華中科技大學

课 程 实 验 报 告

课程名称： 数据结构实验

专业班级 CS2309

学　　号 U202315715

姓　　名 邹晋

指导教师 李剑军

报告日期 2024 年 6 月 13 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 1
1.1　问题描述 1
1.2　系统设计 2
1.3　系统实现 5
1.4　系统测试 18
1.5　实验小结 23
2　基于邻接表的图实现 24
2.1　问题描述 24
2.2　系统设计 26
2.3　系统实现 29
2.4　系统测试 40
2.5　实验小结 45
3　课程的收获和建议 46
3.1　基于顺序存储结构的线性表实现 46
3.2　基于链式存储结构的线性表实现 47
3.3　基于二叉链表的二叉树实现 47
3.4　基于邻接表的图实现 47
4　附录 A 基于顺序存储结构线性表实现的源程序 48
5　附录 B 基于邻接表图实现的源程序 68

1 基于顺序存储结构的线性表实现

线性表是最常用且最简单的一种数据结构，是零个或多个数据元素的有限序列。线性表的数据集合为 a1,a2,…,an，假设每个元素的类型均为 DataType。其中，除第一个元素 a1 外，每一个元素有且只有一个直接前驱元素，除了最后一个元素 an 外，每一个元素有且只有一个直接后继元素。如果 ai 为第 i 位数据元素，我们称 i 为 ai 在线性表中的位序。事实上，线性表是一个相当灵活的数据结构，长度可以按需改变，同时元素不仅可以访问，还可以增删等。

1.1 问题描述

实验要求：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 种基本运算及几种拓展运算。

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。

1.1.1 功能实现

1. 基础功能：

   (a) 初始化表，InitList；
   (b) 销毁表，DestroyList；
   (c) 清空表，ClearList；
   (d) 判定空表，ListEmpty；
   (e) 求表长，ListLength；
   (f) 获得元素，GetElem；
   (g) 查找元素，LocateElem；
   (h) 获得前驱，PriorElem；
   (i) 获得后继，NextElem；
   (j) 插入元素，ListInsert；
   (k) 删除元素，ListDelete；
   (l) 遍历表，ListTraverse；

2. 进阶功能：

(a) 最大连续子数组和,MaxSubArray;

(b) 和为 K 的子数组,SubArrayNum;

(c) 顺序表排序,sortList；

(d) 线性表保存到文件，SaveList;

(e) 从文件读取线性表，LoadList;

(f) 添加线性表,AddList;

(g) 移除线性表，RemoveList;

(h) 查找线性表，LocateList;

(i) 依次输出所有线性表，AllShow;

1.1.2 实验目的

1. 加深对线性表的概念、基本运算的理解；
2. 熟练掌握线性表的逻辑结构与物理结构的关系；
3. 物理结构采用顺序表, 熟练掌握顺序表的基本运算的实现。

分析：需要了解链表的结构，同时演示系统要清晰明了。

1.2 系统设计

1.2.1 系统总体设计

本实验以顺序储存结构线性表为主体，并定义新结构 **LISTS** 来实现多线性表功能，用户可以使用至多 10 个相互**独立**的线性表，使用创建、销毁、清空、排序等操作，并在它们之间相互切换。系统能实现的功能分为基础功能与进阶功能，已在上一节列出。

系统通过一个简易的菜单来给使用者选择功能，每项功能后均有文字说明，使用时也有文字提示。通过清屏函数等操作，并用 **system** 相关函数调整窗口大小，使得菜单始终位于界面上方，便于使用。

主函数则是定义一些全局变量、编写引导并将各种功能函数实现，利用 **switch 分支结构**进行不同功能的实现，以达到连接各项功能函数的目的。

系统界面便于理解，功能均有文字解释，如图：

图 1-1　菜单样式

1.2.2　数据结构设计

1. 首先, 我定义了一些常量

Algorithm 1 Macros and Typedefs

```
1:  define TRUE 1
2:  define MAX 10
3:  define FALSE 0
4:  define OK 1
5:  define ERROR 0
6:  define INFEASIBLE -1
7:  define OVERFLOW -2
8:  typedef int status;
9:  typedef int ElemType;
10: define LIST_INIT_SIZE 100
11: define LISTINCREMENT 20
```

2. 对于功能的实现，设计了两个结构体。

Algorithm 2 SqList Structure

(a)
```
1: typedef struct {
2:     ElemType * elem;
3:     int length;
4:     int listsize;
5: } SqList;
```

Algorithm 3 LISTS Structure

(b)
```
1: typedef struct {
2:     struct {
3:         char name[30];
4:         SqList L;
5:     } elem[10];
6:     int length;
7:     int now;
8: } LISTS;
```

1.3 系统实现

1.3.1 算法设计

1. 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；

   操作思路：若线性表 L 不存在，为 L 分配存储空间后，将表长设为 0。流程图如下：

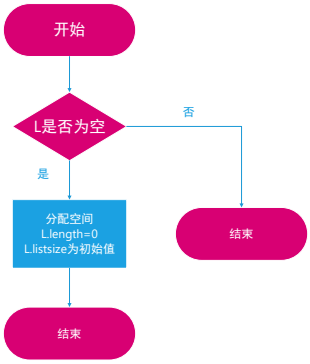

图 1-2 初始化线性表

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；

   操作思路：若线性表存在，销毁 L 的存储空间后，将 L.elem=NULL。流程图如下：

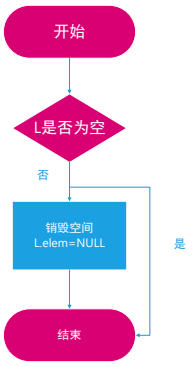

图 1-3 销毁线性表

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；

操作思路：若 L 存在，将 L 的长度重置为 0。流程图如下：

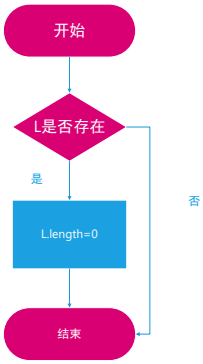

图 1-4　清空线性表

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；

   操作思路：若 L 已存在，检查 L.length 是否为 0。流程图如下：

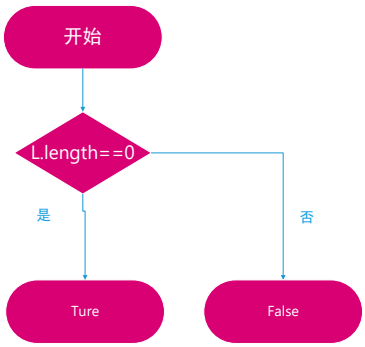

图 1-5　判空线性表

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；

   操作思路：若线性表 L 存在，返回 L.length。流程图如下：

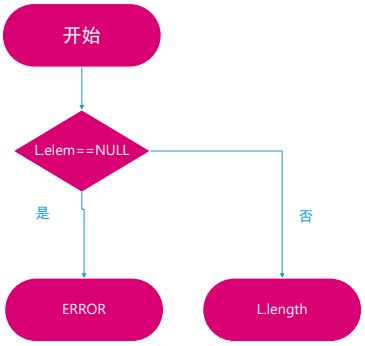

图 1-6　线性表长度

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；
   操作思路：若线性表 L 存在，若线性表 L 存在，判断 i 是否在 1 到 L.length 之间，真则令 e=L.elem[i-1], 假则返回 ERROR。流程图如下：

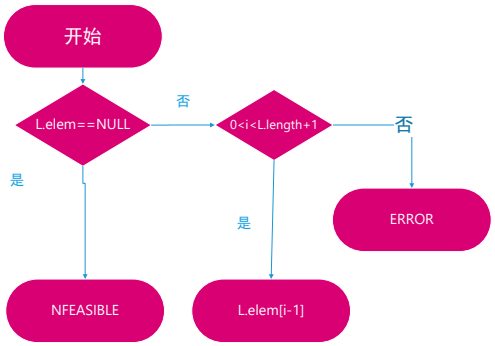

图 1-7　获得元素

7. 查找元素：函数名称是 LocateElem(L,e)；初始条件是线性表已存在；操作结果是返回 L 中第 1 个与 e 满足相等关系的数据元素的位序，若这样的数据元素不存在，则返回值为 0；
   操作思路：若线性表 L 已存在，遍历每一个元素，若有与 e 相等的值就返回，遍历结束则返回 0。流程图如下：

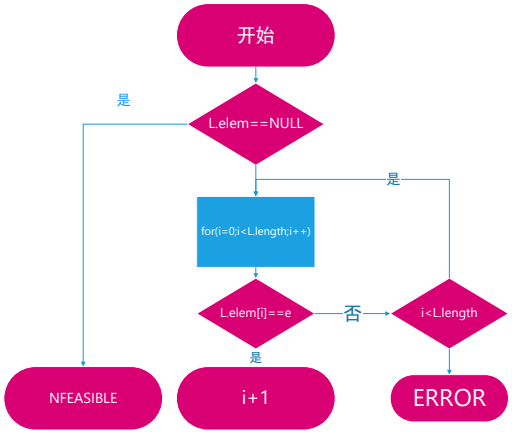

图 1-8　查找元素

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

   操作思路：若线性表 L 已存在，使用 LocateElem 函数确定 cur 位序 e，再判断 e 是否属于区间（0，L.length]。流程图如下：

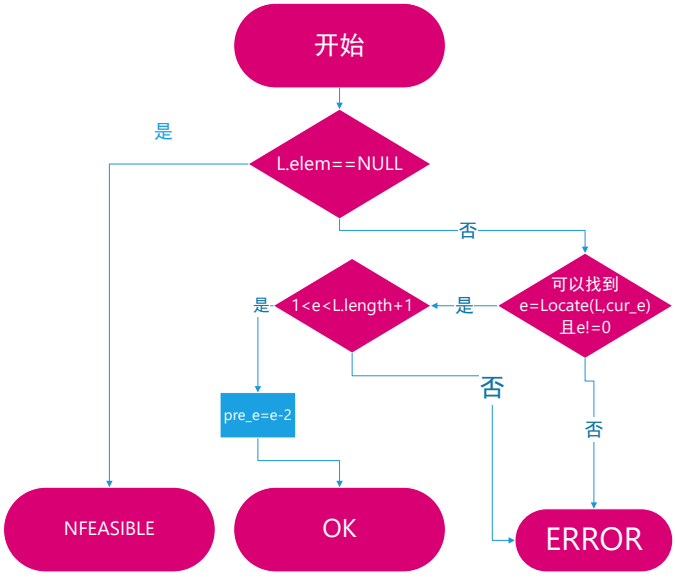

图 1-9　获得前驱

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

后继，否则操作失败，next 无定义；

操作思路：与 PriorElem 类似，先定位，再处理。

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，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。

操作思路：若线性表 L 存在，判断 i 是否合法，判断线性表空间是否超出，将第 i 个及以后的元素后移一位，赋值，长度加一。流程图如下：

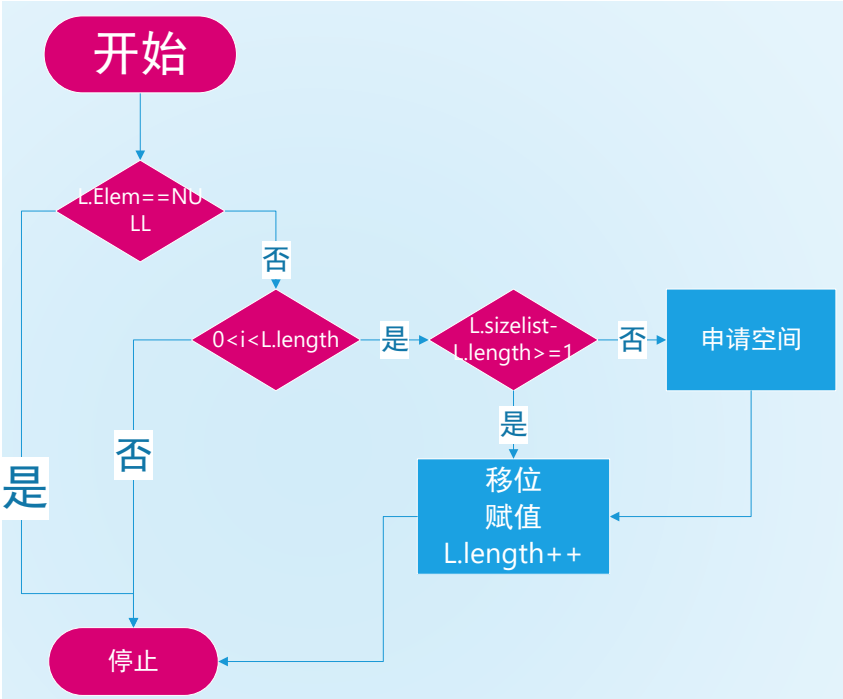

图 1-10　插入元素

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 返回的值；

操作思路：与上一函数类似，还需判断线性表长度是否大于等于 1，删除元素时将其后面元素前移，长度减一，且删元素时可将其元素删完（长度为 0）。

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()。

操作思路：若线性表 L 存在，使用 **for 循环**，将线性表中所有元素输出。

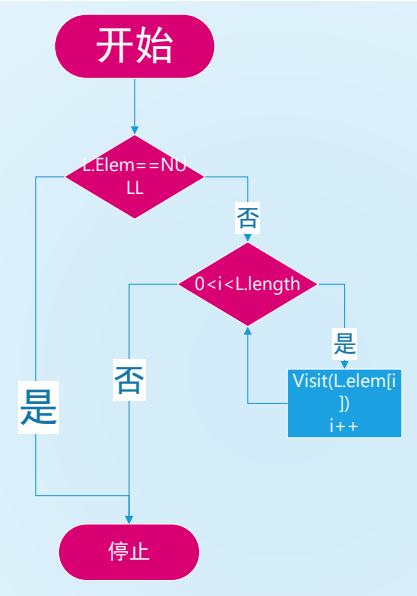

图 1-11　遍历输出

13. 最大连续子数组和：函数名称是 MaxSubArray(L); 初始条件是线性表 L 已存在且非空，请找出一个具有最大和的连续子数组（子数组最少包含一个元素），操作结果是其最大和;

Algorithm 4 求连续最大子数组和

Require: L is an array of integers, n is the length of the array

Ensure: Returns the maximum sum of a contiguous subarray

if $L = \text{NULL}$ **then**
    return INFEASIBLE
end if
$max\_sum \leftarrow L[0]$
$curr\_sum \leftarrow L[0]$
for $i \leftarrow 1$ to n **do**
    $curr\_sum \leftarrow \max(curr\_sum + L[i], L[i])$
    $max\_sum \leftarrow \max(max\_sum, curr\_sum)$
end for
return max_sum

14. 和为 K 的子数组：函数名称是 SubArrayNum(L,k); 初始条件是线性表 L 已存在且非空, 操作结果是该数组中和为 k 的连续子数组的个数；

Algorithm 5 和为 k 连续子数组个数

Require: L is an array of integers, n is the length of the array, k is the target sum

Ensure: Returns the number of subarrays whose sum is equal to k

```
if L = NULL then
    return INFEASIBLE
end if
count ← 0
for i ← 0 to n − 1 do
    sum ← 0
    for j ← i to n − 1 do
        sum ← sum + L[j]
        if sum = k then
            count ← count + 1
        end if
    end for
end for
return count
```

15. 顺序表排序：函数名称是 sortList(L)；初始条件是线性表 L 已存在；操作结果是将 L 由小到大排序；

Algorithm 6 顺序表排序

Require: L is an array of integers, n is the length of the array

Ensure: Sorts the array in non-decreasing order

```
if L = NULL then
    return
end if
for i ← 0 to n − 2 do
    flag ← 0
    for j ← 0 to n − i − 2 do
        if L[j] > L[j + 1] then
            temp ← L[j]
            L[j] ← L[j + 1]
            L[j + 1] ← temp
            flag ← 1
        end if
    end for
    if flag = 0 then
        break
    end if
end for
```

16. 实现线性表的文件形式保存：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；

Algorithm 7 保存线性表

Require: L is a sequence list, $FileName$ is a string

Ensure: Saves the list to a file

1: **if** $L.elem =$ NULL **then**
2: **return** INFEASIBLE
3: **end if**
4: $fp \leftarrow$ fopen($FileName$, "w")
5: **if** $fp =$ NULL **then**
6: **return** ERROR
7: **end if**
8: **for** $j \leftarrow 0$ to $L.length - 1$ **do**
9: fprintf(fp, "%d ", $L.elem[j]$)
10: **end for**
11: fclose(fp)
12: **return** OK

需要设计线性表文件保存和加载操作合理模式。

注：保存到文件后，可以由一个空线性表加载文件生成线性表。

Algorithm 8 加载线性表

Require: L is a sequence list, $FileName$ is a string

Ensure: Loads the list from a file

if $L.elem \neq$ NULL **then**

return INFEASIBLE

end if

$fp \leftarrow$ fopen($FileName$, "r")

if $fp =$ NULL **then**

return ERROR

end if

$L.elem \leftarrow$ malloc($LIST\_INIT\_SIZE \times$ sizeof(ElemType))

$L.listsize \leftarrow LIST\_INIT\_SIZE$

$L.length \leftarrow 0$

while fscanf(fp, "%d", &$L.elem[L.length]$) $= 1$ **do**

$L.length \leftarrow L.length + 1$

end while

fclose(fp)

return OK

17. 实现多个线性表管理：设计相应的数据结构管理多个线性表的查找、添加、移除等功能。对于多线性表管理，我定义了如下的结构体：

Algorithm 9 Definition of LISTS

```
typedef struct {
    struct {
        char name[30];
        SqList L;
    } elem[10];
    int length;
    int now;
} LISTS;
```

并且定义了如下函数：

(a) AddList(lists,ListName), 为 lists 增添一个线性表，并用 ListName 命名；

Algorithm 10 添加线性表

```
1: procedure AddList(Lists, ListName)
2:     if Lists.length ≥ 20 then
3:         return ERROR
4:     end if
5:     Lists.elem[Lists.length].L.elem ← NULL
6:     InitList(Lists.elem[Lists.length].L)
7:     Lists.elem[Lists.length].L.elem ← malloc(LIST_INIT_SIZE × sizeof(ElemType))
8:     k ← 0
9:     while k < 30 && ListName[k] ≠' \0' do
10:        Lists.elem[Lists.length].name[k] ← ListName[k]
11:        k ← k + 1
12:    end while
13:    if !Lists.elem[Lists.length].L.elem then
14:        return ERROR
15:    end if
16:    Lists.elem[Lists.length].L.length ← 0
17:    Lists.elem[Lists.length].L.listsize ← LIST_INIT_SIZE
18:    Lists.length ← Lists.length + 1
19:    return OK
20: end procedure
```

(b) RemoveList(lists,ListName), 查找名字为 ListName 的线性表，若存在，将其删去；

Algorithm 11 移除线性表

```
1:  procedure RemoveList(Lists, ListName)
2:      k ← 0
3:      j ← 0
4:      while k < 10 do
5:          if strcmp(Lists.elem[k].name, ListName) == 0 then
6:              j ← DestroyList(Lists.elem[k].L)
7:              break
8:          end if
9:          k ← k + 1
10:     end while
11:     if j == OK then
12:         for m ← k; m < Lists.length − 1; m + + do
13:             Lists.elem[m] ← Lists.elem[m + 1]
14:         end for
15:         Lists.elem[Lists.length − 1].name ← NULL
16:         Lists.length ← Lists.length − 1
17:         Lists.now ← Lists.length − 1
18:         return OK
19:     else
20:         return ERROR
21:     end if
22: end procedure
```

(c) LocateList(lists,ListName), 查找名字为 ListName 的线性表，并返回位序；

Algorithm 12 LocateList

```
1: procedure LocateList(Lists, ListName)
2:     k ← 0
3:     while k < Lists.length do
4:         if strcmp(Lists.elem[k].name, ListName) == 0 then
5:             return k + 1
6:         end if
7:         k ← k + 1
8:     end while
9:     return INFEASIBLE
10: end procedure
```

(d) Allput(lists)，输出所有的线性表中的元素，以 \n 换行；

Algorithm 13 Allput

```
1: procedure Allput(lists)
2:     for n ← 0; n < lists.length; n++ do
3:         printf("%s ", lists.elem[n].name)
4:         ListTraverse(lists.elem[n].L)
5:         putchar('\n')
6:     end for
7: end procedure
```

1.3.2 程序源代码

见《附录 A 基于顺序存储结构线性表实现的源程序》

1.4 系统测试

下面依次对重要功能进行测试，测试样例包括

1. list1{1,2,-4,4,5,-5,-3,-4}
2. null(空表)
3. 线性表不存在

表 1-1　线性表清空与线性表判空测试表

测试用例	输入	理论结果	测试结果
list1	4	线性表不为空!	线性表不为空!
	3	已清空线性表!	已清空线性表!
	4	线性表为空!	线性表为空!
	2	已销毁线性表!	已销毁线性表!
	4	判断失败！线性表不存在!	判断失败！线性表不存在!

1. 清空线性表与线性表判空的测试

对 list1 进行判空操作，然后清空 list1，再进行一次判空，最后销毁 list1，此时线性表不存在，再进行一次判空。测试数据如表1-1所示。测试结果如下图1-12所示。

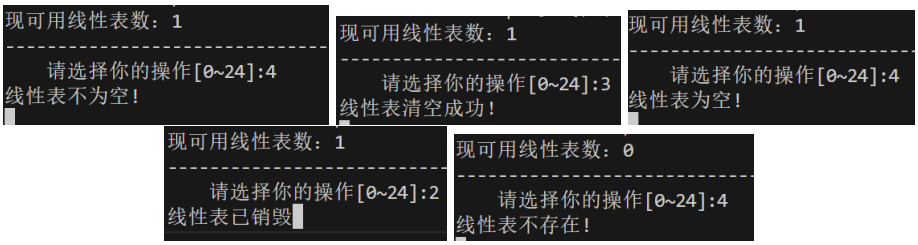

图 1-12　清空线性表与线性表判空测试组的截图

除标准输入外，我还进行了一些非法输入的测试，如下图1-13所示。

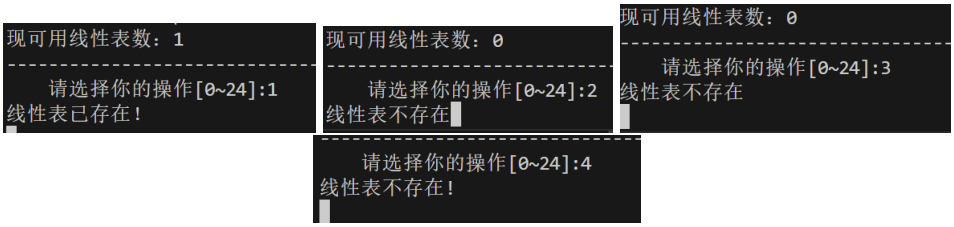

图 1-13　清空线性表与线性表判空测试组的截图（错误）

2. 获得、查找元素及其前驱后继的测试

对 list1 进行获得第三个元素的操作，然后对其进行查找并获取其前驱后继。测试数据如表1-2所示。

表 1-2　获得、查找元素及其前驱后继测试表

测试用例	输入	理论结果	测试结果
list1	6 3	获取元素值为-4	获取元素值为-4
	7 -4	查找元素在第 3 个	查找元素在第 3 个
	8 -4	查询元素前驱为 2	查询元素前驱为 2
	9 -4	查询元素后继为 4	查询元素后继为 4

测试结果如下图1-14所示。

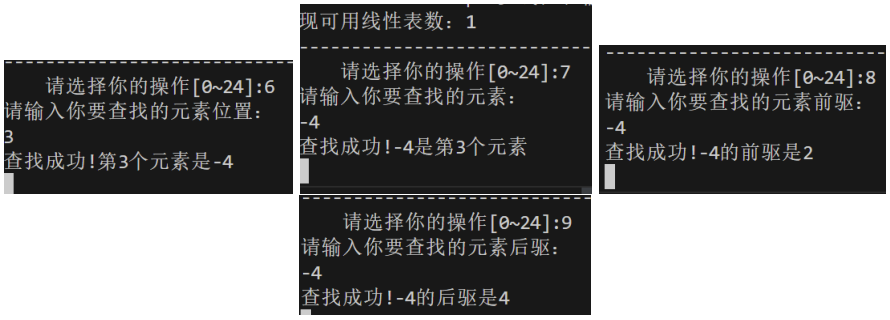

图 1-14　获得、查找元素及其前驱后继测试组的截图

除标准输入外，我还测试了一些非法输入，如下图1-15所示.

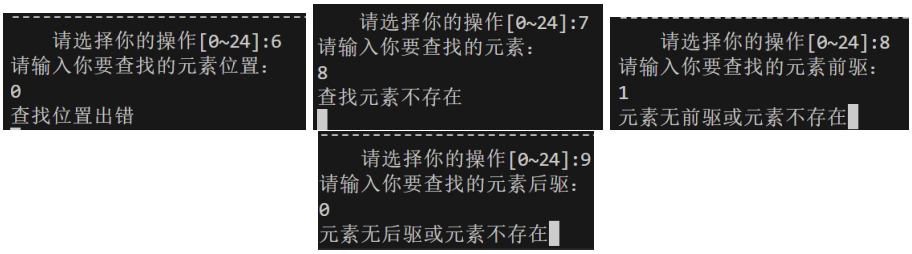

图 1-15　获得、查找元素及其前驱后继测试组的截图 (错误)

3. 插入、删除、遍历元素的测试对 list1 依次进行插入、删除和遍历操作，测试数据如表1-3所示。测试结果如下图1-16所示。

表 1-3　插入、删除、遍历元素测试表

测试用例	输入	理论结果	测试结果
list1	10 2 3	插入成功!	插入成功!
	12	1 2 2 -4 4 5 -5 -3 -4	1 2 2 -4 4 5 -5 -3 -4
	11 4	删除的元素为-4	删除的元素为-4
	12	1 2 2 4 5 -5 -3 -4	1 2 2 4 5 -5 -3 -4

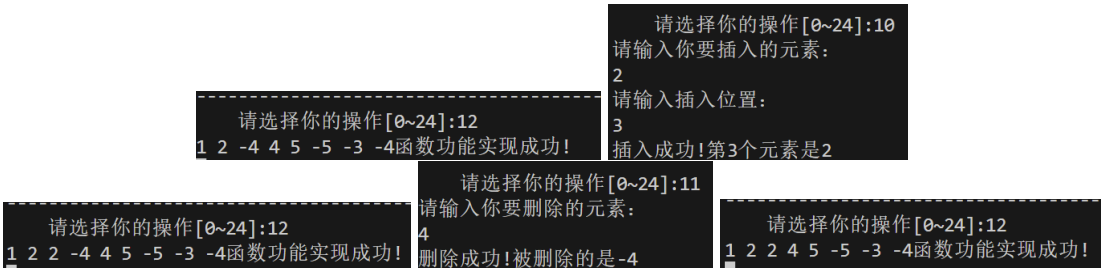

图 1-16　插入、删除、遍历元素测试组的截图

除标准输入外，我还测试了了非法输入，如下图1-17所示.

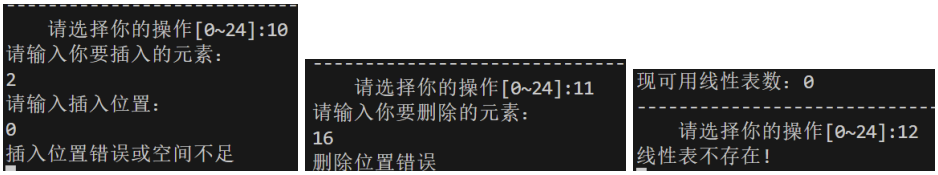

图 1-17　插入、删除、遍历元素测试组的截图 (错误)

4. 之后，我进行了对线性表文件保存与读取的测试，测试结果如下图1-18，1-19所示。

图 1-18　线性表文件保存

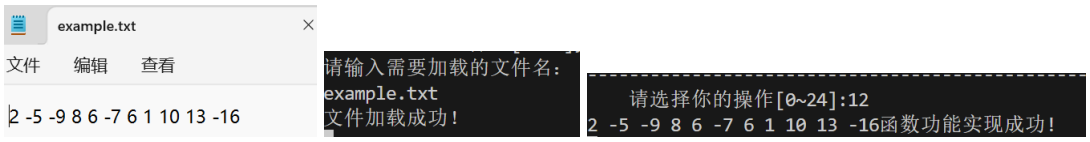

图 1-19　线性表文件读取

5. 使用从文件读取的数列进行最大连续子数组和和排序测试，结果如下图1-22所示.

图 1-20　最大连续子数组和和排序

6. 使用数列**-2,1,-3,4,-1,2,1,-5,4** 进行和为-1 的子数组和个数计算，结果如下图1-21所示。

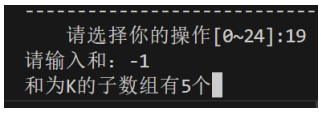

图 1-21　和为-1 的子数组和个数

7. 多线性表演示如图**??**所示，分别是：创建、输出、搜索、切换、移除：

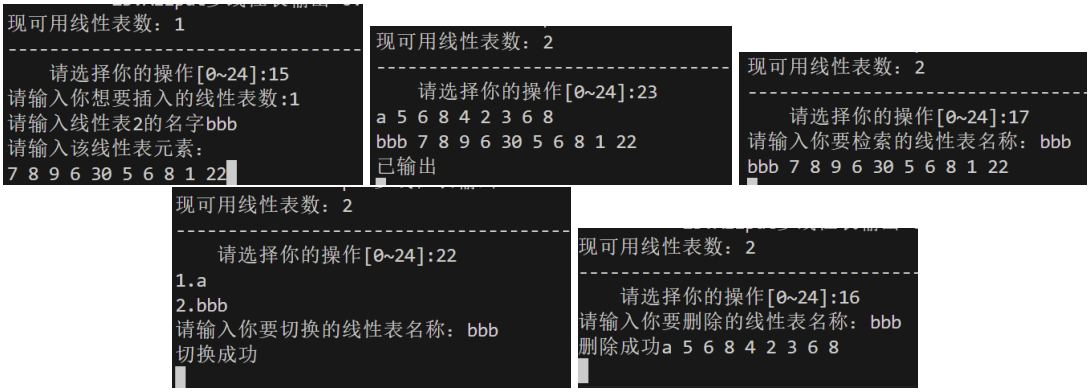

图 1-22　多线性表管理

以上结果均符合预期。

1.5 实验小结

本次实验让我对顺序储存结构的线性表的了解更进一步，基础功能整体难度不大，附加功能难度适中，多线性表管理较为复杂，最大连续子数组和、和为 K 的子数组等功能在基础算法方面也有一定的要求。

在这次实验中，最让人望而生畏的是要将是多种函数组合起来，而且不能照搬头歌的代码，要自己去写一些新的函数才能够让函数组合的更好，如何去调整主函数与子函数的关系，如何让代码的效率变得更高，如何提高代码的健壮性，都是需要认真思考并动手解决的问题。在排序等一些功能的实现中，并不需要函数将调整后的数组（或运行完的结果）传回主函数，可以直接输出，这样可以简化主函数的代码，子函数也没有影响。

在实现多表时，我第一次使用的是通过一个指针去指向一个线性表的表头，这样写会使得代码比较复杂，容易漏写代码导致程序出错，于是我又设计了上文中的结构，以 L.now 去表示当前使用的线性表的位序，在切换线性表时，只需要改变 L.now 的值就行。

总的来说，这次数据结构实验提高了我的编程能力，加深了对线性表的概念、基本运算的理解，掌握了线性表的逻辑结构与物理结构的关系，能熟练实现顺序表的基本运算，使我受益匪浅。

2 基于邻接表的图实现

图的集合 G 是由集合 V 和集合 E 组成，即 G=V,E，V 表示图中所有顶点的集合，E 表示顶点之间所有边的集合。

在线性表中，数据元素之间是被串起来的，仅有线性关系，每个数据元素只有一个直接前驱和一个直接后继。在树形结构中，数据元素之间有着明显的层次关系，并且每一层上的数据元素可能和下一层中多个元素相关，但只能和上一层中一个元素相关。图是一种较线性表和树更加复杂的数据结构。在图形结构中，结点之间的关系可以是任意的，图中任意两个数据元素之间都可能相关。

2.1 问题描述

要求图顶点类型为结构型，至少包含二个部分，一个是能唯一标识一个顶点的关键字（类似于学号或职工号），另一个是其它部分。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。如此，图就从一种数据结构变为了抽象数据类型。本次实验中，默认图为无向图。

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数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。

2.1.1 功能实现

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。具体运算功能定义和说明如下。具体运算功能定义如下：

1. 基础功能：
   (a) 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；
   注：
      i. 要求图 G 中顶点关键字具有唯一性。后面各操作的实现，也都要满足一个图中关键字的唯一性，不再赘述；

ii. V 和 VR 对应的是图的逻辑定义形式，比如 V 为顶点序列，VR 为关键字对的序列。不能将邻接矩阵等物理结构来代替 V 和 VR。

(b) 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；

(c) 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；

(d) 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；

(e) 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；

(f) 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；

(g) 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）；

(h) 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；

(i) 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>；

(j) 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>；

(k) 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；

(l) 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。

2. 附加功能：

(a) 距离小于 k 的顶点集合：函数名称是 VerticesSetLessThanK(G,v,k)，初始条件是图 G 存在；操作结果是返回与顶点 v 距离小于 k 的顶点集合；

(b) 顶点间最短路径和长度：函数名称是 ShortestPathLength(G,v,w); 初始条件是图 G 存在；操作结果是返回顶点 v 与顶点 w 的最短路径的长度；

(c) 图的连通分量：函数名称是 ConnectedComponentsNums(G)，初始条件是图 G 存在；操作结果是返回图 G 的所有连通分量的个数；

(d) 实现图的文件形式保存：其中，

i. 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；

ii. 需要设计图文件保存和加载操作合理模式。

(e) 实现多个图管理：设计相应的数据结构管理多个图的查找、添加、移除等功能。

2.1.2 实验目的

1. 加深对图的概念、基本运算的理解；
2. 熟练掌握图的逻辑结构与物理结构的关系；
3. 熟练掌握图的逻辑结构与物理结构的关系。

2.2 系统设计

2.2.1 系统总体设计

本实验以邻接表为主体，并定义新结构 **GQueen** 来实现多无向图功能，用户可以使用至多 20 个***互相独立***的无向图，对该结构整体有创建图、移除图、查找图、添加图和切换图等功能。系统实现的主要功能已在上一节列出。

系统通过一个简易的菜单来给使用者选择功能，每项功能后均有文字说明，使用时也有文字提示。通过清屏函数等操作，并用 **system** 相关函数调整窗口大小，使得菜单始终位于界面上方，便于使用。

主函数则是定义一些全局变量、编写引导并将各种功能函数实现，利用 **switch 分支结构**进行不同功能的实现，以达到连接各项功能函数的目的。

系统界面便于理解，功能均有文字解释，如图2-1所示：

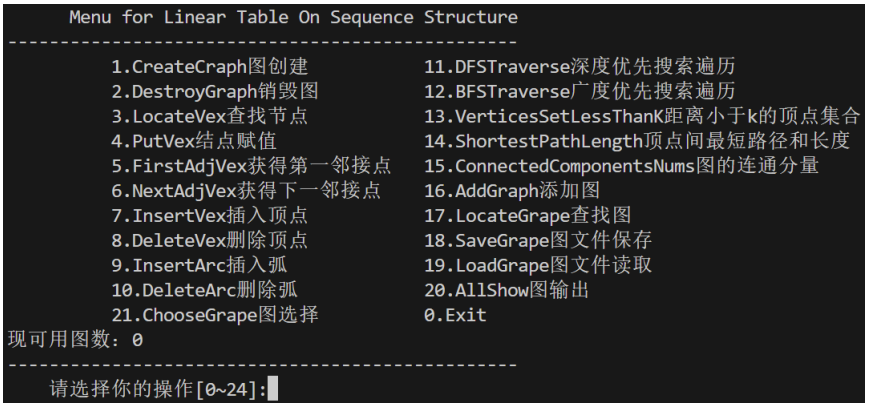

图 2-1　图菜单样式

2.2.2　数据结构设计

1. 首先，我定义了一些常量以及一些在代码中要用到的全局变量：

Algorithm 14 Macros and Typedefs

```
1:  define TRUE 1
2:  define FALSE 0
3:  define OK 1
4:  define ERROR 0
5:  define INFEASIBLE -1
6:  define OVERFLOW -2
7:  define MAX_VERTEX_NUM 20
8:  typedef int status
9:  typedef int KeyType
10: VertexType V[30]
11: KeyType VR[100][2]
12: VertexType value
13: int visited[MAX_VERTEX_NUM]
14: int dist[MAX_VERTEX_NUM]
```

2. 其次，我设计了如下结构体：

Algorithm 15 Queen Structures

(a)

```
1: typedef struct {
2:     KeyType elements[MAX_VERTEX_NUM]
3:     int front
4:     int rear
5: } Queue
```

Algorithm 16 VertexType Structures

(b)

```
1: typedef struct {
2:     KeyType key
3:     char others[300]
4: } VertexType
```

Algorithm 17 ArcNode Structures

(c)
```
typedef struct {
    int adjvex
    struct ArcNode *nextarc
} ArcNode
```

Algorithm 18 VNode Structures

(d)
```
typedef struct {
    VertexType data
    ArcNode *firstarc
} VNode, AdjList[MAX_VERTEX_NUM]
```

Algorithm 19 ALGraph Structures

(e)
```
typedef struct {
    AdjList vertices
    int vexnum, arcnum
    GraphKind kind
} ALGraph
```

Algorithm 20 GQueen Structures

(f)
```
typedef struct {
    char name[30]
    ALGraph G
} elem[MAX_VERTEX_NUM]
int length
int now
} GQueen
```

2.3 系统实现

2.3.1 算法设计

再次说明：我使用菜单栏的方式对基于邻接表的图进行实现。具体设计是，每次先输入所选用的操作序号，然后根据提示输入该操作所需要的操作，以此对

邻接表进行实现。为实现多图管理，我们建立结构体以存放多图，并实现图的增删和建立与初始化。我们下面一一列举每一功能的具体实现。

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。

操作思路：先一一增加图的顶点，如果有两个顶点的关键词相同，则返回 ERROR；再一一读取边 (u, v)，由于我们实现的是无向图，只需分别在 u 顶点添加边 v，并在 v 顶点添加边 u。

特别指出：由于使用的是尾插法，在进行图的保存与读取时要对原本的输入顺序进行还原。流程图如下：

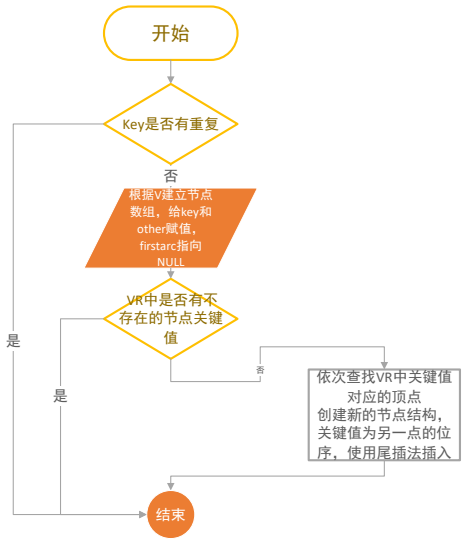

图 2-2　创建图

2. 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；该算法的设计思想是，对每一顶点后续的弧进行 free 操作，而后将图 G 的顶点个数与弧个数置零。由于其实现简单，只需遍历各个顶点链表即可，此处不再赘述。

3. 查找结点

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；

操作思路：遍历结点数组，寻找 key 值与 u 相等的顶点，返回其位序，如果没找到，则返回-1。如下所示：

Algorithm 21 Locate Vertex

```
1: function LocateVex(G, u)
2:     for i ← 0 to G.vexnum − 1 do
3:         if G.vertices[i].data.key == u then
4:             return i
5:         end if
6:     end for
7:     return −1
8: end function
```

4. 结点赋值

   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；操作思路：在 LocateVex(G,u) 的基础上，对于返回位序所指的顶点的 key 与 other 重新赋值，思路简单，无需赘述。

5. 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；

   操作思路：先找到该顶点，直接返回它的 firstarc->adjvex，无需赘述。

6. 获得下一邻接点

   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；

   操作思路：先定位两个顶点，确定两个位序，在 v 对应顶点的邻接顶点中找到 adjvex==w 的边，返回它的 nextarc->adjvex。流程图如下：

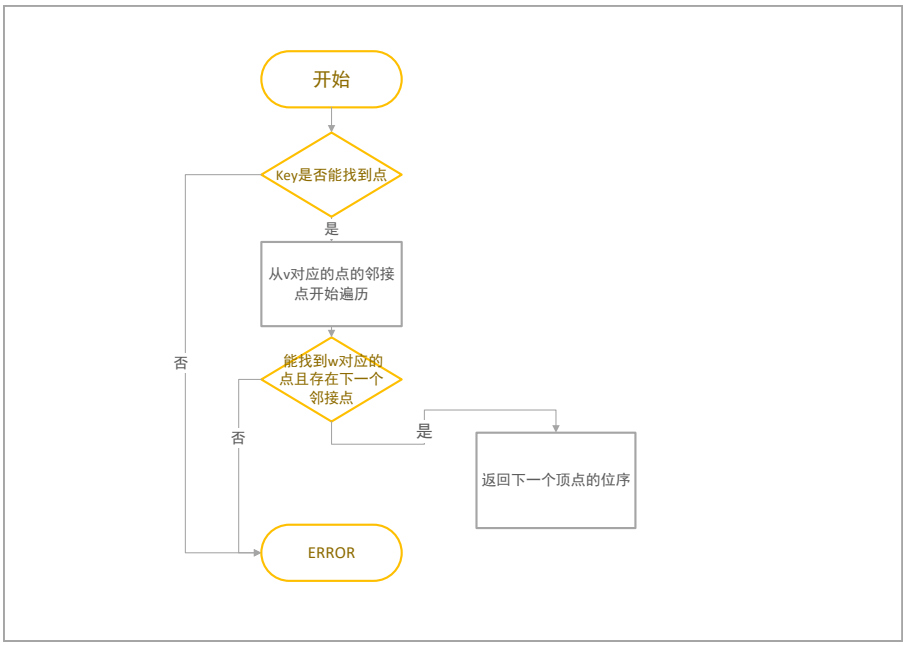

图 2-3　下一邻接点

7. 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)；

操作思路：在顶点数组的末尾加一个顶点元素，顶点数加一。算法如下：

Algorithm 22 Insert Vertex

```
1:  function InsertVex(G, v)
2:      for i ← 0 to G.vexnum − 1 do
3:          if G.vertices[i].data.key == v.key then
4:              return ERROR
5:          end if
6:      end for
7:      if G.vexnum < MAX_VERTEX_NUM then
8:          G.vertices[G.vexnum].data ← v
9:          G.vexnum ← G.vexnum + 1
10:         return OK
11:     else
12:         return ERROR
13:     end if
14: end function
```

8. 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；

操作思路：先找到需删除的顶点 x 的位置，将顶点删除之后，将后续的顶点依次向前挪动一格；对于边的记录，只需把关键字等于 x 的边删去，调整点数与边数即可。流程图如下：

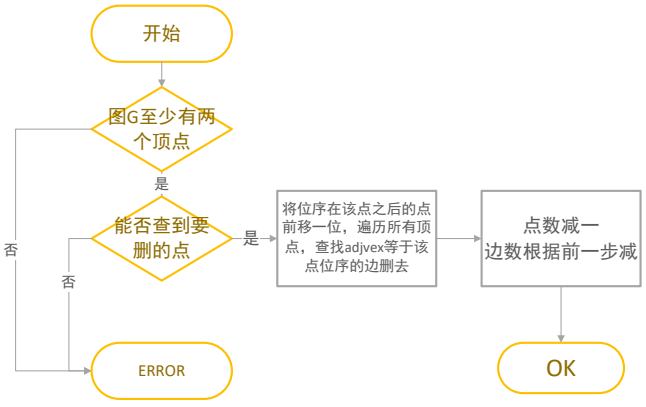

图 2-4　删除顶点

9. 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>；

   操作思路：找到两个顶点，使用尾插法分别给两个顶点插上边结构。是 CreateCraph(G,V,VR) 的部分功能，思路简单，细节较多，详细参见源代码。

10. 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>；

    操作思路：找到结点 w，遍历结点 w 的弧链表，查找与 v 相等的结点，如果找到，则删去；否则返回错误。对结点 v 重复以上流程，边数减一。是 DeleteVex(G,v)，的部分功能，思路简单，细节较多，详细参见源代码。

11. 深度优先搜索遍历

    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；

    操作思路：建立一个 vis[] 数组以保存结点是否以访问的信息，以保证每个结点只被访问一次。如果我们正在访问一个结点，并且如果和它相邻的结点没有被访问，那么我们将跳转至下一结点进行进一步的搜索。这是深度优先搜索 (Depth-First Search) 的实现思想。算法如下：

Algorithm 23 Depth-First Search (DFS)

```
1:  function DFS(G, v, visited, visit)
2:      visited[v] ← 1
3:      VISIT(G.vertices[v].data)
4:      p ← G.vertices[v].firstarc
5:      while p ≠ NULL do
6:          w ← p.adjvex
7:          if ¬visited[w] then
8:              DFS(G, w, visited, visit)
9:          end if
10:         p ← p.nextarc
11:     end while
12: end function
```

Algorithm 24 Depth-First Search Traversal

```
1:  function DFSTRAVERSE(G, visit)
2:      visited[MAX_VERTEX_NUM] ← {0}
3:      for i ← 0 to G.vexnum − 1 do
4:          if ¬visited[i] then
5:              DFS(G, i, visited, visit)
6:          end if
7:      end for
8:      return OK
9:  end function
```

12. 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。

操作思路：建立一个 vis[] 数组以保存结点是否以访问的信息，以保证每个结点只被访问一次。如果我们正在访问一个结点，那么我们将和它相邻的并且和未被访问的结点压入访问队列中。每次在访问队列中去队首元素进行访问，并在 vis 数组中标记其已被访问。这是广度优先搜索 (Breadth-First

Search) 的实现思想，算法如下：

Algorithm 25 Breadth-First Search Traversal

Require: G is a graph, $visit$ is a function to visit a vertex

Ensure: Traverses the graph G using BFS and visits each vertex exactly once

```
1:  function BFSTraverse(G, visit)
2:      visited ← [FALSE] × MAX_VERTEX_NUM
3:      Queue Q
4:      InitQueue(Q)
5:      for i ← 0 to G.vexnum − 1 do
6:          if ¬visited[i] then
7:              visited[i] ← TRUE
8:              visit(G.vertices[i].data)
9:              EnQueue(Q, i)
10:             while ¬QueueEmpty(Q) do
11:                 DeQueue(Q, v)
12:                 p ← G.vertices[v].firstarc
13:                 while p ≠ NULL do
14:                     w ← p.adjvex
15:                     if ¬visited[w] then
16:                         visited[w] ← TRUE
17:                         visit(G.vertices[w].data)
18:                         EnQueue(Q, w)
19:                     end if
20:                     p ← p.nextarc
21:                 end while
22:             end while
23:         end if
24:     end for
25:     return OK
26: end function
```

13. 距离小于 k 的顶点集合

    函数名称是 VerticesSetLessThanK(G,v,k)，初始条件是图 G 存在；操作结果是返回与顶点 v 距离小于 k 的顶点集合；

    操作思路：使用广度优先遍历 k-1 次，将遍历到的点写入顶点数组即可。思路简单，是 BFSTraverse(G,visit()) 的部分功能，不需赘述。

14. 顶点间最短路径和长度

    函数名称是 ShortestPathLength(G,v,w); 初始条件是图 G 存在；操作结果是返回顶点 v 与顶点 w 的最短路径的长度；

    操作思路：依旧使用广度优先遍历，记录已遍历的点，直到遍历到另一点为止，返回遍历层数。思路简单，是 BFSTraverse(G,visit()) 的部分功能，不需赘述。

15. 图的连通分量

    函数名称是 ConnectedComponentsNums(G)，初始条件是图 G 存在；操作结果是返回图 G 的所有连通分量的个数；

    操作思路：依旧使用广度优先遍历，记录已遍历的点，直到遍历完成，检查是否有未遍历的点，若无，返回连通分量数（默认 1）；若有，连通分量数加一，重复以上步骤。思路简单，是 BFSTraverse(G,visit()) 的部分功能，不需赘述。

16. 实现图的文件形式保存

    (a) 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；操作思路：上文已经提到，要还原最开始的输入顺序，使用一个递归算法26完成。保存只是建立两个数组记录顶点数据和边数据。

Algorithm 26 Write Arc Information

Require: VR is a 2D array, p is a pointer to an ArcNode, cnt is a reference to an integer, i is an integer

Ensure: Writes arc information into VR

```
1:  function WArc(VR, p, cnt, i)
2:      if p = NULL then
3:          return
4:      end if
5:      if p → nextarc ≠ NULL then
6:          WArc(VR, p → nextarc, cnt, i)
7:      end if
8:      for j ← 0 to cnt − 1 do
9:          if VR[j][1] = i and VR[j][0] = p → adjvex then
10:             return
11:         end if
12:     end for
13:     VR[cnt][0] ← i
14:     VR[cnt][1] ← p → adjvex
15:     cnt ← cnt + 1
16: end function
```

(b) 要设计图文件加载操作合理模式。

操作思路：从文件中读取数组 V，数组 VR，使用 CreateCraph(G,V,VR) 函数生成图。思路简单，不需赘述。

注：保存到文件后，可以直接加载文件生成图。

17. 实现多个图管理

设计相应的数据结构管理多个图的查找、添加、移除等功能。

注：实现多个图管理应能创建、添加、移除多个图，单图和多图的区别仅仅在于图管理的个数不同，多图管理应可自由切换到管理的每个图，并可单独对某图进行单图的所有操作。

对此，我定义了如下的结构体：

Algorithm 27 GQueen Structures

```
1: typedef struct {
2:     char name[30]
3:     ALGraph G
4: } elem[MAX_VERTEX_NUM]
5: int length
6: int now
7: } GQueen
```

多图操作只是对 GQeen 结构中的 elem 数组进行操作，使用 now 指示当前图是数组 elem 中的哪一个元素，详见源代码，在此不再赘述。

2.3.2 程序源代码

见《附录 B 基于邻接表无向图实现的源程序》

2.4 系统测试

下面依次对重要功能进行测试，测试样例包括

1. Graph1={5 线性表, 8 集合, 7 二叉树, 6 无向图, -1 nil, 5 6, 5 7, 6 7, 7 8, -1 -1}
2. Graph2={1 a, 2 b, 3 c, 4 d, 5 e, 6 f, 7 g, 8 h, 9 i, -1 nil, 1 2, 1 3, 2 4, 2 8, 3 6, 4 5, 4 6, 5 9, 6 7, -1 -1}
3. 图不存在

1. 查找顶点、顶点赋值、获取邻接点的测试

对 Graph1 进行操作，先查找关键字为 5 的结点，然后将该节点的关键字赋为 9，查询其第一邻接点及第一邻接点后的下一邻接点。测试数据如表2-1所示：

表 2-1　查找顶点、顶点赋值、获取邻接点测试表

测试用例	输入	理论结果	测试结果
Graph1	3 5 4 5 9 线性表 5 9 6 9 7	5 线性表 赋值成功! 7 二叉树 6 无向图	5 线性表 赋值成功! 7 二叉树 6 无向图

测试结果如下图2-5所示：

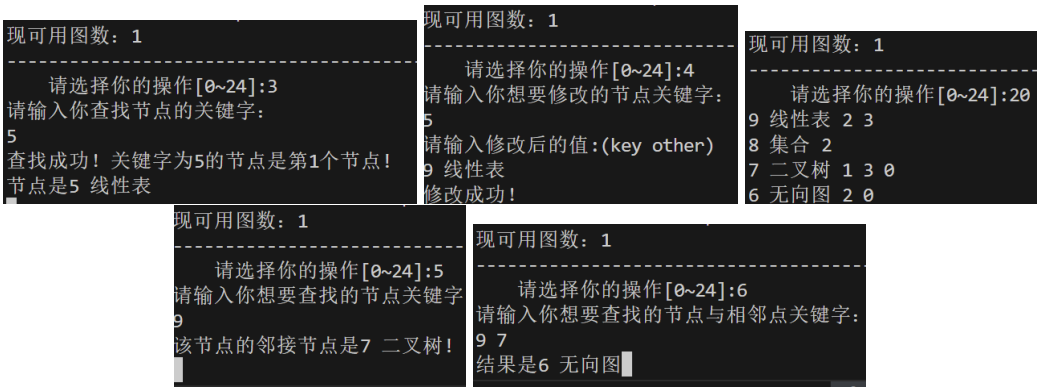

图 2-5　查找顶点、顶点赋值、获取邻接点测试组的截图

之后我还进行了错误输入的测试，依次为查找结点不存在、赋值结点不存在、没有邻接点的情况，测试结果如下图2-6所示：

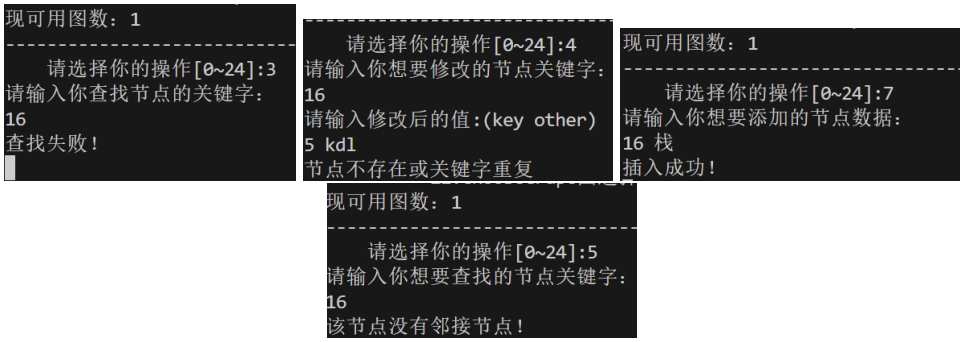

图 2-6　查找顶点、顶点赋值、获取邻接点测试组的截图（错误）

2. 插入、删除、深度优先遍历、广度优先遍历的测试

对 Graph1 进行操作，先插入关键字为 9 的结点，再插入关联关键字为 5 和 9 的弧，利用深度优先搜索遍历检验正确性。再删除关键字为 5 的结点，删除关联关键字为 7 和 8 的弧，用广度优先搜索遍历检验正确性。测试数据如表2-2所示：

表 2-2　插入、删除、两种特殊遍历测试表

测试用例	输入	理论结果	测试结果
Graph1	7 9 有向图	插入成功!	插入成功!
	9 5 9	插入成功!	插入成功!
	11	5 9 7 8 6	5 9 7 8 6
	8 5	删除成功!	删除成功!
	10 7 8	删除成功!	删除成功!
	12	8 7 6 9	8 7 6 9

测试结果如下图2-7所示：

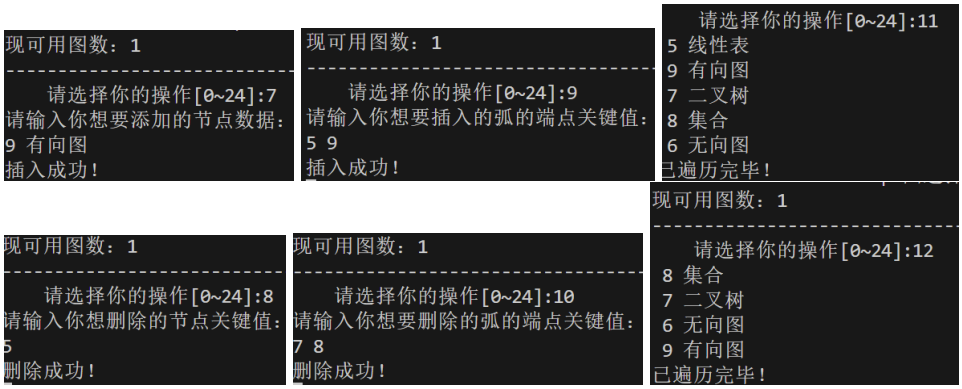

图 2-7　插入、删除、两种特殊遍历测试组的截图

之后我还进行了错误输入的测试，依次为为插入结点关键字重复、插入弧已存在、删除节点不存在、删除弧不存在的情况。测试结果如下图2-8所示：

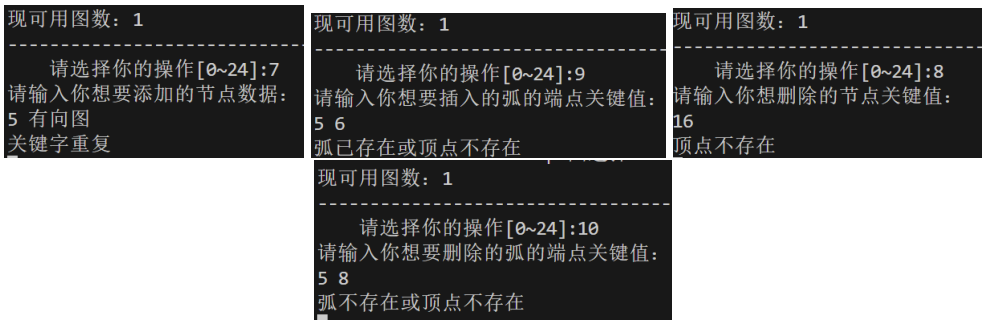

图 2-8　插入、删除、两种特殊遍历测试组的截图（错误）

3. 附加功能的测试

对 Graph2 进行操作，求与关键字为 2 的结点距离小于 2 的所有结点，计算关键字为 2 和 7 的结点之间的最短路径，删除关键字为 5 的结点后，求图的连通分量。测试数据如表2-3表示：

表 2-3　附加功能测试表

测试用例	输入	理论结果	测试结果
Graph1	13 2 2 14 2 7 8 5 15	1 a 2 b 4 d 8 h 3 2	1 a 2 b 4 d 8 h 3 2

测试结果如下图2-9所示：

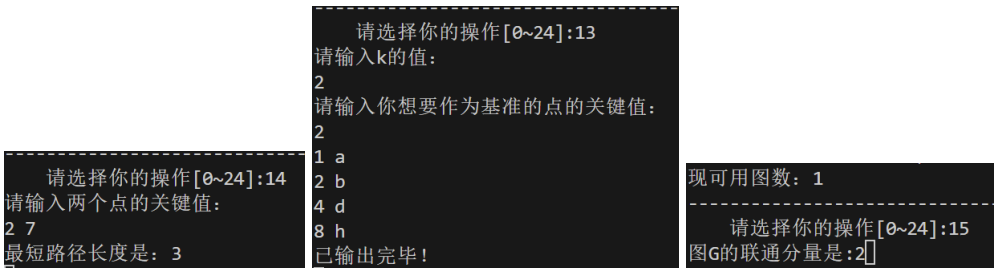

图 2-9　附加功能测试组的截图

4. 多图演示如图2-10所示, 分别是创建、切换、查找，移除就依赖于上文的 DestroyGraph(G) 功能了：

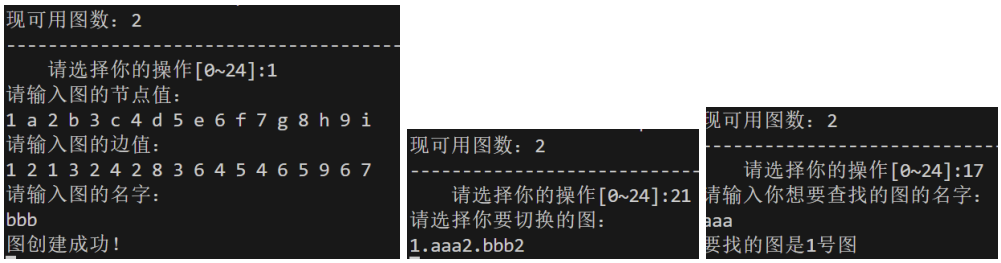

图 2-10　多线性表管理

5. 图文件的保存，如图2-11所示：

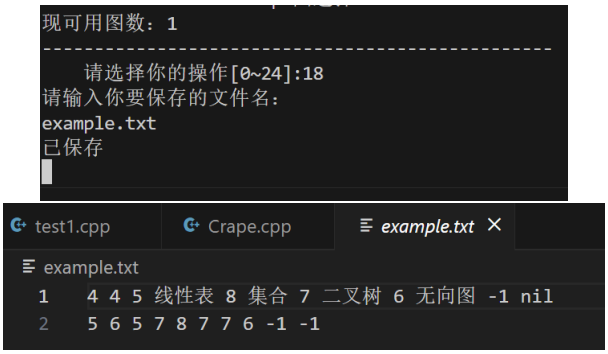

图 2-11　图文件保存

将图销毁后进行图文件的读取，如图2-12所示：

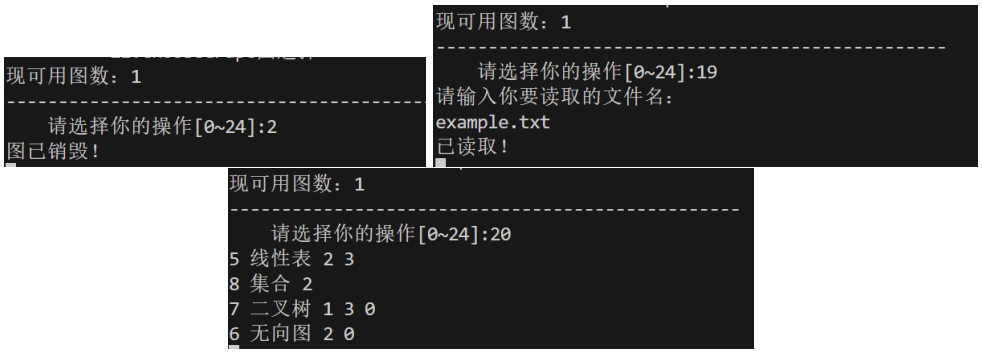

图 2-12　图文件读取

以上测试均符合预期。

2.5 实验小结

本次实验以实现基于邻接表的图这一数据结构及相关功能为主要目的。本章实验的难度应当是所有章节中最难的，耗时也是最长。本章的基础功能较为困难，插入和删除结点和弧实现较为复杂，使用尾插法又对图文件的保存提出了要求，而附加功能中文件和多图系统可参照前三次实验，最短路径等功能算法依靠灵感改造了广度优先算法，难度不大。

在这次实验中，我学会了很多，比如如何通过递归去实现深度优先算法；如何通过队列去实现广度优先算法，以及通过广度优先算法去实现最短路径等功能（尽管实际意义不大）。关于图的功能算法帮我巩固了图论的有关知识。

由于掌握不牢，我在使用尾插法插入边节点时，我经过了多次的失败，最终发现是在初始化时没有让顶点的指针域指向 NULL，所幸在后来写插入边的代码时想到了这一点，将其改正了过来；在删除图中顶点时，如果输入了不存在的关键值，会自动删去第一个顶点，后来发现是因为用来标记被删除顶点的变量没有初始化，改正了这一点后，功能就正常了。

总的来说，这次数据结构实验提高了我的编程能力，加深了对图这一数据结构的理解，

3 课程的收获和建议

本次数据结构实验给我的感觉与上学期的 C 语言实验完全不同，后者只是去完成头歌上的题目，最后汇集所有代码，写出一篇报告即可；前者不仅仅需要汇总所有的代码，还需要自己去写菜单，去写将所有的功能串联起来的核心，难度提升了许多。当然，难度的提升也伴随着技能的成长，我在这次数据结构实验中学到了很多，对于编程有了许多经验，也算是突破了自我。这是我第一次写下了超过一千行的代码文件（甚至不止一个）；也是我第一次构造一个图形化的“菜单”去罗列功能；也让我第一次以一个用户的视角去看待一个程序实现功能，这也使我明白，在程序中，提示语是不可或缺的，否则用户得不到反馈，也不知道如何去使用程序；这也是我第一次使用 LaTeX 去写实验报告，感觉与 Word 各有千秋。最后，感谢一直以来辅导以及检查我的系统的老师，也感谢一直努力的自己。

关于课程的一些小建议：

1. 因为实验课以前还有物理实验，课程早开不太好，那么可以提前把任务书发给学生，让学生试着去尝试，这样也免得一些同学在写代码时发现编译器存在问题，拖慢了速度。
2. 由于开学初学生的负担是比较轻的，实验课可以去填补这份空缺，而且可以拉长授课时间，改为两周一次线下课，这样可以有更加充足的时间去写程序和实验报告。

3.1 基于顺序存储结构的线性表实现

我认为这一章基于数组的应用，是最简单的一章。最大的作用是让学生初次接触通过主函数串联各个子函数去实现一个大的数据结构；并且复习了上学期关于数组的知识，顺序存储结构的线性表既能随机存取，也方便实现排序等功能，以上学期的水平来讲是可以完成的；关于结构体和文件读写部分的知识，锦上添花。

3.2 基于链式存储结构的线性表实现

链式存储结构线性表与上一章顺序表差别不大，基础功能只是将数组实现改为链表实现。通过本章以及与上一章的比较可以发现，算法并不因数据结构的实现方式而改变，例如对于单链表，冒泡排序、选择排序等排序算法依然成立，且实现方法相似。最大的难点在于指针的使用，毕竟上学期就学得头疼。这学期数据结构理论课上还学习了双链表、十字链表、循环链表等知识，这一章有助于同学们掌握指针及链表的知识。

3.3 基于二叉链表的二叉树实现

我认为二叉树是《数据结构》课程中极为重要的一章。本章中四种遍历方式都运用了递归的思想，前序遍历非递归的实现也帮助我巩固了堆栈这一数据结构的用法。附加功能中最近公共祖先、翻转二叉树等功能再次体现了递归的思想。空间的释放是最容易出错的部分，常常因为空间的释放导致运行错误。由于链表的复杂，我觉得给结点的结构增加一个指向其父亲节点的指针是很有用的。

3.4 基于邻接表的图实现

我认为基于邻接表的图是课程最为复杂的一章，用到了先前练习的顺序表和链表。本章中关于深度优先搜索和广度优先搜素两种遍历方法的实现对我有较大的帮助，深入理解了递归思想和队列的使用。得益于对图论的了解和对链表的掌握，在写某些子函数时还是比较得心应手的。

4 附录 A 基于顺序存储结构线性表实现的源程序

```
/* Linear Table On Sequence Structure */
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

/*---------page 10 on textbook ---------*/
#define TRUE 1
#define MAX 10
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
typedef int status;
typedef int ElemType;
#define LIST_INIT_SIZE 100
#define LISTINCREMENT  20
#define MAXSIZE 100
typedef struct{
      ElemType * elem;
      int length;
      int listsize;
 }SqList;
typedef struct{
    struct {
        char name[30];
        SqList L;
    } elem[10];
    int length;
    int now;
}LISTS;

status InitList(SqList &L);
status DestroyList(SqList &L);
status ClearList(SqList& L);
status ListEmpty(SqList L);
```

```
39 status ListLength(SqList L);
40 status GetElem(SqList L,int i,ElemType &e);
41 status LocateElem(SqList L,ElemType e);
42 status PriorElem(SqList L,ElemType e,ElemType &pre);
43 status NextElem(SqList L,ElemType e,ElemType &next);
44 status ListInsert(SqList &L, int i, ElemType e);
45 status ListDelete(SqList &L, int i, ElemType &e);
46 status ListTraverse(SqList L);
47 status SaveList(SqList L, char FileName[]);
48 status LoadList(SqList &L, char FileName[]);
49 status AddList(LISTS &Lists,char ListName[]);
50 status RemoveList(LISTS &Lists,char ListName[]);
51 int LocateList(LISTS Lists,char ListName[]);
52 int MaxSubArray(int *L,int n);
53 int SubArrayNum(int *L,int n,int k);
54 void sortList(int *L,int n);
55 status Make_Person(SqList &L);
56 void Allput(LISTS &lists);
57 //主函数
58 int main()
59 {
60 char name[30];
61 LISTS lists;
62 lists.length=0;
63 lists.now=-1;
64 int op=1;
65 while(op){
66         system("cls");  printf("\n\n");
67         printf("      Menu for Linear Table On Sequence Structure \n");
68         printf("-------------------------------------------------\n");
69         printf("          1. InitList创建表\t          11. ListDelete删除元
              素\n");
70         printf("          2. DestroyList销毁表\t          12.
              ListTrabverse遍历输出线性表 \n");
71         printf("          3. ClearList清空表\t          13. SaveList保存线性
              表到文件  \n");
72         printf("          4. ListEmpty判空表\t          14. LoadList从文件加
              载线性表\n");
73         printf("          5. ListLength表长\t          15. AddList添加线性
              表\n");
```

```
74          printf("          6. GetElem获得元素\t          16. RemoveList移除一
                个线性表\n");
75          printf("          7. LocateElem定位元素\t          17. LocateList查
                找一个线性表\n");
76          printf("          8. PriorElem获得先驱元素\t    18. MaxSubArray最大
                连续子数组和\n");
77          printf("          9. NextElem获得后驱元素\t     19. SubArrayNum和为k
                的子数组数\n");
78          printf("          10. ListInsert插入元素\t      20. sortList升序排列
                \n");
79          printf("          21. Make_person输入元素, 覆写\t22.Listchoose线性表
                选择\n");
80          printf("          23.Allput多线性表输出\t       0. Exit\n");
81          printf("现可用线性表数: %d\n",lists.length);
82          printf("-------------------------------------------------\n");
83          printf("    请选择你的操作[0~24]:");
84          scanf("%d",&op);
85          ElemType e=0;
86          ElemType n=0;
87          ElemType k=0;
88          ElemType j=0;
89      switch(op){
90              case 1:
91                  if(lists.now==-1) printf("线性表未创建\n");
92                  else if (lists.elem[lists.now].L.elem!=NULL) printf("线性表
                        已存在! \n");
93                  else
94                  {
95                          if(InitList(lists.elem[lists.now].L)==OK)
96                          {
97                                  printf("线性表创建成功! \n");
98                                  printf("请输入线性表名称: ");
99                                  scanf("%s",lists.elem[lists.now].name);
100                         }
101                         else printf("线性表创建失败! \n");
102                 }
103                  getchar();getchar();
104                  break;
105             case 2:
106                 if(lists.elem[lists.now].L.elem==NULL)
107                 {
```

```
108                     printf("线性表不存在");
109                 }
110                 else if(DestroyList(lists.elem[lists.now].L)==OK)
111                 {
112                     printf("线性表已销毁");
113                     lists.length--;
114                     lists.now--;
115                 }
116                 else printf("线性表销毁失败! \n");
117                 getchar();getchar();
118                 break;
119         case 3:
120                 if(ClearList(lists.elem[lists.now].L)==OK) printf("线性表清
                        空成功! \n");
121                 else if(ClearList(lists.elem[lists.now].L)!=OK) printf("线性
                        表不存在\n");
122                 else printf("线性表清空失败!\n");
123             getchar();getchar();
124                 break;
125         case 4:
126                 if(ListEmpty(lists.elem[lists.now].L)==OK) printf("线性表为
                        空!\n");
127                 else if(ListEmpty(lists.elem[lists.now].L)==INFEASIBLE)
                        printf("线性表不存在!\n");
128                 else printf("线性表不为空!\n");
129                  getchar();getchar();
130                  break;
131             case 5:
132              if(lists.elem[lists.now].L.elem==NULL) printf("线性表不存在!\n")
                    ;
133                 else if(ListLength(lists.elem[lists.now].L)==lists.elem[
                        lists.now].L.length) printf("函数功能实现成功!表长为%d\n"
                        ,lists.elem[lists.now].L.length);
134                 else printf("函数功能实现失败!\n");
135                  getchar();getchar();
136                  break;
137             case 6:
138              printf("请输入你要查找的元素位置: \n");
139                 scanf("%d",&j);
140                  if(GetElem(lists.elem[lists.now].L,j,e)==OK) printf("查找成
                        功!第%d个元素是%d\n",j,e);
```

```
141                 else if(lists.elem[lists.now].L.elem==NULL) printf("线性表
                        不存在!\n");
142                 else if(GetElem(lists.elem[lists.now].L,j,e)==ERROR) printf
                        ("查找位置出错\n");
143                 else printf("查找失败!\n");
144                  getchar();getchar();
145                  break;
146             case 7:
147                 printf("请输入你要查找的元素: \n");
148                 scanf("%d",&j);
149                 e=LocateElem(lists.elem[lists.now].L,j);
150                 if(e!=0) printf("查找成功!%d是第%d个元素\n",j,e);
151                 else if(lists.elem[lists.now].L.elem==NULL) printf("线性表不
                        存在!\n");
152                 else printf("查找元素不存在\n");
153                  getchar();getchar();
154                  break;
155             case 8:
156             e=0;
157             printf("请输入你要查找的元素前驱: \n");
158             scanf("%d",&j);
159                 if(PriorElem(lists.elem[lists.now].L,j,e)==OK) printf("查找
                        成功!%d的前驱是%d\n",j,e);
160                 else if(lists.elem[lists.now].L.elem==NULL) printf("线性表不
                        存在!\n");
161                 else if(PriorElem(lists.elem[lists.now].L,j,e)==ERROR)
                        printf("元素无前驱或元素不存在");
162                 else printf("函数功能实现失败!\n");
163                 e=0;
164                  getchar();getchar();
165                  break;
166             case 9:
167             e=0;
168             printf("请输入你要查找的元素后驱: \n");
169             scanf("%d",&j);
170                 if(NextElem(lists.elem[lists.now].L,j,e)==OK) printf("查找成
                        功!%d的后驱是%d\n",j,e);
171                 else if(lists.elem[lists.now].L.elem==NULL) printf("线性表不
                        存在!\n");
172                 else if(NextElem(lists.elem[lists.now].L,j,e)==ERROR) printf
                        ("元素无后驱或元素不存在");
```

```
173                 else printf("函数功能实现失败!\n");
174                 e=0;
175                  getchar();getchar();
176                  break;
177             case 10:
178              printf("请输入你要插入的元素: \n");
179              scanf("%d",&e);
180                 printf("请输入插入位置: \n") ;
181                 scanf("%d",&j);
182                 if(lists.elem[lists.now].L.elem==NULL&&ListInsert(lists.elem
                        [lists.now].L,j,e)==INFEASIBLE) printf("线性表不存在!\n")
                        ;
183                 else if(ListInsert(lists.elem[lists.now].L,j,e)==OK) printf(
                        "插入成功!第%d个元素是%d\n",j,lists.elem[lists.now].L.
                        elem[j-1]);
184                 else if(ListInsert(lists.elem[lists.now].L,j,e)==ERROR)
                        printf("插入位置错误或空间不足\n");
185                 else printf("函数功能实现失败!\n");
186                  getchar();getchar();
187                  break;
188             case 11:
189             e=0;
190             printf("请输入你要删除的元素: \n");
191             scanf("%d",&j) ;
192                 if(ListDelete(lists.elem[lists.now].L,j,e)==OK) printf("删除
                        成功!被删除的是%d\n",e);
193                 else if(lists.elem[lists.now].L.elem==NULL) printf("线性表不
                        存在!\n");
194                 else if(ListDelete(lists.elem[lists.now].L,j,e)==ERROR)
                        printf("删除位置错误\n");
195                 else printf("函数功能实现失败!\n");
196                  getchar();getchar();
197                  break;
198             case 12:
199                 if(ListTraverse(lists.elem[lists.now].L)==OK) printf("函数功
                        能实现成功!\n");
200                 else if(lists.elem[lists.now].L.elem==NULL) printf("线性表不
                        存在!\n");
201                 else printf("函数功能实现失败!\n");
202                 getchar();getchar();
203                  break;
```

```
204                 case 13:
205                 printf("请输入你想要保存的文件名: \n");
206                 scanf("%s",name);
207                 if(SaveList(lists.elem[lists.now].L,name)==OK) printf("线性
                        表写入成功!\n");
208                 else if(SaveList(lists.elem[lists.now].L,name)!=OK&&lists.
                        elem[lists.now].L.elem==NULL) printf("线性表不存在");
209                 else printf("函数功能实现失败!\n");
210                 getchar();getchar();
211                 break;
212                 case 14:
213                 if(lists.elem[lists.now].L.elem!=NULL) printf("线性表已存
                        在! \n");
214                 else if(lists.elem[lists.now].L.elem!=NULL)
215                 {
216                         printf("请输入你想要读取的文件名: \n");
217                         scanf("%s",name);
218                         if(LoadList(lists.elem[lists.now].L,name)==OK)
219                         {
220                                 printf("文件导出成功!\n");
221                         }
222                 }
223                 else printf("函数功能实现失败!\n");
224                 getchar();getchar();
225                 break;
226                 case 15:
227                 printf("请输入你想要插入的线性表数:");
228                 scanf("%d", &n);
229                 while(n--)
230                 {
231                         printf("请输入线性表%d的名字",lists.length+1);
232                 scanf("%s",name);
233                 if(LocateList(lists ,name)==-1)
234                 {
235                         AddList(lists ,name);
236                                 printf("请输入该线性表元素: \n");
237                         while (scanf("%d",&e))
238                         {
239                                 ListInsert(lists.elem[lists.length-1].L,
                                        lists.elem[lists.length-1].L.length+1,e);
240                                 if(getchar()=='\n') break;
```

```
241                         }
242                         }
243                         else
244                         {
245                                 printf("你重复输入线性表名称, 请重新输入\n")
                                        ;
246                                 n++;
247                         }
248                 }
249                 lists.now=lists.length-1;
250                 getchar();getchar();
251                 break;
252                 case 16:
253                 printf("请输入你要删除的线性表名称: ");
254                 scanf("%s",name);
255                 if (RemoveList(lists,name)==OK)
256                 {
257                         printf("删除成功");
258                 for(n=0;n<lists.length;n++)
259                         {
260                                 printf("%s ",lists.elem[n].name);
261                                 ListTraverse(lists.elem[n].L);
262                         putchar('\n');
263                         }
264                 }
265                 else if(lists.length==0) printf("无线性表\n");
266                 else printf("删除失败");
267                 getchar();getchar();
268                 break;
269                 case 17:
270                 printf("请输入你要检索的线性表名称: ");
271                 scanf("%s",name);
272                 if(n=LocateList(lists,name))
273                 {
274                         printf("%s ",lists.elem[n-1].name);
275                         ListTraverse(lists.elem[n-1].L);
276         putchar('\n');
277                 }
278                 else if(lists.length==0) printf("无线性表\n");
279                 else printf("没有该线性表");
280                 getchar();getchar();
```

```
281                 break;
282                 case 18:
283                 if(lists.elem[lists.now].L.elem==NULL)
284                 {
285                         printf("线性表为空\n");
286                         break;
287                  }
288                 e=MaxSubArray(lists.elem[lists.now].L.elem,lists.elem[lists.
                        now].L.length);
289                 printf("最大连续子数组和: %d\n",e);
290                 e=0;
291                 getchar();getchar();
292                 break;
293                 case 19:
294                 printf("请输入和: ");
295                 scanf("%d",&k);
296                 e=SubArrayNum(lists.elem[lists.now].L.elem,lists.elem[lists.
                        now].L.length,k);
297                 printf("和为K的子数组有%d个",e);
298                 k=0;
299                 e=0;
300                 getchar();getchar();
301                 break;
302         case 20:
303                 if(lists.elem[lists.now].L.elem==NULL)
304                 {
305                         printf("线性表为空\n");
306                         break;
307                  }
308                 sortList(lists.elem[lists.now].L.elem,lists.elem[lists.now].
                        L.length);
309                 printf("排序后如下: \n");
310                 ListTraverse(lists.elem[lists.now].L);
311                 getchar();getchar();
312                 break;
313                 case 21:
314                 Make_Person(lists.elem[lists.now].L);
315                 getchar();getchar();
316                 break;
317                 case 22:
318                 for(int i=0;i<lists.length;i++)
```

```
319                     {
320                             printf("%d.%s\n",i+1,lists.elem[i].name);
321                     }
322                     printf("请输入你要切换的线性表名称: ");
323                     scanf("%s",name);
324                     if(n=LocateList(lists,name))
325                     {
326                             lists.now=n-1;
327                             printf("切换成功");
328             putchar('\n');
329                     }
330                     else printf("切换失败,无该线性表\n");
331                     getchar();getchar();
332                     break;
333                     case 23:
334                     Allput(lists);
335                     printf("已输出\n");
336                     getchar();getchar();
337                     break;
338                     case 0:
339             break;
340           }//end of switch
341   }//end of while
342 printf("欢迎下次再使用本系统! \n");
343 }//end of main()
344 /*--------page 23 on textbook --------------------*/
345 status InitList(SqList &L)
346 {
347     /********** Begin *********/
348     if(L.elem!=NULL)
349     {
350         return INFEASIBLE;
351     }
352     L.elem=(ElemType*)malloc(LIST_INIT_SIZE*sizeof(ElemType));
353     L.length=0;
354     L.listsize=LIST_INIT_SIZE;
355     if(L.elem!=NULL)
356     {
357         return OK;
358     }
359     else return INFEASIBLE;
```

```
360     /********** End **********/
361 }
362 status AddList(LISTS &Lists,char ListName[])
363 {
364     /********** Begin *********/
365     if (Lists.length >= 20) {
366         return ERROR;
367     }
368     Lists.elem[Lists.length].L.elem=NULL;
369     InitList(Lists.elem[Lists.length].L);
370     Lists.elem[Lists.length].L.elem = (ElemType*)malloc(LIST_INIT_SIZE *
            sizeof(ElemType));
371     for(int k=0;k<30||ListName[k+1]!='\0';k++)
372     {
373         Lists.elem[Lists.length].name[k]=ListName[k];
374     }
375     if (!Lists.elem[Lists.length].L.elem) {
376         return ERROR;
377     }
378     Lists.elem[Lists.length].L.length = 0;
379     Lists.elem[Lists.length].L.listsize = LIST_INIT_SIZE;
380     Lists.length++;
381     return OK;
382     /********** End **********/
383 }
384 status SaveList(SqList L, char FileName[])
385 {
386     if (L.elem == NULL) {
387         return INFEASIBLE;
388     }
389     FILE *fp;
390     fp = fopen(FileName, "w");
391     if (fp == NULL) {
392         return ERROR;
393     }
394     for (int j = 0; j < L.length; j++) {
395         fprintf(fp, "%d ", L.elem[j]);
396     }
397
398     fclose(fp);
399     return OK;
```

```
400 }
401 status LoadList(SqList &L, char FileName[])
402 {
403     if (L.elem != NULL) {
404         return INFEASIBLE;
405     }
406 
407     FILE *fp;
408     fp = fopen(FileName, "r");
409     if (fp == NULL) {
410         return ERROR;
411     }
412     L.elem=(ElemType*)malloc(LIST_INIT_SIZE*sizeof(ElemType));
413     L.listsize=LIST_INIT_SIZE;
414     L.length=0;
415     while (fscanf(fp, "%d", &L.elem[L.length]) == 1) {
416         L.length++;
417     }
418 
419     fclose(fp);
420     return OK;
421 }
422 status ListTraverse(SqList L)
423 {
424     /********** Begin *********/
425     if(L.elem==NULL)
426     {
427         return INFEASIBLE;
428     }
429     else{
430         int a=L.length;
431         for(int i=0;i<a;i++)
432         {
433             printf("%d",L.elem[i]);
434             if(i!=L.length-1) printf(" ");
435         }
436     }
437     return OK;
438     /********** End **********/
439 }
440 status ListDelete(SqList &L, int i, ElemType &e)
```

```
441 {
442     if (L.elem == NULL) {
443         return INFEASIBLE;
444     } else if (i < 1 || i > L.length) {
445         return ERROR;
446     } else {
447         e = L.elem[i - 1];
448         for (int j = i - 1; j < L.length - 1; j++) {
449             L.elem[j] = L.elem[j + 1];
450         }
451 
452         L.length--;
453         return OK;
454     }
455 }
456 status ListInsert(SqList &L, int i, ElemType e)
457 {
458     if (L.elem == NULL) {
459         return INFEASIBLE;
460     } else if (i < 1 || i > L.length + 1) {
461         return ERROR;
462     } else {
463         if (L.listsize - L.length < 1) {
464             ElemType *newbase = NULL;
465             newbase = (ElemType*)realloc(L.elem, (L.listsize + LISTINCREMENT
                ) * sizeof(ElemType));
466             if (!newbase) {
467                 return ERROR;
468             }
469             L.elem = newbase;
470             L.listsize += LISTINCREMENT;
471         }
472         for (int j = L.length - 1; j >= i - 1; j--) {
473             L.elem[j + 1] = L.elem[j];
474         }
475         L.elem[i - 1] = e;
476         L.length++;
477         return OK;
478     }
479 }
480 status NextElem(SqList L,ElemType e,ElemType &next)
```

```
{
    /********** Begin *********/
    if(L.elem==NULL)
    {
        return INFEASIBLE;
    }
    for(int i=0;i<L.length;i++)
    {
        if(L.elem[i]==e)
        {
            if(L.length-i>1)
            {
                next=L.elem[i+1];
                return OK;
            }
            else return ERROR;
        }
    }
    return ERROR;
    /********** End **********/
}
status PriorElem(SqList L,ElemType e,ElemType &pre)
{
    /********** Begin *********/
    if(L.elem==NULL)
    {
        return INFEASIBLE;
    }
    else{
        for(int i=0;i<L.length;i++)
        {
            if(L.elem[i]==e)
            {
                if(i>=1)
                {
                    pre=L.elem[i-1];
                    return OK;
                }
                else return ERROR;
            }
        }
```

```
522             return ERROR;
523         }
524         /********** End **********/
525     }
526     int LocateElem(SqList L,ElemType e)
527     {
528         /********** Begin *********/
529         if(L.elem==NULL)
530         {
531             return INFEASIBLE;
532         }
533         else{
534             for(int i=0;i<L.length;i++)
535             {
536                 if(L.elem[i]==e)
537                 {
538                     return i+1;
539                 }
540             }
541             return 0;
542         }
543         /********** End **********/
544     }
545     status GetElem(SqList L,int i,ElemType &e)
546     {
547
548         /********** Begin *********/
549         if(L.elem==NULL)
550         {
551             return INFEASIBLE;
552         }
553         else if(i>L.length||i<=0)
554         {
555             return ERROR;
556         }
557         else
558         {
559             e=L.elem[i-1];
560             return OK;
561         }
562
```

```
563     /********** End **********/
564 }
565 status ListLength(SqList L)
566 {
567     /********** Begin *********/
568     if(L.elem==NULL)
569     {
570         return INFEASIBLE;
571     }
572     else return L.length;
573 
574     /********** End **********/
575 }
576 status ListEmpty(SqList L)
577 {
578     /********** Begin *********/
579     if(L.elem==NULL)
580     {
581         return INFEASIBLE;
582     }
583     else
584     {
585         if(L.length==0)
586         {
587             return TRUE;
588         }
589         else return FALSE;
590     }
591     /********** End **********/
592 }
593 status ClearList(SqList& L)
594 {
595         /********** Begin *********/
596     if(L.elem==NULL)
597     {
598         return INFEASIBLE;
599     }
600     L.length=0;
601     return OK;
602 
603     /********** End **********/
```

```
604 }
605 status DestroyList(SqList &L)
606 {
607     /********** Begin *********/
608     if(L.elem==NULL)
609     {
610         return INFEASIBLE;
611     }
612     free(L.elem);
613     L.elem=NULL;
614     L.length=0;
615     L.listsize=0;
616     return OK;
617     /********** End **********/
618 }
619 int LocateList(LISTS Lists,char ListName[])
620 {
621     /********** Begin *********/
622     int k=0;
623     for(;k<Lists.length;k++)
624     {
625         if(strcmp(Lists.elem[k].name,ListName)==0)
626         {
627             return k+1;
628         }
629     }
630     return INFEASIBLE;
631
632     /********** End **********/
633 }
634 int MaxSubArray(int *L,int n)
635 {
636     if(L==NULL)
637     {
638         return INFEASIBLE;
639     }
640     int max_sum=L[0];
641     int curr_sum=L[0];
642     for (int i = 1; i < n; i++)
643     {
644         curr_sum=(curr_sum + L[i] > L[i])?curr_sum+L[i]:L[i];
```

```
645             max_sum=(max_sum > curr_sum)?max_sum:curr_sum;
646         }
647         return max_sum;
648     }
649     int SubArrayNum(int *L,int n,int k)
650     {
651         if(L==NULL)
652         {
653             return INFEASIBLE;
654         }
655         int count=0;
656         for (int i=0;i<n;i++)
657         {
658             int sum=0;
659             for (int j=i;j<n;j++)
660             {
661                 sum+=L[j];
662                 if (sum==k)
663                 {
664                     count++;
665                 }
666             }
667         }
668         return count;
669     }
670     void sortList(int *L,int n)
671     {
672         if(L==NULL)
673         {
674             return ;
675         }
676         for(int i=0;i<n-1;i++)
677         {
678             int flag = 0;
679             for(int j=0;j<n-i-1;j++)
680             {
681                 if (L[j]>L[j + 1])
682                 {
683                     int temp=L[j];
684                     L[j]=L[j+1];
685                     L[j+1]=temp;
```

```
686                     flag=1;
687                 }
688             }
689             if (!flag)
690                 break;
691         }
692
693 }
694 status RemoveList(LISTS &Lists,char ListName[])
695 {
696     /********** Begin *********/
697     int k=0,j=0,m=0;
698     for(;k<10;k++)
699     {
700         if(strcmp(Lists.elem[k].name,ListName)==0)
701         {
702             j=DestroyList(Lists.elem[k].L);
703             break;
704         }
705     }
706     if(j==OK)
707     {
708         for(int m=k;j<Lists.length;j++)
709         {
710             Lists.elem[k]=Lists.elem[k+1];
711         }
712         Lists.elem[Lists.length-1].name==NULL;
713         Lists.length--;
714         Lists.now=Lists.length-1;
715         return OK;
716     }
717     else return ERROR;
718     /********** End **********/
719 }
720 status Make_Person(SqList &L)
721 {
722     if(L.elem==NULL) return INFEASIBLE;
723     else{
724         printf("请输入需要的元素: ");
725         int i=0;
726         ElemType e;
```

```
727             while(scanf("%d",&e))
728             {
729                 L.elem[L.length++]=e;
730                 if(getchar()=='\n') break;
731             }
732             return OK;
733         }
734 }
735 void Allput(LISTS &lists)
736 {
737 for(int n=0;n<lists.length;n++)
738             {
739                     printf("%s ",lists.elem[n].name);
740                     ListTraverse(lists.elem[n].L);
741             putchar('\n');
742             }
743 }
```

5 附录 B 基于邻接表图实现的源程序

```
1  /* Linear Table On Sequence Structure */
2  #include <stdio.h>
3  #include <malloc.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <stdbool.h>
7
8  /*---------page 10 on textbook ---------*/
9  #define TRUE 1
10 #define FALSE 0
11 #define OK 1
12 #define ERROR 0
13 #define INFEASIBLE -1
14 #define OVERFLOW -2
15 #define MAX_VERTEX_NUM 20
16
17 typedef int status;
18 typedef int KeyType;
19
20 typedef struct {
21     KeyType elements[MAX_VERTEX_NUM];
22     int front;
23     int rear;
24 } Queue;
25 typedef enum {DG,DN,UDG,UDN} GraphKind;
26 typedef struct {
27     KeyType  key;
28     char others[300];
29 } VertexType; //顶点类型定义
30 typedef struct ArcNode{          //表结点类型定义
31         int adjvex;               //顶点位置编号
32     struct ArcNode *nextarc;        //下一个表结点指针
33 } ArcNode;
34 typedef struct {                                  //头结点及其数组类型定义
35         VertexType data;          //顶点信息
36         ArcNode *firstarc;                  //指向第一条弧
37 } VNode, AdjList[MAX_VERTEX_NUM];
38 typedef  struct {  //邻接表的类型定义
```

```
39      AdjList vertices;                //头结点数组
40      int vexnum,arcnum;               //顶点数、弧数
41      GraphKind  kind;          //图的类型
42  } ALGraph;
43  typedef struct {
44          struct{
45          char name[30];            //图的名字
46                  ALGraph G;            //图
47      }elem[MAX_VERTEX_NUM];
48          int length;
49      int now;                 //图的数量
50  }GQueen;
51
52  VertexType V[30];
53  KeyType VR[100][2];
54  VertexType value;
55  int visited[MAX_VERTEX_NUM];
56  int dist[MAX_VERTEX_NUM];
57
58  void InitQueue(Queue *Q);
59  int QueueEmpty(Queue *Q);
60  int EnQueue(Queue *Q, KeyType item);
61  int DeQueue(Queue *Q, KeyType *item);
62  status CreateCraph(ALGraph &G,VertexType V[],KeyType VR[][2]);
63  status DestroyGraph(ALGraph &G);
64  int LocateVex(ALGraph G,KeyType u);
65  status PutVex(ALGraph &G,KeyType u,VertexType value);
66  int FirstAdjVex(ALGraph G,KeyType u);
67  int NextAdjVex(ALGraph G,KeyType v,KeyType w);
68  status InsertVex(ALGraph &G,VertexType v);
69  status DeleteVex(ALGraph &G,KeyType v);
70  status InsertArc(ALGraph &G,KeyType v,KeyType w);
71  status DeleteArc(ALGraph &G,KeyType v,KeyType w);
72  status DFSTraverse(ALGraph &G,void (*visit)(VertexType));
73  status BFSTraverse(ALGraph &G,void (*visit)(VertexType));
74  void WArc(int VR[][2], ArcNode *p, int *cnt, int i);
75  status SaveGraph(ALGraph G, char FileName[]);
76  status LoadGraph(ALGraph &G, char FileName[]);
77  void BFS(ALGraph &G, int v, int k, int visited[], int &dist);
78  status VerticesSetLessThanK(ALGraph &G, int v, int k, int visited[], int
        dist[]);
```

```
79  void dfs(ALGraph &G, int v, int w, int visited[], int path[], int &pathLen);
80  int ShortestPathLength(ALGraph &G, int v, int w);
81  void DFS_simple(ALGraph &G, int v, int visited[]);
82  int ConnectedComponentsNums(ALGraph &G);
83  void ShowAll(ALGraph G);
84  void visit(VertexType v);
85
86  //主函数
87  int main()
88  {
89      char name[30];
90      char FileName[30];
91      int op=1;
92      GQueen lists;
93      lists.length=0;
94      while(op){
95              system("cls");       printf("\n\n");
96              printf("      Menu for Linear Table On Sequence Structure \n");
97              printf("-------------------------------------------------\n");
98              printf("              1.CreateCraph图创建        \t11.DFSTraverse
                    深度优先搜索遍历\n");
99              printf("              2.DestroyGraph销毁图       \t12.BFSTraverse
                    广度优先搜索遍历 \n");
100             printf("              3.LocateVex查找节点        \t13.
                    VerticesSetLessThanK距离小于k的顶点集合  \n");
101             printf("              4.PutVex结点赋值       \t14.
                    ShortestPathLength顶点间最短路径和长度\n");
102             printf("              5.FirstAdjVex获得第一邻接点\t15.
                    ConnectedComponentsNums图的连通分量\n");
103             printf("              6.NextAdjVex获得下一邻接点\t16.AddGraph添
                    加图\n");
104             printf("              7.InsertVex插入顶点        \t17.LocateGrape
                    查找图\n");
105             printf("              8.DeleteVex删除顶点        \t18.SaveGrape图
                    文件保存\n");
106             printf("              9.InsertArc插入弧      \t19.LoadGrape图文件
                    读取\n");
107             printf("              10.DeleteArc删除弧         \t20.AllShow图输
                    出\n");
108             printf("              21.ChooseGrape图选择       \t0.Exit\n");
109             printf("现可用图数: %d\n",lists.length);
```

```
110             printf("-----------------------------------------------\n");
111             printf("    请选择你的操作[0~24]:");
112             scanf("%d",&op);
113             KeyType e=0;
114             KeyType n=0;
115             KeyType k=0;
116             KeyType j=0;
117         switch(op){
118                 case 1:
119                     e=0;
120                 if(lists.elem[lists.now].G.vexnum!=0)
121                 {
122                     printf("图已存在! \n");
123                     break;
124                 }
125                 printf("请输入图的节点值: \n");
126                     do
127                         {
128                                 scanf("%d %s",&V[e].key,V[e].others);
129                                 e++;
130                         } while (getchar()!='\n');
131                 V[e].key=-1;
132                 V[e].others[0]='a';
133                         e=0;
134                         printf("请输入图的边值: \n");
135                         do
136                         {
137                             scanf("%d %d",&VR[e][0],&VR[e][1]);
138                                 e++;
139                         } while (getchar()!='\n');
140                 VR[e][0]=-1;
141                 VR[e][1]=-1;
142                 if(CreateCraph(lists.elem[lists.now].G,V,VR)==OK)
143                         {
144                                 printf("请输入图的名字: \n");
145                                 scanf("%s",lists.elem[lists.now].name);
146                                 printf("图创建成功! \n");
147                 }
148                 else printf("创建失败! \n");
149                         getchar();getchar();
150                         break;
```

```
151             case 2:
152                 if(lists.length==0)
153                 {
154                     printf("图不存在! \n");
155                 }
156                 else
157                 {
158                     if(DestroyGraph(lists.elem[lists.now].G)==OK)
159                     {
160                         printf("图已销毁! \n");
161                         lists.length--;
162                         lists.now--;
163                     }
164                 }
165                 getchar();getchar();
166                 break;
167             case 3:
168                 if(lists.length==0)
169                 {
170                     printf("图不存在! \n");
171                 }
172                 else
173                 {
174                     printf("请输入你查找节点的关键字: \n");
175                     scanf("%d",&n);
176                     e=-1;
177                     e=LocateVex(lists.elem[lists.now].G,n);
178                     if(e==-1) printf("顶点不存在\n");
179                     else printf("查找成功! 关键字为%d的节点是第%d个节点!\n节
                        点是%d %s\n",n,e+1,n,lists.elem[lists.now].G.vertices
                        [e].data.others);
180                 }
181                 getchar();getchar();
182                 break;
183             case 4:
184                 if(lists.length==0)
185                 {
186                     printf("图不存在! \n");
187                 }
188                 else
189                 {
```

```
190                     printf("请输入你想要修改的节点关键字: \n");
191                     scanf("%d",&n);
192                     printf("请输入修改后的值:(key other)\n");
193                     scanf("%d %s",&value.key,value.others);
194                     if(PutVex(lists.elem[lists.now].G,n,value)==OK) printf("
                            修改成功! \n");
195                     else printf("节点不存在或关键字重复\n");
196                 }
197                 getchar();getchar();
198                 break;
199             case 5:
200                 if(lists.length==0)
201                 {
202                     printf("图不存在! \n");
203                 }
204                 else
205                 {
206                     printf("请输入你想要查找的节点关键字: \n");
207                     scanf("%d",&n);
208                     e=-1;
209                     e=LocateVex(lists.elem[lists.now].G,n);
210                     if(e==-1) printf("该节点不存在! \n");
211                     else{
212                         j=-1;
213                         j=FirstAdjVex(lists.elem[lists.now].G,n);
214                         if(j==-1) printf("该节点没有邻接节点! \n");
215                         else
216                         {
217                             printf("该节点的邻接节点是%d %s!\n",lists.elem[
                                    lists.now].G.vertices[j].data.key,lists.elem[
                                    lists.now].G.vertices[j].data.others);
218                         }
219                     }
220                 }
221                 getchar();getchar();
222                 break;
223             case 6:
224                 if(lists.length==0)
225                 {
226                     printf("图不存在! \n");
227                 }
```

```
228                 else
229                 {
230                     printf("请输入你想要查找的节点与相邻点关键字: \n");
231                     scanf("%d %d",&n,&e);
232                     j=-1;
233                     j=NextAdjVex(lists.elem[lists.now].G,n,e);
234                     if(e==-1)
235                     {
236                         printf("无后继\n");
237                     }
238                     else
239                     {
240                         printf("结果是%d %s",lists.elem[lists.now].G.
                                vertices[j].data.key,lists.elem[lists.now].G.
                                vertices[j].data.others);
241                     }
242                 }
243                 getchar();getchar();
244                 break;
245             case 7:
246                 printf("请输入你想要添加的节点数据: \n");
247                 scanf("%d %s",&value.key,value.others);
248                 if(InsertVex(lists.elem[lists.now].G,value)==OK)
249                 {
250                     printf("插入成功! \n");
251                 }
252                 else
253                 {
254                     printf("关键字重复\n");
255                 }
256                 getchar();getchar();
257                 break;
258             case 8:
259                 printf("请输入你想删除的节点关键值: \n");
260                 scanf("%d",&e);
261                 if(DeleteVex(lists.elem[lists.now].G,e)==OK) printf("删除成
                    功! \n");
262                 else printf("顶点不存在\n");
263                 getchar();getchar();
264                 break;
265             case 9:
```

```
266                 printf("请输入你想要插入的弧的端点关键值: \n");
267                 scanf("%d %d",&e,&n);
268                 if(InsertArc(lists.elem[lists.now].G,e,n)==OK) printf("插入
                        成功! \n");
269                 else printf("弧已存在或顶点不存在\n");
270                 getchar();getchar();
271                 break;
272             case 10:
273                 printf("请输入你想要删除的弧的端点关键值: \n");
274                 scanf("%d %d",&e,&n);
275                 if(DeleteArc(lists.elem[lists.now].G,e,n)==OK) printf("删除
                        成功! \n");
276                 else printf("弧不存在或顶点不存在\n");
277                 getchar();getchar();
278                 break;
279             case 11:
280                 if(DFSTraverse(lists.elem[lists.now].G,visit)==OK) printf("
                        已遍历完毕! \n");
281                 getchar();getchar();
282                 break;
283             case 12:
284                 if(BFSTraverse(lists.elem[lists.now].G,visit)==OK) printf("
                        已遍历完毕! \n");
285                 getchar();getchar();
286                 break;
287             case 13:
288                 printf("请输入k的值: \n");
289                 scanf("%d",&e);
290                 printf("请输入你想要作为基准的点的关键值: \n");
291                 scanf("%d",&j);
292                 n=LocateVex(lists.elem[lists.now].G,j);
293                 if(n!=-1)
294                 {
295                     if(VerticesSetLessThanK(lists.elem[lists.now].G,n,e,
                            visited,dist)==OK) printf("已输出完毕! \n");
296                 }
297                 getchar();getchar();
298                 break;
299             case 14:
300                 if(lists.length==0) printf("图不存在\n");
301                 else
```

```
302                 {
303                     printf("请输入两个点的关键值: \n");
304                     scanf("%d %d",&e,&n);
305                     k=LocateVex(lists.elem[lists.now].G,e);
306                     j=LocateVex(lists.elem[lists.now].G,n);
307                     if(k==-1||j==-1) printf("点不存在! \n");
308                     else
309                                         {
310                                             e=ShortestPathLength(lists.
                                                elem[lists.now].G,k,j);
311                                             printf("最短路径长度是: %d\n
                                                ",e);
312                                         }
313                 }
314                 getchar();getchar();
315                 break;
316             case 15:
317                 if(lists.length==0) printf("图不存在! \n");
318                 else
319                 {
320                     e=ConnectedComponentsNums(lists.elem[lists.now].G);
321                     printf("图G的联通分量是:%d",e);
322                 }
323                 getchar();getchar();
324                 break;
325             case 16:
326                 if(lists.length<20)
327                 {
328                     lists.now=lists.length;
329                     lists.length++;
330                 }
331                 else printf("超出内存! \n");
332                 getchar();getchar();
333                 break;
334             case 17:
335                 printf("请输入你想要查找的图的名字: \n");
336                 scanf("%s",name);
337                 e=0;
338                 for(int i=0;i<lists.length;i++)
339                 {
340                     if(strcmp(name,lists.elem->name)==0)
```

```
341                     {
342                         e=1;
343                         printf("要找的图是%d号图\n",i+1);
344                         break;
345                     }
346                 }
347                 if(e){
348                         printf("图不存在\n");
349                                 }
350                 getchar();getchar();
351                 break;
352             case 18:
353                 if(lists.length==0)
354                 {
355                     printf("图不存在! \n");
356                     break;
357                 }
358                 printf("请输入你要保存的文件名: \n");
359                 scanf("%s",FileName);
360                 if(SaveGraph(lists.elem[lists.now].G,FileName)==OK) printf("
                        已保存\n");
361                 else printf("保存失败! \n");
362                 getchar();getchar();
363                 break;
364             case 19:
365                 printf("请输入你要读取的文件名: \n");
366                 scanf("%s",FileName);
367                 if(LoadGraph(lists.elem[lists.now].G,FileName)==OK) printf("
                        已读取! \n");
368                 else printf("读取失败! \n");
369                 getchar();getchar();
370                 break;
371             case 20:
372                 ShowAll(lists.elem[lists.now].G);
373                 getchar();getchar();
374                 break;
375             case 21:
376                 printf("请选择你要切换的图: \n");
377                 for(int i=0;i<lists.length;i++)
378                 {
379                     printf("%d.%s",i+1,lists.elem[i].name);
```

```
380                     }
381                     scanf("%d",&e);
382                     lists.now=e-1;
383                     getchar();getchar();
384                     break;
385                 case 0:
386                     break;
387                 }
388     }
389     printf("欢迎下次使用本系统! \n");
390 }
391 //-----------------------------------------------------------
392 status CreateCraph(ALGraph &G,VertexType V[],KeyType VR[][2])
393 /*根据V和VR构造图T并返回OK，如果V和VR不正确，返回ERROR
394 如果有相同的关键字，返回ERROR。此题允许通过增加其它函数辅助实现本关任务*/
395 {
396     // 请在这里补充代码，完成本关任务
397     /********** Begin *********/
398     int i=0, j=0, k=0,flag=0;
399     G.vexnum = 0;
400     G.arcnum = 0;
401     G.kind = UDG;
402     while(V[i].key!=-1)
403     {
404         flag=1;
405         if(i>=MAX_VERTEX_NUM) return ERROR;
406         G.vertices[i].data=V[i];
407         G.vertices[i].firstarc=NULL;
408         i++;
409     }
410     if(!flag) return ERROR;
411     G.vexnum=i;
412     i=0;
413     ArcNode *p;
414     int fir,sec;
415     while(VR[i][0]!=-1)
416     {
417         G.arcnum++;
418         fir=-1;
419         for(int j=0;j<G.vexnum;j++)
420         {
```

```
421                 if(VR[i][0]==G.vertices[j].data.key)
422                 {
423                     fir=j;
424                     break;
425                 }
426             }
427             if(fir==-1) return ERROR;
428             sec=-1;
429             for(int j=0;j<G.vexnum;j++)
430             {
431                 if(VR[i][1]==G.vertices[j].data.key)
432                 {
433                     sec=j;
434                     break;
435                 }
436             }
437             if(sec==-1) return ERROR;
438             p=(ArcNode*)malloc(sizeof(ArcNode));
439             p->adjvex=sec;
440             p->nextarc=G.vertices[fir].firstarc;
441             G.vertices[fir].firstarc=p;
442             p=(ArcNode*)malloc(sizeof(ArcNode));
443             p->adjvex=fir;
444             p->nextarc=G.vertices[sec].firstarc;
445             G.vertices[sec].firstarc=p;
446             i++;
447         }
448         return OK;
449     }
450     status DestroyGraph(ALGraph &G)
451     /*销毁无向图G,删除G的全部顶点和边*/
452     {
453         // 请在这里补充代码，完成本关任务
454         /********** Begin *********/
455         for (int i = 0; i < G.vexnum; i++) {
456             ArcNode *p = G.vertices[i].firstarc;
457             while (p != NULL) {
458                 ArcNode *q = p;
459                 p = p->nextarc;
460                 free(q);
461             }
```

```
462             G.vertices[i].firstarc = NULL;
463         }
464         G.vexnum = 0;
465         G.arcnum = 0;
466         return OK;
467 
468         /********** End **********/
469     }
470     int LocateVex(ALGraph G,KeyType u)
471     //根据u在图G中查找顶点，查找成功返回位序，否则返回-1；
472     {
473         // 请在这里补充代码，完成本关任务
474         /********** Begin *********/
475         for (int i = 0; i < G.vexnum; i++) {
476             if (G.vertices[i].data.key == u) {
477                 return i;
478             }
479         }
480         return -1;
481 
482         /********** End **********/
483     }
484     status PutVex(ALGraph &G,KeyType u,VertexType value)
485     //根据u在图G中查找顶点，查找成功将该顶点值修改成value，返回OK;
486     //如果查找失败或关键字不唯一，返回ERROR
487     {
488         // 请在这里补充代码，完成本关任务
489         /********** Begin *********/
490         for (int i = 0; i < G.vexnum; i++) {
491             if (G.vertices[i].data.key == u) {
492                 G.vertices[i].data = value;
493                 for (int j = 0; j < G.vexnum; j++) {
494                     if (j != i && G.vertices[j].data.key == value.key) {
495                         return ERROR;
496                     }
497                 }
498                 return OK;
499             }
500         }
501         return ERROR;
502 
```

```
503     /********** End **********/
504 }
505 int FirstAdjVex(ALGraph G,KeyType u)
506 //根据u在图G中查找顶点，查找成功返回顶点u的第一邻接顶点位序，否则返回-1；
507 {
508     // 请在这里补充代码，完成本关任务
509     /********** Begin *********/
510     for (int i = 0; i < G.vexnum; i++) {
511         if (G.vertices[i].data.key == u) {
512             if (G.vertices[i].firstarc != NULL) {
513                 return G.vertices[i].firstarc->adjvex;
514             } else {
515                 return -1;
516             }
517         }
518     }
519     return -1;
520 
521     /********** End **********/
522 }
523 int NextAdjVex(ALGraph G,KeyType v,KeyType w)
524 //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，或v、w对应顶点不存在，则返回-1。
525 {
526     // 请在这里补充代码，完成本关任务
527     /********** Begin *********/
528     int v1;
529     for(int i=0;i<G.vexnum;i++)
530     {
531         if(G.vertices[i].data.key == w)
532         v1=i;
533     }
534     for (int i = 0; i < G.vexnum; i++)
535     {
536         if (G.vertices[i].data.key == v)
537         {
538             ArcNode *p = G.vertices[i].firstarc;
539             while (p != NULL)
540             {
541                 if (p->adjvex == v1)
542                 {
```

```
543                         if (p->nextarc != NULL)
544                         {
545                             return p->nextarc->adjvex;
546                         }
547                         else return -1;
548                     }
549                     p = p->nextarc;
550                 }
551                 return -1;
552             }
553         }
554         return -1;
555 
556         /********** End **********/
557     }
558     status InsertVex(ALGraph &G, VertexType v)
559     //在图G中插入顶点v，成功返回OK,否则返回ERROR
560     {
561         // 请在这里补充代码，完成本关任务
562         /********** Begin *********/
563         for (int i = 0; i < G.vexnum; i++) {
564             if (G.vertices[i].data.key == v.key) {
565                 return ERROR;
566             }
567         }
568         if (G.vexnum < MAX_VERTEX_NUM) {
569             G.vertices[G.vexnum].data = v;
570             G.vexnum++;
571             return OK;
572         } else {
573             return ERROR;
574         }
575 
576         /********** End **********/
577     }
578     status DeleteVex(ALGraph &G, KeyType v)
579     //在图G中删除关键字v对应的顶点以及相关的弧，成功返回OK,否则返回ERROR
580     {
581         // 请在这里补充代码，完成本关任务
582         /********** Begin *********/
```

```
583     if(G.vertices[0].data.key != -1 && G.vertices[1].data.key == -1) return
            ERROR;//不能删成空表
584
585     int  x=-1;
586         for(int i=0;i<G.vexnum;i++)
587     {
588         if(G.vertices[i].data.key==v)
589         {
590             x=i;
591             break;
592         }
593     }
594         if(x==-1) return ERROR;//没找到
595
596     ArcNode *p,*q;
597         p=G.vertices[x].firstarc;
598         while(p){
599                 q=p;
600                 p=p->nextarc;
601                 free(q);
602                 G.arcnum--;
603         }
604         for(int i=x;i<G.vexnum;i++)
605                 G.vertices[i]=G.vertices[i+1];
606     G.vexnum--;
607         for(int i=0;i<G.vexnum;i++){
608                 p=G.vertices[i].firstarc;
609                 while(p) {
610                         if(p->adjvex==x){
611                                 if(p==G.vertices[i].firstarc){
612                                         G.vertices[i].firstarc=p->nextarc;
613                                         free(p);
614                                         p = G.vertices[i].firstarc;
615                                 }else{
616                                         q->nextarc=p->nextarc;
617                                         free(p);
618                                         p=q->nextarc;
619                                 }
620                         }else{
621                                 if(p->adjvex>x)
622                                         p->adjvex--;
```

```
623                                     q=p;
624                                     p=p->nextarc;
625                                 }
626                             }
627                         }
628                         return OK;
629                     /********** End **********/
630 }
631 status InsertArc(ALGraph &G,KeyType v,KeyType w)
632 //在图G中增加弧<v,w>，成功返回OK,否则返回ERROR
633 {
634     // 请在这里补充代码，完成本关任务
635     /********** Begin *********/
636     int vIndex = -1, wIndex = -1;
637     for (int i = 0; i < G.vexnum; ++i) {
638         if (G.vertices[i].data.key == v) {
639             vIndex = i;
640         }
641         if (G.vertices[i].data.key == w) {
642             wIndex = i;
643         }
644     }
645     if (vIndex == -1 || wIndex == -1) {
646         return ERROR;
647     }
648     ArcNode *p = G.vertices[vIndex].firstarc;
649     while (p) {
650         if (p->adjvex == wIndex) {
651             return ERROR;
652         }
653         p = p->nextarc;
654     }
655     ArcNode *newArcV = (ArcNode*)malloc(sizeof(ArcNode));
656     if (!newArcV) {
657         exit(OVERFLOW);
658     }
659     newArcV->adjvex = wIndex;
660     newArcV->nextarc = G.vertices[vIndex].firstarc;
661     G.vertices[vIndex].firstarc = newArcV;
662     p = G.vertices[wIndex].firstarc;
663     while (p) {
```

```
664             if (p->adjvex == vIndex) {
665                 return ERROR;
666             }
667             p = p->nextarc;
668         }
669         ArcNode *newArcW = (ArcNode*)malloc(sizeof(ArcNode));
670         if (!newArcW) {
671             exit(OVERFLOW);
672         }
673         newArcW->adjvex = vIndex;
674         newArcW->nextarc = G.vertices[wIndex].firstarc;
675         G.vertices[wIndex].firstarc = newArcW;
676         G.arcnum++;
677 
678         return OK;
679         /********** End **********/
680 }
681 status DeleteArc(ALGraph &G,KeyType v,KeyType w)
682 //在图G中删除弧<v,w>，成功返回OK,否则返回ERROR
683 {
684         // 请在这里补充代码，完成本关任务
685         /********** Begin *********/
686         int vIndex = -1, wIndex = -1;
687         for (int i = 0; i < G.vexnum; ++i) {
688             if (G.vertices[i].data.key == v) {
689                 vIndex = i;
690             }
691             if (G.vertices[i].data.key == w) {
692                 wIndex = i;
693             }
694         }
695         if (vIndex == -1 || wIndex == -1) {
696             return ERROR;
697         }
698         int flag=0;
699         ArcNode *p = G.vertices[vIndex].firstarc;
700         ArcNode *prev = NULL;
701         while (p) {
702             if (p->adjvex == wIndex) {
703                 flag=1;
704                 if (prev) {
```

```
705                 prev->nextarc = p->nextarc;
706             } else {
707                 G.vertices[vIndex].firstarc = p->nextarc;
708             }
709             free(p);
710             break;
711         }
712         prev = p;
713         p = p->nextarc;
714     }
715     if(!flag) return ERROR;
716     p = G.vertices[wIndex].firstarc;
717     prev = NULL;
718     while (p) {
719         if (p->adjvex == vIndex) {
720             if (prev) {
721                 prev->nextarc = p->nextarc;
722             } else {
723                 G.vertices[wIndex].firstarc = p->nextarc;
724             }
725             free(p);
726             break;
727         }
728         prev = p;
729         p = p->nextarc;
730     }
731     G.arcnum--;
732     return OK;
733
734     /********** End **********/
735 }
736 void DFS(ALGraph &G, int v, int visited[], void (*visit)(VertexType)) {
737     visited[v] = 1;
738     visit(G.vertices[v].data);
739     ArcNode *p = G.vertices[v].firstarc;
740     while (p) {
741         int w = p->adjvex;
742         if (!visited[w]) {
743             DFS(G, w, visited, visit);
744         }
745         p = p->nextarc;
```

```
746     }
747 }
748 status DFSTraverse(ALGraph &G,void (*visit)(VertexType))
749 //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
750 {
751     // 请在这里补充代码，完成本关任务
752     /********** Begin *********/
753     visited[MAX_VERTEX_NUM]={0};
754     for (int i = 0; i < G.vexnum; ++i) {
755         if (!visited[i]) {
756             DFS(G, i, visited, visit);
757         }
758     }
759     return OK;
760     /********** End **********/
761 }
762 void InitQueue(Queue *Q) {
763     Q->front=Q->rear=0;
764 }
765 int QueueEmpty(Queue *Q) {
766     return Q->front==Q->rear;
767 }
768 int EnQueue(Queue *Q, KeyType item) {
769     if ((Q->rear+1)%MAX_VERTEX_NUM==Q->front) {
770         return ERROR;
771     }
772     Q->elements[Q->rear] = item;
773     Q->rear = (Q->rear + 1) % MAX_VERTEX_NUM;
774     return OK;
775 }
776 int DeQueue(Queue *Q, KeyType *item) {
777     if (QueueEmpty(Q)) {
778         return ERROR;
779     }
780     *item = Q->elements[Q->front];
781     Q->front = (Q->front + 1) %MAX_VERTEX_NUM;
782     return OK;
783 }
784 status BFSTraverse(ALGraph &G,void (*visit)(VertexType))
```

```
785 //对图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且
        仅访问一次
786 {
787     // 请在这里补充代码，完成本关任务
788     /********** Begin *********/
789     int visited[MAX_VERTEX_NUM] = {FALSE};
790     Queue Q;
791     InitQueue(&Q);  // 初始化队列
792     for (int i = 0; i < G.vexnum; i++) {
793         if (!visited[i]) {
794             visited[i] = TRUE;
795             visit(G.vertices[i].data);  // 访问顶点
796             EnQueue(&Q, i);  // 顶点入队
797             while (!QueueEmpty(&Q)) {
798                 int v;
799                 DeQueue(&Q, &v);  // 顶点出队
800                 ArcNode *p = G.vertices[v].firstarc;
801                 while (p != NULL) {
802                     int w = p->adjvex;
803                     if (!visited[w]) {
804                         visited[w] = TRUE;
805                         visit(G.vertices[w].data);  // 访问顶点
806                         EnQueue(&Q, w);  // 顶点入队
807                     }
808                     p = p->nextarc;
809                 }
810             }
811         }
812     }
813     return OK;
814     /********** End **********/
815 }
816 void WArc(int VR[][2], ArcNode *p, int &cnt, int i) {
817         if(p==NULL) return;
818         if(p->nextarc) WArc(VR, p->nextarc, cnt, i);
819         for(int j=0;j<cnt;j++)
820     {
821                 if(VR[j][1]==i&&VR[j][0]==p->adjvex)
822                         return;
823     }
824         VR[cnt][0]=i;
```

```
825             VR[cnt++][1]=p->adjvex;
826     }
827     status SaveGraph(ALGraph G, char FileName[])
828     //将图的数据写入到文件FileName中
829     {
830         // 请在这里补充代码，完成本关任务
831         /********** Begin 1 *********/
832         int VR[100][2]= {0};
833         FILE* fp=fopen(FileName,"w");
834         fprintf(fp,"%d %d ",G.vexnum,G.arcnum);
835             int cnt=0;//记录边条数
836             for(int i=0;i<G.vexnum;i++){
837                     fprintf(fp,"%d %s ",G.vertices[i].data.key,G.vertices[i].
                            data.others);
838             }
839             fprintf(fp,"-1 nil \n");
840
841             for(int i=0; i<G.vexnum; i++) {//边信息存入数组
842                     WArc(VR, G.vertices[i].firstarc , cnt, i);
843             }
844             int x,y;
845             for(int i=0;i<cnt;i++) {//写入边信息
846                     x=VR[i][0];
847                     y=VR[i][1];
848                     fprintf(fp,"%d %d ", G.vertices[x].data.key, G.vertices[y].
                            data.key);
849             }
850             fprintf(fp,"-1 -1 ");
851             fclose(fp);
852             return OK;
853
854         /********** End 1 **********/
855     }
856     status LoadGraph(ALGraph &G, char FileName[])
857     //读入文件FileName的图数据，创建图的邻接表
858     {
859         // 请在这里补充代码，完成本关任务
860         /********** Begin 2 *********/
861         FILE *fp = fopen(FileName, "r");
862         if (fp == NULL) {
863             return ERROR;
```

```
    }
    VertexType V[20];
    int VR[100][2]= {0};
    fscanf(fp, "%d %d", &G.vexnum, &G.arcnum);
    for (int i = 0; i <=G.vexnum; i++) {
        G.vertices[i].firstarc = NULL;
        fscanf(fp, "%d %s", &V[i].key, V[i].others);
    }
    int v1, v2;
    int j=0;
    while (fscanf(fp, "%d%d", &v1, &v2)==2) {
        VR[j][0]=v1;
        VR[j][1]=v2;
        j++;
    }
    fclose(fp);
    if(CreateCraph(G,V,VR)==OK){
                if(G.vertices[2].firstarc!=NULL){
                        ArcNode *p, *q;
                        p=G.vertices[2].firstarc;
                        q=p->nextarc;
                        p->nextarc=q->nextarc;
                        q->nextarc=p;
                        G.vertices[2].firstarc=q;
                }
                return OK;
        }
        else return ERROR;
    /********** End 2 **********/
}
void BFS(ALGraph &G, int v, int k, int visited[], int dist[]) {
    Queue Q;
    InitQueue(&Q);
    visited[v] = TRUE;
    dist[v] = 0;
    EnQueue(&Q, v);
    while (!QueueEmpty(&Q)) {
        int u;
        DeQueue(&Q, &u);
        ArcNode *p = G.vertices[u].firstarc;
        while (p) {
```

```
905                 int w = p->adjvex;
906                 if (!visited[w]) {
907                     visited[w] = TRUE;
908                     dist[w] = dist[u] + 1;
909                     EnQueue(&Q, w);
910                 }
911                 p = p->nextarc;
912             }
913         }
914     }
915
916     status VerticesSetLessThanK(ALGraph &G, int v, int k, int visited[], int
        dist[]) {
917         for(int i=0;i<20;i++)
918         {
919             visited[i]=0;
920             dist[i]=0;
921         }
922         BFS(G, v, k, visited, dist);
923         for (int i = 0; i < G.vexnum; i++) {
924             if (visited[i] && dist[i] < k) {
925                 printf("%d %s\n", G.vertices[i].data.key, G.vertices[i].data.
                    others);
926             }
927         }
928         return OK;
929     }
930     void dfs(ALGraph &G, int v, int w, int visited[], int path[], int &pathLen)
        {
931         Queue Q;
932         InitQueue(&Q);
933         visited[v] = TRUE;
934         dist[v] = 0;
935         EnQueue(&Q, v);
936         while (!QueueEmpty(&Q)) {
937             int u;
938             DeQueue(&Q, &u);
939             ArcNode *p = G.vertices[u].firstarc;
940             while (p) {
941                 int m = p->adjvex;
942                 if (!visited[m]) {
```

```
943                     visited[m] = TRUE;
944                     dist[m] = dist[u] + 1;
945                     EnQueue(&Q, m);
946                 }
947                 p = p->nextarc;
948             }
949         }
950         pathLen=dist[w];
951     }
952
953     int ShortestPathLength(ALGraph &G, int v, int w) {
954         int visited[MAX_VERTEX_NUM] = {false};
955         int path[MAX_VERTEX_NUM];
956         int pathLen = 0;
957         dfs(G, v, w, visited, path, pathLen);
958         return pathLen;
959     }
960     void DFS_simple(ALGraph &G, int v, int visited[]) {
961         visited[v] = TRUE;
962         ArcNode *p = G.vertices[v].firstarc;
963         while (p) {
964             int w = p->adjvex;
965             if (!visited[w]) {
966                 DFS_simple(G, w, visited);
967             }
968             p = p->nextarc;
969         }
970     }
971     int ConnectedComponentsNums(ALGraph &G) {
972         int visited[MAX_VERTEX_NUM] = {FALSE};
973         int count = 0;
974         for (int v = 0; v < G.vexnum; v++) {
975             if (!visited[v]) {
976                 DFS_simple(G, v, visited);
977                 count++;
978             }
979         }
980         return count;
981     }
982     void ShowAll(ALGraph G)
983     {
```

```
984     for(int j=0;j<G.vexnum;j++)
985         {
986         ArcNode *p=G.vertices[j].firstarc;
987         printf("%d %s",G.vertices[j].data.key,G.vertices[j].data.others);
988         while (p)
989         {
990             printf(" %d",p->adjvex);
991             p=p->nextarc;
992         }
993         printf("\n");
994     }
995 }
996 void visit(VertexType v)
997 {
998     printf(" %d %s\n",v.key,v.others);
999 }
```