華中科技大学

课 程 实 验 报 告

课程名称： 数据结构实验

专业班级： 计算机本硕博 2301 班

学　　号： U202315763

姓　　名： 王家乐

指导教师： 向文

报告日期： 2024 年 5 月 22 日

计算机科学与技术学院

目 录

1 基于链式存储结构的线性表实现

1.1 问题描述

该实验要解决的基本问题是实现单链表的各个基本功能，如初始化表、销毁表、清空表、判定空表、求表长和获得元素等等。

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。可选择以文件的形式进行存储和加载，也即将生成的单链表存入到相应的文件中，也可以从文件中获单链表进行操作。

1.1.1 需实现的基本运算

依据最小完备性和常用性相结合的原则，以函数形式定义了单链表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等 20 种基本运算，具体运算功能定义如下。

⑴初始化表：函数名称是 InitList(L)；初始条件是单链表 L 不存在；操作结果是构造一个空的单链表。

⑵销毁表：函数名称是 DestroyList(L)；初始条件是单链表 L 已存在；操作结果是销毁单链表 L。

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

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

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

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

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

⑻获得前驱：函数名称是 PriorElem(L,cur_e,pre_e)；初始条件是单链表 L

已存在；操作结果是若 cur_e 是 L 的数据元素，且不是第一个，则用 pre_e 返回它的前驱，否则操作失败，pre_e 无定义。

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

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

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

⑿遍历表：函数名称是 ListTraverse(L,visit())，初始条件是单链表 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()。

⒀保存单链表:函数名称是 SaveList(Lists),初始条件是单链表集合 Lists 已存在；操作结果是将单链表元素按顺序保存至文件名为 FileName 的文件中。

⒁加载单链表：函数名称是 LoadList(Lists)，将名为 FileName 的文件中的所有单链表加载到 Lists 集合中。

⒂在单链表集合中插入一个单链表：函数名称是 AddList(Lists,Name)，无初始条件；操作结果是将名为 Name 的单链表添加到单链表集合中。

⒃在单链表集合中删除一个单链表:函数名称是 RemoveList(Lists, Name)，初始条件是单链表集合非空；操作结果是将名为 Name 的单链表删除。

⒄在单链表集合中查找一个单链表:函数名称是 LocateList(Lists, Name)，初始条件是单链表集合非空；操作结果是查找名为 Name 的单链表的位置。

⒅链表翻转：函数名称是 ReverseList(L)，初始条件是线性表 L 已存在；操作结果是将 L 翻转。

⒆删除链表的倒数第 n 个结点：函数名称是 RemoveNthFromEnd(L,n)， 初始条件是线性表 L 已存在且非空，操作结果是该链表中倒数第 n 个节点;。

⒇链表排序：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。

1.2 系统设计

本次实验的系统设计如下：将菜单演示和用户选择写入到 while 循环中，用 OP 获取用户的选择，OP 初始化为 1,以便第一次能进入循环。进入循环后系统首先显示功能菜单，然后用户输入选择 0-7,其中 1-7 分别代表对单链表集合一个基本运算，在主函数中通过 switch 语句对应到相应的函数功能，执行完该功能后 break 跳出 switch 语句，继续执行 while 循环，直至用户输入 0 退出当前演示系统。若输入 4，则根据用户输入的单链表名称跳转至相应的单个单链表管理界面，采用与主界面相同的设计思路。

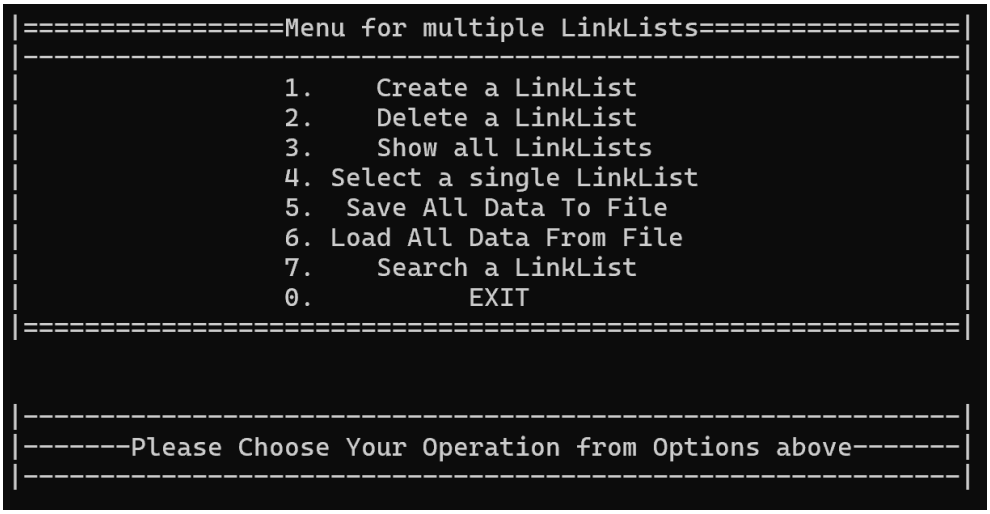


图 1-1 演示系统主菜单

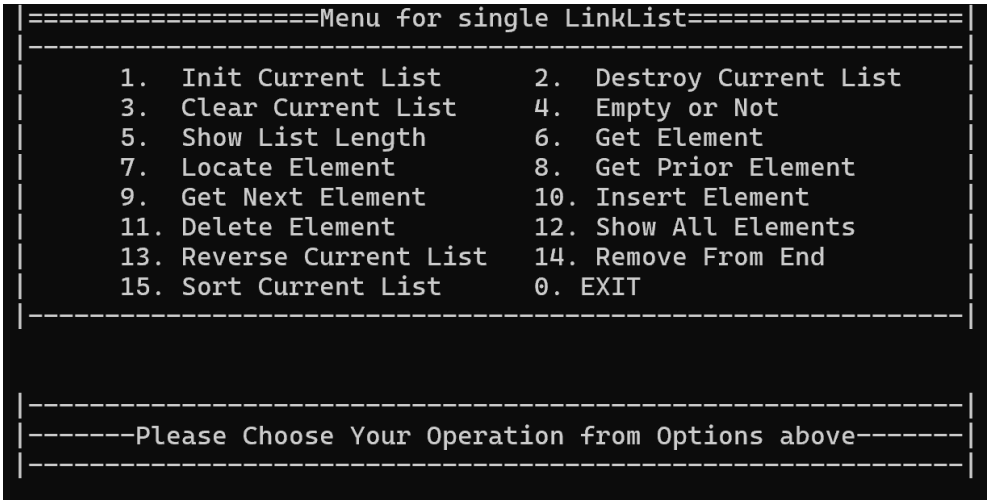


图 1-2 演示系统子菜单

本次实验使用的数据结构定义为：

```
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define LIST_INIT_SIZE 10
#define LISTINCREMENT    10
typedef int status;
typedef int ElemType; //数据元素类型定义
typedef struct LNode{    //单链表（链式结构）结点的定义
        ElemType data;
        struct LNode *next;
}LNode,*LinkList;
typedef struct{    //单链表的管理表定义
      struct { char name[30];
               LinkList L;
        }elem[10];
        int length;
}LISTS;
```

1.3 系统实现

单链表运算算法思想与设计如下：

⑴初始化单链表：将单链表初始化过程写成函数，其中传入函数的参数是主函数中定义的结构型变量 L 的引用。在函数中，首先判断 L 是否创建，如果未创建则使用 malloc 函数分配一个结点的内存空间，并把 L 的 next 指针域置为空。经分析，算法的时间复杂度为 0(1)。

⑵销毁单链表：将销毁单链表的过程写成函数，其中传入函数的参数是主函数中定义的结构性变量 L 的引用。在函数中，使用循环依次将每个结点的内存空间释放，最后将头指针置为空。经分析，该算法的时间复杂度为 O(n)。

⑶清空单链表：将清空单链表的过程写成函数，其中将主函数中定义的结构性变量 L 的引用作为函数参数。在函数中，将头节点的后继指针置空，使用循环从首元节点开始依次释放内存。经分析，该算法的时间复杂度为 O(n)。

⑷判定单链表是否为空表：将判空过程写成函数，其中将主函数中定义的结构性变量 L 作为函数参数。直接判断 L 是否为 NULL，若是则返回 TRUE,否则返回 FALSE。经分析，该算法的时间复杂度为 O(1)。

⑸求单链表表长：将求表长过程写成函数，其中主函数中定义的结 构性变量 L 的引用作为函数的参数，在函数中，通过循环和计数变量记录结点数，最后返回计数结果即为表长。经分析，该算法的时间复杂度为 O(n)。

⑹获得元素：将获得单链表元素写成函数，其中函数的参数是结构型变量 L 以及数据元素的序号 i,由于采取的是链式存储结构，故通过循环依次访问结点并比较数据域的方式来获取元素，当然，如果位置不合法应返回 ERROR。经分析，该算法的时间复杂度为 O(n)。

⑺查找元素：将查找单链表特定值的数据元素写成函数，其中函数的参数是主函数中定义的结构类型变量 L 以及查找的数据元素的值，通过循环对单链表中的每一个结点的数据域与给定值比较是否相等，如果相等就返回该元素的位置。经分析，该算法的时间复杂度为 O(n)。

⑻获得直接前驱：将获得直接前驱元素的过程写成函数，函数的参数是结构体类型变量以及特定数据元素的值，接受前驱的变量。在函数中，首先判断单链表是否为空，若不为空则通过循环遍历元素比较，如果头结点元素即为所找元素则表明无直接前驱，如果头结点不是则用另一个指针指向当前遍历结点的前一个结点，当找到元素时将前一个结点的值赋给 pre，若找不到则返回 ERROR。经分析，该算法的时间复杂度为 O(n)。

⑼获得直接后继：将获得直接后继元素的过程写成函数，函数的参数是结构体类型变量以及特定数据元素的值，接受后继的变量。在函数中，首先判断单链表是否为空，若不为空则通过循环遍历元素比较，当找到元素时将后一个

结点的值赋给 next，若该元素对应最后一个结点则表明无直接后继，若找不到元素则返回 ERROR。经分析，该算法的时间复杂度为 O(n)。

⑽插入元素：将插入元素写成函数，函数的参数是结构型变量的引用，插入元素的值以及插入位置。在函数中，首先判断插入位置的合法性， 即是否在单链表中合适的位置，插入元素时如果单链表结点数为 0，则直接令 L 指向新创建的待插入结点，若不是则先定位至插入位置处更改结点的指针域来实现插入。经分析，该算法的时间复杂度为 O(n)。

⑾删除元素：将删除单链表中元素写成函数，函数的参数是结构类型变量的引用，待删除元素的位置，接受待删除元素的变量。在函数中，先定位至待删除元素的位置，将其赋值给接受变量，然后通过改变指针域移除该结点，最后释放该结点的内存空间。经分析，该算法的时间复杂度为 O(n)。

⑿遍历单链表：将遍历单链表写成一个函数，函数的参数是结构类型变量，直接用一个循环来对单链表中的每一个元素进行操作（此处进行输出操作）。经分析，该算法的时间复杂度为 O(n)。

⒀保存单链表到文件中：函数的参数是单链表集合 Lists 和待保存文件的路径或名称。在函数中，创建一个指向待保存文件的文件指针，通过循环依次将每个单链表的名称、长度及节点信息写入文件中。经分析，该算法的算法复杂度是 $O(n^2)$。

⒁加载文件到单链表集合中：函数的参数是单链表集合 Lists 的引用和待加载文件的路径。在函数中，创建一个指向待加载文件的文件指针，并根据读写逐个分配单链表 L 中结点的内存空间，最后将尾结点的指针域置为空并释放文件指针。经分析，该算法的算法复杂度是 $O(n^2)$。

⒂插入一个单链表：函数的参数是单链表集合 Lists 的引用和待插入单链表的名称。在函数中，先判断该名称的单链表是否已经存在，若不存在，则在单链表集合 Lists 的末尾处开辟新的单链表的存储空间，并将管理表长度增加 1。经分析，该算法的时间复杂度是 O(1)。

⒃删除一个单链表：函数的参数是管理表的引用和待删除单链表的名称。在函数中，通过循环比较单链表名称和待删除单链表的名称，如果没有找到则返回 ERROR，否则就先摧毁待删除单链表，然后自该位置起从前往后将下一个

单链表复制到前一个单链表中。经分析，该算法复杂度是 O（n^2）。

⑰定位一个单链表：函数参数是单链表集合 Lists 和待查找的单链表名称。在函数中，通过循环比较待查找单链表名称和管理表中单链表的名称，如果匹配成功则返回该单链表位序，否则返回 ERROR。经分析，该算法复杂度是 O(n)。

⑱链表翻转：函数参数是结构型变量 L 的引用。在函数中，用一个指针 p 指向首元节点，再将头节点指针域置空，用指针 p 遍历链表，采用头插法插入，则得到逆置链表。经分析，该算法复杂度是 O(n)。

⑲删除链表的倒数第 n 个节点：函数参数是结构型变量 L 的引用。先求出表长 len，再调用 ListDeleteh()函数删除第 len - n + 1 个节点。经分析，该算法复杂度是 O(n)。

⑳链表排序：函数参数是结构型变量 L 的引用。采用交换数据域的冒泡排序算法，共进行 len-1 次，每次把最大数移到末尾。经分析，该算法复杂度是 $O(n^2)$。

1.4 系统测试

输入样例解释：以下测试按照从上到下的顺序进行，且均先按照菜单输入相应的功能选项。

函数	输入	预期输出	实际输出
AddList 添加单链表	name1	The linear table (name: name1) is created!	Please enter the name of the linear table you want to name1 The linear table (name: name1) is created!
LocateList 定位单链表	name1	The location of the list is 1.	Please enter the name of the list you want to query: name1 The location of the list is 1.
InitList 初始化单链表		Successfully initialized!	Successfully initialized!
ListEmpty 判空		The List is empty.	The List is empty.
ListInsert 插入元素	1 65	Successfully inserted.	Position: (between 1 to 1) Please enter the position and the element you want to insert:(s paced by space) 1 65 Successfully inserted.

ListInsert 插入元素	2 4	Successfully inserted.	Position: (between 1 to 2) Please enter the position and the element you want to insert:(spaced by space) 2 4 Successfully inserted.
ListInsert 插入元素	2 13	Successfully inserted.	Position: (between 1 to 3) Please enter the position and the element you want to insert:(spaced by space) 2 13 Successfully inserted.
ListInsert 插入元素	4 22	Successfully inserted.	Position: (between 1 to 4) Please enter the position and the element you want to insert:(spaced by space) 4 22 Successfully inserted.
ListInsert 插入元素	3 16	Successfully inserted.	Position: (between 1 to 5) Please enter the position and the element you want to insert:(spaced by space) 3 16 Successfully inserted.
ListInsert 插入元素	7 4	The position is illegal.	Position: (between 1 to 6) Please enter the position and the element you want to insert:(spaced by space) 7 4 The position is illegal.
ListTraverse 遍历单链表		65 13 16 4 22	65 13 16 4 22 Successfully traveled all elements.
ListEmpty 判空		The List is not empty.	The List is not empty.
ListLength 求表长		The length of the List is : 5.	The length of the List is : 5.
GetElem 某位置元素	2	The element is 13.	Please enter the position (between 1 to 5) you want to query: 2 The element is 13.
GetElem 某位置元素	8	The position is illegal.	Please enter the position (between 1 to 5) you want to query: 8 The position is illegal.
LocateElem 查找某元素	13	The position of 13 is 2.	Please enter the element you want to locate. 13 The position of 13 is 2.
LocateElem 查找某元素	5	The element does not exist.	Please enter the element you want to locate. 5 The element does not exist.
PriorElem 获取前驱	16	The prior element of 16 is 13.	Please enter the element you want to query: 16 The prior element of 16 is 13.
PriorElem 获取前驱	65	failed to find.	Please enter the element you want to query: 65 failed to find.
NextElem 获取后继	4	The next element of 4 is 22.	Please enter the element you want to query: 4 The next element of 4 is 22.
NextElem 获取后继	22	failed to find.	Please enter the element you want to query: 22 failed to find.

ListDelete 删除元素	2	Delete 13 in position 2.	Position: (between 1 to 5) Please enter the position you want to delete: 2 Delete 13 in position 2.
ListDelete 删除元素	8	The position is illegal.	Position: (between 1 to 4) Please enter the position you want to delete: 8 The position is illegal.
ListInsert 插入元素	4 99	Successfully inserted.	Position: (between 1 to 5) Please enter the position and the element you want to insert:(spaced by space) 4 99 Successfully inserted.
ListTraverse 遍历单链表		65 16 4 99 22	65 16 4 99 22 Successfully traveled all elements.
SaveList 保存到文件		Successfully Saved.	Successfully Saved. 
ReverseList 翻转单链表		Successfully reversed.	Successfully reversed.
ListTraverse 遍历单链表		22 99 4 16 65	22 99 4 16 65 Successfully traveled all elements.
SortList 从小到大排序		Successfully sorted.	Successfully sorted.
ListTraverse 遍历单链表		4 16 22 65 99	4 16 22 65 99 Successfully traveled all elements.
RemoveNthFromEnd 删除倒数第 n 个节点	2	Successfully deleted.	Which element do you want to remove from end? 2 Successfully deleted.
ListTraverse 遍历单链表		4 16 22 99	4 16 22 99 Successfully traveled all elements.
ClearList 清空单链表		Successfully cleared!	Successfully cleared!
ListTraverse 遍历单链表		There is no element.	There is no element.
DestroyList 销毁单链表		Successfully destroyed!	Successfully destroyed! Don't forget to initialize current List.
ListTraverse 遍历单链表		You haven't initialized.	You haven't initialized.
AddList 添加单链表	name2	The linear table (name: name2) is created!	Please enter the name of the linear table you want to add : name2 The linear table (name: name2) is created!

DeleteList 删除单链表	name2	The linear table (name: name2) is deleted!	Please enter the name of the linear table you want to delete : name2 The linear table (name: name2) is deleted!
LoadList 从文件中加载		Successfully Loaded.	Are you sure you want to load from the file? The data that is not currently saved will be gone. confirm:1 cancel:0 1 Reading a list with the name name1. element 65 is being read. element 16 is being read. element 4 is being read. element 99 is being read. element 22 is being read. Successfully Loaded.
退出系统	0	Welcome to use this system next time!	Welcome to use this system next time! 请按任意键继续. . .

1.5 实验小结

在实验中，我遇到了如下几个错误：

（1）在写各个函数的代码中，总是因为混淆 L 和 L->next 而导致程序出现千奇百怪的输出结果，后来重新看了一遍关于单链表的结点的有关知识才避免了各种错误，这告诉我，在处理一种数据结构之前先要了解其逻辑结构。

（2）在编写程序时，我有几次都忘记了要释放链表中不需要的结点或者文件指针。这与上次实验中我犯的错误有相似之处，我会吸取教训，不再给代码埋下这种隐患。

（3）在编写排序函数时没有正确考虑循环次数以及每次 p 指针的起点，导致程序出错。这告诉我在动手写代码之前一定要理清思路，抓住每个细节避免因为考虑不周而出现错误。

2 基于邻接表的图实现

2.1 问题描述

该实验要解决的基本问题是实现邻接表的各个基本功能。

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。可选择以文件的形式进行存储和加载，也即将生成的邻接表保存到相应的文件中，也可以从文件中获取邻接表进行操作。同时，也可以对图的集合进行添加、删除和查找操作。

2.1.1 需实现的基本运算

依据最小完备性和常用性相结合的原则，以函数形式定义了邻接表的创建图、销毁图、查找顶点、获得顶点的位序和顶点赋值等 19 种基本运算，具体运算功能定义如下。

⑴创建邻接表：函数名称是 CreateCraph (G,V,VR)；初始条件是邻接表 G 未创建；操作结果是构造一个邻接表。

⑵销毁邻接表：函数名称是 DestroyGraph(G)；初始条件是邻接表 G 已创建；操作结果是销毁邻接表 G。

⑶查找顶点：函数名称是 LocateVex(G,e)；初始条件是邻接表 G 已创建；操作结果是返回关键字为 e 的顶点的位序。

⑷赋值顶点：函数名称是 PutVex(G,e, value)；初始条件是邻接表 G 已创建；操作结果是将关键字为 e 的顶点的信息赋值为 value。

⑸获得第一邻接点：函数名称是 FirstAdjVex(G, e)；初始条件是邻接表 G 已创建；操作结果是返回关键字为 e 的顶点的第一邻接点的位序。

⑹获得下一邻接点：函数名称是 NextAdjVex(G, v,w)；初始条件是邻接表 G 已创建；操作结果是返回关键字为 v 的顶点的相对于邻接点 w 的下一邻接点的位序。

⑺插入顶点：函数名称是 InsertVex(G, value)；初始条件是邻接表 G 已创建；操作结果是在邻接表的顶点数组中插入信息为 value 的顶点。

⑻删除顶点：函数名称是 DeleteVex(G,e)；初始条件是邻接表 G 已创建；

操作结果是删除关键字为 e 的顶点。

⑼插入边：函数名称是 InsertArc(G,v,w)；初始条件是邻接表 G 已创建；操作结果是插入与关键字为 v 和 w 的顶点关联的边。

⑽删除边：函数名称是 DeleteArc(G,v,w)；初始条件是邻接表 G 已创建；操作结果是删除与关键字为 v 和 w 的顶点关联的边。

⑾深度优先搜索遍历图：函数名称是 DFSTraverse(G,visit())，初始条件是邻接表 G 已创建；操作结果是通过 DFS 算法依次对 G 的每个表头顶点调用函数 visit()。

⑿广度优先搜索遍历图：函数名称是 BFSTraverse(G,visit())，初始条件是邻接表 G 已创建；操作结果是通过 BFS 算法依次对 G 的每个表头顶点调用函数 visit()。

⒀保存图：函数名称是 SaveGraph(GL)，初始条件是图集合 GL 已存在；操作结果是依次将图的顶点和边按顺序保存至文件名为 FileName 的文件中。

⒁加载图：函数名称是 LoadGraph(GL)，将名为 FileName 的文件中的所有图加载到 GL 集合中。

⒂在图的集合中添加一个图：函数名称是 AddGraph(GL, Name)；初始条件是图的集合非空；操作结果是将名为 Name 的图添加到图集合中。

⒃在图的集合中删除一个图：函数名称是 RemoveGraph(GL, Name)；初始条件是图的集合非空；操作结果是将名为 Name 的图从图的集合中删除。

⒄在图的集合中查找一个图：函数名称是 LocateGraph(GL，Name)；初始条件是图的集合非空；操作结果是查找名为 Name 的图并定位以可对其进行操作。

⒅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。

⒆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。

⒇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。

2.2 系统设计

本次实验的系统设计如下：将菜单演示和用户选择写入到 while 循环中，用 OP 获取用户的选择，OP 初始化为 1,以便第一次能进入循环。进入循环后系统首先显示功能菜单，然后用户输入选择 0-7,其中 1-7 分别代表对单链表集合一个基本运算，在主函数中通过 switch 语句对应到相应的函数功能，执行完该功能后 break 跳出 switch 语句，继续执行 while 循环，直至用户输入 0 退出当前演示系统。若输入 3，则根据用户输入的单链表名称跳转至相应的单个单链表管理界面，采用与主界面相同的设计思路。

```
|===============Menu for multiple Graphs================|
|-------------------------------------------------------|
|              1.    Create a Graph                      |
|              2.    Delete a Graph                     |
|              3. Select a single Graph                 |
|              4. Save All Data To File                 |
|              5. Load All Data From File               |
|              6.    Show All Graph                     |
|              7.    Search a Graph                     |
|              0.         EXIT                          |
|=======================================================|


|-------------------------------------------------------|
|----Please Choose Your Operation from Options above----|
|-------------------------------------------------------|
```

图 4-1 演示系统主菜单

```
|================Menu for single Graph==================|
|-------------------------------------------------------|
|  1.  DestroyGraph        2.  LocateVex                |
|  3.  PutVex              4.  FirstAdjVex              |
|  5.  NextAdjVex          6.  InsertVex                |
|  7.  DeleteVex           8.  InsertArc                |
|  9.  DeleteArc           10. DFSTraverse              |
|  11. BFSTraverse         12. VerticesSetLessThanK     |
|  13. ShortestPathLength  14. ConnectedComponentsNums  |
|                    0. EXIT                            |
|=======================================================|


|-------------------------------------------------------|
|----Please Choose Your Operation from Options above----|
|-------------------------------------------------------|
```

图 4-2 演示系统子菜单

本次实验用到的数据结构定义为：

```
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define FileName "data.txt"
#define MAX_VERTEX_NUM 20
#define MAX_NAME_LENGTH 20
#define MAX_GRAPH_NUM 10
typedef int status;
typedef int KeyType;
// 图的类型定义
typedef enum
{
    DG,
    DN,
    UDG,
    UDN
} GraphKind;
// 顶点类型定义
typedef struct
{
    KeyType key;
    char others[20];
} VertexType;
// 表结点类型定义
typedef struct ArcNode
{
```

```
    int adjvex;                 // 顶点位置编号
    struct ArcNode *nextarc; // 下一个表结点指针
} ArcNode;
// 头结点及其数组类型定义
typedef struct VNode
{
    VertexType data;      // 顶点信息
    ArcNode *firstarc;    // 指向第一条弧
} VNode, AdjList[MAX_VERTEX_NUM];
// 邻接表的类型定义
typedef struct
{
    AdjList vertices;          // 头结点数组
    int vexnum, arcnum;    // 顶点数、弧数
    GraphKind kind;         // 图的类型
} ALGraph;
// 图的集合类型定义
typedef struct
{
    struct
    {
        ALGraph G;
        char name[MAX_NAME_LENGTH];
    } elem[MAX_GRAPH_NUM];
    int length;     // 图的数量
} GraphList;
GraphList GL;   // 全局变量，图的集合
```

2.3 系统实现

本次实验函数的算法思想及设计如下：

⑴创建邻接表：函数的参数是邻接表 G 的引用，顶点数组 V，存储边的信息的二维数组 VR。在函数中先判断顶点关键字是否唯一，然后将顶点读入头结点数组中，并采用头插法依次插入边，同时要修改边数和顶点数。经分析，该算法的时间复杂度为 O(n^2)。

⑵销毁邻接表：函数的参数是图 G 的引用。在函数中，先判断图是否创建，如未创建则通过循环依次邻接链表中的结点，并将 firstarc 结点置为空。经分析，该算法的时间复杂度为 O(n^2)。

⑶查找顶点：函数的参数是图 G，待查找关键字 u。在函数中，通过循环比较关键字查找待查找结点,如未找到返回-1。经分析,该算法的时间复杂度为O(n)。

⑷赋值顶点：函数的参数是图 G 的引用，被赋值的结点的关键字 u，待赋值的结点的信息 value。在函数中，先查找关键字为 u 的结点是否在图中，如不在返回 ERROR；然后查找 value.key 是否在图中并且是否与 u 相等，如果在图中且不与 u 相等返回 ERROR；其余情况下，直接赋值。经分析，该算法的时间复杂度为 O(n)。

⑸获得第一邻接点：函数参数是图 G，待查找第一邻接结点的结点的关键字 u。在函数中,先判断关键字为 u 的结点是否在图中,如果不在返回 INFEASIBLE，否则判断 firstarc 是否为空,不为空则返回其存储的第一邻接点信息。经分析，该算法的时间复杂度为 O(n)。

⑹获得下一邻接点：函数参数是图 G，待查找下一邻接结点的结点的关键字 v，相对的邻接点的关键字 w。在函数中，先判断关键字为 v 和 w 的结点是否在图中，如果不在返回 INFEASIBLE，否则通过循环查找关键字为 w 的邻接链表结点，如果其下一邻接点不为空则返回下一邻接点关键字，否则返回-1。经分析，该算法的时间复杂度为 O(n)。

⑺插入顶点：函数参数是图 G 的引用，被插入顶点的信息 value。在函数中，先判断关键字为 value.key 的结点是否在图中，如果在返回 ERROR，否则在表头顶点数组中插入顶点并修改 vexnum 信息。经分析,该算法的时间复杂度为 O(n)。

⑻删除顶点：函数参数是图 G 的引用，待删除顶点的关键字 v。在函数中，

先判断关键字为 v 的结点是否在图中，如果不在返回 ERROR，否则清空找到的表头结点的邻接链表，覆盖被删除的表头结点并修改 vexnum 信息，删除与被删除的表头结点相关的边并更新结点的位序。经分析，该算法的时间复杂度为 $O(n^2)$。

⑼插入边：函数参数是图 G，待插入边的两端的关键字 v 和 w。在函数中，先判断关键字为 v 或 w 的结点是否在图中，如果不在返回 ERROR，否则先判断该边是否已经在图中，如果不在则插入边。经分析，该算法的时间复杂度为 O(n)。

⑽删除边：函数参数是图 G，待删除边的两端的关键字 v 和 w。在函数中，先判断关键字为 v 或 w 的结点是否在图中，如果不在返回 ERROR，否则删除相关联顶点的邻接链表中与此边相关的结点（如果此边不在图中则不会删除而返回 INFEASIBLE）。经分析，该算法的时间复杂度为 O(n)。

⑾深度优先搜索遍历图：函数参数是图 G，访问函数的指针 visit。在函数中，先将 visited 数组初始化为 0，代表未访问；然后从某个顶点 v 出发，首先访问该顶点，然后利用 DFS 核心函数依次从它的各个未被访问的邻接点出发深度优先搜索遍历图，直至图中所有和 v 有路径相通的顶点都被访问到。若此时尚有其他顶点未被访问到，则另选一个未被访问的顶点作起始点，重复上述过程，直至图中所有顶点都被访问到为止。经分析，该算法的时间复杂度为 O(n+e)。

⑿广度优先搜索遍历图：函数参数是图 G，访问函数的指针 visit。在函数中，利用循环队列的数组实现方法进行遍历；首先将 visited 数组初始化为 0，代表未访问；然后将第一个访问的结点 v 入队；然后通过判断队列是否为空控制循环，在循环中，先将队首出队，并将与该顶点相连的顶点入队，最后通过一次次的循环使得图中所有和 v 有路径相通的顶点都被访问到。若此时尚有其他顶点未被访问到，则另选一个未被访问的顶点作起始点，重复上述过程，直至图中所有顶点都被访问到为止。经分析，该算法的时间复杂度为 O(n+e)。

⒀保存图：函数参数是图的集合 GL，待保存文件的名称。在函数中，将图的名称，顶点数，边数依次写入文件，再将表头顶点数组写入，然后，通过循环一次将边写入到文件中。经分析，该算法的时间复杂度为 $O(n^2)$。

⒁加载图：函数参数是图的集合 GL 的引用，待加载文件的名称。在函数中，先将数据读到存放顶点和边的数组 V 和 VR 中，然后调用 CreateGraph 函数创建图。经分析，该算法的时间复杂度为 $O(n^2)$。

⑮在图的集合中添加一个图：函数的参数是图的集合 GL 的引用，待添加图的名称 Name。在函数中，首先在管理表的末端添加一个名称为 Name 的图，然后将数据读到存放顶点和边的数组 V 和 VR 中，调用 CreateGraph 函数创建图。经分析，该算法的时间复杂度为 $O(n^2)$。

⑯在图的集合中删除一个图：函数的参数是图的集合 GL 的引用，待删除图的名称 Name。在函数中，通过循环比较名称查找待删除的图，查找成功则先销毁图，并从前至后移动集合中的图。经分析，该算法的时间复杂度为 $O(n^2)$。

⑰在图的集合中查找一个图：函数的参数是图的集合 GL，待查找图的名称 Name。在函数中，通过循环比较名称查找待查找的图，查找成功则定位至该图并返回其位序。经分析，该算法的时间复杂度为 O(n)。

⑱距离小于 k 的顶点集合：使用广度优先搜索（BFS）在图 G 中找到距离顶点 v 小于 k 的所有顶点。它通过层次遍历，将起始顶点 v 入队，从第 1 层开始，依次访问每一层的所有邻接顶点，直到遍历 k-1 层，并将每一层中未被访问过的顶点记录在结果数组中。最终，结果数组中存储了所有符合条件的顶点，并在末尾添加 -1 作为结束标志。

⑲顶点间最短路径和长度： 使用广度优先搜索（BFS）计算图 G 中顶点 v 到顶点 w 的最短路径长度，通过层次遍历从起始顶点 v 出发，依次访问各顶点的邻接顶点，更新并记录最短路径长度，直到找到目标顶点 w 或遍历完所有可达顶点，最终返回顶点 v 到 w 的最短路径长度。

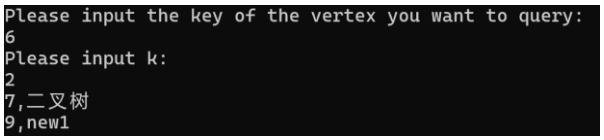
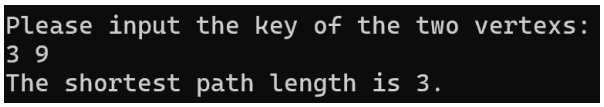
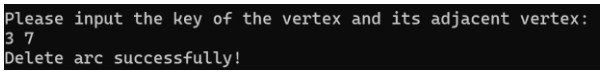
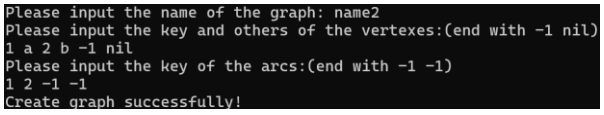
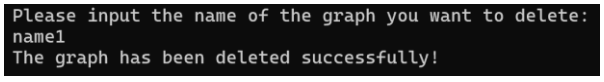
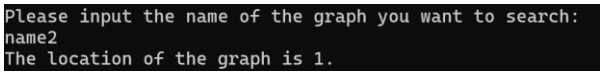
⑳图的连通分量：使用广度优先搜索（BFS）计算图 G 的连通分量数。通过遍历每个顶点，若顶点未被访问过，则启动一次 BFS，从该顶点开始标记所有与其连通的顶点，并将连通分量计数器加一。最终返回图中连通分量的总数。

2.4 系统测试

输入样例解释：以下测试按照从上到下的顺序进行，且均先按照菜单输入相应的功能选项。

函数	输入	预期输出	实际输出
CreateCraph 创建图	name1 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	Create graph successfully!	Please input the name of the graph: name1 Please input the key and others of the vertexes:(end with -1 nil) 5 线性表 8 集合 7 二叉树 6 无向图 -1 nil Please input the key of the arcs:(end with -1 -1) 5 6 5 7 6 7 7 8 -1 -1 Create graph successfully!
LocateGraph 查找图	name1	The location of the graph is 1.	Please input the name of the graph you want to search: name1 The location of the graph is 1.
LocateVex 查找顶点	5	5 线性表	Please input the key of the vertex: 5 5 线性表
LocateVex 查找顶点	9	The vertex does not exist.	Please input the key of the vertex: 9 The vertex does not exist.
DFSTraverse 深度优先搜索遍历		5 线性表 7 二叉树 8 集合 6 无向图	5 线性表 7 二叉树 8 集合 6 无向图
PutVex 给顶点赋值	5 9 new1	Modify vertex successfully!	Please input the key of the vertex you want to modify: 5 Please input the new key and others of the vertex: 9 new1 Modify vertex successfully!
DFSTraverse 深度优先搜索遍历		9 new1 7 二叉树 8 集合 6 无向图	9 new1 7 二叉树 8 集合 6 无向图
FirstAdjVex 第一邻接点	9	The key of the next adjvex is 7.	Please input the key of the vertex: 9 The location of the next adjvex is 2. The key of the next adjvex is 7.
NextAdjVex 下一邻接点	7 8	The key of the next adjvex is 6.	Please input the key of the vertex and its adjacent vertex: 7 8 The location of the next adjvex is 3. The key of the next adjvex is 6.

NextAdjVex 下一邻接点	7 9	The vertex does not exist or has no adjacent vertex.	Please input the key of the vertex and its adjacent vertex: 7 9 The vertex does not exist or has no adjacent vertex.
InsertVex 插入顶点	2 aaa	Insert vertex successfully!	Please input the key and others of the vertex: 2 aaa Insert vertex successfully!
InsertVex 插入顶点	8 bbb	Insert vertex failed!	Please input the key and others of the vertex: 8 bbb Insert vertex failed!
BFSTraverse 广度优先搜索遍历		9 new1 7 二叉树 6 无向图 8 集合 2 aaa	9 new1 7 二叉树 6 无向图 8 集合 2 aaa
InsertVex 插入顶点	3 ccc	Insert vertex successfully!	Please input the key and others of the vertex: 3 ccc Insert vertex successfully!
BFSTraverse 广度优先搜索遍历		9 new1 7 二叉树 6 无向图 8 集合 2 aaa 3 ccc	9 new1 7 二叉树 6 无向图 8 集合 2 aaa 3 ccc
DeleteVex 删除顶点	2	Delete vertex successfully!	Please input the key of the vertex you want to delete: 2 Delete vertex successfully!
BFSTraverse 广度优先搜索遍历		9 new1 7 二叉树 6 无向图 8 集合 3 ccc	9 new1 7 二叉树 6 无向图 8 集合 3 ccc
InsertArc 插入边	3 7	Insert arc successfully!	Please input the key of the vertex and its adjacent vertex: 3 7 Insert arc successfully!
BFSTraverse 广度优先搜索遍历		9 new1 7 二叉树 6 无向图 3 ccc 8 集合	9 new1 7 二叉树 6 无向图 3 ccc 8 集合
DeleteArc 删除边	7 9	Delete arc successfully!	Please input the key of the vertex and its adjacent vertex: 7 9 Delete arc successfully!
BFSTraverse 广度优先搜索遍历		9 new1 6 无向图 7 二叉树 3 ccc 8 集合	9 new1 6 无向图 7 二叉树 3 ccc 8 集合

DFSTraverse 深度优先搜索遍历	19	9 new1 6 无向图 7 二叉树 3 ccc 8 集合	
VerticesSetLessThanK 距离小于 k 的顶点集合	6 2	7,二叉树 9,new1	
ShortestPathLength 顶点间最短路径	3 9	The shortest path length is 3.	
ConnectedComponentsNums 连通分量个数		The number of connected components is 1.	
DeleteArc 删除边	3 7	Delete arc successfully!	
ConnectedComponentsNums 连通分量个数	9	The number of connected components is 2.	
SaveGraph 保存图		Save successfully!	
DestroyGraph 销毁图		Destroy graph successfully!	
AddGraph 添加图	name2 1 a 2 b -1 nil 1 2 -1 -1	Create graph successfully!	
DeleteGraph 删除图	name1	The graph has been deleted successfully!	
LocateGraph 查找图	name2	The location of the graph is 1.	
LoadGraph 加载图		Load successfully!	

退出系统	0		Welcome to use this system next time! 请按任意键继续. . .

2.5 实验小结

在实验中，我遇到了如下几个错误：

（1）本次实验中有许多函数的边界情况很多，最突出的比如插入删除弧和顶点。在写这几个函数的过程中，我常常会因为没有进行特判而无法通过一些特殊样例。这是我以后写程序的时候需要考虑的健壮性问题。

（2）在书写创建图的函数的过程中，由于未理解头插法是什么意思，以及样例的特殊性让我以为是将结点按照顶点位序插入到链表中，结果导致浪费了很长时间，以后在写程序时，我会注意理解清楚要求再开始写程序。

参考文献

[1] 严蔚敏等.数据结构（C 语言版）.清华大学出版社

[2] Larry Nyhoff. ADTs, Data Structures, and Problem Solving with C++. Second Edition,Calvin College,2005

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

[4] 严蔚敏等.数据结构题集（C 语言版）.清华大学出版社