

**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201904

**学 号： U201913158**

**姓 名： 苏宇**

**指导教师： 袁凌**

**报告日期： 2020年 10月 9 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](#_Toc22083)

[1.1 问题描述 2](#_Toc24437)

[1.2 系统设计 4](#_Toc3487)

[1.3 系统实现 6](#_Toc335)

[1.4 系统测试 7](#_Toc22880)

[1.5 实验小结 12](#_Toc28346)

[2 基于链式存储结构的线性表实现 13](#_Toc28981)

[2.1 问题描述 13](#_Toc4250)

[2.2 系统设计 15](#_Toc12956)

[2.3 系统实现 16](#_Toc11766)

[2.4 系统测试 18](#_Toc12791)

[2.5 实验小结 25](#_Toc13627)

[3基于二叉链表的二叉树实现 26](#_Toc26952)

[3.1 问题描述 26](#_Toc4225)

[3.2 系统设计 29](#_Toc6630)

[3.3 系统实现 31](#_Toc25851)

[3.4 系统测试 33](#_Toc24734)

[3.5 实验小结 41](#_Toc24323)

[4基于邻接表的图实现 42](#_Toc14367)

[4.1 问题描述 42](#_Toc30553)

[4.2 系统设计 44](#_Toc17326)

[4.3 系统实现 46](#_Toc17299)

[4.4 系统测试 47](#_Toc19559)

[4.5 实验小结 56](#_Toc13007)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

采用顺序表作为线性表的物理结构，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。

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

### 1.1.1 12种基本运算定义

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

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

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

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

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

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

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

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

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

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

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

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

### 1.1.2 2种文件操作定义

(1)存储元素：函数名称是SaveList(SqList L, char FileName[])，初始条件是线性表L已经存在；操作结果：将线性表L的的元素写到FileName文件中

(2)读入元素：函数名称是LoadList(SqList &L,char FileName[])，初始条件是线性表L不存在：操作结果是：将FileName文件中的数据读入到线性表L中

### 1.1.3 3种集合运算定义

用顺序表来保存线性表的集合，实现多线性表操作。

(1)添加线性表：函数名称是AddList(LISTS& Lists, char ListName[])，初始条件是线性表集合已经存在；操作结果是对线性表集合中添加一个名为ListName的线性表

(2)删除线性表：函数名称是RemoveList(LISTS& Lists, char ListName[])，初始条件是线性表集合已经存在；操作结果是对线性表集合中删除一个名为ListName的线性表

(3)查找线性表：函数名称是LocateList(LISTS Lists, char ListName[])，初始条件是线性表集合已经存在；操作结果是在线性表集合中查找一个名称为ListName的线性表，成功返回逻辑序号

## 1.2 系统设计

### 1.2.1 顺序表存储结构定义

如下图1-2-1来定义顺序表的存储结构，顺序表类型包括三个元素整型指针elem、整型length、整型listsize，elem指向顺序表存储的元素数组的地址，length表示顺序表含有的数据元素的个数，listsize表示顺序表最大存储的数据元素的个数。

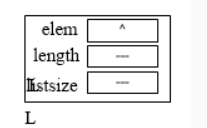


图1-2-1顺序表类型的存储结构

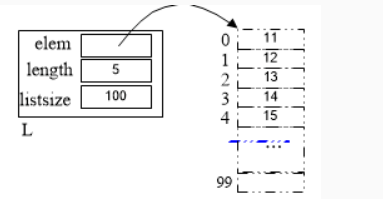


图1-2-2顺序表元素存储的示意图

### 1.2.2 线性表集合类型定义

如下图1-2-3中，线性表集合类型定义了整型指针elem，整型变量length，其中elem用来存储线性表集合所包含的线性表数组的地址，length表示线性表集合中含有的线性表个数。

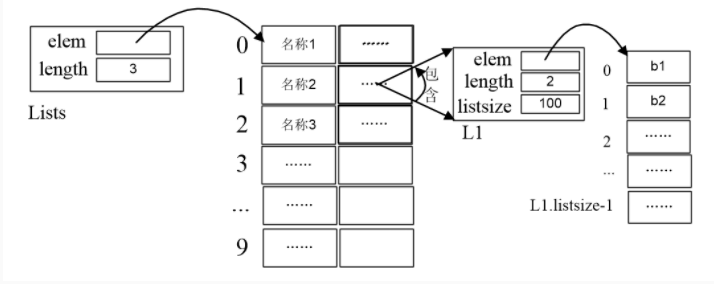


图1-2-3线性表集合存储类型示意图

### 1.2.3 各类函数的设计

文件读取函数LoadList(SqList &L,char FileName[])中调用了线性表的初始化函数InitList(SqList& L)。使用文件读取时初始化线性表并将文件中数据输入线性表。

线性表集合的添加操作AddList(LISTS& Lists, char ListName[])调用了线性表的初始化函数InitList(SqList& L)。

线性表的插入操作ListInsert(L,i,e)调用了ListLength(SqList L)函数。

删除ListDelete(SqList& L, int i, ElemType& e)调用了ListLength(SqList L)函数。

定位操作GetElem(SqList L, int i, ElemType& e)调用了ListLength(SqList L)函数。

获取前驱操作PriorElem(SqList L, ElemType e, ElemType& pre)调用了ListLength(SqList L)函数。

获取后继操作NextElem(SqList L, ElemType e, ElemType& next)调用了ListLength(SqList L)函数。

## 1.3 系统实现

### 1.3.1 重要函数的实现

1. InitList(SqList& L)：

将线性表的长度length设置为0，并为数据分配相应的内存，将内存地址赋给elem。设置线性表最大长度listsize。

时间复杂度为：O(1)

（2）DestroyList(SqList& L)

释放elem指向地址的内存空间，将elem指针置为NULL，将线性表的长度和分配大小置为0

时间复杂度为：O(1)

（3）ListEmpty(SqList L)

若线性表长度为0,则返回True；

时间复杂度为：O(1)

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

判断插入位置是否合理及其线性表是否存在，如果合理，则将第i个数据及其之后的数据全部后移，然后插入。

时间复杂度为：O(n)

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

若线性表L不存在，返回INFEASIBLE；否则删除线性表L的第i个元素，并将后面的元素一次前移，若删除成功则将删除的值赋给e并返回OK，若删除失败则返回ERROR。

时间复杂度为：O(n)

（6）PriorElem(SqList L, ElemType e, ElemType& pre)

线性表L不存在，返回INFEASIBLE；若没有找到指定元素e的前驱，则查找失败，返回ERROR；若查找成功，则将值赋给pre，并返回OK。

时间复杂度为：O(n)

（7）NextElem(SqList L, ElemType e, ElemType& next)

若线性表L不存在，返回INFEASIBLE；若没有找到指定元素e的后继，则查找失败，返回ERROR；若查找成功，则将值赋给next，并返回OK。

时间复杂度为：O（n）

（8）ListTraverse(SqList L)

如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，返回OK；如果线性表L不存在，返回INFEASIBLE。

时间复杂度为：O(n);

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

判断要删除的线性表是否为线性集合的最后一个线性表，如果是，直接调用线性表销毁函数。

查找线性表在集合中的位置，将处于该位置及之后的位置的线性前移，之后销毁线性表，返回OK。

时间复杂度为O（n）

## 1.4 系统测试

编译环境：visual c++ 6.0

使用集成IDE：visual 2019

### 1.4.1 线性表功能的操作

（1）InitList(SqList& L)

创建结果如图1-4-1-1



图1-4-1-1 线性表创建

1. DestroyList(SqList& L)

线性表销毁结果如图1-4-1-2

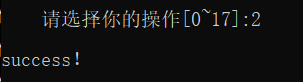


图1-4-1-2 线性表销毁

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

线性表原有数据 2、3、4、5、6

1. 在2之前插入数据56

如图1-4-1-3显示线性表插入后的结果

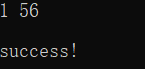


图1-4-1-3线性表插入操作

如图1-4-1-4显示线性表插入后遍历的结果



图1-4-1-4线性表遍历结果

1. 在6之后插入数据45

如图1-4-1-4显示线性表插入后的结果

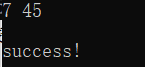


图1-4-1-5线性表插入操作

如图1-4-1-5显示线性表插入后遍历结果



图1-4-1-6线性表遍历结果

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

线性表原有数据56 、2 、3 、4 、5 、6 、45

1. 删除第一个位置的数据

如图1-4-1-7删除线性表第一位置后的结果

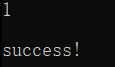


图1-4-1-7线性表删除操作

如图1-4-1-8显示线性表删除后遍历的结果



图1-4-1-8线性表遍历结果

1. 删除最后一个位置的数据

如图1-4-1-9删除线性表第一位置后的结果

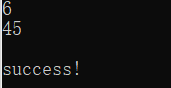


图1-4-1-9线性表删除操作

如图1-4-1-10显示线性表删除后遍历的结果



图1-4-1-10线性表遍历结果

1. ListTraverse(SqList L)

线性表原有数据2 、3 、4 、5 、6

如图1-4-1-11显示线性表遍历结果



图1-4-1-11线性表遍历结果

（6）ListEmpty(SqList L)

线性表原有数据2 、3 、4 、5 、6

线性表判空显示结果如图1-4-1-12



图1-4-1-12线性表判空结果

1. ListLength(SqList L)

线性表原有数据2 、3 、4 、5 、6

线性表长度显示结果如图1-4-1-13



图1-4-1-13线性表长度结果

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

线性表原有数据2 、3 、4 、5 、6

线性表得到第一个位置数据的结果如图1-4-1-14



图1-4-1-14线性表得到数据结果

1. LocateElem(SqList L, ElemType e)

线性表原有数据2 、3 、4 、5 、6

线性表查找数据6并返回其位置的结果如图1-4-1-15



图1-4-1-15线性表取得数据位置结果

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

线性表原有数据2 、3 、4 、5 、6

1. 取得第一个元素2的前驱

运行结果如图1-4-1-16



图1-4-1-16线性表取得前驱数据

1. 取得最后一个元素6的前驱

运行结果如图1-4-1-17



图1-4-1-17线性表取得前驱数据

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

线性表原有数据2 、3 、4 、5 、6

1. 取得第一个元素2的后驱

运行结果如图1-4-1-18



图1-4-1-18线性表取得数据后驱

1. 取得最后一个元素6的后驱

运行结果如图1-4-1-19



图1-4-1-19线性表取得后数据

（11）ClearList(SqList& L)

运行结果见图1-4-1-20



图1-4-1-20清除线性表的操作

### 1.4.2 线性表文件的操作

（1）SaveList(SqList L, char FileName[])

线性表原有数据2 、3 、4 、5 、6

运行结果如图1-4-1-21



图1-4-1-21线性表保存

保存结果在文件中如图1-4-1-22



图1-4-1-22文件中的数据

（2）LoadList(SqList& L, char FileName[])

文件中的数据见图1-4-1-23



图1-4-1-23文件中的数据

遍历结果见图1-4-1-24



图1-4-24导入数据后遍历的结果

### 1.4.3 线性表集合的操作

（1）AddList(LISTS& Lists, char ListName[])

运行结果如图1-4-1-25

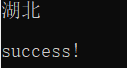
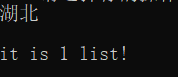


图1-4-1-25插入线性表

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

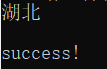
寻找一个名称的线性表返回其位置，如图1-4-1-26



如图1-4-1-26寻找线性表

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

删除一个名称的线性表，运行结果如图1-4-1-27



如图1-4-1-27删除一个线性表

## 1.5 实验小结

本次实验遇到的困难是线性表集合操作中删除某一名称的线性表这一操作。因为这一操作要满足删除线性表后，其余的线性表需要前移，而困难在于线性表集合中保存线性表的elem类型是指针，所以常规的将数据前移之后再删除的做法行不通，经过长时间的思考，最后我将要删除地元素保存在一个指针中，在所有元素前移后再将该指针指向的地址内存释放。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

采用顺序表作为线性表的物理结构，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。

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

### 2.1.1 12种基本运算定义

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

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

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

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

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

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

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

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

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

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

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

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

采用顺序表作为线性表的物理结构，实现相关的基本运算。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。

### 2.1.2 2种文件操作定义

(1)存储元素：函数名称是SaveList(SqList L, char FileName[])，初始条件是线性表L已经存在；操作结果：将线性表L的的元素写到FileName文件中

(2)读入元素：函数名称是LoadList(SqList &L,char FileName[])，初始条件是线性表L不存在：操作结果是：将FileName文件中的数据读入到线性表L中

### 2.1.3 5种集合运算定义

(1)添加线性表：函数名称是AddList(LISTS& Lists, char ListName[])，初始条件是线性表集合已经存在；操作结果是对线性表集合中添加一个名为ListName的线性表

(2)删除线性表：函数名称是RemoveList(LISTS& Lists, char ListName[])，初始条件是线性表集合已经存在；操作结果是对线性表集合中删除一个名为ListName的线性表

(3)查找线性表：函数名称是LocateList(LISTS Lists, char ListName[])，初始条件是线性表集合已经存在；操作结果是在线性表集合中查找一个名称为ListName的线性表，成功返回逻辑序号

（4）显示线性表名：函数名称是show\_lists(LISTS Lists)，初始条件是线性表集合已经存在；操作结果是返回线性表集合中所有线性表的名称

（5）初始化集合：函数名称是IinitList(LISTS Lists),初始条件是线性表集合不存在；操作结果是将文件中的所有数据存入线性表集合

## 2.2 系统设计

### 2.2.1 线性表类型定义

如图2-2-1定义链表的存储结构



图2-2-1链表的存储结构

### 2.2.2 线性表集合类型定义

如下图2-2-2中，线性表集合类型定义了整型指针elem，整型变量length，其中elem用来存储线性表集合所包含的线性表数组的地址，length表示线性表集合中含有的线性表个数。

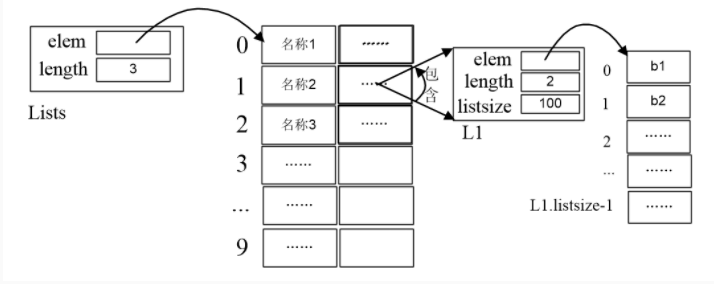


图2-2-2线性表集合的存储结构

链表集合的管理菜单中包含了对单链表的操作的switch菜单。

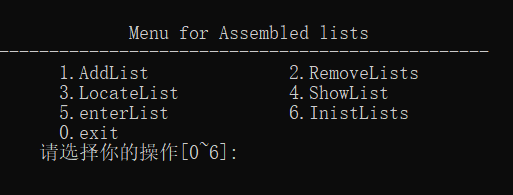


图2-2-3线性表集合操作界面

链表集合的创建操作包括了文件读取操作，以及调用链表的初始化函数InitList(SqList& L)。

### 2.2.3 各类函数的设计

链表的ListInsert(L,i,e)调用ListLength(L)函数。

取元素操作GetElem(LinkList L, int i, ElemType& e)函数调用ListLength(L)函数

获得元素位置操作LocateElem(LinkList L, ElemType e)函数调用ListLength(L)函数。

取前驱元素操作PriorElem(LinkList L, ElemType e, ElemType& pre)函数调用ListLength(L)函数。

取后驱元素操作NextElem(LinkList L, ElemType e, ElemType& next)函数调用调用ListLength(L)函数。

## 2.3 系统实现

### 2.3.1 重要功能的实现

（1）InitList(LinkList& L)

将线性表的长度设置为0，并为头指针分配相应的内存，头指针next设为null。

时间复杂度为：O(1)

（2）DestroyList(LinkList& L)

逐次遍历将节点对应的内存释放。

时间复杂度为：O(n)

（3）ListEmpty(LinkList L)

若线性表长度为0,则为空。

时间复杂度为：O(1)

（4）ListInsert(LinkList& L, int i, ElemType e)

判断插入位置是否合理及其线性表是否存在，如果合理，到达插入位置，将待插入的结点的next指向被插入位置的结点，再让插入位置的前一个结点的前驱的next指针指向待插入的结点 。

时间复杂度为：O(n)

（5）ListDelete(LinkList& L, int i, ElemType& e)

若线性表L不存在，返回INFEASIBLE；若存在，则遍历到删除位置的前一个结点，保存删除位置的结点并释放，让插入位置前的结点的next指向插入位置后的结点。若删除成功则将删除的值赋给e并返回OK，若删除失败则返回ERROR。

时间复杂度为：O(n)

（6）PriorElem(LinkList L, ElemType e, ElemType& pre)

线性表L不存在，返回INFEASIBLE；若没有找到指定元素e的前驱，则查找失败，返回ERROR；若查找成功，则将值赋给pre，并返回OK。

时间复杂度为：O(n)

（7）NextElem(LinkList L, ElemType e, ElemType& next)

若线性表L不存在，返回INFEASIBLE；若没有找到指定元素e的后继，则查找失败，返回ERROR；若查找成功，则将值赋给next，并返回OK。

时间复杂度为：O（n）

（8）ListTraverse(LinkList L)

若线性表L不存在，返回INFEASIBLE；否则输出线性表的每一个元素，并返回OK。

时间复杂度为：O(n);

## 2.4 系统测试

编译环境：visual c++ 6.0

使用集成IDE：visual 2019

### 2.4.1线性表集合的操作

1. InitList(SqList& L)

初始化线性表集合结果如图2-4-1-1



图2-4-1-1初始化线性表集合结果

1. show\_lists(LISTS Lists)

外部txt文件中有5个线性表，名字是北京、上海、广州、深圳、湖南

执行结果如图2-4-1-2

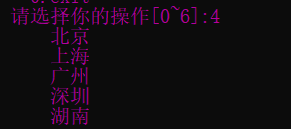


图2-4-1-2显示线性表集合中的线性表

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

线性表集合中有5个线性表，名字是北京、上海、广州、深圳、湖南

输入北京，显示其在线性表集合中的位置如图2-4-1-3

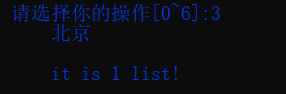


图2-4-1-3获取线性表在集合中的位置

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

线性表集合中有5个线性表，名字是北京、上海、广州、深圳、湖南

移除一个名字叫广州的线性表，执行结果如图2-4-1-4

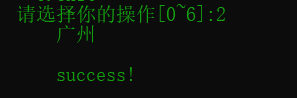


图2-4-1-4移除一个线性表

1. AddList(LISTS& Lists, char ListName[])

线性表集合中有4个线性表，名字是北京、上海、深圳、湖南

加入一个叫湖北的线性表，执行结果如图2-4-1-5

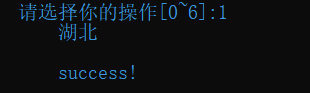


图2-4-1-5添加线性表

线性表中有的线性表名称见图2-4-1-6

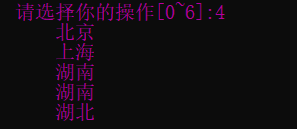


图2-4-1-6遍历线性表集合

1. enterList

选择进入名为湖南的线性表，执行结果如图2-4-1-7

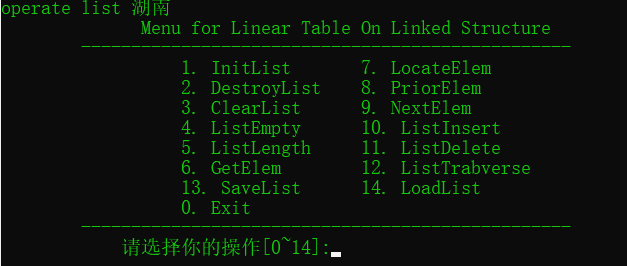


图2-4-1-7进入一个线性表

### 2.4.2 线性表功能的操作

（1）ListTraverse(LinkList L)

遍历结果如图2-4-2-1



图2-4-2-1线性表遍历

（2）ListLength(LinkList L)

线性表中含有数据45、78 、34 、98

线性表长度结果见图2-4-2-2



图2-4-2-2线性表长度

（3）GetElem(LinkList L, int i, ElemType& e)

线性表中含有数据45、78 、34 、98

线性表得到第一个位置数据的结果如图2-4-2-3



图2-4-2-3线性表得到数据结果

1. LocateElem(LinkList L, ElemType e)

线性表中含有数据45、78 、34 、98

线性表查找数据45并返回其位置的结果图2-4-2-4

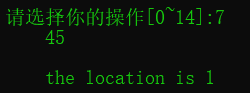


图2-4-2-4线性表取得数据位置结果

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

线性表中含有数据45、78 、34 、98

1. 取得第一个元素45的前驱

运行结果如图2-4-2-5

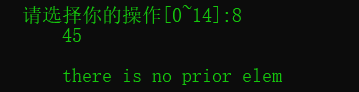


图2-4-2-5线性表取得前驱数据结果

1. 取得最后一个元素98的前驱

运行结果如图2-4-2-6

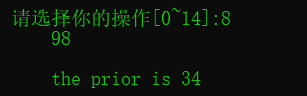


图2-4-2-6线性表取得前驱结果

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

线性表中含有数据45、78 、34 、98

1. 取得第一个元素45的后驱

运行结果如图2-4-2-7

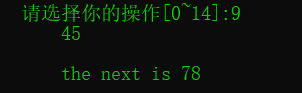


图2-4-2-7线性表取得数据后驱

1. 取得最后一个元素98的后驱

运行结果如图2-4-2-8

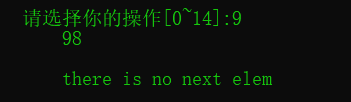


图2-4-2-8线性表取得数据后驱

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

线性表中含有数据45、78 、34 、98

1. 在45之前插入数据23

如图2-4-2-9显示线性表插入后的结果

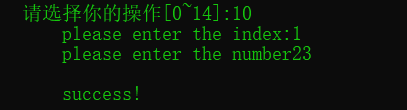


图2-4-2-9线性表插入操作

如图2-4-2-10显示线性表插入后遍历的结果



图2-4-2-10线性表遍历结果

1. 在98之后插入数据4

如图2-4-2-11显示线性表插入后的结果

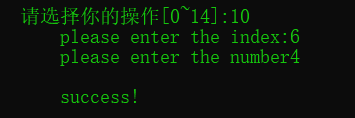


图2-4-2-11线性表插入操作

如图2-4-2-12显示线性表插入后遍历结果



图2-4-2-12线性表遍历结果

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

线性表原有数据23、45、78 、34 、98、 4

1. 删除第一个位置的数据

如图2-4-2-13删除线性表第一位置后的结果

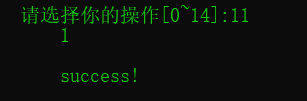


图2-4-2-13线性表删除操作

如图2-4-2-14显示线性表删除后遍历的结果



图2-4-2-14线性表遍历结果

1. 删除最后一个位置的数据

如图2-4-2-15删除线性表第一位置后的结果

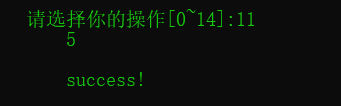


图2-4-2-15线性表删除操作

如图2-4-2-16显示线性表删除后遍历的结果



图2-4-2-16线性表遍历结果

1. ListEmpty(LinkList L)

线性表中含有数据45、78 、34 、98

线性表判空显示结果如图2-4-2-17

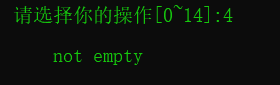


图2-4-2-17线性表判空

1. ClearList(LinkList& L)

线性表中含有数据45、78 、34 、98

运行结果如图2-4-2-18

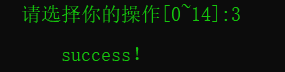


图2-4-2-18线性表清空

1. DestroyList(LinkList& L)

运行结果如图2-4-2-19

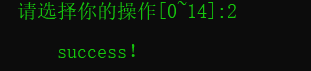


图2-4-2-19线性表销毁

### 2.4.3 线性表文件的操作

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

文件中数据如图2-4-2-20



图2-4-2-20文件中数据

运行结果如图2-4-2-21

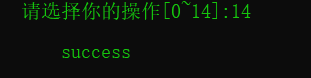


图2-4-2-21导入文件中数据结果

遍历结果如图2-4-2-22



图2-4-2-22导入数据后遍历的结果

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

线性表中数据2、 9、 8、 5、 45、 34

线性表保存数据如图1-4-2-23

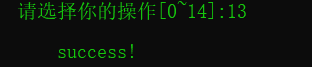


图1-4-2-23线性表保存数据

文件中数据如图2-4-2-24



图2-4-2-24文件中数据

## 2.5 实验小结

本次实验遇到的困难是如何将文件中的数据读取到线性表集合中以及如何用一个线性表读取文件并保存数据到文件，这三个困难都涉及到文件操作这一知识点，C语言的文件操作需要明白文件读取时文件指针的运动位置及文件指针返回值，通过调试和查询最终找到了解决方案。

# 3基于二叉链表的二叉树实现

## 3.1 问题描述

采用二叉链表表作为二叉树的物理结构，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。

依据最小完备性和常用性相结合的原则，以函数形式定义了二叉树的创建二叉树、销毁二叉树、清空二叉树、判定空二叉树和求二叉树深度等14种基本运算。

### 3.1.1 14种基本运算定义

⑴初始化二叉树：函数名称是InitBiTree(BiTree& T)；初始条件是二叉树T不存在，操作结果是构建一个空的二叉树。

⑵销毁二叉树：函数名称是DestroyBiTree(BiTree& T);；初始条件是二叉树T已存在；操作结果是销毁二叉树T，将二叉树头结点设置为空。

⑶清空二叉树：函数名称是ClearBiTree(BiTree& T)；初始条件是二叉树T存在；操作结果是将二叉树设置成空，并删除除头结点的所有结点，释放结点空间。

⑷判定空二叉树：函数名称是BiTreeEmpty(BiTree T)；初始条件是二叉树头结点不为空，操作结果是若首节点为空二叉树则返回TRUE，否则返回FALSE。

⑸求二叉树深度：函数名称是BiTreeDepth(BiTree T)；初始条件是二叉树T存在；操作结果是返回T的深度。

⑹查找结点：函数名称是LocateNode(BiTree T, KeyType e)；初始条件是二叉树T已存在，e是和T中结点关键字类型相同的给定值；操作结果是返回查找到的结点指针，如无关键字为e的结点，返回NULL。

⑺结点赋值：函数名称是Assign(BiTree& T, KeyType e, TElemType value)；初始条件是二叉树T已存在，e是和T中结点关键字类型相同的给定值；操作结果是关键字为e的结点赋值为value。

⑻获得兄弟结点：函数名称是GetSibling(BiTree T, KeyType e)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值；操作结果是返回关键字为e的结点的（左或右）兄弟结点指针。若关键字为e的结点无兄弟，则返回NULL。

⑼插入结点：函数名称是InsertNode(BiTree& T, KeyType e, int LR, TElemType c)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值，LR为0或1，c是待插入结点；操作结果是根据LR为0或者1，插入结点c到T中，作为关键字为e的结点的左或右孩子结点，结点e的原有左子树或右子树则为结点c的右子树。

特殊情况，c插入作为根结点可以考虑LR为-1时，作为根结点插入，原根结点作为c的右子树。

⑽删除结点：函数名称是DeleteNode(BiTree& T, KeyType e)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值。操作结果是删除T中关键字为e的结点；同时，如果关键字为e的结点度为0，删除即可；如关键字为e的结点度为1，用关键字为e的结点孩子代替被删除的e位置；如关键字为e的结点度为2，用e的左孩子代替被删除的e位置，e的右子树作为e的左子树中最右结点的右子树。

⑾前序遍历：函数名称是PreOrderTraverse(BiTree T, void (\*visit)(BiTree))；初始条件是二叉树T存在，Visit是一个函数指针的形参（可使用该函数对结点操作）；操作结果：先序遍历，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。

⑿中序遍历：函数名称是InOrderTraverse(BiTree T, void (\*visit)(BiTree))；初始条件是二叉树T存在，Visit是一个函数指针的形参（可使用该函数对结点操作）；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。

⒀后序遍历：函数名称是PostOrderTraverse(BiTree T, void (\*visit)(BiTree))；初始条件是二叉树T存在，Visit是一个函数指针的形参（可使用该函数对结点操作）；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。

⒁按层遍历：函数名称是LevelOrderTraverse(BiTree T, void (\*visit)(BiTree))；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。

### 3.1.2 2种文件操作定义

(1)存储元素：函数名称是SaveBiTree(BiTree T, char FileName[])，初始条件是二叉树T已经存在；操作结果：将线性表L的的元素写到FileName文件中。

(2)读入元素：函数名称是LoadBiTree(BiTree& T, char FileName[])，初始条件是二叉链表头结点T不存在：操作结果是：将FileName文件中的数据读入到线性表L中。

### 3.1.3 5种集合运算定义

(1)添加二叉链表：函数名称是AddBiTree(FOREST& forest, char ListName[])，初始条件是二叉链表集合已经存在；操作结果是对二叉链表集合中添加一个名为ListName的二叉链表。

(2)删除二叉链表：函数名称是RemoveBiTree(FOREST& forest, char ListName[])，初始条件是二叉链表集合已经存在；操作结果是对线性表集合中删除一个名为ListName的二叉链表。

(3)查找线性表：函数名称是LocateBiTree(FOREST forest, char ListName[])，初始条件是二叉链表集合已经存在；操作结果是在二叉链表集合中查找一个名称为ListName的线性表，成功返回其逻辑序号。

（4）显示二叉链表名：函数名称是ShowBiTree(FOREST forest)，初始条件是二叉链表集合已经存在；操作结果是返回二叉链表集合中所有二叉链表的名称。

（5）初始化集合：函数名称是InitBiTree(FOREST& forest, char FileName[]),初始条件是二叉链表集合不存在；操作结果是将文件中的所有数据存入二叉链表集合。

## 3.2 系统设计

### 3.2.1 二叉链表类型定义

采用二叉链表作为二叉树的物理结构。二叉树节点的定义是一个名为TElemType结构体，存储int类型的key和char类型的value值，。二叉链表节点的定义是一个名为BiTNode的结构体，存储有二叉树树节点和指向左孩子和右孩子的两个指针。见图3-2-1 二叉链表的存储结构定义。

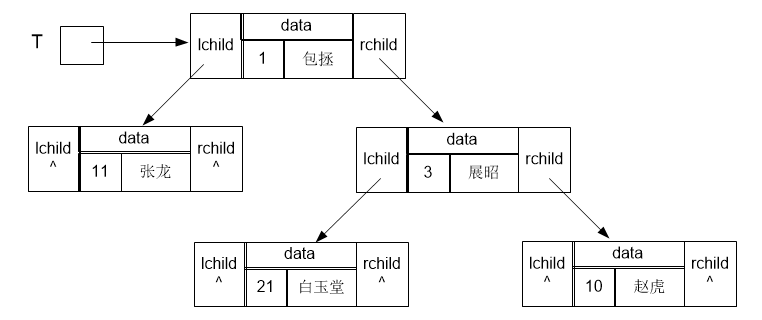


图3-2-1 二叉链表的存储结构定义

### 3.2.2 线性表集合类型定义

演示系统可实现多个二叉树管理。二叉链表的管理表定义是一个名为FOREST的结构体，其中定义了10个元素的结构体elem用以保存二叉链表和表名和一个int类型的变量用以保存二叉链表的个数，其物理结构的参考设计如图3-2-2所示。



图3-2-2多二叉树管理的物理结构示意图

二叉链表集合的管理菜单中包含了对单链表的操作的switch菜单。

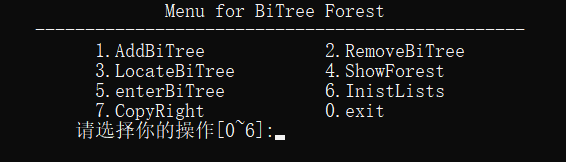


图3-2-3二叉链表集合操作界面

二叉链表集合的创建操作包括了文件读取操作，以及调用二叉链表的初始化函数InitBiTree(BiTree& T)。

### 3.2.3 各类函数的设计

二叉链表的创建函数CreateBiTree(BiTree& T, TElemType definition[])调用了判断是否有元素重复的函数isrepeat(TElemType definition[])和建立二叉树函数BuildTree(TElemType definition[],BiTree &T)。

二叉链表的销毁函数DestroyBiTree(BiTree& T)调用了二叉树的销毁函数Destroyhelper(BiTree& T)，二叉树的销毁函数输入二叉树的根节点用以销毁整个树，二叉链表销毁函数用以将首节点置为空。

二叉链表的清空函数ClearBiTree(BiTree& T)调用了二叉树的销毁函数Destroyhelper(BiTree& T)，二叉树的销毁函数输入二叉树的根节点用以销毁整个树，二叉链表销毁函数用以将首节点指向的根节点置为空。

二叉树的节点赋值函数Assign(BiTree& T, KeyType e, TElemType value)调用了查找二叉树节点函数LocateParentNode(BiTree T, KeyType e)。

二叉树的插入节点函数InsertNode(BiTree& T, KeyType e, int LR, TElemType c)调用了查找二叉树节点函数LocateNode(BiTree T, KeyType e)和初始化二叉链表函数InitBiTree(BiTree& T)。

二叉树的删除节点操作DeleteNode(BiTree& T, KeyType e)调用了查找二叉树节点函数LocateNode(BiTree T, KeyType e)和查找父节点函数LocateParentNode(BiTree T, KeyType e)。

二叉树按层遍历并且删除二叉树逻辑结构函数LevelOrderTraverse(BiTree T, void (\*visit)(BiTree))调用了二叉树深度函数BiTreeDepth(BiTree T)以及判断是否为2的指数函数IsTwo(int num)。

二叉链表集合删除二叉链表函数RemoveBiTree(FOREST& forest, char ListName[])调用了清空二叉链表函数ClearBiTree(BiTree& T)。

二叉链表集合初始化函数InitBiTree(FOREST& forest,char FileName[])调用了初始化二叉链表函数InitBiTree(BiTree& T)。

## 3.3 系统实现

### 3.3.1 重要功能的实现

（1）InitBiTree(BiTree& T)

将二叉链表的头结点分配相应的内存，左孩子和右孩子指向NULL，头指针的next即根节点设为null。

时间复杂度为：O(1)

（2）DestroyBiTree(BiTree& T)

逐次遍历二叉链表将节点对应的内存释放。

时间复杂度为：O(n)

（3）BiTreeEmpty(BiTree T)

若二叉链表表头结点为NULL返回INFEASIBLE,头结点的next即根节点为NULL则为空。

时间复杂度为：O(1)

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

判断插入位置是否合理及其插入的节点是否在树中存在，如果合理并且插入的节点不在树中，到达插入位置，根据LR为0或者1，插入结点c到T中，作为关键字为e的结点的左或右孩子结点，结点e的原有左子树或右子树则为结点c的右子树，返回OK。如果插入失败，返回ERROR。

时间复杂度为：O(n)

（5）DeleteNode(BiTree& T, KeyType e)

e是和T中结点关键字类型相同的给定值。删除T中关键字为e的结点；同时，如果关键字为e的结点度为0，删除即可；如关键字为e的结点度为1，用关键字为e的结点孩子代替被删除的e位置；如关键字为e的结点度为2，用e的左孩子代替被删除的e位置，e的右子树作为e的左子树中最右结点的右子树。 成功删除结点后返回OK，否则返回ERROR。

时间复杂度为：O(n)

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

如果二叉树存在，对二叉树T进行先序遍历，Visit是一个函数指针的形参（可使用该函数对结点操作），对每个结点调用函数Visit访问一次且一次。。

时间复杂度为：O(n)

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

对二叉树进行层序遍历，并根据遍历时的深度打印空格来输出二叉树的逻辑结构并返回OK。

时间复杂度为：O（n）

（8）SaveBiTree(BiTree T, char FileName[])

如果二叉链表不存在，返回ERROR，如果二叉链表存在，将数据写入文件并返回OK。

时间复杂度为：O(n);

## 3.4 系统测试

编译环境：visual c++ 6.0

使用集成IDE：visual 2019

### 3.4.1二叉树集合的操作

（1）InitForest(FOREST& forest, char FileName[])

创建结果如图3-4-1-1。

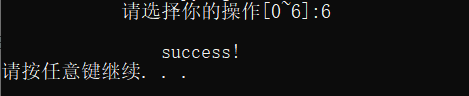


图3-4-1-1 初始化二叉链表集合

（2）AddBiTree(FOREST& forest, char ListName[])

向二叉链表集合中添加二叉链表操作结果如图3-4-1-2

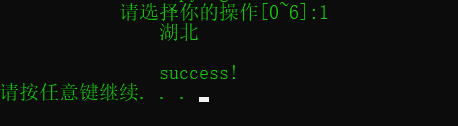


图3-4-1-2添加二叉链表

1. RemoveBiTree(FOREST& forest, char ListName[])

向二叉链表集合中移除某个二叉链表，操作结果如图3-4-1-3



图3-4-1-3移除二叉链表

1. ShowBiTree(FOREST forest)

展示二叉链表集合中所有的二叉链表名字。操作结果如图3-4-1-4。

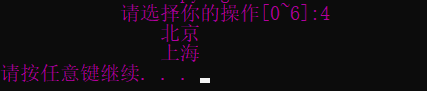


图3-4-1-4打印所有二叉链表

1. LocateBiTree(FOREST forest, char ListName[])

获取二叉链表的逻辑序号。操作结果如图3-4-1-5。

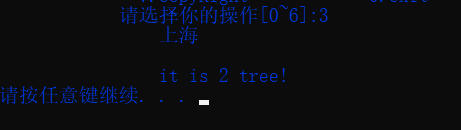


图3-4-1-5输出二叉链表的逻辑序号

1. EnterBitTree

选择进入上海的二叉链表，操作结果如图3-4-1-6。

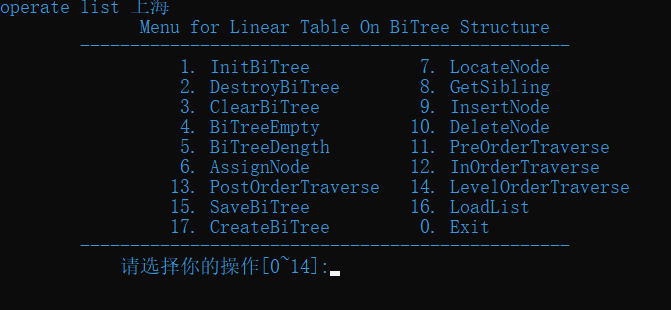


图3-4-1-6进入一个二叉链表操作

### 3.4.2 二叉链表功能的操作

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

前序遍历二叉树，操作结果如图3-4-2-1。

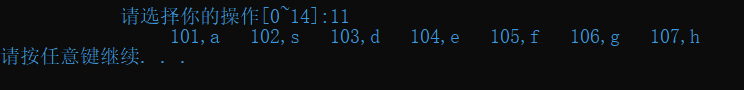


图3-4-2-1前序遍历

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

中序遍历二叉树，操作结果如图3-4-2-2。

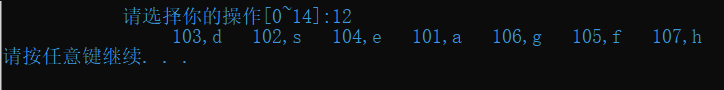


图3-4-2-2中序遍历

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

后序遍历二叉树，操作结果如图3-4-2-3。

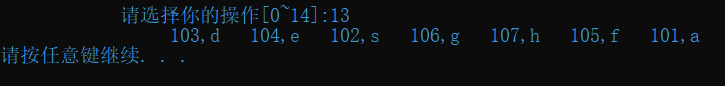


图3-4-2-3后序遍历

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

层序遍历并打印逻辑结构，操作结果如图3-4-2-4。

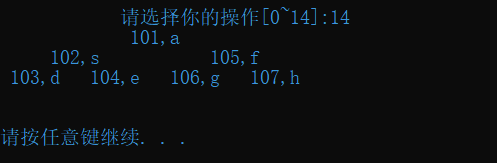


图3-4-2-4层序遍历

1. LocateNode(BiTree T, KeyType e)
2. 查找一个不存在的key，操作结果如图3-4-2-5。

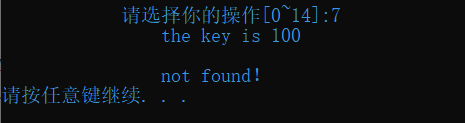


图3-4-2-5查找一个不存在的key

1. 查找一个存在的key，操作结果如图3-4-2-6。

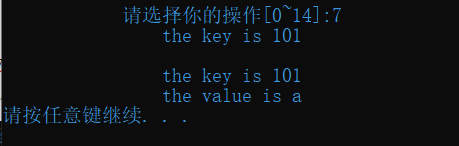


图3-4-2-6查找一个存在的key

1. Assign(BiTree& T, KeyType e, TElemType value)
2. 改变一个不存在的key，操作结果如图3-4-2-7。

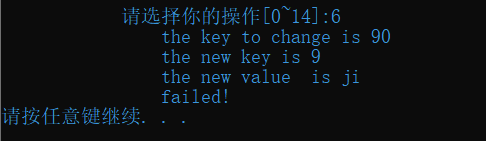


图3-4-2-7改变一个不存在的key

1. 改变一个存在的key，操作结果如3-4-2-8，树的逻辑结构见图3-4-2-9

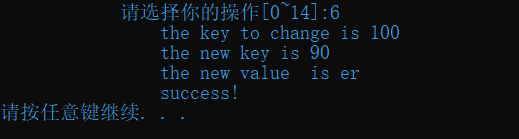


图3-4-2-8改变一个存在的key

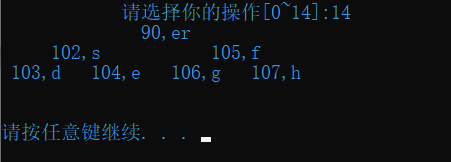


图3-4-2-9改变后的逻辑结构

1. BiTreeDepth(BiTree T)

获取树的深度见图3-4-2-10。

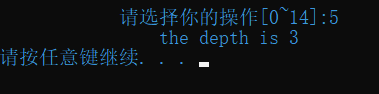


图3-4-2-10获取树的深度

1. GetSibling(BiTree T, KeyType e)
2. 获取不存在的节点的兄弟节点

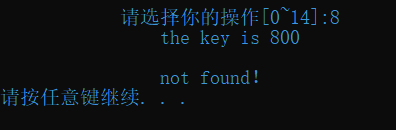


图3-4-2-11获取不存在的节点的兄弟节点

1. 获取存在节点且兄弟节点存在的兄弟节点

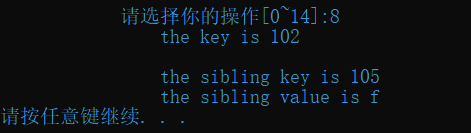


图3-4-2-12获取存在节点的兄弟节点

1. 获取存在节点但兄弟节点不存在的兄弟节点

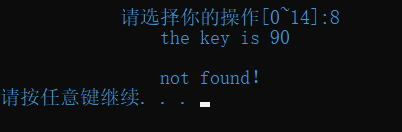


图3-4-2-13获取存在节点但兄弟节点不存在的兄弟节点

1. InsertNode(BiTree& T, KeyType e, int LR, TElemType c)
2. 插入一个不存在父节点的节点，操作结果如图3-4-2-14。

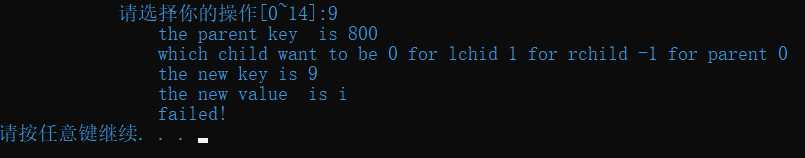


图3-4-2-14插入一个不存在父节点的节点

1. 插入一个某节点的左孩子，且插入节点的key值与树中的节点key值重复

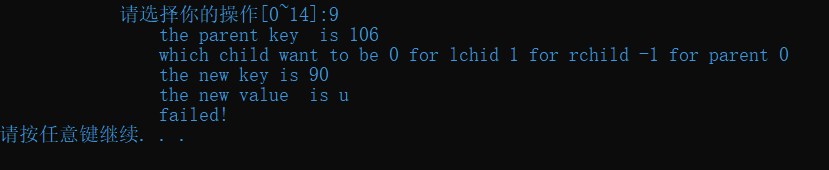


图3-4-2-15插入的key值重复

1. 插入一个某节点的左孩子，且插入节点的key值与树中的节点key值不重复

操作结果如图3-4-2-16，树结构见图3-4-2-17。

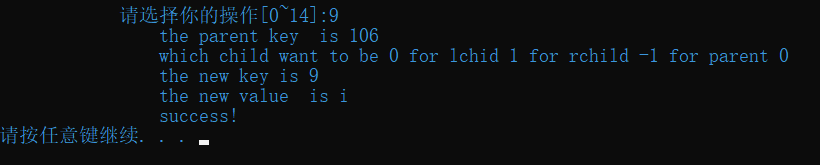


图3-4-2-16正确插入

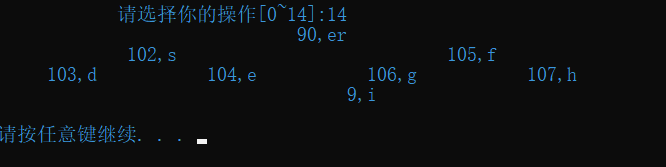


图3-4-2-17树结构示意图

1. DeleteNode(BiTree& T, KeyType e)
2. 删除不存在的节点，操作结果如图3-4-2-18。

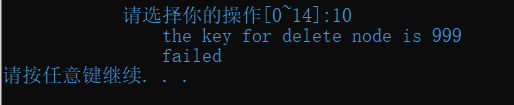


图3-4-2-18删除不存在的节点

1. 删除树叶节点

操作结果如图3-4-2-19，树的逻辑结构如图3-4-2-20

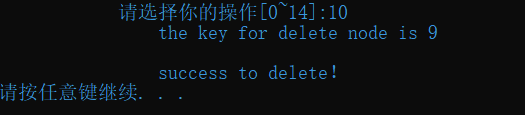


图3-4-2-19删除树叶节点

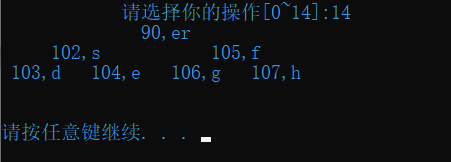


图3-4-2-20树的逻辑结构

1. 删除根节点

操作结果如图3-4-2-21

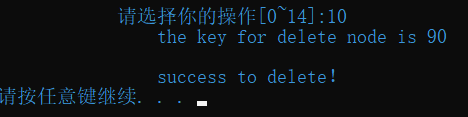


图3-4-2-21删除根节点

1. ClearBiTree(BiTree& T)

清空二叉树，操作结果见图3-4-2-22。

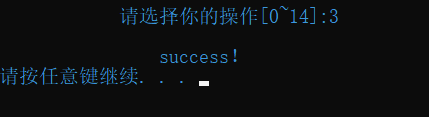


图3-4-2-22清空二叉树

1. DestroyBiTree(BiTree& T)

销毁二叉树，操作结果如图3-4-2-23。

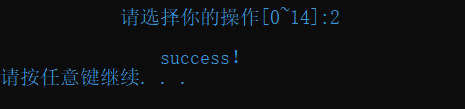


图3-4-2-23销毁二叉树

### 3.4.3 文件的操作

（1）SaveBiTree(BiTree T, char FileName[])

保存二叉树到文件，操作结果如图3-4-3-1，文件中数据如图3-4-3-2

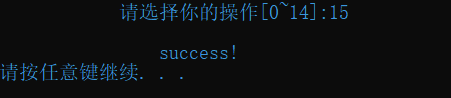


图3-4-3-1保存二叉树



图3-4-3-2文件中数据

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

加载文件中数据到二叉树中，操作结果如图3-4-3-4，文件中数据如图3-4-3-3，树的逻辑结构如图3-4-3-5。

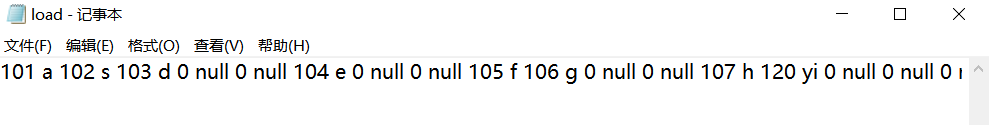


图3-4-3-3文件中数据

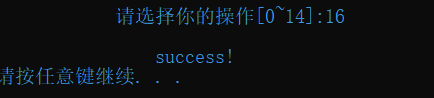


图3-4-3-4操作结果

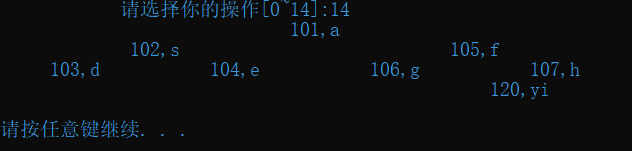


图3-4-3-5树的逻辑结构

## 3.5 实验小结

本次实验遇到的一个难题是如何在层序遍历二叉树时将二叉树的逻辑结构同时打印出来。将二叉树的逻辑结构打印出来意味着将二叉树不存在的节点以空格打印，同时将二叉树的各个节点摆放在一定的位置。考虑到二叉树每层的最多节点数量与该层的高度有关，即为2的该层高度次方，所以定义一个函数用以判断当前遍历到的节点是否为2的指数次方，如果是，则打印“\n”，换到下一层。而每个节点的位置与最底层的长度及这个节点的高度有关，根据相关关系，可以打印出空格。

# 4基于邻接表的图实现

## 4.1 问题描述

采用邻接表作为图的物理结构，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。

实现图的基本运算，可任选无向图、有向图、无向网和有向网这四种图中的一种实现。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12种基本运算。

### 4.1.1 12种基本运算定义

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。

⑵销毁图：函数名称是DestroyGraph(G)；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是和G中顶点关键字类型相同的给定值；操作结果是若u在图G中存在，返回关键字为u的顶点位置序号（简称位序），否则返回其它表示“不存在”的信息。

⑷顶点赋值：函数名称是PutVex (G,u,value)；初始条件是图G存在，u是和G中顶点关键字类型相同的给定值；操作结果是对关键字为u的顶点赋值value。

⑸获得第一邻接点：函数名称是FirstAdjVex(G, u)；初始条件是图G存在，u是G中顶点的位序；操作结果是返回u对应顶点的第一个邻接顶点位序，如果u的顶点没有邻接顶点，否则返回其它表示“不存在”的信息。

⑹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和w是G中两个顶点的位序，v对应G的一个顶点,w对应v的邻接顶点；操作结果是返回v的（相对于w）下一个邻接顶点的位序，如果w是最后一个邻接顶点，返回其它表示“不存在”的信息。

⑺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。（在这里也保持顶点关键字的唯一性）

⑻删除顶点：函数名称是DeleteVex(G,v)；初始条件是图G存在，v是和G中顶点关键字类型相同的给定值；操作结果是在图G中删除关键字v对应的顶点以及相关的弧。

⑼插入弧：函数名称是InsertArc(G,v,w)；初始条件是图G存在，v、w是和G中顶点关键字类型相同的给定值；操作结果是在图G中增加弧<v,w>，如果图G是无向图，还需要增加<w,v>。

⑽删除弧：函数名称是DeleteArc(G,v,w)；初始条件是图G存在，v、w是和G中顶点关键字类型相同的给定值；操作结果是在图G中删除弧<v,w>，如果图G是无向图，还需要删除<w,v>。

⑾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。

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

### 4.1.2 2种文件操作定义

(1)存储元素：函数名称是SaveGraph(ALGraph G, char FileName[])，初始条件是图G已经存在；操作结果：将图G的的元素写到FileName文件中。

(2)读入元素：函数名称是LoadGraph(ALGraph\*& G, char FileName[])，初始条件是图头结点G不存在：操作结果是：将FileName文件中的数据读入到线性图G中。

### 4.1.3 5种集合运算定义

(1)添加图：函数名称是AddGraph(GRAPHLIST& GraphList, char GraphName[])，初始条件是图集合已经存在；操作结果是对图集合中添加一个名为GraphName的图。

(2)删除图：函数名称是RemoveGraph(GRAPHLIST& GraphList, char GraphName[])，初始条件是图集合已经存在；操作结果是对图集合中删除一个名为GraphName的图。

(3)查找图：函数名称是LocateGraph(GRAPHLIST GraphList, char GraphName[])，初始条件是图集合已经存在；操作结果是在图集合中查找一个名称为GraphName的图，成功返回其逻辑序号。

（4）显示图名：函数名称是ShowGraphList(GRAPHLIST GraphList)，初始条件是图集合已经存在；操作结果是返回图集合中所有图的名称。

（5）初始化图：函数名称是InitGraphList(GRAPHLIST& GraphList,char FileName[]),初始条件是图集合不存在；操作结果是将文件中的所有数据存入图集合。

## 4.2 系统设计

### 4.2.1 邻接表类型定义

采用邻接表作为图的物理结构。图顶点的类型定义是一个名为VertexType结构体，存储int类型的key和char类型的value值，。图边的类型定义是一个名为ArcNode的结构体，存储有顶点位置编号和下一个表结点指针以及网的权重。图的头结点定义及其数组类型定义是AdjList的数组存有顶点信息和指向第一条弧的指针。见图4-2-1 图的存储结构定义。

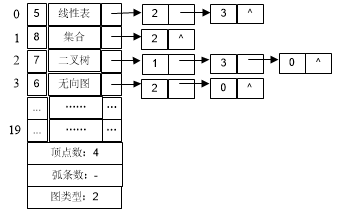


图4-2-1 图的存储结构定义

### 4.2.2 图集合类型定义

演示系统可实现多个图管理。图的管理表定义是一个名为GRAPHLIST的结构体，其中定义了10个元素的结构体elem用以保存图和图名和一个int类型的变量length用以保存图的个数。

图集合的管理菜单中包含了对单图的操作的switch菜单。图集合操作界面见图4-2-3。

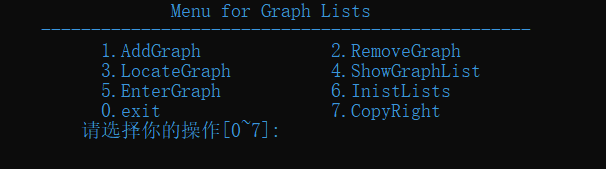


图4-2-3图集合操作界面

图集合的创建操作包括了文件读取操作，以及调用图集合的初始化函数InitGraphList(GRAPHLIST& GraphList,char FileName[])。

### 4.2.3 各类函数的设计

图表的创建函数CreateGraph(ALGraph &G, VertexType V[], KeyType \*\* VR)调用了判断是否有元素重复的函数isrepeat(VertexType V[],int length)、是否有关键字输入错误的函数isKeyTypeRight(VertexType V[], KeyType \*\*VR,int arcnum)和插入图的弧函数InsertArcofGraph(G, VR[i]);。

图的销毁函数DestroyGraph(ALGraph\*& G)调用了图的清空函数ClearGraph(ALGraph& G)，将首节点置为空。

图的节点赋值函数PutVex(ALGraph& G, KeyType u, VertexType value)调用了查找图节点函数LocateKey(ALGraph G, KeyType u)。

图的插入节点函数DeleteVex(ALGraph& G, KeyType v)、图的删除节点操作DeleteNode(BiTree& T, KeyType e)、图的插入弧函数InsertArc(ALGraph& G, KeyType v, KeyType w)调用了查找图节点函数LocateNode(BiTree T, KeyType e)。

图的删除弧函数DeleteArc(ALGraph& G, KeyType v, KeyType w)调用了查找图节点函数LocateKey(ALGraph G, KeyType u)以及删除一条有向边的函数DeleteArcofVex(ALGraph &G, int u, int v)。

图深度遍历函数DFSTraverse(ALGraph G, void (\*visit)(VertexType))调用了图单节点深度遍历函数DFS(ALGraph G, int i)。

图广度遍历函数BFSTraverse(ALGraph G, void (\*visit)(VertexType))调用了图单节点广度遍历函数BFS(ALGraph G, int index)。

图集合删除图函数RemoveGraph(GRAPHLIST& GraphList, char GraphName[])调用了销毁图函数DestroyGraph(ALGraph\*& G)。

## 4.3 系统实现

### 4.3.1 重要功能的实现

（1）InitGraph(ALGraph\*& G)

将图的头结点分配相应的内存，图的边数、节点数设置为0。

时间复杂度为：O(1)

（2）DestroyGraph(ALGraph\*& G)

逐次遍历图将节点对应的内存释放。

时间复杂度为：O(n)

（3）InsertVex(ALGraph& G, VertexType v)

在图G中插入新顶点关v（要求关键字具有唯一性）。成功赋值返回OK，否则返回ERROR。

时间复杂度为：O(n)

（4）DeleteVex(ALGraph& G, KeyType v)

v是和G中顶点关键字类型相同的给定值；操作结果是在图G中删除关键字v对应的顶点以及相关的弧。成功赋值返回OK，否则返回ERROR。

时间复杂度为：O(n)

（5）InsertArc(ALGraph& G, KeyType v, KeyType w)

在图G中增加弧<v,w>，成功返回OK,否则返回ERROR。

时间复杂度为：O(n)

（7）DFSTraverse(ALGraph G, void (\*visit)(VertexType))

对图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。

时间复杂度为：O（n）

（8）SaveGraph(ALGraph G, char FileName[])

如果图不存在，返回ERROR，如果图存在，将数据写入文件并返回OK。

时间复杂度为：O(n);

## 4.4 系统测试

编译环境：visual c++ 6.0

使用集成IDE：visual 2019

### 4.4.1图集合的操作

（1）InitGraphList(GRAPHLIST& GraphList,char FileName[])

创建结果如图4-4-1-1。

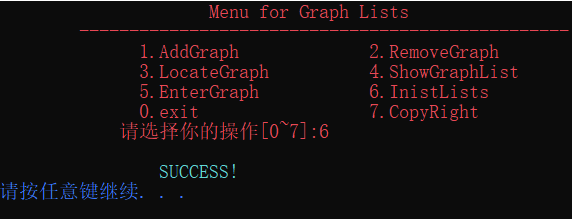


图4-4-1-1 初始化图集合

（2）AddGraph(GRAPHLIST& GraphList, char GraphName[])

向图集合中添加图操作结果如图4-4-1-2

图4-4-1-2添加图

1. RemoveGraph(GRAPHLIST& GraphList, char GraphName[])

向图集合中移除某个图，操作结果如图4-4-1-3

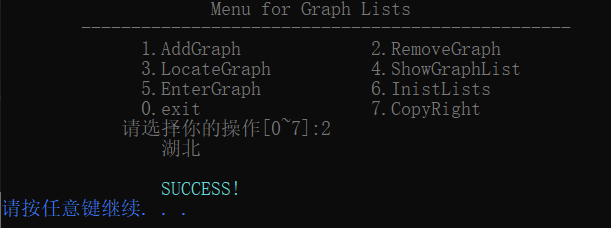


图4-4-1-3移除图

1. ShowGraphList(GRAPHLIST GraphList)

展示图集合中所有的图名字。操作结果如图4-4-1-4。



图4-4-1-4打印所有图

1. LocateGraph(GRAPHLIST GraphList, char GraphName[])

获取图的逻辑序号。操作结果如图4-4-1-5。

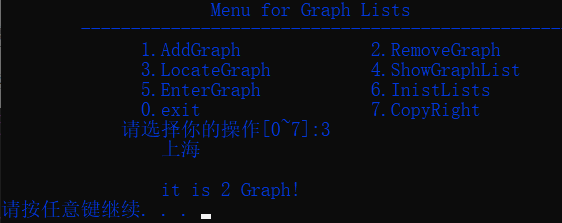


图4-4-1-5输出图的逻辑序号

1. EnterGraph

选择进入上海的图，操作结果如图4-4-1-6。

图4-4-1-6进入一个图操作

### 4.4.2 图功能的操作

（1）ShowGraph(ALGraph G)

遍历邻接表数组并打印邻接表信息，操作结果如图4-4-2-1。

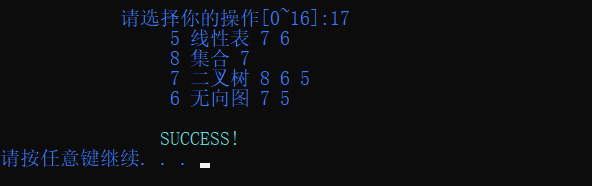


图4-4-2-1遍历邻接表

（2）LocateKey(ALGraph G, KeyType u)

1、查找一个不存在的key，操作结果如图4-4-2-2。

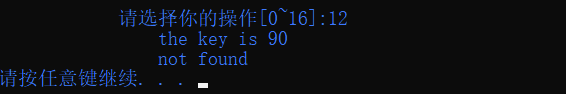


图4-4-2-2查找一个不存在的key

2、查找一个存在的key，操作结果如图4-4-2-3。

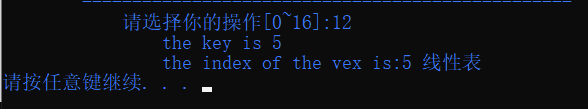


图4-4-2-3查找一个存在的key

（3）PutVex(ALGraph& G, KeyType u, VertexType value)

1、改变一个不存在的key，操作结果如图4-4-2-4。

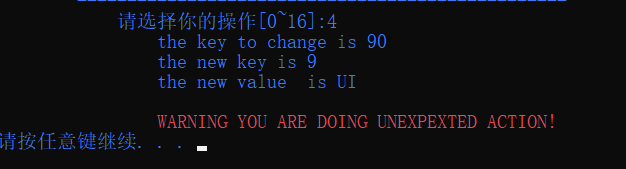


图4-4-2-4改变一个不存在的key

2、改变一个存在的key，操作结果如4-4-2-5，邻接表见图4-4-2-6

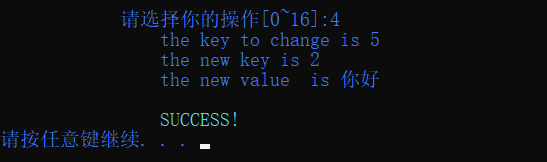


图4-4-2-5改变一个存在的key

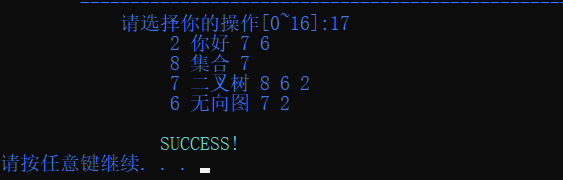


图4-4-2-6改变后的邻接表

（4）FirstAdjVex(ALGraph G, KeyType u)

1.获取不存在的顶点的第一邻接顶点

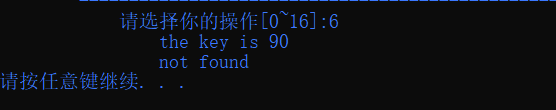


图4-4-2-7获取不存在的顶点的第一邻接顶点

2.获取存在的顶点的第一邻接顶点

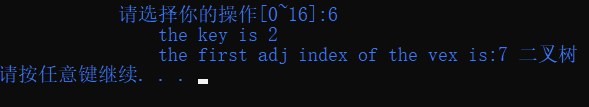


图4-4-2-8获取存在的顶点的第一邻接顶点

（5）NextAdjVex(ALGraph G, KeyType v, KeyType w)

1.输入不存在的顶点

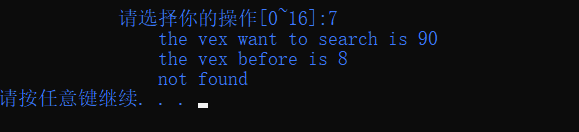


图4-4-2-9不存在的顶点

2.输入存在的节点v，w不存在

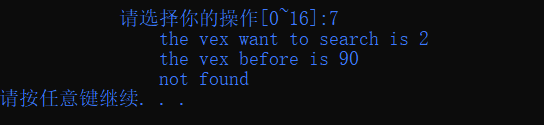


图4-4-2-10存在的节点v，w不存在

3.输入存在的节点v，w存在

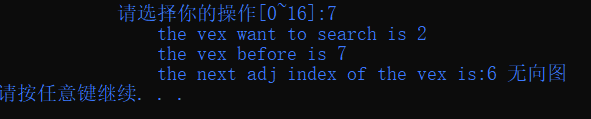


图4-4-2-11存在节点v，w

（6）InsertVex(ALGraph& G, VertexType v)

1.输入与已知顶点key值重复的顶点

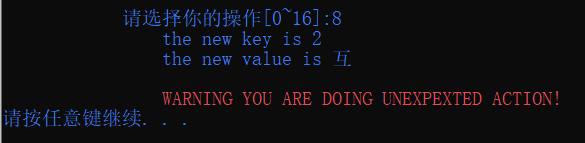


图4-4-2-12插入顶点重复的顶点

2.输入正常的顶点

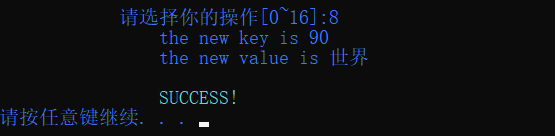


图4-4-2-13插入正常的顶点

1. DeleteVex(ALGraph& G, KeyType v)
2. 删除不存在key值的顶点

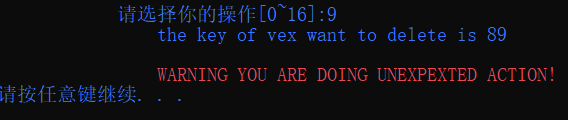


图4-4-2-13删除不存在的顶点

1. 删除存在key值的顶点

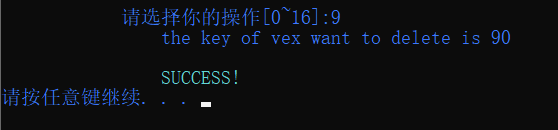


图4-4-2-14删除正常的顶点

1. InsertArc(ALGraph& G, KeyType v, KeyType w)
2. 插入节点不存在的弧

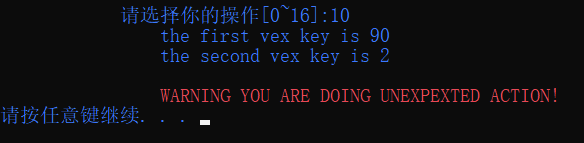


图4-4-2-15插入节点不存在的弧

1. 插入已经存在的弧

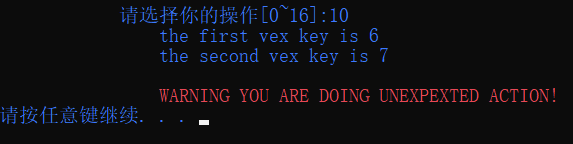


图4-4-2-16插入已经存在的弧

1. 插入正常的弧

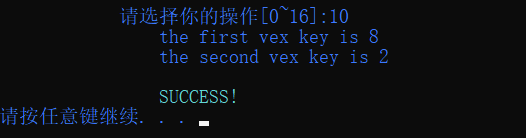


图4-4-2-17插入正常的弧

1. DeleteArc(ALGraph& G, KeyType v, KeyType w)
2. 删除不存在的弧

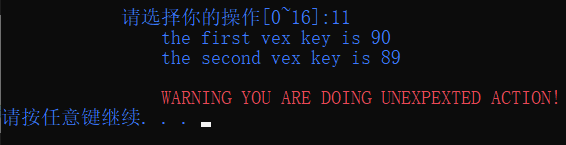


图4-4-2-18删除不存在的弧

1. 删除已经存在的弧

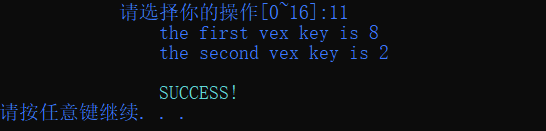


图4-4-2-19删除已经存在的弧

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

广度遍历结果见图4-4-2-20

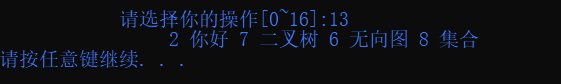


图4-4-2-20广度遍历

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

深度遍历结果见图4-4-2-21

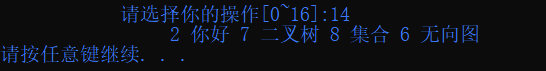


图4-4-2-21深度遍历

（12）ClearGraph(ALGraph& G)

清空图，操作结果见图4-4-2-22。

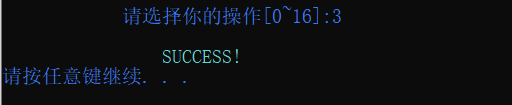


图4-4-2-22清空图

（14）DestroyGraph(ALGraph\*& G)

销毁图，操作结果如图4-4-2-23。

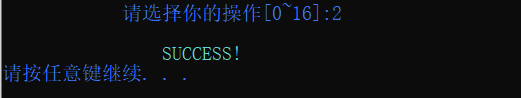


图4-4-2-23销毁图

### 4.4.3 文件的操作

（1）SaveGraph(ALGraph G, char FileName[])

保存图到文件，操作结果如图4-4-3-1，文件中数据如图4-4-3-2

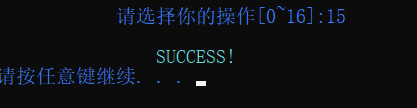


图4-4-3-1保存图

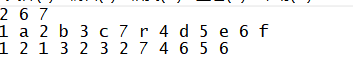


图4-4-3-2文件中数据

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

加载文件中数据到图中，操作结果如图4-4-3-4，文件中数据如图4-4-3-3，邻接表如图4-4-3-5。

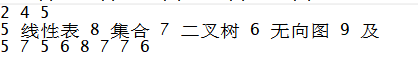


图4-4-3-3文件中数据

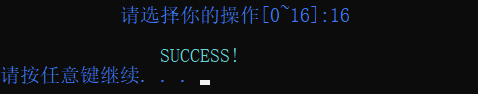


图4-4-3-4操作结果

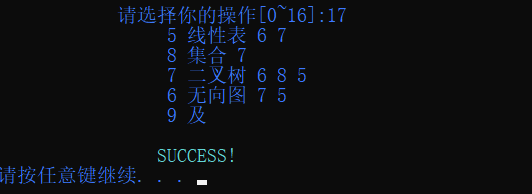


图3-4-3-5邻接表

## 4.5 实验小结

本次图实验遇到的最大困难在于图顶点函数的实现，因为如果图的顶点删除还涉及到该顶点相关联的边，如果图的顶点不是顶点数组中最后一个，那么删除顶点后，数组中该顶点后面的顶点也要发生改变，而由于顶点的逻辑序号发生改变，邻接表中边的记录也要相应发生改变，所以这个函数的实现需要考虑很多方面。最后我通过三步实现这个函数，一、删除关键字顶点的临接边，二、删除其他顶点与需要删除顶点关联的边，在删除过程中加入判断，将相应的边存储的逻辑序号改变，三、如果顶点位置序号是最后一个直接删除否则后面的顶点逐个前移。

# 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