

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 计算机科学与技术（校际交流）1601班**

**学 号： U201612696**

**姓 名： 陈淏睿**

**指导教师： 魏巍**

**报告日期： 2018年 1月 11 日**

**计算机科学与技术学院**

目 录

[1 基于顺序存储结构的线性表实现 2](#_Toc501754122)

[1.1 问题描述 2](#_Toc501754123)

[1.2 系统设计 4](#_Toc501754127)

[1.3 系统实现 10](#_Toc501754133)

[1.4 实验小结 20](#_Toc501754138)

[2 基于链式存储结构的线性表实现 21](#_Toc501754139)

[2.1 问题描述 21](#_Toc501754140)

[2.2 系统设计 23](#_Toc501754144)

[2.3 系统实现 30](#_Toc501754150)

[2.4 实验小结 40](#_Toc501754155)

[3 基于二叉链表的二叉树实现 41](#_Toc501754156)

[3.1 问题描述 41](#_Toc501754157)

[3.2 系统设计 44](#_Toc501754161)

[3.3 系统实现 50](#_Toc501754167)

[3.4 实验小结 62](#_Toc501754172)

[4 基于邻接表的图实现 63](#_Toc501754173)

[4.1 问题描述 63](#_Toc501754174)

[4.2 系统设计 65](#_Toc501754178)

[4.3 系统实现 72](#_Toc501754184)

[4.4 实验小结 83](#_Toc501754189)

[参考文献 84](#_Toc501754190)

[指导教师评定意见 85](#_Toc501754191)

[附录A 基于顺序存储结构线性表实现的源程序 87](#_Toc501754192)

[附录B 基于链式存储结构线性表实现的源程序 102](#_Toc501754193)

[附录C 基于二叉链表二叉树实现的源程序 117](#_Toc501754194)

[附录D 基于邻接表图实现的源程序 135](#_Toc501754195)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

* + 1. 线性表的基本概念

线性表是最常用且最简单的一种数据结构，即n个数据元素的有限序列。线性表中元素的个数n定义为线性表的长度，n=0时成为空表。在非空表中的每个数据元素都有一个确定的位置，如a1是第一个数据元素，an是最后一个数据元素，ai是第i个数据元素。线性表的存储结构分为线性存储和链式存储。

* + 1. 逻辑结构与基本运算

线性表的数据逻辑结构定义如下:

ADT List｛

数据对象：D=｛ai|ai∈ElemSet，i=1，2，„„，n，n≥0｝

数据关系：R1=｛ <ai-1，ai> | ai-1，ai∈D，i=2，„„，n｝

｝

依据最小完备性和常用性相结合的原则，以函数形式定义了包括线性表的初始化表、加载表、保存表、销毁表、清空表、判定空表、求表长、获得元素、查找元素、获得前驱、获得后继、插入元素、删除元素、遍历表 14 个基本运算， 要求分别定义函数来实现上述功能，具体功能运算如下：

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

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

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

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

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

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

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

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

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

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

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

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

* + 1. 多线性表抽象数据类型

基于之前实现的线性表数据类型，设计了多线性表的数据对象、数据关系/并以定义了多线性表的种基本运算，具体数据和运算功能定义如下。

ADT MulList{

数据对象:D = { ei|ei ∈ SqList, i = 1,2, ……,n, n≥0 }

数据关系:R1 = { <ei-1,ei>|ei-1,ei∈D, i = 2,……,n }

}

以函数形式定义了包括线性多表的更改操作对象、遍历共2个基本运算， 要求分别定义函数来实现上述功能，具体功能运算如下：

1.Chooselist(vol, &L, List)

初始条件是表MList非空；操作结果是遍历将L指向List中的第vol个表。

2.ChartTraverse(&MList, visit())

初始条件是表MList非空；操作结果是遍历MList中的所有线性表，并对其中中元素遍历调用visit()。

## 1.2 系统设计

* + 1. 数据物理结构

线性表有关的数据物理结构如下：

typedef struct { //顺序表（顺序结构）的定义

ElemType \*elem; //定义整型指针，为存储空间基址

int length; //线性表的长度

int listsize; //当前分配的存储容量(以 sizeof(ElemType)为单位)

}SqList;

要实现同时对多个线性表管理，只需要定义一个结构数组即可。

在本程序中，数据原子类型ElemType被定义为int整型。

* + 1. 演示系统

采用顺序表作为线性表的物理结构，实现§1.1.2的基本运算。其中ElemType为数据元素的类型名，具体含义可自行定义，其它有关类型和常量的定义和引用详见文献[1]的p10。

要求构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。

演示系统可选择实现线性表的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存线性表数据逻辑结构(D,{R})的完整信息；②需要设计线性表文件保存和加载操作合理模式。

演示系统选择实现多个线性表管理。

通过printf函数在每轮循环中输出菜单，op初始化为1以供首次进入循环。op=0为循环唯一退出条件，在菜单中给出。1-14分别代表线性表的一个基本运算，每轮循环中更新op的值后在main函数中通过switch语句执行相应的函数，执行完该功能后break跳出switch语句，开始下一轮while循环，直到用户输入0选择退出，退出系统。系统设计结构如图1-1所示。

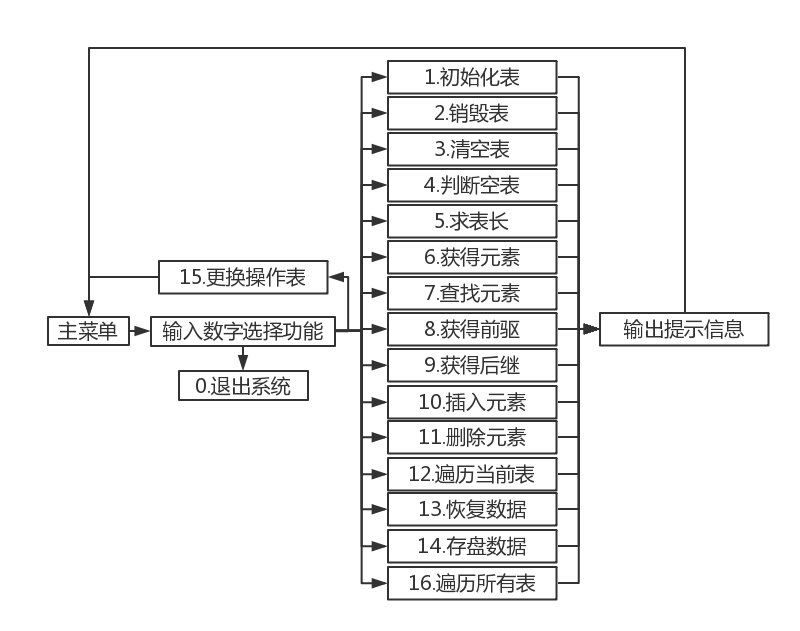


图1-1 系统设计结构图

* + 1. 数据文件存储格式

考虑到简洁高效的原则，按如下方式设计数据文件存储格式。

文件格式:dat，便于程序快速读取数据。

存放方式：顺序按相应的数据类型分配相应的文件空间进行存放，数据之间无间隔。

* + 1. 运算算法思想与设计

线性表运算算法思想与设计如下：

1. 初始化线性表思想：函数的参数是主函数中定义的结构体类型指针变量L的引用，在函数中，首先使用malloc函数分配LISTSIZE大小的连续内存空间，并将首地址返回赋值给L->elem，由于线性表的长度为0，将L->length初始化为0，即完成了线性表的初始化。

2. 销毁线性表思想：函数的参数是结构体类型指针变量L。一开始需要判断合法性。定义一个与表的数量同样大小的判定数组与其专用下标。对于数组中每个判定变量，TRUE（1）记为对应的表已被初始化，反之表示未被初始化（对未修改过的此数组中元素其值不确定）。数组与下标变量均为全局变量。修改操作的表的同时修改下标，将销毁线性表的过程写成函数，其中传入函数是主函数中定义的结构体类型指针变量L。在函数中，首先使用free函数释放掉以L->elem为首地址的连续内存空间，将L->length,L->listsize重新赋值为0，同时对应判定变量修改为FALSE（-1）。

3. 清空线性表的思想：函数的参数是结构体类型指针变量L的引用，一开始需要判断合法性。在函数中，由于清空操作并不释放内存空间，故只需将线性表的长度置为0即可。

4. 判断是否为空表的思想：函数的参数是主函数中定义的指针变量L的引用，一开始需要判断合法性。在调用此函数时直接判断L->length是否为0，若为0则表空，反之非空。

5. 求线性表表长的思想：函数的参数是主函数中定义的指针变量L的引用，一开始需要判断合法性。在函数中，直接返回L->length即为所求线性表的表长。

6. 获得元素的算法思想：函数的参数是结构型变量L以及数据元素的序号i，一开始需要判断合法性。由于采取的是线性存储结构，故直接通过访问数组的方式即L->elem[i-1]来获取元素，当然，在这之前需要判断合法性。

7. 查找元素的算法思想：函数的参数是主函数中定义的结构体类型指针变量L以及查找的数据元素的值，一开始需要判断合法性。通过循环对线性表中的每一个元素与给定值比较看是否相等，如果相等就返回该元素的次序。

8. 获得前驱算法思想：函数的参数是结构体类型指针变量以及特定数据元素的值，接受前驱的变量作为另一个参数，一开始需要判断合法性。首先调用获得元素的函数判断该线性表中特定数据元素的位序，首先判断不为1，否则的话返回FALSE，然后直接返回其前一个元素即L->elem[j-2]。其流程图如图1-2所示。

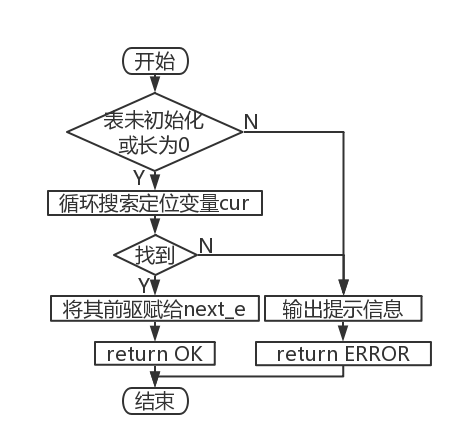
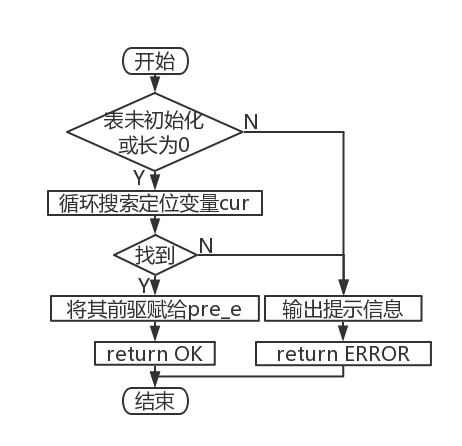


图1-2 获得前驱实现流程图 图1-3 获得后继实现流程图

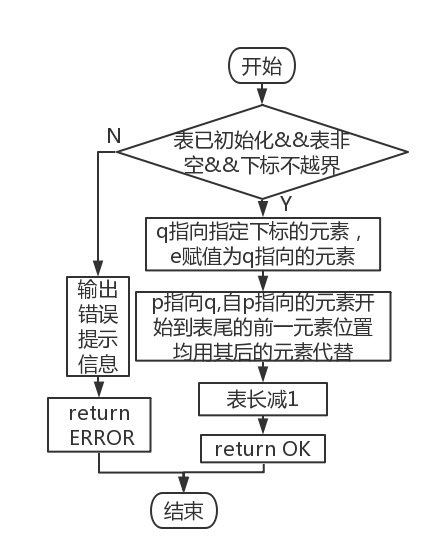
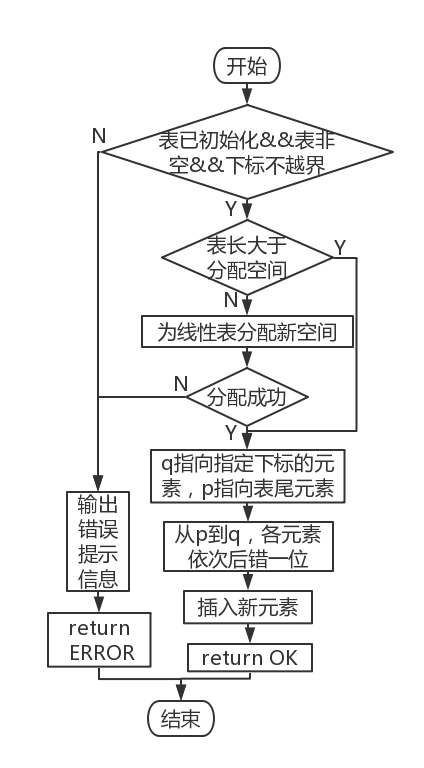


图1-4 插入元素实现流程图 图1-5 删除元素实现流程图

9. 获得后继算法思想：函数的参数是结构体类型指针变量以及特定数据元素的值。一开始需要判断合法性。首先判断是否为最后一个元素，如果不是则直接返回其后一个元素，否则的话返回FALSE，同样该算法需要调用获得元素的函数来确定该特定元素在线性表中的次序。其流程图如图1-3所示。

10. 插入元素算法思想：函数的参数是结构体类型指针变量以及插入元素的值大小以及插入位置。一开始需要判断合法性。在函数中，首先判断插入位置的合法性，即是否在线性表中合适的位置，其次还要判断当前存储空间是否已满，如果满了的话要malloc函数重新分配空间，插入元素时从该位置起到最后一个元素从后开始以此往后移一个单元。其流程图如图1-4所示。

11. 删除元素算法思想：函数的参数是结构体类型指针变量，首先判断位序的合法性，在之后直接将删除元素位置后一个元素直到最后一个元素以此从前往后向前移动一个单元。其流程图如图1-5所示。

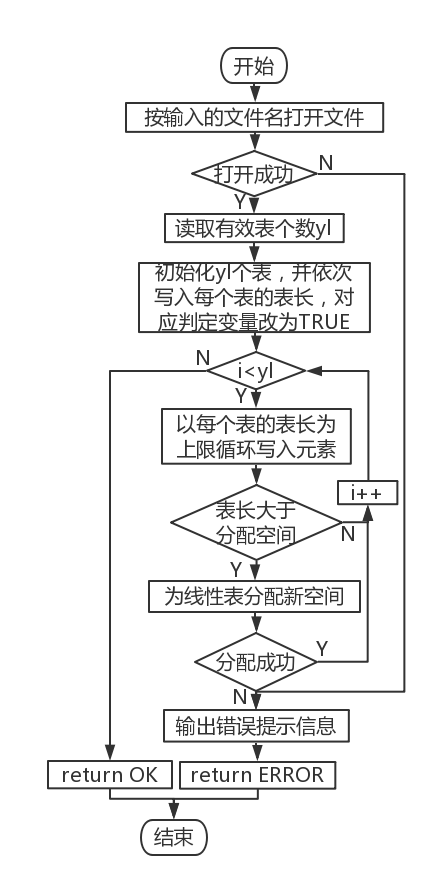
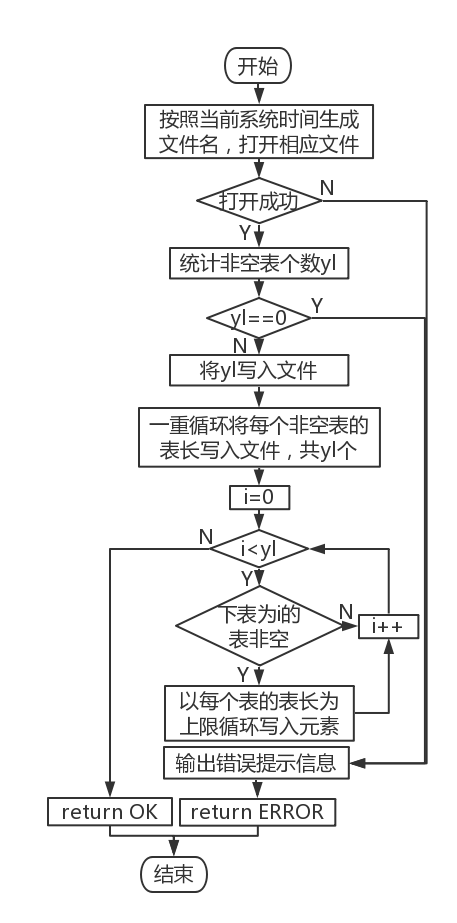


图1-6 数据存盘实现流程图 图1-7 数据读取实现流程图

12. 遍历线性表算法思想：函数的参数是结构体类型指针变量，一开始需要判断合法性。直接用一个循环来对线性表中的每一个元素进行操作。

13. 数据存盘算法思想：函数的参数是文件指针及结构体数组变量。由系统时间自动生成文件名，生成对应二进制文件。先统计已初始化且非空的表的个数，若没有非空表则直接报错返回。将此数据存盘；其次对每个非空表，存盘其表长；最后对每个非空表，循环读入表中各个元素，最后关闭文件。其流程图如图1-6所示。

14．数据读取算法思想：函数的参数是文件指针及结构体数组变量。先读取有效表的个数，按此个数将相同数量的表初始化，读取相同数量的各个表的表长并按顺序写入不同的表中，最后按照已经读取的表长循环读取各个表中的元素，关闭文件。其流程图如图1-7所示。

15．换表算法思想：函数的参数是要操作的表的下标，结构体二级指针变量及结构体数组变量。一开始需要判断合法性。将要操作的表地址赋给同结构的指针，若此表未初始化则输出提示信息。

16．跨表遍历算法思想：函数的参数是结构体数组变量。从第一个表开始，若当前表未初始化或为空，则输出对应提示信息，否则输出表中所有元素。

* + 1. 算法时空复杂度分析

本系统中为对顺序表操作的演示系统。由于对顺序表的操作比较简单，本程序中也没有刻意采用复杂度较高的算法，因此本程序中所涉及的空间/时间复杂度最高也仅为O(n2)。

表1-1 算法时空复杂度分析

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 函数名 | 时间复杂度 | 空间复杂度 | 函数名 | 时间复杂度 | 空间复杂度 |
| IntiaList | O（1） | O（1） | NextElem | O（n） | O（1） |
| DestroyList | O（1） | O（1） | ListInsert | O（n） | O（1） |
| ClearList | O（1） | O（1） | ListDelete | O（n） | O（1） |
| ListEmpty | O（1） | O（1） | ListTrabverse | O（n） | O（1） |
| ListLength | O（1） | O（1） | LoadList | O（n2） | O（1） |
| GetElem | O（1） | O（1） | SaveList | O（n2） | O（1） |
| LocateElem | O（n） | O（1） | ChooseList | O（1） | O（1） |
| PriorElem | O（n） | O（1） | ChartTraverse | O（n2） | O（1） |

## 1.3 系统实现

* + 1. 编程环境，运行环境，项目工程描述

本次实验采用Codeblocks编程软件编写，并用Codeblocks进行编译运行，项目名称是Sequential Linear Chart(Cpp).演示界面菜单如图1-8所示。



图1-8演示系统菜单界面

* + 1. 头文件及预定义常量说明

1.头文件

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

2.预定义常量

#define OK 1

#define TRUE 1

#define FALSE -1

#define ERROR -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

3.类型表达式

typedef int status;

typedef int ElemType;

typedef struct{

ElemType \* elem;

int length;

int listsize;

}SqList;

* + 1. 测试计划

对系统的线性表操作功能按以下计划测试：

1. 检测常规状态下系统的工作情况；
2. 重点检查非法边界操作时的工作情况。

表1-2 演示系统测试计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 待测试功能 | 测试顺序 | 测试输入 | 预计输出 | 线性表状态 |
| 1. IntiaList | 3 | 无 | 初始化L成功 | 表长0，线性表被创建 |
| 2. DestroyList | 11 | 无 | 销毁成功 | 不存在 |
| 3. ClearList | 10 | 无 | 表清空成功 | 空表 |
| 4. ListEmpty | 多次调用 | 无 | L指向未初始化的表时显示为未初始化，指向空表时显示为空表，指向非空表时显示为非空表 | 与其前一步骤相同 |
| 5. ListLength | 多次调用 | 无 | L指向未初始化的表时显示为未初始化，指向已初始化的表时显示表长 | 与其前一步骤相同 |
| 6. GetElem | 5 | 分别输入1和8 | 第一个输入应输出7362，第二个输入下标越界，无法找到 | 与步骤4相同 |
| 7. LocateElem | 6 | 分别输入6363和53463 | 第一个输入应输出2，第二个输入无法找到 | 与步骤5相同 |
| 8. PriorElem | 7 | 分别输入8585，7362和8614 | 第一个输入应输出6363，第二个输入没有前驱，查找失败，第三个输入无法找到 | 与步骤6相同 |
| 9. NextElem | 8 | 分别输入7362，8585和7361 | 第一个输入应输出6363，第二个输入没有后继，查找失败，第三个输入无法找到 | 与步骤7相同 |
| 10. ListInsert | 4 | 分别输入1，7362,；2，6363；3，8585和-52,2 | 对于前三个输入，显示插入成功；对于最后一个输入，应提示下标越界，插入失败 | 随着在合理范围内的下标对应的数据的更新而更新 |
| 11. ListDelete | 9 | 分别输入3和5 | 对于第一个输入，应输出删除成功，删除的元素为8585；对于第二个输入，应提示下标越界，删除失败 | 删除第三个元素8585后其余元素重新按顺序排列 |
| 12. ListTrabverse | 多次调用 | 无 | 若表中有元素，循环打印其所有元素；若表为空表，输出提示此表为空；若表未初始化，输出提示此表未初始化 | 与其前一步骤相同 |
| 13. LoadList | 1，13 | 二进制文件名 | 打开二进制文件成功应输出文件中存储的各个表，未找到对应文件或打开文件失败时输出错误提示信息 | （默认指向第一个表）打开二进制文件成功时应为二进制文件中存储的第一个表，打开文件失败时保持未初始化状态 |
| 14. SaveList | 12 | 无 | 源文件目录下一个新的二进制数据文件，文件名为存盘操作的时间，格式为“星期\_月份缩写\_日期\_时\_分\_秒\_年” | 与步骤11相同 |
| 15. ChooseList | 2 | 20 | L指向内存中最后一个表 | 取决于最后一个表本来的状态，可能未初始化，可能为空，也可能非空 |
| 16. ChartTraverse | 多次调用 | 无 | 对于有元素的表，循环打印其所有元素；对于空表，输出提示此表为空；对于未初始化的表，输出提示此表未初始化 | 与其前一步骤相同 |

* + 1. 测试

1．开始时20个线性表均为空，系统默认操作第一个表。



图1-9 导入数据前

2．通过输入文件名导入数据。

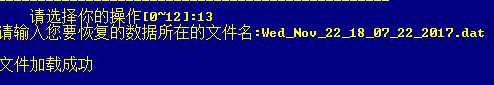


图1-10 输入文件名，导入数据

3．再次遍历，发现数据已成功导入。

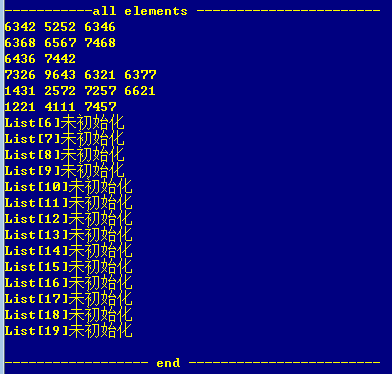


图1-11 导入数据成功

4. 在List[19]，即最后一个表上进行操作。可以发现输出的提示与对整个表遍历时的信息吻合。



图1-12 更改操作表



图1-13 后续操作时菜单统一提示

5. 此时对表进行任何形式的访问（函数2-12），都会提示此表未初始化.举例如下：



图1-14 销毁线性表出错提示（未初始化）



图1-15 判断空表出错提示（未初始化）

6. 对此表初始化，此时表长为0。



图1-16 初始化当前表



图1-17 判断表长

7. 此时对表进行任何形式的访问（函数2-12），都会提示此表为空.举例如下：

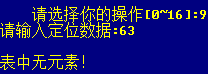


图1-18 求后继出错提示（空表）



图1-19 遍历出错提示（空表）

8. 先后插入元素7362,6363,8585。

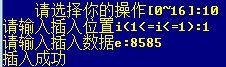


图1-20 插入元素成功

9. 下标越界时，插入失败。

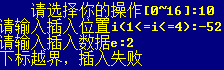


图1-21 插入元素失败

10. 遍历当前表，此时当前表中共3个元素。

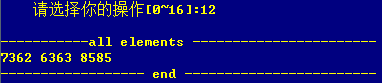


图1-22 遍历当前表



图1-23 判断空表



图1-24 求表长

11. 获取第一个元素。



图1-25 获取元素成功



图1-26 获取元素失败

12. 查找当前表中元素。



图1-27 查找元素成功



图1-28 查找元素失败

13. 获取元素前驱。



图1-29 获取元素前驱成功



图1-30 获取首元前驱



图1-31 未找到此元素

14. 获取元素后继。



图1-32 获取元素后继成功



图1-33 获取尾元后继

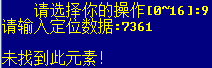


图1-34 未找到此元素

15. 删除元素。



图1-35 删除元素成功



图1-36 删除元素失败（下标越界）

16. 再次遍历当前表，发现8585消失。



图1-37 遍历当前表检查删除元素情况

17. 清空当前表，检查清空情况。



图1-38 清空当前表



图1-39 ListEmpty检查清空情况

18. 删除此表。



图1-40 删除当前表

19. 复查表的删除情况。



图1-41 ListEmpty检查删除情况



图1-42 ListLength检查删除情况

20. 将第7、9、14、20个表初始化，向第12、15、18各表中插入若干元素，遍历全部表。



图1-43 若干输入后遍历全部表

21. 将改动后的表以二进制文件方式存盘

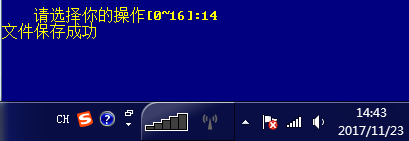


图1-44 文件存盘成功



图1-45 存盘后的文件

22. 向系统中读取刚刚存盘的文件。

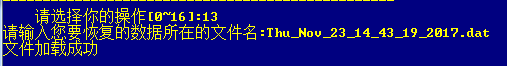


图1-46 读取文件成功

23. 重新遍历全部表，可知只进行初始化，内无元素的表与未初始化的表在存盘时被跳过，造成新表单中表存储的密集化。

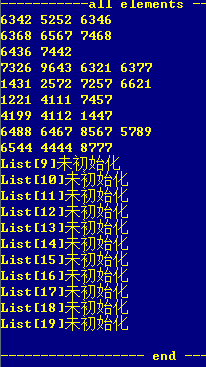


图1-47 遍历全部表

24. 打开错误格式的文件时，显示文件打开失败。



图1-48 目标文件不可读取

25. 退出系统。

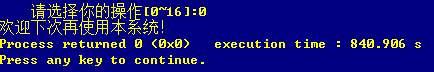


图1-49 退出系统

## 1.4 实验小结

经过本次试验，我充分了解到了线性表的线性物理结构，并且通过切身的体会熟练掌握了线性表的基本操作，提高了自己写有关线性表的代码的能力，尤其是在写的过程中遇到了许多困难，在多次寻求同学的帮助下终于解决了。还发现了自己的薄弱之处，就是在文件的处理时存在许多纰漏，导致文件读取不成功。并且我还加深了对线性表的存在与空的区别。还有对“&”引用符号的理解，加强了其与“\*”的区别，“&e”使得在函数中调用主函数中的值“e”时可以同时更改其值，如在GetElem函数中调用了“e”的值然后在主函数中输出，如果要更改其值的话就必须要用“&”符号。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

1. * 1. 线性表的基本概念

线性表是最常用且最简单的一种数据结构，即n个数据元素的有限序列。线性表中元素的个数n定义为线性表的长度，n=0时成为空表。在非空表中的每个数据元素都有一个确定的位置，如a1是第一个数据元素，an是最后一个数据元素，ai是第i个数据元素。线性表的存储结构分为线性存储和链式存储。

* + 1. 逻辑结构与基本运算

线性表的数据逻辑结构定义如下:

ADT List｛

数据对象：D=｛ai|ai∈ElemSet，i=1，2，„„，n，n≥0｝

数据关系：R1=｛ <ai-1，ai> | ai-1，ai∈D，i=2，„„，n｝

｝

依据最小完备性和常用性相结合的原则，以函数形式定义了包括线性表的初始化表、加载表、保存表、销毁表、清空表、判定空表、求表长、获得元素、查找元素、获得前驱、获得后继、插入元素、删除元素、遍历表 14 个基本运算， 要求分别定义函数来实现上述功能，具体功能运算如下：

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

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

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

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

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

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

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

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

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

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

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

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

* + 1. 多线性表抽象数据类型

基于之前实现的线性表数据类型，设计了多线性表的数据对象、数据关系/并以定义了多线性表的种基本运算，具体数据和运算功能定义如下。

ADT MulList{

数据对象:D = { ei|ei ∈ SqList, i = 1,2, ……,n, n≥0 }

数据关系:R1 = { <ei-1,ei>|ei-1,ei∈D, i = 2,……,n }

}

以函数形式定义了包括线性多表的更改操作对象、遍历共2个基本运算， 要求分别定义函数来实现上述功能，具体功能运算如下：

1.Chooselist(vol, &L, List)

初始条件是表MList非空；操作结果是遍历将L指向List中的第vol个表。

2.ChartTraverse(&MList, visit())

初始条件是表MList非空；操作结果是遍历MList中的所有线性表，并对其中中元素遍历调用visit()。

## 2.2 系统设计

* + 1. 数据物理结构

1．元素结点的数据物理结构

typedef struct lnode{ //元素节点定义

ElemType data; //数据域

struct lnode \*next; //指针域

}LNode;

2．头结点的数据物理结构

typedef struct llist{ //头结点定义

int len; //len存储此表长度

struct lnode \*head, \*tail; //head指向表内首元，tail指向尾元

struct llist \*fore, \*aft; //aft指向下一个表头结点，ore指向前一个表头结点，这样的定义方式决定了在第一个头结点前应有一空白头结点。

}Linklist;

在本程序中，数据原子类型ElemType被定义为int整型。

* + 1. 演示系统

采用链式表表作为线性表的物理结构，实现§2.1.2的基本运算。其中ElemType为数据元素的类型名，具体含义可自行定义，其它有关类型和常量的定义和引用详见文献[1]的p10。

要求构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。

演示系统可选择实现线性表的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存线性表数据逻辑结构(D,{R})的完整信息；②需要设计线性表文件保存和加载操作合理模式。

演示系统选择实现多个线性表管理。

通过printf函数在每轮循环中输出菜单，op初始化为1以供首次进入循环。op=0为循环唯一退出条件，在菜单中给出。1-14分别代表线性表的一个基本运算，每轮循环中更新op的值后在main函数中通过switch语句执行相应的函数，执行完该功能后break跳出switch语句，开始下一轮while循环，直到用户输入0选择退出，退出系统。系统设计结构如图2-1所示。

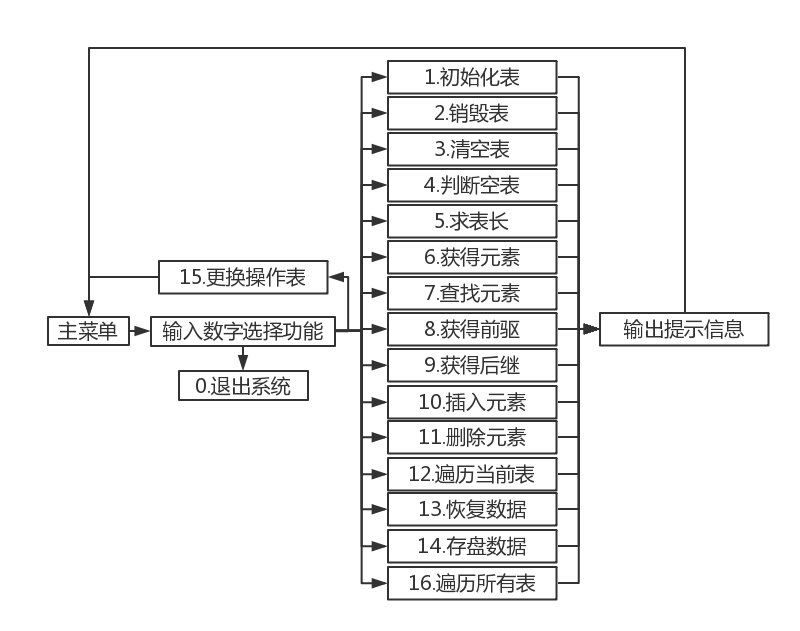


图2-1 系统设计结构图

* + 1. 数据文件存储格式

考虑到简洁高效的原则，按如下方式设计数据文件存储格式。

文件格式:dat，便于程序快速读取数据。

存放方式：顺序按相应的数据类型分配相应的文件空间进行存放，数据之间无间隔。

* + 1. 运算算法思想与设计

线性表运算算法思想与设计如下：

1. 初始化线性表思想：函数的参数是主函数中定义的表头类型指针变量L的地址。在函数中，首先使用malloc函数分配一个内存空间p作为总表表头，在其后接续一个额外空间作为第一个表的小表头，此时内存中有一个空表，即p的下一节点中len成员为0，总表个数length与操作位序vol均改为1，将即完成了线性表的初始化。

2. 销毁线性表思想：一开始需要判断合法性。函数的参数是主函数中定义的表头类型指针变量L，若当前表有两个或以上元素，从前向后依次释放元素，直至表头直接接到仅剩的一个元素。删除此元素后，将空表表头的前驱与后继直接相连，注意由于表头采用双向链表，因此须同时更改其前驱的aft指针与后继的fore指针，删除最后一个表时其前驱的aft直接指向NULL,同时操作位序vol减1。流程图如图2-2所示。

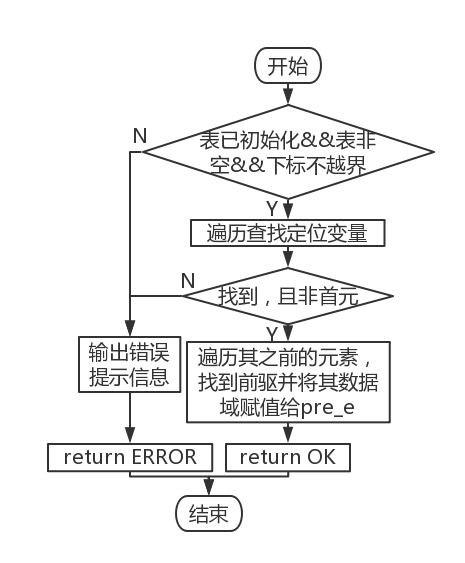
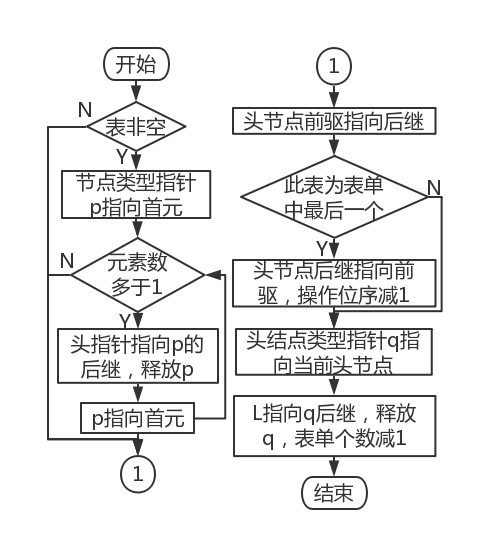


图2-2 销毁线性表实现流程图 图2-3 获得前驱实现流程图

3. 清空线性表的思想：一开始需要判断合法性。函数的参数是主函数中定义的表头类型指针变量L。表非空时，按照与销毁线性表相同的方式清空表中元素。

4. 判断是否为空表的思想：一开始需要判断合法性。函数的参数是主函数中定义的表头类型指针变量L。在调用此函数时直接判断L->head是否为NULL，若为0则表空，反之非空。

5. 求线性表表长的思想：一开始需要判断合法性。函数的参数是主函数中定义的表头类型指针变量L。在函数中，直接返回L->len即为所求线性表的表长。

6. 获得元素的算法思想：一开始需要判断合法性。函数的参数是主函数中定义的表头类型指针变量L,数据元素的序号i以及获取元素的变量e。由于采取的是链式存储结构，故需要通过一重循环找到相应位置的节点后访问节点的数据域获取目标元素。

7. 查找元素的算法思想：一开始需要判断合法性。函数的参数是主函数中定义的表头类型指针变量L以及获取元素的变量e。通过循环对链式表中的每一个节点的数据域与给定值比较是否相等，如果相等就返回该元素的位序。

8. 获得前驱算法思想：一开始需要判断合法性。函数的参数是主函数中定义的表头类型指针变量L，定位变量cur以及接受前驱的变量pre\_e。首先用一重循环在链表中寻找与定位变量cur相同的元素，若找到则记录当前位置，用第二重循环获取其前驱的数据域，对于定位变量在表头时单独讨论。将其赋给pre\_e，否则返回ERROR。流程图如图2-3所示。

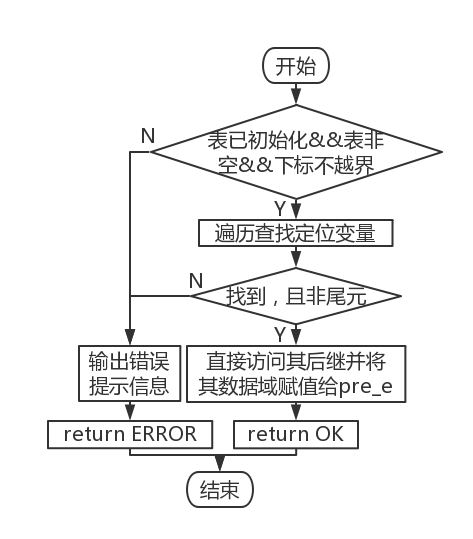


图2-4 获得后继实现流程图

9. 获得后继算法思想：一开始需要判断合法性。函数的参数是主函数中定义的表头类型指针变量L，定位变量cur以及接受后继的变量next\_e。首先用一重循环在链表中寻找与定位变量cur相同的元素，若找到则记录当前位置，此时可直接访问其后继的数据域。将其赋给next\_e，对于定位变量在表尾时单独讨论。否则返回FALSE。流程图如图2-4所示。

a

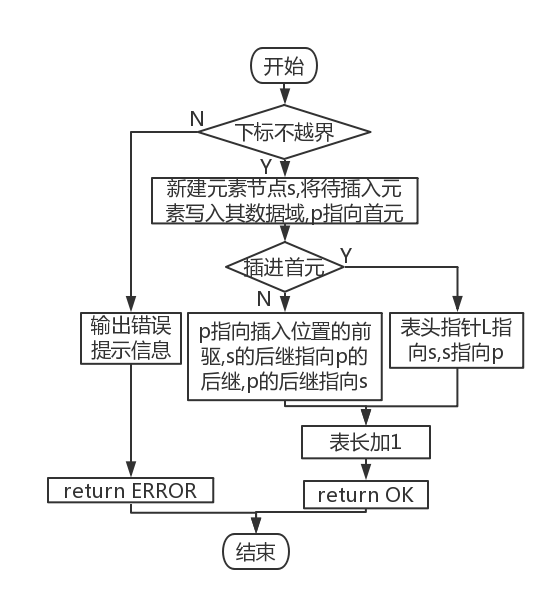


图2-5 插入元素实现流程图

10. 插入元素算法思想：一开始需要判断合法性。函数的参数是主函数中定义的表头类型指针变量L，定位变量i以及接受待插入的变量next\_e。在函数中，首先判断插入位置是否在线性表中。合法时新建一元素节点s，插入变量作为其数据域。对于首元的插入，直接让表头指向新节点，新节点指向之前的首元。一重循环使指针指向目标位置的前驱p，通过修改s与p的后继指针指向插入新节点。流程图如图2-5所示。

11. 删除元素算法思想：一开始需要判断合法性。函数的参数是主函数中定义的表头类型指针变量L，定位变量i以及返回删除的变量next\_e。在函数中，首先判断删除位置是否在线性表中。位置合法时用一重循环定位其前驱p,将前驱的next指针直接指向后继，对于首元的删除，直接让表头指向首元后继。将被删除的节点的数据域赋给next\_e在之后直接将删除元素位置后一个元素直到最后一个元素以此从前往后向前移动一个单元。流程图如图2-6所示。

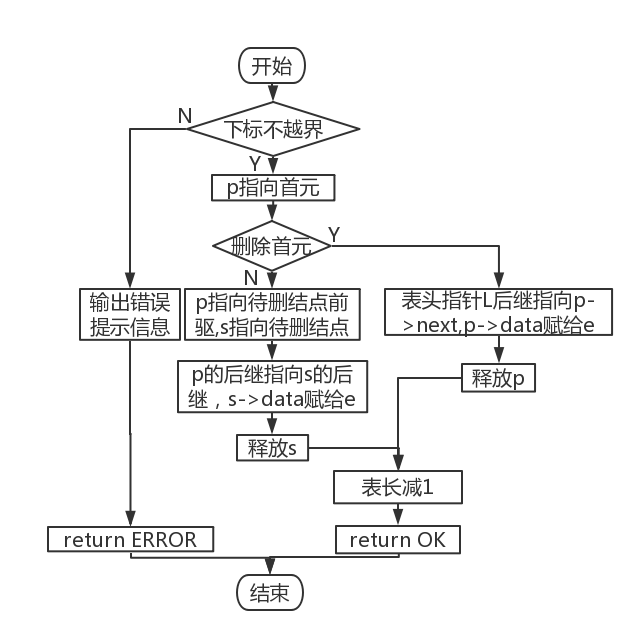


图2-6 删除元素实现流程图

12. 遍历线性表算法思想：一开始需要判断合法性。传入函数的参数是主函数中定义的表头类型指针变量L。从首元开始用一个循环访问所有结点，表空时报错。

13. 数据存盘算法思想：一开始需要判断合法性。函数的参数是文件指针fp及总表头类型指针变量List。由系统时间自动生成文件名，生成对应二进制文件。先用一重循环统计非空的表的个数，若没有非空表则直接报错返回。将此数据存盘；其次一重循环对每个非空表存盘其表长；接着对每个非空表，循环读入表中各个元素，最后关闭文件。流程图如图2-7所示。

14．数据恢复算法思想：一开始需要判断合法性。函数的参数是文件指针fp及总表头类型指针变量List。按文件名成功打开文件后先读取有效表的个数，按此个数用一重循环将相同数量的表头结点初始化，每初始化一个节点，将文件指针当前所指的对应表长写入其数据域，最后按照各个表的表长用二重循环读取相同个数的的元素，关闭文件。流程图如图2-8所示。

15．换表算法思想：一开始需要判断合法性。函数的参数是主函数中定义的表头类型指针变量的地址&L及总表头类型指针变量List。提供的操作范围为1到表长加1，即可选择在最后一个表后新建一张表。若照此选择，直接在表头尾节点后新建一个表头，否则用一重循环定位目标表，将其位置赋给L的地址。

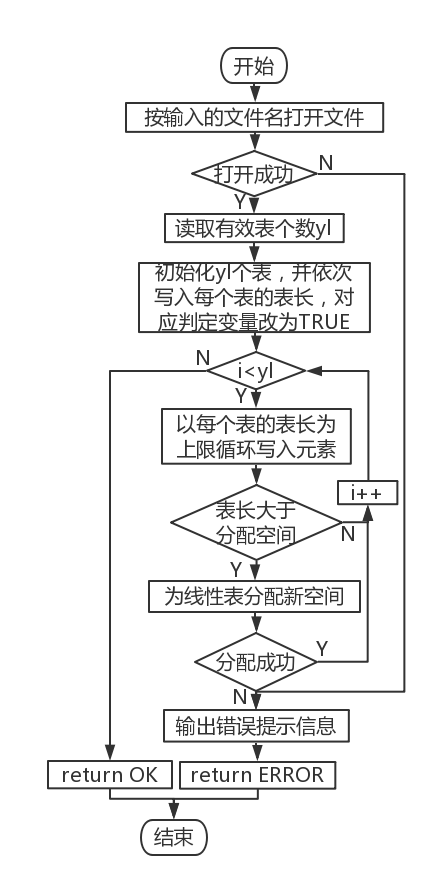
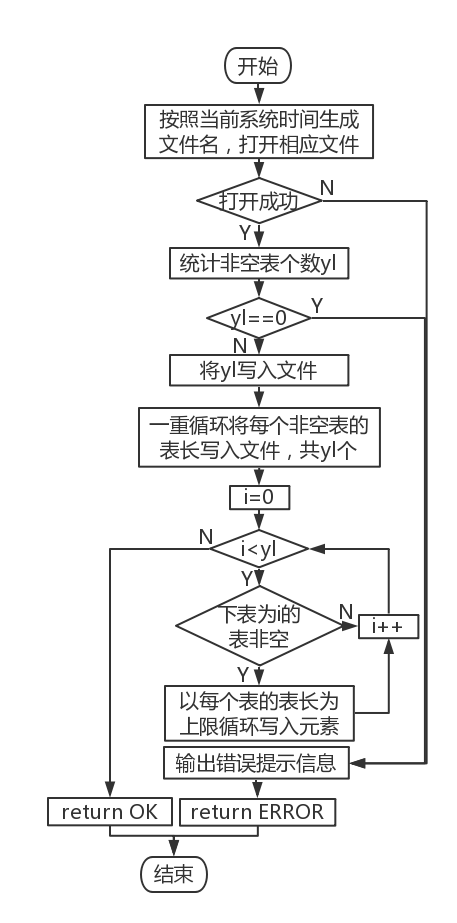


图2-7 数据存盘实现流程图 图2-8 数据恢复实现流程图

16．跨表遍历算法思想：一开始需要判断合法性。函数的参数是总表头类型指针变量List。从第一个表开始进行循环，若当前表未初始化或为空，则输出对应提示信息，否则第二重循环输出表中所有元素。

* + 1. 算法时空复杂度分析

各函数的时空开销与实验一中大抵相同，但由于采用链式表物理结构，因此不能通过元素下标快速定位，少部分函数的时间复杂度由O（1）增长至O（n），具体见下表。

表2-1 算法时空复杂度分析

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 函数名 | 时间复杂度 | 空间复杂度 | 函数名 | 时间复杂度 | 空间复杂度 |
| IntiaList | O（1） | O（1） | NextElem | O（n） | O（1） |
| DestroyList | O（n） | O（1） | ListInsert | O（n） | O（1） |
| ClearList | O（n） | O（1） | ListDelete | O（n） | O（1） |
| ListEmpty | O（1） | O（1） | ListTrabverse | O（n） | O（1） |
| ListLength | O（1） | O（1） | LoadList | O（n2） | O（1） |
| GetElem | O（n） | O（1） | SaveList | O（n2） | O（1） |
| LocateElem | O（n） | O（1） | ChooseList | O（1） | O（1） |
| PriorElem | O（n2） | O（1） | ChartTraverse | O（n2） | O（1） |

## 2.3 系统实现

1. * 1. 编程环境，运行环境，项目工程描述

本次实验采用Codeblocks编程软件编写，并用Codeblocks进行编译运行，项目名称是Node-based Linear Chart(Cpp)。演示系统菜单界面如图2-9所示



图2-9 演示系统菜单界面

* + 1. 头文件及预定义常量说明

1.头文件

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

2.预定义常量

#define OK 1

#define TRUE 1

#define FALSE -1

#define ERROR -2

3.类型表达式

typedef int status;

typedef int ElemType;

typedef struct lnode{

ElemType data;

struct lnode \*next;

}LNode;

typedef struct llist{

int len;

struct lnode \*head, \*tail;

struct llist \*fore, \*aft;

}Linklist;

* + 1. 测试计划

对系统的链式表操作功能按以下计划测试：

1. 检测常规状态下系统的工作情况；
2. 重点检查非法边界操作时的工作情况。

表2-2 演示系统测试计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 待测试功能 | 测试顺序 | 测试输入 | 预计输出 | 线性表状态 |
| 1. IntiaList | 1 | 无 | 初始化L成功 | 表长0，线性表被创建 |
| 2. DestroyList | 10 | 无 | 销毁成功 | 不存在 |
| 3. ClearList | 9 | 无 | 表清空成功 | 空表 |
| 4. ListEmpty | 多次调用 | 无 | L指向未初始化的表时显示为未初始化，指向空表时显示为空表，指向非空表时显示为非空表 | 与其前一步骤相同 |
| 5. ListLength | 多次调用 | 无 | L指向未初始化的表时显示为未初始化，指向已初始化的表时显示表长 | 与其前一步骤相同 |
| 6. GetElem | 4 | 分别输入1和8 | 第一个输入应输出7362，第二个输入下标越界，无法找到 | 与步骤4相同 |
| 7. LocateElem | 5 | 分别输入6363和53463 | 第一个输入应输出2，第二个输入无法找到 | 与步骤5相同 |
| 8. PriorElem | 6 | 分别输入8585，7362和8614 | 第一个输入应输出6363，第二个输入没有前驱，查找失败，第三个输入无法找到 | 与步骤6相同 |
| 9. NextElem | 7 | 分别输入7362，8585和7361 | 第一个输入应输出6363，第二个输入没有后继，查找失败，第三个输入无法找到 | 与步骤7相同 |
| 10. ListInsert | 3 | 分别输入1，6473,；2，7144；3，6369和-52,2 | 对于前三个输入，显示插入成功；对于最后一个输入，应提示下标越界，插入失败 | 随着在合理范围内的下标对应的数据的更新而更新 |
| 11. ListDelete | 8 | 分别输入3和5 | 对于第一个输入，应输出删除成功，删除的元素为8585；对于第二个输入，应提示下标越界，删除失败 | 删除第三个元素8585后其余元素重新按顺序排列 |
| 12. ListTrabverse | 多次调用 | 无 | 若表中有元素，循环打印其所有元素；若表为空表，输出提示此表为空；若表未初始化，输出提示此表未初始化 | 与其前一步骤相同 |
| 13. LoadList | 2，12 | 二进制文件名 | 打开二进制文件成功应输出文件中存储的各个表，未找到对应文件或打开文件失败时输出错误提示信息 | （默认指向第一个表）打开二进制文件成功时应为二进制文件中存储的第一个表，打开文件失败时保持未初始化状态 |
| 14. SaveList | 11 | 无 | 源文件目录下一个新的二进制数据文件，文件名为存盘操作的时间，格式为“星期\_月份缩写\_日期\_时\_分\_秒\_年” | 与步骤11相同 |
| 15. ChooseList | 3 | 20 | L指向内存中最后一个表 | 取决于最后一个表本来的状态，可能未初始化，可能为空，也可能非空 |
| 16. ChartTraverse | 多次调用 | 无 | 对于有元素的表，循环打印其所有元素；对于空表，输出提示此表为空；对于未初始化的表，输出提示此表未初始化 | 与其前一步骤相同 |

* + 1. 测试

1．开始时系统中无表。



图2-10 导入数据前



图2-11 当前内存中无表

2. 此时对表进行任何形式的访问（函数2-12），都会提示此表未初始化.举例如下：



图2-12 DestroyList出错提示（未初始化）



图2-13 ClearList出错提示（未初始化）



图2-14 ClearList出错提示（未初始化）

3. 初始化表单。



图2-15 输入文件名，导入数据



图2-16 当前系统中有1个表

4. 此时对表进行任何形式的访问（函数2-12），都会提示此表为空.举例如下：

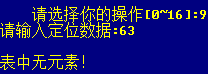


图2-17 NextElem出错提示（空表）

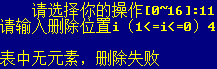


图2-18 ListDelete出错提示（空表）



图2-19 ListTrabverse出错提示（空表）

5．通过输入文件名导入数据，此时系统中共4个表。

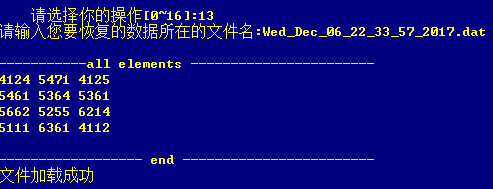


图2-20 输入文件名，导入数据成功



图2-21 导入数据成功

6. 在表单末尾新增一个表，即List[5]。可以发现输出的提示与对整个表遍历时的信息吻合。



图2-22 更改操作表



图2-23 后续操作时菜单统一提示

7. 此时表长为0.



图2-24 判断表长

8. 先后插入元素6473，7144，6369。

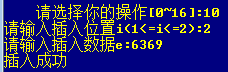


图2-25 插入元素成功

9. 下标越界时，插入失败。

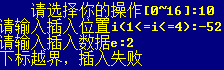


图2-26 插入元素失败

10. 遍历当前表，此时当前表中共3个元素。



图2-27 遍历当前表



图2-28 求表长

11. 获取第一个元素。



图2-29 获取元素成功



图2-30 获取元素失败

12. 查找当前表中元素。



图2-31 查找元素成功



图2-32 查找元素失败

13. 获取元素前驱。



图2-33 获取元素前驱成功



图2-34 获取首元前驱



图2-35 未找到此元素

14. 获取元素后继。



图2-36 获取元素后继成功



图2-37 获取尾元后继

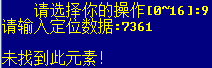


图2-38 未找到此元素

15. 删除元素。



图2-39 删除元素成功



图2-40 删除元素失败（下标越界）

16. 再次遍历当前表，发现8585消失。



图2-41 遍历当前表检查删除元素情况

17. 清空当前表。



图2-42 清空当前表

18. 检查清空情况。



图2-43 ListEmpty函数检查清空情况

19. 删除此表。



图2-44 删除当前表

20. 复查表的删除情况。



图2-45 ListEmpty函数检查删除情况



图2-46 ListLength函数检查删除情况



图2-47 LocateElem函数检查删除情况

21. 创建第5、7、9、10、12个表，向第6、8、11各表中插入若干元素，遍历全部表。



图2-48 若干输入后遍历全部表

22. 将改动后的表以二进制文件方式存盘

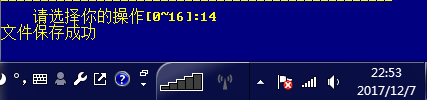


图2-49 文件存盘成功



图2-50 存盘后的文件

23. 向系统中读取刚刚存盘的文件。

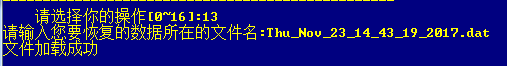


图2-51 读取文件成功

24. 重新遍历全部表，可知只进行初始化，内无元素的表与未初始化的表在存盘时被跳过，造成新表单中表存储的密集化。

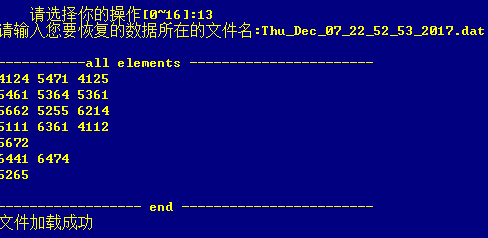


图2-52 遍历全部表

25. 打开错误格式的文件时，显示文件打开失败。



图2-53 目标文件不可读取

26. 退出系统。

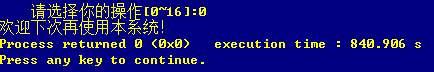


图2-54 退出系统

## 2.4 实验小结

经过本次试验，我充分了解到了链式表的物理结构，并且通过切身的体会熟练掌握了链式表的基本操作，提高了自己写有关链式表的代码的能力，尤其是在写的过程中遇到了许多困难，在多次寻求同学的帮助下终于解决了。通过本次试验得多表操作部分，我复习了十字链表的构造，且在实践中第一次用上了双链表。通过我写代码与debug用了同样长时间的教训，我明白了在课下还是要多加练习，这样在实验的时候才不会对一些基本语句感到不熟悉。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

* + 1. 二叉树的基本概念

二叉树是每个节点最多有两个子树的树结构。通常子树被称作“左子树”（left subtree）和“右子树”（right subtree）。二叉树常被用于实现二叉查找树和二叉堆。

二叉树的每个结点至多只有二棵子树(不存在度大于2的结点)，二叉树的子树有左右之分，次序不能颠倒。二叉树的第i层至多有2^{i-1}个结点；深度为k的二叉树至多有2^k-1个结点；对任何一棵二叉树T，如果其终端结点数为n\_0，度为2的结点数为n\_2，则n\_0=n\_2+1。二叉树的存储结构分为线性存储和链式存储。

* + 1. 逻辑结构与基本运算

抽象数据类型二叉树的定义如下：

ADT BinaryTree {

数据对象D：D是具有相同特性的数据元素的集合。

数据关系R:

若D=Φ，则R=Φ，称BinaryTree为空二叉树；

若D≠Φ，则R={H},H是如下二元关系：

1. 在D中存在唯一的成为根的数据元素root，它在关系H中无前驱；
2. 若D-{root}≠Φ,则存在D-{root}={D1,Dr},且D1∩Dr=Φ;
3. 若D1≠Φ，则D1中存在唯一的元素X1，<root,X1>∈H，且存在D1上的关系H1包含于H；若Dr≠Φ,则Dr中存在唯一的元素Xr，<root,Xr>∈H,且存在Dr上的关系属于H；
4. （D，{H1}）是一棵符合本定义的二叉树，称为根的左子树，（Dr，{Hr}）是一棵符合本定义的二叉树，称为根的右子树。

}

依据最小最小完备性和常用性相结合的原则，以函数形式定义了二叉树的初始化、销毁二叉树、创建二叉树、清空二叉树、判定空二叉树和求二叉树深度等20种基本运算，具体运算功能定义如下：

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 3.2 系统设计

* + 1. 数据物理结构

二叉树的数据物理结构如下：

typedef struct BiTNode

{

ElemType key; //结点编号

char data; //结点名称

struct BiTNode \*lchild, \*rchild;//左、右子树指针

}BiTNode, \*BitTree;//typedef是将结构类型定义struct BiTNode取别名为BiTNode

在本程序中，数据原子类型ElemType被定义为int整型。

* + 1. 演示系统

采用二叉链表表作为二叉树的物理结构，实现§3.1.2的基本运算。其中TElemType为数据元素的类型名，具体含义可自行定义，其它有关类型和常量的定义和引用详见文献[1]的p127。

要求构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。

演示系统可选择实现二叉树的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存二叉树数据逻辑结构(D,{R})的完整信息；②需要设计二叉树文件保存和加载操作合理模式。附录B提供了文件存取的方法。

通过printf函数在每轮循环中输出菜单，op初始化为1以供首次进入循环。op=0为循环唯一退出条件，在菜单中给出。1-22分别代表线性表的一个基本运算，每轮循环中更新op的值后在main函数中通过switch语句执行相应的函数，执行完该功能后break跳出switch语句，开始下一轮while循环，直到用户输入0选择退出，退出系统。系统设计结构如图3-1所示。

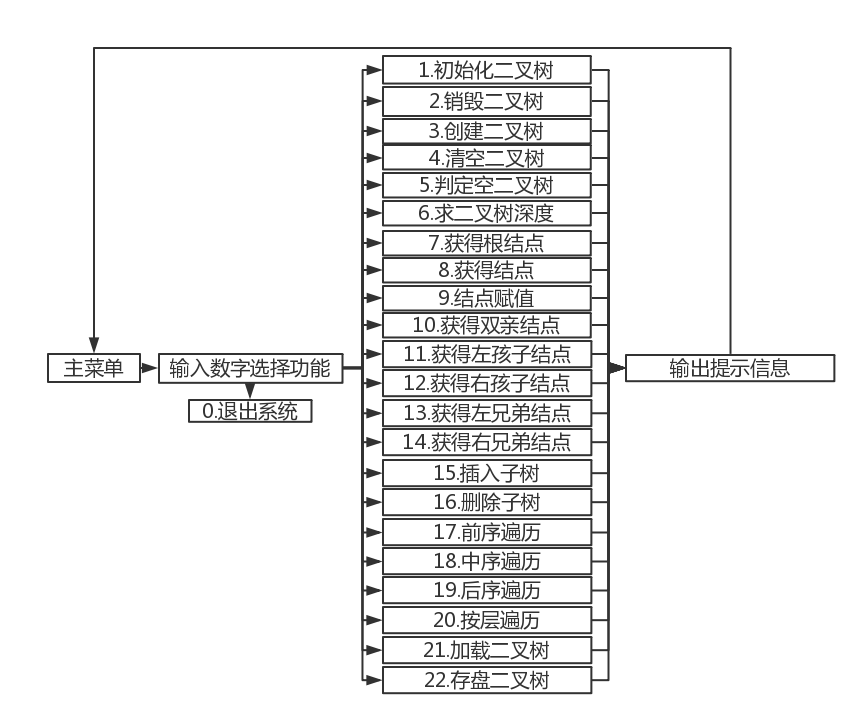


图3-1 系统设计结构图

* + 1. 数据文件存储格式

考虑到简洁高效的原则，按如下方式设计数据文件存储格式。

文件格式:dat，便于程序快速读取数据。

存放方式：顺序按相应的数据类型分配相应的文件空间进行存放，数据之间无间隔。

* + 1. 运算算法思想与设计

二叉树运算算法思想与设计如下：

1. 初始化二叉树算法思想：一开始需要判断合法性（L是否初始化）。函数的参数是树根结点指针L。为L分配1个节点空间，其key置为-1标记空树，左右孩子指针置NULL。

2. 销毁二叉树算法思想：一开始需要判断合法性（L是否初始化）。函数的参数是树根结点指针L，采取递归的方式先销毁二叉树的左子树，再销毁二叉树的右子树，最后用free函数释放掉当前结点对应的内存空间。

3. 创建二叉树算法思想：一开始需要判断合法性（L是否初始化）。函数的参数是树根结点指针L，按照先序次序输入二叉树中结点的值，如果第一个字符为#，则T为空二叉树。否则，malloc函数分配一个单元的空间作为树的根结点，并为其赋值，采取递归的方式继续创建根结点的左子树和右子树。流程图如图3-2 所示。

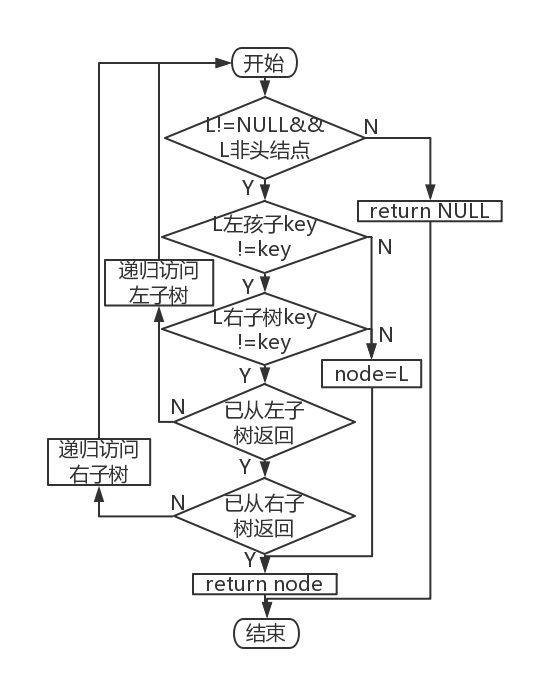
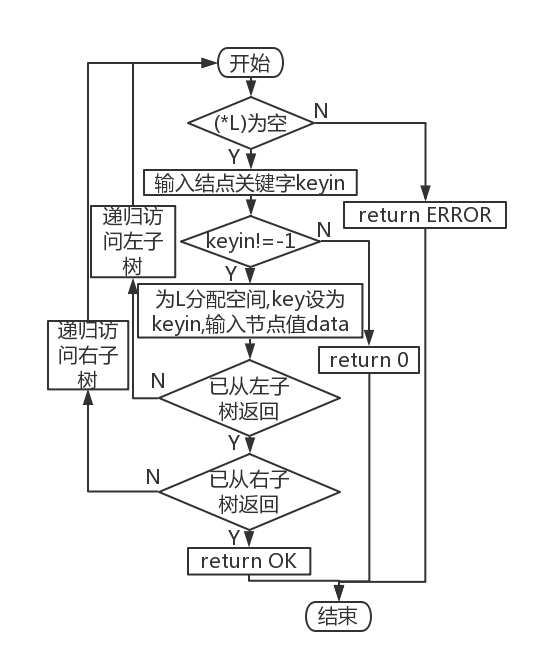


图3-2 创建二叉树流程图 图3-3 获得双亲节点流程图

4. 清空二叉树算法思想：函数的参数是树根结点指针L。将L指向的二叉树销毁，若销毁失败返回ERROR；L置空后再以其为根节点初始化新的空二叉树，若初始化失败返回ERROR。

5. 判定空二叉树算法思想：函数的参数是树根结点指针L。若L的关键字为-1，返回TRUE，否则返回FALSE。

6. 求二叉的深度算法思想：函数的参数是树根结点指针L。采取递归的方式求二叉树深度，先递归访问其左子树并返回左子树深度，再归访问其右子树并返回右子树深度。当前结点深度为左右字数深度中较大的一个加1。若左右子树均为空，则当前结点深度为0。若左右子树均已访问或为空，则向上一级函数返回当前的深度。

7. 获得二叉树根结点算法思想：函数的参数是树根结点指针L。直接返回根节点。

8. 获得结点算法思想：函数的参数是树根结点指针L和待查关键字key，将两个参数传入Search函数并返回指针M，若M非空则返回M所指结点的名称，否则返回0。

9. 结点赋值算法思想：函数的参数是树根节点指针L的地址，待修改结点关键字key和新的名称data，将L和key传入Search函数并返回指针M，若M非空则将M将其名称改为M，否则返回0。

10. 获得双亲结点算法思想：函数的参数是树根节点指针L，比对关键字key和树结点node的地址。若L的左孩子或右孩子的关键字与key匹配，将L赋给node并返回，否则，先递归访问其左孩子，再递归访问其右孩子。访问至叶结点时返回NULL。流程图如图3-3 所示。

11. 获得左孩子结点算法思想：函数的参数是树根节点指针L，比对关键字key和树结点node的地址。若L的关键字与key匹配，在L左孩子存在时将其赋给node并返回，不存在则返回NULL，否则，先递归访问其左孩子，再递归访问其右孩子。访问至叶结点时返回NULL。流程图如图3-4 所示。

12. 获得右孩子结点算法思想同上。

13. 获得左兄弟结点算法思想：函数的参数是树根节点指针L，比对关键字key和树结点node的地址。若L的右孩子关键字与key匹配，在L左孩子存在时将其赋给node并返回，不存在则返回NULL，否则，先递归访问其左孩子，再递归访问其右孩子。访问至叶结点时返回NULL。

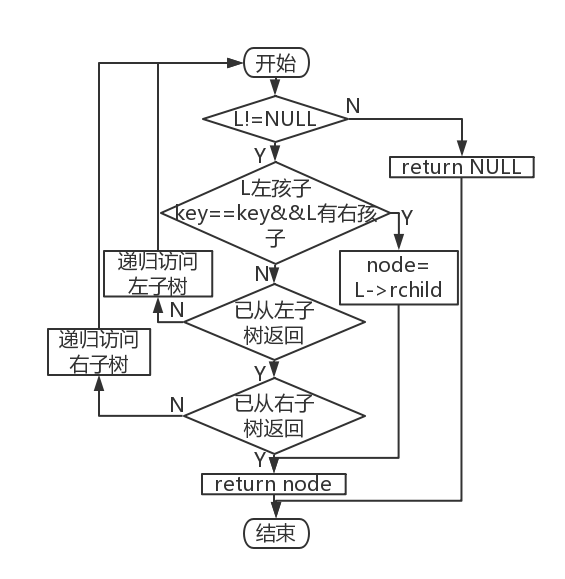
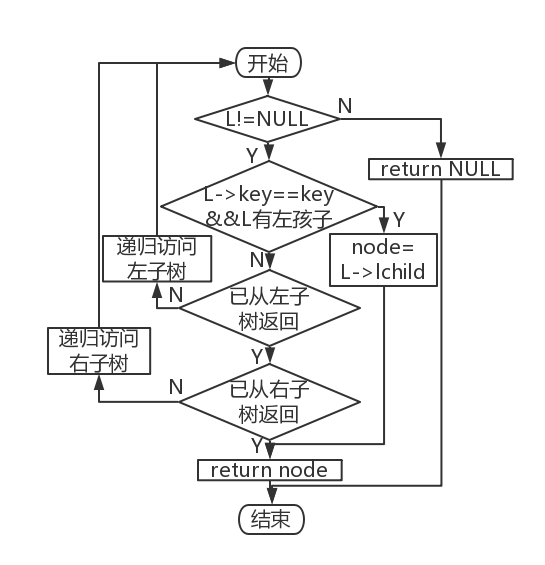


图3-4 获得左孩子流程图 图3-5 获得右兄弟流程图

14. 获得右兄弟结点算法思想同上。流程图如图3-5 所示。

15. 插入子树算法思想：函数的参数是树根节点指针L的地址，目标二叉树结点指针p, 定位变量LR和新树根结点指针c。在左子树插入时，若左子树非空则返回ERROR，否则将c赋给其左子树；在右子树插入时，若左子树非空则返回ERROR，否则将c赋给其右子树。

16. 删除子树算法思想：函数的参数是树根节点指针L的地址，目标二叉树结点指针p,和定位变量LR，销毁左子树时，若左子树已为空则返回FALSE，否则调用销毁二叉树函数销毁其左子树并将左孩子指针置空；销毁右子树时，若右子树已为空则返回FALSE，否则调用销毁二叉树函数销毁其右子树并将右孩子指针置空。流程图如图3-6 所示。

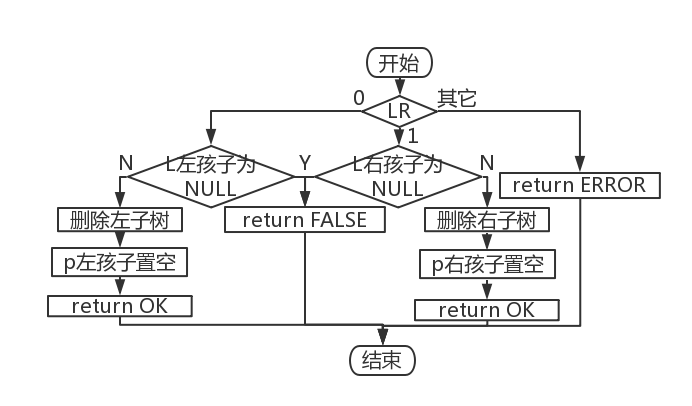


图3-6 删除子树流程图

17. 前序遍历算法思想：函数的参数是树根节点指针L。若L为空则向上一级函数返回0，否则打印其结点信息。若其左孩子非空，递归访问其左孩子；若其右孩子非空，递归访问其右孩子；

18. 中序遍历算法思想：函数的参数是树根节点指针L。若L为空则向上一级函数返回0。若其左孩子非空，递归访问其左孩子。打印其结点信息。若其右孩子非空，递归访问其右孩子。

19. 后序遍历算法思想：函数的参数是树根节点指针L。若L为空则向上一级函数返回0。若其左孩子非空，递归访问其左孩子；若其右孩子非空，递归访问其右孩子。最后打印其结点信息。

20. 层序遍历算法思想：函数的参数是树根节点指针L。建立一足够长的树结点结构数组，令树根节点为首元。定义两下标front和rear，q[front]有左孩子时将其赋给q[rear]，rear自增；q[front]有右孩子时将其赋给q[rear] ，rear自增。front自增。重复以上过程直至树中不再有新的节点被加入数组。流程图如图3-7 所示。

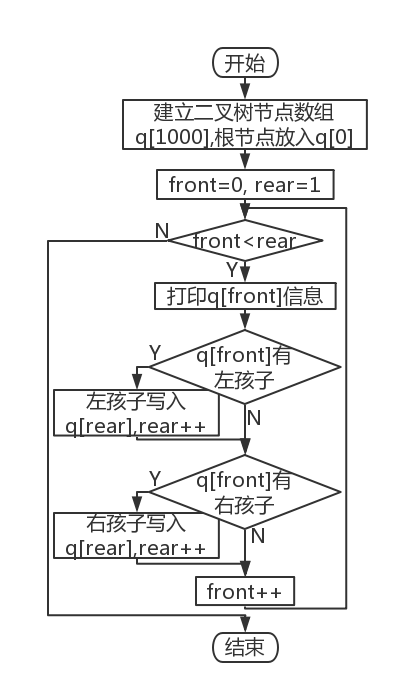


图3-7 层序遍历流程图

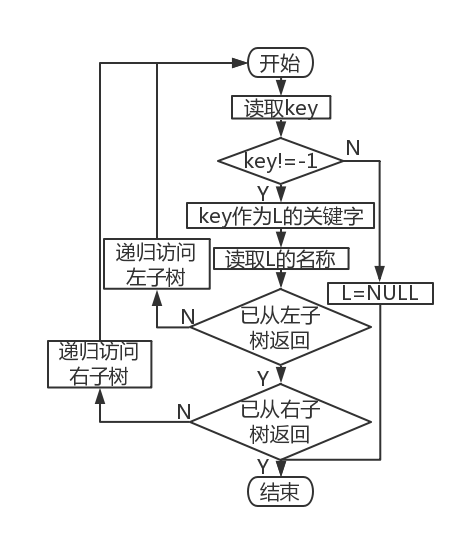
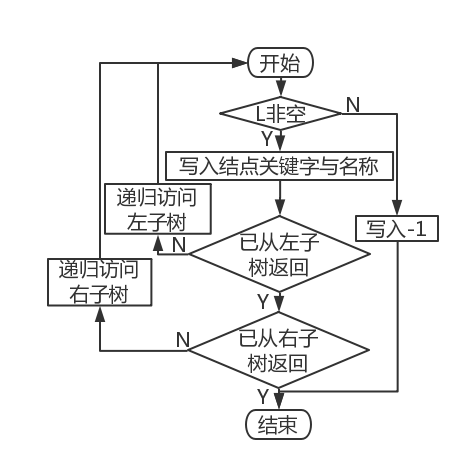


图3-8 数据存盘流程图 图3-9 数据读取流程图

21. 数据存盘算法思想：函数的参数是树根节点指针L和文件指针fp。若L为空，向文件中写入-1，否则写入节点关键字与名称。递归访问其左孩子，之后递归访问其右孩子，全部完成后向上一级返回OK。流程图如图3-8 所示。

22. 数据恢复算法思想：函数的参数是树根节点指针L和文件指针fp。读取当前节点的key，key为-1则将当前结点置空，否则创建当前节点并写入其关键字与名称。递归访问其左孩子，之后递归访问其右孩子，全部完成后向上一级返回OK。流程图如图3-9 所示。

* + 1. 算法时空复杂度分析

表3-1 算法时空复杂度分析

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 函数名 | 时间复杂度 | 空间复杂度 | 函数名 | 时间复杂度 | 空间复杂度 |
| InitBiTree | O（1） | O（1） | RightChild | O（n） | O（1） |
| DestroyBiTree | O（n） | O（1） | LeftSibling | O（n） | O（1） |
| CreateBiTree | O（n） | O（1） | RightSibling | O（n） | O（1） |
| ClearBiTree | O（n） | O（1） | InsertChild | O（1） | O（1） |
| BiTreeEmpty | O（1） | O（1） | DeleteChild | O（n） | O（1） |
| BiTreeDepth | O（n） | O（1） | PreOrderTraverse | O（n） | O（1） |
| Root | O（1） | O（1） | InOrderTraverse | O（n） | O（1） |
| Value | O（n） | O（1） | PostOrderTraverse | O（n） | O（1） |
| Assign | O（n） | O（1） | LevelOrderTraverse | O（n） | O（1） |
| Parent | O（n） | O（1） | LoadBiTree | O（n） | O（1） |
| LeftChild | O（n） | O（1） | SaveBiTree | O（n） | O（1） |

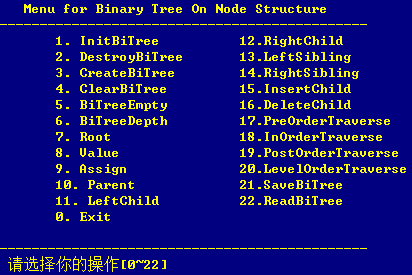


图3-10 演示系统菜单界面

## 3.3 系统实现

* + 1. 编程环境，运行环境，项目工程描述

本次实验采用Codeblocks编程软件编写，并用Codeblocks进行编译运行，项目名称是Binary-Tree-Sample。演示系统菜单界面如图3-10所示。

* + 1. 头文件及预定义常量说明

1.头文件

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

2.预定义常量

#define OK 1

#define TRUE 1

#define FALSE -1

#define ERROR -2

3.类型表达式

typedef int Status; //是函数的类型，其值是函数结果状态代码

typedef int ElemType; //数据元素类型定义

typedef struct BiTNode

{

ElemType key;

char data;

struct BiTNode \*lchild, \*rchild;//左、右子树指针

}BiTNode, \*BitTree;

* + 1. 测试计划

对系统的二叉树操作功能按以下计划测试：

1. 检测常规状态下系统的工作情况；
2. 重点检查非法边界操作时的工作情况

表3-2 演示系统测试计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 待测试功能 | 测试顺序 | 测试输入 | 预计输出 | 二叉树状态 |
| 1. InitBiTree | 1 | 无 | 初始化L成功 | 树深0，二叉树被创建 |
| 2. DestroyBiTree | 23 | 无 | 树销毁成功 | 不存在 |
| 3. CreateBiTree | 9 | 无 | 表清空成功 | 空表 |
| 4. ClearBiTree | 21 | 无 | 树清空成功 | 空树 |
| 5. BiTreeEmpty | 多次调用 | 无 | L未初始化的表时显示为未初始化，为空时显示空，非空时显示非空 | 与其前一步骤相同 |
| 6. BiTreeDepth | 5 | 无 | 二叉树深度为5 | 与步骤4相同 |
| 7. Root | 9 | 无 | 二叉树根的key为1 | 与步骤4相同 |
| 8. Value | 10 | 分别输入6和12 | 第一个输入应输出data为f，第二个输入输出查找失败 | 与步骤4相同 |
| 9. Assign | 11 | 分别输入2，@和85,23 | 第一个输入应输出赋值成功，第二个输入输出查找失败 | 与步骤4相同 |
| 10. Parent | 12 | 分别输入7，1和76 | 第一个输入输出双亲key为5，第二个输入输出无双亲，返回失败；最后一个输入输出未找到目标节点 | 与步骤4相同 |
| 11. LeftChild | 13 | 分别输入2,6和63 | 第一个输入输出左孩子key为3，第二个输入输出无左孩子，返回失败；最后一个输入输出未找到目标节点 | 与步骤4相同 |
| 12. RightChild | 14 | 分别输入1,4和51 | 第一个输入输出右孩子key为4，第二个输入输出无右孩子，返回失败；最后一个输入输出未找到目标节点 | 与步骤4相同 |
| 13. LeftSibling | 15 | 分别输入4,1和73 | 第一个输入输出左兄弟key为2，第二个输入输出无左兄弟，返回失败；最后一个输入输出未找到目标节点 | 与步骤4相同 |
| 14.RightSibling | 16 | 分别输入6,8和69 | 第一个输入输出右兄弟key为7，第二个输入输出无右兄弟，返回失败；最后一个输入输出未找到目标节点 | 与步骤4相同 |
| 15. InsertChild | 4，17，19 | 输入一：5-e;6-f;-1;-1;7-g;8-h;-1;-1;4;0  输入二：-1；2；-1  输入三：9—1；-1；-1；1；1  输入四：9-）；-1；10-=；-1；-1；2；1 | 对第一、第四个输入，应显示插入成功；后两个输入均显示插入失败 | 输入一后为图3-中的二叉树；输入二、三与步骤4中相同；输入四后为图3-中的二叉树 |
| 16. DeleteChild | 18 | 分别输入7-0,2-8和9 | 对于第一个输入输出删除成功；后两个输入均显示删除失败 | 图3- 中的树去掉节点8-h的状态 |
| 17. PreOrderTraverse | 3 | 无 | 前序下各个节点的key与data | 与步骤2相同 |
| 18. InOrderTraverse | 6 | 无 | 中序下各个节点的key与data | 与步骤4相同 |
| 19. PostOrderTraverse | 7 | 无 | 后序下各个节点的key与data | 与步骤4相同 |
| 20. LevelTraverse | 8 | 无 | 层序下各个节点的key与data | 与步骤4相同 |
| 21. SaveBiTree | 20 | 无 | 源文件目录下一个新的二进制数据文件，文件名为存盘操作的时间，格式为“星期\_月份缩写\_日期\_时\_分\_秒\_年” | 与步骤19相同 |
| 22. ReadBiTree | 2，22 | 二进制文件名 | 打开二进制文件成功应输出文件中存储的各个表，未找到对应文件或打开文件失败时输出错误提示信息 | 导入成功则为图3-/3-中的二叉树，失败则仅有一为空节点 |

* + 1. 测试

1．开始时系统中无表。此时对表进行任何形式的访问（函数2-22），都会提示此表未初始化.举例如下：



图3-11 DestroyBiTree出错提示（未初始化）



图3-12 CreateBiTree出错提示（未初始化）



图3-13 BiTreeDepth出错提示（未初始化）



图3-14 Parent出错提示（未初始化）



图3-15 ReadBiTree出错提示（未初始化）

2. 初始化表单。



图3-16 初始化二叉树

3. 此时对表进行任何形式的访问（函数4-21），都会提示此表为空.举例如下：



图3-17 BiTreeEmpty出错提示（空树）



图3-18 Value出错提示（空树）



图3-19 SaveBiTree出错提示（空树）

4．通过输入文件名导入数据，此时系统中共4个节点。



图3-20 输入文件名，导入数据成功

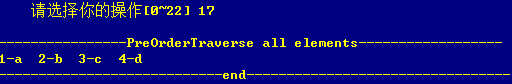


图3-21 先序遍历

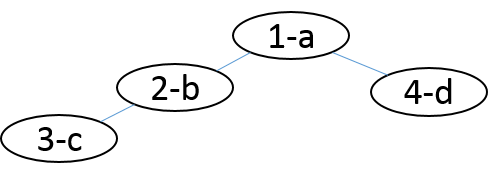


图3-22 此时的二叉树结构图

5. key为4的结点左子树下加一新子树。

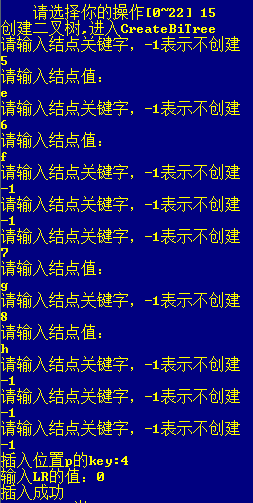


图3-23 更改操作表

6. 此时深度为5.



图3-24 判断表长

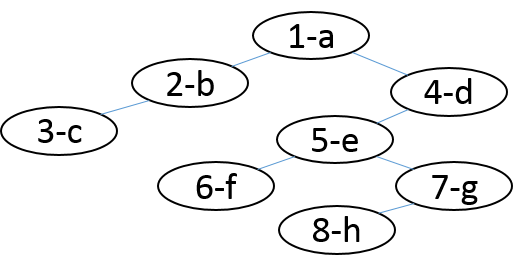


图3-25 此时的二叉树结构图

7. 中序遍历。

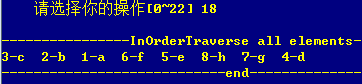


图3-26 中序遍历

8. 后序遍历。

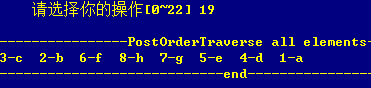


图3-27 后序遍历

9. 层序遍历。

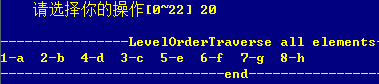


图3-28 层序遍历

图1-24，1-25，1-26遍历当前表，判断空表，求表长

10. 获取根节点。



图3-29 获取根节点

11. 查找当前表中节点。

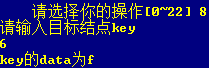


图3-30 查找节点成功

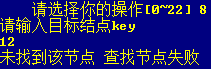


图3-31 查找节点失败

12. 修改节点data。

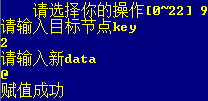


图3-32 修改节点data成功

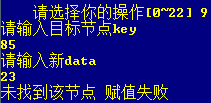


图3-33 修改节点data失败

13. 获取双亲节点。



图3-34 获取双亲节点成功



图3-35 获取双亲节点失败



图3-36 未找到此元素

14. 获取左孩子。



图3-37 获取左孩子节点成功



图3-38 获取左孩子节点失败



图3-39 未找到此元素

15. 获取右孩子。



图3-40 获取右孩子节点成功



图3-41 获取右孩子节点失败



图3-42 未找到此元素

16. 获取左兄弟。



图3-43 获取左兄弟节点成功



图3-44 获取左兄弟节点失败



图3-45 未找到此元素

17. 获取右兄弟。



图3-46 获取左兄弟节点成功



图3-47 获取左兄弟节点失败



图3-48 未找到此元素

18. 插入子树失败边界检查。

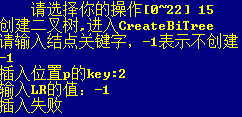


图3-49 插入子树失败（LR下标越界）

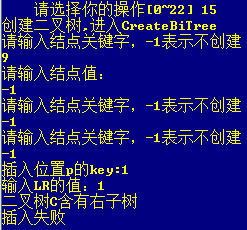


图3-50 插入子树失败（对应子树被占）

19. 删除子树。

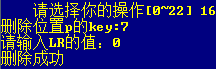


图3-51 删除子树成功



图3-52 删除子树失败



图3-53 未找到此元素

20. 插入新的二叉树后存盘。

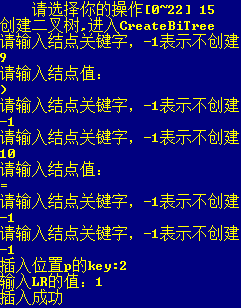


图3-54 插入二叉树

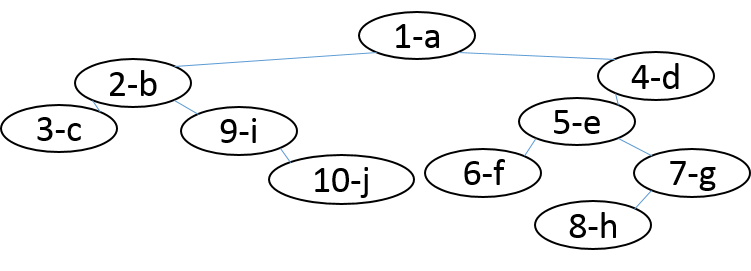


图3-55 此时的二叉树结构图



图3-56 文件存盘



图3-57 新存盘的文件

21. 清空二叉树。



图3-58 清空二叉树



图3-59 清空检查

22. 向系统中读取存盘的文件。



图3-60 文件读取成功

23. 层序遍历检查读入结果。

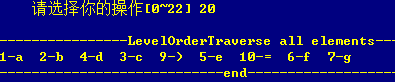


图3-61 层序遍历

24. 打开错误格式的文件时，显示文件打开失败。



图3-62 目标文件不可读取

25. 销毁二叉树。



图3-63 销毁二叉树成功

26. 退出系统。

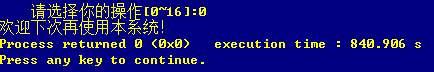


图3-64 退出系统

## 3.4 实验小结

这次实验感觉较之前几次的都要难一些，所以花的时间也比较多，中间写先序、中序、后序遍历算法比较纠结，问过同学后发现用递归算法很简单，之后很快就搞定了。这次实验学会了如何创建二叉树以及对二叉树的各类简单操作，对递归的思想更加熟悉，对指针的运用更加熟练，收获很大，来日方长，在编程的路上还会遇到很多困难，都要努力的寻找办法去解决，这是成为一个好的程序员的必经之路！

# 4 基于邻接表的图实现

## 4.1 问题描述

* + 1. 图的基本概念

图G是一个有序二元组(V,E)，其中V称为顶集(Vertices Set)，E称为边集(Edges set)，E与V不相交。它们亦可写成V(G)和E(G)。

E的元素都是二元组，用(x,y)表示，其中x,y∈V。

* + 1. 逻辑结构与基本运算

抽象数据类型二叉树的定义如下：

ADT BinaryTree {

数据对象D：D是具有相同特性的数据元素的集合。

数据关系R:

若D=Φ，则R=Φ，称BinaryTree为空二叉树；

若D≠Φ，则R={H},H是如下二元关系：

1. 在D中存在唯一的成为根的数据元素root，它在关系H中无前驱；
2. 若D-{root}≠Φ,则存在D-{root}={D1,Dr},且D1∩Dr=Φ;
3. 若D1≠Φ，则D1中存在唯一的元素X1，<root,X1>∈H，且存在D1上的关系H1包含于H；若Dr≠Φ,则Dr中存在唯一的元素Xr，<root,Xr>∈H,且存在Dr上的关系属于H；
4. （D，{H1}）是一棵符合本定义的二叉树，称为根的左子树，（Dr，{Hr}）是一棵符合本定义的二叉树，称为根的右子树。

}

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。

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

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

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

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

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

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

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

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

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

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

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

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

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

## 4.2 系统设计

* + 1. 数据物理结构

1.弧结点的数据物理结构

typedef struct ArcNode{

int adjvex; //该弧所指向的顶点序号

struct ArcNode \* nextarc; //指向下一条弧的指针

InfoType \*info; //该弧相关信息的指针（网中弧的弧长）

}ArcNode;

2.顶点结点的数据物理结构

typedef struct VNode{

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

char data; //顶点名称

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

3.图的数据物理结构

typedef struct{

AdjList vertices;

int vexnum, arcnum; //图的当前顶点数和弧数

int kind; //图的种类标志

}ALGraph;

在本程序中，数据原子类型VertexType被定义为char字符型，InfoType被定义为int\* 整形指针。

* + 1. 演示系统

采用邻接表作无向网的物理结构，实现§4.1.2的基本运算。其中VertexType为数据元素的类型名，具体含义可自行定义，其它有关类型和常量的定义和引用详见文献[1]的p163。

要求构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。

演示系统可选择实现无向网的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存图的数据逻辑结构(D,{R})的完整信息；②需要设计无向网文件保存和加载操作合理模式。附录B提供了文件存取的方法。

演示系统可选择实现多个无向网管理。

将菜单演示和用户选择输入写入到while循环中，用op获取用户的选择。Op初始化为1，以便第一次能进入循环。进入while循环后，系统首先显示功能菜单，然后提示用户输入选择（0-20），其中1-20对应二叉树的一个基本操作，分别对应一个函数，通过switch语句将用户输入的序号对应到相应的函数功能，执行完该语句后break跳出switch语句，执行while循环，直到用户输入0选择退出，退出系统。

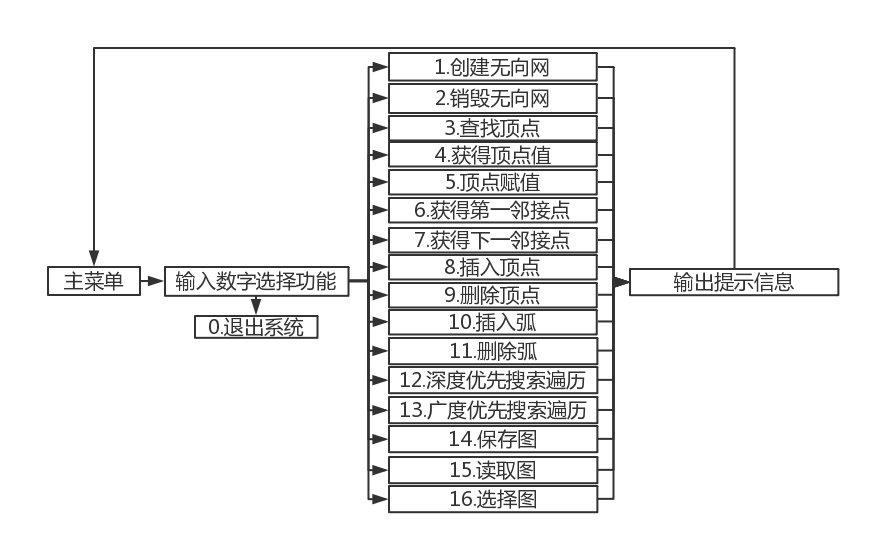


图4-1 系统设计结构图

* + 1. 数据文件存储格式

考虑到简洁高效的原则，按如下方式设计数据文件存储格式。

文件格式:dat，便于程序快速读取数据。

存放方式：顺序按相应的数据类型分配相应的文件空间进行存放，数据之间无间隔。

* + 1. 运算算法思想与设计

无向网运算算法思想与设计如下：

1.创建无向网算法思想：一开始需要判断合法性（图是否已创建）。函数的参数是主函数中所定义的图结构指针G。输入顶点数、边数及对应数目的顶点、弧以及相应信息（含弧长、弧头节点及弧尾节点的名称），注意在无向网条件下应在邻接表中同时建立逆邻接关系，即完成了无向网的初始化。流程图如图4-2 所示。

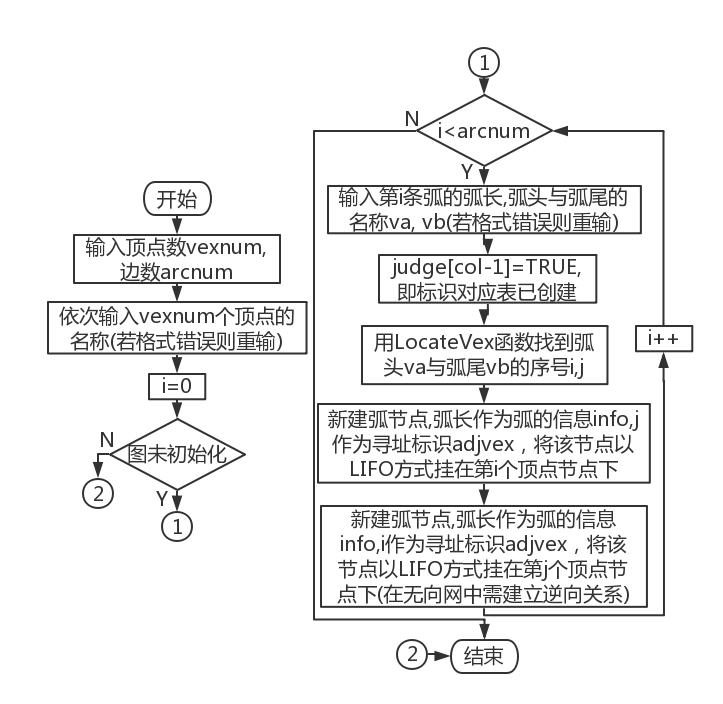


图4-2 创建无向网流程图

2.销毁无向网算法思想：一开始需要判断合法性（图是否已创建）。函数的参数是图结构指针G，若图中有顶点结点，则先删除顶点节点下的弧结点，在无向网情况下应同时删除弧长。将G的点数边数重置为0，即完成图的销毁。流程图如图4-3 所示。

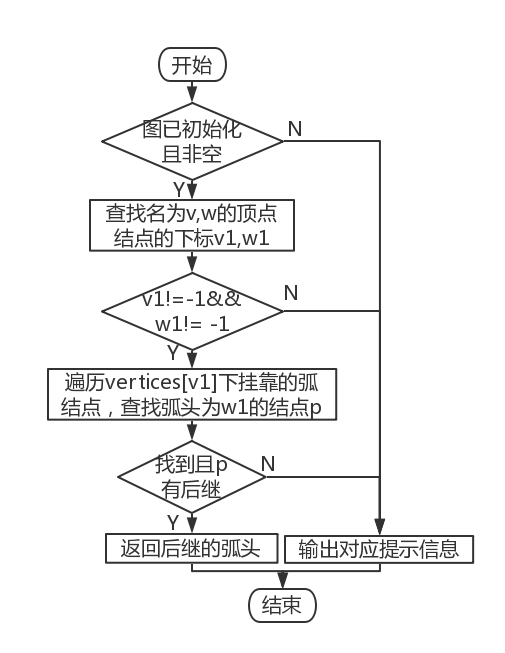
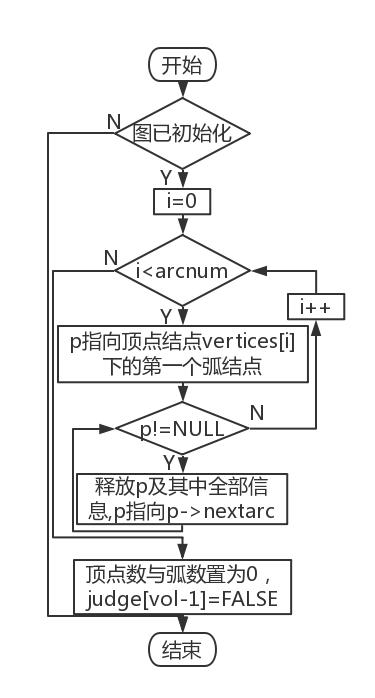


图4-3 销毁无向网流程图 图4-4 获得下一邻接点流程图

3. 查找顶点算法思想：一开始需要判断合法性（图是否已创建+图是否为空）。函数的参数是图结构指针G和待查顶点名称u，遍历顶点结点，查找与u具有相同名称的结点，查找成功返回结点下标，失败返回-1。

4. 获得顶点值算法思想：一开始需要判断合法性（图是否已创建+图是否为空+下标是否越界）。函数的参数是图结构指针G和待查顶点下标rec，直接返回下表对应顶点的名称。

5. 顶点赋值算法思想：一开始需要判断合法性（图是否已创建+图是否为空+下标是否越界）。函数的参数是图结构指针G，结点下标rec和相应结点新名称v。将下标对应的顶点结点名称赋值为v即完成。

6. 获得第一邻接点算法思想：一开始需要判断合法性（图是否已创建+图是否为空）。函数的参数是图结构指针G和结点名称v。先调用查找顶点函数查找名称为v的顶点结点的下标，不存在该顶点则返回ERROR；找到后判断其是否有挂靠弧结点，若有则返回第一弧结点的弧头，否则返回ERROR。

7. 获得下一邻接点算法思想：一开始需要判断合法性（图是否已创建+图是否为空）。函数的参数是图结构指针G，基准结点名称v和定位结点名称w。先调用查找顶点函数查找名称为v和w的顶点结点的下标，若其中一个未找到则返回ERROR。找到后在名称为v的顶点结点下查找弧头为w的下标的结点，若未找到返回ERROR。找到后返回其后继结点的下标，若无后继返回ERROR。流程图如图4-4 所示。

8. 插入顶点算法思想：一开始需要判断合法性（图是否已创建+图是否为空+顶点数是否已达最大值）。函数的参数是图结构指针G和新结点名称v。提取图中现有顶点数作为新结点的下标，将v赋给新开结点的data并将其指向弧结点的指针置空（新建结点均为孤立结点）即完成。

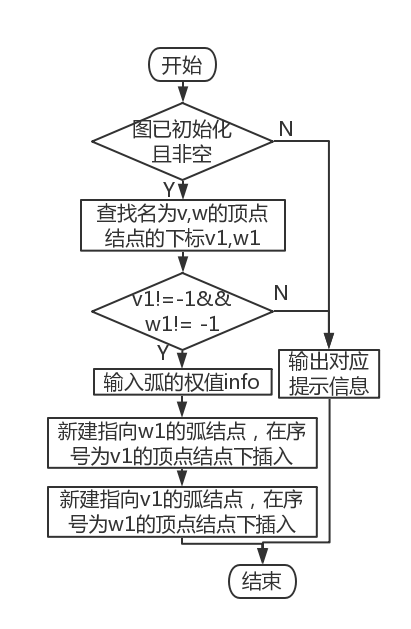
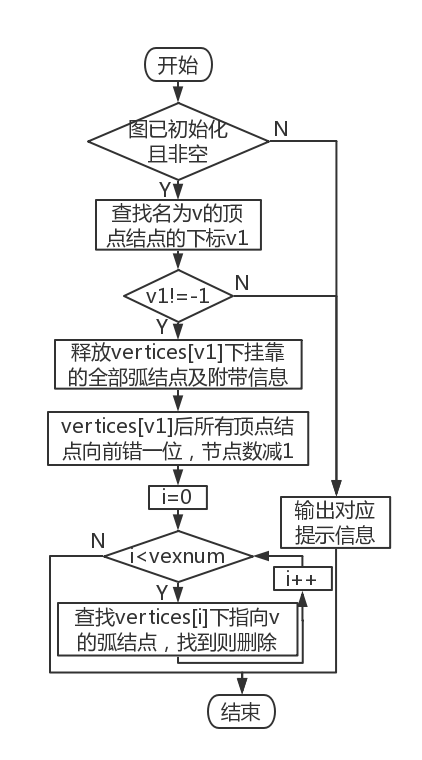


图4-5 删除顶点结点流程图 图4-6 插入弧结点流程图

9. 删除顶点算法思想：一开始需要判断合法性（图是否已创建+图是否为空）。函数的参数是图结构指针G和待删结点名称v。先调用查找顶点函数查找名称为v的顶点结点的下标，不存在该顶点则返回ERROR。删除所有目标结点下的弧结点，其后的顶点结点依次向前错一位以覆盖目标顶点结点，最后遍历邻接表，删除所有弧头名称为v的弧结点，即完成。流程图如图4-5 所示。

10. 插入弧算法思想：一开始需要判断合法性（图是否已创建+图是否为空）。函数的参数是图结构指针G，结点名称v和w。先调用查找顶点函数查找名称为v和w的顶点结点的下标，若其中至少一个未找到则返回ERROR。输入弧的权值，以v和w分别先后作为弧头和弧尾在表中建立对应的邻接关系，即完成。流程图如图4-6 所示。

11. 删除弧算法思想：一开始需要判断合法性（图是否已创建+图是否为空）。函数的参数是图结构指针G，结点名称v和w。先调用查找顶点函数查找名称为v和w的顶点结点的下标i和j，若其中至少一个未找到则返回ERROR。在下标为i的顶点结点下查找并删除指向j的弧结点；同样，在下标为j的顶点结点下查找并删除指向i的弧结点，即完成。

12. 深度优先搜索遍历算法思想：一开始需要判断合法性（图是否已创建+图是否为空）。函数的参数是图结构指针G。遍历顶点结点，运用辅助数组判断当前结点是否已访问，若未访问则调用深搜递归函数DFS。在DFS函数中，先将当前结对应的辅助数组元素置为TRUE（已访问），打印节点信息后通过挂靠弧结点查找其最近的未访问的顶点结点，照此递归，即完成。

13. 广度优先搜索遍历算法思想：一开始需要判断合法性（图是否已创建+图是否为空）。函数的参数是图结构指针G。先初始化队列，从首个顶点结点开始访问。将对应的辅助数组元素置为TRUE并打印节点信息后，该元素进入队列，只要队列非空便不断弹出队头元素，对每个队头，遍历其挂靠的弧结点，若弧结点对应的弧头未被访问，则将对应的辅助数组元素置为TRUE并打印节点信息，如此循环直到图中所有顶点结点对应的辅助数组元素为TRUE，即完成。

14 . 数据存盘算法思想：函数的参数是图结构数组List。由系统时间自动生成文件名，生成对应二进制文件。用一重循环统计非空的表的个数，若没有非空表则直接报错返回。将此数据存盘。对每个图，先存储图结构属性（类型+顶点数+边数），对每个顶点节点，调用函数求其出度（有向图）/或度（无向图）,与名称一同存储，接着遍历其挂靠的弧结点，存储弧头与弧长，即完成。流程图如图4-7 所示。

15. 数据恢复算法思想：函数的参数是图结构数组List。按文件名成功打开文件后先读取有效表的个数，对每个图，读取图结构属性（类型+顶点数+边数）对每个顶点节点，读取其出度（有向图）/或度（无向图），先后建立与出度相同个数的弧结点，读取其序号与信息并依次挂靠在顶点结点下，即完成。流程图如图4-8 所示。

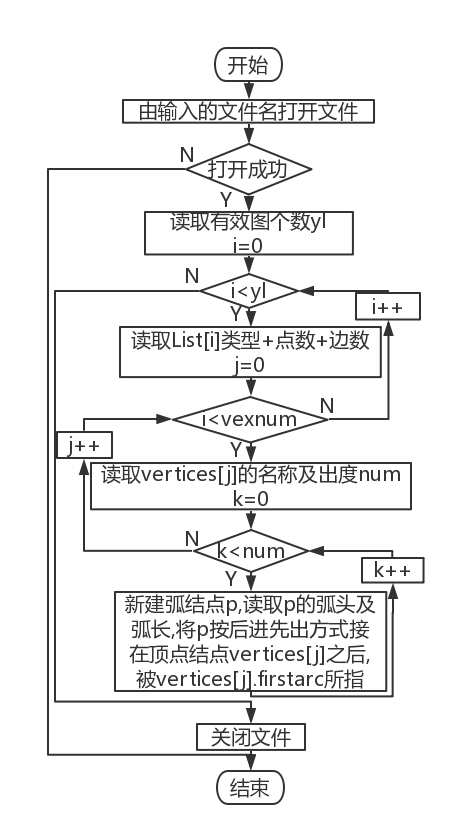
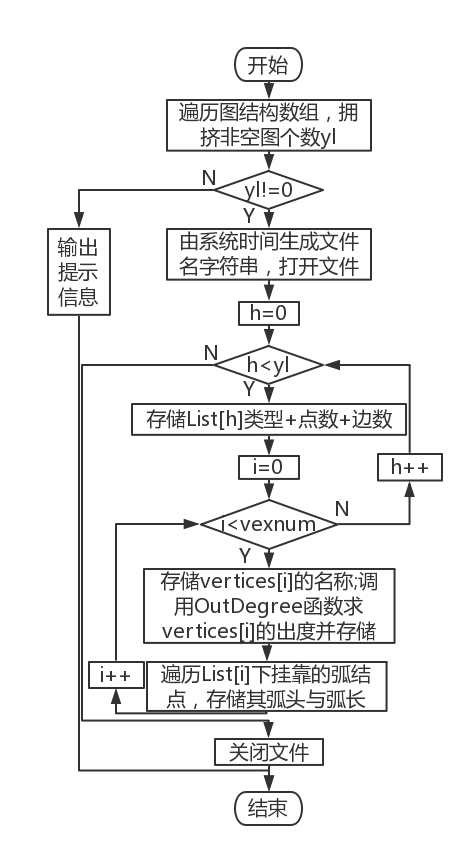


图4-7 数据存储流程图 图4-8 数据恢复流程图

16. 换表算法思想：函数的参数是目标图的位序col，图结构指针L的地址和图结构数组List。一开始需要判断合法性（图是否已创建+下标是否越界）。将要操作的表地址赋给同结构的指针，若此表未初始化则输出提示信息，即完成。

* + 1. 算法时空复杂度分析

表4-1 算法时空复杂度分析

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 函数名 | 时间复杂度 | 空间复杂度 | 函数名 | 时间复杂度 | 空间复杂度 |
| CreateGraph | O（n） | O（1） | DeleteVex | O（n2） | O（1） |
| DestroyGraph | O（n2） | O（1） | InsertArc | O（n） | O（1） |
| LocateVex | O（n） | O（1） | DeleteArc | O（n） | O（1） |
| GetVex | O（1） | O（1） | DFSTraverse | O（nn） | O（1） |
| PutVex | O（1） | O（1） | BFSTraverse | O（n3） | O（n） |
| FirstAdjVex | O（n） | O（1） | SaveGraph | O（n3） | O（1） |
| NextAdjVex | O（n） | O（1） | LoadGraph | O（n3） | O（1） |
| InsertVex | O（1） | O（1） | ChooseList | O（1） | O（1） |

## 4.3 系统实现

* + 1. 编程环境，运行环境，项目工程描述

本次实验采用Codeblocks编程软件编写，并用Codeblocks进行编译运行，项目名称是Adjacent-Chart-Based-MultiGraph。演示系统菜单界面如图4-9所示。



图4-9 演示系统菜单界面

* + 1. 头文件及预定义常量说明

1.头文件

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

2.预定义常量

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR -1

3.类型表达式

typedef int Status; //是函数的类型，其值是函数结果状态代码

typedef int QElemType; //对应顶点结点的下标

typedef struct QNode{

QElemType data; //节点信息

struct QNode \*next;

}QNode,\*QueuePtr; //队列节点结构

typedef struct{

QueuePtr front,rear; //队头、队尾指针

}LinkQueue; //队列结构

typedef int Boolean;

typedef int InfoType; //网中弧长

typedef char VertexType; //节点名称

typedef struct ArcNode{

int adjvex; //该弧所指向的顶点位置

struct ArcNode \* nextarc; //指向下一条弧的指针

InfoType \*info; //该弧相关信息的指针

}ArcNode; //弧结点结构

typedef struct VNode{

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

char data; //顶点信息

}VNode, AdjList[MAX\_VERTEX\_NUM]; //顶点结点结构

typedef struct{

AdjList vertices;

int vexnum, arcnum; //图的当前顶点数和弧数

int kind; //图的种类标志

}ALGraph; //图结构

* + 1. 测试计划

对系统的无向网操作功能按以下计划测试：

1. 检测常规状态下系统的工作情况；
2. 重点检查非法边界操作时的工作情况.

表4-2 演示系统测试计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 待测试功能 | 测试顺序 | 测试输入 | 预计输出 | 无向网状态 |
| 1. CreateGraph | 1,14 | 步骤1:  第一组输入:1;4;5;a b c d;1 a b;2 a c;3 a d;2 b c;1 b d  步骤14:  第二组输入:1;0;@  第三组输入:  3;2;e f g;2 e g;4 e f | 创建List[0] List[1] List[2]成功 | 步骤1:List[0]被创建  步骤14:  List[1]和List[2]被创建 |
| 2. DestroyGraph | 19 | 无 | List[2]销毁成功 | List[2]不存在 |
| 3. LocateVex | 2 | 第一个输入:a  第二个输入:\_ | 第一个输入：序号为0  第二个输入：查找失败 | 与步骤1相同 |
| 4. GetVex | 3 | 第一个输入:3  第二个输入:9 | 第一个输入：名称为d  第二个输入：查找失败 | 与步骤1相同 |
| 5. PutVex | 13 | 第一个输入:6 1  第二个输入:3 d | 第一个输入：修改失败  第二个输入：修改成功 | 名为e的结点改名为d |
| 6. FirstAdjVex | 5,7 | 步骤5:  第一个输入:e  第二个输入:q  步骤7:  第三个输入:e | 第一+二个输入:查找失败  第三个输入:第一邻接顶点序号为3 | 步骤5：与步骤:4相同  步骤7：与步骤6相同 |
| 7. NextAdjVex | 8 | 第一个输入:d e  第二个输入:e d  第三个输入:` /  第四个输入:e f  第五个输入:a e | 第一个输入:下一邻接点序号为1  第二至五个输入:返回失败与失败原因 | 与步骤6相同 |
| 8. InsertVex | 4 | e | 插入成功 | List[0]中加入节点e |
| 9. DeleteVex | 11,18 | 步骤11:  第一个输入: ？  第二个输入: d  步骤18:  第二个输入: @ | 第一个输入:不存在此顶点  第二个输入:删除成功  第三个输入:删除成功，图中已无顶点 | 步骤11:  List[0]中名为d的节点顶点与所有指向d的弧结点被删除  步骤18:  List[1]中无结点 |
| 10. InsertArc | 6 | 第一个输入: a g  第一个输入: e d 4 | 第一个输入:顶点v或w不存在  第二个输入:添加成功 | List[0]中加入长为4的弧e-d |
| 11. DeleteArc | 9 | 第一个输入: d f  第一个输入: e d | 第一个输入:顶点不存在  第二个输入:删除成功 | 与步骤4相同 |
| 12. DFSTraverse | 10，17 | 无 | 当前List的各顶点序号及名称 | 步骤10:与步骤4相同  步骤17: 与步骤14相同 |
| 13. BFSTraverse | 12 | 无 | List[0]各顶点序号及名称 | 与步骤11相同 |
| 14. SaveGraph | 15,20 | 无 | 程序中打印表的信息，程序根目录下名为存档时系统时间的dat文件 | 步骤15:  与步骤14相同  步骤20:  与步骤19相同 |
| 15. LoadGraph | 16 | 文件名字符串 | 程序中打印表的信息，读取成功 | 与步骤14相同 |
| 16. ChooseList | 多次调用 | 图的顺序，即目标图的下标减1 | 若表未初始化，输出提示信息 | 与前一步骤相同 |

* + 1. 测试

1．开始时系统中无图。此时对图进行任何形式的访问（函数2-15），都会提示此图未初始化.举例如下：



图4-10 DestroyGraph出错提示（未初始化）



图4-11 NextAdjVex出错提示（未初始化）



图4-12 InsertVex出错提示（未初始化）



图4-13 DFSTraverse出错提示（未初始化）

2. 创建图。图4-15中顶点结点数字为其下标，字母为其名称；弧结点左边数字为弧头下标，右边数字为弧长，后面的邻接表均遵从此种表示方式。

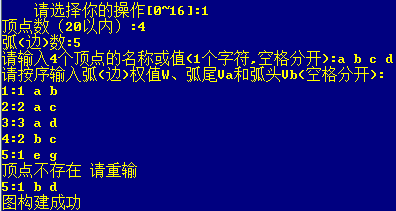


图4-14 初始化无向图，其中涉及查找顶点失败的特殊情况

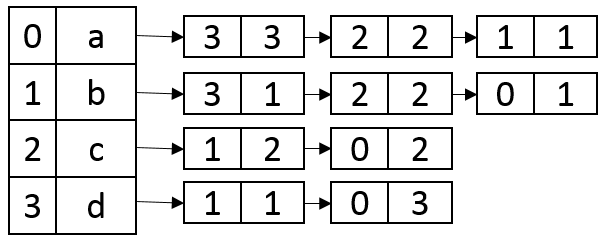


图4-15 List[0]下的邻接表

3. 此时再创建图，会提示此位置下已有图：



图4-16 SaveBiTree出错提示（空树）

4．查找结点。



图4-17 查找结点成功



图4-18 查找结点失败

5. 取结点名称。



图4-19 取结点名称成功



图4-20 取结点名称失败

6. 添加名称为e的结点。



图4-21 添加节点

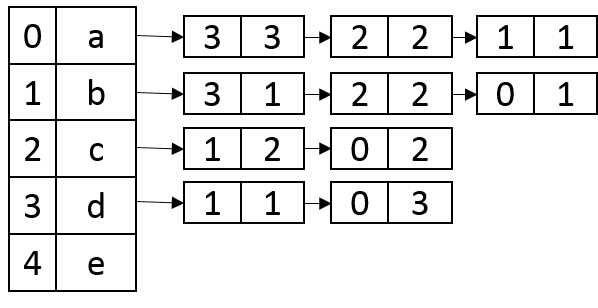


图4-22 List[0]下的邻接表

7. 此时e无邻接点。



图4-23 返回第一邻接结点



图4-24 查找基准结点失败

8. 插入长为4的弧e-d。



图4-25 插入弧失败

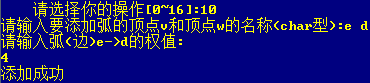


图4-26 插入长为4的弧e-d

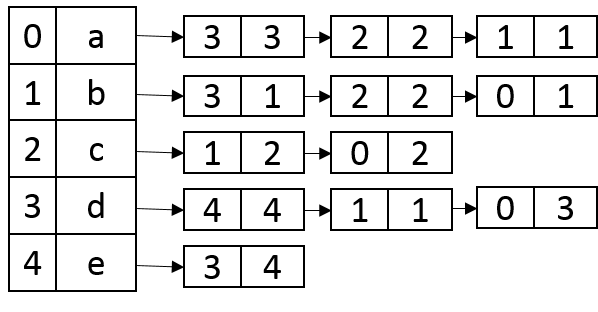


图4-27 List[0]下的邻接表

9. 检查插入情况，此时e只有d一个邻接节点。



图4-28 e的第一邻接点为d



图4-29 d相对e的下一邻接点为b

10. 返回下一邻接点的诸多边界条件。



图4-30 e相对d的下一邻接点不存在



图4-31 基准顶点v不存在



图4-32 定位顶点w不存在



图4-33 两顶点不邻接

11. 删除弧d-e,此时邻接表恢复至图4-22所示状态。



图4-34 删除弧失败



图4-35 删除弧成功

12. 深度优先搜索，e在最后被访问，证明弧删除成功。



图4-36 深度优先搜索

13. 删除结点d。



图4-37 删除结点失败



图4-38 删除结点成功

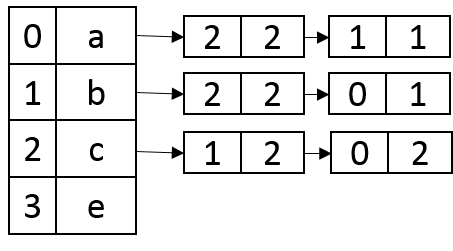


图4-39 List[0]下的邻接表

14. 广度优先搜索，证明结点d被成功删除。



图4-40 广度优先搜索

15. 将e重新赋值为d，之后添加弧

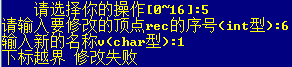


图4-41 顶点赋值失败

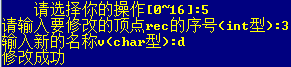


图4-42 顶点赋值成功

16. 切换操作对象，在List[1]和List[2]中先后添加图。



图4-43 操作List[1]

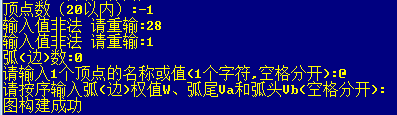


图4-44 List[1]中仅有一孤立顶点



图4-45 操作List[2]

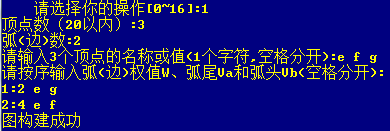


图4-46 创建List[2]

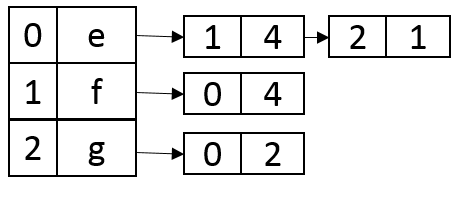


图4-47 List[2]下的邻接表

17. 保存多图。

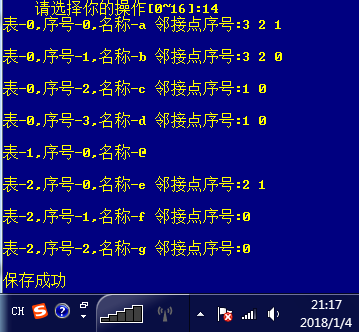


图4-48 保存多图



图4-49 程序根目录下的存档

18. 关闭程序再打开，从文件中恢复数据并检查。



图4-50 恢复数据失败

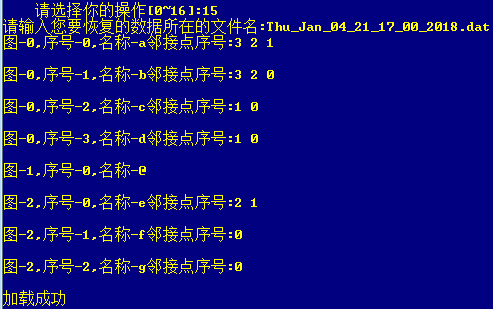


图4-51 恢复数据成功



图4-50 遍历List[0]



图4-52 遍历List[1]



图4-53 遍历List[2]

19. 删除List[1]中最后的顶点并检查删除效果。



图4-54 操作List[1]



图4-55 删除顶点



图4-56 检查删除结果

20. 销毁List[2]中的图。



图4-57 操作List[2]



图4-58 销毁List[2]

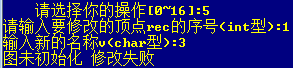


图4-59 检查销毁结果

21. 文件存盘。

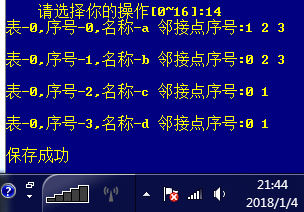


图4-60 文件存盘



图4-61 程序根目录下的文件

22. 退出系统。

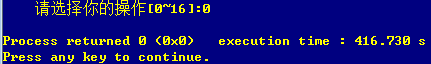


图4-62 插入二叉树

## 4.4 实验小结

这次实验基本能够熟练掌握关于图的各项基本操作，个人认为这次实验核心在于用邻接表储存图的结构，以及深度、广度优先遍历图的方法的掌握。实验过程中遇到些小问题，都通过问同学一一解决了。数据结构的实验到此差不多快要结束了，希望自己能通过寒假的课设更加系统的增强自己在编程这方面的能力，将数据结构课上所学加以实践，编程之路还很漫长，还需更加努力！

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

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

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

[4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

## 指导教师评定意见

一、对实验报告的评语

|  |
| --- |
|  |

二、对实验报告评分

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 评分项目  (分值) | 程序内容  (36.8分) | 程序规范  (9.2分) | 报告内容  (36.8分) | 报告规范  (9.2分) | 考勤  （8分） | 逾期扣分 | 合 计  (100分) |
| 得分 |  |  |  |  |  |  |  |