

**课 程 实 验 报 告**

**课程名称： 数 据 结 构 实 验**

**专业班级： 计算机卓越工程师**

**学 号： U201514482**

**姓 名： 魏硕**

**指导教师： 许贵平**

**报告日期： 2016年9月29日 \_\_ \_\_**

**计算机科学与技术学院**

目录

[1基于顺序存储结构实现线性表的基本运算 4](#_Toc469001567)

[1.1 实验目的 4](#_Toc469001568)

[1.1.1 实验目标 4](#_Toc469001569)

[1.1.2 问题描述 4](#_Toc469001570)

[1.2 线性表演示系统设计 4](#_Toc469001571)

[1.2.1 系统总体设计 4](#_Toc469001572)

[1.2.2 线性表有关定义 5](#_Toc469001573)

[1.2.3 系统详细设计 7](#_Toc469001574)

[1.3 线性表演示系统实现与测试 11](#_Toc469001575)

[1.3.1 系统实现 11](#_Toc469001576)

[1.3.2 系统测试 12](#_Toc469001577)

[1.4 实验小结 13](#_Toc469001578)

[2 基于链式实现线性表的基本运算 14](#_Toc469001579)

[2.1实验目的 14](#_Toc469001580)

[2.1.1实验目标 14](#_Toc469001581)

[2.1.2 问题描述 14](#_Toc469001582)

[2.2 线性表演示系统设计 14](#_Toc469001583)

[2.2.1 系统总体设计 14](#_Toc469001584)

[2.2.2 有关常量和类型定义 15](#_Toc469001585)

[2.2.3 算法详细设计 17](#_Toc469001586)

[2.3 线性表演示系统实现与测试 23](#_Toc469001587)

[2.3.1 系统实现 23](#_Toc469001588)

[2.3.2 系统测试 23](#_Toc469001589)

[2.4 实验小结 25](#_Toc469001590)

[3基于二叉链表实现二叉树的基本运算 26](#_Toc469001591)

[3.1 实验目的 26](#_Toc469001592)

[3.1.1 实验目标 26](#_Toc469001593)

[3.1.2 问题描述 26](#_Toc469001594)

[3.2.1 系统总体设计 26](#_Toc469001595)

[3.2.2 有关常量和类型定义 28](#_Toc469001596)

[3.2.3 算法设计 30](#_Toc469001597)

[3.3 二叉树演示系统实现与测试 35](#_Toc469001598)

[3.3.1 系统实现 35](#_Toc469001599)

[3.3.2 系统测试 36](#_Toc469001600)

[3.4 实验小结 40](#_Toc469001601)

[4基于邻接表的图实现 41](#_Toc469001602)

[4.1 实验目的 41](#_Toc469001603)

[4.1.1 实验目标 41](#_Toc469001604)

[4.1.2 问题描述 41](#_Toc469001605)

[4.2.1 系统总体设计 41](#_Toc469001606)

[4.2.2 有关常量和类型定义 43](#_Toc469001607)

[4.2.3 算法设计 45](#_Toc469001608)

[4.3 图的演示系统实现与测试 51](#_Toc469001609)

[4.3.1 系统实现 51](#_Toc469001610)

[3.3.2 系统测试 52](#_Toc469001611)

[3.4 实验小结 56](#_Toc469001612)

[参考文献 57](#_Toc469001613)

[指导教师评定意见 58](#_Toc469001614)

[附录A 基于顺序存储结构线性表实现的源程序 59](#_Toc469001615)

[附录B 基于链式存储结构线性表实现的源程序 69](#_Toc469001616)

[附录C 基于二叉链表二叉树实现的源程序 81](#_Toc469001617)

[附录D 基于邻接表图实现的源程序 104](#_Toc469001618)

# 1基于顺序存储结构实现线性表的基本运算

* 1. 实验目的
     1. 实验目标

（1）加深对线性表的概念、基本运算的理解；

（2）熟练掌握线性表的逻辑结构与物理结构的关系；

（3）能熟练掌握线性表的基本运算的实现方法并灵活地扩展应用；

（4）通过实验正确认识学习中的不足并加以改正。

1.1.2 问题描述

基于顺序存储结构实现线性表的十二个基本运算（建空表，清空表，添加删除元素等等，其中比较函数做了适当简化，变为寻找与目标值相等的第一个结点的函数），并根据实际需要适当添加相应的辅助函数（比如新增了文件存储和大量写入数据的函数）。

要求做成一个简易的文本界面便于功能的检查和使用。形成良好的用户交互菜单，保证程序质量，提高程序容错性和健壮性。

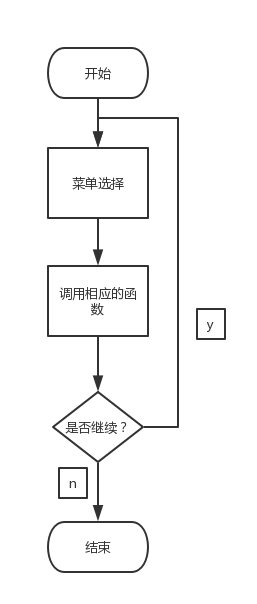
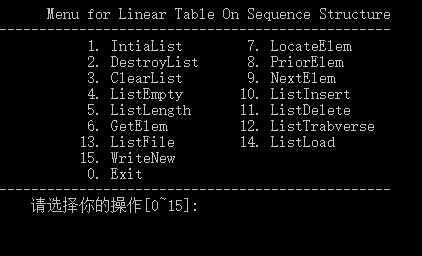
1.2 线性表演示系统设计

1.2.1 系统总体设计

本系统主要存储运算结构为一个顺序存储的线性表，其中数据类型为整型。

系统下的子菜单（即具体的实现函数）有：建立空表、销毁表、清空表、判空表、求表长、获取数据元素、查找数据元素、获得前驱、获得后继、创建线性表、插入数据元素、删除数据元素、表的遍历。并额外增加了文件的读入，写出以及大量数据的增加。

本系统提供一个简易的文本界面，在输入相应的数字后调用相应的函数，并执行下一步操作。具体菜单图及流程图如下：



1.2.2 线性表有关定义

（1）数据元素类型的定义：

typedef int status;

typedef int ElemType;

（2）有关常量的定义：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

（3）顺序表的定义：

typedef struct { //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

由上可知顺序表主要包括三部分，即数据的指针，当前长度及可用的最大长度，操作时可视数据的指针为一个数组，每次操作前都要确认数组当前长度是否超过了最大长度，如若超过，则应补加。

（4）基本运算定义

依据最小完备性和常用性相结合的原则，以函数形式定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等12种基本运算，具体运算功能定义如下。

⑴初始化表：函数名称是InitaList(L)；初始条件是线性表L不存在已存在；操作结果是构造一个空的线性表。

⑵销毁表：函数名称是DestroyList(L)；初始条件是线性表L已存在；操作结果是销毁线性表L。

⑶清空表：函数名称是ClearList(L)；初始条件是线性表L已存在；操作结果是将L重置为空表。

⑷判定空表：函数名称是ListEmpty(L)；初始条件是线性表L已存在；操作结果是若L为空表则返回TRUE,否则返回FALSE。

⑸求表长：函数名称是ListLength(L)；初始条件是线性表已存在；操作结果是返回L中数据元素的个数。

⑹获得元素：函数名称是GetElem(L,i,e)；初始条件是线性表已存在，1≤i≤ListLength(L)；操作结果是用e返回L中第i个数据元素的值。

⑺查找元素：函数名称是LocateElem(L,e,compare())；初始条件是线性表已存在；操作结果是返回L中第1个与e满足关系compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为0。

⑻获得前驱：函数名称是PriorElem(L,cur\_e,pre\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是第一个，则用pre\_e返回它的前驱，否则操作失败，pre\_e无定义。

⑼获得后继：函数名称是NextElem(L,cur\_e,next\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是最后一个，则用next\_e返回它的后继，否则操作失败，next\_e无定义。

⑽插入元素：函数名称是ListInsert(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。

⑾删除元素：函数名称是ListDelete(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。

⑿遍历表：函数名称是ListTraverse(L,visit())，初始条件是线性表L已存在；操作结果是依次对L的每个数据元素调用函数visit()。

1.2.3 系统详细设计

系统框架为主菜单，但是详细的实现主要依靠函数来实现，现对主要的几个函数做具体的结构和算法思路说明：

1. ClearList(SqList \*L)

**函数输入：**链表指针

**函数输出：**完成状态（正常执行返回OK）

**算法描述：**具体操作就是将当前长度赋为0即可，因为下一步加入数据时就会从0开始，如果原来有数据就会加以覆盖，此即为清空。摧毁也较为类似，不过还要加一个释放原有空间的操作。

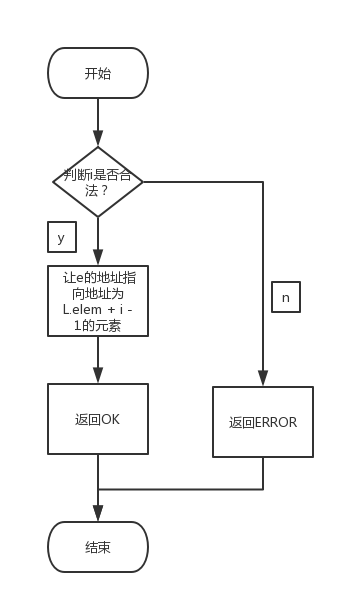
**时间复杂度：**T（n）=O(n)

1. GetElem(SqList L,int i,ElemType \*e)

**函数输入**：链表指针，查询元素位置，查询元素地址

**函数输出：**完成状态（正常执行返回OK）

**算法描述：**需要注意的是数组元素是从0开始存储的，第i个元素的地址应该是首地址加上i-1个单位，而不是i。并且每次开始应先看i是不是在指定的范围内，不然会返回一个什么都没有的地址，出现错误。



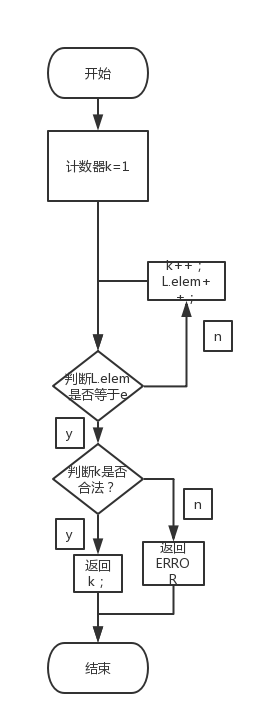
**时间复杂度：**T（n）=O(1)

1. LocateElem(SqList L, ElemType e)

**函数输入：**链表指针，查询数据的值

**函数输出：**查询数据的位置

**算法描述：**寻找并返回与给定值相等的位置，首先必须进行遍历，遍历的结束条件为计数器i>长度n，所有可能遍历次数取平均值，结果为n/2，复杂度为O（n）。寻找到后并加以返回即可。流程图如下：



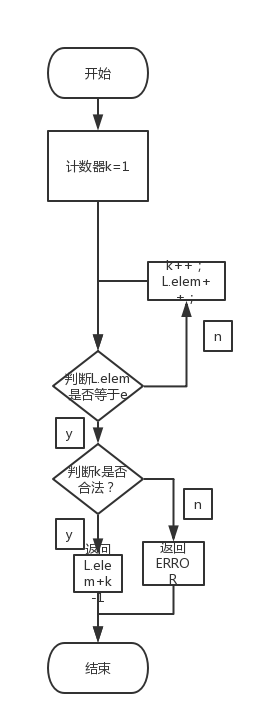
**时间复杂度**：T（n）=O（n）

1. PriorElem(SqList L, ElemType cur\_e, ElemType \*pre)

**函数输入**：链表指针，查询元素的值，前序元素的地址

**函数输出**：完成状态（正常执行返回OK）

**算法描述：**具体实现大致同上，不过返回值变成了前一个的数据的值。



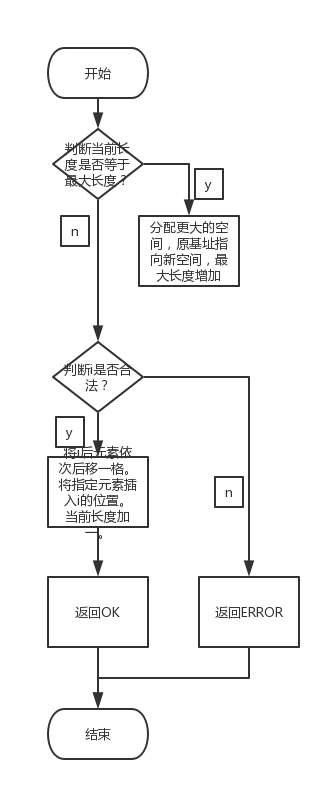
**时间复杂度**：O（T）=O（n）

1. ListInsert(SqList \*L, int i, ElemType e)

**函数输入：**链表指针，插入元素位置，插入元素的值

**函数输出**：完成状态（正常执行返回OK）

**算法描述**：首先必须看当前长度是否等于最大长度，不然插入后就超界了。当等于最大元素后，需要重新分配存储空间，并将基址指向新的空间。其次需要判断i是否合法。最后由于是顺序存储，插入前必须将插入位置后面的元素依次后移一个，空出需要插入的位置，再插入。当前长度加一即可。



时间复杂度：O（T）=O（n）

1. ListTrabverse(SqList L)

函数输入：链表指针

函数输出：完成状态（正常执行返回OK）

算法描述：先设定值为一的计数器，然后输出一个就加一，直至大于当前长度，在这个过程中依次将地址增加，输出当前地址所指的所有的数据即可。为了用户方便，应适当使用空格换行。

时间复杂度：O（T）=O（n）

1.3 线性表演示系统实现与测试

1.3.1 系统实现

电脑系统：windows10 专业版

编程环境：Dev c++

编程语言：c语言

函数清单及其调用关系如下：

status IntiaList(SqList \*L);// 创建链表

status DestroyList(SqList \*L);//摧毁链表

status ClearList(SqList \*L);//清空链表

status ListEmpty(SqList L);//判空表

status ListLength(SqList L);//求表长

status GetElem(SqList L, int i, ElemType \*e);//获取元素

status LocateElem(SqList L, ElemType e); //简化过 定位元素

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

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

status ListInsert(SqList \* L,int i,ElemType e);//插入元素

status ListDelete(SqList \* L, int i, ElemType \* e);//删除元素

status ListTrabverse(SqList L); //简化过 遍历链表

status load\_List(SqList \*L);// 从文件读出

status Listfile(SqList \*L);//保存到文件

1.3.2 系统测试

表1-1 线性表算法测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 用例1 | 1 | 线性表创建成功 |  |
| 用例2 | 2 | 线性表销毁成功 |  |
| 用例3 | 3 | 线性表清空成功 |  |
| 用例4 | 4 | 线性表已清空 |  |
| 用例5 | 5 | 线性表长度为0 |  |
| 用例6 | 6 | 表中第1个数据为1 |  |
| 用例7 | 7 | 表中第一个与查询数据相等的数据的位序为5 |  |
| 用例8 | 8 | 前驱为1 |  |
| 用例9 | 9 | 后驱为12 |  |
| 用例10 | 10 | 线性表插入成功 |  |
| 用例11 | 11 | 元素删除成功 |  |
| 用例12 | 12 | 输出结果 |  |
| 用例13 | 13 | 已保存到文件 |  |
| 用例14 | 14 | 已从文件导出 |  |
| 用例15 | 15 | 插入相应的数据 |  |

1.4 实验小结

1.在写代码之前最好有一个总体的结构的意识，即哪一块要实现哪些功能，一个功能分成哪几个函数。首先就要把每一块分布好，千万不要边想边做，可能会使前后不衔接，有可能写到后面发现前面的代码必须全部修改，不然就不可以用的现象。

2.变量命名要清晰，不要图方便用a，b，c等歧义较多的东西去命名变量，这样做的结果是写到后面不知道自己写的什么东西，容易混乱。其次，尽量不要有太多重复的名字，不然也很容易弄混。

3.代码风格也十分重要，不要自己想到哪里写到哪里，到最后括号是谁的都不知道，更不用说优先级之类的东西。函数前面写清楚变量的意义，要做的事，不然都不知道自己写的什么。还有一个教训就是尽量单一出口，东西多了，这个地方要是出错了根本看不出来，可能你要算的东西还没有算完它就返回了，你也不知道，检查也检查不出来。

4.要考虑到用户可能的输入情况并仔细分析每一种情况，寻找好的解决办法，以便于设计比较完善的界面。其次，若不好统一，则给足够多的提示，不要忘记使用getchar（）读掉可能的空格，避免不必要的麻烦。

5.写函数时要想清楚函数传参，不要多了，也不要少，同时应该分析清楚到底是参数本身还是传参数的地址。

# 基于链式实现线性表的基本运算

2.1实验目的

2.1.1实验目标

（1）加深对线性表的概念、基本运算的理解；

（2）熟练掌握线性表的逻辑结构与物理结构的关系；

（3）能熟练掌握线性表的基本运算的实现方法并灵活地扩展应用；

（4）通过实验正确认识学习中的不足并加以改正。

2.1.2 问题描述

基于链式存储结构实现线性表的十二个基本运算（建空表，清空表，添加删除元素等等，其中比较函数做了适当简化，变为寻找与目标值相等的第一个结点的函数），并根据实际需要适当添加相应的辅助函数（比如新增了文件存储及读出函数）。

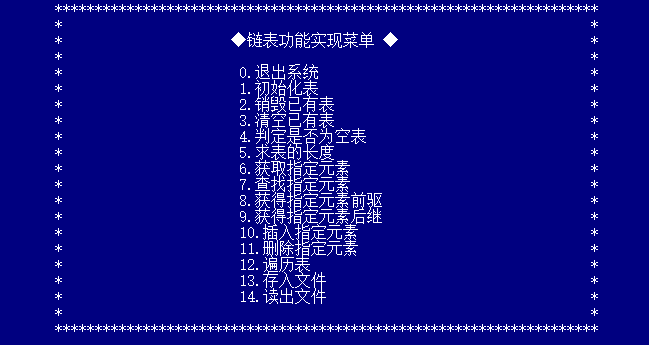
要求做成一个简易的文本界面便于功能的检查和使用。形成良好的用户交互菜单，保证程序质量，提高程序容错性和健壮性。

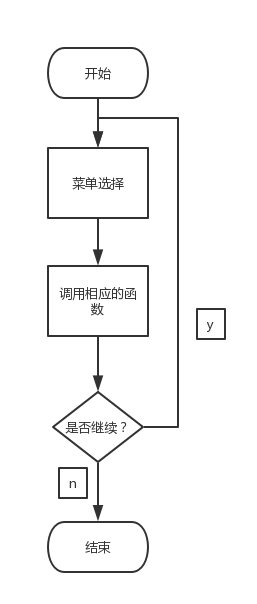
2.2 线性表演示系统设计

2.2.1 系统总体设计

本系统主要存储运算结构为一个链式存储的线性表，其中数据类型为整型。

系统下的子菜单（即具体的实现函数）有：建立空表、销毁表、清空表、判空表、求表长、获取数据元素、查找数据元素、获得前驱、获得后继、创建线性表、插入数据元素、删除数据元素、表的遍历。并额外增加了文件的读入，写出函数。

本系统提供一个简易的文本界面，在输入相应的数字后调用相应的函数，并执行下一步操作。具体菜单图及流程图如下：



2.2.2 有关常量和类型定义

（1）数据元素类型的定义：

typedef int status;

typedef int ElemType;

（2）有关常量的定义：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

（3）链表的定义：

typedef struct ListNode {

ElemType data;

struct ListNode \*next;

}ListNode, \*pListNode;

由上可知顺序表主要包括两部分，即数据域和指向下一个数据结点的指针。设立一个头结点，头结点的数据存放总的数据个数，指针指向第一个元素结点。而最后一个结点的指针应该赋为NULL，方便判断表尾。每次操作时新增或删除都应该自己分配和清空存储空间，因为链表的空间大小比较灵活，随取随用。

（4）基本运算定义

依据最小完备性和常用性相结合的原则，以函数形式定义了链表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等12种基本运算，具体运算功能定义如下。

⑴初始化表：函数名称是InitaList(L)；初始条件是线性表L不存在已存在；操作结果是构造一个空的线性表。

⑵销毁表：函数名称是DestroyList(L)；初始条件是线性表L已存在；操作结果是销毁线性表L。

⑶清空表：函数名称是ClearList(L)；初始条件是线性表L已存在；操作结果是将L重置为空表。

⑷判定空表：函数名称是ListEmpty(L)；初始条件是线性表L已存在；操作结果是若L为空表则返回TRUE,否则返回FALSE。

⑸求表长：函数名称是ListLength(L)；初始条件是线性表已存在；操作结果是返回L中数据元素的个数。

⑹获得元素：函数名称是GetElem(L,i,e)；初始条件是线性表已存在，1≤i≤ListLength(L)；操作结果是用e返回L中第i个数据元素的值。

⑺查找元素：函数名称是LocateElem(L,e)；初始条件是线性表已存在；操作结果是返回L中第1个与e满足关系相等关系的数据元素的位序，若这样的数据元素不存在，则返回值为0。

⑻获得前驱：函数名称是PriorElem(L,cur\_e,pre\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是第一个，则用pre\_e返回它的前驱，否则操作失败，pre\_e无定义。

⑼获得后继：函数名称是NextElem(L,cur\_e,next\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是最后一个，则用next\_e返回它的后继，否则操作失败，next\_e无定义。

⑽插入元素：函数名称是ListInsert(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。

⑾删除元素：函数名称是ListDelete(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。

⑿遍历表：函数名称是ListTraverse(L,visit())，初始条件是线性表L已存在；操作结果是依次对L的每个数据元素调用函数visit()。

2.2.3 算法详细设计

（1）IntiaList(ListNode \*\*L)

**函数输入**：链表指针

**函数输出**：执行状态（正常返回OK）

**算法描述：**即先为头结点分配存储空间，并且将头指针的数据赋值为0，头指针的指针域赋值为NULL，完成初始化操作。

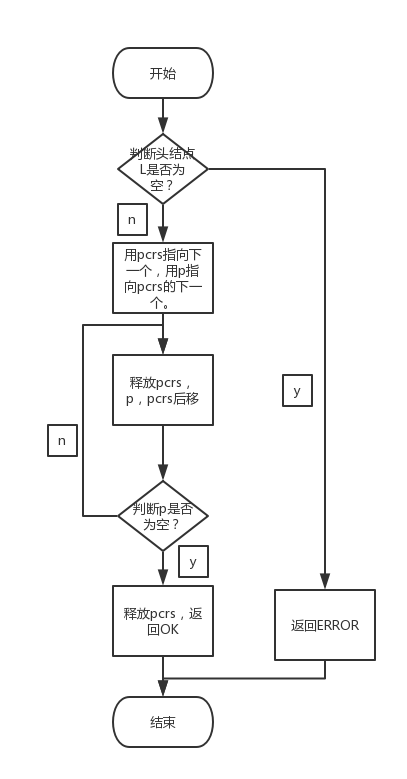
**时间复杂度分析**：O（1）

（2）ClearList(ListNode \*L)

**函数输入**：链表指针

**函数输出**：执行状态（正常返回OK）

**算法描述：**先判断表头是否存在，不存在则出错，存在就进行遍历，将不是头指针的结点都释放掉，并且将头指针的数据赋值为0。



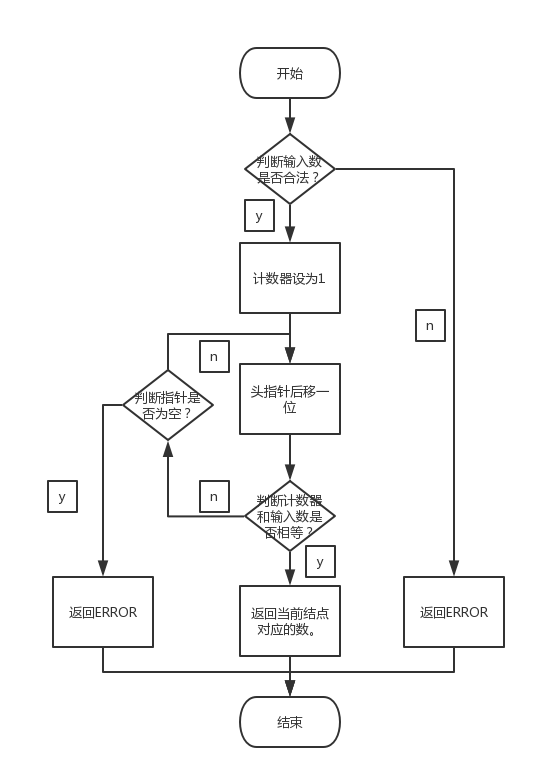
**时间复杂度：**O（n）

（3）GetElem(ListNode \* L,int i,ElemType \*e)

**函数输入**：链表指针，查询元素位置，位置的地址

**函数输出**：执行状态（正常返回OK）

**算法描述：**与顺序存储不同，知道结点的位置后无法直接访问，只能从头结点开始遍历。所以应该先将计数器置为0，然后指针往后走一位就加一，直到与传入数值相同，返回当前结点的数据。



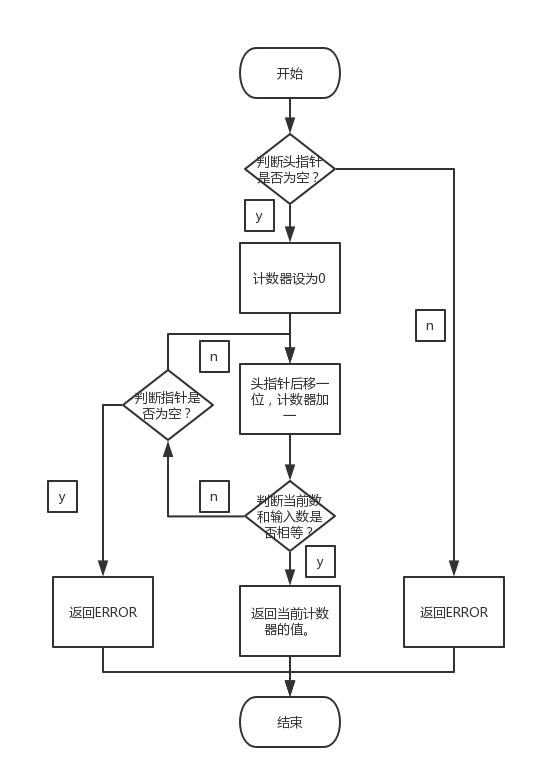
**时间复杂度：**O（n）

（4）LocateElem(ListNode \* L, ElemType e)

**函数输入**：链表指针，查询元素的值

**函数输出**：元素位置

**算法描述**：寻找并返回与给定值相等的位置，首先必须进行遍历，遍历的结束条件为计数器i>长度n，所有可能遍历次数取平均值，结果为n/2，复杂度为O（n）。寻找到后并加以返回即可。流程图如下：



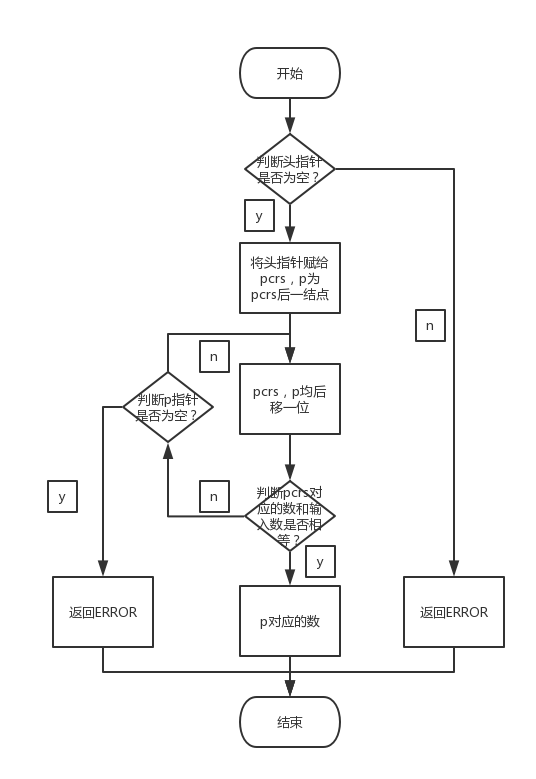
时间复杂度：O（n）

（5）PriorElem(ListNode \* L, ElemType e, ElemType\* pre)

**函数输入**：链表指针，查询元素位置，位置的地址

**函数输出**：执行状态（正常返回OK）

**算法描述：**由于链表的存储特点，不能访问后返回上一个结点，所以必须设置两个指针，一个跑的比另一个快一个结点。然后进行遍历，直到后一个指针指向的结点的数据等于指定的数据，然后返回前一个指针指向结点的数据。如果一直到文件尾都没有发现就说明没有。



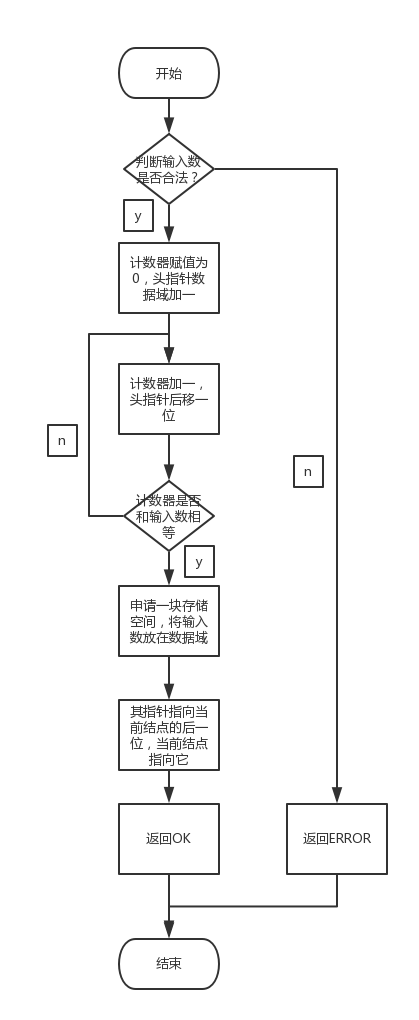
**时间复杂度：**  O（n）

（6）ListInsert(ListNode \*\*L, int i, ElemType e)

**函数输入**：链表指针的指针，插入元素位置，元素的值

**函数输出**：执行状态（正常返回OK）

**算法描述：**寻找插入位置时必须使用遍历，但好处是不需要将后面的结点存储位置依次后移。从头结点开始遍历。所以应该先将计数器置为0，然后指针往后走一位就加一，直到与传入数值相同。然后申请一块存储空间，数据为要插入的数据，让它指向插入位置结点的下一个结点，然后插入位置结点指向它即可。



时间复杂度：O（n）

2.3 线性表演示系统实现与测试

2.3.1 系统实现

电脑系统：windows10 专业版

编程环境：Dev c++

编程语言：c语言

函数原型如下：

status IntiaList(ListNode \*\*L);// 建立链表

status DestroyList(ListNode \*L); //摧毁链表

status ClearList(ListNode \*L);//清空链表

status ListEmpty(ListNode \* L);//判空链表

status ListLength(ListNode \* L);//求链表长度

status GetElem(ListNode \*L, int i, ElemType \*e); //获取指定元素

status LocateElem(ListNode \* L, ElemType e);//指定元素位置

status PriorElem(ListNode \* L, ElemType e, ElemType\* pre);//指定元素前驱

status NextElem(ListNode \* L, ElemType cur\_e, ElemType\* next\_e);//指定元素后继

status ListInsert(ListNode \*\*L, int i, ElemType e);//插入指定元素

status ListDelete(ListNode \*\*L, int i, ElemType\* e);//删除指定元素

status ListTrabverse(ListNode \* L);//遍历链表

status SaveFile(ListNode \* L);//保存文件

status LoadFile(ListNode \*\*L);//读出文件

调用插入函数

status creat\_kindlist(ListNode \*\*L);//大量写入

2.3.2 系统测试

表2-1 线性表算法测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 用例1 | 1 | 线性表创建成功 | GYM38FRFKT@Z8X)TTDCZ]HQ |
| 用例2 | 2 | 线性表销毁成功 | {WYOX543%Z`KGW}U2C)GK2I |
| 用例3 | 3 | 线性表清空成功 | 4[X1LF@1O2BKR@8[%TKE)`K |
| 用例4 | 4 | 线性表已清空 | QD8(TTV(U2}5F616K2H${2J |
| 用例5 | 5 | 线性表长度为0 | T)U6N%R@B$6TV_P4$GF_CFV |
| 用例6 | 6 | 表中第2个数据为2 | PTN7MP990TU$]W)A]Q]NOKX |
| 用例7 | 7 | 表中第一个与查询数据相等的数据的位序为6 | @60H72WU75WEL0[I5DO1[BY |
| 用例8 | 8 | 前驱为5 | ]YAWB]2FB99I@HDQK(ENK8F |
| 用例9 | 9 | 线性表插入成功 | JS[L6NK91Q87QA]WZEOIMNR |

2.4 实验小结

本次实验主要的问题出在销毁上，即表被清空后无法被销毁，原因是清空时将头结点的空间也释放了，导致了销毁的操作无法进行。还有一个问题就是函数的传参，有一些只需传递链表的头指针就可以了，有一些必须传递头指针的指针，不然参数的改变就无效，这是与c语言的特性息息相关的。其次是插入操作的问题，最开始在空表时无法插入（因为没有具体元素，无法插入在谁的前面），但是这样就无法输入数据，所以对这个函数做了自己的修改。在写代码前应该将题目要求看清楚，思路要清晰，且一些细节要注意到，不然容易导致错误。

链表实验完成最大的缺点在于没有将数据元素多元化，链表的数据元素可以有多个，也可以有多种，甚至可以实现十字链表或者多重链表。其次是没有联系实际，让代码具有实用性，主要还是因为数据元素单一。

优点在于通过自己的思考想出了一些新的函数的操作，比如清空和销毁可以用递归来实现（但是没什么必要），并且重新认识了链表式的存储结构，对其优缺点也更加了然于心。

虽然有些问题，收获还是挺多的。

# 3基于二叉链表实现二叉树的基本运算

3.1 实验目的

3.1.1 实验目标

（1）加深对二叉树的概念、基本运算的理解；

（2）熟练掌握二叉树的逻辑结构与物理结构的关系；

（3）以二叉链表作为物理结构，熟练掌握二叉树基本运算的实现。

3.1.2 问题描述

基于二叉链表实现二叉树的二十个个基本运算（建立树，摧毁树，查找结点，插入树等等，其中查找结点，遍历结点等函数做了适当简化，变为寻找与目标值相等的第一个结点的函数），并根据实际需要适当添加相应的辅助函数（比如新增了文件存储函数）。

要求做成一个简易的文本界面便于功能的检查和使用。形成良好的用户交互菜单，保证程序质量，提高程序容错性和健壮性。

**3.2 二叉树演示系统设计**

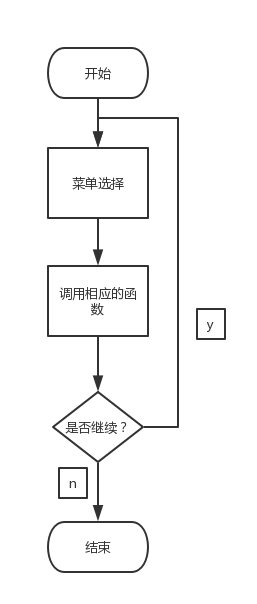
3.2.1 系统总体设计

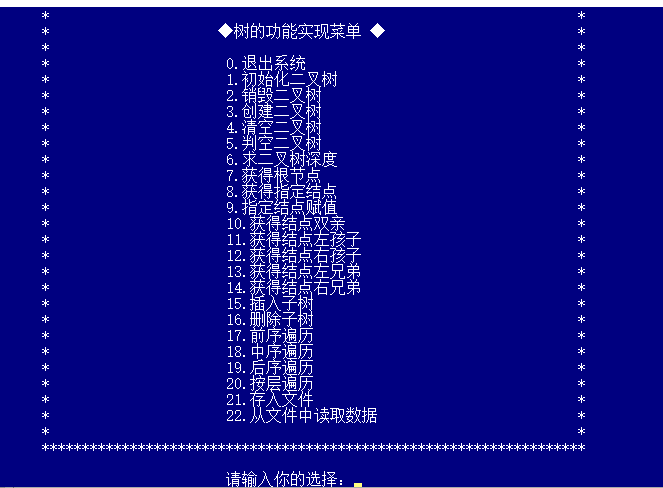
本系统主要存储运算结构为一个存储树的根结点指针的数组，该数组为包含20棵树的森林。树结点包含左右子树的指针，以及数据域。数据域中含有结点编号及名字信息。

为完成树的非递归算法，额外新增栈的基本操作（未实现全部操作，只写了需要用的几个函数）。栈的存储结构为一个数组，其中包含基地址，top为当前所在位置的编号，初始大小为20，与森林大小匹配。

系统下的子菜单（即具体的实现函数）有：初始二叉树、销毁二叉树、创建二叉树、清空二叉树、判空二叉树、求二叉树深度、获得根结点、获得指定结点、获得双亲结点、获得左兄弟结点等20个子功能，并额外增加了文件的读入与写出功能。

本系统提供一个简易的文本界面，在输入相应的数字后调用相应的函数，并执行下一步操作。具体菜单图及流程图如下：





3.2.2 有关常量和类型定义

（1）数据元素类型的定义：

typedef int status;

typedef struct Data TElemType;

struct Data {

int number;//结点的编号

char name[20];//结点存的名字

};

数据域包含结点编号和名字。

（2）有关树的定义：

typedef struct BiTNode {

TElemType data;

struct BiTNode \*lchild,\*rchild;

}BiTNode,\*BiTree;

树结点包含数据域和左右子树的指针，每棵树的头结点指针放在森林数组中。

（3）有关栈的定义：

typedef struct Stack {

BiTree \*base;

int top;

int stacksize;

}\*Stack;

栈用数组实现，数据元素主要包括数组的基地址，所处的当前位置和最大的容量。

（4）基本运算定义

⑴初始化二叉树：函数名称是InitBiTree(T)；初始条件是二叉树T不存在；操作结果是构造空二叉树T。

⑵销毁二叉树：树函数名称是DestroyBiTree(T)；初始条件是二叉树T已存在；操作结果是销毁二叉树T。

⑶创建二叉树：函数名称是CreateBiTree(T,definition)；初始条件是definition 给出二叉树T的定义；操作结果是按definition构造二叉树T。

⑷清空二叉树：函数名称是ClearBiTree (T)；初始条件是二叉树T存在； 操作结果是将二叉树T清空。

⑸判定空二叉树：函数名称是BiTreeEmpty(T)；初始条件是二叉树T存在；操作结果是若T为空二叉树则返回TRUE，否则返回FALSE。

⑹求二叉树深度：函数名称是BiTreeDepth(T)；初始条件是二叉树T存在；操作结果是返回T的深度。

⑺获得根结点：函数名称是Root(T)；初始条件是二叉树T已存在；操作结果是返回T的根。

⑻获得结点：函数名称是Value(T,e)；初始条件是二叉树T已存在，e是T中的某个结点；操作结果是返回e的值。

⑼结点赋值：函数名称是Assign(T,&e,value)；初始条件是二叉树T已存在，e是T中的某个结点；操作结果是结点e赋值为value。

⑽获得双亲结点：函数名称是Parent(T,e) ；初始条件是二叉树T已存在，e是T中的某个结点；操作结果是若e是T的非根结点，则返回它的双亲结点指针，否则返回NULL。

⑾获得左孩子结点：函数名称是LeftChild(T,e)；初始条件是二叉树T存在，e是T中某个节点；操作结果是返回e的左孩子结点指针。若e无左孩子，则返回NULL。

⑿获得右孩子结点：函数名称是RightChild(T,e)；初始条件是二叉树T已存在，e是T中某个结点；操作结果是返回e的右孩子结点指针。若e无右孩子，则返回NULL。

⒀获得左兄弟结点：函数名称是LeftSibling(T,e)；初始条件是二叉树T存在，e是T中某个结点；操作结果是返回e的左兄弟结点指针。若e是T的左孩子或者无左兄弟，则返回NULL。

⒁获得右兄弟结点：函数名称是RightSibling(T,e)；初始条件是二叉树T已存在，e是T中某个结点；操作结果是返回e的右兄弟结点指针。若e是T的右孩子或者无有兄弟，则返回NULL。

⒂插入子树：函数名称是InsertChild(T,p,LR,c)；初始条件是二叉树T存在，p指向T中的某个结点，LR为0或1，,非空二叉树c与T不相交且右子树为空；操作结果是根据LR为0或者1，插入c为T中p所指结点的左或右子树，p 所指结点的原有左子树或右子树则为c的右子树。

⒃删除子树：函数名称是DeleteChild(T.p.LR)；初始条件是二叉树T存在，p指向T中的某个结点，LR为0或1。 操作结果是根据LR为0或者1，删除c为T中p所指结点的左或右子树。

⒄前序遍历：函数名称是PreOrderTraverse(T,Visit())；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果：先序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⒅中序遍历：函数名称是InOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果是中序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⒆后序遍历：函数名称是PostOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果是后序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⒇按层遍历：函数名称是LevelOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果是层序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

3.2.3 算法设计

（1）InitBiTree(void)

**算法输入**：无输入

**算法输出**：实现状态（完成返回OK）

**算法处理步骤描述**：

将森林数组的每个指针置为空，成功则返回OK

**算法时间复杂度分析**：T（n）=O(n)

（2）DestroyBiTree(BiTree \*T)

**算法输入**：树的根结点的指针

**算法输出**：实现状态（完成返回OK）

**算法处理步骤描述**：

1. 首先判断传递的参数是不是空指针，为空指针直接返回。
2. 再判断左子树为不为空，不为空，则递归调用销毁函数，为空就不管；再判断右子树为不为空，不为空，则递归调用销毁函数，为空就不管。
3. 最后销毁掉最开始传入的指针，将头指针置为空。（相当于后序的递归销毁树）

**算法时间复杂度分析**：T（n）=O(n)

（3）CreateBiTree(BiTree T)

**算法输入**：树的根结点

**算法输出**：实现状态（完成返回OK）

**算法处理步骤描述**：

1. 首先判断创建的结点是否为空结点。若为空结点，直接将该节点赋为空，然后退出函数；否则，进入下一步。
2. 为结点分配空间，要求输入结点的具体信息。
3. 递归创建左子树，并将创建结果赋给左子树（因为传过去的是指针，不是指针的指针，必须有这个赋值）；然后递归创建右子树，并将创建结果赋给右子树。（相当于先序遍历创建树）

**算法时间复杂度分析**：T（n）=O(n)

（4）BiTreeDepth(BiTree T)

**算法输入**：树的根结点

**算法输出**：树的深度

**算法处理步骤描述**：

1. 首先判断传入结点是否为空。为空，直接返回0；不为空，进入下一步操作。
2. 递归计算左子树的深度，递归计算右子树的深度，然后比较两者的值，将较大的加一后返回。

**算法时间复杂度分析**：T（n）=O(n)

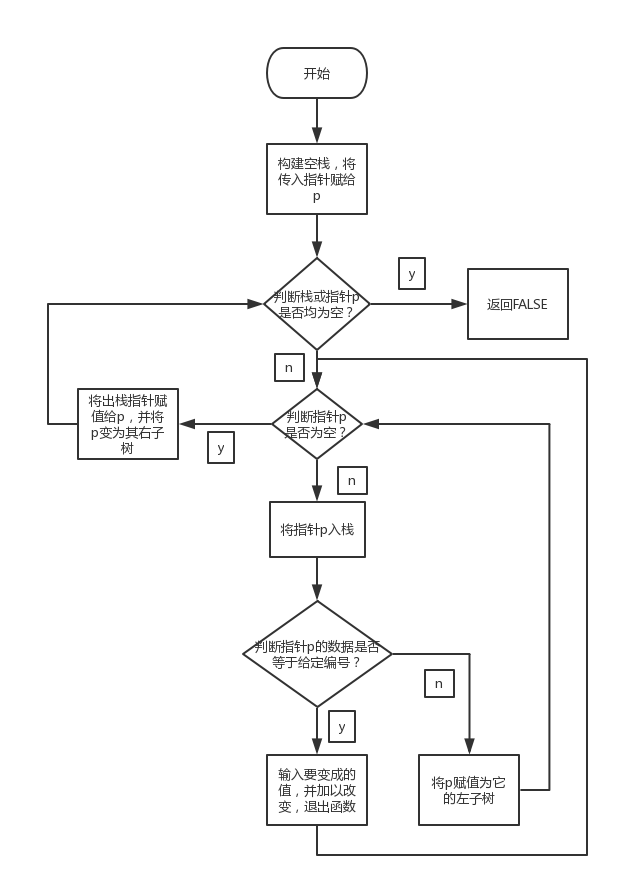
（5）Assign(BiTree \*T,int e)

**算法输入**：需要改变的树的根结点，以及需要改变结点编号

**函数调用**：构建栈，判空栈，入栈和出栈函数

**算法输出**：实现状态（完成返回OK）

**算法处理步骤描述**：



相当于先序遍历寻找结点。

**算法时间复杂度分析**：T（n）=O(n)

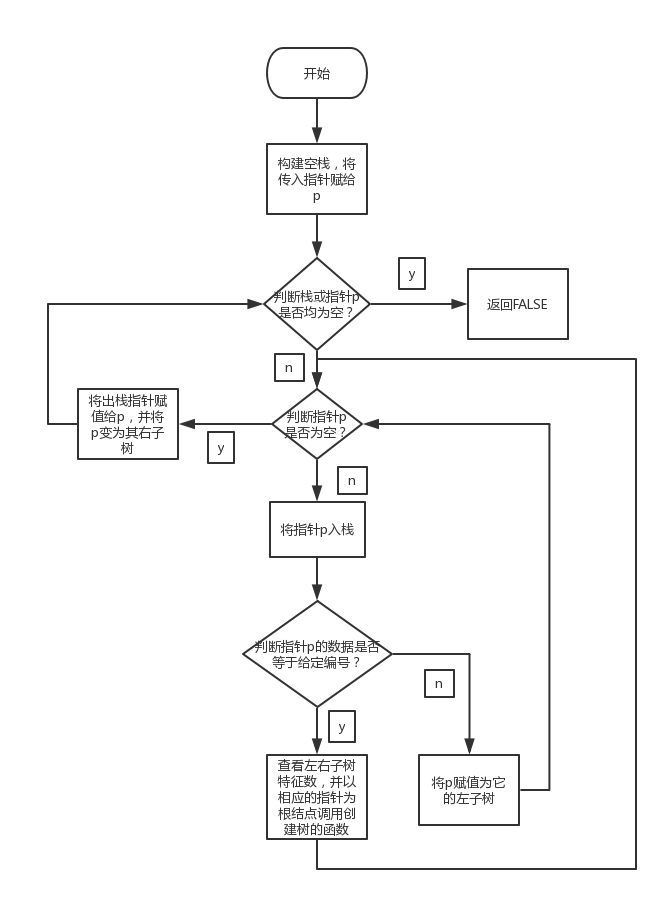
（6）InsertChild(BiTree \*T,int e,int LR)

**算法输入**：树根结点的指针，需要改变的结点编号，左右子树的判定值

**函数调用**：构建栈，判空栈，入栈，出栈函数以及创建树函数

**算法输出**：实现状态（完成返回OK）

**算法处理步骤描述**：



**算法时间复杂度分析**：T（n）=O(n)

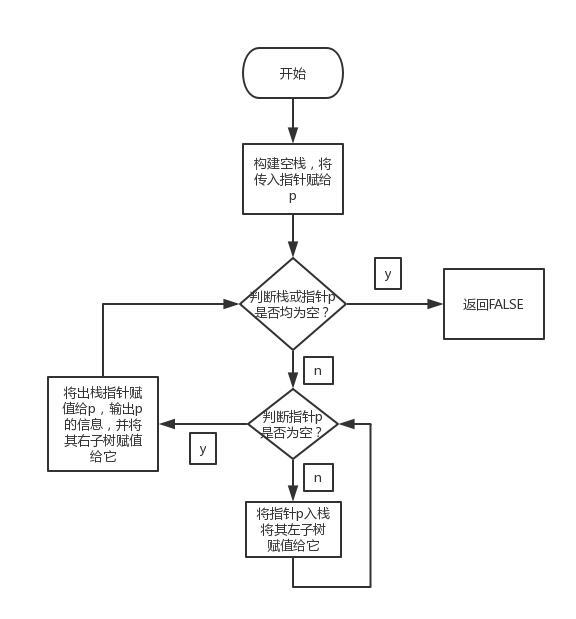
（7）InOrderTraverse(BiTree T)

**算法输入**：树的根结点

**函数调用**：构建栈，判空栈，入栈和出栈函数

**算法输出**：实现状态（完成返回OK）

**算法处理步骤描述**：



**算法时间复杂度分析**：T（n）=O(n)

（8）LevelTraverse(BiTree T)

**算法输入**：树的根结点

**函数调用**：求树的深度的函数以及分层打印结点的函数

**算法输出**：实现状态（完成返回OK）

**算法处理步骤描述**：

1. 首先调用求树的深度的函数求出树的深度。
2. 调用循环，循环次数为树的深度，调用输出每层结点数据函数
3. PrintLevel函数：首先判断树结点为不为空以及level为不为一，不满足就退出。如果level等于一就输出结点信息。否则递归对其左右子树调用这个函数。

**算法时间复杂度分析**：T（n）=O(n)

3.3 二叉树演示系统实现与测试

3.3.1 系统实现

电脑系统：windows10 专业版

编程环境：Dev c++

编程语言：C语言

函数清单及调用关系：

//栈的操作 均无调用关系，无特殊说明

int InitStack(Stack S);//栈的初始化

int Push(Stack S, BiTree BT);//入栈

BiTree Pop(Stack S);//出栈

int IsStackEmpty(Stack S);//判空栈

//树的操作

int InitBiTree(void);//初始化森林 （递归算法）

int DestroyBiTree(BiTree \*T);//摧毁树 （递归算法）

BiTree CreateBiTree(BiTree T);//创建树 （递归算法）

int ClearBiTree(BiTree \*T);//清空树 （递归算法）

int BiTreeEmpty(BiTree T);//判断树是否为空

int BiTreeDepth(BiTree T);//求树的深度 （递归算法）

int Root(BiTree T);//返回根节点

BiTree Value(BiTree T,int e);//输入结点的编号，返回结点

调用了构建栈，判空栈，入栈和出栈函数构建栈，判空栈，入栈和出栈函数

int Assign(BiTree \*T,int e);//找到结点e，并改变它的值

调用了构建栈，判空栈，入栈和出栈函数

BiTree Parent(BiTree T,int e);//返回双亲结点

调用了构建栈，判空栈，入栈和出栈函数

BiTree LeftChild(BiTree T,int e);//返回左孩子结点

调用了构建栈，判空栈，入栈和出栈函数

BiTree RightChild(BiTree T,int e);//返回右孩子

调用了构建栈，判空栈，入栈和出栈函数

BiTree LeftSibling(BiTree T,int e);//返回左兄弟

调用了构建栈，判空栈，入栈和出栈函数

BiTree RightSibling(BiTree T,int e);//返回右兄弟

调用了构建栈，判空栈，入栈和出栈函数

int InsertChild(BiTree \*T,int e,int LR);//插入子树 ,LR为1则是左子树

调用了构建栈，判空栈，入栈和出栈函数以及创建树的函数

int DeleteChild(BiTree \*T,int e,int LR);//删除子树，LR为1则是左子树

调用了构建栈，判空栈，入栈和出栈函数以及删除树的函数

int PreOrderTraverse(BiTree T);//前序遍历（遍历算法）

int InOrderTraverse(BiTree T);//中序遍历

调用了构建栈，判空栈，入栈和出栈函数

int PostOrderTraverse(BiTree T);//后序遍历（遍历算法）

int LevelTraverse(BiTree T);//层次遍历

void PrintLevel(BiTree T,int level);//层次遍历的一部分（遍历算法）

3.3.2 系统测试

**1.创建树函数的测试**

表3-1测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 用例1 | 1 a  2 b  3 c | 创建成功 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\92{XL9[$}DJ9JVXG2WD[8OX.png  C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\FOZX@QXT44M`)QCP[N3@ZKX.png |
| 用例2 | Y(即为空二叉树) | 创建成功 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\B2V2YX0MS~_WX$O~8AVXC2X.png |

**2.指定结点赋值测试**

表3-2测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 用例1 | 已有结点：1 a  2 b  3 c  4 d  5 e  输入：3  3 f | 该节点的值修改成功 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\`$AKD0I@EQ`DV7@9MGBEHDE.png |
| 用例2 | 已有结点：1 a  2 b  3 c  4 d  5 e  输入： 6 | 该结点不存在 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\L[JD1HOYDR$9MCFJFX_E@}8.png |
| 用例3 | 无结点，为空树 | 该树为空树 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\9AT}$}@~6`M4M4U_L{I_KRB.png |

**3.获得结点左兄弟测试**

表3-3测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 用例1 | 已有结点：1 a  2 b  3 c  4 d  5 e  输入：3 | 2 b | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\SZS@}YBO$JT}J~VRP59Y%}1.png |
| 用例2 | 已有结点：1 a  2 b  3 c  4 d  5 e  输入： 1 | 该结点不存在 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\4$TP{7Y6@RD1XA{E8[[TE)Q.png |
| 用例3 | 已有结点：1 a  2 b  3 c  4 d  5 e  输入： 6 | 该结点不存在 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\XE7[~@2Y[5CH1AV)6{EPWBC.png |

**4.插入子树测试**

表3-4测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 用例1 | 已有结点：1 a  2 b  3 c  4 d  5 e  输入：2  1  6 f | 插入成功 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\]%C@@5S)_I6W[{YH312_L@B.png  C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\(WY6HG3J3H2{CHZ{K1WTMCP.png |
| 用例2 | 已有结点：1 a  2 b  3 c  4 d  5 e  输入： 7 | 该结点不存在 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\{91_5HGEBD~9@7NN%M8DPOB.png |
| 用例3 | 无结点，为空树 | 该树为空树 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\9AT}$}@~6`M4M4U_L{I_KRB.png |

**5.中序遍历测试**

表3-5测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 用例1 | 已有结点：1 a  2 b  3 c  4 d  5 e  6 f | 6 f  2 b  1 a  4 d  3 c  5 e | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\EV3@ED6`40T3_F9@J)KINV0.png |
| 用例2 | 已有结点：1 a  2 b  3 c  4 d  5 e | 2 b  1 a  4 d  3 c  5 e | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\EV{@7LXOURX$F)HBI~$KK1W.png |
| 用例3 | 无结点，为空树 | 该树为空树 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\9AT}$}@~6`M4M4U_L{I_KRB.png |

**6.层次遍历的测试**

表3-6测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 用例1 | 已有结点：1 a  2 b  3 c  4 d  5 e | 1 a  2 b  3 c  4 d  5 e | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\0WX3NM807IWCA9YJ1[JHWA5.png |
| 用例2 | 已有结点：1 a  2 b  3 c  4 d  5 e  6 f | 1 a  2 b  3 c  6 f  4 d  5 e | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\5Z5{X0U9~J$)NKL2FVKM5}Y.png |
| 用例3 | 无结点，为空树 | 该树为空树 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\9AT}$}@~6`M4M4U_L{I_KRB.png |

本次实验用例调试基本正常，每个功能能对正确用例做出正确反馈，对错误用例也能返回正确的结论。

3.4 实验小结

树的基本操作的实现，最重要的是想好树的结构，应该有哪几部分，子树与双亲结点的联系与区别，有子树和无子树的区别是什么。又该如何管理多棵树构成的森林，栈的操作的实现等等，只有把基础想好才能进行下一步的操作。

过程中遇到的主要的一个问题是，什么时候传根结点的指针，什么时候传指针的指针。解决是，如果对该链表有修改，不管是增加还是删除，必须传指针的指针（或者通过赋值达到延续的效果）；反之，没有删除操作，只有遍历，只需传根结点的指针。不论是栈的操作，还是树的操作，乃至对所有链表都是适用的。其次是递归操作的思路要清晰，写代码时要注意递归结束条件和对下一次递归所有情况的普遍适用性。还有一个问题是free函数的应用，在load函数中，如果子树为空树而直接free子树是无法完成的，因为与双亲的联系还在，即使free还是有乱码；必须是free（p->lchild）这样才能真实地将左子树置为空。

本次实验还存在的问题在于没能实现通过两种遍历唯一确定一棵树的文件读写操作，用空格容易导致空间的浪费和溢出情况。

# 4基于邻接表的图实现

4.1 实验目的

4.1.1 实验目标

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。

4.1.2 问题描述

基于邻接表实现图的十三个基本运算（建立图，摧毁图，查找结点，插入结点，插入弧等等，其中创建图，查找结点等函数做了适当简化和修改，将有些传入参数变成了在函数中输入参数），并根据实际需要适当添加相应的辅助函数（比如新增了文件存储和文件读写函数）。

要求做成一个简易的文本界面便于功能的检查和使用。形成良好的用户交互菜单，保证程序质量，提高程序容错性和健壮性。

**4.2 图的演示系统设计**

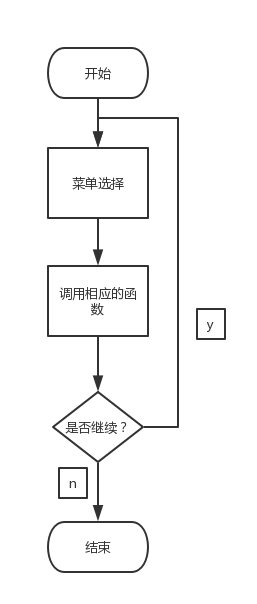
4.2.1 系统总体设计

本系统主要存储运算结构为一个存储图指针的数组，该数组可以同时操作20个图。图的结点信息包括图当前的结点数和弧数，图的种类，以及一个结点数组的基地址。结点的基本信息包括顶点的值和指向第一条依附该顶点的弧的指针。而弧的信息包括弧指向的顶点的位置，弧的权值以及指向下一条弧的指针。

为完成图的广度优先遍历，额外新增队列的基本操作（未实现全部操作，只写了需要用的几个函数）。队列的存储结构为一个数组，其中包含基地址，当前占用空间，最大空间，队头位置以及队尾位置。

系统下的子菜单（即具体的实现函数）有：造图，销毁图，返回结点值，增加顶点或弧，遍历等20个子功能，并额外增加了文件的读入与写出功能。

本系统提供一个简易的文本界面，在输入相应的数字后调用相应的函数，并执行下一步操作。具体菜单图及流程图如下：





4.2.2 有关常量和类型定义

（1）数据元素类型的定义：

typedef int status;

typedef char VertexType;

（2）有关图的定义：

typedef struct {//图信息

AdjList vertices;

int vexnum, arcnum; //图的当前顶点数和弧数

int kind; //图的种类标志

}aLGraph, \*ALGraph;

图结点包含当前定点数，图的种类和图的顶点数组的基地址。

typedef struct VNode { //结点信息

VertexType data;//顶点信息

ArcNode \*firstarc; //指向第一条依附该顶点的弧的指针

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

顶点包含顶点的值和指向第一条依附该顶点的弧的指针。

typedef struct ArcNode { //弧的信息

int adjvex; //该弧所指向的顶点的位置

struct ArcNode \*nextarc; //指向下一条弧的指针

int Weight; //用来记录网的权值

}ArcNode;

弧包含弧指向的顶点的位置，弧的权值以及指向下一条弧的指针。

（3）有关队列的定义：

typedef struct Queue {

int \*base;

int capacity; //最大容量

int front;

int rear;

int now; //当前个数

}\*Queue;

队列用数组实现，数据元素主要包括数组的基地址，当前容量，最大的容量，队头位置和队尾位置。

（4）基本运算定义

（1）CreateCraph(&G,V,VR)

初始条件：V是图的顶点集，VR是图的关系集。

操作结果：按V和VR的定义构造图G。

（2）DestroyCraph(&G)

初始条件：图G存在。

操作结果：销毁图G。

（3）LocateVex(G,u)

初始条件：图G存在，u和G中的顶点具有相同特征。

操作结果：若u在图G中存在，返回顶点u的位置信息，否则返回其它信息。

（4）FirstAdjVex(&G, v)

初始条件：图G存在，v是G的一个顶点。

操作结果：返回v的第一个邻接顶点，如果v没有邻接顶点，返回空。

（5）NextAdjVex(&G, v, w)

初始条件：图G存在，v是G的一个顶点,w是v的邻接顶点。

操作结果：返回v的（相对于w）下一个邻接顶点，如果w是最后一个邻接

顶点，返回空。

（6）InsertVex(&G,v)

初始条件：图G存在，v和G中的顶点具有相同特征。

操作结果：在图G中增加新顶点v。

（7）DeleteVex(&G,v)

初始条件：图G存在，v是G的一个顶点。

操作结果：在图G中删除顶点v和与v相关的弧。

（8）InsertArc(&G,v,w)

初始条件：图G存在，v、w是G的顶点。

操作结果：在图G中增加弧<v,w>，如果图G是无向图，还需要增加<w,v>。

（9）DeleteArc(&G,v,w)

初始条件：图G存在，v、w是G的顶点。

操作结果：在图G中删除弧<v,w>，如果图G是无向图，还需要删除<w,v>。

（10）DFSTraverse(G,visit())

初始条件：图G存在。

操作结果：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。

（11）BFSTraverse(G,visit())

初始条件：图G存在。

操作结果：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。

4.2.3 算法设计

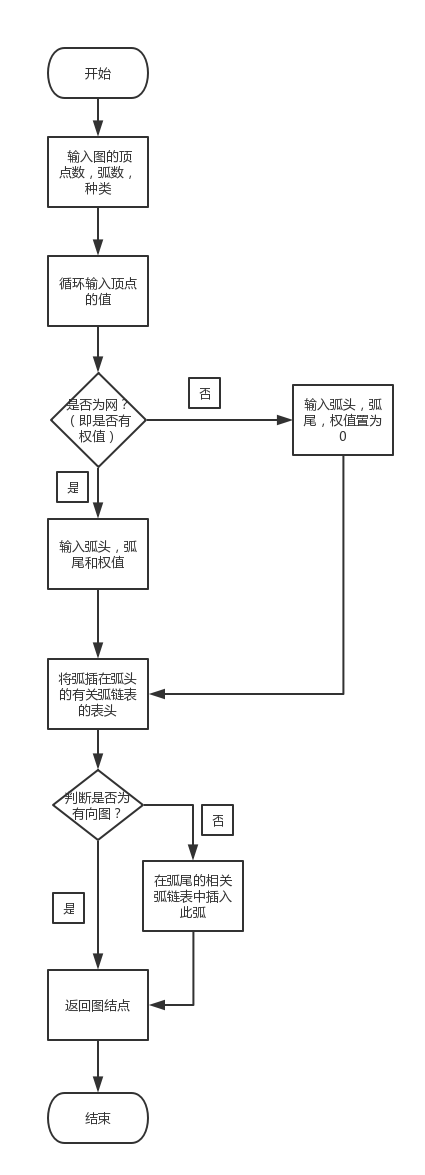
（1）CreateGraph(ALGraph G)

**算法输入**：需要创建的图的指针

**函数调用**：定位函数

**算法输出**：创建好的图的指针

**算法处理步骤描述**：



**算法时间复杂度分析**：T（n）=O(n)

（2）DestroyGraph(ALGraph G)

**算法输入**：需要摧毁的图的指针

**算法输出**：修改后指针

**算法处理步骤描述**：

1. 首先判断传递的参数是不是空指针，为空指针直接返回。
2. 从第一个节点开始，对每一个结点进行销毁，并沿着与其有关的弧的指针将每一个弧结点都销毁。
3. 最后销毁掉最开始传入的指针，将头指针置为空，然后返回头指针。

**算法时间复杂度分析**：T（n）=O(n)

（3）PutVex(ALGraph G, VertexType v, VertexType ch)

**算法输入**：需要改变的图的指针，需要改变的顶点的值，改变的值

**函数调用**：定位函数

**算法输出**：实现状态（完成返回OK）

**算法处理步骤描述**：

1. 首先定位需要修改的结点的值，看这个节点是不是图中的，如果是，找出其位置，如果不是，直接返回错误。
2. 依据返回的位置，将修改的值赋给相应结点的数据域，并且返回OK。

**算法时间复杂度分析**：T（n）=O(n)

（4）NextAdjVex(ALGraph G, VertexType v, VertexType ch)

**算法输入**：需要查找的图的指针，指定顶点的值，指定顶点的值

**函数调用**：定位函数

**算法输出**：实现状态（完成返回OK）

**算法处理步骤描述**：

1. 首先定位传入的两个结点的值的位置，如果两个结点有一个不在图中，就返回错误。
2. 然后沿着与指定结点的有关的弧的指针搜索，直到找到第二个指定结点。
3. 如果没有找到，返回错误；如果找到且其后继不为空，则返回其后继的值。

**算法时间复杂度分析**：T（n）=O(n)

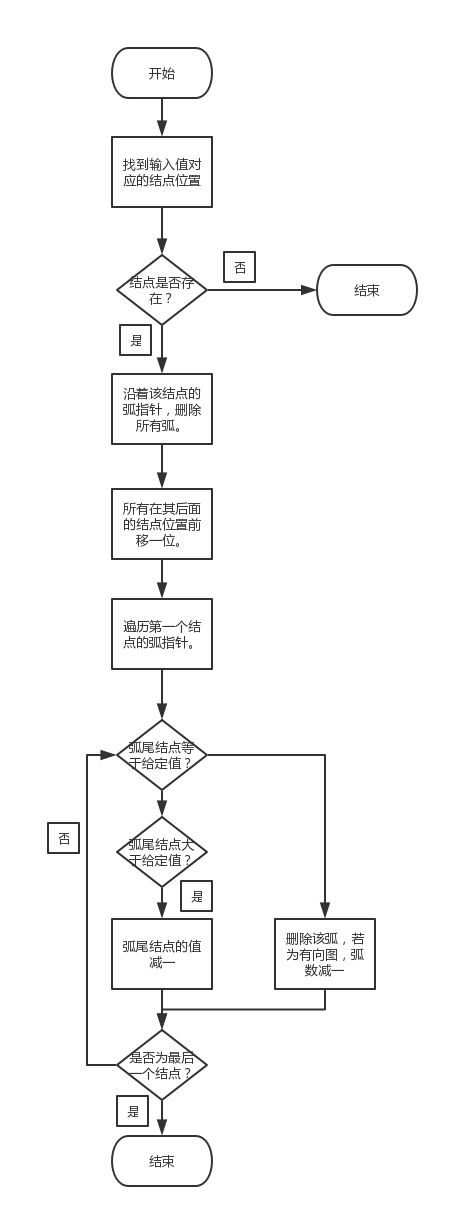
（5）DeleteVex(ALGraph G, VertexType ch)

**算法输入**：需要改变的图的指针，需要删除顶点的值

**函数调用**：定位函数

**算法输出**：实现状态（完成返回OK）

**算法处理步骤描述**：



需要注意结点是邻接表方式存储，所以要逐个前移。

**算法时间复杂度分析**：T（n）=O(n)

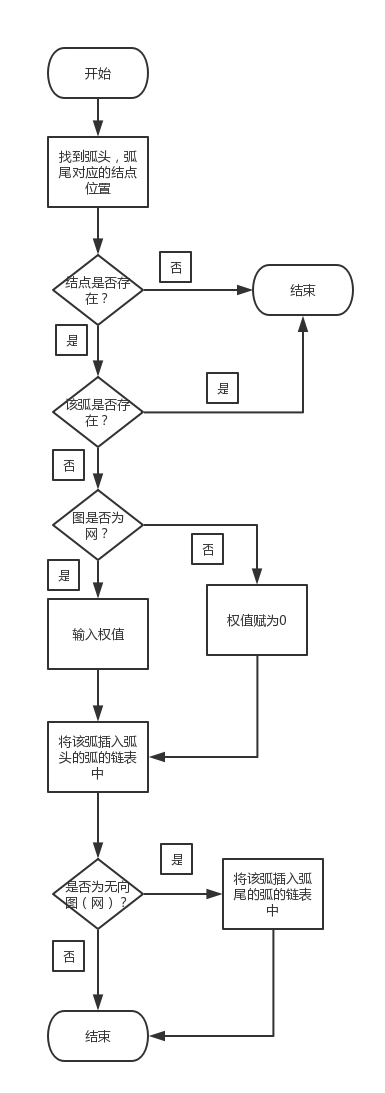
（6）InsertArc(ALGraph G, VertexType v1, VertexType v2)

**算法输入**：需要改变的图的指针，需要插入的弧的弧头和弧尾

**函数调用**：定位函数

**算法输出**：实现状态（完成返回OK）

**算法处理步骤描述**：



注意到图的类型，采取不同处理

**算法时间复杂度分析**：T（n）=O(n)

（7）DFSTraverse(ALGraph G)

**算法输入**：图的指针

**函数调用**：DFS的辅助函数

**算法输出**：实现状态（完成返回OK）

**算法处理步骤描述**：

1. 首先建立一个访问数组（最好是外部数组。访问过置为1），并将所有结点对应的值置为0。
2. 做一个循环，循环次数为结点数，对于没有访问过的结点调用辅助函数。
3. （辅助函数）传入结点没有访问过则输出，并且对应的值置为1。
4. 然后对其邻接点都递归调用辅助函数，直至访问完毕。

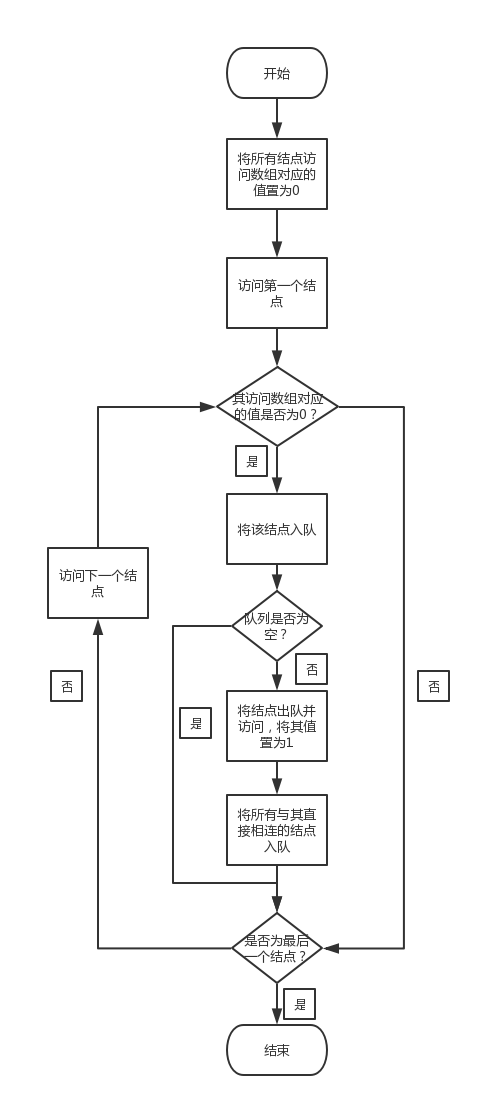
**算法时间复杂度分析**：T（n）=O(n)

（8）BFSTraverse(ALGraph G)

**算法输入**：图的指针

**算法输出**：实现状态（完成返回OK）

**算法处理步骤描述**：



**算法时间复杂度分析**：T（n）=O(n)

4.3 图的演示系统实现与测试

4.3.1 系统实现

电脑系统：windows10 专业版

编程环境：Dev c++

编程语言：C语言

函数清单及调用关系：

//队列操作函数 无特殊说明，均无调用关系

int InitQueue(Queue Q);//创建队列

int EnQueue(Queue Q, int e);//入队

int DeQueue(Queue Q);//出队

int IsQueueEmpty(Queue Q); //判空

int DestroyQueue(Queue Q);//摧毁队列

//图的操作

ALGraph CreateGraph(ALGraph G);//创建图

调用了定位函数

ALGraph DestroyGraph(ALGraph G);//销毁图

status LocateVex(ALGraph G, VertexType ch);//给定数据，判定该节点位置

VNode\* GetVex(ALGraph G, VertexType ch);//获得指定顶点指针

调用了定位函数

status PutVex(ALGraph G, VertexType v, VertexType ch);//替换指定结点的值

调用了定位函数

status FirstAdjVex(ALGraph G, VertexType ch);//获得指定顶点第一个邻接点位置

调用了定位函数

status NextAdjVex(ALGraph G, VertexType v, VertexType ch);//返回下一个邻接点

调用了定位函数

status InsertVex(ALGraph G, VertexType ch);//插入顶点

调用了定位函数

status DeleteVex(ALGraph G, VertexType ch);//删除顶点

调用了定位函数

status InsertArc(ALGraph G, VertexType v1, VertexType v2);//新增一条弧

调用了定位函数

status DeleteArc(ALGraph G, VertexType v1, VertexType v2);//删除一条弧

调用了定位函数

status DFSTraverse(ALGraph G);//深度优先遍历

status DFS(ALGraph G, int v);

status BFSTraverse(ALGraph G);//广度优先遍历

3.3.2 系统测试

**1.创建图函数的测试**

表3-1测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 用例1 | 0  0  0  （即一个空图） | 创建成功 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\E)8){WA5%B1XI@`HGS[G@Z5.png |
| 用例2 | 顶点数4（…）  边数2（…）  种类 0 | 创建成功 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\S)WPJDM@@@T8SYAX)4[V8W8.png |

**2.摧毁图测试**

表3-2测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 用例1 | 已有结点：1 a  2 b  3 c  4 d  5 e | 摧毁成功 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\%}(KADLAJCPT`T(6LD]9APS.png |
| 用例2 | 无结点，为空图 | 该图为空图 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\`NFYLLO$M_DFV0%0R[K2NJD.png |

**3.获得第一个邻接点测试**

表3-3测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 用例1 | 已有结点：1 a  2 b  3 c  4 d  5 e  输入：a | 该第一邻接点位置为1 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\(9H~@}X1LDJ7_$CUT09M@31.png |
| 用例2 | 已有结点：1 a  2 b  3 c  4 d  5 e  输入： f | 该第一邻接点不存在 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\FKZ$BOC6MJ{`CM$_{{7LZ@F.png |
| 用例3 | 已有结点：1 a  2 b  3 c  4 d  5 e  输入： e | 该第一邻接点不存在 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\FKZ$BOC6MJ{`CM$_{{7LZ@F.png |

**4.删除弧测试**

表3-4测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 用例1 | 已有结点：1 a  2 b  3 c  4 d  5 e  输入：a b | 删除成功 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\3K7FF2]}KE4448SJVO30)~2.png |
| 用例2 | 已有结点：1 a  2 b  3 c  4 d  5 e  输入： e f | 结点不存在 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\XH_V_3%IASB_~]$XI4${Z@C.png |
| 用例3 | 无结点，为空图 | 该图为空图 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\OE)6[A`_{6[@D3SY{)6VWSY.png |

**5.深度优先遍历测试**

表3-5测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 用例1 | 已有结点：1 a  2 b  3 c  4 d  5 e | 0 a  1 b  3 d  4 e  2 c | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\1KJ7W[G6XNT(E)%B[])DO$N.png |
| 用例2 | 已有结点：1 a  2 b  3 c  4 d | 0 a  1 b  3 d  2 c | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\0OWF%A~@3W9)72{O~%WRH7Y.png |
| 用例3 | 无结点，为空图 | 该图为空图 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\]H~05Z`N]$XDF{HN]BFM$1H.png |

**6.广度优先遍历的测试**

表3-6测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 用例1 | 已有结点：1 a  2 b  3 c  4 d  5 e | 0 a  1 b  2 c  3 d  4 e | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\6@5PXK0%FK7GL{86(]DN16O.png |
| 用例2 | 已有结点：1 a  2 b  3 c  4 d | 0 a  1 b  2 c  3 d | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\A02]F[_@TG1HVWO0X92N}LD.png |
| 用例3 | 无结点，为空图 | 该图为空图 | C:\Users\魏硕\AppData\Roaming\Tencent\Users\2411602273\QQ\WinTemp\RichOle\ZX}XC0E5KO1_AP293B[XY5E.png |

本次实验用例调试基本正常，每个功能能对正确用例做出正确反馈，对错误用例也能返回正确的结论。

3.4 实验小结

图的操作较树简单，最大的难点在于结点位置是以邻接表存储，而弧的信息是用的链表在存。这种混合结构造成的问题是在进行结点删除操作的时候必须把每一个在其后面的结点的位置进行移动，相应的每一条弧存的位置的信息也需要发生改变。这个对于之前的基本知识有所考验。

有了之前的经验一些细节的处理还算比较轻松。遇到的一个主要的问题就是声明的时候到底有没有分配空间，在结构体声明（或者说具体类型的声明）时一般空间都给了，即使有数组，数组也分配了相应大小的空间。声明指针时就没有给空间，但是可以赋值。第二个问题就是函数之间的传递性，传参和返回值不能只考虑函数自己的方便还需要考虑到其他函数的调用，不能有太多输入输出在函数内进行，会影响函数使用的灵活性。比如在写读出函数时需要调用插入函数，于是要返回去对原先的插入函数进行修改，不能让结点值之类的在函数内输入，而是通过传参传进去。

本次实验还存在的问题在于写入文件的时候没有办法额外增加权值信息，因为其是以结点的方式存的，所以文件的读写只能实现有向图和无向图的存储读写操作。

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

指导教师评定意见

一、对实验报告的评语

|  |
| --- |
|  |

二、对实验报告评分

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 评分项目  (分值) | 程序内容  (40分) | 程序规范  (10分) | 报告内容  (40分) | 报告规范  (10分) | 逾 期  扣 分 | 合 计  (100分) |
| 得分 |  |  |  |  |  |  |

# 附录A 基于顺序存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

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

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType; //数据元素类型定义

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

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef struct { //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

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

status IntiaList(SqList \*L);// 创建链表

status DestroyList(SqList \*L);//摧毁链表

status ClearList(SqList \*L);//清空链表

status ListEmpty(SqList L);//判空表

status ListLength(SqList L);//求表长

status GetElem(SqList L, int i, ElemType \*e);//获取元素

status LocateElem(SqList L, ElemType e); //简化过 定位元素

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

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

status ListInsert(SqList \* L,int i,ElemType e);//插入元素

status ListDelete(SqList \* L, int i, ElemType \* e);//删除元素

status ListTrabverse(SqList L); //简化过 遍历链表

status load\_List(SqList \*L);// 从文件读出

status Listfile(SqList \*L);//保存到文件

status write(SqList \*L);//

status writenew(SqList \*L);//

/\*--------------------------------------------\*/

void main(void) {

SqList L;

int op = 1, m,i;

ElemType e,cur\_e, pre, next;

L.length=0;

while (op)

{

system("cls");

printf("\n\n");

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

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

printf(" 1. IntiaList 7. LocateElem\n");

printf(" 2. DestroyList 8. PriorElem\n");

printf(" 3. ClearList 9. NextElem \n");

printf(" 4. ListEmpty 10. ListInsert\n");

printf(" 5. ListLength 11. ListDelete\n");

printf(" 6. GetElem 12. ListTrabverse\n");

printf(" 13. ListFile 14. ListLoad\n");

printf(" 15. WriteNew\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~15]:");

scanf("%d", &op);getchar();

switch (op)

{

case 1:

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

else printf(" 线性表创建失败！\n");

getchar();

break;

case 2:

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

else printf(" 线性表销毁失败！\n");

getchar();

break;

case 3:

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

else printf(" 线性表清空失败！\n");

getchar();

break;

case 4:

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

else printf(" 线性表不为空！\n");

getchar();

break;

case 5:

printf(" 线性表长度为%d\n", ListLength(L));

getchar();

break;

case 6:

if(L.length == 0)

{

printf(" 表为空，无法操作\n");

getchar();

}

else

{

printf(" 请输入要查询的序数：");

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

GetElem(L, i,&e);

if (GetElem(L, i,&e) == OK) printf(" 表中第%d个数据为%d\n", i, e);

else printf(" 查询失败！\n");

getchar();

}

break;

case 7:

if(L.length == 0)

{

printf(" 表为空，无法操作\n");

getchar();

}

else

{

printf(" 请输入要查询的数据：");

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

m = LocateElem(L, e);

if (m != ERROR)

{

printf(" L中第一个与查询数据相等的数据的位序为%d\n", m);

}

else

{

printf(" 这样的数据元素不存在！\n");

}

getchar();

}

break;

case 8:

if(L.length == 0)

{

printf(" 表为空，无法操作\n");

getchar();

}

else

{

printf(" 请输入要查询的元素：");

scanf("%d", &cur\_e);getchar();

if (PriorElem(L, cur\_e, &pre) == OK) printf(" 前驱为%d\n", pre);

else printf(" 无此前驱\n");

getchar();

}

break;

case 9:

if(L.length == 0)

{

printf(" 表为空，无法操作\n");

getchar();

}

else

{

printf(" 请输入要查询的元素：");

scanf("%d", &cur\_e);getchar();

if (NextElem(L, cur\_e, &next) == OK) printf(" 后驱为%d\n", next);

else printf(" 无此后驱\n");

getchar();

}

break;

case 10:

printf(" 请输入位置i：");

scanf("%d", &i);

printf(" 请输入e：");

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

if (ListInsert(&L, i, e) == OK) printf(" 线性表插入成功\n");

else printf(" 线性表插入失败\n");

getchar();

break;

case 11:

if(L.length == 0)

{

printf(" 表为空，无法操作\n");

getchar();

}

else

{

printf(" 请输入要删除的元素的序列：");

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

if (ListDelete(&L, i, &e) == OK)

{

printf(" 元素删除成功\n");

printf(" 删除元素值为%d\n",e);

}

else printf(" 元素删除失败\n");

getchar();

}

break;

case 12:

if(L.length == 0)

{

printf(" 表为空，无法操作\n");

getchar();

}

else

{

ListTrabverse(L);

getchar();

}

break;

case 13:

if(L.length == 0)

{

printf(" 表为空，无法操作\n");

getchar();

}

else

{

Listfile(&L);

getchar();

}

break;

case 14:

load\_List(&L);

getchar();

break;

case 15:

writenew(&L);

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

/\*--------page 23 on textbook --------------------\*/

status IntiaList(SqList \*L) //创建一个空的链表

{

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

if (!L->elem) exit(OVERFLOW);

L->listsize = LIST\_INIT\_SIZE;

L->length=0;

return OK;

}

status DestroyList(SqList \*L) //清空一个链表并释放存储空间

{

free(L->elem);

L->elem = NULL;

L->length=0;

return OK;

}

status ClearList(SqList \*L)//将一个链表置为空

{

L->length = 0;

return OK;

}

status ListEmpty(SqList L)//判断表是否为空

{

if (L.length == 0)

return TRUE;

else

return FALSE;

}

int ListLength(SqList L) //计算表长

{

return L.length;

}

status GetElem(SqList L,int i,ElemType \*e)//取链表中第i个元素并返回

{

if((i < 0) || (i >= L.length))

return FALSE;

\*e = \*(L.elem+i-1);

return OK;

}

int LocateElem(SqList L,ElemType e)//返回第一个和e相等的数的位置

{

int i=1;

while (\*L.elem != e)

{

L.elem++;

i++;

}

if (i > L.length)

return ERROR;

return i;

}

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

{

int i;

if((i<0) || (i>=L.length))

return FALSE;

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

{

if (L.elem[i] == cur\_e)

{

\*pre = L.elem[i-1];

return OK;

}

}

return FALSE;

}

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

{

int i;

if((i<0) || (i>=L.length))

return FALSE;

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

{

if (L.elem[i] == cur\_e)

{

\*next=L.elem[i+1];

return OK;

}

}

return FALSE;

}

status ListInsert(SqList \*L,int i,ElemType e)//在第i个元素前插入值e

{

ElemType \*newone,\*q,\*p;

if (!L->elem) return ERROR;

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

if (L->length >= L->listsize)

{

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

if (!newone) return ERROR;

L->elem =newone;

L->listsize += LISTINCREMENT;

}

q=&(L->elem[i-1]);

for (p=&(L->elem[L->length-1]);p >= q;p--)

\*(p+1)=\*p;

\*q = e;

L->length++;

return OK;

}

status ListDelete(SqList \*L, int i, ElemType \*e)//删除第i个元素，并用e返回

{

ElemType \*q,\*p;

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

p=&(L->elem[i-1]);

\*e=\*p;

q=&(L->elem[L->length-1]);

for (p++;p <= q;++p)

\*(p-1)=\*p;

L->length--;

return OK;

}

status ListTrabverse(SqList L) //输出所有的元素

{

int i;

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

for (i=0;i<L.length;i++) printf("%d ", L.elem[i]);

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

return OK;

}

status Listfile(SqList \*L)//将数据写到文件

{

FILE \*out1;

if((out1 = fopen("sunxu.txt","w")) == NULL)

{

printf(" 文件打开失败！\n");

return 1;

}

printf(" 已保存到文件！\n");

fwrite(L->elem,sizeof(ElemType),L->length,out1);

fclose(out1);

}

status load\_List(SqList \*L) //数据调出

{

FILE \*in1;

if((in1 = fopen("sunxu.txt","r")) == NULL)

{

printf(" 文件打开失败！\n");

return 1;

}

printf(" 已从文件导出！\n");

IntiaList(L);

L->length=0;

while(fread(&L->elem[L->length],sizeof(ElemType),1,in1))

L->length++;

fclose(in1);

}

status writenew(SqList \*L)//在表后加元素

{

char ch;

ch='y';

ElemType \*nw;

while(ch=='y' || ch=='Y')

{

printf(" 输入数据:");

scanf("%d",&(L->elem[L->length]));

getchar();

L->length++;

if (L->length >= L->listsize)

{

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

if (!nw) return ERROR;

L->elem = nw;

L->listsize += LISTINCREMENT;

}

printf(" 继续输入？y or n\n");

printf(" 输入你的选择:");

ch=getchar();

getchar();

}

return 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 <malloc.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType;

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef struct ListNode {

ElemType data;

struct ListNode \*next;

}ListNode,\*pListNode;

status IntiaList(ListNode \*\*L);// 建立链表

status DestroyList(ListNode \*L); //摧毁链表

status ClearList(ListNode \*L);//清空链表

status ListEmpty(ListNode \* L);//判空链表

status ListLength(ListNode \* L);//求链表长度

status GetElem(ListNode \*L, int i, ElemType \*e); //获取指定元素

status LocateElem(ListNode \* L, ElemType e);//指定元素位置

status PriorElem(ListNode \* L, ElemType e, ElemType\* pre);//指定元素前驱

status NextElem(ListNode \* L, ElemType cur\_e, ElemType\* next\_e);//指定元素后继

status ListInsert(ListNode \*\*L, int i, ElemType e);//插入指定元素

status ListDelete(ListNode \*\*L, int i, ElemType\* e);//删除指定元素

status ListTrabverse(ListNode \* L);//遍历链表

status SaveFile(ListNode \* L);//保存文件

status LoadFile(ListNode \*\*L);//读出文件

status creat\_kindlist(ListNode \*\*L);//大量写入

int main(void)

{

int op = 1, e,pre, next, m;

ListNode \*L;

int i;

while (op) {

system("cls");

system("color 1F");//系统命令，调颜色

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\* \*\n");

printf("\t\t\* ◆链表功能实现菜单 ◆ \*\n");

printf("\t\t\* \*\n");

printf("\t\t\* 0.退出系统 \*\n");

printf("\t\t\* 1.初始化表 \*\n");

printf("\t\t\* 2.销毁已有表 \*\n");

printf("\t\t\* 3.清空已有表 \*\n");

printf("\t\t\* 4.判定是否为空表 \*\n");

printf("\t\t\* 5.求表的长度 \*\n");

printf("\t\t\* 6.获取指定元素 \*\n");

printf("\t\t\* 7.查找指定元素 \*\n");

printf("\t\t\* 8.获得指定元素前驱 \*\n");

printf("\t\t\* 9.获得指定元素后继 \*\n");

printf("\t\t\* 10.插入指定元素 \*\n");

printf("\t\t\* 11.删除指定元素 \*\n");

printf("\t\t\* 12.遍历表 \*\n");

printf("\t\t\* 13.存入文件 \*\n");

printf("\t\t\* 14.读出文件 \*\n");

printf("\t\t\* \*\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

printf("\t\t 请输入你的选择：");

scanf("%d",&op);

switch (op)

{

case 1:

//printf("\n----IntiaList功能待实现！\n");

if (IntiaList(&L) == OK)

printf("线性表创建成功！\n");

else

printf("线性表创建失败！\n");

getchar();getchar();

break;

case 2:

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

if (DestroyList(L) == OK)

printf("线性表销毁成功！\n");

else

printf("线性表销毁失败！\n");

getchar();getchar();

break;

case 3:

//printf("\n----ClearList功能待实现！\n");

if (ClearList(L) == OK)

printf("线性表清空成功！\n");

else

printf("线性表清空失败！\n");

getchar();getchar();

break;

case 4:

//printf("\n----ListEmpty功能待实现！\n");

if (ListEmpty(L) == OK)

printf("线性表为空！\n");

else

printf("线性表不为空！\n");

getchar();getchar();

break;

case 5:

//printf("\n----ListLength功能待实现！\n");

printf("线性表长度为%d\n", ListLength(L));

getchar();getchar();

break;

case 6:

//printf("\n----GetElem功能待实现！\n");

if (ListEmpty(L) == OK)

{

printf("表为空，不可进行该项操作\n");

getchar();getchar();

}

else

{

printf("请输入要查询的数据的序数：");

scanf("%d", &i);

if (GetElem(L, i, &e) == OK)

printf("表中第%d个数据为%d\n", i, e);

else

printf("查询失败！\n");

getchar();getchar();

}

break;

case 7:

//printf("\n----LocateElem功能待实现！\n");

if (ListEmpty(L) == OK)

{

printf("表为空，不可进行该项操作\n");

getchar();getchar();

}

else

{

printf("请输入要查询的数据：\n");

scanf("%d", &e);

m = LocateElem(L, e);

if (m != ERROR)

{

printf("L中第一个与查询数据相等的数据的位序为%d\n", m);

}

else

{

printf("该数据元素不存在！\n");

}

getchar();getchar();

}

break;

case 8:

if (ListEmpty(L) == OK)

{

printf("表为空，不可进行该项操作\n");

getchar();getchar();

}

else

{

printf("请输入要查询的元素：");

scanf("%d", &e);

if (PriorElem(L, e, &pre) == OK) printf("前驱为%d\n", pre);

else printf("此元素无前驱\n");

getchar();getchar();

}

break;

case 9:

if (ListEmpty(L) == OK)

{

printf("表为空，不可进行该项操作\n");

getchar();getchar();

}

else

{

printf("请输入要查询的元素：");

scanf("%d", &e);

if (NextElem(L, e, &next) == OK) printf("后继为%d\n", next);

else printf("此元素无后继\n");

getchar();getchar();

}

break;

case 10:

printf("请输入插入位置i：");

scanf("%d", &i);

printf("请输入数字e：");

scanf("%d", &e);

if (ListInsert(&L, i, e) == OK)

printf("线性表插入成功\n");

else

printf("线性表插入失败\n");

getchar();getchar();

break;

case 11:

if (ListEmpty(L))

{

printf("表为空，不可进行该项操作\n");

getchar();getchar();

}

else

{

printf("请输入要删除的元素的序列：");

scanf("%d", &i);

if (ListDelete(&L, i, &e) == OK)

{

printf("元素删除成功\n");

printf("删除元素为%d\n",e);

}

else printf("元素删除失败\n");

getchar();getchar();

}

break;

case 12:

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

if (!ListTrabverse(L))

printf("线性表是空表！\n");

getchar();getchar();

break;

case 13:

if(SaveFile(L))

printf("数据保存成功!\n");

getchar();getchar();

break;

case 14:

if(LoadFile(&L))

printf("数据载入成功！\n");

getchar();getchar();

break;

case 0:

SaveFile(L);

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}

status IntiaList(ListNode \*\*L)

{

(\*L)= (ListNode \*)malloc(sizeof(ListNode));

(\*L)->next = NULL;

(\*L)->data = 0;

return OK;

}

status DestroyList(ListNode \*L)

{

if(L==NULL) return OK;

ListNode \*pcrs,\*p;

pcrs=L;

p=pcrs->next;

while(p != NULL)

{

free(pcrs);

pcrs =p ;

p = p->next;

}

free(pcrs);

return OK;

}

status ClearList(ListNode \*L)

{

if(L==NULL) return ERROR;

L->data=0;

ListNode \*pcrs,\*p;

pcrs=L->next;

p=pcrs->next;

while(p != NULL)

{

free(pcrs);

pcrs = p;

p =p->next;

}

free(pcrs);

return OK;

}

status ListEmpty(ListNode \* L)

{

if (L == NULL || L->data == 0)

return TRUE;

else

return FALSE;

}

int ListLength(ListNode \* L)

{

if(L == NULL)

return 0;

return L->data;

}

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

{

int j=1; //计数器

ListNode \*p;

if(!ListEmpty(L))

{

if (i<1 || i>L->data)

return ERROR;

p = L->next;

while(j!=i)

{

p = p->next;

j++;

}

(\*e)=p->data;

return OK;

}

else

return FALSE;

}

status LocateElem(ListNode \* L, ElemType e)

{

pListNode p = L->next;

int i = 1;

while (p)

{

if(p->data==e)

return i;

else

{

i++;

p = p->next;

}

}

return 0;

}

status PriorElem(ListNode \* L, ElemType e, ElemType\* pre)

{

pListNode pcrs,p;

pcrs = L->next,p= pcrs->next;

while (p)

{

if(p->data==e)

{

\*pre=pcrs->data;

return OK;

}

pcrs=p;

p=p->next;

}

return ERROR;

}

status NextElem(ListNode \* L, ElemType e, ElemType\* next)

{

pListNode p = L->next;

while(p)

{

if(p->data==e)

{

\*next=(p->next)->data;

return OK;

}

p=p->next;

}

return ERROR;

}

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

{

int j=1;//计数器

if ( \*L != NULL )

{

pListNode Lhead = \*L,p,p1;

if (i<1|| i>Lhead->data + 1)

return ERROR;

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

p->data=e;

p1=Lhead;

if (Lhead->data == 0)

{

Lhead->next = p;

p->next = NULL;

Lhead->data = 1;

return OK;

}

while(p1->next!=NULL)

{

if(j==i)

{

p->next=p1->next;

p1->next=p;

p->data = e;

Lhead->data+=1;

return OK;

}

else

{

p1=p1->next;

j++;

}

}

}

else

return ERROR;

}

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

{

if( !ListEmpty(\*L) )

{

pListNode Lhead = \*L,p1,p2;

int j;

if (i<1 || i>Lhead->data)

return ERROR;

if (Lhead->data)

{

p1 = Lhead, p2 = Lhead->next;

j = 1;

while (p2 && j<i)

{

p1 = p2;

p2 = p2->next;

j++;

}

p1->next = p2->next;

\*e = p2->data;

Lhead->data-=1;

free(p2);

return OK;

}

return ERROR;

}

else

return ERROR;

}

status ListTrabverse(pListNode L)

{

if (!L->data)return ERROR;

pListNode p = L->next;

int i = 1;

printf("\n-----------对所有元素进行间访-----------------------\n");

while (p)

{

printf("第%d个元素的元素值为%d\n\n", i, p->data);

i++;

p = p->next;

}

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

return OK;

}

status SaveFile(pListNode L)

{

FILE \*in;

if((in = fopen("shuju.txt","w")) == NULL)

{

printf("文件打开失败！\n");

return ERROR;

}

L = L->next;

while(L != NULL)

{

fprintf(in, "%d\n", L->data);

L = L->next;

}

fclose(in);

return OK;

}

status LoadFile(ListNode \*\*L)

{

FILE \*out;

ElemType e;

pListNode p;

if((out= fopen("shuju.txt","r")) == NULL)

{

printf("文件打开失败！\n");

return ERROR;

}

IntiaList(&p);

while( !feof(out) )

{

fscanf(out, "%d\n", &e);

ListInsert(&p, 1, e);//以插入形式读元素

}

fclose(out);

\*L = p;

return OK;

}

# 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>

#include <malloc.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

typedef int status;

typedef struct Data TElemType;

#define FORESTSIZE 20

#define STACK\_INIT\_SIZE 100

#define STACKINCREMENT 20

//树的数据域的定义

struct Data {

int number;//结点的编号

char name[20];//结点存的名字

};

//树节点定义

typedef struct BiTNode {

TElemType data;

struct BiTNode \*lchild,\*rchild;

}BiTNode,\*BiTree;

//栈的定义

typedef struct Stack {

BiTree \*base;

int top;

int stacksize;

}\*Stack,stack;

BiTree Forest[FORESTSIZE];//森林

//栈的操作

int InitStack(Stack S);//栈的初始化

int Push(Stack S, BiTree BT);//入栈

BiTree Pop(Stack S);//出栈

int IsStackEmpty(Stack S);//判空栈

int DestroyStack(Stack S);//摧毁栈

//树的操作

int InitBiTree(void);//初始化森林

int DestroyBiTree(BiTree \*T);//摧毁树

BiTree CreateBiTree(BiTree T);//创建树

int ClearBiTree(BiTree \*T);//清空树

int BiTreeEmpty(BiTree T);//判断树是否为空

int BiTreeDepth(BiTree T);//求树的深度

int Root(BiTree T);//返回根节点

BiTree Value(BiTree T,int e);//输入结点的编号，返回结点

int Assign(BiTree \*T,int e);//找到结点e，并改变它的值

BiTree Parent(BiTree T,int e);//返回双亲结点

BiTree LeftChild(BiTree T,int e);//返回左孩子结点

BiTree RightChild(BiTree T,int e);//返回右孩子

BiTree LeftSibling(BiTree T,int e);//返回左兄弟

BiTree RightSibling(BiTree T,int e);//返回右兄弟

int InsertChild(BiTree \*T,int e,int LR);//插入子树 ,LR为1则是左子树

int DeleteChild(BiTree \*T,int e,int LR);//删除子树，LR为1则是左子树

int PreOrderTraverse(BiTree T);//前序遍历

int InOrderTraverse(BiTree T);//中序遍历

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

int LevelTraverse(BiTree T);//层次遍历

void PrintLevel(BiTree T,int level);//层次遍历的一部分

int SaveBiTree(BiTree T);//写入文件

int LoadBiTree(BiTree \*T);//读出文件

int main(void)

{

BiTree T,p;

int e,LR,num,op=1;

while (op)

{

system("cls");

system("color 1F");//系统命令，调颜色

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\* \*\n");

printf("\t\t\* ◆树的功能实现菜单 ◆ \*\n");

printf("\t\t\* \*\n");

printf("\t\t\* 0.退出系统 \*\n");

printf("\t\t\* 1.初始化二叉树 \*\n");

printf("\t\t\* 2.销毁二叉树 \*\n");

printf("\t\t\* 3.创建二叉树 \*\n");

printf("\t\t\* 4.清空二叉树 \*\n");

printf("\t\t\* 5.判空二叉树 \*\n");

printf("\t\t\* 6.求二叉树深度 \*\n");

printf("\t\t\* 7.获得根节点 \*\n");

printf("\t\t\* 8.获得指定结点 \*\n");

printf("\t\t\* 9.指定结点赋值 \*\n");

printf("\t\t\* 10.获得结点双亲 \*\n");

printf("\t\t\* 11.获得结点左孩子 \*\n");

printf("\t\t\* 12.获得结点右孩子 \*\n");

printf("\t\t\* 13.获得结点左兄弟 \*\n");

printf("\t\t\* 14.获得结点右兄弟 \*\n");

printf("\t\t\* 15.插入子树 \*\n");

printf("\t\t\* 16.删除子树 \*\n");

printf("\t\t\* 17.前序遍历 \*\n");

printf("\t\t\* 18.中序遍历 \*\n");

printf("\t\t\* 19.后序遍历 \*\n");

printf("\t\t\* 20.按层遍历 \*\n");

printf("\t\t\* 21.存入文件 \*\n");

printf("\t\t\* 22.从文件中读取数据 \*\n");

printf("\t\t\* \*\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

printf("\t\t 请输入你的选择：");

scanf("%d",&op);getchar();

switch(op)

{

case 1:

system("cls");

if(InitBiTree())

printf("成功创建森林！\n");

getchar();

break;

case 2:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else if(DestroyBiTree(&(Forest[num])))

printf("摧毁成功！\n");

getchar();

break;

case 3:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

Forest[num]=CreateBiTree(Forest[num]);

printf("创建成功！\n");

getchar();

break;

case 4:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else if(ClearBiTree(&(Forest[num])))

printf("清空成功！\n");

getchar();

break;

case 5:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(BiTreeEmpty(Forest[num]))

printf("该树为空树！\n");

else

printf("该树不为空树！\n");

getchar();break;

case 6:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

printf("该树的深度为%d\n",BiTreeDepth(Forest[num]));

getchar();break;

case 7:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

Root(Forest[num]);

getchar();break;

case 8:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else

{

printf("请输入结点的编号：");

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

p=Value(Forest[num],e);

if(!p)

printf("该节点不存在！\n");

else

printf("该节点为 %d %s\n",p->data.number,p->data.name);

}

getchar();break;

case 9:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else

{

printf("请输入结点的编号：");

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

if(!Assign(&Forest[num],e))

printf("该节点不存在！\n");

else

printf("该节点的值修改成功！\n");

}

getchar();break;

case 10:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else

{

printf("请输入结点的编号：");

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

p=Parent(Forest[num],e);

if(!p)

printf("该节点不存在！\n");

else

printf("该节点双亲为 %d %s\n",p->data.number,p->data.name);

}

getchar();break;

case 11:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else

{

printf("请输入结点的编号：");

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

p=LeftChild(Forest[num],e);

if(!p)

printf("该节点不存在！\n");

else

printf("该节点左孩子为 %d %s\n",p->data.number,p->data.name);

}

getchar();break;

case 12:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else

{

printf("请输入结点的编号：");

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

p=RightChild(Forest[num],e);

if(!p)

printf("该节点不存在！\n");

else

printf("该节点右孩子为 %d %s\n",p->data.number,p->data.name);

}

getchar();break;

case 13:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else

{

printf("请输入结点的编号：");

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

p=LeftSibling(Forest[num],e);

if(!p)

printf("该节点不存在！\n");

else

printf("该节点左兄弟为 %d %s\n",p->data.number,p->data.name);

}

getchar();break;

case 14:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else

{

printf("请输入结点的编号：");

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

p=RightSibling(Forest[num],e);

if(!p)

printf("该节点不存在！\n");

else

printf("该节点右兄弟为 %d %s\n",p->data.number,p->data.name);

}

getchar();break;

case 15:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else

{

printf("请输入结点的编号：");

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

printf("请输入插入左子树还是右子树(左为1，右为0)：");

scanf("%d",&LR);getchar();

if(!InsertChild(&Forest[num],e,LR))

printf("该节点不存在！\n");

else

printf("插入成功！\n");

}

getchar();break;

case 16:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else

{

printf("请输入结点的编号：");

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

printf("请输入删除左子树还是右子树(左为1，右为0)：");

scanf("%d",&LR);getchar();

if(!DeleteChild(&Forest[num],e,LR))

printf("该节点不存在！\n");

else

printf("删除成功！\n");

}

getchar();break;

case 17:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else

{

printf("前序遍历的结果为：\n");

PreOrderTraverse(Forest[num]);

}

getchar();break;

case 18:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else

{

printf("中序遍历的结果为：\n");

InOrderTraverse(Forest[num]);

}

getchar();break;

case 19:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else

{

printf("中序遍历的结果为：\n");

PostOrderTraverse(Forest[num]);

}

getchar();break;

case 20:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else

{

printf("层次遍历的结果为：\n");

LevelTraverse(Forest[num]);

}

getchar();break;

case 21:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

printf("该树为空树！\n");

else

{

SaveBiTree(Forest[num]);

printf("写入文件成功！\n");

}

getchar();break;

case 22:

system("cls");

printf("请输入该树的编号（0-19）：");

scanf("%d",&num);getchar();

LoadBiTree(&Forest[num]);

printf("文件读出成功！\n");

getchar();break;

}

}

printf("欢迎下次再使用本系统！\n");

}

//栈

int InitStack(Stack S)

{

S->base=(BiTree \*)malloc(sizeof(BiTree)\*STACK\_INIT\_SIZE);

if(!S->base) exit(OVERFLOW);

S->stacksize=STACK\_INIT\_SIZE;

S->top=-1;

return OK;

}

int Push(Stack S, BiTree T)

{

if ((S->top+1) >= S->stacksize)

{

S->base=(BiTree \*)realloc(S->base, sizeof(BiTree)\*(S->stacksize+STACKINCREMENT));

if(!S->base) exit(OVERFLOW);

S->stacksize=S->stacksize+STACKINCREMENT;

}

S->top++;

S->base[S->top]=T;

return OK;

}

BiTree Pop(Stack S)

{

if (IsStackEmpty(S) == TRUE)

return NULL;

return S->base[S->top--];

}

int IsStackEmpty(Stack S)

{

if (S->top == -1)

return TRUE;

else

return FALSE;

}

//销毁栈S

int DestroyStack(Stack S)

{

free(S->base);

free(S);

S = NULL;

return OK;

}

//树

int InitBiTree(void)//构造空的森林

{

int i;

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

Forest[i]=NULL;

return OK;

}

int DestroyBiTree(BiTree \*T)

{

BiTree p=\*T;

if(p == NULL)

return ERROR;

if(p->lchild != NULL)

DestroyBiTree(&p->lchild);

if(p->rchild != NULL)

DestroyBiTree(&p->rchild);

free(p);

\*T=NULL;

return OK;

}

BiTree CreateBiTree(BiTree T)

{

char ch;

int num;

char name[20];

printf("是否为空结点？y or n\n");

ch=getchar();getchar();

if((ch == 'y') || (ch == 'Y'))

T = NULL;

else

{

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

printf("输入结点编号及名字\n");

printf("编号：");

scanf("%d",&(T->data.number));

printf("名字：");

scanf("%s",T->data.name);getchar();

T->lchild=CreateBiTree(T->lchild);

T->rchild=CreateBiTree(T->rchild);

}

return T;

}

int ClearBiTree(BiTree \*T)

{

BiTree p=\*T;

if(p->lchild != NULL)

ClearBiTree(&p->lchild);

if(p->rchild != NULL)

ClearBiTree(&p->rchild);

free(p);

\*T=NULL;

return OK;

}

int BiTreeEmpty(BiTree T)

{

if((T == NULL) || T->data.number==0)

return TRUE;

else

return FALSE;

}

int BiTreeDepth(BiTree T)

{

BiTree p=T;

int l,r,max;

if(p != NULL)

{

l=BiTreeDepth(p->lchild);

r=BiTreeDepth(p->rchild);

max=(l>r)?l:r;

return max+1;

}

else

return 0;

}

int Root(BiTree T)

{

if(T == NULL)

{

printf("这棵树不存在！\n");

return 0;

}

else

{

printf("根节点编号为%d,名字为%s\n",T->data.number,T->data.name);

return 1;

}

}

BiTree Value(BiTree T,int e)//先序遍历

{

BiTree p=T;

Stack S;

InitStack(S);

if(p == NULL)

return NULL;

while((p != NULL) || !IsStackEmpty(S) )

{

while(p != NULL)

{

Push(S,p);

if(p->data.number == e)

{

DestroyStack(S);

return p;

}

p=p->lchild;

}

if(IsStackEmpty(S) == FALSE)

{

p=Pop(S);

p=p->rchild;

}

}

return NULL;

}

int Assign(BiTree \*T,int e)

{

BiTree p=\*T;

Stack S;

InitStack(S);

if(p == NULL)

return FALSE;

while((p != NULL) || !IsStackEmpty(S))

{

while(p != NULL)

{

Push(S,p);

if(p->data.number == e)

{

printf("输入结点编号：");

scanf("%d",&p->data.number);

printf("输入结点名字：");

scanf("%s",p->data.name);getchar();

DestroyStack(S);

return OK;

}

p=p->lchild;

}

if(IsStackEmpty(S) == FALSE)

{

p=Pop(S);

p=p->rchild;

}

}

return FALSE;

}

BiTree Parent(BiTree T,int e)

{

BiTree p=T;

Stack S;

InitStack(S);

if(p == NULL)

return NULL;

while((p != NULL) || !IsStackEmpty(S))

{

while(p != NULL)

{

Push(S,p);

if(p->lchild != NULL)

if(p->lchild->data.number == e)

return p;

if(p->rchild != NULL)

if(p->rchild->data.number == e)

return p;

p=p->lchild;

}

if(IsStackEmpty(S) == FALSE)

{

p=Pop(S);

p=p->rchild;

}

}

return NULL;

}

BiTree LeftChild(BiTree T,int e)

{

BiTree p=T;

Stack S;

InitStack(S);

if(p == NULL)

return NULL;

while((p != NULL) || !IsStackEmpty(S))

{

while(p != NULL)

{

Push(S,p);

if(p->data.number == e)

return p->lchild;

p=p->lchild;

}

if(IsStackEmpty(S) == FALSE)

{

p=Pop(S);

p=p->rchild;

}

}

return NULL;

}

BiTree RightChild(BiTree T,int e)

{

BiTree p=T;

Stack S;

InitStack(S);

if(p == NULL)

return NULL;

while((p != NULL) || !IsStackEmpty(S))

{

while(p != NULL)

{

Push(S,p);

if(p->data.number == e)

return p->rchild;

p=p->lchild;

}

if(IsStackEmpty(S) == FALSE)

{

p=Pop(S);

p=p->rchild;

}

}

return NULL;

}

BiTree LeftSibling(BiTree T,int e)

{

BiTree p=T;

Stack S;

InitStack(S);

if(p == NULL)

return NULL;

while((p != NULL) || !IsStackEmpty(S))

{

while(p != NULL)

{

Push(S,p);

if(p->rchild != NULL)

if(p->rchild->data.number == e)

return p->lchild;

p=p->lchild;

}

if(IsStackEmpty(S) == FALSE)

{

p=Pop(S);

p=p->rchild;

}

}

return NULL;

}

BiTree RightSibling(BiTree T,int e)

{

BiTree p=T;

Stack S;

InitStack(S);

if(p == NULL)

return NULL;

while((p != NULL) || !IsStackEmpty(S))

{

while(p != NULL)

{

Push(S,p);

if(p->lchild != NULL)

if(p->lchild->data.number == e)

return p->rchild;

p=p->lchild;

}

if(IsStackEmpty(S) == FALSE)

{

p=Pop(S);

p=p->rchild;

}

}

return NULL;

}

int InsertChild(BiTree \*T,int e,int LR)

{

BiTree p=\*T;

Stack S;

int num;

InitStack(S);

if(p == NULL)

return 0;

while((p != NULL) || !IsStackEmpty(S))

{

while(p != NULL)

{

Push(S,p);

if(p->data.number == e)

{

printf("将第几棵树作为子树插入？\n");

scanf("%d",&num);getchar();

if(Forest[num] == NULL)

return FALSE;

if(LR == 1)

p->lchild=Forest[num];

if(LR == 0)

p->rchild=Forest[num];

return OK;

}

p=p->lchild;

}

if(IsStackEmpty(S) == FALSE)

{

p=Pop(S);

p=p->rchild;

}

}

return 0;

}

int DeleteChild(BiTree \*T,int e,int LR)

{

BiTree p=\*T;

Stack S;

InitStack(S);

if(p == NULL)

return 0;

while((p != NULL) || !IsStackEmpty(S))//无法处理结点不存在的情况，貌似是栈有问题

{

while(p != NULL)

{

Push(S,p);

if(p->data.number == e)

{

if(LR == 1)

if(DestroyBiTree(&(p->lchild)) == 0)

printf("该树为空树\n");

if(LR == 0)

if(DestroyBiTree(&(p->rchild)) == 0)

printf("该树为空树\n");

return OK;

}

p=p->lchild;

}

if(IsStackEmpty(S) == FALSE)

{

p=Pop(S);

p=p->rchild;

}

}

return 0;

}

int PreOrderTraverse(BiTree T)

{

BiTree p=T;

if(p != NULL)

{

printf("%d %s\n",p->data.number,p->data.name);

PreOrderTraverse(p->lchild);

PreOrderTraverse(p->rchild);

return OK;

}

else

return 0;

}

int InOrderTraverse(BiTree T)

{

BiTree p=T;

Stack S;

InitStack(S);

while((p != NULL) || !IsStackEmpty(S) )

{

while(p != NULL)

{

Push(S,p);

p=p->lchild;

}

if(IsStackEmpty(S) == FALSE)

{

p=Pop(S);

printf("%d %s\n",p->data.number,p->data.name);

p=p->rchild;

}

}

return OK;

}

int PostOrderTraverse(BiTree T)

{

BiTree p=T;

if(p != NULL)

{

PostOrderTraverse(p->lchild);

PostOrderTraverse(p->rchild);

printf("%d %s\n",p->data.number,p->data.name);

}

else

return 0;

}

int LevelTraverse(BiTree T)

{

int depth,i;

depth=BiTreeDepth(T);

for(i=1;i<=depth;i++)

PrintLevel(T, i);

return OK;

}

void PrintLevel(BiTree T,int level)

{

if (T == NULL || level < 1 )

return;

if (level == 1)

{

printf("%d %s\n",T->data.number,T->data.name);

return;

}

PrintLevel(T->lchild, level-1);// 左子树的 level - 1 级

PrintLevel(T->rchild, level-1); // 右子树的 level - 1 级

}

int SaveBiTree(BiTree T)

{

BiTree p=T;

Stack S;

InitStack(S);

FILE \*in;

if((in = fopen("shuju.txt","w")) == NULL)

{

printf("文件打开失败！\n");

return ERROR;

}

if(p == NULL)

return ERROR;

while((p != NULL) || !IsStackEmpty(S))

{

while(p != NULL)

{

Push(S,p);

fprintf(in, "\*%d %s\n",p->data.number,p->data.name);

p=p->lchild;

}

fprintf(in, "#\n");

if(IsStackEmpty(S) == FALSE)

{

p=Pop(S);

p=p->rchild;

}

}

fprintf(in, "#\n");

fclose(in);

return OK;

}

int LoadBiTree(BiTree \*T)

{

FILE \*out;

int flag=0;

BiTree p,q;

Stack S=(Stack)malloc(sizeof(struct Stack));

InitStack(S);

char ch;

if((out= fopen("shuju.txt","r")) == NULL)

{

printf("文件打开失败！\n");

return ERROR;

}

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

while((ch=fgetc(out))!='#' || !IsStackEmpty(S))

{

if(ch == '#')

fgetc(out);

while(ch != '#')

{

fscanf(out,"%d %s\n",&p->data.number,p->data.name);

if(flag == 0)

{

\*T = p;

flag = 1;

}

Push(S,p);

q=p;

p->lchild = (BiTree)malloc(sizeof(BiTNode));

p=p->lchild;

ch=fgetc(out);

if(ch == '#')

fgetc(out);

}

q->lchild = NULL;

q->rchild=NULL;

if(IsStackEmpty(S) == FALSE)

{

p=Pop(S);

p->rchild=(BiTree)malloc(sizeof(BiTNode));

q=p;

p=p->rchild;

}

}

free(q->rchild);

q->rchild=NULL;

fclose(out);

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 INFEASTABLE -1

#define OVERFLOW -2

#define MAX\_VERTEX\_NUM 20

typedef int status;

typedef char VertexType;

typedef struct ArcNode { //弧的信息

int adjvex; //该弧所指向的顶点的位置

struct ArcNode \*nextarc; //指向下一条弧的指针

int Weight; //用来记录网的权值

}ArcNode;

typedef struct VNode { //结点信息

VertexType data;//顶点信息

ArcNode \*firstarc; //指向第一条依附该顶点的弧的指针

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

typedef struct {//图信息

AdjList vertices;

int vexnum, arcnum; //图的当前顶点数和弧数

int kind; //图的种类标志

}aLGraph, \*ALGraph;

typedef struct Queue {

int \*base;

int capacity; //最大容量

int front;

int rear;

int now; //当前个数

}\*Queue;

ALGraph G[MAX\_VERTEX\_NUM];//存储多个图

int visited[MAX\_VERTEX\_NUM];//访问标记，0表示未访问过，1表示访问过

//图

int InitGraphs(void);//图置为空

ALGraph CreateGraph(ALGraph G);//创建图

ALGraph DestroyGraph(ALGraph T);//销毁图

status LocateVex(ALGraph G, VertexType ch);//给定数据，判定该节点位置

VNode\* GetVex(ALGraph G, VertexType ch);//获得指定顶点指针

status PutVex(ALGraph G, VertexType v, VertexType ch);//替换指定结点的值

status FirstAdjVex(ALGraph G, VertexType ch);//获得指定顶点第一个邻接点位置

status NextAdjVex(ALGraph G, VertexType v, VertexType ch);//返回下一个邻接点

status InsertVex(ALGraph G, VertexType ch);//插入顶点

status DeleteVex(ALGraph G, VertexType ch);//删除顶点

status InsertArc(ALGraph G, VertexType v1, VertexType v2);//新增一条弧

status DeleteArc(ALGraph G, VertexType v1, VertexType v2);//删除一条弧

status DFSTraverse(ALGraph G);//深度优先遍历

status DFS(ALGraph G, int v);

status BFSTraverse(ALGraph G);//广度优先遍历

int Save(ALGraph G);//保存到文件

ALGraph Load(ALGraph G);//从文件中读出

//队列操作函数

int InitQueue(Queue Q);//创建队列

int EnQueue(Queue Q, int e);//入队

int DeQueue(Queue Q);//出队

int IsQueueEmpty(Queue Q); //判空

int DestroyQueue(Queue Q);//摧毁队列

int main(void)

{

int i,op=1,back;

char e1,e2;

VNode\* p=NULL;

InitGraphs();

while (op)

{

system("cls");

system("color 1F");//系统命令，调颜色

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\* \*\n");

printf("\t\t\* ◆图的功能实现菜单 ◆ \*\n");

printf("\t\t\* \*\n");

printf("\t\t\* 0.退出系统 \*\n");

printf("\t\t\* 1.创建图 \*\n");

printf("\t\t\* 2.销毁图 \*\n");

printf("\t\t\* 3.查找顶点 \*\n");

printf("\t\t\* 4.获得顶点值 \*\n");

printf("\t\t\* 5.顶点赋值 \*\n");

printf("\t\t\* 6.获得第一邻接点 \*\n");

printf("\t\t\* 7.获得下一邻接点 \*\n");

printf("\t\t\* 8.插入顶点 \*\n");

printf("\t\t\* 9.删除顶点 \*\n");

printf("\t\t\* 10.插入弧 \*\n");

printf("\t\t\* 11.删除弧 \*\n");

printf("\t\t\* 12.深度优先遍历 \*\n");

printf("\t\t\* 13.广度优先遍历 \*\n");

printf("\t\t\* 14.存入文件 \*\n");

printf("\t\t\* 15.从文件中读数据 \*\n");

printf("\t\t\* \*\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

printf("\t\t 请输入你的选择：");

scanf("%d",&op);getchar();

switch(op)

{

case 1:

system("cls");

printf("请输入该图的编号（0-19）：");

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

if(G[i] == NULL)

{

printf("该图为空图！\n");

getchar();

break;

}

G[i]=CreateGraph(G[i]);

printf("成功创建图！\n");

getchar();

break;

case 2:

system("cls");

printf("请输入该图的编号（0-19）：");

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

if(G[i] == NULL)

{

printf("该图为空图！\n");

getchar();

break;

}

G[i]=DestroyGraph(G[i]);

printf("该图销毁成功！\n");

getchar();

break;

case 3:

system("cls");

printf("请输入该图的编号（0-19）：");

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

if(G[i] == NULL)

{

printf("该图为空图！\n");

getchar();

break;

}

printf("请输入需要查找的结点的值：");

e1=getchar();getchar();

back=LocateVex(G[i],e1);

if(back == -1)

printf("该结点不存在\n");

else

printf("该结点位置为%d\n",back);

getchar();

break;

case 4:

system("cls");

printf("请输入该图的编号（0-19）：");

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

if(G[i] == NULL)

{

printf("该图为空图！\n");

getchar();

break;

}

printf("请输入需要查找的结点的值：");

e1=getchar();getchar();

p=GetVex(G[i],e1);

if(p == NULL)

printf("该结点不存在\n");

else

printf("该结点值为%c\n",p->data);

getchar();

break;

case 5:

system("cls");

printf("请输入该图的编号（0-19）：");

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

if(G[i] == NULL)

{

printf("该图为空图！\n");

getchar();

break;

}

printf("请输入需要改变的结点的值：");

e1=getchar();getchar();

printf("改变后的值：");

e2=getchar();getchar();

if(PutVex(G[i],e1,e2))

printf("修改成功！\n");

else

printf("该结点不存在!\n");

getchar();break;

case 6:

system("cls");

printf("请输入该图的编号（0-19）：");

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

if(G[i] == NULL)

{

printf("该图为空图！\n");

getchar();

break;

}

printf("请输入需要查找的结点的值：");

e1=getchar();getchar();

back=FirstAdjVex(G[i],e1);

if(back == -1)

printf("该第一邻接点不存在\n");

else

printf("该第一邻接点位置为%d\n",back);

getchar();break;

case 7:

system("cls");

printf("请输入该图的编号（0-19）：");

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

if(G[i] == NULL)

{

printf("该图为空图！\n");

getchar();

break;

}

printf("请输入需要查找的结点的值：");

e1=getchar();getchar();

printf("请输入指定的邻接点：");

e2=getchar();getchar();

back=NextAdjVex(G[i],e1,e2);

if(back == -1)

printf("该邻接点不存在\n");

else

printf("该邻接点位置为%d\n",back);

getchar();break;

case 8:

system("cls");

printf("请输入该图的编号（0-19）：");

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

printf("请输入需要插入的结点的值：");

e1=getchar();getchar();

back=InsertVex(G[i],e1);

printf("插入成功！\n");

getchar();break;

case 9:

system("cls");

printf("请输入该图的编号（0-19）：");

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

if(G[i] == NULL)

{

printf("该图为空图！\n");

getchar();

break;

}

printf("请输入需要删除的结点的值：");

e1=getchar();getchar();

back=DeleteVex(G[i],e1);

if(back == 0)

printf("该结点不存在\n");

else

printf("删除成功！\n");

getchar();break;

case 10:

system("cls");

printf("请输入该图的编号（0-19）：");

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

if(G[i] == NULL)

{

printf("该图为空图！\n");

getchar();

break;

}

printf("请输入新增的弧的第一个结点的值：");

e1=getchar();getchar();

printf("请输入新增的弧的第二个结点的值：");

e2=getchar();getchar();

back=InsertArc(G[i],e1,e2);

if(back == 0)

printf("结点不存在\n");

else

printf("插入成功！\n");

getchar();break;

case 11:

system("cls");

printf("请输入该图的编号（0-19）：");

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

if(G[i] == NULL)

{

printf("该图为空图！\n");

getchar();

break;

}

printf("请输入删除的弧的第一个结点的值：");

e1=getchar();getchar();

printf("请输入删除的弧的第二个结点的值：");

e2=getchar();getchar();

back=DeleteArc(G[i],e1,e2);

if(back == 0)

printf("结点不存在\n");

else

printf("删除成功！\n");

getchar();break;

case 12:

system("cls");

printf("请输入该图的编号（0-19）：");

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

if(G[i] == NULL)

{

printf("该图为空图！\n");

getchar();

break;

}

printf("图的深度优先遍历结果为：\n");

DFSTraverse(G[i]);

getchar();break;

case 13:

system("cls");

printf("请输入该图的编号（0-19）：");

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

if(G[i] == NULL)

{

printf("该图为空图！\n");

getchar();

break;

}

printf("图的广度优先遍历结果为：\n");

BFSTraverse(G[i]);

getchar();break;

case 14:

system("cls");

printf("请输入该图的编号（0-19）：");

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

if(G[i] == NULL)

{

printf("该图为空图！\n");

getchar();

break;

}

if(Save(G[i]))

printf("文件保存成功！\n");

else

printf("文件保存失败！\n");

getchar();break;

case 15:

system("cls");

printf("请输入该图的编号（0-19）：");

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

G[i]=Load(G[i]);

printf("文件读取成功！\n");

getchar();break;

}

}

printf("欢迎下次再使用本系统！\n");

}

int InitQueue(Queue Q)

{

Q->base = (int \*)malloc(sizeof(int) \* MAX\_VERTEX\_NUM);

Q->capacity = MAX\_VERTEX\_NUM;

Q->front = 0;

Q->rear = 0;

Q->now = 0;

return OK;

}

int EnQueue(Queue Q, int e)

{

Q->now++;

Q->base[Q->rear] = e;

Q->rear++;

return OK;

}

int DeQueue(Queue Q)

{

int e;

if (IsQueueEmpty(Q))

return -1;

else

{

e = Q->base[Q->front];

Q->now--;

Q->front++;

return e;

}

}

int IsQueueEmpty(Queue Q)

{

if (Q->now == 0)

return TRUE;

else

return FALSE;

}

int DestroyQueue(Queue Q)

{

free(Q->base);

free(Q);

Q = NULL;

return OK;

}

int InitGraphs(void)

{

int i;

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

G[i] = NULL;

return OK;

}

ALGraph CreateGraph(ALGraph T)

{

int i, j, weight;

char tou, wei;

ArcNode \*p;

T = (ALGraph)malloc(sizeof(aLGraph));

printf("请输入图的类型:");

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

if(i<1 || i>4)

{

printf("类型错误！\n");

return NULL;

}

T->kind=i;

printf("请输入图的顶点数:");

scanf("%d", &T->vexnum); getchar();

printf("请输入图的边数:");

scanf("%d", &T->arcnum); getchar();

printf("有向图:0 有向网:1 无向图:2 无向网:3\n");

for (i = 0; i < T->vexnum; i++)

{

printf("输入第%d个结点的值:", i);

scanf("%c", &T->vertices[i].data); getchar();

T->vertices[i].firstarc = NULL;

}

for (i = 0; i < T->arcnum; i++)

{

if (T->kind == 1 || T->kind == 3)

{

printf("请输入第%d条弧的权值:",i);

scanf("%d", &weight); getchar();

printf("请输入弧头:");

scanf("%c", &tou); getchar();

printf("请输入弧尾:");

scanf("%c", &wei); getchar();

}

else

{

printf("请输入弧头:");

scanf("%c", &tou); getchar();

printf("请输入弧尾:");

scanf("%c", &wei); getchar();

}

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

j = LocateVex(T, wei);

i = LocateVex(T, tou);

p->adjvex = j;

if (T->kind == 1 || T->kind == 3)

p->Weight = weight;

else

p->Weight = 0;

p->nextarc = T->vertices[i].firstarc;//插在表头

T->vertices[i].firstarc = p;

if (T->kind >= 2)//为无向的话，需要额外增加弧

{

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

p->adjvex = i;

p->nextarc = T->vertices[j].firstarc; //插在表头

T->vertices[j].firstarc = p;

}

}

return T;

}

ALGraph DestroyGraph(ALGraph T)

{

int i;

ArcNode \*p, \*pcrs;

if (T == NULL)

return NULL;

for (i = 0; i < T->vexnum; i++)

{

p = T->vertices[i].firstarc;

while (p != NULL)

{

pcrs = p->nextarc;

free(p);

p = pcrs;

}

}

free(T);

T = NULL;

return T;

}

status LocateVex(ALGraph G, VertexType ch)

{

int i;

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

if (G->vertices[i].data == ch)

return i;

return -1;

}

VNode\* GetVex(ALGraph G, VertexType ch)

{

int e;

e = LocateVex(G, ch);

if (e != -1)

return &G->vertices[e];

else

return NULL;

}

status PutVex(ALGraph T, VertexType v, VertexType ch)

{

int e;

e = LocateVex(T, v);

if (e != -1)

{

T->vertices[e].data = ch;

return OK;

}

else

return 0;

}

status FirstAdjVex(ALGraph G, VertexType ch)

{

ArcNode \*p;

int e;

e = LocateVex(G, ch);

if (e == -1)

return -1;

else

{

p = G->vertices[e].firstarc;

if (p != NULL)

return p->adjvex;

else

return -1;

}

}

status NextAdjVex(ALGraph G, VertexType v, VertexType ch)

{

ArcNode \*p;

int e1, e2;

e1 = LocateVex(G, v);

e2 = LocateVex(G, ch);

if (e1 == -1 || e2 == -1)

return -1;

p = G->vertices[e1].firstarc;

while ((p != NULL) && (p->adjvex != e2))

p = p->nextarc;

if (p == NULL)

return -1;

else if (p->nextarc == NULL)

return -1;

else

return p->nextarc->adjvex;

}

status InsertVex(ALGraph T, VertexType ch)//插在最后的位置

{

if(LocateVex(T,ch) != -1)

{

printf("结点已经存在！\n");

return FALSE;

}

T->vertices[T->vexnum].data = ch;

T->vertices[T->vexnum].firstarc = NULL;

T->vexnum++;

return OK;

}

status DeleteVex(ALGraph G, VertexType ch)

{

ArcNode \*p, \*pcrs;

int i, j;

j = LocateVex(G, ch);

if (j == -1)

return ERROR;

p = G->vertices[j].firstarc;

while (p)//释放所有以ch为头结点的弧

{

pcrs = p;

p = p->nextarc;

free(pcrs);

G->arcnum--;

}

for (i = j; i < G->vexnum - 1; i++) //所有节点前移一位

G->vertices[i] = G->vertices[i + 1];

G->vexnum--;

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

{

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

while (p != NULL)

{

if (p->adjvex != j)

{

if (p->adjvex > j)//节点序号改变

p->adjvex--;

pcrs = p;

p = p->nextarc;

}

else

{

if (G->vertices[i].firstarc == p)

{

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

free(p);

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

if (G->kind < 2)

G->arcnum--;

}

else

{

pcrs->nextarc = p->nextarc;

free(p);

p = pcrs->nextarc;

if (G->kind < 2)

G->arcnum--;

}

}

}

}

return OK;

}

status InsertArc(ALGraph T, VertexType v1, VertexType v2)

{

ArcNode \*p,\*adj;

int i, j;

i = LocateVex(T, v1);

j = LocateVex(T, v2);

if (i == -1 || j == -1)

return ERROR;

adj=T->vertices[i].firstarc;

while(adj != NULL)

{

if(adj->adjvex == j)

return FALSE;

adj=adj->nextarc;

}

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

if (T->kind == 2 || T->kind == 4)

{

printf("请输入弧的权值:");

scanf("%d", &p->Weight); getchar();

}

else

p->Weight = 0;

p->adjvex = j;

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

T->vertices[i].firstarc = p;

T->arcnum++;

if (T->kind >= 2)

{

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

p->adjvex = i;

p->nextarc = T->vertices[j].firstarc;

T->vertices[j].firstarc = p;

}

return OK;

}

status DeleteArc(ALGraph T, VertexType v1, VertexType v2)

{

int i, j;

ArcNode \*p, \*pcrs;

i = LocateVex(T, v1);

j = LocateVex(T, v2);

if (i == -1 || j == -1)

return ERROR;

p = T->vertices[i].firstarc;

if (p->adjvex == j)

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

else

{

while ((p != NULL) && (p->adjvex != j))

{

pcrs = p;

p = p->nextarc;

}

}

if (p->adjvex == j)

{

pcrs->nextarc = p->nextarc;

free(p);

T->arcnum--;

if (T->kind >= 2)

{

p = T->vertices[j].firstarc;

if (p->adjvex == i)

T->vertices[j].firstarc = p->nextarc;

else

{

while ((p != NULL) && (p->adjvex != i))

{

pcrs = p;

p = p->nextarc;

}

}

if (p->adjvex == i)

{

pcrs->nextarc = p->nextarc;

free(p);

}

if (p == NULL)

return FALSE;

}

}

if (p == NULL)

return FALSE;

return OK;

}

status DFSTraverse(ALGraph G)

{

int i, j;

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

visited[i] = 0;

for (j = 0; j < G->vexnum; j++)

if (visited[j] == 0)

DFS(G, j);

return OK;

}

status DFS(ALGraph G, int v)

{

ArcNode\* arc;

if (visited[v] != 1)

{

printf("%d %c\n", v, G->vertices[v].data);

visited[v] = 1;

}

arc = G->vertices[v].firstarc;

while (arc != NULL)

{

if (!visited[arc->adjvex])

DFS(G, arc->adjvex);

arc = arc->nextarc;

}

return OK;

}

status BFSTraverse(ALGraph G)

{

int i, v;

ArcNode\* arc;

Queue Q;

Q = (Queue)malloc(sizeof(struct Queue));

InitQueue(Q);

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

visited[i] = 0;

for (v = 0; v < G->vexnum; v++)

{

if (visited[v] == 0)

{

EnQueue(Q,v);

while (!IsQueueEmpty(Q))

{

v = DeQueue(Q);

if (visited[v] != 1)

{

printf("%d %c\n", v, G->vertices[v].data);

visited[v] = 1;

}

arc = G->vertices[v].firstarc;

for(;arc != NULL;arc = arc->nextarc)

EnQueue(Q, arc->adjvex);

}

}

}

DestroyQueue(Q);

return OK;

}

int Save(ALGraph G)

{

FILE \*fp;

int i;

ArcNode\* arc;

if ((fp = fopen("shuju.txt", "w+")) == NULL)

{

printf("文件打开失败！\n");

return ERROR;

}

fprintf(fp, "结点数： %d\n", G->vexnum);

fprintf(fp, "弧数： %d\n", G->arcnum);

fprintf(fp, "图的种类为： %d\n", G->kind);

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

fprintf(fp, "%c ", G->vertices[i].data);

fprintf(fp, "\n");

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

{

fprintf(fp, "%d %c ", i, G->vertices[i].data);

arc = G->vertices[i].firstarc;

while (arc != NULL)

{

fprintf(fp, "%d ", arc->adjvex);

arc = arc->nextarc;

}

fprintf(fp, "-1\n");

}

fclose(fp);

return OK;

}

ALGraph Load(ALGraph G)

{

FILE \*fp;

char line[256];

int i, e;

G = (ALGraph)malloc(sizeof(aLGraph));

fp = fopen("shuju.txt", "r");

fgets(line, 250, fp);

sscanf(line, "%\*s%d", &(G->vexnum));

fgets(line, 250, fp);

sscanf(line, "%\*s%d", &G->arcnum);

fgets(line, 250, fp);

sscanf(line, "%\*s%d", &G->kind);

fgets(line, 250, fp);

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

G->vertices[i].firstarc = NULL;

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

G->vertices[i].data = line[i \* 2];

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

{

fscanf(fp, "%\*d %c", &G->vertices[i].data);

fscanf(fp, "%d", &e);

while (e != -1)

{

InsertArc(G, G->vertices[i].data, G->vertices[e].data);

G->arcnum--;

fscanf(fp, "%d", &e);

}

fgetc(fp);

}

fclose(fp);

return G;

}