextAdjVex | O(n) |
| InsertVex | O(n) |
| DeleteVex | O(n) |
| InsertArc | O(n) |
| DeleteArc | O(n) |
| DFSTraverse | O(n) |
| BFSTraverse | O(n) |
| SaveGraph | O(n) |
| LoadGraph | O(n) |
| AddList | O(n) |
| RemoveList | O(n) |
| LocateList | O(n) |

## 4.3 系统实现

（1）status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2])

首先用一个数组记录所有关键字出现的次数，如果数组中出现次数大于或等于2的元素，返回ERROR；循环遍历V创建图的数据顶点，复制data后将首个弧指针置空。再循环遍历VR数组，将符合关键字的结点i和m新建一个弧链接起来。最后返回OK。

（2）status DestroyGraph(ALGraph &G)

循环遍历图的所有结点，用p表示弧指针，循环遍历每个结点的所有弧，用q释放p指向的空间，p继续指向下一个空间，直到p为空。最后把G.vexnum设为0。返回OK。

（3）int LocateVex(ALGraph G,KeyType u)

循环遍历图的所有结点，如果找到一个关键字相同的，就返回循环计数变量i；如果没找到，返回-1。

（4）status PutVex(ALGraph &G,KeyType u,VertexType value)

循环遍历图的所有结点，如果找到一个关键字和value相同的，就返回ERROR。

再次循环遍历结点，如果找到关键字和u相同的，将data赋值为value，返回OK；如果没找到，返回ERROR。

（5）int FirstAdjVex(ALGraph G,KeyType u)

循环遍历图的所有结点，如果找到一个关键字相同的，就返回第一邻接点位序G.vertices[i].firstarc->adjvex；如果没找到，返回-1。

（6）int NextAdjVex(ALGraph G,KeyType v,KeyType w)

循环遍历所有结点，如果找到关键字和v相同的，就用p循环遍历该结点的所有弧，如果有和w相同的弧，返回该弧的下一邻接点的adjvex位序；如果没找到弧或者结点，返回-1。

（7）status InsertVex(ALGraph &G,VertexType v)

首先判断顶点个数是否为20，如果是直接返回ERROR。接着遍历所有顶点，如果找到关键字和v重复的，返回ERROR。再将图的最后一个结点的data赋值为v，vexnum自增，将第一邻接点指针置空，返回OK。

（8）status DeleteVex(ALGraph &G,KeyType v)

首先寻找到v结点，用销毁图中的代码，将顶点v删除，释放空间。再循环遍历其他所有顶点，遍历他们的弧，如果找到有v的邻接点（弧），删除该弧，将上一个指针指向下一个邻接点，释放掉该弧空间。最后遍历所有在v后面的顶点，将他们向后都移一位。最后vexnum自减，最后一个顶点的关键字置为-1，返回OK。如果没有找到v，返回ERROR。

（9）status InsertArc(ALGraph &G,KeyType v,KeyType w)

首先遍历所有顶点，找到关键字v和w的两个顶点，找到一个k加1，遍历完成后k为2表示都找到，若k不等于2返回ERROR。K等于2后，用p指针遍历两个顶点的弧，如果找到<v,w>的弧，返回ERROR。如果没有找到，创建新的邻接点，位序为v和w，分别插入到w顶点和v顶点中，将第一邻接点指向他们，再指向原来的后一个邻接点，返回OK。

（10）status DeleteArc(ALGraph &G,KeyType v,KeyType w)

首先遍历所有顶点，找到关键字为v的顶点后，遍历它的所有邻接点，如果找到w，直接用p指针删除掉这个弧，将上一个指向下一个，释放掉<v,w>空间。如果没找到顶点或者弧，返回ERROR。

（11）status DFSTraverse(ALGraph &G,void (\*visit)(VertexType))

用visit作为标记数组，已经访问过为1，没有为0。循环遍历所有顶点，用DFS函数访问所有结点，DFS中，将该结点visit标记为1后，循环遍历所有邻接点，如果没有访问过，调用DFS访问，返回OK。

（12）status BFSTraverse(ALGraph &G,void (\*visit)(VertexType))

用一个队列对图进行保存，循环遍历所有顶点，用一个标记数组对顶点进行标记，依次将未进队的顶点以及它的邻接点进栈=队，进队的标记为1，直到遍历完所有结点。然后依次把顶点出队，访问顶点，直到队列为空，返回OK。

（13）status SaveGraph(ALGraph G, char FileName[])

用一个文件指针fout打开文件，循环遍历所有顶点，用fprint函数将顶点的相关数据写入文件，再单独将邻接点指针写入文件，关闭文件，返回OK。

（14）status LoadGraph(ALGraph& G, char FileName[])

用一个文件指针fin打开文件，用fscanf获取文件中的data数据，再次获取邻接点数据，创建每个顶点相关的邻接点，关闭文件，返回OK。

（15）status AddList(LISTS &Lists,char ListName[])

循环将ListName 赋给Lists.elem[Lists.length].name,再新建G，Lists.elem[Lists.length].G为G，同时Lists.length自增。

（16）status RemoveList(LISTS &Lists,char ListName[])

循环遍历所有图名Lists.elem[i].name中循环判断Lists.elem[i].name是否为ListName;如果没找到返回ERROR;如果找到，从i开始循环，用i后面的线性表覆盖前一个，达到将第i个删除的效果，最后Lists.length自减。

（17）int LocateList(LISTS Lists,char ListName[])

循环遍历所有图名Lists.elem[i].name中循环判断Lists.elem[i].name是否为ListName;找到返回i+1；没找到返回0。

## 4.4 系统测试

表4-2测试步骤表

|  |  |  |
| --- | --- | --- |
| 操作 | 理论结果 | 步骤 |
| 输入“1 a” | 增加无向图成功 | 1 |
| 输入“1 b” | 增加无向图成功 | 2 |
| 输入“2 b” | 删除无向图成功 | 3 |
| 输入“3 a” | 查找的无向图序号为1 | 4 |
| 输入“4 1” | 无向图管理菜单 | 5 |
| 输入“1  5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1” | 5 线性表 2 3  8 集合 2  7 二叉树 1 3 0  6 无向图 2 0  创建成功！ | 6 |
| 输入“3 5” | 5 线性表 | 7 |
| 输入“4 5 9 kkk” | 9 kkk  8 集合  7 二叉树  6 无向图  赋值成功！ | 8 |
| 输入“4 9 5 线性表” | 5 线性表  8 集合  7 二叉树  6 无向图  赋值成功！ | 9 |
| 输入“5 5” | 7 二叉树 | 10 |
| 输入“6 5 2” | 无下一邻接顶点 | 11 |
| 输入“6 5 7” | 6 无向图 | 12 |
| 输入“7 9 kkk” | 5 线性表 2 3  8 集合 2  7 二叉树 1 3 0  6 无向图 2 0  9 kkk | 13 |
| 输入“8 9” | 5 线性表 2 3  8 集合 2  7 二叉树 1 3 0  6 无向图 2 0 | 14 |
| 输入“9 5 8” | 5 线性表 1 2 3  8 集合 0 2  7 二叉树 1 3 0  6 无向图 2 0 | 15 |
| 输入“10 5 8” | 5 线性表 2 3  8 集合 2  7 二叉树 1 3 0  6 无向图 2 0 | 16 |
| 输入“11” | 5 线性表 7 二叉树 8 集合 6 无向图 | 17 |
| 输入“12” | 5 线性表 7 二叉树 6 无向图 8 集合 | 18 |
| 输入“13” | 文件保存成功 | 19 |
| 输入“2” | 销毁无向图成功 | 20 |
| 输入“14” | 文件输入成功 | 21 |
| 输入“15” | 5 线性表 2 3  8 集合 2  7 二叉树 1 3 0  6 无向图 2 0 | 22 |
| 输入“0” | 多图管理菜单 | 23 |
| 输入“0” | 结束 | 24 |

表4-3实验结果图

|  |  |
| --- | --- |
| 步骤 | 程序运行图 |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 |  |
| 20 |  |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 |  |

## 4.5 实验小结

在本次实验中，我对以邻接表为物理结构的图有了更深入的认识，图的遍历最大的不同点在于，它有关于邻接表，即弧的遍历。在执行删除操作时，需要特别考虑将其他顶点中关于该顶点的弧删除的情况，这花费了我很多时间。图的深度优先遍历与广度优先遍历也是两种很好的遍历算法的例子，一个用到比较经典的标记数组递归方法遍历，另外一个则是利用队列来保存无向图，这有点类似二叉树的层序遍历算法，我写的时候便是往这个方向靠拢的。这是最后一次实验，无向图特殊的存储结构与复杂的结构定义再一次强化了我们对复杂类型的操作训练。

# 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](http://www.calvin.edu/~nyhl/index.html). [ADTs, Data Structures, and Problem Solving with C++.](http://vig.prenhall.com/catalog/academic/product/0,1144,0131409093,00.html)Second Edition, [Calvin College](http://cs.calvin.edu/), 2005

[3] 殷立峰. Qt C++跨平台图形界面程序设计基础. 清华大学出版社,2014:192～197

[4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 附录A 基于顺序存储结构线性表实现的源程序

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

//函数返回状态定义

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

//数据元素类型定义

typedef int status;

typedef int ElemType;

//顺序表（顺序结构）的定义

typedef struct{

ElemType \* elem;

int length;

int listsize;

}SqList;

//线性表的集合类型定义

typedef struct{

struct { char name[30];

SqList L;

} elem[10];

int length;

int listsize;

}LISTS;

LISTS Lists; //线性表集合的定义Lists

//函数声明

status InitList(SqList& L);

status DestroyList(SqList& L);

status ClearList(SqList& L);

status ListEmpty(SqList L);

status ListLength(SqList L);

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

status LocateElem(SqList L,ElemType e);

status PriorElem(SqList L,ElemType e,ElemType &pre);

status NextElem(SqList L,ElemType e,ElemType &next);

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

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

status ListTraverse(SqList L);

status SaveList(SqList L,char FileName[]);

status LoadList(SqList &L,char FileName[]);

status AddList(LISTS &Lists,char ListName[]);

status RemoveList(LISTS &Lists,char ListName[]);

int LocateList(LISTS Lists,char ListName[]);

ElemType e;

int main(void)

{

SqList L;

int operate1=1; //多表管理操作数

int operate2 = 1; //单表管理操作数

int i,pre,next,k; //序号

char listname[30];

while(operate1){

system("cls");

printf("\n\n");

printf(" 欢迎来到多线性表管理菜单 \n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

printf(" 1. 创建线性表 2. 删除线性表\n");

printf(" 3. 查找线性表 4. 选择序号进入线性表管理菜单\n");

printf(" 0. 退出\n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

printf(" Please select your operation[0~4]:\n");

scanf("%d", &operate1);

switch (operate1)

{

case 1:

//新增线性表

printf("\n请输入要增加的线性表名称\n");

scanf("%s", listname);

AddList(Lists,listname);

printf("\n增加线性表成功\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 2:

//删除线性表

printf("\n请输入要删除的线性表名称\n");

scanf("%s", listname);

if(RemoveList(Lists,listname)==OK)

printf("\n删除线性表成功\n");

else

printf("\n删除线性表失败\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 3:

//查找线性表

printf("\n请输入要查找的线性表名称\n");

scanf("%s", listname);

if(LocateList(Lists,listname)!=0)

printf("\n查找的线性表序号为%d\n",LocateList(Lists,listname));

else

printf("\n查找线性表失败\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 4:

//选择线性表进入单表操作系统

printf("选择需要操作的线性表序号\n");

scanf("%d",&k);

L=Lists.elem[k-1].L;

operate2=1;

while (operate2) {

system("cls");

printf("\n\n");

printf(" Welcome to the sequential linear table menu \n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

printf(" 1. 创建线性表 2. 销毁线性表\n");

printf(" 3. 删除线性表 4. 线性表判空\n");

printf(" 5. 线性表求长 6. 获取固定值\n");

printf(" 7. 查找元素号 8. 获取元素前驱\n");

printf(" 9. 获取元素后继 10. 插入元素\n");

printf(" 11. 删除元素 12. 显示元素\n");

printf(" 13. 文件存储 14. 文件读取\n");

printf(" 0. 退出\n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

printf(" Please select your operation[0~14]:\n");

scanf("%d", &operate2);

switch (operate2)

{

case 1:

//创建线性表

if (InitList(L)==OK) {

printf("创建线性表成功");

L.elem[0]=1;

L.elem[L.listsize-1]=2;

}

else

{

printf("创建新线性表失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 2:

//销毁线性表

if (L.elem==NULL)

printf("线性表不存在");

else

{

if (DestroyList(L)==OK)

printf("销毁线性表成功");

else printf("销毁线性表失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 3:

//清空线性表

if (L.elem==NULL)

printf("线性表不存在");

else

{

if (ClearList(L)==OK)

printf("清空线性表成功");

else printf("清空线性表失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 4:

//线性表判空

if (L.elem==NULL)

printf("线性表不存在");

else

{

if (ListEmpty(L)==TRUE)

printf("线性表为空");

else printf("线性表非空");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 5:

//求线性表长

if (L.elem==NULL)

printf("线性表不存在");

else

{

printf("该线性表的长度为%d",ListLength(L));

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 6:

//获取固定序号元素

if (L.elem==NULL)

printf("线性表不存在");

else

{

printf("输入需要查询的序号\n");

scanf("%d",&i);

if(GetElem(L,i,e)==OK) printf("第%d个数的值为%d",i,e);

else printf("查询失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 7:

//查询固定元素序号

if (L.elem==NULL)

printf("线性表不存在");

else

{

printf("输入需要查询的元素\n");

scanf("%d",&e);

if(LocateElem(L,e)!=ERROR) printf("元素%d的序号为%d",e,LocateElem(L,e));

else printf("查询失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 8:

//获取前驱元素

if (L.elem==NULL)

printf("线性表不存在");

else

{

printf("输入需要查询的元素\n");

scanf("%d",&e);

if(PriorElem(L,e,pre)!=ERROR) printf("元素%d的前驱为%d",e,pre);

else printf("查找前驱失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 9:

//获取后继元素

if (L.elem==NULL)

printf("线性表不存在");

else

{

printf("输入需要查询的元素\n");

scanf("%d",&e);

if(NextElem(L,e,next)!=ERROR) printf("元素%d的后继为%d",e,next);

else printf("查找后继失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 10:

//插入元素

if (L.elem==NULL)

printf("线性表不存在");

else

{

printf("输入需要插入的元素\n");

scanf("%d",&e);

printf("输入需要插入的位置\n");

scanf("%d",&i);

if(ListInsert(L,i,e)!=ERROR)

{

printf("插入成功，显示线性表中元素：\n");

for (int k = 1; k <= ListLength(L); k++)

{

GetElem(L, k, e);

printf("%d ", e);

}

}

else printf("插入失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 11:

//删除元素

if (L.elem==NULL)

printf("线性表不存在");

else

{

printf("输入需要删除元素的位置\n");

scanf("%d",&i);

if(ListDelete(L,i,e)!=ERROR)

{

printf("删除成功，显示线性表中元素：\n");

for (int k = 1; k <= ListLength(L); k++)

{

GetElem(L, k, e);

printf("%d ", e);

}

}

else printf("删除失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 12:

//显示线性表元素

if (L.elem==NULL)

printf("线性表不存在");

else

{

ListTraverse(L);

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 13:

//保存文件

if (L.elem==NULL)

printf("线性表不存在");

else

{

printf("\n请输入保存到的文件名称\n");

char filename[30];

scanf("%s", filename);

if (SaveList(L, filename)!=ERROR)

printf("\n保存成功\n");

else

printf("\n保存出错\n");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 14:

//读取文件

if (L.elem==NULL)

printf("线性表不存在");

else

{

printf("\n请输入要加载的文件名称\n");

char filename[30];

scanf("%s", filename);

if (LoadList(L, filename)!=ERROR)

printf("\n文件读取成功\n");

else

printf("\n文件读取错误\n");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

}

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

}

}

return 0;

}

status InitList(SqList& L)

// 线性表L不存在，构造一个空的线性表，返回OK，否则返回INFEASIBLE。

{

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

L.length=0;

L.listsize=100;

return 1;

}

status DestroyList(SqList& L)

// 如果线性表L存在，销毁线性表L，释放数据元素的空间，返回OK，否则返回INFEASIBLE。

{

if(L.elem==NULL) return INFEASIBLE;

else{

free(L.elem);

L.elem=NULL;

return OK;

}

}

status ClearList(SqList& L)

// 如果线性表L存在，删除线性表L中的所有元素，返回OK，否则返回INFEASIBLE。

{

if(L.elem==NULL) return INFEASIBLE;

else{

int i;

for(i=0;i<=L.length-1;i++){

L.elem[i]=NULL;

}

L.length=0;

return OK;

}

}

status ListEmpty(SqList L)

// 如果线性表L存在，判断线性表L是否为空，空就返回TRUE，否则返回FALSE；如果线性表L不存在，返回INFEASIBLE。

{

if(L.elem==NULL) return INFEASIBLE;

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

else return FALSE;

}

status ListLength(SqList L)

// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

{

if(L.elem==NULL) return INFEASIBLE;

else return L.length;

}

status GetElem(SqList L,int i,ElemType &e)

// 如果线性表L存在，获取线性表L的第i个元素，保存在e中，返回OK；如果i不合法，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

if(L.elem==NULL) return INFEASIBLE;

else{

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

else{

e=L.elem[i-1];

return OK;

}

}

}

status LocateElem(SqList L,ElemType e)

// 如果线性表L存在，查找元素e在线性表L中的位置序号并返回OK；如果e不存在，返回ERROR；当线性表L不存在时，返回INFEASIBLE。

{

if(L.elem==NULL) return INFEASIBLE;

else{

int i;

for(i=0;i<=L.length-1;i++){

if(L.elem[i]==e) return i+1;

}

return ERROR;

}

}

status PriorElem(SqList L,ElemType e,ElemType &pre)

// 如果线性表L存在，获取线性表L中元素e的前驱，保存在pre中，返回OK；如果没有前驱，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

if(L.elem==NULL) return INFEASIBLE;

else{

int i;

for(i=0;i<=L.length-1;i++){

if((L.elem[i]==e)&&(i!=0)){

pre=L.elem[i-1];

return OK;

}

}

return ERROR;

}

}

status NextElem(SqList L,ElemType e,ElemType &next)

// 如果线性表L存在，获取线性表L元素e的后继，保存在next中，返回OK；如果没有后继，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

if(L.elem==NULL) return INFEASIBLE;

else{

int i;

for(i=0;i<=L.length-1;i++){

if((L.elem[i]==e)&&(i!=L.length-1)){

next=L.elem[i+1];

return OK;

}

}

return ERROR;

}

}

status ListInsert(SqList &L,int i,ElemType e)

// 如果线性表L存在，将元素e插入到线性表L的第i个元素之前，返回OK；当插入位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

if(L.elem==NULL) return INFEASIBLE;

else{

if((i>L.length+1)||(i<=0)) return ERROR;

else{

int k;

if(L.length>=L.listsize){

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

L.elem=newbase;

L.listsize+=LISTINCREMENT;

}

for(k=L.length;k>=i;k--)

L.elem[k]=L.elem[k-1];

L.elem[i-1]=e;

L.length++;

return OK;

}

}

}

status ListDelete(SqList &L,int i,ElemType &e)

// 如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回OK；当删除位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

if(L.elem==NULL) return INFEASIBLE;

else{

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

else{

int k;

e=L.elem[i-1];

for(k=i-1;k<=L.length-2;k++)

L.elem[k]=L.elem[k+1];

L.length--;

return OK;

}

}

}

status ListTraverse(SqList L)

// 如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，返回OK；如果线性表L不存在，返回INFEASIBLE。

{

if(L.elem==NULL) return INFEASIBLE;

else{

int i;

for(i=0;i<=L.length-1;i++){

if(i==L.length-1) printf("%d",L.elem[i]);

else printf("%d ",L.elem[i]);

}

return OK;

}

}

status SaveList(SqList L,char FileName[])

// 如果线性表L存在，将线性表L的的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

FILE \*fp;

if ((fp = fopen(FileName, "wb")) == NULL)

{

printf("File open error\n ");

return ERROR;

}

fwrite(L.elem, sizeof(ElemType), L.length, fp);

fclose(fp);

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status LoadList(SqList &L,char FileName[])

// 如果线性表L不存在，将FileName文件中的数据读入到线性表L中，返回OK，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

L.length = 0;

FILE \*fp;

if ((fp = fopen(FileName, "rb")) == NULL)

{

printf("File open error\n ");

return ERROR;

}

while (fread(&L.elem[L.length], sizeof(ElemType), 1, fp))

L.length++;

fclose(fp);

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status AddList(LISTS &Lists,char ListName[])

// 只需要在Lists中增加一个名称为ListName的空线性表，线性表数据又后台测试程序插入。

{

int j=0;

for(j=0;ListName[j]!='\0';j++){

Lists.elem[Lists.length].name[j]=ListName[j];

}

Lists.elem[Lists.length].name[j]='\0';

SqList L;

//ElemType \*L.elem=(ElemType \* )malloc(sizeof(ElemType)\*10);

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

L.length=0;

L.listsize=100;

Lists.elem[Lists.length].L=L;

Lists.length+=1;

}

status RemoveList(LISTS &Lists,char ListName[])

// Lists中删除一个名称为ListName的线性表

{

int i;

int k=0;

for(i=0;i<=Lists.length-1;i++){

while(Lists.elem[i].name[k]==ListName[k]){

k++;

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0'))

break;

}

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0'))

break;

}

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0')){

int m;

int n=0;

for(m=i;m<=Lists.length-1;m++)

{

Lists.elem[m].L=Lists.elem[m+1].L;

while(Lists.elem[m].name[n]!='\0'){

Lists.elem[m].name[n]=Lists.elem[m+1].name[n];

n++;

}

}

Lists.length--;

return OK;

}

else return ERROR;

}

int LocateList(LISTS Lists,char ListName[])

// 在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，否则返回0

{

int i;

int k=0;

for(i=0;i<=Lists.length-1;i++){

while(Lists.elem[i].name[k]==ListName[k]){

k++;

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0'))

break;

}

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0'))

break;

}

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0')){

return i+1;

}

else return 0;

}

# 附录B 基于链式存储结构线性表实现的源程序

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#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

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

typedef struct LNode

{ //链表的结构定义

ElemType data;

struct LNode \*next;

}LNode, \*LinkList;

typedef struct{ //线性表的集合类型定义

struct { char name[30];

LinkList L;

} elem[10];

int length;

int listsize;

}LISTS;

LISTS Lists; //线性表集合的定义Lists

status InitList(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);

status PriorElem(LinkList L,ElemType e,ElemType &pre);

status NextElem(LinkList L,ElemType e,ElemType &next);

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

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

status ListTraverse(LinkList L);

status SaveList(LinkList L,char FileName[]);

status LoadList(LinkList &L,char FileName[]);

status AddList(LISTS &Lists,char ListName[]);

status RemoveList(LISTS &Lists,char ListName[]);

int LocateList(LISTS Lists,char ListName[]);

ElemType e;

int main(void)

{

LinkList L;

int operate1=1;

int operate2 = 1;

int i,pre,next,k;//序号

char listname[30];

while(operate1){

system("cls");

printf("\n\n");

printf(" 欢迎来到多线性表管理菜单 \n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

printf(" 1. 创建线性表 2. 删除线性表\n");

printf(" 3. 查找线性表 4. 选择序号进入线性表管理菜单\n");

printf(" 0. 退出\n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

printf(" Please select your operation[0~4]:\n");

scanf("%d", &operate1);

switch (operate1)

{

case 1:

//AddList

printf("\n请输入要增加的线性表名称\n");

scanf("%s", listname);

AddList(Lists,listname);

printf("\n增加线性表成功\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 2:

//RemoveList

printf("\n请输入要删除的线性表名称\n");

scanf("%s", listname);

if(RemoveList(Lists,listname)==OK)

printf("\n删除线性表成功\n");

else

printf("\n删除线性表失败\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 3:

//locateList

printf("\n请输入要查找的线性表名称\n");

scanf("%s", listname);

if(LocateList(Lists,listname)!=0)

printf("\n查找的线性表序号为%d\n",LocateList(Lists,listname));

else

printf("\n查找线性表失败\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 4:

printf("选择需要操作的线性表序号\n");

scanf("%d",&k);

L=Lists.elem[k-1].L;

operate2=1;

while (operate2) {

system("cls");

printf("\n\n");

printf(" Welcome to the sequential linear table menu \n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

printf(" 1. 创建线性表 2. 销毁线性表\n");

printf(" 3. 删除线性表 4. 线性表判空\n");

printf(" 5. 线性表求长 6. 获取固定值\n");

printf(" 7. 查找元素号 8. 获取元素前驱\n");

printf(" 9. 获取元素后继 10. 插入元素\n");

printf(" 11. 删除元素 12. 显示元素\n");

printf(" 13. 文件存储 14. 文件读取\n");

printf(" 0. 退出\n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

printf(" Please select your operation[0~14]:\n");

scanf("%d", &operate2);

switch (operate2)

{

case 1:

//InitList

if (InitList(L)==OK) {

printf("创建线性表成功");

}

else

{

printf("创建新线性表失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 2:

//DestroyList

if (L==NULL)

printf("线性表不存在");

else

{

if (DestroyList(L)==OK)

printf("销毁线性表成功");

else printf("销毁线性表失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 3:

//ClearList

if (L==NULL)

printf("线性表不存在");

else

{

if (ClearList(L)==OK)

printf("清空线性表成功");

else printf("清空线性表失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 4:

//ListEmpty

if (L==NULL)

printf("线性表不存在");

else

{

if (ListEmpty(L)==TRUE)

printf("线性表为空");

else printf("线性表非空");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 5:

//listLenth

if (L==NULL)

printf("线性表不存在");

else

{

printf("该线性表的长度为%d",ListLength(L));

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 6:

//GetElem

if (L==NULL)

printf("线性表不存在");

else

{

printf("输入需要查询的序号\n");

scanf("%d",&i);

if(GetElem(L,i,e)==OK) printf("第%d个数的值为%d",i,e);

else printf("查询失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 7:

//LocateElem

if (L==NULL)

printf("线性表不存在");

else

{

printf("输入需要查询的元素\n");

scanf("%d",&e);

if(LocateElem(L,e)!=ERROR) printf("元素%d的序号为%d",e,LocateElem(L,e));

else printf("查询失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 8:

//PriorElem

if (L==NULL)

printf("线性表不存在");

else

{

printf("输入需要查询的元素\n");

scanf("%d",&e);

if(PriorElem(L,e,pre)!=ERROR) printf("元素%d的前驱为%d",e,pre);

else printf("查找前驱失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 9:

//NextElem

if (L==NULL)

printf("线性表不存在");

else

{

printf("输入需要查询的元素\n");

scanf("%d",&e);

if(NextElem(L,e,next)!=ERROR) printf("元素%d的后继为%d",e,next);

else printf("查找后继失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 10:

//ListInsert

if (L==NULL)

printf("线性表不存在");

else

{

printf("输入需要插入的元素\n");

scanf("%d",&e);

printf("输入需要插入的位置\n");

scanf("%d",&i);

if(ListInsert(L,i,e)!=ERROR)

{

printf("插入成功，显示线性表中元素：\n");

for (int k = 1; k <= ListLength(L); k++)

{

GetElem(L, k, e);

printf("%d ", e);

}

}

else printf("插入失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 11:

//ListDelete

if (L==NULL)

printf("线性表不存在");

else

{

printf("输入需要删除元素的位置\n");

scanf("%d",&i);

if(ListDelete(L,i,e)!=ERROR)

{

printf("删除成功，显示线性表中元素：\n");

for (int k = 1; k <= ListLength(L); k++)

{

GetElem(L, k, e);

printf("%d ", e);

}

}

else printf("删除失败");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 12:

//ListTraverse

if (L==NULL)

printf("线性表不存在");

else

{

ListTraverse(L);

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 13:

//SaveList

if (L==NULL)

printf("线性表不存在");

else

{

printf("\n请输入保存到的文件名称\n");

char filename[30];

scanf("%s", filename);

if (SaveList(L, filename)!=ERROR)

printf("\n保存成功\n");

else

printf("\n保存出错\n");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 14:

//LoadList

if (L==NULL)

printf("线性表不存在");

else

{

printf("\n请输入要加载的文件名称\n");

char filename[30];

scanf("%s", filename);

if (LoadList(L, filename)!=ERROR)

printf("\n文件读取成功\n");

else

printf("\n文件读取错误\n");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

}

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

}

}

return 0;

}

status InitList(LinkList &L)

// 线性表L不存在，构造一个空的线性表，返回OK，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L!=NULL) return INFEASIBLE;

else

{

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

L->next=NULL;

return OK;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status DestroyList(LinkList &L)

// 如果线性表L存在，销毁线性表L，释放数据元素的空间，返回OK，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL) return INFEASIBLE;

else

{

LinkList p,q;

p=L;

while(p!=NULL)

{

q=p->next;

free(p);

p=q;

}

L=NULL;

return OK;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ClearList(LinkList &L)

// 如果线性表L存在，删除线性表L中的所有元素，返回OK，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL) return INFEASIBLE;

else

{

LinkList p,q;

p=L->next;

while(p!=NULL)

{

q=p->next;

free(p);

p=q;

}

L->next=NULL;

return OK;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListEmpty(LinkList L)

// 如果线性表L存在，判断线性表L是否为空，空就返回TRUE，否则返回FALSE；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL) return INFEASIBLE;

else

{

if(L->next) return FALSE;

else return TRUE;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

int ListLength(LinkList L)

// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL) return INFEASIBLE;

else

{

LinkList p;

int i=0;

p=L->next;

while(p)

{

p=p->next;

i++;

}

return i;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status GetElem(LinkList L,int i,ElemType &e)

// 如果线性表L存在，获取线性表L的第i个元素，保存在e中，返回OK；如果i不合法，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL) return INFEASIBLE;

else if(i<1) return ERROR;

else

{

LinkList p;

p=L;

for(int k=0;k<=i-1;k++)

{

p=p->next;

if(p==NULL) return ERROR;

}

if(p->data!=NULL)

{

e=p->data;

return OK;

}

else return ERROR;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status LocateElem(LinkList L,ElemType e)

// 如果线性表L存在，查找元素e在线性表L中的位置序号；如果e不存在，返回ERROR；当线性表L不存在时，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL) return INFEASIBLE;

else

{

LinkList p;

p=L;

int i=0;

while((p!=NULL)&&(p->data!=e))

{

p=p->next;

i++;

}

if(p)

{

return i;

}

else return ERROR;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status PriorElem(LinkList L,ElemType e,ElemType &pre)

// 如果线性表L存在，获取线性表L中元素e的前驱，保存在pre中，返回OK；如果没有前驱，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL) return INFEASIBLE;

else

{

LinkList p,q;

p=L;

int i=0;

while((p!=NULL)&&(p->data!=e))

{

q=p;

p=p->next;

i++;

}

if((p)&&(q!=L))

{

pre=q->data;

return OK;

}

else return ERROR;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status NextElem(LinkList L,ElemType e,ElemType &next)

// 如果线性表L存在，获取线性表L元素e的后继，保存在next中，返回OK；如果没有后继，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL) return INFEASIBLE;

else

{

LinkList p;

p=L;

int i=0;

while((p!=NULL)&&(p->data!=e))

{

p=p->next;

i++;

}

if((p)&&(p->next!=NULL))

{

next=p->next->data;

return OK;

}

else return ERROR;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

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

// 如果线性表L存在，将元素e插入到线性表L的第i个元素之前，返回OK；当插入位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL) return INFEASIBLE;

else if(i<1) return ERROR;

else

{

LinkList p,q,l;

p=L;

for(int k=0;k<=i-1;k++)

{

q=p;

if(p==NULL) return ERROR;

p=p->next;

}

l=(LinkList)malloc(sizeof(LNode));

l->next=q->next;

q->next=l;

l->data=e;

return OK;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListDelete(LinkList &L,int i,ElemType &e)

// 如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回OK；当删除位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL) return INFEASIBLE;

else if(i<1) return ERROR;

else

{

LinkList p,q,l;

p=L;

for(int k=0;k<=i-1;k++)

{

q=p;

p=p->next;

if(p==NULL) return ERROR;

}

e=p->data;

q->next=p->next;

p->next=NULL;

free(p);

return OK;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListTraverse(LinkList L)

// 如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，返回OK；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL) return INFEASIBLE;

else

{

LinkList p,q,l;

p=L->next;

if(p)

{

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

p=p->next;

while(p)

{

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

p=p->next;

}

return OK;

}

else printf("空线性表");

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status SaveList(LinkList L,char FileName[])

// 如果线性表L存在，将线性表L的的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 1 \*\*\*\*\*\*\*\*\*/

if (L==NULL) return INFEASIBLE;

else {

LinkList p;

p=L->next;

FILE\* fp;

fp = fopen(FileName, "wb");

if (!fp) exit(-1);

while(p)

{

fwrite(p, 2\*sizeof(LinkList), 1, fp);

p=p->next;

}

fclose(fp);

return OK;

}

/\*\*\*\*\*\*\*\*\*\* End 1 \*\*\*\*\*\*\*\*\*\*/

}

status LoadList(LinkList &L,char FileName[])

// 如果线性表L不存在，将FileName文件中的数据读入到线性表L中，返回OK，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 2 \*\*\*\*\*\*\*\*\*/

L->data = 0;

LinkList p = L;

FILE \*fp;

if ((fp = fopen(FileName, "rb")) == NULL)

{

printf("File open error\n ");

return ERROR;

}

LNode temp;

while (fread(&temp, 2\*sizeof(LinkList), 1, fp))

{

p->next = (LinkList)malloc(2 \* sizeof(LinkList));

p = p->next;

\*p = temp;

L->data++;

}

p->next = NULL;

fclose(fp);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 2 \*\*\*\*\*\*\*\*\*\*/

}

status AddList(LISTS &Lists,char ListName[])

// 只需要在Lists中增加一个名称为ListName的空线性表，线性表数据又后台测试程序插入。

{

int j=0;

for(j=0;ListName[j]!='\0';j++){

Lists.elem[Lists.length].name[j]=ListName[j];

}

Lists.elem[Lists.length].name[j]='\0';

LinkList L;

//ElemType \*L.elem=(ElemType \* )malloc(sizeof(ElemType)\*10);

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

L->next=NULL;

Lists.elem[Lists.length].L=L;

Lists.length+=1;

}

status RemoveList(LISTS &Lists,char ListName[])

// Lists中删除一个名称为ListName的线性表

{

int i;

int k=0;

for(i=0;i<=Lists.length-1;i++){

while(Lists.elem[i].name[k]==ListName[k]){

k++;

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0'))

break;

}

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0'))

break;

}

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0')){

int m;

int n=0;

for(m=i;m<=Lists.length-1;m++)

{

Lists.elem[m].L=Lists.elem[m+1].L;

while(Lists.elem[m].name[n]!='\0'){

Lists.elem[m].name[n]=Lists.elem[m+1].name[n];

n++;

}

}

Lists.length--;

return OK;

}

else return ERROR;

}

int LocateList(LISTS Lists,char ListName[])

// 在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，否则返回0

{

int i;

int k=0;

for(i=0;i<=Lists.length-1;i++){

while(Lists.elem[i].name[k]==ListName[k]){

k++;

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0'))

break;

}

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0'))

break;

}

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0')){

return i+1;

}

else return 0;

}

# 附录C 基于二叉链表二叉树实现的源程序

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status;

typedef int KeyType;

typedef struct

{

KeyType key;

char others[20];

} TElemType; //二叉树结点类型定义

typedef struct

{

int pos;

TElemType data;

} DEF;

typedef struct BiTNode

{ //二叉链表结点的定义

TElemType data;

struct BiTNode \*lchild, \*rchild;

} BiTNode, \*BiTree;

typedef struct Forest

{ //森林定义

BiTree tree;

char name[30];

struct Forest \*next;

} Forest, \*Fp;

void visit(BiTree T)

{

printf(" %d,%s", T->data.key, T->data.others);

}

BiTree case1(BiTree &T, bool flag = true);

int \*getkeys(BiTree &T, int \*keys = NULL) //获取所有key值

{

if (T)

{

if (!keys)

keys = new int[1000]{0};

keys[T->data.key]++;

getkeys(T->lchild, keys);

getkeys(T->rchild, keys);

}

return keys;

}

status checkKey(TElemType \*keys) //检查序列中有无重复key值

{

int marks[10000]{0};

for (int i = 0; i < 100; i++, keys++)

{

if (keys->key)

{

if (keys->key == -1)

return i;

if (++marks[keys->key - 1] == 2)

return ERROR;

}

}

return OK;

}

BiTNode \*LocateParent(BiTree T, KeyType e)

//查找结点

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

BiTNode \*temp = (BiTNode \*)malloc(sizeof(BiTNode));

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

{

if (T->lchild && T->lchild->data.key == e || T->rchild && T->rchild->data.key == e)

return T;

if ((temp = LocateParent(T->lchild, e)))

return temp;

else

return LocateParent(T->rchild, e);

}

return NULL;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status CreateBiTree(BiTree &T, DEF definition[])

{

static int m=0;

if(m==0)

{

m++;

int k;

int b[200];

for(k=0;k<=99;k++) b[k]=0;

k=0;

while(definition[k].pos!=0)

{

if(definition[k].data.key!=0)

{

b[definition[k].data.key]+=1;

}

k++;

}

for(k=0;k<=99;k++)

{

if(b[k]>=2)

{

return ERROR;

}

}

}

int i = 0, j;

static BiTNode \*p[100];

while (j = definition[i].pos)

{

p[j] = (BiTNode \*)malloc(sizeof(BiTNode));

p[j]->data = definition[i].data;

p[j]->lchild = NULL;

p[j]->rchild = NULL;

if (j != 1)

if (j % 2)

p[j / 2]->rchild = p[j];

else

p[j / 2]->lchild = p[j];

i++;

}

T = p[1];

return OK;

}

status ClearBiTree(BiTree &T)

//将二叉树设置成空，并删除所有结点，释放结点空间

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (T->lchild)

ClearBiTree(T->lchild);

if (T->rchild)

ClearBiTree(T->rchild);

free(T);

T = NULL;

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

int BiTreeDepth(BiTree T)

//求二叉树T的深度

{

static int i = 0, count = 0;

if (T)

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

i++;

if (i > count)

count = i;

BiTreeDepth(T->lchild);

BiTreeDepth(T->rchild);

i--;

}

else

{

return INFEASIBLE;

}

return count;

}

BiTNode \*LocateNode(BiTree T, KeyType e)

//查找结点

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

BiTNode \*temp = (BiTNode \*)malloc(sizeof(BiTNode));

if (T)

{

if (T->data.key == e)

return T;

if ((temp = LocateNode(T->lchild, e)))

return temp;

else

return LocateNode(T->rchild, e);

}

return NULL;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status Assign(BiTree &T, KeyType e, TElemType value)

//实现结点赋值。此题允许通过增加其它函数辅助实现本关任务

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

BiTree Node = LocateNode(T, e);

int \*keys = getkeys(T);

if (!Node)

return INFEASIBLE;

if (Node->data.key != value.key && keys[value.key])

return ERROR;

else

Node->data.key = value.key;

strcpy(Node->data.others, value.others);

return OK;

}

BiTNode \*GetSibling(BiTree T, KeyType e)

//实现获得兄弟结点

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

BiTree parent = LocateParent(T, e);

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

{

if (parent->lchild->data.key == e)

return parent->rchild;

else

return parent->lchild;

}

return NULL;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status InsertNode(BiTree &T, KeyType e, int LR, TElemType c)

//插入结点。此题允许通过增加其它函数辅助实现本关任务

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int \*keys = getkeys(T);

if (keys[c.key])

return ERROR;

if (LR == -1)

{

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

Node->data = c;

Node->rchild = T;

Node->lchild = NULL;

T = Node;

return OK;

}

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

if (root = LocateNode(T, e))

{

if (LR == 0)

{

temp = root->lchild;

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

Node->data = c;

Node->rchild = temp;

Node->lchild = NULL;

root->lchild = Node;

}

else

{

temp = root->rchild;

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

Node->data = c;

Node->rchild = temp;

Node->lchild = NULL;

root->rchild = Node;

}

return OK;

}

return ERROR;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status DeleteNode(BiTree &T, KeyType e)

//删除结点。此题允许通过增加其它函数辅助实现本关任务

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

BiTree parent = LocateParent(T, e);

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

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

int flag = -1;

if (!parent)

if (T->data.key == e)

{

root = T;

flag = 1;

}

else

return ERROR;

else if (parent->rchild->data.key == e)

{

root = parent->rchild;

flag = 2;

}

else

{

root = parent->lchild;

flag = 3;

}

if (root->lchild)

{

BiTree maxrchild = root->lchild;

while (maxrchild->rchild)

maxrchild = maxrchild->rchild;

maxrchild->rchild = root->rchild;

temp = root->lchild;

}

else if (root->rchild)

temp = root->rchild;

else

temp = NULL;

free(root);

if (flag == 1)

T = temp;

else if (flag == 2)

parent->rchild = temp;

else

parent->lchild = temp;

return OK;

}

status PreOrderTraverse(BiTree T, void (\*visit)(BiTree))

//先序遍历二叉树T

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (T)

{

visit(T);

PreOrderTraverse(T->lchild, visit);

PreOrderTraverse(T->rchild, visit);

}

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status InOrderTraverse(BiTree T, void (\*visit)(BiTree))

//中序遍历二叉树T

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (T)

{

InOrderTraverse(T->lchild, visit);

visit(T);

InOrderTraverse(T->rchild, visit);

}

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status PostOrderTraverse(BiTree T, void (\*visit)(BiTree))

//后序遍历二叉树T

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (T)

{

PostOrderTraverse(T->lchild, visit);

PostOrderTraverse(T->rchild, visit);

visit(T);

}

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status LevelOrderTraverse(BiTree T, void (\*visit)(BiTree))

//按层遍历二叉树T

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

BiTree s[100];

for(int x=0;x<=99;x++) s[x]=NULL;

if(T)

{

s[0]=T;

}

BiTree p;

int i=0,k=1,m=0;

while(k)

{

p=s[m];

m++;

visit(p);

k--;

if(p->lchild)

{

i++;

k++;

s[i]=p->lchild

}

if(p->rchild)

{

i++;

k++;

s[i]=p->rchild;

}

}

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status SaveBiTree(BiTree T, char FileName[])

//将二叉树的结点数据写入到文件FileName中

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 1 \*\*\*\*\*\*\*\*\*/

if (!T)

return ERROR;

BiTree \*list = (BiTree \*)malloc(sizeof(BiTree) \* 1000);

list[0] = T;

BiTree Tlist = (BiTree)malloc(sizeof(BiTNode) \* 1000);

Tlist[0] = \*T;

int head = -1, tail = 1;

FILE \*fp;

if (!(fp = fopen(FileName, "w+b")))

return ERROR;

while (++head < tail)

{

if (list[head])

{

list[tail++] = list[head]->lchild;

list[tail++] = list[head]->rchild;

Tlist[head] = \*(list[head]);

}

}

fwrite(&tail, sizeof(int), 1, fp);

fwrite(list, sizeof(BiTree), tail, fp);

fwrite(Tlist, sizeof(BiTNode), tail, fp);

fclose(fp);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 1 \*\*\*\*\*\*\*\*\*\*/

}

status LoadBiTree(BiTree &T, char FileName[])

//读入文件FileName的结点数据，创建二叉树

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 2 \*\*\*\*\*\*\*\*\*/

if (T)

return ERROR;

BiTree list[1000];

int head = -1, tail = 1;

FILE \*fp;

if (!(fp = fopen(FileName, "r+b")))

return ERROR;

fread(&tail, sizeof(int), 1, fp);

fread(list, sizeof(BiTree), tail, fp);

while (head++ < tail - 1)

{

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

fread(tp, sizeof(BiTNode), 1, fp);

if (list[head])

{

list[head] = tp;

}

}

head = -1, tail = 1;

while (head++ < tail - 1)

{

BiTree tp = (BiTree)malloc(sizeof(BiTNode) \* 1);

tp = list[head];

if (!T)

T = tp;

if (tp)

{

tp->lchild = list[tail++];

tp->rchild = list[tail++];

}

}

fclose(fp);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 2 \*\*\*\*\*\*\*\*\*\*/

}

Fp AddTree(Fp &F, char \*name)

{

strcpy(F->name, name);

F->next = NULL;

F->tree = NULL;

case1(F->tree, false);

return F;

}

Fp LocateTree(Fp F, char \*name)

{

while (F)

{

if (!strcmp(F->name, name))

return F;

F = F->next;

}

return NULL;

}

Fp deleteTree(Fp &F, char \*name)

{

Fp temp = F, pre; //从多表的首元结点开始查询

while (temp) //多表的当前结点存在继续遍历

{

if (!strcmp(temp->name, name)) //当前链表的名字与输入名字相同

{

pre->next = pre->next->next; //从多表链表中删除当前链表

ClearBiTree(temp->tree); //释放当前链表空间

temp->tree = NULL;

free(temp);

return

}

pre = temp; //遍历到下一链表

temp = temp->next;

}

return NULL;

}

status switchList(Fp F, char \*name, BiTree &T)

{

if (LocateTree(F, name))

{

T = LocateTree(F, name)->tree;

return OK;

}

return ERROR;

}

void printTree(BiTree T, int h = 0) {

int i;

if (T != NULL) {

printTree(T->rchild, h+1);

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

putchar('\t');

printf("%d %s", T->data.key, T->data.others);

putchar('\n');

printTree(T->lchild, h+1);

}

}

BiTree case1(BiTree &T, bool flag)

{

if (T)

printf("二叉树存在");

else

{

printf("请输入新的pos值, key值和others值\n");

DEF definition[100];

int i = 0;

do

{

scanf("%d%d%s", &definition[i].pos, &definition[i].data.key, definition[i].data.others);

} while (definition[i++].pos);

if (CreateBiTree(T, definition))

printf("二叉树创建成功\n");

else

printf("二叉树创建失败\n");

}

if(flag)

printf("回车键回到主菜单\n");

return T;

}

void case2(BiTree &T)

{

if (!T)

printf("二叉树不存在\n");

else

{

if (ClearBiTree(T))

printf("二叉树清空成功\n");

else

printf("二叉树清空失败\n");

}

printf("输入回车键返回主菜单\n");

}

void case3(BiTree &T)

{

int depth = BiTreeDepth(T);

if (depth)

printf("二叉树的深度为%d\n", depth);

else if (depth == 0)

printf("二叉树深度为0\n");

else

printf("二叉树不存在\n");

printf("输入回车键返回主菜单\n");

}

void case4(BiTree &T)

{

int key;

if (!T)

printf("二叉树不存在\n");

else

{

printf("请输入你想要查找的结点的key值:\n");

scanf("%d", &key);

BiTree Node = LocateNode(T, key);

if (Node)

printf("该key值对应的结点的内容为%s\n", Node->data.others);

else

printf("查找失败\n");

}

printf("输入回车键返回主菜单\n");

}

void case5(BiTree &T)

{

if (!T)

printf("二叉树不存在");

else

{

int key;

TElemType def;

printf("请输入你想要赋值的结点的key值:\n");

scanf("%d", &key);

printf("请输入新的key值和others值\n");

scanf("%d %s", &def.key, def.others);

int f = Assign(T, key, def);

if (f == INFEASIBLE)

printf("结点查找失败\n");

else if (f)

printf("结点赋值成功\n");

else

printf("二叉树中已有待赋值的key值\n");

}

printf("输入回车键返回主菜单\n");

}

void case6(BiTree &T)

{

if (!T)

printf("二叉树不存在\n");

else

{

int key;

printf("请输入结点的key值:\n");

scanf("%d", &key);

BiTree Node = GetSibling(T, key);

if (!Node)

printf("结点的兄弟结点不存在\n");

else

{

printf("结点的兄弟结点的key值为%d,others值为%s", Node->data.key, Node->data.others);

}

}

printf("输入回车键返回主菜单\n");

}

void case7(BiTree &T)

{

if (!T)

printf("二叉树不存在\n");

else

{

int key, LR;

TElemType value;

printf("请输入你想要插入的结点的父结点key值:\n");

scanf("%d", &key);

printf("插入位置为左叶子结点输入0，右叶子结点输入1\n");

scanf("%d", &LR);

printf("请输入你想要插入的结点的key值和others值:\n");

scanf("%d %s", &value.key, value.others);

int flag = InsertNode(T, key, LR, value);

if (flag)

printf("插入成功\n");

else

printf("插入失败\n");

}

printf("输入回车键返回主菜单\n");

}

void case8(BiTree &T)

{

if (!T)

printf("二叉树不存在\n");

else

{

int key;

printf("请输入你想要删除的结点的key值\n");

scanf("%d", &key);

int flag = DeleteNode(T, key);

if (flag)

printf("删除成功\n");

else

printf("结点不存在\n");

}

printf("输入回车键返回主菜单\n");

}

void case9(BiTree &T)

{

if (!T)

printf("二叉树不存在\n");

else

PreOrderTraverse(T, visit);

printf("\n输入回车键返回主菜单\n");

}

void case10(BiTree &T)

{

if (!T)

printf("二叉树不存在\n");

else

InOrderTraverse(T, visit);

printf("\n输入回车键返回主菜单\n");

}

void case11(BiTree &T)

{

if (!T)

printf("二叉树不存在\n");

else

PostOrderTraverse(T, visit);

printf("\n输入回车键返回主菜单\n");

}

void case12(BiTree &T)

{

if (!T)

printf("二叉树不存在\n");

else

LevelOrderTraverse(T, visit);

printf("\n输入回车键返回主菜单\n");

}

void save(BiTree &T)

{

if (!T)

printf("二叉树不存在");

else

{

char filename[30];

strcpy(filename, "./list.dat");

int flag = SaveBiTree(T, filename);

if (flag)

printf("二叉树保存成功\n");

else

printf("二叉树保存失败\n");

}

printf("输入回车键返回主菜单\n");

}

void read(BiTree &T)

{

if (T)

printf("二叉树已存在");

else

{

char filename[30];

strcpy(filename, "./list.dat");

int flag = LoadBiTree(T, filename);

if (flag)

printf("二叉树读取成功\n");

else

printf("二叉树读取失败\n");

}

printf("输入回车键返回主菜单\n");

}

void case14(Fp &F)

{

if(!F){

F = (Fp)malloc(sizeof(Forest));

F->next = NULL;

F->tree = NULL;

}

Fp tf = F;

while (tf->next)

{

tf = tf->next;

}

int count;

printf("请输入想要种多少颗二叉树:\n");

scanf("%d", &count);

char s[30];

while (count--)

{

tf->next = (Fp)malloc(sizeof(Forest));

printf("请输入二叉树的名字:\n");

scanf("%s", s);

AddTree(tf->next, s);

tf = tf->next;

}

tf = F;

while (tf)

{

PreOrderTraverse(tf->tree, visit);

tf = tf->next;

printf("\n");

}

printf("输入回车键返回主菜单\n");

}

void case15(Fp &F)

{

if (!F)

printf("森林不存在\n");

else

{

char s[30];

printf("请输入二叉树的名字\n");

scanf("%s", s);

if (LocateTree(F, s))

{

PreOrderTraverse(LocateTree(F, s)->tree, visit);

printf("\n");

}

else

printf("查找失败\n");

}

printf("输入回车键返回主菜单\n");

}

void case16(Fp &F)

{

Fp tf = F;

if (!F)

printf("森林不存在\n");

else

{

char s[30];

printf("请输入二叉树的名字\n");

scanf("%s", s);

if (deleteTree(F, s))

{

while (tf)

{

PreOrderTraverse(tf->tree, visit);

tf = tf->next;

printf("\n");

}

}

else

printf("查找失败\n");

}

printf("输入回车键返回主菜单\n");

}

void case17(Fp &F, BiTree &T)

{

if (!F)

printf("森林不存在\n");

else

{

char name[30];

printf("请输入你想要切换的表的名字\n");

scanf("%s", name);

if (switchList(F, name, T) == OK)

printf("切换成功\n");

else

printf("切换失败，请检查你输入的名称\n");

}

printf("回车返回主菜单\n");

}

void case18(BiTree T){

if (!T)

printf("二叉树不存在");

else{

printTree(T);

}

printf("回车返回主菜单\n");

}

int main(void)

{

BiTree T = NULL;

Fp F = NULL;

int op = 1;

while (op)

{

system("cls");

printf("\n\n");

printf(" Welcome to the sequential linear table menu \n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

printf(" 1. 创建二叉树 2. 清空二叉树\n");

printf(" 3. 求二叉树深度 4. 查找结点\n");

printf(" 5. 结点赋值 6. 获得兄弟结点\n");

printf(" 7. 插入结点 8. 删除结点\n");

printf(" 9. 先序遍历 10. 中序遍历\n");

printf(" 11. 后序遍历 12. 按层遍历\n");

printf(" 13. 文件存储 14. 文件读取\n");

printf(" 15. 添加树 16. 查找树\n");

printf(" 17. 删除树 18. 选择树\n");

printf(" 19.打印树 0. 结束 \n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

printf(" 请选择你的操作[0~19]:");

scanf("%d", &op);

switch (op)

{

case 1:

case1(T);

getchar();

getchar();

break;

case 2:

case2(T);

getchar();

getchar();

break;

case 3:

case3(T);

getchar();

getchar();

break;

case 4:

case4(T);

getchar();

getchar();

break;

case 5:

case5(T);

getchar();

getchar();

break;

case 6:

case6(T);

getchar();

getchar();

break;

case 7:

case7(T);

getchar();

getchar();

break;

case 8:

case8(T);

getchar();

getchar();

break;

case 9:

case9(T);

getchar();

getchar();

break;

case 10:

case10(T);

getchar();

getchar();

break;

case 11:

case11(T);

getchar();

getchar();

break;

case 12:

case12(T);

getchar();

getchar();

break;

case 13:

save(T);

getchar();

getchar();

break;

case 14:

read(T);

getchar();

getchar();

break;

case 15:

case14(F);

getchar();

getchar();

break;

case 16:

case15(F);

getchar();

getchar();

break;

case 17:

case16(F);

getchar();

getchar();

break;

case 18:

case17(F, T);

getchar();

getchar();

break;

case 19:

case18(T);

getchar();

getchar();

break;

case 0:

break;

} //end of switch

} //end of while

printf("欢迎下次再使用本系统！\n");

return 1;

} //end of main()

# 附录D 基于邻接表图实现的源程序

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#include<string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define MAX\_VERTEX\_NUM 20

#define MAXLENGTH 20

typedef int status;

typedef int KeyType;

typedef enum {DG,DN,UDG,UDN} GraphKind;

typedef struct {

KeyType key;

char others[20];

} VertexType; //顶点类型定义

typedef struct ArcNode { //表结点类型定义

int adjvex; //顶点位置编号

struct ArcNode \*nextarc; //下一个表结点指针

} ArcNode;

typedef struct VNode{ //头结点及其数组类型定义

VertexType data; //顶点信息

ArcNode \*firstarc; //指向第一条弧

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

typedef struct { //邻接表的类型定义

AdjList vertices; //头结点数组

int vexnum,arcnum; //顶点数、弧数

GraphKind kind; //图的类型

} ALGraph;

typedef struct{ //线性表的集合类型定义

struct { char name[30];

ALGraph G;

} elem[10];

int length;

int listsize;

}LISTS;

LISTS Lists;

status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2]);

status DestroyGraph(ALGraph &G);

int LocateVex(ALGraph G,KeyType u);

status PutVex(ALGraph &G,KeyType u,VertexType value);

int FirstAdjVex(ALGraph G,KeyType u);

int NextAdjVex(ALGraph G,KeyType v,KeyType w);

status InsertVex(ALGraph &G,VertexType v);

status DeleteVex(ALGraph &G,KeyType v);

status InsertArc(ALGraph &G,KeyType v,KeyType w);

status DeleteArc(ALGraph &G,KeyType v,KeyType w);

status DFSTraverse(ALGraph &G,void (\*visit)(VertexType));

status BFSTraverse(ALGraph &G,void (\*visit)(VertexType));

status SaveGraph(ALGraph G, char FileName[]);

status LoadGraph(ALGraph &G, char FileName[]);

status AddList(LISTS &Lists,char ListName[]);

status RemoveList(LISTS &Lists,char ListName[]);

int LocateList(LISTS Lists,char ListName[]);

int visited[20];//标记访问数组，已经访问为1，为访问为0；

void DFS(ALGraph &G,void (\*visit)(VertexType),int i)

{

ArcNode \* p;

(\*visit)(G.vertices[i].data);

visited[i]=1;

for(p=G.vertices[i].firstarc;p!=NULL;p=p->nextarc)

{

if(!visited[p->adjvex]) DFS(G,visit,p->adjvex);

}

}

void visit(VertexType v)

{

printf(" %d %s",v.key,v.others);

}

int inqueue[20];//标记是否已经在访问队列，或已经访问

typedef struct QUEUE {

int elem[MAXLENGTH];

int front,length;

} QUEUE;

void iniQueue(QUEUE &Q)

//该函数实现初始化Q

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Q.front=0;

Q.length=0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

int enQueue(QUEUE &Q,int e)

//将元素e入队Q。成功入栈返回1，否则返回0

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(Q.length<MAXLENGTH)

{

Q.elem[(Q.front+Q.length)%MAXLENGTH]=e;

Q.length++;

return 1;

}

else return 0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

int deQueue(QUEUE &Q, int &e)

//将Q队首元素出队，赋值给e。成功出队返回1，否则返回0

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(Q.length>0)

{

e=Q.elem[Q.front];

Q.front=(Q.front+1)%MAXLENGTH;

Q.length--;

return 1;

}

return 0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

int emptyQueue(QUEUE Q)

{

if(Q.length==0) return TRUE;

else return FALSE;

}

int main(void)

{

VertexType V[30];

KeyType VR[100][2];

VertexType value;

ALGraph G;

int operate1=1;

int operate2 = 1;

int i,k,v,w,u;//序号

char listname[30];

char x[] = "C:\\Users\\86153\\高博文\\a.txt";

while(operate1){

system("cls");

printf("\n\n");

printf(" 欢迎来到多图管理菜单 \n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

printf(" 1. 创建无向图 2. 删除无向图\n");

printf(" 3. 查找无向图 4. 选择无向图\n");

printf(" 0. 退出\n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

printf(" Please select your operation[0~4]:\n");

scanf("%d", &operate1);

switch (operate1)

{

case 1:

//AddList

printf("\n请输入要增加的无向图名称\n");

scanf("%s", listname);

AddList(Lists,listname);

printf("\n增加无向图成功\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 2:

//RemoveList

printf("\n请输入要删除的无向图名称\n");

scanf("%s", listname);

if(RemoveList(Lists,listname)==OK)

printf("\n删除无向图成功\n");

else

printf("\n删除无向图失败\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 3:

//locateList

printf("\n请输入要查找的无向图名称\n");

scanf("%s", listname);

if(LocateList(Lists,listname)!=0)

printf("\n查找的无向图序号为%d\n",LocateList(Lists,listname));

else

printf("\n查找无向图失败\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 4:

printf("选择需要操作的无向图序号\n");

scanf("%d",&k);

G=Lists.elem[k-1].G;

operate2=1;

while (operate2) {

system("cls");

printf("\n\n");

printf(" Welcome to the sequential linear table menu \n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

printf(" 1. 创建无向图 2. 销毁无向图\n");

printf(" 3. 查找顶点 4. 顶点赋值\n");

printf(" 5. 获得第一邻接点 6. 获得下一邻接点\n");

printf(" 7. 插入顶点 8. 删除顶点\n");

printf(" 9. 插入弧 10. 删除弧\n");

printf(" 11. 深度优先搜索遍历 12. 广度优先搜索遍历\n");

printf(" 13. 文件存储 14. 文件读取\n");

printf(" 15. 打印图 0. 退出\n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");

printf(" Please select your operation[0~14]:\n");

scanf("%d", &operate2);

switch (operate2)

{

case 1:

//CreateGraph

i=0;

printf("输入无向图的结点数据\n");

do

{

scanf("%d%s",&V[i].key,V[i].others);

} while(V[i++].key!=-1);

i=0;

printf("输入弧的关系\n");

do

{

scanf("%d%d",&VR[i][0],&VR[i][1]);

} while(VR[i++][0]!=-1);

if (CreateGraph(G,V,VR)==ERROR) printf("输入数据错，无法创建");

else for(i=0;i<G.vexnum;i++)

{

ArcNode \*p=G.vertices[i].firstarc;

printf("%d %s",G.vertices[i].data.key,G.vertices[i].data.others);

while (p)

{

printf(" %d",p->adjvex);

p=p->nextarc;

}

printf("\n");

}

printf("创建成功！");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 2:

//DestroyGraph

if (DestroyGraph(G)==OK)

printf("销毁无向图成功\n");

else printf("销毁无向图失败\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 3:

//LocateVex

printf("输入需要查找的结点\n");

scanf("%d",&u);

i=LocateVex(G,u);

if (i!=-1)

printf("%d %s",G.vertices[i].data.key,G.vertices[i].data.others);

else

printf("查找失败\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 4:

//PutVex

printf("输入需要赋值的结点u\n");

scanf("%d",&u);

printf("输入新的结点value的数据结构\n");

scanf("%d%s",&value.key,value.others);

i=PutVex(G,u,value);

if (i==OK)

{

for(i=0;i<G.vexnum;i++)

printf(" %d %s\n",G.vertices[i].data.key,G.vertices[i].data.others);

printf("赋值成功！");

}

else

printf("赋值操作失败");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 5:

//FirstAdjVex

printf("输入要查找的结点关键字\n");

scanf("%d",&u);

i=FirstAdjVex(G,u);

if (i!=-1)

printf("%d %s",G.vertices[i].data.key,G.vertices[i].data.others);

else

printf("查找失败\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 6:

//NextAdjVex

printf("输入结点v和它的邻接点w\n");

scanf("%d%d",&v,&w);

i=NextAdjVex(G,v,w);

if (i!=-1)

printf("%d %s",G.vertices[i].data.key,G.vertices[i].data.others);

else

printf("无下一邻接顶点\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 7:

//InsertVex

printf("输入需要插入的新顶点\n");

scanf("%d%s",&value.key,value.others);

i=InsertVex(G,value);

if (i==OK)

for(i=0;i<G.vexnum;i++)

{

ArcNode \*p=G.vertices[i].firstarc;

printf("%d %s",G.vertices[i].data.key,G.vertices[i].data.others);

while (p)

{

printf(" %d",p->adjvex);

p=p->nextarc;

}

printf("\n");

}

else printf("插入操作失败\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 8:

//DeleteVex

printf("输入需要删除的顶点\n");

scanf("%d",&v);

i=DeleteVex(G,v);

if (i==OK)

{

for(i=0;i<G.vexnum;i++)

{

ArcNode \*p=G.vertices[i].firstarc;

printf("%d %s",G.vertices[i].data.key,G.vertices[i].data.others);

while (p)

{

printf(" %d",p->adjvex);

p=p->nextarc;

}

printf("\n");

}

}

else printf("删除顶点操作失败\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 9:

//InsertArc

printf("输入需要插入的弧\n");

scanf("%d%d",&v,&w);

i=InsertArc(G,v,w);

if (i==OK)

{

for(i=0;i<G.vexnum;i++)

{

ArcNode \*p=G.vertices[i].firstarc;

printf("%d %s",G.vertices[i].data.key,G.vertices[i].data.others);

while (p)

{

printf(" %d",p->adjvex);

p=p->nextarc;

}

printf("\n");

}

}

else printf("插入弧操作失败\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 10:

//DeleteArc

printf("输入需要删除的弧\n");

scanf("%d%d",&v,&w);

i=DeleteArc(G,v,w);

if (i==OK)

{

for(i=0;i<G.vexnum;i++)

{

ArcNode \*p=G.vertices[i].firstarc;

printf("%d %s",G.vertices[i].data.key,G.vertices[i].data.others);

while (p)

{

printf(" %d",p->adjvex);

p=p->nextarc;

}

printf("\n");

}

}

else printf("删除弧操作失败");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 11:

//DFSTraverse

DFSTraverse(G,visit);

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 12:

//BFSTraverse

BFSTraverse(G,visit);

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 13:

//SaveGraph

if (k = SaveGraph(G, x) == OK)

printf("文件保存成功\n");

else printf("文件保存失败\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 14:

//LoadGraph

if (i = LoadGraph(G, x) == OK)

printf("文件输入成功\n");

else printf("文件输入失败\n");

getchar();

printf("\n按回车键继续\n");

getchar();

break;

case 15:

//printGraph

for(i=0;i<G.vexnum;i++)

{

ArcNode \*p=G.vertices[i].firstarc;

printf("%d %s",G.vertices[i].data.key,G.vertices[i].data.others);

while (p)

{

printf(" %d",p->adjvex);

p=p->nextarc;

}

printf("\n");

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

}

}

getchar();

printf("\n按回车键继续\n");

getchar();

break;

}

}

return 0;

}

status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2])

/\*根据V和VR构造图T并返回OK，如果V和VR不正确，返回ERROR

如果有相同的关键字，返回ERROR。此题允许通过增加其它函数辅助实现本关任务\*/

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i=0,k=0,m=0,n=0,b=0;

int a[100];

for(i=0;i<=99;i++) a[i]=0;

for(i=0;V[i].key!=-1;i++)

{

if(i>19) return ERROR;

a[V[i].key]+=1;

if(a[V[i].key]==2) return ERROR;

}

for(k=0;VR[k][1]!=-1;k++)

{

if((a[VR[k][0]]==0)||(a[VR[k][1]]==0)) return ERROR;

}

//判断合法

G.vexnum=0;

G.arcnum=0;

ArcNode \*p;

for(i=0;V[i].key!=-1;i++)

{

G.vertices[i].data=V[i];

G.vertices[i].firstarc=NULL;

G.vexnum++;

}

for(k=0;VR[k][0]!=-1;k++)

{

b++;

for(i=0;V[i].key!=-1;i++)

{

if(V[i].key==VR[k][0])

break;

}

for(m=0;V[m].key!=-1;m++)

{

if(V[m].key==VR[k][1])

break;

}

p=(ArcNode\*)malloc(sizeof(ArcNode));

p->adjvex=m;

p->nextarc=G.vertices[i].firstarc;

G.vertices[i].firstarc=p;

p=(ArcNode\*)malloc(sizeof(ArcNode));

p->adjvex=i;

p->nextarc=G.vertices[m].firstarc;

G.vertices[m].firstarc=p;

}

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status DestroyGraph(ALGraph &G)

/\*销毁无向图G\*/

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i=0;

ArcNode \*p,\*q;

for(i=0;i<G.vexnum;i++)

{

p=G.vertices[i].firstarc->nextarc;

G.vertices[i].firstarc=NULL;

while(p)

{

q=p;

p=p->nextarc;

free(q);

}

}

G.vexnum=0;

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

int LocateVex(ALGraph G,KeyType u)

//根据u在图G中查找顶点，查找成功返回位序，否则返回-1；

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i,k=0;

for(int i=0;i<G.vexnum;i++)

{

if(u==G.vertices[i].data.key)

{

k++;

return i;

}

}

if(k==0)

return -1;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status PutVex(ALGraph &G,KeyType u,VertexType value)

//根据u在图G中查找顶点，查找成功将该顶点值修改成value，返回OK；

//如果查找失败或关键字不唯一，返回ERROR

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i,k=0;

for(i=0;i<G.vexnum;i++)

{

if(value.key==G.vertices[i].data.key)

return ERROR;

}

for(i=0;i<G.vexnum;i++)

{

if(u==G.vertices[i].data.key)

{

k++;

G.vertices[i].data=value;

return OK;

}

}

if(k==0)

return ERROR;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

int FirstAdjVex(ALGraph G,KeyType u)

//根据u在图G中查找顶点，查找成功返回顶点u的第一邻接顶点位序，否则返回-1；

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i,k=0;

for(int i=0;i<G.vexnum;i++)

{

if(u==G.vertices[i].data.key)

{

if(G.vertices[i].firstarc)

{

k++;

return G.vertices[i].firstarc->adjvex;

}

}

}

if(k==0)

return -1;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

int NextAdjVex(ALGraph G,KeyType v,KeyType w)

//根据u在图G中查找顶点，查找成功返回顶点v的邻接顶点相对于w的下一邻接顶点的位序，查找失败返回-1；

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i;

ArcNode \*p;

for(int i=0;i<G.vexnum;i++)

{

if(v==G.vertices[i].data.key)

{

p=G.vertices[i].firstarc;

while(p)

{

if(G.vertices[p->adjvex].data.key==w)

{

p=p->nextarc;

if(p) return p->adjvex;

else return -1;

}

p=p->nextarc;

}

}

}

return -1;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status InsertVex(ALGraph &G,VertexType v)

//在图G中插入顶点v，成功返回OK,否则返回ERROR

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(G.vexnum==20) return ERROR;

int i;

for(i=0;i<G.vexnum;i++)

{

if(v.key==G.vertices[i].data.key)

return ERROR;

}

G.vertices[i].data=v;

G.vexnum++;

G.vertices[i].firstarc=NULL;

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status DeleteVex(ALGraph &G,KeyType v)

//在图G中删除关键字v对应的顶点以及相关的弧，成功返回OK,否则返回ERROR

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i=0,k=0,m=0;

ArcNode \*p,\*q;

if(G.vexnum==1) return ERROR;

for(i=0;i<=G.vexnum-1;i++)

{

if(v==G.vertices[i].data.key)

{

if(G.vertices[i].firstarc!=NULL)

{

p=G.vertices[i].firstarc->nextarc;

G.vertices[i].firstarc=NULL;

while(p)

{

q=p;

p=p->nextarc;

free(q);

}

}

for(m=0;m<=G.vexnum-1;m++)

{

if(m!=i)

{

p=G.vertices[m].firstarc;

if(p)

{

if(G.vertices[p->adjvex].data.key==v)

{

G.vertices[m].firstarc=p->nextarc;

free(p);

}

else

{

while(p->nextarc)

{

if(G.vertices[p->nextarc->adjvex].data.key==v)

{

q=p->nextarc;

p->nextarc=p->nextarc->nextarc;

free(q);

break;

}

p=p->nextarc;

}

}

}

}

}

for(m=0;m<=G.vexnum-1;m++)

{

if(m!=i)

{

p=G.vertices[m].firstarc;

while(p)

{

if(p->adjvex>=i) p->adjvex--;

p=p->nextarc;

}

}

}

for(k=i;k<=G.vexnum-2;k++)

G.vertices[k]=G.vertices[k+1];

G.vexnum--;

G.vertices[k].data.key=-1;

return OK;

}

}

return ERROR;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status InsertArc(ALGraph &G,KeyType v,KeyType w)

//在图G中增加弧<v,w>，成功返回OK,否则返回ERROR

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i,k=0,x=0,y=0;

ArcNode \*p,\*q;

for(x=0;x<G.vexnum;x++)

{

if(v==G.vertices[x].data.key)

{

k++;

break;

}

}

for(y=0;y<G.vexnum;y++)

{

if(w==G.vertices[y].data.key)

{

k++;

break;

}

}

if(k!=2) return ERROR;

p=G.vertices[x].firstarc;

if(p)

{

if(G.vertices[p->adjvex].data.key==w)

{

return ERROR;

}

while(p->nextarc)

{

if(G.vertices[p->nextarc->adjvex].data.key==w)

{

return ERROR;

}

p=p->nextarc;

}

p=(ArcNode\*)malloc(sizeof(ArcNode));

p->adjvex=y;

p->nextarc=G.vertices[x].firstarc;

G.vertices[x].firstarc=p;

p=(ArcNode\*)malloc(sizeof(ArcNode));

p->adjvex=x;

p->nextarc=G.vertices[y].firstarc;

G.vertices[y].firstarc=p;

return OK;

}

else

{

p=(ArcNode\*)malloc(sizeof(ArcNode));

p->adjvex=y;

p->nextarc=G.vertices[x].firstarc;

G.vertices[x].firstarc=p;

p=(ArcNode\*)malloc(sizeof(ArcNode));

p->adjvex=x;

p->nextarc=G.vertices[y].firstarc;

G.vertices[y].firstarc=p;

return OK;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status DeleteArc(ALGraph &G,KeyType v,KeyType w)

//在图G中删除弧<v,w>，成功返回OK,否则返回ERROR

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i,k=0;

ArcNode \*p,\*q;

for(i=0;i<G.vexnum;i++)

{

if(v==G.vertices[i].data.key)

{

p=G.vertices[i].firstarc;

if(G.vertices[p->adjvex].data.key==w)

{

G.vertices[i].firstarc=p->nextarc;

free(p);

break;

}

while(p->nextarc)

{

if(G.vertices[p->nextarc->adjvex].data.key==w)

{

q=p->nextarc;

p->nextarc=p->nextarc->nextarc;

free(q);

break;

}

p=p->nextarc;

}

}

}

for(i=0;i<G.vexnum;i++)

{

if(w==G.vertices[i].data.key)

{

p=G.vertices[i].firstarc;

if(G.vertices[p->adjvex].data.key==v)

{

G.vertices[i].firstarc=p->nextarc;

free(p);

return OK;

}

while(p->nextarc)

{

if(G.vertices[p->nextarc->adjvex].data.key==v)

{

q=p->nextarc;

p->nextarc=p->nextarc->nextarc;

free(q);

return OK;

}

p=p->nextarc;

}

}

}

return ERROR;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status DFSTraverse(ALGraph &G,void (\*visit)(VertexType))

//对图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i;

for(i=0;i<G.vexnum;i++)

{

if(!visited[i]) DFS(G,visit,i);

}

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status BFSTraverse(ALGraph &G,void (\*visit)(VertexType))

//对图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int e;//本函数中无用

int i,j;

ArcNode \*p;

QUEUE Q;

iniQueue(Q);

for(j=0;j<G.vexnum;j++)

{

if(!inqueue[j])

{

enQueue(Q,j);

inqueue[j]=1;

for(i=Q.front;i-Q.front+1<=Q.length;i++)//一个联通子图中的结点未全部入队列

{

for(p=G.vertices[Q.elem[i]].firstarc;p!=NULL;p=p->nextarc)

{

if(!inqueue[p->adjvex])

{

enQueue(Q,p->adjvex);

inqueue[p->adjvex]=1;

}

}

}

while(!emptyQueue(Q))

{

deQueue(Q,e);

(\*visit)(G.vertices[e].data);

}

}

}

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status SaveGraph(ALGraph G, char FileName[])

//将图的数据写入到文件FileName中

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 1 \*\*\*\*\*\*\*\*\*/

FILE\* fout = fopen(FileName, "wt");

if (fout == NULL) exit(-1);

int i;

ArcNode\* temp;

for (i = 0;i < G.vexnum;i++)

{

fprintf(fout, "%d %s ", G.vertices[i].data.key, G.vertices[i].data.others);//先输出节点数据

temp = G.vertices[i].firstarc;

while (temp != NULL)

{

fprintf(fout, "%d ", temp->adjvex);

temp = temp->nextarc;

}

fprintf(fout, "-1 ");//表示一个节点输出完了（包括邻接表）

}

//结尾输入-2表示所有输出结束

fclose(fout);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 1 \*\*\*\*\*\*\*\*\*\*/

}

status LoadGraph(ALGraph& G, char FileName[])

//读入文件FileName的图数据，创建图的邻接表

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 2 \*\*\*\*\*\*\*\*\*/

int vexn = 0, arcn = 0;

ArcNode\* te, \* pre;

FILE\* fin = fopen(FileName, "rt");

if (fin == NULL) exit(-1);

int tkey, t;

char tothers[20];//暂时储存的数据

while (fscanf(fin, "%d%s", &tkey, tothers) != EOF)

{

G.vertices[vexn].data.key = tkey;

strcpy(G.vertices[vexn].data.others, tothers);

fscanf(fin, "%d", &t);

if (t != -1)

{

te = (ArcNode\*)malloc(sizeof(ArcNode));

G.vertices[vexn].firstarc = te;

te->adjvex = t;

te->nextarc = NULL;

fscanf(fin, "%d", &t);

while (t != -1)

{

pre = te;

te = (ArcNode\*)malloc(sizeof(ArcNode));

pre->nextarc = te;

te->adjvex = t;

te->nextarc = NULL;

fscanf(fin, "%d", &t);

}

}

else

{

G.vertices[vexn].firstarc = NULL;

}

vexn++;

}

G.vexnum = vexn;

fclose(fin);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 2 \*\*\*\*\*\*\*\*\*\*/

}

status AddList(LISTS &Lists,char ListName[])

// 只需要在Lists中增加一个名称为ListName的无向图

{

int j=0;

for(j=0;ListName[j]!='\0';j++){

Lists.elem[Lists.length].name[j]=ListName[j];

}

Lists.elem[Lists.length].name[j]='\0';

ALGraph G;

Lists.elem[Lists.length].G=G;

Lists.length+=1;

}

status RemoveList(LISTS &Lists,char ListName[])

// Lists中删除一个名称为ListName的无向图

{

int i;

int k=0;

for(i=0;i<=Lists.length-1;i++){

while(Lists.elem[i].name[k]==ListName[k]){

k++;

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0'))

break;

}

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0'))

break;

}

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0')){

int m;

int n=0;

for(m=i;m<=Lists.length-1;m++)

{

Lists.elem[m].G=Lists.elem[m+1].G;

while(Lists.elem[m].name[n]!='\0'){

Lists.elem[m].name[n]=Lists.elem[m+1].name[n];

n++;

}

}

Lists.length--;

return OK;

}

else return ERROR;

}

int LocateList(LISTS Lists,char ListName[])

// 在Lists中查找一个名称为ListName的无向图，成功返回逻辑序号，否则返回0

{

int i;

int k=0;

for(i=0;i<=Lists.length-1;i++){

while(Lists.elem[i].name[k]==ListName[k]){

k++;

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0'))

break;

}

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0'))

break;

}

if((Lists.elem[i].name[k]=='\0')&&(ListName[k]=='\0')){

return i+1;

}

else return 0;

}