

**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202001**

**学 号： U202015314**

**姓 名： 杨玉林**

**指导教师： 李建军**

**报告日期： 2021年 7月 1 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 2](#_Toc458159882)

[1.3 系统实现 2](#_Toc458159883)

[1.4 实验小结 2](#_Toc458159884)

[**2 基于链式存储结构的线性表实现 2**](#_Toc458159885)

[2.1 问题描述 2](#_Toc458159886)

[2.2 系统设计 2](#_Toc458159887)

[2.3 系统实现 2](#_Toc458159888)

[2.4 实验小结 2](#_Toc458159889)

[**3 基于二叉链表的二叉树实现 2**](#_Toc458159890)

[3.1 问题描述 2](#_Toc458159891)

[3.2 系统设计 2](#_Toc458159892)

[3.3 系统实现 2](#_Toc458159893)

[3.4 实验小结 2](#_Toc458159894)

[**4 基于二叉链表的二叉树实现 2**](#_Toc458159895)

[4.1 问题描述 2](#_Toc458159896)

[4.2 系统设计 2](#_Toc458159897)

[4.3 系统实现 2](#_Toc458159898)

[4.4 实验小结 2](#_Toc458159899)

[**参考文献 2**](#_Toc458159900)

[**附录A 基于顺序存储结构线性表实现的源程序 2**](#_Toc458159901)

[**附录B 基于链式存储结构线性表实现的源程序 2**](#_Toc458159902)

[**附录C 基于二叉链表二叉树实现的源程序 2**](#_Toc458159903)

[**附录D 基于邻接表图实现的源程序 2**](#_Toc458159904)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

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。

通过实验达到⑴加深对线性表的概念、基本运算的理解；⑵熟练掌握线性表的逻辑结构与物理结构的关系；⑶物理结构采用顺序表,熟练掌握线性表的基本运算的实现。

## 1.2 系统设计

主函数接收输入的数据，接收操作数调用对应函数。(如图1-1所示)

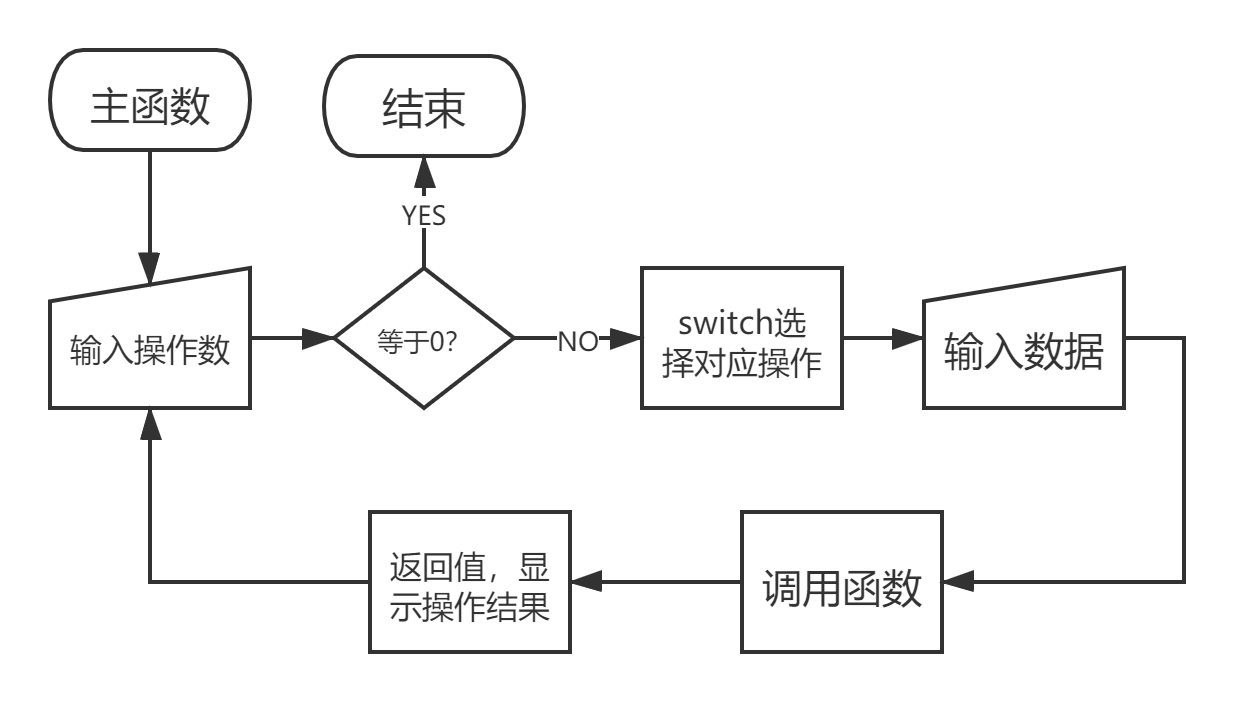


图1-1

数据结构设置：

typedef int ElemType;

typedef int status;

typedef int ElemType; //数据元素类型定义

typedef int status;

typedef int ElemType; //数据元素类型定义

typedef struct

{ //顺序表（顺序结构）的定义

char name[30];

ElemType \*elem;

int length;

int listsize;

} SqList;

typedef struct

{ //线性表的管理表定义

SqList \*elem;

int length;

int listsSize;

} LISTS;

## 1.3 系统实现

主要函数：

1. status InitList(SqList &L);

初始化表：如果线性表L不存在，构造一个空的线性表；否则返回INFEASIBLE。

1. status DestroyList(SqList &L);

销毁表：如果线性表L已存在，销毁线性表L；否则返回INFEASIBLE。

1. status ClearList(SqList &L);

清空表：如果线性表L已存在，操作结果是将L重置为空表，即length置为0；否则返回INFEASIBLE；

1. status ListEmpty(SqList L);

判断空表：如果线性表L已存在，L为空表，返回TURE；否则返回ERROR；否则返回INFEASIBLE。

1. status ListLength(SqList L);

求表长：如果线性表已存在，是返回L中数据元素的个数；否则返回INFEASINBLE。

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

获得元素：线性表L已存在，如果1≤i≤ListLength(L)，用e返回L中第i个数据元素的值，否则返回ERROR；如果线性表L不存在，返回INFEASIBLE。

1. status LocateElem(SqList L, ElemType e);

查找元素：线性表已存在，查找数据元素e不存在，则返回值为0。

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

获得前驱：线性表L已存在，e是L的数据元素，且不是第一个，则用pre返回它的前驱，返回OK；否则操作失败，pre无定义,返回ERROR；如果线性表L不存在，返回INFEASIBLE。（如图1-2所示）

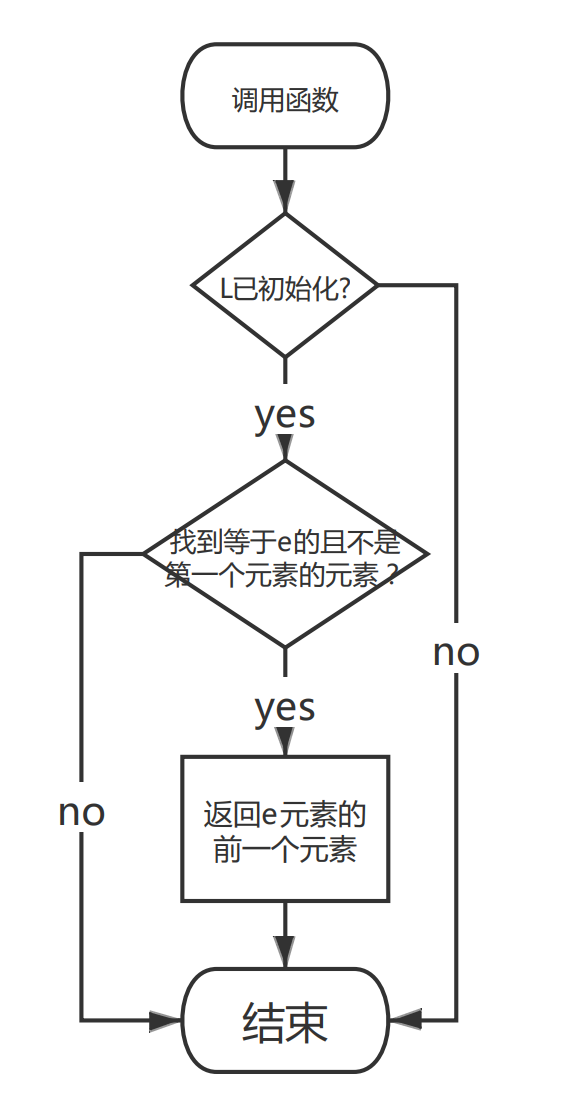


图1-2 PriorElem流程图

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

获得后继：线性表L已存在，若e是L的数据元素，且不是最后一个，则用next返回它的后继，返回OK；否则操作失败，next无定义,返回ERROR；如果线性表L不存在，返回INFEASIBLE。

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

插入元素：线性表L已存在，如果1≤i≤ListLength(L)+1，在L的第i个位置之前插入新的数据元素e，返回OK；否则插入失败，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

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

删除元素：线性表L已存在且非空，如果1≤i≤ListLength(L)，删除L的第i个数据元素，用e返回其值，返回OK；否则删除失败，e无定义，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

1. status ListTraverse(SqList L);

遍历表：线性表L已存在，依次遍历输出L的每个数据元素，返回OK；否则返回INFEASIBLE。

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

存储表：线性表已经存在，将线性表数据存入文件。（如图1-3所示）

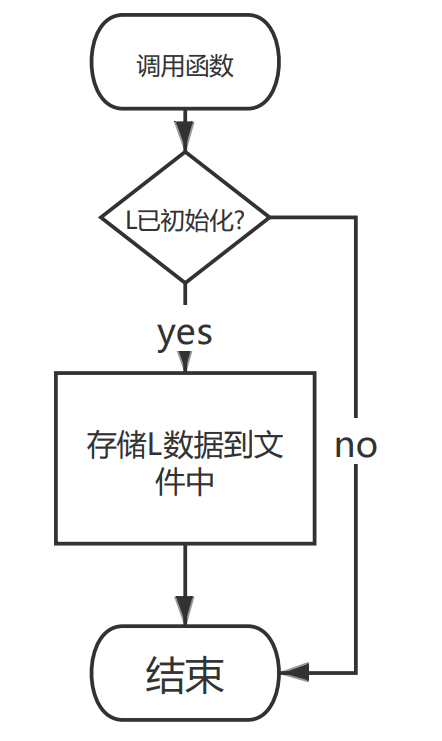


图1-3 SaveList流程图

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

读取表：读取文件，线性表不存在，将读取数据存入线性表。

1. status AddList(LISTS &Lists, SqList L);

加入链表：Lists已初始化，将当前线性表加入Lists中。

1. status InitLists(LISTS &Lists);

初始化Lists：Lists未初始化，将Lists初始化。

1. status ListsTraverse(LISTS Lists);

遍历Lists：Lists已存在，按顺序输出Lists中存储的线性表的每个数据元素。

1. status LocateList(LISTS Lists, char ListName[]);

查找表：Lists已存在，输入线性表名字查找并返回其在Lists中的顺序，不存在返回ERROR。

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

删除表：Lists已存在，输入要删除的线性表的名字在Lists中将其删去，不存在返回ERROR。

* 1. **系统测试**

1. 初始化线性表：初始化未初始化的线性表输出Success，已初始化的输出INFEASIBLE。

表1-1 InitList测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 1 | Success！ |  |
| {{1}{1，2，3}} | 1 | INFEASIBLE！ |  |

1. 销毁线性表：销毁已初始化的线性表返回success，未初始化的返回INFEASIBLE。

表1-2 DestroyList测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 2 | INFEASIBLE！ |  |
| {{1}{1，2，3}} | 2 | Success！ |  |

1. 清空线性表：将已初始化的线性表长度置为0，未初始化的返回INFEASIBLE。

表1-3 ClearList测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 3 | INFEASIBLE！ |  |
| {{1}{1，2，3}} | 3 | Success！ |  |

1. 判断是否是空表：判断已初始化的线性表是否为空表，未初始化返回INFEASIBLE。

表1-4 ListEmpty测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 4 | INFEASIBLE！ |  |
| {{1}{}} | 4 | is Empty! |  |
| {{1}{1，2，3}} | 4 | isn’t Empty! |  |

1. 删除表中元素：删除已存在线性表中的存在的第i个元素，若第i个元素不存在则返回ERROR，线性表未初始化则返回INFEASIBLE。

表1-5 ListDelete测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 11 1 | INFEASIBLE！ |  |
| {{1}{}} | 11 1 | Error! |  |
| {{1}{1，2，3}} | 11 1 | Success! |  |

1. 遍历输出线性表元素：遍历输出已初始化的线性表元素，返回OK；若未初始化则返回INFEASIBLE。

表1-6 ListTraverse测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 12 | INFEASIBLE！ |  |
| {{1}{}} | 12 | 1 L.elem is Empty! |  |
| {{1}{1，2，3}} | 12 | 1 1 2 3 |  |

1. 存储线性表数据到文件中：线性表已存在，将线性表数据写到文件中，返回OK；线性表未初始化则返回INFEASIBLE。

表1-7 SaveList测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 13 | INFEASIBLE！ |  |
| {{1}{}} | 13 | Success! |  |
| {{1}{1，2，3}} | 13 | Success! |  |

1. 读取文件到线性表中：线性表不存在，读取文件数据到线性表中，返回OK；线性表存在则返回INFEASIBLE。

表1-8 LoadList测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 13 12 | Success!  1 1 2 3 |  |
| {{1}{}} | 13 | INFEASIBLE! |  |
| {{1}{1，2，3}} | 13 | INFEASIBLE! |  |

1. 查找元素e线性表中的位置：线性表存在，查找元素e在线性表中的位置并返回位置，未找到返回ERROR，线性表不存在则返回INFEASIBLE。

表1-9 LocateList测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 7 1 | INFEASIBLE! |  |
| {{1}{}} | 7 1 | INFEASIBLE! |  |
| {{1}{1，2，3}} | 7 2 | The elem 2 is 2th! |  |

1. 将当前线性表加入到线性管理链表：线性表且线性管理链表存在，加入到线性管理表中，返回OK；否则返回INFEASIBLE。

表1-10 AddList测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| {{1}{}} | 1 | Success! |  |
| {{1}{1，2，3}} | 1 | Success! |  |
| {{2}{1，2}} | 1 | Success! |  |

## 1.5 实验小结

本次实验主要内容是关于线性表的练习，因为老师给的文件中已给出基础框架，只需对实验中要求的函数进行补充，这减小了我们的学习压力，更能突出对课程内容的考查与训练。

本次实验加深了我对线性表的理解，也使我发现了自身很多的不足之处。首先，对文件读写掌握不足，从而也导致了写读写文件时无从下手。其次，我还发现了我对指针空间管理等理解还不够深刻，导致经常出现bug却不知道哪里出现了问题。例如，InitList时总是报错，找了许久才发现是因为L.elem空间里面不为空,需要在前面将L.elem置为空。因为educoder实验时只用写一个函数，不需要将他们整合到一个里面并写成一个系统，所以当我做实验报告时发现才发现做出一个系统并没有那么简单，只写好单独的函数是没有用的，更重要的是将他们整合在一起，做出自己想要的效果。

本次实验锻炼了我们理论与实践结合的能力，更加深了我们对于数据结构这门课程的学习。

# 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。

通过实验达到⑴加深对线性表的概念、基本运算的理解；⑵熟练掌握线性表的逻辑结构与物理结构的关系；⑶物理结构采用单链表,熟练掌握线性表的基本运算的实现。

## 2.2 系统设计

主函数接收输入的数据，接收操作数调用对应函数。(如图2-1所示)

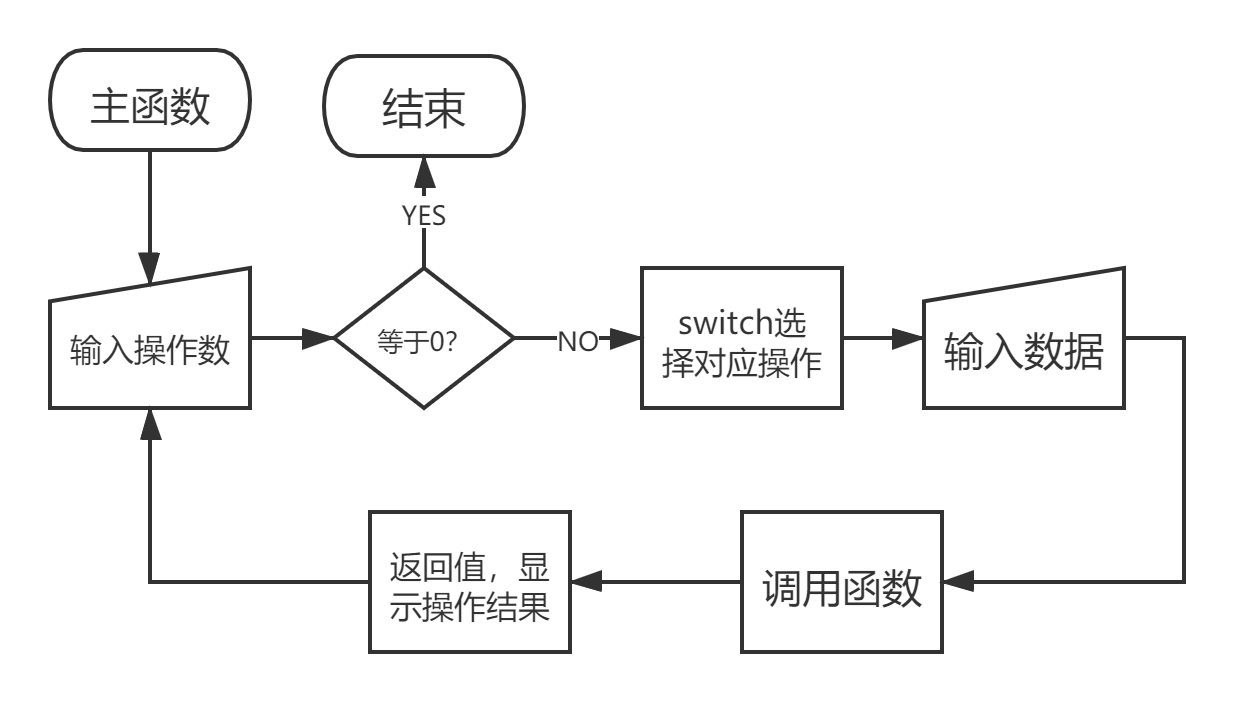


图2-1

数据结构设置：

typedef int status;

typedef int ElemType; //数据元素类型定义

typedef struct LNode

{ //单链表（链式结构）结点的定义

ElemType data;

struct LNode \*next;

} LNode, \*LinkList;

typedef struct List

{

LinkList L;

char name[30];

} List;

typedef struct Lists

{

List \*elem;

ElemType length;

ElemType listsSize;

} LISTS;

## 2.3 系统实现

主要函数：

1. status InitList(LinkList &L);

初始化表：如果线性表L不存在，构造一个空的线性表；否则返回INFEASIBLE。

1. status DestroyList(LinkList &L);

销毁表：如果线性表L已存在，销毁线性表L；否则返回INFEASIBLE。

1. status ClearList(LinkList &L);

清空表：如果线性表L已存在，操作结果是将L重置为空表，即length置为0；否则返回INFEASIBLE；

1. status ListEmpty(LinkList L);

判断空表：如果线性表L已存在，L为空表，返回TURE；否则返回ERROR；否则返回INFEASIBLE。

1. status ListLength(LinkList L);

求表长：如果线性表已存在，是返回L中数据元素的个数；否则返回INFEASINBLE。

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

获得元素：线性表L已存在，如果1≤i≤ListLength(L)，用e返回L中第i个数据元素的值，否则返回ERROR；如果线性表L不存在，返回INFEASIBLE。

1. status LocateElem(LinkList L, ElemType e);

查找元素：线性表已存在，查找数据元素e不存在，则返回值为0。

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

获得前驱：线性表L已存在，e是L的数据元素，且不是第一个，则用pre返回它的前驱，返回OK；否则操作失败，pre无定义,返回ERROR；如果线性表L不存在，返回INFEASIBLE。（如图2-2所示）

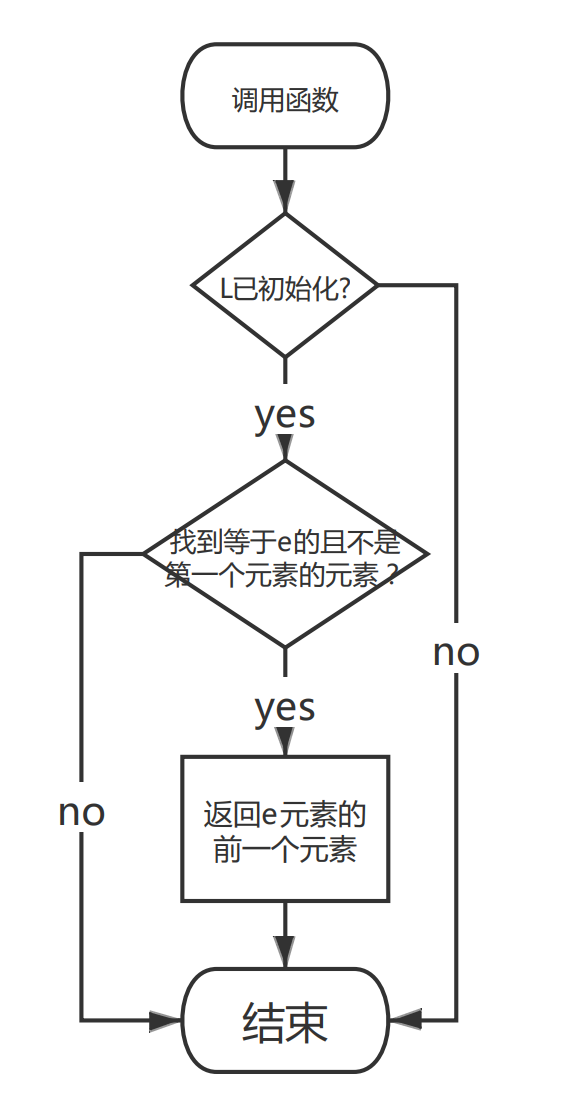


图2-2 PriorElem流程图

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

获得后继：线性表L已存在，若e是L的数据元素，且不是最后一个，则用next返回它的后继，返回OK；否则操作失败，next无定义,返回ERROR；如果线性表L不存在，返回INFEASIBLE。

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

插入元素：线性表L已存在，如果1≤i≤ListLength(L)+1，在L的第i个位置之前插入新的数据元素e，返回OK；否则插入失败，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

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

删除元素：线性表L已存在且非空，如果1≤i≤ListLength(L)，删除L的第i个数据元素，用e返回其值，返回OK；否则删除失败，e无定义，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

1. status ListTraverse(LinkList L);

遍历表：线性表L已存在，依次遍历输出L的每个数据元素，返回OK；否则返回INFEASIBLE。

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

存储表：线性表已经存在，将线性表数据存入文件。（如图1-3所示）

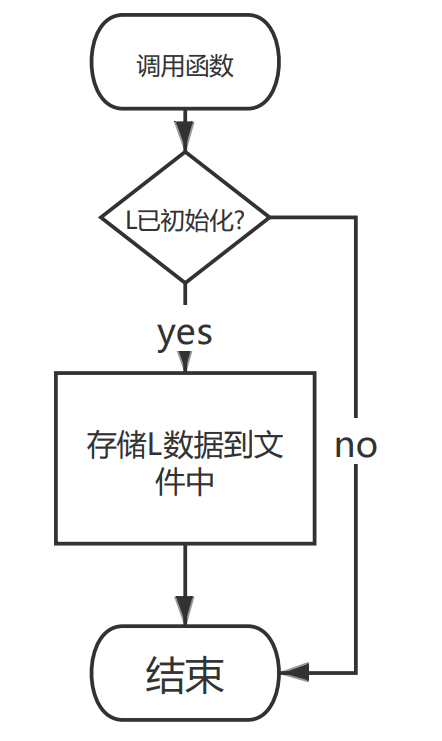


图1-3 SaveList流程图

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

读取表：读取文件，线性表不存在，将读取数据存入线性表。

1. status AddList(LISTS &Lists, LinkList L);

加入链表：Lists已初始化，将当前线性表加入Lists中。

1. status InitLists(LISTS &Lists);

初始化Lists：Lists未初始化，将Lists初始化。

1. status ListsTraverse(LISTS Lists);

遍历Lists：Lists已存在，按顺序输出Lists中存储的线性表的每个数据元素。

1. status LocateList(LISTS Lists, char ListName[]);

查找表：Lists已存在，输入线性表名字查找并返回其在Lists中的顺序，不存在返回ERROR。

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

删除表：Lists已存在，输入要删除的线性表的名字在Lists中将其删去，不存在返回ERROR。

## 2.4 系统测试

1. 初始化线性表：初始化未初始化的线性表输出Success，已初始化的输出INFEASIBLE。

表2-1 InitList测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 1 | Success！ |  |
| {1，2，3} | 1 | INFEASIBLE！ |  |

1. 销毁线性表：销毁已初始化的线性表返回success，未初始化的返回INFEASIBLE。

表2-2 DestroyList测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 2 | INFEASIBLE！ |  |
| {1，2，3} | 2 | Success！ |  |

1. 清空线性表：将头节点以外的空间全部释放，未初始化的返回INFEASIBLE。

表2-3 ClearList测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 3 | INFEASIBLE！ |  |
| {1，2，3} | 3 | Success！ |  |

1. 判断是否是空表：判断已初始化的线性表是否为空表，未初始化返回INFEASIBLE。

表2-4 ListEmpty测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 4 | INFEASIBLE！ |  |
| {} | 4 | is Empty! |  |
| {1，2，3} | 4 | isn’t Empty! |  |

1. 删除表中元素：删除已存在线性表中的存在的第i个元素，若第i个元素不存在则返回ERROR，线性表未初始化则返回INFEASIBLE。

表2-5 ListDelete测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 11 1 | INFEASIBLE！ |  |
| {} | 11 1 | Error! |  |
| {1，2，3} | 11 1 | Success! |  |

1. 遍历输出线性表元素：遍历输出已初始化的线性表元素，返回OK；若未初始化则返回INFEASIBLE。

表2-6 ListTraverse测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 12 | INFEASIBLE！ |  |
| {} | 12 | 1 L.elem is Empty! |  |
| {1，2，3} | 12 | 1 1 2 3 |  |

1. 存储线性表数据到文件中：线性表已存在，将线性表数据写到文件中，返回OK；线性表未初始化则返回INFEASIBLE。

表2-7 SaveList测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 13 | INFEASIBLE！ |  |
| {} | 13 | Success! |  |
| {1，2，3} | 13 | Success! |  |

1. 读取文件到线性表中：线性表不存在，读取文件数据到线性表中，返回OK；线性表存在则返回INFEASIBLE。

表2-8 LoadList测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 13 12 | Success!  1 1 2 3 |  |
| {} | 13 | INFEASIBLE! |  |
| {1，2，3} | 13 | INFEASIBLE! |  |

1. 查找元素e线性表中的位置：线性表存在，查找元素e在线性表中的位置并返回位置，未找到返回ERROR，线性表不存在则返回INFEASIBLE。

表2-9 LocateList测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 7 1 | INFEASIBLE! |  |
| {} | 7 1 | INFEASIBLE! |  |
| {1，2，3} | 7 2 | The elem 2 is 2th! |  |

1. 将当前线性表加入到线性管理链表：线性表且线性管理链表存在，加入到线性管理表中，返回OK；否则返回INFEASIBLE。

表2-10 AddList测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| {{q}{}} | 1 | Success! |  |
| {{1}{1，2，3}} | 1 | Success! |  |
| {{w}{1，2}} | 1 | Success! |  |

## 2.5 实验小结

本次实验主要内容是关于链式线性表的练习，因为上次和这次的基础框架一样，所以不需要修改许多，基本上只需要修改函数就可以了。

本次实验加深了我对线性表和链表的理解，也使我发现了自身很多的不足之处。首先，我发现了我对链表的掌握还不够熟练，导致在写链表初始化等操作时出错，最后看了之前学习链表时的书才正确写出来。其次，还是对指针掌握不够，在写程序时经常因为这个空间问题出现各种错误或者是程序直接停止运行。

本次实验锻炼了我们理论与实践结合的能力，更加深了我们对于数据结构这门课程的学习。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

通过实验达到⑴加深对二叉树的概念、基本运算的理解；⑵熟练掌握二叉树的逻辑结构与物理结构的关系；⑶以二叉链表作为物理结构，熟练掌握二叉树基本运算的实现。

## 3.2 系统设计

主函数接收输入的数据，接收操作数调用对应函数。(如图3-1所示)

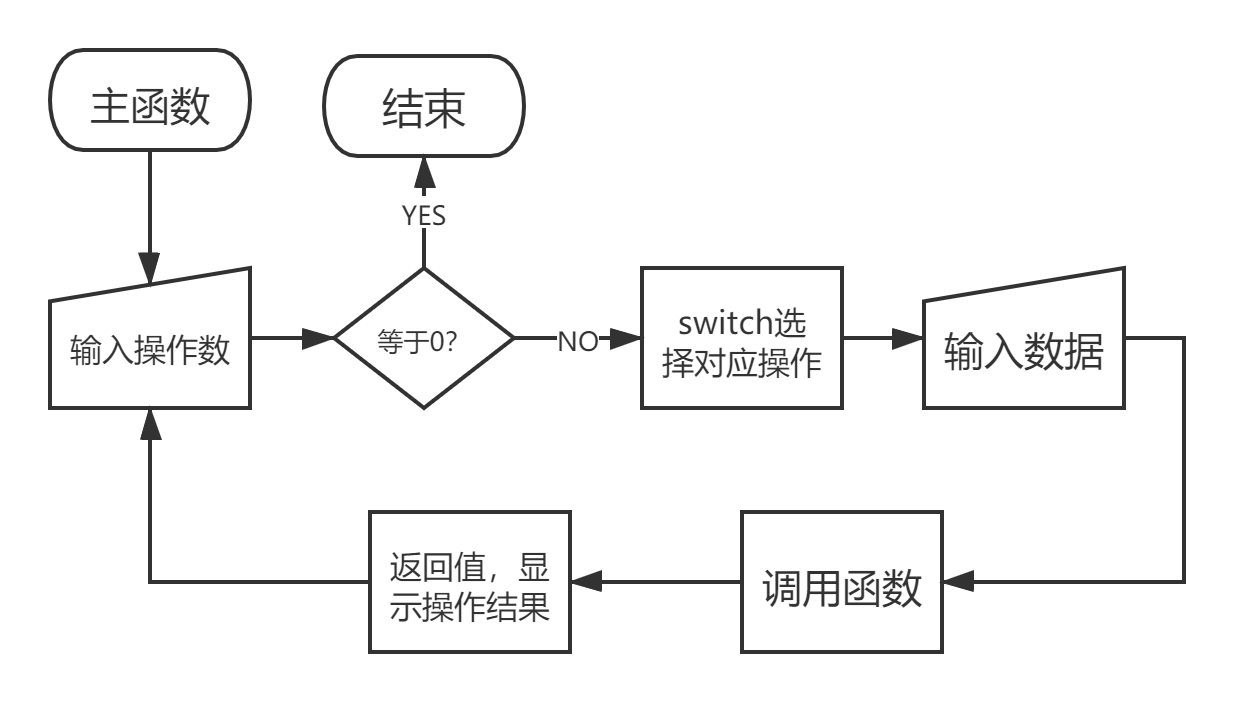


图3-1

数据结构设置：

typedef int status;

typedef int KeyType;

typedef struct

{

KeyType key;

char others[100];

} TElemType; //二叉树结点类型定义

typedef struct BiTNode

{ //二叉链表结点的定义

TElemType data;

struct BiTNode \*lchild, \*rchild;

} BiTNode, \*BiTree;

typedef struct

{

BiTree T;

char name[30];

} List;

## 3.3 系统实现

主要函数：

* + 1. status CreateBiTree(BiTree &T, TElemType definition[]);

创建二叉树：definition 给出二叉树T的定义，各节点关键值具有唯一性，否则返回ERROR，按definition构造二叉树T。成功返回OK，失败返回ERROR。

1. status DestroyBiTree(BiTree &T);

销毁二叉树：二叉树T已存在；销毁二叉树T，返回OK。

1. status ClearBiTree(BiTree &T);

清空二叉树：二叉树T存在；将二叉树T清空。

1. status BiTreeEmpty(BiTree T);

判定空二叉树：二叉树T存在；若T为空二叉树则返回TRUE，否则返回FALSE。

1. int BiTreeDepth(BiTree T);

求二叉树深度：二叉树T存在；返回T的深度。

1. BiTNode \*LocateNode(BiTree T, KeyType e);

查找结点：二叉树T已存在，e是和T中结点关键字类型相同的给定值；返回查找到的结点指针，如无关键字为e的结点，返回NULL。

1. BiTNode \*LocateFatherNode(BiTree T, KeyType e);

查找父亲结点：二叉树T已存在，e是和T中结点关键字类型相同的给定值；返回查找到的结点指针的父亲结点，如无关键字为e的结点，返回NULL。（如图3-2所示）

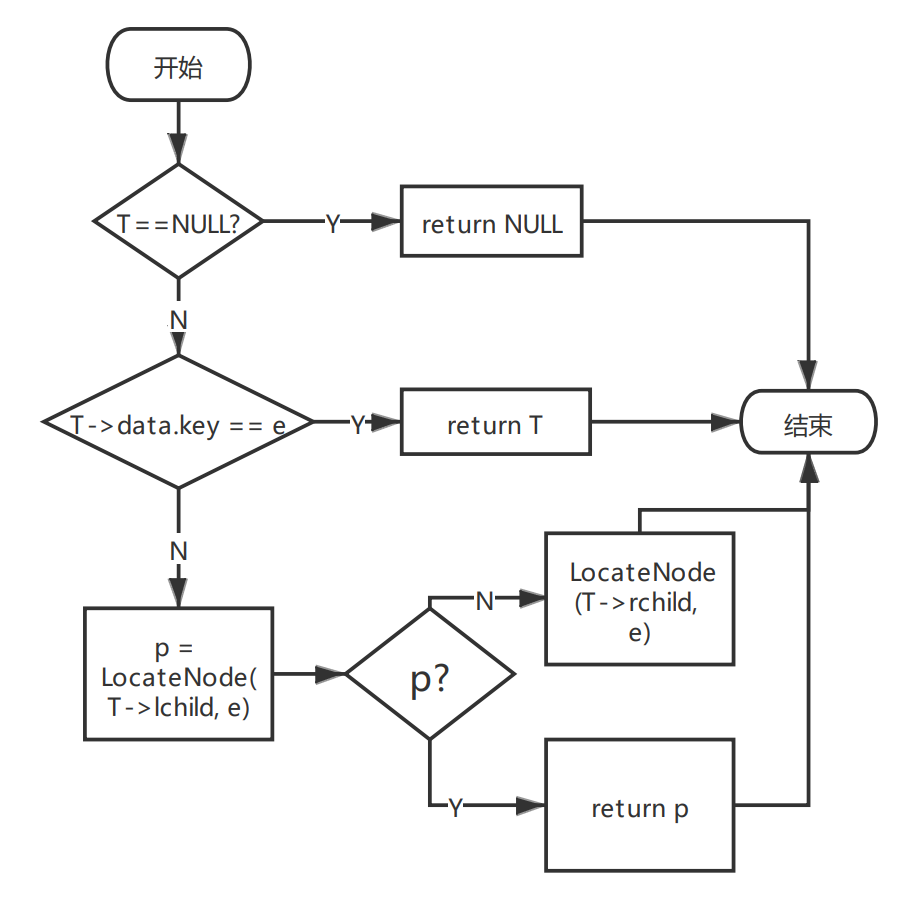


图3-2 LocateFatherNode流程图

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

结点赋值：二叉树T已存在，e是和T中结点关键字类型相同的给定值，将关键字为e的结点赋值为value。成功返回OK，失败返回ERROR。

1. BiTNode \*GetSibling(BiTree T, KeyType e);

获得兄弟结点：二叉树T存在，e是和T中结点关键字类型相同的给定值；返回关键字为e的结点的（左或右）兄弟结点指针。若关键字为e的结点无兄弟，则返回NULL。（如图3-3所示）

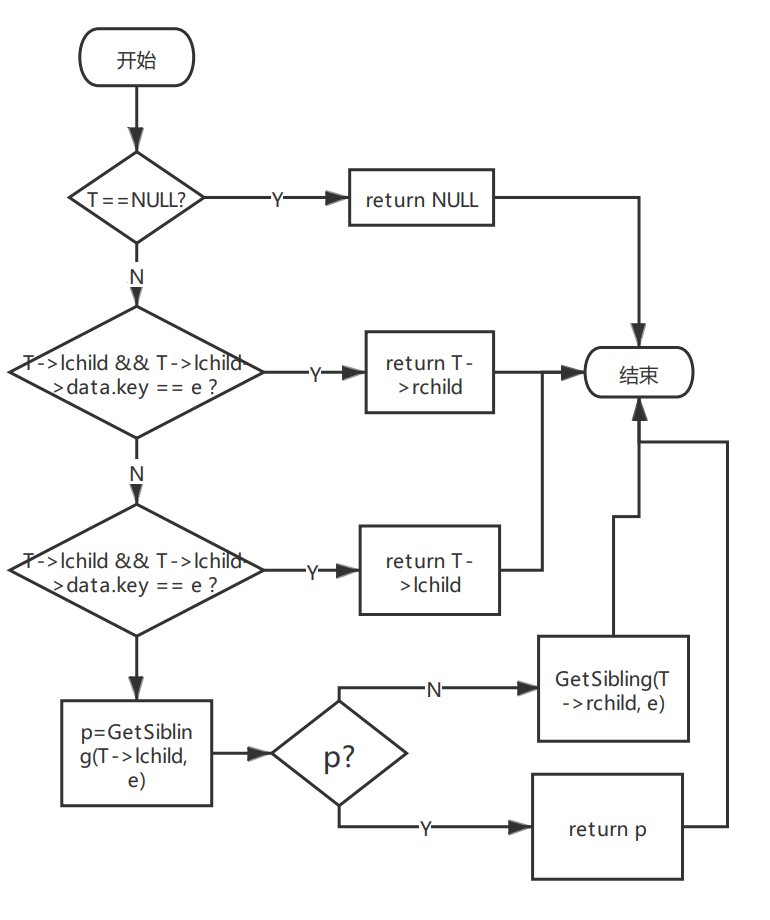


图3-3 GetSibling流程图

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

插入结点：二叉树T存在，e是和T中结点关键字类型相同的给定值，LR为0或1，c是待插入结点；根据LR为0或者1，插入结点c到T中，作为关键字为e的结点的左或右孩子结点，结点e的原有左子树或右子树则为结点c的右子树。特殊情况，c插入作为根结点， LR为-1时，作为根结点插入，原根结点作为c的右子树。

1. status DeleteNode(BiTree &T, KeyType e);

删除结点：二叉树T存在，e是和T中结点关键字类型相同的给定值。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。

1. status PreOrderTraverse(BiTree T, void (\*visit)(BiTree));

前序遍历：二叉树T存在，先序遍历T，对每个结点调用函数Visit一次且一次。

1. status InOrderTraverse(BiTree T, void (\*visit)(BiTree));

中序遍历：二叉树T存在，中序遍历T，对每个结点调用函数Visit一次且一次。

1. status PostOrderTraverse(BiTree T, void (\*visit)(BiTree));

后序遍历：二叉树T存在，后序遍历T，对每个结点调用函数Visit一次且一次。

1. status LevelOrderTraverse(BiTree T, void (\*visit)(BiTree));

按层遍历：二叉树T存在，层序遍历T，对每个结点调用函数Visit一次且一次。

1. status SaveBiTree(BiTree T, char FileName[]);

存储树：初始条件是二叉树T存在，操作结果前序遍历T，将结点数据存入文件中，成功返回OK，失败返回ERROR。

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

读取树：二叉树T不存在，读取文件数据，前序创建链表，成功返回OK，失败返回ERROR。

## 3.4 系统测试

创建的树为：1 a 2 b 0 0 0 0 3 c 0 0 0 0 -1 -1

1. 创建二叉树：构造二叉树T，成功输出Success，失败输出ERROR。

表3-1 CreateBiTree测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 1 | Success！ |  |
| {{1,a},{2,b},{3,c}} | 1 | Success！ |  |

1. 销毁二叉树：销毁二叉树T，成功输出Success，失败输出INFEASIBLE。

表3-2 DestroyBiTree测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 2 | INFEASIBLE! |  |
| {{1,a},{2,b},{3,c}} | 2 | Success！ |  |

1. 树的深度：返回树的深度并输出。

表3-3 BiTreeDepth测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 5 | 0 |  |
| {{1,a},{2,b},{3,c}} | 5 | 2 |  |

1. 查找结点：查找到关键值为e的结点，成功输出该节点，失败输出ERROR。

表3-4 Assign测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 6 1 | ERROR！ |  |
| {{1,a},{2,b},{3,c}} | 6 4 | ERROR！ |  |
| {{1,a},{2,b},{3,c}} | 6 1 | Success! |  |

1. 结点赋值：给结点关键字为e的结点重新赋值。成功输出Success，失败输出ERROER。

表3-5 Assign测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 7 1 1 w | ERROR！ |  |
| {{1,a},{2,b},{3,c}} | 7 1 2 w | ERROR！ |  |
| {{1,a},{2,b},{3,c}} | 7 1 4 w | Success! |  |
| {{1,a},{2,b},{3,c}} | 7 1 1 w | Success! |  |

1. 插入结点：成功输出Success，失败输出ERROER。

表3-6 InsertNode测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 9 1 0 4 w | ERROR！ |  |
| {{1,a},{2,b},{3,c}} | 9 1 1 2 w | ERROR！ |  |
| {{1,a},{2,b},{3,c}} | 9 4 -1 4 w | ERROR！ |  |
| {{1,a},{2,b},{3,c}} | 9 1 -1 9 w | Success! |  |

1. 删除结点：成功输出Success，失败输出ERROER。

表3-7 DeleteNode测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 10 1 | ERROR！ |  |
| {{1,a},{2,b},{3,c}} | 10 4 | ERROR！ |  |
| {{1,a},{2,b},{3,c}} | 10 1 | Success! |  |

1. 输出树：输出树（包括前序中序后序按层输出）。

表3-8 输出树测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 11 12 13 14 |  |  |
| {{1,a},{2,b},{3,c}} | 11 12 13 14 | 1,a 2,b 3,c  2,b 1,a 3,c  2,b 3,c 1,a  1,a 2,b 3,c |  |

1. 存储树：树非空情况下存储树，成功输出Success，失败输出ERROR。

表3-9 SaveBiTree测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 15 | ERROR！ |  |
| {{1,a},{2,b},{3,c}} | 15 | Success！ |  |

1. 读取数据生成树：树为空情况下读取数据生成树，成功输出Success，失败输出ERROR。

表3-10 LoadBiTree测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 16 | Success！ |  |
| {{1,a},{2,b},{3,c}} | 16 | ERROR！ |  |

3.5 实验小结

本次实验主要内容是关于树的练习， 因为树是一个比较陌生的新的数据结构，所以相对而言比较复杂一点。

本次实验让我学会了树的基本操作，如何创建树，插入新的结点等，同时也发现自身的不足。树的许多操作都需要利用递归操作，而递归本就比较难理解，而且还需要判断好什么时候结束递归或者继续下一个递归。这耗费了我不少时间，但当我解决了问题时，我对递归的理解更深刻了。树的遍历中需要用非递归实现，这比用递归实现更困难，需要利用栈的后进先出实现，这加深了我对栈的理解。

本次实验锻炼了我们理论与实践结合的能力，更加深了我们对于数据结构这门课程的学习。

# 4 基于邻接表的图实现

## 4.1 问题描述

通过实验达到⑴加深对图的概念、基本运算的理解；⑵熟练掌握图的逻辑结构与物理结构的关系；⑶以邻接表作为物理结构，熟练掌握图基本运算的实现。

## 4.2 系统设计

主函数接收输入的数据，接收操作数调用对应函数。(如图4-1所示)

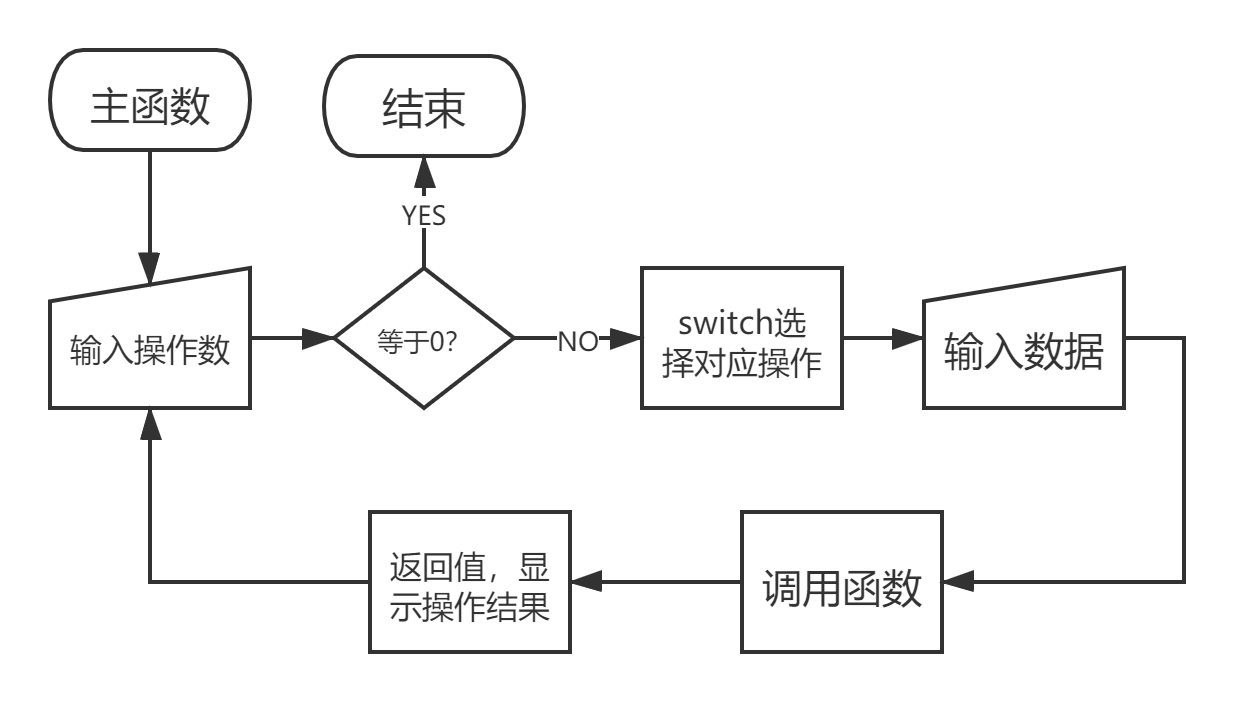


图4-1

数据结构设置：

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

{

int elem[MAX\_VERTEX\_NUM];

int front, tail;

} QUEUE;

typedef struct

{

ALGraph G;

char name[30];

} LIST;

typedef struct

{

LIST elem[20];

int length;

} LISTS;

## 4.3 系统实现

主要函数：

1. status CreateCraph(ALGraph &G, VertexType V[], KeyType VR[][2]);

创建图： V是图的顶点集，VR是图的关系集；按V和VR的定义构造图G。成功返回OK，失败返回ERROR。

1. status DestroyGraph(ALGraph &G);

销毁图：图G已存在，销毁图G。成功返回OK，失败返回ERROR。

1. int LocateVex(ALGraph G, KeyType u);

查找顶点：图G存在，u是和G中顶点关键字类型相同的给定值；若u在图G中存在，返回关键字为u的顶点位置序号（简称位序），否则返回-1。

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

顶点赋值：图G存在，u是和G中顶点关键字类型相同的给定值；对关键字为u的顶点赋值value。成功返回OK，失败返回ERROR。

1. int FirstAdjVex(ALGraph G, KeyType u);

获得第一邻接点：图G存在，u是G中顶点的位序；操作结果是返回u对应顶点的第一个邻接顶点位序，如果u的顶点没有邻接顶点，否则返回-1。

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

获得下一邻接点：图G存在，v和w是G中两个顶点的位序，v对应G的一个顶点,w对应v的邻接顶点；返回v的（相对于w）下一个邻接顶点的位序，如果w是最后一个邻接顶点，返回-1。

1. status InsertVex(ALGraph &G, VertexType v);

插入顶点：图G存在，v和G中的顶点具有相同特征；在图G中增加新顶点v。（顶点关键字唯一）成功返回OK，失败返回ERROR。

1. status DeleteVex(ALGraph &G, KeyType v);

删除顶点：图G存在，v是和G中顶点关键字类型相同的给定值；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，失败返回ERROR。

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

插入弧：图G存在，v、w是和G中顶点关键字类型相同的给定值；在图G中增加弧<v,w>。成功返回OK，失败返回ERROR。（如图4-2所示，下页）

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

删除弧：图G存在，v、w是和G中顶点关键字类型相同的给定值；操作结果是在图G中删除弧<v,w>。成功返回OK，失败返回ERROR。

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

深度优先搜索遍历：图G存在；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。

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

广深度优先搜索遍历：图G存在；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。

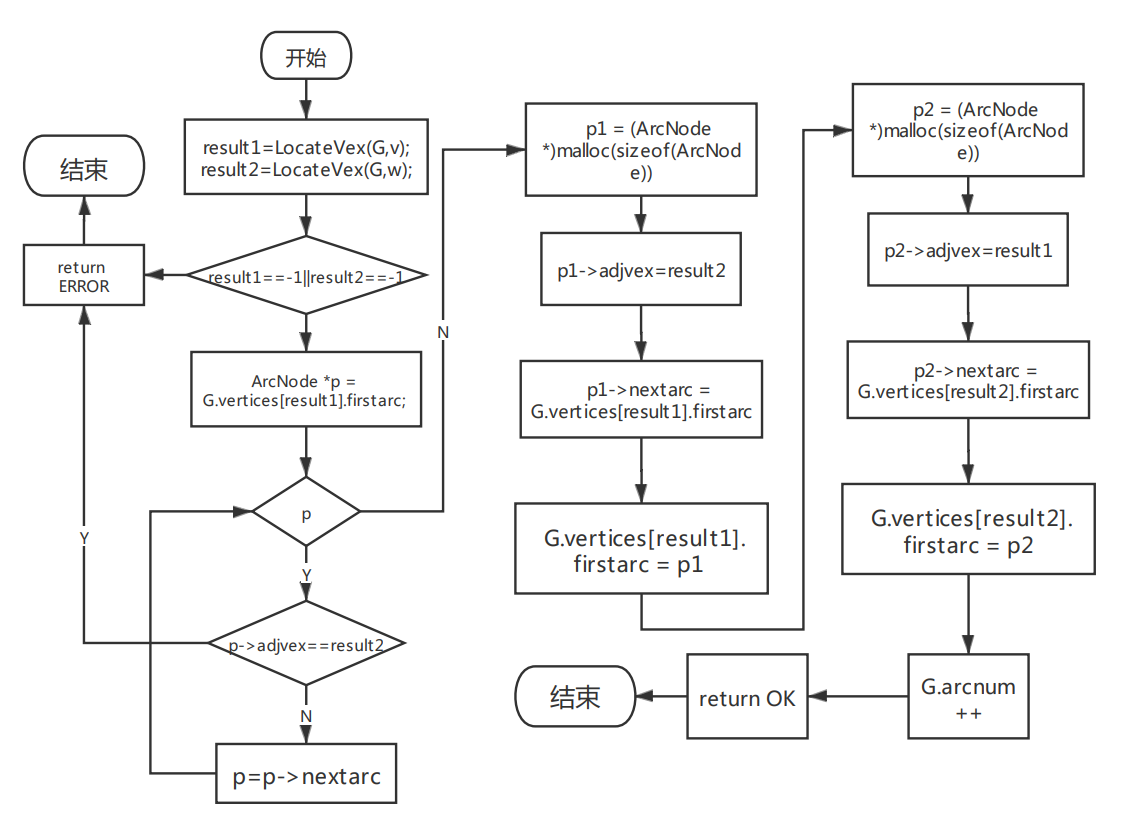


图4-2 InsertArc流程图

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

广深度优先搜索遍历：图G存在；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。

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

存储图：图G存在，存储图中的信息。成功返回OK，失败返回ERROR。

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

读取图：图G不存在，读取存储在文件中的信息创建图。成功返回OK，失败返回ERROR。（如图4-3所示，下页）

1. status ALGraphTraverse(ALGraph G);

输出图：图G存在，输出图和后序结点的位序，返回OK。

辅助函数：

1. status Find(VertexType V[], KeyType VR[][2]);

检查生成图的数据：检查生成图的数据是否合法，合法返回OK，否则返回ERROR。

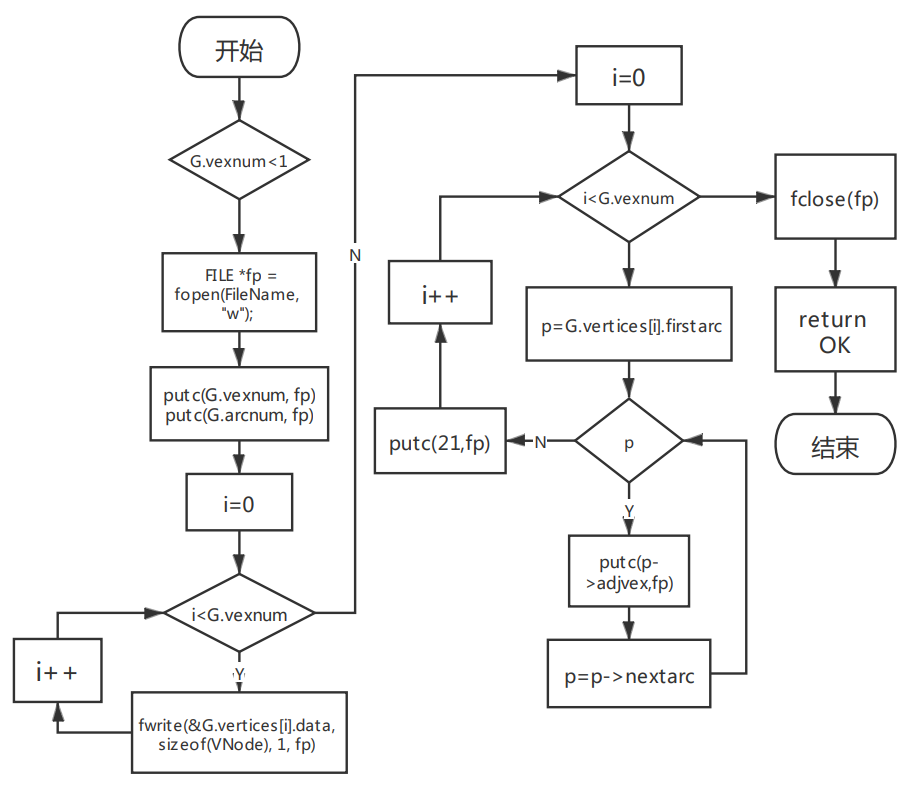


图4-3 SaveGraph流程图

## 4.4 系统测试

1. 输入的数据为5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1
2. 输入的数据为5 线性表 8 集合 8二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1
3. 输入的数据为1 G1 2 G2 3 G3 4 G4 5 G5 6 G6 7 G7 8 G8 9 G9 10 G10 11 G11 12 G12 13 G13 14 G14 15 G15 16 G16 17 G17 18 G18 19 G19 20 G20 21 G21 -1 nil -1 -1
4. 创建图：构造图G，成功输出Success，失败输出ERROR。

表4-1 CreateCraph测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 1 a | Success！ |  |
| NULL | 1 b | ERROR! |  |
| NULL | 1 c | ERROR! |  |

1. 销毁图：销毁图，成功输出Success，失败输出INFEASIBLE。

表4-2 DestoryGraph测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| a | 2 | Success！ |  |
| NULL | 2 | INFEASIBLE! |  |

1. 查找顶点：成功输出位序，失败输出ERROR。

表4-3 LocateVex测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| a | 3 7 | The Vex 二叉树 is the 2 vex |  |
| a | 3 1 | ERROR! |  |
| NULL | 3 7 | ERROR! |  |

1. 顶点赋值：给顶点赋值，成功输出Success，失败输出ERROR。

表4-4 PutVex测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| a | 4 7 1 w | Success！ |  |
| a | 4 7 8 w | ERROR! |  |
| a | 4 1 1 w | ERROR! |  |
| NULL | 4 7 1 w | ERROR! |  |

1. 获得第一邻接点：成功输出第一邻接点位序，失败输出ERROR。

表4-6 NextAdjVex测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| a | 5 7 | The FirstAdjVex is 1 |  |
| a | 5 9 | ERROR! |  |
| NULL | 5 7 | ERROR! |  |

1. 获得下一邻接点：成功输出下一邻接点位序，失败输出ERROR。

表4-6 NextAdjVex测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| a | 6 5 7 | The NextAdjVex is 3 |  |
| a | 6 7 5 | ERROR! |  |
| a | 6 1 7 | ERROR! |  |
| NULL | 6 7 7 | ERROR! |  |

1. 插入顶点：成功输出Success，失败输出ERROR。

表4-7 InsertVex测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| a | 7 1 w | Success！ |  |
| a | 7 7 w | ERROR! |  |
| a | 7 8 w | ERROR! |  |
| NULL | 7 2 w | ERROR! |  |

1. 删除顶点：成功输出Success，失败输出ERROR。

表4-8 DeleteVex测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| a | 8 5 | Success！ |  |
| a | 8 1 | ERROR! |  |
| NULL | 8 7 | ERROR! |  |

1. 插入弧：成功输出Success，失败输出ERROR。

表4-9 InsertArc测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| a | 9 5 8 | Success！ |  |
| a | 9 5 7 | ERROR! |  |
| NULL | 9 7 10 | ERROR! |  |

1. 删除弧：成功输出Success，失败输出ERROR。

表4-10 DeleteArc测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| a | 10 5 7 | Success！ |  |
| a | 10 5 10 | ERROR! |  |
| NULL | 10 7 10 | ERROR! |  |

1. 深度优先搜索遍历：输出树。

表4-11 DFSTraverse测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| a | 11 | 5 线性表 8 集合 7 二叉树 6 无向图 |  |
| NULL | 11 |  |  |

1. 深度优先搜索遍历：输出树。

表4-12 BFSTraverse测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| a | 12 | 5 线性表 8 集合 7 二叉树 6 无向图 |  |
| NULL | 12 |  |  |

1. 存储树：成功输出Success，失败输出ERROR。

表4-13 SaveGraph测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| a | 13 | Success！ |  |
| NULL | 13 | ERROR！ |  |

1. 读取树：成功输出Success，失败输出ERROR。

表4-14 LoadGraph测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| NULL | 13 | Success！ |  |
| a | 13 | ERROR！ |  |

1. 输出树：输出树。

表4-15 ALGraphTraverse测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| a | 14 | Success！ |  |
| a | 13 | ERROR！ |  |

## 4.5 实验小结

本次实验主要内容是关于图的练习， 因为图是一个陌生的新的数据结构，所以相对而言复杂一点。

本次实验让我学会了图的基本操作，如何创建图，插入新的结点等，同时也发现自身的不足。图的许多操作都需要对指针进行操作，而指针需要小心操作。在本次实验中，我写出来一个严重错误，导致我的程序死循环了，并且还在不停的分配空间，随后我的电脑死机并重启了。 本次实验中图的操作多且复杂，需要检查结点存不存在和结点相连的边等等，不然很容易出现错误。

本次实验锻炼了我们理论与实践结合的能力，更加深了我们对于数据结构这门课程的学习。

# 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 <stdlib.h>

#include <string.h>

#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

#define LISTS\_INIT\_SIZE 10

#define LISTSINCREMENT 10

typedef int ElemType;

typedef int status;

typedef int ElemType; //数据元素类型定义

typedef int status;

typedef int ElemType; //数据元素类型定义

typedef struct

{ //顺序表（顺序结构）的定义

char name[30];

ElemType \*elem;

int length;

int listsize;

} SqList;

typedef struct

{ //线性表的管理表定义

SqList \*elem;

int length;

int listsSize;

} LISTS;

status InitList(SqList &L);

status DestroyList(SqList &L);

status ClearList(SqList &L);

status CreateNewList(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, SqList L);

status InitLists(LISTS &Lists);

status SaveLists(LISTS &Lists, char FileName[]);

status LoadLists(LISTS &Lists, char FileName[]);

status ListsTraverse(LISTS Lists);

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

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

status ReLists(LISTS &Lists);

int main()

{

LISTS Lists;

Lists.length = 0;

Lists.listsSize = 10;

Lists.elem = (SqList \*)malloc(sizeof(SqList) \* LISTS\_INIT\_SIZE);

char filename1[30] = "LinkLists.txt";

char filename2[30] = "Lists.txt";

char Listname[30];

ElemType i, e, pre, next, result;

int op = 1, now = 0;

while (op)

{

printf("The op of Lists\n");

printf("1.AddList\n");

printf("2.ListsTraverse\n");

printf("3.LocateList\n");

printf("4.RemoveList\n");

printf("5.choose List\n");

printf("0.exit\n");

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

printf("choose [0-5]:");

scanf("%d", &op);

while (op)

{

switch (op)

{

case 1:

if (Lists.length == Lists.listsSize)

{

Lists.listsSize += LISTSINCREMENT;

Lists.elem = (SqList \*)realloc(Lists.elem, sizeof(SqList) \* Lists.listsSize);

}

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

if (InitList(Lists.elem[Lists.length]) != OK)

printf("ERROR!");

int temp;

printf("input the elem of list:");

scanf("%d", &temp);

while (temp)

{

ListInsert(Lists.elem[Lists.length], Lists.elem[Lists.length].length + 1, temp);

scanf("%d", &temp);

}

now++;

Lists.length++;

break;

case 2:

ListsTraverse(Lists);

break;

case 3:

printf("input the name of List:");

scanf("%s", Listname);

result = LocateList(Lists, Listname);

if (result)

printf("the List %s is the %dth List\n", Listname, result);

else

printf("ERROR!\n");

break;

case 4:

printf("input the name of List:");

scanf("%s", Listname);

result = RemoveList(Lists, Listname);

if (result)

printf("Success!\n");

else

printf("ERROR!\n");

break;

case 5:

printf("input the name of List:");

scanf("%s", Listname);

result = LocateList(Lists, Listname);

if (result == 0)

{

printf("Don't exist!\n");

}

else

{

system("cls");

now = result - 1;

printf(" The List you choose is %s\n", Lists.elem[now].name);

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

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

printf(" The op of List\n");

printf(" 1. InitList 2. DestroyList\n");

printf(" 3. ClearList 4. ListEmpty\n");

printf(" 5. ListLength 6. GetElem \n");

printf(" 7. LocateElem 8. PriorElem\n");

printf(" 9. NextElem 10. ListInsert\n");

printf(" 11. ListDelete 12. ListTrabverse\n");

printf(" 13. SaveList 14.LoadList\n");

printf(" 0.exit \n");

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

printf("choose [0-14]:");

scanf("%d", &op);

while (op >= 0)

{

switch (op)

{

case 1:

//status InitList(LinkList &L)

if (InitList(Lists.elem[now]) == OK)

printf("Success!\n");

else

printf("INFEASIBLE!\n");

getchar();

break;

case 2:

//status DestroyList(LinkList &L)

if (DestroyList(Lists.elem[now]) == OK && Lists.elem[now].elem == NULL)

printf("Success!\n");

else

printf("INFEASIBLE!\n");

getchar();

break;

case 3:

//status ClearList(LinkList &L)

if (ClearList(Lists.elem[now]) == OK)

printf("Success!\n");

else

printf("INFEASIBLE!\n");

getchar();

break;

case 4:

//status ListEmpty(LinkList L)

result = ListEmpty(Lists.elem[now]);

if (result == TRUE)

printf("is Empty!\n");

else if (result == FALSE)

printf("isn't Empty!\n");

else

printf("INFEASIBLE!\n");

getchar();

break;

case 5:

//status ListLength(LinkList L)

result = ListLength(Lists.elem[now]);

if (result != INFEASIBLE)

printf("The length is %d\n", result);

else

printf("INFEASIBLE!\n");

getchar();

break;

case 6:

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

printf("please input the number i:\n");

scanf("%d", &i);

result = GetElem(Lists.elem[now], i, e);

if (result == INFEASIBLE)

printf("INFEASIBLE!\n");

else if (result == ERROR)

printf("Error!\n");

else

printf("The %dth number is %d\n", i, e);

getchar();

break;

case 7:

//status LocateElem(LinkList L, ElemType e)

printf("please input the number e:\n");

scanf("%d", &e);

result = LocateElem(Lists.elem[now], e);

if (result == INFEASIBLE)

printf("INFEASIBLE\n");

else if (result == ERROR)

printf("Error!\n");

else

printf("the elem %d is %dth!\n", e, result);

getchar();

break;

case 8:

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

printf("please input the number e:\n");

scanf("%d", &e);

result = PriorElem(Lists.elem[now], e, pre);

if (result == INFEASIBLE)

printf("INFEASIBLE\n");

else if (result == ERROR)

printf("Error!\n");

else if (result == OK)

printf("Success! The prior elem is %d!\n", pre);

getchar();

break;

case 9:

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

printf("please input the number e:\n");

scanf("%d", &e);

result = NextElem(Lists.elem[now], e, next);

if (result == INFEASIBLE)

printf("INFEASIBLE\n");

else if (result == ERROR)

printf("Error!\n");

else if (result == OK)

printf("Success! The next elem is %d!\n", next);

getchar();

break;

case 10:

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

printf("please input the number i, e:\n");

scanf("%d%d", &i, &e);

result = ListInsert(Lists.elem[now], i, e);

if (result == INFEASIBLE)

printf("INFEASIBLE\n");

else if (result == ERROR)

printf("Error!\n");

else if (result == OK)

printf("Success!\n");

getchar();

break;

case 11:

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

printf("please input the number i:\n");

scanf("%d", &i);

result = ListDelete(Lists.elem[now], i, e);

if (result == OK)

printf("Success!\n");

else if (result == INFEASIBLE)

printf("INFEASIBLE\n");

else if (result == ERROR)

printf("Error!\n");

getchar();

break;

case 12:

//status ListTraverse(LinkList L);

result = ListTraverse(Lists.elem[now]);

if (result == OK)

printf("");

else if (result == INFEASIBLE)

printf("INFEASIBLE\n");

else if (result == ERROR)

printf("Error!\n");

getchar();

break;

case 13:

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

result = SaveList(Lists.elem[now], filename1);

if (result == INFEASIBLE)

printf("INFEASIBLE!\n");

else

printf("Success!\n");

getchar();

break;

case 14:

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

result = LoadList(Lists.elem[now], filename1);

if (result == INFEASIBLE)

printf("INFEASIBLE!\n");

else

printf("Success!\n");

getchar();

break;

case 0:

break;

}

if (op == 0)

{

system("cls");

printf("The op of Lists\n");

printf("1.AddList\n");

printf("2.ListsTraverse\n");

printf("3.LocateList\n");

printf("4.RemoveList\n");

printf("5.choose List\n");

printf("0.exit\n");

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

ReLists(Lists);

op = -1;

break;

}

printf("choose [0-14]:");

scanf("%d", &op);

}

}

break;

case 0:

printf("Welcome to use it next time!\n");

}

if (op == 0)

return 0;

printf("choose [0-5]:");

scanf("%d", &op);

}

}

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。

{

//检查L是否存在

if (L.elem)

{

return INFEASIBLE;

}

//不存在则构造一个空的线性表

else

{

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

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

printf("input the name of list:");

scanf("%s", L.name);

}

return OK;

}

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。

{

//检查线性表L是否存在

if (!L.elem)

return INFEASIBLE;

//销毁线性表L

else

{

free(L.elem);

L.length = L.listsize = 0;

L.elem = NULL;

L.name[0] = '\0';

}

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。

{

//检测L是否存在

if (!L.elem)

return INFEASIBLE;

//长度置为0

else

{

L.length = 0;

}

return OK;

}

status CreateNewList(SqList &L)

{

SqList L1;

L1.elem = NULL;

L = L1;

if (L.elem)

return ERROR;

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)

return INFEASIBLE;

else

{

if (L.length)

return FALSE;

return TRUE;

}

}

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。

{

//判断线性表L是否存在

if (!L.elem)

return INFEASIBLE;

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)

return INFEASIBLE;

else if (i > L.length || i < 1)

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)

return INFEASIBLE;

for (int i = 0; i < L.length; 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)

return INFEASIBLE;

for (int i = 1; i < L.length; i++)

{

if (L.elem[i] == e)

{

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)

return INFEASIBLE;

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

{

if (L.elem[i] == e)

{

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。

{

//检测线性表L是否存在

if (!L.elem)

return INFEASIBLE;

//检测插入位置是否正确

else if (i < 1 || i > L.length + 1)

return ERROR;

//检测插入后是否会溢出

else if (L.length == L.listsize)

{

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

L.listsize += LISTINCREMENT;

}

//插入

ElemType t1, t2;

t1 = e;

L.length++;

for (int j = i - 1; j < L.length; j++)

{

t2 = L.elem[j];

L.elem[j] = t1;

t1 = t2;

}

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)

return INFEASIBLE;

//检查删除位置是否正确

if (i < 1 || i > L.length)

return ERROR;

//进行删除操作

e = L.elem[i - 1];

L.length--;

for (int j = i - 1; j < L.length; j++)

{

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

}

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)

return INFEASIBLE;

printf("%s ", L.name);

if (L.length == 0)

printf("L.elem is Empty!\n");

for (int i = 0; i < L.length; i++)

{

if (i != L.length - 1)

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

else

printf("%d\n", 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。

{

if (!L.elem)

return INFEASIBLE;

FILE \*fp = fopen(FileName, "w");

putc(L.length, fp);

putc(L.listsize, fp);

fwrite(L.name, sizeof(char) \* 30, 1, fp);

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

fclose(fp);

return OK;

}

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。

{

if (L.elem)

return INFEASIBLE;

FILE \*fp = fopen(FileName, "r");

L.length = getc(fp);

L.listsize = getc(fp);

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

fread(L.name, sizeof(char) \* 30, 1, fp);

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

fclose(fp);

return OK;

}

status AddList(LISTS &Lists, SqList L)

// 将当前线性表L加入到Lists中

{

if (!L.elem)

return INFEASIBLE;

if (!Lists.elem)

return INFEASIBLE;

if (Lists.length == LISTS\_INIT\_SIZE)

{

Lists.listsSize += LISTSINCREMENT;

Lists.elem = (SqList \*)realloc(Lists.elem, sizeof(SqList) \* Lists.listsSize);

}

Lists.elem[Lists.length] = L;

Lists.length++;

return OK;

}

status InitLists(LISTS &Lists)

//初始化Lists

{

if (Lists.elem)

return INFEASIBLE;

else

{

Lists.length = 0;

Lists.listsSize = 10;

Lists.elem = (SqList \*)malloc(sizeof(SqList) \* LISTS\_INIT\_SIZE);

for (int i = 0; i < LISTS\_INIT\_SIZE; i++)

{

Lists.elem[i].elem = NULL;

}

}

return OK;

}

status SaveLists(LISTS &Lists, char FileName[])

// 如果Lists存在，将Lists的的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。

{

if (!Lists.elem)

return INFEASIBLE;

FILE \*fp = fopen(FileName, "w");

putc(Lists.length, fp);

putc(Lists.listsSize, fp);

for (int i = 0; i < Lists.length; i++)

{

putc(Lists.elem[i].length, fp);

putc(Lists.elem[i].listsize, fp);

fwrite(Lists.elem[i].name, sizeof(char) \* 30, 1, fp);

fwrite(Lists.elem[i].elem, sizeof(ElemType), Lists.elem[i].length, fp);

}

fclose(fp);

return OK;

}

status LoadLists(LISTS &Lists, char FileName[])

// 如果Lists不存在，将FileName文件中的数据读入到Lists中，返回OK，否则返回INFEASIBLE。

{

if (Lists.elem)

return INFEASIBLE;

FILE \*fp = fopen(FileName, "r");

Lists.length = getc(fp);

Lists.listsSize = getc(fp);

Lists.elem = (SqList \*)malloc(Lists.listsSize \* sizeof(SqList));

for (int i = 0; i < Lists.length; i++)

{

Lists.elem[i].length = getc(fp);

Lists.elem[i].listsize = getc(fp);

fread(Lists.elem[i].name, sizeof(char) \* 30, 1, fp);

Lists.elem[i].elem = (ElemType \*)malloc(sizeof(ElemType) \* Lists.elem[i].listsize);

fread(Lists.elem[i].elem, sizeof(ElemType), Lists.elem[i].length, fp);

}

fclose(fp);

return OK;

}

status ListsTraverse(LISTS Lists)

//依次输出Lists中的线性表的元素

{

if (!Lists.elem)

return INFEASIBLE;

if (Lists.length == 0)

{

printf("Is Empty!\n");

return 0;

}

for (int i = 0; i < Lists.length; i++)

{

ListTraverse(Lists.elem[i]);

}

return 1;

}

status 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

{

if (!Lists.elem)

return INFEASIBLE;

int i;

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

{

if (strcmp(Lists.elem[i].name, ListName) == 0)

return i + 1;

}

return 0;

}

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

// Lists中删除一个名称为ListName的线性表

{

if (!Lists.elem)

return INFEASIBLE;

int i;

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

{

if (strcmp(Lists.elem[i].name, ListName) == 0)

{

DestroyList(Lists.elem[i]);

Lists.length--;

for (; i < Lists.length; i++)

{

Lists.elem[i] = Lists.elem[i + 1];

}

return OK;

}

}

return ERROR;

}

status ReLists(LISTS &Lists)

{

for (int i = 0; i < Lists.length; i++)

{

if (Lists.elem[i].elem == NULL)

{

Lists.length--;

for (int j = i; j < Lists.length; j++)

{

Lists.elem[j] = Lists.elem[j + 1];

}

}

}

}

附录B 基于链式存储结构线性表实现的源程序

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#include "stdio.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 LISTS\_INIT\_SIZE 10

#define LISTSINCREMENT 10

typedef int ElemType;

typedef struct LNode

{ //单链表（链式结构）结点的定义

ElemType data;

struct LNode \*next;

} LNode, \*LinkList;

typedef struct List

{

LinkList L;

char name[30];

} List;

typedef struct Lists

{

List \*elem;

ElemType length;

ElemType listsSize;

} LISTS;

status InitList(LinkList &L);

status DestroyList(LinkList &L);

status ClearList(LinkList &L);

status ListEmpty(LinkList L);

status ListLength(LinkList L);

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

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, LinkList L);

status SaveLists(LISTS &Lists, char FileName[]);

status LoadLists(LISTS &Lists, char FileName[]);

status ListsTraverse(LISTS Lists);

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

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

status reLists(LISTS &Lists);

int main()

{

LISTS Lists;

Lists.length = 0;

Lists.listsSize = 10;

Lists.elem = (List \*)malloc(sizeof(List) \* LISTS\_INIT\_SIZE);

char filename1[30] = "LinkLists.txt";

char filename2[30] = "Lists.txt";

char Listname[30];

ElemType i, e, pre, next, result;

int op = 1, now = 0;

while (op)

{

printf("The op of Lists\n");

printf("1.AddList\n");

printf("2.ListsTraverse\n");

printf("3.LocateList\n");

printf("4.RemoveList\n");

printf("5.choose List\n");

printf("0.exit\n");

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

printf("choose [0-5]:");

scanf("%d", &op);

while (op)

{

switch (op)

{

case 1:

if (Lists.length == Lists.listsSize)

{

Lists.listsSize += LISTSINCREMENT;

Lists.elem = (List \*)realloc(Lists.elem, sizeof(List) \* Lists.listsSize);

}

printf("input the name of list:");

scanf("%s", Listname);

strcpy(Lists.elem[Lists.length].name, Listname);

printf("input the elem of list:");

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

if (InitList(Lists.elem[Lists.length].L) != 1)

printf("ERROR!");

int temp;

scanf("%d", &temp);

while (temp)

{

ListInsert(Lists.elem[Lists.length].L, ListLength(Lists.elem[Lists.length].L) + 1, temp);

scanf("%d", &temp);

}

now++;

Lists.length++;

break;

case 2:

ListsTraverse(Lists);

break;

case 3:

printf("input the name of List:");

scanf("%s", Listname);

result = LocateList(Lists, Listname);

if (result)

printf("the List %s is the %dth List\n", Listname, result);

else

printf("ERROR!\n");

break;

case 4:

printf("input the name of List:");

scanf("%s", Listname);

result = RemoveList(Lists, Listname);

if (result)

printf("Success!\n");

else

printf("ERROR!\n");

break;

case 5:

printf("input the name of List:");

scanf("%s", Listname);

result = LocateList(Lists, Listname);

if (result == 0)

{

printf("Don't exist!\n");

}

else

{

system("cls");

now = result - 1;

printf(" The List you choose is %s\n", Lists.elem[now].name);

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

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

printf(" The op of List\n");

printf(" 1. InitList 2. DestroyList\n");

printf(" 3. ClearList 4. ListEmpty\n");

printf(" 5. ListLength 6. GetElem \n");

printf(" 7. LocateElem 8. PriorElem\n");

printf(" 9. NextElem 10. ListInsert\n");

printf(" 11. ListDelete 12. ListTrabverse\n");

printf(" 13. SaveList 14.LoadList\n");

printf(" 0.exit \n");

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

printf("choose [0-14]:");

scanf("%d", &op);

while (op >= 0)

{

switch (op)

{

case 1:

//status InitList(LinkList &L)

if (InitList(Lists.elem[now].L) == OK)

printf("Success!\n");

else

printf("INFEASIBLE!\n");

getchar();

break;

case 2:

//status DestroyList(LinkList &L)

if (DestroyList(Lists.elem[now].L) == OK && Lists.elem[now].L == NULL)

printf("Success!\n");

else

printf("INFEASIBLE!\n");

getchar();

break;

case 3:

//status ClearList(LinkList &L)

if (ClearList(Lists.elem[now].L) == OK)

printf("Success!\n");

else

printf("INFEASIBLE!\n");

getchar();

break;

case 4:

//status ListEmpty(LinkList L)

result = ListEmpty(Lists.elem[now].L);

if (result == TRUE)

printf("is Empty!\n");

else if (result == FALSE)

printf("isn't Empty!\n");

else

printf("INFEASIBLE!\n");

getchar();

break;

case 5:

//status ListLength(LinkList L)

result = ListLength(Lists.elem[now].L);

if (result != INFEASIBLE)

printf("The length is %d\n", result);

else

printf("INFEASIBLE!\n");

getchar();

break;

case 6:

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

printf("please input the number i:\n");

scanf("%d", &i);

result = GetElem(Lists.elem[now].L, i, e);

if (result == INFEASIBLE)

printf("INFEASIBLE!\n");

else if (result == ERROR)

printf("Error!\n");

else

printf("The %dth number is %d\n", i, e);

getchar();

break;

case 7:

//status LocateElem(LinkList L, ElemType e)

printf("please input the number e:\n");

scanf("%d", &e);

result = LocateElem(Lists.elem[now].L, e);

if (result == INFEASIBLE)

printf("INFEASIBLE\n");

else if (result == ERROR)

printf("Error!\n");

else

printf("the elem %d is %dth!\n", e, result);

getchar();

break;

case 8:

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

printf("please input the number e:\n");

scanf("%d", &e);

result = PriorElem(Lists.elem[now].L, e, pre);

if (result == INFEASIBLE)

printf("INFEASIBLE\n");

else if (result == ERROR)

printf("Error!\n");

else if (result == OK)

printf("Success! The prior elem is %d!\n", pre);

getchar();

break;

case 9:

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

printf("please input the number e:\n");

scanf("%d", &e);

result = NextElem(Lists.elem[now].L, e, next);

if (result == INFEASIBLE)

printf("INFEASIBLE\n");

else if (result == ERROR)

printf("Error!\n");

else if (result == OK)

printf("Success! The next elem is %d!\n", next);

getchar();

break;

case 10:

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

printf("please input the number i, e:\n");

scanf("%d%d", &i, &e);

result = ListInsert(Lists.elem[now].L, i, e);

if (result == INFEASIBLE)

printf("INFEASIBLE\n");

else if (result == ERROR)

printf("Error!\n");

else if (result == OK)

printf("Success!\n");

getchar();

break;

case 11:

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

printf("please input the number i:\n");

scanf("%d", &i);

result = ListDelete(Lists.elem[now].L, i, e);

if (result == OK)

printf("Success!\n");

else if (result == INFEASIBLE)

printf("INFEASIBLE\n");

else if (result == ERROR)

printf("Error!\n");

getchar();

break;

case 12:

//status ListTraverse(LinkList L);

result = ListTraverse(Lists.elem[now].L);

if (result == OK)

printf("");

else if (result == INFEASIBLE)

printf("INFEASIBLE\n");

else if (result == ERROR)

printf("Error!\n");

getchar();

break;

case 13:

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

result = SaveList(Lists.elem[now].L, filename1);

if (result == INFEASIBLE)

printf("INFEASIBLE!\n");

else

printf("Success!\n");

getchar();

break;

case 14:

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

result = LoadList(Lists.elem[now].L, filename1);

if (result == INFEASIBLE)

printf("INFEASIBLE!\n");

else

printf("Success!\n");

getchar();

break;

default:

break;

}

if (op == 0)

{

system("cls");

printf("The op of Lists\n");

printf("1.AddList\n");

printf("2.ListsTraverse\n");

printf("3.LocateList\n");

printf("4.RemoveList\n");

printf("5.choose List\n");

printf("0.exit\n");

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

reLists(Lists);

op = -1;

break;

}

printf("choose [0-14]:");

scanf("%d", &op);

}

}

break;

case 0:

printf("Welcome to use it next time!\n");

default:

break;

}

if (op == 0)

return 0;

printf("choose [0-5]:");

scanf("%d", &op);

}

}

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。

{

if (L)

return INFEASIBLE;

else

{

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

L->next = NULL;

}

return OK;

}

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。

{

if (!L)

return INFEASIBLE;

else

{

free(L);

L = NULL;

}

return OK;

}

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。

{

if (!L)

return INFEASIBLE;

else

{

LinkList p = L;

LinkList p1 = p->next;

while (p1 != NULL)

{

p = p1;

p1 = p->next;

free(p);

}

L->next = NULL;

}

return OK;

}

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。

{

if (!L)

return INFEASIBLE;

else

{

if (L->next == NULL)

return TRUE;

else

return FALSE;

}

}

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。

{

if (!L)

return INFEASIBLE;

else

{

int i = 0;

LinkList p = L->next;

while (p)

{

i++;

p = p->next;

}

return i;

}

}

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。

{

if (!L)

return INFEASIBLE;

else if (i < 1)

return ERROR;

else

{

int j = 1;

LinkList p = L->next;

while (p)

{

if (i == j)

{

e = p->data;

return OK;

}

p = p->next;

j++;

}

}

return ERROR;

}

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。

{

if (!L)

return INFEASIBLE;

else

{

int i = 1;

LinkList p = L->next;

while (p)

{

if (p->data == e)

{

return i;

}

p = p->next;

i++;

}

}

return ERROR;

}

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。

{

if (!L)

return INFEASIBLE;

else

{

LinkList p = L;

LinkList p1 = p->next;

while (p1 && p1->next)

{

p = p1;

p1 = p->next;

if (p1->data == e)

{

pre = p->data;

return OK;

}

}

}

return ERROR;

}

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。

{

if (!L)

return INFEASIBLE;

else

{

LinkList p = L;

LinkList p1 = p->next;

while (p1 && p1->next)

{

p = p1;

p1 = p1->next;

if (p->data == e)

{

next = p1->data;

return OK;

}

}

}

return ERROR;

}

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。

{

if (!L)

return INFEASIBLE;

else if (i < 1)

return ERROR;

else

{

int j = 1;

LinkList p = L;

LinkList p1 = p->next;

while (p)

{

if (j == i)

//找到当前插入位置

{

LinkList p2;

p2 = (LinkList)malloc(sizeof(LNode));

p2->data = e;

p->next = p2;

p2->next = p1;

return OK;

}

if (!p1)

//已经到结尾还未找到插入位置

{

return ERROR;

}

p = p1;

p1 = p->next;

j++;

}

}

return ERROR;

}

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。

{

if (!L)

return INFEASIBLE;

if (i < 1)

return ERROR;

else

{

int j = 1;

LinkList p = L;

LinkList p1 = p->next;

while (p1)

{

if (j == i)

{

e = p1->data;

p->next = p1->next;

free(p1);

return OK;

}

p = p1;

p1 = p->next;

j++;

}

}

return ERROR;

}

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。

{

if (!L)

return INFEASIBLE;

else

{

L = L->next;

while (L)

{

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

L = L->next;

}

printf("\n");

}

return OK;

}

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。

{

if (!L)

return INFEASIBLE;

else

{

FILE \*fp = fopen(FileName, "w");

L = L->next;

while (L)

{

putc(L->data, fp);

L = L->next;

}

fclose(fp);

}

return OK;

}

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。

{

if (L)

return INFEASIBLE;

else

{

FILE \*fp = fopen(FileName, "r");

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

L->next = NULL;

LinkList p = L, p1;

ElemType t;

while ((t = getc(fp)) != EOF)

{

p1 = (LinkList)malloc(sizeof(LNode));

p1->data = t;

p->next = p1;

p = p1;

}

p->next = NULL;

fclose(fp);

}

return OK;

}

status AddList(LISTS &Lists, LinkList L)

// 将当前线性表L加入到Lists中

{

if (!L)

return INFEASIBLE;

if (!Lists.elem)

return INFEASIBLE;

if (Lists.length == LISTS\_INIT\_SIZE)

{

Lists.listsSize += LISTSINCREMENT;

Lists.elem = (List \*)realloc(Lists.elem, sizeof(List) \* Lists.listsSize);

}

printf("please input the name of List: ");

scanf("%s", Lists.elem[Lists.length].name);

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

Lists.length++;

return OK;

}

status SaveLists(LISTS &Lists, char FileName[])

// 如果Lists存在，将Lists的的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。

{

if (!Lists.elem)

return INFEASIBLE;

FILE \*fp = fopen(FileName, "w");

putc(Lists.length, fp);

putc(Lists.listsSize, fp);

for (int i = 0; i < Lists.length; i++)

{

fwrite(Lists.elem[i].name, sizeof(char) \* 30, 1, fp);

LinkList p = Lists.elem[i].L->next;

while (p)

{

putc(p->data, fp);

p = p->next;

}

putc('#', fp);

}

fclose(fp);

return OK;

}

status LoadLists(LISTS &Lists, char FileName[])

// 如果Lists不存在，将FileName文件中的数据读入到Lists中，返回OK，否则返回INFEASIBLE。

{

if (Lists.elem)

return INFEASIBLE;

FILE \*fp = fopen(FileName, "r");

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

ElemType t;

p->next = NULL;

Lists.length = getc(fp);

Lists.listsSize = getc(fp);

Lists.elem = (List \*)malloc(Lists.listsSize \* sizeof(List));

for (int i = 0; i < Lists.length; i++)

{

fread(Lists.elem[i].name, sizeof(char) \* 30, 1, fp);

Lists.elem[i].L = p;

while ((t = getc(fp)) != EOF && t != '#')

{

LinkList p1 = (LinkList)malloc(sizeof(LNode));

p1->data = t;

p->next = p1;

p1->next = NULL;

p = p1;

}

}

fclose(fp);

return OK;

}

status ListsTraverse(LISTS Lists)

//依次输出Lists中的线性表的元素

{

if (!Lists.elem)

return INFEASIBLE;

if (Lists.length == 0)

{

printf("Is Empty!\n");

return 0;

}

for (int i = 0; i < Lists.length; i++)

{

printf("%s ", Lists.elem[i].name);

ListTraverse(Lists.elem[i].L);

}

return 1;

}

status 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

{

if (!Lists.elem)

return INFEASIBLE;

int i;

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

{

if (strcmp(Lists.elem[i].name, ListName) == 0)

return i + 1;

}

return 0;

}

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

// Lists中删除一个名称为ListName的线性表

{

if (!Lists.elem)

return INFEASIBLE;

int i;

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

{

if (strcmp(Lists.elem[i].name, ListName) == 0)

{

DestroyList(Lists.elem[i].L);

Lists.elem[i].name[0] = '/0';

Lists.length--;

for (; i < Lists.length; i++)

{

Lists.elem[i] = Lists.elem[i + 1];

}

return OK;

}

}

return ERROR;

}

status reLists(LISTS &Lists)

{

for (int i = 0; i < Lists.length; i++)

{

if (Lists.elem[i].L == NULL)

{

Lists.length--;

for (int j = i; j < Lists.length; j++)

{

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

strcpy(Lists.elem[j].name, Lists.elem[j + 1].name);

}

}

}

}

# 附录C 基于二叉链表二叉树实现的源程序

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status;

typedef int KeyType;

typedef struct

{

  KeyType key;

  char others[100];

} TElemType; //二叉树结点类型定义

typedef struct BiTNode

{ //二叉链表结点的定义

  TElemType data;

  struct BiTNode \*lchild, \*rchild;

} BiTNode, \*BiTree;

typedef struct

{

  BiTree T;

  char name[30];

} List;

void visit(BiTree T)

{

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

}

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

status FindKey(TElemType definition[]);

status DestroyBiTree(BiTree &T);

status ClearBiTree(BiTree &T);

BiTree CreateTree(TElemType definition[]);

status BiTreeEmpty(BiTree T);

int BiTreeDepth(BiTree T);

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

BiTNode \*LocateFatherNode(BiTree T, KeyType e);

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

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

status find(BiTree T, KeyType key);

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

status DeleteNode(BiTree &T, KeyType e);

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

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

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

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

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

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

int main()

{

  char filename[30] = "BiTree.txt";

  int e, result;

  int op = 1;

  TElemType defination[100];

  TElemType value;

  BiTree p, T = NULL;

  int LR;

  int def\_i;

  while (op)

  {

    system("cls");

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

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

    printf("                The op of List\n");

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

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

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

    printf("        7. Assign     8. GetSibling\n");

    printf("        9. InsertNode       10. DeleteNode\n");

    printf("        11. PreOrderTraverse    12. InOrderTraverse\n");

    printf("        13. PostOrderTraverse      14.LevelOrderTraverse\n");

    printf("        15.SaveBiTree     16.LoadBiTree   \n");

    printf("        0.exit   \n");

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

    printf("choose [0-16]:");

    scanf("%d", &op);

    while (op >= 0)

    {

      switch (op)

      {

      case 1:

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

        printf("input the elem of list:");

        def\_i = 0;

        scanf("%d%s", &defination[def\_i].key, defination[def\_i].others);

        while (defination[def\_i].key != -1)

        {

          def\_i++;

          scanf("%d%s", &defination[def\_i].key, defination[def\_i].others);

        }

        if (CreateBiTree(T, defination) == OK)

          printf("Success!\n");

        else

          printf("INFEASIBLE!\n");

        getchar();

        break;

      case 2:

        //status DestroyBiTree(BiTree &T)

        if (T != NULL && DestroyBiTree(T) == OK)

          printf("Success!\n");

        else

          printf("INFEASIBLE!\n");

        getchar();

        break;

      case 3:

        //status ClearBiTree(BiTree &T)

        if (ClearBiTree(T) == OK)

          printf("Success!\n");

        else

          printf("INFEASIBLE!\n");

        getchar();

        break;

      case 4:

        //status BiTreeEmpty(BiTree T)

        result = BiTreeEmpty(T);

        if (result == TRUE)

          printf("is Empty!\n");

        else if (result == FALSE)

          printf("isn't Empty!\n");

        else

          printf("INFEASIBLE!\n");

        getchar();

        break;

      case 5:

        //int BiTreeDepth(BiTree T)

        result = BiTreeDepth(T);

        printf("The depth is %d\n", result);

        getchar();

        break;

      case 6:

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

        printf("please input the key e:\n");

        scanf("%d", &e);

        p = LocateNode(T, e);

        if (p == NULL)

          printf("ERROR!\n");

        else

        {

          printf("Success!\np:");

          visit(p);

          printf("\n");

        }

        getchar();

        break;

      case 7:

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

        printf("please input the key e and value.key, value.others:\n");

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

        result = Assign(T, e, value);

        if (result == NULL)

          printf("ERROR!\n");

        else

          printf("Success!\n");

        getchar();

        break;

      case 8:

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

        printf("please input the key e:\n");

        scanf("%d", &e);

        p = GetSibling(T, e);

        if (p == NULL)

          printf("ERROR!\n");

        else

        {

          printf("Success!\np:");

          visit(p);

          printf("\n");

        }

        getchar();

        break;

      case 9:

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

        printf("please input the key e, LR, value.key, value.others:\n");

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

        result = InsertNode(T, e, LR, value);

        if (result == ERROR)

          printf("ERROR!\n");

        else if (result == OK)

          printf("Success!\n");

        getchar();

        break;

      case 10:

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

        printf("please input the number e:\n");

        scanf("%d", &e);

        result = DeleteNode(T, e);

        if (result == ERROR)

          printf("ERROR!\n");

        else if (result == OK)

          printf("Success!\n");

        getchar();

        break;

      case 11:

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

        result = PreOrderTraverse(T, visit);

        printf("\n");

        getchar();

        break;

      case 12:

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

        result = InOrderTraverse(T, visit);

        printf("\n");

        getchar();

        break;

      case 13:

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

        result = PostOrderTraverse(T, visit);

        printf("\n");

        getchar();

        break;

      case 14:

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

        result = LevelOrderTraverse(T, visit);

        printf("\n");

        getchar();

        break;

      case 15:

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

        result = SaveBiTree(T, filename);

        if (result == ERROR)

          printf("ERROR!\n");

        else

          printf("Success!\n");

        getchar();

        break;

      case 16:

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

        result = LoadBiTree(T, filename);

        if (result == ERROR)

          printf("ERROR!\n");

        else

          printf("Success!\n");

        getchar();

        break;

      default:

        break;

      }

      if (op == 0)

        break;

      printf("choose [0-16]:");

      scanf("%d", &op);

    }

  }

  return 0;

}

int def\_Create\_i = 0;

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。此题允许通过增加其它函数辅助实现本关任务\*/

{

  if (!FindKey(definition))

    return ERROR;

  else

  {

    T = CreateTree(definition);

    def\_Create\_i = 0;

  }

  return OK;

}

BiTree CreateTree(TElemType definition[])

{

  BiTree T;

  if (definition[def\_Create\_i].key == 0)

  {

    def\_Create\_i++;

    T = NULL;

  }

  else

  {

    T = new BiTNode;

    T->data = definition[def\_Create\_i];

    def\_Create\_i++;

    T->lchild = CreateTree(definition);

    T->rchild = CreateTree(definition);

  }

  return T;

}

status FindKey(TElemType definition[])

{

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

  {

    for (int j = i + 1; definition[j].key != -1; j++)

    {

      if (definition[i].key == definition[j].key && definition[i].key != 0)

        return ERROR;

    }

  }

  return OK;

}

status DestroyBiTree(BiTree &T)

{

  if (!T)

    return OK;

  else

  {

    DestroyBiTree(T->lchild);

    DestroyBiTree(T->rchild);

    free(T);

    T = NULL;

    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

{

  if (!T)

    return OK;

  ClearBiTree(T->lchild);

  ClearBiTree(T->rchild);

  free(T);

  T = NULL;

  return OK;

}

status BiTreeEmpty(BiTree T)

{

  if (!T)

    return TRUE;

  else

    return FALSE;

}

int BiTreeDepth(BiTree T)

//求二叉树T的深度

{

  if (!T)

    return 0;

  else

  {

    int l = BiTreeDepth(T->lchild);

    int r = BiTreeDepth(T->rchild);

    return l > r ? l + 1 : r + 1;

  }

}

BiTNode \*LocateNode(BiTree T, KeyType e)

//查找结点

{

  BiTree p;

  if (!T)

    return NULL;

  else

  {

    if (T->data.key == e)

    {

      return T;

    }

    else

      p = LocateNode(T->lchild, e);

    if (p)

      return p;

    else

      return LocateNode(T->rchild, e);

  }

}

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

{

  if (!find(T, value.key) && e != value.key)

    return NULL;

  BiTree p = LocateNode(T, e);

  if (!p)

    return NULL;

  else

  {

    p->data = value;

    return OK;

  }

}

BiTNode \*GetSibling(BiTree T, KeyType e)

//实现获得兄弟结点

{

  BiTree p;

  if (!T)

    return NULL;

  else

  {

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

      return T->rchild;

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

      return T->lchild;

    else

      p = GetSibling(T->lchild, e);

    if (p)

      return p;

    else

      return GetSibling(T->rchild, e);

  }

}

status find(BiTree T, KeyType key)

//查找key为e的结点，存在返回ERROR,不存在返回OK

{

  int i;

  if (!T)

    return OK;

  else

  {

    if (T->data.key == key)

    {

      return ERROR;

    }

    else

      i = find(T->lchild, key);

    if (i == 0)

      return ERROR;

    else

      return find(T->rchild, key);

  }

}

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

{

  if ((find(T, c.key) == ERROR))

  {

    //printf("ERROR 1");

    return ERROR;

  }

  if ((find(T, e) == OK))

  {

    //printf("ERROR 2");

    return ERROR;

  }

  else

  {

    BiTree p = LocateNode(T, e);

    if (!p)

    {

      //printf("ERROR 3");

      return ERROR;

    }

    BiTree q;

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

    m->data = c;

    m->lchild = NULL;

    m->rchild = NULL;

    if (LR == -1)

    {

      q = T;

      m->rchild = T;

      T = m;

    }

    else if (LR == 0)

    {

      q = p->lchild;

      p->lchild = m;

      m->rchild = q;

    }

    else if (LR == 1)

    {

      q = p->rchild;

      p->rchild = m;

      m->rchild = q;

    }

    return OK;

  }

}

BiTNode \*LocateFatherNode(BiTree T, KeyType e)

//查找父亲结点

{

  BiTree p;

  if (!T)

    return NULL;

  if ((!T->lchild) && (!T->rchild))

    return NULL;

  else

  {

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

    {

      return T;

    }

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

      return T;

    else

      p = LocateFatherNode(T->lchild, e);

    if (p)

      return p;

    else

      return LocateFatherNode(T->rchild, e);

  }

}

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

{

  if (!T)

    return ERROR;

  if (T->data.key == e)

  {

    if (T->lchild && T->rchild)

    {

      BiTree l = T->lchild;

      BiTree r = T->rchild;

      free(T);

      T = l;

      while (l->rchild)

        l = l->rchild;

      l->rchild = r;

    }

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

    {

      free(T);

      T = NULL;

    }

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

    {

      BiTree q = T->lchild;

      free(T);

      T = q;

    }

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

    {

      BiTree q = T->rchild;

      free(T);

      T = q;

    }

    return OK;

  }

  BiTree p = LocateFatherNode(T, e);

  if (!p)

    return ERROR;

  BiTree q = p->lchild && p->lchild->data.key == e ? p->lchild : p->rchild;

  int i = p->lchild && p->lchild->data.key == e ? 0 : 1;

  if (p && q)

  {

    if (q->lchild && q->rchild)

    {

      BiTree l = q->lchild;

      BiTree r = q->rchild;

      free(q);

      q = l;

      while (l->rchild)

        l = l->rchild;

      l->rchild = r;

      if (i)

        p->lchild = q;

      else

        p->lchild = q;

    }

    else if ((!q->lchild) && (!q->rchild))

    {

      free(q);

      q = NULL;

      if (i)

        p->lchild = NULL;

      else

        p->lchild = NULL;

    }

    else if ((q->lchild && (!q->rchild)))

    {

      BiTree m = q->lchild;

      free(q);

      q = m;

      if (i)

        p->lchild = q;

      else

        p->lchild = q;

    }

    else if ((!q->lchild) && q->rchild)

    {

      BiTree m = q->rchild;

      free(q);

      q = m;

      if (i)

        p->lchild = q;

      else

        p->lchild = q;

    }

    return OK;

  }

}

typedef struct

{

  BiTree \*elem;

  int top;

  int base;

} SqStack;

#define MAXSIZE 100

int initStack(SqStack &S)

{

  S.elem = (BiTree \*)malloc(sizeof(BiTree) \* 100);

  S.top = S.base = 0;

  return 1;

}

int pop(SqStack &S, BiTree &q)

{

  if (S.top == S.base)

    return 0;

  else

  {

    q = S.elem[--S.top];

    return 1;

  }

}

int push(SqStack &S, BiTree &q)

{

  if (S.top - S.base == MAXSIZE)

    return 0;

  else

  {

    S.elem[S.top++] = q;

    return 1;

  }

}

int StackEmpty(SqStack S)

{

  if (S.top == S.base)

    return 1;

  return 0;

}

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

{

  SqStack S;

  initStack(S);

  BiTree q;

  if (T)

    push(S, T);

  while (!StackEmpty(S))

  {

    q = NULL;

    pop(S, q);

    if (q)

      visit(q);

    if (q && q->rchild)

      push(S, q->rchild);

    if (q && q->lchild)

      push(S, q->lchild);

  }

  return OK;

}

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

{

  SqStack S;

  initStack(S);

  BiTree q = T;

  while (!StackEmpty(S) || q)

  {

    if (q)

    {

      push(S, q);

      q = q->lchild;

    }

    else

    {

      pop(S, q);

      visit(q);

      q = q->rchild;

    }

  }

  return OK;

}

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

//先序遍历二叉树T

{

  if (T)

  {

    visit(T);

    PreOrderTraverse(T->lchild, visit);

    PreOrderTraverse(T->rchild, visit);

  }

  return OK;

}

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

//中序遍历二叉树T

{

  if (T)

  {

    InOrderTraverse(T->lchild, visit);

    visit(T);

    InOrderTraverse(T->rchild, visit);

  }

  return OK;

}\*/

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

//后序遍历二叉树T

{

  if (T)

  {

    PostOrderTraverse(T->lchild, visit);

    PostOrderTraverse(T->rchild, visit);

    visit(T);

  }

  return OK;

}

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

//按层遍历二叉树T

{

  int length = 0;

  BiTree List[100];

  BiTree p = NULL;

  if (T)

  {

    List[length++] = T;

  }

  while (length > 0)

  {

    p = List[0];

    for (int j = 0; j + 1 < length; j++)

    {

      List[j] = List[j + 1];

    }

    length--;

    visit(p);

    if (p->lchild) //左孩子不空，入队列

    {

      List[length++] = p->lchild;

    }

    if (p->rchild) //右孩子不空，入队列

    {

      List[length++] = p->rchild;

    }

  }

  return OK;

}

status write(BiTree T, FILE \*fp)

{

  if (!T)

  {

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

    p->data.key = 0;

    strcpy(p->data.others, "NULL");

    fwrite(p, sizeof(BiTree), 1, fp);

    return NULL;

  }

  else

  {

    fwrite(T, sizeof(BiTree), 1, fp);

    write(T->lchild, fp);

    write(T->rchild, fp);

    return OK;

  }

}

status read(BiTree &T, FILE \*fp)

{

  if (feof(fp))

    return OK;

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

  fread(p, sizeof(BiTree), 1, fp);

  if (p->data.key)

  {

    T = p;

    T->lchild = T->rchild = NULL;

    read(T->lchild, fp);

    read(T->rchild, fp);

  }

  else

  {

    T = NULL;

  }

  return OK;

}

status SaveBiTree(BiTree T, char FileName[])

{

  if (!T)

    return ERROR;

  else

  {

    FILE \*fp = fopen(FileName, "w");

    write(T, fp);

    fclose(fp);

    return OK;

  }

}

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

{

  if (T)

    return ERROR;

  else

  {

    FILE \*fp = fopen(FileName, "r");

    read(T, fp);

    fclose(fp);

    return OK;

  }

}

# 附录D 基于邻接表图实现的源程序

#include "stdio.h"

#include "stdlib.h"

#include <string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define MAX\_VERTEX\_NUM 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

{

int elem[MAX\_VERTEX\_NUM];

int front, tail;

} QUEUE;

typedef struct

{

ALGraph G;

char name[30];

} LIST;

typedef struct

{

LIST elem[20];

int length;

} LISTS;

int visited[MAX\_VERTEX\_NUM];

void visit(VertexType v)

{

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

}

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

status Find(VertexType V[], KeyType VR[][2]);

status CreateVR(VNode &V1, 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 DeleteVR(ALGraph &G, int adjvex, int result);

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 ALGraphsTraverse(LISTS Lists);

status LocateALGraph(LISTS Lists, char name[]);

status RemoveALGraph(LISTS &Lists, char name[]);

status ALGraphTraverse(ALGraph G);

int main()

{

LISTS Lists;

Lists.length = 0;

char FileName[30] = "ALGraph.txt";

char Listname[30];

KeyType u, w;

int op = 1, now = 0, result, i;

VertexType value;

VertexType V[21];

KeyType VR[100][2];

for (int i = 0; i < MAX\_VERTEX\_NUM; i++)

{

Lists.elem[i].G.arcnum = 0;

Lists.elem[i].G.vexnum = 0;

}

while (op)

{

printf("The op of ALGraphs\n");

printf("1.AddALGraph\n");

printf("2.ALGraphsTraverse\n");

printf("3.LocateALGraph\n");

printf("4.RemoveALGraph\n");

printf("5.choose ALGraph\n");

printf("0.exit\n");

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

printf("choose [0-5]:");

scanf("%d", &op);

while (op)

{

switch (op)

{

case 1:

i = 0;

printf("input the name of ALGraph:");

scanf("%s", Listname);

strcpy(Lists.elem[Lists.length].name, Listname);

printf("input the elem of ALGraph:");

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

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

{

i++;

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

}

i = 0;

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

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

{

i++;

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

}

if (CreateCraph(Lists.elem[Lists.length].G, V, VR) != OK)

{

printf("ERROR!\n");

Lists.elem[Lists.length].G.vexnum = 0;

}

else

{

printf("Success!\n");

now++;

Lists.length++;

}

break;

case 2:

ALGraphsTraverse(Lists);

break;

case 3:

printf("input the name of ALGraph:");

scanf("%s", Listname);

result = LocateALGraph(Lists, Listname);

if (result)

printf("the ALGraph %s is the %dth ALGraph\n", Listname, result);

else

printf("ERROR!\n");

break;

case 4:

printf("input the name of ALGraph:");

scanf("%s", Listname);

result = RemoveALGraph(Lists, Listname);

if (result)

printf("Success!\n");

else

printf("ERROR!\n");

break;

case 5:

printf("input the name of ALGraph:");

scanf("%s", Listname);

result = LocateALGraph(Lists, Listname);

if (result == 0)

{

printf("Don't exist!\n");

}

else

{

system("cls");

now = result - 1;

printf(" The ALGraph you choose is %s\n", Lists.elem[now].name);

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

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

printf(" The op of ALGraph\n");

printf(" 1. CreateCraph 2. DestroyGraph\n");

printf(" 3. LocateVex 4. PutVex\n");

printf(" 5. FirstAdjVex 6. NextAdjVex \n");

printf(" 7. InsertVex 8. DeleteVex\n");

printf(" 9. InsertArc 10. DeleteArc\n");

printf(" 11. DFSTraverse 12. BFSTraverse\n");

printf(" 13. SaveGraph 14.LoadGraph\n");

printf(" 15. ALGraphTraverse 0.exit \n");

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

printf("choose [0-15]:");

scanf("%d", &op);

while (op >= 0)

{

switch (op)

{

case 1:

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

printf("input the elem of ALGraph:");

i = 0;

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

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

{

i++;

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

}

i = 0;

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

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

{

i++;

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

}

if (CreateCraph(Lists.elem[now].G, V, VR) == OK)

printf("Success!\n");

else

printf("INFEASIBLE!\n");

getchar();

break;

case 2:

//status DestroyGraph(ALGraph &G);

if (DestroyGraph(Lists.elem[now].G) == OK)

printf("Success!\n");

else

printf("INFEASIBLE!\n");

getchar();

break;

case 3:

//int LocateVex(ALGraph G, KeyType u);

printf("please input the key u:\n");

scanf("%d", &u);

result = LocateVex(Lists.elem[now].G, u);

if (result >= 0)

printf("The Vex %s is the %d vex\n", Lists.elem[now].G.vertices[result].data.others, result);

else

printf("ERROR!\n");

getchar();

break;

case 4:

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

printf("please input the key u, value:\n");

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

result = PutVex(Lists.elem[now].G, u, value);

if (result == OK)

printf("Success!\n");

else

printf("ERROR!\n");

getchar();

break;

case 5:

//int FirstAdjVex(ALGraph G, KeyType u);

printf("please input the key u:\n");

scanf("%d", &u);

result = FirstAdjVex(Lists.elem[now].G, u);

if (result != INFEASIBLE)

printf("The FirstAdjVex is %d\n", result);

else

printf("ERROR!\n");

getchar();

break;

case 6:

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

printf("please input the key u, w:\n");

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

result = NextAdjVex(Lists.elem[now].G, u, w);

if (result == INFEASIBLE)

printf("ERROR!\n");

else

printf("The NextAdjVex is %d\n", result);

getchar();

break;

case 7:

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

printf("please input the value:\n");

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

result = InsertVex(Lists.elem[now].G, value);

if (result == ERROR)

printf("ERROR!\n");

else

printf("Success!\n");

getchar();

break;

case 8:

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

printf("please input the key u:\n");

scanf("%d", &u);

result = DeleteVex(Lists.elem[now].G, u);

if (result == ERROR)

printf("ERROR!\n");

else

printf("Success!\n");

getchar();

break;

case 9:

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

printf("please input the key u, w:\n");

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

result = InsertArc(Lists.elem[now].G, u, w);

if (result == ERROR)

printf("ERROR!\n");

else

printf("Success!\n");

getchar();

break;

case 10:

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

printf("please input the key u, w:\n");

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

result = DeleteArc(Lists.elem[now].G, u, w);

if (result == ERROR)

printf("ERROR!\n");

else

printf("Success!\n");

getchar();

break;

case 11:

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

result = DFSTraverse(Lists.elem[now].G, visit);

if (result == OK)

printf("\n");

getchar();

break;

case 12:

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

result = BFSTraverse(Lists.elem[now].G, visit);

if (result == OK)

printf("\n");

getchar();

break;

case 13:

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

result = SaveGraph(Lists.elem[now].G, FileName);

if (result == ERROR)

printf("ERROR!\n");

else

printf("Success!\n");

getchar();

break;

case 14:

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

result = LoadGraph(Lists.elem[now].G, FileName);

if (result == ERROR)

printf("ERROR!\n");

else

printf("Success!\n");

getchar();

break;

case 15:

//status ALGraphTraverse(ALGraph G)

ALGraphTraverse(Lists.elem[now].G);

getchar();

break;

default:

break;

}

if (op == 0)

{

system("cls");

printf("The op of ALGraphs\n");

printf("1.AddALGraph\n");

printf("2.ALGraphsTraverse\n");

printf("3.LocateALGraph\n");

printf("4.RemoveALGraph\n");

printf("5.choose ALGraph\n");

printf("0.exit\n");

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

op = -1;

break;

}

printf("choose [0-14]:");

scanf("%d", &op);

}

}

break;

case 0:

printf("Welcome to use it next time!\n");

default:

break;

}

if (op == 0)

return 0;

printf("choose [0-5]:");

scanf("%d", &op);

}

}

return 0;

}

status Find(VertexType V[], KeyType VR[][2])

{

int i, j;

KeyType a[100];

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

{

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

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

{

if (V[i].key == V[j].key)

return ERROR;

}

}

a[i] = -1;

int flag;

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

{

flag = 0;

for (j = 0; a[j] != -1; j++)

{

if (VR[i][0] == a[j])

flag++;

if (VR[i][1] == a[j])

flag++;

}

if (flag != 2)

return ERROR;

}

return OK;

}

status CreateVR(VNode &V1, VertexType V[], KeyType VR[][2])

{

int i, j, k;

ArcNode \*head, \*q;

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

{

if (VR[i][0] == V1.data.key || VR[i][1] == V1.data.key)

{

k = VR[i][0] == V1.data.key ? 1 : 0;

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

{

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

break;

}

head = V1.firstarc;

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

q->adjvex = j;

q->nextarc = head;

V1.firstarc = q;

}

}

return OK;

}

status CreateCraph(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\*/

{

if (!Find(V, VR))

return ERROR;

if (G.vexnum > 0)

{

printf("G.vexnum:%d\n", G.vexnum);

return ERROR;

}

else

{

int i, j;

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

{

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

G.vertices[i].firstarc = NULL;

CreateVR(G.vertices[i], V, VR);

}

if (i > MAX\_VERTEX\_NUM || i < 1)

{

//printf("%d\n", i);

return ERROR;

}

//printf("%d\n", i);

G.vexnum = i;

//printf("%d\n", G.vexnum);

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

j++;

G.arcnum = j;

}

return OK;

}

status DestroyGraph(ALGraph &G)

/\*销毁无向图G,删除G的全部顶点和边\*/

{

if (G.vexnum < 1)

return ERROR;

ArcNode \*p, \*q;

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

{

p = G.vertices[i].firstarc;

while (p)

{

q = p->nextarc;

free(p);

p = q;

}

}

G.vexnum = G.arcnum = 0;

return OK;

}

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；

{

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

{

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

return i;

}

return -1;

}

status FindKey(ALGraph G, KeyType u, VertexType value)

//判断value中的key值和其它key值有无冲突

{

if (u == value.key)

return OK;

else

{

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

{

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

return -1;

}

}

return OK;

}

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

{

int i = LocateVex(G, u);

if (i == -1)

return ERROR;

if (FindKey(G, u, value) == -1)

return ERROR;

G.vertices[i].data = value;

return OK;

}

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；

{

int result = LocateVex(G, u);

if (result == -1)

return -1;

else if (G.vertices[result].firstarc)

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

return -1;

}

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；

{

int result1, result2;

result1 = LocateVex(G, v);

result2 = LocateVex(G, w);

if (result1 == -1 || result2 == -1)

return -1;

else

{

ArcNode \*p = G.vertices[result1].firstarc;

while (p)

{

if (p->adjvex == result2)

{

p = p->nextarc;

if (p)

return p->adjvex;

else

return -1;

}

p = p->nextarc;

}

}

return -1;

}

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

{

if (LocateVex(G, v.key) != -1 || G.vexnum >= 20)

return ERROR;

else

{

G.vertices[G.vexnum].data = v;

G.vertices[G.vexnum].firstarc = NULL;

G.vexnum++;

return OK;

}

}

status DeleteVR(ALGraph &G, int adjvex, int result)

//adjvex是待删除的结点序号，result是待操作的结点序号

{

ArcNode \*p1, \*p2;

p1 = G.vertices[result].firstarc;

if (p1)

p2 = p1->nextarc;

else

return OK;

if (p1->adjvex == adjvex)

{

free(p1);

p1 = NULL;

G.vertices[result].firstarc = p2;

G.arcnum--;

}

else

{

while (p2)

{

if (p2->adjvex == adjvex)

{

G.arcnum--;

p1->nextarc = p2->nextarc;

free(p2);

p2 = NULL;

return OK;

}

else

{

p1 = p2;

p2 = p1->nextarc;

}

}

}

return OK;

}

status DeleteV(ALGraph &G, int adjvex)

{

ArcNode \*p1;

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

{

DeleteVR(G, adjvex, i);

p1 = G.vertices[i].firstarc;

while (p1)

{

if (p1->adjvex > adjvex)

p1->adjvex = p1->adjvex - 1;

p1 = p1->nextarc;

}

}

return OK;

}

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

{

if (G.vexnum < 1)

return ERROR;

int result = LocateVex(G, v);

if (result == -1)

return ERROR;

else

{

G.vexnum--;

ArcNode \*p1, \*p2;

p1 = G.vertices[result].firstarc;

while (p1)

{

p2 = p1->nextarc;

free(p1);

p1 = NULL;

p1 = p2;

}

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

{

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

}

DeleteV(G, result);

return OK;

}

}

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

{

int result1 = LocateVex(G, v);

int result2 = LocateVex(G, w);

if (result1 == -1 || result2 == -1)

return ERROR;

else

{

ArcNode \*p = G.vertices[result1].firstarc;

while (p)

{

if (p->adjvex == result2)

return ERROR;

p = p->nextarc;

}

ArcNode \*p1, \*p2;

p1 = (ArcNode \*)malloc(sizeof(ArcNode));

p1->adjvex = result2;

p1->nextarc = G.vertices[result1].firstarc;

G.vertices[result1].firstarc = p1;

p2 = (ArcNode \*)malloc(sizeof(ArcNode));

p2->adjvex = result1;

p2->nextarc = G.vertices[result2].firstarc;

G.vertices[result2].firstarc = p2;

G.arcnum++;

return OK;

}

}

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

{

int result1 = LocateVex(G, v);

int result2 = LocateVex(G, w);

if (result1 == -1 || result2 == -1)

return ERROR;

else

{

ArcNode \*p = G.vertices[result1].firstarc;

int flag = 0;

while (p)

{

if (p->adjvex == result2)

{

flag = 1;

break;

}

p = p->nextarc;

}

if (flag == 0)

return ERROR;

else

{

DeleteVR(G, result1, result2);

DeleteVR(G, result2, result1);

G.arcnum++;

}

return OK;

}

}

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

{

visited[i] = TRUE;

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

ArcNode \*p;

p = G.vertices[i].firstarc;

while (p)

{

if (!visited[p->adjvex])

{

DFS(G, p->adjvex, visit);

}

p = p->nextarc;

}

}

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

{

int i;

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

{

visited[j] = FALSE;

}

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

{

if (!visited[i])

{

DFS(G, i, visit);

}

}

return OK;

}

status InitQueue(QUEUE &Q)

{

Q.front = Q.tail = 0;

return OK;

}

status QueueEmpty(QUEUE Q)

{

if (Q.front == Q.tail)

return OK;

return ERROR;

}

status QueueFull(QUEUE Q)

{

if ((Q.tail + MAX\_VERTEX\_NUM - Q.front) % MAX\_VERTEX\_NUM == MAX\_VERTEX\_NUM - 1)

return OK;

return ERROR;

}

status EnQueue(QUEUE &Q, int v)

{

if (QueueFull(Q))

return ERROR;

Q.elem[Q.tail] = v;

Q.tail = (Q.tail + 1) % MAX\_VERTEX\_NUM;

return OK;

}

status DeQueue(QUEUE &Q, int &v)

{

if (QueueEmpty(Q))

return ERROR;

v = Q.elem[Q.front];

Q.front = (Q.front + 1) % MAX\_VERTEX\_NUM;

return OK;

}

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

{

QUEUE Q;

InitQueue(Q);

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

{

visited[i] = FALSE;

}

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

{

if (!visited[i])

{

visited[i] = TRUE;

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

EnQueue(Q, i);

while (!QueueEmpty(Q))

{

DeQueue(Q, i);

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

while (p)

{

if (!visited[p->adjvex])

{

visited[p->adjvex] = TRUE;

visit(G.vertices[p->adjvex].data);

EnQueue(Q, p->adjvex);

}

p = p->nextarc;

}

}

}

}

return OK;

}

status SaveGraph(ALGraph G, char FileName[])

//将图的数据写入到文件FileName中

{

if (G.vexnum < 1)

return ERROR;

FILE \*fp = fopen(FileName, "w");

ArcNode \*p;

putc(G.vexnum, fp);

putc(G.arcnum, fp);

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

{

fwrite(&G.vertices[i].data, sizeof(VNode), 1, fp);

}

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

{

p = G.vertices[i].firstarc;

while (p)

{

putc(p->adjvex, fp);

p = p->nextarc;

}

putc(21, fp);

}

fclose(fp);

return OK;

}

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

//读入文件FileName的图数据，创建图的邻接表

{

if (G.vexnum > 0)

return ERROR;

FILE \*fp = fopen(FileName, "r");

ArcNode \*p;

G.vexnum = getc(fp);

G.arcnum = getc(fp);

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

{

fread(&G.vertices[i].data, sizeof(VNode), 1, fp);

G.vertices[i].firstarc = NULL;

}

int j;

int adj[20];

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

{

for (j = 0; (adj[j] = getc(fp)) >= 0; j++)

{

if (adj[j] == 21)

break;

}

j--;

while (j >= 0)

{

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

p->adjvex = adj[j--];

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

G.vertices[i].firstarc = p;

}

}

fclose(fp);

return OK;

}

status ALGraphTraverse(ALGraph G)

{

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

{

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

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

while (p)

{

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

p = p->nextarc;

}

printf("\n");

}

return OK;

}

status ALGraphsTraverse(LISTS Lists)

{

for (int i = 0; i < Lists.length; i++)

{

printf("%s ", Lists.elem[i].name);

BFSTraverse(Lists.elem[i].G, visit);

printf("\n");

}

return OK;

}

status LocateALGraph(LISTS Lists, char name[])

{

for (int i = 0; i < Lists.length; i++)

{

if (strcmp(Lists.elem[i].name, name) == 0)

{

return i + 1;

}

}

return ERROR;

}

status RemoveALGraph(LISTS &Lists, char name[])

{

int result = LocateALGraph(Lists, name);

if (!result)

{

return ERROR;

}

else

{

Lists.length--;

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

{

Lists.elem[i] = Lists.elem[i + 1];

}

DestroyGraph(Lists.elem[Lists.length].G);

}

return OK;

}