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 CS2209

学　　号 U202215650

姓　　名 严浩洋

指导教师 李剑军

报告日期 2022 年 6 月 1 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 问题描述

实验要求构造顺序表，并创建一个菜单来实现功能的演示。功能包括**基础的**创建销毁与增删改查等操作，以及**进阶的**文件存取、多线性表管理与求最大子数组和等实用功能。

1.1.1 功能实现

具体实现的功能如下：

1. 基础功能
   (a) 初始化表
   (b) 销毁表
   (c) 清空表
   (d) 判定空表
   (e) 求表长
   (f) 获取元素
   (g) 查找元素
   (h) 获得前驱
   (i) 获得后继
   (j) 插入元素
   (k) 删除元素
   (l) 遍历表
2. 进阶功能
   (a) 最大连续子数组和
   (b) 和为 K 的子数组
   (c) 顺序表排序
   (d) 文件形式存取线性表
   (e) 多线性表管理

1.1.2 实验目的

1. 加强对线性表与顺序存储结构的理解和使用，掌握实现线性表基础功能的方法。
2. 通过对顺序存储结构的实现，了解顺序表逻辑结构与物理结构的关系。
3. 会运用线性表实现常用的功能，并解决常见的问题。

1.2 系统设计

1.2.1 系统总体设计

本系统在初始条件下提供一个顺序存储结构的线性表，用户可自行添加新的线性表或删除原有的线性表，总表数不超过十个。每个线性表**相互独立**，均可独立实现所有的功能。系统所能实现的功能包括基础功能和进阶功能两部分，已在上一节中列出。

系统利用一个简易菜单以供用户选择需要的功能。通过**清屏函数**等操作，并用 **system** 相关函数调整窗口大小，可以使菜单选项始终位于界面上方，方便用户选择。

主函数中主要包括相关变量的定义以及菜单的实现，利用 **switch 分支结构**以实现功能的选择，并将特定的功能分别封装于不同的函数中。在主函数中只需调用相应函数即可。

1.2.2 数据结构设计

本实验以顺序存储结构线性表为主体，但定义了新的结构：Lists，用于多线性表管理。

Lists 包含两部分：Lists 的长度和最多可存储十个 SqList 的数组。前者用于表示管理线性表的个数，后者用于存储线性表。

1.3 系统实现

1.3.1 算法设计

1. 初始化线性表

函数名称：InitList

初始条件：线性表 L 不存在。

操作结果：构造一个空的线性表。

算法思路：若线性表不存在，分配存储空间后，将表长设为 0，再将线性表容量设为预定义的初始存储容量，图1-1为函数的流程图。

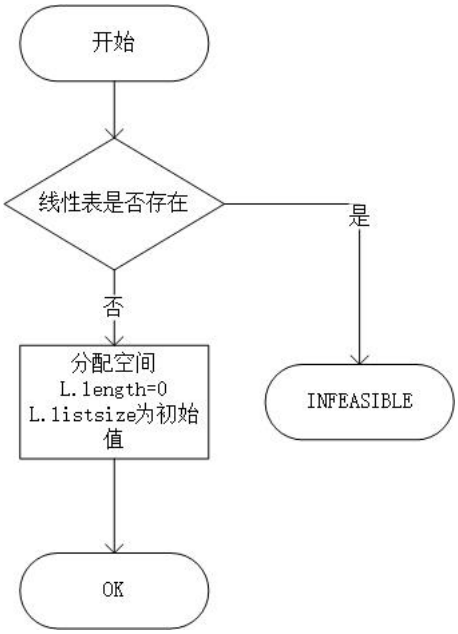

图 1-1　初始化线性表流程图

2. 销毁线性表

函数名称：DestroyList。

初始条件：线性表 L 已存在。

操作结果：销毁线性表 L。

算法思路：若线性表存在，释放内存并将各数据设置为初值，图1-2为 DestroyList 的流程图。

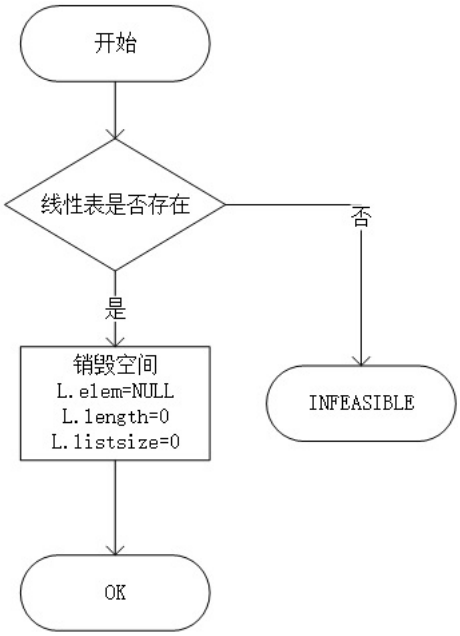

图 1-2　初始化线性表流程图

3. 清空线性表

函数名称：ClearList。

初始条件：线性表 L 已存在。

操作结果：将 L 重置为空表。

算法思路：若线性表存在，则将长度设置为 0 即可。图1-3为 ClearList 的流程图。

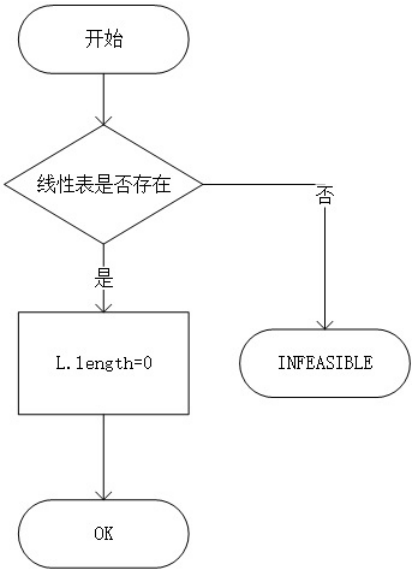

图 1-3　清空线性表流程图

4. 判断空线性表

函数名称：ListEmpty。

初始条件：线性表 L 已存在。

操作结果：若 L 为空表则返回 TRUE, 否则返回 FALSE。

算法思路：若线性表存在，则检验 L.length 是否为 0 即可。图1-4为 ListEmpty 的流程图。

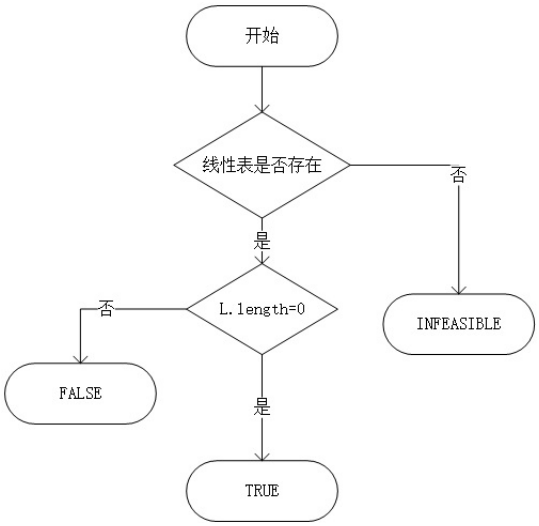

图 1-4　判断空线性表流程图

5. 求线性表长

函数名称：ListLength。

初始条件：线性表 L 已存在。

操作结果：返回 L 中数据元素的个数。

算法思路：若线性表存在，直接返回表长即可。图1-5是 ListLength 的流程图。

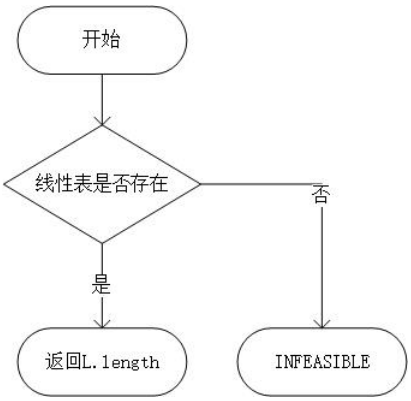

图 1-5　求线性表长流程图

6. 获得线性表元素

函数名称：GetElem。

初始条件：线性表 L 已存在。

操作结果：用 e 返回 L 中第 i 个数据元素的值。

算法思路：若线性表存在，返回 L.elem[i-1] 即可。图1-6是获得线性表元素的流程图。

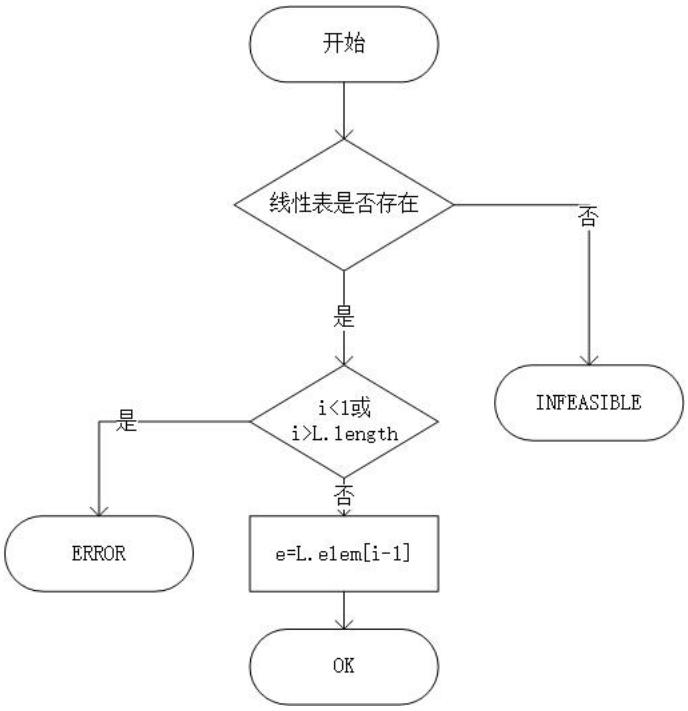

图 1-6　获得线性表元素流程图

7. 查找元素

函数名称：LocateElem。

初始条件：线性表 L 已存在。

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

算法思路：若线性表存在，遍历线性表找到值为 e 的点，返回其下标值加一即可。图1-7为查找元素的流程图。

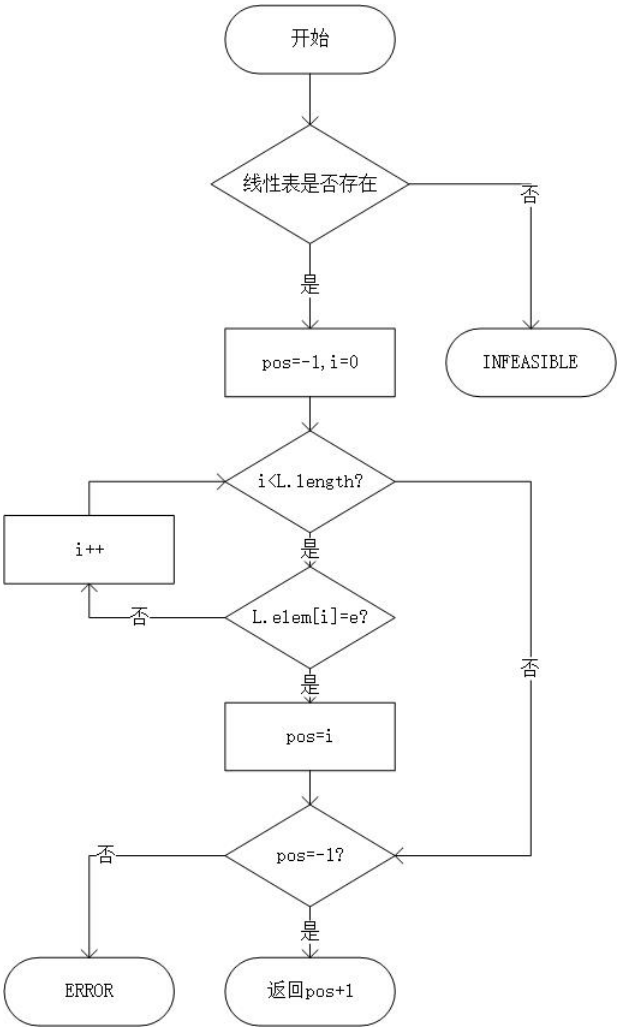

图 1-7　查找元素流程图

8. 获得前驱

函数名称：PriorElem。

初始条件：线性表 L 已存在。

操作结果：若 e 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。

算法思路：若线性表存在，对其从第二个数进行遍历，把 L.elem[i-1] 的值赋给 pre 即可。图1-8是获得前驱的流程图。

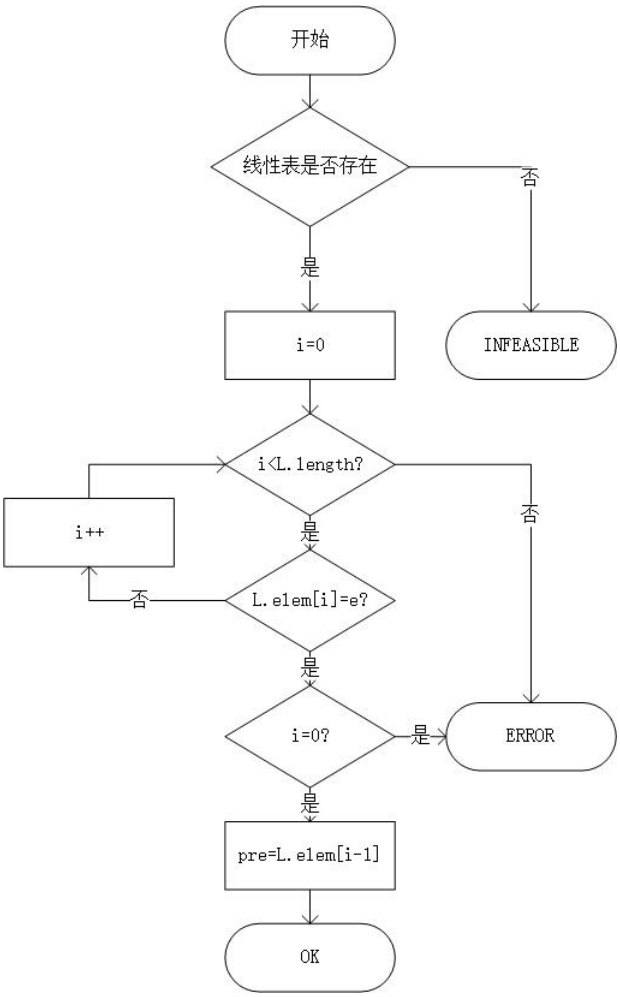

图 1-8　获得前驱流程图

9. 获得后继

函数名称：NextElem。

初始条件：线性表 L 已存在。

操作结果：若 e 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。

算法思路：若线性表存在，对其进行遍历直至倒数第二个数，把 L.elem[i+1] 的值赋给 next 即可。图1-9是获得后继的流程图。

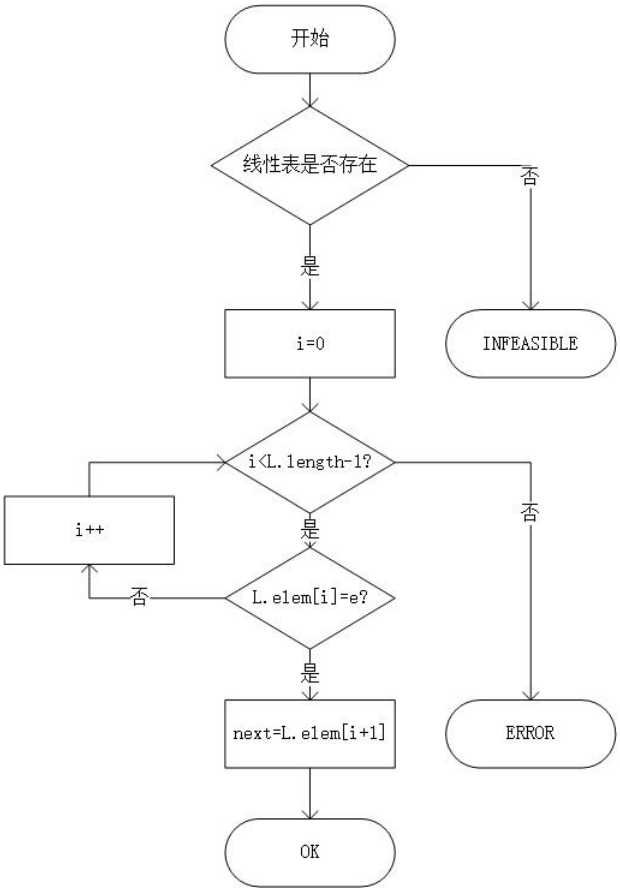

图 1-9　获得后继流程图

10. 插入元素

函数名称：ListInsert。

初始条件：线性表 L 已存在。

操作结果：在 L 的第 i 个位置之前插入新的数据元素 e。

算法思路：若线性表存在且下标合法，那么线性表第 i 个位置后的元素向后移一位，在第 i 个位置插入该数即可。图1-10是插入元素的流程图。

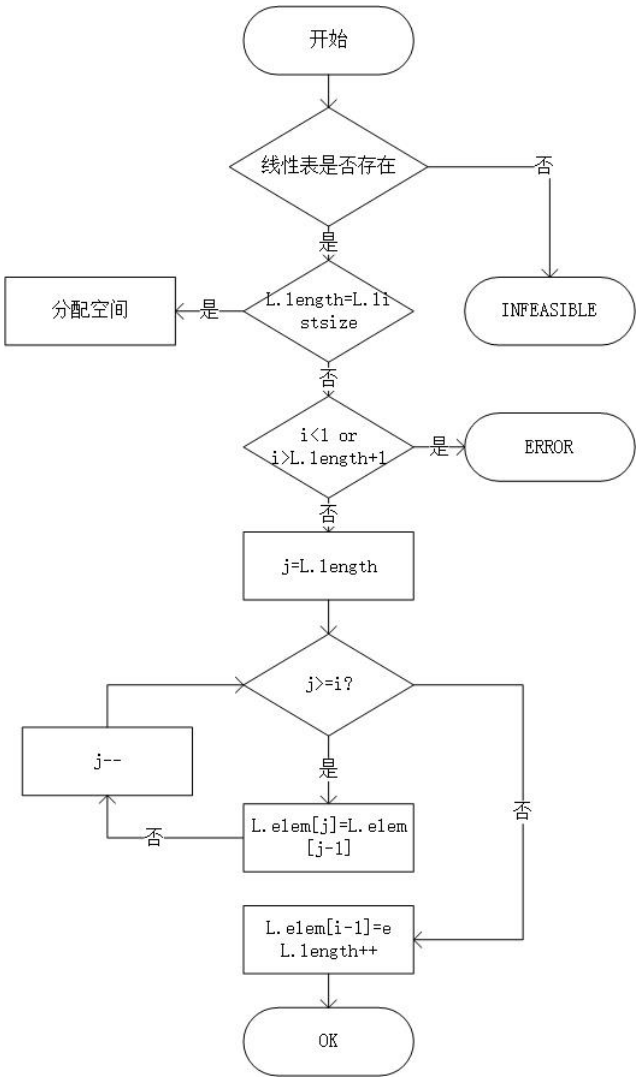

图 1-10　插入元素流程图

11. 删除元素

函数名称：ListDelete。

初始条件：线性表 L 已存在。

操作结果：删除 L 的第 i 个数据元素，用 e 返回删除的值。

算法思路：若线性表存在，则将第 i 个位置之后的元素向前移一位即可。图1-11是删除元素的流程图。

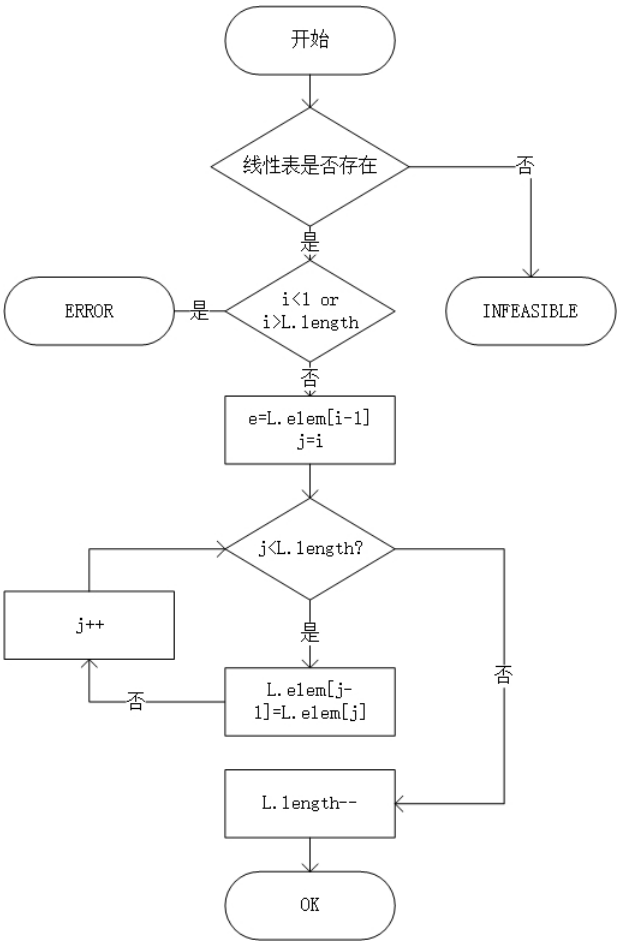

图 1-11　删除元素流程图

12. 遍历线性表

函数名称：ListTraverse。

初始条件：线性表 L 已存在。

操作结果：依次访问 L 的每个数据元素。

算法思路：若线性表存在，从第一个位置遍历到最后一个位置即可。图1-12是遍历线性表的流程图。

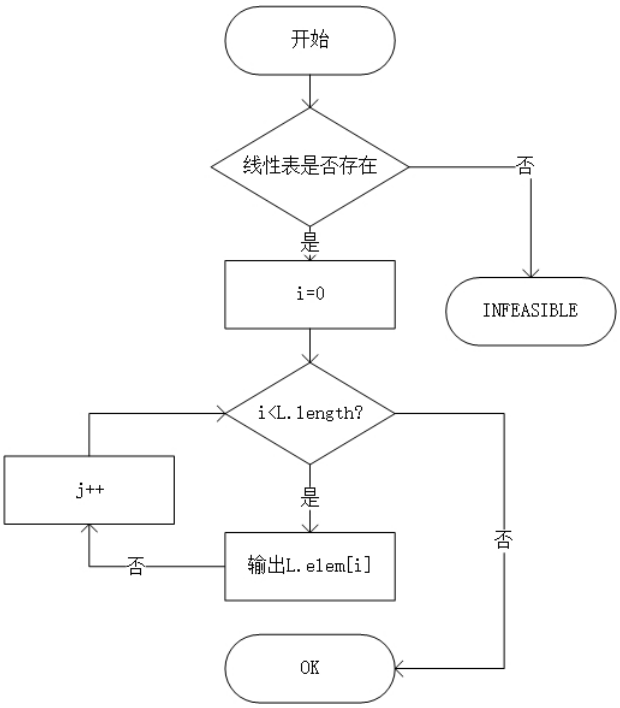

图 1-12　遍历线性表流程图

13. 最大连续子数组和

函数名称：MaxSubArray。

初始条件：线性表 L 已存在。

操作结果：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。

算法 1.1. 求最大连续子数组和

Input: List L

Output: the sum of max subarray

procedure MAXSUBARRAY(L)

for $i = 0 \rightarrow L.length$ **do**

```
        sum += L.elem[i]
        ans=max(sum,ans)
        if sum < 0 then
            sum = 0
        end if
    end for
    return ans
end procedure
```

14. 和为 K 的子数组

函数名称：SubArrayNum

初始条件：线性表 L 已存在。

操作结果：返回该数组中和为 k 的连续子数组的个数。

算法 1.2. 和为 K 的子数组

```
Input: List L, int k
Output: the num of array with sum of k
  procedure SubArrayNum(L,k)
    for start = 0 → L.length do
      sum = 0
      for end = start → 0 do
        sum+=L.elem[end]
        if sum=k then
          cnt++
        end if
      end for
    end for
    return cnt
  end procedure
```

15. 顺序表排序

函数名称：SortList

初始条件：线性表 L 已存在。

操作结果：将 L 由小到大排序。

算法 1.3. 顺序表排序

Input: List L

Output: L from small to large

```
procedure SortList(L)
    for i = 0 → L.length do
        for j = 1 → L.length − i do
            if L.elem[j]<L.elem[j-1] then
                swap(L.elem[j],L.elem[j-1])
            end if
        end for
    end for
    return OK
end procedure
```

16. 顺序表的文件形式储存

本功能主要包括两个函数 SaveList 和 LoadList，分别用于储存线性表到文件和读取文件中的线性表。

对于 SaveList 函数，若线性表存在，则先将当前线性表的元素个数和元素空间大小输入到文件中，然后将线性表中的元素按物理顺序输入即可。

对于 LoadList 函数，若当前线性表不存在，则可进行读取操作，先读入文件中保存的顺序表的空间大小和元素个数，分配相同的空间后，依次读入文件中顺序表的元素即可。

17. 实现多个线性表管理

本功能主要包括三个函数 AddList、RemoveList 和 LocateList，分别用于添加线性表、移除线性表和定位线性表。

对于 AddList 函数，若输入的名称不重复，则新建一个顺序表，长度加一，

长度和空间赋值为 0。

对于 RemoveList 函数，若存在线性表的名称与用户输入的名称相同，则将其从多表系统中移除，将序号大于该表的线性表前移一位，长度减一即可。

对于 LocateList 函数，若存在线性表的名称与用户输入的名称相同，则将该表的序号赋给用户操作的线性表的指针。

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1-13所示。

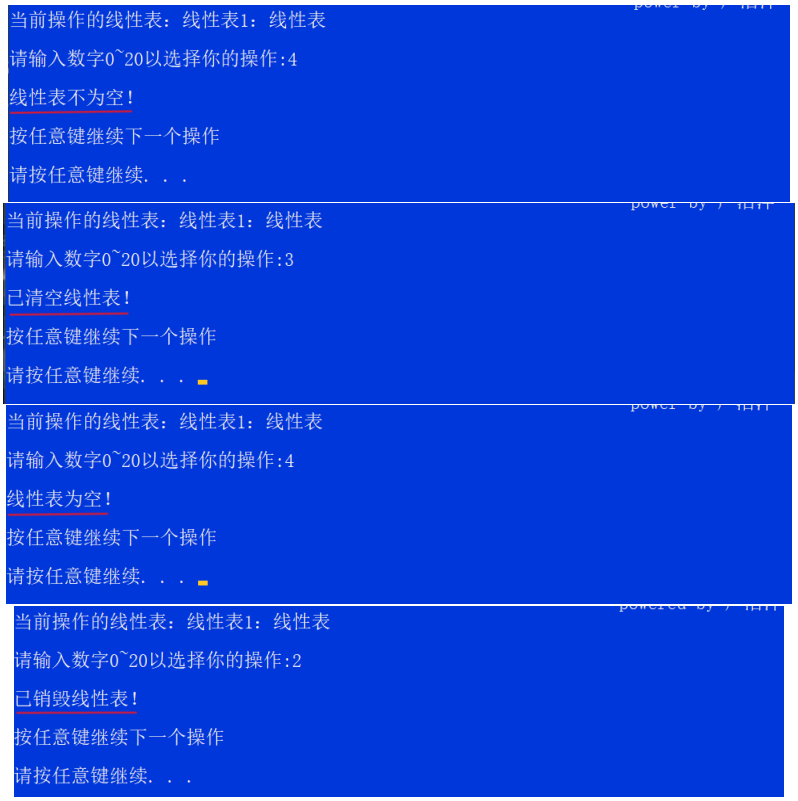

图 1-13　清空线性表与线性表判空测试组的截图

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

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。

测试结果如下图1-14所示。

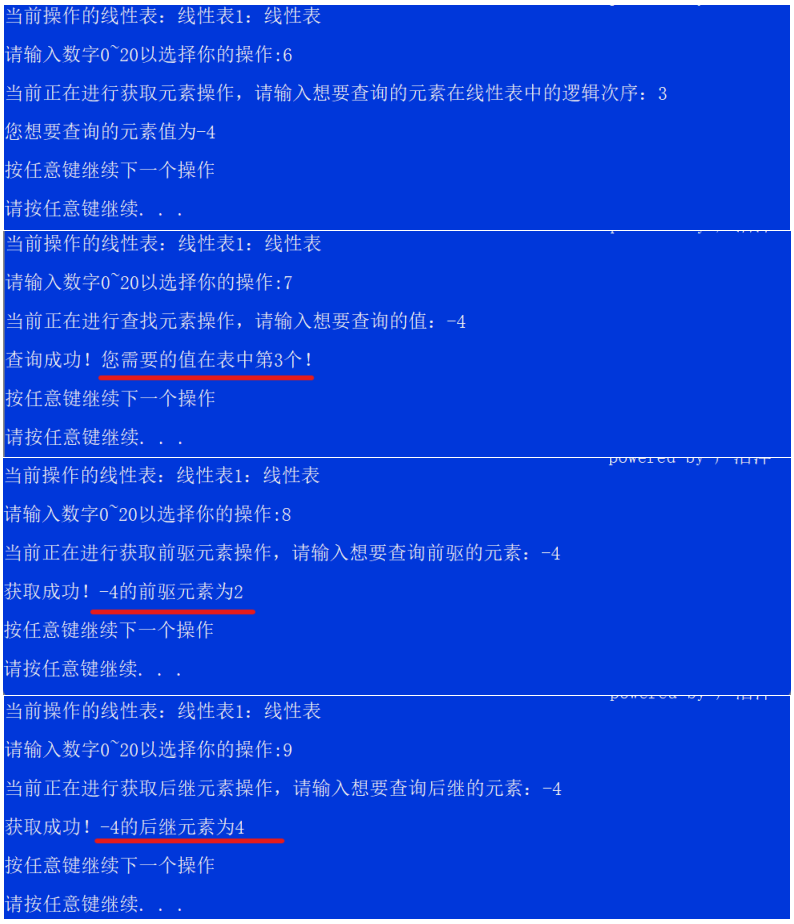

图 1-14　获得、查找元素及其前驱后继测试组的截图

除标准数据外，下面测试了一些不合法数据、临界数据，依次为输入下标不合法、获取元素不在线性表中、获取前驱的元素为表头、获取后继的元素不

在线性表中的情况。测试结果如下图1-15所示。

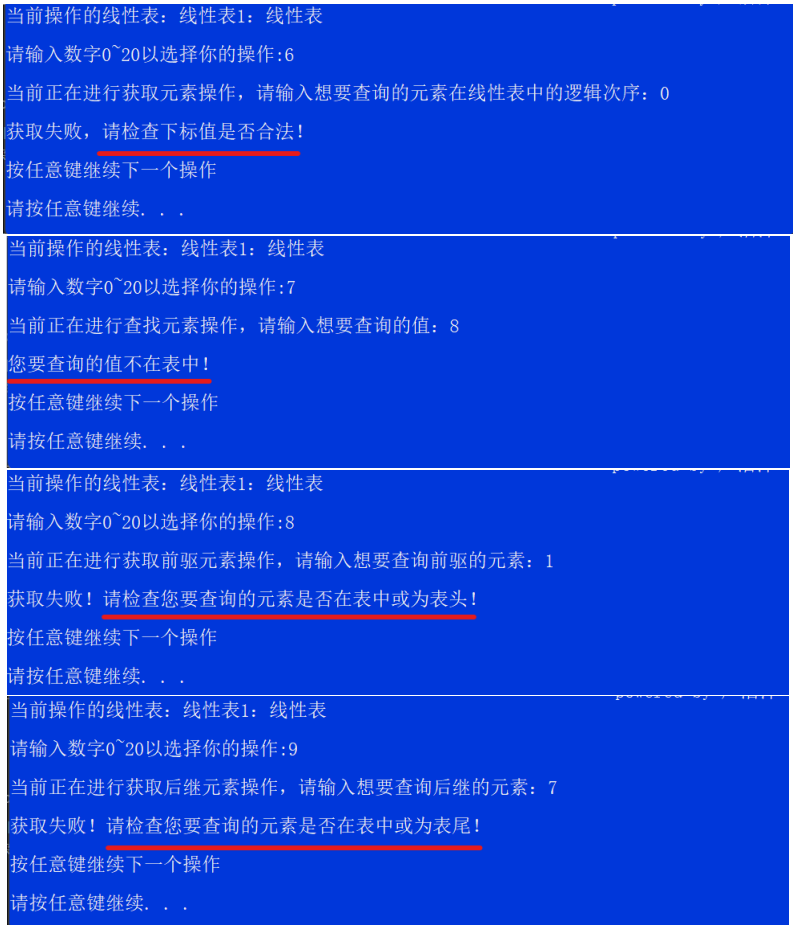

图 1-15　不合法数据、临界数据的测试

以上结果均符合预期。

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 3 2	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 3	删除的元素为 2	删除的元素为 2
	12	1 2 -4 4 5 -5 -3 -4	1 2 -4 4 5 -5 -3 -4

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、图1-17所示。

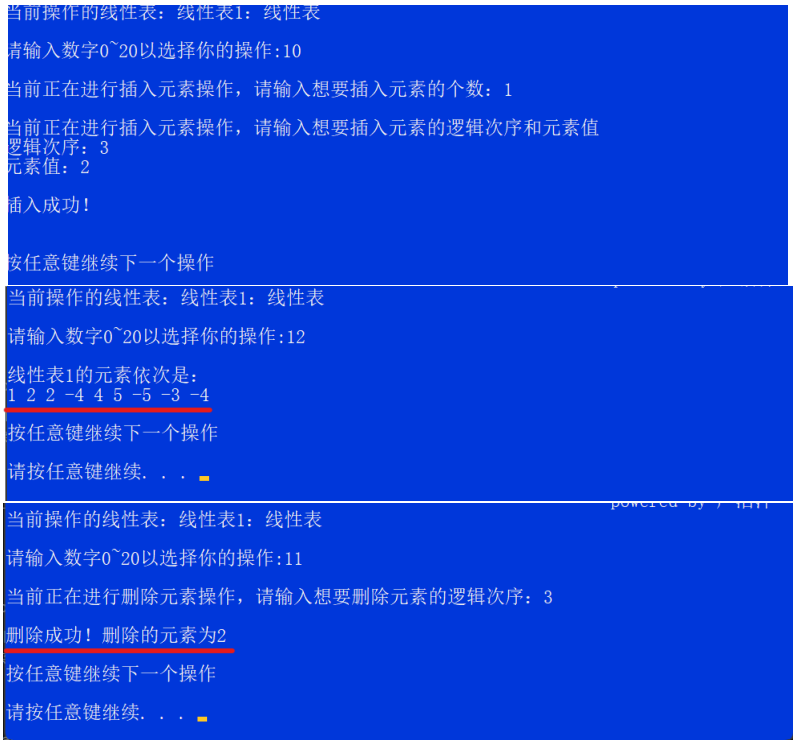

图 1-16　插入、删除、遍历元素测试组的截图 (1)

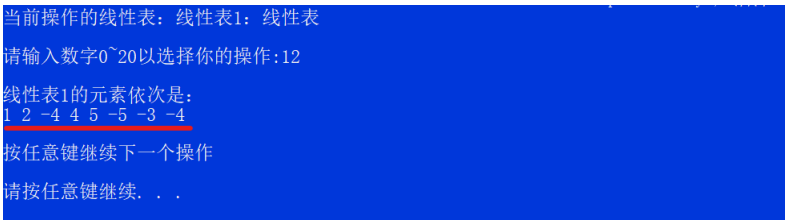

图 1-17　插入、删除、遍历元素测试组的截图 (2)

除标准数据外，下面测试了一些不合法数据、临界数据，依次为插入、删除下标不合法，线性表不存在的情况。测试结果下图1-18所示。

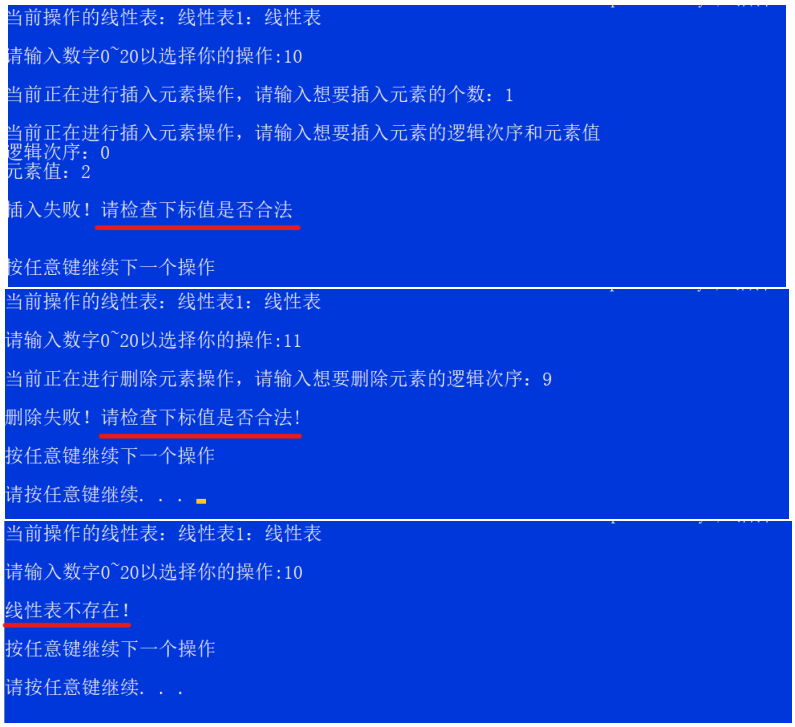

图 1-18　不合法数据、临界数据的测试

以上结果均符合预期。

1.5 实验小结

本次实验以实现顺序表的逻辑结构及其功能为主要目的。基础功能整体难度不大，附加功能中，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。

这次实验中我学习并练习了许多东西。最大连续子数组和题目中，我首先使用动态规划进行实现，后来我学习并使用了在线处理这一算法，有效降低了算法的空间复杂度。顺序表排序中，我也练习了《数据结构》课程第十章中的多种排序方法。文件存取和多线性表管理的功能也让我复习了上学期《C 语言程序设计》课程中学到的结构与文件的知识。为美化程序界面，我学习了一些关于 system 函数的知识，改变了程序框的大小和颜色。

本次实验中我也有一些常见的错误。在进行多线性表管理时，常常会错误地判断线性表个数，这是由于在相关功能上不能正确地增减线性表个数，如“初始化线性表”、“销毁线性表”、“读取线性表”、“添加/移除线性表”等等。最终建立了正确合理的增减线性表数目的规则。在设计系统主体时，我没有首先考虑后期多线性表管理的实现，后期再进行删改，耗费了一些时间。

编写实验报告时，我复习了 visio 画流程图的方法，学习了使用伪代码描述算法的格式，锻炼了综合能力，为将来的科研奠定了基础。

2 基于邻接表的图实现

2.1 问题描述

实验要求以邻接表的形式实现图这一数据结构，并创建一个菜单来实现功能的演示。功能包括**基础的**创建、销毁与增删改查等操作，其中增删包括结点与弧的增删，搜索包括常见的深度优先搜索 (DFS) 与广度优先搜索 (BFS)，以及进阶的文件存取、多图管理与距离小于 k 的顶点集合、最短路径等使用功能。

2.1.1 功能实现

具体实现的功能如下：

1. 基础功能
   (a) 创建图
   (b) 销毁图
   (c) 查找顶点
   (d) 顶点赋值
   (e) 获得第一邻接点
   (f) 获得下一邻接点
   (g) 插入顶点
   (h) 删除顶点
   (i) 插入弧
   (j) 删除弧
   (k) 深度优先搜索遍历
   (l) 广度优先搜索遍历
2. 进阶功能
   (a) 距离小于 k 的顶点集合
   (b) 顶点间最短路径和长度
   (c) 图的连通分量
   (d) 实现图的文件形式保存
   (e) 实现多个图管理

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. 以邻接表作为物理结构，熟练掌握图基本运算的实现

2.2 系统设计

2.2.1 系统总体设计

本系统再初始条件下提供一个以邻接表形式存储的图，用户可自行添加新的图或移除原有的图，图的个数不超过十个。每个图之间**相互独立**，均可独立实现所有的功能，也可通过“选择图”功能切换要操作的图。系统所实现的功能包括基础功能和进阶功能两部分，已在上一节列出。

系统利用一个简易菜单以供用户选择需要的功能。通过**清屏函数**等操作，并用 **system** 相关函数调整窗口大小，可以使菜单选项始终位于界面上方，方便用户选择。

主函数中主要包括相关变量的定义以及菜单的实现，利用 **switch 分支结构**以实现功能的选择，并将特定的功能分别封装于不同的函数中。在主函数中只需调用相应函数即可。

2.2.2 数据结构设计

本实验使用了多个结构体以定义图的各个部分，包括：

1. VertexType：表示图的结点
2. ArcNode：表示邻接表结点，即表示弧
3. VNode：表示头节点
4. ALGraph：表示图

除以上图的相关定义外，本实验沿用顺序表的结构：Lists，以实现多图管理。

2.3 系统实现

2.3.1 算法设计

1. 创建图

函数名称：CreateGraph

初始条件：无向图 G 不存在，给定结点与邻接数组，结点不超过二十个。

操作结果：构造一个无向图 G，要求满足各顶点关键字的唯一性。

算法思路：先计算弧与结点的个数，判断关键字是否重复、是否为空图、是否超过二十个结点，若以上情况均不存在，则对每条边使用 InsertArc 函数。图2-1为 CreateGraph 的流程图。

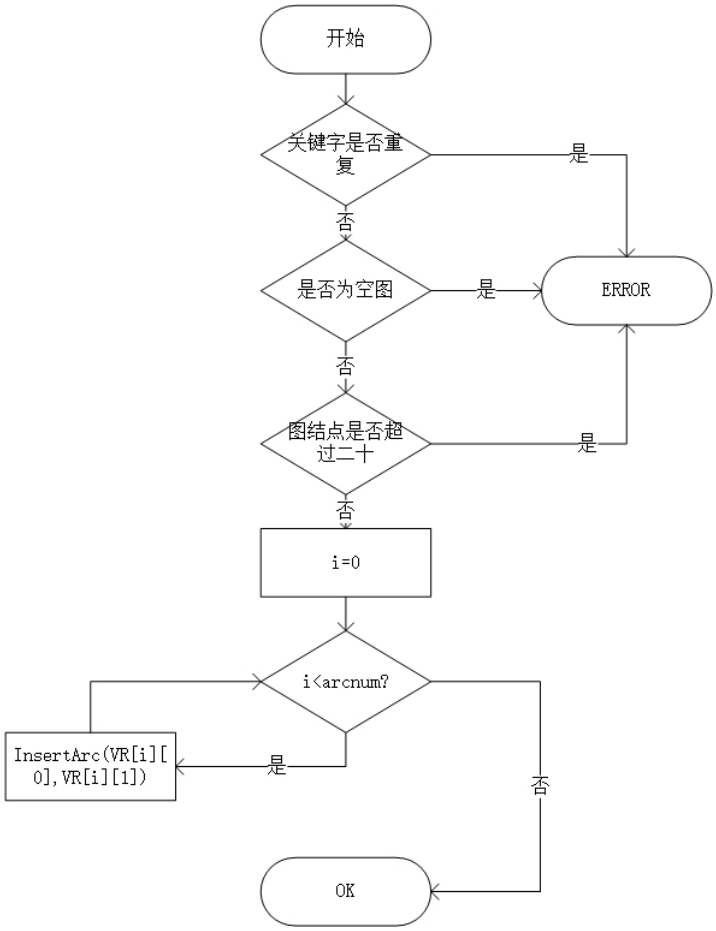

图 2-1 创建图流程图

2. 销毁图

函数名称：DestroyGraph

初始条件：无向图 G 存在。

操作结果：邻接表表示的无向图销毁，并释放所有表结点的空间。

算法思路：对每一个表头结点，由 firstarc 开始依次释放弧结点，最后把 G.vexnum、G.arcnum 改为 0 即可。图2-2为 DestroyGraph 的流程图。

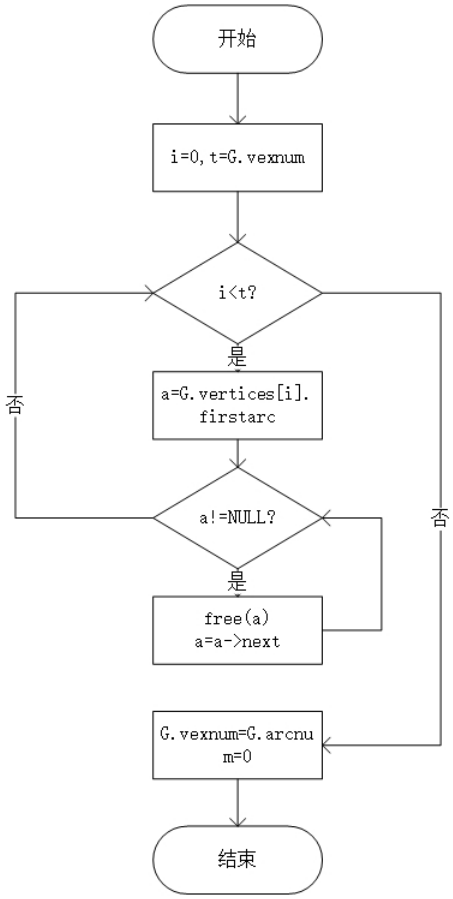

图 2-2　销毁图流程图

3. 查找顶点

函数名称：LocateVex

初始条件：无向图 G 存在。

操作结果：成功时返回关键字为 u 的顶点位置序号（简称位序），否则返回-1。

算法思路：遍历每一个头节点即可，判断结点关键字是否与传入的关键字 u 相等，若查询成功返回当前位序，否则返回-1。图2-3为 LocateVex 的流程图。

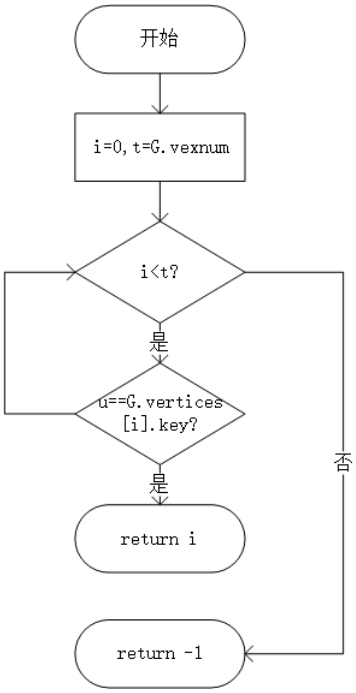

图 2-3　查找顶点流程图

4. 顶点赋值

函数名称：PutVex

初始条件：无向图 G 存在。

操作结果：对关键字为 u 的顶点赋值 value(要求关键字具有唯一性)。

算法思路：先判断关键字是否重复，若不重复则遍历头结点，找到对应关键字 u 的点，更改关键字即可。图2-4为 PutVex 的流程图。

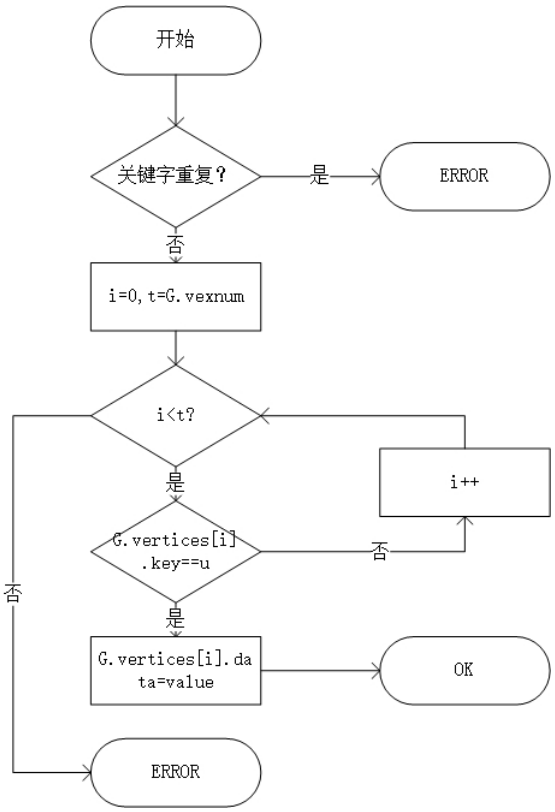

图 2-4　顶点赋值流程图

5. 获得第一邻接点

函数名称：FirstAdjVex

初始条件：无向图 G 存在。

操作结果：返回关键字为 u 的顶点第一个邻接顶点的位序，否则返回-1。

算法思路：先判断关键字是否重复，若不重复则遍历头结点，找到对应关键字 u 的点，更改关键字即可。图2-5为 FirstAdjVex 的流程图。

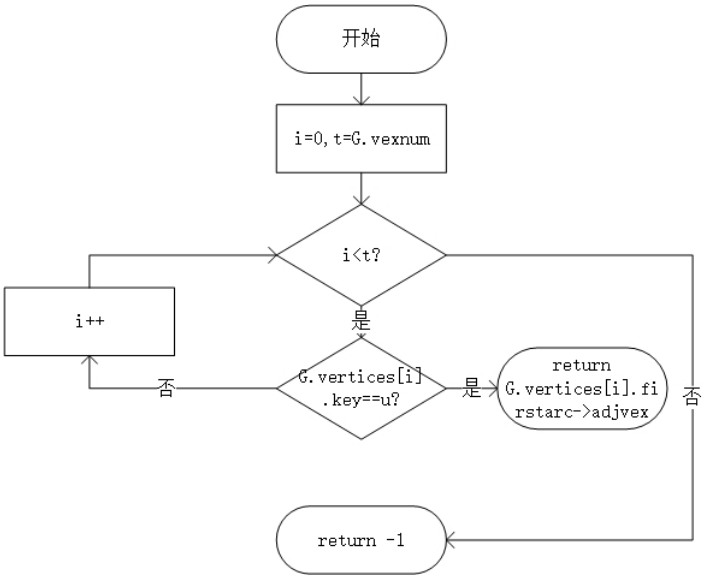

图 2-5　获得第一邻接点流程图

6. 获得下一邻接点

函数名称：NextAdjVex

初始条件：无向图 G 存在。

操作结果：返回 v 相对于 w 的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。

算法思路：先遍历寻找结点 v，然后在 v 的邻接结点中寻找 w。若 w 不是最后一个邻接结点，则返回其 nextarc 指向的 adjvex。

7. 插入顶点

函数名称：InsertVex

初始条件：无向图 G 存在。

操作结果：在图 G 中插入新顶点 V，要求满足关键字唯一性。

算法思路：先判断结点数是否超出上限，关键字是否重复。然后在头节点数组末尾插入新节点，G.vexnum+1 即可。图2-6是 InsertVex 的流程图。

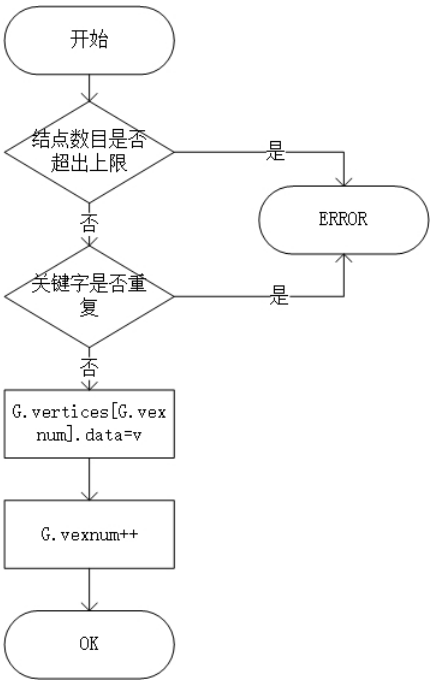

图 2-6　插入顶点流程图

8. 删除顶点

函数名称：DeleteVex

初始条件：无向图 G 存在。

操作结果：在图 G 中删除关键字 v 对应的顶点以及相关的弧。

算法思路：先遍历头结点数组，找到关键字对应的结点，释放其所有弧结点后，将该节点后的全部结点向前移一位，G.vexnum-1。然后对每一个头结点进行遍历，遍历每个结点的所有弧结点，若找到有关联删去点的弧则释放。

9. 插入弧

函数名称：InsertArc

初始条件：无向图 G 存在。

操作结果：在图 G 中增加弧 <v,w>。

算法思路：先遍历头结点，找到 v 和 w 对应头结点的位序，若未找到则返回 ERROR。然后新建弧结点，关联结点 w，将其作为结点 v 的 firstarc。同理可关联 w 到 v 的弧。

10. 删除弧

函数名称：DeleteArc

初始条件：无向图 G 存在。

操作结果：在图 G 中删除弧 <v,w>。

算法思路：先遍历头结点，找到 v 和 w 对应头结点的位序，若未找到则返回 ERROR。然后遍历 V 的弧结点，找到关联 W 的弧则释放，同理释放 W 弧中关联 V 的弧结点，若未找到关联 v、w 的弧则返回 ERROR。

11. 深度优先搜索遍历

函数名称：DFSTraverse

初始条件：无向图 G 存在。

操作结果：对图 G 进行深度优先搜索遍历。

可以使用深度优先搜索 (DFS) 递归函数进行辅助，对每个头结点进行 DFS 操作，用标记数组 flag 记录结点是否遍历。

算法 2.1. 深度优先搜索

Input: ALGraph G,int v

Output: Elements of G in DFS sequence

```
procedure DFS(G,v)
    arc=G.vertices[v].firstarc
    visit(G.vertices[v].data)
    flag[v]=1
    while arc do
        if !flag[arc->adjvex] then
            DFS(G,arc->adjvex)
        end if
        arc=arc->nextarc
    end while
    return
end procedure
```

12. 广度优先搜索遍历

函数名称：BFSTraverse

初始条件：无向图 G 存在。

操作结果：对图 G 进行广度优先搜索遍历。

可以使用队列 (queue) 这一数据结构进行辅助，用标记数组 flag 记录结点是

否遍历过。

算法 2.2. 广度优先搜索

Input: ALGraph G

Output: Elements of G in BFS sequence

```
procedure DFS(G)
    for i = 0 → G.vexnum do
        if !flag[i] then
            visit(G.vertices[i].data)
            queue[tail++]=G.vertices[i].firstarc
            while front<tail do
                arc=queue[front++]
                if !flag[arc->adjvex] then
                    flag[arc->adjvex]=1
                    visit(G.vertices[arc->adjvex].data)
                    邻接结点全部入队
                end if
            end while
        end if
    end for
    return
end procedure
```

13. 距离小于 k 的顶点集合

函数名称：VerticesSetLessThanK

初始条件：无向图 G 存在。

操作结果：返回与顶点 v 距离小于 k 的顶点集合。

算法思路：先遍历头结点数组，找到关键字对应的结点，由于“深度”与“距离”的相似性，可以利用 dfs 函数递归遍历所有距离小于 K 的点。

14. 顶点间最短路径和长度

函数名称：ShortestPathLength

初始条件：无向图 G 存在。

操作结果：返回顶点 v 与顶点 w 的最短路径的长度。

可以利用 Floyd-Warshall 算法求任意两点间最短路径，然后带入这两点的值返回即可。

算法 2.3. 最短路径算法

Input: ALGraph G,int a,int$b$

Output: the shortest length between a and b

```
procedure ShortestPathLength(G)
    for i → G.vexnum do
        for j → G.vexnum do
            for k → G.vexnum do
                if G[i][j]>G[i][k]+G[k][j] then
                    G[i][j]>G[i][k]+G[k][j]
                end if
            end for
        end for
    end for
    return G[a][b]
end procedure
```

15. 图的连通分量

函数名称：ConnectedComponentsNums

初始条件：无向图 G 存在。

操作结果：返回图 G 的所有连通分量的个数。

算法思路：同样利用深度优先搜索算法，从一个头结点开始遍历，遇到未遍历的头结点，则连通分量加一。

16. 图的文件形式储存

本功能主要包括两个函数 SaveGraph 和 LoadGraph，分别用于储存图到文件和读取文件中的图。

对于 SaveGraph 函数，若图存在，则先遍历头结点数组，将所有弧存储在 VR

二维数组中，然后将节点数组和关系数组依次写入到文件中。

对于 LoadGraph 函数，若当前图不存在，则可进行读取操作，将文件中的节点数组和关系数组读入后，直接调用 CreateGraph 函数即可。

17. 实现多个图管理

本功能主要包括三个函数 AddGraph、RemoveGraph 和 LocateGraph，分别用于添加图、移除图和定位图。

对于 AddGraph 函数，若输入的名称不重复，则新建一个图，长度加一，结点数和边数赋值为 0。

对于 RemoveGraph 函数，若存在图的名称与用户输入的名称相同，则将其从多图系统中移除，将序号大于该表的图前移一位，长度减一即可。

对于 LocateGraph 函数，若存在图的名称与用户输入的名称相同，则将该图的序号赋给用户操作的图的指针。

2.3.2 程序源代码

见《附录 B 基于邻接表图实现的源程序》

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

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

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。

测试结果如下图2-7所示。

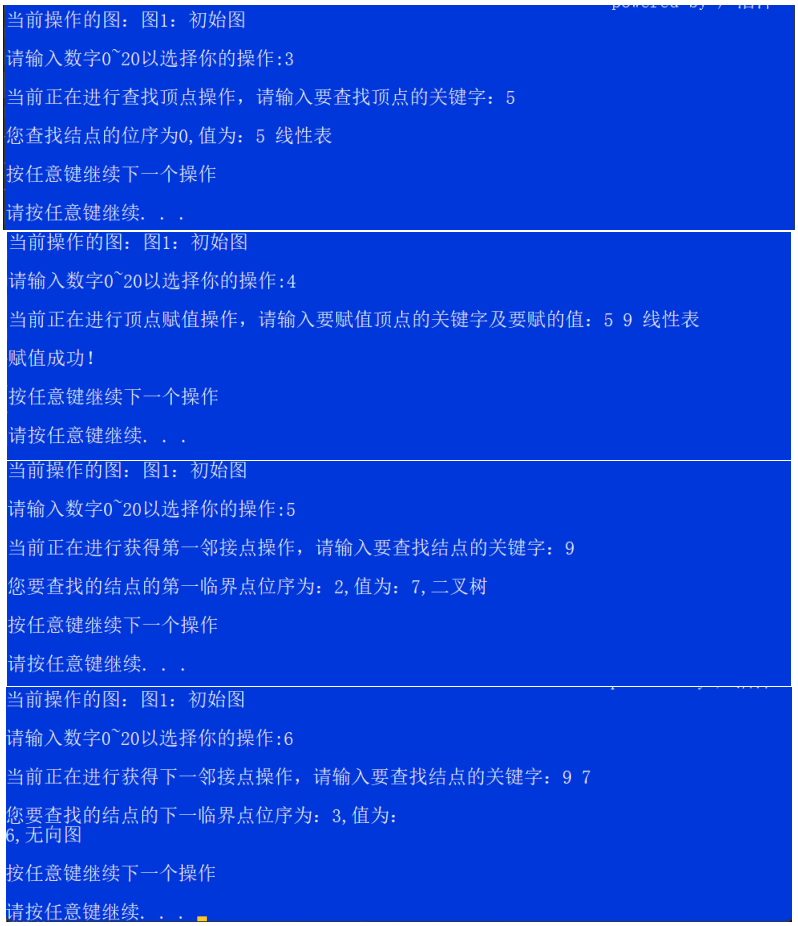

图 2-7　查找顶点、顶点赋值、获取邻接点测试组的截图

除标准数据外，下面测试了一些不合法数据、临界数据，依次为查找结点不

存在、赋值结点不存在、获取第一邻接点时图为空、没有下一邻接点的情况，测试结果如下图2-8所示。

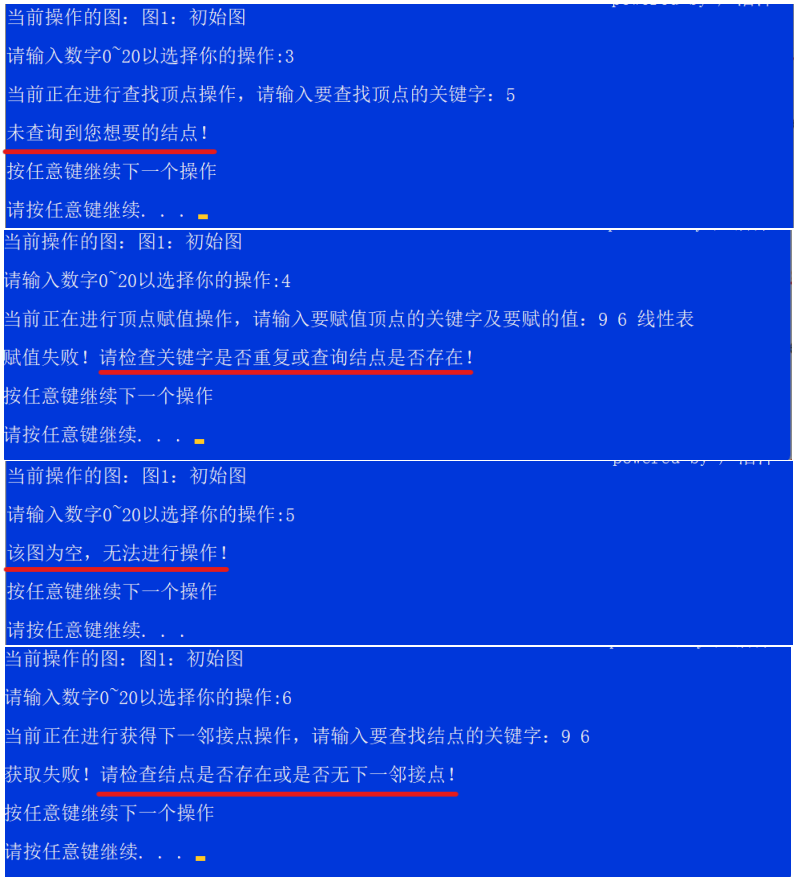

图 2-8　不合法数据、临界数据测试组的截图

表 2-2　插入、删除测试表

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！
	8 5 9	插入成功！	插入成功！
	11	5 9 7 8 6	5 9 7 8 6
	9 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

2. 插入、删除的测试

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。

测试结果如下图2-9、图2-10所示。

图 2-9　插入、删除测试组的截图 (1)

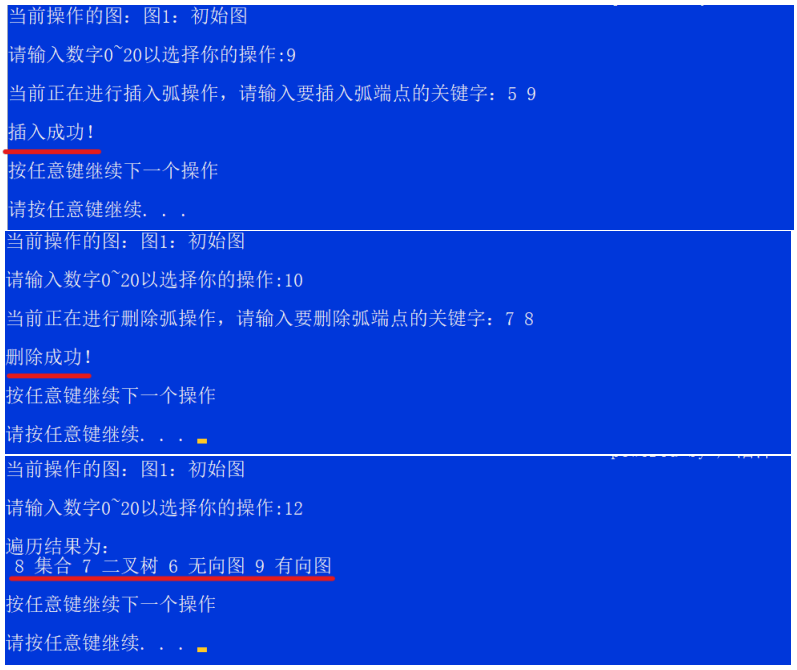

图 2-10　插入、删除测试组的截图 (2)

除标准数据外，下面测试了一些不合法数据、临界数据，依次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2-11、图2-12所示。

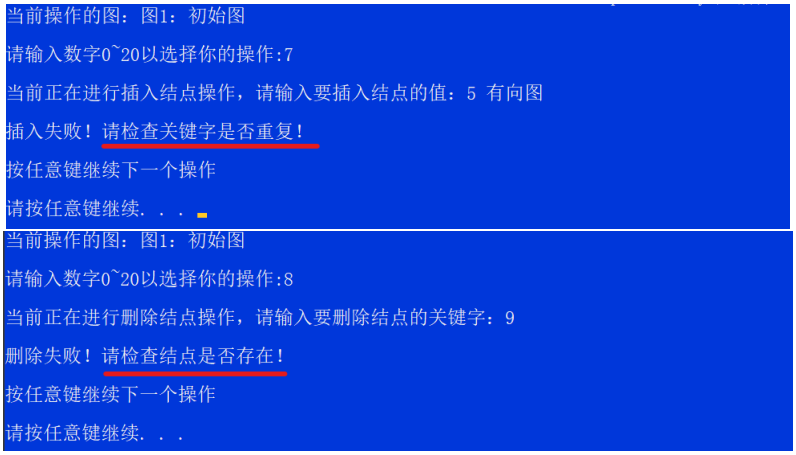

图 2-11　不合法数据、临界数据测试组的截图 (1)

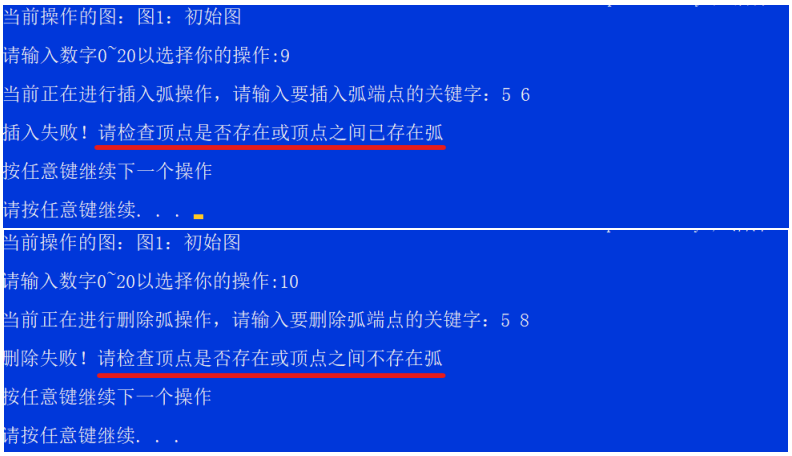

图 2-12　不合法数据、临界数据测试组的截图 (2)

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	15 2 3 16 2 7 17	3 c 6 f 1 a 3 2	3 c 6 f 1 a 3 2

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 3 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。

测试结果如下图2-13所示。

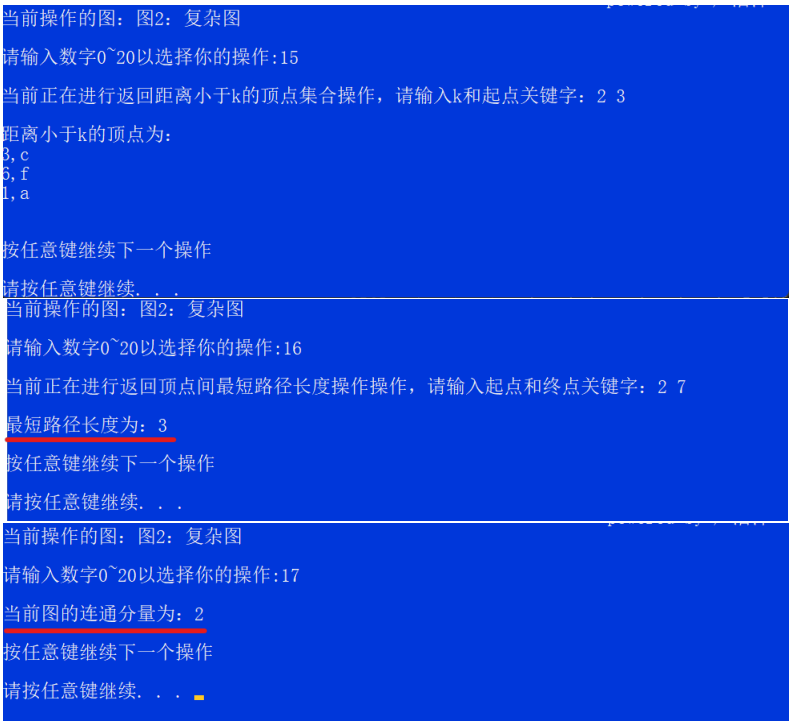

图 2-13　附加功能测试组的截图

2.5　实验小结

本次实验以实现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，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也较为经典，难度不大。

这次试验中我也学到了很多。深度优先搜索和广度优先搜素是在算法竞赛中甚至工程上常用的遍历算法，本次实验中多次运用了这两种算法，提高了熟练度。此外，深度优先搜索帮助我更好地理解递归的思想，广度优先搜素则让我更熟练地运用队列这一数据结构。附加功能中的最短路径问题则帮助我巩固了图论的相关知识。

本次实验中也有些地方由于掌握生疏而出现了错误。在进行删除、销毁操作时，我常常不清楚哪些结点空间应当被释放，从哪里入手，导致空间释放不正确，百思不得其解，也耗费了很多时间。这实际上是概念掌握不熟练导致的。在回顾课本、询问同学之后，最终还是正确地释放了结点空间，从错误中学到了知识。

3 课程的收获和建议

本次数据结构实验课程中我学到了很多，也突破了自我。这是我第一次写下超过千行的代码文件，第一次构建一个可供他人使用的“系统”，也是第一次用 LaTex 写下程序设计相关的实验报告。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。

关于课程整体建议提前 1-2 周开课，并且课程中间可以间隔一周，这样可以有更充足的时间来编写代码。

3.1 基于顺序存储结构的线性表实现

顺序表是课程中最为简单的一章。顺序存储结构使用数组实现，可以进行随机存取，也方便实现排序等功能，较为简便。本次实验最大的收获则是初步练习如何将已经编写好的函数整合到一个代码文件中，也复习了文件存取和结构体的知识。

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。

3.3 基于二叉链表的二叉树实现

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，中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。本章结点空间的释放是实现过程中最常犯的错误，有时删除节点后再添加会导致运行错误，这就是未处理好空间分配的问题。由于在程序设计竞赛中使用数组实现二叉树较为方便，建议本章允许学生在数组和单链表中任选其一实现二叉树即可。

3.4 基于邻接表的图实现

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。邻接表的实现方式也给最短路径等算法的实现带来了挑战。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。

附录 A 基于顺序存储结构线性表实现的源程序

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<string.h>
#include<windows.h>

/**************相关定义与声明***************/
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2                          //溢出
typedef int status;
typedef int ElemType;
#define LIST_INIT_SIZE 100        //顺序表的初始大小
#define LISTINCREMENT  10         //顺序表每次分配增加的大小
typedef int ElemType;
typedef struct{                              //顺序表（顺序结构）的定义
       ElemType * elem;
       int length;
       int listsize;
 }SqList;
 typedef struct{   //线性表的集合类型定义
     struct { char name[30];
              SqList L;
     } elem[11];
     int length;
}LISTS;
LISTS Lists;        //线性表集合的定义Lists
/*****************************************/

/******************函数表*******************/
status InitList(SqList& L);
status DestroyList(SqList& L);
status ClearList(SqList& L);
status ListEmpty(SqList L);
status ListLength(SqList L);
status GetElem(SqList L,int i,ElemType &e);
int LocateElem(SqList L,ElemType e);
status PriorElem(SqList L,ElemType e,ElemType &pre);
status NextElem(SqList L,ElemType e,ElemType &next);
status ListInsert(SqList &L,int i,ElemType e);
status ListDelete(SqList &L,int i,ElemType &e);
status ListTraverse(SqList L);
status  SaveList(SqList L,char FileName[]);
status  LoadList(SqList &L,char FileName[]);
status AddList(LISTS &Lists,char ListName[]);
status RemoveList(LISTS &Lists,char ListName[]);
int LocateList(LISTS Lists,char ListName[]);
int MaxSubArray( SqList &L );
int SubArrayNum( SqList &L,int k );
status sortList( SqList &L );
```

```
54  /*****************************************/
55
56  /*****************主函数********************/
57  int main(){
58          /*****************************************************************/
59          /*                                                程序框设置
                                */
60          /*****************************************************************/
61          system("mode con cols=82 lines=32 ");    //更改程序框大小
62          system("title 线性表：顺序存储结构");    //更改程序框名称
63          system("color 1F");                                          //背景蓝色，字体亮白色
64          /*****************************************************************/
65          /*                                                初始化
                                    */
66          /*****************************************************************/
67          int op=1;                           //用户操作选项
68          SqList L[11];               //多线性表处理
69          LISTS list;                         //多线性表初始化
70          list.length=1;
71          char filename[50];          //存储文件名
72          char listname[50];          //存储线性表名
73          int lp=1;                           //当前处理的线性表指针
74          int val,pos;                //获取/查找元素用，val:元素的值,pos:元素的逻辑次序
75          ElemType pre,next;          //获取前驱/后继用，pre:前驱，next:后继
76          ElemType ins;               //插入元素用，存储插入元素的值
77          int cnt=0;                          //记录操作次数
78          //线性表初始化
79          for( int i=1 ; i<11 ; i++ ){
80                  list.elem[i].L.elem=NULL;
81                  list.elem[i].L.length=0;
82                  list.elem[i].L.listsize=0;
83          }
84          char *defaultname="线性表";
85          for( int i=0 ; defaultname[i]!='\0' ; i++ )
86                  list.elem[1].name[i]=defaultname[i];
87          /*****************************************************************/
88          /*                                                菜单主体
                                */
89          /*****************************************************************/
90          while(op){
91                  system("cls");
92              printf("\n\n");
93                  printf("           \t\t\t线性表顺序结构菜单\t\t\n\n");
94                  printf(" 可在最多10个顺序表进行多表操作，初始表尚未初始化，名称默认为“线性表”\n");
95                  printf(" ------------------------------------------------------------------------------\n"
                        );
96                  printf(" *1. 初始化线性表          \t\t7.  查找元素\t\t\t\t*\n");
97                  printf(" *2. 摧毁线性表         \t\t\t8.  获取前驱元素\t\t\t*\n");
98                  printf(" *3. 清空线性表           \t\t9.  获取后继元素 \t\t\t*\n");
99                  printf(" *4. 线性表判空           \t\t10. 插入元素\t\t\t\t*\n");
100                 printf(" *5. 线性表长度         \t\t\t11. 删除元素\t\t\t\t*\n");
101                 printf(" *6. 获取元素             \t\t12. 遍历线性表\t\t\t\t*\n");
102                 printf(" ------------------------------------------------------------------------------\n"
                        );
103                 printf(" *13.线性表存储           \t\t14. 线性表读取\t\t\t\t*\n");
104                 printf(" *15.最大连续子数组和      \t\t16. 和为K的子数组\t\t\t*\n");
105                 printf(" *17.从小到大排序          \t\t18. 添加线性表\t\t\t\t*\n");
```

```
106             printf(" *19.移除线性表          \t\t20. 选择要操作的线性表\t\t\t*\n\n");
107             printf(" *0. 退出\t\t\t\n");
108             printf(" ----------------------------------------------------------------powered by 严浩洋\n
                    ");
109             printf("当前操作的线性表：线性表%d：%s\n\n",lp,list.elem[lp].name);
110             printf("请输入数字0~20以选择你的操作:");
111             scanf("%d",&op);//选择op的值,用于switch
112             printf("\n");
113             switch(op){
114                     case 0:
115                             break;
116                     case 1:
117                             //初始化线性表
118                             if( list.elem[lp].L.elem ){
119                                     printf("创建失败，线性表已存在！");
120                                     break;
121                             }
122                             if( InitList(list.elem[lp].L)==OK )
123                                     printf("线性表创建成功！");
124                             else
125                                     printf("创建失败，线性表已存在！");
126                             break;
127                     case 2:
128                             //摧毁线性表
129                             if( !list.elem[lp].L.elem ){
130                                     printf("销毁失败，线性表不存在！");
131                                     break;
132                             }
133                             if( DestroyList(list.elem[lp].L)==OK )
134                                     printf("已销毁线性表！");
135                             else
136                                     printf("销毁失败，线性表不存在！");
137                             break;
138                     case 3:
139                             //清空线性表
140                             if( !list.elem[lp].L.elem ){
141                                     printf("清空失败，线性表不存在！");
142                                     break;
143                             }
144                             if( ClearList(list.elem[lp].L)==OK )
145                                     printf("已清空线性表！");
146                             else
147                                     printf("清空失败，线性表不存在！");
148                             break;
149                     case 4:
150                             //判断线性表是否为空
151                             if( ListEmpty(list.elem[lp].L)==INFEASIBLE )
152                                     printf("判断失败，线性表不存在！");
153                             else if( ListEmpty(list.elem[lp].L)==TRUE )
154                                     printf("线性表为空！");
155                             else    //ListEmpty(L[lp])==FALSE
156                                     printf("线性表不为空！");
157                             break;
158                     case 5:
159                             //线性表长度
160                             if( !list.elem[lp].L.elem ){
161                                     printf("线性表不存在！");
```

```
162                             break;
163                     }
164                     if( ListLength(list.elem[lp].L)==INFEASIBLE )
165                             printf("线性表不存在！");
166                     else
167                             printf("线性表长度为%d",ListLength(list.elem[lp].L));
168                     break;
169             case 6:
170                     //获取元素
171                     if( !list.elem[lp].L.elem ){
172                             printf("线性表不存在！");
173                             break;
174                     }
175                     printf("当前正在进行获取元素操作，请输入想要查询的元素在线性表中的逻辑次
                            序：");
176                     scanf("%d",&pos);
177                     if( GetElem(list.elem[lp].L,pos,val)==INFEASIBLE )
178                             printf("\n线性表不存在！");
179                     else if( GetElem(list.elem[lp].L,pos,val)==ERROR )
180                             printf("\n获取失败，请检查下标值是否合法！");
181                     else
182                             printf("\n您想要查询的元素值为%d",val);
183                     break;
184             case 7:
185                     //查找元素
186                     if( !list.elem[lp].L.elem ){
187                             printf("线性表不存在！");
188                             break;
189                     }
190                     printf("当前正在进行查找元素操作，请输入想要查询的值：");
191                     scanf("%d",&val);
192                     if( (pos=LocateElem(list.elem[lp].L,val))==INFEASIBLE ) //把返回值存在pos
                            中，避免重复调用
193                             printf("\n线性表不存在！");
194                     else if( pos!=ERROR )
195                             printf("\n查询成功！您需要的值在表中第%d个！",pos);
196                     else
197                             printf("\n您要查询的值不在表中！");
198                     break;
199             case 8:
200                     //获取前驱元素
201                     if( !list.elem[lp].L.elem ){
202                             printf("线性表不存在！");
203                             break;
204                     }
205                     printf("当前正在进行获取前驱元素操作，请输入想要查询前驱的元素：");
206                     scanf("%d",&val);
207                     if( PriorElem(list.elem[lp].L,val,pre)==INFEASIBLE )
208                             printf("\n线性表不存在！");
209                     else if( PriorElem(list.elem[lp].L,val,pre)==ERROR )
210                             printf("\n获取失败！请检查您要查询的元素是否在表中或为表头！");
211                     else
212                             printf("\n获取成功！%d的前驱元素为%d",val,pre);
213                     break;
214             case 9:
215                     //获取后继元素
216                     if( !list.elem[lp].L.elem ){
```

```
217                     printf("线性表不存在！");
218                     break;
219                 }
220                 printf("当前正在进行获取后继元素操作，请输入想要查询后继的元素：");
221                 scanf("%d",&val);
222                 if( PriorElem(list.elem[lp].L,val,pre)==INFEASIBLE )
223                     printf("\n线性表不存在！");
224                 else if( PriorElem(list.elem[lp].L,val,pre)==ERROR )
225                     printf("\n获取失败！请检查您要查询的元素是否在表中或为表尾！");
226                 else
227                     printf("\n获取成功！%d的后继元素为%d",val,pre);
228                 break;
229             case 10:
230                 //插入元素
231                 if( !list.elem[lp].L.elem ){
232                     printf("线性表不存在！");
233                     break;
234                 }
235                 printf("当前正在进行插入元素操作，请输入想要插入元素的逻辑次序和元素值\n")
                        ;
236                 printf("逻辑次序：");
237                 scanf("%d",&pos);
238                 printf("元素值：");
239                 scanf("%d",&ins);
240                 if( ListInsert(list.elem[lp].L,pos,ins)==OK )
241                     printf("\n插入成功！");
242                 else if( ListInsert(list.elem[lp].L,pos,ins)==INFEASIBLE )
243                     printf("\n线性表不存在");
244                 else
245                     printf("\n插入失败！请检查下标值是否合法");
246                 break;
247             case 11:
248                 //删除元素
249                 if( !list.elem[lp].L.elem ){
250                     printf("线性表不存在！");
251                     break;
252                 }
253                 printf("当前正在进行删除元素操作，请输入想要删除元素的逻辑次序：");
254                 scanf("%d",&pos);
255                 if( ListDelete(list.elem[lp].L,pos,ins)==OK )
256                     printf("\n删除成功！删除的元素为%d",ins);
257                 else if( ListDelete(list.elem[lp].L,pos,ins)==INFEASIBLE )
258                     printf("\n线性表不存在");
259                 else
260                     printf("\n删除失败！请检查下标值是否合法!");
261                 break;
262             case 12:
263                 //遍历线性表
264                 //这里需要先判断线性表是否存在
265                 if( !list.elem[lp].L.elem )
266                     printf("线性表不存在！");
267                 else{
268                     printf("线性表%d的元素依次是：\n",lp);
269                     ListTraverse(list.elem[lp].L);
270                 }
271                 break;
272             case 13:
```

```
273                     //线性表存储
274                     if( !list.elem[lp].L.elem ){
275                             printf("线性表不存在！");
276                             break;
277                     }
278                     printf("当前正在进行线性表存储操作，请输入想要生成的文件名：\n");
279                     scanf("%s",filename);
280                     if( SaveList(list.elem[lp].L,filename)==INFEASIBLE )
281                             printf("线性表不存在！");
282                     else
283                             printf("存储成功！");
284                     break;
285             case 14:
286                     //线性表读取
287                     printf("当前正在进行线性表读取操作，请输入想要读取的文件名(包括扩展名)：\n
                            ");
288                     scanf("%s",filename);
289                     if( LoadList(list.elem[lp].L,filename)==OK )
290                             printf("\n读取成功！");
291                     else if( LoadList(list.elem[lp].L,filename)==INFEASIBLE )
292                             printf("\n线性表不为空！不能进行读取操作！");
293                     else
294                             printf("\n读取失败！请检查文件是否存在！");
295                     break;
296             case 15:
297                     //最大连续子数组和
298                     int ret15;
299                     if( (ret15=MaxSubArray(list.elem[lp].L))==INFEASIBLE )
300                             printf("线性表不存在或为空！");
301                     else
302                             printf("最大连续子数组和为：%d",ret15);
303                     break;
304             case 16:
305                     //返回和为K的子数组数目
306                     int ret16,k;
307                     printf("当前正在进行求和为K的子数组数目的操作，请输入K的值：");
308                     scanf("%d",&k);
309                     if( (ret16=SubArrayNum(list.elem[lp].L,k))==INFEASIBLE )
310                             printf("\n线性表不存在或为空！");
311                     else
312                             printf("\n和为K的子数组数目为：%d",ret16);
313                     break;
314             case 17:
315                     //从小到大排序
316                     if( sortList(list.elem[lp].L)==INFEASIBLE )
317                             printf("线性表不存在或为空！");
318                     else
319                             printf("排序成功！");
320                     break;
321             case 18:
322                     //多线性表管理——添加新表
323                     if( list.length>=10 )
324                             printf("当前线性表数目已超过上限！");
325                     else{
326                             printf("当前正在进行添加线性表的操作，请输入想要建立新表的名称：\n
                                    ");
327                             scanf("%s",listname);
```

```
328                                     AddList(list ,listname);
329                                     printf("\n添加成功！");
330                             }
331                             break;
332                     case 19:
333                             //多线性表管理——移除表
334                             if( !list.elem )
335                                     printf("线性表组中没有线性表，先创建一个吧！");
336                             else{
337                                     int ret19;
338                                     printf("当前正在进行移除线性表的操作，请输入您要移除的线性表的名
                                            称：\n");
339                                     scanf("%s",listname);
340                                     if((ret19=RemoveList(list ,listname))==ERROR)
341                                             printf("\n没有找到您输入的线性表耶，请检查您是否输入错误！
                                                    ");
342                                     else{
343                                             printf("\n移除成功了！\n");
344                                             if( ret19<lp )
345                                                     lp--;
346                                             else if( ret19==lp ){
347                                                     printf("\n由于你删除了当前线性表，下次操作默认对第
                                                            一个线性表进行！");
348                                                     lp=1;
349                                             }
350                                     }
351                             }
352                             break;
353                     case 20:
354                             //多线性表管理——选择要操作的线性表
355                             printf("当前正在进行选择操作线性表的操作，请输入您要操作的线性表的名称：\n
                                    ");
356                             scanf("%s",listname);
357                             int ret20;
358                             if( (ret20=LocateList(list ,listname))==INFEASIBLE )
359                                     printf("\n线性表组中没有线性表！先创建一个吧！");
360                             else if( ret20==ERROR )
361                                     printf("\n没有找到您输入的线性表耶，请检查您是否输入错误！");
362                             else{
363                                     lp=ret20;
364                                     printf("\n定位成功！");
365                             }
366                             break;
367                     default:
368                             //输入op错误
369                             printf("请输入正确的选项！");
370                             break;
371             }
372             if( op!=0 ){
373                     printf("\n\n按任意键继续下一个操作\n\n");
374                     system("pause");
375                     cnt++;
376             }
377     }
378     printf("谢谢使用！您本次共进行了%d次操作！欢迎下次再来！\n",cnt);
379     Sleep(2000);
380     return 0;
```

```
381 }
382 /*****************************************/
383
384
385 /*****************************************************************
386 函数名称：InitList
387 初始条件：线性表L不存在
388 功能说明：如果线性表L不存在，操作结果是构造一个空的线性表，返回OK，
389 否则返回INFEASIBLE。
390 返回值类型：
391 *****************************************************************/
392 status InitList(SqList& L)
393 {
394     if(L.elem)                              //线性表已存在
395         return INFEASIBLE;          //不可行
396     L.elem=(ElemType*)malloc(sizeof(ElemType)*LIST_INIT_SIZE);
397     L.length=0;
398     L.listsize=LIST_INIT_SIZE;
399     return OK;
400 }
401
402 /*****************************************************************
403 函数名称：DestroyList
404 初始条件：线性表L已存在
405 功能说明：如果线性表L存在，该操作释放线性表的空间，使线性表成为未初
406 始化状态，返回OK；否则对于一个未初始的线性表，是不能进行销毁操作的，
407 返回INFEASIBLE。
408 返回值类型：  status
409 *****************************************************************/
410 status DestroyList(SqList& L)
411 {
412     if(L.elem){                             //若线性表存在
413         free(L.elem);
414         //相关数值清空
415         L.elem=NULL;
416         L.length=0;
417         L.listsize=0;
418     }
419     else                                            //线性表不存在
420         return INFEASIBLE;          //不可行
421     return OK;
422 }
423
424 /*****************************************************************
425 函数名称：ClearList
426 初始条件：线性表L已存在
427 功能说明：若线性表L不存在，返回INFEASIBLE。否则清空线性表L，返回OK；
428 返回值类型：  status
429 *****************************************************************/
430 status ClearList(SqList& L)
431 {
432     if( L.elem ){                           //若线性表存在
433         L.length=0;
434     }
435     else                                            //线性表不存在
436         return INFEASIBLE;          //不可行
437     return OK;
```

```
438 }
439
440 /**************************************************************************
441 函数名称：ListEmpty
442 初始条件：线性表L已存在
443 功能说明：若线性表L不存在，则返回INFEASIBLE；若线性表L长度为0，则返
444 回TRUE；不为0则返回FALSE。
445 返回值类型： status
446 **************************************************************************/
447 status ListEmpty(SqList L)
448 {
449     if( !L.elem )                       //若线性表不存在
450         return INFEASIBLE;      //不可行
451     if( L.length==0 )           //线性表为空
452         return TRUE;
453     else                                //线性表不为空
454         return FALSE;
455 }
456
457 /**************************************************************************
458 函数名称：ListLength
459 初始条件：线性表L已存在
460 功能说明：若线性表L不存在，返回INFEASIBLE；否则返回线性表L的长度。
461 返回值类型： status
462 **************************************************************************/
463 status ListLength(SqList L)
464 {
465     if( !L.elem )                       //若线性表不存在
466         return INFEASIBLE;      //不可行
467     else
468         return L.length;
469 }
470
471 /**************************************************************************
472 函数名称：GetElem
473 初始条件：线性表L已存在
474 功能说明：若线性表L不存在，返回INFEASIBLE；若i<1或者i超过线性表L的长
475 度，则返回ERROR；若获取成功，则将值赋给e，并返回OK。
476 返回值类型： status
477 **************************************************************************/
478 status GetElem(SqList L,int i,ElemType &e)
479 {
480     if( !L.elem )                       //若线性表不存在
481         return INFEASIBLE;      //不可行
482     if( i<1||i>L.length )       //若下标值不合法
483         return ERROR;           //出错
484     e=L.elem[i-1];
485     return OK;
486 }
487
488 /**************************************************************************
489 函数名称：LocateElem
490 初始条件：线性表L已存在
491 功能说明：若线性表L不存在，返回INFEASIBLE；若没有找到指定的元素e，则
492 查找失败，返回ERROR；若查找成功，则返回元素逻辑序号i。
493 返回值类型： int
494 **************************************************************************/
```

```
495 int LocateElem(SqList L,ElemType e)
496 {
497     if( !L.elem )                       //若线性表不存在
498         return INFEASIBLE;      //不可行
499     int pos=-1;                         //所求元素下标，初始化为-1
500     for( int i=0 ; i<L.length ; i++ ){
501         if( L.elem[i]==e ){
502             pos=i;
503             break;
504         }
505     }
506     if( pos==-1 )                       //pos仍未原值，即未找到
507         return ERROR;           //出错
508     return pos+1;
509 }
510
511 /*****************************************************************
512 函数名称：PriorElem
513 初始条件：线性表L已存在
514 功能说明：若线性表L不存在，返回INFEASIBLE；若没有找到指定元素e的前驱，
515 则查找失败，返回ERROR；若查找成功，则将值赋给pre，并返回OK。
516 返回值类型：  status
517 *****************************************************************/
518 status PriorElem(SqList L,ElemType e,ElemType &pre)
519 {
520     if( !L.elem )                       //若线性表不存在
521         return INFEASIBLE;      //不可行
522     for( int i=0 ; i<L.length ; i++ ){
523         if( L.elem[i]==e ){     //找到该值
524             if( i==0 )          //所求值对应表中首元素，无前驱
525                 return ERROR;
526             pre=L.elem[i-1];
527             return OK;          //元素存在且不为首元素
528         }
529     }
530     return ERROR;                       //循环结束，即未找到该元素
531 }
532 //注：这里第一次写时没有考虑首元素没有前驱，于是在修改时直接进行特判所查
533 //元素是否为首元素，还有一种改法与下面查找后继所用的方法类似，即在循环时
534 //直接从i=1开始循环。
535
536 /*****************************************************************
537 函数名称：NextElem
538 初始条件：线性表L已存在
539 功能说明：若线性表L不存在，返回INFEASIBLE；若没有找到指定元素e的后继，
540 则查找失败，返回ERROR；若查找成功，则将值赋给next，并返回OK。
541 返回值类型：  status
542 *****************************************************************/
543 status NextElem(SqList L,ElemType e,ElemType &next)
544 {
545     if( !L.elem )                       //若线性表不存在
546         return INFEASIBLE;      //不可行
547     for( int i=0 ; i<L.length-1 ; i++ ){
548         //这里循环到L.length-2即可，因为即使L.elem[L.length-1]==e也没有后继
549         if( L.elem[i]==e ){     //找到该值
550             next=L.elem[i+1];
551             return OK;
```

```
552             }
553         }
554         return ERROR;                           //循环结束，即未找到该元素
555     }
556
557     /**************************************************************
558     函数名称：ListInsert
559     初始条件：线性表L已存在
560     功能说明：如果线性表L不存在，返回INFEASIBLE；否则在线性表L的第i个元素
561     前插入新的元素e，插入成功返回OK，失败返回ERROR。
562     返回值类型：  status
563     **************************************************************/
564     status ListInsert(SqList &L,int i,ElemType e)
565     {
566         if( !L.elem )                                       //若线性表不存在
567             return INFEASIBLE;                          //不可行
568         if( L.length==L.listsize ){             //线性表已满
569             L.elem=(ElemType*)realloc(L.elem,sizeof(ElemType)*(L.listsize+LISTINCREMENT));
570             L.listsize+=LISTINCREMENT;
571         }
572         if( i<1||i>L.length+1)                      //若下标不合法
573             return ERROR;                           //出粗
574         for( int j=L.length ; j>=i ; j-- )          //移动元素
575             L.elem[j]=L.elem[j-1];
576         L.elem[i-1]=e;
577         L.length++;
578         return OK;
579     }
580
581     /**************************************************************
582     函数名称：ListDelete
583     初始条件：线性表L已存在
584     功能说明：若线性表L不存在，返回INFEASIBLE；否则删除线性表L的第i个元
585     素，若删除成功则将删除的值赋给e并返回OK，若删除失败则返回ERROR。
586     返回值类型：  status
587     **************************************************************/
588     status ListDelete(SqList &L,int i,ElemType &e)
589     {
590         if( !L.elem )                           //若线性表不存在
591             return INFEASIBLE;          //不可行
592         if( i<1||i>L.length)            //若下标不合法
593             return ERROR;               //出错
594         e=L.elem[i-1];
595         for( int j=i ; j<L.length ; j++ )           //移动元素
596             L.elem[j-1]=L.elem[j];
597         L.length--;
598         return OK;
599     }
600
601     /**************************************************************
602     函数名称：ListTraverse
603     初始条件：线性表L已存在
604     功能说明：若线性表L不存在，返回INFEASIBLE；否则输出线性表的每一个元
605     素，并返回OK。
606     返回值类型：  status
607     **************************************************************/
608     status ListTraverse(SqList L)
```

```
{
    if( !L.elem )                       //若线性表不存在
        return INFEASIBLE;      //不可行
    for( int i=0 ; i<L.length ; i++ ){
        printf("%d",L.elem[i]);
        if( i!=L.length-1 )
        printf(" ");
    }
    return OK;
}

/**************************************************************************
函数名称：SaveList
初始条件：线性表L已存在
功能说明：如果线性表L不存在，返回INFEASIBLE；否则将线性表L的全部元素
写入到文件名为FileName的文件中，返回OK。
返回值类型：  status
**************************************************************************/
status  SaveList(SqList L,char FileName[])
{
    if( !L.elem )                       //若线性表不存在
        return INFEASIBLE;      //不可行
    FILE *fp;
    fp=fopen(FileName,"w+");
    fprintf(fp,"%d ",L.length);                 //元素个数和表长都写入文件，方便操作
    fprintf(fp,"%d ",L.listsize);
    for( int i=0 ; i<L.length ; i++ )
        fprintf(fp,"%d ",L.elem[i]);
    fclose(fp);
    return OK;
}

/**************************************************************************
函数名称：LoadList
初始条件：线性表L为空
功能说明：如果线性表L存在，表示L中已经有数据，读入数据会覆盖原数据造
成数据丢失，返回INFEASIBLE；否则将文件名为FileName的数据读入到线性表
L中，返回OK。本实验不考虑用追加的方式读入文件数据追加到现有线性表中。
返回值类型：  status
**************************************************************************/
status  LoadList(SqList &L,char FileName[])
{
    if( L.elem )                        //若线性表不为空
        return INFEASIBLE;      //不可行
    FILE *fp;
    fp=fopen(FileName,"r+");
    if( fp==NULL )                      //文件不存在
        return ERROR;
    fscanf(fp,"%d %d",&L.length,&L.listsize);           //先读入元素个数和表长
    L.elem=(ElemType*)malloc(sizeof(ElemType)*L.listsize);
    for( int i=0 ; i<L.length ; i++ )
        fscanf(fp,"%d",&L.elem[i]);
    fclose(fp);
    return OK;
}

/**************************************************************************
```

```
666  函数名称：AddList
667  初始条件：想要添加的线性表L为空
668  功能说明：Lists是一个以顺序表形式管理的线性表的集合，在集合中增加一
669  个新的空线性表。增加成功返回OK，否则返回ERROR。
670  返回值类型：  status
671  ********************************************************************/
672  status AddList(LISTS &Lists,char ListName[])
673  {
674      if( Lists.length>=10 )        //超出Lists最大范围
675          return ERROR;             //出错
676          //初始化操作
677          Lists.length++;
678      Lists.elem[Lists.length].L.length=0;
679      Lists.elem[Lists.length].L.listsize=0;
680      int i;
681      for( i=0 ; ListName[i]!='\0' ; i++ )
682          Lists.elem[Lists.length].name[i]=ListName[i];
683      Lists.elem[Lists.length].name[i]='\0';
684      return OK;
685  }
686
687  /********************************************************************
688  函数名称：RemoveList
689  初始条件：Lists存在
690  功能说明：Lists是一个以顺序表形式管理的线性表的集合，在集合中查找名
691  称为ListName的线性表，有则删除，返回OK，无则返回ERROR。
692  返回值类型：  status
693  ********************************************************************/
694  status RemoveList(LISTS &Lists,char ListName[])
695  {
696          if( !Lists.length )               //Lists为空
697                  return INFEASIBLE;        //不可行
698      for( int i=1 ; i<=Lists.length ; i++ ){
699          if( !strcmp(ListName,Lists.elem[i].name) ){
700              //free(Lists.elem[i].L.elem);free(Lists.elem[i].name);
701              for( int j=i ; j<Lists.length ; j++ )
702                  Lists.elem[j]=Lists.elem[j+1];
703              Lists.elem[Lists.length].L.elem=NULL;
704              Lists.elem[Lists.length].L.listsize=0;
705              Lists.elem[Lists.length].L.length=0;
706              Lists.length--;
707              return i;
708          }
709      }
710      return ERROR;
711  }
712
713  /********************************************************************
714  函数名称：LocateList
715  初始条件：Lists存在
716  功能说明：Lists是一个以顺序表形式管理的线性表的集合，在集合中查找名称
717  为ListName的线性表，有则返回线性表的逻辑序号，无则返回0。
718  返回值类型：  int
719  ********************************************************************/
720  int LocateList(LISTS Lists,char ListName[])
721  {
722          if( !Lists.length )               //Lists为空
```

```
723                 return INFEASIBLE;        //不可行
724     for( int i=1 ; i<=Lists.length ; i++ ){
725         if( !strcmp(ListName,Lists.elem[i].name) )
726             return i;
727     }
728     return 0;
729 }
730 
731 /*****************************************************************
732 函数名称：MaxSubArray
733 初始条件：线性表L存在且非空
734 功能说明：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），
735 返回其最大和
736 返回值类型：  int
737 *****************************************************************/
738 int MaxSubArray( SqList &L ){
739         if( !L.elem||L.length==0 )              //线性表不存在或为空
740                 return INFEASIBLE;                      //不可行
741         int sum=0;
742         int ans=L.elem[0];
743         for( int i=0 ; i<L.length ; i++ ){      //在线处理
744                 sum+=L.elem[i];
745                 ans=(sum>ans)?sum:ans;          //模拟max函数
746                 if( sum<0 )                                     //sum小于0则舍弃
747                         sum=0;
748         }
749         return ans;
750 }
751 
752 /*****************************************************************
753 函数名称：SubArrayNum
754 初始条件：线性表L存在且非空
755 功能说明：返回该数组中和为k的连续子数组的个数
756 返回值类型：  int
757 *****************************************************************/
758 int SubArrayNum( SqList &L,int k ){
759         if( !L.elem||L.length==0 )              //线性表不存在或为空
760                 return INFEASIBLE;                      //不可行
761         int cnt=0;
762         for( int start=0 ; start<L.length ; start++ ){
763                 int sum=0;
764                 for( int end=start ; end>=0 ; end-- ){
765                         sum+=L.elem[end];
766                         if( sum==k )
767                                 cnt++;
768                 }
769         }
770         return cnt;
771 }
772 
773 /*****************************************************************
774 函数名称：sortList
775 初始条件：线性表L存在且非空
776 功能说明：将L由小到大排序
777 返回值类型：  status
778 *****************************************************************/
779 status sortList( SqList &L ){
```

```
780         if( !L.elem||L.length==0 )                  //线性表不存在或为空
781                 return INFEASIBLE;                          //不可行
782         for( int i=0 ; i<L.length ; i++ ){          //BubbleSort
783                 for( int j=1 ; j<L.length-i ; j++ ){
784                         if( L.elem[j]<L.elem[j-1] ){
785                                 int t=L.elem[j];
786                                 L.elem[j]=L.elem[j-1];
787                                 L.elem[j-1]=t;
788                         }
789                 }
790         }
791         return OK;
792 }
```

附录 B 基于邻接表图实现的源程序

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<string.h>
#include<windows.h>

/*************************预定义*************************/
#define TRUE 1                                  //判断正确
#define FALSE 0                                 //判断错误
#define OK 1                                    //成功
#define ERROR 0                                 //逻辑错误
#define INFEASIBLE -1                           //不可行错误
#define OVERFLOW -2                             //溢出错误
#define MAX_VERTEX_NUM 20                       //最大结点数

typedef int status;                                                 //状态类型
typedef int KeyType;                                              //关键字类型
typedef enum {DG,DN,UDG,UDN} GraphKind;         //图类型

typedef struct {
      KeyType  key;
      char others[20];
} VertexType;                                   //顶点类型定义


typedef struct ArcNode {          //表结点类型定义
          int adjvex;                           //顶点位置编号
          struct ArcNode  *nextarc;             //下一个表结点指针
} ArcNode;

typedef struct VNode{                           //头结点及其数组类型定义
          VertexType data;                      //顶点信息
          ArcNode *firstarc;                    //指向第一条弧
        } VNode,AdjList[MAX_VERTEX_NUM];

typedef  struct {                               //邻接表的类型定义
     AdjList vertices;                          //头结点数组
     int vexnum,arcnum;                         //顶点数、弧数
     int  kind;                                 //图的类型
   } ALGraph;

typedef struct{                                 //线性表的集合类型定义
     struct {
                char name[30];                  //表名
          ALGraph G;                            //线性表内容
     } elem[11];
     int length;                                //表长
}LISTS;

//全局变量
char flag[100]={0};                             //判断结点是否遍历过
int adjvex[100];                                //存储结点位序
int nums=0;                                             //存储节点数目
```

```
54  int Min=100;                                        //最短路径
55
56  /*****************************************************/
57
58  /*************************函数表*************************/
59  status InsertArc(ALGraph &G,KeyType v,KeyType w);
60  status CreateCraph(ALGraph &G,VertexType V[],KeyType VR[][2]);
61  status DestroyGraph(ALGraph &G);
62  int LocateVex(ALGraph G,KeyType u);
63  status PutVex(ALGraph &G,KeyType u,VertexType value);
64  int FirstAdjVex(ALGraph G,KeyType u);
65  int NextAdjVex(ALGraph G,KeyType v,KeyType w);
66  status InsertVex(ALGraph &G,VertexType v);
67  status DeleteVex(ALGraph &G,KeyType v);
68  status DeleteArc(ALGraph &G,KeyType v,KeyType w);
69  void visit(VertexType v);
70  void dfs(ALGraph &G,int v,void (*visit)(VertexType));
71  status DFSTraverse(ALGraph &G,void (*visit)(VertexType));
72  status BFSTraverse(ALGraph &G,void (*visit)(VertexType));
73  status SaveGraph(ALGraph G, char FileName[]);
74  status LoadGraph(ALGraph &G, char FileName[]);
75  status AddGraph(LISTS &graph,char graphname[]);
76  int DeleteGraph(LISTS &graph,char graphname[]);
77  int ChooseGraph(LISTS &graph,char graphname[]);
78  status VerticesSetLessThanK(ALGraph &G,KeyType v,int k);
79  void dfs_lessk(ALGraph &G,int k,int adj);
80  int ShortestPathLength(ALGraph &G,KeyType v,KeyType w);
81  void dfs_connect(ALGraph &G,char flag[],int v);
82  int ConnectedComponentsNums(ALGraph &G);
83  /*****************************************************/
84
85  /*************************主函数*************************/
86
87  int main(){
88
89      /*****************************************************************/
90      /*                                                 程序框设置
                        */
91      /*****************************************************************/
92
93      system("mode con cols=82 lines=31 ");    //更改程序框大小
94      system("title 线性表：链式存储结构");    //更改程序框名称
95      system("color 1F");                                          //背景蓝色，字体亮白色
96
97      /*****************************************************************/
98      /*                                                 初始化
                            */
99      /*****************************************************************/
100
101     int op=1;                          //用户操作选项
102     char filename[50];          //存储文件名
103     char graphname[50];         //存储图名
104     int lp=1;                          //记录当前处理的图指针
105     int cntnum=0;               //记录操作数目
106     int kv,kw,ku;               //记录关键字值
107     VertexType input[100];
108     int VR[100][2];             //弧数组
```

```
109
110         //多图管理
111         LISTS graph;              //线性表集合管理多图
112         graph.length=0;           //长度初始化
113
114         //图初始化
115         for( int i=0 ; i<11 ; i++ ){
116                 graph.elem[i].G.arcnum=0;                 //结点与弧数目归零
117                 graph.elem[i].G.vexnum=0;
118                 graph.elem[i].G.kind=0;
119         }
120
121         //名称初始化
122         char defaultname[50]="初始图";            //默认名称
123         for( int i=0 ; defaultname[i]!='\0' ; i++ )
124                 graph.elem[1].name[i]=defaultname[i];
125
126         /*****************************************************************/
127         /*                                                菜单主体
                        */
128         /*****************************************************************/
129
130         while(op){
131                 system("cls");
132             printf("\n\n");
133                 printf("           \t\t\t\t图菜单\t\t\n\n");
134                 printf(" \t可在最多10个图进行多图操作，二叉树尚未初始化，名称默认为“图”\n");
135                 printf(" ------------------------------------------------------------------------------\n"
                        );
136                 printf(" *1. 创建图          \t\t\t7.  插入顶点\t\t\t\t*\n");
137                 printf(" *2. 销毁图        \t\t\t8.  删除顶点\t\t\t\t*\n");
138                 printf(" *3. 查找顶点          \t\t\t9.  插入弧 \t\t\t\t*\n");
139                 printf(" *4. 顶点赋值          \t\t\t10. 删除弧\t\t\t\t*\n");
140                 printf(" *5. 获得第一邻接点         \t\t11. 深度优先搜索\t\t\t*\n");
141                 printf(" *6. 获得下一邻接点            \t\t12. 广度优先搜索\t\t\t*\n");
142                 printf(" ------------------------------------------------------------------------------\n"
                        );
143                 printf(" *13.图文件存储           \t\t14. 图文件读取\t\t\t\t*\n");
144                 printf(" *15.距离小于k的顶点集合 \t\t16. 顶点间最短路径的长度\t\t*\n");
145                 printf(" *17.图的连通分量         \t\t18. 添加图\t\t\t\t*\n");
146                 printf(" *19.移除图           \t\t\t20. 选择要操作的图\t\t\t*\n");
147                 printf(" *0. 退出\t\t\t\n");
148                 printf(" ------------------------------------------------------------------powered by 严浩洋\n
                        ");
149                 printf("当前操作的图：图%d：%s\n\n",lp,graph.elem[lp].name);
150                 printf("请输入数字0~20以选择你的操作:");
151                 scanf("%d",&op);//选择op的值,用于switch
152                 printf("\n");
153                 int i=0;
154                 switch(op){
155
156                         case 0:
157                                 //结束操作
158                                 break;
159
160                         case 1:
161                                 //创建图
```

```
162                     if( graph.elem[lp].G.arcnum||graph.elem[lp].G.vexnum ){
163                             printf("建立失败！图已存在！");
164                             break;
165                     }
166
167                     //输入顶点数组
168                     printf("当前正在进行创建图操作，请输入图的顶点与弧数组：\n");
169
170                     do{
171                             scanf("%d %s",&input[i].key,input[i].others);
172                     }while(input[i++].key!=-1);
173
174                     //超过二十个点
175                     if( i>20 ){
176                             printf("\n创建失败！超过二十个点！");
177                             break;
178                     }
179
180                     //空图
181                     else if( i<=1 ){
182                             printf("\n创建失败！结点数为0！");
183                             break;
184                     }
185
186                     //输入弧数组
187                     i=0;
188                     do{
189                             scanf("%d %d",&VR[i][0],&VR[i][1]);
190                     }while(VR[i++][0]!=-1);
191
192                     if( CreateCraph(graph.elem[lp].G,input,VR)==ERROR )
193                             printf("\n创建失败！请检查关键字是否重复！");
194                     else{
195                             printf("\n创建成功！");
196                             if( graph.elem[lp].G.kind==0 )
197                                     graph.length++;
198                     }
199                     break;
200
201             case 2:
202                     //销毁图
203                     if( !graph.elem[lp].G.vexnum ){
204                             printf("销毁失败！图不存在！");
205                             break;
206                     }
207                     DestroyGraph(graph.elem[lp].G);
208                     printf("销毁成功！");
209                     graph.length--;
210                     break;
211
212             case 3:
213                     //查找顶点
214                     if( graph.elem[lp].G.vexnum==0 ){
215                             printf("该图为空，无法进行操作！");
216                             break;
217                     }
218                     printf("当前正在进行查找顶点操作，请输入要查找顶点的关键字：");
```

```
219                     scanf("%d",&ku);
220                     int ret3;
221                     if( (ret3=LocateVex(graph.elem[lp].G,ku))==-1 )
222                             printf("\n未查询到您想要的结点！");
223                     else{
224                             printf("\n您查找结点的位序为%d,",ret3);
225                             printf("值为：%d %s",graph.elem[lp].G.vertices[ret3].data.key,
                                    graph.elem[lp].G.vertices[ret3].data.others);
226                     }
227                     break;
228
229             case 4:
230                     //顶点赋值
231                     if( graph.elem[lp].G.vexnum==0 ){
232                             printf("该图为空，无法进行操作！");
233                             break;
234                     }
235                     printf("当前正在进行顶点赋值操作，请输入要赋值顶点的关键字及要赋的值：");
236                     scanf("%d %d %s",&ku,&input[0].key,input[0].others);
237                     if( PutVex(graph.elem[lp].G,ku,input[0])==ERROR )
238                             printf("\n赋值失败！请检查关键字是否重复或查询结点是否存在！");
239                     else
240                             printf("\n赋值成功！");
241                     break;
242
243             case 5:
244                     //获得第一邻接点
245                     if( graph.elem[lp].G.vexnum==0 ){
246                             printf("该图为空，无法进行操作！");
247                             break;
248                     }
249                     printf("当前正在进行获得第一邻接点操作，请输入要查找结点的关键字：");
250                     scanf("%d",&ku);
251                     int ret5;
252                     if( (ret5=FirstAdjVex(graph.elem[lp].G,ku))==-1 )
253                             printf("\n您要查找的结点不存在！");
254                     else{
255                             printf("\n您要查找的结点的第一临界点位序为：%d,值为：",ret5);
256                             printf("%d,%s",graph.elem[lp].G.vertices[ret5].data.key,graph.elem
                                    [lp].G.vertices[ret5].data.others);
257                     }
258                     break;
259
260             case 6:
261                     //获得下一邻接点
262                     if( graph.elem[lp].G.vexnum==0 ){
263                             printf("该图为空，无法进行操作！");
264                             break;
265                     }
266                     printf("当前正在进行获得下一邻接点操作，请输入要查找结点的关键字：");
267                     scanf("%d %d",&kv,&kw);
268                     int ret6;
269                     if( (ret6=NextAdjVex(graph.elem[lp].G,kv,kw))==-1 )
270                             printf("\n获取失败！请检查结点是否存在或是否无下一邻接点！");
271                     else{
272                             printf("\n您要查找的结点的下一临界点位序为：%d,值为：",ret6);
```

```
273                             printf("\n%d,%s",graph.elem[lp].G.vertices[ret6].data.key,graph.
                                    elem[lp].G.vertices[ret6].data.others);
274                     }
275                     break;
276
277             case 7:
278                     //插入顶点
279                     if( graph.elem[lp].G.vexnum==0 ){
280                             printf("该图为空，无法进行操作！");
281                             break;
282                     }
283                     if( graph.elem[lp].G.vexnum>=20 ){
284                             printf("结点数已达到最大，无法插入！");
285                             break;
286                     }
287                     printf("当前正在进行插入结点操作，请输入要插入结点的值：");
288                     scanf("%d %s",&input[0].key,input[0].others);
289                     if( InsertVex(graph.elem[lp].G,input[0])==ERROR )
290                             printf("\n插入失败！请检查关键字是否重复！");
291                     else
292                             printf("插入成功！");
293                     break;
294
295             case 8:
296                     //删除顶点
297                     if( graph.elem[lp].G.vexnum==0 ){
298                             printf("该图为空，无法进行操作！");
299                             break;
300                     }
301                     if( graph.elem[lp].G.vexnum==1 ){
302                             printf("该图只有一个结点，无法删除！");
303                             break;
304                     }
305                     printf("当前正在进行删除结点操作，请输入要删除结点的关键字：");
306                     scanf("%d",&ku);
307                     if( DeleteVex(graph.elem[lp].G,ku)==ERROR )
308                             printf("\n删除失败！请检查结点是否存在！");
309                     else
310                             printf("\n删除成功！");
311                     break;
312
313             case 9:
314                     //插入弧
315                     if( graph.elem[lp].G.vexnum==0 ){
316                             printf("该图为空，无法进行操作！");
317                             break;
318                     }
319                     printf("当前正在进行插入弧操作，请输入要插入弧端点的关键字：");
320                     scanf("%d %d",&kv,&kw);
321                     if( InsertArc(graph.elem[lp].G,kv,kw)==ERROR )
322                             printf("\n插入失败！请检查顶点是否存在或顶点之间已存在弧");
323                     else
324                             printf("\n插入成功！");
325                     break;
326
327             case 10:
328                     //删除弧
```

```
329                     if( graph.elem[lp].G.vexnum==0 ){
330                             printf("该图为空，无法进行操作！");
331                             break;
332                     }
333                     printf("当前正在进行删除弧操作，请输入要删除弧端点的关键字：");
334                     scanf("%d %d",&kv,&kw);
335                     if( DeleteArc(graph.elem[lp].G,kv,kw)==ERROR )
336                             printf("\n删除失败！请检查顶点是否存在或顶点之间不存在弧");
337                     else
338                             printf("\n删除成功！");
339                     break;
340
341             case 11:
342                     //深度优先搜索
343                     if( graph.elem[lp].G.vexnum==0 ){
344                             printf("该图为空，无法进行操作！");
345                             break;
346                     }
347                     for( int i=0 ; i<100 ; i++ )
348                             flag[i]=0;
349                     printf("遍历结果为：\n");
350                     DFSTraverse(graph.elem[lp].G,visit);
351                     break;
352
353             case 12:
354                     //广度优先搜索
355                     if( graph.elem[lp].G.vexnum==0 ){
356                             printf("该图为空，无法进行操作！");
357                             break;
358                     }
359                     printf("遍历结果为：\n");
360                     BFSTraverse(graph.elem[lp].G,visit);
361                     break;
362
363             case 13:
364                     //存储图
365                     if( graph.elem[lp].G.vexnum==0 ){
366                             printf("该图为空，无法进行操作！");
367                             break;
368                     }
369                     printf("当前正在进行存储图操作，请输入要存储的文件名：");
370                     scanf("%s",filename);
371                     SaveGraph(graph.elem[lp].G,filename);
372                     printf("\n存储成功！");
373                     break;
374
375             case 14:
376                     //读取图
377                     if( graph.elem[lp].G.vexnum ){
378                             printf("该图不为空，无法进行操作！");
379                             break;
380                     }
381                     printf("当前正在进行图的读取操作，请输入文件名：\n");
382                     scanf("%s",filename);
383                     LoadGraph(graph.elem[lp].G,filename);
384                     if( graph.elem[lp].G.kind==0 )
385                             graph.length++;
```

```
386                     printf("\n读取成功！");
387                     break;
388
389             case 15:
390                     //距离小于k的顶点集合
391                     if( !graph.elem[lp].G.vexnum ){
392                             printf("该图为空，无法进行操作！");
393                             break;
394                     }
395                     printf("当前正在进行返回距离小于k的顶点集合操作，请输入k和起点关键字：");
396                     int k;
397                     nums=0;
398                     for( int i=0 ; i<100 ; i++ ){
399                             adjvex[i]=-1;
400                             flag[i]=0;
401                     }
402                     scanf("%d %d",&k,&ku);
403                     if( VerticesSetLessThanK(graph.elem[lp].G,ku,k)==ERROR ){
404                             printf("\n查找失败！请检查结点是否存在！");
405                             break;
406                     }
407                     else{
408                             printf("\n距离小于k的顶点为：\n");
409                             for( int i=0 ; adjvex[i]!=-1 ; i++ )
410                                     printf("%d,%s\n",graph.elem[lp].G.vertices[adjvex[i]].data
                                            .key,graph.elem[lp].G.vertices[adjvex[i]].data.others
                                            );
411                     }
412                     break;
413
414             case 16:
415                     //顶点间最短路径长度
416                     if( !graph.elem[lp].G.vexnum ){
417                             printf("该图为空，无法进行操作！");
418                             break;
419                     }
420                     int ret16;
421                     for( int i=0 ; i<100 ; i++ )
422                             flag[i]=0;
423                     printf("当前正在进行返回顶点间最短路径长度操作操作，请输入起点和终点关键
                            字：");
424                     scanf("%d %d",&kv,&kw);
425                     if( (ret16=ShortestPathLength(graph.elem[lp].G,kv,kw))==ERROR )
426                             printf("\n查找失败！请检查结点是否存在！");
427                     else if( ret16==10000 )
428                             printf("\n两点之间不存在路径！");
429                     else
430                             printf("\n最短路径长度为：%d",ret16);
431                     break;
432
433             case 17:
434                     //图的连通分量
435                     if( !graph.elem[lp].G.vexnum ){
436                             printf("该图为空，无法进行操作！");
437                             break;
438                     }
```

```
439                         printf("当前图的连通分量为：%d",ConnectedComponentsNums(graph.elem[lp].G))
                                ;
440                         break;
441
442                 case 18:
443                         //添加图
444                         if( graph.length>=10 ){
445                                 printf("当前图的数目已达上限，无法添加！");
446                                 break;
447                         }
448                         printf("当前正在进行添加图操作，请输入图名称：");
449                         scanf("%s",graphname);
450                         if( AddGraph(graph,graphname)==ERROR )
451                                 printf("\n当前名称已存在，换一个名字吧！");
452                         else
453                                 printf("\n添加成功！");
454                         break;
455
456                 case 19:
457                         //移除图
458                         if( graph.length<=0 ){
459                                 printf("当前已没有图，无法进行移除操作！");
460                                 break;
461                         }
462                         int ret19;
463                         printf("当前正在进行移除图操作，请输入移除图的名称：");
464                         scanf("%s",graphname);
465                         if( (ret19=DeleteGraph(graph,graphname))==0 )
466                                 printf("\n删除失败！请检查图是否存在！");
467                         else{
468                                 printf("\n删除成功！");
469                                 if( ret19==lp ){
470                                         printf("由于您删除了当前操作的图，下次操作默认对图1进行！"
                                                );
471                                         lp=1;
472                                 }
473                         }
474                         break;
475
476                 case 20:
477                         //选择要操作的图
478                         printf("当前正在进行选择图操作，请输入要选择的图的名称：");
479                         scanf("%s",graphname);
480                         int ret20;
481                         if( (ret20=ChooseGraph(graph,graphname))==0 )
482                                 printf("\n选择失败！未找到您输入的图！");
483                         else{
484                                 lp=ret20;
485                                 printf("\n选择成功！即将对图%d进行操作！",lp);
486                         }
487                         break;
488         }
489         if( op!=0 ){
490                 printf("\n\n按任意键继续下一个操作\n\n");
491                 system("pause");
492                 cntnum++;
493         }
```

```
494             }
495             printf("谢谢使用！您本次共进行了%d次操作！欢迎下次再来！\n",cntnum);
496             Sleep(2000);
497             return 0;
498     }
499
500     /*********************************************************/
501
502     /*********************************************************
503     函数名:    CreateCraph
504     初始条件：无向图G不存在，给定结点与邻接数组
505     函数功能：根据顶点序列V和关系对序列VR构造一个无向图G（要
506     求无向图G中各顶点关键字具有唯一性，结点数目不多于MAX_VERT
507     EX_NUM）。
508     返回类型：status
509     辅助函数：插入弧函数
510     *********************************************************/
511     status InsertArc(ALGraph &G,KeyType v,KeyType w)
512     {
513         int t=G.vexnum,a=-1,b=-1;                    //结点数与v和w的下标
514
515         //判断v和w是否存在
516         for( int i=0 ; i<t ; i++ ){
517             if( G.vertices[i].data.key==v ){
518                 a=i;
519                 break;
520             }
521         }
522         if(a==-1)
523             return ERROR;
524         for( int i=0 ; i<t ; i++ ){
525             if( G.vertices[i].data.key==w ){
526                 b=i;
527                 break;
528             }
529         }
530         if(b==-1)
531             return ERROR;
532
533         //判断原先是否已存在相应的边
534         ArcNode *arc=G.vertices[a].firstarc;
535         while(arc){
536             if( arc->adjvex==b )
537                 return ERROR;
538             arc=arc->nextarc;
539         }
540
541         //插入弧（双向）
542         G.arcnum++;
543         ArcNode *insert=(ArcNode*)malloc(sizeof(ArcNode));
544         insert->adjvex=b;
545         insert->nextarc=G.vertices[a].firstarc;
546         G.vertices[a].firstarc=insert;
547         ArcNode *insert2=(ArcNode*)malloc(sizeof(ArcNode));
548         insert2->adjvex=a;
549         insert2->nextarc=G.vertices[b].firstarc;
550         G.vertices[b].firstarc=insert2;
```

```
551     return OK;
552 }
553
554 status CreateCraph(ALGraph &G,VertexType V[],KeyType VR[][2])
555 {
556     //计算边数和弧数
557     int vexnum=0,arcnum=0;
558     for( int i=0 ; V[i].key!=-1 ; i++ )
559         vexnum++;
560     for( int i=0 ; VR[i][0]!=-1 ; i++ )
561         arcnum++;
562
563     //判断关键字重复
564     char bull[1000]={0};
565     for( int i=0 ; i<vexnum ; i++ ){
566         bull[V[i].key]++;
567         if( bull[V[i].key]>1 )
568             return ERROR;
569     }
570
571     //特判：空图
572     if( vexnum==0 )
573         return ERROR;
574     //特判：超过二十个点
575     if( vexnum>20 )
576         return ERROR;
577
578     //判断弧没有错误
579     for( int i=0 ; i<arcnum ; i++ )
580         if( !bull[VR[i][0]]||!bull[VR[i][1]] )
581             return ERROR;
582
583     //更改图相应值
584     G.vexnum=vexnum;
585     for( int i=0 ; i<vexnum ; i++ ){
586         G.vertices[i].data=V[i];
587         G.vertices[i].firstarc=NULL;
588     }
589
590     //建立边
591     for( int i=0 ; i<arcnum ; i++ ){
592         InsertArc(G,VR[i][0],VR[i][1]);
593     }
594     return OK;
595 }
596
597 /*********************************************************
598 函数名:   DestroyGraph
599 初始条件：无向图G存在
600 函数功能：将邻接表表示的无向图销毁，并释放所有表结点的空间。
601 返回类型：status
602 *********************************************************/
603 status DestroyGraph(ALGraph &G)
604 {
605     int t=G.vexnum;
606
607     //从firstarc依次释放空间
```

```
608     for( int i=0 ; i<t ; i++ ){
609         ArcNode *a=G.vertices[i].firstarc;
610         while(a){
611             ArcNode *next=a->nextarc;
612             free(a);
613             a=next;
614         }
615     }
616 
617     //相关值归零
618     G.vexnum=0;G.arcnum=0;
619     return OK;
620 }
621 
622 /*********************************************************
623 函数名:   LocateVex
624 初始条件：无向图G存在
625 函数功能：u是和G中顶点关键字类型相同的给定值；根据u查找顶
626 点，成功时返回关键字为u的顶点位置序号（简称位序），否则返
627 回-1。
628 返回类型：int
629 *********************************************************/
630 int LocateVex(ALGraph G,KeyType u)
631 {
632     int t=G.vexnum;
633 
634     //遍历顶点数组
635     for( int i=0 ; i<t ; i++ )
636         if( u==G.vertices[i].data.key )
637             return i;
638     return -1;
639 }
640 
641 /*********************************************************
642 函数名:   PutVex
643 初始条件：无向图G存在
644 函数功能：u是和G中顶点关键字类型相同的给定值；操作结果是
645 对关键字为u的顶点赋值value（要求关键字具有唯一性）。成功
646 赋值返回OK，否则返回ERROR。
647 返回类型：status
648 *********************************************************/
649 status PutVex(ALGraph &G,KeyType u,VertexType value)
650 {
651     int t=G.vexnum;
652 
653     int num[100]={0};           //判定关键字重复数组
654     for( int i=0 ; i<t ; i++ )
655         num[G.vertices[i].data.key]++;
656     if( u!=value.key&&num[value.key] )
657         return ERROR;           //关键字重复
658     for( int i=0 ; i<t ; i++ ){
659         if( u==G.vertices[i].data.key ){
660             G.vertices[i].data=value;
661             return OK;
662         }
663     }
664     return ERROR;
```

```
665 }
666
667 /*********************************************************
668 函数名:    FirstAdjVex
669 初始条件:无向图G存在
670 函数功能:u是和G中顶点关键字类型相同的给定值;返回关键字
671 为u的顶点第一个邻接顶点位置序号(简称位序),否则返回-1。
672 返回类型:int
673 *********************************************************/
674 int FirstAdjVex(ALGraph G,KeyType u)
675 {
676     int t=G.vexnum,a;
677
678     //遍历顶点数组
679     for( int i=0 ; i<t ; i++ ){
680         if( u==G.vertices[i].data.key ){
681             return G.vertices[i].firstarc->adjvex;
682         }
683     }
684     return -1;
685 }
686
687 /*********************************************************
688 函数名:    NextAdjVex
689 初始条件:无向图G存在
690 函数功能: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
691 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
692 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
693 回-1。
694 返回类型:int
695 *********************************************************/
696 int NextAdjVex(ALGraph G,KeyType v,KeyType w)
697 {
698     int t=G.vexnum,a=-1;
699
700     //寻找对应v的结点
701     for( int i=0 ; i<t ; i++ )
702         if( v==G.vertices[i].data.key ){
703             a=i;
704             break;
705         }
706     if( a==-1 )
707         return -1;
708
709     //在v的邻接结点中寻找w
710     ArcNode *arc=G.vertices[a].firstarc;
711     while( arc ){
712         if( G.vertices[arc->adjvex].data.key==w )
713             break;
714         arc=arc->nextarc;
715     }
716     if( !arc )
717         return -1;
718     if( !(arc->nextarc) )    //最后一个邻接顶点
719         return -1;
720     return arc->nextarc->adjvex;
721 }
```

```
722
723 /*************************************************************
724 函数名:    InsertVex
725 初始条件:无向图G存在
726 函数功能:在图G中插入新顶点关v(要求关键字具有唯一性,注意
727 判断顶点个数是否已满)。成功返回OK,否则返回ERROR。
728 返回类型:status
729 *************************************************************/
730 status InsertVex(ALGraph &G,VertexType v)
731 {
732     //判断顶点数是否超过最大值
733     if( G.vexnum>=MAX_VERTEX_NUM )
734         return ERROR;
735
736     int t=G.vexnum;
737
738     //判断关键字唯一性
739     int num[1000]={0};
740     for( int i=0 ; i<t ; i++ )
741         num[G.vertices[i].data.key]++;
742     if( num[v.key] )
743         return ERROR;
744     G.vertices[t].data=v;
745     G.vexnum++;
746     return OK;
747 }
748
749 /*************************************************************
750 函数名:    DeleteVex
751 初始条件:无向图G存在
752 函数功能:v是和G中顶点关键字类型相同的给定值;操作结果是在
753 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
754 回ERROR。
755 返回类型:status
756 *************************************************************/
757 status DeleteVex(ALGraph &G,KeyType v)
758 {
759     int t=G.vexnum,a=-1;
760
761     //顶点数组移位
762     for( int i=0 ; i<t ; i++ )
763         if( G.vertices[i].data.key==v )
764             a=i;
765     if( a==-1 )
766         return ERROR;
767     if( t==1 )
768         return ERROR;
769     ArcNode *arc=G.vertices[a].firstarc;
770     while(arc){
771         ArcNode *c=arc->nextarc;
772         free(arc);
773         arc=c;
774     }
775     for( int i=a ; i<t-1 ; i++ )
776         G.vertices[i]=G.vertices[i+1];
777     G.vexnum--;t--;
778
```

```
779     //遍历找相关弧
780     for( int i=0 ; i<t ; i++ ){
781         ArcNode *arc=G.vertices[i].firstarc ,*pre=NULL,*usearc=NULL,*usepre=NULL;
782         while( arc ){
783             if( arc->adjvex==a ){
784                 usearc=arc;
785                 usepre=pre;
786             }
787             if( arc->adjvex>a )
788                 arc->adjvex--;
789             pre=arc;
790             arc=arc->nextarc;
791         }
792         if( !usearc )
793             continue;
794         G.arcnum--;
795         if(usepre==NULL){
796             G.vertices[i].firstarc=usearc->nextarc;
797             free(usearc);
798             continue;
799         }
800         if( !(usearc->nextarc) ){
801             free(usearc);
802             usepre->nextarc=NULL;
803             continue;
804         }
805         usepre->nextarc=usearc->nextarc;
806         free(usearc);
807     }
808     return OK;
809 }
810
811 /*********************************************************
812 函数名:   DeleteArc
813 初始条件:无向图G存在
814 函数功能:v、w是和G中顶点关键字类型相同的给定值;操作结果
815 是在图G中删除弧<v,w>。成功返回OK,否则返回ERROR。
816 返回类型:status
817 *********************************************************/
818 status DeleteArc(ALGraph &G,KeyType v,KeyType w)
819 {
820     int t=G.vexnum,a=-1,b=-1;
821
822     //遍历寻找v和w对应结点
823     for( int i=0 ; i<t ; i++ )
824         if( G.vertices[i].data.key==v ){
825             a=i;
826             break;
827         }
828     for( int i=0 ; i<t ; i++ )
829         if( G.vertices[i].data.key==w ){
830             b=i;
831             break;
832         }
833     if( a==-1||b==-1 )
834         return ERROR;
835
```

```
836     //删除弧（双向）
837     G.arcnum--;
838     ArcNode *arc=G.vertices[a].firstarc,*pre=NULL;
839     while(arc){
840         if( arc->adjvex==b ){
841             if( !pre ){
842                 G.vertices[a].firstarc=arc->nextarc;
843                 free(arc);
844             }
845             else{
846                 pre->nextarc=arc->nextarc;
847                 free(arc);
848             }
849             break;
850         }
851         pre=arc;
852         arc=arc->nextarc;
853     }
854     if( !arc )          //未找到对应的弧，即v和w不邻接
855         return ERROR;
856     arc=G.vertices[b].firstarc;pre=NULL;
857     while(arc){
858         if( arc->adjvex==a ){
859             if( !pre ){
860                 G.vertices[b].firstarc=arc->nextarc;
861                 free(arc);
862             }
863             else{
864                 pre->nextarc=arc->nextarc;
865                 free(arc);
866             }
867             break;
868         }
869         pre=arc;
870         arc=arc->nextarc;
871     }
872     return OK;
873 }
874
875 /*********************************************************
876 函数名:   DFSTraverse
877 初始条件：无向图G存在
878 函数功能：对图G进行深度优先搜索遍历，依次对图中的每一个顶
879 点使用函数visit访问一次，且仅访问一次。
880 返回类型：status
881 辅助函数：深度优先搜索函数
882 *********************************************************/
883 void visit(VertexType v)
884 {
885     printf(" %d %s",v.key,v.others);
886 }
887
888 void dfs(ALGraph &G,int v,void (*visit)(VertexType)){
889     ArcNode *arc=G.vertices[v].firstarc;
890     visit(G.vertices[v].data);
891     flag[v]=1;                  //标记已遍历结点
892     while( arc ){
```

```
893             if( !flag[arc->adjvex] )
894                 dfs(G,arc->adjvex,visit);
895             arc=arc->nextarc;
896         }
897         return ;
898     }
899
900     status DFSTraverse(ALGraph &G,void (*visit)(VertexType))
901     {
902         int t=G.vexnum;
903         for( int i=0 ; i<t ; i++ ){
904             if( !flag[i] )
905                 dfs(G,i,visit);
906         }
907         return OK;
908     }
909
910     /*******************************************************
911     函数名:   BFSTraverse
912     初始条件:无向图G存在
913     函数功能:对图G进行广度优先搜索遍历,依次对图中的每一个顶
914     点使用函数visit访问一次,且仅访问一次。
915     返回类型:status
916     *******************************************************/
917     status BFSTraverse(ALGraph &G,void (*visit)(VertexType))
918     {
919         int t=G.vexnum;
920
921         char flag[100]={0};          //标记已遍历结点
922
923             //构建队列
924         ArcNode *queue[100];int front=0,tail=0;
925
926         //遍历阶段
927         for( int i=0 ; i<t ; i++ ){
928             if( !flag[i] ){          //若结点未遍历过
929                 visit(G.vertices[i].data);
930                 front=tail=0;        //队列初始化
931                 queue[tail++]=G.vertices[i].firstarc;        //进队
932                 while(front<tail){
933                     ArcNode* arc=queue[front++];
934                     if( !arc )
935                             break;
936                     if( !flag[arc->adjvex] ){
937                         flag[arc->adjvex]=1;
938                         visit(G.vertices[arc->adjvex].data);
939                         arc=arc->nextarc;
940                         while(arc){
941                             queue[tail++]=arc;                   //邻接结点全部进队
942                             arc=arc->nextarc;
943                         }
944                     }
945                 }
946             }
947         }
948         //这个大括号结尾真丑(bushi)
949         return OK;
```

```
950 }
951
952 /**************************************************
953 函数名:    SaveGrapgh
954 初始条件:无向图G存在
955 函数功能:将图G写到文件名为FileName的文件中,返回OK
956 返回类型:status
957 **************************************************/
958 status SaveGraph(ALGraph G, char FileName[])
959 {
960         //文件操作初始化
961     FILE *fp;
962     fp=fopen(FileName,"w+");
963
964     //文件内容数组
965     VertexType V[100];
966     KeyType VR[100][2];
967
968     //构造V和VR数组
969     char flag[100][100]={0};              //用于判断边是否重复
970     int num=0;
971     for( int i=0 ; i<G.vexnum ; i++ )
972         V[i]=G.vertices[i].data;
973     for( int i=0 ; i<G.vexnum ; i++ ){
974         ArcNode *arc=G.vertices[i].firstarc;
975         while(arc){
976             int a=G.vertices[i].data.key,b=G.vertices[arc->adjvex].data.key;
977             if( flag[a][b]||flag[b][a] ){
978                 arc=arc->nextarc;
979                 continue;
980             }
981             flag[a][b]=1;
982             VR[num][0]=a;VR[num][1]=b;
983             num++;
984             arc=arc->nextarc;
985         }
986     }
987     for( int i=0 ; i<num ; i++ ){
988         if( VR[i][0]>VR[i][1] ){
989             int t=VR[i][0];
990             VR[i][0]=VR[i][1];
991             VR[i][1]=t;
992         }
993     }
994
995     //对VR数组进行插入排序(第一维)
996     for( int i=0 ; i<num ; i++ ){
997         int Min=VR[i][0],Mini=i;
998         for( int j=i ; j<num ; j++ ){
999             if( VR[j][0]<Min ){
1000                Min=VR[j][0];
1001                Mini=j;
1002            }
1003        }
1004        int t1=VR[i][0],t2=VR[i][1];
1005        VR[i][0]=VR[Mini][0];VR[i][1]=VR[Mini][1];
1006        VR[Mini][0]=t1;VR[Mini][1]=t2;
```

```
1007        }
1008
1009        //对VR数组进行插入排序（第二维）
1010        for( int i=0 ; i<num ; i++ ){
1011            int tail=i;
1012            while(VR[tail][0]==VR[i][0])
1013                tail++;
1014            if( tail-i==0 )
1015                continue;
1016            for( int k=i ; k<tail ; k++ ){
1017                int Min=VR[k][1],Mini=k;
1018                for( int j=k ; j<tail ; j++ ){
1019                    if( VR[j][1]<Min){
1020                        Min=VR[j][1];
1021                        Mini=j;
1022                    }
1023                }
1024                int t=VR[k][1];
1025                VR[k][1]=VR[Mini][1];
1026                VR[Mini][1]=t;
1027            }
1028            i=tail-1;
1029        }
1030
1031        //文件输入
1032        for( int i=0 ; i<G.vexnum ; i++ )
1033            fprintf(fp,"%d %s ",V[i].key,V[i].others);
1034        fprintf(fp,"%d %s ",-1,"nil");
1035        for( int i=0 ; i<num ; i++ )
1036            fprintf(fp,"%d %d ",VR[i][0],VR[i][1]);
1037        fprintf(fp,"%d %d",-1,-1);
1038        fclose(fp);             //关闭文件好习惯
1039        return OK;
1040    }
1041
1042    /*********************************************************
1043    函数名:    LoadGraph
1044    初始条件：无向图G不存在
1045    函数功能：将图G写到文件名为FileName的文件中，返回OK
1046    返回类型：status
1047    *********************************************************/
1048    status LoadGraph(ALGraph &G, char FileName[])
1049    {
1050            //文件初始化
1051        FILE *fp;
1052        fp=fopen(FileName,"r+");
1053
1054        //从文件中读入V和VR数组
1055        int vexnum=0,arcnum=0,v1,v2;
1056        VertexType V[100];
1057        KeyType VR[100][2];
1058        fscanf(fp,"%d %s",&V[0].key,V[0].others);
1059        while(V[vexnum].key!=-1){
1060            vexnum++;
1061            fscanf(fp,"%d %s",&V[vexnum].key,V[vexnum].others);
1062        }
1063        fscanf(fp,"%d %d",&VR[0][0],&VR[0][1]);
```

```
1064        while(VR[arcnum][0]!=-1){
1065            arcnum++;
1066            fscanf(fp,"%d %d",&VR[arcnum][0],&VR[arcnum][1]);
1067        }
1068
1069        //利用CreateCraph函数构造无向图
1070        CreateCraph(G,V,VR);
1071        fclose(fp);
1072        return OK;
1073    }
1074
1075    /*************************************************************************
1076    函数名称：AddGraph
1077    初始条件：当前图数目未超过上限
1078    功能说明：添加一个新的图，赋予其图名
1079    返回值类型：status
1080    *************************************************************************/
1081    status AddGraph(LISTS &graph,char graphname[])
1082    {
1083            for( int i=1 ; i<11 ; i++ )
1084                    if( !strcmp(graphname,graph.elem[i].name) )
1085                            return ERROR;
1086            graph.length++;
1087            graph.elem[graph.length].G.vexnum=0;
1088            graph.elem[graph.length].G.arcnum=0;
1089            int i;
1090            for( i=0 ; graphname[i]!='\0' ; i++ )
1091                    graph.elem[graph.length].name[i]=graphname[i];
1092            graph.elem[graph.length].name[i]='\0';
1093            graph.elem[graph.length].G.kind=1;
1094            return OK;
1095    }
1096
1097    int DeleteGraph(LISTS &graph,char graphname[]){
1098            for( int i=1 ; i<11 ; i++ ){
1099                    if( !strcmp(graphname,graph.elem[i].name) ){
1100                            for( int j=i ; j<graph.length ; j++ )
1101                                    graph.elem[j]=graph.elem[j+1];
1102                            graph.elem[graph.length].G.vexnum=0;
1103                            graph.elem[graph.length].G.arcnum=0;
1104                            graph.length--;
1105                            return i;
1106                    }
1107            }
1108            return 0;
1109    }
1110
1111    int ChooseGraph(LISTS &graph,char graphname[])
1112    {
1113            for( int i=1 ; i<11 ; i++ )
1114                    if( !strcmp(graphname,graph.elem[i].name) )
1115                            return i;
1116            return 0;
1117    }
1118
1119    /*************************************************************************
1120    函数名称：VerticesSetLessThanK
```

```
1121  初始条件：图G存在
1122  功能说明：返回与顶点v距离小于k的顶点集合
1123  返回值类型：status
1124  辅助函数：深度优先搜索函数
1125  ******************************************************************/
1126  void dfs_lessk(ALGraph &G,int k,int adj){
1127          if( k<=0 )
1128                  return ;
1129          if( flag[adj] )
1130                  return ;
1131          flag[adj]++;
1132          adjvex[nums++]=adj;
1133          ArcNode *arc=G.vertices[adj].firstarc;
1134          while(arc){
1135                  dfs_lessk(G,k-1,arc->adjvex);
1136                  arc=arc->nextarc;
1137          }
1138  }
1139  status VerticesSetLessThanK(ALGraph &G,KeyType v,int k){
1140          int t=G.vexnum,a=-1;
1141          for( int i=0 ; i<t ; i++ )
1142                  if( G.vertices[i].data.key==v ){
1143                          a=i;
1144                          break;
1145                  }
1146          if( a==-1 )
1147                  return ERROR;
1148          dfs_lessk(G,k,a);
1149          return OK;
1150  }
1151
1152  /******************************************************************
1153  函数名称：ShortestPathLength
1154  初始条件：图G存在
1155  功能说明：返回顶点v与顶点w的最短路径的长度
1156  返回值类型：int
1157  ******************************************************************/
1158  int ShortestPathLength(ALGraph &G,KeyType v,KeyType w){
1159          int t=G.vexnum,a=-1,b=-1,Min,u;
1160          for( int i=0 ; i<t ; i++ )
1161                  if( G.vertices[i].data.key==v ){
1162                          a=i;
1163                          break;
1164                  }
1165          for( int i=0 ; i<t ; i++ )
1166                  if( G.vertices[i].data.key==w ){
1167                          b=i;
1168                          break;
1169                  }
1170          if( a==-1||b==-1 )
1171                  return ERROR;
1172          int grid[100][100]={0};
1173          for( int i=0 ; i<t ; i++ ){
1174                  ArcNode *arc=G.vertices[i].firstarc;
1175                  while(arc){
1176                          grid[i][arc->adjvex]=1;
1177                          arc=arc->nextarc;
```

```
1178                    }
1179            }
1180            for( int i=0 ; i<100 ; i++ )
1181                    for( int j=0 ; j<100 ; j++ )
1182                            if( !grid[i][j] )
1183                                    grid[i][j]=10000;
1184            for( int i=0 ; i<100 ; i++ )
1185                    grid[i][i]=0;
1186            for( int i=0 ; i<t ; i++ )
1187                    for( int j=0 ; j<t ; j++ )
1188                            for( int k=0 ; k<t ; k++ )
1189                                    if( grid[i][j]>grid[i][k]+grid[k][j] )
1190                                            grid[i][j]=grid[i][k]+grid[k][j];
1191            return grid[a][b];
1192    }
1193
1194    /*****************************************************************
1195    函数名称：ConnectedComponentsNums
1196    初始条件：图G存在
1197    功能说明：返回图G的所有连通分量的个数
1198    返回值类型：int
1199    *****************************************************************/
1200    void dfs_connect(ALGraph &G,char flag[],int v){
1201            ArcNode *arc=G.vertices[v].firstarc;
1202        flag[v]=1;                      //标记已遍历结点
1203        while( arc ){
1204            if( !flag[arc->adjvex] )
1205                dfs_connect(G,flag,arc->adjvex);
1206            arc=arc->nextarc;
1207        }
1208        return ;
1209    }
1210
1211    int ConnectedComponentsNums(ALGraph &G){
1212            char flag[100]={0};
1213            int t=G.vexnum;
1214            int ret=0;
1215            for( int i=0 ; i<t ; i++ ){
1216                    if( !flag[i] ){
1217                            ret++;
1218                            dfs_connect(G,flag,i);
1219                    }
1220            }
1221            return ret;
1222    }
```