

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级：**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期：**

目 录

[1 基于顺序存储结构的线性表实现 4](#_Toc503456861)

[1.1 问题描述 4](#_Toc503456862)

[1.1.1 线性表基本运算定义 4](#_Toc503456863)

[1.2 系统设计 6](#_Toc503456864)

[1.2.1 系统总体设计 6](#_Toc503456865)

[1.2.2 有关常量和类型的定义 7](#_Toc503456866)

[1.2.3 算法设计 8](#_Toc503456867)

[1.3 系统实现 13](#_Toc503456868)

[1.3.1系统实现 13](#_Toc503456869)

[1.3.2系统测试 13](#_Toc503456870)

[1.4 实验小结 22](#_Toc503456871)

[2 基于链式存储结构的线性表实现 23](#_Toc503456872)

[2.1 问题描述 23](#_Toc503456873)

[2.1.1 线性表基本运算定义 23](#_Toc503456874)

[2.2 系统设计 25](#_Toc503456875)

[2.2.1 系统总体设计 25](#_Toc503456876)

[2.2.2 有关常量和类型的定义 26](#_Toc503456877)

[2.2.3 算法设计 27](#_Toc503456878)

[2.3 系统实现 32](#_Toc503456879)

[2.3.1系统实现 32](#_Toc503456880)

[2.3.2系统测试 32](#_Toc503456881)

[2.4 实验小结 40](#_Toc503456882)

[3基于二叉链表的二叉树实现 42](#_Toc503456883)

[3.1 问题描述 42](#_Toc503456884)

[3.1.1 二叉树基本运算定义 42](#_Toc503456885)

[3.2 系统设计 46](#_Toc503456886)

[3.2.1 系统总体设计 46](#_Toc503456887)

[3.2.2 有关常量和类型的定义 48](#_Toc503456888)

[3.2.3 算法设计 49](#_Toc503456889)

[3.3 系统实现 58](#_Toc503456890)

[3.3.1系统实现 58](#_Toc503456891)

[3.3.2系统测试 59](#_Toc503456892)

[3.4 实验小结 71](#_Toc503456893)

[4基于邻接表的图实现 72](#_Toc503456894)

[4.1 问题描述 72](#_Toc503456895)

[4.1.1 图基本运算定义 72](#_Toc503456896)

[4.2 系统设计 74](#_Toc503456897)

[4.2.1 系统总体设计 74](#_Toc503456898)

[4.2.2 有关常量和类型的定义 75](#_Toc503456899)

[4.2.3 算法设计 77](#_Toc503456900)

[4.3 系统实现 83](#_Toc503456901)

[4.3.1系统实现 83](#_Toc503456902)

[4.3.2系统测试 84](#_Toc503456903)

[4.4 实验小结 96](#_Toc503456904)

[参考文献 97](#_Toc503456905)

[附录A 基于顺序存储结构线性表实现的源程序 98](#_Toc503456906)

[附录B基于链式存储结构线性表实现的源程序 113](#_Toc503456907)

[附录C基于二叉链表二叉树实现的源程序 129](#_Toc503456908)

[附录D 基于邻接表图实现的源程序 157](#_Toc503456909)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

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

## 1.1.1 线性表基本运算定义

依据最小完备性和常用性相结合的原则，以函数形式定义了12种基本运算，具体运算功能定义如下。

⑴初始化表：

函数名称:InitaList(L)；

初始条件:线性表L已存在；

操作结果:构造一个空的线性表。

此时应注意与表不存在之间的区分。

⑵销毁表：

函数名称:DestroyList(L)；

初始条件:线性表L已存在；

操作结果:销毁线性表L。

应是从物理上彻底释放。

⑶清空表：

函数名称:ClearList(L)；

初始条件:线性表L已存在；

操作结果:将L重置为空表。

仅是释放包含数据元素的结点

⑷判定空表：

函数名称:ListEmpty(L)；

初始条件:线性表L已存在；

操作结果:若L为空表则返回TRUE,否则返回FALSE。

⑸求表长：

函数名称:ListLength(L)；

初始条件:线性表已存在；

操作结果:返回L中数据元素的个数。

长度不包含头结点。

⑹获得元素：

函数名称:GetElem(L,i,e)；

初始条件:线性表已存在，1≤i≤ListLength(L)；

操作结果:用e返回L中第i个数据元素的值。

头结点不参与计数，计数从头结点一侧开始

⑺查找元素：

函数名称:LocateElem(L,e,compare())；

初始条件:线性表已存在；

操作结果:返回L中第1个与e满足关系compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为0。

⑻获得前驱：

函数名称:PriorElem(L,cur\_e,pre\_e)；

初始条件:线性表L已存在；

操作结果:若cur\_e是L的数据元素，且不是第一个，则用pre\_e返回它的前驱，否则操作失败。

需注意判断该元素是否合法。

⑼获得后继：

函数名称:NextElem(L,cur\_e,next\_e)；

初始条件:线性表L已存在；

操作结果:若cur\_e是L的数据元素，且不是最后一个，则用next\_e返回它的后继，否则操作失败。

需注意判断该元素是否合法。

⑽插入元素：

函数名称:ListInsert(L,i,e)；

初始条件:线性表L已存在，1≤i≤ListLength(L)+1；

操作结果:在L的第i个位置之前插入新的数据元素e。

操作为前插。即用新结点代替原来位置的结点。

⑾删除元素：

函数名称:ListDelete(L,i,e)；

初始条件:线性表L已存在且非空，1≤i≤ListLength(L)；

操作结果：删除L的第i个数据元素，用e返回其值。

头结点不参与计数。

⑿遍历表：

函数名称:ListTraverse(L,visit());

初始条件:线性表L已存在；

操作结果:依次对L的每个数据元素调用函数visit()。

## 1.2 系统设计

### 1.2.1 系统总体设计

采用简易菜单，总体结构为循环结构，内部为对线性表的13种操作和退出选项，除12种基本操作外，还增加了切换线性表的选项，其中退出与切换表的实现位于主函数中；外部为表管理模块，可在内存中操作多个表。使用switch语句支持操作的选择（输入数字进行选择），使用while语句支持主循环。程序主流程图如下

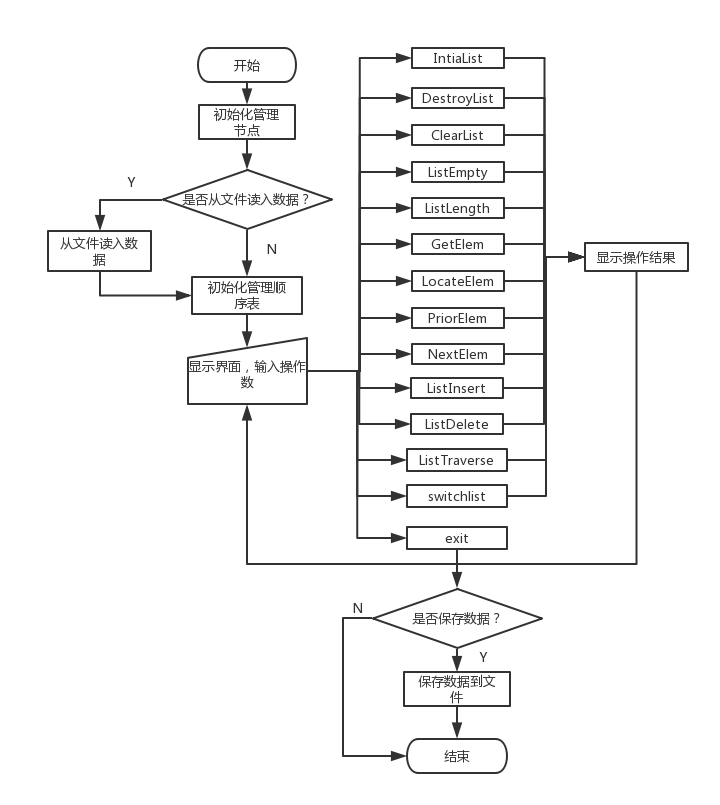


图1.2.1.1 系统主流程图

### 1.2.2 有关常量和类型的定义

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<malloc.h> //包含头文件

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10 //定义状态量

typedef int status;

typedef int ElemType; //数据元素类型定义

typedef struct SqList{ //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

typedef struct ListInfo{ //表信息管理结构定义

char name[20];

struct SqList\* elem;

}ListInfo;

typedef struct ManageList{ //表管理顺序表的信息结构定义

struct ListInfo\* elem;

int length;

int listsize;

int cur;

}ManageList;

### 1.2.3 算法设计

（1） 初始化表：status IntiaList(SqList \*L))

算法输入：指向顺序表的指针L。

算法输出：函数功能运行状态。

算法思想描述：

①：为顺序表动态分配内存。

②：判断是否成功分配，如果分配不成功，退出程序，退出参数OVERFLOW。

③：在L指向的结构中初始化此顺序表的表长与当前分配长度，返回OK。

时空复杂度分析：S（n）=O（1）；T（n）=O（1）；

（2） 摧毁表：status DestroyList(SqList \* L)

算法输入：指向顺序表的指针L。

算法输出：函数功能运行状态。

算法思想描述：

①：若顺序表不存在，返回ERROR。

②：释放L指向的结构中的指针elem指向的顺序表空间；

③：修改L指向的结构中的表信息，返回OK。

时空复杂度分析：S（n）=O(1)；T(n)=O(1)；

（3） 清空表：status ClearList(SqList \*L)

算法输入：指向顺序表的指针L。

算法输出：函数功能运行状态。

算法思想描述：

①：若顺序表不存在，返回ERROR。

②：释放原顺序表内存。

③：重新动态分配顺序表内存空间，若分配失败，退出程序，退出 参数OVERFLOW。

④：修改信息节点中表的长度，返回OK。

时空复杂度分析：S（n）=O（1）；T（n）=O（1）；

（4） 判定空表：status ListEmpty(SqList \*L)

算法输入：指向顺序表的指针L。

算法输出：函数功能运行状态。

算法思想描述：

①：判断L->length是否为0。

②：如果是，返回TRUE。如果不是，返回FALSE。

时空复杂度分析：S（n）=O（1）；T（n）=O（1）；

（5）获得元素：int ListLength(SqList \*L)

算法输入：指向顺序表的指针L。

算法输出：若顺序表存在，输出表长；若不存在，输出-1。

算法思想描述：

①:若顺序表不存在，返回-1。

②:返回L->length。

时空复杂度分析：S（n）=O（1） T（n）=O（1）；

（6）获得元素：status GetElem(SqList \*L,int i,ElemType \*e)

算法输入：指向顺序表的指针L，元素位序i，ElemType \*型指针e。

算法输出：函数功能运行状态。

算法思想描述：

①:判断所取元素的位置是否合法，若不合法，返回ERROR。

②: 用e指向的空间记录该元素的值，返回OK。

时空复杂度分析：S（n）=O（1） T（n）=O（1）；

（7）查找元素：status LocateElem(SqList \*L,ElemType e,int (\*compare)(ElemType,ElemType))

算法输入：指向顺序表的指针L，目标元素e，函数指针compare。

算法输出：顺序表中第1个与e满足关系compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回0。

①: 若顺序表不存在，返回ERROR。

②:遍历顺序表，查找第1个与e满足关系compare（）关系的数据元素。

③:若查找结果存在，返回该元素位序值。反之，返回0。

时空复杂度分析：S（n）=O（1） T（n）=O（n）；

（8）获得前驱：status PriorElem(SqList \*L,ElemType cur,ElemType \*pre\_e)

算法输入：指向顺序表的指针L，目标元素cur，ElemType \*型指针pre\_e。

算法输出：函数功能运行状态。

算法思想描述：

①：调用LocateElem（），获得元素cur的位序。

②：判断返回的位序是否合法，若合法，用pre\_e指向的空间记录该元素的前驱元素的值，返回OK；若不合法，返回ERROR。

时空复杂度分析：S（n）=O（1）T（n）=O（n）；

（9）获得后继：status NextElem(SqList \*L,ElemType cur,ElemType \*next\_e)

算法输入：指向顺序表的指针L，目标元素cur，ElemType \*型指针next\_e。

算法输出：函数功能运行状态。

算法思想描述：

①：调用LocateElem（），获得元素cur的位序。

②：判断返回的位序是否合法，若合法，用next\_e指向的空间记录该元素的前驱元素的值，返回OK；若不合法，返回ERROR。

算法时空复杂度分析：S（n）=O（1）T（n）=O（n）；

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

算法输入：指向顺序表的指针L，插入位置的位序i，插入元素的元素值e。

算法输出：函数功能运行状态。

算法思想描述：

①：若顺序表不存在或插入位置不合法，返回ERROR。

②：若分配空间不够可能造成溢出，重新给顺序表动态分配更大的内存空间，若分配失败，退出程序，退出参数OVERFLOW。

③：将位序为i的元素以及之后的所有元素依次后移一位。

④：表长度加1。

⑤：将位序为i的元素赋值为e,返回OK。

时空复杂度分析：S（n）=O（1）T（n）=O（n）;

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

算法输入：指向顺序表的指针L，删除位置的位序i，ElemType \*型指针e。

算法输出：函数功能运行状态。

算法思想描述：

①：若顺序表不存在或删除位置不合法，返回ERROR。

②：用e指向的空间记录删除位置元素的值

③：将该元素之后的所有元素依次前移一位。

④：表长度减1，返回OK。

时空复杂度分析：S（n）=O（1） T（n）=O（n）；

（12）遍历表：status ListTraverse(SqList \*L,void (\*visit)(ElemType))

算法输入：指向顺序表的指针L，函数指针visit。

算法输出：函数功能运行状态。

算法思想描述：

①：若顺序表不存在，返回ERROR。

②：依次对顺序表的每个数据元素调用函数visit()，输出元素值，返回OK。

时空复杂度分析：S（n）=O（1） T（n）=O（n）；

## 1.3 系统实现

**1.3.1系统实现**

实现语言为c语言，编译器为mingw32-gcc-4.9.2，操作系统为windows7。

代码中的主要功能除了12种基本操作之外，还包括表数据的文件读写部分和多表管理部分。文件读写采用单个二进制文件形式，保存所有信息，包括表管理相关单链表与结点信息。在进入主菜单之前可以选择是否读文件，退出程序之前可以选择是否保存表数据到文件。从文件读入数据后可追加表。使用一个包含当前表信息的结点来指示表，以避免判断表的存在性时的困难。

多表管理部分主要体现在main()函数中，采用元素为结构体的顺序表管理多个线性表，主要维护当前表的个数，当前表在管理表中的位置，管理表当前物理长度，各表的名称与指针。其中前三个内容保存在一个独立的结点中，此结点中某一指针指向此顺序表。摧毁当前表后，操作权默认切换至第一个表。从文件初始化表后，操作权默认属于当前第一个表，非文件初始化时，操作默认属于当前最后一个表。表的名称默认为表的序号。

**1.3.2系统测试**

本系统首先会给出是否从文件读入数据的选择，为方便后续测试，已经事先创建了5个顺序表以供测试。本测试选择从文件读入数据，同时测试文件读取功能。

表1.3.2.1从文件读入数据功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 数据文件 | 选项y | 恢复成功 | 与理论相符 |

根据提示按回车后，出现主选单

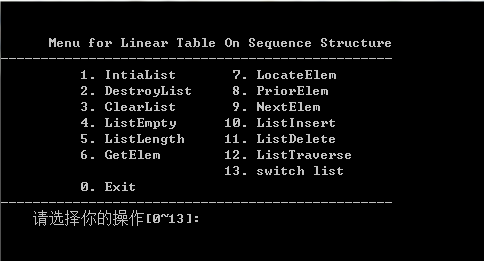


图1.3.2.2 操作界面截图

1、初始化表功能测试。

表1.3.2.3初始化表功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无特定用例 | 结构体指针 | 线性表创建成功 | 与理论相符 |

2、判断空表功能测试：

表1.3.2.4判断空表功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无表 | 输出：没有表可以判断 | 与理论相符 |
| 表1 | 非空表1 | 输出：线性表不为空 | 与理论相符 |
| 表6 | 空表 | 输出：线性表为空表 | 与理论相符 |

3、求表长功能测试：

表1.3.2.5求表长功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无表 | 输出：没有表可以进行计算 | 与理论相符 |
| 表6 | 空表 | 输出：线性表元素个数为0个 | 与理论相符 |
| 表4 | 长度为5的表4 | 输出：线性表元素个数为5个 | 与理论相符 |

4、摧毁表功能测试：

表1.3.2.6摧毁表功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无表 | 输出：没有表可以摧毁 | 与理论相符 |
| 表6 | 空表6 | 线性表销毁成功 | 与理论相符 |

5、清空表功能测试：

表1.3.2.7清空表功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无表 | 输出：没有表可以置空 | 符合理论结论 |
| 表6 | 空表6 | 输出：线性表已置空 | 符合理论结论 |
| 表1 | 长度为2的表1 | 输出：线性表已置空 | 符合理论结论 |

6、插入功能测试：

表1.3.2.8插入功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 无 | 无表 | 输出：没有表！ | 与理论相符 |
| 表6 | 空表6，第1个元素前插1 | 要求输入下标和元素，元素插入成功 | 与理论相符 |
| 表6 | 空表6，第5个元素前插5 | 插入失败 | 与理论相符 |
| 表6 | 空表6，第0个元素前插0 | 插入失败 | 与理论相符 |

7、删除功能测试：

表1.3.2.9删除功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 无 | 无表 | 删除失败 | 与理论相符 |
| 表6 | 空表6，删除第1个元素 | 删除失败 | 与理论相符 |
| 表4 | 表4删除第2个元素 | 删除成功，第2个元素值为2 | 与理论相符 |
| 表4 | 表4删除第0个元素 | 删除失败 | 与理论相符 |
| 表4 | 表4删除第100个元素 | 删除失败 | 与理论相符 |

8、定位功能测试：

表1.3.2.10定位功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 无 | 无表 | 没有表 | 与理论相符 |
| 表6 | 空表6，查找8 | 数据元素不存在 | 与理论相符 |
| 表3 | 长度为4的表3，查找2 | 元素位序为2 | 与理论相符 |
| 表3 | 长度为4的表3，查找100 | 数据元素不存在 | 与理论相符 |

9、获得元素值功能测试：

表1.3.2.11获取元素功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 无 | 无表 | 输出：没有表 | 与理论相符 |
| 表6 | 空表6，获得第1个元素 | 该数据元素不存在 | 与理论相符 |
| 表2 | 表2，获得第3个元素 | 第3个元素的值为3 | 与理论相符 |
| 表2 | 表2，获得第100个元素 | 该数据元素不存在 | 与理论相符 |

10、获得前驱功能测试：

表1.3.2.12获得前驱功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 无 | 无表 | 输出：没有表 | 与理论相符 |
| 表6 | 空表6，获得1的前驱 | 操作失败 | 与理论相符 |
| 表4 | 表4，获得4的前驱 | 元素的前驱元素为3 | 与理论相符 |
| 表4 | 表4，获得100的前驱 | 操作失败 | 与理论相符 |

11、获得后继功能测试：

表1.3.2.13获得后继功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 无 | 无表 | 输出：没有表 | 与理论相符 |
| 表6 | 空表6，获得1的后继 | 操作失败 | 与理论相符 |
| 表4 | 表4，获得4的后继 | 元素的后继元素为5 | 与理论相符 |
| 表4 | 表4，获得5的后继 | 操作失败 | 与理论相符 |

12、遍历功能测试：

表1.3.2.14遍历功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无 | 输出：没有表 | 与理论相符 |
| 表3 | 表3 | 输出1 2 3 4 | 与理论相符 |

13、表控制切换功能测试：

如下图，选择功能13，即可显示当前内存中的所有表，以及其名称与长度。可以选择一个表进行操作。

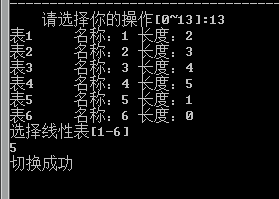


图1.3.2.15切换功能演示图

14、数据保存功能测试：

表1.3.2.16存储功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 结构中所有数据 | 结构中所有数据，删除一个表，保存后再读取 | 保存成功，可供下次正确读取 | 与理论相符 |

综上：该系统实现了全部14个功能，并且对边界条件的反应程度良好，异常判断较齐全，符合实验要求。

## 1.4 实验小结

本次实验核心的12个操作难度不大，但是如何构造表管理模块以及具体的管理方式需要仔细认真地研究和界定，将其设计出来也花了不少的时间。但是起初有很多方面的问题没有考虑到，表管理比较粗糙，后来经过好几遍的改进逐渐完善起来，虽然仍然不是非常完备，但是能保证基本的管理需求。另外，文件操作部分还有较大的改进空间，可以改进储存结构来减小文件体积，界面方面也不是很美观。这一次上机进一步锻炼了我的动手能力，也让我的查错与问题界定能力有了提升，更加具有系统观。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

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

### 2.1.1 线性表基本运算定义

依据最小完备性和常用性相结合的原则，以函数形式定义了12种基本运算，具体运算功能定义如下。

⑴初始化表：

函数名称:InitaList(L)；

初始条件:线性表L已存在；

操作结果:构造一个空的线性表。

此时应注意与表不存在之间的区分。

⑵销毁表：

函数名称：DestroyList(L)；

初始条件：线性表L已存在；

操作结果:销毁线性表L。

应是从物理上彻底释放。

⑶清空表：

函数名称：ClearList(L)；

初始条件：线性表L已存在；

操作结果：将L重置为空表。

仅是释放包含数据元素的结点

⑷判定空表：

函数名称：ListEmpty(L)；

初始条件：线性表L已存在；

操作结果：若L为空表则返回TRUE,否则返回FALSE。

⑸求表长：

函数名称：ListLength(L)；

初始条件：线性表已存在；

操作结果：返回L中数据元素的个数。

长度不包含头结点。

⑹获得元素：

函数名称：GetElem(L,i,e)；

初始条件：线性表已存在，1≤i≤ListLength(L)；

操作结果：用e返回L中第i个数据元素的值。

头结点不参与计数，计数从头结点一侧开始

⑺查找元素：

函数名称：LocateElem(L,e,compare())；

初始条件：线性表已存在；

操作结果：返回L中第1个与e满足关系compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为0。

⑻获得前驱：

函数名称：PriorElem(L,cur\_e,pre\_e)；

初始条件：线性表L已存在；

操作结果：若cur\_e是L的数据元素，且不是第一个，则用pre\_e返回它的前驱，否则操作失败。

需注意判断该元素是否合法。

⑼获得后继：

函数名称：NextElem(L,cur\_e,next\_e)；

初始条件：线性表L已存在；

操作结果：若cur\_e是L的数据元素，且不是最后一个，则用next\_e返回它的后继，否则操作失败。

需注意判断该元素是否合法。

⑽插入元素：

函数名称：ListInsert(L,i,e)；

初始条件：线性表L已存在，1≤i≤ListLength(L)+1；

操作结果：在L的第i个位置之前插入新的数据元素e。

操作为前插。即用新结点代替原来位置的结点。

⑾删除元素：

函数名称：ListDelete(L,i,e)；

初始条件：线性表L已存在且非空，1≤i≤ListLength(L)；

操作结果：删除L的第i个数据元素，用e返回其值。

头结点不参与计数。

⑿遍历表：

函数名称：ListTraverse(L,visit())；

初始条件：线性表L已存在；

操作结果：依次对L的每个数据元素调用函数visit()。

## 2.2 系统设计

### 2.2.1 系统总体设计

采用简易菜单，总体结构为循环结构，内部为对线性表的13种操作和退出选项，除12种基本操作外，还增加了切换线性表的选项，其中退出与切换表的实现位于主函数中；外部为表管理模块，可在内存中操作多个表。使用switch语句支持操作的选择（输入数字进行选择），使用while语句支持主循环。程序主流程图如下：

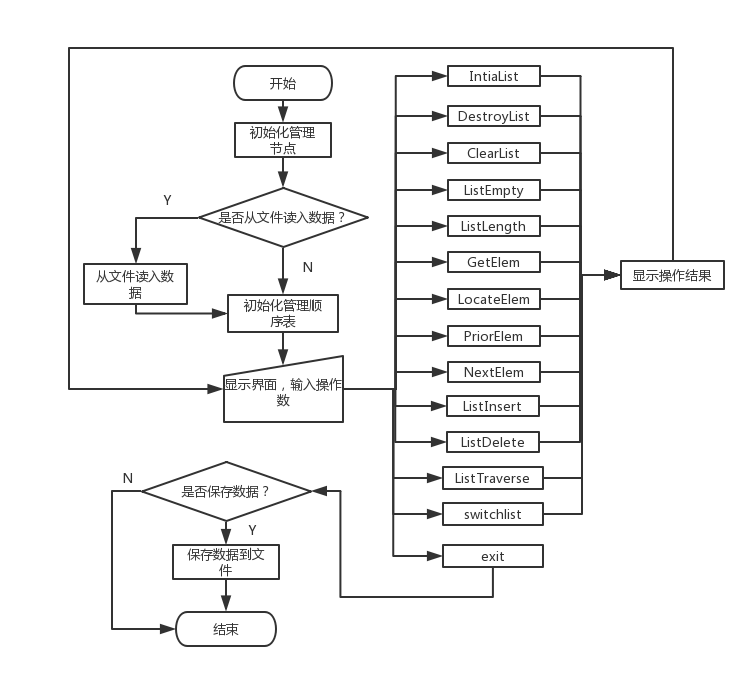


图2.2.1.1 系统主流程图

### 2.2.2 有关常量和类型的定义

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<malloc.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef int status;

typedef int ElemType;

typedef struct LNode{

ElemType data;

struct LNode\* next;

}LNode, \*LNode\_ptr, \*Position;

typedef struct LinkList{

struct LNode\* head;

//struct LNode\* tail;

int length;

}LinkList;

typedef struct ListInfo{

char name[20];

struct LinkList\* elem;

}ListInfo;

typedef struct ManageList{

struct ListInfo\* elem;

int length;

int listsize;

int cur;

}ManageList;

### 2.2.3 算法设计

（1） 初始化表：status IntiaList(LinkList \*L)

算法输入：指向单链表的指针L。

算法输出：函数功能运行状态。

算法思想描述：

①：为单链表动态分配内存。

②：判断是否成功分配，如果分配不成功，退出程序，退出参数OVERFLOW。

③：在L指向的结构中初始化此单链表的表长。

④初始化头结点中的数据域与指针域。返回OK

时空复杂度分析：S（n）=O（1）；T（n）=O（1）；

（2） 摧毁表：status DestroyList(LinkList \*L)

算法输入：指向单链表的指针L。

算法输出：函数功能运行状态。

算法思想描述：

①：若单链表不存在，返回ERROR。

②：使用循环将单链表从头结点开始依次销毁；

③：修改L指向的结构中的表信息，返回OK。

时空复杂度分析：S（n）=O(1)；T(n)=O(n)；

（3） 清空表：status ClearList(LinkList \*L)

算法输入：指向单链表的指针L。

算法输出：函数功能运行状态。

算法思想描述：

①：若单链表不存在，返回ERROR。

②：若单链表长度为0，返回OK。

③：若单链表长度不为0，销毁头结点之后的所有结点。

④：修改信息节点中表的长度和头结点的next指针域，返回OK。

时空复杂度分析：S（n）=O（1）；T（n）=O（n）；

（4） 判定空表：status ListEmpty(LinkList \*L)

算法输入：指向单链表的指针L。

算法输出：函数功能运行状态。

算法思想描述：

①：若单链表不存在，返回-1。

②：判断L->length是否为0。

③：如果是，返回TRUE。如果不是，返回FALSE。

时空复杂度分析：S（n）=O（1）；T（n）=O（1）；

（5）获得元素：int ListLength(LinkList \*L)

算法输入：指向单链表的指针L。

算法输出：若单链表存在，输出表长；若不存在，输出-1。

算法思想描述：

①:若单链表不存在，返回-1。

②:返回L->length。

时空复杂度分析：S（n）=O（1） T（n）=O（1）；

（6）获得元素：status GetElem(LinkList \*L,int i,ElemType \*e)

算法输入：指向单链表的指针L，元素位序i，ElemType \*型指针e。

算法输出：函数功能运行状态。

算法思想描述：

①:判断所取元素的位置是否合法，若不合法，返回ERROR。

②: 使用循环找到位序为i的元素，用e指向的空间记录该元素的值，返回OK。

时空复杂度分析：S（n）=O（1） T（n）=O（n）；

（7）查找元素：status LocateElem(LinkList \*L,ElemType e,int (\*compare)(ElemType,ElemType))

算法输入：指向单链表的指针L，目标元素e，函数指针compare。

算法输出：单链表中第1个与e满足关系compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回0。

①: 若单链表不存在，返回ERROR。

②:遍历单链表，查找第1个与e满足关系compare（）关系的数据元素。

③:若查找结果存在，返回该元素位序值。反之，返回0。

时空复杂度分析：S（n）=O（1） T（n）=O（n）；

（8）获得前驱：status PriorElem(LinkList \*L,ElemType cur\_e,ElemType \*pre\_e)

算法输入：指向单链表的指针L，目标元素cur\_e，ElemType \*型指针pre\_e。

算法输出：函数功能运行状态。

算法思想描述：

①：调用LocateElem（），获得元素cur\_e的位序。

②：判断返回的位序是否合法，若合法，扫描单链表，用pre\_e指向的空间记录该元素的前驱元素的值，返回OK；若不合法，返回ERROR。

时空复杂度分析：S（n）=O（1）T（n）=O（n）；

（9）获得后继：status NextElem(LinkList \*L,ElemType cur\_e,ElemType \*next\_e)

算法输入：指向单链表的指针L，目标元素cur\_e，ElemType \*型指针next\_e。

算法输出：函数功能运行状态。

算法思想描述：

①：调用LocateElem（），获得元素cur\_e的位序。

②：判断返回的位序是否合法，若合法，扫描单链表,用next\_e指向的空间记录该元素的后继元素的值，返回OK；若不合法，返回ERROR。

算法时空复杂度分析：S（n）=O（1）T（n）=O（n）；

（10） 插入元素：status ListInsert(LinkList \*L,int i,ElemType e)

算法输入：指向单链表的指针L，插入位置的位序，插入元素的元素值。

算法输出：函数功能运行状态。

算法思想描述：

①：若单链表不存在或插入位置不合法，返回ERROR。

②：使用指针移动到目标位置，建立新结点。

③：将此结点的数据域赋值为e。

④：使第i-1个结点的next指针指向新结点，使新结点的next指针指向原第i个结点。

⑤：表长加1,返回OK。

时空复杂度分析：S（n）=O（1）T（n）=O（n）;

（11） 删除元素：status ListDelete(LinkList \*L,int i,ElemType \*e)

算法输入：指向单链表的指针L，删除位置的位序i，ElemType \*型指针e。

算法输出：函数功能运行状态。

算法思想描述：

①：若单链表不存在或删除位置不合法，返回ERROR。

②：使用指针移动到目标位置，用e指向的空间记录删除位置元素的值。

③：使第i-1个结点的next指针指向第i+1个结点，释放目标结点。

④：表长度减1，返回OK。

时空复杂度分析：S（n）=O（1） T（n）=O（n）；

（12）遍历表：status ListTraverse(LinkList \*L,void (\*visit)(ElemType))

算法输入：指向单链表的指针L，函数指针visit。

算法输出：函数功能运行状态。

算法思想描述：

①：若单链表不存在，返回ERROR。

②：依次对单链表的每个数据元素调用函数visit()，输出元素值，返回OK。

时空复杂度分析：S（n）=O（1） T（n）=O（n）；

## 2.3 系统实现

**2.3.1系统实现**

实现语言为c语言，编译器为mingw32-gcc-4.9.2，操作系统为windows7。

代码中的主要功能除了12种基本操作之外，还包括表数据的文件读写部分和多表管理部分。文件读写采用单个二进制文件形式，保存所有信息，包括表管理相关单链表与结点信息。在进入主菜单之前可以选择是否读文件，退出程序之前可以选择是否保存表数据到文件。从文件读入数据后可追加表。单链表采用带头结点的单链表，同时使用一个包含当前表信息的结点来指示表，以避免判断表的存在性时的困难。

多表管理部分主要体现在main()函数中，采用元素为结构体的顺序表管理多个线性表，主要维护当前表的个数，当前表在管理表中的位置，管理表当前物理长度，各表的名称与指针。其中前三个内容保存在一个独立的结点中，此结点中某一指针指向此顺序表。摧毁当前表后，操作权默认切换至第一个表。从文件初始化表后，操作权默认属于当前第一个表，非文件初始化时，操作默认属于当前最后一个表。表的名称默认为表的序号。

**2.3.2系统测试**

本系统首先会给出是否从文件读入数据的选择，为方便后续测试，已经事先创建了7个单链表以供测试。本测试选择从文件读入数据，同时测试文件读取功能。

表2.3.2.1从文件读入数据功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 数据文件 | 选项y | 恢复成功 | 与理论相符 |

根据提示按回车后，出现主选单

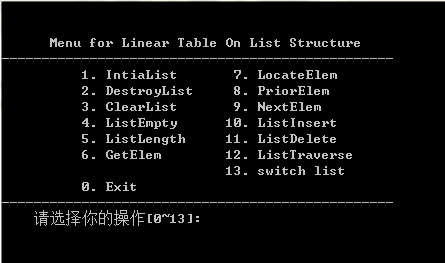


图2.3.2.2 操作界面截图

1、初始化表功能测试。

表2.3.2.3初始化表功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无特定用例 | 结构体指针 | 线性表创建成功 | 与理论相符 |

2、判断空表功能测试：

表2.3.2.4判断空表功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无表 | 输出：没有表可以判断 | 与理论相符 |
| 表3 | 非空表3 | 输出：线性表不为空 | 与理论相符 |
| 表7 | 空表7 | 输出：线性表为空表 | 与理论相符 |

3、求表长功能测试：

表2.3.2.5求表长功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无表 | 输出：没有表可以进行计算 | 与理论相符 |
| 表7 | 空表7 | 输出：线性表元素个数为0个 | 与理论相符 |
| 表1 | 长度为4的表1 | 输出：线性表元素个数为4个 | 与理论相符 |

4、摧毁表功能测试：

表2.3.2.6摧毁表功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无表 | 输出：没有表可以摧毁 | 与理论相符 |
| 表7 | 空表7 | 线性表销毁成功 | 与理论相符 |

5、清空表功能测试：

表2.3.2.7清空表功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无表 | 输出：没有表可以置空 | 符合理论结论 |
| 表7 | 空表7 | 输出：线性表已置空 | 符合理论结论 |
| 表2 | 长度为4的表2 | 输出：线性表已置空 | 符合理论结论 |

6、插入功能测试：

表2.3.2.8插入功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 无 | 无表 | 输出：没有表！ | 与理论相符 |
| 表7 | 空表7，第1个元素前插200 | 要求输入下标和元素，元素插入成功 | 与理论相符 |
| 表6 | 空表6，第5个元素前插300 | 插入失败 | 与理论相符 |
| 表7 | 空表7，第0个元素前插0 | 插入失败 | 与理论相符 |

7、删除功能测试：

表2.3.2.9删除功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 无 | 无表 | 删除失败 | 与理论相符 |
| 表7 | 空表7，删除第10个元素 | 删除失败 | 与理论相符 |
| 表4 | 长度为2的表4删除第1个元素 | 删除成功，第1个元素值为1 | 与理论相符 |
| 表4 | 长度为2的表4删除第0个元素 | 删除失败 | 与理论相符 |
| 表3 | 长度为3的表3删除第4个元素 | 删除失败 | 与理论相符 |

8、定位功能测试：

表2.3.2.10定位功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 无 | 无表 | 没有表 | 与理论相符 |
| 表7 | 空表7，查找1 | 数据元素不存在 | 与理论相符 |
| 表3 | 长度为3的表3，查找3 | 元素位序为3 | 与理论相符 |
| 表3 | 长度为3的表3，查找-1 | 数据元素不存在 | 与理论相符 |

9、获得元素值功能测试：

表2.3.2.11获取元素功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 无 | 无表 | 输出：没有表 | 与理论相符 |
| 表7 | 空表7，获得第1个元素 | 该数据元素不存在 | 与理论相符 |
| 表2 | 长度为4的表2，获得第2个元素 | 第2个元素的值为2 | 与理论相符 |
| 表2 | 表2，获得第5个元素 | 该数据元素不存在 | 与理论相符 |

10、获得前驱功能测试：

表2.3.2.12获得前驱功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 无 | 无表 | 输出：没有表 | 与理论相符 |
| 表7 | 空表7，获得1的前驱 | 操作失败 | 与理论相符 |
| 表4 | 长度为2的表4，获得2的前驱 | 元素的前驱元素为1 | 与理论相符 |
| 表4 | 长度为2的表4，获得4的前驱 | 操作失败 | 与理论相符 |

11、获得后继功能测试：

表2.3.2.13获得后继功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 无 | 无表 | 输出：没有表 | 与理论相符 |
| 表7 | 空表7，获得1的后继 | 操作失败 | 与理论相符 |
| 表1 | 长度为4的表1，获得10的后继 | 元素的后继元素为2 | 与理论相符 |
| 表1 | 长度为4的表1，获得5的后继 | 操作失败 | 与理论相符 |

12、遍历功能测试：

表2.3.2.14遍历功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无 | 输出：没有表 | 与理论相符 |
| 表1 | 表1 | 输出10 2 3 5 | 与理论相符 |

13、表控制切换功能测试：

如下图，选择功能13，即可显示当前内存中的所有表，以及其名称与长度。可以选择一个表进行操作。



图2.3.2.15切换功能演示图

14、数据保存功能测试：

表2.3.2.16存储功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 结构中所有数据 | 结构中所有数据，删除一个表，保存后再读取 | 保存成功，可供下次正确读取 | 与理论相符 |

综上：该系统实现了全部14个功能，并且对边界条件的反应程度良好，异常判断较齐全，符合实验要求。

## 2.4 实验小结

本次实验总体相对难度不大，与第一次实验有许多相同之处。在编写算法的过程中，我体会到了虽然单链表是动态的，插入与删除比线性表方便，但是由于无法直接找到元素，在定位与销毁方面却不太方便。相比讨论理论的课堂，在实验中更能体会到这一点。本次的表管理体系沿用了第一次实验的，存储文件有效数据密度的问题仍然没有得到解决，但是这种机制在逻辑上是简单的，但是我想改进后会更好，因为如果数据规模较大，现有方式的缺点就会更加明显。

# 3基于二叉链表的二叉树实现

## 3.1 问题描述

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

### 3.1.1 二叉树基本运算定义

依据最小完备性和常用性相结合的原则，以函数形式定义了20种基本运算，具体运算功能定义如下。

⑴初始化二叉树：

函数名称: InitBiTree(T)；

初始条件: 二叉树T不存在；

操作结果: 构造空二叉树T。

此时应注意空与不存在之间的区分。

⑵销毁二叉树：

函数名称：DestroyBiTree(T)；

初始条件：二叉树T已存在；

操作结果: 销毁二叉树T。

应是从物理上彻底释放。

⑶创建二叉树：

函数名称：CreateBiTree(T,definition)；；

初始条件：definition 给出二叉树T的定义；

操作结果：按definition构造二叉树T。

⑷清空二叉树：

函数名称：ClearBiTree (T)；

初始条件：二叉树T存在；

操作结果：将二叉树T清空。

⑸判定空二叉树：

函数名称：BiTreeEmpty(T)；

初始条件：二叉树T存在；

操作结果：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：BiTreeDepth(T)；

初始条件：二叉树T存在；

操作结果：返回T的深度。

⑺获得根结点：

函数名称：Root(T)；

初始条件：二叉树T已存在；

操作结果：返回T的根。

⑻获得结点：

函数名称：Value(T,e)；

初始条件：二叉树T已存在，e是T中的某个结点；

操作结果：返回e的值。

⑼结点赋值：

函数名称：Assign(T,&e,value)；

初始条件：二叉树T已存在，e是T中的某个结点；

操作结果：结点e赋值为value。

⑽获得双亲结点：

函数名称：Parent(T,e)；

初始条件：二叉树T已存在，e是T中的某个结点；

操作结果：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：LeftChild(T,e)；

初始条件：二叉树T存在，e是T中某个节点；

操作结果：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：RightChild(T,e)；

初始条件：二叉树T已存在，e是T中某个结点；

操作结果：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：LeftSibling(T,e)；

初始条件：二叉树T存在，e是T中某个结点；

操作结果：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：RightSibling(T,e)；

初始条件：二叉树T已存在，e是T中某个结点；

操作结果：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：InsertChild(T,p,LR,c)；

初始条件：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：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：DeleteChild(T.p.LR)；

初始条件：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：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：PreOrderTraverse(T,Visit())；

初始条件：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：InOrderTraverse(T,Visit))；

初始条件：二叉树T存在，Visit是对结点操作的应用函数；

操作结果：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：PostOrderTraverse(T,Visit))；

初始条件：二叉树T存在，Visit是对结点操作的应用函数；

操作结果：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：LevelOrderTraverse(T,Visit))；

初始条件：二叉树T存在，Visit是对结点操作的应用函数；

操作结果：层序遍历T，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

## 3.2 系统设计

### 3.2.1 系统总体设计

采用简易菜单，总体结构为循环结构，内部为对二叉树的20种操作和退出选项，除20种基本操作外，还增加了切换二叉树的选项，其中退出与切换二叉树，以及一些相关量的维护的实现位于主函数中；外部为多树管理模块，可在内存中操作多个二叉树。使用switch语句支持操作的选择（输入数字进行选择），使用while语句支持主循环。程序主流程图如下：

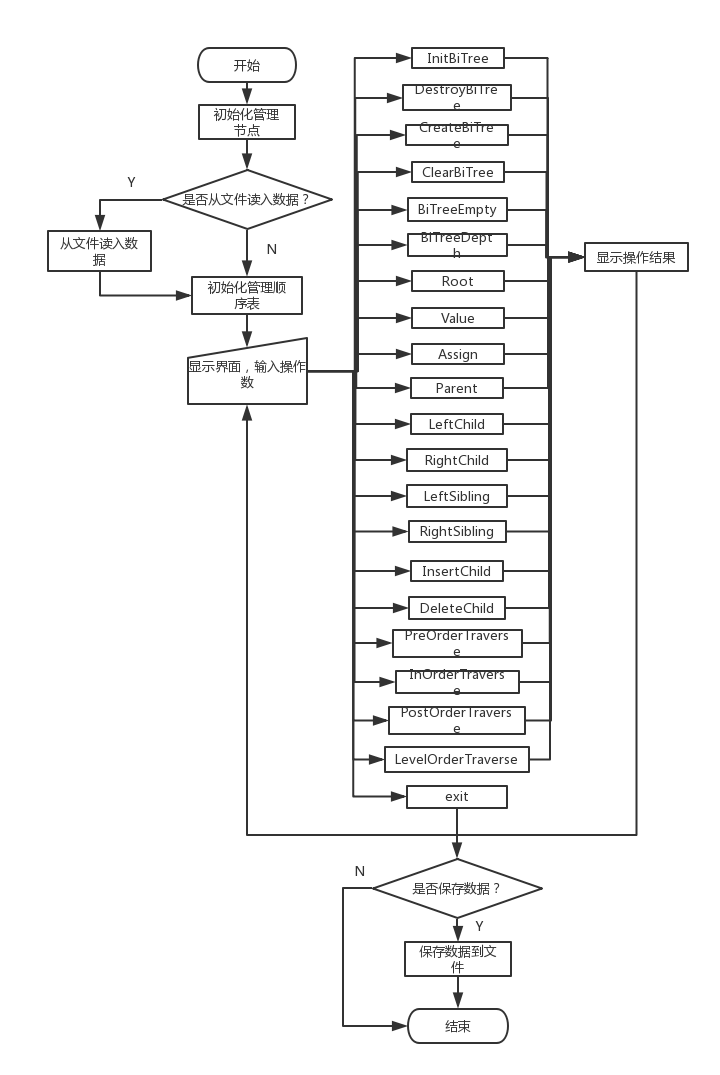


图3.2.1.1 系统主流程图

### 3.2.2 有关常量和类型的定义

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<malloc.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

#define MAX\_LENGTH 512

typedef int status;

typedef int TElemType;

typedef struct BiTNode{

TElemType index;

TElemType grade;

struct BiTNode \*lchild,\*rchild;

}BiTNode, \*Tree, \*TreeNode, \*BiTree;

typedef struct Bitreeinfo{

int ver\_num;

int depth;

struct BiTNode \*root;

}Bitreeinfo, \*Treeinfo;

typedef struct ListInfo{

char name[20];

struct Bitreeinfo\* elem;

}ListInfo;

typedef struct ManageList{

struct ListInfo\* elem;

int length;

int listsize;

int cur;

}ManageList;

### 3.2.3 算法设计

（1）初始化二叉树：status InitBiTree(Tree\* T)

算法输入：指向指向二叉树根节点的指针的指针T。

算法输出：函数功能运行状态。

算法思想描述：

①：将T指向的指针赋值为NULL。

②：返回OK。

时空复杂度分析：S（n）=O（1）；T（n）=O（1）；

（2）销毁二叉树：status DestroyBiTree(Tree T)

算法输入：指向二叉树根节点的指针T。

算法输出：函数功能运行状态。

算法思想描述：

本算法使用递归方法实现

①：若二叉树为空，返回ERROR。

②：若二叉树不为空：若根节点左子树不为空，销毁左子树，指针lchild置为NULL；若根节点右子树不为空，销毁右子树，指针rchild置为NULL；若根节点存在，释放节点内存，根节点指针置为NULL。返回OK；

时空复杂度分析：S（n）=O(n)；T(n)=O(n)；

（3）创建二叉树：TreeNode CreateBiTree(Treeinfo L)

算法输入：指向二叉树信息节点的指针L。

算法输出：当前二叉树的根节点指针。

算法思想描述：

本算法使用递归方法实现

①：从输入流获取1个字符，若此字符为#或\n，返回NULL。

②：若此字符不服从以上情况，从输入流获取5个字符，组成6个字符。

③：创建新结点，若创建失败，退出程序，退出参数OVERFLOW。

④：取前3个字符转换为整数，赋值给新结点的index；取后3个字符转换为整数，赋值给新结点的grade。

⑤：信息节点中节点个数加1。

⑥：创建左子树。（递归调用）。

⑦：创建右子树。（递归调用）。

时空复杂度分析：S（n）=O（n）；T（n）=O（n）；

（4）清空二叉树：status ClearBiTree(Tree T)

算法输入：指向二叉树根节点的指针T。

算法输出：函数功能运行状态。

算法思想描述：

本算法使用递归方法实现

①：若二叉树为空，返回ERROR。

②：若二叉树不为空：若根节点左子树不为空，清空左子树，指针lchild置为NULL；若根节点右子树不为空，清空右子树，指针rchild置为NULL；若根节点存在，释放节点内存，根节点指针置为NULL。返回OK；

时空复杂度分析：S（n）=O(n)；T(n)=O(n)；

（5）判定空二叉树：status BiTreeEmpty(Tree T)

算法输入：指向二叉树根节点的指针T。

算法输出：函数功能运行状态。

算法思想描述：

①:若T为NULL，返回TRUE。

②:若T不为NULL，返回FALSE。

时空复杂度分析：S（n）=O（1） T（n）=O（1）；

（6）求二叉树深度：status BiTreeDepth(Tree T)

算法输入：指向二叉树根节点的指针T。

算法输出：函数功能运行状态。

算法思想描述：

①:若二叉树为空，返回0。

②:求左子树深度，赋值给NLeft（递归调用）。

③：求右子树深度，赋值给NRight（递归调用）。

④：返回NLeft和NRight中的较大值。

时空复杂度分析：S（n）=O（n） T（n）=O（n）；

（7）获得根结点：TreeNode Root(Tree T)

算法输入：指向二叉树根节点的指针T。

算法输出：指向根节点的指针。

①: 若二叉树不存在，返回NULL。

②:若二叉树存在，返回T的值。

时空复杂度分析：S（n）=O（1） T（n）=O（1）；

（8）获得结点：TElemType Value(Tree T,int e)

算法输入：指向二叉树根节点的指针T，目标节点标识e。

算法输出：目标节点值。

算法思想描述：

①：如果二叉树为空，返回ERROR。

②：按非递归中序遍历遍历二叉树，若节点标识等于e，返回该节点值。

③：若未找到相应节点，返回INFEASIBLE。

时空复杂度分析：S（n）=O（n）T（n）=O（n）；

（9）结点赋值：status Assign(Tree T,int e,int value)

算法输入：指向二叉树根节点的指针T，目标节点标识e，值value。

算法输出：函数功能运行状态。

算法思想描述：

①：如果二叉树为空，返回ERROR。

②：按非递归中序遍历遍历二叉树，若节点标识等于e，将该节点值赋为value，返回OK。

③：若未找到相应节点，返回INFEASIBLE。

算法时空复杂度分析：S（n）=O（n）T（n）=O（n）；

（10）获得双亲结点：TreeNode Parent(Tree T,int e)

算法输入：指向二叉树根节点的指针T，目标节点标识e。

算法输出：指向双亲节点的指针。

算法思想描述：

①：如果二叉树为空，返回ERROR。

②：按非递归后序遍历遍历二叉树，若当前节点标识等于e，返回指针栈中的当前栈顶元素（即双亲节点指针）。

③：若未找到相应节点，返回NULL。

时空复杂度分析：S（n）=O（n）T（n）=O（n）;

（11）获得左孩子结点：TreeNode LeftChild(Tree T,int e)

算法输入：指向二叉树根节点的指针T，目标节点标识e。

算法输出：指向目标节点左孩子节点的指针。

算法思想描述：

①：调用获得双亲函数，获得该标识的节点的双亲节点的指针。

②：按非递归中序遍历遍历二叉树，若当前节点标识等于e，返回其左孩子指针。

③：若未找到相应节点，返回NULL。

时空复杂度分析：S（n）=O（n） T（n）=O（n）；

（12）获得右孩子结点：TreeNode RightChild(Tree T,int e)

算法输入：指向二叉树根节点的指针T，目标节点标识e。

算法输出：指向目标节点右孩子节点的指针。

算法思想描述：

①：如果二叉树为空，返回ERROR。

②：按非递归中序遍历遍历二叉树，若当前节点标识等于e，返回其右孩子指针。

③：若未找到相应节点，返回NULL。

时空复杂度分析：S（n）=O（n） T（n）=O（n）；

（13）获得左兄弟结点：TreeNode LeftSibling(Tree T,int e)

算法输入：指向二叉树根节点的指针T，目标节点标识e。

算法输出：指向目标节点左兄弟节点的指针。

算法思想描述：

①：调用Parent，获取对应标识的节点的双亲节点指针。

②：遍历二叉树，获取对应标识的节点指针。

③：若双亲节点存在：

若目标节点是双亲节点的左孩子或双亲节点的左孩子为空，返回ERROR。若不满足上述两种情况，返回双亲节点的左孩子指针。

④：若双亲节点不存在，返回ERROR

时空复杂度分析：S（n）=O（n） T（n）=O（n）；

（14）获得右兄弟结点：TreeNode RightSibling(Tree T,int e)

算法输入：指向二叉树根节点的指针T，目标节点标识e。

算法输出：指向目标节点右兄弟节点的指针。

算法思想描述：

①：调用Parent，获取对应标识的节点的双亲节点指针。

②：遍历二叉树，获取对应标识的节点指针。

③：若双亲节点存在：

若目标节点是双亲节点的右孩子或双亲节点的右孩子为空，返回ERROR。若不满足上述两种情况，返回双亲节点的右孩子指针。

④：若双亲节点不存在，返回ERROR

时空复杂度分析：S（n）=O（n） T（n）=O（n）；

（15）插入子树：status InsertChild(Tree T,int index,int LR,BiTree c)

算法输入：指向二叉树根节点的指针T，目标节点标识index，指示左右的整数LR，被插入的树的根节点指针c。

算法输出：函数功能运行状态。

算法思想描述：

①：如果二叉树为空，返回ERROR。

②：拷贝树c至树insertion。

③：若insertion的右孩子不为空，销毁其右子树。

④：调用Search()找到index对应的节点，获取其指针。

⑤：如果未找到，返回ERROR。

⑥：若找到

如果LR为0，将insertion插入为该节点左子树，原左子树成为insertion的右子树。如果LR为1，将insertion插入为该节点右子树，原右子树成为insertion的右子树。返回OK。

时空复杂度分析：S（n）=O（n） T（n）=O（n）；

（16）删除子树：status DeleteChild(Tree T,int e,int LR)

算法输入：指向二叉树根节点的指针T，目标节点标识e，指示左右的整数LR。

算法输出：函数功能运行状态。

算法思想描述：

①：如果二叉树为空，返回ERROR。

②：调用Search()找到e对应的节点，获取其指针。

③：如果未找到，返回ERROR。

④：若找到

如果LR为0，销毁左子树。如果LR为1，销毁右子树。返回OK。

时空复杂度分析：S（n）=O（n） T（n）=O（n）；

（17）前序遍历：status PreOrderTraverse(Tree T,void (\*visit)(TElemType))

算法输入：指向二叉树根节点的指针T，函数指针visit。

算法输出：函数功能运行状态。

算法思想描述：

①：如果二叉树为空，返回ERROR。

②：初始化指针栈。

③：若T不为NULL或栈顶不为0：

④：若T不为NULL：

若栈满，退出程序，退出参数OVERFLOW。

访问T指向的节点。

T入栈。

将T赋值为其左孩子指针。

转④。

⑤：栈顶元素出栈，赋值给T。

⑥：若双亲节点不存在，返回ERROR

⑦：将T赋值为其右孩子指针。转③

⑧：返回OK。

时空复杂度分析：S（n）=O（n） T（n）=O（n）；

（18）中序遍历：status InOrderTraverse(Tree T,void (\*visit)(TElemType))

算法输入：指向二叉树根节点的指针T，函数指针visit。

算法输出：函数功能运行状态。

算法思想描述：

①：如果二叉树为空，返回ERROR。

②：初始化指针栈。

③：若T不为NULL：

若栈满，退出程序，退出参数OVERFLOW。

T入栈。

将T赋值为其左孩子指针。

④：若栈顶不为0。

栈顶元素出栈，赋值给T。

访问T指向的节点。

将T赋值为其右孩子指针。

⑤：若T不为NULL或栈顶不为0，转③。

⑥：若T为NULL且栈顶为0，返回OK。

时空复杂度分析：S（n）=O（n） T（n）=O（n）；

（19）后序遍历：status PostOrderTraverse(Tree T,void (\*visit)(TElemType))

算法输入：指向二叉树根节点的指针T，函数指针visit。

算法输出：函数功能运行状态。

算法思想描述：

①：如果二叉树为空，返回ERROR。

②：初始化指针栈和标志栈。

③：若T不为NULL或栈顶不为0：

④：若T不为NULL：

若栈满，退出程序，退出参数OVERFLOW。

T入栈。

标志栈中相应位置置为0。

将T赋值为其左孩子指针。

转④。

⑤：如果标志栈栈顶元素的值为0：

将T赋值为指针栈栈顶元素值。

标志栈栈顶元素的值置为1。

将T赋值为其右孩子指针。

⑥：如果标志栈栈顶元素的值为1：

⑦：如果标志栈栈顶元素的值为1：

指针栈栈顶元素出栈，并赋值给T。

访问T指向的节点。

转⑦。

⑧：如果标志栈栈顶元素的值不为1：

将T赋值为NULL。

转③

⑨：返回OK。

时空复杂度分析：S（n）=O（n） T（n）=O（n）；

（20）按层遍历：status LevelOrderTraverse(Tree T,void (\*visit)(TElemType))

算法输入：指向二叉树根节点的指针T，函数指针visit。

算法输出：函数功能运行状态。

算法思想描述：

①：初始化指针队列。

②：如果树为空，返回ERROR。

③：如果树不为空：

④：T进入队列。

⑤：若队列不为空：

队首元素出列，赋值给指针p。

访问p指向的节点。

如果p的左子树不为空，左子树根节点指针入队。

如果p的右子树不为空，右子树根节点指针入队。

转⑤

⑥：返回OK。

时空复杂度分析：S（n）=O（n） T（n）=O（n）；

## 3.3 系统实现

**3.3.1系统实现**

实现语言为c语言，编译器为mingw32-gcc-4.9.2，操作系统为windows7。

代码中的主要功能除了20种基本操作之外，还包括二叉树数据的文件读写部分和多二叉树管理部分，以及一些除以定义运算外必要的函数。这些函数包括Search()，copy\_tree()和get\_ver\_num()，分别实现了二叉树中的信息匹配,二叉树的复制以及二叉树节点数计算。

文件读写采用单个二进制文件形式，该文件中保存所有信息，包括多树管理的管理信息（依次包括：管理用顺序表的信息，管理用顺序表自身）与具体的各二叉树信息及二叉树节点数据。在进入主菜单之前可以选择是否读文件，退出程序之前可以选择是否保存二叉树数据到文件。从文件读入数据后可追加二叉树。对于每一棵独立的二叉树，使用一个包含当前二叉树信息的结点来指示二叉树，以提供二叉树总览时必要的信息，以及快速提供一些关于该树的数据，例如节点数、层数。

多二叉树管理部分主要体现在main()函数中，采用元素为结构体的顺序表管理多个二叉树，主要维护当前二叉树的个数，当前二叉树在管理用顺序表中的位置，管理用顺序表当前物理长度，各二叉树的名称与指针。管理用顺序表的信息保存于一个独立的节点中，管理用顺序表自身只用于管理各二叉树信息。

对相关功能的陈述：

①摧毁当前二叉树后，操作权默认切换至第一个二叉树。

②从文件初始化二叉树后，操作权默认属于当前第一个二叉树，非文件初始化或在文件数据的基础上追加二叉树时，操作权默认属于当前最后一个二叉树。

③二叉树的名称默认为二叉树的序号。

④20个已界定的运算只对二叉树自身负责，相关管理功能实现中的数据修改置于主函数中

⑤创建二叉树时输入完全先序序列，不存在的节点使用#代替，正常节点的单位为由6个字符构成的串，前三位为标识，后三位为数据。

⑥保存文件时，若某节点左孩子或右孩子为空，则写入一个特殊的空节点进行标志。

**3.3.2系统测试**

本系统首先会给出是否从文件读入数据的选择。为方便后续测试，已经事先创建了5个二叉树以供测试。本测试选择从文件读入数据，同时测试文件读取功能。

5棵二叉树分别如下：

树1：层数为5的单支树，从上到下依次为100 99 98 97 96。

树2：层数为5的满二叉树去掉右子树形成的树，各节点按照先序顺序递增，依次值为1至16。

树3：层数为5的满二叉树去掉左子树但保留左子树根节点形成的树，各节点按照先序顺序依次递减，值为50至34。

树4：只有一个根节点的树，根节点值为100。

树5：层数为4的满二叉树，各节点按照先序顺序依次递减，值为80至66

表3.3.2.1从文件读入数据功能测试二叉树

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 数据文件 | 选项y | 恢复成功 | 与理论相符 |

根据提示按回车后，出现主选单



图3.3.2.2 操作界面截图

1、初始化二叉树功能测试。

表3.3.2.3初始化二叉树功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无特定用例 | 指向指向二叉树根节点的指针的指针 | 二叉树初始化成功 | 与理论相符 |

2、销毁二叉树功能测试：

表3.3.2.4销毁二叉树功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树可以判断 | 与理论相符 |
| 二叉树1 | 二叉树1 | 输出：二叉树销毁成功 | 与理论相符 |

3、创建二叉树功能测试：

表3.3.2.5创建二叉树功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：当前没有已初始化的二叉树 | 与理论相符 |
| 二叉树6 | 空二叉树6，先序序列001001002002##003003## | 输出：二叉树创建成功 | 与理论相符 |

4、清空二叉树功能测试：

表3.3.2.6清空二叉树功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树可以置空 | 符合理论结论 |
| 二叉树6 | 空二叉树6 | 输出：二叉树置空失败 | 符合理论结论 |
| 二叉树2 | 二叉树2 | 输出：二叉树已置空 | 符合理论结论 |

5、判断二叉树是否空功能测试：

表3.3.2.7判断二叉树是否空功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树可以判断 | 符合理论结论 |
| 二叉树6 | 空二叉树6 | 输出：二叉树为空 | 符合理论结论 |
| 二叉树2 | 二叉树2 | 输出：二叉树不为空 | 符合理论结论 |

6、求二叉树深度功能测试：

表3.3.2.8求二叉树深度功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树可以进行计算 | 符合理论结论 |
| 二叉树6 | 空二叉树6 | 输出：二叉树深度为0 | 符合理论结论 |
| 二叉树2 | 二叉树2 | 输出：二叉树深度为5 | 符合理论结论 |

7、获取根节点功能测试：

表3.3.2.9获取根节点功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树 | 符合理论结论 |
| 二叉树6 | 空二叉树6 | 输出：二叉树还没有节点 | 符合理论结论 |
| 二叉树4 | 二叉树4 | 输出：根节点找到，标识为1，数据为100 | 符合理论结论 |

8、获取节点值功能测试：

表3.3.2.10获取节点值功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树 | 符合理论结论 |
| 二叉树6 | 空二叉树6，节点标识1 | 输出：该节点不存在 | 符合理论结论 |
| 二叉树2 | 二叉树2，节点标识12 | 输出：节点值为12 | 符合理论结论 |

9、节点赋值功能测试：

表3.3.2.11节点赋值功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树 | 符合理论结论 |
| 二叉树6 | 空二叉树6，节点标识1，节点值23 | 输出：操作失败 | 符合理论结论 |
| 二叉树2 | 二叉树2，节点标识10，节点值100 | 输出：赋值成功 | 符合理论结论 |

10、求双亲节点功能测试：

表3.3.2.12求双亲节点功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树 | 符合理论结论 |
| 二叉树6 | 空二叉树6，节点标识12 | 输出：操作失败 | 符合理论结论 |
| 二叉树2 | 二叉树2，节点标识9 | 输出：父节点找到，标识为7，值为7 | 符合理论结论 |
| 二叉树4 | 二叉树4，节点标识1 | 输出：操作失败 | 符合理论结论 |

11、求左孩子功能测试：

表3.3.2.13求左孩子功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树 | 符合理论结论 |
| 二叉树6 | 空二叉树6，节点标识12 | 输出：查找失败 | 符合理论结论 |
| 二叉树2 | 二叉树2，节点标识7 | 输出：左孩子标识为8，值为8 | 符合理论结论 |
| 二叉树4 | 二叉树4，节点标识1 | 输出：查找失败 | 符合理论结论 |

12、求右孩子功能测试：

表3.3.2.14求右孩子功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树 | 符合理论结论 |
| 二叉树6 | 空二叉树6，节点标识12 | 输出：查找失败 | 符合理论结论 |
| 二叉树2 | 二叉树2，节点标识7 | 输出：右孩子标识为9，值为9 | 符合理论结论 |
| 二叉树4 | 二叉树4，节点标识1 | 输出：查找失败 | 符合理论结论 |

13、求左兄弟功能测试：

表3.3.2.15求左兄弟功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树 | 符合理论结论 |
| 二叉树6 | 空二叉树6，节点标识12 | 输出：查找失败 | 符合理论结论 |
| 二叉树2 | 二叉树2，节点标识16 | 输出：左兄弟标识为15，值为15 | 符合理论结论 |
| 二叉树1 | 二叉树1，节点标识2 | 输出：查找失败 | 符合理论结论 |

14、求右兄弟功能测试：

表3.3.2.16求右兄弟功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树 | 符合理论结论 |
| 二叉树6 | 空二叉树6，节点标识12 | 输出：查找失败 | 符合理论结论 |
| 二叉树2 | 二叉树2，节点标识4 | 输出：右兄弟标识为7，值为7 | 符合理论结论 |
| 二叉树1 | 二叉树1，节点标识2 | 输出：查找失败 | 符合理论结论 |

15、插入子树功能测试：

表3.3.2.17插入子树功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树 | 符合理论结论 |
| 二叉树6 | 空二叉树6，节点标识12，左子树（0） | 输出：插入失败 | 符合理论结论 |
| 二叉树2 | 二叉树2，节点标识2，左子树（0） | 输出：插入成功 | 符合理论结论 |
| 二叉树2 | 二叉树2，节点标识2，右子树（1） | 输出：插入成功 | 符合理论结论 |

16、删除子树功能测试：

表3.3.2.18删除子树功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树 | 符合理论结论 |
| 二叉树6 | 空二叉树6，节点标识12，左子树（0） | 输出：操作失败 | 符合理论结论 |
| 二叉树1 | 二叉树1，节点标识2，左子树（0） | 输出：左子树已删除 | 符合理论结论 |
| 二叉树1 | 二叉树1，节点标识2，右子树（1） | 输出：右子树已删除 | 符合理论结论 |

17、先序遍历功能测试：

表3.3.2.19先序遍历功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树 | 符合理论结论 |
| 二叉树6 | 空二叉树6 | 输出：操作失败 | 符合理论结论 |
| 二叉树1 | 二叉树1 | 输出：100 99 98 97 96 遍历完成 | 符合理论结论 |

18、中序遍历功能测试：

表3.3.2.20中序遍历功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树 | 符合理论结论 |
| 二叉树6 | 空二叉树6 | 输出：操作失败 | 符合理论结论 |
| 二叉树1 | 二叉树1 | 输出：96 97 98 99 100 遍历完成 | 符合理论结论 |

19、后序遍历功能测试：

表3.3.2.21后序遍历功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树 | 符合理论结论 |
| 二叉树6 | 空二叉树6 | 输出：操作失败 | 符合理论结论 |
| 二叉树1 | 二叉树1 | 输出：96 97 98 99 100 遍历完成 | 符合理论结论 |

20、层序遍历功能测试：

表3.3.2.22层序遍历功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无二叉树 | 输出：没有二叉树 | 符合理论结论 |
| 二叉树6 | 空二叉树6 | 输出：操作失败 | 符合理论结论 |
| 二叉树1 | 二叉树1 | 输出：100 99 98 97 96 遍历完成 | 符合理论结论 |

21、二叉树控制切换功能测试：

如下图，选择功能21，即可显示当前内存中的所有二叉树，以及其名称与长度。可以选择一个二叉树进行操作。

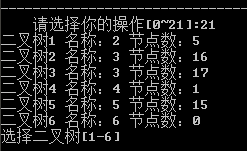


图3.3.2.23切换功能演示图

22、数据保存功能测试：

表3.3.2.24存储功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 结构中所有数据 | 结构中所有数据，删除一个二叉树，保存后再读取 | 保存成功，可供下次正确读取 | 与理论相符 |

综上：该系统实现了全部20个功能，并且对边界条件的反应程度良好，异常判断较齐全，符合实验要求。

## 3.4 实验小结

本次实验需要实现的功能较多，也较复杂，因此花了较多的时间。虽然绝大多数函数使用了非递归算法，但是仍有一些实现较复杂的函数使用了递归。虽然形式上简单了，但是仍然要注意递归的终止条件与各层递归结果之间的连接。同时撰写实验报告的同时了解了递归的空间复杂度的来源，即函数调用过程的内存活动，每一个函数调用过程都有一个独立的栈帧，因此递归实际上是压栈出栈操作支持的。这次实验让我对二叉树的操作有了实际经验，加深了对二叉树的理解。

# 4基于邻接表的图实现

## 4.1 问题描述

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

### 4.1.1 图基本运算定义

依据最小完备性和常用性相结合的原则，以函数形式定义了创建图、销毁图、查找顶点、获得顶点值和顶点赋值等13种基本运算，具体运算功能定义如下。

⑴创建图：

函数名称：CreateCraph(&G,V,VR)；

初始条件：V是图的顶点集，VR是图的关系集；

操作结果：按V和VR的定义构造图G。

⑵销毁图：

函数名称：DestroyGraph(T)；

初始条件：图G已存在；

操作结果：销毁图G。

⑶查找顶点：

函数名称：LocateVex(G,u)；

初始条件：图G存在，u和G中的顶点具有相同特征；

操作结果：若u在图G中存在，返回顶点u的位置信息，否则返回其它信息。

⑷获得顶点值：

函数名称：GetVex (G,v)；

初始条件：图G存在，v是G中的某个顶点；

操作结果：返回v的值。

⑸顶点赋值：

函数名称：PutVex (G,v,value)；

初始条件：图G存在，v是G中的某个顶点；

操作结果：对v赋值value。

⑹获得第一邻接点：

函数名称：FirstAdjVex(&G, v)；

初始条件：图G存在，v是G的一个顶点；

操作结果：返回v的第一个邻接顶点，如果v没有邻接顶点，返回“空”。

⑺获得下一邻接点：

函数名称：NextAdjVex(&G, v, w)；

初始条件：图G存在，v是G的一个顶点,w是v的邻接顶点；

操作结果：返回v的（相对于w）下一个邻接顶点，如果w是最后一个邻接顶点，返回“空”。

⑻插入顶点：

函数名称：InsertVex(&G,v)；

初始条件：图G存在，v和G中的顶点具有相同特征；

操作结果：在图G中增加新顶点v。

⑼删除顶点：

函数名称：DeleteVex(&G,v)；

初始条件：图G存在，v是G的一个顶点；

操作结果：在图G中删除顶点v和与v相关的弧。

⑽插入弧：

函数名称：InsertArc(&G,v,w)；

初始条件：图G存在，v、w是G的顶点；

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

⑾删除弧：

函数名称：DeleteArc(&G,v,w)；

初始条件：图G存在，v、w是G的顶点；

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

⑿深度优先搜索遍历：

函数名称：DFSTraverse(G,visit())；

初始条件：图G存在；

操作结果：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：BFSTraverse(G,visit())；

初始条件：图G存在；

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

## 4.2 系统设计

### 4.2.1 系统总体设计

采用简易菜单，总体结构为循环结构，内部为对图的13种操作和退出选项，除13种基本操作外，还增加了切换图的选项，其中退出与切换图，以及一些相关量的维护的实现位于主函数中；外部为多图管理模块，可在内存中操作多个图。支持无向图、无向网、有向图、有向网四种图类型。使用switch语句支持操作的选择（输入数字进行选择），使用while语句支持主循环。程序主流程图如下：

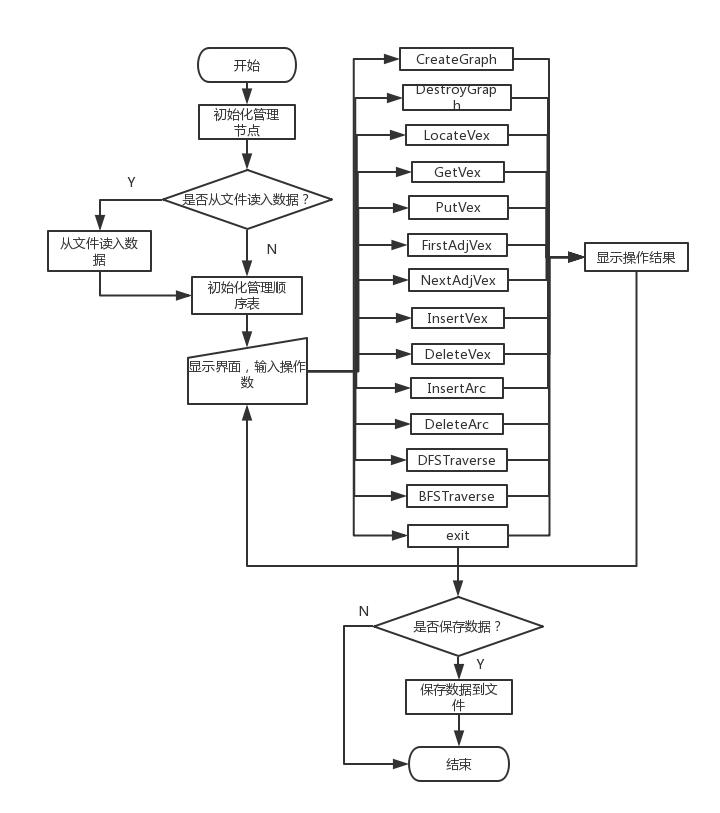


图4.2.1.1 系统主流程图

### 4.2.2 有关常量和类型的定义

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<malloc.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

//#define MAX\_LENGTH 512

#define MAX\_VERTEX\_NUM 20

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

typedef int status;

typedef char Vertex;

typedef int Arcinfo;

typedef struct ArcNode{//表结点结构类型

int adjvex; //该弧(边)的终点位置

struct ArcNode\* nextarc; //指向下一条弧的指针

Arcinfo weight; //该弧的相关信息的指针

} ArcNode, \*p\_ArcNode;

typedef struct Vnode {//头结点的类型

Vertex data; //顶点信息

struct ArcNode \*firstarc; //指向第一条弧

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

typedef struct ALGraph{//邻接表

AdjList vertices;

int vexnum, arcnum; //图中顶点数n和边数e

GraphKind kind; //图的类型

} ALGraph, \*GraphPtr, \*Graph;

typedef struct ListInfo{

char name[20];

struct ALGraph\* elem;

}ListInfo;

typedef struct ManageList{

struct ListInfo\* elem;

int length;

int listsize;

int cur;

}ManageList;

typedef struct qu{

struct qu\* pre;

int v;

struct qu\* next;

}qu, \*que;

### 4.2.3 算法设计

（1）创建图：Graph CreateGraph(Graph g);

算法输入：图所在结构体的指针g。

算法输出：指向该图的指针。

算法思想描述：

①：输入图的类型。

②：输入顶点数和边数。

③：输入各顶点值。

④：若为有向图

⑤：输入边的头和尾。

⑥：在尾部对应的顶点后添加边节点。

⑦：若为有向网

⑧：输入边的头和尾及权值。

⑨：在尾部对应的顶点后添加边节点。

⑩：若为无向图

⑪：输入相邻接的两个顶点。

⑫：在两个顶点后分别添加边节点。

⑬：若为无向网

⑭：输入相邻接的两个顶点和权值。

⑮：在两个顶点后分别添加边节点。

时空复杂度分析：S（n）=O（n）；T（n）=O（n2）；

（2）销毁图：status DestroyGraph(Graph g);

算法输入：图所在结构体的指针g。

算法输出：函数功能运行状态。

算法思想描述：

①：依次销毁各顶点对应的边节点。

②：销毁图所在的结构体。

③：返回OK；

时空复杂度分析：S（n）=O(1)；T(n)=O(n)；

（3）查找顶点：int LocateVex(Graph g,Vertex u);

算法输入：图所在结构体的指针g，节点值u。

算法输出：顶点位置编号。

算法思想描述：

①：遍历顶点顺序表，若顶点的值等于u，返回编号。

②：若未找到符合相应条件的顶点，返回-1。

时空复杂度分析：S（n）=O（1）；T（n）=O（n）；

（4）获得结点值：Vertex GetVex(Graph g,int v);

算法输入：图所在结构体的指针g，顶点编号v。

算法输出：顶点值。

算法思想描述：

①：若v小于0或v大于等于顶点数，返回#。

②：若v合法，返回该顶点的值；

时空复杂度分析：S（n）=O(1)；T(n)=O(1)；

（5）顶点赋值：status PutVex(Graph g,int v,Vertex value);

算法输入：图所在结构体的指针g，顶点编号v，期望的顶点值value。

算法输出：函数功能运行状态。

算法思想描述：

①：若v小于0或v大于等于顶点数，返回ERROR。

②：若v合法，将该顶点的值改为value的值；

时空复杂度分析：S（n）=O（1） T（n）=O（1）；

（6）获得第一邻接点：int FirstAdjVex(Graph g,int v);

算法输入：图所在结构体的指针g，顶点编号v。

算法输出：第一邻接点编号。

算法思想描述：

①:若v小于0或v大于等于顶点数，返回-2。

②:若该顶点不存在第一邻接点，返回-1。

③：返回第一邻接点的编号。

时空复杂度分析：S（n）=O（1） T（n）=O（1）；

（7）获得下一邻接点：int NextAdjVex(Graph g,int v, int w);

算法输入：图所在结构体的指针g，顶点编号v，邻接顶点编号w。

算法输出：下一邻接点编号。

①: 若v小于0或v大于等于顶点数或w小于0或w大于等于顶点数，返回-2。

②:在编号为v的顶点的边中搜索邻接点编号为w的边，若找到，转③，若未找到，返回-2。

③：若该边节点的后一个边节点存在，返回后一个节点对应的邻接点编号，否则返回-1。

时空复杂度分析：S（n）=O（1） T（n）=O（n）；

（8）插入顶点：status InsertVex(Graph g,Vertex v);

算法输入：图所在结构体的指针g，目标顶点值v。

算法输出：函数运行状态。

算法思想描述：

①：如果顶点数大于等于最大顶点数，返回ERROR。

②：在顶点数组的末尾添加记录。

③：顶点数加1。

④：返回OK。

时空复杂度分析：S（n）=O（1）T（n）=O（1）；

（9）删除顶点：status DeleteVex(Graph g,int v);

算法输入：图所在结构体的指针g，目标顶点编号v。

算法输出：函数功能运行状态。

算法思想描述：

①：若v小于0或v大于等于顶点数，返回ERROR。

②：若为有向图或有向网

③：销毁编号为v的顶点之后的所有边节点，即以该顶点为尾的边

④：销毁以该顶点为头的边对应的边节点

⑤：若为无向图或无向网

⑥：销毁编号为v的顶点之后的所有边节点

⑦：销毁其他所有顶点之后与该顶点有关的边节点

⑧：将目标顶点之后的顶点依次前移一个单位

⑨：将边节点中大于v的编号全部-1

⑩：更新边数与顶点数

⑪：返回OK

算法时空复杂度分析：S（n）=O（1）T（n）=O（m\*n）；

（10）插入弧：status InsertArc(Graph g,int v,int w);

算法输入：图所在结构体的指针g，顶点编号v，顶点编号w。

算法输出：函数功能运行状态。

算法思想描述：

①：如果v或w不合法，返回ERROR。

②：如果该图为有向图或有向网，在v对应顶点对应的边链表最后添加边节点，其中邻接点编号为w。若图为有向网，要求输入权值，并更新边权值。

③：如果该图为无向图或无向网，在v对应顶点对应的边链表最后添加边节点，其中邻接点编号为w。若图为无向网，要求输入权值，并更新边权值。对w所对应的顶点做相同操作，但无需再次输入权值。

④：边数加1

⑤：返回OK

时空复杂度分析：S（n）=O（1）T（n）=O（n）;

（11）删除弧：status DeleteArc(Graph g,int v,int w);

算法输入：图所在结构体的指针g，顶点编号v，顶点编号w。

算法输出：函数功能运行状态。

算法思想描述：

①：如果v或w不合法，返回ERROR。

②：如果该图为有向图或有向网，删除v对应的顶点后的边链表中邻接点编号为w的边节点。

③：如果该图为无向图或无向网，删除v对应的顶点后的边链表中邻接点编号为w的边节点。删除w对应的顶点后的边链表中邻接点编号为v的边节点。

④：边数-1。

⑤：返回OK。

时空复杂度分析：S（n）=O（1） T（n）=O（n）；

（12）深度优先搜索遍历：status DFSTraverse(Graph g,void (\*visit)(Vertex));

算法输入：图所在结构体的指针g，函数指针visit。

算法输出：函数功能运行状态。

算法思想描述：

①：初始化visited数组，使各元素都为FALSE。

②：从第一个顶点开始，依次地，若顶点未被访问过，对其调用DFS函数。

DFS函数：

①：初始化栈。

②：起始顶点入栈。

③：当栈不为空

④：栈顶出栈，为v，若顶点v未被访问过，访问顶点，设置访问标记为TRUE。

⑤：获取v的第一邻接点为w。

⑥：若w未被访问过，w入栈。

⑦：获取v相对于w的下一邻接点为w，若w存在，转⑥，若w不存在，转③

时空复杂度分析：S（n）=O（n） T（n）=O（n+e）；

（13）广度优先搜索遍历：status BFSTraverse(Graph g,void (\*visit)(Vertex));

算法输入：图所在结构体的指针g，函数指针visit。

算法输出：函数功能运行状态。

算法思想描述：

①：初始化visited数组，使各元素都为FALSE。

②：初始化链队列。

③：依次对各顶点进行如下操作（当前顶点为v）

④：若该顶点未访问，访问该顶点，设置访问标记为TRUE。v入队。若访问过，操作进行到下一个顶点。

⑤：若队列不为空，转⑥，若为空，操作进行到下一个顶点。

⑥：队头出队并为u

⑦：获得u的第一邻接点为w

⑧：若w未访问过，访问w并设访问标记为TRUE,w入队

⑨：获取u相对于w的下一邻接点为w,若w存在，转⑧。若w不存在，转⑤

⑩：返回OK

时空复杂度分析：S（n）=O（n） T（n）=O（n+e）；

## 4.3 系统实现

**4.3.1系统实现**

实现语言为c语言，编译器为mingw32-gcc-4.9.2，操作系统为windows7。

代码中的主要功能除了13种基本操作之外，还包括图数据的文件读写部分和多图管理部分。

文件读写采用单个二进制文件形式，该文件中保存所有信息，包括多图管理的管理信息（依次包括：管理用顺序表的信息，管理用顺序表自身）与图节点数据。在进入主菜单之前可以选择是否读文件，退出程序之前可以选择是否保存图数据到文件。从文件读入数据后可追加图。管理用顺序表主要维护图的名称和图的指针。

多图管理部分主要体现在main()函数中，采用元素为结构体的顺序表管理多个图，主要维护当前图的个数，当前图在管理用顺序表中的位置，管理用顺序表当前物理长度，各图的名称与指针。管理用顺序表的信息保存于一个独立的节点中，管理用顺序表自身只用于管理各图信息。

对相关功能的陈述：

①摧毁当前图后，操作权默认切换至第一个图。

②从文件初始化图后，操作权默认属于当前第一个图，非文件初始化或在文件数据的基础上追加图时，操作权默认属于当前最后一个图。

③图的名称默认为图的序号。

④13个已界定的运算只对图自身负责，相关管理功能实现中的数据修改置于主函数中

⑤保存文件时，各顶点之后的边链表达到末尾时，写入一个空节点以示该链表结束。

**4.3.2系统测试**

本系统首先会给出是否从文件读入数据的选择。为方便后续测试，已经事先创建了4个图以供测试。本测试选择从文件读入数据，同时测试文件读取功能。

4个图分别如下：

图1：

图2：

5

6

4

3

2

1

图3:

图4：

8

7

6

5

4

3

2

1

表4.3.2.1从文件读入数据功能测试图

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 数据文件 | 选项y | 恢复成功 | 与理论相符 |

根据提示按回车后，出现主选单



图4.3.2.2 操作界面截图

1、创建图功能测试。

表4.3.2.3创建图功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 3个顶点的无向图 | 顶点数3，边数3  顶点值a,b,c,边<a,b>,<b,c>,<c,a> | 3 arcs done  3 arcs inputed  Create success | 见图4.3.2.4 |
| 3个顶点的无向网 | 顶点数3，边数3  顶点值a,b,c,边<a,b>,<b,c>,<c,a>  权值1,2,3 | 3 arcs done  3 arcs inputed  Create success | 见图4.3.2.5 |
| 3个顶点的有向网 | 顶点数3，边数3  顶点值a,b,c,边<a,b>,<b,c>,<c,a>  权值1,2,3 | 3 arcs done  3 arcs inputed  Create success | 见图4.3.2.6 |
| 3个顶点的有向图 | 顶点数3，边数3  顶点值a,b,c,边<a,b>,<b,c>,<c,a> | 3 arcs done  3 arcs inputed  Create success | 见图4.3.2.7 |

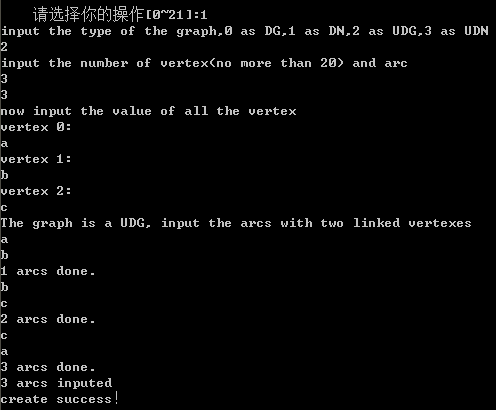


图4.3.2.4

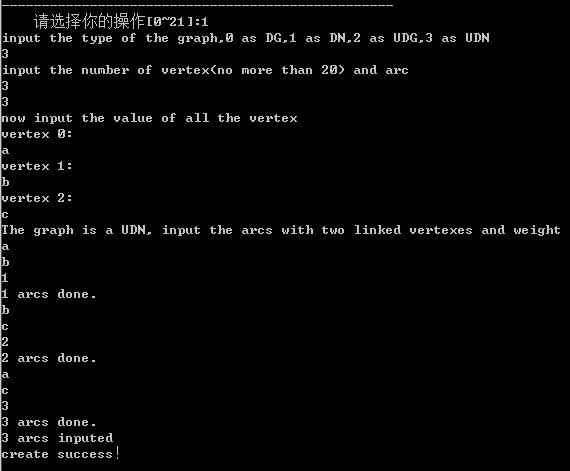


图4.3.2.5

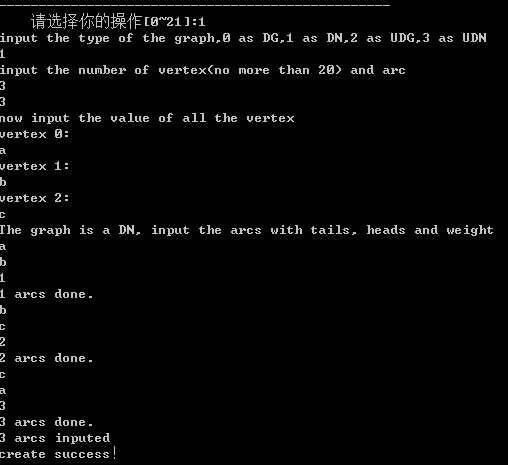


图4.3.2.6

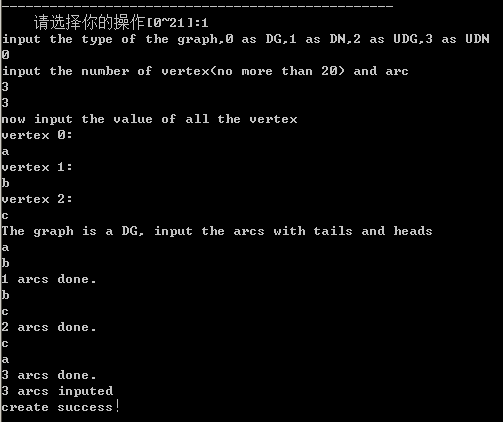


图4.3.2.7

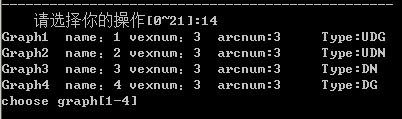


图4.3.2.8

可以看出，与理论结果相符

2、销毁图功能测试：

表4.3.2.9销毁图功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无图 | No graph to destroy | 与理论相符 |
| 图1 | 图1 | Destroy success | 与理论相符 |

3、查找顶点功能测试：

表4.3.2.10查找顶点功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无图 | No created graph | 与理论相符 |
| 图1 | 顶点值c | The location is 2 | 与理论相符 |

4、获得顶点值功能测试：

表4.3.2.11清空图功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无图 | No created graph | 符合理论结论 |
| 图1 | 顶点编号3 | The value is d! | 符合理论结论 |

5、顶点赋值功能测试：

表4.3.2.12顶点赋值功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无图 | No created graph | 符合理论结论 |
| 图2 | 顶点编号1和值b | done | 符合理论结论 |

6、获得第一邻接点功能测试：

表4.3.2.13获得第一邻接点功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无图 | No created graph | 符合理论结论 |
| 图4 | 顶点编号0 | The location of the first adjvex is 1 | 符合理论结论 |

7、获得下一邻接点功能测试：

表4.3.2.14获得下一邻接点功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无图 | No created graph | 符合理论结论 |
| 图4 | 顶点编号0，上一邻接点编号1 | The location of the next adjvex is 3 | 符合理论结论 |

8、插入顶点功能测试：

表4.3.2.15插入顶点功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无图 | No created graph | 符合理论结论 |
| 图1 | 顶点值z | Insertion done | 符合理论结论 |

9、删除顶点功能测试：

表4.3.2.16删除顶点功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无图 | No created graph | 符合理论结论 |
| 图4 | 顶点编号4 | deleted | 符合理论结论 |

10、插入弧功能测试：

表4.3.2.17插入弧功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无图 | No created graph | 符合理论结论 |
| 图1 | 弧尾编号1，弧头编号3 | Insertion done | 符合理论结论 |

11、删除弧功能测试：

表4.3.2.18删除弧功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无图 | No created graph | 符合理论结论 |
| 图4 | 一对邻接顶点的编号0,1 | deleted | 符合理论结论 |

12、深度优先搜索遍历功能测试：

表4.3.2.19深度优先搜索遍历功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无图 | No created graph | 符合理论结论 |
| 图1 | 图1 | a f g d e h b c  done | 符合理论结论 |
| 图3 | 图3 | a e c d b  done | 符合理论结论 |

13、广深度优先搜索遍历功能测试：

表4.3.2.20广深度优先搜索遍历功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 无 | 无图 | No created graph | 符合理论结论 |
| 图2 | 图2 | a b c d e f g  done | 符合理论结论 |
| 图4 | 图4 | a b d e c | 符合理论结论 |

14、图控制切换功能测试：

如下图，选择功能21，即可显示当前内存中的所有图，以及其名称与顶点数，边数与类别。可以选择一个图进行操作。

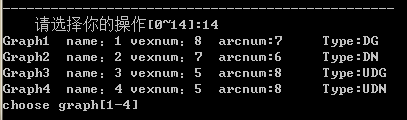


图4.3.2.21切换功能演示图

15、数据保存功能测试：

表4.3.2.22存储功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行功能 |
| 结构中所有数据 | 结构中所有数据，删除一个图，保存后再读取 | 保存成功，可供下次正确读取 | 与理论相符 |

综上：该系统实现了全部13个功能，并且对边界条件的反应程度良好，异常判断较齐全，符合实验要求。

## 4.4 实验小结

本次实验实践了图的邻接表表示法，并且实现了4种图的操作，从实际角度明白了各种类型的图在使用邻接表表示时的操作有什么不同，也了解了使用邻接表表示图的优点和缺点。在动手写过2个遍历算法后对这两个算法有了更深的理解。总体上促使了我编程水平的提高，也使我用数据结构的观点去看问题。

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

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

[3] Computer Systems A Programmer’s Perspective, Third Edition, Randal E.Bryant

[4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 附录A 基于顺序存储结构线性表实现的源程序

def.h:

//主要定义

#ifndef DEF\_H\_INCLUDED

#define DEF\_H\_INCLUDED

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<malloc.h> //包含头文件

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10 //定义状态量

typedef int status;

typedef int ElemType;

typedef struct SqList{ //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

typedef struct ListInfo{ //表信息管理结构定义

char name[20];

struct SqList\* elem;

}ListInfo;

typedef struct ManageList{ //表管理顺序表的信息结构定义

struct ListInfo\* elem;

int length;

int listsize;

int cur;

}ManageList;

//函数声明

status IntiaList(SqList \*L);

status DestroyList(SqList \*L);

status ClearList(SqList \*L);

status ListEmpty(SqList \*L);

int ListLength(SqList \*L);

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

status LocateElem(SqList \*L,ElemType e,int (\*compare)(ElemType,ElemType));

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

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

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

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

status ListTraverse(SqList \*L,void (\*visit)(ElemType));

void visit(ElemType e);

status compare(ElemType e,ElemType f);

void save(ManageList\* M);

void recover(ManageList\* M,SqList \*\*L);

#endif // DEF\_H\_INCLUDED

main.c:

//主函数

#include "def.h"

int main()

{

SqList\* L; int op=1; char rec;

ManageList M;

//M.elem=(ListInfo\*)malloc(sizeof(ListInfo)\*10);

M.length=0;

M.listsize=10;

M.cur=-1; //表管理初始化

printf("是否从文件读入数据？y/n\n");

scanf("%c",&rec); fflush(stdin);

if(rec=='y') recover(&M,&L);

else M.elem=(ListInfo\*)malloc(sizeof(ListInfo)\*10); //从文件恢复数据

while(op) //主循环

{

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

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

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

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

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

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

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

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

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

printf(" 13. switch list\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~13]:"); //界面输出

scanf("%d",&op);

switch(op){

case 1: //表初始化

{

if(M.length+1>=M.listsize)

{

ListInfo\* newelem=(ListInfo\*)realloc(M.elem,(M.listsize+1)\*sizeof(ListInfo));

if(!newelem) exit(OVERFLOW);

M.elem=newelem;

M.listsize++;

}

M.elem[M.length].elem=(SqList\*)malloc(sizeof(SqList));

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

L->elem=NULL;

L->length=0;

L->listsize=0;

if(IntiaList(L)==OK)

{

printf("线性表创建成功！\n");

itoa(M.length+1,M.elem[M.length].name,10);

++(M.length);

M.cur=M.length-1;

fflush(stdin); getchar();

break;

}

else

{

free(L);

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

printf("线性表创建失败！\n");

fflush(stdin); getchar();

break;

}

}

case 2: //表摧毁

if(M.length==0)

{

printf("没有表可以摧毁！\n"); fflush(stdin); getchar();

break;

}

if(DestroyList(L)==OK)

{

int i,j;

free(L);

for(i=M.cur;i<M.length;i++)

{

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

}

for(j=0;j<M.length;j++)

{

itoa(j+1,M.elem[j].name,10);

}

M.length--;

M.cur=0;

L=M.elem[0].elem;

printf("线性表销毁成功！\n");

}

else printf("线性表销毁失败！\n");

fflush(stdin); getchar();

break;

case 3: //表置空

if(M.length==0)

{

printf("没有表可以置空！\n"); fflush(stdin); getchar();

break;

}

if(ClearList(L)==OK) printf("线性表已置空！\n");

else printf("线性表置空失败！\n");

fflush(stdin); getchar();

break;

case 4: //判断表是否空

{

if(M.length==0)

{

printf("没有表可以判断！\n"); fflush(stdin); getchar();

break;

}

status s=ListEmpty(L);

if(s==-1) printf("线性表不存在！\n");

if(s==TRUE) printf("线性表为空表！\n");

if(s==FALSE) printf("线性表不为空！\n");

fflush(stdin); getchar();

break;

}

case 5: //计算表长度

{

if(M.length==0)

{

printf("没有表可以进行计算！\n"); fflush(stdin); getchar();

break;

}

int length=ListLength(L);

if(length==-1) printf("线性表不存在！\n");

else printf("线性表元素个数为%d个\n",length);

fflush(stdin); getchar();

break;

}

case 6: //获取元素值

{

if(M.length==0)

{

printf("没有表！\n"); fflush(stdin); getchar();

break;

}

int i;

ElemType e;

printf("第i个元素的值\n输入i:");

scanf("%d",&i); fflush(stdin);

if(GetElem(L,i,&e)==OK) printf("第%d个元素的值为%d\n",i,e);

else printf("该数据元素不存在\n");

fflush(stdin); getchar();

break;

}

case 7: //元素定位

{

if(M.length==0)

{

printf("没有表！\n"); fflush(stdin); getchar();

break;

}

ElemType e;

int i;

printf("元素定位\n输入元素e:");

scanf("%d",&e); fflush(stdin);

i=LocateElem(L,e,compare);

if(i) printf("e元素位序为%d\n",i);

else printf("数据元素不存在\n");

fflush(stdin); getchar();

break;

}

case 8: //获取前驱

{

if(M.length==0)

{

printf("没有表！\n"); fflush(stdin); getchar();

break;

}

ElemType pre\_e,cur;

printf("元素前驱查找\n输入元素cur:");

scanf("%d",&cur); fflush(stdin);

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

else printf("操作失败！\n");

fflush(stdin); getchar();

break;

}

case 9: //获取后继

{

if(M.length==0)

{

printf("没有表！\n"); fflush(stdin); getchar();

break;

}

ElemType next\_e,cur;

printf("元素后继查找\n输入元素cur:");

scanf("%d",&cur); fflush(stdin);

if(NextElem(L,cur,&next\_e)==OK) printf("元素cur的后继元素为%d\n",next\_e);

else printf("操作失败！\n");

fflush(stdin); getchar();

break;

}

case 10: //插入元素

{

if(M.length==0)

{

printf("没有表！\n"); fflush(stdin); getchar();

break;

}

int i;

ElemType e;

printf("在第i个元素之前插入元素e\n输入i:");

scanf("%d",&i); fflush(stdin);

printf("输入元素e\n");

scanf("%d",&e); fflush(stdin);

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

else printf("插入失败！\n");

fflush(stdin);getchar();

break;

}

case 11: //删除元素

{

if(M.length==0)

{

printf("没有表！\n"); fflush(stdin); getchar();

break;

}

int i;

ElemType e;

printf("删除第i个元素\n输入i:");

scanf("%d",&i); fflush(stdin);

if(ListDelete(L,i,&e)==OK) printf("元素删除成功！第%d个元素值为%d\n",i,e);

else printf("删除失败！\n");

fflush(stdin); getchar();

break;

}

case 12: //遍历表

{

if(M.length==0)

{

printf("没有表！\n"); fflush(stdin); getchar();

break;

}

if(ListEmpty(L)) printf("线性表是空表！\n");

else

{

ListTraverse(L,visit); printf("\n遍历完成\n");

}

fflush(stdin); getchar();

break;

}

case 13: //切换表

{

int i,j;

if(M.length==0)

{

printf("当前没有线性表\n");

fflush(stdin); getchar();

break;

}

if(M.length==1)

{

printf("当前只有1个线性表\n");

fflush(stdin); getchar();

break;

}

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

{

printf("表%d\t名称：%s\t长度：%d\n",i+1,(M.elem+i)->name,(M.elem+i)->elem->length);

}

printf("选择线性表[1-%d]\n",M.length);

scanf("%d",&j); fflush(stdin);

L=(M.elem+j-1)->elem;

M.cur=j-1;

printf("切换成功\n");

fflush(stdin); getchar();

break;

}

case 0:

break;

}

}

char choose;

printf("是否保存数据到文件？\ty/n:"); fflush(stdin); //文件保存

scanf("%c",&choose); fflush(stdin);

printf("\n");

if(choose=='y') save(&M);

printf("欢迎下次再使用本系统！\n\n---回车键退出---\n");

getchar();

return 0;

}

operates.c:

//各具体操作与功能实现

#include "def.h"

/\*

函数名称：IntiaList

函数参数：指向顺序表的指针

函数功能：初始化线性表

\*/

status IntiaList(SqList \*L)

{

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

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

L->length=0;

L->listsize=LIST\_INIT\_SIZE;

return OK;

}

/\*

函数名称：DestroyList

函数参数：指向顺序表的指针

函数功能：摧毁线性表

\*/

status DestroyList(SqList \*L)

{

if(!L->elem) return ERROR;

free(L->elem);

L->elem=NULL;

L->length=0;

L->listsize=0;

return OK;

}

/\*

函数名称：ClearList

函数参数：指向顺序表的指针

函数功能：清空线性表

\*/

status ClearList(SqList \*L)

{

if(!L->elem) return ERROR;

free(L->elem);

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

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

L->length=0;

return OK;

}

/\*

函数名称：ListEmpty

函数参数：指向顺序表的指针

函数功能：判断线性表是否为空

\*/

status ListEmpty(SqList \*L)

{

if(!L->elem) return -1;

if(L->length==0) return TRUE;

else return FALSE;

}

/\*

函数名称：ListLength

函数参数：指向顺序表的指针

函数功能：求线性表的表长

\*/

int ListLength(SqList \*L)

{

if(!L->elem) return -1;

return L->length;

}

/\*

函数名称：GetElem

函数参数：指向顺序表的指针L,元素位序i

函数功能：取线性表中的元素

\*/

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

{

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

\*e=L->elem[i-1];

return OK;

}

/\*

函数名称：compare

函数参数：元素e,f

函数功能：比较两元素是否相等

\*/

status compare(ElemType e,ElemType f)

{

return e==f?TRUE:FALSE;

}

/\*

函数名称：LocateElem

函数参数：指向顺序表的指针L，目标元素e，函数指针compare

函数功能：定位线性表中的元素

\*/

status LocateElem(SqList \*L,ElemType e,int (\*compare)(ElemType,ElemType))

{

if(!L->elem) return ERROR;

int i=1;

ElemType \*p=L->elem;

while(i<=L->length&&!(\*compare)(\*p++,e)) ++i;

if(i<=L->length) return i;

else return 0;

}

/\*

函数名称：PriorElem

函数参数：指向顺序表的指针L，目标元素cur，指针pre\_e

函数功能：获取目标元素的前驱

\*/

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

{

int i=LocateElem(L,cur,compare);

if(i==1||i>L->length||i==0) return ERROR;

else

{

\*pre\_e=L->elem[i-2];

return OK;

}

}

/\*

函数名称：NextElem

函数参数：指向顺序表的指针L，目标元素cur，指针next\_e

函数功能：获取目标元素的后继

\*/

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

{

int i=LocateElem(L,cur,compare);

if(i>=L->length||i==0) return ERROR;

else

{

\*next\_e=L->elem[i];

return OK;

}

}

/\*

函数名称：ListInsert

函数参数：指向顺序表的指针L，要插入元素的位置i，插入的元素e

函数功能：插入元素

\*/

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

{

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

if(L->length>=L->listsize-1)

{

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

if(!newbase) exit(OVERFLOW);

L->elem=newbase;

L->listsize+=LISTINCREMENT;

}

ElemType \*q=L->elem+i-1;

ElemType \*p;

for(p=L->elem+L->length-1;p>=q;--p)

{

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

}

\*q=e;

++L->length;

return OK;

}

/\*

函数名称：ListDelete

函数参数：指向顺序表的指针L，要删除元素的位序i，指针e

函数功能：删除元素

\*/

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

{

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

ElemType \*p,\*q;

p=L->elem+i-1;

\*e=\*p;

q=L->elem+L->length-1;

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

{

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

}

--L->length;

return OK;

}

/\*

函数名称：visit

函数参数：元素e

函数功能：输出元素e

\*/

void visit(ElemType e)

{

printf("%d\t",e);

}

/\*

函数名称：ListTraverse

函数参数：指向顺序表的指针L，函数指针visit

函数功能：遍历输出顺序表中所有元素的值

\*/

status ListTraverse(SqList \*L,void (\*visit)(ElemType))

{

if(!L->elem) return ERROR;

ElemType \*p;

for(p=L->elem;(p-L->elem)<=L->length-1;p++)

{

(\*visit)(\*p);

}

return OK;

}

/\*

函数名称：save

函数参数：指向管理表信息节点的指针

函数功能：保存数据到文件

\*/

void save(ManageList\* M)

{

if(M->length==0) return;

FILE\* fp=NULL;

int i;

fp=fopen("save.dat","wb");

fwrite(M,sizeof(ManageList),1,fp);

fwrite(M->elem,sizeof(ListInfo),M->listsize,fp);

for(i=0;i<M->length;i++)

{

fwrite((M->elem+i)->elem,sizeof(SqList),1,fp);

fwrite((M->elem+i)->elem->elem,sizeof(ElemType),(M->elem+i)->elem->listsize,fp);

free((M->elem+i)->elem->elem);

free((M->elem+i)->elem);

}

free(M->elem);

printf("已保存\n");

}

/\*

函数名称：recover

函数参数：指向管理表信息节点的指针M，指向指向顺序表的指针的指针L

函数功能：从文件读取数据恢复至内存

\*/

void recover(ManageList\* M,SqList \*\*L)

{

FILE\* fp=NULL;

int i;

fp=fopen("save.dat","rb");

fread(M,sizeof(ManageList),1,fp);

ListInfo\* info=(ListInfo\*)malloc(sizeof(ListInfo)\*(M->listsize));

M->elem=info;

fread(M->elem,sizeof(ListInfo),M->listsize,fp);

for(i=0;i<M->length;i++)

{

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

fread(L,sizeof(SqList),1,fp);

(M->elem+i)->elem=L;

ElemType\* e\_ptr=(ElemType \*)malloc( (L->listsize) \* sizeof (ElemType));

fread(e\_ptr,sizeof (ElemType),L->listsize,fp);

L->elem=e\_ptr;

}

M->cur=0;

\*L=M->elem[0].elem;

printf("恢复成功,按回车继续\n"); getchar();

}

# 附录B基于链式存储结构线性表实现的源程序

def.h:

//主要定义

#ifndef DEF\_H\_INCLUDED

#define DEF\_H\_INCLUDED

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<malloc.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef int status;

typedef int ElemType;

typedef struct LNode{

ElemType data;

struct LNode\* next;

}LNode, \*LNode\_ptr, \*Position;

typedef struct LinkList{

struct LNode\* head;

//struct LNode\* tail;

int length;

}LinkList;

typedef struct ListInfo{

char name[20];

struct LinkList\* elem;

}ListInfo;

typedef struct ManageList{

struct ListInfo\* elem;

int length;

int listsize;

int cur;

}ManageList;

status IntiaList(LinkList \*L);

status DestroyList(LinkList \*L);

status ClearList(LinkList \*L);

status ListEmpty(LinkList \*L);

int ListLength(LinkList \*L);

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

status LocateElem(LinkList \*L,ElemType e,int (\*compare)(ElemType,ElemType));

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

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

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

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

status ListTraverse(LinkList \*L,void (\*visit)(ElemType));

void visit(ElemType e);

status compare(ElemType e,ElemType f);

void save(ManageList\* M);

void recover(ManageList\* M,LinkList \*\*L);

#endif // DEF\_H\_INCLUDED

main.c:

//主函数

#include "def.h"

int main()

{

LinkList\* L; int op=1; char rec;

ManageList M;

M.length=0;

M.listsize=10;

M.cur=-1;

printf("是否从文件读入数据？y/n\n");

scanf("%c",&rec); fflush(stdin);

if(rec=='y') recover(&M,&L);

else M.elem=(ListInfo\*)malloc(sizeof(ListInfo)\*10);

while(op)

{

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

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

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

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

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

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

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

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

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

printf(" 13. switch list\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~13]:");

scanf("%d",&op);

switch(op){

case 1:

{

if(M.length+1>=M.listsize)

{

ListInfo\* newelem=(ListInfo\*)realloc(M.elem,(M.listsize+1)\*sizeof(ListInfo));

if(!newelem) exit(OVERFLOW);

M.elem=newelem;

M.listsize++;

}

M.elem[M.length].elem=(LinkList\*)malloc(sizeof(LinkList));

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

if(IntiaList(L)==OK)

{

printf("线性表创建成功！\n");

itoa(M.length+1,M.elem[M.length].name,10);

++(M.length);

M.cur=M.length-1;

fflush(stdin); getchar();

break;

}

else

{

free(L);

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

printf("线性表创建失败！\n");

fflush(stdin); getchar();

break;

}

}

case 2:

if(M.length==0)

{

printf("没有表可以摧毁！\n"); fflush(stdin); getchar();

break;

}

if(DestroyList(L)==OK)

{

int i;

free(L);

for(i=M.cur;i<M.length;i++)

{

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

}

M.length--;

M.cur=0;

L=M.elem[0].elem;

printf("线性表销毁成功！\n");

}

else printf("线性表销毁失败！\n");

fflush(stdin); getchar();

break;

case 3:

if(M.length==0)

{

printf("没有表可以置空！\n"); fflush(stdin); getchar();

break;

}

if(ClearList(L)==OK) printf("线性表已置空！\n");

else printf("线性表置空失败！\n");

fflush(stdin); getchar();

break;

case 4:

{

if(M.length==0)

{

printf("没有表可以判断！\n"); fflush(stdin); getchar();

break;

}

status s=ListEmpty(L);

if(s==-1) printf("线性表不存在！\n");

if(s==TRUE) printf("线性表为空表！\n");

if(s==FALSE) printf("线性表不为空！\n");

fflush(stdin); getchar();

break;

}

case 5:

{

if(M.length==0)

{

printf("没有表可以进行计算！\n"); fflush(stdin); getchar();

break;

}

int length=ListLength(L);

if(length==-1) printf("线性表不存在！\n");

else printf("线性表元素个数为%d个\n",length);

fflush(stdin); getchar();

break;

}

case 6:

{

if(M.length==0)

{

printf("没有表！\n"); fflush(stdin); getchar();

break;

}

int i;

ElemType e;

printf("第i个元素的值\n输入i:");

scanf("%d",&i); fflush(stdin);

if(GetElem(L,i,&e)==OK) printf("第%d个元素的值为%d\n",i,e);

else printf("该数据元素不存在\n");

fflush(stdin); getchar();

break;

}

case 7:

{

if(M.length==0)

{

printf("没有表！\n"); fflush(stdin); getchar();

break;

}

ElemType e;

int i;

printf("元素定位\n输入元素e:");

scanf("%d",&e); fflush(stdin);

i=LocateElem(L,e,compare);

if(i) printf("e元素位序为%d\n",i);

else printf("数据元素不存在\n");

fflush(stdin); getchar();

break;

}

case 8:

{

if(M.length==0)

{

printf("没有表！\n"); fflush(stdin); getchar();

break;

}

ElemType pre\_e,cur;

printf("元素前驱查找\n输入元素cur:");

scanf("%d",&cur); fflush(stdin);

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

else printf("操作失败！\n");

fflush(stdin); getchar();

break;

}

case 9:

{

if(M.length==0)

{

printf("没有表！\n"); fflush(stdin); getchar();

break;

}

ElemType next\_e,cur;

printf("元素后继查找:\n输入元素cur:");

scanf("%d",&cur); fflush(stdin);

if(NextElem(L,cur,&next\_e)==OK) printf("元素cur的后继元素为%d\n",next\_e);

else printf("操作失败！\n");

fflush(stdin); getchar();

break;

}

case 10:

{

if(M.length==0)

{

printf("没有表！\n"); fflush(stdin); getchar();

break;

}

int i;

ElemType e;

printf("在第i个元素之前插入元素e\n输入i:");

scanf("%d",&i); fflush(stdin);

printf("输入元素e\n");

scanf("%d",&e); fflush(stdin);

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

else printf("插入失败！\n");

fflush(stdin);getchar();

break;

}

case 11:

{

if(M.length==0)

{

printf("没有表！\n"); fflush(stdin); getchar();

break;

}

int i;

ElemType e;

printf("删除第i个元素\n输入i:");

scanf("%d",&i); fflush(stdin);

if(ListDelete(L,i,&e)==OK) printf("元素删除成功！第%d个元素值为%d\n",i,e);

else printf("删除失败！\n");

fflush(stdin); getchar();

break;

}

case 12:

{

if(M.length==0)

{

printf("没有表！\n"); fflush(stdin); getchar();

break;

}

if(!ListTraverse(L,visit)) printf("线性表是空表！\n");

else printf("\n遍历完成\n");

fflush(stdin); getchar();

break;

}

case 13:

{

int i,j;

if(M.length==0)

{

printf("当前没有线性表\n");

fflush(stdin); getchar();

break;

}

if(M.length==1)

{

printf("当前只有1个线性表\n");

printf("表%d\t名称：%s\t长度：%d\n",1,(M.elem)->name,(M.elem)->elem->length);

fflush(stdin); getchar();

break;

}

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

{

printf("表%d\t名称：%s\t长度：%d\n",i+1,(M.elem+i)->name,(M.elem+i)->elem->length);

}

printf("选择线性表[1-%d]\n",M.length);

scanf("%d",&j); fflush(stdin);

L=(M.elem+j-1)->elem;

M.cur=j-1;

printf("切换成功\n");

fflush(stdin); getchar();

break;

}

case 0:

break;

}

}

char choose;

printf("是否保存数据到文件？\ty/n:"); fflush(stdin);

scanf("%c",&choose); fflush(stdin);

printf("\n");

if(choose=='y') save(&M);

printf("欢迎下次再使用本系统！\n\n---回车键退出---\n");

getchar();

return 0;

}

operates.c:

//各具体操作与功能实现

#include "def.h"

/\*

函数名称：IntiaList

函数参数：指向单链表的指针

函数功能：初始化单链表

\*/

status IntiaList(LinkList \*L)

{

L->head = (LNode\_ptr)malloc(sizeof (LNode));

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

L->length=0;

L->head->data=-1;

L->head->next=NULL;

return OK;

}

/\*

函数名称：DestroyList

函数参数：指向单链表的指针

函数功能：摧毁单链表

\*/

status DestroyList(LinkList \*L)

{

if(!L->head) return ERROR;

LNode\_ptr cur,af;

af=L->head;

while(af!=NULL)

{

cur=af;

af=af->next;

free(cur);

}

L->head=NULL;

L->length=0;

return OK;

}

/\*

函数名称：ClearList

函数参数：指向单链表的指针

函数功能：清空单链表

\*/

status ClearList(LinkList \*L)

{

if(!L->head) return ERROR;

LNode\_ptr cur,af;

if(L->length!=0)

{

af=L->head->next;

while(af!=NULL)

{

cur=af;

af=af->next;

free(cur);

}

L->length=0;

L->head->next=NULL;

return OK;

}

else return OK;

}

/\*

函数名称：ListEmpty

函数参数：指向单链表的指针

函数功能：判断单链表是否为空

\*/

status ListEmpty(LinkList \*L)

{

if(!L->head) return -1;

if(L->length==0) return TRUE;

else return FALSE;

}

/\*

函数名称：ListLength

函数参数：指向单链表的指针

函数功能：求单链表的表长

\*/

int ListLength(LinkList \*L)

{

if(!L->head) return -1;

return L->length;

}

/\*

函数名称：GetElem

函数参数：指向单链表的指针L,元素位序i，指针e

函数功能：取单链表中的元素

\*/

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

{

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

Position p=L->head;

while(i)

{

p=p->next;

i--;

}

\*e=p->data;

return OK;

}

/\*

函数名称：compare

函数参数：元素e,f

函数功能：比较两元素是否相等

\*/

status compare(ElemType e,ElemType f)

{

return e==f?TRUE:FALSE;

}

/\*

函数名称：LocateElem

函数参数：指向单链表的指针L，目标元素e，函数指针compare

函数功能：定位单链表中的元素

\*/

status LocateElem(LinkList \*L,ElemType e,int (\*compare)(ElemType,ElemType))

{

int i=1;

LNode\_ptr p=L->head->next;

while(i<=L->length&&!(\*compare)(p->data,e))

{

++i;

p=p->next;

}

if(i<=L->length) return i;

else return 0;

}

/\*

函数名称：PriorElem

函数参数：指向单链表的指针L，目标元素cur\_e，指针pre\_e

函数功能：获取目标元素的前驱

\*/

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

{

int i=LocateElem(L,cur\_e,compare);

if(i==1||i>L->length||i==0) return ERROR;

else

{

Position cur,pre;

cur=L->head->next;

pre=L->head;

while(cur&&cur->data!=cur\_e)

{

cur=cur->next;

pre=pre->next;

}

\*pre\_e=pre->data;

return OK;

}

}

/\*

函数名称：NextElem

函数参数：指向单链表的指针L，目标元素cur\_e，指针next\_e

函数功能：获取目标元素的后继

\*/

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

{

int i=LocateElem(L,cur\_e,compare);

if(i==0||i>=L->length) return ERROR;

else

{

Position cur=L->head->next;

while(cur&&cur->data!=cur\_e)

{

cur=cur->next;

}

\*next\_e=cur->next->data;

return OK;

}

}

/\*

函数名称：ListInsert

函数参数：指向单链表的指针L，要插入元素的位置i，插入的元素e

函数功能：插入元素

\*/

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

{

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

Position pos=L->head->next;

Position pre=L->head;

while(i-1)

{

pos=pos->next;

pre=pre->next;

i--;

}

LNode\_ptr new\_node=(LNode\_ptr)malloc(sizeof(LNode));

new\_node->data=e;

new\_node->next=pos;

pre->next=new\_node;

++L->length;

return OK;

}

/\*

函数名称：ListDelete

函数参数：指向单链表的指针L，要删除元素的位序i，指针e

函数功能：删除元素

\*/

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

{

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

Position pos=L->head->next;

Position pre=L->head;

while(i-1)

{

pos=pos->next;

pre=pre->next;

i--;

}

\*e=pos->data;

pre->next=pos->next;

free(pos);

--L->length;

return OK;

}

/\*

函数名称：visit

函数参数：元素e

函数功能：输出元素e

\*/

void visit(ElemType e)

{

printf("%d\t",e);

}

/\*

函数名称：ListTraverse

函数参数：指向单链表的指针L，函数指针visit

函数功能：遍历输出单链表中所有元素的值

\*/

status ListTraverse(LinkList \*L,void (\*visit)(ElemType))

{

if(!L->head) return ERROR;

Position p=L->head->next;

while(p)

{

(\*visit)(p->data);

p=p->next;

}

return OK;

}

/\*

函数名称：save

函数参数：指向管理表信息节点的指针

函数功能：保存数据到文件

\*/

void save(ManageList\* M)

{

if(M->length==0) return;

FILE\* fp=NULL;

int i;

LinkList \*info\_node;

LNode\_ptr cur,pre;

fp=fopen("save.dat","wb");

fwrite(M,sizeof(ManageList),1,fp);

fwrite(M->elem,sizeof(ListInfo),M->listsize,fp);

for(i=0;i<M->length;i++)

{

info\_node=(M->elem+i)->elem;

fwrite(info\_node,sizeof(LinkList),1,fp);

cur=info\_node->head;

while(cur)

{

fwrite(cur,sizeof(LNode),1,fp);

pre=cur;

cur=cur->next;

free(pre);

}

free(info\_node);

}

free(M->elem);

printf("已保存\n");

}

/\*

函数名称：recover

函数参数：指向管理表信息节点的指针M，指向指向单链表的指针的指针L

函数功能：从文件读取数据恢复至内存

\*/

void recover(ManageList\* M,LinkList \*\*L)

{

FILE\* fp=NULL;

int i,j;

fp=fopen("save.dat","rb");

fread(M,sizeof(ManageList),1,fp);

ListInfo\* info=(ListInfo\*)malloc(sizeof(ListInfo)\*(M->listsize));

M->elem=info;

fread(M->elem,sizeof(ListInfo),M->listsize,fp);

for(i=0;i<M->length;i++)

{

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

fread(L,sizeof(LinkList),1,fp);

(M->elem+i)->elem=L;

LNode\_ptr cur,new\_node;

L->head=(LNode\_ptr)malloc(sizeof(LNode));

fread(L->head,sizeof(LNode),1,fp);

cur=L->head;

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

{

new\_node=(LNode\_ptr)malloc(sizeof(LNode));

fread(new\_node,sizeof(LNode),1,fp);

cur->next=new\_node;

cur=new\_node;

}

cur->next=NULL;

}

M->cur=0;

\*L=M->elem[0].elem;

printf("恢复成功,按回车继续\n"); getchar();

}

# 附录C基于二叉链表二叉树实现的源程序

def.h:

//主要定义

#ifndef DEF\_H\_INCLUDED

#define DEF\_H\_INCLUDED

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<malloc.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

#define MAX\_LENGTH 512

typedef int status;

typedef int TElemType;

typedef struct BiTNode{

TElemType index;

TElemType grade;

struct BiTNode \*lchild,\*rchild;

}BiTNode, \*Tree, \*TreeNode, \*BiTree;

typedef struct Bitreeinfo{

int ver\_num;

int depth;

struct BiTNode \*root;

}Bitreeinfo, \*Treeinfo;

typedef struct ListInfo{

char name[20];

struct Bitreeinfo\* elem;

}ListInfo;

typedef struct ManageList{

struct ListInfo\* elem;

int length;

int listsize;

int cur;

}ManageList;

status InitBiTree(Tree\* T);

status DestroyBiTree(Tree T);

TreeNode CreateBiTree(Treeinfo L);

status ClearBiTree(Tree T);

status BiTreeEmpty(Tree T);

status BiTreeDepth(Tree T);

TreeNode Root(Tree T);

TElemType Value(Tree T,int e);

status Assign(Tree T,int e,int value);

TreeNode Parent(Tree T,int e);

TreeNode LeftChild(Tree T,int e);

TreeNode RightChild(Tree T,int e);

TreeNode LeftSibling(Tree T,int e);

TreeNode RightSibling(Tree T,int e);

status InsertChild(Tree T,int index,int LR,BiTree c);

status DeleteChild(Tree T,int e,int LR);

status PreOrderTraverse(Tree T,void (\*visit)(TElemType));

status InOrderTraverse(Tree T,void (\*visit)(TElemType));

status PostOrderTraverse(Tree T,void (\*visit)(TElemType));

status LevelOrderTraverse(Tree T,void (\*visit)(TElemType));

BiTree copy\_tree(Tree T);

TreeNode Search(Tree T,int index);

int get\_ver\_num(Tree T);

void visit(TElemType e);

void save\_tree(Tree T,FILE\* fp);

void save(ManageList\* M);

TreeNode recover\_tree(FILE\* fp);

void recover(ManageList\* M,Treeinfo\* L);

#endif // DEF\_H\_INCLUDED

main.c:

//主函数

#include "def.h"

int main()

{

Treeinfo L; int op=1; char rec;

ManageList M;

M.length=0;

M.listsize=10;

M.cur=-1;

printf("是否从文件读入数据？y/n\n");

scanf("%c",&rec); fflush(stdin);

if(rec=='y') recover(&M,&L);

else M.elem=(ListInfo\*)malloc(sizeof(ListInfo)\*10);

while(op)

{

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

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

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

printf(" 1. InitBiTree 11. LeftChild\n");

printf(" 2. DestroyBiTree 12. RightChild\n");

printf(" 3. CreateBiTree 13. LeftSibling\n");

printf(" 4. ClearBiTree 14. RightSibling\n");

printf(" 5. BiTreeEmpty 15. InsertChild\n");

printf(" 6. BiTreeDepth 16. DeleteChild\n");

printf(" 7. Root 17. PreOrderTraverse\n");

printf(" 8. Value 18. InOrderTraverse\n");

printf(" 9. Assign 19. PostOrderTraverse\n");

printf(" 10. Parent 20. LevelOrderTraverse\n");

printf(" 21. switch Tree\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~21]:");

scanf("%d",&op);

switch(op){

case 1:

{

if(M.length+1>=M.listsize)

{

ListInfo\* newelem=(ListInfo\*)realloc(M.elem,(M.listsize+1)\*sizeof(ListInfo));

if(!newelem) exit(OVERFLOW);

M.elem=newelem;

M.listsize++;

}

M.elem[M.length].elem=(Bitreeinfo\*)malloc(sizeof(Bitreeinfo));

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

if(InitBiTree(&(L->root))==OK)

{

printf("二叉树初始化成功！\n");

itoa(M.length+1,M.elem[M.length].name,10);

++(M.length);

M.cur=M.length-1;

L->depth=0;

L->ver\_num=0;

fflush(stdin); getchar();

break;

}

else

{

free(L);

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

printf("二叉树创建失败！\n");

fflush(stdin); getchar();

break;

}

}

case 2:

if(M.length==0)

{

printf("没有二叉树可以摧毁！\n"); fflush(stdin); getchar();

break;

}

if(L->root)

{

if(DestroyBiTree(L->root)==OK)

{

int i;

free(L);

for(i=M.cur;i<M.length;i++)

{

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

}

M.length--;

M.cur=0;

L=M.elem[0].elem;

printf("二叉树销毁成功！\n");

}

else printf("二叉树销毁失败！\n");

}

else

{

int i;

free(L);

for(i=M.cur;i<M.length;i++)

{

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

}

M.length--;

M.cur=0;

L=M.elem[0].elem;

printf("二叉树销毁成功！\n");

}

fflush(stdin); getchar();

break;

case 3:

{

Tree root;

if(M.length==0)

{

printf("当前没有已初始化的二叉树！\n"); fflush(stdin); getchar();

break;

}

fflush(stdin);

printf("请输入先序序列：");

root=CreateBiTree(L);

if(root)

{

L->depth=BiTreeDepth(L->root);

L->root=root;

printf("二叉树创建成功\n");

}

else printf("二叉树创建失败\n");

fflush(stdin); getchar();

break;

}

case 4:

if(M.length==0)

{

printf("没有二叉树可以置空！\n"); fflush(stdin); getchar();

break;

}

if(ClearBiTree(L->root)==OK)

{

L->depth=0;

L->ver\_num=0;

L->root=NULL;

printf("二叉树已置空！\n");

}

else printf("二叉树置空失败！\n");

fflush(stdin); getchar();

break;

case 5:

{

if(M.length==0)

{

printf("没有二叉树可以判断！\n"); fflush(stdin); getchar();

break;

}

status s=BiTreeEmpty(L->root);

if(s==TRUE) printf("二叉树为空！\n");

if(s==FALSE) printf("二叉树不为空！\n");

fflush(stdin); getchar();

break;

}

case 6:

{

if(M.length==0)

{

printf("没有二叉树可以进行计算！\n"); fflush(stdin); getchar();

break;

}

int depth=BiTreeDepth(L->root);

printf("二叉树深度为%d\n",depth);

fflush(stdin); getchar();

break;

}

case 7:

{

if(M.length==0)

{

printf("没有二叉树！\n"); fflush(stdin); getchar();

break;

}

TreeNode r=Root(L->root);

if(r==ERROR) printf("二叉树还没有节点\n");

else printf("根节点找到，标识为%d,数据为%d\n",r->index,r->grade);

fflush(stdin); getchar();

break;

}

case 8:

{

if(M.length==0)

{

printf("没有二叉树！\n"); fflush(stdin); getchar();

break;

}

int e;

TElemType v;

printf("元素定位\n输入节点标识:");

scanf("%d",&e); fflush(stdin);

v=Value(L->root,e);

if(v) printf("节点值为%d\n",v);

else printf("该节点不存在\n");

fflush(stdin); getchar();

break;

}

case 9:

{

if(M.length==0)

{

printf("没有二叉树！\n"); fflush(stdin); getchar();

break;

}

TElemType index,value;

printf("给节点赋值：\n输入节点标识:");

scanf("%d",&index); fflush(stdin);

printf("输入节点值\n");

scanf("%d",&value); fflush(stdin);

if(Assign(L->root,index,value)) printf("赋值成功\n");

else printf("操作失败！\n");

fflush(stdin); getchar();

break;

}

case 10:

{

if(M.length==0)

{

printf("没有二叉树！\n"); fflush(stdin); getchar();

break;

}

TElemType index;

TreeNode parent;

printf("查找父节点:\n输入节点标识:");

scanf("%d",&index); fflush(stdin);

parent=Parent(L->root,index);

if(parent) printf("父节点找到，标识为%d，值为%d\n",parent->index,parent->grade);

else printf("操作失败！\n");

fflush(stdin); getchar();

break;

}

case 11:

{

if(M.length==0)

{

printf("没有二叉树！\n"); fflush(stdin); getchar();

break;

}

TElemType index;

TreeNode leftc;

printf("查找左孩子:\n输入节点标识:");

scanf("%d",&index); fflush(stdin);

leftc=LeftChild(L->root,index);

if(leftc) printf("左孩子标识为%d，值为%d\n",leftc->index,leftc->grade);

else printf("查找失败！\n");

fflush(stdin);getchar();

break;

}

case 12:

{

if(M.length==0)

{

printf("没有二叉树！\n"); fflush(stdin); getchar();

break;

}

TElemType index;

TreeNode rightc;

printf("查找右孩子:\n输入节点标识:");

scanf("%d",&index); fflush(stdin);

rightc=RightChild(L->root,index);

if(rightc) printf("右孩子标识为%d，值为%d\n",rightc->index,rightc->grade);

else printf("查找失败！\n");

fflush(stdin);getchar();

break;

}

case 13:

{

if(M.length==0)

{

printf("没有二叉树！\n"); fflush(stdin); getchar();

break;

}

TElemType index;

TreeNode lefts;

printf("查找左兄弟:\n输入节点标识:");

scanf("%d",&index); fflush(stdin);

lefts=LeftSibling(L->root,index);

if(lefts) printf("左兄弟标识为%d，值为%d\n",lefts->index,lefts->grade);

else printf("查找失败！\n");

fflush(stdin);getchar();

break;

}

case 14:

{

if(M.length==0)

{

printf("没有二叉树！\n"); fflush(stdin); getchar();

break;

}

TElemType index;

TreeNode rights;

printf("查找右兄弟:\n输入节点标识:");

scanf("%d",&index); fflush(stdin);

rights=RightSibling(L->root,index);

if(rights) printf("右兄弟标识为%d，值为%d\n",rights->index,rights->grade);

else printf("查找失败！\n");

fflush(stdin);getchar();

break;

}

case 15:

{

int LR;

int index;

if(M.length==0)

{

printf("没有二叉树！\n"); fflush(stdin); getchar();

break;

}

if(M.length==1)

{

printf("当前只有一棵树，插入其自身\n");

printf("输入插入节点标识\n"); scanf("%d",&index); fflush(stdin);

printf("选择插入为该节点的左子树（0）或右子树（1）\n"); scanf("%d",&LR); fflush(stdin);

if(InsertChild(L->root,index,LR,L->root)==OK)

{

printf("插入成功\n");

L->depth=BiTreeDepth(L->root);

(L->ver\_num)\*=2;

}

else printf("插入失败\n");

}

else

{

printf("当前多于一棵树，插入第一棵树\n");

printf("输入插入节点标识\n"); scanf("%d",&index); fflush(stdin);

printf("选择插入为该节点的左子树（0）或右子树（1）\n"); scanf("%d",&LR); fflush(stdin);

if(InsertChild(L->root,index,LR,M.elem->elem->root)==OK)

{

printf("插入成功\n");

L->depth=BiTreeDepth(L->root);

L->ver\_num=get\_ver\_num(L->root);

}

else printf("插入失败\n");

}

fflush(stdin);getchar();

break;

}

case 16:

{

int index,LR;

if(M.length==0)

{

printf("没有二叉树！\n"); fflush(stdin); getchar();

break;

}

printf("输入要操作的结点的标识\n");

scanf("%d",&index); fflush(stdin);

printf("选择左子树（0）或右子树（1）\n");

scanf("%d",&LR); fflush(stdin);

if(DeleteChild(L->root,index,LR)==OK)

{

L->depth=BiTreeDepth(L->root);

L->ver\_num=get\_ver\_num(L->root);

printf("%s\n",LR?"右子树已删除":"左子树已删除");

}

else printf("操作失败\n");

fflush(stdin);getchar();

break;

}

case 17:

{

if(M.length==0)

{

printf("没有二叉树！\n"); fflush(stdin); getchar();

break;

}

printf("先序遍历\n");

if(PreOrderTraverse(L->root,visit)==OK) printf("遍历完成\n");

else printf("操作失败\n");

fflush(stdin);getchar();

break;

}

case 18:

{

if(M.length==0)

{

printf("没有二叉树！\n"); fflush(stdin); getchar();

break;

}

printf("中序遍历\n");

if(InOrderTraverse(L->root,visit)==OK) printf("遍历完成\n");

else printf("操作失败\n");

fflush(stdin);getchar();

break;

}

case 19:

{

if(M.length==0)

{

printf("没有二叉树！\n"); fflush(stdin); getchar();

break;

}

printf("后序遍历\n");

if(PostOrderTraverse(L->root,visit)==OK) printf("遍历完成\n");

else printf("操作失败\n");

fflush(stdin);getchar();

break;

}

case 20:

{

if(M.length==0)

{

printf("没有二叉树！\n"); fflush(stdin); getchar();

break;

}

printf("层序遍历\n");

if(LevelOrderTraverse(L->root,visit)==OK) printf("遍历完成\n");

else printf("操作失败\n");

fflush(stdin);getchar();

break;

}

case 21:

{

int i,j;

if(M.length==0)

{

printf("当前没有二叉树\n");

fflush(stdin); getchar();

break;

}

if(M.length==1)

{

printf("当前只有1棵二叉树\n");

printf("二叉树%d\t名称：%s\t节点数：%d\n",1,(M.elem)->name,(M.elem)->elem->ver\_num);

fflush(stdin); getchar();

break;

}

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

{

printf("二叉树%d\t名称：%s\t节点数：%d\n",i+1,(M.elem+i)->name,(M.elem+i)->elem->ver\_num);

}

printf("选择二叉树[1-%d]\n",M.length);

scanf("%d",&j); fflush(stdin);

L=(M.elem+j-1)->elem;

M.cur=j-1;

printf("切换成功\n");

fflush(stdin); getchar();

break;

}

case 0:

break;

}

}

char choose;

printf("是否保存数据到文件？\ty/n:"); fflush(stdin);

scanf("%c",&choose); fflush(stdin);

printf("\n");

if(choose=='y') save(&M);

printf("欢迎下次再使用本系统！\n\n---回车键退出---\n");

getchar();

return 0;

}

operates.c:

//各具体操作与功能实现

#include "def.h"

/\*初始化二叉树\*/

status InitBiTree(Tree\* T)

{

\*T=NULL;

return OK;

}

/\*销毁二叉树\*/

status DestroyBiTree(Tree T)

{

if(T)

{

if(T->lchild)

{

DestroyBiTree(T->lchild);

T->lchild=NULL;

}

if(T->rchild)

{

DestroyBiTree(T->rchild);

T->rchild=NULL;

}

if(T)

{

free(T);

T=NULL;

}

return OK;

}

else return ERROR;

}

/\*创建二叉树\*/

TreeNode CreateBiTree(Treeinfo L)

{

char content[7];

char content\_part[6];

char s\_index[4];

char s\_grade[4];

TreeNode T;

int i;

char c=getchar();

if (c=='#'||c=='\n')

{

return NULL;

}

content[0]=c;

scanf("%5s",content\_part);

for(i=1;i<=5;i++)

{

content[i]=content\_part[i-1];

}

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

if(!T) exit(OVERFLOW);

sscanf(content,"%3s",s\_index);

sscanf(content+3,"%3s",s\_grade);

T->index=atoi(s\_index);

T->grade=atoi(s\_grade);

(L->ver\_num)++;

T->lchild=CreateBiTree(L);

T->rchild=CreateBiTree(L);

return T;

}

/\*清空二叉树\*/

status ClearBiTree(Tree T)

{

if(T)

{

if(T->lchild)

{

ClearBiTree(T->lchild);

T->lchild=NULL;

}

if(T->rchild)

{

ClearBiTree(T->rchild);

T->rchild=NULL;

}

if(T)

{

free(T);

T=NULL;

}

return OK;

}

else return ERROR;

}

/\*判断二叉树是否为空\*/

status BiTreeEmpty(Tree T)

{

if(T) return FALSE;

else return TRUE;

}

/\*求二叉树深度\*/

status BiTreeDepth(Tree T)

{

if(!T)

{

return 0;

}

int nLeft=BiTreeDepth(T->lchild);

int nRight=BiTreeDepth(T->rchild);

return nLeft>nRight?nLeft+1:nRight+1;

}

/\*求二叉树的根\*/

TreeNode Root(Tree T)

{

if(!T)

{

return NULL;

}

return T;

}

/\*求节点值\*/

TElemType Value(Tree T,int e)

{

BiTree st[MAX\_LENGTH+1]; //定义指针栈st[1..maxleng]

int top=0; //置空栈

if(!T) return ERROR;

do

{

while(T)

{

if(top==MAX\_LENGTH) exit(OVERFLOW);

st[++top]=T;

T=T->lchild;

}

if(top) {

T=st[top--];

if(T->index==e) return T->grade;

T=T->rchild;

}

} while(top||T);

return INFEASIBLE;

}

/\*节点赋值\*/

status Assign(Tree T,int e,int value)

{

BiTree st[MAX\_LENGTH+1];

int top=0;

if(!T) return ERROR;

do

{

while(T)

{

if(top==MAX\_LENGTH) exit(OVERFLOW);

st[++top]=T;

T=T->lchild;

}

if(top)

{

T=st[top--];

if(T->index==e)

{

T->grade=value;

return OK;

}

T=T->rchild;

}

} while(top||T);

return INFEASIBLE;

}

/\*求双亲节点\*/

TreeNode Parent(Tree T,int e)

{

BiTree st[MAX\_LENGTH+1];

int tag[MAX\_LENGTH+1];

int top=0;

if(!T) return NULL;

while(T||top)

{

while(T)

{

if(top==MAX\_LENGTH) exit(OVERFLOW);

st[++top]=T;

tag[top]=0;

T=T->lchild;

}

if(tag[top]==0)

{

T=st[top];

tag[top]=1;

T=T->rchild;

}

else

{

while(tag[top]==1)

{

T=st[top--];

if(T->index==e&&top>0) return st[top];

}

T=NULL;

}

}

return NULL;

}

/\*求左孩子\*/

TreeNode LeftChild(Tree T,int e)

{

BiTree st[MAX\_LENGTH+1];

int top=0;

if(!T) return ERROR;

do

{

while(T)

{

if(top==MAX\_LENGTH) exit(OVERFLOW);

st[++top]=T;

T=T->lchild;

}

if(top)

{

T=st[top--];

if(T->index==e) return T->lchild;

T=T->rchild;

}

} while(top||T);

return NULL;

}

/\*求右孩子\*/

TreeNode RightChild(Tree T,int e)

{

BiTree st[MAX\_LENGTH+1];

int top=0;

if(!T) return ERROR;

do

{

while(T)

{

if(top==MAX\_LENGTH) exit(OVERFLOW);

st[++top]=T;

T=T->lchild;

}

if(top)

{

T=st[top--];

if(T->index==e) return T->rchild;

T=T->rchild;

}

} while(top||T);

return NULL;

}

/\*求左兄弟\*/

TreeNode LeftSibling(Tree T,int e)

{

TreeNode parent=Parent(T,e);

TreeNode cur=Search(T,e);

if(parent)

{

if(parent->lchild==cur||parent->lchild==NULL) return ERROR;

else return parent->lchild;

}

else return ERROR;

}

/\*求右兄弟\*/

TreeNode RightSibling(Tree T,int e)

{

TreeNode parent=Parent(T,e);

TreeNode cur=Search(T,e);

if(parent)

{

if(parent->rchild==cur||parent->rchild==NULL) return ERROR;

else return parent->rchild;

}

else return ERROR;

}

/\*复制二叉树\*/

BiTree copy\_tree(Tree T)

{

TreeNode newNode=NULL;

TreeNode newLp=NULL;

TreeNode newRp=NULL;

if(!T) return NULL;

if(T->lchild)

{

newLp=copy\_tree(T->lchild);

}

else

{

newLp=NULL;

}

if(T->rchild)

{

newRp=copy\_tree(T->rchild);

}

else

{

newRp=NULL;

}

newNode=(TreeNode)malloc(sizeof(BiTNode));

if(!newNode) return NULL;

newNode->lchild=newLp;

newNode->rchild=newRp;

newNode->index=T->index;

newNode->grade=T->grade;

return newNode;

}

/\*搜索节点\*/

TreeNode Search(Tree T,int index)

{

BiTree st[MAX\_LENGTH+1];

int top=0;

if(!T) return ERROR;

do

{

while(T)

{

if(top==MAX\_LENGTH) exit(OVERFLOW);

st[++top]=T;

T=T->lchild;

}

if(top)

{

T=st[top--];

if(T->index==index) return T;

T=T->rchild;

}

} while(top||T);

return NULL;

}

/\*插入子树\*/

status InsertChild(Tree T,int index,int LR,BiTree c)

{

if(!T) return ERROR;

Tree insertion=NULL;

Tree origin=NULL;

insertion=copy\_tree(c);

if(insertion->rchild)

{

DestroyBiTree(insertion->rchild);

insertion->rchild=NULL;

}

TreeNode p=Search(T,index);

if(p)

{

if(!LR)

{

origin=p->lchild;

p->lchild=insertion;

insertion->rchild=origin;

}

else

{

origin=p->rchild;

p->rchild=insertion;

insertion->rchild=origin;

}

return OK;

}

else return ERROR;

}

/\*删除子树\*/

status DeleteChild(Tree T,int e,int LR)

{

if(!T) return ERROR;

TreeNode p=Search(T,e);

if(p)

{

Tree left=p->lchild;

Tree right=p->rchild;

if(!LR)

{

DestroyBiTree(left);

p->lchild=NULL;

}

else

{

DestroyBiTree(right);

p->rchild=NULL;

}

return OK;

}

else return ERROR;

}

/\*访问节点\*/

void visit(TElemType e)

{

printf("%d\t",e);

}

/\*先序遍历\*/

status PreOrderTraverse(Tree T,void (\*visit)(TElemType))

{

BiTree st[MAX\_LENGTH+1];

int top=0;

if(!T) return ERROR;

while(T||top)

{

while(T)

{

if(top==MAX\_LENGTH) exit(OVERFLOW);

visit(T->grade);

st[++top]=T;

T=T->lchild;

}

T=st[top--];

T=T->rchild;

}

return OK;

}

/\*中序遍历\*/

status InOrderTraverse(Tree T,void (\*visit)(TElemType))

{

BiTree st[MAX\_LENGTH+1];

int top=0;

if(!T) return ERROR;

do

{

while(T)

{

if(top==MAX\_LENGTH) exit(OVERFLOW);

st[++top]=T;

T=T->lchild;

}

if(top)

{

T=st[top--];

visit(T->grade);

T=T->rchild;

}

} while(top||T);

return OK;

}

/\*后序遍历\*/

status PostOrderTraverse(Tree T,void (\*visit)(TElemType))

{

BiTree st[MAX\_LENGTH+1];

int tag[MAX\_LENGTH+1];

int top=0;

if(!T) return ERROR;

while(T||top)

{

while(T)

{

if(top==MAX\_LENGTH) exit(OVERFLOW);

st[++top]=T;

tag[top]=0;

T=T->lchild;

}

if(tag[top]==0)

{

T=st[top];

tag[top]=1;

T=T->rchild;

}

else

{

while(tag[top]==1)

{

T=st[top--];

visit(T->grade);

}

T=NULL;

}

}

return OK;

}

/\*层序遍历\*/

status LevelOrderTraverse(Tree T,void (\*visit)(TElemType))

{

BiTree q[2\*MAX\_LENGTH+1];

BiTree p;

int top=0;

int tail=0;

if(T)

{

q[++tail]=T;

while(top!=tail)

{

p=q[++top];

visit(p->grade);

if(p->lchild) q[++tail]=p->lchild;

if(p->rchild) q[++tail]=p->rchild;

}

return OK;

}

else return ERROR;

}

/\*保存树\*/

void save\_tree(Tree T,FILE\* fp)

{

TreeNode empty=(TreeNode)malloc(sizeof(BiTNode));

empty->index=-1;

if(T)

{

fwrite(T,sizeof(BiTNode),1,fp);

}

else

{

fwrite(empty,sizeof(BiTNode),1,fp);

}

if(T)

{

save\_tree(T->lchild,fp);

save\_tree(T->rchild,fp);

}

}

/\*求二叉树节点数\*/

int get\_ver\_num(Tree T)

{

BiTree st[MAX\_LENGTH+1];

int top=0;

int num=0;

if(!T) return ERROR;

while(T||top)

{

while(T)

{

if(top==MAX\_LENGTH) exit(OVERFLOW);

num++;

st[++top]=T;

T=T->lchild;

}

T=st[top--];

T=T->rchild;

}

return num;

}

/\*保存数据到文件\*/

void save(ManageList\* M)

{

if(M->length==0) return;

FILE\* fp=NULL;

int i;

Treeinfo info\_node;

TreeNode root;

fp=fopen("save.dat","wb");

fwrite(M,sizeof(ManageList),1,fp);

fwrite(M->elem,sizeof(ListInfo),M->listsize,fp);

for(i=0;i<M->length;i++)

{

info\_node=(M->elem+i)->elem;

fwrite(info\_node,sizeof(Bitreeinfo),1,fp);

root=info\_node->root;

save\_tree(root,fp);

DestroyBiTree(root);

free(info\_node);

}

free(M->elem);

printf("已保存\n");

}

/\*恢复二叉树\*/

TreeNode recover\_tree(FILE\* fp)

{

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

if(!T) exit(OVERFLOW);

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

if(T->index>0)

{

T->lchild=recover\_tree(fp);

T->rchild=recover\_tree(fp);

return T;

}

else

{

free(T);

return NULL;

}

}

/\*从文件恢复数据\*/

void recover(ManageList\* M,Treeinfo\* L)

{

FILE\* fp=NULL;

int i;

fp=fopen("save.dat","rb");

fread(M,sizeof(ManageList),1,fp);

ListInfo\* info=(ListInfo\*)malloc(sizeof(ListInfo)\*(M->listsize));

M->elem=info;

fread(M->elem,sizeof(ListInfo),M->listsize,fp);

for(i=0;i<M->length;i++)

{

Treeinfo LL=(Treeinfo)malloc(sizeof(Bitreeinfo));

fread(LL,sizeof(Bitreeinfo),1,fp);

(M->elem+i)->elem=LL;

LL->root=recover\_tree(fp);

}

M->cur=0;

\*L=M->elem[0].elem;

printf("恢复成功,按回车继续\n"); getchar();

}

# 附录D 基于邻接表图实现的源程序

def.h:

//主要定义

#ifndef DEF\_H\_INCLUDED

#define DEF\_H\_INCLUDED

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<malloc.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

//#define MAX\_LENGTH 512

#define MAX\_VERTEX\_NUM 20

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

typedef int status;

typedef char Vertex;

typedef int Arcinfo;

typedef struct ArcNode{//表结点结构类型

int adjvex; //该弧(边)的终点位置

struct ArcNode\* nextarc; //指向下一条弧的指针

Arcinfo weight; //该弧的相关信息的指针

} ArcNode, \*p\_ArcNode;

typedef struct Vnode {//头结点的类型

Vertex data; //顶点信息

struct ArcNode \*firstarc; //指向第一条弧

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

typedef struct ALGraph{//邻接表

AdjList vertices;

int vexnum, arcnum; //图中顶点数n和边数e

GraphKind kind; //图的类型

} ALGraph, \*GraphPtr, \*Graph;

typedef struct ListInfo{

char name[20];

struct ALGraph\* elem;

}ListInfo;

typedef struct ManageList{

struct ListInfo\* elem;

int length;

int listsize;

int cur;

}ManageList;

typedef struct qu{

struct qu\* pre;

int v;

struct qu\* next;

}qu, \*que;

Graph CreateGraph(Graph g);

status DestroyGraph(Graph g);

int LocateVex(Graph g,Vertex u);

Vertex GetVex(Graph g,int v);

status PutVex(Graph g,int v,Vertex value);

int FirstAdjVex(Graph g,int v);

int NextAdjVex(Graph g,int v, int w);

status InsertVex(Graph g,Vertex v);

status DeleteVex(Graph g,int v);

status InsertArc(Graph g,int v,int w);

status DeleteArc(Graph g,int v,int w);

status DFSTraverse(Graph g,void (\*visit)(Vertex));

void DFS(Graph g,int v0);

status BFSTraverse(Graph g,void (\*visit)(Vertex));

void visit(Vertex e);

//status compare(Vertex e,Vertex f);

void save(ManageList\* M);

void recover(ManageList\* M,Graph\* g);

#endif // DEF\_H\_INCLUDED

main.c:

//主函数

#include "def.h"

int main()

{

Graph g; int op=1; char rec;

ManageList M;

M.length=0;

M.listsize=10;

M.cur=-1;

printf("是否从文件读入数据？y/n\n");

scanf("%c",&rec); fflush(stdin);

if(rec=='y') recover(&M,&g);

else M.elem=(ListInfo\*)malloc(sizeof(ListInfo)\*10);

while(op)

{

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

printf(" Menu for Graph On Adjacency List \n");

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

printf(" 1. CreateGraph 11. DeleteArc \n");

printf(" 2. DestroyGraph 12. DFSTraverse\n");

printf(" 3. LocateVex 13. BFSTraverse\n");

printf(" 4. GetVex \n");

printf(" 5. PutVex \n");

printf(" 6. FirstAdjVex \n");

printf(" 7. NextAdjVex \n");

printf(" 8. InsertVex \n");

printf(" 9. DeleteVex \n");

printf(" 10. InsertArc \n");

printf(" 14. switch Graph\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~14]:");

scanf("%d",&op);

switch(op){

case 1:

{

if(M.length+1>=M.listsize)

{

ListInfo\* newelem=(ListInfo\*)realloc(M.elem,(M.listsize+1)\*sizeof(ListInfo));

if(!newelem) exit(OVERFLOW);

M.elem=newelem;

M.listsize++;

}

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

g=M.elem[M.length].elem;

g=CreateGraph(g);

if(g)

{

printf("create success！\n");

itoa(M.length+1,M.elem[M.length].name,10);

M.elem[M.length].elem=g;

++(M.length);

// printf("%d\n",M.length);

M.cur=M.length-1;

fflush(stdin); getchar();

break;

}

else

{

free(g);

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

printf("fail to create graph！\n");

fflush(stdin); getchar();

break;

}

}

case 2:

if(M.length==0)

{

printf("no graph to destroy！\n"); fflush(stdin); getchar();

break;

}

if(g)

{

if(DestroyGraph(g)==OK)

{

int i;

for(i=M.cur;i<M.length;i++)

{

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

}

M.length--;

M.cur=0;

g=M.elem[0].elem;

printf("destroy success！\n");

}

else printf("destroy failed！\n");

}

// else

//{

// int i;

//free(L);

//for(i=M.cur;i<M.length;i++)

// {

// M.elem[i]=M.elem[i+1];

//}

//M.length--;

//M.cur=0;

//L=M.elem[0].elem;

// printf("二叉树销毁成功！\n");

//}

fflush(stdin); getchar();

break;

case 3:

{

if(M.length==0)

{

printf("no created graph！\n"); fflush(stdin); getchar();

break;

}

Vertex value;

int i;

printf("Locate\n input the value of the vertex：");

fflush(stdin);

scanf("%c",&value); fflush(stdin);

i=LocateVex(g,value);

if(i+1)

{

printf("the location is %d\n",i);

}

else printf("fail to find the vertex\n");

fflush(stdin); getchar();

break;

}

case 4:

if(M.length==0)

{

printf("no created graph！\n"); fflush(stdin); getchar();

break;

}

Vertex value;

int i;

printf("get the value\n input the number of the vertex：");

scanf("%d",&i); fflush(stdin);

value=GetVex(g,i);

if(value!='#')

{

printf("the value is %c！\n",value);

}

else printf("no such vertex！\n");

fflush(stdin); getchar();

break;

case 5:

{

if(M.length==0)

{

printf("no created graph！\n"); fflush(stdin); getchar();

break;

}

Vertex value;

int i;

printf("assign\n input the number and the value of the vertex：");

scanf("%d",&i); fflush(stdin);

scanf("%c",&value); fflush(stdin);

if(PutVex(g,i,value)==OK)

{

printf("done\n");

}

else printf("fail to find the vertex\n");

fflush(stdin); getchar();

break;

}

case 6:

{

if(M.length==0)

{

printf("no created graph！\n"); fflush(stdin); getchar();

break;

}

int v,u;

printf("find the first adjvex\ninput the number of the vertex:");

scanf("%d",&v); fflush(stdin);

u=FirstAdjVex(g,v);

if(u==-2) printf("no such vertex\n");

if(u==-1) printf("the vertex has no adjvex\n");

if(u>=0) printf("the location of the first adjvex is %d\n",u);

fflush(stdin); getchar();

break;

}

case 7:

{

if(M.length==0)

{

printf("no created graph！\n"); fflush(stdin); getchar();

break;

}

int v,w,u;

printf("find the next adjvex\ninput the number of the vertex and the last vertex:");

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

u=NextAdjVex(g,v,w);

if(u>=0) printf("the location of the next adjvex is %d\n",u);

else printf("failed\n");

fflush(stdin); getchar();

break;

}

case 8:

{

if(M.length==0)

{

printf("no created graph！\n"); fflush(stdin); getchar();

break;

}

Vertex value;

printf("insert a vertex\ninput the value of the vertex:");

fflush(stdin);

scanf("%c",&value); fflush(stdin);

if(InsertVex(g,value)==OK)

{

printf("insertion done\n");

}

else printf("failed\n");

fflush(stdin); getchar();

break;

}

case 9:

{

if(M.length==0)

{

printf("no created graph！\n"); fflush(stdin); getchar();

break;

}

int v;

printf("delete a vertex\ninput the number of the vertex:");

fflush(stdin);

scanf("%d",&v); fflush(stdin);

if(DeleteVex(g,v)==OK)

{

printf("deleted\n");

}

else printf("failed\n");

fflush(stdin); getchar();

break;

}

case 10:

{

if(M.length==0)

{

printf("no created graph！\n"); fflush(stdin); getchar();

break;

}

int v,w,result;

printf("insert an arc\ninput the number of the tail and the head:");

fflush(stdin);

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

result=InsertArc(g,v,w);

if(result==OK)

{

printf("insertion done\n");

}

else if(result==ERROR) printf("no such vertex\n");

else printf("the arc is already there\n");

fflush(stdin); getchar();

break;

}

case 11:

{

if(M.length==0)

{

printf("no created graph！\n"); fflush(stdin); getchar();

break;

}

int v,w,result;

printf("delete an arc\ninput the number of the tail and the head:");

fflush(stdin);

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

result=DeleteArc(g,v,w);

if(result==OK)

{

printf("deleted\n");

}

if(result==ERROR) printf("failed\n");

fflush(stdin); getchar();

break;

}

case 12:

{

if(M.length==0)

{

printf("no created graph！\n"); fflush(stdin); getchar();

break;

}

printf("DFS\n");

if(DFSTraverse(g,visit)==OK)

{

printf("done\n");

}

else printf("failed\n");

fflush(stdin); getchar();

break;

}

case 13:

{

if(M.length==0)

{

printf("no created graph！\n"); fflush(stdin); getchar();

break;

}

printf("BFS\n");

if(BFSTraverse(g,visit)==OK)

{

printf("done\n");

}

else printf("failed\n");

fflush(stdin); getchar();

break;

}

case 14:

{

int i,j,type;

if(M.length==0)

{

printf("no created graph\n");

fflush(stdin); getchar();

break;

}

if(M.length==1)

{

printf("there is only one graph\n");

printf("Graph%d\tname：%s\tvexnum：%d arcnum:%d\t",1,(M.elem)->name,(M.elem)->elem->vexnum,(M.elem)->elem->arcnum);

type=(int)(M.elem)->elem->kind;

printf("Type:%s\n",type==0?"DG":(type==1?"DN":(type==2?"UDG":"UDN")));

fflush(stdin); getchar();

break;

}

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

{

printf("Graph%d\tname：%s\tvexnum：%d arcnum:%d\t",i+1,(M.elem+i)->name,(M.elem+i)->elem->vexnum,(M.elem+i)->elem->arcnum);

type=(int)(M.elem+i)->elem->kind;

printf("Type:%s\n",type==0?"DG":(type==1?"DN":(type==2?"UDG":"UDN")));

}

printf("choose graph[1-%d]\n",M.length);

scanf("%d",&j); fflush(stdin);

g=(M.elem+j-1)->elem;

M.cur=j-1;

printf("切换成功\n");

fflush(stdin); getchar();

break;

}

case 0:

break;

}

}

char choose;

printf("是否保存数据到文件？\ty/n:"); fflush(stdin);

scanf("%c",&choose); fflush(stdin);

printf("\n");

if(choose=='y') save(&M);

printf("欢迎下次再使用本系统！\n\n---回车键退出---\n");

getchar();

return 0;

}

operates.c:

//各具体操作与功能实现

#include "def.h"

int visited[MAX\_VERTEX\_NUM];

Graph CreateGraph(Graph g)

{

int type;

int i,k;

int h\_local,t\_local,location1,location2;

Vertex tail,head,vertex1,vertex2;

Arcinfo weight;

p\_ArcNode arc\_now=NULL;

p\_ArcNode arc\_now1=NULL;

p\_ArcNode arc\_now2=NULL;

p\_ArcNode arc\_end[MAX\_VERTEX\_NUM]={NULL};

g=(Graph)malloc(sizeof(ALGraph)); if(!g) exit(OVERFLOW);

printf("input the type of the graph,0 as DG,1 as DN,2 as UDG,3 as UDN\n");

scanf("%d",&type); fflush(stdin); g->kind=(GraphKind)type;

printf("input the number of vertex(no more than 20) and arc\n");

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

printf("now input the value of all the vertex\n");

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

{

printf("vertex %d:\n",i);

scanf("%c",&((g->vertices[i]).data)); fflush(stdin);

(g->vertices[i]).firstarc=NULL;

}

//printf("%d\n",g->vexnum);

//DG

if(g->kind==DG)

{

printf("The graph is a DG, input the arcs with tails and heads\n");

for(k=0;k<g->arcnum;k++)

{

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

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

h\_local=LocateVex(g,head);

t\_local=LocateVex(g,tail);

// printf("%c%c%c\n%c%c\n",g->vertices[0].data,g->vertices[1].data,g->vertices[2].data,head,tail);

// printf("%d%d\n",h\_local,t\_local);

if(h\_local==-1||t\_local==-1)

{

printf("no such vertex.\n");

k--;

continue;

}

arc\_now=(p\_ArcNode)malloc(sizeof(ArcNode)); if(!arc\_now) exit(OVERFLOW);

if(g->vertices[t\_local].firstarc==NULL)

{

g->vertices[t\_local].firstarc=arc\_now;

arc\_now->adjvex=h\_local;

arc\_now->weight=-1;

arc\_now->nextarc=NULL;

arc\_end[t\_local]=arc\_now;

}

else

{

arc\_end[t\_local]->nextarc=arc\_now;

arc\_now->adjvex=h\_local;

arc\_now->weight=-1;

arc\_now->nextarc=NULL;

arc\_end[t\_local]=arc\_now;

}

printf("%d arcs done.\n",k+1);

}

printf("%d arcs inputed\n",g->arcnum);

}

//DN

if(g->kind==DN)

{

printf("The graph is a DN, input the arcs with tails, heads and weight\n");

for(k=0;k<g->arcnum;k++)

{

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

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

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

h\_local=LocateVex(g,head);

t\_local=LocateVex(g,tail);

if(h\_local==-1||t\_local==-1)

{

printf("no such vertex.\n");

k--;

continue;

}

arc\_now=(p\_ArcNode)malloc(sizeof(ArcNode)); if(!arc\_now) exit(OVERFLOW);

if(g->vertices[t\_local].firstarc==NULL)

{

g->vertices[t\_local].firstarc=arc\_now;

arc\_now->adjvex=h\_local;

arc\_now->weight=weight;

arc\_now->nextarc=NULL;

arc\_end[t\_local]=arc\_now;

}

else

{

arc\_end[t\_local]->nextarc=arc\_now;

arc\_now->adjvex=h\_local;

arc\_now->weight=weight;

arc\_now->nextarc=NULL;

arc\_end[t\_local]=arc\_now;

}

printf("%d arcs done.\n",k+1);

}

printf("%d arcs inputed\n",g->arcnum);

}

//UDG

if(g->kind==UDG)

{

printf("The graph is a UDG, input the arcs with two linked vertexes\n");

for(k=0;k<g->arcnum;k++)

{

scanf("%c",&vertex1); getchar();

scanf("%c",&vertex2); getchar();

location1=LocateVex(g,vertex1);

location2=LocateVex(g,vertex2);

if(location1==-1||location2==-1)

{

printf("no such vertex.\n");

k--;

continue;

}

arc\_now1=(p\_ArcNode)malloc(sizeof(ArcNode)); if(!arc\_now1) exit(OVERFLOW);

if(g->vertices[location1].firstarc==NULL)

{

g->vertices[location1].firstarc=arc\_now1;

arc\_now1->adjvex=location2;

arc\_now1->weight=-1;

arc\_now1->nextarc=NULL;

arc\_end[location1]=arc\_now1;

}

else

{

arc\_end[location1]->nextarc=arc\_now1;

arc\_now1->adjvex=location2;

arc\_now1->weight=-1;

arc\_now1->nextarc=NULL;

arc\_end[location1]=arc\_now1;

}

arc\_now2=(p\_ArcNode)malloc(sizeof(ArcNode)); if(!arc\_now2) exit(OVERFLOW);

if(g->vertices[location2].firstarc==NULL)

{

g->vertices[location2].firstarc=arc\_now2;

arc\_now2->adjvex=location1;

arc\_now2->weight=-1;

arc\_now2->nextarc=NULL;

arc\_end[location2]=arc\_now2;

}

else

{

arc\_end[location2]->nextarc=arc\_now2;

arc\_now2->adjvex=location1;

arc\_now2->weight=-1;

arc\_now2->nextarc=NULL;

arc\_end[location2]=arc\_now2;

}

printf("%d arcs done.\n",k+1);

}

printf("%d arcs inputed\n",g->arcnum);

}

//UDN

if(g->kind==UDN)

{

printf("The graph is a UDN, input the arcs with two linked vertexes and weight\n");

for(k=0;k<g->arcnum;k++)

{

scanf("%c",&vertex1); getchar();

scanf("%c",&vertex2); getchar();

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

location1=LocateVex(g,vertex1);

location2=LocateVex(g,vertex2);

if(location1==-1||location2==-1)

{

printf("no such vertex.\n");

k--;

continue;

}

arc\_now1=(p\_ArcNode)malloc(sizeof(ArcNode)); if(!arc\_now1) exit(OVERFLOW);

if(g->vertices[location1].firstarc==NULL)

{

g->vertices[location1].firstarc=arc\_now1;

arc\_now1->adjvex=location2;

arc\_now1->weight=weight;

arc\_now1->nextarc=NULL;

arc\_end[location1]=arc\_now1;

}

else

{

arc\_end[location1]->nextarc=arc\_now1;

arc\_now1->adjvex=location2;

arc\_now1->weight=weight;

arc\_now1->nextarc=NULL;

arc\_end[location1]=arc\_now1;

}

arc\_now2=(p\_ArcNode)malloc(sizeof(ArcNode)); if(!arc\_now2) exit(OVERFLOW);

if(g->vertices[location2].firstarc==NULL)

{

g->vertices[location2].firstarc=arc\_now2;

arc\_now2->adjvex=location1;

arc\_now2->weight=weight;

arc\_now2->nextarc=NULL;

arc\_end[location2]=arc\_now2;

}

else

{

arc\_end[location2]->nextarc=arc\_now2;

arc\_now2->adjvex=location1;

arc\_now2->weight=weight;

arc\_now2->nextarc=NULL;

arc\_end[location2]=arc\_now2;

}

printf("%d arcs done.\n",k+1);

}

printf("%d arcs inputed\n",g->arcnum);

}

return g;

}

status DestroyGraph(Graph g)

{

int i;

p\_ArcNode pre=NULL;

p\_ArcNode cur=NULL;

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

{

cur=g->vertices[i].firstarc;

if(!cur)

{

continue;

}

while(cur)

{

pre=cur;

cur=cur->nextarc;

free(pre);

}

}

free(g);

return OK;

}

int LocateVex(Graph g,Vertex u)

{

int i;

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

{

if(g->vertices[i].data==u) return i;

else continue;

}

return -1;

}

Vertex GetVex(Graph g,int v)

{

if(v<0||v>=g->vexnum) return '#';

else return g->vertices[v].data;

}

status PutVex(Graph g,int v,Vertex value)

{

if(v<0||v>=g->vexnum) return ERROR;

else

{

g->vertices[v].data=value;

return OK;

}

}

int FirstAdjVex(Graph g,int v)

{

if(v<0||v>=g->vexnum) return -2;

else

{

if(g->vertices[v].firstarc)

{

return g->vertices[v].firstarc->adjvex;

}

else return -1;

}

}

int NextAdjVex(Graph g,int v, int w)

{

if(v<0||v>=g->vexnum) return -2;

if(w<0||w>=g->vexnum) return -2;

p\_ArcNode cur=g->vertices[v].firstarc;

// p\_ArcNode pre=NULL;

while(cur)

{

if(cur->adjvex==w) break;

else cur=cur->nextarc;

}

if(cur)

{

if(cur->nextarc) return cur->nextarc->adjvex;

else return -1;

}

else return -2;

}

status InsertVex(Graph g,Vertex v)

{

if(g->vexnum>=MAX\_VERTEX\_NUM) return ERROR;

g->vertices[g->vexnum].data=v;

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

(g->vexnum)++;

return OK;

}

status DeleteVex(Graph g,int v)

{

p\_ArcNode arc=NULL;

p\_ArcNode pre=NULL;

int arc\_deleted=0;

int i;

if(v<0||v>=g->vexnum) return ERROR;

if(g->kind==DN||g->kind==DG)

{

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

while(arc)

{

pre=arc;

arc=arc->nextarc;

free(pre);

arc\_deleted++;

}

printf("arc\_deleted=%d\n",arc\_deleted);

g->vertices[v].firstarc=NULL;

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

{

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

while(arc)

{

printf("%p\n",arc);

if(g->vertices[i].firstarc==arc)

{

if(arc->adjvex==v)

{

pre=arc;

arc=arc->nextarc;

free(pre);

arc\_deleted++;

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

}

else

{

pre=arc;

arc=arc->nextarc;

}

}

else

{

if(arc->adjvex==v)

{

arc=arc->nextarc;

free(pre->nextarc);

arc\_deleted++;

pre->nextarc=arc;

}

else

{

arc=arc->nextarc;

pre=pre->nextarc;

}

}

}

}

}

if(g->kind==UDG||g->kind==UDN)

{

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

while(arc)

{

pre=arc;

arc=arc->nextarc;

free(pre);

arc\_deleted++;

}

g->vertices[v].firstarc=NULL;

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

{

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

while(arc)

{

if(g->vertices[i].firstarc==arc)

{

if(arc->adjvex==v)

{

pre=arc;

arc=arc->nextarc;

free(pre);

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

}

else

{

pre=arc;

arc=arc->nextarc;

}

}

else

{

if(arc->adjvex==v)

{

arc=arc->nextarc;

free(pre->nextarc);

pre->nextarc=arc;

}

else

{

arc=arc->nextarc;

pre=pre->nextarc;

}

}

}

}

}

for(i=v+1;i<=g->vexnum;i++)

{

g->vertices[i-1]=g->vertices[i];

}

g->arcnum-=arc\_deleted;

g->vexnum--;

p\_ArcNode now=NULL;

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

{

now=g->vertices[i].firstarc;

while(now)

{

if(now->adjvex>v) now->adjvex--;

now=now->nextarc;

}

}

printf("arc\_deleted=%d\n",arc\_deleted);

return OK;

}

status InsertArc(Graph g,int v,int w)

{

if(v<0||v>=g->vexnum) return ERROR;

if(w<0||w>=g->vexnum) return ERROR;

p\_ArcNode new\_arc=NULL;

p\_ArcNode end\_arc=NULL;

int weight;

if(g->kind==DG||g->kind==DN)

{

end\_arc=g->vertices[v].firstarc;

while(end\_arc)

{

if(end\_arc->adjvex==w) return INFEASIBLE;

else

{

if(end\_arc->nextarc==NULL) break;

else

{

end\_arc=end\_arc->nextarc;

continue;

}

}

}

if(end\_arc)

{

new\_arc=(p\_ArcNode)malloc(sizeof(ArcNode));

end\_arc->nextarc=new\_arc;

new\_arc->adjvex=w;

new\_arc->nextarc=NULL;

if(g->kind==DG) new\_arc->weight=-1;

else

{

printf("you are adding an arc in a DN, please enter the weight\n");

scanf("%d",&weight); fflush(stdin);

new\_arc->weight=weight;

}

}

else

{

new\_arc=(p\_ArcNode)malloc(sizeof(ArcNode));

g->vertices[v].firstarc=new\_arc;

new\_arc->adjvex=w;

new\_arc->nextarc=NULL;

if(g->kind==DG) new\_arc->weight=-1;

else

{

printf("you are adding an arc in a DN, please enter the weight\n");

scanf("%d",&weight); fflush(stdin);

new\_arc->weight=weight;

}

}

}

if(g->kind==UDN||g->kind==UDG)

{

end\_arc=g->vertices[v].firstarc;

while(end\_arc)

{

if(end\_arc->adjvex==w) return INFEASIBLE;

else

{

if(end\_arc->nextarc==NULL) break;

else

{

end\_arc=end\_arc->nextarc;

continue;

}

}

}

if(end\_arc)

{

new\_arc=(p\_ArcNode)malloc(sizeof(ArcNode));

end\_arc->nextarc=new\_arc;

new\_arc->adjvex=w;

new\_arc->nextarc=NULL;

if(g->kind==UDG) new\_arc->weight=-1;

else

{

printf("you are adding an arc in an UDN, please enter the weight\n");

scanf("%d",&weight); fflush(stdin);

new\_arc->weight=weight;

}

}

if(!end\_arc)

{

new\_arc=(p\_ArcNode)malloc(sizeof(ArcNode));

g->vertices[v].firstarc=new\_arc;

new\_arc->adjvex=w;

new\_arc->nextarc=NULL;

if(g->kind==UDG) new\_arc->weight=-1;

else

{

printf("you are adding an arc in an UDN, please enter the weight\n");

scanf("%d",&weight); fflush(stdin);

new\_arc->weight=weight;

}

}

end\_arc=g->vertices[w].firstarc;

while(end\_arc)

{

if(end\_arc->adjvex==v) return INFEASIBLE;

else

{

if(end\_arc->nextarc==NULL) break;

else

{

end\_arc=end\_arc->nextarc;

continue;

}

}

}

if(end\_arc)

{

new\_arc=(p\_ArcNode)malloc(sizeof(ArcNode));

end\_arc->nextarc=new\_arc;

new\_arc->adjvex=v;

new\_arc->nextarc=NULL;

if(g->kind==UDG) new\_arc->weight=-1;

else new\_arc->weight=weight;

}

if(!end\_arc)

{

new\_arc=(p\_ArcNode)malloc(sizeof(ArcNode));

g->vertices[w].firstarc=new\_arc;

new\_arc->adjvex=v;

new\_arc->nextarc=NULL;

if(g->kind==UDG) new\_arc->weight=-1;

else new\_arc->weight=weight;

}

}

g->arcnum++;

return OK;

}

status DeleteArc(Graph g,int v,int w)

{

if(v<0||v>=g->vexnum) return ERROR;

if(w<0||w>=g->vexnum) return ERROR;

p\_ArcNode arc=NULL;

p\_ArcNode pre=NULL;

if(g->kind==DG||g->kind==DN)

{

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

while(arc)

{

if(g->vertices[v].firstarc==arc)

{

if(arc->adjvex==w)

{

pre=arc;

arc=arc->nextarc;

free(pre);

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

break;

}

else

{

pre=arc;

arc=arc->nextarc;

continue;

}

}

else

{

if(arc->adjvex==w)

{

arc=arc->nextarc;

free(pre->nextarc);

pre->nextarc=arc;

break;

}

else

{

arc=arc->nextarc;

pre=pre->nextarc;

continue;

}

}

}

}

if(g->kind==UDG||g->kind==UDN)

{

arc=g->vertices[w].firstarc;

while(arc)

{

if(g->vertices[w].firstarc==arc)

{

if(arc->adjvex==v)

{

pre=arc;

arc=arc->nextarc;

free(pre);

g->vertices[w].firstarc=arc;

break;

}

else

{

pre=arc;

arc=arc->nextarc;

continue;

}

}

else

{

if(arc->adjvex==v)

{

arc=arc->nextarc;

free(pre->nextarc);

pre->nextarc=arc;

break;

}

else

{

arc=arc->nextarc;

pre=pre->nextarc;

continue;

}

}

}

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

while(arc)

{

if(g->vertices[v].firstarc==arc)

{

if(arc->adjvex==w)

{

pre=arc;

arc=arc->nextarc;

free(pre);

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

break;

}

else

{

pre=arc;

arc=arc->nextarc;

continue;

}

}

else

{

if(arc->adjvex==w)

{

arc=arc->nextarc;

free(pre->nextarc);

pre->nextarc=arc;

break;

}

else

{

arc=arc->nextarc;

pre=pre->nextarc;

continue;

}

}

}

}

g->arcnum--;

return OK;

}

void visit(Vertex e)

{

printf("%c\t",e);

}

status DFSTraverse(Graph g,void (\*visit)(Vertex))

{

int v;

for(v=0;v<g->vexnum;v++) visited[v]=FALSE;

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

{

if(!visited[v]) DFS(g,v);

}

return OK;

}

void DFS(Graph g,int v0)

{

int st[MAX\_VERTEX\_NUM+1];

int top=0;

int v,w;

st[++top]=v0;

while(top)

{

v=st[top--];

if(!visited[v])

{ //栈中可能有重复顶点

visited[v]=TRUE;

visit(g->vertices[v].data);

}

for(w=FirstAdjVex(g,v); w>=0; w=NextAdjVex(g, v, w))

{

if (!visited[w]) st[++top]=w;

}

}

}

status BFSTraverse(Graph g,void (\*visit)(Vertex))

{

int u,v,w;

for(v=0;v<g->vexnum;v++) visited[v]=FALSE;

que tail=NULL;

que head=NULL;

que new\_node=NULL;

tail=(que)malloc(sizeof(qu)); tail->next=NULL; tail->pre=NULL; tail->v=-1;

head=tail;

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

{

if(!visited[v])

{

visited[v]=TRUE;

visit(g->vertices[v].data);

new\_node=(que)malloc(sizeof(qu));

new\_node->v=v;

if(head!=tail)

{

new\_node->next=tail->next; new\_node->pre=tail; tail->next->pre=new\_node; tail->next=new\_node;

}

else

{

tail->next=new\_node; new\_node->pre=tail; new\_node->next=tail; tail->pre=new\_node;

head=new\_node;

}

while(head!=tail)

{

head=head->pre;

u=head->next->v;

free(head->next); head->next=tail; tail->pre=head;

for(w=FirstAdjVex(g,u);w>=0;w=NextAdjVex(g,u,w))

{

if(!visited[w])

{

visited[w]=TRUE;

visit(g->vertices[w].data);

new\_node=(que)malloc(sizeof(qu));

new\_node->v=w;

if(head!=tail)

{

new\_node->next=tail->next; new\_node->pre=tail; tail->next->pre=new\_node; tail->next=new\_node;

}

else

{

tail->next=new\_node; new\_node->pre=tail; new\_node->next=tail; tail->pre=new\_node;

head=new\_node;

}

}

}

}

}

}

return OK;

}

void save(ManageList\* M)

{

if(M->length==0) return;

FILE\* fp=NULL;

int i,j;

Graph g;

p\_ArcNode arc,pre;

p\_ArcNode blank=(p\_ArcNode)malloc(sizeof(ArcNode));

blank->weight=-2;

fp=fopen("graph.dat","wb");

fwrite(M,sizeof(ManageList),1,fp);

fwrite(M->elem,sizeof(ListInfo),M->listsize,fp);

for(i=0;i<M->length;i++)

{

g=(M->elem+i)->elem;

fwrite(g,sizeof(ALGraph),1,fp);

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

{

arc=g->vertices[j].firstarc;

while(arc)

{

fwrite(arc,sizeof(ArcNode),1,fp);

pre=arc;

arc=arc->nextarc;

free(pre);

}

fwrite(blank,sizeof(ArcNode),1,fp);

}

free(g);

}

free(M->elem);

printf("已保存\n");

}

void recover(ManageList\* M,Graph\* g)

{

FILE\* fp=NULL;

int i,j,done=0;

p\_ArcNode new\_arc,end\_arc;

fp=fopen("graph.dat","rb");

fread(M,sizeof(ManageList),1,fp);

ListInfo\* info=(ListInfo\*)malloc(sizeof(ListInfo)\*(M->listsize));

M->elem=info;

fread(M->elem,sizeof(ListInfo),M->listsize,fp);

for(i=0;i<M->length;i++)

{

Graph g=(Graph)malloc(sizeof(ALGraph));

fread(g,sizeof(ALGraph),1,fp);

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

{

new\_arc=(p\_ArcNode)malloc(sizeof(ArcNode));

fread(new\_arc,sizeof(ArcNode),1,fp);

if(new\_arc->weight==-2)

{

g->vertices[j].firstarc=NULL;

free(new\_arc);

continue;

}

else

{

g->vertices[j].firstarc=new\_arc;

new\_arc->nextarc=NULL;

end\_arc=new\_arc;

}

while(!done)

{

new\_arc=(p\_ArcNode)malloc(sizeof(ArcNode));

fread(new\_arc,sizeof(ArcNode),1,fp);

if(new\_arc->weight==-2)

{

free(new\_arc);

done=1;

}

else

{

end\_arc->nextarc=new\_arc;

new\_arc->nextarc=NULL;

end\_arc=new\_arc;

}

}

done=0;

}

(M->elem+i)->elem=g;

}

M->cur=0;

\*g=M->elem[0].elem;

printf("恢复成功,按回车继续\n"); getchar();

}