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 计卓 2101

学　　号 U202112071

姓　　名 王彬

指导教师 许贵平

报告日期 2022 年 6 月 4 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

线性表是最常用且最简单的一种数据结构。在非空线性表中，每一非首结点都有其前驱；同时，每一非尾结点都有其后继。在非空表中每一个元素都有一个确定的位置，即如果 a_i 为第 i 位数据元素，我们称 i 为 a_i 在线性表中的位序。事实上，线性表是一个相当灵活的数据结构，长度可以按需改变，同时元素不仅可以访问，还可以增删等。通过实验达到：

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

1.1 问题描述

采用顺序表作为线性表的物理结构，实现 1.2 小节的运算。其中 ElemType 为数据元素的类型名，具体含义可自行定义。

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

1.2 系统设计

我们使用命令行的形式实现顺序存储结构的线性表。

换言之，我们对每次输入的指令字符串进行分析，并自动归类指令的含义。

为实现多表管理，并便捷用户使用该系统，在一行中用户需要键入操作名和该操作所需要的变量，如表名、数字、文件名等。例如，如需创建一个名为”cst” 的新线性表，须键入”LISTINIT cst”；再例如，如果需要销毁名为”wb” 的线性表，应输入”LISTDESTROY wb”。

与此同时，为便利用户使用，即使用户操作名输入错误，甚至是本应输入数字的地方误键入了字符串，程序会提出警告，而不是出错而闪退。

具体的指令集，详见源代码的 MENU 部分，其中，L 指的是表名，I 和 E 均为数字，F 指代文件名。

依据最小完备性和常用性相结合的原则，以函数形式定义了线性表的初始

化表、销毁表、清空表、判定空表、求表长和获得元素等 12 种基本运算，具体运算功能定义如下：

1) 初始化表：函数名称是 InitList(L)；初始条件是线性表 L 不存在；操作结果是构造一个空的线性表；
2) 销毁表：函数名称是 DestroyList(L)；初始条件是线性表 L 已存在；操作结果是销毁线性表 L；
3) 清空表：函数名称是 ClearList(L)；初始条件是线性表 L 已存在；操作结果是将 L 重置为空表；
4) 判定空表：函数名称是 ListEmpty(L)；初始条件是线性表 L 已存在；操作结果是若 L 为空表则返回 TRUE, 否则返回 FALSE；
5) 求表长：函数名称是 ListLength(L)；初始条件是线性表已存在；操作结果是返回 L 中数据元素的个数；
6) 获得元素:函数名称是 GetElem(L,i,e);初始条件是线性表已存在,1≤i≤ListLength(L)；操作结果是用 e 返回 L 中第 i 个数据元素的值；
7) 查找元素：函数名称是 LocateElem(L,e,compare())；初始条件是线性表已存在；操作结果是返回 L 中第 1 个与 e 满足关系 compare () 关系的数据元素的位序，若这样的数据元素不存在，则返回值为 0；
8) 获得前驱：函数名称是 PriorElem(L,cur_e,pre_e)；初始条件是线性表 L 已存在；操作结果是若 cur_e 是 L 的数据元素，且不是第一个，则用 pre_e 返回它的前驱，否则操作失败，pre_e 无定义；
9) 获得后继：函数名称是 NextElem(L,cur_e,$next_e$)；初始条件是线性表 L 已存在；操作结果是若 cur_e 是 L 的数据元素，且不是最后一个，则用 $next_e$ 返回它的后继，否则操作失败，$next_e$ 无定义；
10) 插入元素:函数名称是 ListInsert(L,i,e);初始条件是线性表 L 已存在,1≤i≤ListLength(L)+1；操作结果是在 L 的第 i 个位置之前插入新的数据元素 e。
11) 删除元素：函数名称是 ListDelete(L,i,e)；初始条件是线性表 L 已存在且非空，1≤i≤ListLength(L)；操作结果：删除 L 的第 i 个数据元素，用 e 返回其值；
12) 遍历表：函数名称是 ListTraverse(L,visit())，初始条件是线性表 L 已存在；操作结果是依次对 L 的每个数据元素调用函数 visit()。
13) 最大连续子数组和：函数名称是 MaxSubArray(L); 初始条件是线性表 L 已存

在且非空，请找出一个具有最大和的连续子数组（子数组最少包含一个元素），操作结果是其最大和；

14) 和为 K 的子数组：函数名称是 SubArrayNum(L,k); 初始条件是线性表 L 已存在且非空, 操作结果是该数组中和为 k 的连续子数组的个数；
15) 顺序表排序：函数名称是 sortList(L)；初始条件是线性表 L 已存在；操作结果是将 L 由小到大排序；
16) 实现线性表的文件形式保存：其中，□ 需要设计文件数据记录格式，以高效保存线性表数据逻辑结构 (D,R) 的完整信息；□ 需要设计线性表文件保存和加载操作合理模式。附录 B 提供了文件存取的参考方法。
17) 实现多个线性表管理：设计相应的数据结构管理多个线性表的查找、添加、移除等功能。

1.3 系统实现

为实现多表管理，在建立每一个新表之时，都需要对它进行命名。新建时，系统会自动给新表分配物理空间；删除线性表时，系统亦会自动撤销线性表的存储空间。线性表的管理使用数据结构 LISTS 进行储存，存储形式亦为一种线性表。如果线性表的个数超出预期，系统亦会开辟新的空间；如果计算机物理空间不足，系统将抛出错误。

在相应的操作命令之后，必须输入所操作的对象线性表，以实现对于对象线性表的施动。

1.3.1 初始化表

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

请参考算法1.1

算法 1.1. 初始化表

Input: name of the Linear Sequence

Output: Initialized sequence

procedure InitList(L)

if then $L.elem! = NULL$
  return $INFEASIBLE$
end if
$L.length \leftarrow 0$
$L.listsize \leftarrow LISTINITSIZE$
$MallocL$
return OK
end procedure

具体的实现是，我们先通过表名查找是否在 LISTS 中存在和新表名相同的线性表。如果有，则创建失败；如果没有，即继续创建。

该算法空间复杂度为 $O(n)$，时间复杂度为 $O(1)$.

1.3.2 销毁线性表

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

该函数设计思想是，若 L 非空，将 L 置空；若 L 已空，返回异常即可。

1.3.3 清空线性表

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

该函数设计思想是，将 L 的长度设为 0.

1.3.4 判定空表

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

请参考算法1.2

算法 1.2. 判定空表

Input: Linear Sequence: L
Output: status
procedure ListEmpty(L)
if then$L.elem == NULL$
return $INFEASIBLE$
end if
if then$L.length! = 0$
return $FALSE$
end if
return $TRUE$
end procedure

经分析，该算法时间复杂度为 O(1).

1.3.5 求表长

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

该算法的设计思想是，若线性表不存在，返回 INFEASIBLE；若线性表存在，直接返回 L.length 即可。

1.3.6 获得元素

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

请参考算法1.3

算法 1.3. 获得元素
Input: Linear Sequence: L,i
Output: status, e
procedure GetElem(L)
if then$L.elem == NULL$

return $INFEASIBLE$
end if
if then$i < 1 or i > length$
return $ERROR$
end if
$e = L.elem[i-1]$
return $TRUE$
end procedure

经分析，该算法时间复杂度为 O(1).

1.3.7 查找元素

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

算法的设计思想是，遍历线性表 L，如果查找到与 e 满足 compare() 关系的元素，则返回它的次序；否则，返回 0.

请参考算法1.4

算法 1.4. 获得元素

Input: Linear Sequence: L,e
Output: i
procedure LocateElem(L)
if then$L.elem == NULL$
return $INFEASIBLE$
end if
while $i < L.length$ **do**
if then$Compare(L.elem[i-1], e)$
return i

```
        return 0
```

经分析，该算法时间复杂度为 O(n).

1.3.8 获得前驱

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

算法的设计思想是，在线性表 L 中查找到 cur_e 所在的位置，如果找到并且不在表头，则将其前驱赋给 pre_e；如果不合法，则返回 ERROR。

相应的算法详见代码部分，简单拼凑即可，此处不再赘述。

1.3.9 获得后继

函数名称是 NextElem(L,cur_e,$next_e$)；初始条件是线性表 L 已存在；操作结果是若 cur_e 是 L 的数据元素，且不是最后一个，则用 $next_e$ 返回它的后继，否则操作失败，$next_e$ 无定义；

算法的设计思想和获得前驱类似，此处亦不再赘述。

1.3.10 插入元素

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

算法的设计思想是，先将线性表长度增加 1，如果线性表长度超过系统分配的物理长度则返回 ERROR；线性表中第 i 位之后的元素各自向后挪一位。待第 i 位空下时，再将 e 赋给线性表 L 中的第 i 位元素。

请参考算法1.5

算法 1.5. 插入元素

Input: Linear Sequence: L,i,e

Output: Status

procedure ListInsert(L)

if then$L.elem == NULL$

return $INFEASIBLE$

end if

if then$number(i)illigal$

return $ERROR$

$j = L.length - 1$

while $j > i$ **do**

$L.elem[j+1] = L.elem[j]$

end while

$L.elem[i-1] = e$

return OK

我们假定需要插入的元素位置等概率地出现在表头到表尾，则平均需要移动的元素个数为 $(n+1)/2$，即，该算法时间复杂度为 O(n).

1.3.11 删除元素

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

该算法的设计思想和插入元素类似，只不过将插入改为删除而已。换言之，只需将线性表第 i 位之后的元素向前移动一格即可。

1.3.12 遍历表

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

该算法的设计思想是，依次对 L 的每个元素调用 visit() 即可。

1.3.13 最大连续子数组和

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

算法的设计思想是，计算前缀和 sum[1..n]，而后枚举 i,j(i<j)，计算 sum[j]-sum[i] 并取最大值，并输出该最大值即可。具体算法附如下：

算法输入：线性表 L

算法输出：最大值

算法流程：

1. 计算前缀和数组 sum，sum[0]=L.elem[0], 对于 i!=0 时，sum[i]=sum[i-1]+L.elem[i]；

2. 枚举 i,j，记录 sum[j]-sum[i] 的最大值。

算法时间复杂度：

因为我们枚举了 i 和 j，故其时间复杂度为 $O(n^2)$.

1.3.14 和为 K 的子数组

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

和求最大子数组和相类似，将前缀和数组求得，并枚举 i,j 记录 sum[j]-sum[i] 和 k 相等的二元组 (i, j) 个数即可。相关的算法和求最大连续子数组和类似，此处不再赘述。

由相似的分析可得，其时间复杂度为 $O(n^2)$.

1.3.15 顺序表排序

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

由于线性表的长度小于 100，数据量较小，我们不妨用冒泡排序 (Bubble Sort) 的方式进行排序。

请参考算法1.6

算法 1.6. 顺序表排序

Input: Linear Sequence: L

Output: Status
procedure sortList(L)
if then$L.elem == NULL$
return $INFEASIBLE$
end if
for do$i := 0 to L.length - 1$
for do$j := i + 1 to L.length - 1$
$k = min L.elem[i], L.elem[j]$
$Swap L.elem[i], L.elem[k]$
return OK

容易分析得，该算法的时间复杂度为 $O(n^2)$.

1.3.16 实现线性表的文件形式保存

其中，□ 需要设计文件数据记录格式，以高效保存线性表数据逻辑结构 (D,R) 的完整信息；□ 需要设计线性表文件保存和加载操作合理模式。附录 B 提供了文件存取的参考方法。

我们首先对文件数据记录格式进行设计。为使得文件空间得到最大化利用，我们只将线性表从表头到表尾一一地输出至文件，而后输出”0” 作为文件结尾标志。如是，线性表元素的先后顺序得以保存，且事实上长度可以再次计算得出。

读写文件较易，亦不再多述。

1.4 系统测试

本次实验使用 CodeBlocks 进行编写。我们将实验划分为三个文件协同进行编译，分别为”main.cpp”, ”def.h”, ”opt.h”。这三个文件分别为主程序文件、数据结构定义文件和操作文件。我们由主程序文件调用数据结构定义文件和操作文件。即，”def.h” 和”opt.h” 从属于”main.cpp”.

1.4.1 测试计划

为便于浏览并且提纲挈领，我们对测试计划绘制了表格如下。其中斜线表示输入为空或者无输出。

表 1-1 基于线性存储结构的线性表测试计划表.

序号	操作	输入	预期输出
1	LISTINIT	list1	/
2	LISTLEN	list1	0
3	LISTINS	list1 1 3	/
4	GETELEM	list1 1	3
5	LISTINS	list1 2 1	/
6	LISTINS	list1 3 6	/
7	LISTSORT	list1	/
8	SHOWLIST	list1	1 3 6
9	LISTDEL	list1 2	/
10	ELEMNEXT	list1 1	6
11	LISTINS	list1 1 8	/
12	LISTINS	list1 1 3	/
13	MAXSUBARRAY	list1	18
14	SUBARRAYNUM	list1	1
15	SAVELIST	list1 cst.dat	/
16	LISTDESTROY	list1	/
17	LISTINIT	list2	/
18	LOADLIST	list2 cst.dat	/
19	SHOWLIST	list2	3 8 1 6
20	EXIT	/	/

1.4.2 实际测试

我们依据上表对程序进行测试。

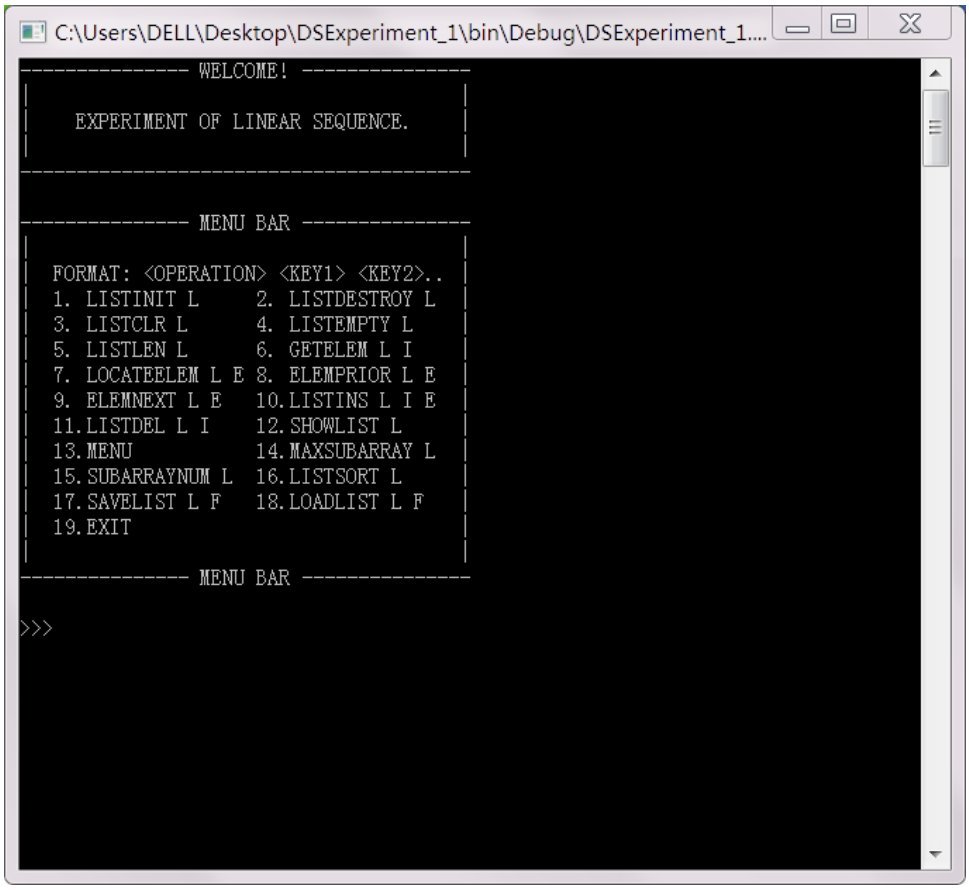

图 1-1　命令行初始界面

图 1-1 为我们开始使用实验程序的初始界面。

下面我们一一对程序的使用进行展示与说明。

图 1-2　新建表

如图 1-2，如图所示，我们新建了 list1 表。”»>” 表示成功，因为我们对本实验默认是成功操作，如果不输出则意味着操作成功。事实上如果输入有误，程序会进行报错处理。但不会出现闪退的情况，除非计算机的内存不足。我们对本实验中的实验程序鲁棒性具有相当高的预期。

如图 1-3，因为表中无数据，所以表是空的，表长为 0.

如图 1-4，在 list1 表的第一位插入数据 3. 插入成功.

如图 1-5，取在 list1 表的第一位元素的值。该值为 3. 读取成功.

如图 1-6，插入数据成功.

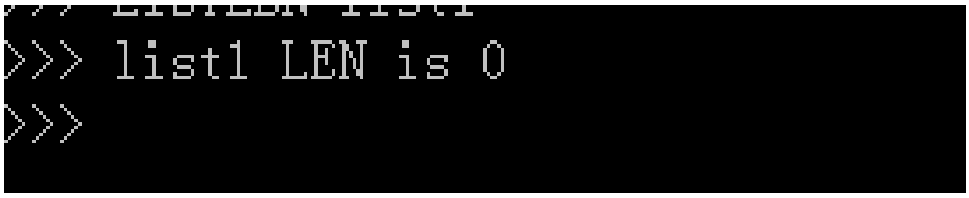

图 1-3　返回 list1 表长

图 1-4　插入数据

如图 1-7，插入数据成功.

如图 1-8，我们对线性表 list1 进行了从小到大的排序。

如图 1-9，原本杂乱的数据完成了由小到大的排序，返回 1，3，6. 说明排序成功。

如图 1-10，我们对 list1 删除第 2 个元素。删除成功。

如图 1-11，第一位的下一个元素值为 6. 正确。

如图 1-12，在 list1 的第一位处插入 8. 插入成功。

如图 1-13，在 list1 的第一位处插入 3. 插入成功。

如图 1-14，求得最大子数组和，和为 18，正确。

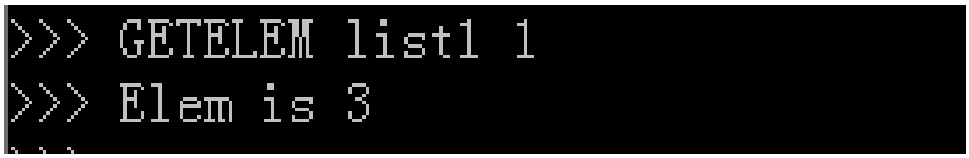

图 1-5　取数据

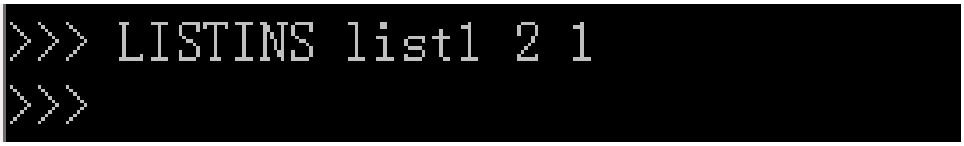

图 1-6　插入数据

如图 1-15，求得子数组和，答为 1，正确。

图 1-7　插入数据

图 1-8　线性表排序

如图 1-16，储存成功！

如图 1-17，销毁成功！

如图 1-18，建立成功！

如图 1-19，读取成功！

如图 1-20，说明正确读取。

如图 1-21，安全退出，返回值为 0.

综上，均符合预期！

1.5　实验小结

在这次实验中，我们的创新之处在于使用了命令行的形式完成了线性表的设计实验。命令行首先需要匹配命令的含义，在这方面我下了不少工夫。另外，我们为了追求程序的鲁棒性，使得即使误将数值输入为字符串也不会出现闪退，我们对程序的输入进行了些许修饰，以达到预期的效果。

```
>>> SHOWLIST list1
>>> #1 3 6
>>>
```

图 1-9　显示排序结果

```
>>> LISTDEL list1 2
>>>
```

图 1-10　元素删除

```
>>> ELEMNEXT list1 1
>>> Elem is 6
>>>
```

图 1-11　下一元素

```
>>> LISTINS list1 1 8
>>>
```

图 1-12　插入数据

```
>>> LISTINS list1 1 3
>>>
```

图 1-13　插入数据

```
>>> MAXSUBARRAY list1
>>> MAX = 18
>>>
```

图 1-14　最大和

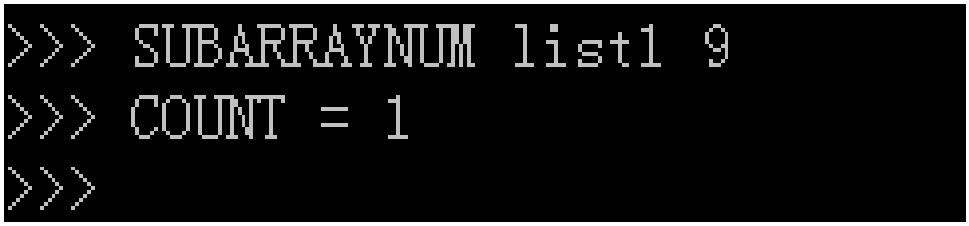

图 1-15　数组和

图 1-16　储存文件

图 1-17　销毁 list1

图 1-18　新建 list2

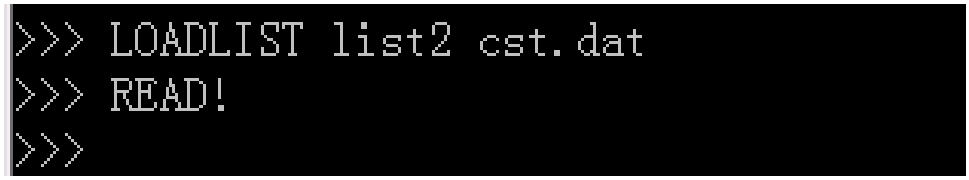

图 1-19　读取文件

图 1-20　命令行初始界面

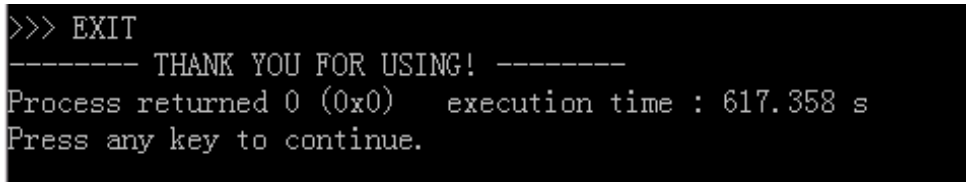

图 1-21　退出

2 基于邻接表的图实现

2.1 问题描述

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

图是一种数据结构，加上一组基本操作，就成了抽象数据类型。依据最小完备性和常用性相结合的原则，以函数形式定义了创建图、销毁图、查找顶点、获得顶点值和顶点赋值等 12 种基本运算。即，我们利用邻接表的形式，完成对图的数据结构的实现。在本设计中，我们假定图为无向图。

2.2 系统设计

依据最小完备性和常用性相结合的原则，以函数形式定义了创建图、销毁图、查找顶点、获得顶点值和顶点赋值等 12 种基本运算。具体运算功能定义和说明如下。具体运算功能定义如下：

1) 创建图：函数名称是 CreateCraph(G,V,VR)；初始条件是 V 是图的顶点集，VR 是图的关系集；操作结果是按 V 和 VR 的定义构造图 G；
2) 销毁图：函数名称是 DestroyGraph(G)；初始条件图 G 已存在；操作结果是销毁图 G；
3) 查找顶点：函数名称是 LocateVex(G,u)；初始条件是图 G 存在，u 是和 G 中顶点关键字类型相同的给定值；操作结果是若 u 在图 G 中存在，返回关键字为 u 的顶点位置序号（简称位序），否则返回其它表示“不存在”的信息；
4) 顶点赋值：函数名称是 PutVex (G,u,value)；初始条件是图 G 存在，u 是和 G 中顶点关键字类型相同的给定值；操作结果是对关键字为 u 的顶点赋值 value；
5) 获得第一邻接点：函数名称是 FirstAdjVex(G, u)；初始条件是图 G 存在，u 是 G 中顶点的位序；操作结果是返回 u 对应顶点的第一个邻接顶点位序，如果 u 的顶点没有邻接顶点，否则返回其它表示“不存在”的信息；
6) 获得下一邻接点：函数名称是 NextAdjVex(G, v, w)；初始条件是图 G 存在，v 和 w 是 G 中两个顶点的位序，v 对应 G 的一个顶点,w 对应 v 的邻接顶点；操作结果是返回 v 的（相对于 w）下一个邻接顶点的位序，如果 w 是最后

一个邻接顶点，返回其它表示“不存在”的信息；

7) 插入顶点：函数名称是 InsertVex(G,v)；初始条件是图 G 存在，v 和 G 中的顶点具有相同特征；操作结果是在图 G 中增加新顶点 v。（在这里也保持顶点关键字的唯一性）；
8) 删除顶点：函数名称是 DeleteVex(G,v)；初始条件是图 G 存在，v 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中删除关键字 v 对应的顶点以及相关的弧；
9) 插入弧：函数名称是 InsertArc(G,v,w)；初始条件是图 G 存在，v、w 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中增加弧 <v,w>，如果图 G 是无向图，还需要增加 <w,v>；
10) 删除弧：函数名称是 DeleteArc(G,v,w)；初始条件是图 G 存在，v、w 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中删除弧 <v,w>，如果图 G 是无向图，还需要删除 <w,v>；
11) 深度优先搜索遍历：函数名称是 DFSTraverse(G,visit())；初始条件是图 G 存在；操作结果是图 G 进行深度优先搜索遍历，依次对图中的每一个顶点使用函数 visit 访问一次，且仅访问一次；
12) 广度优先搜索遍历：函数名称是 BFSTraverse(G,visit())；初始条件是图 G 存在；操作结果是图 G 进行广度优先搜索遍历，依次对图中的每一个顶点使用函数 visit 访问一次，且仅访问一次。
13) 距离小于 k 的顶点集合：函数名称是 VerticesSetLessThanK(G,v,k)，初始条件是图 G 存在；操作结果是返回与顶点 v 距离小于 k 的顶点集合；
14) 顶点间最短路径和长度：函数名称是 ShortestPathLength(G,v,w); 初始条件是图 G 存在；操作结果是返回顶点 v 与顶点 w 的最短路径的长度；
15) 图的连通分量：函数名称是 ConnectedComponentsNums(G)，初始条件是图 G 存在；操作结果是返回图 G 的所有连通分量的个数；
16) 实现图的文件形式保存：其中，□ 需要设计文件数据记录格式，以高效保存图的数据逻辑结构 (D,R) 的完整信息；□ 需要设计图文件保存和加载操作合理模式。附录 B 提供了文件存取的方法；
17) 实现多个图管理：设计相应的数据结构管理多个图的查找、添加、移除等功能。

2.3 系统实现

我们使用菜单栏的方式对基于邻接表的图进行实现。具体设计是，每次先输入所选用的操作名称，然后输入该操作所需要的操作，以此对邻接表进行实现。

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

2.3.1 创建图

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

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

请参考算法2.1

算法 2.1. 创建图

Input: Sequence of (V, E)

Output: Initialized graph

procedure CreateCraph(G)
  for **do** $i := 1..n$
    $Initialize(V[i])$
  end for
  for **do** $i < j$
    if **then** $V[i] == V[j]$
      return $ERROR$
    end if
  end for
  for **do** $i := 1..e$
    $AddEdge(V[u], v)$
    $AddEdge(V[v], u)$
  end for

$G.vexnum \leftarrow n$

$G.arcnum \leftarrow e$

return OK

end procedure=0

该算法空间复杂度为 $O(n+e)$，时间复杂度为 $O(n+e)$.

2.3.2 销毁图

函数名称是 DestroyGraph(G)；初始条件图 G 已存在；操作结果是销毁图 G；

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

该算法空间复杂度为 $O(n+e)$，时间复杂度为 $O(1)$.

2.3.3 查找顶点

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

该算法的设计思想是，一一遍历顶点，并匹配 u 与各个顶点的关键字。如果相等，则返回相应的位序；如果遍历结束均未找到，返回-1 作为未找到的标识。

请参考算法2.2

算法 2.2. 查找顶点

Input: Sequence of (V, E)

Output: int

procedure LocateVex(G)

for **do**$i := 1..n$

if **then**$V[i].key == u$

return i

```
        end if
    end for
    return −1
end procedure
```

该算法空间复杂度为 $O(n)$，时间复杂度为 $O(1)$.

2.3.4 顶点赋值

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

该算法的设计思想是，先利用查找结点找到该结点，而后将该结点的顶点值进行修改即可。实现简单，只“堆砌”即可，亦不赘述。

该算法空间复杂度为 $O(1)$，时间复杂度为 $O(n)$.

2.3.5 获得第一邻接点

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

该算法的设计思想是，先找到该结点，随后直接返回该点的第一邻接点即可。亦简单堆砌即可。

该算法空间复杂度为 $O(1)$，时间复杂度为 $O(n)$.

2.3.6 获得下一邻接点

函数名称是 NextAdjVex(G, v, w)；初始条件是图 G 存在，v 和 w 是 G 中两个顶点的位序，v 对应 G 的一个顶点,w 对应 v 的邻接顶点；操作结果是返回 v 的（相对于 w）下一个邻接顶点的位序，如果 w 是最后一个邻接顶点，返回其它表示“不存在”的信息；

该算法的设计思想是，先利用查找顶点找到结点，而后遍历该顶点的边链表找到 w 的位序，并返回下一邻接点。

请参考算法2.3

算法 2.3. 获得下一邻接点

Input: Sequence of (V, E)

Output: int

procedure NextAdjVex(G)

$u \leftarrow LocateVex(v)$

for do$i = 1..V[u].arcnum$

if then$V[u][i] == w$

return $V[u][i+1]$

end if

end for

return -1

end procedure

该算法空间复杂度为 $O(1)$，时间复杂度为 $O(n + E[u]) = O(n)$. 这是因为图 G 为简单图，每个顶点上边的个数小于等于 n，有 $O(E[u]) = O(n)$.

2.3.7 插入顶点

函数名称是 InsertVex(G,v)；初始条件是图 G 存在，v 和 G 中的顶点具有相同特征；操作结果是在图 G 中增加新顶点 v。（在这里也保持顶点关键字的唯一性）；

该算法的设计思想是，直接在最末处添加结点，并将顶点个数增加一即可。

大概的算法参考如下。

算法 2.4. 插入顶点

Input: Sequence of (V, E)

Output: int

procedure InsertVex(G)

$u \leftarrow LocateVex(v)$

if then$u.exists$

```
        return ERROR
    end if
    G.vertices[++G.vexnum] ← V
    return OK
end procedure
```

该算法空间复杂度为 $O(1)$，时间复杂度为 $O(1)$.

2.3.8 删除顶点

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

该算法的设计思想是，先找到需删除的顶点 x 的位置，将顶点删除之后，将后续的顶点依次向前挪动一格；对于边的记录，只需把关键字等于 x 的边删去，再把关键字大于 x 的顶点减去 1 即可。

具体的实现参考代码。

该算法空间复杂度为 $O(1)$，时间复杂度为 $O(n+e)$.

2.3.9 插入弧

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

该算法的设计思想是，将顶点 v 和顶点 w 找到，并且进行加边操作。

具体的实现参考代码。

该算法空间复杂度为 $O(1)$，时间复杂度为 $O(n)$.

2.3.10 删除弧

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

算法如下：

算法输入：图 G, 边信息

算法输出：图 G'

算法流程：

1. 找到结点 w；
2. 遍历结点 w 的弧链表，查找与 v 相等的结点；
3. 如果（2）找到，则删去；否则返回错误；
4. 对结点 v 重复（1）（2）（3）流程；
5. 返回正确结果。

算法时间复杂度：

$O(n)$.

2.3.11 深度优先搜索遍历

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

该算法的设计思想是，对图 G 进行深度优先搜索。换言之，我们对图的遍历是以深度为第一优先级的。

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

我们将其算法附下。

算法 2.5. 深度优先搜索遍历

Input: Sequence of (V, E)

Output: Search Sequence

procedure DFSTraverse(G, u)

$uvisited.$

for do$Each(u, v)inG.u$

if then$vnotvisited$

$DFSTraverse(G, v)$

end if

end for

end procedure

因为每一结点仅被访问一次，故深度优先搜索的时间复杂度为 $O(n)$.

2.3.12 广度优先搜索遍历

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

该算法的设计思想是，对图 G 进行广度优先搜索。换言之，我们对图的遍历是以广度为第一优先级的。

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

我们将其算法附下。

算法 2.6. 广度优先搜索遍历

Input: Sequence of (V, E)

Output: Search Sequence

procedure DFSTraverse(G, u)

$uvisited.$

while do$Qnotempty$

$u \leftarrow Q.front$

$Mark(u)$

for do$each(u, v)inG$

```
            if thenvunvisited
                Q.push(v)
            end if
        end for
    end while
end procedure
```

因为每一结点仅被访问一次，故深度优先搜索的时间复杂度为 $O(n)$.

2.3.13 距离小于 k 的顶点集合

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

算法简单，只需对连通矩阵进行 k 次方，如果非零即为距离小于 k 的顶点。

2.3.14 顶点间最短路径和长度

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

算法如下：

// 狄杰斯特拉算法

S.empty

Q=V0

While (!Q.empty)

U=min(Q)

S+=U

For each v in (u, v)

Relax(u, v, w)

狄杰斯特拉算法中，每个结点和边仅被搜索一次，可以得到该算法时间复杂度为 $O(nlogn)$.

2.3.15 图的连通分量

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

我们使用 tarjan 算法进行图的连通分量的求得。具体算法如下。

// tarjan

Tarjan(u)

Dfn[u] = low[u] = ++idx

Stack.push(u)

For each (u, v) in E

If (v is not visited)

Tarjan(v)

Low[u] = min(low[u], low[v])

Else if (v in stack)

Low[u] = min(low[u], dfn[v])

If (dfn[u] == low[u])

Repeat

V = stack.pop

Print v

Until (u == v)

由于 tarjan 算法实际上是一种 dfs 算法，仅将图遍历了一遍，故其时间复杂度为 $O(n+e)$.

2.3.16 实现图的文件形式保存

其中，□ 需要设计文件数据记录格式，以高效保存图的数据逻辑结构 (D,R) 的完整信息；□ 需要设计图文件保存和加载操作合理模式。附录 B 提供了文件存取的方法；

我们设计的储存结构是，将其顶点依次存下，在将其边存下即可。

2.3.17 实现多个图管理

设计相应的数据结构管理多个图的查找、添加、移除等功能。

只需创建多图结构数组，后将其实现增加删除创建等函数即可。

2.4 系统测试

本次实验使用 CodeBlocks 进行编写。我们将实验划分为三个文件协同进行编译，分别为”main.cpp”, ”def.h”, ”opt.h”。这三个文件分别为主程序文件、数据结构定义文件和操作文件。我们由主程序文件调用数据结构定义文件和操作文件。即，”def.h” 和”opt.h” 从属于”main.cpp”.

2.4.1 测试计划

为便于浏览并且提纲挈领，我们对测试计划绘制了表格如下。其中斜线表示输入为空或者无输出。

表 2-1 “基于邻接表的图实现”测试计划表.

序号	操作	输入	预期输出
1	CreateCraph	如图示	/
2	LocateVex	6	3
3	InsertVex	9 456	/
4	ConnectedComponentsNums	/	2
5	InsertArc	8 9	/
6	DFSTraverse	/	如图示
7	VerticesSetLessThanK	5 1	0
8	DeleteVex	5	/
9	BFSTraverse	/	如图示
10	SaveGraph	cst.dat	/
11	DestroyGraph	/	/
12	LoadGraph	cst.dat	/
13	BFSTraverse	/	如图示
14	AddGraph	1	/
15	InitNewGraph	如图示	/

2.4.2 实际测试

我们依据上表对程序进行测试。

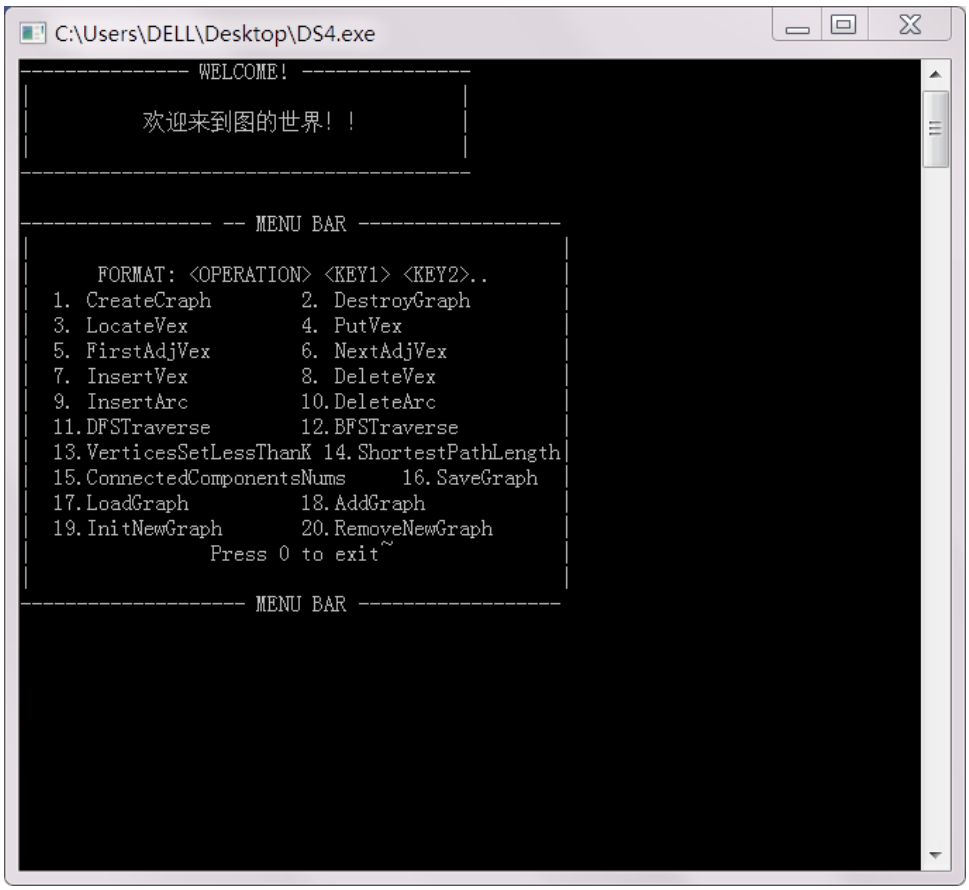

图 2-1 命令行初始界面

图 2-1 为我们开始使用实验程序的初始界面。

下面我们一一对程序的使用进行展示与说明。

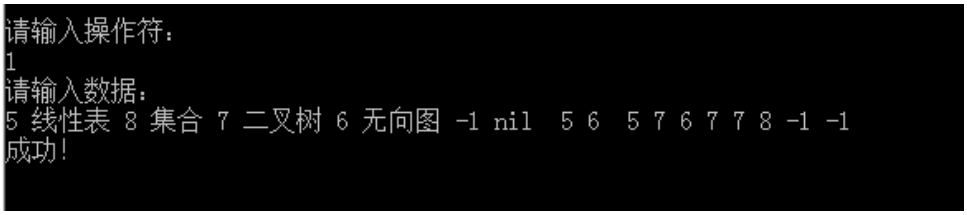

图 2-2 新建图

如图 2-2，新建成功！

如图 2-3，关键字为 6 的位序为 3，正确！

如图 2-4，插入新节点，关键词为 9，名称为“456”，插入成功！

如图 2-5，求得该图连通分量为 2，是正确的答案。

如图 2-6，插入数据成功.

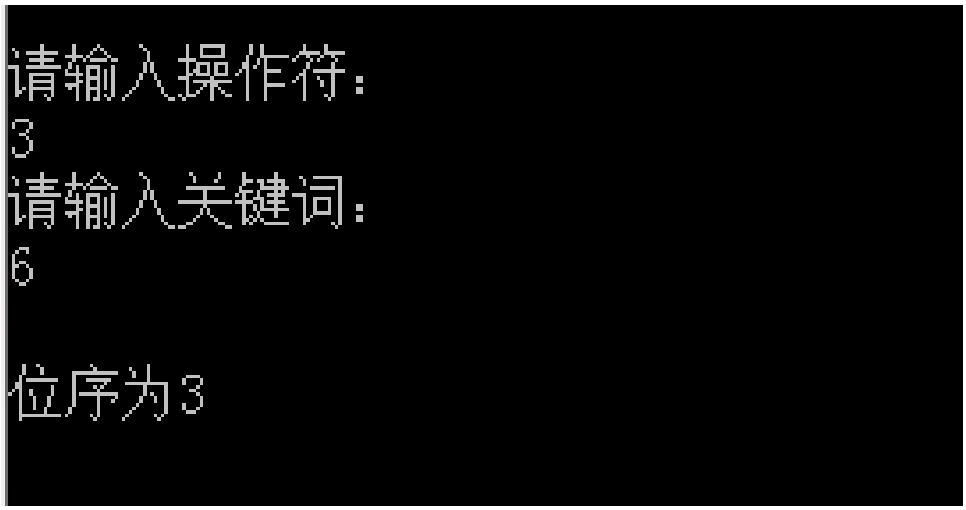

图 2-3　返回位序

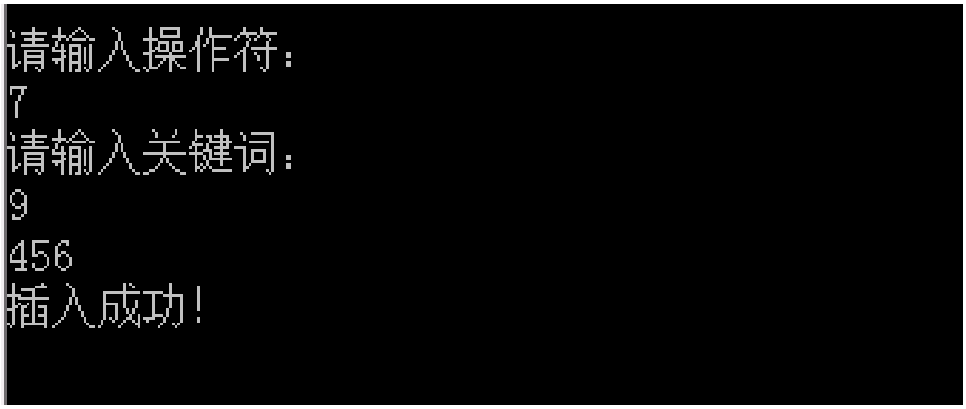

图 2-4　插入结点

如图 2-7，遍历结果显示，插入弧成功。

如图 2-8，距离小于 1 的只有它自身，正确！

如图 2-9，删除成功！

如图 2-10，遍历结果说明删除成功！

如图 2-11，已储存至“cst.dat”

如图 2-12，图已销毁。

如图 2-13，读取成功。

如图 2-14，遍历结果说明读取确实成功。

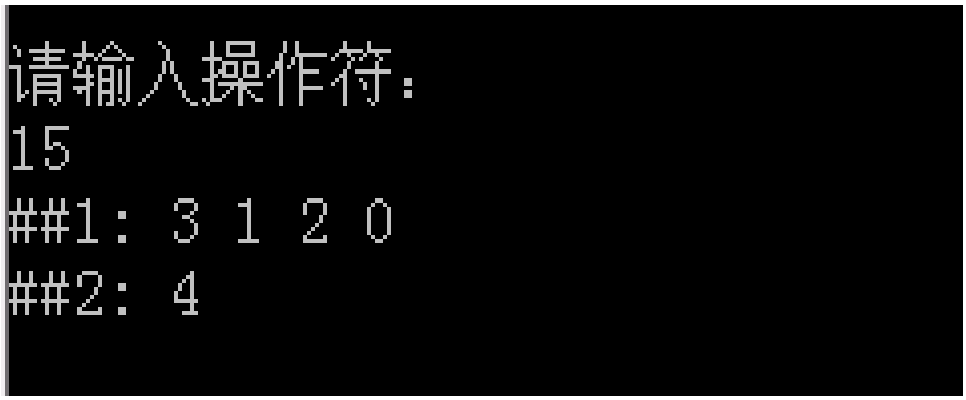

图 2-5　求连通分量

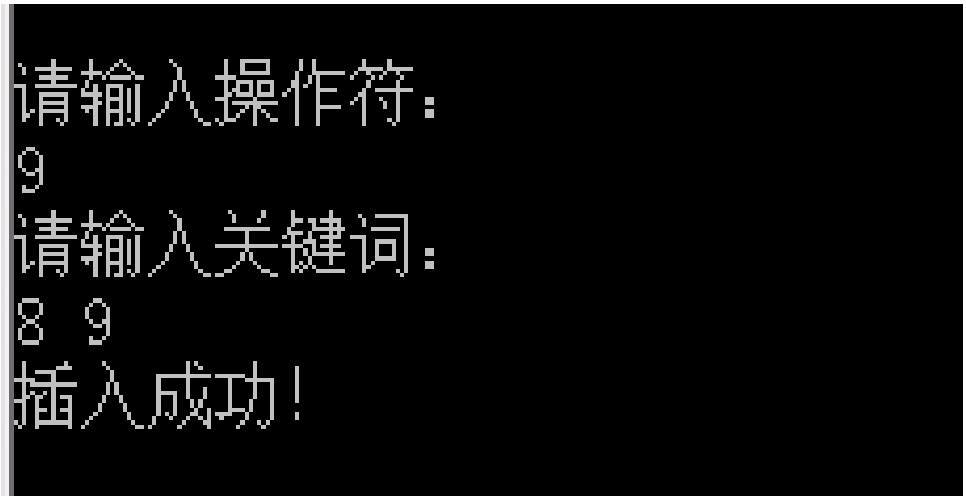

图 2-6　插入弧

如图 2-15，增加图顺利。

图 2-7　遍历结果

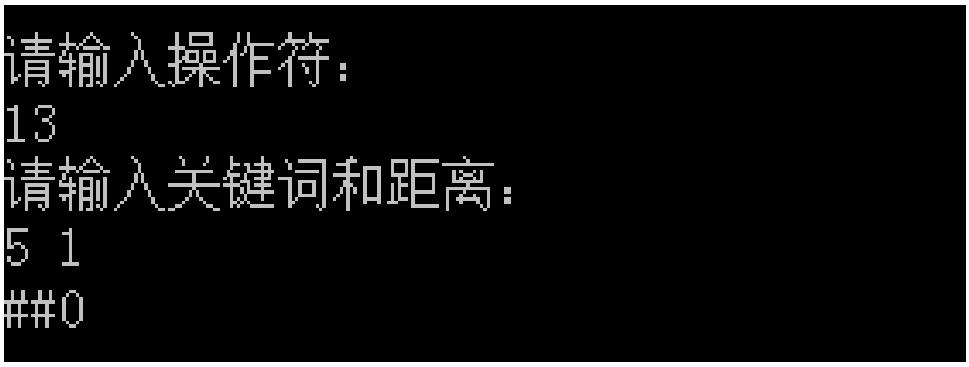

图 2-8　距离小于 k 试验

如图 2-16，可以实现多图的初始化。

2.5　实验小结

图的邻接表构建相对较难，需要写多个函数，并且指针指向也值得玩味。极有意义。

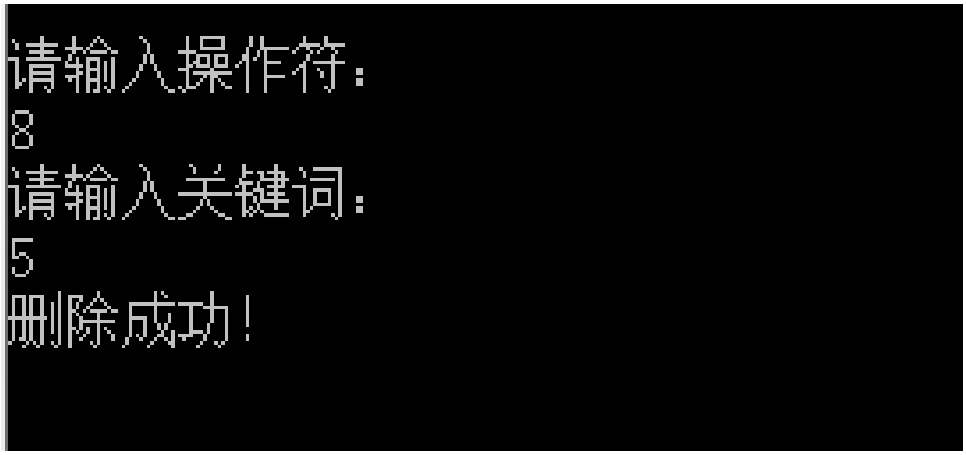

图 2-9　删除结点

图 2-10　遍历结果

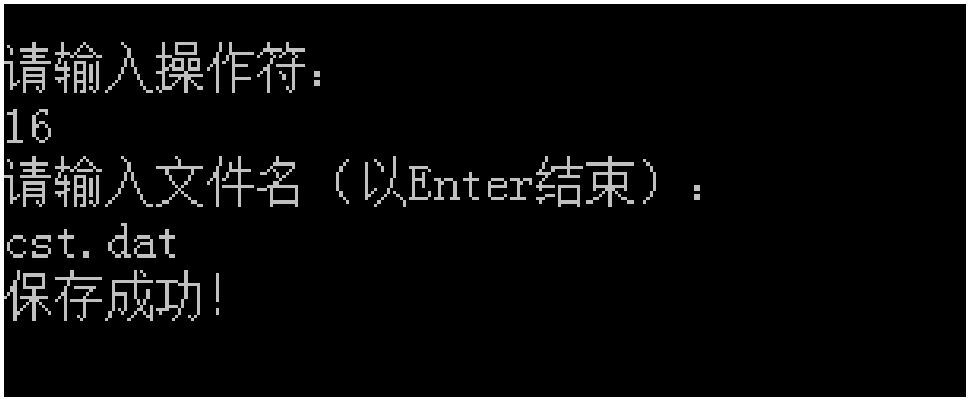

图 2-11　储存至文件

图 2-12　销毁图

图 2-13　读取文件

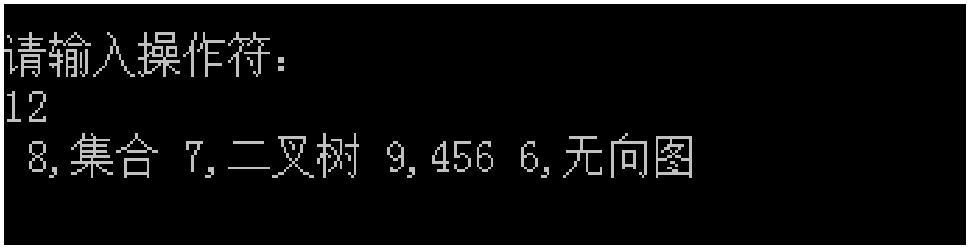

图 2-14　遍历结果

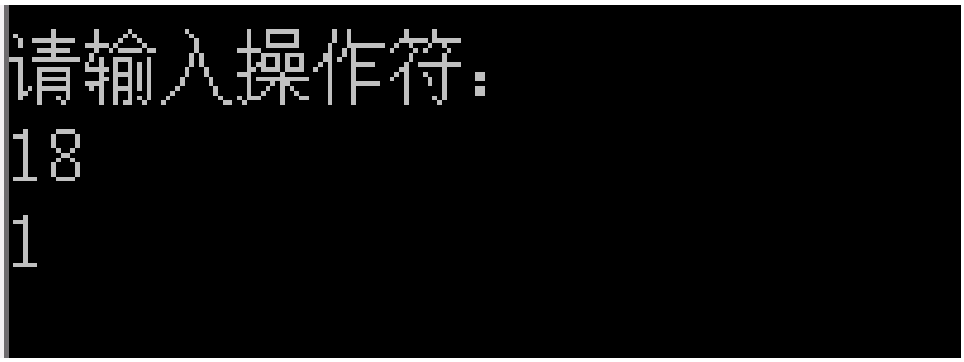

图 2-15　增加图

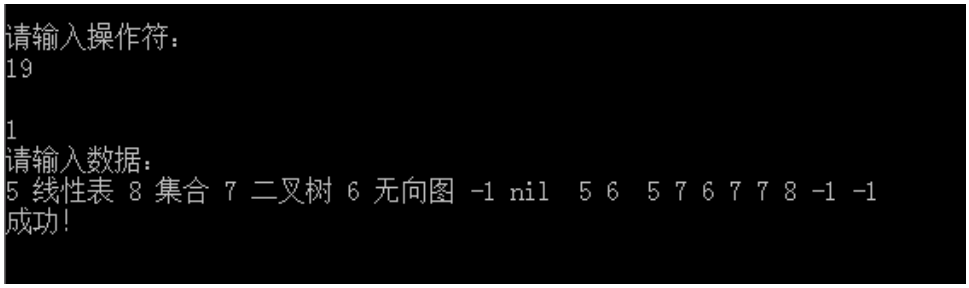

图 2-16　初始化多图

3 课程的收获和建议

通过学习数据结构实验，我深刻理解了不同数据结构的不同具体实现，并加深了对这四种数据结构的认识。我们深入线性表储存的设计，我们纠结链表指针的指向改变，我们对二叉树左孩子和右孩子的遍历深入探寻，我们对邻接表的点和弧进行增删补漏。通过这四则基本的数据结构设计，我们增进了认识和理解，我们动了手，长了知识。

3.1 基于顺序存储结构的线性表实现

顺序存储结构实验较为简单，只需通过数组和动态分配线性物理空间进行线性查找即可。我建议减少该专题课时，将时间用于更具有挑战性的链式存储结构的线性表的结构设计。我在这一专题内使用了 LaTeX 进行了实验报告的撰写。在之前我并没有使用过 LaTeX，LaTeX 是一种优美的文稿处理工具，严肃严谨而直接并较为浅易，兼具优美性和易用性。但从另一方面，因为本学期课业繁重，事务繁忙，难以抽出更多时间进行对于 LaTeX 的具体功能进行全面具体的学习，故对于 LaTeX 我仅是浅尝辄止，囫囵吞枣，并无全面立体的了解！在实验报告中，对于算法的撰写，有所缺漏，不尽歉意！

3.2 基于邻接表的图实现

邻接表的编码和实现相对较难，尤其几个附加实验诸如连通分量的计算之类需要掌握特定算法才能求得。然而，这么多的任务仅仅给了我们两次课的时间加班加点地完成，实在不够！我建议增加该专题的课时。图和二叉树的构建本就较难，时间却和两个线性表完全一样，我认为这是需要调整的！

4 附录 A 基于顺序存储结构线性表实现的源程序

/* Linear Table On Sequence Structure */

```
/* In File "def.h" */


#ifndef DEF_H_INCLUDED
#define DEF_H_INCLUDED

#include "stdio.h"
#include "stdlib.h"
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2

typedef int status;
typedef int ElemType; //数据元素类型定义

#define LIST_INIT_SIZE 100
#define LISTINCREMENT  10
typedef int ElemType;
typedef struct{  //顺序表（顺序结构）的定义
      ElemType * elem;
      int length;
      int listsize;
         }SqList;
```

```
typedef struct{  //线性表的管理表定义
     struct { char name[30];
        SqList L;
      } elem[10];
      int length;
      int listsize;
 }LISTS;

#endif // DEF_H_INCLUDED



/* In File "opt.h" */


#ifndef OPT_H_INCLUDED
#define OPT_H_INCLUDED

#include <string.h>
#include <math.h>

status InitList(SqList& L)
// 线性表L不存在，构造一个空的线性表，返回OK，否则返回INFEASIBLE。
{
    if (L.elem != NULL){
        return INFEASIBLE;
    }
    L.elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof (ElemType));
    if (!L.elem) exit(OVERFLOW);
    L.length = 0;
    L.listsize = LIST_INIT_SIZE;
```

```
    return OK;
}

status DestroyList(SqList& L)
// 如果线性表L存在，销毁线性表L，释放数据元素的空间，返回OK，否则返回INFEASIBLE。
{
    if (!L.elem) return INFEASIBLE;
    free(L.elem);
    L.elem = NULL;
    L.length = 0;
    L.listsize = 0;
    return OK;
}

status ClearList(SqList& L)
// 如果线性表L存在，删除线性表L中的所有元素，返回OK，否则返回INFEASIBLE。
{
    if (!L.elem) return INFEASIBLE;
    L.length = 0;
    return OK;
}

status ListEmpty(SqList L)
// 如果线性表L存在，判断线性表L是否为空，空就返回TRUE，否则返回FALSE；如果线性表L不
{
    if (!L.elem) return INFEASIBLE;
    if (L.length) return FALSE;
    return TRUE;
}

status ListLength(SqList L)
```

```
// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。
{
    if (!L.elem) return INFEASIBLE;
    return L.length;
}

status GetElem(SqList L,int i,ElemType &e)
// 如果线性表L存在，获取线性表L的第i个元素，保存在e中，返回OK；如果i不合法，返回ER
{
    if (!L.elem) return INFEASIBLE;
    if (i < 1 || i > L.length) return ERROR;
    e = *(L.elem + i - 1);
    return OK;
}

int LocateElem(SqList L,ElemType e)
// 如果线性表L存在，查找元素e在线性表L中的位置序号并返回该序号；如果e不存在，返回0
{
    if (!L.elem) return INFEASIBLE;
    for (int i = 0; i < L.length; ++i){
        if (*(L.elem + i) == e)
            return i + 1;
    }
    return 0;
}

status PriorElem(SqList L,ElemType e,ElemType &pre)
// 如果线性表L存在，获取线性表L中元素e的前驱，保存在pre中，返回OK；如果没有前驱，
{
    if (!L.elem) return INFEASIBLE;
    for (int i = 1; i < L.length; ++i){
```

```
        if (*(L.elem + i) == e) {
            pre = *(L.elem + i - 1);
            return OK;
        }
    }
    return ERROR;
}

status NextElem(SqList L,ElemType e,ElemType &next)
// 如果线性表L存在，获取线性表L元素e的后继，保存在next中，返回OK；如果没有后继，返
{
    if (!L.elem) return INFEASIBLE;
    for (int i = 0; i < L.length - 1; ++i){
        if (*(L.elem + i) == e) {
            next = *(L.elem + i + 1);
            return OK;
        }
    }
    return ERROR;
}

status ListInsert(SqList &L,int i,ElemType e)
// 如果线性表L存在，将元素e插入到线性表L的第i个元素之前，返回OK；当插入位置不正确时
{
    if (!L.elem) return INFEASIBLE;
    if (i < 1 || i > L.length + 1) return ERROR;
    if (L.length >= L.listsize){
        ElemType *newbase = (ElemType * )realloc(L.elem, (L.listsize + LISTINCREME
        if (!newbase) exit(OVERFLOW);
        L.elem = newbase;
        L.listsize += LISTINCREMENT;
```

```
    }
    ElemType *q = L.elem + i - 1;
    for (ElemType *p = L.elem + L.length - 1; p >= q; --p)
        *(p + 1) = *p;
    *q = e;
    ++L.length;
    return OK;
}

status ListDelete(SqList &L,int i,ElemType &e)
// 如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回OK；当删除位置不正确时
{
    if (!L.elem) return INFEASIBLE;
    if (i < 1 || i > L.length) return ERROR;
    ElemType *p = L.elem + i - 1;
    e = *p;
    for (ElemType *q = L.elem + L.length - 1; p < q; ++p)
        *p = *(p + 1);
    --L.length;
    return OK;
}

status ListTraverse(SqList L)
// 如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，返回OK；如果线性表L不
{
    if (!L.elem) return INFEASIBLE;
    ElemType *p = L.elem, *q = L.elem + L.length - 1;
    printf(">>> #");
    for (; p < q; ++p)
        printf("%d ", *p);
    if (L.length) printf("%d", *q);
```

```
    printf("\n");
    return OK;
}

status  SaveList(SqList L,char FileName[])
// 如果线性表L存在，将线性表L的的元素写到FileName文件中，返回OK，否则返回INFEASIBL
{
    if (!L.elem) return INFEASIBLE;
    FILE *fp = fopen(FileName, "w");
    ElemType *p = L.elem, *q = L.elem + L.length;
    //fprintf(fp, "%d\n", L.length);
    for (; p < q; ++p)
        fprintf(fp, "%d ", *p);
    fprintf(fp, "0");
    fclose(fp);
    return OK;
}
status  LoadList(SqList &L,char FileName[])
// 如果线性表L不存在，将FileName文件中的数据读入到线性表L中，返回OK，否则返回INFEA
{
    //if (L.elem) return INFEASIBLE;
    // ERROR!
    FILE *fp = fopen(FileName, "r");
    L.length = 0;
    L.listsize = 100;
    L.elem = (ElemType * )malloc(LIST_INIT_SIZE * sizeof(ElemType));
    ElemType q;
    fscanf(fp, "%d", &q);
    while (q){
        L.elem[L.length++] = q;
        fscanf(fp, "%d", &q);
```

```
    }
    fclose(fp);
    return OK;
}

status AddList(LISTS &Lists,char ListName[])
// 只需要在Lists中增加一个名称为ListName的空线性表，线性表数据又后台测试程序插入。
{
    if (Lists.length > 10) return ERROR;
    strcpy((Lists.elem + Lists.length)->name, ListName);
    SqList Lp;
    Lp.elem = NULL;
    if (InitList(Lp) == ERROR) return ERROR;
    (Lists.elem + Lists.length)->L = Lp;
    ++Lists.length;
    return OK;
}

status RemoveList(LISTS &Lists,char ListName[])
// Lists中删除一个名称为ListName的线性表
{
    for (int i = 0; i < Lists.length; ++i){
        if (strcmp(Lists.elem[i].name, ListName) == 0){
            for (int j = i + 1; j < Lists.length; ++j){
                strcpy(Lists.elem[j - 1].name, Lists.elem[j].name);
                Lists.elem[j - 1].L = Lists.elem[j].L;
            }
            Lists.length--;
            return OK;
        }
    }
```

```
    return ERROR;
}

int LocateList(LISTS Lists,char ListName[])
// 在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，否则返回0
{
    for (int i = 0; i < Lists.length; ++i){
        if (strcmp(Lists.elem[i].name, ListName) == 0) return i + 1;
    }
    return 0;
}



int MaxSubArray(SqList &L){
    //返回最大子数列和
    if (!L.elem) return INFEASIBLE;
    int sum[100];
    memset(sum, 0, sizeof sum);
    sum[0] = L.elem[0];
    int maxn = 0;
    for (int i = 1; i < L.length; ++i){
        sum[i] = sum[i - 1] + L.elem[i];
    }
    for (int i = 0; i < L.length; ++i)
        for (int j = i; j < L.length; ++j)
            if (maxn < sum[j] - sum[i]) maxn = sum[j] - sum[i];
    return maxn;
}

int SubArrayNum(SqList &L, int k){
```

```
    //返回和为k的连续子数组的个数
    if (!L.elem) return INFEASIBLE;
    int sum[100];
    memset(sum, 0, sizeof sum);
    sum[0] = L.elem[0];
    int cnt = 0;
    for (int i = 1; i < L.length; ++i){
        sum[i] = sum[i - 1] + L.elem[i];
    }
    for (int i = 0; i < L.length; ++i)
        for (int j = i; j < L.length; ++j)
            if (sum[j] - sum[i] == k) ++cnt;
    return cnt;
}

int sortList(SqList &L){
    //按升序排序线性表L
    if (!L.elem) return INFEASIBLE;
    for (int i = 0; i < L.length; ++i){
        int t = i;
        for (int j = i + 1; j < L.length; ++j)
            if (L.elem[t] > L.elem[j]) t = j;
        int qw;
        qw = L.elem[t];
        L.elem[t] = L.elem[i];
        L.elem[i] = qw;
    }
    return OK;
}

#endif // OPT_H_INCLUDED
```

```
/* In File "main.cpp" */


#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include "def.h"
#include "opt.h"

LISTS Lists;//总表

void menuBar(){
    //system("cls");
    //list_read!
    printf("--------------- MENU BAR ---------------\n");
    printf("|                                       |\n");
    printf("|  FORMAT: <OPERATION> <KEY1> <KEY2>.. |\n");
    printf("|  1. LISTINIT L     2. LISTDESTROY L  |\n");
    printf("|  3. LISTCLR L      4. LISTEMPTY L    |\n");
    printf("|  5. LISTLEN L      6. GETELEM L I    |\n");
    printf("|  7. LOCATEELEM L E 8. ELEMPRIOR L E  |\n");
    printf("|  9. ELEMNEXT L E   10.LISTINS L I E  |\n");
    printf("|  11.LISTDEL L I    12.SHOWLIST L     |\n");
    printf("|  13.MENU           14.MAXSUBARRAY L  |\n");
    printf("|  15.SUBARRAYNUM L  16.LISTSORT L     |\n");
    printf("|  17.SAVELIST L F   18.LOADLIST L F   |\n");
    printf("|  19.EXIT                             |\n");
```

```
    printf("|                                          |\n");
    printf("--------------- MENU BAR ---------------\n\n");
}

void initialBar(){
    printf("--------------- WELCOME! ---------------\n");
    printf("|                                          |\n");
    printf("|    EXPERIMENT OF LINEAR SEQUENCE.    |\n");
    printf("|                                          |\n");
    printf("------------------------------------------\n\n");
}

int main()
{
    initialBar();
    menuBar();
    char opt[80], opt1[80], opt2[80], opt3[80], state;
    printf(">>> ");
    scanf("%s", opt);
    while(strcmp(opt, "EXIT") != 0){
        if (strcmp(opt, "LISTINIT") == 0){
            scanf("%s", opt1);
            state = AddList(Lists, opt1);
            if (state == ERROR) printf(">>> ERROR: INVALID LIST! \n");
        }
        else if (strcmp(opt, "LISTDESTROY") == 0){
            scanf("%s", opt1);
            state = RemoveList(Lists, opt1);
            if (state == ERROR) printf(">>> ERROR: LISTNAME NOT FOUND! \n");
        }
        else if (strcmp(opt, "LISTCLR") == 0){
```

```
        scanf("%s", opt1);
        state = LocateList(Lists, opt1);
        if (state == 0) {
            printf(">>> ERROR: LISTNAME NOT FOUND! \n");
            goto PPP;
        }
        int p = state - 1;
        int q = ClearList(Lists.elem[p].L);
        if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
    }
    else if (strcmp(opt, "LISTEMPTY") == 0){
        scanf("%s", opt1);
        state = LocateList(Lists, opt1);
        if (state == 0) {
            printf(">>> ERROR: LISTNAME NOT FOUND! \n");
            goto PPP;
        }
        int p = state - 1;
        int q = ListEmpty(Lists.elem[p].L);
        if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
        else if (q == TRUE) printf(">>> TRUE! \n");
        else if (q == FALSE) printf(">>> FALSE! \n");
    }
    else if (strcmp(opt, "LISTLEN") == 0){
        scanf("%s", opt1);
        state = LocateList(Lists, opt1);
        if (state == 0) {
            printf(">>> ERROR: LISTNAME NOT FOUND! \n");
            goto PPP;
        }
        int p = state - 1;
```

```
        int q = ListLength(Lists.elem[p].L);
        if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
        else printf(">>> %s LEN is %d \n", Lists.elem[p].name, q);
    }
    else if (strcmp(opt, "GETELEM") == 0){
        scanf("%s", opt1);
        scanf("%s", opt2);
        state = LocateList(Lists, opt1);
        int tmp = atoi(opt2);
        if (tmp == 0) {
            printf(">>> ERROR: INVALID NUMBER! \n");
            goto PPP;
        }
        if (state == 0) {
            printf(">>> ERROR: LISTNAME NOT FOUND! \n");
            goto PPP;
        }
        int p = state - 1;
        int ee;
        int q = GetElem(Lists.elem[p].L, tmp, ee);
        if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
        if (q == ERROR) printf(">>> ERROR: INVALID NUMBER! \n");
        else printf(">>> Elem is %d \n", ee);
    }
    else if (strcmp(opt, "LOCATEELEM") == 0){
        scanf("%s", opt1);
        scanf("%s", opt2);
        state = LocateList(Lists, opt1);
        int tmp = atoi(opt2);
        if (tmp == 0) {
            printf(">>> ERROR: INVALID NUMBER! \n");
```

```
            goto PPP;
        }
        if (state == 0) {
            printf(">>> ERROR: LISTNAME NOT FOUND! \n");
            goto PPP;
        }
        int p = state - 1;
        int q = LocateElem(Lists.elem[p].L, tmp);
        if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
        if (q == 0) printf(">>> NOT FOUND! \n");
        else if (q > 0) printf(">>> Elem is #%d \n", q);
    }
    else if (strcmp(opt, "ELEMPRIOR") == 0){
        scanf("%s", opt1);
        scanf("%s", opt2);
        state = LocateList(Lists, opt1);
        int tmp = atoi(opt2);
        if (tmp == 0) {
            printf(">>> ERROR: INVALID NUMBER! \n");
            goto PPP;
        }
        if (state == 0) {
            printf(">>> ERROR: LISTNAME NOT FOUND! \n");
            goto PPP;
        }
        int p = state - 1;
        int ee;
        int q = PriorElem(Lists.elem[p].L, tmp, ee);
        if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
        if (q == ERROR) printf(">>> ERROR: INVALID NUMBER! \n");
        else if (q > 0) printf(">>> Elem is %d \n", ee);
```

```
}
else if (strcmp(opt, "ELEMNEXT") == 0){
    scanf("%s", opt1);
    scanf("%s", opt2);
    state = LocateList(Lists, opt1);
    int tmp = atoi(opt2);
    if (tmp == 0) {
        printf(">>> ERROR: INVALID NUMBER! \n");
        goto PPP;
    }
    if (state == 0) {
        printf(">>> ERROR: LISTNAME NOT FOUND! \n");
        goto PPP;
    }
    int p = state - 1;
    int ee;
    int q = NextElem(Lists.elem[p].L, tmp, ee);
    if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
    if (q == ERROR) printf(">>> ERROR: INVALID NUMBER! \n");
    else if (q > 0) printf(">>> Elem is %d \n", ee);
}
else if (strcmp(opt, "LISTINS") == 0){
    scanf("%s", opt1);
    scanf("%s", opt2);
    scanf("%s", opt3);
    state = LocateList(Lists, opt1);
    int tmp = atoi(opt2), tmp2 = atoi(opt3);
    if (tmp == 0 || tmp2 == 0) {
        printf(">>> ERROR: INVALID NUMBER! \n");
        goto PPP;
    }
```

```
        if (state == 0) {
            printf(">>> ERROR: LISTNAME NOT FOUND! \n");
            goto PPP;
        }
        int p = state - 1;
        int q = ListInsert(Lists.elem[p].L, tmp, tmp2);
        if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
        if (q == ERROR) printf(">>> ERROR: INVALID NUMBER! \n");
    }
    else if (strcmp(opt, "LISTDEL") == 0){
        scanf("%s", opt1);
        scanf("%s", opt2);
        state = LocateList(Lists, opt1);
        int tmp = atoi(opt2);
        if (tmp == 0) {
            printf(">>> ERROR: INVALID NUMBER! \n");
            goto PPP;
        }
        if (state == 0) {
            printf(">>> ERROR: LISTNAME NOT FOUND! \n");
            goto PPP;
        }
        int p = state - 1;
        int ee;
        int q = ListDelete(Lists.elem[p].L, tmp, ee);
        if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
        if (q == ERROR) printf(">>> ERROR: INVALID NUMBER! \n");
    }
    else if (strcmp(opt, "SHOWLIST") == 0){
        scanf("%s", opt1);
        state = LocateList(Lists, opt1);
```

```
        if (state == 0) {
            printf(">>> ERROR: LISTNAME NOT FOUND! \n");
            goto PPP;
        }
        int p = state - 1;
        int q = ListTraverse(Lists.elem[p].L);
        if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
    }
    else if (strcmp(opt, "MENU") == 0){
        menuBar();
    }
    else if (strcmp(opt, "MAXSUBARRAY") == 0){
        scanf("%s", opt1);
        state = LocateList(Lists, opt1);
        if (state == 0) {
            printf(">>> ERROR: LISTNAME NOT FOUND! \n");
            goto PPP;
        }
        int p = state - 1;
        int q = MaxSubArray(Lists.elem[p].L);
        if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
        else printf(">>> MAX = %d \n", q);
    }
    else if (strcmp(opt, "SUBARRAYNUM") == 0){
        scanf("%s", opt1);
        scanf("%s", opt2);
        state = LocateList(Lists, opt1);
        int tmp = atoi(opt2);
        if (tmp == 0) {
            printf(">>> ERROR: INVALID NUMBER! \n");
            goto PPP;
```

```
        }
        if (state == 0) {
            printf(">>> ERROR: LISTNAME NOT FOUND! \n");
            goto PPP;
        }
        int p = state - 1;
        int q = SubArrayNum(Lists.elem[p].L, tmp);
        if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
        else printf(">>> COUNT = %d \n", q);
    }
    else if (strcmp(opt, "LISTSORT") == 0){
        scanf("%s", opt1);
        state = LocateList(Lists, opt1);
        if (state == 0) {
            printf(">>> ERROR: LISTNAME NOT FOUND! \n");
            goto PPP;
        }
        int p = state - 1;
        int q = sortList(Lists.elem[p].L);
        if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
        else printf(">>> SORT SUCCEEDED! \n");
    }
    else if (strcmp(opt, "SAVELIST") == 0){
        scanf("%s", opt1);
        scanf("%s", opt2);
        state = LocateList(Lists, opt1);
        if (state == 0) {
            printf(">>> ERROR: LISTNAME NOT FOUND! \n");
            goto PPP;
        }
        int p = state - 1;
```

```
            int q = SaveList(Lists.elem[p].L, opt2);
            if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
            else printf(">>> SAVED! \n");
        }
        else if (strcmp(opt, "LOADLIST") == 0){
            scanf("%s", opt1);
            scanf("%s", opt2);
            state = LocateList(Lists, opt1);
            if (state == 0) {
                printf(">>> ERROR: LISTNAME NOT FOUND! \n");
                goto PPP;
            }
            int p = state - 1;
            int q = LoadList(Lists.elem[p].L, opt2);
            if (q == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
            else printf(">>> READ! \n");
        }

        else printf(">>> INVALID INPUT_OPERATION! \n");
    PPP:
        memset(opt, 0, sizeof opt);
        printf(">>> ");
        scanf("%s", opt);
    }
    printf("-------- THANK YOU FOR USING! --------");
    return 0;
}
```

5 附录 B 基于邻接表图实现的源程序

/* Graph Structure */

```
/* In File "def.h" */


#ifndef DEF_H_INCLUDED
#define DEF_H_INCLUDED

    #include "stdio.h"
#include "stdlib.h"
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define MAX_VERTEX_NUM 20
typedef int status;
typedef int KeyType;
   typedef enum {DG,DN,UDG,UDN} GraphKind;
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
         GraphKind  kind;         //图的类型
        } ALGraph;

#endif // DEF_H_INCLUDED



/* In File "opt.h" */


#ifndef OPT_H_INCLUDED
#define OPT_H_INCLUDED


#include <queue>
#include <cstring>

using namespace std;

status CreateCraph(ALGraph &G,VertexType V[],KeyType VR[][2])
/*根据V和VR构造图T并返回OK，如果V和VR不正确，返回ERROR
如果有相同的关键字，返回ERROR。此题允许通过增加其它函数辅助实现本关任务*/
{
```

```
// 请在这里补充代码，完成本关任务
/********** Begin *********/
int p = 0;
//G.vertices=(VNode*)malloc(MAX_VERTEX_NUM*sizeof(VNode));
while(V[p].key!=-1){
    G.vertices[p].data = V[p];
    G.vertices[p].firstarc=NULL;
    for (int i = 0; i < p; ++i)
        if (V[p].key == V[i].key) return ERROR;
    ++p;
    if (p>20) return ERROR;
}
if (p==0) return ERROR;
//return OK;//1
int t = 0;
while (VR[t][0] != -1){
    int u=-1;
    for (int i = 0; i < p; ++i){
        if (V[i].key == VR[t][0]) {
            u=i;
            break;
        }
    }
    if (u==-1) return ERROR;
    int v=-1;
    for (int i = 0; i < p; ++i){
        if (V[i].key == VR[t][1]) {
            v=i;
            break;
        }
    }
```

```
        if (v==-1) return ERROR;
            ArcNode* ps=G.vertices[u].firstarc,*pp;
            pp=(ArcNode *)malloc(sizeof(ArcNode));
            pp->adjvex=v;
            pp->nextarc=ps;
            G.vertices[u].firstarc=pp;

            ArcNode* pt=G.vertices[v].firstarc;
            pp=(ArcNode *)malloc(sizeof(ArcNode));
            pp->adjvex=u;
            pp->nextarc=pt;
            G.vertices[v].firstarc=pp;
        //printf("%d,%d  ", u, v);
        ++t;
        //printf("%d",t);
    }
    G.vexnum=p;
    G.arcnum=t;
    G.kind=UDG;
    return OK;
    /********** End **********/
}


status DestroyGraph(ALGraph &G)
/*销毁无向图G,删除G的全部顶点和边*/
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    for (int i = 0;i<G.vexnum;++i){
        ArcNode *pt = G.vertices[i].firstarc,*ps;
```

```
        while (pt){
            ps=pt->nextarc;
            free(pt);
            pt = ps;
        }

    }
    G.vexnum = 0;
    G.arcnum=0;
    return OK;
    /********** End **********/
}

int LocateVex(ALGraph G,KeyType u)
//根据u在图G中查找顶点，查找成功返回位序，否则返回-1;
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    for (int i = 0; i<G.vexnum;++i)
    if (G.vertices[i].data.key==u) return i;
    return -1;

    /********** End **********/
}

status PutVex(ALGraph &G,KeyType u,VertexType value)
//根据u在图G中查找顶点，查找成功将该顶点值修改成value，返回OK;
//如果查找失败或关键字不唯一，返回ERROR
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
```

```
    int l=-1;
    for (int i =0;i<G.vexnum;++i)
        if (G.vertices[i].data.key==u)
            l=i;
    if (l==-1) return ERROR;
    for (int i = 0; i<G.vexnum;++i)
        if (i != l && G.vertices[i].data.key==value.key) return ERROR;
    G.vertices[l].data=value;
    return OK;

    /********** End **********/
}

int FirstAdjVex(ALGraph G,KeyType u)
//根据u在图G中查找顶点，查找成功返回顶点u的第一邻接顶点位序，否则返回-1;
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    for (int i = 0; i < G.vexnum;++i) {
        if (G.vertices[i].data.key == u)
            return G.vertices[i].firstarc->adjvex;
    }
    return -1;
    /********** End **********/
}

int Get_index(ALGraph G,int key)
{
    for(int i=0;i<G.vexnum;i++)
        if(G.vertices[i].data.key==key)
```

```
            return i;
    return -1;
}

int NextAdjVex(ALGraph G,KeyType v,KeyType w)
//根据u在图G中查找顶点，查找成功返回顶点v的邻接顶点相对于w的下一邻接顶点的位序，查
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    int s1=Get_index(G,v);
    int s2=Get_index(G,w);
    if (s1==-1||s2==-1) return -1;
    ArcNode* p=G.vertices[s1].firstarc;
    while(p && p->nextarc){
        if (p->adjvex==s2) return p->nextarc->adjvex;
        p=p->nextarc;
    }
    return -1;


    /********** End **********/
}

status InsertVex(ALGraph &G,VertexType v)
//在图G中插入顶点v，成功返回OK,否则返回ERROR
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    int tmp = v.key;
    for (int i = 0; i < G.vexnum;++i)
        if (tmp==G.vertices[i].data.key) return ERROR;
```

```
    if (G.vexnum==MAX_VERTEX_NUM) return ERROR;
    G.vertices[G.vexnum].data=v;
    G.vertices[G.vexnum].firstarc=NULL;
    ++G.vexnum;
    return OK;

    /********** End **********/
}

status DestroyList(ArcNode * f)
{
    if(f==NULL)return INFEASIBLE;
    ArcNode * p=f;
    while(p)
    {
        f=f->nextarc;
        free(p);
        p=f;
    }
    return OK;
}
void DeleteVex_A_Node(ALGraph &G,VNode &Node,int index)
{
    ArcNode * p =Node.firstarc;
    while(Node.firstarc!=NULL && Node.firstarc->adjvex==index)
    {
        G.arcnum--;
        Node.firstarc=Node.firstarc->nextarc;
        free(p);
        p=Node.firstarc;
    }
```

```
    if(Node.firstarc==NULL)return;
    ArcNode * q =Node.firstarc->nextarc;
    for(;q!=NULL;)
    {
        if(q->adjvex==index)
        {
            G.arcnum--;
            p->nextarc=q->nextarc;
            free(q);
            q=p->nextarc;
        }
        else
        {
            p=p->nextarc;
            q=q->nextarc;
        }
    }
}

status DeleteVex(ALGraph &G,KeyType v)
//在图G中删除关键字v对应的顶点以及相关的弧，成功返回OK,否则返回ERROR
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    int loc=Get_index(G,v);
    if (loc==-1||G.vexnum==1) return ERROR;
    for(int i=0;i<G.vexnum;i++)
    {
        DeleteVex_A_Node(G,G.vertices[i],loc);
    }
    for(int i=0;i<G.vexnum;i++)
```

```
    {
        ArcNode * f=G.vertices[i].firstarc;
        for(;f!=NULL;f=f->nextarc)
        {
            if(f->adjvex>loc)
            {
                (f->adjvex)--;
            }
        }
    }
    DestroyList(G.vertices[loc].firstarc);
    for (int i=loc;i<G.vexnum-1;++i)
        G.vertices[i] = G.vertices[i+1];
    --G.vexnum;
    return OK;
    /********** End **********/
}

void DeleteArc_A_Node(VNode &Node,int index)
{
    ArcNode * p =Node.firstarc;
    if(p==NULL)return;
    if(p->adjvex==index)//首个链表节点就是
    {
        Node.firstarc=Node.firstarc->nextarc;
        free(p);
        p=NULL;
        return;
    }
    ArcNode * q =Node.firstarc->nextarc;
    for(;q!=NULL;)
```

```
    {
        if(q->adjvex==index)
        {
            p->nextarc=q->nextarc;
            free(q);
            q=p->nextarc;
            break;
        }
        else
        {
            p=p->nextarc;
            q=q->nextarc;
        }
    }
}

status InsertArc(ALGraph &G,KeyType v,KeyType w)
//在图G中增加弧<v,w>，成功返回OK,否则返回ERROR
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    int s1=Get_index(G,v);
    int s2=Get_index(G,w);
    if (s1==-1 || s2==-1) return ERROR;
    ArcNode *q=G.vertices[s1].firstarc;
    while (q){
        if (q->adjvex==s2) return ERROR;
        q=q->nextarc;
    }

    ArcNode* p=(ArcNode*)malloc(sizeof(ArcNode));
```

```
    p->adjvex = s2;
    p->nextarc=G.vertices[s1].firstarc;
    G.vertices[s1].firstarc=p;

    p=(ArcNode*)malloc(sizeof(ArcNode));
    p->adjvex = s1;
    p->nextarc=G.vertices[s2].firstarc;
    G.vertices[s2].firstarc=p;
    ++G.arcnum;
    return OK;
    /********** End **********/
}

status DeleteArc(ALGraph &G,KeyType v,KeyType w)
//在图G中删除弧<v,w>, 成功返回OK,否则返回ERROR
{
    // 请在这里补充代码, 完成本关任务
    /********** Begin *********/
    int s1=Get_index(G,v);
    int s2=Get_index(G,w);
    if (s1==-1 || s2==-1) return ERROR;
    ArcNode *q=G.vertices[s1].firstarc;
    int flag=0;
    while(q){
        if (q->adjvex==s2) flag=1;
        q=q->nextarc;
    }
    if (flag==0) return ERROR;
    DeleteArc_A_Node(G.vertices[s1],s2);
    DeleteArc_A_Node(G.vertices[s2],s1);
    --G.arcnum;
```

```
    return OK;
    /********** End **********/
}

int vis2[30];

void dfs(ALGraph G,int n,void (*visit)(VertexType)){
    if (G.vertices[n].data.key==0) return;
    vis2[n]=1;
    visit(G.vertices[n].data);
    ArcNode *p=G.vertices[n].firstarc;
    while(p){
        if(vis2[p->adjvex]==0) {
            vis2[p->adjvex]=1;
            dfs(G,p->adjvex,visit);
        }
        p=p->nextarc;
    }
}

status DFSTraverse(ALGraph &G,void (*visit)(VertexType))
//对图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    memset(vis2,0,sizeof(vis2));
    for (int i=0;i<G.vexnum;++i)
    {
        if (vis2[i]==0) dfs(G,i,visit);
    }
    return OK;
```

```
    /********** End **********/
}

int vis1[30];

status BFSTraverse(ALGraph &G,void (*visit)(VertexType))
//对图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    memset(vis1,0,sizeof(vis1));
    queue<int> q;
    for (int i = 0;i<G.vexnum;++i){
    if (vis1[i]==0) q.push(i);
    while(!q.empty()){
        int u=q.front();
        q.pop();
        if (vis1[u]==0) visit(G.vertices[u].data);
        vis1[u]=1;
        ArcNode *p=G.vertices[u].firstarc;
        while(p){
            if (vis1[p->adjvex]==0) {
                q.push(p->adjvex);
            }
            p=p->nextarc;
        }
    }
    }
    return OK;
    /********** End **********/
}
```

```
int Get_v_index(ALGraph G,int key)
{
    for(int i=0;i<G.vexnum;i++)
    {
        if(G.vertices[i].data.key==key)
        {
            return i;
        }
    }
    return -1;
}

status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2])
/*根据V和VR构造图T并返回OK，如果V和VR不正确，返回ERROR
如果有相同的关键字，返回ERROR。此题允许通过增加其它函数辅助实现本关任务*/
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    int p = 0;
    while(V[p].key!=-1){
        G.vertices[p].data = V[p];
        G.vertices[p].firstarc=NULL;
        for (int i = 0; i < p; ++i)
            if (V[p].key == V[i].key) return ERROR;
        ++p;
        if (p>20) return ERROR;
    }
    if (p==0) return ERROR;
    int t = 0;
    while (VR[t][0] != -1){
```

```
int u=-1;
for (int i = 0; i < p; ++i){
    if (V[i].key == VR[t][0]) {
        u=i;
        break;
    }
}
if (u==-1) return ERROR;
int v=-1;
for (int i = 0; i < p; ++i){
    if (V[i].key == VR[t][1]) {
        v=i;
        break;
    }
}
if (v==-1) return ERROR;


    ArcNode* ps=G.vertices[u].firstarc,*ss;
    ss=(ArcNode *)malloc(sizeof(ArcNode));
    ss->adjvex=v;
    ss->nextarc=ps;
    G.vertices[u].firstarc=ss;

    ArcNode* pt=G.vertices[v].firstarc;
    ss=(ArcNode *)malloc(sizeof(ArcNode));
    ss->adjvex=u;
    ss->nextarc=pt;
    G.vertices[v].firstarc=ss;
//printf("%d,%d  ", u, v);
++t;
```

```
    }
    G.vexnum=p;
    G.arcnum=t;
    G.kind=UDG;
    /********** End **********/
}

status SaveGraph(ALGraph G, char FileName[])
//将图的数据写入到文件FileName中
{
    FILE*fp1;
    fp1=fopen(FileName,"w");
    int i,j,tt=-1;
    int nnn[30][2];
    i=0;
    while(i<G.vexnum)
    {
        fprintf(fp1,"%d %s ",G.vertices[i].data.key,G.vertices[i].data.others);
        ++i;
    }
    i=0;
    fprintf(fp1,"%d null ",tt);
    while(i<G.vexnum)
    {
        for(j=0;j<30;++j)
        {
            nnn[j][0]=-1;
            nnn[j][1]=-1;
        }
        j=0;
        if(G.vertices[i].firstarc!=NULL)
```

```
        {
            if(G.vertices[i].data.key<G.vertices[G.vertices[i].firstarc->adjvex].d
            {
                nnn[j][0]=G.vertices[i].data.key;
                nnn[j][1]=G.vertices[G.vertices[i].firstarc->adjvex].data.key;
                ++j;
            }
            ArcNode*p=G.vertices[i].firstarc;
            while(p->nextarc!=NULL)
            {
                p=p->nextarc;
                if(G.vertices[i].data.key<G.vertices[p->adjvex].data.key)
                {
                    nnn[j][0]=G.vertices[i].data.key;
                    nnn[j][1]=G.vertices[p->adjvex].data.key;
                    ++j;
                }
            }
        }
        --j;
        while(j>=0)
        {
           fprintf(fp1,"%d %d ",temp[j][0],temp[j][1]);
            --j;
        }
        ++i;
    }
    fprintf(fp1,"%d %d ",tt,tt);
    fclose(fp1);
    return OK;
```

```
}

status LoadGraph(ALGraph &G, char FileName[])
//读入文件FileName的图数据，创建图的邻接表
{
    if(G.vexnum!=0) return ERROR;
    VertexType V[30];
    KeyType VR[100][2];
    int i;
    FILE*fp2;
    fp2=fopen(FileName,"r");
    i=0;
    do {
        fscanf(fp2,"%d %s ",&V[i].key,V[i].others);
    } while(V[i++].key!=-1);
    i=0;
    do {
        fscanf(fp2,"%d %d",&VR[i][0],&VR[i][1]);
    } while(VR[i++][0]!=-1);

    fclose(fp2);
    CreateGraph(G,V,VR);
    return OK;
}

#endif // OPT_H_INCLUDED


/* In File "main.cpp" */
```

```
#include <iostream>
#include <sstream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <string>
#include "def.h"
#include "opt.h"
//#include <iostream>
#include <stack>
//#include <cstring>
#include <vector>
#include <queue>
#define MAXN 100

using namespace std;

ALGraph G;

ALGraph G1[20];
int available[20];
int totot=0;

int g[40][40];

void MatrixTransformer(ALGraph G){
    for (int i = 0;i<20;++i)
        for (int j=0;j<20;++j)
        g[i][j]=0;
    for (int i=0;i<G.vexnum;++i)
    {
```

```
        ArcNode *p=G.vertices[i].firstarc;
        while(p){
            g[i][p->adjvex]=1;
            p=p->nextarc;
        }
    }
}

int anss[20];

status VerticesSetLessThanK(ALGraph G,KeyType v,int k){
    MatrixTransformer(G);
    int p=Get_v_index(G,v);
    for (int i = 0;i<20;++i) anss[i]=0;
    if (p==-1) return ERROR;
    anss[p]=1;
    for (int i = 1;i < k;++i){
        for (int j = 0;j < 20;++j)
        if (anss[j]==1){
            for (int k=0;k<20;++k)
                if (g[j][k]==1) anss[k] = 1;
        }
    }
    return OK;
}

stack<int> stk;
vector<int> p[MAXN];
int low[MAXN], dfn[MAXN], vis[MAXN], ans[MAXN];
int tot = 0, sum = 0;
```

```
void VectorTransformer(ALGraph G){
    for(int i=0;i<20;++i)
        p[i].clear();
    for (int i=0;i<G.vexnum;++i)
    {
        ArcNode *t=G.vertices[i].firstarc;
        while(t){
            p[i].push_back(t->adjvex);
            t=t->nextarc;
        }
    }
}

void tarjan(int u){
//    printf("#%d# ", u);
//    printf("\n");
//    printf("##dfn ");
//    for (int i = 0; i < n; ++i) printf("%d ", dfn[i]);
//    printf("\n");
//    printf("##low ");
//    for (int i = 0; i < n; ++i) printf("%d ", low[i]);
//    printf("\n");
//    printf("\n");

    dfn[u] = low[u] = ++tot;
    vis[u] = 1;
    stk.push(u);
    //return;
    for (int i = 0; i < p[u].size(); ++i){
        int v = p[u][i];
```

```
            if (!dfn[v]) {
                tarjan(v);
                low[u] = min(low[u], low[v]);
            }
            else if (vis[u]) low[u] = min(low[u], dfn[v]);
        }
        if (dfn[u] == low[u]) {
            int j = -1;
            printf("##%d: ", ++sum);
            while (u != j) {
                j = stk.top();
                printf("%d ", stk.top());
                vis[stk.top()] = 0;
                stk.pop();
            }
            printf("\n");
        }
}

void ConnectedComponentsNums(ALGraph G){
    VectorTransformer(G);
    memset(dfn,0,sizeof(dfn));
    memset(low,0,sizeof(low));
    memset(vis,0,sizeof(vis));
    memset(ans,0,sizeof(ans));
    for (int i = 0; i < G.vexnum; ++i)
        if (!dfn[i]) tarjan(i);
}

int tm[20][20];
```

```
void MatrixMul(){
    for (int i = 0;i<20;++i)
        for (int j=0;j<20;++j)
        tm[i][j]=0;
    for (int i = 0;i<20;++i)
    for (int j = 0;j<20;++j){
        for (int k=0;k<20;++k)
            tm[i][j]+=g[i][k]*g[k][j];
    }
    for (int i = 0;i<20;++i)
        for (int j=0;j<20;++j)
        g[i][j]=tm[i][j];
}

int ShortestPathLength(ALGraph G,KeyType v,KeyType w){
    int s1=Get_v_index(G,v),s2=Get_v_index(G,w);
    if (s1==-1 || s2==-1) return -1;
    MatrixTransformer(G);
    if (g[s1][s2]) return 1;
    for (int i = 1;i<=G.vexnum;++i){
        MatrixMul();
        if (g[s1][s2]) return i+1;
    }
    return -1;
/********/
}

void menuBar(){
    //system("cls");
    //list_read!
    printf("----------------- -- MENU BAR -----------------\n");
```

```
    printf("|                                                  |\n");
    printf("|      FORMAT: <OPERATION> <KEY1> <KEY2>..         |\n");
    printf("|  1. CreateCraph        2. DestroyGraph           |\n");
    printf("|  3. LocateVex          4. PutVex                 |\n");
    printf("|  5. FirstAdjVex        6. NextAdjVex             |\n");
    printf("|  7. InsertVex          8. DeleteVex              |\n");
    printf("|  9. InsertArc          10.DeleteArc              |\n");
    printf("|  11.DFSTraverse        12.BFSTraverse            |\n");
    printf("|  13.VerticesSetLessThanK 14.ShortestPathLength|\n");
    printf("|  15.ConnectedComponentsNums     16.SaveGraph  |\n");
    printf("|  17.LoadGraph          18.AddGraph               |\n");
    printf("|  19.InitNewGraph       20.RemoveNewGraph         |\n");
    printf("|                Press 0 to exit~                  |\n");
    printf("|                                                  |\n");
    printf("-------------------- MENU BAR ------------------\n\n");
}

void initialBar(){
    printf("--------------- WELCOME! ---------------\n");
    printf("|                                       |\n");
    printf("|          欢迎来到图的世界！！         |\n");
    printf("|                                       |\n");
    printf("-----------------------------------------\n\n");
}

void putError(int state, string str = "NUMBER"){
    if (state == INFEASIBLE) printf(">>> ERROR: INFEASIBLE! \n");
    if (state == OVERFLOW) printf(">>> ERROR: OVERFLOW! \n");
    if (state == ERROR) std::cout << ">>> ERROR: INVALID " <<str << "! \n";
    printf(">>>  \n");
}
```

```
void visit(VertexType v)
{
    printf(" %d,%s",v.key,v.others);
}

int getopt(){
    getchar();
    getchar();
    system("cls");
    int st;
    printf("\n请输入操作符: \n");
    scanf("%d", &st);
    return st;
}

int main()
{
    memset(available,0,sizeof(available));
    initialBar();
    menuBar();
    getchar();
    int ps,ans;
    while (ps=getopt()){
        if(ps==1){
                printf("请输入数据: \n");
            VertexType V[30];
            KeyType VR[100][2];
            int i=0;
            do {
                scanf("%d%s",&V[i].key,V[i].others);
```

```
    } while(V[i++].key!=-1);
    i=0;
    do {
        scanf("%d%d",&VR[i][0],&VR[i][1]);
    } while(VR[i++][0]!=-1);
    if (CreateCraph(G,V,VR)==ERROR) printf("输入数据错, 无法创建\n");
    else printf("成功! \n");
}
if(ps==2){
    DestroyGraph(G);
    printf("销毁成功! \n");
}
if(ps==3){
    printf("请输入关键词: \n");
    int uu;
    scanf("%d", &uu);
    int ans=LocateVex(G,uu);
    printf("\n位序为%d\n",ans);
}
if(ps==4){
    printf("请输入关键词和所赋的新值（用空格间隔）: \n");
    int uu;
    VertexType vlu;
    scanf("%d%d%s", &uu,&vlu.key,vlu.others);
    int ans=PutVex(G,uu,vlu);
    if (ans==ERROR) printf("赋值失败! \n");
    else printf("赋值成功! \n");
}
if(ps==5){
    printf("请输入关键词: \n");
    int uu;
```

```
        scanf("%d", &uu);
        int ans=FirstAdjVex(G,uu);
        if (ans==-1) printf("不存在! \n");
        else printf("第一邻接点为%d! \n",ans);
    }
    if(ps==6){
        printf("请输入两个关键词: \n");
        int uu,vv;
        scanf("%d%d", &uu,&vv);
        int ans=NextAdjVex(G,uu,vv);
        if (ans==-1) printf("不存在! \n");
        else printf("下一邻接点为%d! \n",ans);
    }
    if(ps==7){
        printf("请输入关键词: \n");
        VertexType uu;
        scanf("%d%s", &uu.key,uu.others);
        int ans=InsertVex(G,uu);
        if (ans==ERROR) printf("插入失败! \n");
        else printf("插入成功! \n");
    }
    if(ps==8){
        printf("请输入关键词: \n");
        int uu;
        scanf("%d", &uu);
        int ans=DeleteVex(G,uu);
        if (ans==ERROR) printf("删除失败! \n");
        else printf("删除成功! \n");
    }
    if(ps==9){
        printf("请输入关键词: \n");
```

```
        int uu,vv;
        scanf("%d%d", &uu,&vv);
        int ans=InsertArc(G,uu,vv);
        if (ans==ERROR) printf("删除失败! \n");
        else printf("删除成功! \n");
    }
    if(ps==10){
        printf("请输入关键词: \n");
        int uu,vv;
        scanf("%d%d", &uu,&vv);
        int ans=DeleteArc(G,uu,vv);
        if (ans==ERROR) printf("删除失败! \n");
        else printf("删除成功! \n");
    }
    if(ps==11){
        DFSTraverse(G,visit);
    }
    if(ps==12){
        BFSTraverse(G,visit);
    }
    if(ps==13){
        printf("请输入关键词和距离: \n");
        int uu,vv;
        scanf("%d%d", &uu,&vv);
        int ans=VerticesSetLessThanK(G,uu,vv);
        if (ans==ERROR) printf("删除失败! \n");
        else {
                printf("##");
            for (int i=0;i<G.vexnum;++i) if (anss[i]) printf("%d ",i);
        printf("\n");
        }
```

```
}
if(ps==14){
    printf("请输入关键词: \n");
    int uu,vv;
    scanf("%d%d", &uu,&vv);
    int ans=ShortestPathLength(G,uu,vv);
    if (ans==-1) printf("不连通! \n");
    else {
           printf("%d\n",ans);
    }
}
if(ps==15){
    ConnectedComponentsNums(G);
}
if (ps==16){
    printf("请输入文件名（以Enter结束）: \n");
    char str[80];
    scanf("%s", str);
    SaveGraph(G, str);
    printf("保存成功! \n");
}
if (ps==17){
    printf("请输入文件名（以Enter结束）: \n");
    char str[80];
    scanf("%s", str);
    LoadGraph(G, str);
    printf("读取成功! \n");
}
if (ps==18) {
    available[totot++]=1;
}
```

```
        if (ps==19) {
            int u;
            scanf("%d",&u);
            if (available[u]==0) continue;
        printf("请输入数据: \n");
            VertexType V[30];
            KeyType VR[100][2];
            int i=0;
            do {
                scanf("%d%s",&V[i].key,V[i].others);
            } while(V[i++].key!=-1);
            i=0;
            do {
                scanf("%d%d",&VR[i][0],&VR[i][1]);
            } while(VR[i++][0]!=-1);
            if (CreateCraph(G1[u],V,VR)==ERROR) printf("输入数据错, 无法创建\n");
            else printf("成功! \n");
        }
        if (ps==20) {
            int u;
            scanf("%d",&u);
            available[u]=0;
        }
    }
    printf("\n\n欢迎下次光临哟~\n");
    return 0;
}

/* End of Code */
```