

**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201707**

**学 号： U201714801**

**姓 名： 杨旭东**

**指导教师： 祝建华**

**报告日期： 2018.12.30**

**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](#_Toc533849803)

[1.1顺序表演示系统设计 2](#_Toc533849804)

[1.2 顺序表演示系统设计 2](#_Toc533849805)

[1.3 顺序表的系统实现和测试 11](#_Toc533849806)

[1.4 实验小结 15](#_Toc533849807)

[2 基于链式存储结构的线性表实现 1](#_Toc533849808)

[2.1 问题描述 1](#_Toc533849809)

[2.2 链表演示系统设计 1](#_Toc533849810)

[2.3 单链表系统实现与测试 8](#_Toc533849811)

[2.4 实验小结 12](#_Toc533849812)

[3 基于二叉链表的二叉树实现 14](#_Toc533849813)

[3.1 实验目的 14](#_Toc533849814)

[3.2 链表演示系统设计 14](#_Toc533849815)

[3.3 顺序表的系统实现和测试 24](#_Toc533849816)

[3.4 实验小结 31](#_Toc533849817)

[4 基于邻接表的图实现 34](#_Toc533849818)

[4.1 实验目的 34](#_Toc533849819)

[4.2 图演示系统设计 34](#_Toc533849820)

[4.3 图的系统实现和测试 41](#_Toc533849821)

[4.4实验小结 47](#_Toc533849822)

[参考文献 48](#_Toc533849823)

[附录A 基于顺序存储结构的线性表的源程序 1](#_Toc533849824)

[附录B 基于链式存储结构的线性表的源程序 12](#_Toc533849825)

[附录C 基于二叉链表二叉树的源程序 25](#_Toc533849826)

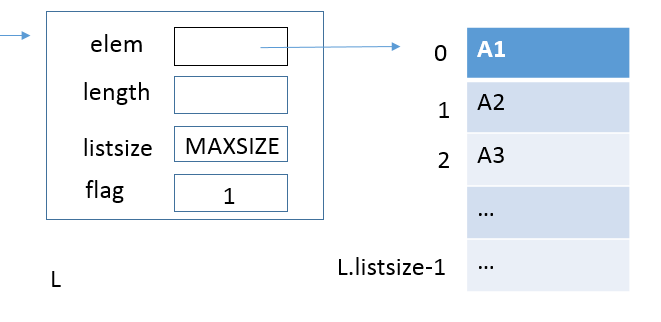
[附录D 基于邻接表的图的源程序 55](#_Toc533849827)

### 基于顺序存储结构的线性表实现

## 1.1顺序表演示系统设计

本次试验目的是实现顺序表的基本操作，用菜单的形式完整的实现各个功能，并且相互配合，能将顺序表的相关知识融会贯通。还增加了文件的读写功能方便用户更好的保存当前的数据和读取以前的数据，提供更好的用户体验。

项目中数据元素之间的逻辑关系如图1-1。



**图 1-1**

## 顺序表演示系统设计

**1.2.1 系统总体设计**

**系统中提供了总共18个命令来实现对多表的操作，包括添**

**加、删除、切换3个简单命令，以及对单表的插入，删除，遍历…,如图1-2所示。**

****

**图 1-2**

**各个模块的相关功能描述如下：**

**（1）status IntiaList(SqList &L);**

**初始条件：表不存在**

**实现结果：初始化顺序表**

**（2）status DestroyList(SqList &L);**

**初始条件：表存在**

**实现结果：销毁顺序表**

**（3）status ClearList(SqList &L);**

**初始条件：表存在**

**实现结果：清空顺序表**

**（4）status ListEmpty(SqList L);**

**初始条件：表存在**

**实现结果：判断顺序表是否为空**

**（5）int ListLength(SqList L,ElemType &e);**

**初始条件：表存在**

**实现结果：返回表长**

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

**初始条件：表存在并且i合法**

**实现结果：查找第i个元素，并用e返回其值**

**（7）status LocateElem(SqList L,ElemType e);**

**初始条件：表存在**

**实现结果：查找元素e的位置并返回**

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

**初始条件：表存在**

**实现结果：查找元素cur的前驱并用pre\_e返回**

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

**初始条件：表存在**

**实现结果：查找cur的后继并用next\_e返回**

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

**初始条件：表存在并且i合法**

**实现结果：在第i个位置插入元素e**

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

**初始条件：表存在并且i合法**

**实现结果：删除第i个元素并用e返回其值**

**（12）status ListTravers(SqList L);**

**初始条件：表存在且不为空**

**实现结果：输出表中的元素**

**（13）status ListSave(SqList &L);**

**初始条件：表存在**

**实现结果：将当前表中的内容保存在文件中**

**（14）status List\_Load\_from\_File(SqList&L);**

**初始条件：文档存在**

**实现结果：如果文档中有数据，选择加载到当前表后部或者新建表加载,亦可以选择清空当前表再加载**

**（15）status Add\_List(M\_List &M);**

**实现结果：添加一张新表**

**（16）status Change\_List(M\_List &M)**

**初始条件：列表中表不为空**

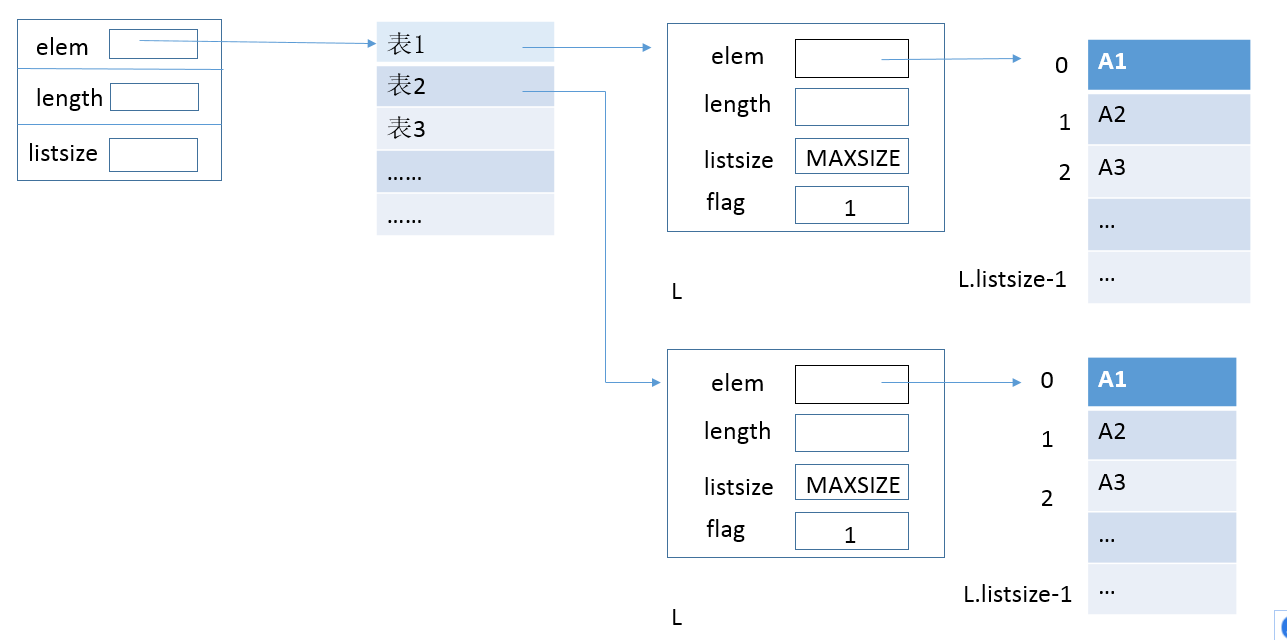
**实现结果：如果表中元素不止一个，切换到另一个表进行操作**

**（17）status List\_Delete(M\_List &M);**

**初始条件：列表不为空**

**实现结果：删除当前表**

**通过用顺序表管理顺序表的方法实现多表操作，结构如图1-3所示。**

****

**表1-3**

* + 1. **有关常量的类型和定义**

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define FALSE -1

#define INFEASIBLE -1

#define OVERFLOW -2

//顺序表结构

typedef struct

{

ElemType \*elem;

int length;

int listsize;

int flag; //用于标记表初始化状态

}SqList;

//管理线性表的多表结构类型

typedef struct //用于保存多个线性表的结构

{

SqList\* Sq; //顺序表指针

int length;

int listsize;

}M\_List;

数据类型以及函数返回状态定义：

typedef int status;

typedef int ElemType;

线性表初始化容量,以及容量不足时扩充容量等：

#define LIST\_INIT\_SIZE 10

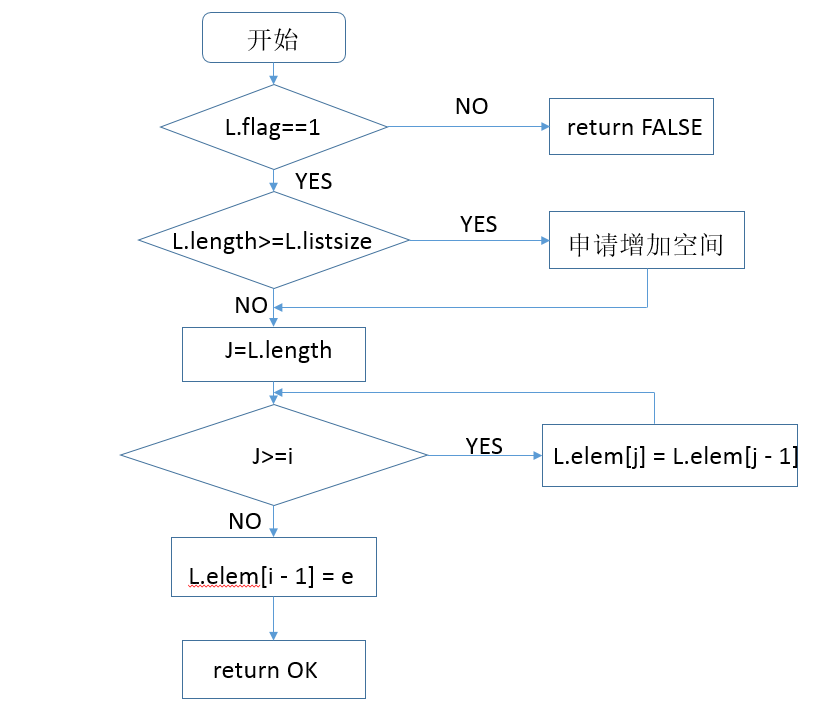
#define LISTINCREMENT 10

#define MAXSIZE 10

* + 1. **算法设计**

实验中需完成对顺序表的初始化，插入，销毁，查找……等17个操作，函数清单请见1.2.1，源代码请看附录A,现选择顺序表插入，查找后继，元素位置的查找，写入文件，删除数据等几个比较难的算法进行设计和分析，如下：

算法1、顺序表的插入ListInsert(SqList &L,I , e)

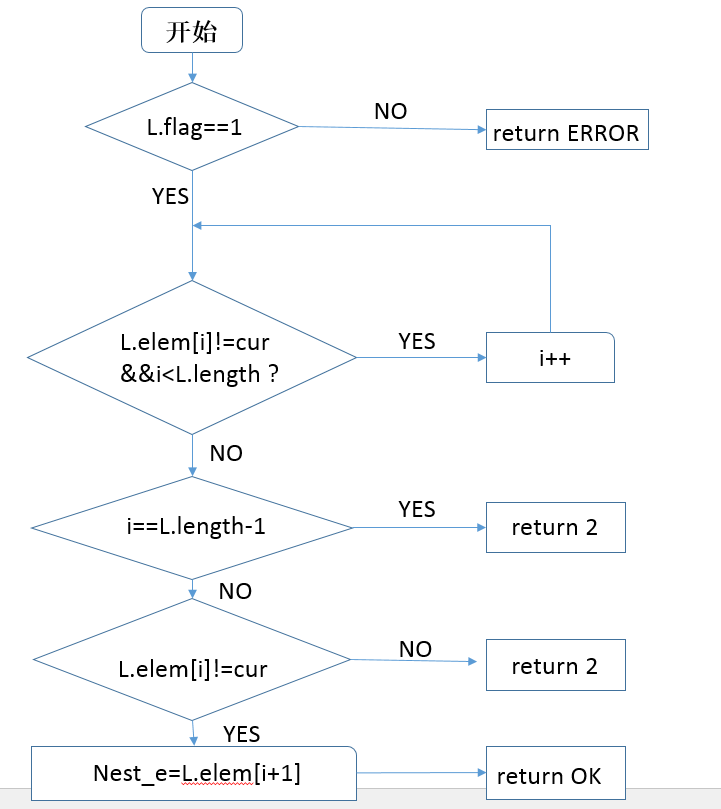


**图1-4**

算法分析：寻找插入结点O（n），其他步骤均为O（1），故总时间复杂度为O（n）。最坏的情况考虑，由于需要移动n个元素，所以空间复杂度也为O(n).

算法2：查找后继元素NextElem(L, cur, &next\_e)

程序框图如图1-5。

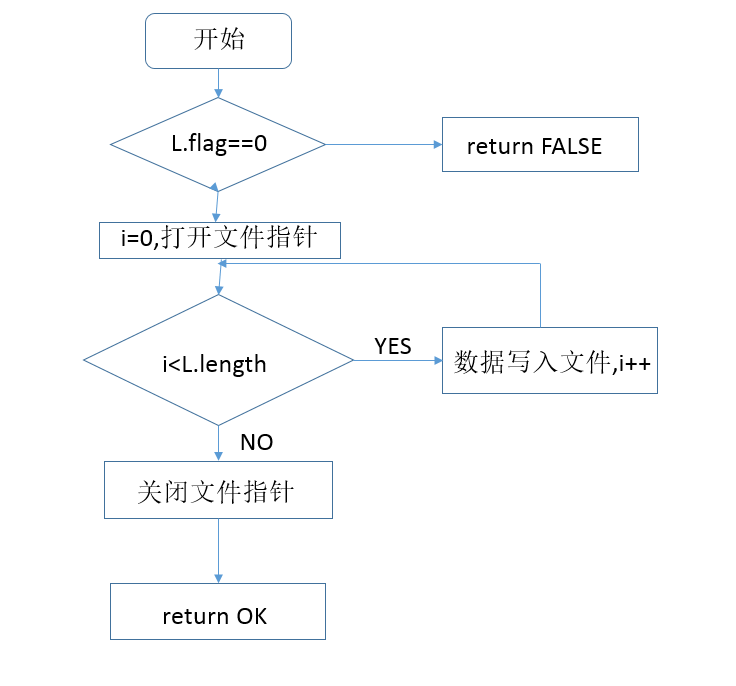


**图1-24**

算法分析：查找元素时最坏情况下需要遍历整个顺序表，时间复杂度为O(n),其他步骤时间复杂度均为O(1),因此总的时间复杂度为O(n).空间复杂度为O(1).

算法3：数据写入文件ListSave(SqList &L)

程序框图如图1-6。

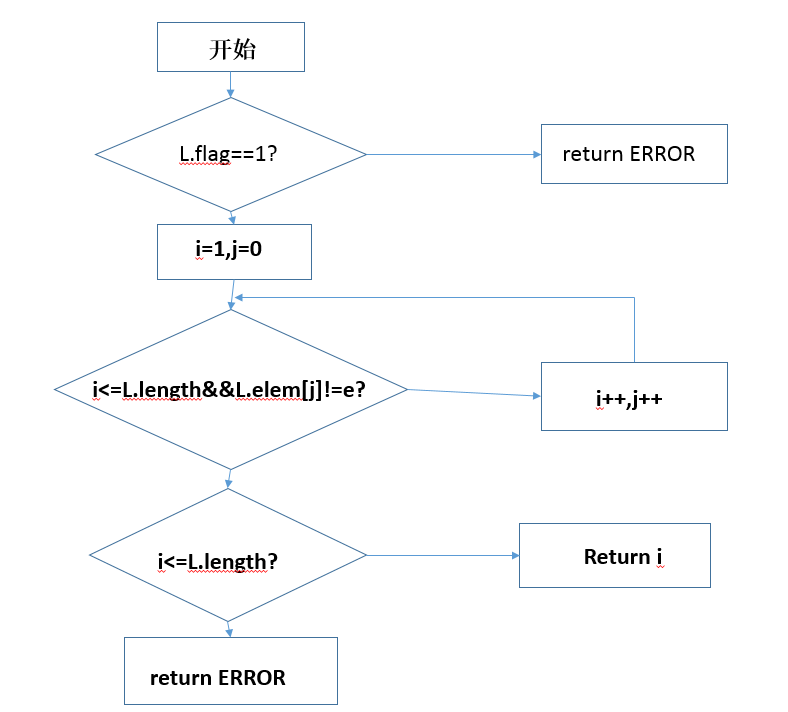


**图1-6**

算法分析：数据的遍历时间复杂度为O(n),文件的写入时间复杂度为O(n),总的复杂度为O(n),空间复杂度为O(n).

算法4：查找元素位置LocateElem(L, e)

程序框图如下：

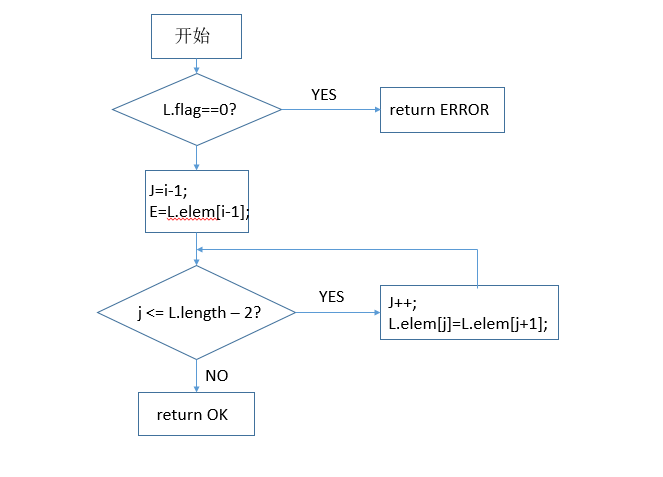


**图1-7**

算法分析：需要对顺序表进行遍历，时间复杂度为O(n),空间复杂度为O(n).

算法5：删除元素ListDelete(SqList &L,int i , ElemType &e)

程序框图如图：



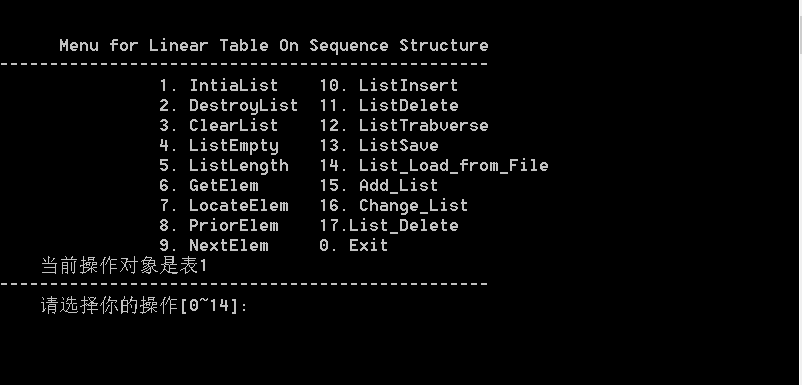
**图1-8**

删除过程平均需要移动一半的数据元素，因此时间复杂度为O(n).空间复杂度为O(n).

## 1.3 顺序表的系统实现和测试

**1.3.1 系统实现**

此次我所使用的编译环境是VS2013，本次试验中，我提供了18个命令的菜单，通过这个菜单，可以实现顺序表的初始化，销毁，清空，插入，删除，查找，定位，获取头元素，获取表长，判断是否为空，查找元素前驱，查找元素后继，保存顺序表元素到文件，加载文件等18个功能，如图1.31



**图1-9**

**1.3.2 测试用例及结果**

**表1-1**

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | 描述 | 预期结果 | **运行结果** |
| （1） | 初始化顺序表 | 如果表不存在，则初始化成功 | **C:\Users\Administrator\Desktop\2.png** |
| （2） | 依次插入11，22，33 | 成功插入11，22，33 | **C:\Users\Administrator\Desktop\1.png** |
| （3） | 遍历顺序表 | 输出表中所有元素 | **C:\Users\Administrator\Desktop\捕获.PNG** |
| （4） | 将数据保存在文件中 | 显示保存成功 | **C:\Users\Administrator\Desktop\捕获.PNG** |
| (5) | 查看第1个元素和第6个元素 | 第一个元素存在，输出33，第6个元素不存在，提示出错 | **C:\Users\Administrator\Desktop\捕获.PNGC:\Users\Administrator\Desktop\1.PNG** |
| （6） | 查看元素33和11的后继元素 | 33后继为22，11没有后继，提示没有后继 | **C:\Users\Administrator\Desktop\捕获.PNGC:\Users\Administrator\Desktop\1.PNG** |
| （7） | 删除元素22，遍历表 | 删除后只剩33和11 | **C:\Users\Administrator\Desktop\捕获.PNGC:\Users\Administrator\Desktop\1.PNG** |
| (8) | 查找元素33和44是否在表中，如果在，返回它们的位置 | 33在表中，输出位置为1,44不在表中，输出不存在 | **C:\Users\Administrator\Desktop\2.PNGC:\Users\Administrator\Desktop\1.PNG** |
| （9） | 切换顺序表操作 | 从当前顺序表切换到其他表操作 | **C:\Users\Administrator\Desktop\2.PNGC:\Users\Administrator\Desktop\1.PNG** |

**上面一共测试了9组用例，程序均成功的执行完毕，没有发生异常，完美达到预期效果，文件的写入和读取也正常执行，基本达到预期规划。**

## 1.4 实验小结

本次实验，让我对顺序表的知识又系统的学习了一遍，单个算法写起来很容易，但是把这十几个算法组合到一起，并且要防止其相互干扰，并要让他们相互配合，互相补充，实现起来却很麻烦。实验中遇到不少问题。

1. 如果顺序表存在的情况下用户不小心又从新初始化，那么以前的数据和空间就都浪费了，如何判断顺序表已经存在呢？一开始我选择判断L.elem是否存在，但是并没有达到目的，是乎起不到判断的作用，然后我在结构中增加了一个flag标志，初始化之后将flag置为1，不存在时为0，成功解决了问题。
2. 数据写入文件，刚开始我用fprintf函数写入文件数据，然后用fread函数读取数据，然而能正常写入，但是读取却总是失败，我找了很久原因也没有找到，后来我改用fwrite函数，问题就解决了。

这次实验总体来说不是很难，算法也都比较简单，但是确实整个线性数据结构的基础，因此学好这部分的内容十分重要，总之收获颇丰。

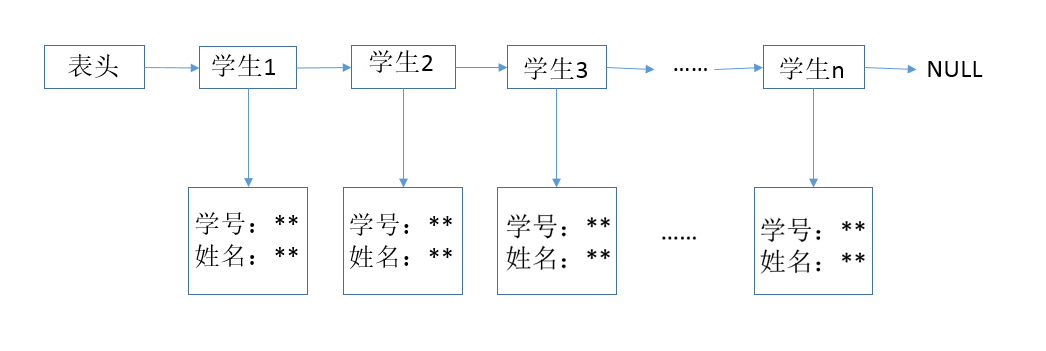
1. 对于多表操作，我完成单表的操作之后又定义一个结构来对多表进行操作，依然用顺序表的结构类型，本以为很简单，然而我整个代码中都没有定义结构指针，我用L=M.elem[count]语句来对切换顺序表，但是我忽略了L只是一个结构类型而不是指针类型，这只是一个赋值语句，对表的操作没有保存下来。这个问题折磨了我一个多小时，真的好粗心，后来我放弃了L,直接将L.elem[count]传过去，很快就成功了。

### 基于链式存储结构的线性表实现

## 问题描述

本次试验目的是做一个用链表来管理学生系统的控制台应用程序，同时还提供了多表操作，如表一存储一班学生信息，表二存储2班学生信息，以此达到对整个学校的学生系统化管理的目的。程序中提供了16个命令对链表进行操作，比如创建，销毁，清空，求表长，查找元素位置，查找元素前驱后继，删除，插入等等，在完成实验目的的基础上又增加了对表中元素的保存和加载等操作，基本形成了一个功能相对完善的小程序。单个班级的信息用链表结构保存，而多张表用顺序表的结构进行保存。

单个表的存储结构如图2-1所示。

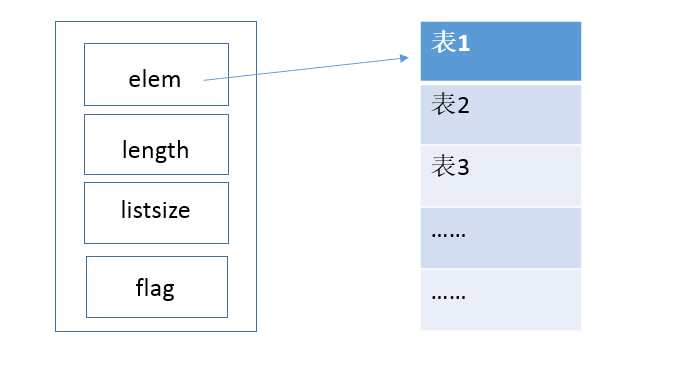


**图2-1**

## 链表演示系统设计

**2.2.1 系统总体设计**

系统中我提供了17个命令来实现对链表的操作，包括初始化，销毁，清空，查找，遍历，查找前驱、后继，获取元素位置，插入……以及多表的添加和切换。考虑到数据写入的方便性，我将学号和姓名分开存放在两个不同的文件中，读取的时候按照一定的顺序读取便不会出错。因为可能会出现两个人同名的情况，所以我查找元素时都是基于学号进行操作的。学号和姓名用一个结构体进行保存，因此如果需要的话还可以很方便的增加更多的属性，例如家庭住址，身份证号码等等，程序的可扩展性很高。利用一个顺序表，又可以所有班级进行管理，可以很方便的切换到不同的班级进行修改，多表的物理存储结构如图2-1所示。

****

**图2-2**

各个命令模块的功能见表2-1。

**表2-1**

|  |  |  |  |
| --- | --- | --- | --- |
| 命令 | 函数 | 初始条件 | 操作结果 |
| 1、初始化链表 | IntiaList(SqList &L) | 表L不存在 | 初始化L |
| 2、销毁链表 | DestroyList(SqList &L) | 表L存在 | 销毁L |
| 3、清空表 | ClearList(SqList &L) | 表L存在 | 将表中数据清空 |
| 4、判断表空 | ListEmpty(SqList L) | 表 L存在 | 表空返回TRUE，否则返回FALSE |
| 5、求表长 | ListLength(SqList &L) | 表 L存在 | 返回表长 |
| 6、获取元素 | GetElem(L, i , e) | L存在，0<i<Listlength(L) | 用e 返回第i 个元素 |
| 7、查找元素位置 | LocateElem(L, e, loc) | 表 L存在 | 如果元素e 存在表中，用loc返回位置 |
| 8、查找元素前驱 | PriorElem(L ,cur, pre\_e) | 表 L存在，cur在表中且存在前驱 | 用结构pre\_e 返回cur的前一个学生信息 |
| 9、查找元素后继 | NextElem(L, cur, next\_e) | 表 L存在，cur在表中且存在后继 | 用结构next\_e 返回cur的后一个学生信息 |
| 10、插入元素 | ListInsert(SqList &L, int i) | 表L存在，且i 的位置合法 | 在第i 个位置插入元素 |
| 11、删除元素 | ListDelete(L, i, e) | 表L存在，且i 的位置合法 | 删除第i个元素，并用结构指针返回被删除的结构 |
| 12、遍历链表 | ListTrabverse(SqList L) | 表L存在 | 遍历并输出L中的信息 |
| 13、保存数据 | Save\_List(SqList L) | 表L存在 | 将L中的信息保存到文件中 |
| 14、从文件加载 | load\_List(SqList &L) | 文件中有数据且表L存在 | 将文件中的信息加载到L 尾部 |
| 15、添加新表 | Add\_List(M\_List &M) | 多表的顺序存储结构已经初始化 | 增加一张表 |
| 16、切换链表 |  | 存在多张表 | 顺序切换到下一张表 |
| 0、 |  |  | 退出程序 |

#### 2.2.2 有关常量的类型和定义

#define TRUE 1

#define FALSE -1

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 10

#define LISTINCREMENT 10

typedef int status;

typedef int ElemType; //数据元素类型定义

#define MAXSIZE 20

//学生信息结构体

typedef struct student

{

char name[MAXSIZE];

int number;

}Student;

//链表结构体

typedef struct SqList

{

Student\* Stu;

struct SqList\*next;

int flag;

}SqList, \*LinkList;

//用于保存多个线性表的结构

typedef struct

{

SqList \*elem; //顺序表指针

int length; //表长

int listsize; //表最大容量

}M\_List;

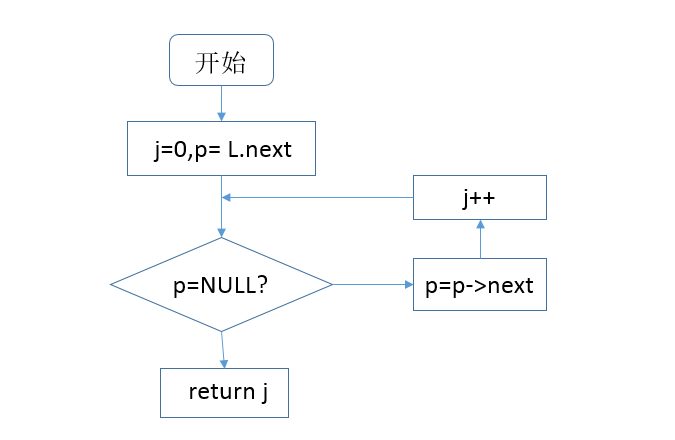
#### 2.2.3 算法设计

程序中共17个命令，有15个函数，函数清单已在上面表格中给出，现对其中比较难的几个算法进行设计和分析：

算法（1）：获取表长度ListLength(SqList &L)

这个算法比较简单只需要在遍历链表的同时每次计数器加1就可以了。

程序框图如下：



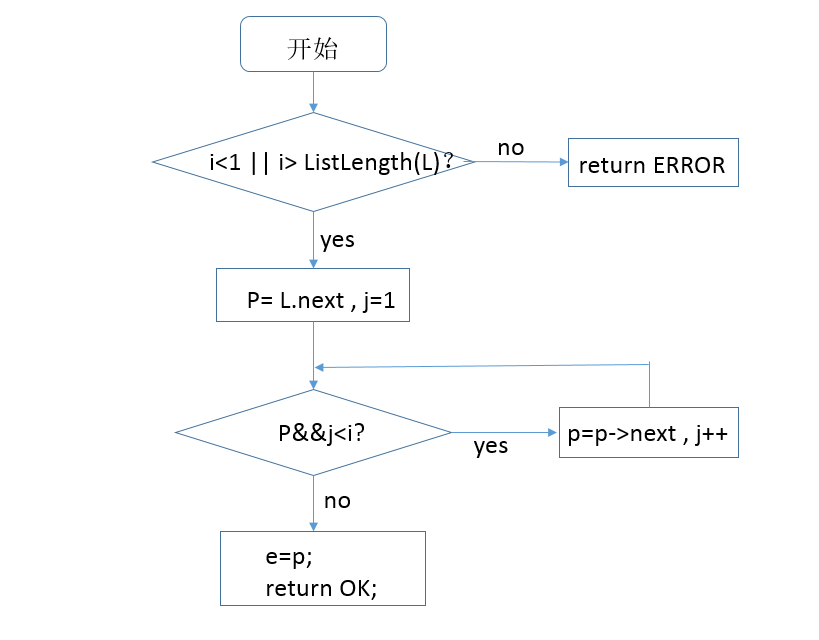
**图 2-3**

整个过程只需对链表循环一遍即可得到结果，时间复杂度为O(n);空间复杂度为O(n).

算法2：获取第i 个元素GetElem(L, i , e)

1. 如果i的值超出正常范围则返回ERROR。
2. 对表进行遍历，用一个计数器进行判断知道找到第i个元素，将它赋值给e，返回OK。

算法的程序框图如下。



**图 2-4**

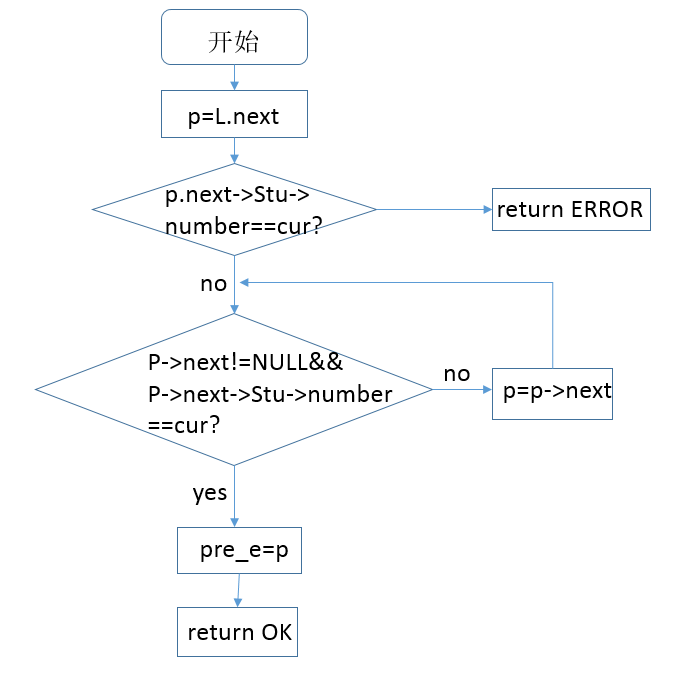
算法分析：此算法也是基于链表的遍历的基础上操作的，找到第i个位置即返回。平均需要遍历一半长度链表，时间复杂度为O(n)，空间复杂度为O(n).

算法（3）：获取元素前驱PriorElem(L ,cur, pre\_e)

算法设计：（1）如果cur是第一个元素的值则返回ERROR，说明没有前驱。

（2）用一个指针p对链表进行遍历，如果p->next->stu->number==cur，则将p赋值给pre\_e，如果遍历完链表都没有找到，说明不存在，返回ERROR。

程序框图如图2-5.

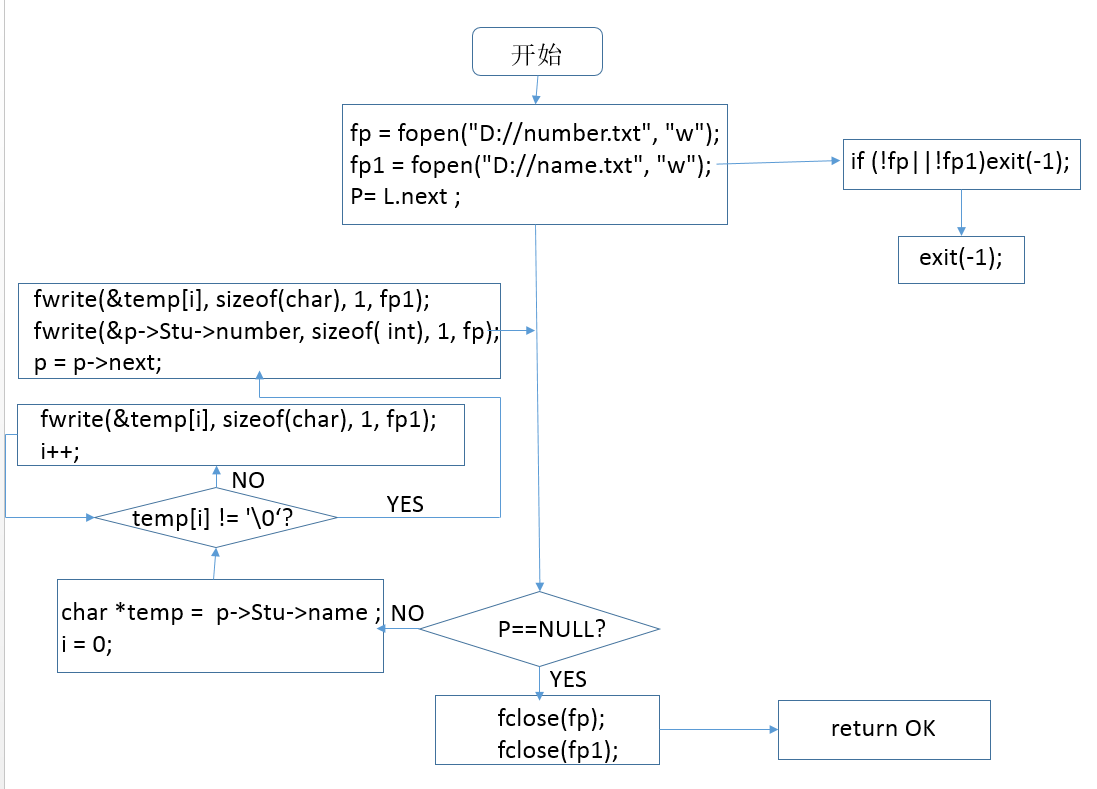


**图 2-5**

算法分析：该算法平均需要遍历一班长度链表，时间复杂度为O(n),其他步骤均为O(1),故总的时间复杂度为O(n).

算法（4）：链表元素写入文件Save\_List(SqList L)

算法设计思路：只需一边遍历链表，一边把读到的数据写入对应文件即可，程序框图如下：



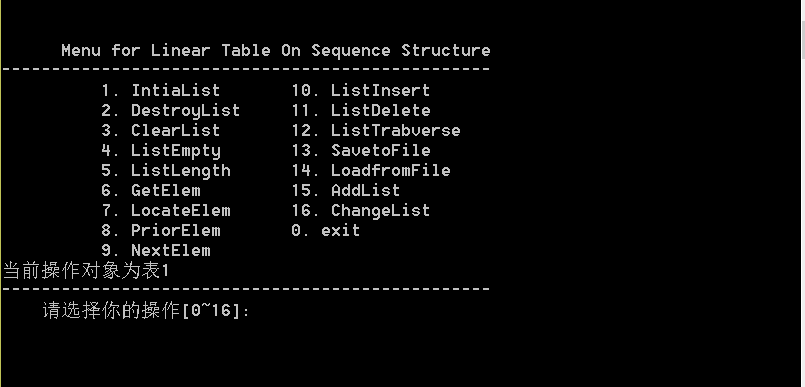
**图 2-6**

算法分析：此算法也是基于链表遍历的基础上进行操作的，思路也很简单，每遍历完一个节点便调用函数进行保存，时间复杂度为O(n),空间复杂度为O(n).

## 2.3 单链表系统实现与测试

**2.3.1 系统实现**

本次实现我使用的编译器是vs2013,操作系统是windows8，各函数基本独立，根据如图2-7所示的菜单可实现各个函数的调用。



**图 2-7**

**2.3.2 系统测试**

测试用例及其预期结果如下表所示。

**表2-2**

|  |  |  |  |
| --- | --- | --- | --- |
| 实验用例 | 描述 | 测试数据 | 预期结果 |
| （1） | 初始化链表 | 无 | 显示初始化成功 |
| （2） | 初始位置插入1个学生 | 位置i:1  姓名：露露  学号：2015 | 插入成功，遍历链表时能成功输出 |
| （3） | 在第1个和第5个位置插入 | 位置i:1  姓名：梅梅  学号：2012  位置i:5  姓名：花花  学号：2017 | 第一个位置正确插入，第5个插入失败 |
| （4） | 根据学号查找元素位置 | 学号：2010  学号：2011 |  |
| （5） | 根据学号查找前驱 | 学号：2010  学号：2011 |  |
| （6） | 根据学号查找后继 | 学号：2019  学号：2014 |  |

测试结果：

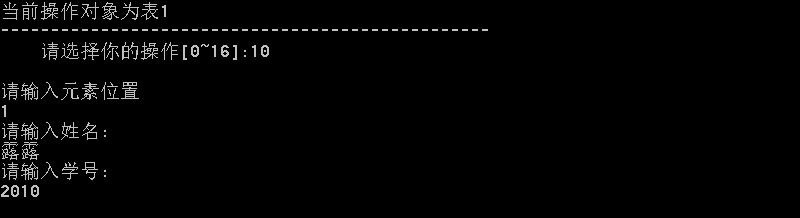
1. 操作结果如图2-8.



**图2-8**

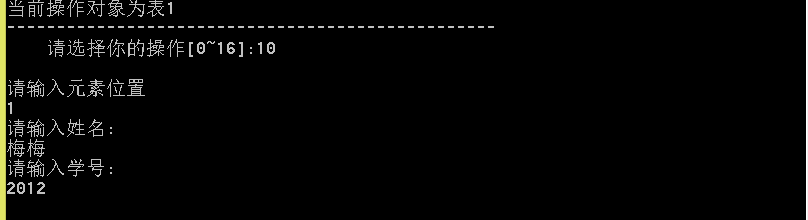
评价：初始化成功，达到实验目的。

1. 操作结果如图2-9.

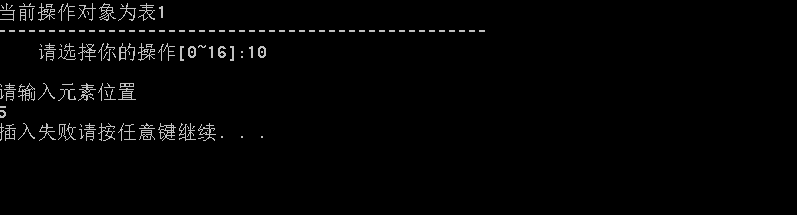


**图 2-9**

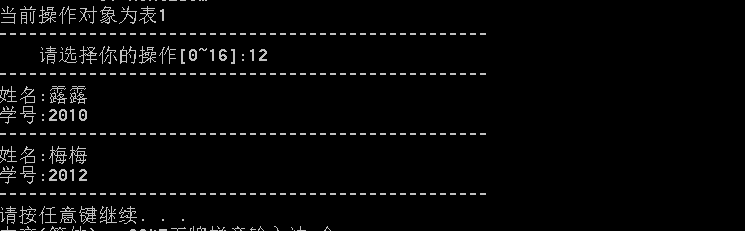
1. 操作结果如图2-10、2-11和2-12。



**图 2-10**



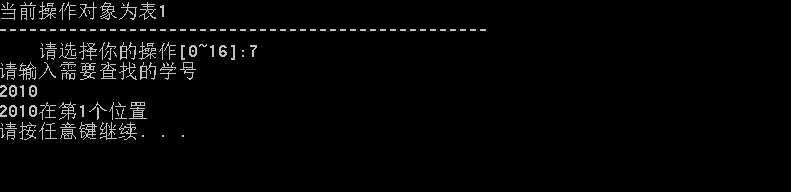
**图 2-11**



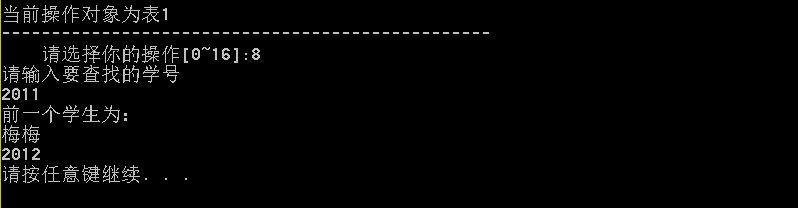
**图 2-12**

评价：位置1 插入成功，位置5插入失败，和预期结果相同。

1. 操作结果如图2-13，2-14.



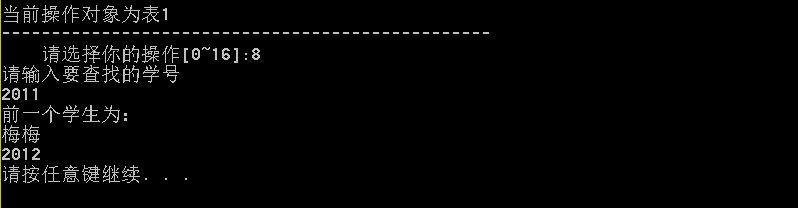
**图 2-13**



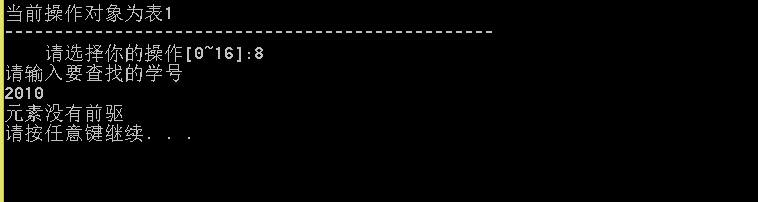
**图 2-15**

评价：两个都成功查找到并返回位置，达到预期目的。

1. 查找前驱，操作结果如图2-16,2-17.



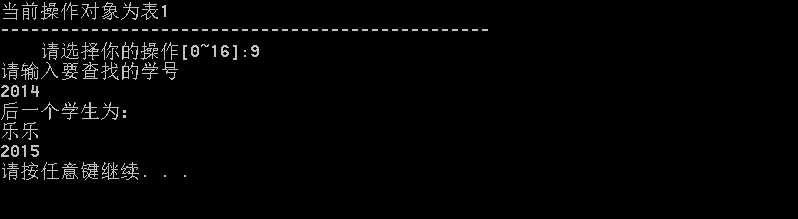
**图 2-16**



**图 2-17**

评价：成功查找到元素，并对没有前驱或者不存在的元素给出了正确提示，完美达到预期的。

1. 查找后继，实验结果如图2-18，2-19.



**图 2-18**



**图 2-19**

评价：2019是最后一个元素，没有后继，显示正确，2014的后继也查找成功，达到实验目的。

## 2.4 实验小结

这次实验又系统的温习了一遍链表的知识，实验过程中也遇到了不少麻烦，比如，我定义多表的结构是用顺序表做的物理结构，然而第一个数据项是用SqList,是一个结构性，导致后面每次调用时都要用‘.’来调用，本来第一个节点用做头结点用来处理链表很方便，但是因为我定义的类型不合适，每次又都要对头结点单独处理，很不方便。文件储存也出了一个问题，我保存文件总会把之前的覆盖掉，比如第二张表会把第一张表的内容覆盖，因而储存多张表时会出现错误，应该每次建立新表时对应建立一个对应的文件进行储存。

链表节点中有一个数据项是结构类型，刚开始做时我只为结点申请了空间，没有对里面的结构申请空间，因此报错了好几次但是一直找不到原因，后来多次测试终于找到了原因。

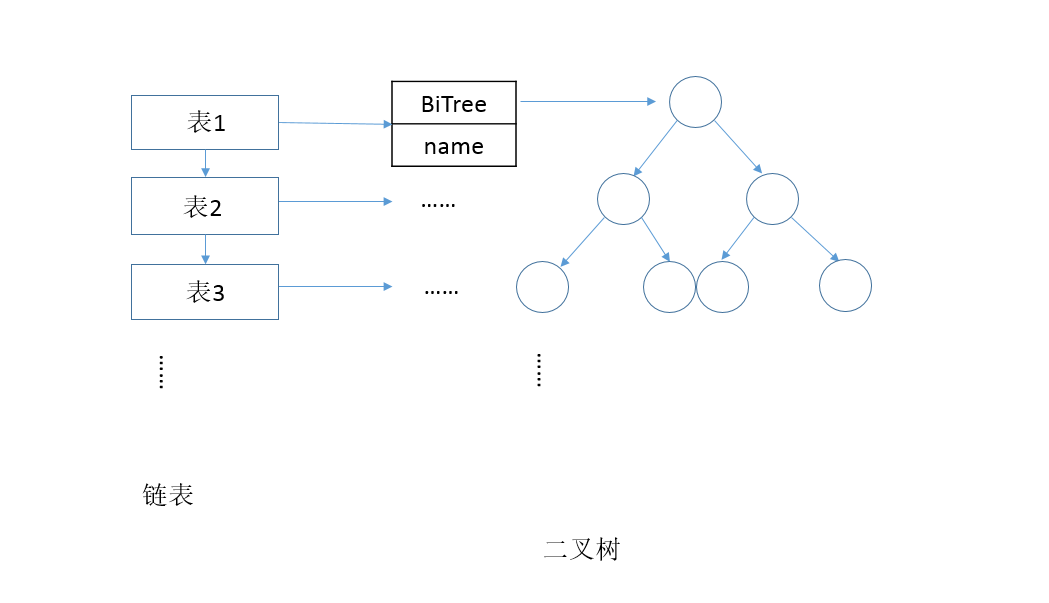
整个数据结构都是以链表和顺序表作为基础的，对链表的熟练操作至关重要，这次实验让我又对链表有了更深层次的认识。

# 3 基于二叉链表的二叉树实现

## 3.1 实验目的

本次试验将会运用二叉链表和顺序表构造一个管理多树的程序，程序提供了24个命令给用户调用，包括树的生成创建，存储，加载，销毁等命令。

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。(4)综合运用链表和二叉链表，以链表存储二叉链表，以此达到管理多棵树的目的。用链表管理树，其物理结构如图3-1。



**图 3-1**

## 链表演示系统设计

**3.2.1 系统总体设计**

为有效的管理一个树状目录，程序至少应该提供初始化，清空，销毁，插入结点，修改节点，查找子树，查找双亲，赋值，以及保存，加载等操作，如果树的深度不是太大，还可以提供图形化显示，以更直观的显示给用户，下面是对程序中提供的24个命令的简单介绍。

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。

(21)保存数据到文件：函数名称是SaveBiTree(T)；初始条件是二叉树T存在且T不为空，则根据用户输入的文件名以前序遍历顺序将数据保存到文件中，同时需要保存该元素节点相对于满二叉树的位置以及左右节点的位置，方便还原。

(22)从文件加载数据：函数名称是LoadBiTree(T)；初始条件是T存在且文件不为空。根据文件中保存的对应节点的对应元素的对应的位置关系，先将所有元素都读到一个数组中，然后根据且节点的对应关系建立各结点的关系，还原二叉树。

（23）添加多棵树：函数名称是AddTree(L)；初始条件是链表L存在，由用户输入需要创建的树的名称，并将创建好的树返回给用户。

（24）切换到另一棵树：函数名称是ChangeTree（L）；初始条件是目前存在多棵树，由用户输入需要切换到的树的名称，然后遍历查找整个链表，如果找到返回该树，如果没有找到，返回ERROR。

（25）删除树：函数名称是DeleteTree（L）；初始条件是链表L不为空，由用户输入需要删除的树的名字，遍历树查找，如果找到则删除，返回OK，否则返回ERROR。

（26）图形化显示树：函数名称是Graph（T）；条件是树T存在且不为空，树的深度小于5，将树中的元素相对于满二叉树的位置输出，形象直观地显示出树的形状和各个节点间的位置关系。

**3.2.2 有关常量的类型和定义**

//相关常量定义

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType;

#define MAXSIZE 100

char definition[MAXSIZE];

typedef struct MyStru

{

char name[MAXSIZE]; //姓名

int number; //学号 唯一标识

}Student;

//二叉树结构类型

typedef struct tree

{

Student \*stu;

struct tree \*lchild; //左孩子

struct tree \*rchild; //右孩子

int exit; //节点中是否有值

int loc; //记录相对于满二叉树的位置，方便图形化显示和保存

}BiTnode, \*BiTree;

//顺序表结构，用于管理多棵树

typedef struct M\_tree

{

BiTree tree;

int flag; //检验是否存在的标志

char name[20]; //存储树的名字

}M\_Tree;

typedef struct SList

{

M\_Tree\* elem;

struct SList\* next;

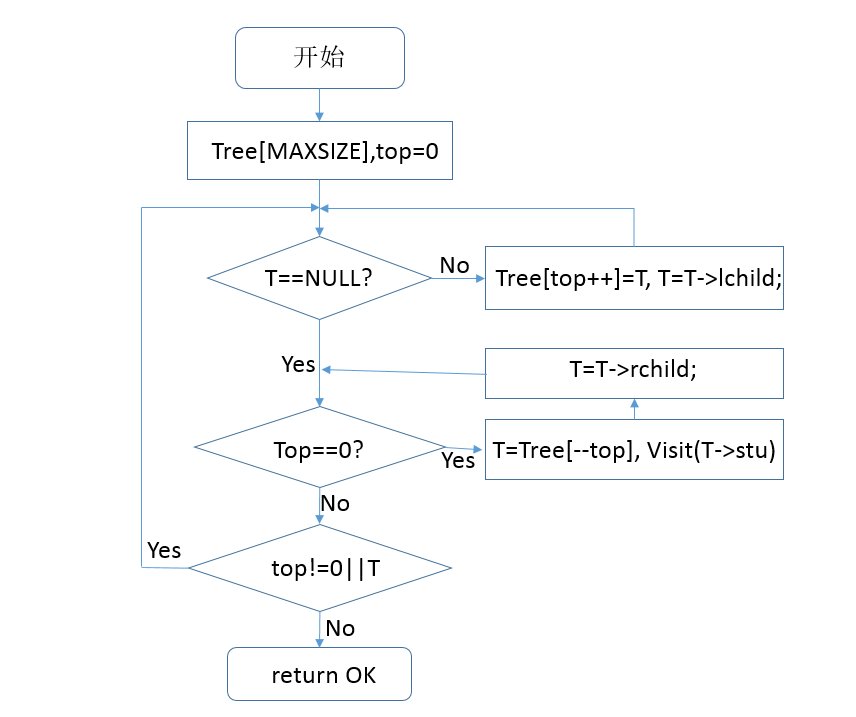
}\*SqList,Sqnode;

#### 3.2.3 算法设计

因为程序中的函数过多，不方便分析算法，因此只给出一部分重点算法详细步骤和程序框图。

1. 中序非递归遍历树，函数名称是InOrderTraverse(T).

算法设计：使用一个节点指针栈遍历二叉树，算法的程序框图 3-2。



**图 3-2**

算法分析：假设二叉树中有n个节点，则每个节点都有一次入栈出栈和读取的操作，时间复杂度为O（3n）,即O(n),空间复杂度为O(n).

(2) 保存树，函数名称SaveBiTree(T)。

首先构造一个指针数组tree[MAXSIZE],top=0;

步骤如下：①如果T为空则返回，否则执行以下步骤。

②用户输入保存到的文件的名称，然后拼接成文件名路径。

③以二进制写入方式打开文件。

④根指针入栈，即tree[top++]=T;

⑤栈顶元素出栈T，首先写入T序号，如果左右孩子不为空的话，将左右孩子的序号写入，否则写入-1.然后写入结构中的数据。T的左右孩子如果存在的话分别入栈。

⑥top--。当top不为0，返回第⑤步，否则执行下一步。

⑦关闭文件，返回OK。

算法分析：保存过程需要遍历整棵树，时间复杂度为O(n),空间复杂度也为O(n)。

1. 从文件中加载树，函数名称LoadBiTree（T）。

算法设计：之前保存树的时候我们已经保存了节点相对于二叉树的位置以及左右节点的位置，因此我们可以用一个结构指针数组BiTree tree[MAXSIZE]读出所有的数据，再用一个整形二维数组Relation[MAXSIZE][2]保存节点的左右孩子的位置关系，然后根据节点的位置关系还原树。步骤如下：

①将数组tree[]全部置为NULL，Relation[MAXSIZE][]中元素全部置为-1.

②根据用户输入的文件路径和名称打开文件，如果文件不存在则要求用户重新输入。如果不能打开则返回错误。

③读出文件中一个结构的全部数据以及位置信息

④判断是否到达文件尾，如果没有，则申请一个结构空间，把3中读到的信息依次赋值，用一个变量count记录下节点位置的最大信息，并且返回第3步，如果以及到达了文件尾，则执行第5步。

⑤当i<count, 遍历数组tree[]，如果某个tree[i]不为空，则根据Relation[i][0]和Relation[MAXSIZE][0]中记录的左右孩子的信息建立起各个节点间的位置关系。

⑥返回OK。

算法分析：假设文件中有n个元素，则读取过程时间复杂度为O（n）,之后遍历数组tree，时间复杂度也为O（n），总的时间复杂度为O（n），空间复杂度也为O（n）.

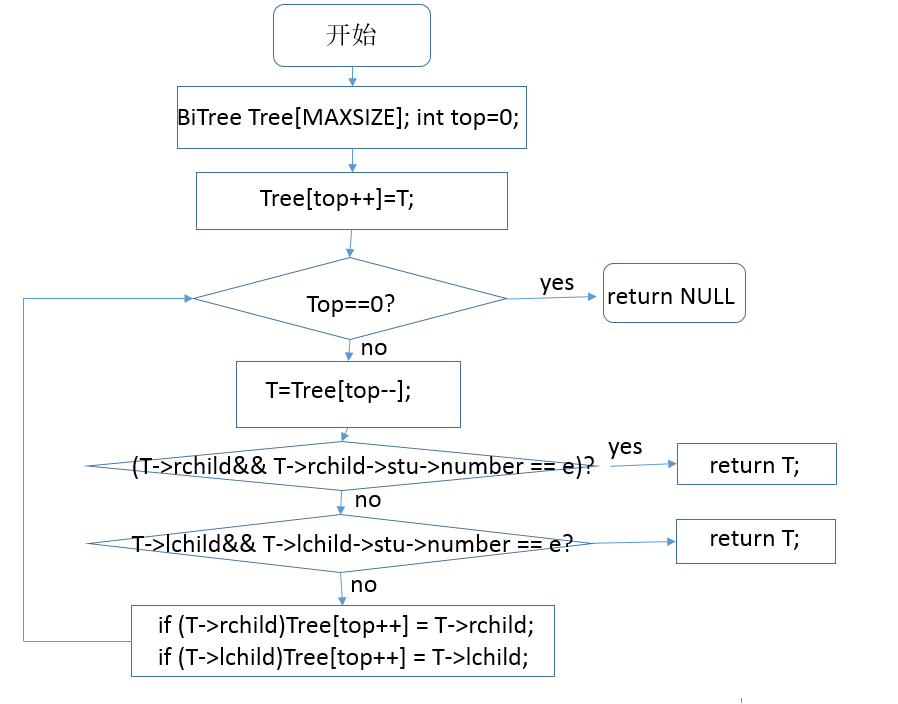
(4) 清空树，函数名称是ClearBiTree(T)。

算法设计：使用递归算法最为简单，只要递归的依次的释放左右孩子的空间，同时把头结点的标志位exit置为0即可。

算法分析：理论上需要遍历整个树，所有时间复杂度和空间复杂度都为O(n).

(5) 查找双亲，函数名称是Parent(T,e)。

算法设计：e是需要查找的元素的唯一ID，用前序遍历的方式查找整棵树，如果找到某个结点的左孩子或者右孩子的ID和e相同，则返回该节点，如果遍历完整棵树都没有找到，则返回NULL。程序框图3-3。



**图3-3**

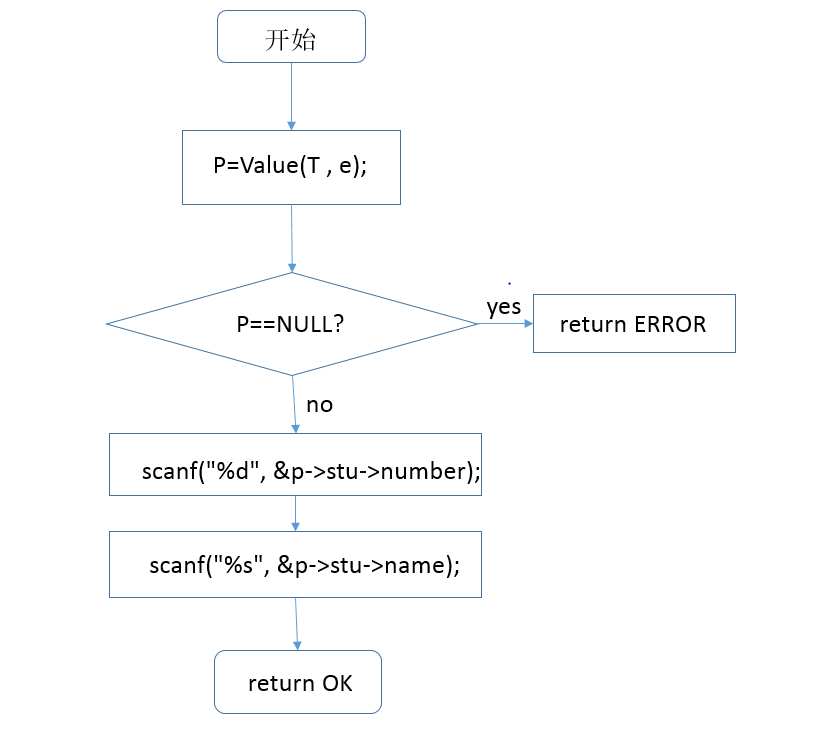
算法分析：最坏情况选需要遍历整个树，平均时间复杂度为O(n/2),所以时间复杂度为 O（n）,空间复杂度为O(n).

(6) 更改节点的值，函数名称Assign（T,e,p）。

算法设计：①调用函数value（T,e）查找看是否书中有节点，如果有则赋值给p，如果没有则返回ERROR；

②分别修改节点的name和number属性，返回OK。

流程框图如图3-4。



**图 3-4**

算法分析：调用函数value需要遍历整个树，最坏情况下时间复杂度为O(n),平均为O(n)，故时间复杂度为O(n),空间复杂度为O(n)。

（7）插入子树，函数名称InsertChild(T, p, LR, c)

算法设计：c是一个已经建立好的子树，其中保证c的右子树为空，根据LR的值确定插入位置P，0表示插入到左子树，1表示插入到右子树，P所指结点的原有左子树或右子树则为c的右子树。

①调用value函数查找到p的位置，如果P为空，返回ERROR。

②如果LR为0，把P的左子树赋值给c的右子树，并把c插入到该节点左子树上。

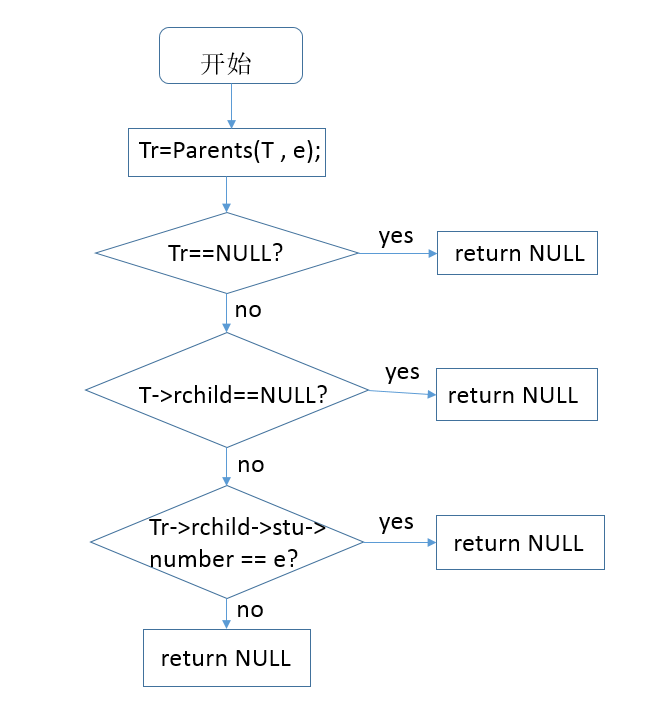
③如果LR为1，把P的右子树赋值给c的右子树，并把c插入到该节点右子树上。

④如果LR为其他值，返回ERROR，否则返回OK。程序结束。

算法分析，调用Value查找p节点时间复杂度为O（n），其他步骤时间复杂度为O(1),故总的时间复杂度为O（n）,空间复杂度为O(1).

(8) 获得右兄弟结点：函数名称是RightSibling(T,e)。

算法设计：e是节点的ID，要查找右兄弟，必然要先找到其双亲，如果本身就是根节点，则没有双亲。找到双亲后，右兄弟可能不存在，不存在则返回NULL，存在则返回右兄弟的结点。程序框图如3-5。



**图 3-5**

算法分析：整个过程关键步骤是遍历二叉树，时间复杂度是O(n),其他步骤时间复杂度是O(1),所以总的时间复杂度为O（1）。

(9) 添加一棵新树，函数名称是AddTree(L)。

算法设计：L是存储树的链表的头结点因为整个树是无序保存在链表中，所以添加树时不必考虑插入位置，每次都插入到尾部，查找时根据树的名称查找即可。

算法步骤：

①判断L是否为空，如果为空，则返回ERROR，否则执行以下步骤。

②从表头遍历找到表尾tail。

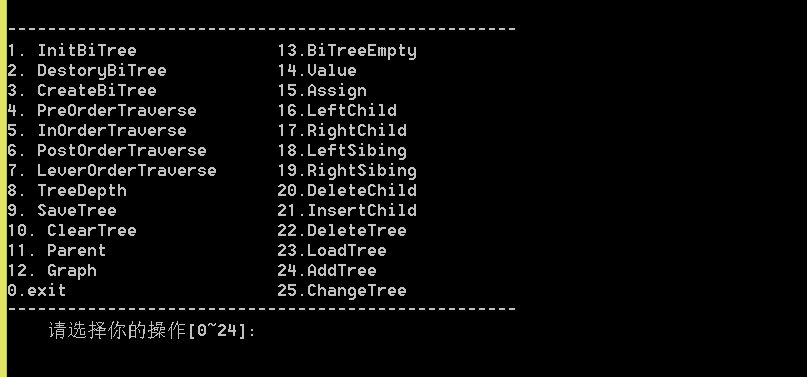
③申请一个树的头结点，用户输入树的名字。

④将该节点添加到表尾，返回OK。

## 3.3 顺序表的系统实现和测试

**3.3.1 系统实现**

本次试验实验的操作系统是windows8，编译软件是VS2013。程序是win32控制台应用程序，以交互式的操作模式供给用户使用，其操作界面如图3-6所示。



**图 3-6**

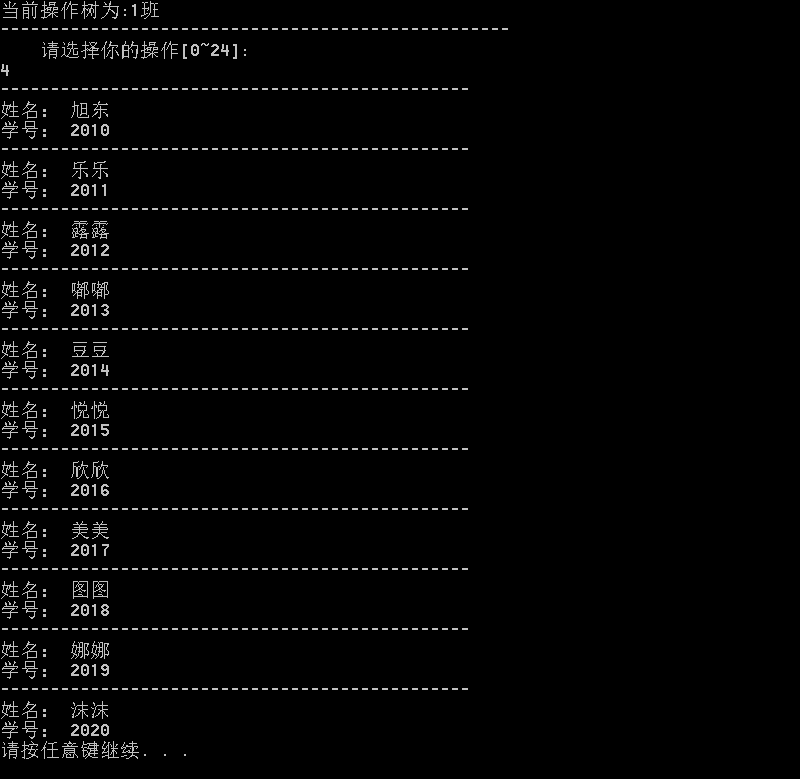
**3.3.2 系统测试**

测试用例以及预期结果如下。

⑴创建链表，按先序序列输入以下10个节点，（旭东，2010），（乐乐，2011），（露露，2012），（嘟嘟，2013），（豆豆，2014），（悦悦，2015），（欣欣，2016），（美美，2017），（图图，2018），（娜娜，2019），（沫沫，2020）。预期结果，树中成功创建好10个节点。

⑵先序遍历二叉树，预期结果，输出顺序为：（旭东，2010），（乐乐，2011），（露露，2012），（嘟嘟，2013），（豆豆，2014），（悦悦，2015），（欣欣，2016），（美美，2017），（图图，2018），（娜娜，2019），（沫沫，2020）。

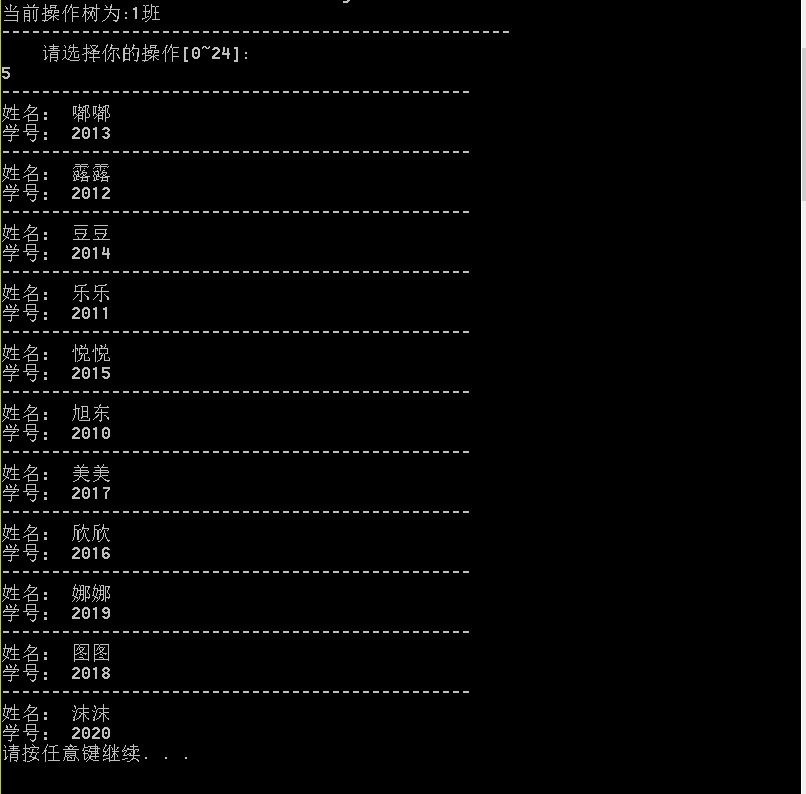
实验结果如图3-7。



**图 3-7**

⑶中序遍历二叉树，预期结果，输出顺序为：（嘟嘟吗，2013），（露露，2012），（豆豆，2014），（乐乐，2011），（悦悦，2015），（旭东，2010），（美美，2017），（欣欣，2016），（娜娜，2018），（图图，2018），（沫沫，2020）。

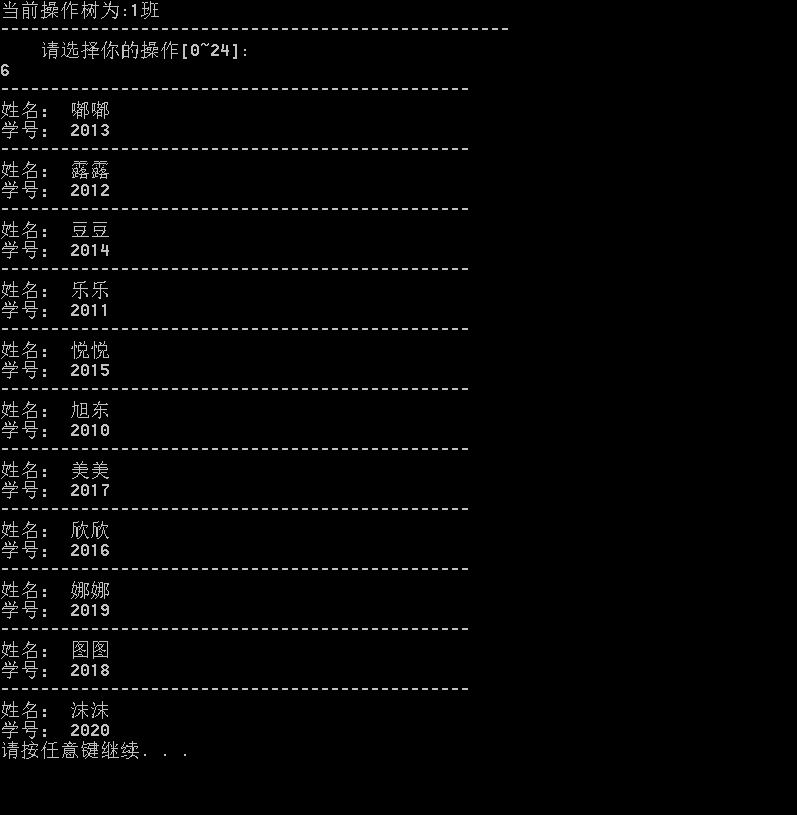
实验结果如图3-8。



**图 3-8**

⑷后续遍历二叉树

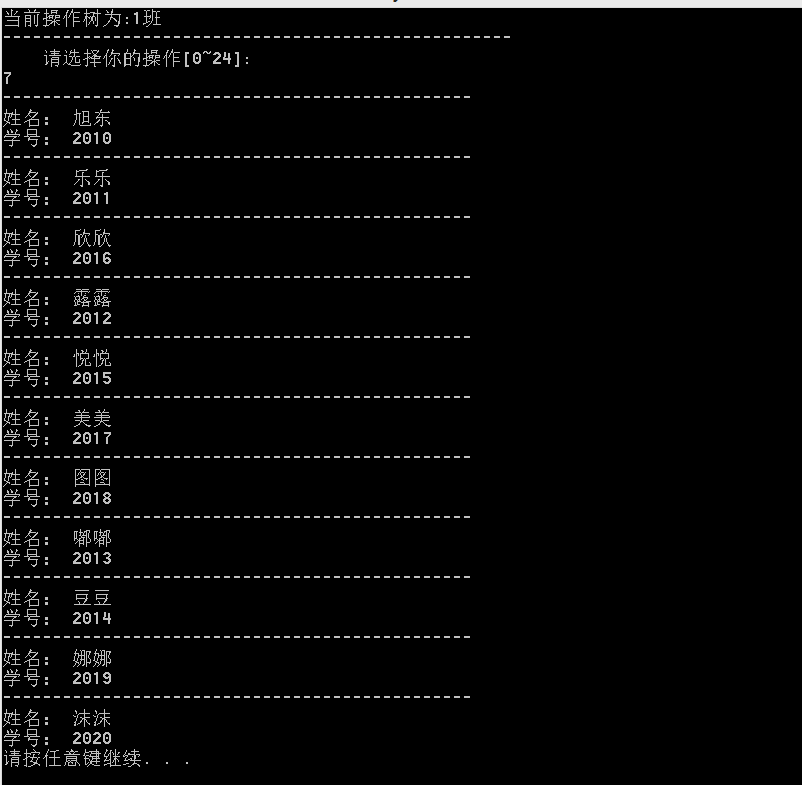
实验结果如图：



**图 3-9**

⑸层序遍历二叉树

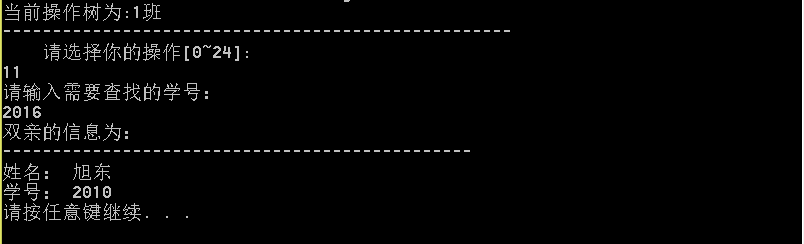
实验结果如图。



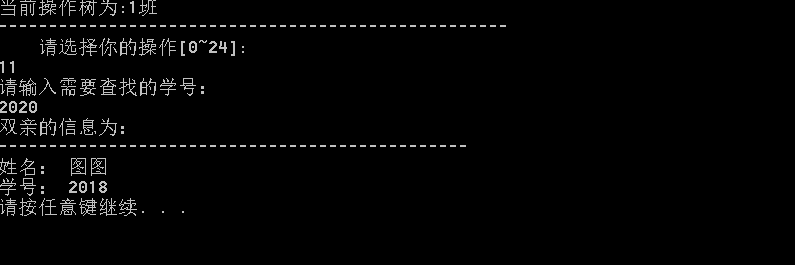
**图 3-10**

⑹查找ID为2016和2020的双亲。预期结果，2016的双亲是2010，2020的双亲是2018，程序成功输出ID为2018和2010和节点的相关信息。

实验结果如图3-11和3-12所示。



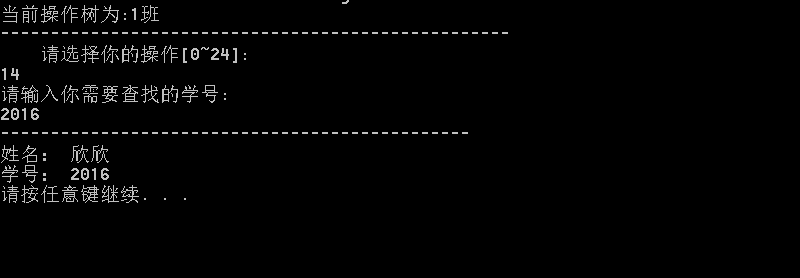
**图 3-11**



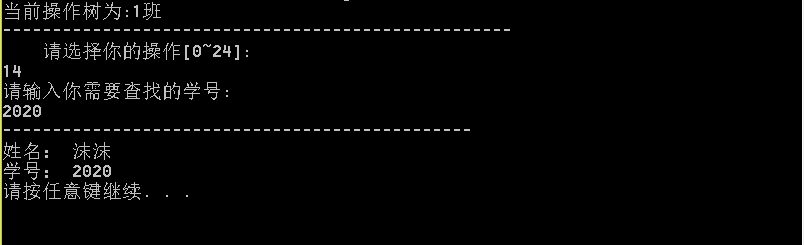
**图 3-12**

⑺查找ID为2016和2020的结点。预期结果，2016和2020都是树中的结点，查找之后能成功输出这两个节点的相关信息。

实验结果如图3-13和3-14。



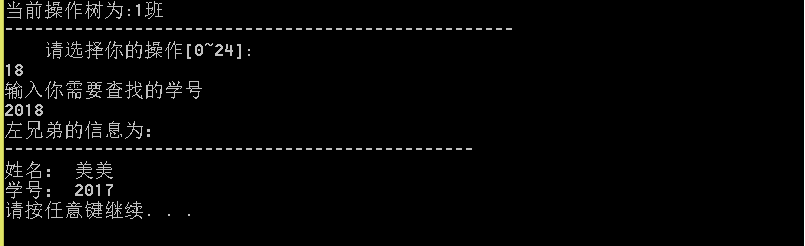
**图 3-13**



**图 3-14**

⑻查找2018的左兄弟。预期结果，2018的左兄弟是ID为2017的节点，程序能正常输出节点ID为2017的所有信息。

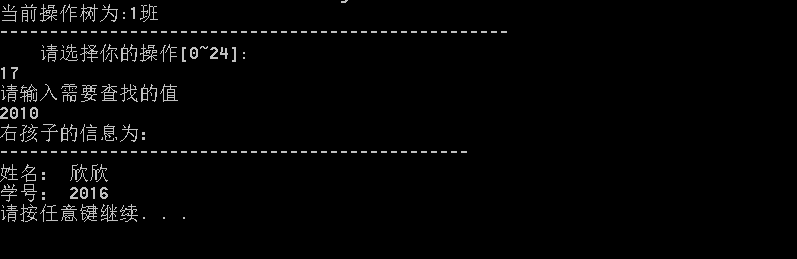
实验结果如图3-15.



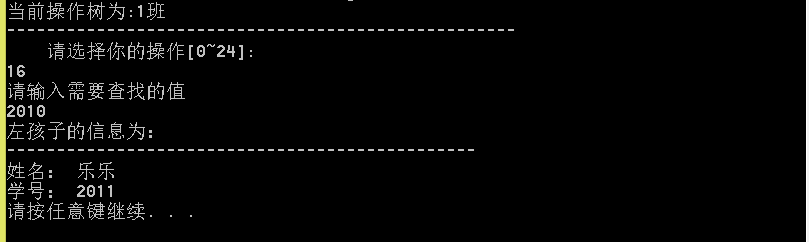
**图 3-15**

⑼查找ID为2010的左孩子和右孩子。预期结果，2010的右孩子为2016，2010的左孩子为2011，能正确的查找到并输出相关信息。

实验结果如图3-16和3-17。



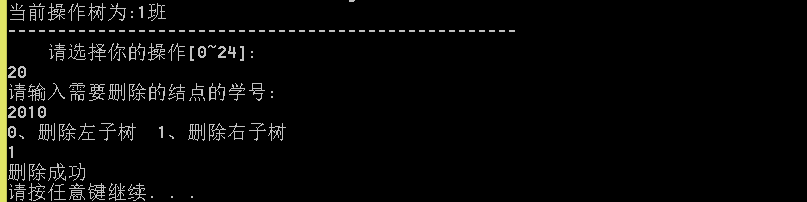
**图 3-16**



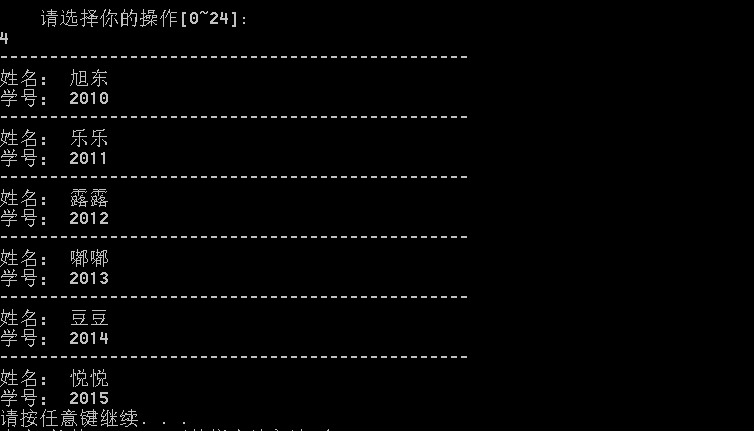
**图 3-17**

⑽删除ID为2010的右子树并前序遍历查看结果。预期结果2010的右子树中含有4个节点，删除之后将只有6个节点输出。

实验结果如图3-18和3-19。



**图 3- 18**



**图 3-19**

测试结果总评：整个过程中程序没有出现BUG，测试的所有用例也全部都正确的输出了实验结果，有些命令没有给出测试用例，但是自己测试的时候都能正确执行，对各种情况都给出了对应的处理方式，是一次很成功的实验。

## 实验小结

本次试验遇到很多问题，花了很多时间才完成了整个实验，同时也暴露了自

己编程能力的不足，经过这次实验使我对树的认识又更深一步，同时，更加巩固了自己对链表的运用，在实验中遇到的问题主要有如下几个。

①创建二叉树时我一开始选择递归的方式创建，definition是一个例如\*\*@@@的字符串，可是递归的过程需要根据definition的字符选择是否创建一个节点还是将该节点置为空，’@’则表示当前节点为空，我用CreateBiTree（T，++definition）进行传参，然而每次都不能正确的创建完整个二叉树就结束了。查阅资料得知递归也是调用栈实现的，definition放在参数表中必然就参与了递归，达不到实验目的，经过冥思苦想，我自己用栈的思想实现了一个栈才解决了这个问题。

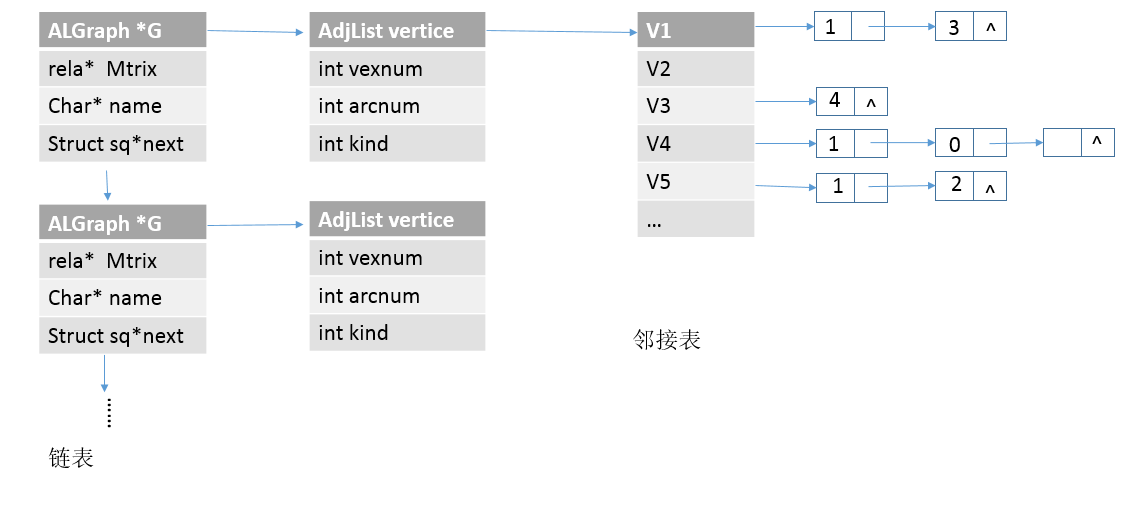
②文件的写入和加载问题。怎样保存树的节点后读取出来还能正确的还原他们的关系呢？一开始想到保存各结点的指针关系，然而这种方案不仅浪费空间而且不能实现，因为读取出来后其指针地址就变了，后来我选择在节点中增加一个位置标记，即相对于满二叉树的位置，这样节点n的左孩子必然为2\*n（假设存在）,右孩子为2\*（n+1），读取出来后我们很方便的就可以根据位置标记还原二叉树了。

# 4基于邻接表的图实现

## 4.1 实验目的

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)实验中通过用链表管理图的方法，每个节点对应一张图，方便同时对多个图进行操作，其物理结构如图4-1。



**图 4-1**

## 图演示系统设计

**4.2.1 系统总体设计**

依据最小完备性和常用性相结合的原则，以函数形式定义了创建图、销毁图、查找顶点、获得顶点值、添加图、切换图，保存图、加载图和顶点赋值等18种基本运算，具体运算功能定义如下。

⑴创建图：函数名称是CreateCraph(&G,V,VR)；初始条件是V是图的顶点集，VR是图的关系集；操作结果是按V和VR的定义构造图G。

⑵销毁图：树函数名称是DestroyBiTree(T)；初始条件图G已存在；操作结果是销毁图G。

⑶查找顶点：函数名称是LocateVex(G,u)；初始条件是图G存在，u和G中的顶点具有相同特征；操作结果是若u在图G中存在，返回顶点u的位置信息，否则返回其它信息。

⑷获得顶点值：函数名称是GetVex (G,v)；初始条件是图G存在，v是G中的某个顶点；操作结果是返回v的值。

⑸顶点赋值：函数名称是PutVex (G,v,value)；初始条件是图G存在，v是G中的某个顶点；操作结果是对v赋值value。

⑹获得第一邻接点：函数名称是FirstAdjVex(&G, v)；初始条件是图G存在，v是G的一个顶点；操作结果是返回v的第一个邻接顶点，如果v没有邻接顶点，返回“空”。

⑺获得下一邻接点：函数名称是NextAdjVex(&G, v, w)；初始条件是图G存在，v是G的一个顶点,w是v的邻接顶点；操作结果是返回v的（相对于w）下一个邻接顶点，如果w是最后一个邻接顶点，返回“空”。

⑻插入顶点：函数名称是InsertVex(&G,v)；初始条件是图G存在，v和G中的顶点具有相同特征；操作结果是在图G中增加新顶点v。

⑼删除顶点：函数名称是DeleteVex(&G,v)；初始条件是图G存在，v是G的一个顶点；操作结果是在图G中删除顶点v和与v相关的弧。

⑽插入弧：函数名称是InsertArc(&G,v,w)；初始条件是图G存在，v、w是G的顶点；操作结果是在图G中增加弧<v,w>，如果图G是无向图，还需要增加<w,v>。

⑾删除弧：函数名称是DeleteArc(&G,v,w)；初始条件是图G存在，v、w是G的顶点；操作结果是在图G中删除弧<v,w>，如果图G是无向图，还需要删除<w,v>。

⑿深度优先搜索遍历：函数名称是DFSTraverse(G,visit())；初始条件是图G存在；操作结果是图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。

⒀广深度优先搜索遍历：函数名称是BFSTraverse(G,visit())；初始条件是图G存在；操作结果是图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。

⒁保存图：函数名称是 SaveGraph(G，VR);初始条件是图G存在且图中存在元素；操作结果是将图G中的元素保存到文件中，并且同时保存对应的邻接矩阵，方便加载。

⒂加载图：函数名称是 LoadGraph(G); 初始条件是图文件存在且不为空；操作结果是新建一张图并且把文件中的元素加载到图中并且建立起个顶点间的关系。

⒃添加图：函数名称是AddGraph(L); 初始条件是链表L存在；操作结果是增加一个链表节点，即增加了一张图。

⒄切换图：函数名称是ChangeGraph(L); 初始条件是链表L存在而且不为空；操作结果是根据用户输入的图的名字返回相应的图，如果不存在返回NULL。

⒅删除图：函数名称是 DeleteGraph(L); 初始条件是链表L存在而且不为空；操作结果是根据用户输入的图的名字查找到相应的图节点并删除，如果不存在返回ERROR。

* + 1. **有关常量的类型和定义**

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status;

typedef int InfoType;

#define MAXSIZE 100

#define MAX\_VERTEX\_NUM 20

typedef int KeyType;

int ID[100],num; //用来存储唯一ID，防止重复

bool visited[MAX\_VERTEX\_NUM]; //访问标志数组

typedef struct

{

char localName[20]; //地点名称

int ID; //唯一标识ID

}province;

typedef struct ArcNode

{

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

InfoType info; //该弧相关信息的指针

}ArcNode;

typedef struct VNode

{

province\* data; //顶点信息

ArcNode \*firstarc; //指向第一条依附于该顶点的弧的指针

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

typedef struct

{

AdjList vertice;

int vexnum, arcnum; //弧的当前顶点数和弧度数

int kind; //图的种类标志

}ALGraph;

//邻接关系矩阵

typedef struct

{

int relation[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM];

}rela;

//顶点信息

typedef struct

{

char Vnode[MAX\_VERTEX\_NUM][20];

}Vnodes;

//多表结构，用于保存多个图

typedef struct Sq

{

ALGraph\* G; //图

rela\* Mtrix; //关系矩阵

char name[20]; //图的名字

struct Sq \*next;

}SqNode,\*SqList;

* + 1. **算法设计**

⑴创建图：函数名称是CreateCraph(\*G,V,VR)。

描述：G是图指针，V是定点集，VR是关系集，用一个结构二维数组保存各顶点的关系，相当于邻接矩阵。

算法设计：因为过程比较长，画程序框图比较繁琐，所有我用自然语言描述整个过程。步骤如下。

①如果G为空，返回ERROR。

②i =0，如果i 小于顶点数目，则为G的每个顶点数据域申请空间，并把顶点集V中的数据拷贝到G的数据域中，i++；如果i>=顶点数目，执行下一步，否则继续执行第2步。

③i=0，如果i小于顶点数目，则由用户输入每个顶点的ID，并检查ID是否重复，如果重复，要求重新输入。i++，如果i>=顶点数目，执行下一步，否则继续执行第3步。

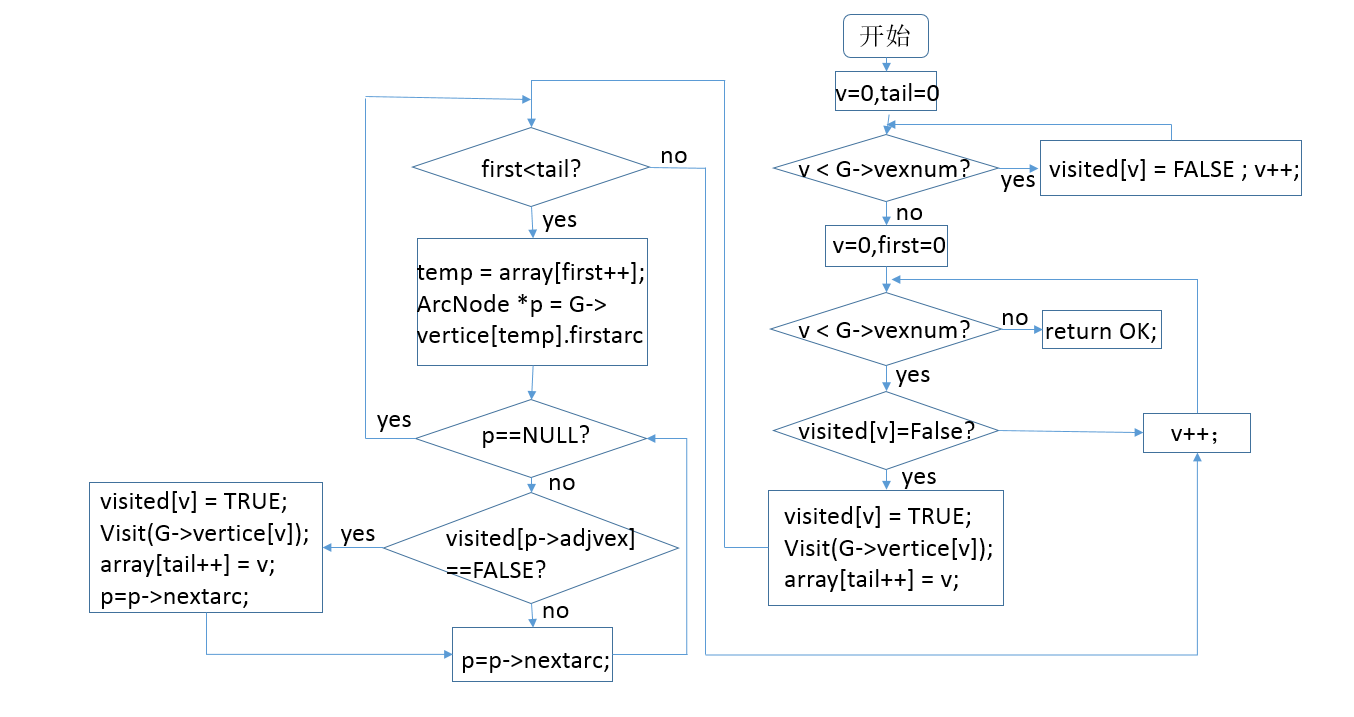
④遍历关系集VR，如果VR中对于顶点的关系为1，则说明该两个顶点之间有弧连接，则把该弧插到顶点后面。

⑤G->kind=1; 既表明图的类型同时也以此可以作为判断图是否存在的依据。返回OK。

算法分析：设顶点数目为n，则步骤2的频度为O(n),步骤3的频度为O(n),步骤4的频度为O(n\*n)。故时间复杂度为O(n\*n),空间复杂度为O(1)。

⑵广深度优先搜索遍历：函数名称是BFSTraverse(G,visit())；

算法设计：首先构造一个足够大的空的数组，然后将每一个顶点序号对应的值赋值为FALSE,标记为为读状态。再构造一个足够大的数组作为队列使用，用队列的思想遍历邻接表，每次从队列中读到一个顶点，输出它的信息，并将它出队，将它的邻接顶点入队。程序框图如图4-2.



**图 4-2**

算法分析：依次将每个顶点入队，然后从队列中读取队首元素并遍历它的邻接表，平均时间复杂度为O(n\*n),空间复杂度为O(n)。

⑶深度优先搜索遍历：函数名称是DFSTraverse(G,visit())。

算法设计：

①首先构造一个足够大的空的数组visited[MAX]，然后将每一个顶点序号对应的值赋值为FALSE,标记为为读状态，v=0。

②v小于图的顶点数吗？如果是，执行下一步，函数结束，否则返回OK。

③调用函数DFS(G, v)；v++，返回第2步。

DFS函数设计如下：

①visited[v]=True。 调用函数Visit访问第v个节点信息，p指向顶点v的第一条弧。

②p为空吗？如果是，返回OK；否则执行下一步。

③p所指向的顶点已经被访问过吗？如果没有，递归调用DFS函数访问该顶点。

④p指向下一条弧，返回②。

算法分析：时间复杂度为O(n\*n),空间复杂度为O(n)。

⑷销毁图：树函数名称是DestroyBiTree(T)；

算法设计：只需要判断图是否为空，如果不为空，释放图的空间，返回OK；如果为空，返回ERROR。

算法分析：因为图结构中，各顶点是存放在一个数组中，所以释放操作只需要一步即可完成，时间复杂度为O(1)，空间复杂度为O(1)。

⑸顶点赋值：函数名称是PutVex (G,v,value)；

算法设计：vn表示节点序号，value中包含有需要赋值的信息，步骤如下。

①G为空或者G->kind!=1吗？如果是，返回ERROR；否则执行下一步.

②将value中包含的信息赋值给顶点v，返回OK。

算法分析：时间复杂度为O(1),空间复杂度为O(1)。

⑹获得第一邻接点：函数名称是FirstAdjVex(&G, v)；

算法设计：因为参数v是节点的ID，所以我们先通过ID找到该顶点在结构数组中的位置，便很容易的找到了该节点的第一邻接顶点。步骤如下。

①调用search(G,v)函数查找顶点v在图中的位置，如果返回的是-1，说明该ID不存在于图中，返回NULL；如果不是，则将返回的结果赋值给m；

②图的第m个节点的第一个邻接点是否为空，如果为空，返回NULL；否则返回第一邻接点。

算法分析：函数search需要遍历整个图查找ID对于的序号，因此该过程时间复杂度为O(n)，其他步骤的时间复杂度均为O(1)。综上，时间复杂度为O(n),空间复杂度为O(1)。

⑺删除顶点：函数名称是DeleteVex(&G,v)；

算法设计：v是需要删除的节点的ID，根据ID首先查找该ID在图中对应的位置，然后将该节点后面的顶点都向前移动一个位置，然后将图的顶点数减1即可。步骤如下。

① 如果图G为空，返回ERROR，如果不为空，执行下一步。

②调用search(G,v)函数查找顶点v在图中的位置，如果返回的是-1，说明该ID不存在于图中，返回NULL；如果不是，则将返回的结果赋值给m；

③遍历所有的顶点，如果该定点的邻接表中有指向顶点v的弧，删除该弧。

④从第m个位置开始，将后面的每个节点都向前移动一个位置。

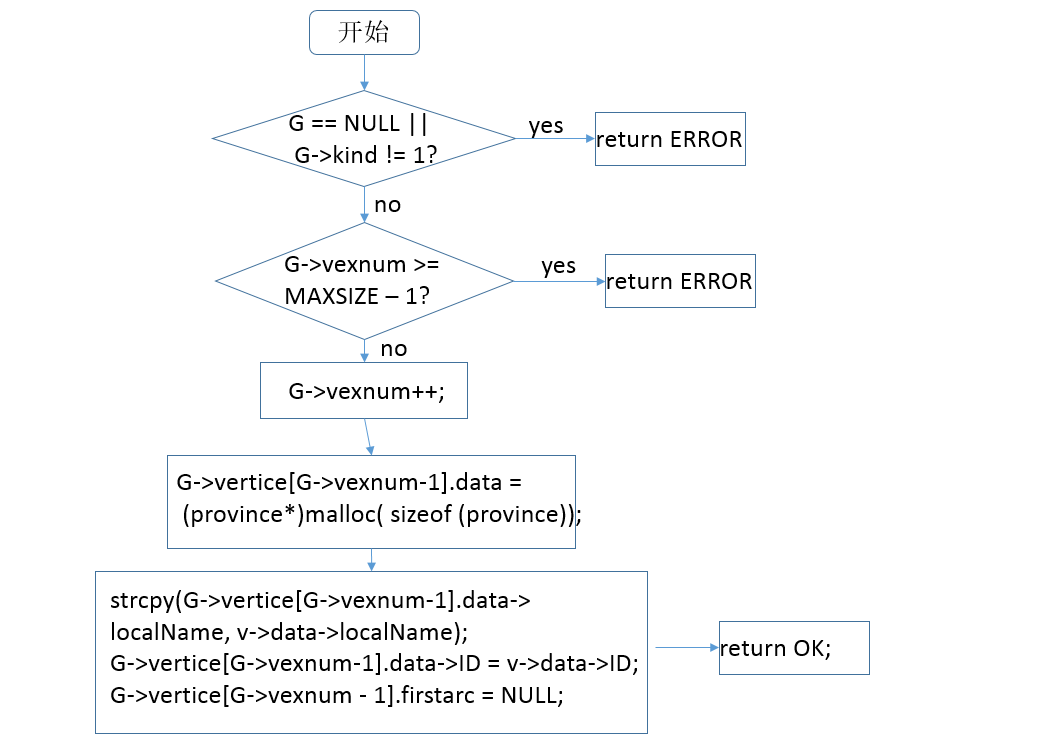
⑤图G的顶点数目减1。返回OK，程序结束。

算法分析:调用search函数查找顶点v的位置需要遍历图，平均时间复杂度为O(n)。

遍历所有顶点和顶点的邻接表，时间复杂度为O(n\*n),移动图顶点时间复杂度为O(n/2),综上，时间复杂度为O(n\*n),空间复杂度为O(1)。

⑻插入顶点：函数名称是InsertVex(&G,v)；

算法设计：因为各结点是保存在结构数组中的，因此在结构数组未满的情况下，只需要将图的顶点数加一，即将新插入的顶点保存到数组最后一个即可，程序框图如图 4-3。



**图 4-3**

算法分析：程序中不存在循环的情况，都是单步执行，因此时间复杂度为O(1),空间复杂度为O(1)。

⑼保存图：函数名称是 SaveGraph(G, VR);

算法设计：保存图时，我们不仅要保存图的元素，还要保存图中各顶点之间的关系，以便于以后恢复图。算法步骤如下。

①如果图不存在，返回ERROR。

②由用户输入需要保存到的文件，以二进制写入的方式打开该文件，如果打开失败，返回ERROR。

③写入图的顶点数目vexnum。

④遍历图的所有顶点，将每个顶点数据域中的数据写入到文件中。

⑤遍历关系矩阵，将矩阵中的信息写入到文件中。

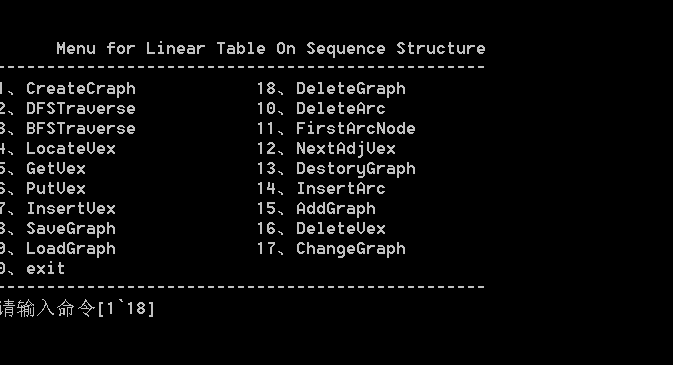
⑥关闭文件指针，返回OK。

算法分析：第4步需要遍历图的所有顶点，时间复杂度为O(n),步骤5需要遍历二维矩阵，时间复杂度为O(n\*n)，综上，时间复杂度为O(n\*n),空间复杂度为O(1)。

## 4.3 图的系统实现和测试

**4.3.1 系统实现**

本次试验实验的操作系统是windows8，编译软件是VS2013。程序是win32控制台应用程序，以交互式的操作模式供给用户使用，其操作界面如图4-4所示。

****

**图 4-4**

**4.3.2 系统测试**

测试用例如下。

⑴创建一个如图4-5的有向图，并分别深度和广度遍历。

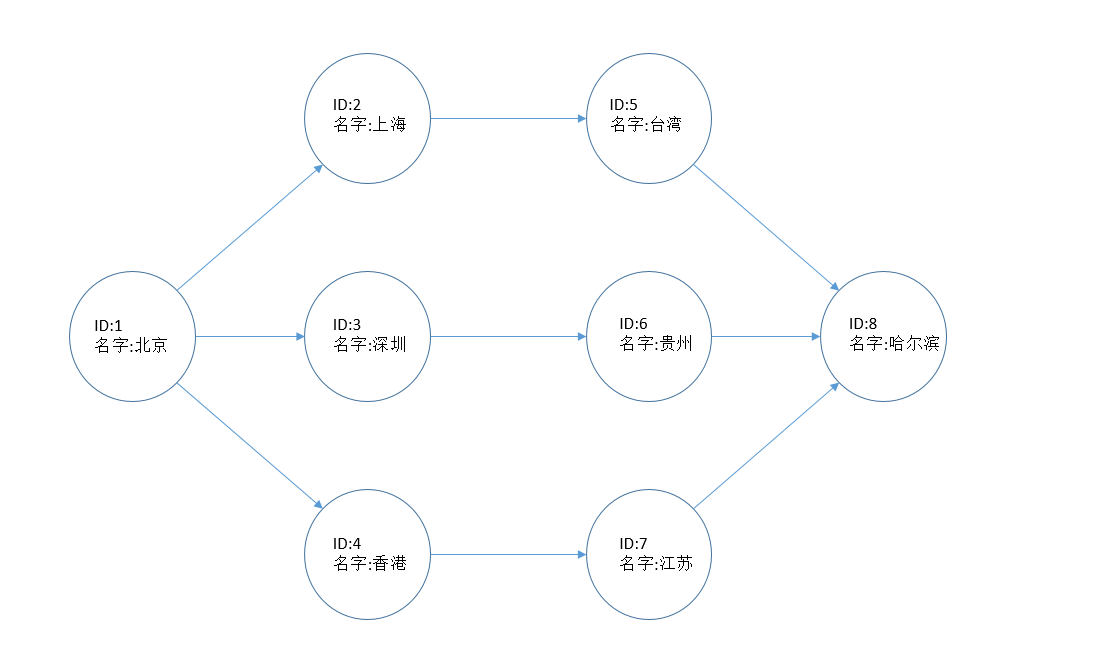
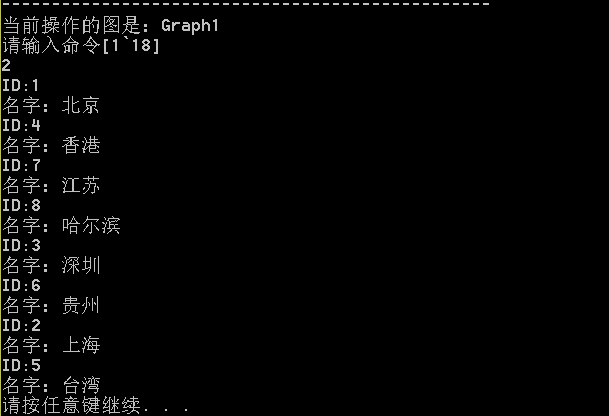


图 4-5

测试结果如图4-6和4-7。



**图 4-6**



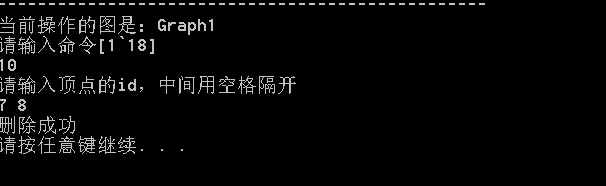
**图 4-7**

结果评价：新建图后，深度和广度遍历都成功执行，符合预期结果，而且没有出现异常。

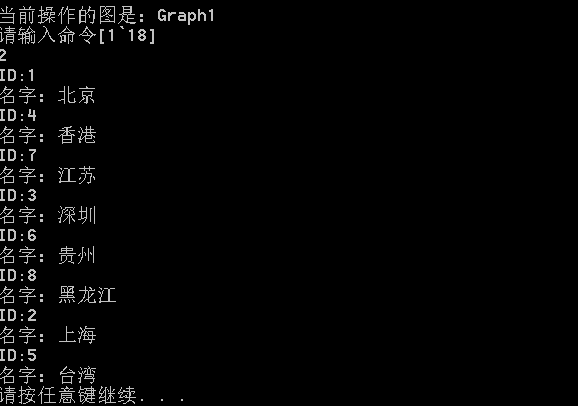
⑵删除7和8之间的弧，然后遍历，如图4-8，4-9。

预期结果，删除ID为7和8之间的弧之后，深度优先遍历ID的输出顺序将变为

1，4，7，3，6，8，2，5。



**图 4-8**

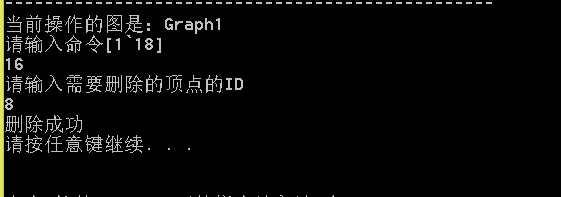


**图 4-9**

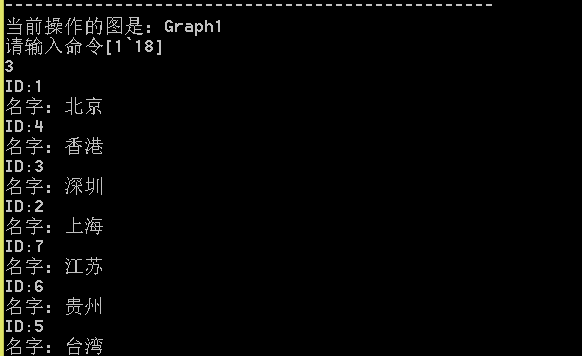
测试评价：测试结果和预期结果一样，满足实验要求。

⑶删除顶点8，然后广度优先遍历。

预期结果，删除ID 8之后，ID输出顺序将变为1，4，3，2，7，6，5.



**图 4-10**



**图 4-11**

测试结果和预期结果一致，达到了实验目的。

⑷将ID为8的顶点的名字由哈尔滨修改为黑龙江如图4-12和4-13。

预期结果，修改之后将输出黑龙江而不是哈尔滨。



**图 4-12**

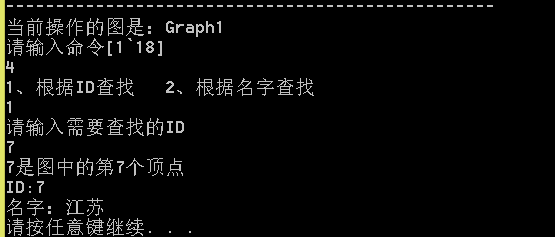


**图 4-13**

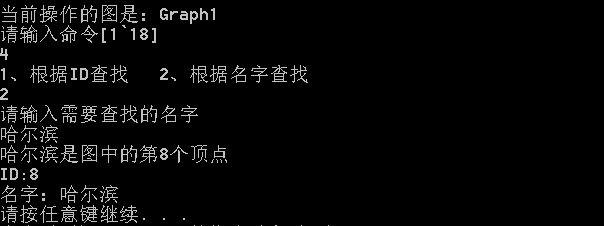
测试评价，测试结果和预期结果一致，达到预期目的。

⑸查找顶点ID为7和名字为哈尔滨的位置，如图4-14和4-15。

预期结果，因为元素顶点是保存在结构数组中，因此他们是按顺序存储，ID 7的位置是7，哈尔滨的位置是8。



**图 4-14**



**图 4-15**

测试评价：成功找到了两个元素的位置，达到了实验预期的目的。

⑹查找ID为1的顶点的第一邻接点，如图4-16。

预期结果，由以上的深度遍历输出结果我们知道1的邻接点为4，将输出ID为4的顶点信息。



**图 4-16**

测试评价，和预期结果一致，达到实验目的。

⑺插入顶点ID为9，名字为台湾，如图4-17。

预期结果，由于插入的顶点和其他顶点之间没有建立关系，所以输出时将在最后输出，结果如图4-17。



**图 4-17**

测试评价，测试结果和预期结果一致，达到预期目的。

⑻保存图。

预期结果，根据上面对保存函数的解析知道，保存图时将在D盘中新建一个TXT文件，里面存储了图的顶点信息和关系矩阵，打开D盘将看到这个文件，但是由于是以二进制写入的，所以不可读。

⑼加载图，重新建立一个图，读入刚才保存的图。

预期结果，，将得到一个和刚才一模一样的图。

## 4.4实验小结

经过前面几次试验，这次写的比较顺利，在向老师征求一些意见之后，老师建议我在新建顶点关系时通过输入ID新建关系，提高操作的便捷性，最好提供给用户傻瓜操作模式。还有就是忘了做查重操作，新建一个顶点时应该检查是否已经存在相同的顶点ID，助教说这算是系统异常，以后的设计中需要注意这一点。

这是最后一次实验了，总结之前的几次，我深刻的感悟就是做软件时无论怎么仔细都不为过，每次我都信心满满的找助教检查，可是还是存在这样那样的问题，因此以后的学习和工作中必须注意这一点。有一点心里不太舒服的是第一次助教就说只允许检查一次，可是我还是看到某些同学错了就拿回去重新写，改好又来检查，我觉得这很不公平，不知道助教为什么允许这样做。

虽然这门课即将结束了，但我知道我对数据结构的学习还很浅薄，以后要经常复习，不能考了就放，经过这个学期的实验，我感觉我的动手能力有了很大的提高。

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

[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 基于顺序存储结构的线性表的源程序

#include<stdlib.h>

#include<malloc.h>

#include<stdio.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;

#define LIST\_INIT\_SIZE 10

#define LISTINCREMENT 10

#define MAXSIZE 10

typedef struct

{

ElemType \*elem;

int length;

int listsize;

int flag; //用于标记表初始化状态

}SqList;

typedef struct //用于保存多个线性表的结构

{

SqList\* Sq; //顺序表指针

int length;

int listsize;

}M\_List;

//如果表存在，删除第i个元素，用e 返回值

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

//如果表存在，遍历表

status ListTrabverse(SqList L);

//如果表不存在，初始化

status IntiaList(SqList &L);

//如果表存在,销毁

status DestroyList(SqList &L);

//如果表存在，清空

status ClearList(SqList &L);

//如果表存在，判断是否为空

status ListEmpty(SqList L);

//如果表存在，返回表长

int ListLength(SqList L,ElemType &e);

//如果表存在，获取第i个元素，用e返回

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

//如果表存在，找到元素e的位置

status LocateElem(SqList L,ElemType e);

//如果表存在，元素cur存在前驱，用e返回

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

//如果表存在，元素cur存在后继，用e返回

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

//如果表存在，在第i个位置插入元素e

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

//如果表存在，删除第i个元素

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

//如果表存在,保存到文件

status ListSave(SqList &L);

//从文件加载表

status List\_Load\_from\_File(SqList&L);

//初始化多表

status Init\_ManyList(M\_List &M);

//添加一张新表

status Add\_List(M\_List &M);

//切换到另一张表

status Change\_List(M\_List &M,SqList &L);

//删除当前表

status Delete\_List(M\_List &M,int &i);

int main()

{

int count = 0; //用来计算当前表的序号

M\_List M;

Init\_ManyList(M);

SqList &L=M.Sq[count];

int op=1;

while (op)

{

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

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

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

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

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

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

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

printf(" 1. IntiaList 10. ListInsert\n");

printf(" 2. DestroyList 11. ListDelete\n");

printf(" 3. ClearList 12. ListTrabverse\n");

printf(" 4. ListEmpty 13. ListSave\n");

printf(" 5. ListLength 14. List\_Load\_from\_File\n");

printf(" 6. GetElem 15. Add\_List\n");

printf(" 7. LocateElem 16. Change\_List\n ");

printf(" 8. PriorElem 17.List\_Delete\n ");

printf(" 9. NextElem 0. Exit\n");

printf(" 当前操作对象是表%d\n", count+1);

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

printf(" 请选择你的操作[0~14]:");

scanf("%d", &op);

int i, j, k, e;

switch (op)

{

case 1:{

if (IntiaList(M.Sq[count]) == -1) printf("已经初始化，请勿重复操作\n");

else if (IntiaList(M.Sq[count]))printf("线性表创建成功\n");

else printf("线性表创建失败\n"); system("pause"); }

break;

case 2:

i = DestroyList(M.Sq[count]);

if (i==OK)

printf("\n已经成功销毁顺序表\n");

if (i == -1)

printf("还没有初始化\n");

system("pause");

break;

case 3:

if (ClearList(M.Sq[count]) == -1)printf("\n请先初始化操作\n");

else

printf("\n表已经清空\n");

system("pause");

break;

case 4:

i = ListEmpty(M.Sq[count]);

if (i)printf("\n表非空\n");

else if (i == -1)

printf("还没有初始化");

else

printf("\n表为空\n");

system("pause");

break;

case 5:

i = ListLength(L,e);

if (i==-1)printf("\n请先初始化表\n");

else

printf("\n表的长度为%d\n",e);

system("pause");

break;

case 6:

printf("请输入需要获取的元素的位置\n");

scanf("%d", &i);

j = GetElem(M.Sq[count], i, e);

if (j == -1)

{

printf("还没有初始化\n");

break;

}

if (j==1)

printf("\n第%d个元素为%d\n",i,e);

else

printf("\n输入错误或者是还没有初始化表\n");

system("pause");

break;

case 7:

printf("请输入元素\n");

scanf("%d", &e);

j = LocateElem(M.Sq[count], e);

if (j == -1)

printf("还没有初始化\n");

if (!j)printf("元素不存在\n");

else

printf("\n元素%d表中第%d位置\n",e,j);

system("pause");

break;

case 8:

printf("请输入元素\n");

scanf("%d", &j);

i = PriorElem(M.Sq[count], j, k);

if (i == -1)

{

printf("还没有初始化"); break;

}

if (i == 0)printf("请先初始化\n");

else if (i == 2)printf("元素不存在或没有前驱\n");

else

printf("\n%d的前驱元素为%d\n",j,k);

system("pause");

break;

case 9:

printf("请输入元素\n");

scanf("%d", &j);

i = NextElem(M.Sq[count], j, k);

if (i == -1){ printf("请先初始化\n"); }

else if (i == 2)printf("元素不存在或没有后继\n");

else

printf("\n%d的后继元素为%d\n", j, k);

system("pause");

break;

case 10:

printf("\n请输入需要插入的元素位置\n");

scanf("%d", &j);

printf("请输入需要插入元素\n");

scanf("%d", &k);

i = ListInsert(M.Sq[count], j, k);

if (i == -1)

{

printf("还没有初始化\n"); break;

}

if (i == 0)printf("输入错误\n");

else

printf("\n插入成功\n", k);

system("pause");

break;

case 11:

printf("请输入需要删除元素位置\n");

scanf("%d", &j);

i = ListDelete(M.Sq[count], j, k);

if (i == -1)

{

printf("还没有初始化\n"); break;

}

if (i == 0)printf("输入错误\n");

else

printf("\n删除成功,删除元素为%d\n",k);

system("pause");

break;

case 12:

i = ListTrabverse(M.Sq[count]);

if (i == -1){ printf("还没有初始化\n");}

if (i==0) printf("线性表是空表！\n");

system("pause");

break;

case 13:

if (ListSave(M.Sq[count]) == -1){ printf("还没有初始化"); break; }

else printf("保存成功!\n");

system("pause");

break;

case 14:

i = List\_Load\_from\_File(M.Sq[count]);

if (i==0){ printf("读入失败!\n"); }

else{ printf("读入成功!\n");}

system("pause");

break;

case 15:

Add\_List(M);

printf("添加表%d成功\n",M.length+1);

count = M.length;

system("pause");

break;

case 16:

if (M.length == 0){ printf("当前还没有表存在\n"); }

else{

if (count < M.length)count++;

else count =0;

printf("已经切换至表%d\n", count+1);

}

system("pause");

break;

case 17:

if (M.length == 0){ printf("当前还没有表存在\n"); }

else{

if (Delete\_List(M, count))printf("已经删除表%d\n", count+1);

else

printf("删除失败\n");

count--;

}

system("pause");

break;

case 0:break;

}

}

printf("欢迎下次使用本系统\n");

}

status IntiaList(SqList &L)

{

if (L.flag==1)return -1; //L.flag==1,说明顺序表已经经过初始化，为避免浪费，不应再为其申请，直接返回

else

{

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

if (!L.elem)return ERROR;

L.flag = 1; //将flag置为1，标记已经经过初始化

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

return OK;

}

}

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

{

if (i<1 || i>L.length+1)return ERROR; //插入位置不正确

if (L.flag == 0)return -1;

if (L.length >= L.listsize) //表满，为其动态增加空间

{

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

if (!L.elem)return ERROR;

L.listsize += LISTINCREMENT;

}

++L.length;

int j = L.length;

for (; j >= i; j--)L.elem[j] = L.elem[j - 1]; //遍历表，找到插入位置

L.elem[i - 1] = e; //完成插入

return OK;

}

status DestroyList(SqList &L)

{

if (L.flag == 0)return -1; //flag为0，说明表不存在

else

{

L.flag = 0; //释放空间，并将标志置为0

free(L.elem);

return OK;

}

}

status ClearList(SqList &L)

{

if (L.flag==0)return -1;

else L.length = 0; //表长置为0，清空表

return OK;

}

int ListLength(SqList L,ElemType &e)

{

if (L.flag==0)return -1; //先判断表是否存在

e = L.length;

return OK;

}

status ListEmpty(SqList L)

{

if (L.flag==0)return -1; //先判断表是否存在

if (L.length == 0)return 0; //表长为0，说明表为空

else return 1;

}

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

{

if (L.flag==0)return -1; //先判断表是否存在

if (i<1 || i>L.length)return ERROR; //获取元素的位置不正确

e = L.elem[i-1];

return OK;

}

status LocateElem(SqList L, ElemType e)

{

int i = 1,j=0;

if (L.flag == 0)return -1; //先判断表是否存在

while (i <= L.length&&L.elem[j++]!=e)i++; //查找元素

if (i <= L.length)return i;

else return 0;

}

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

{

if (L.flag == 0)return -1; //先判断表是否存在

int i = 0;

while (L.elem[i] != cur)i++; //查找元素位置

if (i == 0)return 2; //第一个元素，没有前驱

else

pre\_e = L.elem[i - 1]; //返回元素

return OK;

}

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

{

if (L.flag==0)return -1; //先判断表是否存在

int i=0;

while (L.elem[i] != cur&&i<L.length)i++;

if (i == L.length-1)return 2; //表尾元素没有后继

else if (L.elem[i] != cur)return 2; //元素没有找到

else

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

return OK;

}

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

{

if (L.flag == 0)return -1;

if ((i<1 )||( i>L.length))return ERROR; //元素位置错误

int j = i - 1;

e = L.elem[i - 1];

for (; j <= L.length - 2; j++)L.elem[j] = L.elem[j + 1];

L.length--;

return OK;

}

status Delete\_List(M\_List &M, int &i)

{

if (M.length == 0 || i<0 || i>M.length)return ERROR;

int j = i;

while (j < M.length-1)M.Sq[j] = M.Sq[j + 1];

M.length--;

return OK;

}

status ListTrabverse(SqList L)

{

if (L.flag == 0)return -1; //先判断表是否存在

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

int i = 0;

while (i <= L.length-1)printf("\n%d\t", L.elem[i++]); //遍历顺序表

printf("\n");

return OK;

}

status ListSave(SqList &L)

{

if (L.flag == 0)return -1;

FILE \*fout;

fout = fopen("D://abc.txt", "w");

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

fwrite(&L.elem[i], sizeof(int), 1, fout);//把表中的数据写入到文件中保存

fclose(fout);

return OK;

}

status List\_Load\_from\_File(SqList&L)

{

printf("1. 清空当前表并加载 2. 加载到当前表后部\n");

int op,flag=1;

scanf("%d", &op);

switch (op)

{

case 1:

{ IntiaList(L);

FILE \*fin;

fin = fopen("D://abc.txt", "r");

if (!fin)return ERROR;

int i = 0, temp;

for (i; !feof(fin); i++)

{

fread(&temp, sizeof(int), 1, fin);

if (!feof(fin))

{

ListInsert(L, 1, temp);

}

}

fclose(fin);

}break;

case 2:

{

if (L.flag == 0){ printf("当前表不存在!\n"); flag = 0; }

else{

FILE \*fin;

fin = fopen("D://abc.txt", "r");

if (!fin)return ERROR;

int i = 0, temp;

for (i; !feof(fin); i++)

{

fread(&temp, sizeof(int), 1, fin);

if (!feof(fin))

{

ListInsert(L, L.length, temp);

}

}

fclose(fin);

}

}break;

default:

printf("输入错误\n");

break;

}

if (flag == 0)return 0;

return OK;

}

status Init\_ManyList(M\_List &M)

{

M.Sq = (SqList\*)malloc(MAXSIZE\*sizeof(SqList));

if (!M.Sq) exit(-1);

M.length = 0;

M.listsize = MAXSIZE;

return OK;

}

status Add\_List(M\_List &M)

{

if (M.length >= MAXSIZE)

{

M.Sq = (SqList\*)realloc(M.Sq, (M.listsize + LISTINCREMENT)\*sizeof(SqList));

if (!M.Sq)return ERROR;

}

M.listsize += LISTINCREMENT;

M.length++;

return OK;

}

## 附录B 基于链式存储结构的线性表的源程序

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE -1

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 10

#define LISTINCREMENT 10

typedef int status;

typedef int ElemType; //数据元素类型定义

#define MAXSIZE 20

//学生信息结构体

typedef struct student

{

char name[MAXSIZE];

int number;

}Student;

typedef struct SqList

{

Student\* Stu;

struct SqList\*next;

int flag;

}SqList, \*LinkList;

typedef struct //用于保存多个线性表的结构

{

SqList \*elem; //顺序表指针

int length;

int listsize;

}M\_List;

status IntiaList(SqList &L);

status DestroyList(SqList &L);

status ClearList(SqList &L);

status ListEmpty(SqList L);

int ListLength(SqList &L);

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

status LocateElem(SqList& L, ElemType e, int &loc);

status ListInsert(SqList &L, int i);

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

status ListTrabverse(SqList L);

status Init\_MList(M\_List &M);

status Add\_List(M\_List &M);

status Save\_List(SqList L);

//从文件加载到尾部

status load\_List(SqList &L);

//查找前驱

status PriorElem(SqList L,int cur, LinkList& pre\_e);

//查找后继

status NextElem(SqList L,int cur, LinkList& next\_e);

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

void main(void){

M\_List M;

LinkList next\_e, pre\_e,temp=NULL;

int op = 1;

int i, j, k;

int count = 0;

int cur;

Init\_MList(M);

while (op){

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

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

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

printf(" 1. IntiaList 10. ListInsert\n");

printf(" 2. DestroyList 11. ListDelete\n");

printf(" 3. ClearList 12. ListTrabverse\n");

printf(" 4. ListEmpty 13. SavetoFile\n");

printf(" 5. ListLength 14. LoadfromFile\n");

printf(" 6. GetElem 15. AddList\n");

printf(" 7. LocateElem 16. ChangeList\n");

printf(" 8. PriorElem 0. exit\n");

printf(" 9. NextElem \n");

printf("当前操作对象为表%d\n", count + 1);

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

printf(" 请选择你的操作[0~16]:");

scanf("%d", &op);

switch (op){

case 1:

if (IntiaList(M.elem[count]) == OK) printf("线性表创建成功！\n");

else printf("线性表已经存在，请勿重新创建！\n");

system("pause");

break;

case 2:

i = DestroyList(M.elem[count]);

if (i == OK)

{

printf("链表已经销毁\n");

if (count < M.length - 1)count++;

else count = 0;

}

else

printf("链表不存在\n");

system("pause");

break;

case 3:

if (M.elem[count].flag != 1)

printf("表不存在\n");

else

{

ClearList(M.elem[count]);

printf("表已经清空\n");

}

system("pause");

break;

case 4:

j = ListEmpty(M.elem[count]);

if (j == FALSE)

printf("\n表不存在\n");

else if (j == OK)

printf("表为空\n");

else

printf("表不为空\n");

system("pause");

break;

case 5:

j = ListLength(M.elem[count]);

if (j == -1)

printf("表不存在\n");

else

printf("表长为%d\n",j);

system("pause");

break;

case 6:

if (M.elem[count].flag!=1)

printf("表不存在\n");

else

{

printf("请输入元素位置\n");

scanf("%d", &i);

if (GetElem(M.elem[count], i, temp))

{

printf("第i个学生：\n");

printf("%s\n", temp->Stu->name);

printf("%d\n", temp->Stu->number);

}

else

printf("位置不对\n");

}

system("pause");

break;

case 7:

printf("请输入需要查找的学号\n");

scanf("%d", &i);

j = LocateElem(M.elem[count], i, k);

if (j == FALSE)

printf("链表不存在\n");

else if (j == ERROR)

printf("没有找到\n");

else

printf("%d在第%d个位置\n", i, k);

system("pause");

break;

case 8:

if (M.elem[count].flag != 1)

printf("\n表不存在\n");

else

{

printf("请输入要查找的学号\n");

scanf("%d", &cur);

i = PriorElem(M.elem[count],cur,pre\_e);

if (i == ERROR)

printf("元素没有前驱\n");

else

{

printf("前一个学生为：\n");

printf("%s\n", pre\_e->Stu->name);

printf("%d\n", pre\_e->Stu->number);

}

}

system("pause");

break;

case 9:

if (M.elem[count].flag != 1)

printf("\n表不存在\n");

else

{

printf("请输入要查找的学号\n");

scanf("%d", &cur);

i = NextElem(M.elem[count],cur, next\_e);

if (i == ERROR)

printf("元素没有后继\n");

else

{

printf("后一个学生为：\n");

printf("%s\n", next\_e->Stu->name);

printf("%d\n", next\_e->Stu->number);

}

}

system("pause");

break;

case 10:

printf("\n请输入元素位置\n");

scanf("%d", &i);

j = ListInsert(M.elem[count], i);

if (j == OK)

printf("插入成功\n");

else if (j == FALSE)

printf("表不存在\n");

else

printf("插入失败");

system("pause");

break;

case 11:

if (M.elem[count].flag != 1)

printf("\n表不存在\n");

else

{

printf("请输入需要删除元素位置\n");

scanf("%d", &i);

j = ListDelete(M.elem[count], i, temp);

if (!j)

printf("位置输入不对\n");

else

{

printf("删除的学生为:\n");

printf("%s\n", temp->Stu->name);

printf("%d\n", temp->Stu->number);

}

}

system("pause");

break;

case 12:

if (!ListTrabverse(M.elem[count])) printf("链表不存在！\n");

system("pause");

break;

case 13:

if (M.elem[count].flag != 1)printf("链表不存在\n");

else

{

Save\_List(M.elem[count]);

printf("保存成功\n");

}

system("pause");

break;

case 14:

if (M.elem[count].flag != 1)

printf("链表不存在!\n");

else

{

load\_List(M.elem[count]);

printf("加载成功\n");

}

system("pause");

break;

case 15:

Add\_List(M);

count++;

printf("已经添加新表%d\n", M.length);

system("pause");

break;

case 16:

if (count < M.length-1)count++;

else count = 0;

printf("已经切换至表%d\n", count + 1);

system("pause");

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}

status IntiaList(SqList& L)

{

if (L.flag == 1)return FALSE;

L.Stu = (Student\*)malloc(MAXSIZE\*sizeof(Student));

L.Stu->number = 0;

L.next = NULL;

L.flag = 1;

return OK;

}

status ListTrabverse(SqList L){

if (L.flag != 1)return ERROR;

LinkList p = L.next;

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

while (p){

printf("姓名:%s\n", p->Stu->name);

printf("学号:%d\n", p->Stu->number);

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

p = p->next;

}

return OK;

}

status ListEmpty(SqList L)

{

if (L.flag != 1)return FALSE; //表不存在

if (L.next == NULL)return OK; //表为空，返回OK

return ERROR;

}

status Init\_MList(M\_List &M)

{

//为顺序表申请空间

M.elem = (LinkList)malloc(LIST\_INIT\_SIZE\*sizeof(SqList));

if (!M.elem)return ERROR;

M.length = 1; //初始化为1，表示表中默认开始有一张表

M.listsize = LIST\_INIT\_SIZE;

return OK;

}

status Add\_List(M\_List &M)

{

//储存空间已满

if (M.length >= MAXSIZE)

{

M.elem = (SqList\*)realloc(M.elem, (M.listsize + LISTINCREMENT)\*sizeof(SqList));

if (!M.elem)return ERROR;

}

M.listsize += LISTINCREMENT;

M.length++;

return OK;

}

status ListInsert(SqList &L, int i)

{

if (L.flag != 1)return FALSE;

if (i<1||i > ListLength(L)+1)return ERROR; //位置不对

int j = 1;

LinkList p = L.next, q = NULL;

while (j < i&&p)

{

j++;

p = p->next;

}

if (j < i)return ERROR;

q = (LinkList)malloc(sizeof(SqList));

q->Stu = (Student\*)malloc(sizeof(Student));

printf("请输入姓名:\n");

scanf("%s", &q->Stu->name);

printf("请输入学号:\n");

scanf("%d", &q->Stu->number);

if (p == NULL)

{

L.next = q;

q->next = NULL;

}

else

{

q->next = p->next;

p->next = q;

}

return OK;

}

status LocateElem(SqList& L, ElemType e, int &loc)

{

if (L.flag != 1)return FALSE; //表不存在

LinkList p = L.next;

int j = 1;

while (p&&(p->Stu->number != e)) //根据学号查找

{

j++;

p = p->next;

}

if (!p)return ERROR; //没有找到

loc = j; //返回找到位置

return OK;

}

int ListLength(SqList &L)

{

if (L.flag != 1)return FALSE;

int j = 0;

LinkList p = L.next;

while (p!=NULL)

{

j++;

p = p->next;

}

return j;

}

status PriorElem(SqList L,int cur,LinkList& pre\_e)

{

if (L.flag != 1)return FALSE;

LinkList p = L.next;

if (p->Stu->number == cur)return ERROR; //第一个元素没有前驱

while ((p->next)&&(p->next->Stu->number != cur))p = p->next;

if (p!=NULL)pre\_e = p;

else return ERROR;

return OK;

}

status NextElem(SqList L,int cur,LinkList& next\_e)

{

if (L.flag != 1)return FALSE;

LinkList p = L.next;

while (p&&p->Stu->number != cur)p = p->next;

if (!p || p->next == NULL)return ERROR; //元素不存在后者是最后一个元素没有后继

next\_e = p->next;

return OK;

}

status DestroyList(SqList &L)

{

if (L.flag != 1)return FALSE;

LinkList p = L.next,q;

q = p;

while (q!=NULL&&(q->next!=NULL))

{

p = q->next;

free(q);

q = p;

}

L.next = NULL;

L.flag = 0; //标志置为0,表示表不存在了

return OK;

}

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

{

if (i <= 0 || i > ListLength(L))

return ERROR;

LinkList p = L.next,q;

int count = 1;

if (i == 1)

{

e = L.next;

L.next = p->next;

return OK;

}

while (p->next&&count < i-1)

{

p = p->next;

++count;

}

if (!(p->next))return ERROR;

q = p->next;

p->next = q->next;

e = q;

return OK;

}

status Save\_List(SqList L)

{

LinkList p=L.next;

FILE \*fp,\*fp1;

//姓名和学号分别保存在两个文件中，如果打开失败，退出程序

fp = fopen("D://number.txt", "w");

if (!fp)exit(-1);

fp1 = fopen("D://name.txt", "w");

if (!fp1)exit(-1);

int i;

while (p)

{

char \*temp = p->Stu->name ;

i = 0;

//名字中含有多个字节，逐一遍历然后保存

while (temp[i] != '\0')

{

fwrite(&temp[i], sizeof(char), 1, fp1);

i++;

}

fwrite(&temp[i], sizeof(char), 1, fp1);

fwrite(&p->Stu->number, sizeof(int), 1, fp);

p = p->next;

}

fclose(fp);

fclose(fp1);

return OK;

}

status load\_List(SqList &L)

{

LinkList p=L.next,q=NULL;

if (p) //找到表尾进行加载

while (p->next)p = p->next;

FILE \*fp1, \*fp2;

fp1 = fopen("D://name.txt", "r");

if (!fp1)exit(-1);

fp2 = fopen("D://number.txt", "r");

if (!fp2)exit(-1);

int i = 0,temp,t;

char ch;

for (i; (!feof(fp2))&&(!feof(fp1)); i++)

{

t = 0;

fread(&temp, sizeof(int), 1, fp2);

fread(&ch, sizeof(char), 1, fp1);

if (!feof(fp2)) //再次判断是否到达文件尾，非常重要

{

//如果还有元素，继续增加结点

q = (LinkList)malloc(sizeof(SqList));

q->Stu = (Student\*)malloc(sizeof(Student));

q->Stu->number = temp;

}

else

break;

while (ch != '\0') //当读到一个字符串尾部，表示一个名字结束

{

if (!feof(fp1))

q->Stu->name[t++] = ch;

fread(&ch, sizeof(char), 1, fp1);

}

q->Stu->name[t] = '\0'; //对保存名字的字符串末尾添加结尾符，否则遍历会出错

if (L.next == NULL)

{

L.next = q;

p = L.next;

p->next = NULL;

q = NULL;

}

else

{

p->next = q;

p = p->next;

p->next = NULL;

q = NULL;

}

}

fclose(fp2);

fclose(fp1);

return OK;

}

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

{

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

LinkList p = L.next;

int j=1;

while (p&&j < i)

{

p = p->next;

++j;

}

e = p;

return OK;

}

status ClearList(SqList &L)

{

LinkList p = L.next,q;

//依次释放结点空间，只保留头结点

while (p)

{

q = p;

p = p->next;

free(q);

}

L.next = NULL;

return OK;

}

## 附录C 基于二叉链表二叉树的源程序

#include<malloc.h>

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#include<io.h>

//相关常量定义

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType;

#define LIST\_INIT\_SIZE 10

#define LISTINCREMENT 10

#define MAXSIZE 100

char definition[MAXSIZE];

int sequence,ID[MAXSIZE]; //用来存储唯一标识的ID，防止重复

typedef struct MyStru

{

char name[MAXSIZE]; //姓名

int number; //学号 唯一标识

}Student;

//二叉树结构类型

typedef struct tree

{

Student \*stu;

struct tree \*lchild; //左孩子

struct tree \*rchild; //右孩子

int exit; //节点中是否有值

int loc; //用于记录相对于满二叉树的位置，方便图形化显示和保存

}BiTnode, \*BiTree;

//顺序表结构，用于管理多棵树

typedef struct M\_tree

{

BiTree tree;

int flag;

char name[20]; //存储树的名字

}M\_Tree;

typedef struct SList

{

M\_Tree\* elem;

struct SList\* next;

}\*SqList,Sqnode;

status Inspect(BiTree T, int th)

{

if (!T)return ERROR; //如果树为空，则返回

BiTree Tree[MAXSIZE], temp; //构造一个队列用于保存节点

int i = 0, j = 0;

Tree[i++] = T;

while (i > j)

{

if (th == Tree[j]->stu->number)

return ERROR;//对头出队，输出节点值

if (Tree[j]->lchild) //如果对头元素有左孩子，加入队尾

{

Tree[i] = Tree[j]->lchild; i++;

}

if (Tree[j]->rchild) //如果对头元素有右孩子,加入队尾

{

Tree[i] = Tree[j]->rchild; i++;

}

j++; //出队操作，对头指针向后移动

}

return OK;

}

//初始化顺序表

status InitSqList(SqList& L)

{

L = (SqList)malloc(sizeof(Sqnode));

L->elem = (M\_Tree\*)malloc(sizeof(M\_Tree));

L->elem->flag = 0;

if (!L)return ERROR;

L->next = NULL;

return OK;

}

//检查是否有重复的ID

status Visit(Student\* stu);

//初始化二叉树

status InitBiTree(M\_Tree\* M)

{

if (M->tree!=NULL&&M->flag==1)

{

printf("树已经初始化，请勿重复操作\n");

return ERROR; //树已经存在，避免多次初始化

}

M->tree = (BiTree)malloc(sizeof(BiTnode));

M->flag = 1;

printf("请输入树的名称\n");

scanf("%s", M->name);

return OK;

}

//销毁二叉树,递归的方法

status DestroyBiTree(BiTree T)

{

if (T->lchild)

DestroyBiTree(T->lchild);

if (T->rchild)

DestroyBiTree(T->rchild);

free(T);

T->exit = 0;

return OK;

}

//为每个结点记录下相对于满二叉树的位置

status Location(BiTree& T)

{

if (!T)return ERROR; //如果树为空，则返回

BiTree Tree[MAXSIZE], temp; //构造一个队列用于保存节点

int i = 0, j = 0;

Tree[i++] = T;

Tree[j]->loc = 1; //头结点是第一个位置

while (i > j)

{

if (Tree[j]->lchild) //如果对头元素有左孩子，加入队尾

{

Tree[i] = Tree[j]->lchild;

Tree[i]->loc = 2 \* Tree[j]->loc;

i++;

}

if (Tree[j]->rchild) //如果对头元素有右孩子,加入队尾

{

Tree[i] = Tree[j]->rchild;

Tree[i]->loc = 2 \* Tree[j]->loc + 1;

i++;

}

j++; //出队操作，对头指针向后移动

}

return OK;

}

BiTree CReatenode(BiTree T)

{

BiTree q;

q = (BiTree)malloc(sizeof(BiTnode));

q->stu = (Student\*)malloc(sizeof(Student));

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

printf("请输入姓名:\n");

scanf("%s", q->stu->name);

it:

printf("请输入学号：\n");

scanf("%d", &q->stu->number);

if (!Inspect(T, q->stu->number))

{

printf("ID重复，重新输入\n");

goto it;

}

q->exit = 1;

return q;

}

BiTree Createnode()

{

BiTree q;

q = (BiTree)malloc(sizeof(BiTnode));

q->stu = (Student\*)malloc(sizeof(Student));

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

printf("请输入姓名:\n");

scanf("%s", q->stu->name);

printf("请输入学号：\n");

scanf("%d", &q->stu->number);

q->exit = 1;

return q;

}

//创建二叉树

status CReateBiTree(BiTree root,BiTree &T, char\* definition)

{

BiTree Tree[MAXSIZE]; //栈，用来存放结点

int array[MAXSIZE];

int arr[MAXSIZE][1]; //保存节点和他的左孩子位置

int srr[MAXSIZE][1]; //保存孩子和他的右孩子位置

memset(array, 1, MAXSIZE - 1);

int top = 0, i = 0, peak = 0;

while (definition[i] != '\0') //按照输入的先序遍历

{

if (definition[i] == '\*')

{

Tree[top] = CReatenode(root);

array[top] = 0; //将当前节点标记为0，表示已经访问过了，之后不再访问，相当于出栈

Tree[++peak] = Tree[top]->rchild; //右孩子放到栈顶

srr[top][0] = peak; //记录下当前节点和右孩子位置

Tree[++peak] = Tree[top]->lchild; //左孩子入栈

arr[top][0] = peak; //记录下当前节点和左孩子位置

top = peak;

}

else

{

Tree[top] = NULL;

array[top] = 0;

}

while (array[top] == 0) //标记为0,说明已经访问过了，寻找下一个未访问过元素

top--;

i++;

}

for (int i = 0; i <= peak; i++)

{

if (Tree[i] != NULL)

{

//按照之前存储的结点和它的左右孩子位置，依次建立它们的关系

Tree[i]->lchild = Tree[arr[i][0]];

Tree[i]->rchild = Tree[srr[i][0]];

}

}

T = Tree[0];

Location(T);

return OK;

}

//创建二叉树,非递归算法

status CreateBiTree(BiTree &T, char\* definition)

{

BiTree Tree[MAXSIZE]; //栈，用来存放结点

int array[MAXSIZE];

int arr[MAXSIZE][1]; //保存节点和他的左孩子位置

int srr[MAXSIZE][1]; //保存孩子和他的右孩子位置

memset(array, 1, MAXSIZE - 1);

int top = 0,i=0,peak=0;

while (definition[i] != '\0') //按照输入的先序遍历

{

if (definition[i] == '\*')

{

Tree[top] = Createnode();

array[top] = 0; //将当前节点标记为0，表示已经访问过了，之后不再访问，相当于出栈

Tree[++peak] = Tree[top]->rchild; //右孩子放到栈顶

srr[top][0] = peak; //记录下当前节点和右孩子位置

Tree[++peak] = Tree[top]->lchild; //左孩子入栈

arr[top][0] = peak; //记录下当前节点和左孩子位置

top = peak;

}

else

{

Tree[top] = NULL;

array[top] = 0;

}

while (array[top]==0) //标记为0,说明已经访问过了，寻找下一个未访问过元素

top--;

i++;

}

for (int i = 0; i <= peak; i++)

{

if (Tree[i] != NULL)

{

//按照之前存储的结点和它的左右孩子位置，依次建立它们的关系

Tree[i]->lchild = Tree[arr[i][0]];

Tree[i]->rchild = Tree[srr[i][0]];

}

}

T = Tree[0];

Location(T);

return OK;

}

//前序遍历

status PreOrderTraverse(BiTree T, status(\*Visit)(Student\* stu))

{

if (T){

Visit(T->stu);

PreOrderTraverse(T->lchild, Visit);

PreOrderTraverse(T->rchild, Visit);

}

else

return OK;

}

//中序遍历

status InOrderTraverse(BiTree T, status(\*Visit)(Student\* stu))

{

BiTree tree[MAXSIZE];

int top = 0;

do{

while (T)

{

if (top == MAXSIZE) //栈已满，退出

exit(-1);

tree[top++] = T; //根进栈

T = T->lchild; //移向左子树

}

if (top)

{

T = tree[--top]; //弹出根指针

Visit(T->stu);

T = T->rchild; //准备遍历右子树

}

} while (top || T);

}

//后序遍历

status PostOrderTraverse(BiTree T, status(\*Visit)(Student\* stu))

{

if (T)

{

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

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

Visit(T->stu);

}

else

return OK;

}

//层序遍历

status LevelOrderTraverse(BiTree T, status(\*Visit)(Student\* stu))

{

if (!T)return ERROR; //如果树为空，则返回

BiTree Tree[MAXSIZE],temp; //构造一个队列用于保存节点

int i = 0, j = 0;

Tree[i++] = T;

while (i > j)

{

Visit(Tree[j]->stu); //对头出队，输出节点值

if (Tree[j]->lchild) //如果对头元素有左孩子，加入队尾

{

Tree[i] = Tree[j]->lchild; i++;

}

if (Tree[j]->rchild) //如果对头元素有右孩子,加入队尾

{

Tree[i] = Tree[j]->rchild; i++;

}

j++; //出队操作，对头指针向后移动

}

return OK;

}

//输出节点值

status Visit(Student\* stu)

{

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

printf("姓名： %s\n",stu->name);

printf("学号： %d\n", stu->number);

return OK;

}

//求树的深度

int BiTreeDepth(BiTree T)

{

if (T == NULL)return 0;

int Ldepth = BiTreeDepth(T->lchild);

int Rdepth = BiTreeDepth(T->rchild);

return (((Ldepth > Rdepth) ? Ldepth : Rdepth)+1);

}

//保存二叉树

status SaveBiTree(BiTree T)

{

if (T == NULL)return ERROR;

BiTree tree[MAXSIZE]; //构造一个栈用于前序非递归遍历树

int top = 0,fal=-1;

FILE\*fp;

char Filename[30] = "D://";

char tail[5] = ".txt";

char sh[30];

printf("请输入保存的名称：");

scanf("%s", &sh);

strcat(Filename,sh); //拼接成完整的文件名

strcat(Filename, tail);

fp = fopen(Filename, "w");

if (!fp)return ERROR;

tree[top++] = T;

while (top)

{

T = tree[--top]; //栈顶元素出栈

fwrite(&T->loc, sizeof(int), 1, fp); //将栈顶元素的位置存进文件

if (T->rchild != NULL)

{

tree[top++] = T->rchild;

fwrite(&T->rchild->loc, sizeof(int), 1, fp); //右孩子位置存进文件

}

else

fwrite(&fal, sizeof(int), 1, fp); //右孩子不存在，存入-1

if (T->lchild != NULL)

{

tree[top++] = T->lchild;

fwrite(&T->lchild->loc, sizeof(int), 1, fp);

}

else

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

fwrite(&T->stu->name, sizeof(char), 20, fp);

fwrite(&T->stu->number, sizeof(int), 1, fp);

}

fclose(fp);

return OK;

}

//加载二叉树

status LoadBiTree(BiTree &T)

{

BiTree tree[MAXSIZE]; //将文件中所有的内容读到一个数组中

memset(&tree, NULL, MAXSIZE);

int Relation[MAXSIZE][2]; //保存节点与左右孩子的关系

memset(&Relation, -1, MAXSIZE \* 2);

char filename[100];

int i, j, k,number;

int count=0;

char name[20];

printf("请输入需要加载的文件的名字\n");

loop:scanf("%s", &filename);

if (\_access(filename, 0) == -1) //检查系统中是否存在文件

{

printf("输入文件不存在,请重新输入\n");

goto loop;

}

FILE \*fp;

fp = fopen(filename, "r");

if (!fp)return ERROR;

while (!feof(fp))

{

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

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

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

fread(&name, sizeof(char), 20, fp);

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

if (!feof(fp))

{

tree[i] = (BiTree)malloc(sizeof(BiTnode));

tree[i]->stu = (Student\*)malloc(sizeof(Student));

strcpy(tree[i]->stu->name,name);

tree[i]->stu->number = number;

Relation[i][1] = j; //右孩子位置

Relation[i][0] = k; //左孩子位置

tree[i]->exit = 1;

tree[i]->loc = i;

if (j > k&&j>count)

count = j;

else if(k>j&&k>count)

count = k;

}

else

break; //已经到达文件尾部

}

//文件已经全部读出完毕，现在根据读到的各结点间位置关系为每一个结点建立关系

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

{

if (tree[i] != NULL)

{

if (Relation[i][0] == -1)

tree[i]->lchild = NULL;

else

tree[i]->lchild = tree[Relation[i][0]];

if (Relation[i][1] == -1)

tree[i]->rchild = NULL;

else

tree[i]->rchild = tree[Relation[i][1]];

}

}

for (int i = 0; i < MAXSIZE;i++)

if (tree[i] != NULL)

{

T = tree[i];

break;

}

return OK;

}

status Judge(char \*target,int n)

{

int i = 0;

int m =0, t = 0;

while (i < n)

{

if (target[i] == '\*')

m++;

else if (target[i] == '@')

t++;

else

return ERROR;

i++;

}

if ((t - 1) != m)return ERROR;

return OK;

}

//清空二叉树

status ClearBiTree(BiTree T)

{

//分别删除左右子树，只保留根节点

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

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

T->exit = 0;

return OK;

}

//判断是否为空树

status BiTreeEmpty(BiTree T)

{

if (T->exit != 1)return OK;

else

return ERROR;

}

//获取某个结点的值

BiTree Value(BiTree T, int e)

{

BiTree Tree[MAXSIZE]; //非递归遍历树直到找到节点

int top = 0;

Tree[top++] = T;

while (top)

{

T = Tree[--top];

if (T->stu->number == e)return T; //满足则返回当前结点

if (T->rchild)Tree[top++] = T->rchild;

if (T->lchild)Tree[top++] = T->lchild;

}

return NULL; //遍历完整个树都没有找到，返回空

}

//节点赋值

status Assign(BiTree T, int e, Student\* value)

{

BiTree p=NULL;

int judge;

p = Value(T, e);

if (p == NULL) //没有找到节点，返回ERROR

{

printf("没有找到！");

return ERROR;

}

else

{

printf("请输入修改后的学号:\n");

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

printf("请输入修改后的名字:\n");

scanf("%s", &p->stu->name);

}

return OK;

}

//获取双亲

BiTree Parent(BiTree T, int e)

{

BiTree Tree[MAXSIZE]; //非递归遍历树直到找到节点

int top = 0;

Tree[top++] = T;

while (top)

{

T = Tree[--top];

if (T->rchild&& T->rchild->stu->number == e)return T; //满足则返回当前结点

if (T->lchild&& T->lchild->stu->number == e)return T;

if (T->rchild)Tree[top++] = T->rchild;

if (T->lchild)Tree[top++] = T->lchild;

}

return NULL;

}

//获取左孩子

BiTree LeftChild(BiTree T, int e)

{

BiTree tp;

tp = Value(T, e);

if (tp == NULL)

{

printf("没有找到节点%d\n", e);

return NULL;

}

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

else

printf("没有左孩子!\n");

return NULL;

}

//获得右孩子

BiTree RightChild(BiTree T, int e)

{

BiTree tp;

tp = Value(T, e);

if (tp == NULL)

{

printf("没有找到节点%d\n", e);

return NULL;

}

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

else

printf("没有右孩子!\n");

return NULL;

}

//获得左兄弟

BiTree LeftSibling(BiTree T, int e)

{

BiTree Tr;

Tr = Parent(T, e); //先找到父节点

if (Tr == NULL) //父节点为空，左兄弟也一定不存在

{

printf("没有左兄弟\n");

return NULL;

}

if ((Tr->lchild == NULL)||(Tr->lchild->stu->number == e) ) //当前即为左孩子或者没有左兄弟

{

printf("没有左兄弟\n");

return NULL;

}

else

return Tr->lchild;

}

//获得右兄弟

BiTree RightSibling(BiTree T, int e)

{

BiTree Tr;

Tr = Parent(T, e); //先找到父节点

if (Tr == NULL) //父节点为空，右兄弟也一定不存在

{

printf("没有右兄弟\n");

return NULL;

}

if ((Tr->rchild == NULL)||(Tr->rchild->stu->number == e)) //当前即为左孩子或者没有左兄弟

{

printf("没有右兄弟\n");

return NULL;

}

else

return Tr->rchild;

}

//删除子树

status DeleteChild(BiTree T, BiTree p, int LR)

{

BiTree Tr = NULL;

Tr = Value(T, p->stu->number);

if (Tr == NULL)return ERROR;

if (LR== 0)

{

if (Tr->lchild)

DestroyBiTree(Tr->lchild);

Tr->lchild = NULL;

}

else if (LR == 1)

{

if (Tr->rchild)

DestroyBiTree(Tr->rchild);

Tr->rchild = NULL;

}

else //LR的值不为0或1

{

printf("输入错误\n");

return ERROR;

}

return OK;

}

//插入子树

status InsertChild(BiTree T, BiTree p, int LR, BiTree c)

{

BiTree Tr;

Tr = Value(T, p->stu->number); //先找到p在树中的位置

if (Tr == NULL)

return ERROR;

if (LR == 0)

{

c->rchild = Tr->lchild; //将c插到Tr的左子树上，同时将Tr的左子树插到c的右子树上

Tr->lchild = c;

}

else if (LR == 1)

{

c->rchild = Tr->rchild; //将c插到Tr的右子树上，同时将Tr的左子树插到c的右子树上

Tr->rchild = c;

}

else

{

printf("选项错误\n");

return ERROR;

}

return OK;

}

//打印结点

status PrintTree(BiTree N,int &row,int &flag)

{

int first=0, end; //每行的起始和结尾

int loc = N->loc;

if (loc == 1)

{

for (int t = 0; t <= 32;t++)

printf(" ");

printf("%s ",N->stu->name);

flag = 1;

}

else if ((loc >= 2) && (loc <= 3))

{

loc = loc - 2;

if (flag == 1)

{

printf("\n");

first = 0;

row++;

for (int t = 0; t <= 16; t++)

printf(" ");

flag = 2;

}

for (first; first < loc; first++)

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

printf("%s ", N->stu->name);

first = loc;

}

else if ((loc >= 4) && (loc <= 7))

{

loc = loc - 4;

if (flag == 2)

{

printf("\n");

first = 0;

row++;

for (int t = 0; t <= 8; t++)

printf(" ");

flag = 3;

}

for (first; first < loc; first++)

printf("\_\_\_\_\_\_\_\_\_\_");

printf("%s ", N->stu->name);

first = loc;

}

else if ((loc >= 8) && (loc <= 15))

{

loc = loc - 8;

if (flag == 3)

{

printf("\n");

first = 0;

row++;

for (int t = 0; t <= 4; t++)

printf(" ");

flag = 4;

}

for (first; first < loc; first++)

printf("\_\_\_");

printf("%s ", N->stu->name);

first = loc;

}

else

{

loc = loc - 16;

if (flag == 4)

{

printf("\n");

first = 0;

row++;

for (int t = 0; t <= 2; t++)

printf("");

flag = 5;

}

for (first; first < loc; first++)

printf("\_");

printf("%s", N->stu->name);

first = loc;

}

return OK;

}

//图形化显示树

status Graph(BiTree T)

{

if (!T)return ERROR; //如果树为空，则返回

int deepth = BiTreeDepth(T);

if (deepth > 5)

{

printf("树层数大于5，不能显示\n");

return ERROR;

}

BiTree Tree[MAXSIZE], temp; //构造一个队列用于保存节点

int i = 0, j = 0,flag=0,row=0;

Tree[i++] = T;

while (i > j)

{

PrintTree(Tree[j],row,deepth);

if (Tree[j]->lchild) //如果对头元素有左孩子，加入队尾

{

Tree[i] = Tree[j]->lchild; i++;

}

if (Tree[j]->rchild) //如果对头元素有右孩子,加入队尾

{

Tree[i] = Tree[j]->rchild; i++;

}

j++; //出队操作，对头指针向后移动

}

printf("\n");

return OK;

}

//增加一个新树

SqList AddTree(SqList& L)

{

SqList p,q;

p = L;

while (p->next)

{

p = p->next;

}

q = (SqList)malloc(sizeof(Sqnode));

q->elem = (M\_Tree\*)malloc(sizeof(M\_Tree));

if (!q)return ERROR;

q->next = NULL;

p->next = q;

printf("添加成功\n");

return q;

}

//切换另一棵树进行操作

SqList ChangeTree(SqList L)

{

char name[20];

printf("请输入需要切换的树\n");

scanf("%s", name);

SqList p = L;

while (p&&strcmp(p->elem->name, name))

{

p = p->next;

}

return p; //无论p是否为空都返回，主程序中再判断

}

//删除树

status DeleteSqList(SqList& L)

{

printf("请输入需要删除的树\n");

char name[20];

scanf("%s", &name);

SqList p=L,q;

if (!strcmp(L->elem->name, name))

{

L = L->next;

return OK;

}

else

{

while (p&&(strcmp(p->elem->name, name)))

{

q = p;

p = p->next;

}

if (p == NULL){

printf("没有找到\n");

return ERROR;

}

else

{

q->next = p->next;

free(p);

return OK;

}

}

}

//root

BiTree Root(BiTree T)

{

return T;

}

void main()

{

int op = 1,th;

BiTree T=NULL,temp=NULL;

SqList L=NULL,p=NULL,q=NULL;

sequence = 0;

InitSqList(L);

p = L;

while (op)

{

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

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

printf("1. InitBiTree 13.BiTreeEmpty \n");

printf("2. DestoryBiTree 14.Value \n");

printf("3. CreateBiTree 15.Assign \n");

printf("4. PreOrderTraverse 16.LeftChild \n");

printf("5. InOrderTraverse 17.RightChild \n");

printf("6. PostOrderTraverse 18.LeftSibing \n");

printf("7. LeverOrderTraverse 19.RightSibing \n");

printf("8. TreeDepth 20.DeleteChild \n");

printf("9. SaveTree 21.InsertChild \n");

printf("10. ClearTree 22.DeleteTree \n");

printf("11. Parent 23.LoadTree \n");

printf("12. Graph 24.AddTree \n");

printf("0.exit 25.ChangeTree \n");

printf(" 26.Root\n");

if (p->elem->flag == 1)printf("当前操作树为:%s\n", p->elem->name);

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

printf(" 请选择你的操作[0~24]:\n");

scanf("%d", &op);

switch (op)

{

case 1:

if (InitBiTree(p->elem))

printf("初始化成功!\n");

system("pause");

break;

case 2:

if (p->elem->flag != 1)

printf("还没有初始化哦\n");

else if (p->elem->tree->exit == 1)

{

DestroyBiTree(p->elem->tree);

printf("销毁成功了呢!\n");

}

else

printf("失败\n");

system("pause");

break;

case 3:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else

{

char i=0;

printf("请以前序遍历顺序输入正确的\*@串，\*表示节点存在，@表示不存在,以回车结束。\n");

scanf("%s", &definition);

if(Judge(definition, strlen(definition))==ERROR)printf("输入错误！\n");

else

{

printf("%s", definition);

if (CreateBiTree(p->elem->tree, definition)){

printf("创建成功\n");

Location(p->elem->tree);

}

}

}

system("pause");

break;

case 4:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (p->elem->tree->exit != 1)

printf("当前是棵空树呢！\n");

else

{

PreOrderTraverse(p->elem->tree, Visit);

}

system("pause");

break;

case 5:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (p->elem->tree->exit!=1)

printf("当前是棵空树呢！\n");

else

{

InOrderTraverse(p->elem->tree, Visit);

}

system("pause");

break;

case 6:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (p->elem->tree->exit != 1)

printf("当前是棵空树呢！\n");

else

{

PostOrderTraverse(p->elem->tree, Visit);

}

system("pause");

break;

case 7:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (p->elem->tree->exit != 1)

printf("当前是棵空树呢！\n");

else

{

LevelOrderTraverse(p->elem->tree, Visit);

}

system("pause");

break;

case 8:

if (p->elem->flag != 1)

printf("还没有初始化哦\n");

else if (p->elem->tree->exit != 1)

printf("还是空树哦！\n");

else

{

int count = 0;

count = BiTreeDepth(p->elem->tree);

printf("树的深度为：%d", count);

}

system("pause");

break;

case 9:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (p->elem->tree->exit!=1)

{

printf("树为空\n");

}

else

{

if(SaveBiTree(p->elem->tree))printf("保存成功\n");

}

system("pause");

break;

case 10:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if(ClearBiTree(p->elem->tree))

printf("清空成功!\n");

system("pause");

break;

case 11:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (BiTreeEmpty(p->elem->tree))

printf("当前是棵空树!\n");

else

{

printf("请输入需要查找的学号：\n");

scanf("%d", &th);

temp = Parent(p->elem->tree, th);

if (temp == NULL)printf("不存在或没有双亲!\n");

else

{

printf("双亲的信息为：\n");

Visit(temp->stu);

}

}

system("pause");

break;

case 12:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (BiTreeEmpty(p->elem->tree))

printf("当前是棵空树!\n");

else

Graph(p->elem->tree);

system("pause");

break;

case 13:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (BiTreeEmpty(p->elem->tree))

printf("当前是棵空树!\n");

else

printf("不是空树！\n");

system("pause");

break;

case 14:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (BiTreeEmpty(p->elem->tree))

printf("当前是棵空树!\n");

else

{

printf("请输入你需要查找的学号:\n");

int e;

scanf("%d", &e);

temp = Value(p->elem->tree, e);

if (temp == NULL)

printf("查找的结点不存在!\n");

else

Visit(temp->stu);

}

system("pause");

break;

case 15:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (BiTreeEmpty(p->elem->tree))

printf("当前是棵空树!\n");

else

{

printf("请输入你需要修改的学号：\n");

Student\* stu;

stu = (Student\*)malloc(sizeof(Student));

scanf("%d", &stu->number);

th = stu->number;

if (Assign(p->elem->tree, th, stu))

printf("修改成功！\n");

else

printf("修改失败！\n");

}

system("pause");

break;

case 16:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (BiTreeEmpty(p->elem->tree))

printf("当前是棵空树!\n");

else

{

printf("请输入需要查找的值\n");

scanf("%d", &th);

temp = LeftChild(p->elem->tree, th);

if (temp)

{

printf("左孩子的信息为：\n");

Visit(temp->stu);

}

}

system("pause");

break;

case 17:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (BiTreeEmpty(p->elem->tree))

printf("当前是棵空树!\n");

else

{

printf("请输入需要查找的值\n");

scanf("%d", &th);

temp = RightChild(p->elem->tree, th);

if (temp)

{

printf("右孩子的信息为：\n");

Visit(temp->stu);

}

}

system("pause");

break;

case 18:

if (p->elem->flag != 1)

printf("还没有初始化哦\n");

else if (BiTreeEmpty(p->elem->tree))

printf("当前是棵空树!\n");

else

{

printf("输入你需要查找的学号\n");

scanf("%d", &th);

temp = LeftSibling(p->elem->tree, th);

if (temp != NULL)

{

printf("左兄弟的信息为：\n");

Visit(temp->stu);

}

}

system("pause");

break;

case 19:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (BiTreeEmpty(p->elem->tree))

printf("当前是棵空树!\n");

else

{

printf("输入你需要查找的学号\n");

scanf("%d", &th);

temp = RightSibling(p->elem->tree, th);

if (temp != NULL)

{

printf("右兄弟的信息为：\n");

Visit(temp->stu);

}

}

system("pause");

break;

case 20:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (BiTreeEmpty(p->elem->tree))

printf("当前是棵空树!\n");

else

{

printf("请输入需要删除的结点的学号:\n");

scanf("%d", &th);

temp = Value(p->elem->tree, th);

if (temp == NULL)

{

printf("没有查找到该节点\n");

}

else

{

printf("0、删除左子树 1、删除右子树\n");

scanf("%d", &th);

if (DeleteChild(p->elem->tree, temp, th))

printf("删除成功\n");

else

printf("删除失败\n");

}

}

system("pause");

break;

case 21:

if (p->elem->flag!=1)

printf("还没有初始化哦\n");

else if (BiTreeEmpty(p->elem->tree))

printf("当前是棵空树!\n");

else

{

BiTree t;

printf("请按先序序列构造树，同时保证右子树为空，否则会造成右子树数据丢失\n");

printf("请以前序遍历顺序输入正确的\*@串，\*表示节点存在，@表示不存在,以回车结束。\n");

scanf("%s", &definition);

if (Judge(definition, strlen(definition)) == ERROR)printf("输入错误！\n");

else

{

printf("%s", definition);

if (CReateBiTree(p->elem->tree,t, definition))printf("创建成功\n");

printf("输入你需要插入结点的ID：\n");

scanf("%d", &th);

temp = Value(p->elem->tree, th);

if (temp == NULL)

printf("没有找到\n");

else

{

printf("0:插入到左子树，1：插入到右子树\n");

loop:scanf("%d", &th);

if (th != 0 && th != 1)

{

printf("输入错误,从新输入\n");

goto loop;

}

if(InsertChild(p->elem->tree,temp,th,t))

printf("插入成功\n");

}

}

Location(p->elem->tree);

}

system("pause");

break;

case 23:

if (LoadBiTree(p->elem->tree))printf("加载成功!");

else

printf("加载失败!\n");

system("pause");

break;

case 24:

p = AddTree(L);

system("pause");

break;

case 25:

q = ChangeTree(L);

if (q == NULL)printf("没有查找到\n");

else

{

printf("切换成功\n");

p = q;

}system("pause");

break;

case 22:

if (DeleteSqList(L))

{

printf("删除成功\n");

if (L == NULL)InitSqList(L);

p = L;

}

else

printf("删除失败\n");

system("pause");

break;

case 26:

if (p->elem->flag != 1)

printf("还没有初始化哦\n");

else if (BiTreeEmpty(p->elem->tree))

printf("当前是棵空树!\n");

else

{

BiTree zanshi;

zanshi = Root(T);

printf("根信息为:\n");

Visit(zanshi->stu);

}

system("pause");

break;

case 0:

break;

default:

system("pause");

break;

}

}

}

## 附录D 基于邻接表的图的源程序

#include<stdlib.h>

#include<malloc.h>

#include<stdio.h>

#include<string.h>

#include<io.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status;

typedef int InfoType;

#define MAXSIZE 100

#define MAX\_VERTEX\_NUM 20

typedef int KeyType;

int ID[100],num; //用来存储唯一ID，防止重复

bool visited[MAX\_VERTEX\_NUM]; //访问标志数组

typedef struct

{

char localName[20]; //地点名称

int ID; //唯一标识ID

}province;

typedef struct ArcNode

{

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

InfoType info; //该弧相关信息的指针

}ArcNode;

typedef struct VNode

{

province\* data; //顶点信息

ArcNode \*firstarc; //指向第一条依附于该顶点的弧的指针

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

typedef struct

{

AdjList vertice;

int vexnum, arcnum; //弧的当前顶点数和弧度数

int kind; //图的种类标志

}ALGraph;

//邻接关系矩阵

typedef struct

{

int relation[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM];

}rela;

//顶点信息

typedef struct

{

char Vnode[MAX\_VERTEX\_NUM][20];

}Vnodes;

//多表结构，用于保存多个图

typedef struct Sq

{

ALGraph\* G; //图

rela\* Mtrix; //关系矩阵

char name[20]; //图的名字

struct Sq \*next;

}SqNode,\*SqList;

//销毁图

status DestoryGraph(ALGraph\* G);

status(\*VisitedFunc)(VNode V); //函数变量

//检查ID是否重复

status Inpect(int t);

//创建关系集,用一个一维数组保存

status CreateRela(int vexnum, int arcnum);

//查找节点,根据ID

status LocateVex(ALGraph \*G, int u);

//查找节点，根据地名

status LocateVex(ALGraph \*G, char\* name);

//插入弧

status InsertArc(ALGraph \*G,KeyType v,KeyType w);

//创建顶点集

status CreateNode(Vnodes\* &V, int vexnum);

//创建图

status CreateCraph(ALGraph\* &G,Vnodes \*V,rela \*VR);

//插入顶点

status InsertVex(ALGraph\* &G, VNode\* v);

//获得第一个邻接点

ArcNode\* FirstAdjVex(ALGraph \*G, KeyType v);

status Visit(VNode V)

{

printf("ID:%d\n", V.data->ID);

printf("名字：%s\n", V.data->localName);

return OK;

}

//删除顶点

status DeleteVex(ALGraph\* &G, KeyType v);

//深度优先遍历

void DFSTraverse(ALGraph\* G, status(\*Visit)(VNode V));

void DFS(ALGraph\* G, int v);

//广度优先搜索

void BFSTraverse(ALGraph \*G, status(\*Visit)(VNode V));

//获得顶点值

status GetVex(ALGraph \*G,int v);

//顶点赋值

status PutVex(ALGraph\* G, int v, province\* value);

//删除弧

status DeleteArc(status &G,int v,int w);

//创建关系集,用一个一维数组保存

status CreateRela(int vexnum, int arcnum, rela\* &relation);

//保存图

status SaveGraph(ALGraph\* G,rela\* VR);

//加载图

status LoadGraph(ALGraph\* &G);

//根据ID查找顶点位置

int search(ALGraph \*G, int id);

//删除弧

status DeleteArc(ALGraph\* G, KeyType v, KeyType w);

//初始化多表

status InitSqList(SqList &L);

//获得下一邻接

ArcNode\* NextAdjVex(ALGraph \*G, KeyType v, KeyType w);

//更新关系矩阵

status RenewMtrix(SqList L);

//添加一个新图

SqList AddGraph(SqList &L);

//切换图

SqList ChangeGraph(SqList L, char\* name);

//删除图

status DeleteGraph(SqList &L, char \*name);

//清空ID

void ClearArray();

int main()

{

Vnodes\* V=NULL; //顶点集

num = 0;

ALGraph\* G=NULL;

G = (ALGraph\*)malloc(sizeof(ALGraph));

int op = 1,a,b;

KeyType v, w;

SqList L = NULL,p=NULL,q=NULL;

InitSqList(L);

p = L;

ArcNode \*A=NULL;

while (op)

{

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

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

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

printf("1、CreateCraph 18、DeleteGraph\n");

printf("2、DFSTraverse 10、DeleteArc\n");

printf("3、BFSTraverse 11、FirstArcNode\n");

printf("4、LocateVex 12、NextAdjVex\n");

printf("5、GetVex 13、DestoryGraph\n");

printf("6、PutVex 14、InsertArc\n");

printf("7、InsertVex 15、AddGraph\n");

printf("8、SaveGraph 16、DeleteVex\n");

printf("9、LoadGraph 17、ChangeGraph\n");

printf("0、exit\n");

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

if (p == NULL)InitSqList(p);

if (p->G->kind == 1)

printf("当前操作的图是：%s\n", p->name);

printf("请输入命令[1`18]\n");

scanf("%d", &op);

if (p == NULL) InitSqList(p);

switch (op)

{

case 1:

printf("请输入顶点数和弧数\n");

scanf("%d %d", &a, &b);

if (a <= 0 || a > MAX\_VERTEX\_NUM || b <= 0 || b > MAX\_VERTEX\_NUM)

{

printf("输入错误\n");

}

else

{

p->G = (ALGraph\*)malloc(sizeof(ALGraph));

p->G->vexnum = a; p->G->arcnum = b;

CreateRela(p->G->vexnum, p->G->arcnum, p->Mtrix);

CreateNode(V, p->G->vexnum);

CreateCraph(p->G, V, p->Mtrix);

printf("请主人为小图取个名字吧\n");

scanf("%s", &p->name);

}

system("pause");

break;

case 2:

if (p->G->kind != 1)

{

printf("还没有创建\n");

}

else

{

DFSTraverse(p->G, Visit);

}

system("pause");

break;

case 3:

if (p->G->kind != 1)

{

printf("还没有创建\n");

}

else

BFSTraverse(p->G, Visit);

system("pause");

break;

case 4:

if (p->G->kind != 1)

{

printf("还没有创建\n");

}

else

{

char name[20];

int id;

printf("1、根据ID查找 2、根据名字查找\n");

scanf("%d", &id);

if (id == 1)

{

printf("请输入需要查找的ID\n");

scanf("%d", &id);

if (LocateVex(p->G, id) == ERROR)

printf("没有找到\n");

}

else if (id == 2)

{

printf("请输入需要查找的名字\n");

scanf("%s", &name);

if (!LocateVex(p->G, name))

printf("没有找到\n");

}

else

printf("输入错误\n");

}

system("pause");

break;

case 5:

if (p->G->kind != 1)

{

printf("还没有创建\n");

}

else

{

printf("请输入需要查找的顶点ID\n");

scanf("%d", &a);

if (GetVex(p->G, a)==ERROR)

printf("输入错误\n");

}

system("pause");

break;

case 6:

if (p->G->kind != 1)

{

printf("还没有创建\n");

}

else

{

printf("输入结点ID\n");

scanf("%d", &a);

b = search(p->G, a);

if (b == -1)

{

printf("不存在的ID\n");

}

else

{

province\* po = (province\*)malloc(sizeof(province));

loop5:

printf("请输入修改后的ID\n");

int th;

scanf("%d", &th);

if (Inpect(th))

{

po->ID = th;

printf("请输入修改后的名字\n");

scanf("%s", &po->localName);

if (PutVex(p->G, b, po))

printf("修改成功\n");

else

printf("修改失败\n");

}

else

{

printf("ID重复，重新输入\n");

goto loop5;

}

free(po);

}

}

system("pause");

break;

case 7:

if (p->G->kind != 1)

{

printf("还没有创建\n");

}

else

{

VNode \*node = NULL;

node = (VNode\*)malloc(sizeof(VNode));

node->data = (province\*)malloc(sizeof(province));

printf("请输入要插入顶点名字：\n");

scanf("%s", &node->data->localName);

printf("请输入顶点%s的ID:\n", node->data->localName);

int m;

loop4:scanf("%d", &m);

if (Inpect(m) == OK)

{

node->data->ID = m;

}

else

{

printf("ID重复，重新输入\n");

goto loop4;

}

if (InsertVex(p->G, node))printf("插入成功\n");

else printf("插入失败");

}

system("pause");

break;

case 8:

if (p->G->kind != 1)

{

printf("还没有创建\n");

}

else

{

if (SaveGraph(p->G, p->Mtrix))

printf("保存成功\n");

}

system("pause");

break;

case 9:

if (p->G == NULL)

{

p->G = (ALGraph\*)malloc(sizeof(ALGraph));

if (LoadGraph(p->G))printf("加载成功\n");

else printf("加载失败\n");

}

else

{

printf("继续操作会失去当前图中的信息,继续吗？(回车继续)\n");

char ch;

getchar();

ch = getchar();

if (ch == '\n'){

if (LoadGraph(p->G))

{

printf("加载成功\n");

RenewMtrix(p);

printf("这张新图的名字叫什么呀?\n");

scanf("%s", &p->name);

ClearArray();;

}

else printf("加载失败\n");

}

}

system("pause");

break;

case 10:

if (p->G->kind != 1)

{

printf("还没有创建\n");

}

else

{

printf("请输入顶点的id，中间用空格隔开\n");

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

if (DeleteArc(p->G, v, w))

printf("删除成功\n");

RenewMtrix(p);

}

system("pause");

break;

case 11:

if (p->G->kind != 1)

{

printf("还没有创建\n");

}

else

{

printf("输入需要查找的ID\n");

scanf("%d", &v);

A = FirstAdjVex(p->G, v);

if (A == NULL)

printf("顶点不存在或者没有邻接点\n");

else

{

printf("第一邻接点信息为:\n");

printf("ID:%d\n", p->G->vertice[A->adjvex].data->ID);

printf("名字:%s\n", p->G->vertice[A->adjvex].data->localName);

}

}

system("pause");

break;

case 12:

if (p->G->kind != 1)

{

printf("还没有创建\n");

}

else

{

printf("请输入需要查找的顶点，中间用空格隔开\n");

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

A = NextAdjVex(p->G, v, w);

if (A != NULL)

Visit(p->G->vertice[A->adjvex]);

else

printf("NULL\n");

}

system("pause");

break;

case 13:

if (p->G->kind != 1)

{

printf("还没有创建\n");

}

else

{

if (DestoryGraph(p->G))printf("销毁成功\n");

}

system("pause");

break;

case 14:

if (p->G->kind != 1)

{

printf("还没有创建\n");

}

else

{

printf("请输入需要插入的弧的两端顶点ID\n");

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

if (InsertArc(p->G, v, w))

printf("插入成功\n");

else

printf("插入失败\n");

RenewMtrix(p);

}

system("pause");

break;

case 15:

q = AddGraph(L);

if (q)

p = q;

else

printf("添加失败\n");

system("pause");

break;

case 16:

if (p->G->kind != 1)

{

printf("还没有创建\n");

}

else

{

printf("请输入需要删除的顶点的ID\n");

scanf("%d", &v);

if (search(p->G, v) == -1)

printf("ID不存在\n");

else

{

if (DeleteVex(p->G, v) == OK)

printf("删除成功\n");

else

printf("删除失败\n");

RenewMtrix(p);

}

}

system("pause");

break;

case 17:

printf("请输入需要切换到的图的名字\n");

char Graphname[20];

scanf("%s", &Graphname);

q = ChangeGraph(L, Graphname);

if (q == NULL)

printf("不存在\n");

else

{

p = q;

printf("切换成功\n");

ClearArray();;

}

system("pause");

break;

case 18:

printf("请输入需要删除的图的名字\n");

getchar();

gets(Graphname);

if (DeleteGraph(L, Graphname))

{

printf("删除成功\n");

ClearArray();;

}

else

printf("图不存在\n");

system("pause");

break;

case 0:

break;

default:

system("pause");

break;

}

}

return 0;

}

//创建顶点集

status CreateNode(Vnodes\* &V,int vexnum)

{

V = (Vnodes\*)malloc(sizeof(Vnodes));

for (int i = 0; i < vexnum; i++) //num表示顶点数目，依次为每个顶点赋值

{

printf("请输入第%d个节点的地名\n", i + 1);

scanf("%s", &V->Vnode[i]);

}

return OK;

}

//创建关系集,用一个一维数组保存

status CreateRela(int vexnum,int arcnum,rela\* &realtion)

{

int m, n,t=0;

realtion = (rela\*)malloc(sizeof(rela));

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

memset(&realtion->relation[i], 0, MAX\_VERTEX\_NUM);

printf("请输入两个节点的序号表示它们之间有关系，中间以空格隔开\n");

char ch = '\*';

while (t<arcnum)

{

loop6:

scanf("%d %d", &m, &n);

if (m<0 || m>vexnum || n < 0 || n > vexnum||m==n)

{

printf("输入错误，重新输入\n");

goto loop6;

}

else

{

ch = getchar();

realtion->relation[m-1][n-1] = 1;

}

t++;

}

return OK;

}

//检查ID是否重复

status Inpect(int t)

{

for (int i = 0; i < num; i++)

{

if (ID[i] == t)

{

printf("ID重复\n");

return ERROR;

}

}

ID[num++] = t;

return OK;

}

//创建图

status CreateCraph(ALGraph\* &G, Vnodes\* V, rela \*VR)

{

if (!G)return ERROR;

for (int i = 0; i < G->vexnum; i++) //把顶级集中的顶点信息拷贝到图中

{

G->vertice[i].data = (province\*)malloc(sizeof(province));

strcpy(G->vertice[i].data->localName, V->Vnode[i]);

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

}

for (int i = 0; i < G->vexnum; i++) //输入每个顶点ID

{

printf("请输入顶点%s的ID:\n",G->vertice[i].data->localName);

int m;

it:scanf("%d", &m);

if (Inpect(m) == OK)

{

G->vertice[i].data->ID = m;

}

else

{

printf("ID重复，重新输入\n");

goto it;

}

}

ArcNode\* p = NULL;

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

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

{

if (VR->relation[i][j] == 1)

{

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

p->nextarc = NULL;

p->adjvex = j;

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

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

}

}

G->kind = 1;

return OK;

}

//深度优先搜索

void DFSTraverse(ALGraph\* G, status(\*Visit)(VNode V))

{

for (int v = 0; v < G->vexnum; v++) //访问数组初始化

visited[v] = FALSE;

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

if (!visited[v])

DFS(G, v); //对尚未访问的节点调用DFS

}

void DFS(ALGraph\* G, int v)

{

visited[v] = TRUE;

Visit(G->vertice[v]); //访问第v个顶点

for (ArcNode \*p = G->vertice[v].firstarc; p != NULL; p = p->nextarc)

{

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

}

}

//广度优先搜索

void BFSTraverse(ALGraph \*G, status(\*Visit)(VNode V))

{

int v;

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

visited[v] = FALSE;

int array[MAX\_VERTEX\_NUM\*20], first = 0, tail = 0,temp; //作为辅助队列

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

if (!visited[v])

{

visited[v] = TRUE;

Visit(G->vertice[v]);

array[tail++] = v;

while (first < tail)

{

temp = array[first++];

for (ArcNode \*p = G->vertice[temp].firstarc; p != NULL; p = p->nextarc)

{

if (!visited[p->adjvex])

{

visited[p->adjvex] = TRUE;

Visit(G->vertice[p->adjvex]);

array[tail++] = p->adjvex;

}

}

}

}

}

//根据ID查找

status LocateVex(ALGraph \*G, int u)

{

int v = 0;

int array[MAX\_VERTEX\_NUM], frist=0, tail=0,temp; //辅助对列

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

visited[v] = FALSE;

for (v = 0; v < G->vexnum;v++) //广度优先搜索查找

if (!visited[v])

{

if (G->vertice[v].data->ID == u)

{

printf("%d是图中的第%d个顶点\n", u, v+1);

Visit(G->vertice[v]);

return OK;

}

visited[v] = TRUE;

array[tail++] = v;

while (frist < tail)

{

temp = array[frist++];

for (ArcNode \*p = G->vertice[temp].firstarc; p != NULL; p = p->nextarc)

{

if (!visited[p->adjvex])

{

visited[p->adjvex] = TRUE;

if (G->vertice[p->adjvex].data->ID == u)

{

printf("%d是图中的第%d个顶点\n", u, p->adjvex+1);

Visit(G->vertice[p->adjvex]);

return OK;

} array[tail++] = p->adjvex;

}

}

}

}

return ERROR;

}

//查找节点，根据地名

status LocateVex(ALGraph \*G, char\* name)

{

int v = 0;

int array[MAX\_VERTEX\_NUM], frist = 0, tail = 0, temp; //辅助对列

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

visited[v] = FALSE;

for (v = 0; v < G->vexnum; v++) //广度优先搜索查找

if (!visited[v])

{

if (!strcmp(name,G->vertice[v].data->localName))

{

printf("%s是图中的第%d个顶点\n", name, v+1);

Visit(G->vertice[v]);

return OK;

}

visited[v] = TRUE;

array[tail++] = v;

while (frist < tail)

{

temp = array[frist++];

for (ArcNode \*p = G->vertice[temp].firstarc; p != NULL; p = p->nextarc)

{

if (!visited[p->adjvex])

{

visited[p->adjvex] = TRUE;

if (!strcmp(name, G->vertice[p->adjvex].data->localName))

{

printf("%s是图中的第%d个顶点\n", name, p->adjvex+1);

Visit(G->vertice[p->adjvex]);

return OK;

}

array[tail++] = p->adjvex;

}

}

}

}

return ERROR;

}

//获得顶点值

status GetVex(ALGraph\* G, int vn)

{

int v = 0;

int array[MAX\_VERTEX\_NUM], frist = 0, tail = 0, temp; //辅助对列

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

visited[v] = FALSE;

for (v = 0; v < G->vexnum; v++) //广度优先搜索查找

if (!visited[v])

{

if (G->vertice[v].data->ID == vn)

{

Visit(G->vertice[v]);

return OK;

}

visited[v] = TRUE;

array[tail++] = v;

while (frist < tail)

{

temp = array[frist++];

for (ArcNode \*p = G->vertice[temp].firstarc; p != NULL; p = p->nextarc)

{

if (!visited[p->adjvex])

{

visited[p->adjvex] = TRUE;

if (G->vertice[p->adjvex].data->ID == vn)

{

Visit(G->vertice[p->adjvex]);

return OK;

} array[tail++] = p->adjvex;

}

}

}

}

return ERROR;

}

//顶点赋值

status PutVex(ALGraph\* G, int vn,province\* value )

{

if (!G || G->kind != 1)

return ERROR;

G->vertice[vn].data->ID = value->ID;

strcpy(G->vertice[vn].data->localName, value->localName);

return OK;

}

//插入顶点

status InsertVex(ALGraph\* &G, VNode\* v)

{

if (G == NULL || G->kind != 1)return ERROR;

if (G->vexnum >= MAXSIZE - 1)return ERROR; //结构数组已满

G->vexnum++; //顶点数加一

G->vertice[G->vexnum-1].data = (province\*)malloc(sizeof(province));

strcpy(G->vertice[G->vexnum-1].data->localName, v->data->localName);

G->vertice[G->vexnum-1].data->ID = v->data->ID;

G->vertice[G->vexnum - 1].firstarc = NULL;

return OK;

}

//保存图

status SaveGraph(ALGraph\* G,rela\* VR)

{

FILE\* fp;

char fupan[50] = "D://";

char tail[10] = ".txt";

char name[20];

printf("输入需要保存的文件名字\n");

scanf("%s", &name);

strcat(fupan, name);

strcat(fupan, tail); //拼接成一个完整的文件路径

fp = fopen(fupan,"w");

if (!fp)exit(-1);

fwrite(&G->vexnum, 1, sizeof(int), fp);

for (int i = 0; i < G->vexnum; i++) //保存顶点数目

{

fwrite(&G->vertice[i].data->localName,20, sizeof(char), fp);

fwrite(&G->vertice[i].data->ID, 1, sizeof(int), fp);

}

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

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

{

fwrite(&VR->relation[i][j], 1, sizeof(int), fp); //保存关系矩阵

}

fclose(fp);

return OK;

}

//根据ID查找顶点位置

int search(ALGraph\* G, int id)

{

int v = 0;

int array[MAX\_VERTEX\_NUM], frist = 0, tail = 0, temp; //辅助对列

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

visited[v] = FALSE;

for (v = 0; v < G->vexnum; v++) //广度优先搜索查找

if (!visited[v])

{

if (G->vertice[v].data->ID == id)

{

return v;

}

visited[v] = TRUE;

array[tail++] = v;

while (frist < tail)

{

temp = array[frist++];

for (ArcNode \*p = G->vertice[temp].firstarc; p != NULL; p = p->nextarc)

{

if (!visited[p->adjvex])

{

visited[p->adjvex] = TRUE;

if (G->vertice[p->adjvex].data->ID == id)

{

return p->adjvex;

}

array[tail++] = p->adjvex;

}

}

}

}

return -1; //没有找到

}

//加载图

status LoadGraph(ALGraph\* &G)

{

Vnodes\* V=NULL; // 保存顶点信息

rela\* VR; //保存关系矩阵

V = (Vnodes\*)malloc(sizeof(Vnodes));

VR = (rela\*)malloc(sizeof(rela));

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

memset(&VR->relation[i], 0, MAX\_VERTEX\_NUM);

FILE\* fp;

char filepath[30];

printf("请输入文件名称\n");

file:gets(filepath);

if (\_access(filepath, 0) == -1)

{

printf("文件不存在，从新输入\n");

goto file;

}

G = (ALGraph\*)malloc(sizeof(ALGraph));

fp = fopen(filepath,"r");

int m;

while (!feof(fp))

{

fread(&m, 1, sizeof(int), fp); //读取顶点个数

if (!feof(fp))

G->vexnum = m;

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

{

if (!feof(fp))

{

G->vertice[i].data = (province\*)malloc(sizeof(province));

fread(&G->vertice[i].data->localName, 20, sizeof(char), fp); //读取顶点信息

fread(&G->vertice[i].data->ID, 1, sizeof(int), fp);

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

}

}

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

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

{

fread(&m, 1, sizeof(int), fp); //读取关系矩阵

if (!feof(fp))

VR->relation[i][j] = m;

}

}

ArcNode\* p = NULL;

for (int i = 0; i < G->vexnum; i++) //根据关系矩阵建立邻接表

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

{

if (VR->relation[i][j] == 1)

{

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

p->nextarc = NULL;

p->adjvex = j;

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

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

}

}

G->kind = 1;

fclose(fp);

return OK;

}

//插入弧

status InsertArc(ALGraph \*G, KeyType v, KeyType w)

{

int m = search(G, v); //找到顶点在图中的位置

int n = search(G, w);

if (m == -1 || n == -1) //输入的ID不存在

{ printf("不存在的ID\n");

return ERROR;

}

else

{

ArcNode\* p = NULL;

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

p->nextarc = NULL;

p->adjvex = n;

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

G->vertice[m].firstarc = p;

return OK;

}

}

//删除弧

status DeleteArc(ALGraph\* G, KeyType v, KeyType w)

{

int m = search(G, v); //找到顶点在邻接表中的序号

int n = search(G, w);

if (m == -1 || n == -1)

{

printf("ID不存在\n");

return ERROR;

}

ArcNode\* p = NULL,\*q=NULL;

if (G->vertice[m].firstarc == NULL)

{

printf("%d和%d之间没有弧\n", v, w);

return ERROR;

}

if (G->vertice[m].firstarc->adjvex == n) //对第一个邻接点特殊处理

{

p = G->vertice[m].firstarc;

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

free(p);

return OK;

}

for (p = G->vertice[m].firstarc; p != NULL;)

{

q = p;

p = p->nextarc;

if (p == NULL)break;

if (p->adjvex == n)

{

q->nextarc = p->nextarc;

free(p);

return OK;

}

}

printf("%d和%d之间没有弧\n", v, w);

return ERROR;

}

//初始化多表

status InitSqList(SqList &L)

{

L = (SqList)malloc(sizeof(SqNode));

if (!L)return ERROR;

L->next = NULL;

L->G = (ALGraph\*)malloc(sizeof(ALGraph));

L->G->kind = 0;

L->Mtrix = (rela\*)malloc(sizeof(rela));

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

memset(&L->Mtrix->relation[i], 0, MAX\_VERTEX\_NUM);

return OK;

}

//获得第一个邻接点

ArcNode\* FirstAdjVex(ALGraph \*G, KeyType v)

{

int m = search(G, v);

if (m == -1)return NULL; //顶点不存在

if (G->vertice[m].firstarc == NULL)return NULL; //没有邻接顶点

else

return G->vertice[m].firstarc;

}

//获得下一邻接

ArcNode\* NextAdjVex(ALGraph \*G, KeyType v, KeyType w)

{

int m = search(G, v); //找到v和w的位置

int n = search(G, w);

if (m == -1 || n == -1)

{

printf("顶点不存在\n"); //顶点不存在

return NULL;

}

ArcNode \*p = G->vertice[m].firstarc;

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

{

if (p->adjvex == n&&p->nextarc != NULL) //找到顶点

return p->nextarc;

}

return NULL;

}

//更新关系矩阵

status RenewMtrix(SqList L)

{

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

for (ArcNode\* p = L->G->vertice[i].firstarc; p != NULL; p = p->nextarc)

L->Mtrix->relation[i][p->adjvex] = 1;

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

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

// printf("%d\t", L->Mtrix->relation[i][j]);

return OK;

}

//添加一个新图

SqList AddGraph(SqList &L)

{

SqList p = L,q=NULL;

if (L->G->kind != 1)

{

printf("请输入名字\n");

scanf("%s", &L->name);

printf("添加成功\n");

return L;

}

InitSqList(q);

q->next = L->next;

L->next = q;

printf("请输入名字\n");

scanf("%s", &q->name);

return q;

}

//切换图

SqList ChangeGraph(SqList L, char\* name)

{

if (strcmp(name, L->name) == 0)

return L;

if (L->next == NULL)

{

return ERROR;

}

if (L->G->kind != 1)return ERROR;

SqList p = L;

for (p; p != NULL&&p->G->kind==1;p=p->next)

if (strcmp(name, p->name) == 0)

return p;

return NULL;

}

//删除图

status DeleteGraph(SqList &L, char \*name)

{

SqList p = L, q;

if (L == NULL)return ERROR;

if (L->G->kind != 1)return ERROR;

if (strcmp(L->name, name) == 0)

{

L = L->next;

free(p);

return OK;

}

q = p;

for (p; p != NULL; p = p->next)

{

if (strcmp(p->name, name) == 0)

{

q->next = p->next;

free(p);

return OK;

}

q = p;

}

return ERROR;

}

//销毁图

status DestoryGraph(ALGraph\* G)

{

free(G->vertice);

G->kind = 0;

return OK;

}

//删除顶点

status DeleteVex(ALGraph\* &G, KeyType v)

{

int m = search(G, v); //找到v在邻接表中的位置

ArcNode\*p = NULL, \*q = NULL;

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

{

q=p = G->vertice[i].firstarc; //遍历整个邻接表，删除每个顶点与v相关的弧

if (G->vertice[i].firstarc!=NULL&&G->vertice[i].firstarc->adjvex == m) //对第一个邻接点单独处理

{

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

free(p);

}

else

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

{

if (p->adjvex == m)

{

q->nextarc = p->nextarc;

free(p);

break;

}

q = p;

}

}

for (int i = m; i < G->vexnum-1; i++) //将m后面的表都往前移动

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

G->vexnum--; //顶点数目减1

return OK;

}

//清空ID

void ClearArray()

{

for (int i = 0; i < num; i++)

{

ID[i] = -1;

}

num = 0;

}