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 CS2402

学　　号

姓　　名 cupid-qrq

指导教师

报告日期 2025 年 5 月 7 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

1.1.1 实验目的

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。

1.1.2 线性表运算定义

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经存在，$1 \leq i \leq$ *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 中第一个与 e 满足关系 *compare*() 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，$1 \leq i \leq ListLength(L)+1$；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，$1 \leq i \leq ListLength(L)$；操作结果：删除 L 的第 i 个数据元素，用 e 返回的值；

（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*()。

附加功能：

（1）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；

（2）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；

（3）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；

（4）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。

注：保存到文件后，可以由一个空线性表加载文件生成线性表。

（5）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。

注：附加功能（5）实现多个线性表管理应能创建、添加、移除多个线性表，单线性表和多线性表的区别仅仅在于线性表管理的个数不同，多线性表管理应可自由切换到管理的每个表，并可单独对某线性表进行单线性表的所有操作。

1.1.3 实验任务

采用顺序表作为线性表的物理结构，实现 1.1.2 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。

1.2 系统设计

1.2.1 系统结构设计

本系统实现一个具有菜单的多线性表管理系统。程序先给出了相关的常量，数据类型的定义和函数的声明。接着给出每个函数的实现代码。然后在主函数里面实现了一个菜单，用户可以通过输入数字选择相应的操作。每个操作对应一个函数，函数的参数和返回值都在函数定义中给出。菜单可以选择的操作有：

1) 创建线性表
2) 销毁表
3) 清空表
4) 判定空表
5) 求表长
6) 获得元素
7) 查找元素
8) 获得某元素前驱
9) 获得某元素后继
10) 插入元素
11) 删除元素
12) 遍历线性表
13) 求最大连续子数组和
14) 求和为 K 的子数组个数
15) 线性表排序
16) 线性表文件保存
17) 线性表文件加载
18) 添加线性表

19) 删除线性表
20) 定位线性表
21) 切换线性表
22) 输出线性表列表
23) 退出系统

1.2.2 数据结构设计

本系统使用顺序存储结构实现线性表，并定义了一个线性表的结构体，包含了线性表的当前长度，线性表的名称，和存储线性表数据元素的数组。还定义了一个多线性表管理结构体，包含了一个线性表的数组，当前操作的线性表索引，和当前管理的线性表数量。具体定义如下：

```
typedef struct sqlist
{
    ElemType *elem; // 指向存储线性表数据元素的动态数组
    int length;     // 当前线性表的长度
    int listsize;   // 线性表的最大容量
    char name[30];  // 线性表的名称
} SqList;

typedef struct
{
    SqList elem[MAXSIZE]; // 存储多个线性表的数组
    int length;           // 当前管理的线性表数量
    int cur_index;        // 当前操作的线性表索引
} LISTS;
```

1.3 系统实现

下面给出各个函数的实现思路，对于较为复杂的函数，在分析的基础上给出辅助流程图进行说明。函数和系统实现的源代码放在附录中。具体分析如下：

1.3.1 单线性表函数实现

1.status InitList(SqList &L)

函数的参数是线性表 L，返回值是函数执行状态码。函数的实现思路是：首先判断线性表 L 是否已经存在，如果存在则返回 INFEASIBLE；如果不存在，则申请一个大小为 LIST_INIT_SIZE 的动态数组，并将线性表 L 的长度和最大容量初始化为 0 和 LIST_INIT_SIZE，最后返回 OK。

时间复杂度：O(1)

空间复杂度：O(1)

2.status DestroyList(SqList &L)

函数的参数是线性表 L，返回值是函数执行状态码。函数的实现思路是：首先判断线性表 L 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，则释放线性表 L 的动态数组，并将线性表 L 的长度和最大容量初始化为 0 和 0，最后返回 OK。

时间复杂度：O(1)

空间复杂度：O(1)

3.status ClearList(SqList &L)

函数的参数是线性表 L，返回值是函数执行状态码。函数的实现思路是：首先判断线性表 L 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，则将线性表 L 的长度初始化为 0，最后返回 OK。

时间复杂度：O(1)

空间复杂度：O(1)

4.status ListEmpty(SqList &L)

函数的参数是线性表 L，返回值是函数执行状态码。函数的实现思路是：首

先判断线性表 L 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，则判断线性表 L 的长度是否为 0，如果为 0 则返回 TRUE，否则返回 FALSE。

时间复杂度：O(1)

空间复杂度：O(1)

5.status ListLength(SqList &L)

函数的参数是线性表 L，返回值是函数执行状态码。函数的实现思路是：首先判断线性表 L 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，则返回线性表 L 的长度 L.length。

时间复杂度：O(1)

空间复杂度：O(1)

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

函数的参数是线性表 L，待查找的元素位置 i，以及用于返回的元素值 e，返回值是函数执行状态码。函数的实现思路是：首先判断线性表 L 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，则判断 i 是否在 1 到 L.length 之间，如果不在则返回 ERROR；如果在，则用 e 返回线性表 L 中第 i 个数据元素的值。

时间复杂度：O(1)

空间复杂度：O(1)

7.int LocateElem(SqList &L, ElemType e)

函数的参数是线性表 L，待查找的元素值 e，返回值是函数执行状态码。函数的实现思路是：首先判断线性表 L 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，则遍历线性表 L 中的每个数据元素，判断是否与 e 相等，如果相等则返回该数据元素的位序，否则返回 0。

时间复杂度：O(n)

空间复杂度：O(1)

8.status PriorElem(SqList &L, ElemType e, ElemType &pre)

函数的参数是线性表 L，待查找的元素值 e，以及用于返回的前驱元素的序号的 pre，返回值是函数执行状态码。函数的实现思路是：首先判断线性表 L 是

否已经存在，如果不存在则返回 INFEASIBLE；如果存在，查找线性表 L 中第一个与 e 相等的元素的前驱元素，如果该元素不是第一个，则用 pre 返回它的前驱，否则返回 ERROR。

时间复杂度：O(n)

空间复杂度：O(1)

9.status NextElem(SqList &L, ElemType e, ElemType &next)

函数的参数是线性表 L，待查找的元素值 e，以及用于返回的后继元素的序号的 next，返回值是函数执行状态码。函数的实现思路是：首先判断线性表 L 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，查找线性表 L 中第一个与 e 相等的元素的后继元素，如果该元素不是最后一个，则用 next 返回它的后继，否则返回 ERROR。

时间复杂度：O(n)

空间复杂度：O(1)

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

函数的参数是线性表 L，插入位置 i，以及待插入的元素值 e，返回值是函数执行状态码。函数的实现思路是：首先判断线性表 L 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，则判断 i 是否在 1 到 L.length+1 之间，如果不在则返回 ERROR。

如果 i 在合法范围内，则判断线性表 L 的长度是否等于线性表 L 的最大容量，如果相等则扩展线性表 L 的容量；然后将线性表 L 中第 i 个数据元素及其后面的数据元素依次后移一位，将 e 插入到第 i 个位置，最后返回 OK。

时间复杂度：O(n)

空间复杂度：O(1)

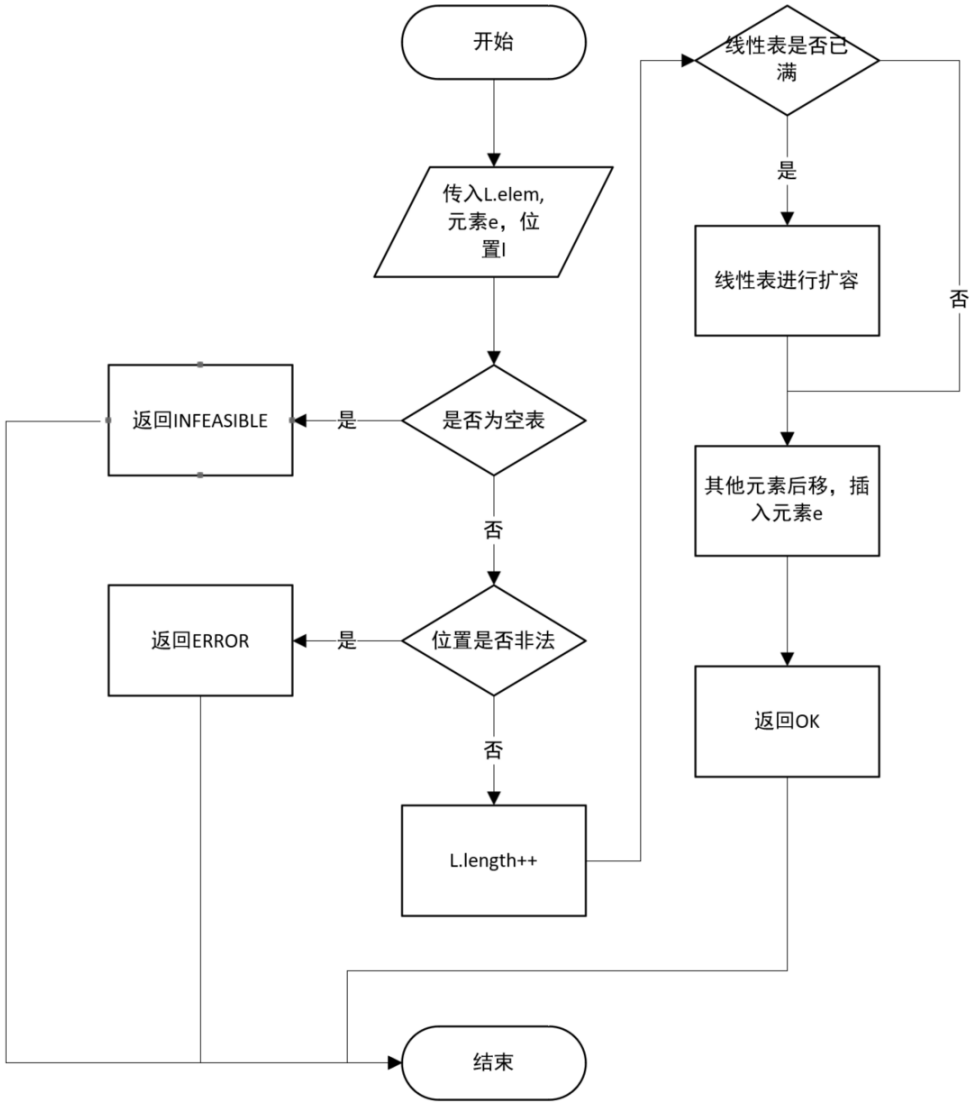

图 1-1　ListInsert 函数的流程图

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

函数的参数是线性表 L，删除位置 i，以及用于接收删除元位置素值的元素 e，返回值是函数执行状态码。函数的实现思路是：首先判断线性表 L 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，则判断 i 是否在 1 到 L.length 之间，如果不在则返回 ERROR；

如果在，则用 e 返回线性表 L 中第 i 个数据元素的值。然后将线性表 L 中第 i+1 个数据元素及其后面的数据元素依次前移一位，最后返回 OK。

时间复杂度：O(n)

空间复杂度：O(1)

12.status ListTraverse(SqList &L)

函数的参数是线性表 L，返回值是函数执行状态码。函数的实现思路是：首先判断线性表 L 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，则遍历线性表 L 中的每个数据元素，依次输出每个数据元素的值，最后返回 OK。

时间复杂度：O(n)

空间复杂度：O(1)

13.int MaxSubArray(SqList &L)

函数的参数是线性表 L, 返回值是最大连续子数组和。函数的实现思路是：首先判断线性表 L 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，则定义两个变量 currentMax 和 Max，分别表示当前最大子数组和和全局最大子数组和。然后遍历线性表 L 中的每个数据元素，更新 currentMax 和 Max 的值，最后返回 Max。

该算法的核心思想是动态规划，笔者在这里给出详细的解析来说明动态规划的过程，并给出代码和流程图，具体算法处理步骤如下：

1) 定义两个变量 currentMax 和 Max，分别表示当前最大子数组和和全局最大子数组和。
2) 如果 currentMax 小于 0，则 currentMax+elem[i] 不可能为最优解，此时舍弃原先旧的 currentMax，重新赋值为 elem[i]。
3) 如果 currentMax 大于 0，此时 currentMax+elem[i] 可能为最优解，故 current-Max+=elem[i]。
4) 比较 currentMax 和 Max 的大小，如果 currentMax 大于 Max，则更新 Max 的值为 currentMax。
5) 遍历线性表 L 中的每个数据元素，依次更新 currentMax 和 Max 的值。
6) 最后返回 Max。
7) 该算法的时间复杂度为 O(n)，空间复杂度为 O(1)。

```
int MaxSubArray(SqList &L)
{
    if (L.elem == NULL)
        return INFEASIBLE;
    int currentMax = L.elem[0];
    int Max = L.elem[0];
    for (int i = 1; i < L.length; i++)
    {
        currentMax =
        max(L.elem[i],currentMax+L.elem[i]);
        Max = max(Max, currentMax);
    }
    return Max;
}
```

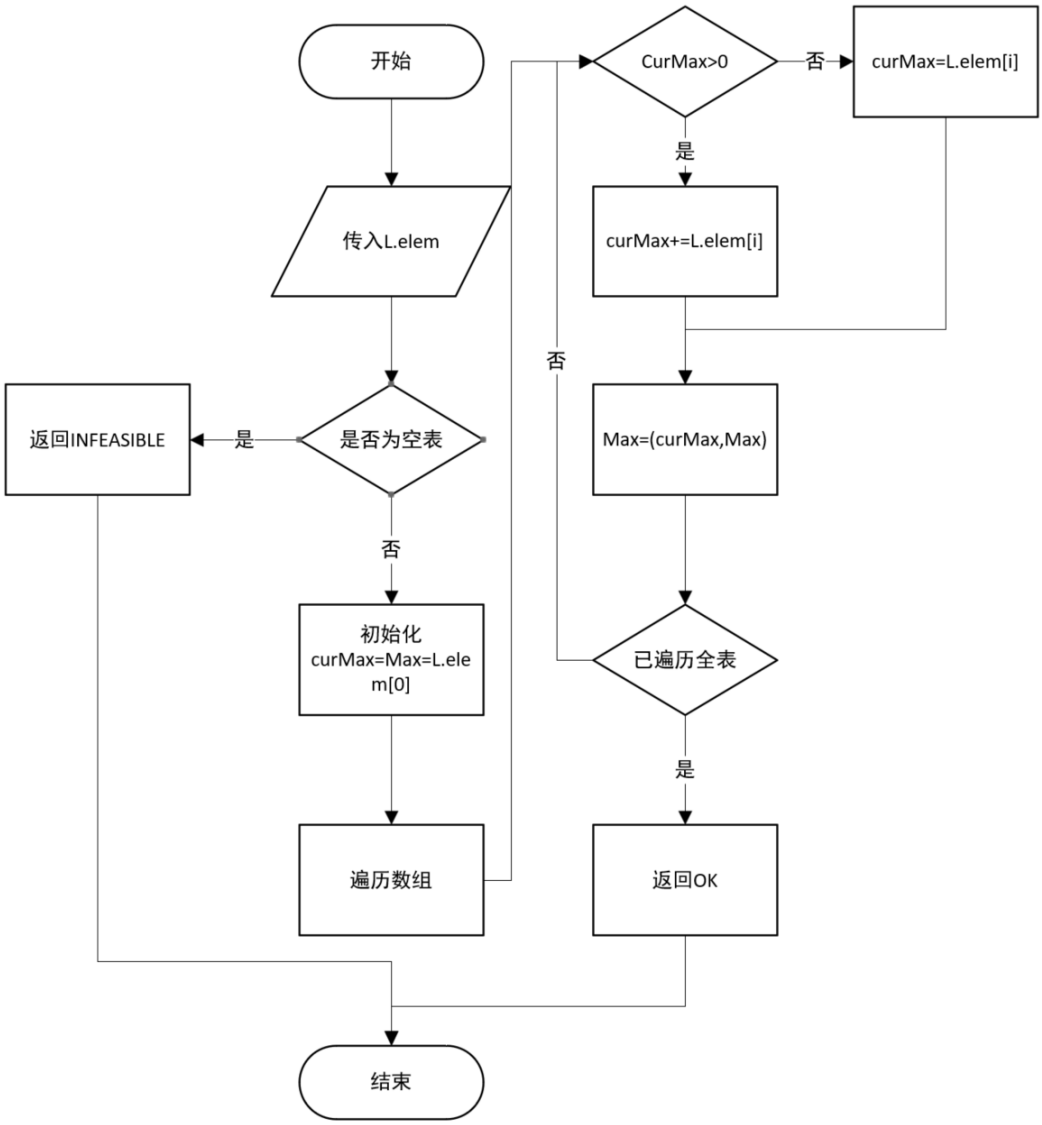

图 1-2　MaxSubArray 函数的流程图

14.int SubArrayNum(SqList &L, int k)

函数的参数是线性表 L 和整数 k，返回值是和为 k 的连续子数组个数。函数的实现思路是：首先判断线性表 k 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，则定义一个哈希表记录前缀和出现的次数，并初始化前缀和为 0 的情况。然后遍历线性表 L 中的每个数据元素，利用前缀和性质更新当前累计和并统计满足条件的子数组数量。最后返回符合条件的子数组总数。

该算法的核心思想是结合动态规划思想与哈希表优化策略，通过维护前缀

和与哈希表来高效地查找是否存在满足条件的子数组。笔者在此给出详细的解析来说明算法的处理过程，并给出代码和流程图，具体算法处理步骤如下：

1) 定义一个哈希表 prefixSum，用于记录前缀和出现的次数，并且我们在一开始时设置 prefixSum[0] = 1，表示前缀和为 0 的情况出现一次。
2) 定义两个变量 sum 和 cnt，分别表示当前累计前缀和和满足条件的子数组数量，初始值均为 0。
3) 遍历线性表 L 中的每个数据元素：
   - 将当前元素加入 sum，得到新的前缀和；
   - 查找是否存在前缀和为 sum - k，若存在，则说明从该位置到当前位置之间的子数组和为 k，此时将 cnt += prefixSum[sum - k]；
   - 更新哈希表中当前前缀和 sum 的出现次数。
4) 最后返回 cnt，即和为 k 的连续子数组个数。

```
int SubArrayNum(SqList &L, int k)
{
    if (L.elem == NULL)
        return INFEASIBLE;

    unordered_map<int, int> prefixSum;
    prefixSum[0] = 1;
    int sum = 0, cnt = 0;

    for (int i = 0; i < L.length; i++)
    {
        sum += L.elem[i];
        cnt += prefixSum[sum - k];
        prefixSum[sum]++;
    }
    return cnt;
}
```

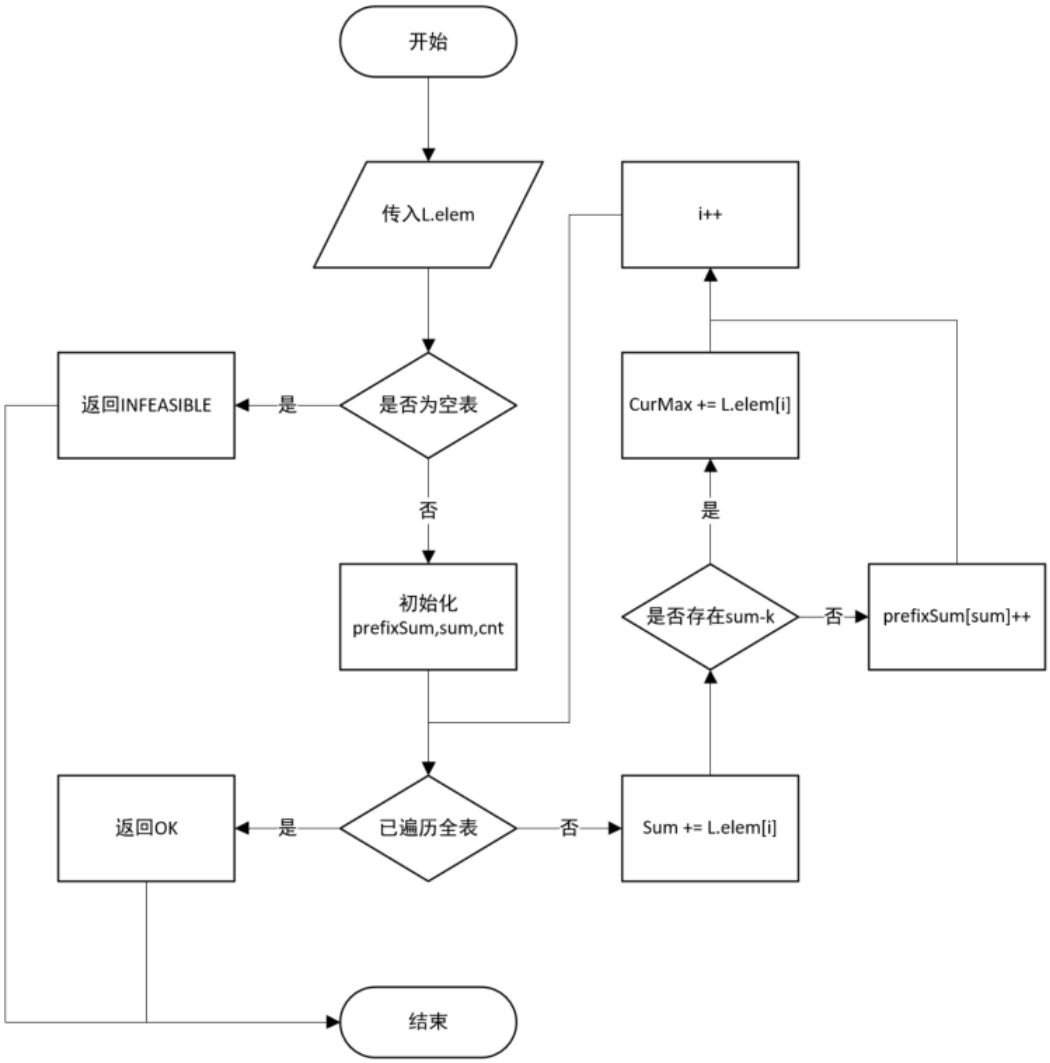

图 1-3　SubArrayNum 函数的流程图

该算法的时间复杂度为 O(n)，空间复杂度为 O(n)，因为我们使用了一个哈希表来存储前缀和的出现次数。但是避免了对每种可能的子数组进行暴力搜索，这样的时间复杂度是 $O(n^2)$，而我们只需要遍历线性表 L 一次，在遍历的过程中维护前缀和和哈希表即可。

值得一提的是，在代码的实现过程中，为了避免 C 语言实现哈希表需要手写哈希函数和冲突解决策略，在这里使用了 C++ STL 中的 unordered_map 来实现哈希表的功能。这样可以大大简化代码的复杂度，提高代码的可读性和可维护性。

15.status sortList(SqList &L)

函数的参数是线性表 L，返回值是函数执行状态码。函数的实现思路是：首先判断线性表 L 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，则使用快速排序算法对线性表 L 进行排序，最后返回 OK。

为了方便，在具体实现利用了 STL 库的 sort 函数来排序，实际上 sort 也是通过快速排序来实现的，在这里不再赘述。

时间复杂度：O(nlogn)

空间复杂度：O(1)

16.status SaveList(SqList &L, char filename)

函数的参数是线性表 L 和文件名 filename，返回值是函数执行状态码。函数的实现思路是：首先判断线性表 L 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，则打开文件 filename，并将线性表 L 中的每个数据元素写入文件中，最后关闭文件并返回 OK。

时间复杂度：O(n)

空间复杂度：O(1)

17.status LoadList(SqList &L, char filename)

函数的参数是线性表 L 和文件名 filename，返回值是函数执行状态码。函数的实现思路是：首先判断线性表 L 是否已经存在，如果不存在则返回 INFEASIBLE；如果存在，则打开文件 filename，并将文件中的数据读入线性表 L 中，最后关闭文件并返回 OK。

时间复杂度：O(n)

空间复杂度：O(1)

1.3.2 多线性表函数实现

下面是多线性表函数的实现思路：

18.status AddList(LISTS &lists, char name)

函数的参数是多线性表管理结构体 lists 和线性表名称 name，返回值是函数执行状态码。

1. 首先判断当前管理的线性表数量是否已经达到最大值，如果达到则返回 INFEASIBLE；
2. 如果没有达到，遍历当前管理的线性表数组，判断是否已经存在同名的线性表，如果存在则返回 ERROR；
3. 如果不存在，则创建一个新的线性表，并将其添加到当前管理的线性表数组中；
4. 然后将新线性表的名称设置为 name，并将新线性表的长度和最大容量初始化为 0；
5. 最后将新线性表的动态数组指针指向 NULL；
6. 接着将当前管理的线性表数量加 1, 把当前操作的线性表索引 lists.cur_index 设置为新添加的线性表的索引；
7. 最后返回 OK。

时间复杂度：O(n)

空间复杂度：O(1)

19.status RemoveList(LISTS &lists，char *name)

函数的参数是多线性表管理结构体 lists 和线性表名称 name，返回值是函数执行状态码。

1. 首先判断当前管理的线性表数量是否为 0，如果为 0 则返回 INFEASIBLE；
2. 如果不为 0，则遍历当前管理的线性表数组，判断是否存在同名的线性表，如果不存在则返回 ERROR；
3. 如果存在，则销毁该线性表，并将当前管理的线性表数量减 1；
4. 接着把从当前位置往后的线性表依次前移一位, 并将原来最后一个线性表的动态数组指针指向 NULL，长度和最大容量初始化为 0，名称清空；
5. 再将当前操作的线性表索引 lists.cur_index 设置为 0；
6. 最后返回 OK。

时间复杂度：O(n)

空间复杂度：O(1)

20.int LocateList(LISTS &lists，char *name)

函数的参数是多线性表管理结构体 lists 和线性表名称 name，返回值是函数执行状态码。函数的实现思路是：首先判断当前管理的线性表数量是否为 0，如果为 0 则返回 INFEASIBLE；如果不为 0，则遍历当前管理的线性表数组，判断是否存在同名的线性表，如果不存在则返回 ERROR；如果存在，则返回该线性表的索引。

时间复杂度：O(n)

空间复杂度：O(1)

21.status SwitchList(LISTS &lists，int index)

函数的参数是多线性表管理结构体 lists 和线性表索引 index，返回值是函数执行状态码。函数的实现思路是：首先判断当前管理的线性表数量是否为 0，如果为 0 则返回 INFEASIBLE；如果不为 0，则判断 index 是否在 0 到 lists.length-1 之间，如果不在则返回 ERROR；如果在，则将当前操作的线性表索引 lists.cur_index 设置为 index，并返回 OK。

时间复杂度：O(1)

空间复杂度：O(1)

22.status PrintList(LISTS &lists)

函数的参数是多线性表管理结构体 lists，返回值是函数执行状态码。函数的实现思路是：首先判断当前管理的线性表数量是否为 0，如果为 0 则返回 INFEASIBLE；如果不为 0，则遍历当前管理的线性表数组，依次输出每个线性表的名称和长度，并返回 OK。

时间复杂度：O(n)

空间复杂度：O(1)

23.void PrintMenu(LISTS Lists)

函数的参数是多线性表管理结构体 lists，返回值是函数执行状态码。函数用于打印菜单，在每次用户操作后重新打印菜单，提示用户操作对应序号和当前操作的线性表名称，一直到最后用户选择退出系统。

至此，整个系统的设计和每一个函数的实现思路都已经给出，下面给出系统

的测试情况。

1.4 系统测试

本部分主要说明针对各个函数正常和异常的测试用例及测试结果：

首先给出进入系统的菜单界面，用户需要先为第一个初始的线性表命名，然后进入系统菜单界面。系统菜单界面如下图所示：

图 1-4 系统菜单界面

1. InitList 函数测试

测试数据：

表 1-1 InitList 测试数据

测试用例	程序输入	理论结果
不存在线性表	1	线性表初始化成功！
已存在线性表	1	线性表创建失败！

测试结果：

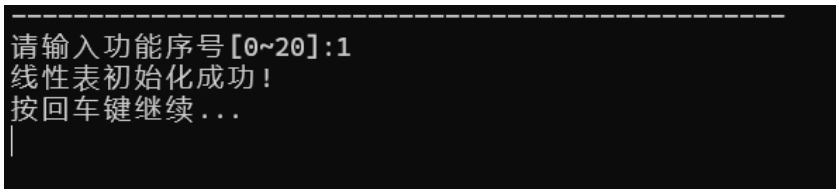

图 1-5　InitList 函数测试 1 结果

图 1-6　InitList 函数测试 2 结果

2. DestroyList 函数测试

测试数据：

表 1-2　DestroyList 测试数据

测试用例	程序输入	理论结果
已存在线性表	2	线性表销毁成功!
不存在线性表	2	线性表销毁失败!

测试结果：

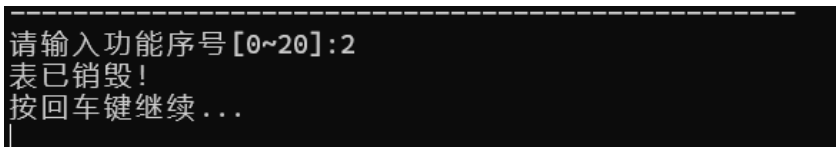

图 1-7　DestroyList 函数测试 1 结果

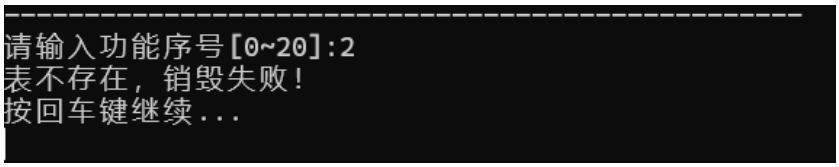

图 1-8　DestroyList 函数测试 2 结果

3. ClearList 函数测试

测试数据：

表 1-3　ClearList 测试数据

测试用例	程序输入	理论结果
已存在线性表	3	线性表清空成功！
不存在线性表	3	线性表清空失败！

测试结果：

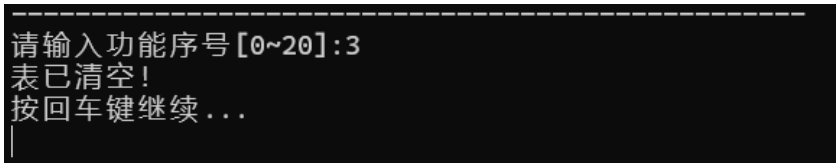

图 1-9　ClearList 函数测试 1 结果

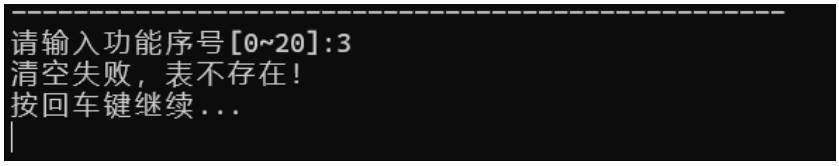

图 1-10　ClearList 函数测试 2 结果

4. ListEmpty 函数测试

测试数据：

表 1-4　ListEmpty 测试数据

测试用例	程序输入	理论结果
已存在空线性表	4	线性表为空！
已存在非空线性表	4	线性表不为空！
不存在线性表	4	线性表不存在！

测试结果：

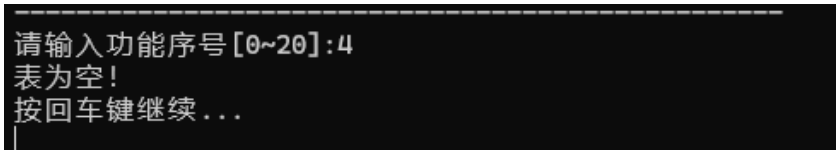

图 1-11　ListEmpty 函数测试 1 结果

图 1-12　ListEmpty 函数测试 2 结果

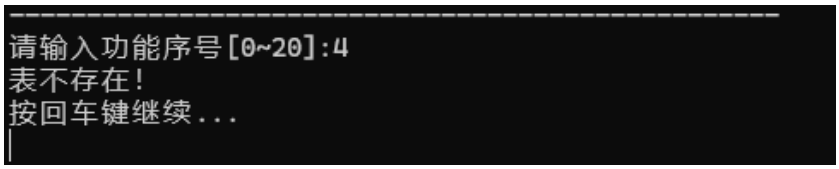

图 1-13　ListEmpty 函数测试 3 结果

5. ListLength 函数测试

测试数据：

表 1-5　ListLength 测试数据

测试用例	程序输入	理论结果
已存在非空线性表	5	线性表长度为 n！
不存在线性表	5	线性表不存在！

测试结果：

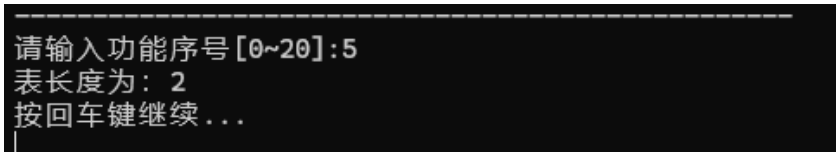

图 1-14　ListLength 函数测试 1 结果

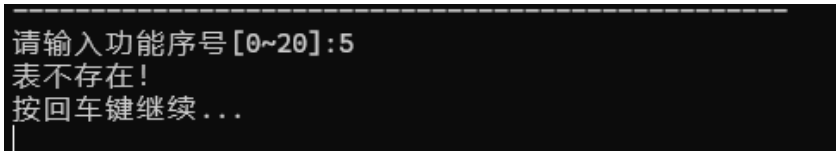

图 1-15　ListLength 函数测试 2 结果

6. GetElem 函数测试

测试数据：

表 1-6　GetElem 测试数据

测试用例	程序输入	理论结果
已存在非空线性表	6	线性表第 i 个元素为 e！
不存在线性表	6	线性表不存在！
i<1 或 i>L.length	6	i 不在合法范围内！

测试结果：

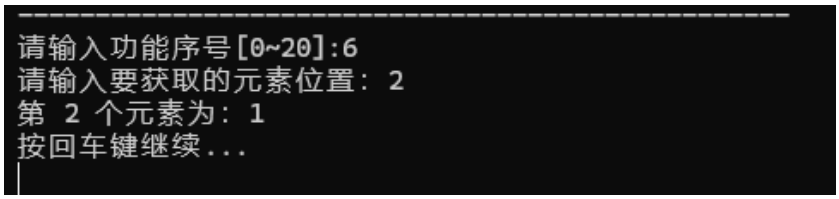

图 1-16　GetElem 函数测试 1 结果

图 1-17　GetElem 函数测试 2 结果

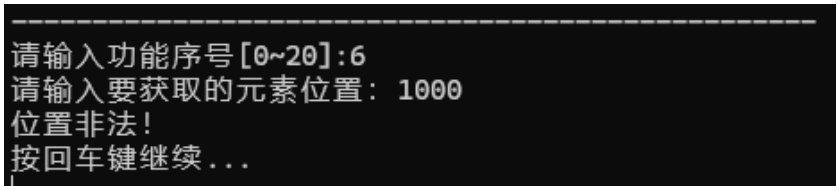

图 1-18　GetElem 函数测试 3 结果

7. LocateElem 函数测试

测试数据：

表 1-7　LocateElem 测试数据

测试用例	程序输入	理论结果
已存在非空线性表且查找元素存在	7	线性表中 e 的位置为 i!
不存在线性表	7	线性表不存在!
查找元素不存在	7	e 不在线性表内!

测试结果：

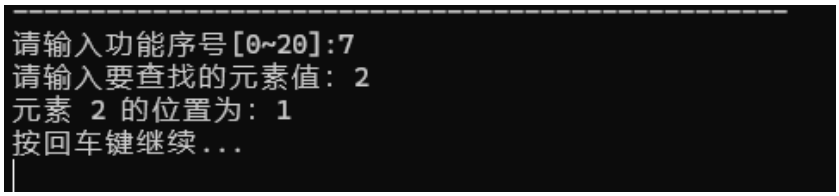

图 1-19　LocateElem 函数测试 1 结果

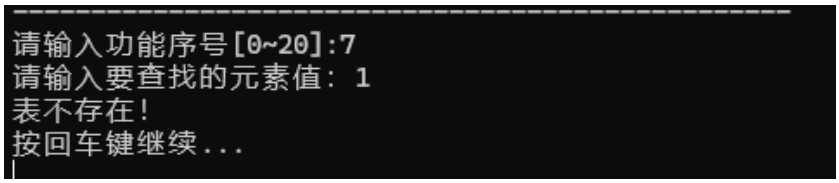

图 1-20　LocateElem 函数测试 2 结果

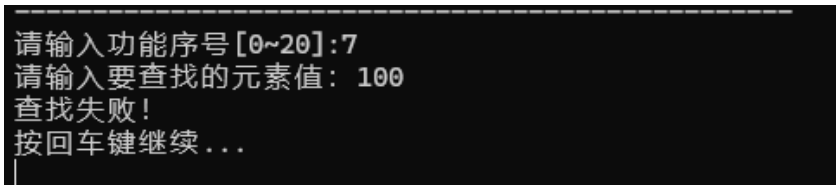

图 1-21　LocateElem 函数测试 3 结果

8. PriorElem 函数测试

测试数据：

表 1-8　PriorElem 测试数据

测试用例	程序输入	理论结果
已存在非空线性表	8	线性表中元素 e 的前驱为 prior!
不存在线性表	8	线性表不存在!
e 为线性表首元素	8	首元素无前驱元素!
e 不存在于线性表	8	e 不在合法范围内!

测试结果：

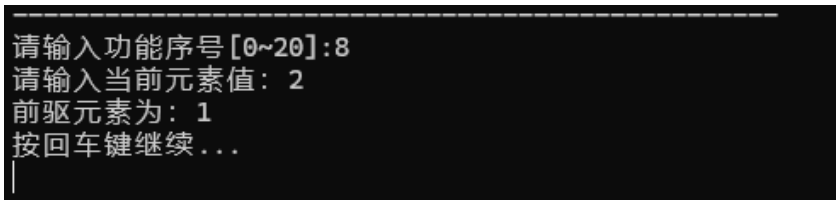

图 1-22　PriorElem 函数测试 1 结果

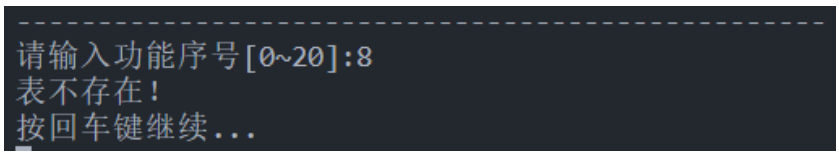

图 1-23　PriorElem 函数测试 2 结果

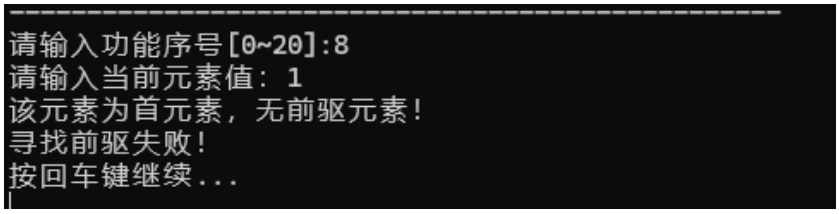

图 1-24　PriorElem 函数测试 3 结果

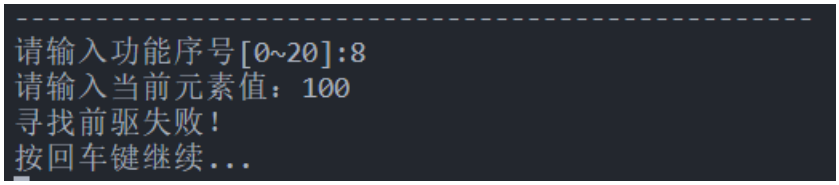

图 1-25　PriorElem 函数测试 4 结果

9. NextElem 函数测试

测试数据：

表 1-9　NextElem 测试数据

测试用例	程序输入	理论结果
已存在非空线性表	9	线性表中第 i 个元素的后继为 next！
不存在线性表	9	线性表不存在！
e 为线性表尾元素	9	尾元素无后继元素！
e 不存在于线性表	9	e 不在合法范围内！

测试结果：

图 1-26　NextElem 函数测试 1 结果

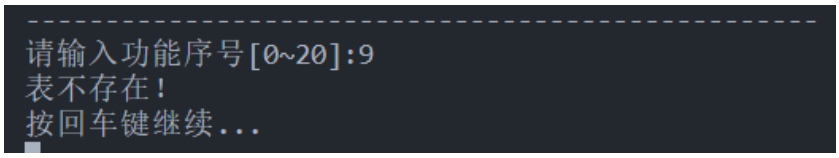

图 1-27　NextElem 函数测试 2 结果

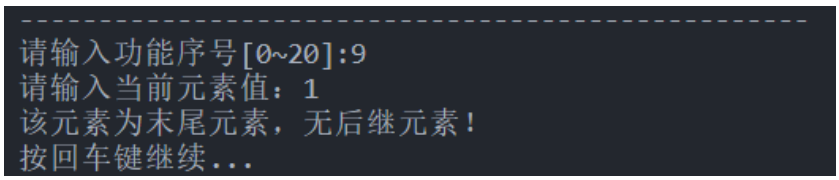

图 1-28　NextElem 函数测试 3 结果

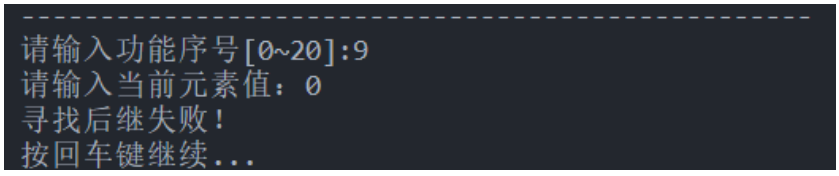

图 1-29　NextElem 函数测试 4 结果

10. ListInsert 函数测试

测试数据：

表 1-10　ListInsert 测试数据

测试用例	程序输入	理论结果
已存在非空线性表	10	线性表插入成功！
不存在线性表	10	线性表不存在！
i<1 或 i>L.length+1	10	i 不在合法范围内！

测试结果：

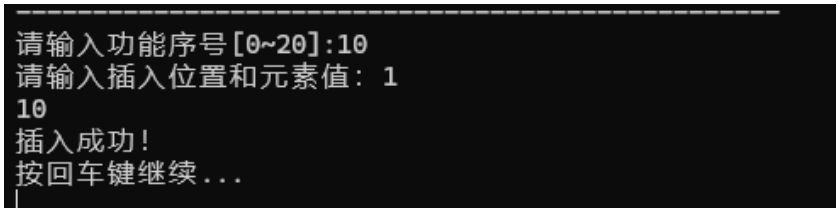

图 1-30　ListInsert 函数测试 1 结果

图 1-31　ListInsert 函数测试 2 结果

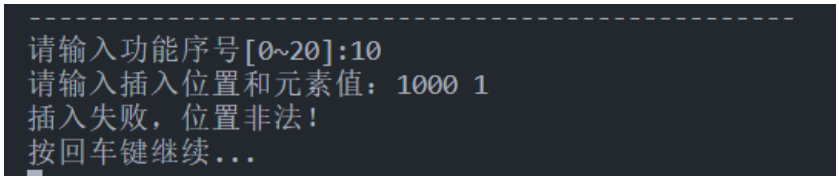

图 1-32　ListInsert 函数测试 3 结果

11. ListDelete 函数测试

测试数据：

表 1-11　ListDelete 测试数据

测试用例	程序输入	理论结果
已存在非空线性表	11	线性表删除成功！
不存在线性表	11	线性表不存在！
i<1 或 i>L.length	11	i 不在合法范围内！

测试结果：

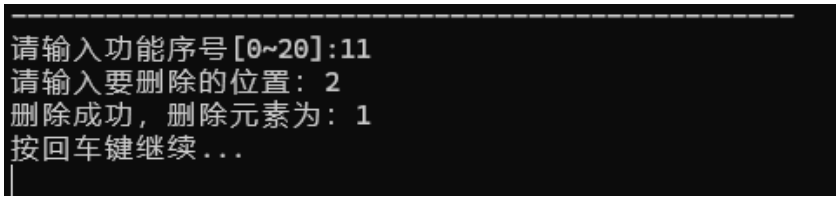

图 1-33　ListDelete 函数测试 1 结果

图 1-34　ListDelete 函数测试 2 结果

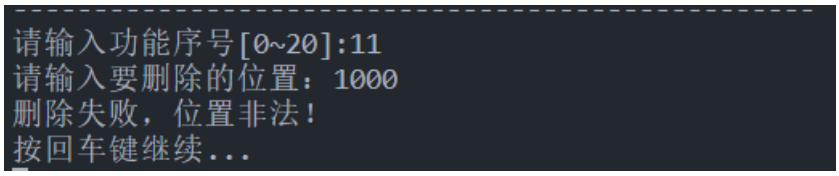

图 1-35　ListDelete 函数测试 3 结果

12. ListTraverse 函数测试

测试数据：

表 1-12　ListTraverse 测试数据

测试用例	程序输入	理论结果
已存在非空线性表	12	线性表遍历成功！
不存在线性表	12	线性表不存在！

测试结果：

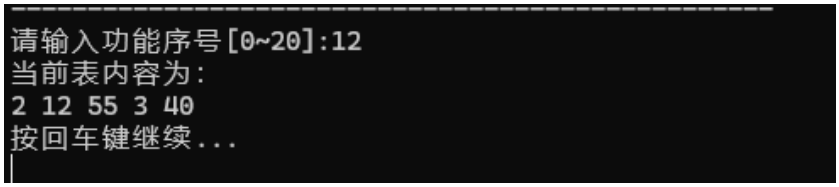

图 1-36　ListTraverse 函数测试 1 结果

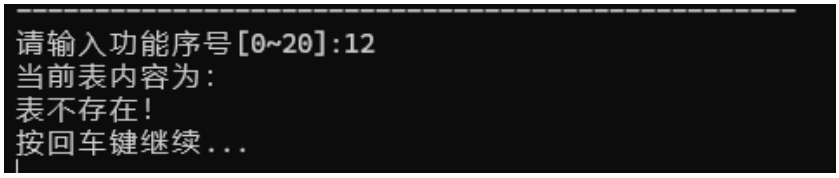

图 1-37　ListTraverse 函数测试 2 结果

13. maxSubArray 函数测试

测试数据：

表 1-13　maxSubArray 测试数据

测试用例	程序输入	理论结果
[-12, 24, -7, 33, -45, 10, -2, 17, -3, 4]	13	50
不存在线性表	13	线性表不存在！

测试结果：

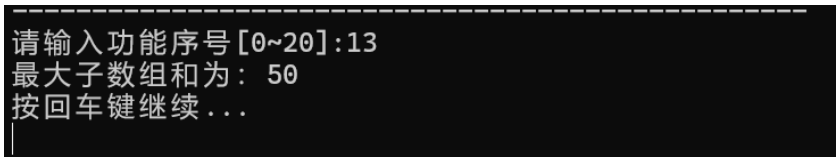

图 1-38　maxSubArray 函数测试 1 结果

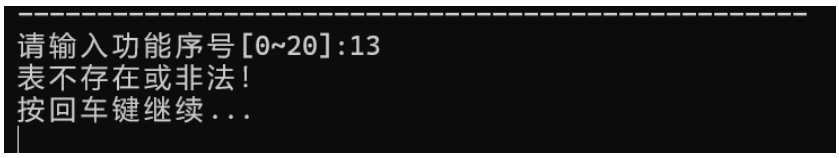

图 1-39　maxSubArray 函数测试 2 结果

14. SubArrayNum 函数测试

测试数据：

表 1-14　SubArrayNum 测试数据

测试用例	程序输入	理论结果
-12, 24, -7, 33, -45, 10, -2, 17, -3, 4（-19）	14	和为 –19 有 2 个
不存在线性表	14	线性表不存在!

测试结果：

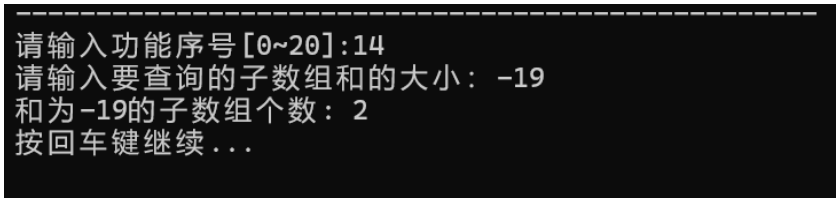

图 1-40　SubArrayNum 函数测试 1 结果

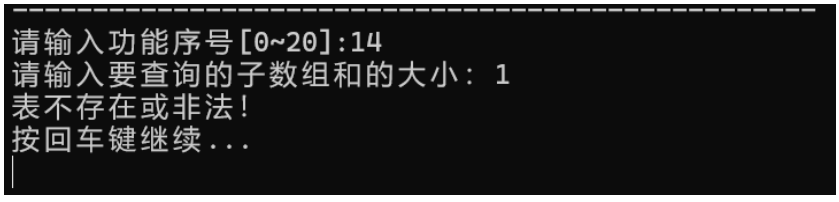

图 1-41　SubArrayNum 函数测试 2 结果

15. sortList 函数测试

测试数据：

表 1-15　sortList 测试数据

测试用例	程序输入	理论结果
随机正负数字同前	15	-45, -12, -7, -3, -2, 4, 10, 17, 24, 33
不存在线性表	15	线性表不存在！

测试结果：

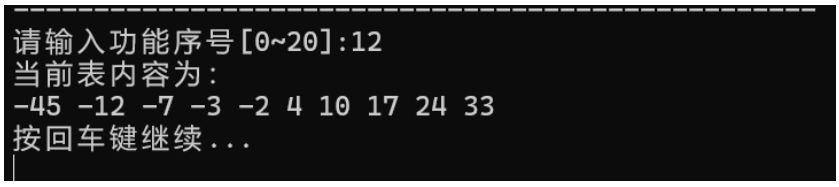

图 1-42　sortList 函数测试 1 结果

图 1-43　sortList 函数测试 2 结果

16. LoadList 函数测试

测试数据：

表 1-16　sortList 测试数据

测试用例	程序输入	理论结果
写有线性表数据的文件	16	文件读入数据成功!

测试结果：

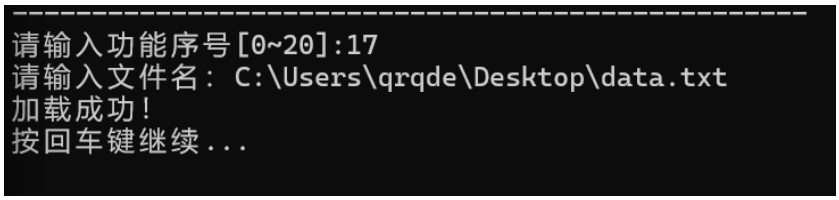

图 1-44　sortList 函数测试 1 结果

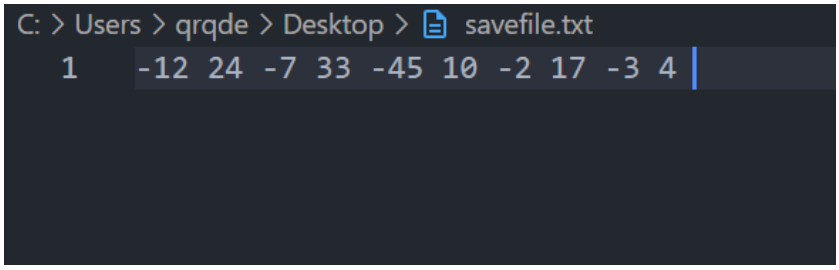

图 1-45　sortList 函数测试 2 结果

17. SaveList 函数测试

测试数据：

表 1-17　SaveList 测试数据

测试用例	程序输入	理论结果
[-12, 24, -7, 33, -45, 10, -2, 17, -3, 4]	17	文件保存成功!

测试结果：

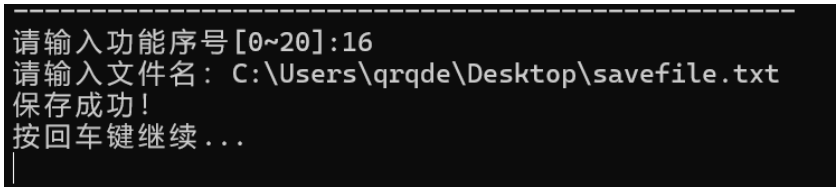

图 1-46　SaveList 函数测试 1 结果

18. AddList 函数测试

测试数据：

选择功能 18，增加线性表的数量，输入新的线性表的名称，并构建一个新的线性表

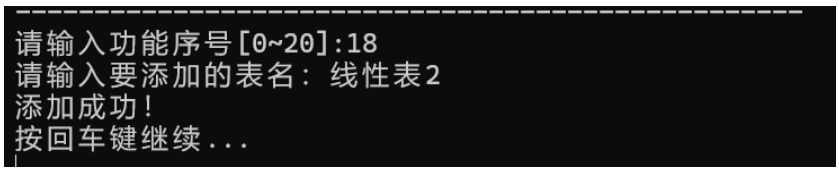

图 1-47　AddList 函数测试 1 结果

19. RemoveList 函数测试

测试数据：

表 1-18　RemoveList 测试数据

测试用例	程序输入	理论结果
已有线性表名称	19	删除线性表成功!
不存在线性表	19	删除线性表失败!

测试结果：

图 1-48　RemoveList 函数测试 1 结果

图 1-49　RemoveList 函数测试 2 结果

20. LoadList 函数测试

测试数据：

表 1-19　LoadList 测试数据

测试用例	程序输入	理论结果
已有线性表名称	20	该线性表序号为 n！
不存在线性表	20	该线性表不存在！

测试结果：

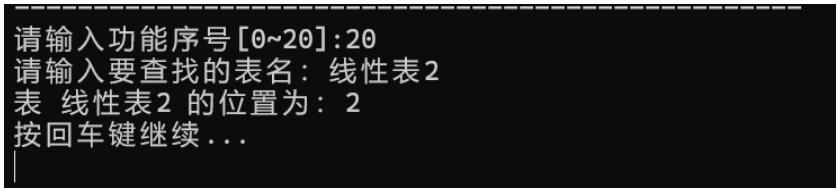

图 1-50　LoadList 函数测试 1 结果

图 1-51　LoadList 函数测试 2 结果

21. SwitchList 函数测试

测试数据：

表 1-20　SwitchList 测试数据

测试用例	程序输入	理论结果
已有线性表名称	21	切换成功，当前为线性表 xxx！
不存在线性表	21	切换失败，该线性表不存在！

测试结果：

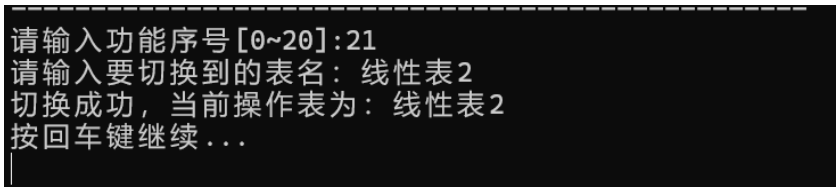

图 1-52　SwitchList 函数测试 1 结果

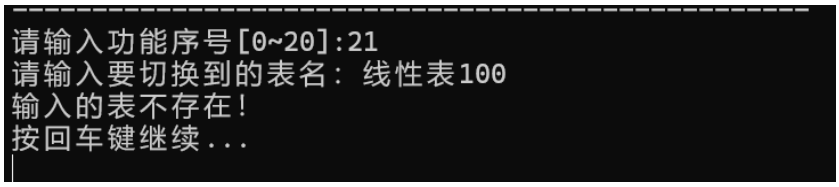

图 1-53　SwitchList 函数测试 2 结果

22. PrintList 函数测试

测试数据：

选择功能 22，打印当前多线性表的所有线性表的名称

图 1-54　PrintList 函数测试 1 结果

至此，所有的测试用例都已经完成，每个函数都进行了多种情况的测试，包括一些边界情况的测试，测试结果都符合预期，说明程序的功能实现是正确的。

1.5 实验小结

本次实验完成了基于链式存储结构的线性表的实现，主要实现了单线性表的基本操作，一些附加的功能比如求子数组的和、子数组为 k 个数等，以及线性表的文件读写操作和多线性表系统的实现。通过这次数据结构实验，我在很多方面都有很大收获。

- **编程思维提升** 通过这次实验，我对数据结构的概念和框架有了更深刻的理解，明白了怎么通过一步步搭建数据结构的定义和操作来实现一个完整的程序，实现了从之前的一个代码只解决一个问题到现在的完成一个完整的系统的转变和进步。
- **编程能力提升** 通过实验我复习看了上学期的 C 语言基础知识，回顾和复习比如文件读写操作的实现，以及嵌套结构体的定义和使用等，还熟悉了使用 VSCode 进行编程和调试的流程。
- **探索新知，拓宽知识面** 在实验中遇到函数参数使用 & 的传递方式，促使我学习了 cpp 语言的引用传递方式，了解了引用传递和指针传递的区别和联系，明白了引用传递的本质是指针传递。还借此学习了一些 cpp 的基础语法和特性，学会运用基础的 STD 库函数，比如 sort 函数、getline 函数等，了解了 STL 的基本使用方法。比如，我使用了 STL 中的 unordered_map 来存储子数组和的值和出现次数，使用了 STL 中的 sort 函数来对线性表进行排序。这些操作大大提高了编程的效率和可读性。
- **算法思维提升** 对于求子数组和的问题，我使用了双指针的算法来解决，避免了暴力破解的时间复杂度为 $O(n^2)$ 的情况，使用了哈希表来存储子数组和的值和出现次数，避免了重复计算，提高了算法的效率。通过 deepseek 和洛谷等平台学习，我学到很多算法思想和技巧，对动态规划、贪心算法、分治算法等有了更深刻的理解和认识。

2 基于邻接表的图实现

2.1 问题描述

2.1.1 实验目的

通过实验达到：（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。

2.1.2 图基本运算定义

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。具体运算功能定义如下：

（1）创建图：函数名称是 CreateGraph(G,V,VR)；初始条件是 V 是图的顶点集，VR 是图的关系集；操作结果是按 V 和 VR 的定义构造图 G。
注：要求图 G 中顶点关键字具有唯一性。后面各操作的实现，也都要满足一个图中关键字的唯一性，不再赘述；V 和 VR 对应的是图的逻辑定义形式，比如 V 为顶点序列，VR 为关键字对的序列。不能将邻接矩阵等物理结构来代替 V 和 VR。

（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，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（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 $\langle v, w\rangle$，如果图 G 是无向图，还需要增加 $\langle w, v\rangle$。

（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 $\langle v, w\rangle$，如果图 G 是无向图，还需要删除 $\langle w, v\rangle$。

（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。

附加功能：

（1）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；

（2）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；

（3）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；

（4）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。

注：保存到文件后，可以直接加载文件生成图。

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

注：附加功能（5）实现多个图管理应能创建、添加、移除多个图，单图和多图的区别仅仅在于图管理的个数不同，多图管理应可自由切换到管理的每个图，并可单独对某图进行单图的所有操作。

2.1.3 实验任务

采用邻接表作为图的物理结构，实现 4.2 节的基本运算，可任选无向图、有向图、无向网和有向网这四种图中的一种实现。本实验选择无向图来实现。其中 ElemType 为数据元素的类型名，具体含义可自行定义，但要求顶点类型为结构型，至少包含二个部分，一个是能唯一标识一个顶点的关键字（类似于学号或职工号），另一个是其它部分。

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

2.2 系统设计

2.2.1 系统结构设计

本系统实现一个具有菜单的多图管理系统。程序先给出了相关的常量，数据类型的定义和函数的声明。接着给出每个函数的实现代码。然后在主函数里面实现了一个菜单，用户可以通过输入数字选择相应的操作。每个操作对应一个函数，函数的参数和返回值都在函数定义中给出。菜单可以选择的操作有：

1) 创建图
2) 销毁图
3) 查找顶点
4) 顶点赋值
5) 获得第一邻接点
6) 获得下一邻接点
7) 插入顶点
8) 删除顶点

9) 插入弧
10) 删除弧
11) 深度优先搜索遍历
12) 广度优先搜索遍历
13) 距离小于 k 的顶点集合
14) 顶点间最短路径长度
15) 图的连通分量数
16) 实现图的文件形式保存
17) 从文件读取图
18) 添加图
19) 删除图
20) 切换图
21) 打印图列表
22) 打印图和图的邻接表
23) 退出系统

2.2.2 数据结构设计

本系统使用邻接表作为无向图的物理结构，同时设计了一个支持多图管理的结构体，便于用户同时操作多个图。具体定义如下：

1. 顶点结构体 *VertexType*，每个顶点包含一个关键字和一个其他部分，其中关键字可以是任意类型体。

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

2. 弧结构体 *ArcNode*，包含一个指向下一个结点的指针和一个顶点位置编号。这里的顶点位置编号是指在邻接表中该顶点的位置序号。

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

3. 邻接表结点类型 *VNode*，每个顶点对应一个邻接表头结点，*data* 保存该顶点的信息，*firstarc* 指向与该顶点相连的第一条边。这一设计形成一个“顶点 + 边链”的结构，体现了邻接表的基本思路。

```
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];
```

4. 图的邻接表类型 *Graph*，该结构表示一张完整的图。*vertices* 是顶点数组（邻接表），*vexnum* 和 *arcnum* 分别记录顶点数和边数，*kind* 指示图的类型（如有向图、无向图等），*name* 是图的名称.

```
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
    char name[30];         // 图的名称
} ALGraph;
```

5. 多图管理结构体 *GRAPHS*，该结构用于统一管理多个图。*elem* 是图的数组，最多可管理 MAX_GRAPH_SIZE 个图，*length* 记录当前已有的图的数量，*cur_index* 表示当前操作的图的索引。这种设计可支持多图的切换、删除、查找等操作，满足实际应用中对多个图同时管理的需求。

```
typedef struct
{
    ALGraph elem[MAX_GRAPH_SIZE];
    int length;
    int cur_index;
} GRAPHS;
```

2.3 系统实现

2.3.1 基础功能函数实现

1. status CreateGraph(ALGraph &G, VertexType V[], KeyType VR[][2])

（1）输入：*G* 为待构建的图结构体引用，*V[]* 为顶点数组，以 *key == -1* 作为结束标志，*VR[][2]* 为边数组，每对表示一条边，以 *{-1, -1}* 结束。

（2）输出：函数执行状态，成功返回 *OK*，输入非法或内存申请失败时返回 *ERROR* 或 *OVERFLOW*。

（3）算法思想描述：通过顺序遍历顶点和边数组，初始化图中各顶点结点，并依次插入边，构造邻接表形式的无向图。过程中排除非法输入、重复边和自环边，确保图结构合法。

（4）算法处理步骤：首先遍历 *V[]* 和 *VR[]* 数组，统计顶点数和边数。之后调用辅助函数 *isKeyTypeUnique* 判断顶点的关键字和附加信息是否唯一，其中使用了 *unordered_map* 提高查重效率。若不合法，直接返回 *ERROR*。

接着初始化图的每个顶点，将顶点信息存入图中，并将邻接表指针设为空。然后遍历边数组，对于每条边，通过 *LocateVertex* 查找对应顶点下标，若顶点不存在、重复或构成自环，则跳过该边。否则，动态创建两个 *ArcNode* 结点，分别插入两个顶点的邻接表，实现双向连接，构造无向图。

（5）时间复杂度：$\mathcal{O}(n+m)$，其中 n 为顶点数，m 为边数。

（6）空间复杂度：$\mathcal{O}(n+m)$，主要用于图结构和邻接表所需的存储。

2. status DestroyGraph(ALGraph &G)

（1）输入：*G* 为待销毁的图结构体引用。

（2）输出：函数执行状态，操作成功返回 *OK*。

（3）算法思想描述：遍历图中所有顶点，释放各自邻接表中动态分配的边结点，并将顶点数和边数清零，从而彻底销毁图结构。

（4）算法处理步骤：通过循环依次访问每个顶点，使用指针 *p* 遍历其邻接表，并用临时指针 *q* 保存当前结点，释放后再移动至下一结点。所有边释放完毕后，清空邻接表指针，并将图的顶点数 *vexnum* 与边数 *arcnum* 置为 0，彻底清除图中的所有信息。

（5）时间复杂度：$\mathcal{O}(n+m)$，其中 n 为顶点数，m 为边数，邻接表中每条边结点都需要被释放。

（6）空间复杂度：$\mathcal{O}(1)$，不引入额外辅助空间，仅在释放原图结构。

3. int LocateVertex(ALGraph G, KeyType u)

（1）输入：图结构体 *G*，以及需要查找的顶点关键字 *u*。

（2）输出：返回顶点 *u* 在图中的位序（下标），若图中不存在该顶点，则返回 *-1*。

（3）算法思想描述：函数通过遍历图中所有顶点，依次比较每个顶点的关键字是否等于给定关键字 *u*，找到即返回该顶点的索引位置。若遍历结束仍未找到，则说明该顶点不存在于图中。

（4）算法处理步骤：从顶点数组起始位置开始，按顺序逐个顶点比较关键字，匹配成功立即返回对应下标；若循环结束都无匹配，则返回 *-1* 表示查找失败。

（5）时间复杂度：$\mathcal{O}(n)$，其中 n 为图中顶点数，最坏情况下需遍历所有顶点。

（6）空间复杂度：$\mathcal{O}(1)$，只使用了少量辅助变量，未占用额外空间。

4. status PutVex(ALGraph &G, KeyType u, VertexType value)

（1）输入：图结构体引用 *G*，需要修改的顶点关键字 *u*，以及新的顶点数据 *value*。

（2）输出：若顶点关键字存在且修改后保证关键字和附加信息唯一，返回 *OK*；查找失败或修改导致关键字重复，返回 *ERROR*。

（3）算法思想描述：函数首先遍历图中顶点数组，查找关键字为 *u* 的顶点。找到后，暂存原顶点数据，将该顶点数据修改为 *value*。随后调用辅助函数 *isKeyTypeUniqueInAdjlist* 验证修改后的顶点数组中，所有顶点关键字和附加信息是否保持唯一性。若唯一性被破坏，则恢复原顶点数据并返回错误，确保图中顶点的

属性一致且无重复。

辅助函数 *isKeyTypeUniqueInAdjlist* 利用两个哈希表分别统计顶点数组中关键字（*key*）和附加信息（*others*）的出现次数，遍历过程中一旦发现重复立即返回 *ERROR*，保证顶点数据唯一性。

（4）算法处理步骤：顺序遍历顶点数组定位目标顶点，尝试修改顶点信息后调用唯一性检测函数；若检测失败回滚数据，否则返回成功。若未找到顶点，返回错误。

（5）时间复杂度：$\mathcal{O}(n)$，其中 n 为顶点数，主要耗时在查找和唯一性检测。

（6）空间复杂度：$\mathcal{O}(n)$，主要用于辅助函数中哈希表存储顶点属性计数。

5. int FirstAdjVex(ALGraph G, KeyType u)

（1）输入：图结构体 *G*，以及需要查找的顶点关键字 *u*。

（2）输出：返回顶点 *u* 的第一个邻接顶点的位序，若不存在则返回 *-1*。

（3）算法思想描述：函数首先调用 *LocateVertex* 查找关键字为 *u* 的顶点，若不存在则返回 *-1*。若存在，则通过该顶点的邻接表头结点获取第一个邻接结点的位序。

（4）算法处理步骤：先查找顶点 *u* 的位序，若存在则返回其邻接表中第一个结点的位序；若不存在，则返回 *-1*。

（5）时间复杂度：$\mathcal{O}(1)$，查找和返回邻接结点均为常数时间操作。

（6）空间复杂度：$\mathcal{O}(1)$，只使用了少量辅助变量，未占用额外空间。

6. int NextAdjVex(ALGraph G, KeyType v, KeyType w)

（1）输入：图结构体 *G*，顶点关键字 *v* 及其邻接顶点关键字 *w*。

（2）输出：若顶点 *v* 和 *w* 均存在且 *w* 是 *v* 的邻接顶点，则返回 *v* 相对于 *w* 的下一个邻接顶点在图中顶点数组中的位序；若 *w* 是 *v* 邻接顶点的最后一个，或者 *v*、*w* 中任意一个顶点不存在，则返回 *-1*。

（3）算法思想描述：首先通过辅助函数 *LocateVertex* 查找关键字 *v* 和 *w* 对应的顶点下标，若任一顶点不存在，返回 *-1*。然后遍历顶点 *v* 的邻接表，查找邻接点链表中对应 *w* 的节点位置。找到该节点后，判断其后继节点是否存在，若存在则返回后继节点对应顶点的位序，否则返回 *-1*。

（4）算法处理步骤：先定位 *v* 和 *w* 的顶点索引，接着遍历 *v* 的邻接链表节

点，匹配邻接顶点 *w*，若匹配成功且存在下一个邻接顶点，则返回该邻接顶点下标；否则返回 *-1*。

（5）时间复杂度：$\mathcal{O}(d)$，其中 d 是顶点 *v* 的邻接点数量。

（6）空间复杂度：$\mathcal{O}(1)$，仅使用少量辅助变量，无额外空间消耗。

7. status InsertVex(ALGraph &G, VertexType v)

（1）输入：图结构体引用 *G* 和待插入的顶点信息 *v*。

（2）输出：函数执行状态，插入成功返回 *OK*，若图中顶点数达到最大容量或插入顶点的关键字或附加信息已存在则返回 *ERROR*。

（3）算法思想描述：该函数用于向图中添加一个新的顶点。首先判断图中顶点数是否已达最大限制，若是则无法插入，返回错误。然后遍历已有顶点，检查待插入顶点的关键字和附加信息是否与已有顶点重复，避免顶点信息冲突。若无重复，将新顶点信息存入图的顶点数组尾部，并初始化该顶点的邻接表头指针为空，最后顶点数加一，表示成功插入。

（4）算法处理步骤：先判断容量限制，再遍历顶点数组做唯一性检查，若通过则直接在末尾添加顶点，邻接表指针置空，更新顶点数。

（5）时间复杂度：$\mathcal{O}(n)$，其中 n 是当前图中顶点数。

（6）空间复杂度：$\mathcal{O}(1)$，仅使用少量辅助变量，无额外空间消耗。

8. status DeleteVex(ALGraph &G, KeyType v)

（1）输入：图结构体引用 *G* 和待删除顶点的关键字 *v*。

（2）输出：函数执行状态，成功删除返回 *OK*，若顶点不存在或图中只有一个顶点时返回 *ERROR*。

（3）算法思想描述：该函数用于删除图中指定关键字对应的顶点及其所有相关边。首先定位待删除顶点在顶点数组中的索引，若不存在则返回错误。若图中只有一个顶点，禁止删除以防图为空。然后分三步操作删除：

- 遍历图中所有顶点的邻接表，查找并删除所有指向待删顶点的边，同时调整相关顶点索引，保证图结构一致性。每删除一条边，边数减一。
- 释放待删顶点自身邻接表中所有边的内存，清空其邻接关系。
- 从顶点数组中删除该顶点，采用覆盖后移方式调整数组，顶点数减一。

（4）算法处理步骤：调用辅助函数定位顶点，判断异常情况后，遍历邻接表

删除指向该顶点的边，释放待删顶点的邻接边内存，最后调整顶点数组顺序并更新顶点数和边数。

（5）时间复杂度：$\mathcal{O}(n+m)$，其中 n 是顶点数，m 是边数。遍历邻接表和顶点数组均为线性时间。

（6）空间复杂度：$\mathcal{O}(1)$，在原图结构上进行修改，无额外大空间开销。

9. status InsertArc(ALGraph &G, KeyType v, KeyType w)

1）输入：图结构体引用 *G*，两个顶点关键字 *v* 和 *w*，表示要在图中插入一条无向边 <*v*,*w*>。

（2）输出：函数执行状态，若插入成功返回 *OK*，若顶点不存在或边已存在返回 *ERROR*。

（3）算法思想描述：本函数旨在为无向图 *G* 中的顶点 *v* 和 *w* 增加一条边。通过查找顶点 *v* 和 *w* 在图中对应的下标，验证两顶点存在性后，再判断两顶点之间是否已存在边，避免重复插入。若不存在边，则为两顶点分别插入指向对方的邻接点，形成无向边的双向连接。

（4）算法处理步骤：

首先调用辅助函数 *LocateVertex* 查找顶点 *v* 和 *w* 在顶点数组中的位置，若任一顶点不存在，立即返回 *ERROR*。

遍历顶点 *w* 的邻接表，检查是否已经存在指向 *v* 的边，如果存在则返回 *ERROR*，避免重复插入。

若不存在，则动态分配两个邻接点结点（*ArcNode*），分别插入到 *w* 和 *v* 的邻接表头，实现双向邻接关系。

最后将图的弧数 *arcnum* 加一，表示新增了一条边。

（5）时间复杂度：$\mathcal{O}(d_w)$，其中 d_w 是顶点 *w* 的邻接点数量。整体来看，插入操作的复杂度主要取决于检查重复边的遍历时间。

（6）空间复杂度：$\mathcal{O}(1)$，主要用于两个新分配的邻接结点内存，属于常数空间开销。

10. status DeleteArc(ALGraph &G, KeyType v, KeyType w)

（1）输入：图结构体引用 *G*，两个顶点关键字 *v* 和 *w*，表示要删除的无向边 <*v*,*w*>。

（2）输出：函数执行状态，成功删除边返回 *OK*，若顶点不存在或边不存在则返回 *ERROR*。

（3）算法思想描述：本函数通过邻接表结构删除无向图中连接顶点 v 和 w 的边。无向边在邻接表中分别以单链表节点的形式记录于两个顶点的邻接表中，因此需要分别在 v 和 w 的邻接链表中找到对应邻接点并删除，从而实现边的移除。

（4）算法处理步骤：

- 利用辅助函数 *LocateVertex* 定位顶点 v 和 w 的索引，若任意顶点不存在，返回 *ERROR*。
- 在顶点 w 的邻接单链表中遍历查找指向 v 的邻接点，若不存在该边，返回 *ERROR*。
- 在顶点 w 的邻接链表中删除该邻接点节点，若该节点是头结点则调整头指针，否则调整前驱节点指针，释放该节点内存。
- 类似地，在顶点 v 的邻接单链表中删除指向 w 的邻接点节点。
- 删除成功后，将图的边数 *arcnum* 减一。

（5）时间复杂度：$\mathcal{O}(d_w + d_v)$，其中 d_w 和 d_v 分别为顶点 w 和 v 的度，即邻接节点数量。

（6）空间复杂度：$\mathcal{O}(1)$，删除操作只涉及指针操作和内存释放，不需要额外空间。

11. status DFSTraverse(ALGraph G)

（1）输入：图结构体引用 G，表示待遍历的无向图。

（2）输出：函数执行状态，始终返回 *OK*。

（3）算法思想描述：本函数使用深度优先搜索（*DFS*）对整个图进行遍历。为防止图不连通而漏掉某些顶点，函数对每个未访问顶点调用一次递归函数 *dfs*，从而实现整个图的深度优先遍历，并在访问过程中调用 *visit* 函数处理每个顶点的数据。

（4）算法处理步骤：

- 创建长度为顶点数的整型数组 *visited*[]，初始值全部设为 0，用于标记顶点是否被访问。
- 对每一个顶点 i，如果其 $visited[i] = 0$，调用递归函数 $dfs(G, i, visited)$ 对以该顶点为起点的连通分量进行深度优先搜索。

- 每次访问顶点时，调用 *visit*($G.vertices[n].data$) 打印其关键字和其他信息。

（5）*dfs* 子函数说明：函数 *dfs*($G, n, visited$) 采用递归方式完成从顶点 n 开始的深度优先遍历，具体处理步骤如下：

- 标记当前顶点 n 为已访问：$visited[n] = 1$。
- 调用 *visit*($G.vertices[n].data$) 输出当前顶点信息。
- 遍历顶点 n 的邻接链表，依次对未被访问的邻接点 $p \rightarrow adjvex$ 递归调用 *dfs*。

（6）时间复杂度：$\mathcal{O}(n+e)$，其中 n 为顶点数，e 为边数，每个顶点和边最多被访问一次。

（7）空间复杂度：$\mathcal{O}(n)$，递归调用栈最大深度为 n，并需 n 大小的 *visited* 数组。

12. status BFSTraverse(ALGraph G)

（1）输入：图结构体引用 *G*，表示待遍历的无向图。

（2）输出：函数执行状态，始终返回 *OK*。

（3）算法思想描述：本函数采用广度优先搜索（*BFS*）对图 *G* 进行遍历。BFS 使用队列实现，依次访问每个顶点的所有邻接点，确保每个顶点仅访问一次。为防止图中存在多个连通分量导致漏遍历，函数对每个未访问顶点均调用一次 BFS 扫描过程。

（4）算法处理步骤：

- 创建长度为顶点数的整型数组 *visited*[]，初始值为 0。
- 创建一个空队列 q。
- 对每个顶点 i，若 $visited[i] = 0$，则：
  - 访问该顶点并标记为已访问，入队。
  - 当队列非空时，取出队头顶点 n，遍历其所有未访问的邻接点 m，依次访问、标记并入队。

（5）时间复杂度：$\mathcal{O}(n+e)$，其中 n 为顶点数，e 为边数。每个顶点入队、出队一次，每条边被遍历一次。

（6）空间复杂度：$\mathcal{O}(n)$，主要为 *visited*[] 数组和辅助队列。

13. status SaveGraph(ALGraph G, const char FileName[])

（1）输入：图结构体引用 *G*，表示待保存的无向图；字符数组常量 *FileName*，表示保存文件的路径和名称。

（2）输出：函数执行状态，若保存成功返回 *OK*，若文件打开失败则返回 *ERROR*。

（3）算法思想描述：本函数将图 *G* 的基本信息（顶点数、边数）、各顶点数据以及邻接表结构保存到指定的文本文件中。采用邻接表遍历的方式，逐个写入顶点和邻接点信息，以便后续的图加载操作。

（4）算法处理步骤：

- 使用 *fopen* 以写模式打开文件 *FileName*，若失败则返回 *ERROR*。
- 先写入顶点数 *G.vexnum* 和边数 *G.arcnum*。
- 遍历每个顶点 i：
  - 写入顶点的 *key* 和 *others* 信息。
  - 遍历顶点 i 的邻接表，逐个写入邻接点索引。
  - 写入结束标志 -1，表示该顶点邻接点输入结束。
  - 写入换行符，准备写下一个顶点。
- 关闭文件并返回 *OK*。

（5）时间复杂度：$\mathcal{O}(n+e)$，其中 n 为顶点数，e 为边数。每个顶点和每条边均被遍历一次。

（6）空间复杂度：$\mathcal{O}(1)$，主要使用了常量空间的辅助变量和指针。

14. status LoadGraph(ALGraph &G, const char FileName[])

（1）输入：图结构体引用 *G*，用于保存读取后的无向图；字符数组常量 *FileName*，表示待加载文件的路径和名称。

（2）输出：函数执行状态，若加载成功返回 *OK*，若文件打开或格式错误则返回 *ERROR*，若内存分配失败则返回 *OVERFLOW*。

（3）算法思想描述：本函数从文本文件中逐行读取图 *G* 的基本信息（顶点数、边数）、顶点数据及邻接表信息。对每个顶点及其邻接点逐个解析，并使用尾插法动态构建邻接表，最终在内存中还原图的完整结构。

（4）算法处理步骤：

- 使用 *fopen* 以读模式打开文件 *FileName*，若失败则返回 *ERROR*。

- 读取顶点数 *G.vexnum* 和边数 *G.arcnum*，若超出最大限制则返回 *ERROR*。
- 对每个顶点 i：
  - 读取顶点的 *key* 和 *others* 信息，初始化该顶点的邻接表为空。
  - 逐个读取邻接点索引，直到遇到 -1 为止：
    - 为每个邻接点申请新的邻接表结点 *ArcNode*。
    - 使用尾插法将邻接点结点接到该顶点的邻接表末尾。
- 关闭文件并返回 *OK*。

（5）时间复杂度：$\mathcal{O}(n+e)$，其中 n 为顶点数，e 为边数。每个顶点及邻接点仅被读取和插入一次。

（6）空间复杂度：$\mathcal{O}(e)$，主要为邻接表的存储空间，每条边在邻接表中均占用一个结点。

15. int ShortestPathLength(ALGraph G, KeyType v, KeyType k)

（1）输入：图结构体 *G*，表示待查找的无向图；起点顶点关键字 *v* 和终点顶点关键字 *k*。

（2）输出：若存在从 v 到 k 的最短路径，返回该最短路径长度；若不存在路径或输入无效，返回 *ERROR*。

（3）算法思想描述：本函数采用广度优先搜索（*BFS*）思想，计算从起点 v 到终点 k 的最短路径长度。首先定位起点和终点在图中顶点数组的索引；再从起点 v 出发，使用队列实现 BFS，逐层遍历邻接点并更新最短距离数组 *distance*[]。若在遍历过程中访问到终点 k，则当前的距离即为最短路径长度；否则表示不存在路径。

（4）算法处理步骤：

- 通过 *LocateVertex* 函数查找起点 v 和终点 k 在邻接表中的索引，若任意一个不存在则返回 *ERROR*。
- 初始化数组：*visited*[$G.vexnum$]、*distance*[$G.vexnum$]，并创建队列 q。
- 将起点索引 *n* 入队，并初始化 *distance*[*n*] = *0*。
- 当队列非空时：
  - 取出队首元素 u，遍历其所有邻接点。
  - 对每个未访问的邻接点 w：

* 标记为已访问，更新最短距离 $distance[w] = distance[u] + 1$。
* 入队 w。

- 遍历结束后，若 $visited[m] = 1$，则返回 $distance[m]$；否则返回 $ERROR$。

（5）时间复杂度：$\mathcal{O}(n+e)$，其中 n 为顶点数，e 为边数。（6）空间复杂度：$\mathcal{O}(n)$，主要为 $visited[G.vexnum]$、$distance[G.vexnum]$。

下面是这个函数的流程图：

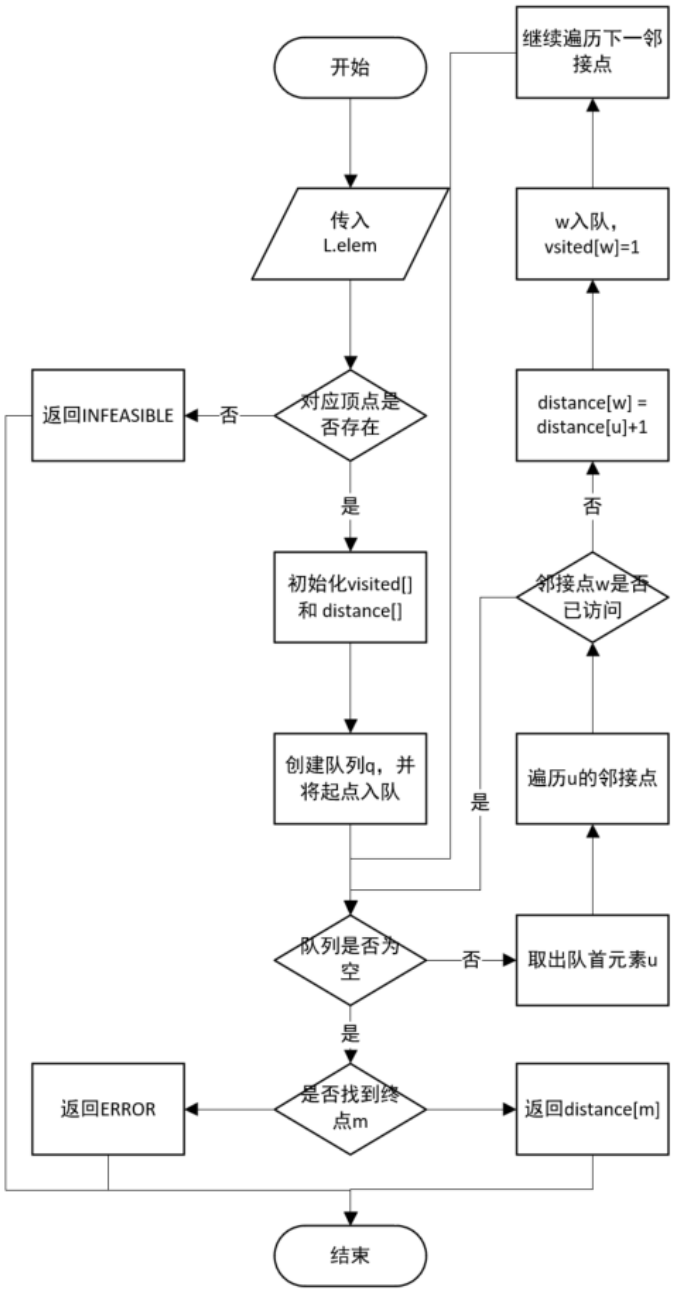

图 2-1　ShortestPathLength 函数流程图

16. void VerticesWithinDistanceK(ALGraph G, KeyType v, KeyType K)

（1）输入：图结构体 *G*，表示待查询的无向图；起点顶点关键字 *v*；距离阈值 *K*，表示需要查找的距离上限。

（2）输出：在图 *G* 中，输出与顶点 v 距离 $< K$ 且不为 v 的所有顶点的关键字和附加信息；若无符合条件的顶点，输出提示信息。

（3）算法思想描述：该函数基于广度优先搜索（*BFS*）的思想。首先进行一系列输入合法性检查：若 $K < 0$、起点不存在、图为空或无边，直接输出提示信息并返回；否则，从起点 v 出发，使用 *BFS* 遍历全图，统计所有顶点到 v 的最短距离，最后遍历并输出距离 $< K$ 且不为起点的所有顶点。通过一次 *BFS*，避免了重复调用最短路径函数。

（4）算法处理步骤：

- 先进行合法性检查，若不满足条件则返回。
- 初始化距离数组 *distance*[*MAX_VERTEX_NUM*] 为 -1，表示未访问。
- 将 *start* 入队列 q，并设置 *distance*[*start*] $= 0$。
- 当队列非空时：
  - 取出队首元素 u，遍历 u 的邻接点。
  - 若邻接点 w 未被访问，则更新 *distance*[*w*] $=$ *distance*[*u*] $+ 1$ 并将 w 入队。
- 遍历 *distance* 数组：
  - 若 $i \neq start$ 且 *distance*[*i*] $\neq -1$ 且 *distance*[*i*] $< K$，则输出该顶点的关键字和附加信息。
- 若最终没有任何顶点满足条件，则输出“没有满足条件的顶点”。

（5）时间复杂度：$\mathcal{O}(n+e)$，其中 n 为顶点数，e 为边数。所有顶点最多被访问一次，所有边最多被遍历一次。

（6）空间复杂度：$\mathcal{O}(n)$，主要为 *distance* 数组和辅助队列 q。

本函数的流程图如下：

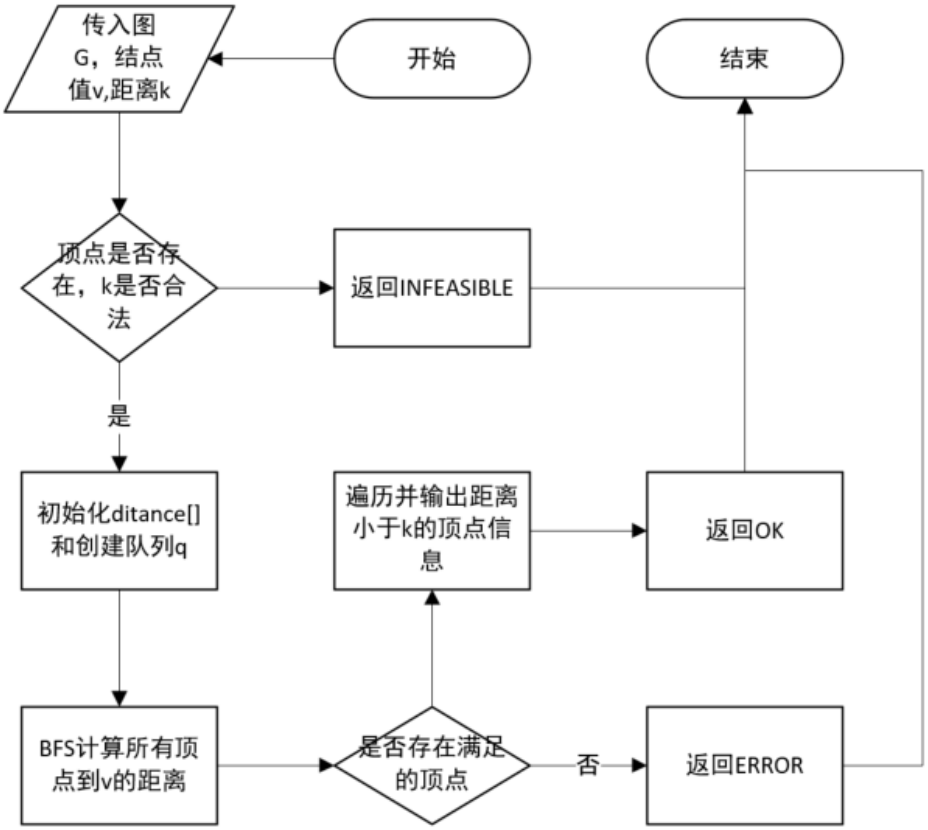

图 2-2　VerticesWithinDistanceK 函数流程图

17. int ConnectedComponentsNum(ALGraph G)

（1）输入：图结构体 *G*，表示一个无向图。

（2）输出：返回 *G* 中的连通分量个数。

（3）算法思想描述：该函数利用广度优先搜索（*BFS*）遍历图中的各个连通分量，统计总数。依次遍历每个顶点，若未访问，则该顶点所在的连通分量尚未计数。以该顶点为起点，执行一次 *BFS*，将该分量内的所有顶点标记为已访问。每次 *BFS* 结束后，连通分量计数器 *cnt* 自增 1。

（4）算法处理步骤：

- 初始化访问数组 *visited*[*G.vexnum*] 为全 0，连通分量计数 $cnt = 0$。
- 从 $i = 0$ 到 $G.vexnum - 1$ 遍历所有顶点：
  - 若 *visited*[*i*] = *0*，说明该顶点属于一个尚未计数的连通分量：
    - 将 *cnt* 加 1，并将 i 入队，标记 *visited*[*i*] = *1*。
    - 使用 *BFS* 遍历该连通分量内所有顶点，将它们都标记为已访问。

– 若 *visited*[*i*] = *1*，说明该顶点已被计数，继续遍历下一个顶点。

• 遍历结束，返回连通分量个数 *cnt*。

（5）时间复杂度：$\mathcal{O}(n+e)$，其中 n 为顶点数，e 为边数。每个顶点和边只被遍历一次。

（6）空间复杂度：$\mathcal{O}(n)$，主要用于访问数组和辅助队列。

本函数的流程图如下：

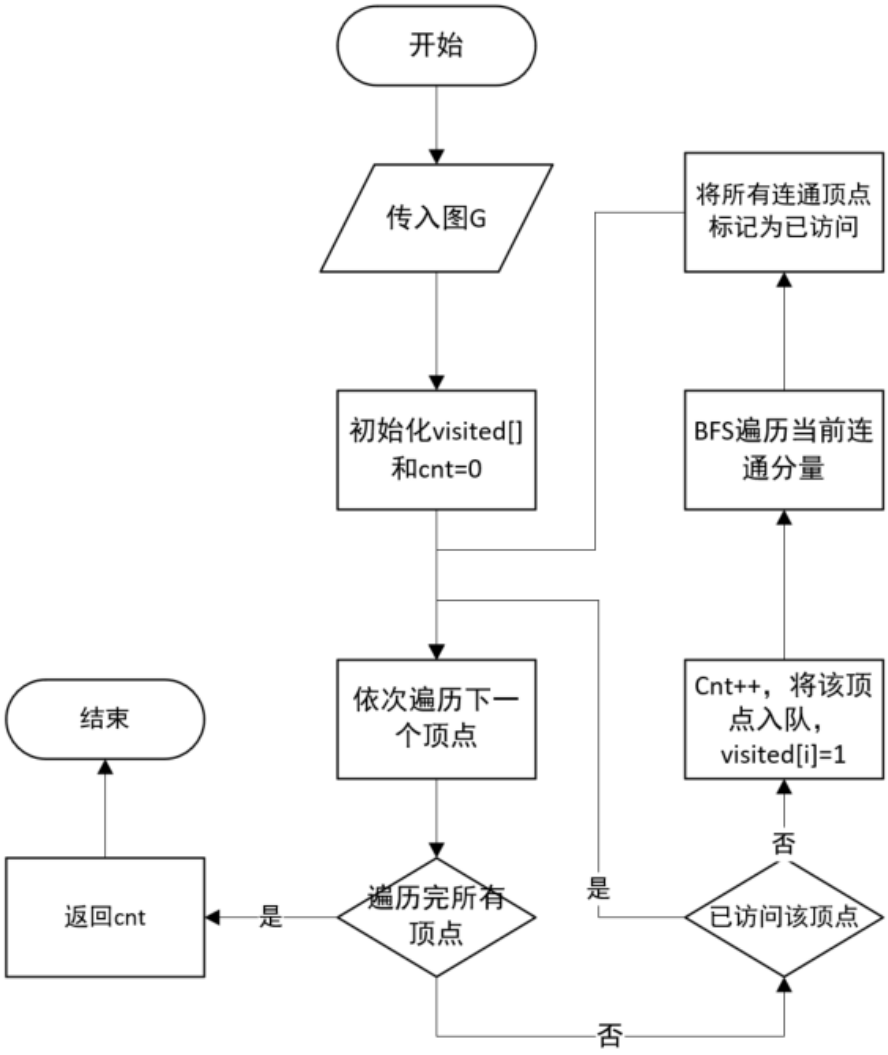

图 2-3　ConnectedComponentsNum 函数流程图

2.3.2　多图操作函数实现

下面是多图操作函数的实现思路：

*18. status AddGraph(GRAPHS &Graphs, char *Graphname)*

（1）输入：多图结构体引用 *Graphs*，表示存储多图的数组；*Graphname*，表示新图的名称。

（2）输出：函数执行状态，若添加成功返回 *OK*，若图名已存在或内存分配失败则返回 *ERROR*。

（3）算法思想描述：本函数用于向多图数组中添加一个新图。首先检查图的数量是否已达到最大限制，若是则返回错误。然后检查图名是否已存在，若存在则返回错误。最后动态分配内存创建新图，并将其添加到多图数组中。

（4）算法处理步骤：

- 检查多图数组的大小是否已达到最大限制，若是则返回 *ERROR*。
- 检查图名是否已存在，若存在则返回 *ERROR*。
- 创建新图的结构体，初始化顶点数、边数为 0。
- 复制 *Graphname* 到新图的名称。
- 将新图添加到多图数组中，*Graphs.length* 增加 1。
- 把新图作为当前操作的图，修改 $Graphs.cur\_index = Graphs.length - 1$。
- 返回 *OK*。

（5）时间复杂度：$\mathcal{O}(1)$，添加操作只涉及常数时间的内存分配和指针操作。

（6）空间复杂度：$\mathcal{O}(1)$，主要用于常数空间的指针和变量。

*19. status DeleteGraph(GRAPHS &Graphs, char *Graphname)*

（1）输入：多图结构体引用 *Graphs*，表示存储多图的数组；*Graphname*，表示删除的图名称。

（2）输出：函数执行状态，若删除成功返回 *OK*，若图名不存在或内存释放失败则返回 *ERROR*。

（3）算法思想描述：本函数用于从多图数组中删除指定名称的图。首先检查图名是否存在，若不存在则返回错误。然后释放该图的邻接表和顶点数组内存，最后将该图从多图数组中移除。

（4）算法处理步骤：

- 调用 *LocateGraph* 函数查找图名对应的 *index*，若不存在则返回 *ERROR*。

- 调用 *DestroyGraph* 函数释放该图的邻接表。
- $memset$(*Graphs.Graphs*[*index*]，0，sizeof(*ALGraph*))，清空该图的结构体。
- 更新数组长度 *Graphs.length*。
- 将 *Graphs.Graphs*[*index*] 后面的图向前移动。
- 返回 *OK*。

（5）时间复杂度：$\mathcal{O}(1)$，删除操作只涉及常数时间的内存释放和指针操作。

（6）空间复杂度：$\mathcal{O}(1)$，主要用于常数空间的指针和变量。

*20. status SwitchGraph(GRAPHS &Graphs, char *Graphname)*

（1）输入：多图结构体引用 *Graphs*，表示存储多图的数组；*Graphname*，表示切换的图名称。

（2）输出：函数执行状态，若切换成功返回 *OK*，若图名不存在则返回 *ERROR*。

（3）算法思想描述：本函数用于切换当前操作的图。首先检查图名是否存在，若不存在则返回错误。然后更新当前图的索引，使后续操作作用于新的图。

（4）算法处理步骤：

- 调用 *LocateGraph* 函数查找图名对应的 *index*，若不存在则返回 *ERROR*。
- 更新当前图的索引 *Graphs.cur_index* = *index*。
- 返回 *OK*。

（5）时间复杂度：$\mathcal{O}(1)$，切换操作只涉及常数时间的指针操作。

（6）空间复杂度：$\mathcal{O}(1)$，主要用于常数空间的指针和变量。

21. status PrintGraphs(GRAPHS &Graphs)

（1）输入：多图结构体引用 *Graphs*，表示存储多图的数组。

（2）输出：函数执行状态，始终返回 *OK*。

（3）算法思想描述：本函数用于打印多图数组中所有图的名称和基本信息。遍历多图数组，依次输出每个图的名称。

（4）算法处理步骤：遍历多图数组 *Graphs.Graphs*，输出每个图的名称和基本信息。

（5）时间复杂度：$\mathcal{O}(n)$，其中 n 为多图数组的长度。

（6）空间复杂度：$\mathcal{O}(1)$，主要用于常数空间的指针和变量。

22. status PrintGraphWithAdjList(GRAPHS &Graphs)

（1）输入：多图结构体引用 *Graphs*，表示存储多图的数组。

（2）输出：函数执行状态，始终返回 *OK*。

（3）算法思想描述：本函数用于打印当前操作的图的邻接表结构。遍历邻接表，输出每个顶点及其邻接点的信息。

（4）算法处理步骤：

- 输出当前图的名称。
- 输出当前图的顶点数和边数。
- 遍历邻接表：
  - 对每个顶点 i，输出其关键字和附加信息。
  - 遍历该顶点的邻接链表，输出所有邻接点的信息。
- 返回 *OK*。

（5）时间复杂度：$\mathcal{O}(n+e)$，其中 n 为顶点数，e 为边数。每个顶点和边均被遍历一次。

（6）空间复杂度：$\mathcal{O}(1)$，主要用于常数空间的指针和变量。

至此，整个系统的设计和每一个函数的实现思路都已经给出，下面给出系统的测试情况。

2.4 系统测试

本部分主要说明针对各个函数正常和异常的测试用例及测试结果：

首先给出进入系统的菜单界面，用户需要先为第一个初始的图命名，然后进入系统菜单界面。系统菜单界面如下图所示：

图 2-4　系统菜单界面

1. *CreateGraph* 函数测试

表 2-1　NextElem 测试数据

测试用例	程序输入	理论结果
不存在图	1	图创建成功
存在图	1	已存在图，创建失败！

测试结果：

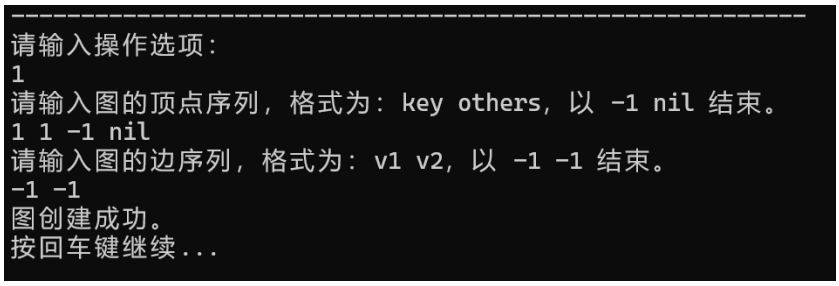

图 2-5　CreateGraph 函数测试 1 结果

图 2-6　CreateGraph 函数测试 2 结果

2. *DestroyGraph* 函数测试

表 2-2　DestroyGraph 测试数据

测试用例	程序输入	理论结果
不存在图	2	图不存在，删除失败！
存在图	2	图删除成功！

测试结果：

图 2-7　DestroyGraph 函数测试 1 结果

图 2-8　DestroyGraph 函数测试 2 结果

3. *LocateVertex* 函数测试

表 2-3　LocateVertex 测试数据

测试用例	程序输入	理论结果
图不存在	3	图不存在，查找失败！
图存在且该顶点存在	3	返回该顶点的索引
图存在且该顶点不存在	3	顶点不存在

测试结果：

图 2-9　LocateVertex 函数测试 1 结果

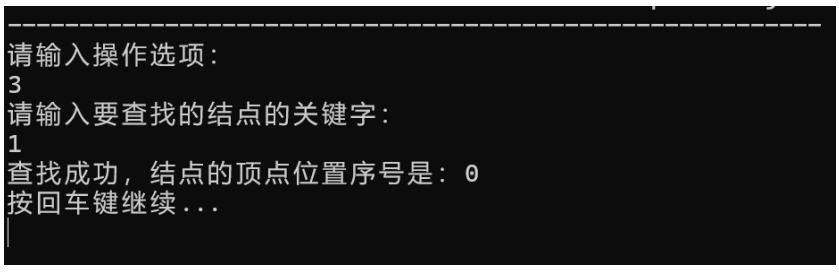

图 2-10　LocateVertex 函数测试 2 结果

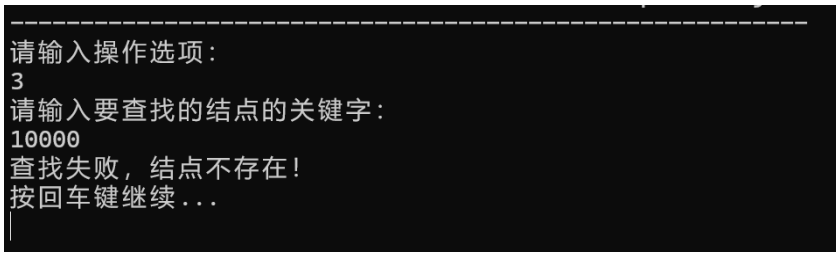

图 2-11　LocateVertex 函数测试 3 结果

4. *PutVex* 函数测试

表 2-4　PutVex 测试数据

测试用例	程序输入	理论结果
图不存在	4	图不存在，修改失败!
图存在且该顶点存在	4	修改成功!
图存在且该顶点不存在	4	顶点不存在，修改失败!

测试结果：

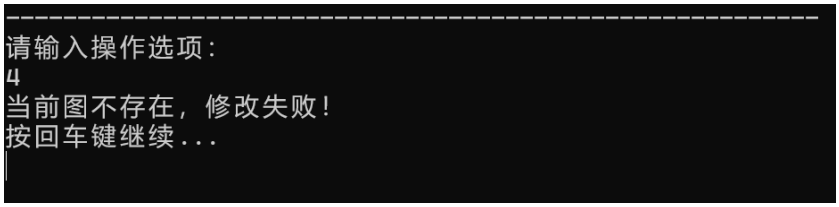

图 2-12　PutVex 函数测试 1 结果

图 2-13　PutVex 函数测试 2 结果

图 2-14　PutVex 函数测试 3 结果

5. *FirstAdjVex* 函数测试

表 2-5　FirstAdjVex 测试数据

测试用例	程序输入	理论结果
图不存在	5	图不存在，查找失败！
图存在，顶点存在	5	返回该点第一个邻接点的索引
图存在，顶点不存在	5	顶点不存在，查找失败！

测试结果：

图 2-15　FirstAdjVex 函数测试 1 结果

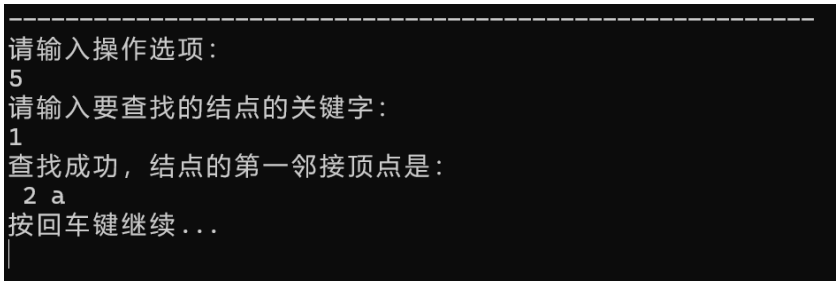

图 2-16　FirstAdjVex 函数测试 2 结果

图 2-17　FirstAdjVex 函数测试 3 结果

6. *NextAdjVex* 函数测试

表 2-6　NextAdjVex 测试数据

测试用例	程序输入	理论结果
图不存在	6	图不存在，查找失败!
正常情况	6	返回下一个邻接点的索引
顶点不存在	6	顶点不存在，查找失败!
w 是 v 的最后一个邻接点	6	w 无下一个邻接点

测试结果：

图 2-18　NextAdjVex 函数测试 1 结果

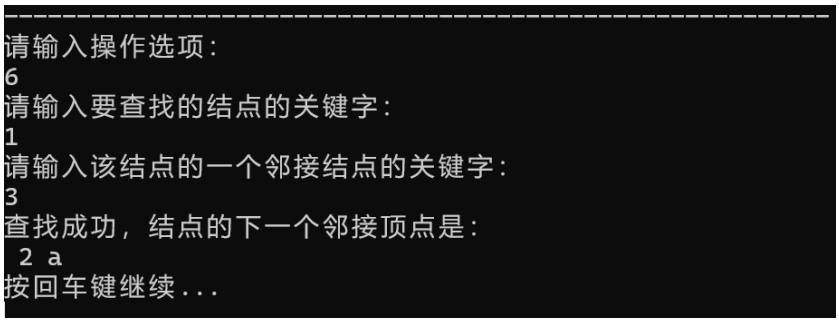

图 2-19　NextAdjVex 函数测试 2 结果

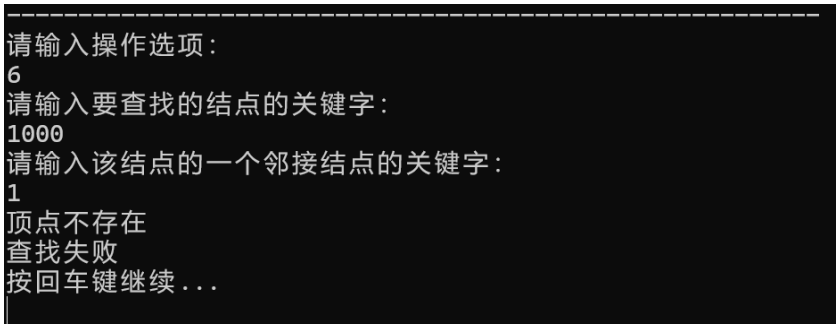

图 2-20　NextAdjVex 函数测试 3 结果

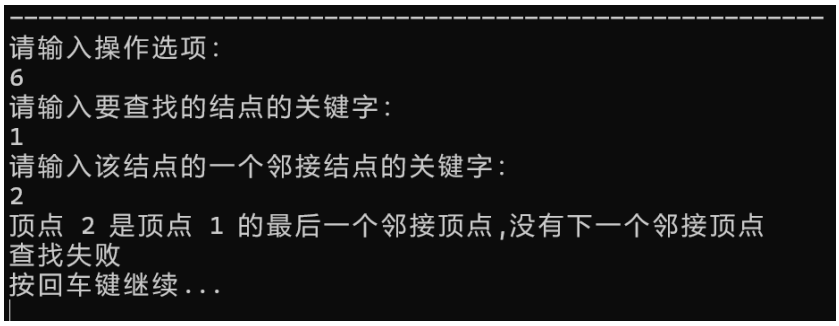

图 2-21　NextAdjVex 函数测试 4 结果

7. *InsertVex* 函数测试

表 2-7　InsertVex 测试数据

测试用例	程序输入	理论结果
图不存在	7	图不存在，插入失败！
顶点已满	7	顶点已满，插入失败！
顶点已存在	7	顶点已存在，插入失败！
正常情况	7	插入成功！

测试结果：

```
--------------------------------------------------------------
请输入操作选项:
7
当前图不存在，插入顶点失败!
按回车键继续...
```

图 2-22　InsertVex 函数测试 1 结果

```
--------------------------------------------------------------
请输入操作选项:
7
请输入要插入的结点的key和others:
21 aaa
图的顶点数已达上限，无法插入新顶点
插入失败
按回车键继续...
```

图 2-23　InsertVex 函数测试 2 结果

```
--------------------------------------------------------------
请输入操作选项:
7
请输入要插入的结点的key和others:
19 aaa
图中已存在该顶点，无法插入
插入失败
按回车键继续...
```

图 2-24　InsertVex 函数测试 3 结果

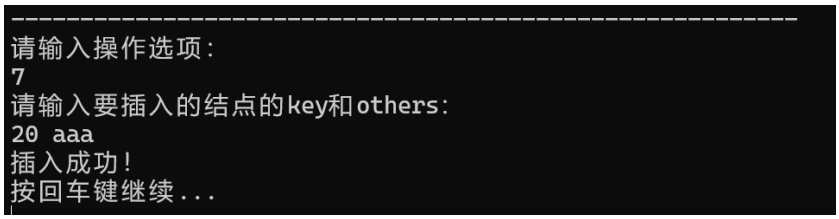

图 2-25　InsertVex 函数测试 4 结果

8. *DeleteVex* 函数测试

表 2-8　DeleteVex 测试数据

测试用例	程序输入	理论结果
图不存在	8	图不存在，删除失败！
顶点不存在	8	顶点不存在，删除失败！
只有一个顶点	8	不可再删除！
正常情况	8	删除成功！

测试结果：

图 2-26　DeleteVex 函数测试 1 结果

图 2-27　DeleteVex 函数测试 2 结果

图 2-28　DeleteVex 函数测试 3 结果

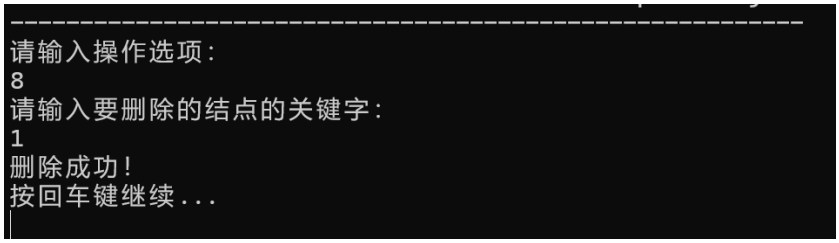

图 2-29　DeleteVex 函数测试 4 结果

9. *InsertArc* 函数测试

表 2-9　InsertArc 测试数据

测试用例	程序输入	理论结果
图不存在	9	图不存在，插入失败！
顶点不存在	9	顶点不存在，插入失败！
弧已存在	9	弧已存在，插入失败！
正常情况	9	插入成功！

测试结果：

图 2-30　InsertArc 函数测试 1 结果

图 2-31　InsertArc 函数测试 2 结果

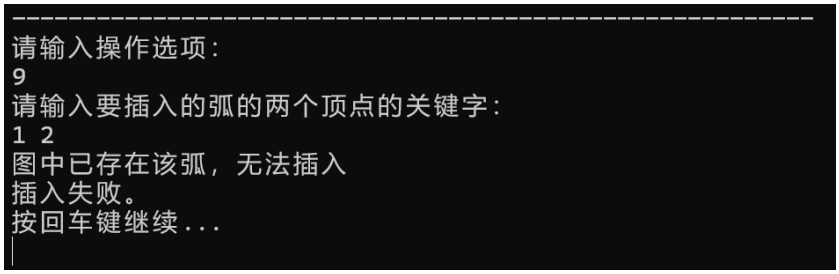

图 2-32　InsertArc 函数测试 3 结果

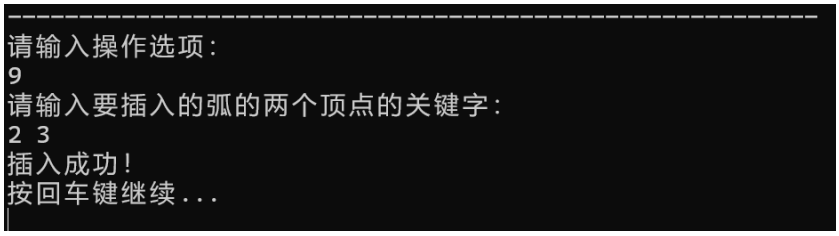

图 2-33　InsertArc 函数测试 4 结果

10. *DeleteArc* 函数测试

表 2-10　DeleteArc 测试数据

测试用例	程序输入	理论结果
图不存在	10	图不存在，删除失败！
顶点不存在	10	顶点不存在，删除失败！
弧不存在	10	弧不存在，删除失败！
正常情况	10	删除成功！

测试结果：

图 2-34　DeleteArc 函数测试 1 结果

图 2-35　DeleteArc 函数测试 2 结果

图 2-36　DeleteArc 函数测试 3 结果

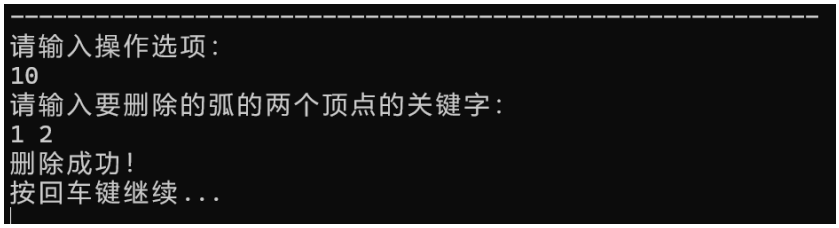

图 2-37　DeleteArc 函数测试 4 结果

11. *DFSTraverse* 函数测试

表 2-11　DFSTraverse 测试数据

测试用例	程序输入	理论结果
图不存在	11	图不存在，遍历失败！
正常情况	11	遍历成功！

测试结果：

```
--------------------------------------------------------
请输入操作选项:
11
当前图不存在，遍历失败!
按回车键继续...
```

图 2-38　DFSTraverse 函数测试 1 结果

```
--------------------------------------------------------
请输入操作选项:
11
深度优先遍历结果:
1 a
4 c
2 b
3 d
```

图 2-39　DFSTraverse 函数测试 2 结果

12. *BFSTraverse* 函数测试

表 2-12　BFSTraverse 测试数据

测试用例	程序输入	理论结果
图不存在	12	图不存在，遍历失败！
正常情况	12	遍历成功！

测试结果：

图 2-40　BFSTraverse 函数测试 1 结果

图 2-41　BFSTraverse 函数测试 2 结果

13. *SaveGraph* 函数测试

表 2-13　SaveGraph 测试数据

测试用例	程序输入	理论结果
图不存在	13	图不存在，保存失败!
正常情况	13	保存成功!

测试结果：

图 2-42　SaveGraph 函数测试 1 结果

图 2-43　SaveGraph 函数测试 2 结果

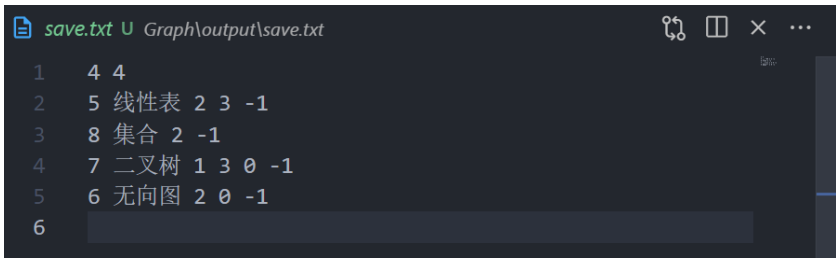

图 2-44　SaveGraph 函数测试 2 结果

14. *LoadGraph* 函数测试

表 2-14　LoadGraph 测试数据

测试用例	程序输入	理论结果
图已经存在	14	加载失败！
正常情况	14	加载成功！

测试结果：

图 2-45　LoadGraph 函数测试 1 结果

图 2-46　LoadGraph 函数测试 2 结果

15. *ShortestPathLength* 函数测试

表 2-15　ShortestPathLength 测试数据

测试用例	程序输入	理论结果
图不存在	15	图不存在，查找失败！
顶点不存在	15	顶点不存在，查找失败！
正常情况	15	输出两点最短路径距离

测试结果：

图 2-47　ShortestPathLength 函数测试 1 结果

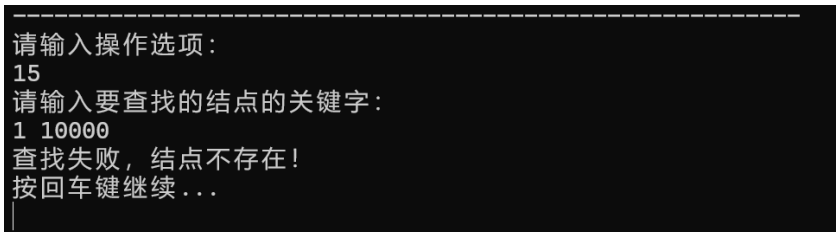

图 2-48　ShortestPathLength 函数测试 2 结果

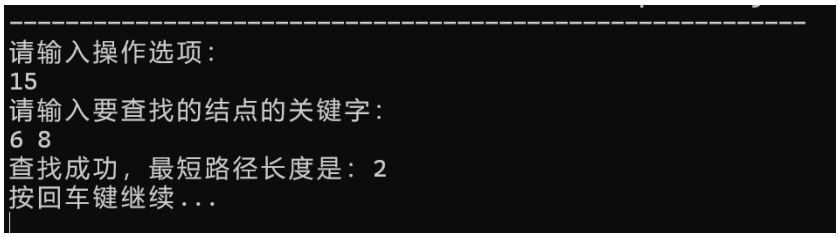

图 2-49　ShortestPathLength 函数测试 3 结果

16. *VerticesWithinDistanceK* 函数测试

表 2-16　VerticesWithinDistanceK 测试数据

测试用例	程序输入	理论结果
图不存在	16	图不存在，查找失败！
顶点不存在	16	顶点不存在，查找失败！
正常情况	16	输出符合条件顶点信息

测试结果：

图 2-50　VerticesWithinDistanceK 函数测试 1 结果

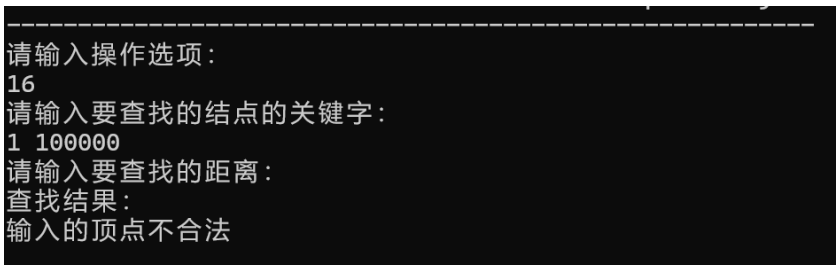

图 2-51　VerticesWithinDistanceK 函数测试 2 结果

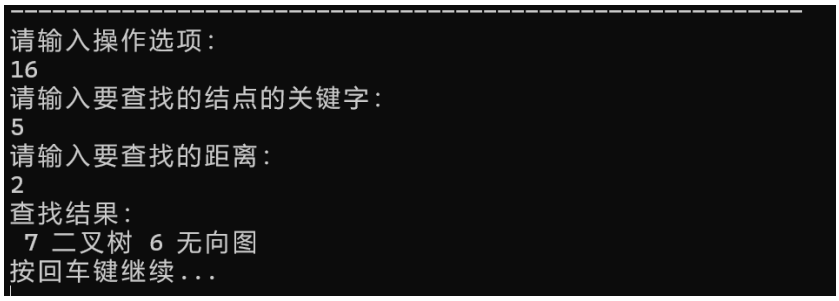

图 2-52　VerticesWithinDistanceK 函数测试 3 结果

17. *ConnectedComponentsNum* 函数测试

表 2-17　ConnectedComponentsNum 测试数据

测试用例	程序输入	理论结果
图不存在	17	图不存在，查找失败！
正常情况	17	输出连通分量数

测试结果：

图 2-53　ConnectedComponentsNum 函数测试 1 结果

图 2-54　ConnectedComponentsNum 函数测试 2 结果

18. *AddGraph* 函数测试

表 2-18　AddGraph 测试数据

测试用例	程序输入	理论结果
图名称重复	18	名称重复，添加失败！
正常情况	18	添加成功！

测试结果：

图 2-55　AddGraph 函数测试 1 结果

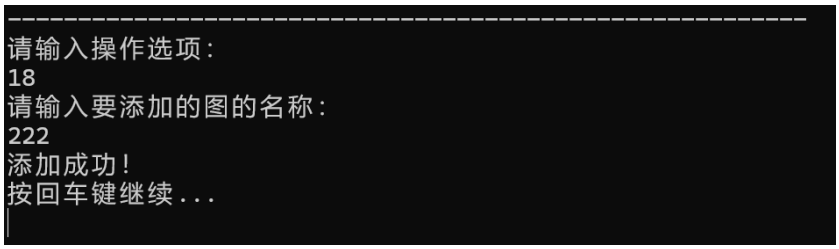

图 2-56　AddGraph 函数测试 2 结果

19. *RemoveGraph* 函数测试

表 2-19　RemoveGraph 测试数据

测试用例	程序输入	理论结果
图不存在	19	图不存在，删除失败！
正常情况	19	删除成功！

测试结果：

图 2-57　RemoveGraph 函数测试 1 结果

图 2-58　RemoveGraph 函数测试 2 结果

20. *SwitchGraph* 函数测试

表 2-20　SwitchGraph 测试数据

测试用例	程序输入	理论结果
图不存在	20	图不存在，切换失败！
正常情况	20	切换成功！

测试结果：

图 2-59　SwitchGraph 函数测试 1 结果

图 2-60　SwitchGraph 函数测试 2 结果

21. *PrintGraphs* 函数测试

输出当前图列表中所有图的信息。

图 2-61　PrintGraphs 函数测试结果

22. *PrintGraphWithAdjList* 函数测试

表 2-21　PrintGraphWithAdjList 测试数据

测试用例	程序输入	理论结果
图不存在	22	图不存在，打印失败！
正常情况	22	打印成功！

测试结果：

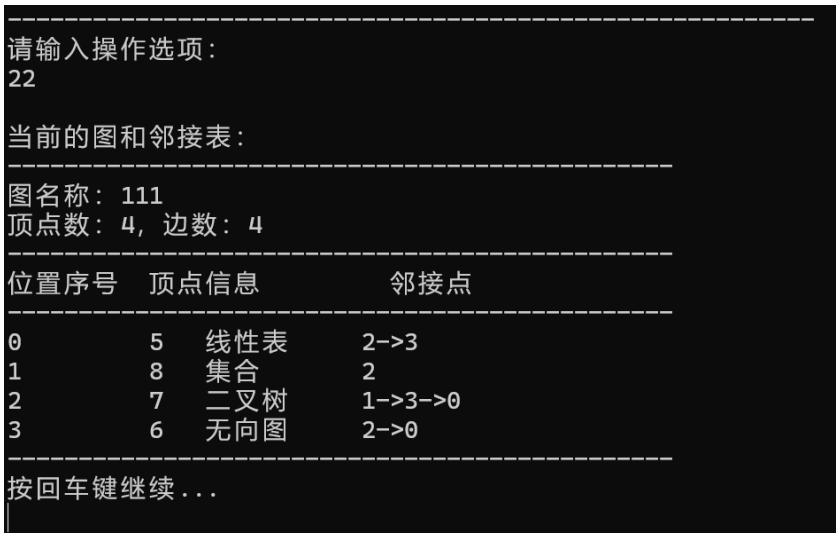

图 2-62　PrintGraphWithAdjList 函数测试结果

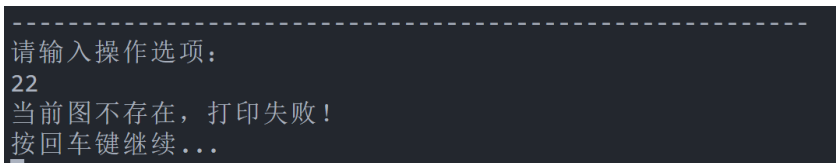

图 2-63　PrintGraphWithAdjList 函数测试结果

至此，所有的测试用例都已经完成，每个函数都进行了多种情况的测试，包括一些边界情况的测试，测试结果都符合预期，说明程序的功能实现是正确的。

2.5　实验小结

本次实验实现了一个基于邻接表的图的存储结构，并实现了对图的基本操作，比如插入、删除、遍历等操作，还有一些附加的功能比如求最短路径、连通分量数，以及怎么文件读写图和图的可视化等功能。通过本次数据结构实验，我在很多方面都有很大收获。

- 图的理解加深　通过实验，我对图的数据结构定义有了更深刻的理解，深刻理解了怎么用邻接表存储图，以及邻接表的各种功能实现。
- 编程思维提升　通过本次实验，我的编程思维有了很大提升，深刻理解了数据结构的设计和实现，比如从顶点的结构体设计到邻接表的设计，再到图

的设计，最后是多图的设计，这样环环相扣的设计和封装的思想，让我在编程思维上有了很大提升。

- **编程能力提升**　通过本次实验，我对图的很多算法实现有了更深刻的理解，比如深度优先遍历、广度优先遍历、最短路径算法，怎么求连通分量数等，还通过 ai 和洛谷等顺带学习了一些其他图的算法，提升了我的编程能力。

3 实验总结

好，写到这里终于可以长舒一口气，截图和排版得人都麻木了，数据结构实验的内容终于完成。四次实验，前两次实验是线性表和链表的实现，第三次实验是树的实现，最后一次实验是图的实现。

从最开始的头哥实现每个关卡，再到自己组装起来每个系统，并不断试错、完善、优化，最后是 LaTeX 报告的编写，整个过程前后跨度达到了一两个月。期间经历了很多的波折，很多的 bug，很多的错误，很多的挫折，有时候忙活一晚上也没什么很大的进展。

但是回过头来看，这期间的收获和进步也是巨大的。从最开始摸索怎么在 vscode 上配置 LaTeX 的环境，慢慢学习 LaTeX 的语法还有排版，学习一些 *Cpp* 的语法和常用 *STL*，再到学习把自己的数据结构实验的代码封装打包，上传到 *GitHub* 上托管等等，都是我之前没有接触过的东西。虽然有时候会觉得很麻烦，但是回过头来看，这些都是我在数据结构实验中收获的宝贵经验。

本实验托管在 *GitHub* 上，地址为：

`https://github.com/Cupid-qrq/Data-Structure-Experiment`。

2025.5.25

参考文献

[1] PRATA S. C Primer Plus[M]. Addison-Wesley, 2014, 6th edition .

[2] 袁玲, 等. 数据结构——从概念到算法（C 语言版 | 微课版）[M]. 北京：人民邮电出版社, 2023, .

[3] PRATA S. C++ Primer Plus（第 6 版）[M]. 北京：人民邮电出版社, 2020.

[4] Nyhoff L. ADTs, Data Structures, and Problem Solving with C++. [M]. Calvin College, 2005.

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        |
 |  Author:qrq 2402                           |
 |  Date:2025.4.7                             |
 --------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unordered_map>
#include <algorithm>

typedef int status;
typedef int ElemType;

#define OK 1
#define ERROR 0
#define TRUE 1
#define FALSE 0
#define INFEASIBLE -1
#define OVERFLOW -2

#define LIST_INIT_SIZE 100
#define LISTINCREMENT 10
#define MAXSIZE 11
using namespace std;

typedef struct sqlist
{
    ElemType *elem;
    int length;
```

```
    int listsize;
    char name[30];
} SqList;

typedef struct
{
    SqList elem[MAXSIZE];
    int length;
    int cur_index;
} LISTS;

/*--------------------------------------------------*/

status InitList(SqList &L);
status DestroyList(SqList &L);
status ClearList(SqList &L);
status ListEmpty(SqList &L);
int ListLength(SqList &L);
status GetElem(SqList &L, int i, ElemType &e);
int LocateElem(SqList &L, ElemType e);
status PriorElem(SqList &L, ElemType cur, ElemType &pre_e);
status NextElem(SqList &L, ElemType cur, ElemType &next_e);
status ListInsert(SqList &L, int i, ElemType &e);
status ListDelete(SqList &L, int i, ElemType &e);
status ListTraverse(SqList &L);

int MaxSubArray(SqList &L);
int SubArrayNum(SqList &L, int k);
status SortList(SqList &L);
status SaveList(SqList L, char FileName[]);
status LoadList(SqList &L, char FileName[]);

int LocateList(LISTS Lists, char ListName[]);
status AddList(LISTS &Lists, char ListName[]);
status RemoveList(LISTS &Lists, char ListName[]);
```

```
status SwitchList(LISTS &Lists, char ListName[]);
void PrintLists(LISTS Lists);
void printMenu(LISTS &Lists);

/*-----------------------------------------------*/

status InitList(SqList &L)
{
    if (L.elem != NULL)
        return INFEASIBLE;

    L.elem = (ElemType *)malloc(sizeof(ElemType) * LIST_INIT_SIZE);
    if (L.elem == NULL)
        return OVERFLOW;

    L.length = 0;
    L.listsize = LIST_INIT_SIZE;

    return OK;
}

status DestroyList(SqList &L)
{
    if (L.elem == NULL)
        return INFEASIBLE;

    free(L.elem);
    L.elem = NULL;
    L.length = 0;
    L.listsize = 0;

    return OK;
}

status ClearList(SqList &L)
```

```
{
    if (L.elem == NULL)
        return INFEASIBLE;

    L.length = 0;

    return OK;
}

status ListEmpty(SqList &L)
{
    if (L.elem == NULL)
        return INFEASIBLE;

    return L.length == 0 ? TRUE : FALSE;
}

int ListLength(SqList &L)
{
    if (L.elem == NULL)
        return INFEASIBLE;

    return L.length;
}

status GetElem(SqList &L, int i, ElemType &e)
{
    if (L.elem == NULL)
        return INFEASIBLE;

    if (i < 1 || i > L.length)
        return ERROR;

    e = L.elem[i - 1];
    return OK;
```

```
}

int LocateElem(SqList &L, ElemType e)
{
    if (L.elem == NULL)
        return INFEASIBLE;

    for (int i = 0; i < L.length; i++)
    {
        if (e == L.elem[i])
            return i + 1;
    }

    return ERROR;
}

status PriorElem(SqList &L, ElemType e, ElemType &pre)
{
    if (L.elem == NULL)
        return INFEASIBLE;

    for (int i = 0; i < L.length; i++)
    {
        if (e == L.elem[i] && i == 0)
        {
            printf(" 该元素为首元素，无前驱元素！\n");
            return ERROR;
        }

        if (e == L.elem[i] && i != 0)
        {
            pre = L.elem[i - 1];
            return OK;
        }
    }
```

```
    return ERROR;
}

status NextElem(SqList &L, ElemType e, ElemType &next)
{
    if (L.elem == NULL)
        return INFEASIBLE;

    for (int i = 0; i < L.length; i++)
    {
        if (e == L.elem[i] && i != L.length - 1)
        {
            next = L.elem[i + 1];
            return OK;
        }
    }

    return ERROR;
}

status ListInsert(SqList &L, int i, ElemType &e)
{
    if (L.elem == NULL)
        return INFEASIBLE;

    if (i < 1 || i > L.length + 1)
        return ERROR;

    if (L.length >= L.listsize)
    {
        ElemType *Newlist = (ElemType *)realloc(L.elem, (L.listsize +
        LISTINCREMENT) * sizeof(ElemType));
        if (Newlist == NULL)
            return OVERFLOW;
```

```
        L.elem = Newlist;
        L.listsize += LISTINCREMENT;
    }

    for (int p = L.length - 1; p >= i - 1; p--)
    {
        L.elem[p + 1] = L.elem[p];
    }
    L.elem[i - 1] = e;
    L.length++;

    return OK;
}

status ListDelete(SqList &L, int i, ElemType &e)
{
    if (L.elem == NULL)
        return INFEASIBLE;

    if (i < 1 || i > L.length)
        return ERROR;

    e = L.elem[i - 1];
    for (int p = i - 1; p < L.length - 1; p++)
    {
        L.elem[p] = L.elem[p + 1];
    }
    L.length--;

    return OK;
}

status ListTraverse(SqList &L)
{
```

```
    if (L.elem == NULL)
        return INFEASIBLE;

    for (int i = 0; i < L.length; i++)
    {
        printf("%d", L.elem[i]);
        if (i != L.length - 1)
            printf(" ");
    }
    printf("\n");

    return OK;
}

int MaxSubArray(SqList &L)
{
    if (L.elem == NULL)
        return INFEASIBLE;

    int currentMax = L.elem[0];
    int Max = L.elem[0];

    for (int i = 1; i < L.length; i++)
    {
        currentMax = max(L.elem[i], currentMax + L.elem[i]);
        Max = max(Max, currentMax);
    }

    return Max;
}

int SubArrayNum(SqList &L, int k)
{
    if (L.elem == NULL)
        return INFEASIBLE;
```

```
    unordered_map<int, int> prefixSum;
    prefixSum[0] = 1;
    int sum = 0, cnt = 0;

    for (int i = 0; i < L.length; i++)
    {
        sum += L.elem[i];
        cnt += prefixSum[sum - k];
        prefixSum[sum]++;
    }

    return cnt;
}

status SortList(SqList &L)
{
    if (L.elem == NULL)
        return INFEASIBLE;

    sort(L.elem, L.elem + L.length);
    return OK;
}

status SaveList(SqList L, char FileName[])
{
    if (L.elem == NULL)
        return INFEASIBLE;

    FILE *fp = fopen(FileName, "w");
    if (fp == NULL)
        return ERROR;

    for (int i = 0; i < L.length; i++)
    {
```

```
        fprintf(fp, "%d ", L.elem[i]);
    }
    fclose(fp);

    return OK;
}

status LoadList(SqList &L, char FileName[])
{
    if (L.elem != NULL)
        return INFEASIBLE;

    FILE *fp = fopen(FileName, "r");
    if (fp == NULL)
        return ERROR;

    L.elem = (ElemType *)malloc(sizeof(ElemType) * LIST_INIT_SIZE);
    if (L.elem == NULL)
    {
        fclose(fp);
        return OVERFLOW;
    }

    L.length = 0;
    L.listsize = LIST_INIT_SIZE;

    ElemType num;
    while (fscanf(fp, "%d", &num) == 1)
    {
        if (L.length >= L.listsize)
        {
            ElemType *Newlist = (ElemType *)realloc(L.elem, (L.listsize +
            LISTINCREMENT) * sizeof(ElemType));
            if (Newlist == NULL)
            {
```

```
                fclose(fp);
                return OVERFLOW;
            }
            L.elem = Newlist;
            L.listsize += LISTINCREMENT;
        }
        L.elem[L.length++] = num;
    }

    fclose(fp);

    return OK;
}

status AddList(LISTS &Lists, char ListName[])
{
    if (Lists.length >= MAXSIZE)
        return ERROR;

    for (int i = 0; i < Lists.length; i++)
    {
        if (strcmp(Lists.elem[i].name, ListName) == 0)
            return ERROR;
    }

    SqList &newList = Lists.elem[Lists.length];
    newList.elem = NULL;
    newList.length = 0;
    strcpy(newList.name, ListName);
    Lists.length++;

    Lists.cur_index = Lists.length - 1;
    return OK;
}
```

```
status RemoveList(LISTS &Lists, char ListName[])
{
    for (int i = 0; i < Lists.length; i++)
    {
        if (strcmp(Lists.elem[i].name, ListName) == 0)
        {
            free(Lists.elem[i].elem);
            for (int j = i; j < Lists.length - 1; j++)
            {
                Lists.elem[j] = Lists.elem[j + 1];
            }

            Lists.elem[Lists.length - 1].elem = NULL;
            Lists.elem[Lists.length - 1].length = 0;
            Lists.elem[Lists.length - 1].listsize = 0;
            memset(Lists.elem[Lists.length - 1].name, 0, 30);

            Lists.length--;

            Lists.cur_index = 0;
            return OK;
        }
    }
    return ERROR;
}

int LocateList(LISTS Lists, char ListName[])
{
    for (int i = 0; i < Lists.length; i++)
    {
        if (strcmp(Lists.elem[i].name, ListName) == 0)
        {
            return i + 1;
        }
    }
```

```
    return ERROR;
}

void PrintLists(LISTS Lists)
{
    printf(" 当前线性表列表: \n");
    for (int i = 0; i < Lists.length; i++)
    {
        printf("%d. %s\n", i + 1, Lists.elem[i].name);
    }
}

void printMenu(LISTS &Lists)
{
    printf("\n\n");
    printf("-------------------------------------------------\n");
    printf("   Menu for Linear Table On Sequence Structure \n");
    printf("-------------------------------------------------\n");
    printf(" 当前操作表为: %s\n", Lists.elem[Lists.cur_index].name);
    printf("        1.  InitList       11. ListDelete   \n");
    printf("        2.  DestroyList    12. ListTraverse \n");
    printf("        3.  ClearList      13. MaxSubArray  \n");
    printf("        4.  ListEmpty      14. SubArrayNum  \n");
    printf("        5.  ListLength     15. SortList     \n");
    printf("        6.  GetElem        16. SaveList     \n");
    printf("        7.  LocateElem     17. LoadList     \n");
    printf("        8.  PriorElem      18. AddList      \n");
    printf("        9.  NextElem       19. RemoveList   \n");
    printf("       10.  ListInsert     20. LocateList   \n");
    printf("       21.  SwitchList     22. PrintLists    \n");
    printf("        0.Exit    \n");
    printf("-------------------------------------------------\n");
    printf(" 请输入功能序号 [0~20]:");
}
```

```
int main()
{
    LISTS Lists;
    Lists.length = 1;
    Lists.cur_index = 0;
    Lists.elem[0].elem = NULL;
    char firstname[30];
    system("cls");
    printf("-----------------------------------------------\n");
    printf(" 这是一个多线性表系统，可以对单个或多个线性表进行操作\n");
    printf("Author: HUST Cupid-qrq\n");
    printf("-----------------------------------------------\n");
    printf(" 在开始之前，请先输入第一个线性表的名称：\n");
    if (scanf("%s", firstname) != 1)
    {
        printf(" 输入错误，请重新输入：\n");
        scanf("%s", firstname);
    }
    printf(" 输入成功，第一个线性表的名称是：%s\n", firstname);
    strcpy(Lists.elem[0].name, firstname);
    printf(" 接下来，进入菜单界面，按任意键继续：\n");
    getchar();
    getchar();

    int op = 1;
    int i, e, k;
    int pre_e, cur_e, next_e;
    int Maxsub;
    int cnt;
    char filename[30], listname[30];

    while (op)
    {
        printMenu(Lists);
```

```
scanf("%d", &op);

switch (op)
{
case 1:
    if (InitList(Lists.elem[Lists.cur_index]) == OK)
        printf(" 线性表初始化成功!\n");
    else
        printf(" 线性表已存在或初始化失败!\n");
    break;

case 2:
    if (DestroyList(Lists.elem[Lists.cur_index]) == OK)
        printf(" 表已销毁! \n");
    else
        printf(" 表不存在, 销毁失败! \n");
    break;

case 3:
    if (ClearList(Lists.elem[Lists.cur_index]) == OK)
        printf(" 表已清空! \n");
    else
        printf(" 清空失败, 表不存在! \n");
    break;

case 4:
    if (ListEmpty(Lists.elem[Lists.cur_index]) == TRUE)
        printf(" 表为空! \n");
    else if (ListEmpty(Lists.elem[Lists.cur_index]) == FALSE)
        printf(" 表非空! \n");
    else
        printf(" 表不存在! \n");
    break;

case 5:
```

```
        if (ListLength(Lists.elem[Lists.cur_index]) != INFEASIBLE)
            printf(" 表长度为: %d\n",
            ListLength(Lists.elem[Lists.cur_index]));
        else
            printf(" 表不存在! \n");
        break;

    case 6:
        printf(" 请输入要获取的元素位置: ");
        scanf("%d", &i);
        if (GetElem(Lists.elem[Lists.cur_index], i, e) == OK)
            printf(" 第 %d 个元素为: %d\n", i, e);
        else
            printf(" 位置非法或表不存在! \n");
        break;

    case 7:
        printf(" 请输入要查找的元素值: ");
        scanf("%d", &e);
        i = LocateElem(Lists.elem[Lists.cur_index], e);
        if (i != ERROR && i != INFEASIBLE)
            printf(" 元素 %d 的位置为: %d\n", e, i);
        else if (i == INFEASIBLE)
            printf(" 表不存在! \n");
        else
            printf(" 查找失败! \n");
        break;

    case 8:
        printf(" 请输入当前元素值: ");
        scanf("%d", &cur_e);
        if (PriorElem(Lists.elem[Lists.cur_index], cur_e, pre_e) ==
        OK)
            printf(" 前驱元素为: %d\n", pre_e);
        else
```

```
            printf(" 寻找前驱失败！\n");
        break;

    case 9:
        printf(" 请输入当前元素值：");
        scanf("%d", &cur_e);
        if (NextElem(Lists.elem[Lists.cur_index], cur_e, next_e) ==
        OK)
            printf(" 后继元素为：%d\n", next_e);
        else
            printf(" 无后继元素或表不存在！\n");
        break;

    case 10:
        printf(" 请输入插入位置和元素值：");
        scanf("%d%d", &i, &e);
        if (ListInsert(Lists.elem[Lists.cur_index], i, e) == OK)
            printf(" 插入成功！\n");
        else
            printf(" 插入失败，位置非法或表不存在！\n");
        break;

    case 11:
        printf(" 请输入要删除的位置：");
        scanf("%d", &i);
        if (ListDelete(Lists.elem[Lists.cur_index], i, e) == OK)
            printf(" 删除成功，删除元素为：%d\n", e);
        else
            printf(" 删除失败，位置非法或表不存在！\n");
        break;

    case 12:
        printf(" 当前表内容为：\n");
        if (ListTraverse(Lists.elem[Lists.cur_index]) == INFEASIBLE)
            printf(" 表不存在！\n");
```

```
        break;

    case 13:
        Maxsub = MaxSubArray(Lists.elem[Lists.cur_index]);
        if (Maxsub != INFEASIBLE)
            printf(" 最大子数组和为: %d\n", Maxsub);
        else
            printf(" 表不存在或非法! \n");
        break;

    case 14:
        printf(" 请输入要查询的子数组和的大小: ");
        scanf("%d", &k);
        cnt = SubArrayNum(Lists.elem[Lists.cur_index], k);
        if (cnt != INFEASIBLE)
            printf(" 和为%d 的子数组个数: %d\n", k, cnt);
        else
            printf(" 表不存在或非法! \n");
        break;

    case 15:
        if (SortList(Lists.elem[Lists.cur_index]) == OK)
            printf(" 线性表已排序成功! \n");
        else
            printf(" 表不存在! \n");
        break;

    case 16:
        printf(" 请输入文件名: ");
        scanf("%s", filename);
        if (SaveList(Lists.elem[Lists.cur_index], filename) == OK)
            printf(" 保存成功! \n");
        else
            printf(" 保存失败或表不存在! \n");
        break;
```

```
case 17:
    printf(" 请输入文件名: ");
    scanf("%s", filename);
    if (LoadList(Lists.elem[Lists.cur_index], filename) == OK)
        printf(" 加载成功! \n");
    else
        printf(" 加载失败或表已存在! \n");
    break;

case 18:
    printf(" 请输入要添加的表名: ");
    scanf("%s", listname);
    if (AddList(Lists, listname) == OK)
        printf(" 添加成功! \n");
    else
        printf(" 添加失败, 表已存在或内存不足! \n");
    break;

case 19:
    printf(" 请输入要删除的表名: ");
    scanf("%s", listname);
    if (RemoveList(Lists, listname) == OK)
        printf(" 删除成功! \n");
    else
        printf(" 删除失败, 表不存在! \n");
    break;

case 20:
    printf(" 请输入要查找的表名: ");
    scanf("%s", listname);
    i = LocateList(Lists, listname);
    if (i != ERROR)
        printf(" 表 %s 的位置为: %d\n", listname, i);
    else
```

```
            printf(" 表未找到！\n");
        break;

    case 21:
        printf(" 请输入要切换到的表名：");
        scanf("%s", listname);
        i = LocateList(Lists, listname);
        if (i != ERROR)
        {
            Lists.cur_index = i - 1;
            printf(" 切换成功，当前操作表为：%s\n",
            Lists.elem[Lists.cur_index].name);
        }
        else
            printf(" 输入的表不存在！\n");
        break;

    case 22:
        PrintLists(Lists);
        break;
    case 0:
        printf(" 退出程序！\n");
        break;

    default:
        printf(" 无效操作，请重新输入！\n");
        break;
    }

    if (op != 0)
    {
        printf(" 按回车键继续...\n");
        getchar();
        getchar();
    }
```

```
    }

    return 0;
}
```

附录 D 基于邻接表图实现的源程序

```
/*-------------------------------------------
|  Graph System On Adjacency List           |
|  Author:qrq 2402                          |
|  Date:2025.5.21                           |
-------------------------------------------*/

#include "stdio.h"
#include "stdlib.h"
#include <string.h>
#include <unordered_map>
#include <queue>
#include <vector>

using namespace std;

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define MAX_VERTEX_NUM 20
#define MAX_GRAPH_SIZE 10

typedef int status;
typedef int KeyType;

typedef enum
{
    DG,
    DN,
    UDG,
```

```
    UDN
} GraphKind; // 本程序以无向图为例

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

typedef struct ArcNode
{                               // 邻接表结点类型定义
    int adjvex;                 // 顶点位置编号
    struct ArcNode *nextarc; // 下一个结点指针
} 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
    char name[30];       // 图的名称
} ALGraph;

typedef struct
{
    ALGraph elem[MAX_GRAPH_SIZE];
    int length;
    int cur_index;
} GRAPHS;

/*----------------------------------------------*/
```

```
// basic operations
status isKeyTypeUnique(VertexType V[]);
status isKeyTypeUniqueInAdjlist(AdjList list, int len);
status visit(VertexType v);
void dfs(ALGraph G, int n, int visited[]);

// ALGraph system operations
status CreateGraph(ALGraph &G, VertexType V[], KeyType VR[][2]);
status DestroyGraph(ALGraph &G);
int LocateVertex(ALGraph G, KeyType u);
status PutVex(ALGraph &G, KeyType u, VertexType value);
int FirstAdjVex(ALGraph G, KeyType u);
int NextAdjVex(ALGraph G, KeyType v, KeyType w);
status InsertVex(ALGraph &G, VertexType v);
status DeleteVex(ALGraph &G, KeyType v);
status InsertArc(ALGraph &G, KeyType v, KeyType w);
status DeleteArc(ALGraph &G, KeyType v, KeyType w);
status DFSTraverse(ALGraph &G);
status BFSTraverse(ALGraph &G);

// Additional operations
status SaveGraph(ALGraph G, const char FileName[]);
status LoadGraph(ALGraph &G, const char FileName[]);
int ShortestPathLength(ALGraph G, KeyType v, KeyType k);
void VerticesWithinDistanceK(ALGraph G, KeyType v, KeyType K);
int ConnectedComponentsNum(ALGraph G);

// multiple graphs operation
int LocateGraph(GRAPHS Graphs, char *Graphname);
status AddGraph(GRAPHS &Graphs, char *Graphname);
status RemoveGraph(GRAPHS &Graphs, char *Graphname);
status SwitchGraph(GRAPHS &Graphs, char *Graphname);
void PrintGraphslist(GRAPHS Graphs);
void PrintMenu(GRAPHS Graphs);
```

```
void PrintGraphWithAdjList(ALGraph G);

/*----------------------------------------------*/

status isKeyTypeUnique(VertexType V[])
{
    unordered_map<int, int> keyCnt;
    unordered_map<string, int> othersCnt;
    int i = 0;
    while (V[i].key != -1)
    {
        keyCnt[V[i].key]++;
        othersCnt[V[i].others]++;
        if (keyCnt[V[i].key] > 1 || othersCnt[V[i].others] > 1)
            return ERROR;
        i++;
    }
    return OK;
}

status CreateGraph(ALGraph &G, VertexType V[], KeyType VR[][2])
{
    // G.kind = UDG;

    // 算出顶点数和边数
    G.vexnum = 0;
    G.arcnum = 0;
    int i = 0, j = 0;

    while (V[i].key != -1)
    {
        i++;
    }

    while (VR[j][0] != -1)
```

```
{
    j++;
}

G.vexnum = i;
G.arcnum = j;

// 判断顶点数和边数是否超过最大值
if (G.arcnum > MAX_VERTEX_NUM * (MAX_VERTEX_NUM - 1) / 2)
    return ERROR;
if (G.vexnum > MAX_VERTEX_NUM)
    return ERROR;
if (isKeyTypeUnique(V) == ERROR)
    return ERROR;

// 创建图
for (int i = 0; i < G.vexnum; i++)
{
    G.vertices[i].data = V[i];
    G.vertices[i].firstarc = NULL;
}

for (int i = 0; i < j; i++)
{
    int v1 = LocateVertex(G, VR[i][0]);
    int v2 = LocateVertex(G, VR[i][1]);

    // 判断边的两个顶点是否在图中
    if (v1 == -1 || v2 == -1)
        return ERROR;
    // 判断边的两个顶点是否相同，如果相同则跳过
    if (v1 == v2)
    {
        G.arcnum--;
        continue;
```

```
    }
    // 判断边的两个顶点是否已经存在
    int flag = 1;
    if (G.vertices[v1].firstarc != NULL)
    {
        ArcNode *p = G.vertices[v1].firstarc;
        while (p != NULL)
        {
            if (p->adjvex == v2)
                flag = 0;
            p = p->nextarc;
        }
    }
    if (!flag)
    {
        G.arcnum--;
        continue;
    }

    // 在 v1 的邻接表中插入 v2
    ArcNode *p = (ArcNode *)malloc(sizeof(ArcNode));
    if (!p)
        return OVERFLOW;
    p->adjvex = v2;
    p->nextarc = G.vertices[v1].firstarc;
    G.vertices[v1].firstarc = p;

    // 在 v2 的邻接表中插入 v1
    ArcNode *q = (ArcNode *)malloc(sizeof(ArcNode));
    if (!q)
        return OVERFLOW;
    q->adjvex = v1;
    q->nextarc = G.vertices[v2].firstarc;
    G.vertices[v2].firstarc = q;
}
```

```
    return OK;
}

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 *p = G.vertices[i].firstarc;
        while (p)
        {
            ArcNode *q = p;
            p = p->nextarc;
            free(q);
        }
        G.vertices[i].firstarc = NULL;
    }

    G.vexnum = 0;
    G.arcnum = 0;

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

int LocateVertex(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 isKeyTypeUniqueInAdjlist(AdjList list, int len)
{
    unordered_map<int, int> keyCnt;
    unordered_map<string, int> othersCnt;
    for (int i = 0; i < len; i++)
    {
        keyCnt[list[i].data.key]++;
        othersCnt[list[i].data.others]++;
        if (keyCnt[list[i].data.key] > 1 || othersCnt[list[i].data.others]
        > 1)
            return ERROR;
    }
    return OK;
}

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 *********/

    for (int i = 0; i < G.vexnum; i++)
    {
        if (G.vertices[i].data.key == u)
        {
            VertexType temp = G.vertices[i].data;
```

```
            G.vertices[i].data = value;
            if (isKeyTypeUniqueInAdjlist(G.vertices, G.vexnum))
            {
                return OK;
            }
            else
            {
                G.vertices[i].data = temp;
                return ERROR;
            }
        }
    }

    return ERROR;
    /********** 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 *********/

    int i = LocateVertex(G, u);
    if (i == -1)
    {

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

```
    else
        return -1;
    /********** End **********/
}
```

```
int NextAdjVex(ALGraph G, KeyType v, KeyType w)
// v 对应 G 的一个顶点,w 对应 v 的邻接顶点；操作结果是返回 v 的（相对于 w）下一个邻接顶点的位序；如果 w 是最后一个邻接顶点，或 v、w 对应顶点不存在，则返回-1。
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    int n = LocateVertex(G, v);
    int m = LocateVertex(G, w);
    if (n == -1 || m == -1)
    {
        printf(" 顶点不存在\n");
        return -1;
    }

    ArcNode *p = G.vertices[n].firstarc;
    while (p && p->adjvex != m)
    {
        p = p->nextarc;
    }

    if (p == NULL)
    {
        printf(" 顶点 %d 不是顶点 %d 的邻接顶点\n", w, v);
        return -1;
    }
```

```
    if (p && p->nextarc)
    {
        return p->nextarc->adjvex;
    }
    else
    {
        printf(" 顶点 %d 是顶点 %d 的最后一个邻接顶点，没有下一个邻接顶
        点\n", w, v);
        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 *********/
    if (G.vexnum == MAX_VERTEX_NUM)
    {
        printf(" 图的顶点数已达上限，无法插入新顶点\n");
        return ERROR;
    }
    for (int i = 0; i < G.vexnum; i++)
    {
        if (G.vertices[i].data.key == v.key ||
        strcmp(G.vertices[i].data.others, v.others) == 0)
        {
            printf(" 图中已存在该顶点，无法插入\n");
            return ERROR;
        }
    }

    G.vertices[G.vexnum].data = v;
    G.vertices[G.vexnum].firstarc = NULL;
```

```
    G.vexnum++;
    return OK;
    /********** End **********/
}

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 n = LocateVertex(G, v);
    if (n == -1)
    {
        printf(" 图中不存在该顶点，无法删除\n");
        return ERROR;
    }

    // 如果只有一个结点，不能删除
    if (G.vexnum == 1)
    {
        printf(" 图中只有一个结点，无法删除\n");
        return ERROR;
    }

    // 1. 删除所有指向该点的邻接表结点
    for (int i = 0; i < G.vexnum; i++)
    {
        if (i == n)
            continue;
        ArcNode *pre = NULL;
        ArcNode *p = G.vertices[i].firstarc;

        while (p)
        {
            if (p->adjvex == n)
```

```
        {
            if (pre == NULL)
            {
                G.vertices[i].firstarc = p->nextarc;
            }
            else
            {
                pre->nextarc = p->nextarc;
            }

            ArcNode *tmp = p;
            p = p->nextarc;
            free(tmp);
            G.arcnum--;
        }
        else
        {
            if (p->adjvex > n)
                p->adjvex--;

            pre = p;
            p = p->nextarc;
        }
    }
}

// 2. 删除该点关联的邻接表结点
ArcNode *p = G.vertices[n].firstarc;
while (p)
{
    ArcNode *tmp = p;
    p = p->nextarc;
    free(tmp);
}
```

```
    // 3. 调整顶点数组
    for (int i = n; i < G.vexnum - 1; i++)
    {
        G.vertices[i] = G.vertices[i + 1];
    }
    G.vexnum--;

    // printf("%d %d",G.vexnum,G.arcnum);

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

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 n = LocateVertex(G, v);
    int m = LocateVertex(G, w);
    if (n == -1 || m == -1)
    {
        printf(" 图中不存在该顶点，无法插入\n");
        return ERROR;
    }

    ArcNode *temp = G.vertices[m].firstarc;
    while (temp)
    {
        if (temp->adjvex == n)
        {
            printf(" 图中已存在该弧，无法插入\n");
            return ERROR;
        }
        temp = temp->nextarc;
```

```
    }

    ArcNode *p = (ArcNode *)malloc(sizeof(ArcNode));
    p->adjvex = n;
    p->nextarc = G.vertices[m].firstarc;
    G.vertices[m].firstarc = p;

    ArcNode *q = (ArcNode *)malloc(sizeof(ArcNode));
    q->adjvex = m;
    q->nextarc = G.vertices[n].firstarc;
    G.vertices[n].firstarc = q;

    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 n = LocateVertex(G, v);
    int m = LocateVertex(G, w);
    if (n == -1 || m == -1)
    {
        printf(" 图中不存在该顶点，无法删除\n");
        return ERROR;
    }

    ArcNode *temp = G.vertices[m].firstarc;
    int flag = 0;
    while (temp)
    {
```

```
        if (temp->adjvex == n)
            flag = 1;
        temp = temp->nextarc;
    }
    if (!flag)
    {
        printf(" 图中不存在该弧，无法删除\n");
        return ERROR;
    }

    ArcNode *p = G.vertices[m].firstarc;
    temp = NULL;
    while (p && p->adjvex != n)
    {
        temp = p;
        p = p->nextarc;
    }

    if (p == G.vertices[m].firstarc)
    {
        G.vertices[m].firstarc = p->nextarc;
    }
    else
    {
        temp->nextarc = p->nextarc;
    }
    free(p);

    ArcNode *q = G.vertices[n].firstarc;
    temp = NULL;
    while (q && q->adjvex != m)
    {
        temp = q;
        q = q->nextarc;
    }
```

```
    if (q == G.vertices[n].firstarc)
    {
        G.vertices[n].firstarc = q->nextarc;
    }
    else
    {
        temp->nextarc = q->nextarc;
    }
    free(q);

    G.arcnum--;

    return OK;

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

void dfs(ALGraph G, int n, int visited[])
{
    visited[n] = 1;
    visit(G.vertices[n].data);

    ArcNode *p = G.vertices[n].firstarc;
    while (p)
    {
        if (!visited[p->adjvex])
        {
            dfs(G, p->adjvex, visited);
        }
        p = p->nextarc;
    }

    return;
}
```

```
status visit(VertexType v)
{
    // 访问顶点的操作
    printf("%d %s\n", v.key, v.others);
    return OK;
}

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

    for (int i = 0; i < G.vexnum; i++)
    {
        if (!visited[i])
        {
            dfs(G, i, visited);
        }
    }

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

status BFSTraverse(ALGraph &G)
// 对图 G 进行广度优先搜索遍历，依次对图中的每一个顶点使用函数 visit 访问一次，且仅访问一次
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    bool visited[G.vexnum] = {FALSE};
```

```
    queue<int> q;

    for (int i = 0; i < G.vexnum; i++)
    {
        if (!visited[i])
        {
            visit(G.vertices[i].data);
            visited[i] = TRUE;
            q.push(i);
        }

        while (!q.empty())
        {
            int n = q.front();
            q.pop();
            ArcNode *p = G.vertices[n].firstarc;

            while (p)
            {
                int m = p->adjvex;
                if (!visited[m])
                {
                    visit(G.vertices[m].data);
                    visited[m] = TRUE;
                    q.push(m);
                }
                p = p->nextarc;
            }
        }
    }

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

```
status SaveGraph(ALGraph G, const char FileName[])
{
    FILE *fp = fopen(FileName, "w");
    if (!fp)
        return ERROR;

    fprintf(fp, "%d %d\n", G.vexnum, G.arcnum); // 写入顶点数和边数

    for (int i = 0; i < G.vexnum; ++i)
    {
        // 写入顶点 key 和 others
        fprintf(fp, "%d %s", G.vertices[i].data.key,
        G.vertices[i].data.others);

        // 写入邻接点索引
        ArcNode *p = G.vertices[i].firstarc;
        while (p != NULL)
        {
            fprintf(fp, " %d", p->adjvex); // 写入邻接点在数组中的索引
            p = p->nextarc;
        }
        fprintf(fp, " -1 "); // 结束邻接点的输入
        // 换行
        fprintf(fp, "\n");
    }

    fclose(fp);
    return OK;
}

status LoadGraph(ALGraph &G, const char FileName[])
{
    FILE *fp = fopen(FileName, "r");
    if (!fp)
        return ERROR;
```

```
// 读取顶点数和边数
fscanf(fp, "%d %d", &G.vexnum, &G.arcnum);
if (G.vexnum > MAX_VERTEX_NUM || G.arcnum > MAX_VERTEX_NUM *
(MAX_VERTEX_NUM - 1) / 2)
    return ERROR;

// 读取顶点和它的邻接点
for (int i = 0; i < G.vexnum; ++i)
{
    fscanf(fp, "%d %s", &G.vertices[i].data.key,
    G.vertices[i].data.others);
    G.vertices[i].firstarc = NULL;
    int adjvex;
    while (fscanf(fp, "%d", &adjvex) == 1 && adjvex != -1)
    {
        ArcNode *p = (ArcNode *)malloc(sizeof(ArcNode));
        if (!p)
            return OVERFLOW;
        p->adjvex = adjvex;
        // 尾插法构建邻接表
        ArcNode *q = G.vertices[i].firstarc;
        if (q == NULL)
        {
            G.vertices[i].firstarc = p;
            p->nextarc = NULL;
        }
        else
        {
            while (q->nextarc != NULL)
                q = q->nextarc;
            q->nextarc = p;
            p->nextarc = NULL;
        }
    }
```

```
    }

    fclose(fp);
    return OK;
}

int ShortestPathLength(ALGraph G, KeyType v, KeyType k)
{
    int n = LocateVertex(G, v);
    int m = LocateVertex(G, k);
    if (n == -1 || m == -1)
        return ERROR;

    int visited[G.vexnum] = {0};
    int distance[G.vexnum] = {0};

    queue<int> q;
    q.push(n);

    visited[n] = 1;
    distance[n] = 0;

    while (!q.empty())
    {
        int u = q.front();
        q.pop();

        ArcNode *p = G.vertices[u].firstarc;
        while (p)
        {
            int w = p->adjvex;
            if (!visited[w])
            {
                visited[w] = 1;
                distance[w] = distance[u] + 1;
```

```
                q.push(w);
            }
            p = p->nextarc;
        }
    }

    if (visited[m])
        return distance[m];
    else
        return ERROR;
}

void VerticesWithinDistanceK(ALGraph G, KeyType v, KeyType K)
{
    if (K < 0)
    {
        printf(" 输入的 K 值不合法\n");
        return;
    }

    int start = LocateVertex(G, v);
    if (start == -1)
    {
        printf(" 输入的顶点不合法\n");
        return;
    }

    if (G.vexnum == 0)
    {
        printf(" 图为空\n");
        return;
    }

    if (G.arcnum == 0)
    {
```

```
    printf(" 图中没有边\n");
    return;
}

// 使用一次 BFS 求所有点到 v 的距离，避免多次 ShortestPathLength 调用
int distance[MAX_VERTEX_NUM];
for (int i = 0; i < G.vexnum; ++i)
    distance[i] = -1;
queue<int> q;
distance[start] = 0;
q.push(start);

while (!q.empty())
{
    int u = q.front();
    q.pop();
    ArcNode *p = G.vertices[u].firstarc;
    while (p)
    {
        int w = p->adjvex;
        if (distance[w] == -1)
        {
            distance[w] = distance[u] + 1;
            q.push(w);
        }
        p = p->nextarc;
    }
}

int flag = 0;
for (int i = 0; i < G.vexnum; i++)
{
    if (i == start)
        continue;
    if (distance[i] != -1 && distance[i] < K)
```

```
        {
            flag = 1;
            printf(" %d %s", G.vertices[i].data.key,
            G.vertices[i].data.others);
        }
    }

    if (!flag)
        printf(" 没有满足条件的顶点\n");

    return;
}

int ConnectedComponentsNum(ALGraph G)
{
    vector<int> visited(G.vexnum, 0);
    int cnt = 0;

    for (int i = 0; i < G.vexnum; i++)
    {
        if (!visited[i])
        {
            cnt++;
            queue<int> q;
            q.push(i);
            visited[i] = 1;

            while (!q.empty())
            {
                int u = q.front();
                q.pop();

                ArcNode *p = G.vertices[u].firstarc;
                while (p)
                {
```

```
                    int w = p->adjvex;
                    if (!visited[w])
                    {
                        visited[w] = 1;
                        q.push(w);
                    }
                    p = p->nextarc;
                }
            }
        }
    }

    return cnt;
}
```

```
int LocateGraph(GRAPHS Graphs, char *Graphname)
{
    for (int i = 0; i < Graphs.length; i++)
    {
        if (strcmp(Graphs.elem[i].name, Graphname) == 0)
            return i;
    }
    return -1;
}

status AddGraph(GRAPHS &Graphs, char *Graphname)
{
    if (Graphs.length >= MAX_GRAPH_SIZE)
    {
        printf(" 图的数量已达上限，无法添加新图\n");
        return ERROR;
```

```
    }

    for (int i = 0; i < Graphs.length; i++)
    {
        if (strcmp(Graphs.elem[i].name, Graphname) == 0)
        {
            printf(" 图已存在，无法添加\n");
            return ERROR;
        }
    }

    // 创建新图
    ALGraph newGraph;
    newGraph.vexnum = 0;
    newGraph.arcnum = 0;
    newGraph.kind = UDG; // 默认无向图
    strcpy(newGraph.name, Graphname);
    Graphs.elem[Graphs.length] = newGraph;
    Graphs.length++;
    Graphs.cur_index = Graphs.length - 1;

    return OK;
}

status RemoveGraph(GRAPHS &Graphs, char *Graphname)
{
    int index = LocateGraph(Graphs, Graphname);
    if (index == -1)
    {
        printf(" 图不存在，无法删除\n");
        return ERROR;
    }

    // 销毁图
    DestroyGraph(Graphs.elem[index]);
```

```
    memset(&Graphs.elem[index], 0, sizeof(ALGraph));

    // 删除图
    for (int i = index; i < Graphs.length - 1; i++)
    {
        Graphs.elem[i] = Graphs.elem[i + 1];
    }
    Graphs.length--;

    return OK;
}

status SwitchGraph(GRAPHS &Graphs, char *Graphname)
{
    int index = LocateGraph(Graphs, Graphname);
    if (index == -1)
    {
        printf(" 图不存在，无法切换\n");
        return ERROR;
    }

    Graphs.cur_index = index;
    return OK;
}

void PrintGraphslist(GRAPHS Graphs)
{
    printf(" 当前图列表: \n");
    for (int i = 0; i < Graphs.length; i++)
    {
        printf("%d: %s\n", i + 1, Graphs.elem[i].name);
    }
}

void PrintGraphWithAdjList(ALGraph G)
```

```
{
    printf("\n当前的图和邻接表: \n");
    printf("-----------------------------------------------\n");
    printf(" 图名称: %s\n", G.name);
    printf(" 顶点数: %d, 边数: %d\n", G.vexnum, G.arcnum);
    printf("-----------------------------------------------\n");
    printf(" 位置序号  顶点信息          邻接点\n");
    printf("-----------------------------------------------\n");

    for (int i = 0; i < G.vexnum; i++)
    {
        printf("%-9d %-3d %-10s ", i, G.vertices[i].data.key,
        G.vertices[i].data.others);

        ArcNode *p = G.vertices[i].firstarc;
        if (!p)
        {
            printf("(无邻接点)");
        }
        else
        {
            while (p)
            {
                printf("%d", p->adjvex);
                p = p->nextarc;
                if (p)
                    printf("->");
            }
        }
        printf("\n");
    }
    printf("-----------------------------------------------\n");
}

void PrintMenu(GRAPHS Graphs)
```

```
{
    printf("\n");
    printf("-----------------------------------------------------\n");
    printf("   Menu for Graph System On Adjacency List \n");

    printf("-----------------------------------------------------\n\n");
    printf(" 当前图: %s\n", Graphs.elem[Graphs.cur_index].name);
    printf("Basic Operations:\n");
    printf("      1. CreateGraph              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("Additional Operations:\n");
    printf("     13. SaveGraph               14. LoadGraph           \n");
    printf("     15. ShortestPathLength      16. VerticesWithinK     \n");
    printf("     17. ConnectedComponentsNum                          \n");
    printf("Multiple Graphs Operations:\n");
    printf("     18. AddGraph                19. RemoveGraph         \n");
    printf("     20. SwitchGraph             21. PrintGraphs         \n");
    printf("     0. Exit                     22.
    PrintGraphWithAdjList\n");
    printf("-----------------------------------------------------\n");
    printf(" 请输入操作选项: \n");
}

int main()
{
    GRAPHS Graphs;
    Graphs.length = 1;
    Graphs.cur_index = 0;
    ALGraph G;
    G.vexnum = 0;
    G.arcnum = 0;
```

```
G.kind = UDG; // 默认无向图
Graphs.elem[0] = G;
char firstname[30];
system("cls");
printf("-------------------------------------------------\n");
printf(" 邻接表实现的图系统，可以对单个或多个无向图进行操作\n");
printf("Author: HUST Cupid-qrq\n");
printf("-------------------------------------------------\n");
printf(" 在开始之前，请先输入第一个图的名称：\n");
if (scanf("%s", firstname) != 1)
{
    printf(" 输入错误，请重新输入：\n");
    scanf("%s", firstname);
}
printf(" 输入成功，第一个图的名称是：%s\n", firstname);
strcpy(Graphs.elem[0].name, firstname);
printf(" 接下来，进入菜单界面，按任意键继续：\n");
getchar();
getchar();

int op = 1;                    // 菜单选项
VertexType V[MAX_VERTEX_NUM];  // 顶点数组
KeyType VR[MAX_VERTEX_NUM][2]; // 边数组
int exist;

char filename[100], treename[100];
// to be continued

while (op)
{
    PrintMenu(Graphs);

    if (scanf("%d", &op) != 1)
    {
        op = -1;
```

```
}

switch (op)
{
case 1:
{
    // 创建图
    if (Graphs.elem[Graphs.cur_index].vexnum != 0)
    {
        printf(" 当前图已存在，创建失败！\n");
        break;
    }

    // 初始化定义数组

    memset(V, 0, sizeof(V));
    memset(VR, 0, sizeof(VR));

    printf(" 请输入图的顶点序列，格式为：key others，以 -1 nil 结束。
    \n");
    int i = 0;
    do
    {
        scanf("%d%s", &V[i].key, V[i].others);
    } while (V[i++].key != -1);
    i = 0;
    printf(" 请输入图的边序列，格式为：v1 v2，以 -1 -1 结束。\n");
    do
    {
        scanf("%d%d", &VR[i][0], &VR[i][1]);
    } while (VR[i++][0] != -1);

    // 构造新图
    status res = CreateGraph(Graphs.elem[Graphs.cur_index], V,
    VR);
```

```
        if (res == OK)
        {
            printf(" 图创建成功。\n");
        }
        else
        {
            printf(" 图创建失败，可能原因：重复 key 或内存不足。\n");
        }

        break;
    }
    case 2:
    {
        if (Graphs.elem[Graphs.cur_index].vexnum == 0)
        {
            printf(" 当前图不存在，销毁失败！\n");
            break;
        }

        if (DestroyGraph(Graphs.elem[Graphs.cur_index]) == OK)
            printf(" 图销毁成功！\n");
        else
            printf(" 图不存在，销毁失败！\n");

        break;
    }
    case 3:
    {
        if (Graphs.elem[Graphs.cur_index].vexnum == 0)
        {
            printf(" 当前图不存在，查找失败！\n");
            break;
        }

        int index;
```

```
        printf(" 请输入要查找的结点的关键字: \n");
        if (scanf("%d", &exist) != 1)
        {
            printf(" 输入错误, 请重新输入: \n");
            continue;
        }

        index = LocateVertex(Graphs.elem[Graphs.cur_index], exist);
        if (index == -1)
        {
            printf(" 查找失败, 结点不存在! \n");
        }
        else
        {
            printf(" 查找成功, 结点的顶点位置序号是: %d\n", index);
        }

        break;
    }
    case 4:
    {
        if (Graphs.elem[Graphs.cur_index].vexnum == 0)
        {
            printf(" 当前图不存在, 修改失败! \n");
            break;
        }

        exist = 0;
        printf(" 请输入要修改的结点的关键字: \n");
        if (scanf("%d", &exist) != 1)
        {
            printf(" 输入错误, 请重新输入: \n");
            continue;
        }
```

```
        int index = LocateVertex(Graphs.elem[Graphs.cur_index],
        exist);
        if (index == -1)
        {
            printf(" 查找失败，结点不存在！\n");
        }
        else
        {
            printf(" 请输入新的结点值：\n");
            VertexType newVex;
            scanf("%d", &newVex.key);
            strcpy(newVex.others,
            Graphs.elem[Graphs.cur_index].vertices[index].data.others);

            if (PutVex(Graphs.elem[Graphs.cur_index], exist, newVex)
            == OK)
            {
                printf(" 修改成功！\n");
            }
            else
            {
                printf(" 修改失败，可能原因：重复 key 或内存不足。\n");
            }
        }

        break;
    }
    case 5:
    {
        if (Graphs.elem[Graphs.cur_index].vexnum == 0)
        {
            printf(" 当前图不存在，查找失败！\n");
            break;
        }
```

```
        exist = 0;
        printf(" 请输入要查找的结点的关键字: \n");
        if (scanf("%d", &exist) != 1)
        {
            printf(" 输入错误, 请重新输入: \n");
            continue;
        }
        int res = FirstAdjVex(Graphs.elem[Graphs.cur_index], exist);
        if (res == -1)
        {
            printf(" 查找失败, 结点不存在! \n");
        }
        else
        {
            printf(" 查找成功, 结点的第一邻接顶点是: \n");
            printf(" %d %s\n",
            Graphs.elem[Graphs.cur_index].vertices[res].data.key,
            Graphs.elem[Graphs.cur_index].vertices[res].data.others);
        }

        break;
    }
    case 6:
    {
        if (Graphs.elem[Graphs.cur_index].vexnum == 0)
        {
            printf(" 当前图不存在, 查找失败! \n");
            break;
        }

        int v, w;
        printf(" 请输入要查找的结点的关键字: \n");
        if (scanf("%d", &v) != 1)
        {
            printf(" 输入错误, 请重新输入: \n");
```

```
            continue;
        }
        printf(" 请输入该结点的一个邻接结点的关键字: \n");
        if (scanf("%d", &w) != 1)
        {
            printf(" 输入错误, 请重新输入: \n");
            continue;
        }
        int res2 = NextAdjVex(Graphs.elem[Graphs.cur_index], v, w);
        if (res2 == -1)
        {
            printf(" 查找失败\n");
        }
        else
        {
            printf(" 查找成功, 结点的下一个邻接顶点是: \n");
            printf(" %d %s\n",
            Graphs.elem[Graphs.cur_index].vertices[res2].data.key,
            Graphs.elem[Graphs.cur_index].vertices[res2].data.others);
        }

        break;
    }
    case 7:
    {
        if (Graphs.elem[Graphs.cur_index].vexnum == 0)
        {
            printf(" 当前图不存在, 插入顶点失败! \n");
            break;
        }

        VertexType newVex;
        printf(" 请输入要插入的结点的 key 和 others: \n");
        if (scanf("%d %s", &newVex.key, newVex.others) != 2)
        {
```

```
            printf(" 输入错误，请重新输入: \n");
            continue;
        }

        if (InsertVex(Graphs.elem[Graphs.cur_index], newVex) == OK)
        {
            printf(" 插入成功! \n");
        }
        else
        {
            printf(" 插入失败\n");
        }

        break;
    }
    case 8:
    {
        if (Graphs.elem[Graphs.cur_index].vexnum == 0)
        {
            printf(" 当前图不存在，删除顶点失败! \n");
            break;
        }

        int exist = 0;
        printf(" 请输入要删除的结点的关键字: \n");
        if (scanf("%d", &exist) != 1)
        {
            printf(" 输入错误，请重新输入: \n");
            continue;
        }

        if (DeleteVex(Graphs.elem[Graphs.cur_index], exist) == OK)
        {
            printf(" 删除成功! \n");
        }
```

```
        else
        {
            printf(" 删除失败。\n");
        }

        break;
    }
    case 9:
    {
        if (Graphs.elem[Graphs.cur_index].vexnum == 0)
        {
            printf(" 当前图不存在，插入边失败！\n");
            break;
        }

        int v1, v2;
        printf(" 请输入要插入的弧的两个顶点的关键字：\n");
        if (scanf("%d %d", &v1, &v2) != 2)
        {
            printf(" 输入错误，请重新输入：\n");
            continue;
        }

        if (InsertArc(Graphs.elem[Graphs.cur_index], v1, v2) == OK)
        {
            printf(" 插入成功！\n");
        }
        else
        {
            printf(" 插入失败。\n");
        }

        break;
    }
    case 10:
```

```
{
    if (Graphs.elem[Graphs.cur_index].vexnum == 0)
    {
        printf(" 当前图不存在，删除边失败！\n");
        break;
    }

    int v1, v2;
    v1 = v2 = 0;
    printf(" 请输入要删除的弧的两个顶点的关键字：\n");
    if (scanf("%d %d", &v1, &v2) != 2)
    {
        printf(" 输入错误，请重新输入：\n");
        continue;
    }

    if (DeleteArc(Graphs.elem[Graphs.cur_index], v1, v2) == OK)
    {
        printf(" 删除成功！\n");
    }
    else
    {
        printf(" 删除失败。\n");
    }

    break;
}
case 11:
{
    if (Graphs.elem[Graphs.cur_index].vexnum == 0)
    {
        printf(" 当前图不存在，遍历失败！\n");
        break;
    }
```

```
        printf(" 深度优先遍历结果: \n");
        DFSTraverse(Graphs.elem[Graphs.cur_index]);
        printf("\n");

        break;
    }
    case 12:
    {
        if (Graphs.elem[Graphs.cur_index].vexnum == 0)
        {
            printf(" 当前图不存在, 遍历失败! \n");
            break;
        }

        printf(" 广度优先遍历结果: \n");
        BFSTraverse(Graphs.elem[Graphs.cur_index]);
        printf("\n");

        break;
    }
    case 13:
    {
        if (Graphs.elem[Graphs.cur_index].vexnum == 0)
        {
            printf(" 当前图不存在, 保存失败! \n");
            break;
        }

        printf(" 请输入要保存的文件名: \n");
        if (scanf("%s", filename) != 1)
        {
            printf(" 输入错误, 请重新输入: \n");
            continue;
        }
```

```
        if (SaveGraph(Graphs.elem[Graphs.cur_index], filename) == OK)
        {
            printf(" 保存成功！\n");
        }
        else
        {
            printf(" 保存失败。\n");
        }

        break;
    }
    case 14:
    {
        if (Graphs.elem[Graphs.cur_index].vexnum != 0)
        {
            printf(" 当前图已存在，加载失败！\n");
            break;
        }

        printf(" 请输入要加载的文件名：\n");
        if (scanf("%s", filename) != 1)
        {
            printf(" 输入错误，请重新输入：\n");
            continue;
        }

        if (LoadGraph(Graphs.elem[Graphs.cur_index], filename) == OK)
        {
            printf(" 加载成功！\n");
        }
        else
        {
            printf(" 加载失败。\n");
        }
```

```
        break;
    }
    case 15:
    {
        if (Graphs.elem[Graphs.cur_index].vexnum == 0)
        {
            printf(" 当前图不存在，查找失败！\n");
            break;
        }

        int v1, v2;
        v1 = v2 = 0;
        printf(" 请输入要查找的结点的关键字：\n");
        if (scanf("%d %d", &v1, &v2) != 2)
        {
            printf(" 输入错误，请重新输入：\n");
            continue;
        }

        int res3 = ShortestPathLength(Graphs.elem[Graphs.cur_index],
        v1, v2);
        if (res3 == ERROR)
        {
            printf(" 查找失败，结点不存在！\n");
        }
        else
        {
            printf(" 查找成功，最短路径长度是：%d\n", res3);
        }

        break;
    }
    case 16:
    {
        if (Graphs.elem[Graphs.cur_index].vexnum == 0)
```

```
        {
            printf(" 当前图不存在，查找失败！\n");
            break;
        }

        int v1 = 0;
        int k = 0;
        printf(" 请输入要查找的结点的关键字：\n");
        if (scanf("%d", &v1) != 1)
        {
            printf(" 输入错误，请重新输入：\n");
            continue;
        }
        printf(" 请输入要查找的距离：\n");
        if (scanf("%d", &k) != 1)
        {
            printf(" 输入错误，请重新输入：\n");
            continue;
        }
        printf(" 查找结果：\n");
        VerticesWithinDistanceK(Graphs.elem[Graphs.cur_index], v1, k);
        printf("\n");

        break;
    }
    case 17:
    {
        if (Graphs.elem[Graphs.cur_index].vexnum == 0)
        {
            printf(" 当前图不存在，查找失败！\n");
            break;
        }

        int num =
        ConnectedComponentsNum(Graphs.elem[Graphs.cur_index]);
```

```
        printf(" 连通分量的个数是: %d\n", num);

        break;
    }
    case 18:
    {
        if (Graphs.length >= MAX_GRAPH_SIZE)
        {
            printf(" 图的数量已达上限，无法添加新图\n");
            break;
        }

        printf(" 请输入要添加的图的名称: \n");
        if (scanf("%s", treename) != 1)
        {
            printf(" 输入错误，请重新输入: \n");
            continue;
        }

        if (AddGraph(Graphs, treename) == OK)
        {
            printf(" 添加成功! \n");
        }
        else
        {
            printf(" 添加失败。\n");
        }

        break;
    }
    case 19:
    {
        printf(" 请输入要删除的图的名称: \n");
        if (scanf("%s", treename) != 1)
        {
```

```
            printf(" 输入错误，请重新输入：\n");
            continue;
        }

        if (RemoveGraph(Graphs, treename) == OK)
        {
            printf(" 删除成功！\n");
        }

        break;
    }
    case 20:
    {
        printf(" 请输入要切换的图的名称：\n");
        if (scanf("%s", treename) != 1)
        {
            printf(" 输入错误，请重新输入：\n");
            continue;
        }

        if (SwitchGraph(Graphs, treename) == OK)
        {
            printf(" 切换成功！\n");
        }

        break;
    }
    case 21:
    {
        PrintGraphslist(Graphs);
        break;
    }

    case 22:
    {
```

```
            if (Graphs.elem[Graphs.cur_index].vexnum == 0)
            {
                printf(" 当前图不存在，打印失败！\n");
                break;
            }
            PrintGraphWithAdjList(Graphs.elem[Graphs.cur_index]);
            break;
        }
        case 0:
        {
            printf(" 退出系统！\n");
            break;
        }
        default:
            printf(" 输入错误，请重新输入：\n");
            break;
        }

        if (op != 0)
        {
            printf(" 按回车键继续...\n");
            getchar();
            getchar();
        }
    }

    return 0;
}
```