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 CS22xx

学　　号 Uxxxxxxxxx

姓　　名 711LLL711

指导教师 李丹

报告日期 2023 年 5 月 30 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

采用顺序表作为线性表的物理结构，要求构造一个具有菜单的功能演示系统，演示系统可选择实现多个线性表管理。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。

需要以函数形式定义线性表的初始化表、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，具体操作功能定义如下。

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

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

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

（4）判定空表：函数名称是 ListEmpty(L)；初始条件是线性表 L 已存在；

操作结果是若 L 为空表则返回 TRUE, 否则返回 FALSE；

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

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

（7）查找元素：函数名称是 LocateElem(L,e)；初始条件是线性表已存在；操作结果是返回 L 中第 1 个与 e 相等的数据元素的位序，若这样的数据元素不存在，则返回 ERROR；

（8）获得前驱：函数名称是 PriorElem(L,e,pre)；初始条件是线性表 L 已存在；操作结果是获取线性表 L 元素 e 的前驱，保存在 pre 中，返回 OK；如果没有前驱，返回 ERROR；

（9）获得后继：函数名称是 NextElem(L,e,next)；初始条件是线性表 L 已存在；操作结果是获取线性表 L 元素 e 的后继，保存在 next 中，返回 OK；如果没有后继，返回 ERROR；

（10）插入元素：函数名称是 ListInsert(L,i,e)；初始条件是线性表 L 已存在，

1≤i≤ListLength(L)+1；操作结果是在 L 的第 i 个位置之前插入新的数据元素 e，操作成功返回 OK；

（11）删除元素：函数名称是 ListDelete(L,i,e)；初始条件是线性表 L 已存在且非空，1≤i≤ListLength(L)；操作结果：删除 L 的第 i 个数据元素，将删除的元素存入 e 中，操作成功返回 OK；

（12）遍历表：函数名称是 ListTraverse(L)，初始条件是线性表 L 已存在；操作结果是依次对 L 的每个数据元素进行读取输出。

对于文件管理以及多表管理，同样需要以函数形式扩展文件保存与读取操作以及多表管理操作的基本操作，具体操作功能定义如下。

（1）文件保存：函数名称是 SaveList(L, FileName)，初始条件是线性表 L 已存在；操作结果是将线性表 L 的的元素写到 FileName 文件中，返回 OK；

（2）文件读取：函数名称是 LoadList(L, FileName)，初始条件是线性表 L 不存在；操作结果是将 FileName 文件中的数据读入到线性表 L 中，返回 OK；

（3）多表操作：通过修改 main 函数中的变量 i_num，切换所要操作的线性表。在创建上述函数的基础上，构建基于顺序存储结构的线性表。过程中合理运用各个函数，完成整个系统的构建。

1.2 系统设计

1.2.1 系统总体设计框架

本系统提供一个顺序存储的线性表。支持多表操作，在用户操作前，可以选择在哪个线性表上进行操作。

界面菜单包含所有操作，菜单可供选择的操作有：初始化线性表、销毁表、清空表、判空表，求表长、得到某元素、查找元素、获得某元素的前驱、获得某元素的后继、插入元素、删除元素、遍历线性表、求最大连续子数组和、和为 K 的子数组、顺序表排序、实现线性表的文件形式保存、实现多个线性表管理。

1.2.2 定义常量、数据类型以及数据结构

为了使得系统源码便于统一化管理和便捷化浏览，定义常数以及数据类型是必不可少的环节，将相关含义与常量或数据类型对应起来可方便操作。

（1）定义常量：一部分常量用来指定函数的返回值，用于判断函数执行的情

况。定义的常量有 TRUE(1),FALSE(0),OK(1),ERROR(0),INFEASIBLE(-1), OVERFLOW(-2)。还有常量用于指定线性表存储空间的分配量以及用于分配更多空间的分配量：LIST_INIT_SIZE (100)，LISTINCREMENT(10)。

（2）定义数据类型：此处定义了部分函数返回值的类型 status 以及线性表中单个数据的类型 ElemType。本实验中将二者都定义为了 int 类型。

（3）定义数据结构：顺序表结构体中包含三种元素，分别为元素存储空间的首指针 elem、当前已存储元素的数量 length 以及当前分配的存储空间的总长度 listsize，并将其类型定义为 SqList。线性表的管理使用了简单的数组，数组元素的数据类型是 SqList。通过改变当前数组的索引值选择要操作的线性表。

1.2.3 基本操作函数算法设计

调用函数返回值包含 OK、ERROR、TRUE、FALSE 以及 INFEASIBLE 五种情况。OK 对应相关操作处理成功；ERROR 对应操作处理失败；INFEASIBLE 对应表空无的判断，如 InitList(L) 初始化表函数在表不为空的情况下，返回 INFEASIBLE。而 DestroyList(L) 销毁表函数在表为空的情况下，返回 INFEASIBLE。具体函数算法设计如下。

（1）构造空表 status InitList(SqList& L) 将表头结点指向存储区域。该函数只包含判断和顺序结构，因此其时间复杂度为 O(1)。

（2）销毁单表 status DestroyList(SqList& L) 函数释放指针指向的线性表的数据存储空间，并将将线性表的数组指针域赋值为空指针。该函数只有判断和顺序结构，因此其时间复杂度为 O(1)。

（3）清空单表 status ClearList(SqList& L) 将线性表中的数据清空。该函数只有判断和顺序结构，因此其时间复杂度为 O(1)。

（4）判断空表 status ListEmpty(SqList L) 判断线性表长度是否为 0，为 0 就是空表。该函数只有判断和顺序结构，因此其时间复杂度为 O(1)。

（5）单表长度 status ListLength(SqList L) 使用循环结构遍历线性表获取单表长度。该函数只有判断和顺序结构，因此其时间复杂度为 O(1)。

（6）获取元素 status GetElem(SqList L, int i, ElemType& e) 通过使用循环结构遍历线性表并记录遍历长度，查找特定位置的元素。该函数只有判断和顺序结构，因此其时间复杂度为 O(1)。

（7）查找元素 status LocateElem(SqList L, ElemType e) 通过使用循环结构遍

历线性表并记录遍历长度，判断该元素是否与目标元素相等，查找元素的位置。该函数包含一个循环结构，在最好的情况下，即线性表中第一个元素即为待查找的元素，则循环执行 1 次，在最坏的情况下，即线性表中最后一个元素为待查找的元素或待查找的元素不在线性表中，则循环执行 n 次 (n 为线性表当前的表长)，因此循环平均执行 (n+1)/2 次，则函数的时间复杂度为 O(n)。

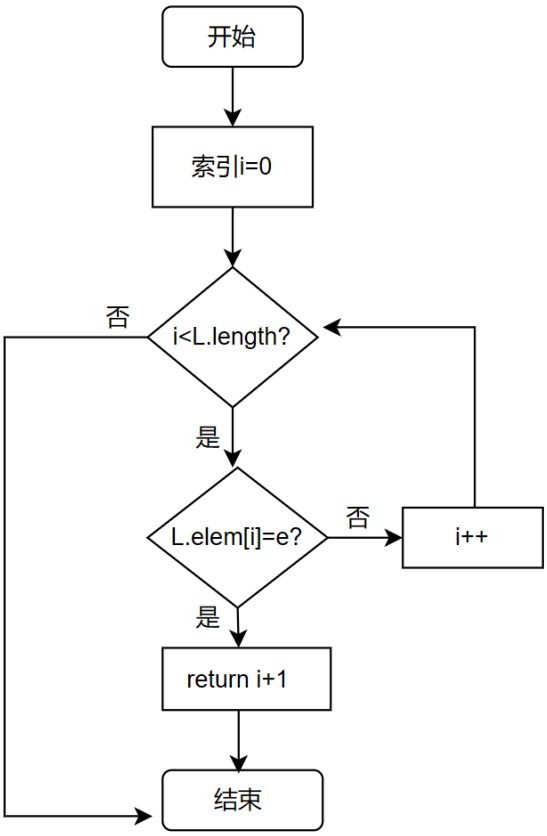


图 1-1　查找元素流程图

（8）获取后继 status NextElem(SqList L, ElemType e, ElemType& next) 使用循环结构查找目标元素，并判断它和它的后继是否存在。该函数包含一个循环结构，在待查找后继的元素为线性表的第一个元素时，循环执行 1 次，当待查找后继的元素为线性表的倒数第二个元素或者不在线性表中时，循环执行 n-1 次 (n 为线性表当前的表长)，因此循环平均执行 n/2 次，则函数的时间复杂度为 O(n)。

（9）获取前驱 status PriorElem(SqList L, ElemType e, ElemType& pre) 使用循环结构查找目标元素，并判断它和它的前驱是否存在。该函数包含一个循环结构，在待查找前驱的元素为第一个元素时，循环不执行，在待查找元素为第二个元素时，循环执行 1 次，这两种情况为最好的情况，当待查找前驱的元素为最后一个元素或者不在线性表中时，循环执行 n 次 (n 为线性表当前的表长)，因此循

环平均执行 n/2 次，则函数的时间复杂度为 O(n)。

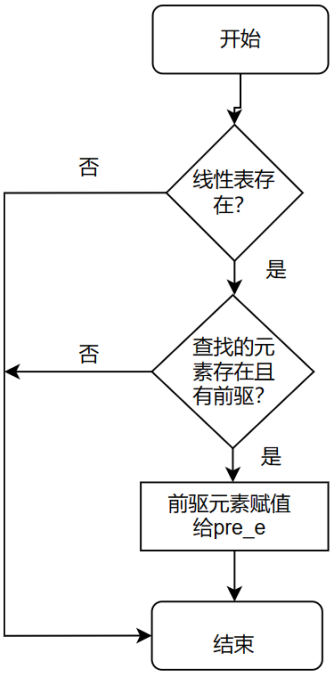


图 1-2　获取前驱元素流程图

（10）插入元素 status ListInsert(SqList& L, int i, ElemType e) 使用循环结构遍历表找到要插入的位置，插入元素。该函数包含一个循环结构，当插入位置错误时，循环不执行；当插入元素位置为线性表的最后一个位置时，循环执行 1 次；当插入元素位置为线性表的第一个位置时，函数执行 n 次。因此循环平均执行 n/2 次，则函数的时间复杂度为 O(n)。

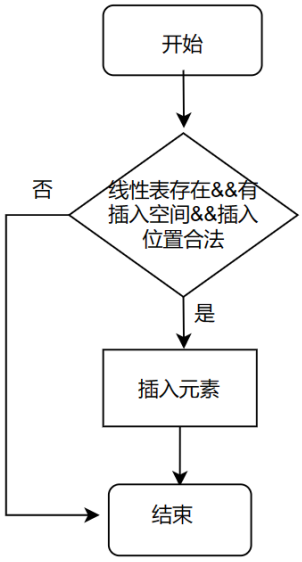


图 1-3　插入流程图

（11）删除元素 status ListDelete(SqList& L, int i,ElemType& e) 使用循环结构遍历表找到要删除的位置，删除元素。该函数包含一个循环结构，当输入位置错误或者删除最后一个元素时，循环不执行；当待删除元素为线性表的倒数第二个元素时，循环执行 1 次；当待删除的元素为第一个元素时，循环执行 n-1 次，因此循环平均执行 (n-1)/2 次，则函数的时间复杂度为 O(n)。

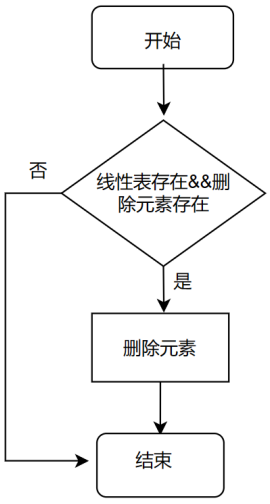


图 1-4　删除元素流程图

（12）遍历单表 status ListTraverse(SqList L) 函数使用循环结构遍历线性表中的每一个元素，并将其逐个输出。该函数包含一个循环结构，执行 n 次，故其时

间复杂度为 O(n)。

1.2.4 附加功能设计

1) 函数名称：Maxsubarray
   函数功能：找出有最大和的连续子数组
   思路：遍历线性表中的元素，运用贪心算法，如果当前累加和为 0，更新 s 结果，如果当前累加和为负，将当前累加和重置为 0，表示从该位置重新开始求和
   返回值类型：int 类型, 返回其最大和

2) 函数名称：subarraynum
   函数功能：找出和为 k 的连续子组的个数
   思路：使用双指针，左指针指向起始位置，分别以每个元素为起始，右指针向右探测子数组的和是否满足条件。
   返回值类型：int 类型, 返回和为 k 的连续子组的个数

3) (3) 函数名称：sortlist
   函数功能：将 L 由小到大排序或从大到小排序；
   思路：采用的是冒泡排序的方法，时间复杂度 O（n^2）
   返回值类型：status

1.2.5 文件及多表操作函数算法设计

1.2.3 中基础操作函数算法设计主要是对单表基础操作提供便利。在此基础上，系统还需扩展文件操作及多表操作，依靠以上基础操作无法完成，则需要构建文件及多表操作函数来完成相关操作。具体函数算法设计如下。

（1）文件保存 status SaveList(SqList L, char *FileName) 使用文件指针用 “w” 模式打开目标文件，循环写入线性表的数据。

（2）文件读取 status LoadList(SqList& L, char *FileName) 使用文件指针用 “r” 模式打开目标文件，循环读取线性表的数据并创建线性表。该函数中含有一个循环结构，用于读取文件中的数据到顺序表中，执行 n 次，因此时间复杂度为 O(n)。

1.2.6 异常情况的处理

由于输入内容具有不可预见性，因此，每次读取输入操作时候都应该对其进行相应的处理，若输入符合输入要求，则进行正常的操作；反之，则提示用户输入不合法并要求用户重新输入。

对每一个用户输入都应该执行上述操作以保证程序得到正确的用户输入，从而保证程序能够正确稳定的运行。例如在函数的返回值上，如果线性表不存在，进行对线性表求表长或插入元素，函数会返回 INFEASIBLE，如果查找不到某一元素的前驱会返回 ERROR。

1.3 系统实现

1.3.1 实现方案与操作环境

本系统使用 c 语言实现，实验源文件保存在 u1.cpp 文件中，生成目标程序名为 u1.exe。

完成本实验的操作系统为 Windows 11（64 位），使用 visual studio code 进行编写代码、调试及功能测试。

1.3.2 多表操作的实现

多表管理表结构体的示意图以及顺序表元素在内存中的存储状态如图所示，图中每一行元素表示管理一个顺序表的相关元素，顺序表 L 的第一个元素 elem 指向顺序表的元素存储空间。

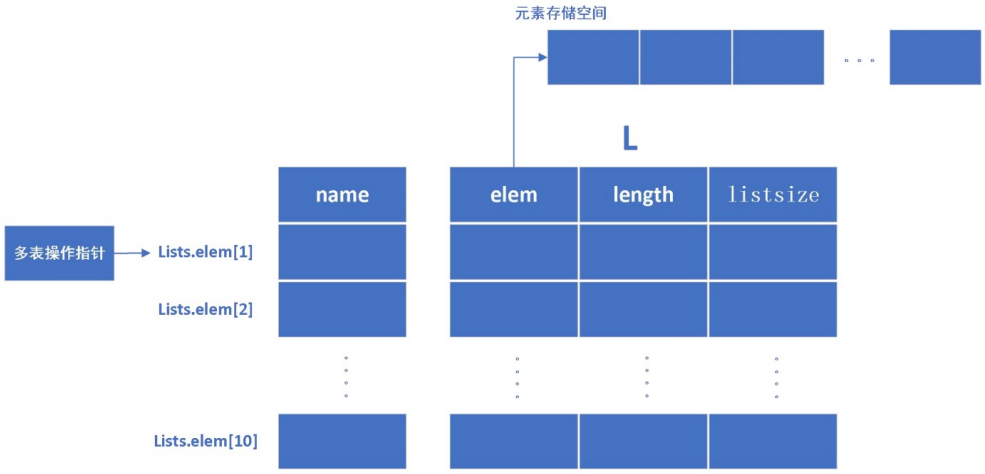


图 1-5 多表管理的存储结构示意图

1.3.3 基本函数功能的实现

1) InitList

该函数接受一个 SqList 类型的指针，为该指针指向的线性表分配 LIST_INIT_SIZE 的存储空间，并将该线性表的当前长度置为 0，总长度置为 LIST_INIT_SIZE。若存储空间分配成功，则返回 OK；若存储空间分配失败，则返回 INFEASIBLE。

2) DestroyList

该函数接受一个 SqList 类型的指针，函数释放指针指向的线性表的数据存储空间，并将指针 elem 置为 NULL，并返回 OK。如果线性表不存在，则返回 INFEASIBLE。

3) ClearList

该函数接受一个 SqList 类型的指针，函数将指针指向的线性表的长度 length 置为 0，并返回 OK。如果线性表不存在，则返回 INFEASTABLE。

4) ListEmpty

该函数接受一个 SqList 类型的值，函数判断 SqList 线性表的长度 length 是否为 0。若为 0，说明线性表为空，函数返回 TRUE；若不为 0，说明线性表不为空，函数返回 FALSE。如果线性表不存在，则返回 INFEASIBLE。

5) ListLength

该函数接受一个 SqList 类型的值，函数返回 SqList 线性表的当前长度 length。如果线性表不存在，则返回 INFEASIBLE。该函数只有判断和顺序结构，因此其时间复杂度为 O(1)。

6) GetElem

该函数接受三个参数，第二个参数用来获取位置，第三个参数用来储存查找的元素，若查到函数，则将查找到的元素存储到 e 中，并返回 OK。若查找位置没有元素，则返回 ERROR。如果线性表不存在，则返回 INFEASIBLE。

7) LocateElem

该函数接受两个参数，第二个参数 e 为待查找元素。循环操作遍历整个线性表，逐个比较，若找到第一个与 e 相等的元素，则返回该元素的位置；若遍历完整个线性表仍未找到该元素，则返回 ERROR。如果线性表不存在，则返回 INFEASIBLE

8) NextElem

该函数接受三个参数，第二个参数为需要查找前驱的元素；第三个参数储存找到的前驱。首先判断需要查找前驱的元素是否为第一个元素，若是，则返回 ERROR；否则，函数遍历线性表查找该元素的位置，若找到，则将其前一个位置的值赋给 pre，并返回 OK；若未找到，则返回 ERROR。如果线性表不存在，则返回 INFEASIBLE。

9) PriorElem

该函数接受三个参数，第二个参数为需要查找后继的元素；第三个参数储存找到的后继。首先判断需要查找后继的元素是否为最后一个元素，若是，则返回 ERROR；否则，函数遍历线性表查找该元素的位置，若找到，则将其后一个位置的值赋给 next，并返回 OK；若未找到，则返回 ERROR。如果线性表不存在，则返回 INFEASIBLE。

10) ListInsert

该函数接受三个参数，第二个参数为插入元素的位置；第三个元素为待插入元素。函数首先判断插入位置是否合法，若插入位置小于 1 或大于表长 length+ 则返回 ERROR。判断是否在最后一个位置插入元素，若是则直接在最后添加元素，将表长 length 加 1，并返回 OK；否则，函数从最后一个元素开始，将待插入位置及其后的所有元素向后移动一个单位，然后将待插入元素存储到待插入的位置，表长 length 加 1，并返回 OK。如果线性表不存在，则返回 INFEASIBLE。

11) ListDelete

该函数接受三个参数，第二个参数为删除元素的位置；第三个参数用于储存删除的元素。首先，若该位置小于 1 或大于表长 length，则返回 ERROR；否则，将删除的元素赋给 e，从该位置开始将其后所有元素向前移动一个单位，并将线性表的当前长度 length 减 1，返回 OK。如果线性表不存在，则返回 INFEASIBLE。

12) ListTraverse

该函数接受表 L，函数遍历线性表 L 的每一个元素，并将其逐个输出，并返回 OK。如果线性表不存在，则返回 INFEASIBLE。

1.3.4 附加功能的实现

1) Maxsubarray

初始化当前最大子序列和为负无穷大，当前连续子序列和为 0。遍历顺序表中的每一个元素，将当前连续子序列和加上这个元素的值。如果当前连续子序列和大于当前最大子序列和，更新当前最大子序列和为当前连续子序列和。如果当前连续子序列和变成了负数，说明这个子序列不可能是最大子序列的一部分，将当前连续子序列和清零。最后返回最大子序列和。

2) subarraynum

初始化计数器为 0，以及当前子序列的和为 0。如果顺序表为空，返回错误代码。遍历顺序表中的每一个元素作为子序列的起始点 i，再从 i 开始遍历顺序表中的每一个元素作为子序列的终止点 j。计算从 i 到 j 这段子序列的和 sum，如果 sum 等于 k，说明这个子序列满足条件，将计数器加一。将当前子序列的和 sum 清零，开始寻找下一个子序列。最后返回计数器的值，即

和为 k 的子序列的个数。

3) sortlist

使用冒泡排序实现线性表的排序。

1.3.5 文件操作功能的实现

1) Loadfile

实现步骤：首先给表 L 分配空间，将表长 length 置为 0。然后以只读模式“r”打开，若目标文件为空，则返回 INFEASIBLE，退出函数。目标文件不为空，则根据空格数读取表长 length。重置 fp 到文件头。从文件中每次读取 Elemtype 类型长度的数据并将其写入到线性表中，然后关闭文件，并返回 OK。如果线性表不存在，则返回 INFEASIBLE。

2) LoadList

该函数接受两个参数，第二个参数为目标文件的文件名。首先给表 L 分配空间，将表长 length 置为 0。然后以只读模式“r”打开，若目标文件为空，则返回 INFEASIBLE，退出函数。目标文件不为空，则根据空格数读取表长 length。重置 fp 到文件头。从文件中每次读取 Elemtype 类型长度的数据并将其写入到线性表中，然后关闭文件，并返回 OK。如果线性表不存在，则返回 INFEASIBLE。

1.4 系统测试

程序采用简易界面，对 12 基本功能、3 个附加功能、2 个文件操作和多表选择操作进行测试。

程序的菜单界面如下图1-6所示。

图 1-6 菜单界面

测试样例 TEST= {-2 1 -3 4 -1 2 1 -5 4}

1.4.1 基本函数功能的测试

初始化、清空、销毁的测试

因为这三个操作都较简单，所以测试结果合并在一个表格里展示。

表 1-1 初始化、清空、销毁测试

操作类型	输入	理论结果	运行结果
初始化	1	初始化成功	请选择你的操作[0--18]: 1 请输入要保存的线性表名称 test 线性表创建成功
清空 (已初始化)	3	清空成功	请选择你的操作[0--18]: 3 线性表重置成功!
清空 (未初始化)	3	线性表不存在	请选择你的操作[0--18]: 3 线性表不存在!
销毁 (已初始化)	2	线性表销毁成功	请选择你的操作[0--18]: 2 销毁线性表成功!
销毁 (未初始化)	2	线性表不存在	请选择你的操作[0--18]: 2 线性表不存在!

求表长的测试

测试情况 1：顺序表已初始化，返回正确表长。

测试情况 2：顺序表未初始化，提示线性表不存在。

表 1-2　求表长测试

操作类型	输入	理论结果	运行结果
TEST（已初始化）	5	9	请选择你的操作[0--18]: 5 线性表表长为9
正常操作（未初始化）	5	线性表不存在	请选择你的操作[0--18]: 5 线性表不存在!

判空的测试

测试情况 1：顺序表已初始化，非空表，显示不是空表。

测试情况 2：顺序表已初始化，是空表，显示是空表。

测试情况 2：顺序表未初始化，显示是不存在。

表 1-3　线性表判空测试

操作类型	输入	理论结果	运行结果
非空表	4	线性表不是空表	请选择你的操作[0--18]: 4 线性表不是空表!
空表	4	线性表是空表	请选择你的操作[0--18]: 4 线性表为空!
null	4	线性表不存在	请选择你的操作[0--18]: 4 线性表不存在!

获取元素的测试

测试情况 1：顺序表已初始化，输入位置合法，显示在该位置的元素。

测试情况 2：顺序表已初始化，输入位置不合法，显示输入未知错误。

测试情况 2：顺序表未初始化，显示是不存在。

表 1-4 getelem 测试

操作类型	输入	理论结果	运行结果
在线性表中存在	6 3	第三个结点的元素是 3	请选择你的操作[0--18]: 6 请输入要取结点的位置: 3 第3个结点的元素是: -3
输入位置不合法	6	输入位置错误	请选择你的操作[0--18]: 6 请输入要取结点的位置: 12 输入位置错误!
null	6	线性表不存在	请选择你的操作[0--18]: 6 线性表不存在!

定位元素的测试

测试情况 1：顺序表已初始化，元素在表中存在，显示在该元素的位置。

测试情况 2：顺序表已初始化，元素在表中不存在，显示不存在。

测试情况 3：顺序表未初始化，显示是线性表不存在。

表 1-5　locateelem 测试

操作类型	输入	理论结果	运行结果
输入元素在线性表中存在	7 4	4 出现在第 4 个位置	请选择你的操作[0--18]: 7 请输入数据元素值: 4 4元素第一次出现位于第4个位置!
元素在线性表不存在	7 10	元素不存在	请选择你的操作[0--18]: 7 请输入数据元素值: 10 该元素不存在!
null	7	线性表不存在	请选择你的操作[0--18]: 7 线性表不存在!

插入元素的测试

测试情况 1：顺序表已初始化，插入位置合法，插入元素，提示插入成功。

测试情况 2：顺序表未初始化，显示是线性表不存在。

表 1-6　listinsert 测试

操作类型	输入	理论结果	运行结果
插入位置正确	10 1 1	插入成功	请选择你的操作[0--18]: 10 请输入您要插入的数据元素: 1 请输入您要插入的数据元素的位置: 1 插入数据元素成功!
null	10	线性表不存在	请选择你的操作[0--18]: 10 线性表不存在!

求前驱元素的测试

测试情况 1：顺序表已初始化，元素在表中存在且有前驱元素，显示该元素的前驱。

测试情况 2：顺序表已初始化，元素在表中不存在，显示不存在。

测试情况 3：顺序表未初始化，显示是线性表不存在。

表 1-7 priorelem 测试

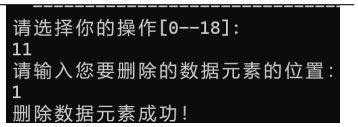
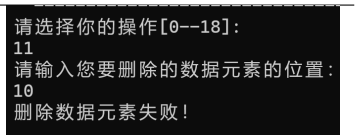
操作类型	输入	理论结果	运行结果
元素存在且有前驱	8 4	前驱元素是-3	请选择你的操作[0--18]: 7 请输入数据元素值: 4 4元素第一次出现位于第4个位置!
没有该元素	8	线性表没有该元素	请选择你的操作[0--18]: 7 请输入数据元素值: 10 该元素不存在!
null	8	线性表不存在	请选择你的操作[0--18]: 7 线性表不存在!

删除元素的测试

测试情况 1：顺序表已初始化，删除位置合法，删除元素，提示删除成功。

测试情况 2：顺序表未初始化，显示是线性表不存在。

表 1-8　listdelete 测试

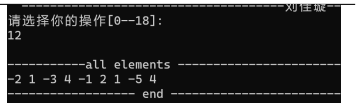
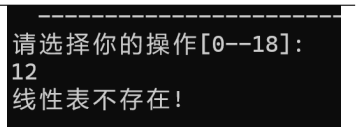
操作类型	输入	理论结果	运行结果
删除位置正确	11 1	删除元素成功	
删除元素不合法	11 10	输入位置错误	

遍历线性表的测试

测试情况 1：顺序表已初始化，非空表，答应遍历结果

测试情况 2：顺序表未初始化，显示是线性表不存在。

表 1-9　listtraverse 测试

操作类型	输入	理论结果	运行结果
非空表	12	-2 1 -3 4 -1 2 1 -5 4	
null	12	线性表不存在	

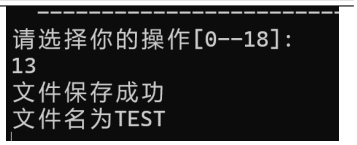
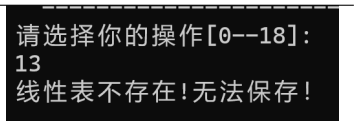
1.4.2 文件操作函数功能的测试

文件保存的测试

测试情况 1：顺序表已初始化，保存到文件中，插入保存成功。

测试情况 2：顺序表未初始化，显示是线性表不存在，无法保存。

表 1-10 文件保存测试

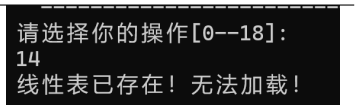
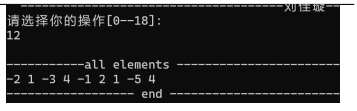
操作类型	输入	理论结果	运行结果
已初始化	13	文件保存成功	
null	13	线性表不存在	

读取文件的测试

测试情况 1：线性表已存在，提示读取失败

测试情况 2：线性表不存在，从文件中加载线性表，提示加载成功

表 1-11 文件读取测试

操作类型	输入	理论结果	运行结果
线性表已存在	14	线性表存在无法加载	
线性表不存在	14 12（遍历）	线性表加载成功	

1.4.3 附加函数功能的测试

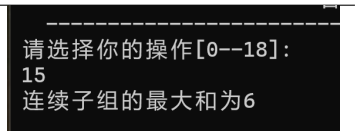
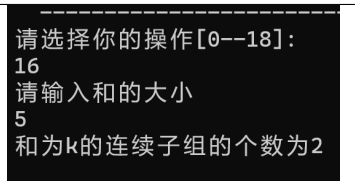
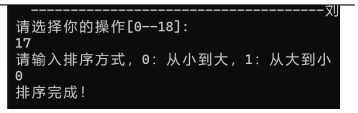
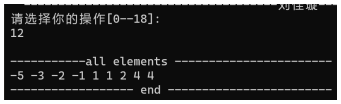
附加功能的测试

附加功能 1：Maxsubarray, 返回最大的连续子数组的和

附加功能 2：Subarraynum，输入一个整数，返回连续的子数组和为此的个数

附加功能 3：Sortlist，将线性表从大到小或从小到大排序

表 1-12 文件读取测试

操作类型	输入	理论结果	运行结果
Maxsubarray	15	6	
Subarraynum	16 5	2	
Sortlist	17 12（遍历）	排序成功	 

1.5 实验小结

本次实验主要内容是关于线性表的练习，由于实验之前老师已给出基础框架，只需对实验中要求的函数进行补充，这减小了我们的学习压力，更能突出对课程内容的考查与训练。

在本次实验中，我学会了从整体到局部的逻辑思维方式。首先，构建系统整体框架，然后在此框架中填补所需内容，例如所需函数及相关定义等。再者，我也学会了分级解决目标问题的方法，将系统分级化，方便管理是其一，其二则是编写目的更加明确化。最后则是一化多的方法，将目标问题分成若干子问题来进行解决，如总系统分为单表操作和多表操作，多表操作中又包含单独操作，层层细化，为问题解决提供了不少便利。

于此同时也遇到了很多的困难和问题，在解决问题和不断完善解决方案的功能的过程中，我也有了很多的收获。在设计函数时，要考虑多种可能的情况，返回不同的值，提升程序设计的健壮性。对于不同的操作结果，也要在终端输出提示性语句，从而使用户体验更好。

其次，我对传引用 & 不太熟悉，导致有的函数调用时也加了 & 导致报错，通过实验我提高了对函数参数调用的熟练程度

最后，本次试验中的对文件的操作也让我对 C 语言程序中文件的读写有了更加深入的理解.

2 基于邻接表的图实现

2.1 问题描述

物理结构为邻接表，创建一个简易菜单的功能演示界面。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，并给出了相应的操作提示。也可选择以文件的形式进行存储和加载，整个系统在主程序中完成函数调用。

创建图、销毁图、查找顶点、获得顶点值和顶点赋值等 13 种基本运算的具体运算功能定义如下：

1) 创建图：函数名称是 CreateCraph(&G,V,VR)；初始条件是 V 是图的顶点集，VR 是图的关系集；操作结果是按 V 和 VR 的定义构造图 G。
2) 销毁图：树函数名称是 DestroyBiTree(T)；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 和 G 中的顶点具有相同特征；操作结果是若 u 在图 G 中存在，返回顶点 u 的位置信息，否则返回其它信息。
4) 获得顶点值：函数名称是 GetVex (G,v)；初始条件是图 G 存在，v 是 G 中的某个顶点；操作结果是返回 v 的值。
5) 顶点赋值：函数名称是 PutVex (G,v,value)；初始条件是图 G 存在，v 是 G 中的某个顶点；操作结果是对 v 赋值 value。
6) 获得第一邻接点：函数名称是 FirstAdjVex(&G, v)；初始条件是图 G 存在，v 是 G 的一个顶点；操作结果是返回 v 的第一个邻接顶点，如果 v 没有邻接顶点，返回“空”。
7) 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 是 G 的一个顶点,w 是 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，如果 w 是最后一个邻接顶点，返回“空”。
8) 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。
9) 删除顶点：函数名称是 DeleteVex(&G,v)；初始条件是图 G 存在，v 是 G 的一个顶点；操作结果是在图 G 中删除顶点 v 和与 v 相关的弧。

10) 插入弧：函数名称是 InsertArc(&G,v,w)；初始条件是图 G 存在，v、w 是 G 的顶点；操作结果是在图 G 中增加弧 <v,w>，如果图 G 是无向图，还需要增加 <w,v>。
11) 删除弧：函数名称是 DeleteArc(G,v,w)；初始条件是图 G 存在，v、w 是 G 的顶点；操作结果是在图 G 中删除弧 <v,w>，如果图 G 是无向图，还需要删除 <w,v>。
12) 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。
13) 广深度优先搜索遍历：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。

对于文件管理以及多图管理，同样需要以函数形式扩展文件保存与读取操作以及多图管理操作的基本操作，具体操作功能定义如下。

1) 保存文件:：函数名称是 SaveGraph(G, FileName)；初始条件是图 G 存在；操作结果是图 G 保存到名字为 FileName 的文件中。
2) 读取文件：函数名称是 LoadGraph(G, FileName)；初始条件是文件存在；操作结果是名字为 FileName 的文件读取到图 G 中。

对于附加功能，函数要求如下。

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；

2.2 系统设计

2.2.1 系统总体设计框架

物理结构为邻接表，创建一个简易菜单的功能演示界面。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 13 种基本运算，并给出了相应的操作提示。也可选择以文件的形式进行存储和加载，整个系统在主程序中完成函数调用，以及程序的退出。

2.2.2 有关常量、数据类型的定义

数据元素类型的定义：

```
typedef int status;
typedef int KeyType;
typedef enum {DG,DN,UDG,UDN} GraphKind;
typedef struct {
        KeyType  key;
        char others[20];
} VertexType; //顶点类型定义

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

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

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

有关常量的定义：

```
#define TRUE 1
#define FALSE 0
#define INFEASIBLE -2
#define MAX_VERTEX_NUM 20
#define graphnum 10
#define MAX_VERTEX_NUM 20
#define MAX_ARC_NUM 40
#define ERROR -1
#define OK 1
#define OVERFLOW -2
```

2.2.3 多图操作的设计

采用数组存储多个图，数组中的元素的数据类型是 ALGraph，通过改变 i_num 的值选择操作的图。多图的示意结构如下图所示。

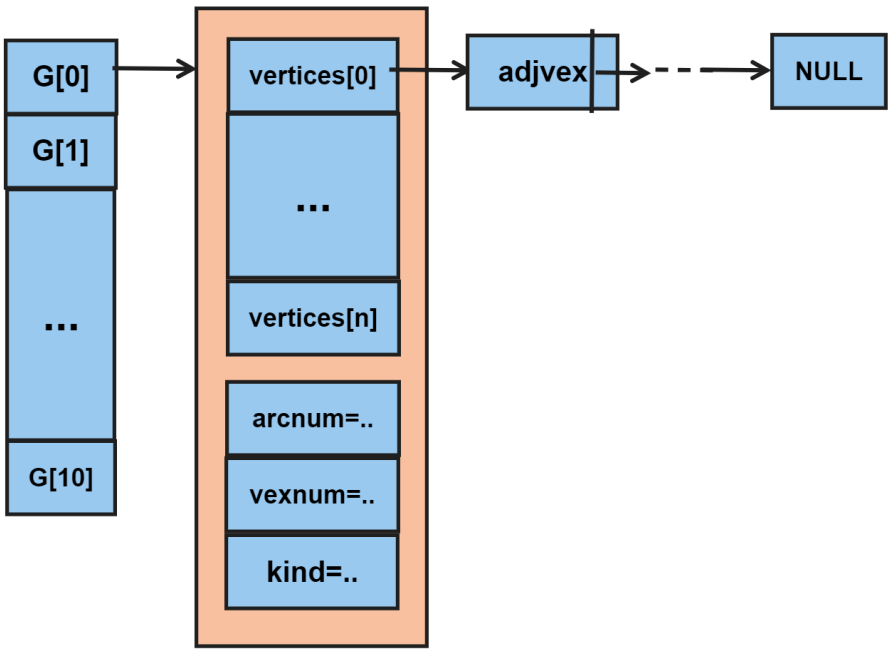


图 2-1 多图管理的存储结构示意图

2.2.4 基本操作功能算法设计

1) 函数名称：CreateGraph(G)

操作结果：构造图 G

算法思路：首先遍历读取图的所有节点存储在数组中，然后找到每条弧的两端节点，利用头插法插入图的邻接表。

时间复杂度：遍历数组验证是否有重复关键字，时间复杂度 $0(n^2)$, 每次根据弧的顶点找到对应结点，时间复杂度 $0(n)$, 总的时间复杂度为 $0(n^2)$

2) 函数名称：DestroyGraph(G)

初始条件：图 G 已存在

操作结果：销毁图 G

算法思路：遍历图，依次释放存储数据元素的空间，最后将图的几个顶点数、弧数和种类均置为零，完成销毁。

时间复杂度：该函数从每个顶点开始释放弧数据，遍历每个顶点和每个弧数据，因此函数的时间复杂度为 O(n+m)。

3) 函数名称：PutVex (G,v,value)

初始条件：图 G 存在，v 是 G 中的某个顶点

操作结果：对 v 赋值 value

算法思路：调用 LocateVex(G,u) 函数，并将找到的顶点的值修改为所给值。

时间复杂度：调用 LocateVex(G,u) 函数循环查找顶点位序，并比较是否与其他关键字冲突，因而遍历所有顶点，时间复杂度为 O(n)，而该顶点赋值函数为简单的顺序结构，因此其总的时间复杂度为 O(n)。

4) 函数名称：LocateVex(G,u)

初始条件：图 G 存在，u 和 G 中的顶点具有相同特征

操作结果：若 u 在图 G 中存在，返回顶点 u 的位置信息，否则返回其它信息

算法思路：遍历图，当图中顶点的值与所给信息的值相同时返回该顶点的 key，完成定位。

时间复杂度：最好的情况为第一次就比较成功，只循环 1 次；最坏的情况为最后一次比较成功或者查找失败，需要循环 n 次，因此函数的时间复杂度

为 O(n)。

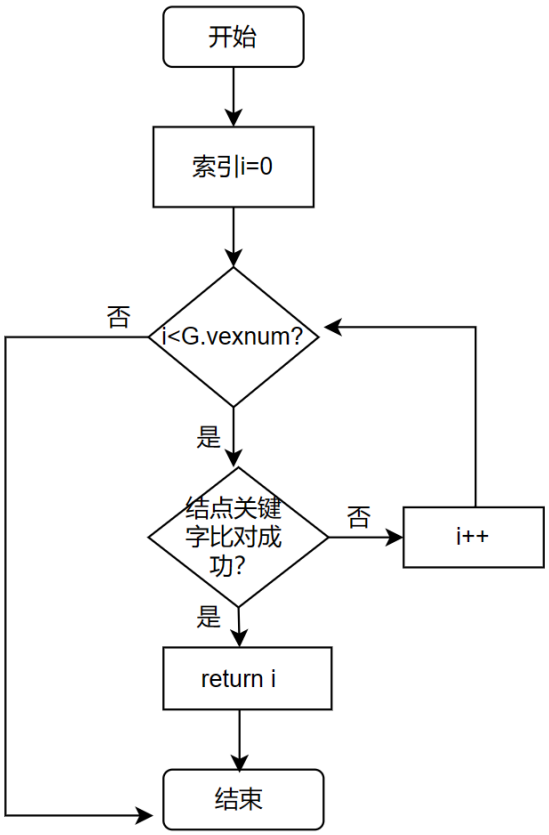


图 2-2　查找结点流程图

5) 函数名称：FirstAdjVex(&G, v)

初始条件：图 G 存在，v 是 G 的一个顶点

操作结果：返回 v 的第一个邻接顶点，如果 v 没有邻接顶点，返回“空”

算法思路：调用 LocateVex(G,u) 函数，并返回找到的顶点的第一个邻接顶点。

时间复杂度：该函数调用 LocateVex 函数，时间复杂度为 O(n)，此获取第一邻接点, 函数为简单的顺序结构，故函数总的时间复杂度为 O(n)。

6) 函数名称：NextAdjVex(&G, v, w)

初始条件：图 G 存在，v 是 G 的一个顶点,w 是 v 的邻接顶点

操作结果：返回 v 的（相对于 w）下一个邻接顶点，如果 w 是最后一个邻接顶点，返回“空”

算法思路：调用 LocateVex(G,u) 函数，并返回找到的顶点相对所给的邻接顶

点的下一个邻接顶点。

时间复杂度：该函数调用 LocateVex 函数，时间复杂度为 O(n)，而该函数中循环查找 v 结点的邻接顶点的平均查找次数为 n/2，因此总的时间复杂度为 O(n)。

7) 函数名称：InsertVex(&G,v)

初始条件：图 G 存在，v 和 G 中的顶点具有相同特征

操作结果：在图 G 中增加新顶点 v

算法思路：输入该顶点的值，插入新结点并对其赋值。

时间复杂度：该函数调用 LocateVex 函数，时间复杂度为 O(n)，其他为顺序结构，因此其时间复杂度为 O(n)。

8) 函数名称：DeleteVex(&G,v)

初始条件：图 G 存在，v 是 G 的一个顶点

操作结果：在图 G 中删除顶点 v 和与 v 相关的弧

算法思路：调用 LocateVex(G,u) 函数，找到该顶点，删除与该顶点相关的所有边以及该顶点。

时间复杂度：该函数调用 LocateVex 函数，时间复杂度分别为 O(n)，而其他部分最好的情况为只遍历 1 个顶点，最坏的情况为遍历每一个结点以及每一条弧，故总的时间复杂度为 O(n+m)（n 为顶点数，m 为弧数）。

9) 函数名称：InsertArc(&G,v,w)

初始条件：图 G 存在，v、w 是 G 的顶点

操作结果：在图 G 中增加弧 <v,w>，如果图 G 是无向图，还需要增加 <w,v>

算法思路：调用 LocateVex(G,u) 函数，找到相应顶点，增添对应的弧 <v,w> 以及 <w,v>。

时间复杂度：该函数调用 LocateVex 函数，时间复杂度分别为 O(n)，而其他部分除需要遍历 v 的邻接顶点外，为顺序结构。在查找邻接结点时为循环结构，最好的结果是循环 1 次，最坏结果为循环 n 次，时间复杂度为 O(n)。故总的时间复杂度为 O(n)。

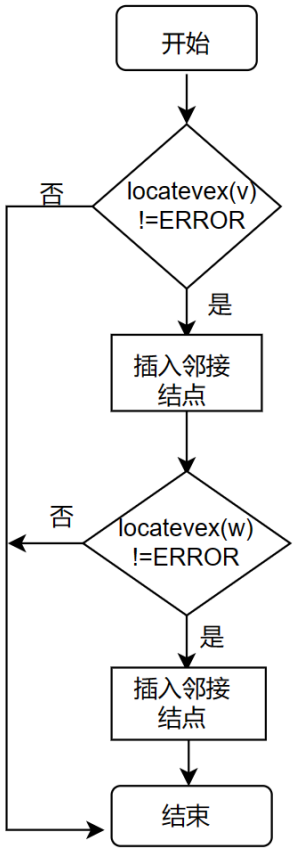


图 2-3　插入弧流程图

10) 函数名称：DeleteArc(&G,v,w)

初始条件：是图 G 存在，v、w 是 G 的顶点

操作结果：在图 G 中删除弧 <v,w>，如果图 G 是无向图，还需要删除 <w,v>

算法思路：调用 LocateVex(G,u) 函数，找到相应顶点，删除对应的弧 <v,w> 以及 <w,v>。

时间复杂度：该函数调用 LocateVex 函数，时间复杂度为 O(n)，而遍历两个顶点的邻接顶点，最好的情况为只循环 1 次，最坏的情况为循环 2n 次，因此函数的时间复杂度为 O(n)。

11) 函数名称：DFSTraverse(G,visit())

初始条件：图 G 存在

操作结果：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

算法思路：首先，访问出发顶点 v 并作访问标记；然后，依次从 v 的未访问过的邻接顶点 w 出发，进行深度优先遍历。

时间复杂度：该函数对无向图每个顶点调用 visit 函数，函数 visit 的时间复杂度为 O(1)，故总的时间复杂度为 O(n)。

12) 函数名称：BFSTraverse(G,visit())

初始条件：图 G 存在

操作结果：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

算法思路：首先，访问出发顶点 v 并作访问标记；然后，依次访问 v 的所有未访问过的邻接顶点 w，再依次广度优先遍历 w 的所有未访问过的邻接顶点，直到出发顶点 v 的所有可达顶点均被访问过为止。

时间复杂度：该函数对无向图每个顶点调用 visit 函数，函数 visit 的时间复杂度为 O(1)，故总的时间复杂度为 O(n)。

2.2.5 附加功能函数设计

1) 函数名称：VerticesSetLessThanK(G, v, k, result, count)

初始条件：图 G 存在，顶点 v 存在于图 G 中，k 为非负整数，result 和 count 为有效的输出参数。

操作结果：在图 G 中，找到所有与顶点 v 的距离小于 k 的顶点，并将这些顶点存储在数组 result 中，同时将符合条件的顶点个数存储在变量 count 中。

算法思路：使用 BFS，将所有节点离目标节点 v 的距离保存到 dist 数组中。最后遍历 dist 数组，把距离小于 k 的保存在 result 数组中，同时增加 count。

时间复杂度：该函数通过 BFS 遍历了图中的所有顶点和边，时间复杂度为 O(n+e)，其中 n 为顶点数，e 为边数。

2) 函数名称：ShortestPathLength(G, v, w)

初始条件：图 G 存在，顶点 v 和 w 存在于图 G 中。

操作结果：计算图 G 中从顶点 v 到顶点 w 的最短路径长度，并返回该长度值。

算法思路：使用 BFS, 找到 v 和 w 间的最短路径。

时间复杂度：该函数通过 BFS 遍历图 G，时间复杂度为 O(n+e)，其中 n 为

顶点数，e 为边数。

3) 函数名称：ConnectedComponentsNums(G)
初始条件：图 G 存在。
操作结果：返回图 G 的所有连通分量的个数。
算法思路：求连通分量个数的问题, 也就是构建图的生成树的个数的问题, 使用 BFS 或 DFS 都是可行的。这里使用 DFS, 每次调用 DFS 函数时，都会访问一个新的连通分量，因此每次调用 DFS 函数后，将计数器 count 加 1。最后 count 的个数就是连通分量的个数。

2.2.6 文件操作函数设计

1) 函数名称：SaveGraph(G, FileName[])
初始条件：图 G 存在。
操作结果：将图 G 的数据写入到文件 FileName 中。
算法思路：遍历图的所有顶点，对每个顶点, 遍历它的所有邻接结点, 并将相关数据写入文件。时间复杂度：该函数需要遍历图的所有顶点，并将相关数据写入文件。因此，时间复杂度取决于图的顶点数和邻接点数，假设图有 V 个顶点和 E 条边，则时间复杂度为 O(V+E)。

2) 函数名称：LoadGraph(G, FileName[])
初始条件：图 G 不存在。
操作结果：从文件 FileName 中读取图的数据，并创建图的邻接表。
算法思路：根据文件中的结点和邻接表中的邻接结点创建图, 注意需要用栈倒置头插法插入, 保证读取后顶点邻接表顺序与原图相同。
时间复杂度：该函数需要读取文件中的数据，并根据数据创建图的邻接表。假设图有 V 个顶点和 E 条边，则时间复杂度为 O(V+E)。

2.3 系统实现

2.3.1 实现方案与操作环境

本系统使用 C 语言实现，实现方案包括一个头文件（func.h）和一个源文件（main.c）。头文件 func.h 中定义了常量和数据类型，构建了无向图的结构，声明了对无向图操作的相关函数，包括 12 个基础单图操作函数、2 个文件操作及队列数据结构的常用函数。源文件 main.c 中通过使用各定义和调用各函数来对单图及多图进行各种操作，实现方式是用一个 switch 结构，根据输入的数字，执行不同的语句，进而调用不同的函数。完成本实验的操作系统为 Windows 11（64 位），使用 visual studio code 进行编写代码、调试及功能测试。

2.3.2 多图操作的实现

利用邻接表进行多图操作, 图的索引数保存在变量 i_num 中. 在选择 choose 时, 可以改变 i_num 的值来改变操作的图的序号数。

举例如下：

V={{5，"线性表"}，{8，"集合"}，{7，"二叉树"}，{6，"无向图"}};

VR={{5，6}，{5，7}，{6，7}，{7，8}};

对应算法生成的邻接表如下图所示：

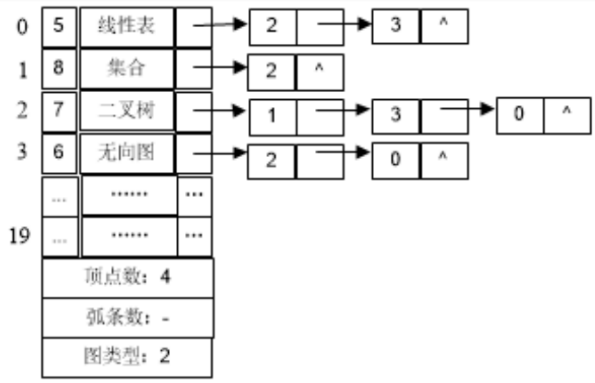


图 2-4 多图邻接表

2.3.3 基本函数功能的实现

1) 创建图 status CreateCraph(ALGraph& G, VertexType V[], KeyType VR[][2])

定义 vexnum=0,arcnum=0, 分别记录顶点和边的数目。如若当前顶点序列

的关键字不为-1 执行 while 循环：如果当前关键字未出现则更新标记数组并继续，否则返回 ERROR，在邻接表添加新顶点，令表头结点为 NULL，更新顶点数，检查是否超过最大数目 MAXVERTEXNUM ，超过则返回 ERROR。循环结束如果 vexnum=0, 即没有顶点，则返回 ERROR，否则令 G.vexnum=vexnum。当前关系序列不为 (-1,-1) 时执行 while 循环：用 for 循环遍历邻接表，查找关系序列相应顶点，插入相应的弧。

2) 销毁图 status DestroyGraph(ALGraph& G)

该函数接受一个 ALGraph 类型的图 G 邻接表。该函数先创建 ArcNode 类型的两个指针 p、q 用于清除表结点。外层循环用于访问每个顶点，内层循环从每个结点的 firstarc 开始遍历后续数据，并将其 free。最后将图 G 的顶点数和弧数置为 0，函数返回 OK，表明无向图被释放。

3) 查找结点 int LocateVex(ALGraph G, KeyType u)

设置一个 ArcNode 指针，用 for 循环遍历邻接表，如果当前顶点关键字与所找关键字相等，则返回当前顶点序号。如果没找到，返回-1。

4) 结点赋值 status PutVex(ALGraph& G, KeyType u, VertexType value)

借用 LocateVex 函数实现，最后将找到的节点赋值。

5) 查找首个邻接点 int FirstAdjVex(ALGraph G, KeyType u)

借用 LocateVex 函数实现，最后返回找到的节点的首个邻接节点。

6) 查找下一邻接点 int NextAdjVex(ALGraph G, KeyType v, KeyType w)

该函数接受一个 ALGraph 类型的参数 G 和两个 KeyType 类型的关键字 v 与 w。该函数分别两次调用 LocateVex 函数，查找 v 与 w 分别对应的位序，如果有顶点未查找成功，则函数返回-1。查找成功则在 v 对应顶点后的表链表中查找是否有 w 对应的顶点，查找成功，则返回 w 后的下一个邻接点的位序，反之返回 -1。

7) 插入结点 status InsertVex(ALGraph& G, VertexType v)

该函数接受一个 ALGraph 类型的参数 G 和 VertexType 类型的待插入顶点 v。首先调用 LocateVex 函数，判断待插入顶点 v 的关键字是否与已有关键字重复，若重复则返回 ERROR ；不重复则判断图 G 的顶点数是否达到最大值 MAX_VERTEX_NUM，若以达到则返回 ERROR；满足上述两种条件后，将待插入顶点 v 插入到表的末端位置，顶点数增 1，返回 OK。

8) 删除结点 status DeleteVex(ALGraph& G, KeyType v)

该函数接受一个 ALGraph 类型的参数 G 和 VertexType 类型的待插入顶点 v。首先调用 LocateVex 函数，查找待删除顶点位序，若查找失败或图 G 的顶点数为 0 或 1 则返回 ERROR；查找成功则创建指针 p 指向 v 的第一条弧，q 指向 p 的 nextarc，接着创建 save 指针指向第一条弧连接的顶点的表链表表头，遍历查找与 v 相连的结点，将其删除。接着 free 掉 p，将下一个结点赋给 p，进行新一轮的循环。当删除掉图 G 中与 v 连接的弧和顶点 v 后，将邻接表顶点 v 后的位序进行调整，然后遍历每一个顶点的弧链表，将存在 v 之后的位序数减一，最后返回 OK。

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

该函数接受三个参数，第一个为 ALGraph 类型的参数 G，v 和 w 则为 KeyType 类型的参数，为待插入弧的顶点关键字。先调用 LocateVex 查找 v 和 w 是否在图 G 中存在，若有一个顶点不存在，则返回 ERROR；若都存在，则将 p 指向 v 的弧链表的首结点，遍历弧链表每个数据，查找是否插入的弧重复，若存在重复情况，则返回 ERROR；如果弧为新弧，则将 v 和 w 的数据分别插入对应顶点的第一邻接结点中，返回 OK。

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

该函数接受三个参数，第一个为 ALGraph 类型的参数 G，v 和 w 则为 KeyType 类型的参数，为待删除弧的顶点关键字。首先调用 LocateVex 函数，查找待删除弧的两端顶点是否存在，若有顶点不存在则返回 ERROR；若两顶点都存在，则先遍历 v 顶点的邻接顶点是否存在 w，如果不存在则返回 ERROR；若存在，则将邻接信息删除，再将 w 的邻接点 v 信息删除，返回 OK。

10) 深度优先遍历 status DFSTraverse(ALGraph G, void (*visit)(VertexType))

该函数接受两个参数，第一个为 ALGraph 类型的参数 G，第二个为函数指针 visit，用于输出当前顶点。定义顶点访问状态数组 visited[100] 用于表示顶点是否被访问，将其中 G 的顶点数个数组元素初始化为 0。如果顶点未访问，则调用 dfshelper 函数，循环次数为顶点数。dfshelper 函数用于处理访问顶点顺序。先将进入函数的顶点对应的顶点参数置为 1，表示已访问，再调用 visit 函数访问该顶点；循环操作，将 v 的第一邻接点置为 w，若 w 存在且未被访问，则调用 dfshelper 函数，若已被访问，则再将 w 置为 v 相对于 w 的下一邻接点，循环操作至 w 不存在为止。

11) 广度优先遍历 status BFSTraverse(ALGraph G, void (*visit)(VertexType))

该函数接受两个参数，第一个为 ALGraph 类型的参数 G，第二个为函数指针 visit，用于输出当前顶点。定义顶点访问状态数组 visited[100] 用于表示顶点是否被访问，将其中 G 的顶点数个数组元素初始化为 0。设置队列 queue。按顶点位序依次选择顶点，当顶点 v 未被访问时，先将访问参数置为 1，再遍历该顶点，将 v 入队。循环操作当队中无数据时停止，队列出队，赋给 u，再依次访问 u 的邻接顶点，若未被访问，则输出邻接顶点，并将访问参数置为 1，将该邻接点进队。重复上述操作，当队中无数据中停止。返回 OK。

2.3.4 附加函数功能的实现

1) 距离小于 k 的结点 status VerticesSetLessThanK(ALGraph G,int v,int k,int *result,int *count);

初始化 dist 和 visited 数组，用于记录顶点之间的距离和访问状态。将顶点 v 转换为数组索引，并对 dist 和 visited 数组进行初始化。设置一个队列 queue，并将顶点 v 入队，同时将其距离设为 0，并标记为已访问。进行 BFS 遍历，通过队列不断取出顶点 u，遍历 u 的邻接顶点，更新距离和访问状态，并将未访问的邻接顶点入队。遍历 dist 数组，找到距离小于 k 的顶点，将其存储在 result 数组中，并更新 count 的值。如果 count 为 0，则返回 ERROR；否则返回 OK。

2) 最短路径 int ShortestPathLength(ALGraph G, KeyType v, KeyType w) ;

初始化 visited 数组、distance 数组和 queue 队列，用于记录顶点的访问状态、

起点到顶点的距离和 BFS 遍历。遍历图 G 中的顶点，找到顶点 v 所在的下标 i。如果找不到 v，则返回-1 表示 v 不在图 G 中。标记顶点 i 为已访问，起点到起点的距离为 0，将 i 入队。进行 BFS 遍历，从队列中取出顶点 i，遍历其邻接顶点 j。如果顶点 j 尚未被访问过，标记 j 为已访问，更新 j 到起点的距离，并将 j 入队。如果找到顶点 w，返回 w 到起点的距离值。如果队列为空，表示无法从 v 到 w 找到路径，返回-1。

3) 连通分量 int ConnectedComponentsNums(ALGraph G);

初始化一个计数器 count 为 0，用于记录连通分量的个数。初始化一个 visited 数组，用于记录每个顶点是否被访问过。初始时，所有顶点的 visited 值设为 0。对于图 G 的每个顶点，如果该顶点尚未被访问过，则进行以下步骤：a. 调用 DFS 函数进行深度优先搜索，从当前顶点出发遍历所有与之连通的顶点。b. 在 DFS 函数中，将访问过的顶点标记为已访问（visited 值设为 1）。每次调用 DFS 函数时，都会访问一个新的连通分量，因此每次调用 DFS 函数后，将计数器 count 加 1。返回计数器 count 的值，即为图 G 的连通分量个数。

2.3.5 文件操作功能的实现

1) 保存文件 status SaveGraph(ALGraph G, char FileName[]);

设置文件指针 fp 打开文件名为 FileName 的文件，遍历节点数组，指针遍历每个节点的邻接结点，将相关信息写入文件，遍历该顶点的每个邻接点，使用 fprintf 函数将邻接点的位置（adjvex）写入文件中。在邻接点写入结束后，使用 fprintf 函数写入-1 作为邻接点的结尾标识。在顶点遍历结束后，使用 fprintf 函数写入-1 和”nil” 作为顶点的结尾标识。

2) 读取文件 status LoadGraph(ALGraph&G, char FileName[]);

需要使用栈的数据结构，保证读取得到的节点顺序与原图相同。按照文件中结点的顺序读取到图中，同时改变 vexnum 和 arcnum, 最后将图的类型设为 UDG。

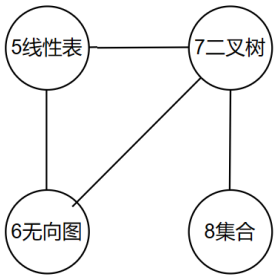
2.3.6 异常输入的处理

由于输入内容具有不可预见性，因此，每次读取输入操作时候都应该对其进行相应的处理，若输入符合输入要求，则进行正常的操作；反之，则提示用户输入不合法并要求用户重新输入。对每一个用户输入都应该执行上述操作以保证程序得到正确的用户输入，从而保证程序能够正确稳定的运行。

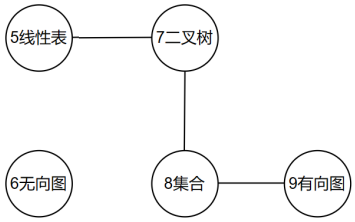
2.4 系统测试

演示系统以一个菜单作为交互界面，用户通过输入命令对应的编号来调用相应的函数来实现创建图，销毁图，清空图，顶点赋值，查找首个邻接点，查找相邻下一个邻接点，插入顶点，删除顶点，插入弧，删除弧，DFS，BFS 等基本操作，以及保存和读取文件，求最短通路，求距某顶点距离小于 d 的顶点，求连通分量等进阶操作。

测试用例 G=(V,VR)，三种图，一个连通，一个非连通，一个不存在的图。分别记为测试样例 1，测试样例 2，测试样例 3。



(a) 测试样例 1



(b) 测试样例 2

图 2-5 测试用例

菜单界面如下

```
      Menu for Undirected Graph On Chain Structure
********************************************************************
        1.  CreateGraph              2.  DestroyGraph
        3.  LocateVex                4.  PutVex
        5.  FirstAdjVex              6.  NextAdjVex
        7.  InsertVex                8.  DeleteVex
        9.  InsertArc                10. DeleteArc
        11. DFSTraverse              12. BFSTraverse
        13. SaveGraph                14. LoadGraph
        15. VerticesSetLessThanK     16. ShortestPathLength
        17. ConnectedComponentsNums  18. choose
        0.Exit
*****************************刘佳璇*************************************
        请选择你的操作[0-18]:
```

图 2-6 菜单界面

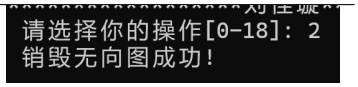
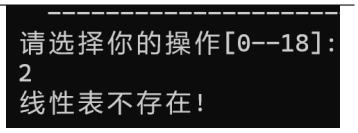
2.4.1 基本函数功能的测试

创建、销毁图的测试

因为创建、销毁图的测试比较简单，合并在同一个表格中展示。

采用测试样例 1 测试。

表 2-1 创建、销毁图的测试

操作类型	输入	理论结果	运行结果
创建图	1	创建成功	
销毁图 (已初始化)	2	销毁成功	
销毁图 (未初始化)	2	图还未创建	

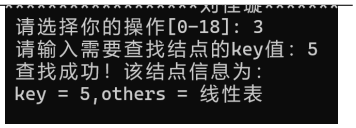
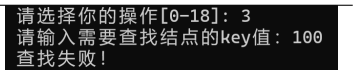
定位结点的测试

测试情况 1：采用测试样例 1 测试。查找结点存在，打印节点的信息

测试情况 2：采用测试样例 1 测试。查找结点不存在，提示查找失败

测试情况 3：采用测试样例 3 测试。图未初始化，提示图还未创建

表 2-2 定位结点的测试

操作类型	输入	理论结果	运行结果
测试样例 1（结点存在）	3 5	打印节点信息	
测试样例 1（结点不存在）	3 100	查找失败	
测试样例 3（图未初始化）	3	图还未创建	

结点赋值的测试

测试情况 1：采用测试样例 1 测试。赋值结点存在，修改节点信息

测试情况 2：采用测试样例 1 测试。修改后关键字重复，提示赋值失败

测试情况 3：采用测试样例 3 测试。图未初始化，提示图还未创建

表 2-3　结点赋值的测试

操作类型	输入	理论结果	运行结果
测试情况 1	4 5 9 线性表	赋值成功	请选择你的操作[0-18]: 4 请输入需要赋值的结点key值:5 请输入赋值后的结点key值:9 请输入赋值后的结点others值:线性表 赋值成功！现在该结点的信息为: key = 9,others = 线性表
测试情况 2	4 5 6 repeated	赋值失败	请选择你的操作[0-18]: 4 请输入需要赋值的结点key值:5 请输入赋值后的结点key值:6 请输入赋值后的结点others值:repeated 赋值失败！赋值后有重复
测试情况 3	4	图还未创建	请选择你的操作[0-18]: 4 该图还未创建！

查找第一邻接点的测试

测试情况 1：采用测试样例 1。测试查找结点的第一邻接点存在

测试情况 2：采用测试样例 2。不存在第一邻接点，提示查找失败

测试情况 3：采用测试样例 3。图未初始化，提示图还未创建

表 2-4　查找第一邻接点的测试

操作类型	输入	理论结果	运行结果
测试情况 1	5 6	打印结果	请选择你的操作[0-18]: 5 请输入要查找的顶点的key: 6 该顶点的第一个邻接顶点为: key = 7,others = 二叉树
测试情况 2	5 6	不存在	请选择你的操作[0-18]: 5 请输入要查找的顶点的key: 6 该顶点没有邻接顶点！
测试情况 3	5	图还未创建	请选择你的操作[0-18]: 5 该图还未创建！

查找下一邻接点的测试

测试情况 1：采用测试样例 1 测试。目标结点存在，打印节点信息

测试情况 2：采用测试样例 2 测试。不存在下一邻接点，提示查找失败

测试情况 3：采用测试样例 3 测试。图未初始化，提示图还未创建

表 2-5　查找下一邻接点的测试

操作类型	输入	理论结果	运行结果
测试情况 1	6 6 7	打印结果	请选择你的操作[0-18]: 6 请输入要查找的顶点 v 的key: 6 请输入和其相对的顶点 w 的key: 7 该顶点v相对于w的下一个邻接顶点为: key = 9,others = 线性表
测试情况 2	6 9 8	不存在	请选择你的操作[0-18]: 6 请输入要查找的顶点 v 的key: 9 请输入和其相对的顶点 w 的key: 8 查找失败!
测试情况 3	6	图还未创建	请选择你的操作[0-18]: 6 该图还未创建!

插入节点的测试

测试情况 1：采用测试样例 1 测试。插入后无关键字重复，插入成功

测试情况 2：采用测试样例 1 测试。插入后关键字重复，插入失败

测试情况 3：采用测试样例 3 测试。图未初始化，提示图还未创建

表 2-6　插入节点的测试

操作类型	输入	理论结果	运行结果
测试情况 1	7 6 无向图	插入成功	请选择你的操作[0-18]: 7 请输入要插入的顶点的key值: 6 请输入要插入的顶点的others值: 无向图 插入成功!
测试情况 2	7 5 repeated	插入失败	请选择你的操作[0-18]: 7 请输入要插入的顶点的key值: 5 请输入要插入的顶点的others值: repeated 关键字重复插入失败!
测试情况 3	7	图还未创建	请选择你的操作[0-18]: 7 该图还未创建!

删除节点的测试

测试情况 1：采用测试样例 1 测试。结点存在，删除成功

测试情况 2：采用测试样例 1 测试。节点不存在，删除失败

测试情况 3：采用测试样例 3 测试。图未初始化，提示图还未创建

表 2-7　删除节点的测试

操作类型	输入	理论结果	运行结果
测试情况 1	8 6	删除成功	请选择你的操作[0-18]: 8 请输入要删除的顶点的key值:6 删除成功!
测试情况 2	8 10	删除节点不存在	请选择你的操作[0-18]: 7 请输入要插入的顶点的key值: 5 请输入要插入的顶点的others值: repeated 关键字重复插入失败!
测试情况 3	8	图还未创建	请选择你的操作[0-18]: 8 该图还未创建!

删除弧的测试

测试情况 1：采用测试样例 1 测试。弧存在，删除弧

测试情况 2：采用测试样例 1 测试。弧不存在，删除失败

测试情况 3：采用测试样例 3 测试。图未初始化，提示图还未创建

表 2-8　删除弧的测试

操作类型	输入	理论结果	运行结果
测试情况 1	10 6 7	删除成功	请选择你的操作[0-18]: 10 请输入边的两个节点的关键字:6 7 删除成功!
测试情况 2	10 5 8	删除失败	请选择你的操作[0-18]: 10 请输入边的两个节点的关键字:5 8 删除失败 ,它们之间没有弧!
测试情况 3	8	图还未创建	请选择你的操作[0-18]: 10 该图还未创建!

插入弧的测试

测试情况 1：采用测试样例 1 测试。节点存在，中间无弧，插入成功

测试情况 2：采用测试样例 1 测试。节点存在，已经有弧，插入失败

测试情况 3：采用测试样例 3 测试。图未初始化，提示图还未创建

表 2-9　插入弧的测试

操作类型	输入	理论结果	运行结果
测试情况 1	9 6 8	插入成功	请选择你的操作[0-18]: 9 请输入边的两个节点的关键字:6 8 添加成功!
测试情况 2	9 5 7	插入失败	请选择你的操作[0-18]: 9 请输入边的两个节点的关键字:5 7 添加失败 ,它们之间已经有弧!
测试情况 3	8	图还未创建	请选择你的操作[0-18]: 9 该图还未创建!

深度优先遍历、广度优先遍历的测试

测试情况 1：采用测试样例 1 测试。进行深度优先遍历

测试情况 2：采用测试样例 1 测试。进行广度优先遍历

表 2-10　DFS,BFS 的测试

操作类型	输入	理论结果	运行结果
测试情况 1	11	输出结果	请选择你的操作[0-18]: 11 该无向图的深度优先搜索遍历为: key = 5,others = 线性表 key = 7,others = 二叉树 key = 6,others = 无向图 key = 8,others = 集合
测试情况 2	12	输出结果	请选择你的操作[0-18]: 12 该无向图的广度优先搜索遍历为: key = 5,others = 线性表 key = 7,others = 二叉树 key = 6,others = 无向图 key = 8,others = 集合

2.4.2　文件操作函数功能的测试

保存文件、读取文件的测试

测试情况 1：采用测试样例 1 测试。保存到“graphA.txt”文件中，提示保存成功

测试情况 2：采用测试样例 1 测试。从“graphA.txt”文件中读取，提示读取成功

表 2-11　保存文件、读取文件的测试

操作类型	输入	理论结果	运行结果
测试情况 1	13 graphA.txt	保存成功	请选择你的操作[0-18]: 13 请输入保存文件的文件名(如a.txt): graphA.txt 保存成功!
测试情况 2	14 graphA.txt	读取成功	请选择你的操作[0-18]: 14 请输入读取文件的文件名(如a.txt): graphA.txt 读取成功!

2.4.3 附加函数功能的测试

距离小于 k 的顶点集合的测试

测试情况 1：采用测试样例 1 测试。返回与顶点 5 距离小于 2 的顶点集合

测试情况 2：采用测试样例 2 测试。返回与顶点 6 距离小于 100 的顶点集合

表 2-12 距离小于 k 的顶点集合的测试

操作类型	输入	理论结果	运行结果
测试情况 1	15 5 2	打印顶点集合	请选择你的操作[0-18]: 15 请输入要查找的顶点的key值: 5 你想找的距离小于多少的顶点集合? 2 距离小于2的顶点集合为: key = 5,others = 线性表 key = 7,others = 二叉树 key = 6,others = 无向图
测试情况 2	15 6 100	打印顶点集合	请选择你的操作[0-18]: 15 请输入要查找的顶点的key值: 6 你想找的距离小于多少的顶点集合? 100 距离小于100的顶点集合为: key = 6,others = 无向图

顶点间最短路径和长度的测试

测试情况 1：采用测试样例 1 测试。返回顶点 5 和 8 之间的最短路径和

测试情况 2：采用测试样例 2 测试。顶点之间没有路径，提示两点之间没有路径

表 2-13 顶点间最短路径和长度的测试

操作类型	输入	理论结果	运行结果
测试情况 1	16 5 8	输出最短路径和	请选择你的操作[0-18]: 16 请输入要查找的两个顶点的key值,空格分隔: 5 8 两个顶点之间的最短路径长度为:2
测试情况 2	16 6 9	提示没有路径	请选择你的操作[0-18]: 16 请输入要查找的两个顶点的key值,空格分隔: 6 9 两个顶点之间没有路径!

求连通分量的测试

测试情况 1：采用测试样例 1 测试。这是连通图，返回图的连通分量个数 1

测试情况 2：采用测试样例 2 测试。这是连通分量为 2 的非连通图，返回图的连通分量个数 2

表 2-14　求连通分量的测试

操作类型	输入	理论结果	运行结果
测试情况 1	17	1	请选择你的操作[0-18]: 17 连通分量个数为:1
测试情况 2	17	2	请选择你的操作[0-18]: 17 连通分量个数为:2

2.5 实验小结

本次实验加深了对图的概念、基本运算的理解，熟练了掌握图的逻辑结构与物理结构的关系，熟练了掌握图基本运算的实现。

在关于图的函数的实现中，也需要联系之前所学的栈和队列的知识，比如广度优先搜索需要用队列实现，深度优先搜索可以用栈实现，不同的数据结构之间相互联系，也是许多更复杂算法的基础，充分体现了数据结构这门课程的重要性。在使用栈和队列实现函数的过程中，我对它们的熟悉程度也加深了。

这次实验锻炼了函数封装和函数调用的能力，比如在编写插入与删除弧的函数时，如果能够在之前定义定位顶点的函数，并调用，将极大地省去冗杂的代码，这次实验让我对函数的工具性，模块性有了直观的感受。

在 debug 的过程中，要善于从表象看到本质。比如，我在删除和插入顶点操作后进行 DFS，出现了栈溢出的情况，但是不进行删除顶点操作进行 DFS 则不会。我检查了很多遍 DFS 函数，发现并没有错误，于是终于想到是删除顶点操作出了问题。由于我在插入顶点时没有把下一邻接点指针设为空指针，导致了栈溢出。由于函数的调用关系，一个函数出错，可能会导致后续的很多操作都出现错误，所以 debug 时要细心严谨。

总的来说，本次数据结构实验提高了我的编程能力，让我对系统整体设计有了更深的认识。

3 课程的收获和建议

3.1 基于顺序存储结构的线性表实现

以数组为物理结构实现线性表算是比较简单的内容，上学期学习 C 语言时对数组已经比较熟悉。

在这次实验中，我熟悉了文件的读写操作，之前对这部分知识运用的比较少，算是一种查漏补缺。特别是判断文件指针是否已到文件尾，我的操作有一些错误，debug 时才纠正过来。我对文件读写的理解是，从 printf 到 fprintf，一个是打印到屏幕上，一个是打印到文件中，两者在使用方法上有细微差别，理解上并没有什么不同。

在附加功能最大连续子数组和的实现中，运用贪心算法实现，充分体现了“程序 = 数据结构 + 算法”的理论，让我对数据结构和算法有了更好的认识。

3.2 基于邻接表的图实现

图这个数据结构非常重要，在许多领域有着广泛的应用，比如工程中的 AOE 网、AOV 网，通信网络设计，神经网络和人工智能等。

图这个数据结构，并不好像线性表一样直观的在屏幕上打印图示，所以在实现函数时需要多演算一下，防止出错。使用邻接表进行存储时，对指针特别要注意邻接结点指针设空指针的操作，稍不注意就可能出现栈溢出的情况。

对于图的存储结构，邻接表并不见得比邻接矩阵更好，比如在对弧删除的操作中，弧查找的操作时间复杂度显然比邻接表高很多，后续的删除操作更是十分复杂。所以合理利用图的各种表示形式能够方便处理不同情况下的问题。

参考文献

[1] 严蔚敏, 吴伟民. 数据结构 (C 语言版)[M]. [S.l.]: 清华大学出版社, 2017.

[2] 严蔚敏, 吴伟民. 数据结构题集 (C 语言版)[M]. [S.l.]: 清华大学出版社, 2017.

[3] WEISS M A. 数据结构与算法分析（C 语言版）[M]. [S.l.]: 机械工业出版社, 2016.

[4] PRATA S. C Primer Plus（第 6 版）[M]. 姜佑, trans. [S.l.]: 人民邮电出版社, 2019.

代码开源声明

本报告中的实验代码已经在 github 上开源，遵循 MIT 许可协议。任何人可以免费使用、修改和分发本代码，但必须保留原作者的版权信息和许可声明。本代码的 github 仓库地址为：https://github.com/711LLL711/HUST$_d atastructure_l ab$。

附上代码仓库的截图：

代码文件目录结构分为四个子文件夹，每个文件夹存一个实验的源码、测试样例、可执行程序：

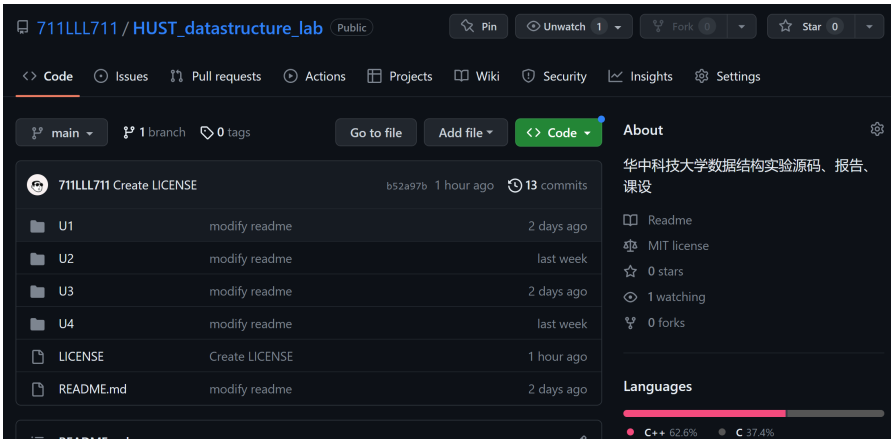


图 1　仓库代码目录结构

主页面的 readme 显示了仓库的包含内容和每个实验的链接，点击可直接跳转。

图 2　仓库 README

每个实验的子页面的 README 声明了该子页面的结构、程序数据输入格式、常量定义、数据类型定义、函数声明等。

截图展示了二叉树实验子页面的目录结构和 README 部分内容。

图 3　二叉树页面

附录 A 基于顺序存储结构线性表实现的源程序

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

/*----------page 10 on textbook ----------*/
#define TRUE 0
#define FALSE -3
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define MAX_NUM 10

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

/*-------page 22 on textbook -------*/
#define LIST_INIT_SIZE 100
#define LISTINCREMENT  10
typedef struct{  //顺序表（顺序结构）的定义
        ElemType * elem;
        int length;
        int listsize;
}SqList;

/*-----page 19 on textbook ---------
*------functiom declaration---------*/
status InitList(SqList& L);
status DestroyList(SqList& L);
status ClearList(SqList& L);
status ListEmpty(SqList L);
```

```
34 status ListLength(SqList L);
35 status GetElem(SqList L,int i,ElemType &e);
36 int LocateElem(SqList L,ElemType e);
37 status PriorElem(SqList L,ElemType e,ElemType &pre);
38 status NextElem(SqList L,ElemType e,ElemType &next);
39 status ListInsert(SqList &L,int i,ElemType e);
40 status ListDelete(SqList &L,int i,ElemType &e);
41 status ListTraverse(SqList L);
42 status  SaveList(SqList L,char FileName[]);
43 status  LoadList(SqList &L,char FileName[]);
44 int Maxsubarray(SqList L);
45 int subarraynum(SqList L ,ElemType k);
46 status sortlist(SqList L,int order);
47
48 /*-------------------------------------------*/
49 int main(void){
50     char filename[40];
51     int op=1;//chosen option
52     int i,order=0;
53     int i_num=1;
54     SqList L[MAX_NUM+1];
55     for(i=0;i<MAX_NUM+1;i++)
56     {
57         L[i].elem = NULL;
58         L[i].listsize = 0;
59         L[i].length = 0;
60     }
61     //上面的for循环是用来生成没有存储空间的线性表
62     ElemType e, cur_e , pre_e, next_e;
63     int k;
64   while(op){
65
66         system("cls");  //用于清屏
67     printf("\n\n");
68         printf("      \t\t\tMenu for Linear Table On Sequence
```

```
            Structure \n");
69          printf("   可在%d个顺序表进行多表操作，初始化请先操作功能
               15,默认在第一个表上操作\n", MAX_NUM);
70          printf("
               ------------------------------------------------\n");
71          printf("**\t\t\t1. IntiaList        7.  LocateElem\t\t\t
               **\n");
72          printf("**\t\t\t2. DestroyList      8.  PriorElem\t\t\t**\
               n");
73          printf("**\t\t\t3. ClearList        9.  NextElem \t\t\t**\
               n");
74          printf("**\t\t\t4. ListEmpty        10. ListInsert\t\t\t
               **\n");
75          printf("**\t\t\t5. ListLength       11. ListDelete\t\t\t
               **\n");
76          printf("**\t\t\t6. GetElem          12. ListTraverse\t\t\t
               **\n");
77          printf("**\t\t\t13.SaveList         14. LoadList\t\t\t
                        **\n");
78      printf("**\t\t\t15.Maxsubarray          16. subarraynum\t\t\t
           **\n");
79      printf("**\t\t\t17.sortlist                          \t\t\t**\n");
80          printf("**\t\t\t0. Exit             制作时间：2023.4.6\t\t
               \t**\n");
81          printf("**\t\t\t18.ChooseList(请先进行此选项以选择在哪个
               表上进行操作)\t**\n");
82          printf("**\t\t\t若本实验已有文件，可通过函数14进行加载\t\
               t\t**\n");
83          printf("
               ------------------------------------------------\n");
84          printf("请选择你的操作[0--18]:\n");
85          scanf("%d",&op);//选择op的值,用于switch
86      switch(op){
87              case 1:
88           //初始化线性表
```

```
89                  if(InitList(L[i_num])==OK)
90          {
91
92              printf("请输入要保存的线性表名称\n");
93              scanf("%s", filename);
94              printf("线性表创建成功\n");
95          }
96                      else printf("线性表创建失败! \n");
97                  getchar();getchar();
98                  break;
99
100           case 2:
101                 //销毁线性表
102                 if(L[i_num].elem==NULL)
103         {
104             printf("线性表不存在!\n");
105             getchar();getchar();
106             break;
107         }
108                 if(DestroyList(L[i_num])==OK)
109         {
110             printf("销毁线性表成功!\n");
111         }
112         else printf("销毁线性表失败! \n");
113                 getchar();getchar();
114                 break;
115
116           case 3:
117                 //重置线性表
118                 if(L[i_num].elem==NULL)
119         {
120             printf("线性表不存在!\n");
121             getchar();getchar();
122             break;
123         }
```

```
124             if(ClearList(L[i_num])==OK)
125             {
126                printf("线性表重置成功! \n");
127             }
128             else printf("线性表重置失败! \n");
129                     getchar();getchar();
130                     break;
131
132               case 4:
133                     //判断是否为空
134                     if(L[i_num].elem==NULL)
135             {
136                 printf("线性表不存在!\n");
137                 getchar();getchar();
138                 break;
139             }
140             if(ListEmpty(L[i_num])==TRUE)
141             {
142                 printf("线性表为空! \n");
143             }
144             else printf("线性表不是空表! \n");
145                     getchar();getchar();
146                     break;
147
148               case 5:
149                   //得到线性表长度
150                   if(L[i_num].elem==NULL)
151             {
152                 printf("线性表不存在!\n");
153                 getchar();getchar();
154                 break;
155             }
156             printf("线性表表长为%d\n",ListLength(L[i_num]));
157                     getchar();getchar();
158                     break;
```

```
159
160           case 6:
161               //得到某个元素
162             if(L[i_num].elem==NULL)
163         {
164             printf("线性表不存在!\n");
165             getchar();getchar();
166             break;
167         }
168
169         printf("请输入要取结点的位置: \n");
170                 scanf("%d",&i);
171                 if(GetElem(L[i_num],i,e)==OK)
172                 printf("第%d个结点的元素是: %d\n",i,e);
173                 else  printf("输入位置错误! \n");
174                 getchar();getchar();
175                 break;
176
177           case 7:
178               //确定元素位置
179                if(L[i_num].elem==NULL)
180         {
181             printf("线性表不存在!\n");
182             getchar();getchar();
183             break;
184         }
185         printf("请输入数据元素值: \n");
186                 scanf("%d",&e);
187         if(LocateElem(L[i_num],e)==ERROR){
188            printf("该元素不存在!\n");
189         }else{
190            i=LocateElem(L[i_num],e);
191            printf("%d元素第一次出现位于第%d个位置! \n",e,i);
192         }
193                 getchar();getchar();
```

```
194                 break;
195
196               case 8:
197                   //求出前驱结点
198                    if(L[i_num].elem==NULL)
199             {
200                 printf("线性表不存在!\n");
201                 getchar();getchar();
202                 break;
203             }
204
205             printf("请输入数据元素: \n");
206                     scanf("%d",&cur_e);
207
208                     if(PriorElem(L[i_num],cur_e,pre_e)==OK)
209                     printf("其前驱元素为: %d\n",pre_e);
210                     else if(PriorElem(L[i_num],cur_e,pre_e)==
                            OVERFLOW)
211                     printf("顺序表中没有该元素! \n");
212                     else  printf("其不存在前驱元素! \n");
213                     getchar();getchar();
214                     break;
215
216               case 9:
217                   //求出后置节点
218             printf("请输入数据元素: \n");
219                     scanf("%d",&cur_e);
220
221                 if(L[i_num].elem==NULL)
222             {
223                 printf("线性表不存在!\n");
224                 getchar();getchar();
225                 break;
226             }
227                     if(NextElem(L[i_num],cur_e,next_e)==OK){
```

```
228                 printf("其后继元素为: %d\n",next_e);
229             }
230                 else if(NextElem(L[i_num],cur_e,pre_e)==FALSE){
231                     printf("其不存在后继元素! \n");
232             }
233                 else{
234                printf("顺序表中没有该元素! \n");
235             }
236                 getchar();getchar();
237                 break;
238
239               case 10:
240                   //插入元素
241                 if(L[i_num].elem==NULL)
242             {
243                 printf("线性表不存在!\n");
244                 getchar();getchar();
245                 break;
246             }
247                 printf("请输入您要插入的数据元素: \n");
248                 scanf("%d",&e);
249                 printf("请输入您要插入的数据元素的位置: \n");
250                 scanf("%d",&i);
251                 if(ListInsert(L[i_num],i,e)==OK)
252                 printf("插入数据元素成功! \n");
253                 else
254                 printf("插入数据元素失败! \n");
255                 getchar();getchar();
256                 break;
257
258               case 11:
259                   //删除元素
260                   if(L[i_num].elem==NULL)
261             {
262                 printf("线性表不存在!\n");
```

```
263             getchar();getchar();
264             break;
265         }
266                 printf("请输入您要删除的数据元素的位置: \n");
267                 scanf("%d",&i);
268                 if(ListDelete(L[i_num],i,e)==OK)
269                 printf("删除数据元素成功! \n");
270                 else
271                 printf("删除数据元素失败! \n");
272                 getchar();getchar();
273                 break;
274
275           case 12:
276               //遍历线性表中的元素
277
278             if(L[i_num].elem==NULL)
279         {
280             printf("线性表不存在!\n");
281             getchar();getchar();
282             break;
283         }
284         if(L[i_num].length==0)
285         {
286            printf("线性表是空表! \n");
287            getchar();getchar();
288            break;
289         }
290        ListTraverse(L[i_num]);
291                 getchar();getchar();
292                 break;
293
294       case 13:
295           //保存文件
296           if(L[i_num].elem == NULL)
297         {
```

```
298             printf("线性表不存在!无法保存! \n");
299             getchar();getchar();
300             break;
301         }
302         if(SaveList(L[i_num], filename)==OK){
303            printf("文件保存成功\n文件名为%s\n",filename);
304         }
305                 getchar();getchar();
306         break;
307
308       case 14:
309        //加载文件, 需要输入需要加载的名称
310        if(L[i_num].elem!=NULL){
311            printf("线性表已存在! 无法加载! ");
312            getchar();getchar();
313            break;
314        }
315         printf("请输入要加载的文件名:\n ");
316                  scanf("%s", filename);
317                  if(LoadList(L[i_num], filename)==OK)
318            {
319                 printf("文件加载成功\n");
320                     }else{
321                 printf("文件加载失败! ");
322            }
323         break;
324
325       case 15:
326           //求连续子组的最大和
327           if(L[i_num].elem==NULL){
328                 printf("线性表不存在! ");
329                 getchar(); getchar();
330                 break;
331           }
332           printf("连续子组的最大和为%d" ,Maxsubarray(L[i_num]));
```

```
333             getchar();getchar();
334             break;
335
336          case 16:
337             //求和为k的连续子组的个数
338             if(L[i_num].elem==NULL){
339                   printf("线性表不存在! ");
340                   getchar(); getchar();
341                   break;
342             }
343             printf("请输入和的大小\n");
344             scanf("%d" ,&k);
345             printf("和为k的连续子组的个数为%d" ,subarraynum(L[
                    i_num] ,k));
346             getchar(); getchar();
347              break;
348
349          case 17:
350             //线性表排序
351             if(L[i_num].elem==NULL){
352                   printf("线性表不存在! ");
353                   getchar(); getchar();
354                   break;
355             }
356             printf("请输入排序方式, 0: 从小到大, 1: 从大到小\n");
357             scanf("%d" ,&order);
358             sortlist(L[i_num],order);
359             printf("排序完成! ");
360             getchar(); getchar();
361             break;
362
363          case 18:
364             //选择在哪个表进行操作
365             printf("请输入要在第几个表操作:\n ");
366             printf("*只支持在%d个顺序表上操作*\n",MAX_NUM);
```

```
367                     scanf("%d",&i_num);
368                   if((i_num<1)||(i_num>MAX_NUM))
369                     {
370                         printf("请选择正确范围! 默认对第一个线性
                              表进行操作\n");
371                         i_num=1;
372                     }
373                     printf("正在对第%d个表进行操作\n",i_num);
374             getchar(); getchar();
375           break;
376
377             case 0:
378          //退出菜单, 退出整个程序
379           break;
380          }//end of switch
381   }//end of while
382
383   printf("欢迎下次再使用本系统!\n");
384   system("pause");
385   return 0;
386 }//end of main()
387
388 /******************function defination*************************/
389
390 /*****************************************************************
391 *函数名称: IntiaList
392 *函数功能: 构造一个空的线性表
393 *初始条件: 线性表L不存在已存在
394 *操作结果: 构造一个空的线性表。
395 *返回值类型: status类型
396 *返回结果: 线性表L不存在, 构造一个空的线性表, 返回OK, 否则返回
      INFEASIBLE。
397 *****************************************************************/
398 status InitList(SqList& L)
399 {
```

```
400     if(L.elem){
401         return INFEASIBLE;
402     }
403     L.elem =(ElemType*)malloc(sizeof(ElemType) * LIST_INIT_SIZE);
404     L.length=0;
405     L.listsize=LIST_INIT_SIZE;
406     return OK;
407 }
408
409
410 /*****************************************************************
411 *函数名称: DestoryList
412 *函数功能: 销毁线性表
413 *初始条件: 线性表L已存在
414 *操作结果: 销毁线性表L
415 *返回值类型: status类型
416 *返回结果: 如果线性表L存在, 销毁线性表L, 释放数据元素的空间, 返回
        OK, 否则返回INFEASIBLE。
417 *****************************************************************/
418 status DestroyList(SqList& L)
419 {
420     if(L.elem==NULL){
421         return INFEASIBLE;
422     }
423     L.length=0;
424     L.listsize=0;
425     free(L.elem);
426     L.elem=NULL;
427     return OK;
428 }
429
430
431 /*****************************************************************
432 *函数名称: ClearList
433 *函数功能: 重置顺序表
```

```
434 *初始条件：线性表L已存在
435 *操作结果：将L重置为空表。
436 *返回值类型：status类型
437 *返回结果：如果线性表L存在，删除线性表L中的所有元素，返回OK，否则
        返回INFEASIBLE。
438 **************************************************************/
439 status ClearList(SqList& L)
440 {
441     if(L.elem==NULL){
442         return INFEASIBLE;
443     }
444     L.length=0;
445     free(L.elem);
446     return OK;
447 }
448
449
450
451 /**************************************************************
452 *函数名称：ListEmpty
453 *函数功能：判断线性表是否为空
454 *初始条件：线性表L已存在
455 *操作结果：若L为空表则返回TRUE,否则返回FALSE。
456 *返回值类型：status类型
457 *返回结果：如果线性表L存在，判断线性表L是否为空，空就返回TRUE，否
        则返回FALSE；如果线性表L不存在，返回INFEASIBLE。
458 **************************************************************/
459 status ListEmpty(SqList L)
460 {
461     if(L.elem==NULL){
462         return INFEASIBLE;
463     }
464     if(L.length==0){
465         return TRUE;
466     }
```

```
467     return FALSE;
468 }
469
470
471
472 /*****************************************************************
473 *函数名称: ListLength
474 *函数功能: 求线性表的表长
475 *初始条件: 线性表已存在
476 *操作结果: 返回L中数据元素的个数。
477 *返回值类型: status类型
478 *返回结果: 如果线性表L存在, 返回线性表L的长度, 否则返回INFEASIBLE
        。
479 *****************************************************************/
480 status ListLength(SqList L)
481 {
482     if(L.elem==NULL){
483         return INFEASIBLE;
484     }
485         return L.length;
486 }
487
488
489 /*****************************************************************
490 *函数名称: GetElem
491 *函数功能: 得到某一个元素的值
492 *初始条件: 线性表已存在, 1 i ListLength(L)
493 *操作结果: 用e返回L中第i个数据元素的值
494 *返回值类型: status类型
495 *返回结果: 如果线性表L存在, 获取线性表L的第i个元素, 保存在e中, 返
        回OK;
496             如果i不合法, 返回ERROR;
497             如果线性表L不存在, 返回INFEASIBLE。
498 *****************************************************************/
499 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;
}


/**************************************************************
*函数名称: LocateElem
*函数功能: 查找元素
*初始条件: 线性表已存在;
*操作结果: 返回L中第1个与e相等的数据元素的位序
*返回值类型: status类型
*返回结果: 如果线性表L存在, 查找元素e在线性表L中的位置序号并返回
    该序号;
          如果e不存在, 返回ERROR;
          当线性表L不存在时, 返回INFEASIBLE。
**************************************************************/
int LocateElem(SqList L,ElemType e)
{
    int i;
    if(L.elem==NULL){
        return INFEASIBLE;
    }
    for(i = 0 ;i < L.length ;i++){
        if(L.elem[i]==e){
            return i+1;
        }
    }
    return ERROR;
```

```
534 }
535
536
537 /**************************************************************
538 *函数名称：PriorElem
539 *函数功能：求元素的前驱
540 *注释：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，
541         则用pre_e返回它的前驱，否则操作失败，pre_e无定义。
542 *返回值类型：status类型
543 *返回结果：如果线性表L存在，获取线性表L中元素e的前驱，保存在pre
    中，返回OK；
544             如果没有前驱，返回ERROR；
545             如果线性表L不存在，返回INFEASIBLE。
546 **************************************************************/
547 status PriorElem(SqList L,ElemType e,ElemType &pre)
548 {
549     int i;
550     if(L.elem==NULL){
551         return INFEASIBLE;
552     }
553     for(i = 0 ;i < L.length ;i++){
554         if(e==L.elem[i]){
555             if(i==0){
556                 return ERROR;
557             }
558                 pre=L.elem[i-1];
559                 return OK;
560         }
561     }
562     return OVERFLOW;
563 }
564
565
566 /**************************************************************
```

```
*函数名称：NextElem
*函数功能：求后继节点
*输入输出：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。
*返回值类型：status类型
*返回结果：如果线性表L存在，获取线性表L元素e的后继，保存在next中，返回OK；
            如果没有后继，返回ERROR；
            如果线性表L不存在，返回INFEASIBLE。
**********************************************************************/
status NextElem(SqList L,ElemType e,ElemType &next)
{
    int i;
    if(L.elem==NULL){
        return INFEASIBLE;
    }
    for(i = 0 ;i < L.length ;i++){
        if(e==L.elem[i]){
            if(i==L.length-1){
                return FALSE;
            }
                next=L.elem[i+1];
                return OK;
        }
    }
    return ERROR;
}


/**********************************************************************
*函数名称：ListInsert
*函数功能：插入元素
*初始条件：线性表L已存在且非空，1 i ListLength(L)+1
*操作结果：在L的第i个位置之前插入新的数据元素e
```

```
*返回值类型：status类型
*返回结果：如果线性表L存在，将元素e插入到线性表L的第i个元素之前，
    返回OK;
            当插入位置不正确时，返回ERROR;
            如果线性表L不存在，返回INFEASIBLE。
*****************************************************************/
status ListInsert(SqList &L,int i,ElemType e)
{
    int j;
    ElemType* newbase;
    if(L.elem==NULL){
        return INFEASIBLE;
    }
    if(i<1 || i > L.length+1){
        return ERROR;
    }
    if(L.length>=L.listsize){
        newbase=(ElemType*)realloc(L.elem,sizeof(ElemType)*(L.
            listsize+LISTINCREMENT));
        if(newbase==NULL){
            return OVERFLOW;
        }
        L.elem=newbase;
        L.listsize+=LISTINCREMENT;
    }
    for(j = L.length - 1; j>=i-1 ;j--){
        L.elem[j+1]=L.elem[j];
    }
    L.elem[i - 1]=e;
    L.length++;
    return OK;
}


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

```
633 *函数名称：ListDelete
634 *函数功能：删除元素
635 *初始条件：线性表L已存在且非空，1 i ListLength(L);
636 *操作结果：删除L的第i个数据元素，用e返回其值。
637 *返回值类型：status类型
638 *返回结果：如果线性表L存在，删除线性表L的第i个元素，并保存在e中，
        返回OK;
639             当删除位置不正确时，返回ERROR;
640             如果线性表L不存在，返回INFEASIBLE。
641 ****************************************************************/
642 status ListDelete(SqList &L,int i,ElemType &e)
643 {
644     int j;
645     if(L.elem==NULL){
646         return INFEASIBLE;
647     }
648     if(i<1 || i >L.length){
649         return ERROR;
650     }
651     e=L.elem[i - 1];
652     for(j = i - 1 ; j <L.length-1;j++){
653         L.elem[j] = L.elem[j+1];
654     }
655     L.length--;
656     return OK;
657 }
658
659
660 /****************************************************************
661 *函数名称：ListTraverse
662 *函数功能：遍历顺序表
663 *操作结果：输出顺序表的值
664 *返回值类型：status类型
665 *返回结果：如果线性表L存在，依次显示线性表中的元素，每个元素间空
        一格，返回OK;
```

```
666             如果线性表L不存在，返回INFEASIBLE。
667 **************************************************************/
668 status ListTraverse(SqList L)
669 {
670     int i;
671     if(L.elem==NULL){
672         return INFEASIBLE;
673     }
674     if(L.length==0){
675         return ERROR;
676     }
677     printf("\n-----------all elements ----------------------\n")
        ;
678         for(i = 0 ; i < L.length ;i++){
679             printf("%d" ,L.elem[i]);
680             if(i < L.length-1){
681                 printf(" ");
682             }
683         }
684     printf("\n------------------ end ------------------------\n")
        ;
685     return OK;
686 }
687
688
689 /****************below are the added function*****************/
690
691 /**************************************************************
692 *函数名称：SaveList
693 *函数功能：保存线性表
694 *注释：将线性表保存到文件中
695 *返回值类型：status
696 *返回结果：如果线性表L存在，将线性表L的的元素写到FileName文件中，
        返回OK，否则返回INFEASIBLE。
697 **************************************************************/
```

```
698 status  SaveList(SqList L,char FileName[])
699 {
700     FILE*p;
701     int i;
702     if(L.elem==NULL){
703         return INFEASIBLE;
704     }
705     p=fopen(FileName ,"wb");
706     if(p!=NULL){
707         for(i = 0 ;i < L.length ;i++){
708             fprintf(p,"%d " ,L.elem[i]);
709         }
710     }else{
711         return ERROR;
712     }
713     fclose(p);
714     return OK;
715 }
716
717
718 /**************************************************************
719 *函数名称：LoadList
720 *函数功能：加载文件
721 *操作结果：加载文件中的数据到线性表中
722 *返回值类型：status类型
723 *返回结果：如果线性表L不存在，将FileName文件中的数据读入到线性表L
        中，返回OK，否则返回INFEASIBLE。
724 **************************************************************/
725 status  LoadList(SqList &L,char FileName[])
726 {
727     FILE* p;
728     int c,j=0;
729     ElemType* newbase;
730
731     if(L.elem){
```

```
732         return INFEASIBLE;
733     }
734         L.elem=(ElemType*)malloc(sizeof(ElemType)*LIST_INIT_SIZE)
                ;
735         L.length=0;
736         L.listsize=LIST_INIT_SIZE;
737 
738         p=fopen(FileName ,"rb");
739        while((fscanf(p,"%d",&c))!=EOF)    {
740             if(L.length>=L.listsize){
741                 newbase=(ElemType*)realloc(L.elem,sizeof(ElemType
                    )*(L.listsize+LISTINCREMENT));
742                 if(newbase==NULL){
743                     return OVERFLOW;
744                 }
745                 L.elem=newbase;
746                 L.listsize+=LISTINCREMENT;
747             }
748             L.elem[j++]=c;
749             L.length++;
750         }
751     fclose(p);
752     return OK;
753 }
754 
755 
756 /*****************************************************************
757 *函数名称: Maxsubarray
758 *函数功能: 找出有最大和的连续子数组
759 *算法思路: 遍历线性表中的元素, 运用贪心算法, 如果当前累加和为0,
        更新结果,
760             如果当前累加和为负, 将当前累加和重置为0, 表示从该位置重
                新开始求和
761 *返回值类型: int类型,返回其最大和
762 *****************************************************************/
```

```
763 int Maxsubarray(SqList L)
764 {
765     int maxsum=INT_MIN ,cursum=0;
766     int i;
767     for(i = 0 ;i < L.length ;i++){
768         cursum+=L.elem[i];
769         if(cursum > maxsum){
770             maxsum=cursum;
771         }
772         if(cursum < 0){
773             cursum=0;
774         }
775     }
776     return maxsum;
777 }
778
779
780 /*****************************************************************
781 *函数名称: subarraynum
782 *函数功能: 找出和为k的连续子组的个数
783 *算法思路: 遍历线性表中元素, 依次作为起始点, 计算从该起始点开始的
        连续子组的和,
784             如果和为k, 计数器加1, 否则继续遍历
785 *返回值类型: int类型,返回和为k的连续子组的个数
786 *****************************************************************/
787 int subarraynum(SqList L ,ElemType k)
788 {
789     int i ,j ,sum=0,cnt=0;
790     if(L.elem==NULL){
791         return INFEASIBLE;
792     }
793     for(i = 0 ;i < L.length ;i++){
794         for(j = i ;j < L.length ;j++){
795             sum+=L.elem[j];
796             if(sum==k){
```

```
797                 cnt++;
798             }
799         }
800         sum=0;
801     }
802     return cnt;
803 }
804
805
806 /*****************************************************************
807 *函数名称: sortlist
808 *函数功能: 将L由小到大排序;
809 *初始条件: 是线性表L已存在
810 *操作结果: 将L由小到大排序
811 *算法思路: 采用的是冒泡排序的方法
812 *返回值类型: status
813 *****************************************************************/
814 status sortlist(SqList L,int order)
815 {
816     if(L.elem==NULL){
817         return INFEASIBLE;
818     }
819     int i ,j;
820     ElemType t;
821     //order=1从大到小排序, order=0从小到大排序
822     if(order==1){
823         for(i = 0 ;i < L.length - 1;i++){
824             for(j  = 0 ;j < L.length - 1 - i ;j++){
825                 if(L.elem[j] < L.elem[j+1]){
826                     t = L.elem[j];
827                     L.elem[j] =L.elem[j+1];
828                     L.elem[j+1]=t;
829                 }
830             }
831         }
```

```
832     }else{
833         for(i = 0 ;i < L.length - 1;i++){
834             for(j  = 0 ;j < L.length - 1 - i ;j++){
835                 if(L.elem[j] > L.elem[j+1]){
836                     t = L.elem[j];
837                     L.elem[j] =L.elem[j+1];
838                     L.elem[j+1]=t;
839                 }
840             }
841         }
842     }
843     return OK;
844 }
```

附录 B 基于链式存储结构线性表实现的源程序

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

/*---------page 10 on textbook ---------*/
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define MAX_NUM 10

typedef int status;
typedef int ElemType;

/*-------page 22 on textbook -------*/
#define LIST_INIT_SIZE 100
#define LISTINCREMENT  10
typedef struct LNode{
        ElemType data;
        struct LNode *next;
}LNode, *LinkList;

status InitList(LinkList &L);
status DestroyList(LinkList &L);
status ClearList(LinkList &L);
status ListEmpty(LinkList L);
int ListLength(LinkList L);
status GetElem(LinkList L,int i,ElemType &e);
status LocateElem(LinkList L,ElemType e);
status PriorElem(LinkList L,ElemType e,ElemType &pre);
```

```
34 status NextElem(LinkList L,ElemType e,ElemType &next);
35 status ListInsert(LinkList &L,int i,ElemType e);
36 status ListDelete(LinkList &L,int i,ElemType &e);
37 status ListTraverse(LinkList L);
38 status reverseList(LinkList L);
39 status RemoveNthFromEnd(LinkList L,int n);
40 status sortlist(LinkList L,int order);
41 status SaveList(LinkList L,char FileName[]);
42 status LoadList(LinkList &L,char FileName[]);
43
44 int main(){
45         char filename[40];
46         int op=1;
47     int i,i_num=1,len,order=0;
48     LinkList L[MAX_NUM];
49     for (i = 0; i<MAX_NUM; i++)
50         {
51         L[i]=NULL;
52         }
53         ElemType e, cur_e, pre_e, next_e;
54     while(op){
55         system("cls");
56     printf("\n\n");
57     printf("      \t\t\tMenu for Linear Table On Sequence
       Structure \n");
58         printf("  可在%d个顺序表进行多表操作, 初始化请先操作功能
           18,默认在第一个表上操作\n", MAX_NUM);
59         printf("
           ----------------------------------------------------\n"
           );
60         printf("**\t\t\t1. InitList         9.  NextElem\t\t\t
                   **\n");
61         printf("**\t\t\t2. DestroyList      10. ListInsert\t\t\t
           **\n");
62         printf("**\t\t\t3. ClearList        11. ListDelete \t\t\t
```

```
            **\n");
63          printf("**\t\t\t4. ListEmpty        12. ListTraverse\t\t\t
                **\n");
64          printf("**\t\t\t5. ListLength       13. reverseList\t\t\t
                **\n");
65          printf("**\t\t\t6. GetElem          14. RemoveNthFromEnd\t
                \t\t **\n");
66      printf("**\t\t\t7. LocateElem       15. sortlist\t\t\t
                    **\n");
67      printf("**\t\t\t8. PriorElem        16. SaveList\t\t\t
                    **\n");
68          printf("**\t\t\t17.LoadList                          \t\t\t
                **\n");
69          printf("**\t\t\t0. Exit             制作时间2023.4.12\t\t\
                t **\n");
70          printf("**\t\t\t18.ChooseList(请先进行此选项以选择在哪个
                表上进行操作)\n");
71          printf("**\t\t\t本实验已有文件sss, 可通过函数17进行加载\n
                ");
72          printf("
                ----------------------------------------------------\n"
                );
73          printf("请选择你的操作[0-18]:\n");
74          scanf("%d",&op);
75      switch(op)
76          {
77              case 1:
78           //初始化线性表
79                      if(InitList(L[i_num])==OK)
80           {
81
82                printf("请输入要保存的线性表名称\n");
83                scanf("%s", filename);
84                printf("线性表创建成功\n");
85           }
```

```
86                  else printf("线性表创建失败! \n");
87              getchar();getchar();
88              break;
89
90          case 2:
91              //销毁线性表
92              if(L[i_num] == NULL)
93          {
94              printf("线性表不存在!\n");
95              getchar();getchar();
96              break;
97          }
98              if(DestroyList(L[i_num])==OK)
99          {
100             printf("销毁线性表成功!\n");
101         }
102         else printf("销毁线性表失败! \n");
103             getchar();getchar();
104             break;
105
106         case 3:
107             //重置线性表
108             if(L[i_num] == NULL)
109         {
110             printf("线性表不存在!\n");
111             getchar();getchar();
112             break;
113         }
114         if(ClearList(L[i_num])==OK)
115         {
116            printf("线性表重置成功! \n");
117         }
118         else printf("线性表重置失败! \n");
119             getchar();getchar();
120             break;
```

```
121
122             case 4:
123                   //判断是否为空
124                   if(L[i_num] == NULL)
125             {
126                 printf("线性表不存在!\n");
127                 getchar();getchar();
128                 break;
129             }
130             if(ListEmpty(L[i_num]))
131             {
132                 printf("线性表为空! \n");
133             }
134             else printf("线性表不是空表! \n");
135                   getchar();getchar();
136                   break;
137
138             case 5:
139                 //得到线性表长度
140                 if(L[i_num] == NULL)
141             {
142                 printf("线性表不存在!\n");
143                 getchar();getchar();
144                 break;
145             }
146             printf("线性表表长为%d\n",ListLength(L[i_num]));
147                   getchar();getchar();
148                   break;
149
150             case 6:
151                   //得到某个元素
152                 if(L[i_num] == NULL)
153             {
154                 printf("线性表不存在!\n");
155                 getchar();getchar();
```

```
156             break;
157         }
158                 printf("请输入要取结点的位置: \n");
159                 scanf("%d",&i);
160                 if(GetElem(L[i_num],i,e)==OK)
161                 printf("第%d个结点的元素是: %d\n",i,e);
162                 else  printf("输入位置错误! \n");
163                 getchar();getchar();
164                 break;
165
166           case 7:
167       //查找e元素在线性表中的位置
168                 if(L[i_num] == NULL)
169         {
170             printf("线性表不存在!\n");
171             getchar();getchar();
172             break;
173         }
174                 printf("请输入数据元素值: \n");
175                 scanf("%d",&e);
176         i = LocateElem(L[i_num],e);
177                 if(i!=ERROR){
178            printf("%d元素第一次出现位于第%d个位置! \n",e,i);
179         }
180                 else  printf("该元素不存在!\n");
181                 getchar();getchar();
182                 break;
183
184           case 8:
185               //求出前驱结点
186               if(L[i_num] == NULL)
187         {
188             printf("线性表不存在!\n");
189             getchar();getchar();
190             break;
```

```
191         }
192                 printf("请输入数据元素: \n");
193                 scanf("%d",&cur_e);
194                 PriorElem(L[i_num],cur_e,pre_e);
195                 if(PriorElem(L[i_num],cur_e,pre_e)==OK)
196                 printf("其前驱元素为: %d\n",pre_e);
197                 else if(PriorElem(L[i_num],cur_e,pre_e)==
                        OVERFLOW)
198                 printf("顺序表中没有该元素! \n");
199                 else  printf("其不存在前驱元素! \n");
200                 getchar();getchar();
201                 break;
202
203
204           case 9:
205               //求出后置节点
206               if(L[i_num] == NULL)
207         {
208             printf("线性表不存在!\n");
209             getchar();getchar();
210             break;
211         }
212         printf("请输入数据元素: \n");
213                 scanf("%d",&cur_e);
214                 if(NextElem(L[i_num],cur_e,next_e)==OK){
215            printf("其后继元素为: %d\n",next_e);
216         }else if(NextElem(L[i_num],cur_e,pre_e)==OVERFLOW){
217            printf("顺序表中没有该元素! \n");
218         }else{
219            printf("其不存在后继元素! \n");
220         }
221                 getchar();getchar();
222                 break;
223
224           case 10:
```

```
225             //插入元素
226             if(L[i_num] == NULL)
227         {
228             printf("线性表不存在!\n");
229             getchar();getchar();
230             break;
231         }
232                 printf("请输入您要插入的数据元素: \n");
233                 scanf("%d",&e);
234                 printf("请输入您要插入的数据元素的位置: \n");
235                 scanf("%d",&i);
236                 if(ListInsert(L[i_num],i,e)==OK)
237                 printf("插入数据元素成功! \n");
238                 else
239                 printf("插入数据元素失败! \n");
240                 getchar();getchar();
241                 break;
242
243           case 11:
244               //删除元素
245               if(L[i_num] == NULL)
246         {
247             printf("线性表不存在!\n");
248             getchar();getchar();
249             break;
250         }
251                 printf("请输入您要删除的数据元素的位置: \n");
252                 scanf("%d",&i);
253                 if(ListDelete(L[i_num],i,e)==OK)
254                 printf("删除数据元素成功! \n");
255                 else
256                 printf("删除数据元素失败! \n");
257                 getchar();getchar();
258                 break;
259
```

```
260             case 12:
261                 //遍历线性表中的元素
262                 if(L[i_num] == NULL)
263             {
264                 printf("线性表不存在!\n");
265                 getchar();getchar();
266                 break;
267             }
268             if(ListEmpty(L[i_num])) {
269                printf("线性表是空表! \n");
270                getchar();getchar();
271                         break;
272             }
273             ListTraverse(L[i_num]);
274                     getchar();getchar();
275                     break;
276
277         case 13:
278             //反转线性表
279             if(L[i_num]==NULL){
280                printf("线性表不存在! ");
281                getchar();getchar();
282                         break;
283             }
284             if(reverseList(L[i_num])==OK){
285                printf("反转完成");
286             }
287             getchar();getchar();
288                     break;
289
290         case 14:
291         //删除倒数第n个元素
292         if(L[i_num]==NULL){
293          printf("线性表是空表");
294          getchar();getchar();
```

```
295                 break;
296         }
297         len=ListLength(L[i_num]);
298         printf("请输入要删除的倒数第几个位置\n");
299         scanf("%d" ,&i);
300         if(i < 1 || i > len){
301          printf("要删除的位置不合法!");
302          getchar();getchar();
303                 break;
304         }
305         RemoveNthFromEnd(L[i_num] ,i);
306         printf("删除成功! ");
307         getchar();getchar();
308                 break;
309
310         case 15:
311         //线性表排序
312         if(L[i_num]==NULL){
313          printf("线性表是空表! ");
314          getchar();getchar();
315                 break;
316         }
317         printf("请输入排序方式, 0: 从小到大, 1: 从大到小\n");
318          scanf("%d" ,&order);
319          sortlist(L[i_num],order);
320         printf("排序成功");
321          getchar();getchar();
322                 break;
323
324            case 16:
325             //保存文件
326             if(L[i_num] == NULL)
327           {
328                 printf("线性表不存在!\n");
329                 getchar();getchar();
```

```
330             break;
331          }
332          if(SaveList(L[i_num], filename)==OK){
333             printf("文件保存成功!文件名为%s\n",filename);
334          }
335                  getchar();getchar();
336          break;
337
338            case 17:
339        //加载文件, 需要输入需要加载的名称
340        if((L[i_num]!=NULL)){
341             printf("顺序表已存在, 无法加载! ");
342        }else{
343             printf("请输入要加载的文件名:\n ");
344                      scanf("%s", filename);
345                      if(LoadList(L[i_num], filename)==OK){
346                  printf("文件加载成功\n");
347                      }
348        }
349                  getchar();getchar();
350          break;
351
352            case 18:
353            //选择在哪个表进行操作
354            printf("请输入要在第几个表操作:\n ");
355            printf("*只支持在%d个顺序表上操作*\n",MAX_NUM);
356                     scanf("%d",&i_num);
357                     printf("正在对第%d个表进行操作\n",i_num);
358                  if((i_num<1)||(i_num>10))
359                  {
360                          printf("请选择正确范围! \n");
361                          i_num=1;
362                  }
363                  getchar(); getchar();
364          break;
```

```
365
366             case 0:
367           break;
368         }//end of switch
369   }//end of while
370         printf("\t\t欢迎再次使用本系统! \n");
371 }//end of main()
372 /*--------page 23 on textbook --------------------*/
373
374
375 /**************************************************************
376 *函数名称: IntiaList
377 *函数功能: 构造一个空的线性表
378 *初始条件: 线性表L不存在已存在
379 *操作结果: 构造一个空的线性表。
380 *返回值类型: status类型
381 **************************************************************/
382 status InitList(LinkList &L)
383 {
384     if(L!=NULL){
385         return INFEASIBLE;
386     }
387     LNode *p;
388     p = (LNode*)malloc(sizeof(LNode));
389     p->next=NULL;
390     L=p;
391     return OK;
392 }
393
394 /**************************************************************
395 *函数名称: DestoryList
396 *函数功能: 销毁线性表
397 *初始条件: 线性表L已存在
398 *操作结果: 销毁线性表L
399 *返回值类型: status类型
```

```
400 ************************************************************/
401 status DestroyList(LinkList &L)
402 {
403     if(L==NULL){
404         return INFEASIBLE;
405     }
406     LNode*p;
407     for(p=L->next;p;p=L->next){
408         L->next=p->next;
409         free(p);
410     }
411     free(L);
412     L=NULL;
413     return OK;
414 }
415 
416 /**************************************************************
417 *函数名称：ClearList
418 *函数功能：重置顺序表
419 *初始条件：线性表L已存在
420 *操作结果：将L重置为空表。
421 *返回值类型：status类型
422 **************************************************************/
423 status ClearList(LinkList &L)
424 {
425     if(L==NULL){
426         return INFEASIBLE;
427     }
428     LNode* p;
429     for(p=L->next;p;p=L->next){
430         L->next=p->next;
431         free(p);
432     }
433     return OK;
434 }
```

```
435
436 /**************************************************************
437 *函数名称: ListEmpty
438 *函数功能: 判断线性表是否为空
439 *初始条件: 线性表L已存在
440 *操作结果: 若L为空表则返回TRUE,否则返回FALSE。
441 *返回值类型: status类型
442 **************************************************************/
443 status ListEmpty(LinkList L)
444 {
445     if(L==NULL){
446         return INFEASIBLE;
447     }
448     if(L->next==NULL){
449         return TRUE;
450     }
451     return FALSE;
452 }
453
454 /**************************************************************
455 *函数名称: ListLength
456 *函数功能: 求线性表的表长
457 *初始条件: 线性表已存在
458 *操作结果: 返回L中数据元素的个数。
459 *返回值类型: status类型
460 **************************************************************/
461 int ListLength(LinkList L)
462 {
463     if(L==NULL){
464         return INFEASIBLE;
465     }
466     LNode* p;
467     int cnt=0;
468     for(p = L->next ;p;p=p->next){
469         cnt++;
```

```
470     }
471     return cnt;
472 }
473
474
475 /*****************************************************************
476 *函数名称: GetElem
477 *函数功能: 得到某一个元素的值
478 *初始条件: 线性表已存在, 1 i ListLength(L)
479 *操作结果: 用e返回L中第i个数据元素的值
480 *返回值类型: status类型
481 *****************************************************************/
482 status GetElem(LinkList L,int i,ElemType &e)
483 {
484     if(L==NULL){
485         return INFEASIBLE;
486     }
487     int index=1;
488     LNode* p;
489     if(i < 1){
490         return ERROR;
491     }
492     for(p=L->next;p ;p=p->next){
493         if(index==i){
494             e = p->data;
495             return OK;
496         }
497         index++;
498     }
499     return ERROR;
500 }
501 /*****************************************************************
502 *函数名称: LocateElem
503 *函数功能: 查找元素
504 *操作结果: 如果在线性表中存在, 返回在线性表中的位序
```

```
*返回值类型：status类型
**********************************************************************/
status LocateElem(LinkList L,ElemType e)
{
    if(L==NULL){
        return INFEASIBLE;
    }
    LNode* p;
    int index=1;
    for(p =L->next ;p ;p=p->next){
        if(p->data==e){
            return index;
        }
        index++;
    }
    return ERROR;
}


/**********************************************************************
*函数名称：PriorElem
*函数功能：求元素的前驱
*初始条件：线性表L已存在
*操作结果：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。
*返回值类型：status类型
**********************************************************************/
status PriorElem(LinkList L,ElemType e,ElemType &pre)
{
    if(L==NULL){
        return INFEASIBLE;
    }
    LNode* p ,*prior;
    for(p =L->next;p ;prior=p,p = p->next){
        if(p->data==e){
```

```
            if(p==L->next){
                return ERROR;
            }else{
                pre=prior->data;
                return OK;
            }
        }
    }
    return OVERFLOW;
}
/**************************************************************
*函数名称：NextElem
*函数功能：求后继节点
*初始条件：线性表L已存在;
*操作结果：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。
*返回值类型：status类型
**************************************************************/
status NextElem(LinkList L,ElemType e,ElemType &next)
{
    if(L==NULL){
        return INFEASIBLE;
    }
    LNode* p;
    for(p =L->next;p ;p = p->next){
        if(p->data==e){
            if(p->next==NULL){
                return ERROR;
            }else{
                next=p->next->data;
                return OK;
            }
        }
    }
    return OVERFLOW;
```

```
575 }
576
577
578 /*****************************************************************
579 *函数名称: ListInsert
580 *函数功能: 插入元素
581 *初始条件: 线性表L已存在且非空, 1 i ListLength(L)+1;
582 *操作结果: 在L的第i个位置之前插入新的数据元素e
583 *返回值类型: status类型
584 *****************************************************************/
585 status ListInsert(LinkList &L,int i,ElemType e)
586 {
587     if(L==NULL){
588         return INFEASIBLE;
589     }
590     if(i<1){
591         return ERROR;
592     }
593     LNode* p ,*newnode;
594     newnode=(LNode*)malloc(sizeof(LNode));
595     newnode->data=e;
596     if(i==1){
597         newnode->next=L->next;
598         L->next=newnode;
599         return OK;
600     }
601     int cnt=0;
602     for(p = L->next ;p;p=p->next){
603         cnt++;
604         if(cnt==i -1){
605            newnode->next = p->next;
606             p->next=newnode;
607             return OK;
608         }
609     }
```

```
610     free(newnode);
611     return ERROR;
612 }
613
614 /*****************************************************************
615 *函数名称: ListDelete
616 *函数功能: 删除元素
617 *初始条件: 线性表L已存在且非空, 1 i ListLength(L);
618 *操作结果: 删除L的第i个数据元素, 用e返回其值。
619 *返回值类型: status类型
620 *****************************************************************/
621 status ListDelete(LinkList &L,int i,ElemType &e)
622 {
623     if(L==NULL){
624         return INFEASIBLE;
625     }
626     if(i<1){
627         return ERROR;
628     }
629     LNode* p ,*del;
630     int cnt=0;
631     for(p=L ;p->next && cnt<i -1;){
632         p=p->next;
633         cnt++;
634     }
635     if(p->next==NULL){
636         return ERROR;
637     }
638     del=p->next;
639     e=del->data;
640     p->next=del->next;
641     free(del);
642     return OK;
643 }
644
```

```
645 /*****************************************************************
646 *函数名称: ListTraverse
647 *函数功能: 遍历顺序表
648 *操作结果: 输出顺序表的值
649 *返回值类型: status类型
650 *****************************************************************/
651 status ListTraverse(LinkList L)
652 {
653     if(L==NULL){
654         return INFEASIBLE;
655     }
656     if(L->next==NULL){
657         return ERROR;
658     }
659     LNode* p;
660     printf("\n---------------------------- all elements
        -------------------------------------\n");
661     for(p =L->next ;p ;p =p->next){
662         printf("%d" ,p->data);
663         if(p->next){
664             printf(" ");
665         }
666     }
667     printf("\n--------------------------------- end
        ----------------------------------------\n");
668     return OK;
669 }
670
671 /*****************************************************************
672 *函数名称: SaveList
673 *函数功能: 保存线性表
674 *操作结果: 将线性表保存
675 *返回值类型: status
676 *****************************************************************/
677 status SaveList(LinkList L,char FileName[])
```

```
678 {
679     if(L==NULL){
680         return INFEASIBLE;
681     }
682     FILE* fp;
683     LNode* p;
684     fp=fopen(FileName,"wb");
685     if(fp==NULL){
686         return ERROR;
687     }
688     for(p=L->next ;p ;p=p->next){
689         fprintf(fp,"%d " ,p->data);
690     }
691     fclose(fp);
692     return OK;
693 }
694 
695 /**************************************************************
696 *函数名称: LoadList
697 *函数功能: 加载文件
698 *操作结果: 加载文件, 以便功能的测试, 文件名要正确
699 *返回值类型: status类型
700 **************************************************************/
701 status LoadList(LinkList &L,char FileName[]){
702     if(L!=NULL){
703         return INFEASIBLE;
704     }
705     LNode* p ,*now;
706     p=(LNode*)malloc(sizeof(LNode));
707     p->next=NULL;
708     L=p;
709     now=L;
710     FILE* fp;
711     int c;
712     fp=fopen(FileName,"rb");
```

```
713     if(fp==NULL){
714         return ERROR;
715     }
716     while(fscanf(fp,"%d " ,&c)==1){
717         LNode* node=(LNode*)malloc(sizeof(LNode));
718         node->data=c;
719         node->next=NULL;
720         now->next=node;
721         now=node;
722     }
723     fclose(fp);
724     return OK;
725 }
726 /*****************************************************************
727 *函数名称: reverseList
728 *函数功能: 反转线性表
729 *返回结果: 线性表不存在或空表返回INFEASIBLE,反转成功返回OK
730 *返回值类型: status类型
731 *****************************************************************/
732 status reverseList(LinkList L)
733 {
734     LinkList p ,prior,post;
735     if(L==NULL ||ListEmpty(L)){
736         return INFEASIBLE;
737     }
738     if(ListLength(L)==1){
739         return OK;
740     }
741     prior=L->next;
742     p=prior->next;
743     post=p->next;
744     while(1){
745         p->next=prior;
746         if(post==NULL){
747             break;
```

```
748         }
749         prior=p;
750         p = post;
751         post =post->next;
752     }
753     L->next->next=NULL;
754     L->next=p;
755     return OK;
756 }
757 /*****************************************************************
758 *函数名称: RemoveNthFromEnd
759 *函数功能: 删除倒数第n个元素
760 *返回结果: 线性表不存在返回INFEASIBLE,
761             删除位置不合法返回ERROR
762             删除成功返回OK
763 *返回值类型: status类型
764 *****************************************************************/
765 status RemoveNthFromEnd(LinkList L,int n)
766 {
767     int temp;
768     int length=ListLength(L);
769     if(L==NULL||L->next==NULL){
770         return INFEASIBLE;
771     }
772     if(n < 1 || n > length){
773         return ERROR;
774     }
775     ListDelete(L, length+1-n, temp);
776     return OK;
777 }
778
779 /*****************************************************************
780 *函数名称: sortlist
781 *函数功能: 线性表从小到大排序
782 *算法思路:冒泡排序
```

```
783 *返回值类型: status类型
784 ***************************************************************/
785 status sortlist(LinkList L,int order)
786 {
787         int t;
788     if(L==NULL){
789         return INFEASIBLE;
790     }
791     if(L->next==NULL){
792         return OK;
793     }
794     LinkList ptr,done;
795     if(order){
796         //order=1,从大到小排序
797         for(done=NULL;L->next->next!=done;done=ptr){
798             for(ptr =L->next ;ptr->next!=done ;){
799                 if(ptr->data < ptr->next->data){
800                     t = ptr->data;
801                     ptr->data=ptr->next->data;
802                     ptr->next->data=t;
803                 }
804                 ptr = ptr->next;
805             }
806         }
807     }else{
808         //order=0,从小到大排序
809         for(done=NULL;L->next->next!=done;done=ptr){
810             for(ptr =L->next ;ptr->next!=done ;){
811                 if(ptr->data > ptr->next->data){
812                     t = ptr->data;
813                     ptr->data=ptr->next->data;
814                     ptr->next->data=t;
815                 }
816                 ptr = ptr->next;
817             }
```

```
818             }
819         }
820         return OK;
821     }
```

附录 C 基于二叉链表二叉树实现的源程序

头文件 def.h 如下：

```
/*definition of function of BiTree*/
#include "stdio.h"
#include "stdlib.h"
#include <string.h>
#define TRUE 1
#define FALSE 0
#define OK 0
#define ERROR -1
#define INFEASIBLE -2
#define MAX 100
#define OVERFLOW -3
#define LISTSIZE 20//支持在20个树中操作
#define ARRSIZE 20

typedef int status;
typedef int KeyType;
typedef struct {
                KeyType  key;
                char others[ARRSIZE];
} TElemType; //二叉树结点类型定义


typedef struct BiTNode{  //二叉链表结点的定义
                TElemType  data;
                struct BiTNode *lchild,*rchild;
} BiTNode, *BiTree;

typedef struct {
        BiTree T;//创建二叉树用的指针T
        char name[ARRSIZE];//用于保存树的名称
}LElemType;

```

```
33 typedef struct {
34         LElemType tree[LISTSIZE];//多个树进行操作
35         int length;
36         int listsize;
37 }SqList;//线性表保存多个树
38
39
40 static int j,end=0,fail=0;
41 static int check[MAX]={0};
42 int nowkey;
43
44 /****************function definition*****************/
45 /**
46  * 函数名称: InitBiTree
47  * 初始条件: 二叉树T不存在
48  * 操作结果: 构造空树二叉树T
49  */
50 status InitBiTree(BiTree &T) {
51         T = (BiTree)malloc(sizeof(BiTNode));
52         T-> lchild = NULL;
53         T -> rchild = NULL;
54         strcpy(T -> data.others,"#");
55         T -> data.key = 0;//初始化二叉树,将左右指针指向空
56         return OK;
57 }
58
59 /**
60  * 函数名称: create
61  * 操作结果: 根据definition先序序列递归构造二叉树结点
62  * 函数功能: 服务于CreateBiTree函数
63  */
64 void create(BiTree &T,TElemType definition[])
65 {
66     if(end || fail){
67         return;
```

```
68      }
69      nowkey=definition[j++].key;
70      if(nowkey==0){
71          T=NULL;
72      }else if(nowkey==-1){
73          end=1;
74          return;
75      }else{
76          if(check[nowkey]){
77              fail=1;
78              return;
79          }else{
80              check[nowkey]=1;
81              T = (BiTree)malloc(sizeof(BiTNode));
82              T->data=definition[j-1];
83              create(T->lchild,definition);
84              create(T->rchild,definition);
85          }
86      }
87      return;
88  }
89
90  /**
91   * 函数名称: DestroyBiTree
92   * 初始条件: 二叉树T存在
93   * 操作结果: 销毁二叉树T
94   */
95  void DestroyBiTree(BiTree &T)
96  {
97      if(T){
98          DestroyBiTree(T->lchild);
99          DestroyBiTree(T->rchild);
100         free(T);
101         T=NULL;
102     }
```

```
103     return;
104 }
105
106 /**
107  * 函数名称：CreateBiTree
108  * 初始条件：二叉树T存在
109  * 操作结果：操作结果是按definition构造二叉树T
110  * 返回值类型：status类型
111  * 返回值：如果关键字重复返回ERROR，创建成功返回OK
112  */
113 status CreateBiTree(BiTree &T,TElemType definition[])
114 {
115     nowkey=definition[0].key;
116     j = 0;
117     create(T,definition);
118     if(fail){
119         return ERROR;
120     }
121     return OK;
122 }
123
124 /**
125  * 函数名称：FreeNode
126  * 函数功能：服务于ClearBiTree函数
127  */
128 void FreeNode(BiTree &T)
129 {
130     if(T==NULL){
131         return;
132     }
133     FreeNode(T->lchild);
134     FreeNode(T->rchild);
135     free(T);
136     T=NULL;
137 }
```

```
138
139 /**
140  * 函数名称: ClearBiTree
141  * 初始条件: 二叉树T已存在
142  * 操作结果: 清空二叉树
143  */
144 status ClearBiTree(BiTree &T)
145 {
146     if(T==NULL){
147         return ERROR;
148     }
149     FreeNode(T);
150     InitBiTree(T);
151     return OK;
152 }
153
154 /**
155  * 函数名称: max
156  * 操作结果:返回两数最大值
157  * 函数功能: 服务于BiTreeDepth函数
158  */
159 int max(int a ,int b)
160 {
161     return a > b?a:b;
162 }
163
164 /**
165  * 函数名称: BiTreeDepth
166  * 初始条件: 二叉树T存在
167  * 操作结果: 返回二叉树的深度
168  */
169 int BiTreeDepth(BiTree T)
170 {
171     if(T==NULL){
172         return 0;
```

```
173     }
174     return 1+max(BiTreeDepth(T->lchild),BiTreeDepth(T->rchild));
175 }
176 
177 /**
178  * 函数名称: LocateNode
179  * 初始条件: 二叉树T存在
180  * 操作结果: 操作结果是返回查找到的结点指针, 如无关键字为e的结
        点, 返回NULL;
181  * 函数参数: e是和T中结点关键字类型相同的给定值;
182  */
183 BiTNode* LocateNode(BiTree T,KeyType e)
184 {
185     BiTree result=NULL,queue[100],now;
186     int front=0,rear=1;
187     if(T==NULL){
188         return result;
189     }
190     queue[front]=T;
191     while(rear > front){
192         now=queue[front++];
193         if(now->data.key==e){
194             return now;
195         }
196         if(now->lchild!=NULL){
197             queue[rear++]=now->lchild;
198         }
199         if(now->rchild!=NULL){
200             queue[rear++]=now->rchild;
201         }
202     }
203     return result;
204 }
205 
206 /**
```

```
207  * 函数名称：locate_parent
208  * 初始条件：二叉树T存在
209  * 操作结果：前序遍历搜索关键字为e的结点，保存在p中，并且把该节点
         的父节点存在parent_p中
210  */
211 void locate_parent(BiTree T, BiTree parent, KeyType e, BiTree& p,
        BiTree& parent_p)
212 {
213     if (T) {
214         if (T->data.key == e) {
215             p = T;
216             parent_p = parent;
217             return;
218         }
219         locate_parent(T->lchild, T, e, p, parent_p);
220         locate_parent(T->rchild, T, e, p, parent_p);
221     }
222 }
223
224 /**
225  * 函数名称：Assign
226  * 初始条件：二叉树T已存在
227  * 操作结果：关键字为e的结点赋值为value
228  * 函数参数：e是和T中结点关键字类型相同的给定值
229  * 返回值：如果赋值后出现关键字重复，返回ERROR，复制成功返回OK
230  */
231 status Assign(BiTree &T, KeyType e, TElemType value) {
232     BiTNode* p = LocateNode(T, e);
233     if (p == NULL) {
234         return INFEASIBLE;
235     }
236     BiTNode* search = LocateNode(T, value.key);
237     if (search != NULL && search != p) {
238         return ERROR;
239     }
```

```
240     p->data = value;
241     return OK;
242 }
243
244 /**
245  * 函数名称: GetSibling
246  * 初始条件: 二叉树T存在
247  * 操作结果: 获取关键字为e的节点的兄弟节点
248  */
249 BiTNode* GetSibling(BiTree T, KeyType e) {
250     if (!T) {
251         return NULL;
252     }
253     BiTNode* p = NULL;
254     BiTNode* parent_p = NULL;
255     locate_parent(T, NULL, e, p, parent_p);
256     if (!p || !parent_p || !(parent_p->lchild && parent_p->rchild
        )) {
257         return NULL;
258     }
259     if (p == parent_p->lchild) {
260         return parent_p->rchild;
261     } else {
262         return parent_p->lchild;
263     }
264 }
265 /**
266  * 函数名称: BiTreeEmpty
267  * 初始条件: 无
268  * 操作结果: 二叉树判空
269  *
270  */
271 status BiTreeEmpty(BiTree T) {
272         if(strcmp(T->data.others ,"#")!=0) return FALSE;
273         else return TRUE;
```

```
274 }
275
276 /**
277  * 函数名称: InsertNode
278  * 初始条件: 二叉树T存在
279  * 操作结果: 根据LR为0或者1, 插入结点c到T中,
280  *           作为关键字为e的结点的左或右孩子结点,
281  *           结点e的原有左子树或右子树则为结点c的右子树;
282  * 函数参数: e是和T中结点关键字类型相同的给定值, LR为0或1, c是待
         插入结点;
283  *           LR为-1时, 作为根结点插入, 原根结点作为c的右子树。
284  * 返回值类型: status类型
285  * 返回值: 值二叉树为空/找不到目标节点/插入节点后关键字重复, 返回
         ERROR;
286  *          插入成功返回OK
287  *
288  */
289 status InsertNode(BiTree &T,KeyType e,int LR,TElemType c)
290 {
291     BiTree insert=NULL,newnode,repeat;
292     newnode=(BiTNode*)malloc(sizeof(BiTNode));
293     newnode->lchild=NULL;
294     newnode->rchild=NULL;
295     if(LR==-1){
296         newnode->data=c;
297         newnode->rchild=T;
298         T=newnode;
299         return OK;
300     }
301     insert=LocateNode(T,e);
302     repeat=LocateNode(T,c.key);
303     if(insert==NULL){
304         return INFEASIBLE;
305     }
306     if(repeat!=NULL){
```

```
307         return ERROR;
308     }
309     newnode->data=c;
310     switch(LR){
311         case 0:
312             newnode->rchild=insert->lchild;
313             insert->lchild=newnode;
314             break;
315         case 1:
316             newnode->rchild=insert->rchild;
317             insert->rchild=newnode;
318             break;
319     }
320     return OK;
321 }
322
323 /**
324  * 函数名称: DeleteNode
325  * 初始条件: 二叉树T已存在
326  * 操作结果: 删除T中关键字为e的结点;
327  *           同时, 如果关键字为e的结点度为0, 删除即可;
328  *           如关键字为e的结点度为1, 用关键字为e的结点孩子代替被删
     除的e位置;
329  *           如关键字为e的结点度为2, 用e的左孩子代替被删除的e位
     置,
330  *           e的右子树作为e的左子树中最右结点的右子树;
331  *
332  */
333 status DeleteNode(BiTree &T,KeyType e)
334 {
335     int du=0,LR=0,lchildflag=0, rchildflag=0;
336     if(T==NULL){
337         return ERROR;
338     }
339     BiTree p = NULL ,search;
```

```
340     BiTNode* parent_p = NULL;
341     locate_parent(T, NULL, e, p, parent_p);
342     if(parent_p && parent_p->rchild==p){//parent_p=NULL,p为根节点
343         LR=1;
344     }
345     if(p==NULL){
346         return ERROR;//找不到目标节点
347     }
348
349     if(p->lchild){
350         du++;
351         lchildflag++;
352     }
353     if(p->rchild){
354         du++;
355         rchildflag++;
356     }
357     if(p==T){//删除根节点的情况单独处理
358         switch (du){
359             case 0:
360                 free(T);
361                 T=NULL;
362                 break;
363             case 1:
364                 if(lchildflag==1){
365                     T=T->lchild;
366                 }else{
367                     T=T->rchild;
368                 }
369                 free(p);
370                 break;
371             case 2:
372                 T=T->lchild;
373                 for(search=p->lchild;search->rchild;search=search
                        ->rchild);
```

```
374                 search->rchild=p->rchild;
375                 free(p);
376                 break;
377         }
378         return OK;
379     }
380     switch(du){
381         case 0:
382             if(LR==0){
383                 parent_p->lchild=NULL;
384             }else{
385                 parent_p->rchild=NULL;
386             }
387             free(p);
388             break;
389         case 1:
390             if(LR==0){
391                 if(lchildflag){
392                     parent_p->lchild=p->lchild;
393                 }else{
394                     parent_p->lchild=p->rchild;
395                 }
396             }else{
397                 if(lchildflag){
398                     parent_p->rchild=p->lchild;
399                 }else{
400                     parent_p->rchild=p->rchild;
401                 }
402             }
403             free(p);
404             break;
405         case 2:
406             for(search=p->lchild;search->rchild;search=search->
                    rchild);
407             search->rchild=p->rchild;
```

```
408             if(LR==0){
409                 parent_p->lchild=p->lchild;
410             }else{
411                 parent_p->rchild=p->lchild;
412             }
413             free(p);
414             break;
415     }
416     return OK;
417 }
418 
419 
420 /**
421  * 函数名称: visit
422  * 初始条件: 二叉树存在
423  * 操作结果: 打印结点关键字
424  */
425 void visit(BiTree T)
426 {
427     if(T){
428         printf("%d %s\n",T->data.key,T->data.others);
429     }
430     return;
431 }
432 /**
433  * 函数名称: PreOrderTraverse
434  * 初始条件: 二叉树T存在
435  * 操作结果: 先序遍历, 对每个结点调用函数Visit一次且一次, 一旦调
        用失败, 则操作失败。
436  */
437 void PreOrderTraverse(BiTree t,void (*visit)(BiTree))
438 {
439     BiTree stack[MAX],p;
440     int top=0;//置空栈
441     if(t!=NULL){
```

```
442         stack[top++]=t;
443         while(top){
444             p = stack[--top];
445             //根节点退栈，访问根节点，左右子树入栈
446             visit(p);
447             if(p->rchild!=NULL){
448                 stack[top++]=p->rchild;
449             }
450             if(p->lchild!=NULL){
451                 stack[top++]=p->lchild;
452             }
453         }
454     }
455 }
456
457
458 /**
459  * 函数名称：InOrderTraverse
460  * 初始条件：二叉树T存在
461  * 操作结果：中序遍历t，对每个结点调用函数Visit一次且一次，一旦调
        用失败，则操作失败；
462  */
463 void InOrderTraverse(BiTree T,void (*visit)(BiTree))
464 {
465     if(T==NULL){
466         return;
467     }
468     InOrderTraverse(T->lchild ,visit);
469     visit(T);
470     InOrderTraverse(T->rchild ,visit);
471     return;
472 }
473
474 /**
475  * 函数名称：PostOrderTraverse
```

```
476  * 初始条件：二叉树T存在
477  * 操作结果：序遍历t，对每个结点调用函数Visit一次且一次，一旦调用
         失败，则操作失败。
478  */
479 void PostOrderTraverse(BiTree T,void (*visit)(BiTree))
480 {
481     if(T==NULL){
482         return;
483     }
484     PostOrderTraverse(T->lchild,visit);
485     PostOrderTraverse(T->rchild,visit);
486     visit(T);
487     return;
488 }
489
490 /**
491  * 函数名称：LevelOrderTraverse
492  * 初始条件：二叉树T存在
493  * 操作结果：层序遍历t，对每个结点调用函数Visit一次且一次，一旦调
         用失败，则操作失败。
494  */
495
496 void LevelOrderTraverse(BiTree T,void (*visit)(BiTree))
497 {
498     BiTree queue[100];
499     BiTNode* now;
500     int front=0,rear=1;
501     if(T==NULL){
502         return;
503     }
504     queue[front]=T;
505     while(rear > front){
506         now=queue[front++];
507         visit(now);
508         if(now->lchild!=NULL){
```

```
509             queue[rear++]=now->lchild;
510         }
511         if(now->rchild!=NULL){
512             queue[rear++]=now->rchild;
513         }
514     }
515     return;
516 }
517 
518 /***********added function*****************/
519 
520 /**
521  * 函数名称: SaveBiTree
522  * 初始条件: 二叉树T存在
523  * 操作结果: 将二叉树T的数据以先序保存在文件中
524  */
525 status SaveBiTree(BiTNode *T,char FileName[]) { //非递归先序储存
526         FILE *fp;
527         BiTree pforsave=T;
528         BiTree stack[100];
529         int top=0;
530         if((fp=fopen(FileName,"wb"))==NULL) {
531                 //printf("存储发生错误\n");
532                 return ERROR;
533         }
534         do {
535                 while(pforsave) {
536                         if(top==MAX){
537                 return OVERFLOW;
538             }
539                         stack[top++]=pforsave;
540                         fprintf(fp," %d ",pforsave->data.key);
541                         fprintf(fp," %s ",pforsave->data.others);
542                         pforsave=pforsave->lchild;
543                 }
```

```
544                     fprintf(fp," %d ",0);
545                     if(top) {
546                             top--;
547                             pforsave=stack[top]->rchild;
548                     }
549             } while(top||pforsave);
550             fprintf(fp," %d ",0);
551             fclose(fp);
552             return OK;
553     }
554
555     /**
556      * 函数名称: read
557      * 初始条件: 文件存在
558      * 操作结果: 递归将文件中的数据读取到二叉树T中
559      * 函数功能: 服务于LoadBiTree
560      */
561     void read(BiTree &T,FILE *fp) {
562             int definition;
563             fscanf(fp," %d ",&definition);
564             if(!definition) T=NULL;
565             else {
566                     T=(BiTNode*)malloc(sizeof(BiTNode));
567                     T->data.key=definition;
568                     fscanf(fp," %s ",T->data.others);
569                     read(T->lchild,fp);
570                     read(T->rchild,fp);
571             }
572     }
573
574     /**
575      * 函数名称: LoadBiTree
576      * 初始条件: 文件存在
577      * 操作结果: 将文件中的数据读取到二叉树T中
578      */
```

```
579 status LoadBiTree(BiTree &T,char FileName[]) {
580         FILE *fp;
581         if ((fp=fopen(FileName,"rb"))==NULL) {
582                 //printf("读取发生错误\n");
583                 return ERROR;
584         }
585         read(T,fp);
586         fclose(fp);
587         return OK;
588 }
589
590  /**
591  * 函数名称: MaxPathSum
592  * 初始条件: 二叉树T存在
593  * 操作结果: 返回二叉树中的最大路径和, 从一个结点到另一个节点的路
        径
594  * 算法思路: 递归的求左右子树的最大路径和, 最后加上该结点的值。
595  */
596 int MaxPathSum(BiTree T, int &maxSum)
597 {
598     if (T == NULL){
599          return 0;
600     }
601     int left = MaxPathSum(T->lchild, maxSum);
602     int right = MaxPathSum(T->rchild, maxSum);
603     if(left < 0)left=0;
604     if(right < 0)right=0;
605     if(T->data.key+max(left,right) > maxSum){
606         maxSum=T->data.key+max(left,right);
607     }
608     return T->data.key + max(left, right);
609 }
610
611 /**
612  * 函数名称: LowestCommonAncestor
```

```
613  * 初始条件：二叉树T存在
614  * 函数参数：二叉树T，关键字e1，e2
615  * 操作结果：该二叉树中e1节点和e2节点的最近公共祖先
616  * 函数思路：如果e1和e2分别是T的左右子树的节点，
617  *           那么T就是他们的最近公共祖先
618  *            如果e1和e2都是T的左子树的节点，
619  *            则在T的左子树中继续寻找他们的最近公共祖先
620  *            如果e1和e2都是T的右子树的节点，
621  *            则在T的右子树中继续寻找他们的最近公共祖先
622  */
623 BiTree LowestCommonAncestor(BiTree T, KeyType e1, KeyType e2)
624 {
625    if(T==NULL || T->data.key==e1 || T->data.key==e2)
626        return T;
627     BiTree left = LowestCommonAncestor(T->lchild, e1, e2);
628     BiTree right = LowestCommonAncestor(T->rchild, e1, e2);
629     if(left!=NULL && right!=NULL)
630         return T;
631     if(left!=NULL)
632         return left;
633     if(right!=NULL)
634         return right;
635     return NULL;
636 }
637
638 /**
639  * 函数名称：InvertTree
640  * 初始条件：线性表L已存在
641  * 操作结果：将T翻转，使其所有节点的左右节点互换
642  * 算法思路：递归翻转，在前序位置交换左右节点
643  */
644 void InvertTree(BiTree T)
645 {
646     if(T == NULL)
647         return;
```

```
    BiTree temp = T->lchild;
    T->lchild = T->rchild;
    T->rchild = temp;
    InvertTree(T->lchild);
    InvertTree(T->rchild);
}


/**
 * 函数名称: printTree
 * 初始条件: 线性表L已存在
 * 函数功能: 将树形图形象的打印出来
 */
void printTree(BiTree T, int type, int level)
{
        int i;
        if (T==NULL){
        return;
    }
        printTree(T->rchild, 2, level+1);
        switch (type)
        {
        case 0:
            printf("%2d%s\n", T->data.key,T->data.others);
            break;
        case 1:
            for (i = 0; i < level; i++)
                printf("\t");
            printf("\\%2d%s\n", T->data.key,T->data.others);
            break;
        case 2:
            for (i = 0; i < level; i++)
                printf("\t");
            printf("/%2d%s\n", T->data.key,T->data.others);
            break;
```

```
683     }
684         printTree(T->lchild,1,level+1);
685 }
```

main 函数 u3.cpp 如下：

```
/*main function of BiTree*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include "def.h"

/*function delclaration*/
void create(BiTree &T,TElemType definition[]);
void DestroyBiTree(BiTree &T);
void FreeNode(BiTree &T);
status ClearBiTree(BiTree &T);
status BiTreeEmpty(BiTree T);
int max(int a ,int b);
int BiTreeDepth(BiTree T);
BiTNode* LocateNode(BiTree T,KeyType e);
void locate_parent(BiTree T, BiTree parent, KeyType e, BiTree& p,
    BiTree& parent_p);
status Assign(BiTree &T, KeyType e, TElemType value);
BiTNode* GetSibling(BiTree T, KeyType e);
status InsertNode(BiTree &T,KeyType e,int LR,TElemType c);
status DeleteNode(BiTree &T,KeyType e);
void PreOrderTraverse(BiTree T,void (*visit)(BiTree));
void InOrderTraverse(BiTree T,void (*visit)(BiTree));
void PostOrderTraverse(BiTree T,void (*visit)(BiTree));
void LevelOrderTraverse(BiTree T,void (*visit)(BiTree));
int MaxPathSum(BiTree T, int &maxSum);
BiTree LowestCommonAncestor(BiTree T, KeyType e1, KeyType e2);
void InvertTree(BiTree T);
void read(BiTree &T,FILE *fp);
status SaveBiTree(BiTNode *T,char FileName[]);
status LoadBiTree(BiTree &T,char FileName[]);
void printTree(BiTree T, int type, int level);

```

```
// /**********global var(for creating the BiTree)**********/
// static int j,end=0,fail=0;
// static int check[MAX]={0};
// int nowkey;

int main(){
        char filename[30];//保存树的名称
        SqList L;//相当于用数组构建多树操作的框架
        BiTree p;
        int op=1;//op用来case,便于用户操作
        int e;//用于存查找结点的关键字
    int i, num=1 ,LR=0;
        FILE* fp;
        TElemType definition[MAX];//用于创建二叉树
        int index=0 ,createkey;//用于definition的索引
        TElemType newnode;//用于插入结点
    BiTree commonancestor;
        int e1,e2;
    int maxsum =INT_MIN;
        status opreateresult;

        //线性表初始化
        L.length = 0;
        L.listsize=LISTSIZE;
    for(i=0;i<20;i++)
    {
        L.tree[i].T = NULL;//相当于创建20个定义中类型的树的指针,
            同时指空, 即20个树;
    }
     while(op){
        system("cls");
    printf("\n\n");
        printf("\t\t      Menu for Binary Tree On Binary Linked
           List \n");
        printf("\t
```

```
            ************************************************\n");
67          printf("\t**\t\t1.  InitBiTree          \t  2.
               CreateBiTree \t\t**\n");
68          printf("\t**\t\t3.  DestroyBiTree       \t  4.
               ClearBiTree  \t\t**\n");
69          printf("\t**\t\t5.  BiTreeEmpty         \t  6.
               BiTreeDepth\t\t**\n");
70          printf("\t**\t\t7.  LocateNode          \t  8.  Assign \t\
               t\t**\n");
71          printf("\t**\t\t9.  GetSibling         \t  10. InsertNode\
               t\t**\n");
72          printf("\t**\t\t11. DeleteNode          \t  12.
               PreOrderTraverse\t\t**\n");
73          printf("\t**\t\t13. InOrderTraverse     \t  14.
               PostOrderTraverse\t\t**\n");
74          printf("\t**\t\t15. LevelOrderTraverse \t  16. MaxPathSum
               \t\t**\n");
75          printf("\t**\t\t17. LowestCommonAncestor  18. InvertTree\
               t\t**\n");
76          printf("\t**\t\t19. printTree
               \t\t**\n");
77          printf("\t**\t\t20. SaveBiTree      \t  21. LoadBiTree\t\t
               **\n");
78          printf("\t**\t\t22. Choose(多树操作)       0.  Exit\t\t\t
               **\n");
79          printf("\t**\t\t若已有文件，可通过函数21进行加载\t\t\t**\
               n");
80          printf("\t**    进行操作前，请先选择要操作的二叉树序号，默
               认在第一个树上进行操作\t**\n");
81          printf("\t
               ************************************************\n");
82          printf("\t\t\t请选择你的操作[0~22]:\n");
83          scanf("%d",&op);
84      switch(op)
85          {
```

```
86
87              case 1:
88              //初始化二叉树
89               InitBiTree((L.tree[num-1].T));
90               printf("\t\t\t二叉树创建成功! \n");
91                   getchar();getchar();
92                   break;
93
94              case 2:
95              //创建二叉树
96                   if (L.tree[num-1].T == NULL){
97                           printf("\t\t\t二叉树不存在! \n");
98                           getchar();getchar();
99                           break;
100                  }else if(!BiTreeEmpty(L.tree[num-1].T)){
101                          printf("\t\t\t二叉树不为空, 请在空二叉树
                                开始创建! \n");
102                          getchar();getchar();
103                          break;
104                  }
105                  fflush(stdin);
106                  printf("请用前序方式输入序列构建二叉树, \n每个结
                        点对应一个整型的关键字和一个字符串, \n当关键字
                        为0时, 表示空子树, 为-1表示输入结束\n");
107                  scanf("%d" ,&createkey);
108                  while(createkey!=-1){
109                          definition[index].key=createkey;
110                          scanf("%s" ,definition[index].others);
111                          index++;
112                          scanf("%d" ,&createkey);
113                  }
114                  fflush(stdin);
115                  printf("输入结束\n");
116                  if(CreateBiTree(L.tree[num-1].T,definition)==OK){
117                          L.length++;
```

```
118                     printf("\t\t\t二叉树构造成功! \n\t\t\t请
                            输入二叉树的名字\n");
119                     scanf("%s",&L.tree[num-1].name);
120                     printf("OK!你创建的二叉树名字是%s\n",L.
                            tree[num-1].name);
121             }else{
122                     ClearBiTree(L.tree[num-1].T);
123                     printf("输入了重复关键字, 构造失败\n");
124             }
125             for(j = 0 ;j < MAX ;j++){
126                     check[j]=0;
127             }
128             end=0;fail=0;
129             getchar();getchar();
130             break;
131
132             case 3:
133         //销毁二叉树
134             if (L.tree[num-1].T == NULL)
135             {
136                     printf("\t\t\t二叉树不存在! \n");
137                     getchar();getchar();
138                     break;
139             }
140          DestroyBiTree((L.tree[num-1].T));
141             L.length--;
142          printf("\t\t\t二叉树销毁成功!\n");
143             getchar();getchar();
144             break;
145
146         case 4:
147         //清空二叉树
148             if (L.tree[num-1].T == NULL)
149             {
150                     printf("\t\t\t二叉树不存在! \n");
```

```
151                     getchar();getchar();
152                     break;
153                 }
154              if(ClearBiTree(L.tree[num-1].T) == OK){
155                     printf("\t\t\t二叉树清空成功! \n");
156                 }else{
157                     printf("\t\t\t二叉树清空失败! \n");
158                 }
159                 getchar();getchar();
160                 break;
161
162             case 5:
163             //二叉树判空
164                 if (L.tree[num-1].T==NULL)
165                 {
166                     printf("\t\t\t二叉树不存在! \n");
167                     getchar();getchar();
168                     break;
169                 }
170              if(BiTreeEmpty(L.tree[num-1].T) == TRUE){
171                     printf("\t\t\t二叉树为空树! \n");
172                 }else{
173                     printf("\t\t\t二叉树不为空树! \n");
174                 }
175                 getchar();getchar();
176                 break;
177
178             case 6:
179             //求二叉树深度
180                 if (L.tree[num-1].T == NULL)
181                 {
182                     printf("\t\t\t二叉树不存在! \n");
183                     getchar();getchar();
184                     break;
185                 }
```

```
186             if(BiTreeEmpty(L.tree[num-1].T) == TRUE){
187                     printf("\t\t\t二叉树为空树! \n");
188             }else{
189                     printf("\t\t\t二叉树的深度为:%d\n",
                           BiTreeDepth(L.tree[num-1].T));
190             }
191             getchar();getchar();
192             break;
193
194         case 7:
195         //查找关键字为e的结点
196             if (L.tree[num-1].T == NULL)
197             {
198                     printf("\t\t\t二叉树不存在! \n");
199                     getchar();getchar();
200                     break;
201             }
202             printf("请输入要查找的关键字(注意是int类型)\n");
203             scanf("%d" ,&e);
204             p = LocateNode(L.tree[num-1].T,e);
205          if(p){
206                     printf("找到了! \n");
207                     printf("该结点关键字为%d,信息为%s" ,p->
                           data.key,p->data.others);
208             }else{
209                     printf("没找到o( 🄵 )o\n");
210             }
211             getchar();getchar();
212             break;
213
214         case 8:
215         //结点赋值
216             if (L.tree[num-1].T == NULL)
217             {
218                     printf("\t\t\t二叉树不存在! \n");
```

```
                getchar();getchar();
                break;
            }
         printf("\t\t\t请输入你要赋值的key:\n");
            scanf("%d", &e);
            printf("请输入你要修改后的关键字\n");
            scanf("%d" ,&newnode.key);
            printf("请输入你要修改后的信息\n");
            scanf("%s",newnode.others);
            opreateresult=Assign(L.tree[num-1].T,e,newnode);
            if(opreateresult==OK){
                printf("修改成功");
            }else if(opreateresult==ERROR){
                printf("修改后会存在关键字重复，修改失败\
                   n");
            }else{
                printf("要赋值的结点不存在");
            }
            getchar();getchar();
            break;

        case 9:
        //查找节点的兄弟
            if (L.tree[num-1].T == NULL)
            {
                printf("\t\t\t二叉树不存在! \n");
                getchar();getchar();
                break;
            }
         printf("\t\t\t请输入你要查找的key:\n");
            scanf("%d", &e);

            if(LocateNode(L.tree[num-1].T,e)==NULL){
                printf("该节点不存在\n");
                getchar();getchar();
```

```
253                     break;
254             }
255             p = GetSibling(L.tree[num-1].T,e);
256             if(p==NULL){
257                     printf("该节点不存在兄弟节点\n");
258             }else{
259                     printf("找到了, 该节点的兄弟结点关键字为%
                            d,信息为%s\n",p->data.key,p->data.
                            others);
260             }
261             getchar();getchar();
262             break;
263
264     case 10:
265     //插入结点
266             if (L.tree[num-1].T == NULL)
267             {
268                     printf("\t\t\t二叉树不存在! \n");
269                     getchar();getchar();
270                     break;
271             }
272             printf("请输入你要插入的结点的父结点的关键字\n");
273             scanf("%d" ,&e);
274             printf("请输入LR: 0-->左边, 1-->右边, -1-->在根节
                    点插入\n");
275             scanf("%d" ,&LR);
276             if(LR!=0 &&LR!=1 &&LR!=-1){
277                     printf("LR输入错误! \n");
278                     getchar();getchar();
279                     break;
280             }
281             printf("请输入你要插入的结点的关键字,类型必须是
                    int\n");
282             scanf("%d" ,&newnode.key);
283             printf("请输入你要插入的结点的信息\n");
```

```
284                 scanf("%s" ,newnode.others);
285                 opreateresult=InsertNode(L.tree[num-1].T,e,LR,
                        newnode);
286                 if(opreateresult==INFEASIBLE){
287                         printf("你要插入的父节点不存在\n");
288                 }else if(opreateresult==ERROR){
289                         printf("你要插入的关键字有重复\n");
290                 }else{
291                         printf("插入成功\n");
292                 }
293                 getchar();getchar();
294                 break;
295
296             case 11:
297         //删除结点
298                 if (L.tree[num-1].T == NULL){
299                         printf("\t\t\t二叉树不存在! \n");
300                         getchar();getchar();
301                         break;
302                 }
303                 printf("请输入你要删除的结点的关键字\n");
304                 scanf("%d" ,&e);
305                 if(DeleteNode(L.tree[num-1].T,e)==ERROR){
306                         printf("你要删除的结点不存在\n");
307                 }else{
308                         printf("删除成功\n");
309                 }
310                 getchar();getchar();
311                 break;
312
313             case 12:
314         //先序遍历
315                 if(L.tree[num-1].T==NULL){
316                         printf("二叉树不存在\n");
317                         getchar();getchar();
```

```
318                     break;
319             }
320             PreOrderTraverse(L.tree[num-1].T,visit);
321             getchar();getchar();
322                     break;
323
324         case 13:
325      //中序遍历
326             if(L.tree[num-1].T==NULL){
327                     printf("二叉树不存在\n");
328                     getchar();getchar();
329                     break;
330             }
331             InOrderTraverse(L.tree[num-1].T,visit);
332             getchar();getchar();
333                     break;
334         case 14:
335      //后序遍历
336             if(L.tree[num-1].T==NULL){
337                     printf("二叉树不存在\n");
338                     getchar();getchar();
339                     break;
340             }
341             PostOrderTraverse(L.tree[num-1].T,visit);
342             getchar();getchar();
343                     break;
344
345         case 15:
346      //层序遍历
347             if(L.tree[num-1].T==NULL){
348                     printf("二叉树不存在! \n");
349                     getchar();getchar();
350                     break;
351             }
352             LevelOrderTraverse(L.tree[num-1].T,visit);
```

```
353                 getchar();getchar();
354                         break;
355
356             case 16:
357          //最大路径和，从根节点到叶子结点的最大路径和
358                 maxsum=INT_MIN;
359                 if(L.tree[num-1].T==NULL){
360                         printf("二叉树不存在! \n");
361                 }else{
362                         MaxPathSum(L.tree[num-1].T,maxsum);
363                         printf("最大路径和为%d\n",maxsum);
364                 }
365                 getchar();getchar();
366                         break;
367
368             case 17:
369         //两个节点的最近公共祖先
370                 if(L.tree[num-1].T==NULL){
371                         printf("二叉树不存在! \n");
372                 }else{
373                         printf("请输入你要查找的两个结点的关键字\
                            n");
374                         scanf("%d %d",&e1,&e2);
375                         commonancestor=LowestCommonAncestor(L.
                            tree[num-1].T,e1,e2);
376                         if(commonancestor==NULL){
377                                 printf("你要查找的两个结点不存
                                    在! \n");
378                         }else{
379                                 printf("最近公共祖先关键字为%d,信
                                    息为%s\n",commonancestor->data
                                    .key,commonancestor->data.
                                    others);
380                         }
381                 }
```

```
382                 getchar();getchar();
383                         break;
384
385             case 18:
386         //将二叉树所有节点的左右子树反转
387                 if(L.tree[num-1].T==NULL){
388                         printf("二叉树不存在! \n");
389                 }else{
390                         InvertTree(L.tree[num-1].T);
391                         printf("反转成功! \n");
392                 }
393                 getchar();getchar();
394                         break;
395
396                 case 19:
397                 //打印二叉树
398                 if(L.tree[num-1].T==NULL){
399                         printf("二叉树不存在! \n");
400                 }else if(BiTreeEmpty(L.tree[num-1].T)){
401                         printf("");
402                 }else{
403                         printTree(L.tree[num-1].T,0,0);
404                 }
405                 getchar();getchar();
406                         break;
407
408
409             case 20:
410         //保存二叉树
411                 if(L.tree[num-1].T==NULL){
412                         printf("二叉树不存在! \n");
413                 }else{
414                         printf("请输入你要保存的文件名\n");
415                         scanf("%s" ,filename);
416                         SaveBiTree(L.tree[num-1].T,filename);
```

```
417                     printf("保存成功! 已将二叉树以先序序列保
                            存在文件%s中\n",filename);
418             }
419             getchar();getchar();
420                     break;
421
422         case 21:
423     //加载二叉树
424             printf("请输入你要加载的文件名\n");
425             scanf("%s" ,filename);
426             if((fp=fopen(filename,"r"))==NULL){
427                     printf("打开文件失败! \n");
428             }else{
429                     LoadBiTree(L.tree[num-1].T,filename);
430                     printf("加载成功! 已将文件%s中的二叉树加
                            载到第%d棵二叉树中\n",filename,num);
431             }
432             getchar();getchar();
433                     break;
434
435         case 22:
436         //choose
437             printf("\t\t\t在第几个树操作?只支持%d个树进行操作
                    : \n",LISTSIZE);
438             scanf("%d",&num);
439             if(num<1||num>20){
440                     printf("\t\t\t请选择正确范围! 已默认在第
                            一个树操作\n");
441                     num=1;
442             }else{
443                     printf("\t\t\t你选择了第%d个树\n",num);
444             }
445             getchar(); getchar();
446             break;
447         case 0:
```

```
448             break;
449             }//end of switch
450       }//end of while
451         printf("\n");
452             printf("\t\t\t欢迎下次再使用本系统! \n\n");
453             getchar();
454   }//end of main()
```

附录 D 基于邻接表图实现的源程序

头文件 func.h 如下：

```
/*definition of function of graph*/
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#define TRUE 1
#define FALSE 0
#define INFEASIBLE -2
#define MAX_VERTEX_NUM 20
#define graphnum 10
#define MAX_VERTEX_NUM 20
#define MAX_ARC_NUM 40
#define ERROR -1
#define OK 1
#define OVERFLOW -2

/*队列*/
typedef int ElemType;
typedef struct node{
    ElemType data;
    struct node * next;
}node,*nodeptr;
typedef struct {
    nodeptr head;//表头结点,不存数据
    nodeptr rear;
}queue,*queueptr;

/*栈*/
typedef struct stack
{
    ElemType elem[MAX_VERTEX_NUM];
    int p;
} stack; //栈的定义
```

```
33
34
35  typedef int status;
36  typedef int KeyType;
37  typedef enum {DG,DN,UDG,UDN} GraphKind;
38  typedef struct {
39          KeyType  key;
40          char others[20];
41  } VertexType; //顶点类型定义
42
43  typedef struct ArcNode {          //表结点类型定义
44          int adjvex;               //顶点位置编号
45          struct ArcNode  *nextarc;          //下一个表结点指针
46  } ArcNode;
47
48  typedef struct VNode{                              //头结点及其数组
        类型定义
49          VertexType data;         //顶点信息
50          ArcNode *firstarc;                //指向第一条弧
51  } VNode,AdjList[MAX_VERTEX_NUM];
52
53  typedef  struct {  //邻接表的类型定义
54      AdjList vertices;             //头结点数组
55      int vexnum,arcnum;             //顶点数、弧数
56      GraphKind  kind;         //图的类型
57  } ALGraph;
58  status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2]);
59  status DestroyGraph(ALGraph &G);
60  status LocateVex(ALGraph G,KeyType u);
61  status PutVex(ALGraph &G,KeyType u,VertexType value);
62  status FirstAdjVex(ALGraph G,KeyType u);
63  status NextAdjVex(ALGraph G,KeyType v,KeyType w);
64  status InsertVex(ALGraph &G,VertexType v);
65  status DeleteVex(ALGraph &G,KeyType v);
66  status InsertArc(ALGraph &G,KeyType v,KeyType w);
```

```
67 status DeleteArc(ALGraph &G,KeyType v,KeyType w);
68 void dfshelper(const ALGraph &G,int i ,int *visited,void (*visit)
       (VertexType));
69 status DFSTraverse(const ALGraph& G,void (*visit)(VertexType));
70 status BFSTraverse(const ALGraph& G,void (*visit)(VertexType));
71 void visit(VertexType v);
72 status SaveGraph(ALGraph G, char FileName[]);
73 status LoadGraph(ALGraph &G, char FileName[]);
74 status VerticesSetLessThanK(ALGraph G,int v,int k,int *result,int
       *count);
75 int ShortestPathLength(ALGraph G, KeyType v, KeyType w) ;
76 void DFS(ALGraph G, int v, int visited[]);
77 int ConnectedComponentsNums(ALGraph G);
78
79 /*栈和队列的函数说明*/
80 void iniStack(stack &S);
81 int isEmptyStack(stack &S);
82 int push(stack &S, ElemType e);
83 int pop(stack &S);
84 void initqueue(queueptr q);
85 status enqueue(queueptr q,ElemType x);
86 status dequeue(queueptr q ,ElemType &e);
87 int isempty(queueptr q);
88
89 /*队列函数*/
90 void initqueue(queueptr q)
91 {
92     q->head=(nodeptr)malloc(sizeof(node));
93     q->head->next=NULL;
94     q->rear=q->head;
95 }
96
97 status enqueue(queueptr q,ElemType x)
98 {
99     nodeptr newnode=(nodeptr)malloc(sizeof(node));
```

```
100     if(newnode==NULL){
101         return OVERFLOW;
102     }
103     newnode->data=x;
104     newnode->next=NULL;
105     q->rear->next=newnode;
106     q->rear=newnode;
107     return OK;
108 }
109 status dequeue(queueptr q ,ElemType &e)
110 {
111     if(q->head->next==NULL){
112         return ERROR;
113     }
114     nodeptr p=q->head->next;
115     e=p->data;
116     q->head->next=p->next;
117     if(q->rear==p){
118         q->rear=q->head;
119     }
120     free(p);
121     return OK;
122 }
123 int isempty(queueptr q)
124 {
125     if(q->head->next==NULL){
126         return 1;
127     }
128     return 0;
129 }
130
131 /*栈函数*/
132 void iniStack(stack &S)
133 //该函数初始化栈S
134 {
```

```
135     S.p = 0;
136 }
137 int isEmptyStack(stack &S)
138 //判断是不是空栈，是则返回1，不是则返回0
139 {
140     if (S.p)
141         return 0;
142     else
143         return 1;
144 }
145 int push(stack &S, ElemType e)
146 //该函数将元素进栈，成功则返回1，失败返回0
147 {
148     if (S.p == MAX_VERTEX_NUM - 1)
149         return OVERFLOW;
150     else
151     {
152         S.elem[++S.p] = e;
153     }
154     return OK;
155 }
156 int pop(stack &S)
157 //该函数将元素出栈，返回出栈的元素值
158 {
159     if (S.p == 0)
160         return ERROR;
161     else
162         return S.elem[S.p--];
163 }
164
165
166 /************************
167  * 函数名称：CreateGraph
168  * 初始条件：图G不存在
169  * 操作结果：创建图G
```

```
 ************************/
status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2]) {
        int m = 0,n,one = -1,two = -1 ;
    G.vexnum=G.arcnum=0;
    //添加头结点
    while(V[m].key != -1){
                G.vertices[m].data = V[m];
                G.vertices[m].firstarc = NULL;
        m++;
        if(m > MAX_VERTEX_NUM) return ERROR;
    }

        G.vexnum = m;
    if(G.vexnum==0){
        return ERROR;
    }
    for(m = 0; m < G.vexnum; m++) {
                for(n = m+1; n < G.vexnum ; n++) {
                        if(V[m].key == V[n].key)
                                return ERROR;
                }
        }

        m = 0;
        ArcNode *p;

    //保存VR边信息的数组，VR[i][0]和VR[i][1]分别表示第i条边的两个
        顶点的key值
        while (VR[m][0] != -1) {
                one = -1;two=-1;//one和two分别表示第i条边的两个顶
                    点在头结点数组中的位置
                for(n = 0; n < G.vexnum; n++){
            if (VR[m][0] == G.vertices[n].data.key) {
                                one = n;
                                break;
```

```
203                         }
204         }
205         if(n==G.vexnum){
206                 return ERROR;
207         }
208                 for(n = 0; n < G.vexnum; n++){
209             if (VR[m][1] == G.vertices[n].data.key) {
210                                 two = n;
211                                 break;
212                         }
213         }
214         if(n==G.vexnum){
215                 return ERROR;
216         }
217         if(one==two){
218             return ERROR;
219         }
220
221         G.arcnum++;
222         //头插法
223                 p = (ArcNode*)malloc(sizeof(ArcNode));
224                 p -> adjvex = two;
225                 p -> nextarc = G.vertices[one].firstarc;
226                 G.vertices[one].firstarc = p;
227
228                 p = (ArcNode*)malloc(sizeof(ArcNode));
229                 p -> adjvex = one;
230                 p -> nextarc = G.vertices[two].firstarc;
231                 G.vertices[two].firstarc = p;
232                 m++;
233         }
234     G.kind = UDG;
235     return OK;
236 }
237
```

```
238 /*************************
239  * 函数名称: DestroyGraph
240  * 初始条件: 图G已存在
241  * 操作结果: 销毁图G
242  *************************/
243 status DestroyGraph(ALGraph &G) {
244         int i;
245         ArcNode *p, *q;//边类型的指针p,q
246         for(i = 0; i < G.vexnum; i++) {
247                 p = G.vertices[i].firstarc;//p指向第i个节点的 第
                        一条依附于该节点的指针
248                 while(p) { //p不为空,就依次将q指向下一条弧,释放p,
                        最后再将p指向q, 开始下一次循环
249                         q = p->nextarc;
250                         free(p);
251                         p = q;
252                 }
253         }
254         G.vexnum = 0;//之后将图的信息,顶点数目,边数目,kind置为DG
255         G.arcnum = 0;
256     G.kind=DG;
257         return OK;
258 }
259
260 /************************************************
261  * 函数名称: LocateVex
262  * 初始条件: 图G存在, u和G中的顶点具有相同特征
263  * 函数功能: 若u在图G中存在, 返回顶点u的索引, 否则返回ERROR
264  ************************************************/
265 status LocateVex(ALGraph G,KeyType u) {
266         int i;
267         for(i=0; i<G.vexnum; i++) {
268                 if(u == G.vertices[i].data.key)//判断节点名称是否
                        相同
269                         return i;//如果查找成功, 返回i的值
```

```
270         }
271         return ERROR;
272 }
273
274
275 /*************************************
276  * 函数名称: PutVex
277  * 初始条件: 图G存在, v是G中的某个顶点
278  * 函数功能: 根据u在图G中查找顶点v, 对v赋值value
279  * 返回值: 如果v在图G中存在, 返回OK, 如果查找失败或关键字不唯一,
         返回ERROR
280  *************************************/
281 status PutVex(ALGraph &G,KeyType u,VertexType value)
282 {
283     int i ,j;
284     for(i = 0 ;i < G.vexnum ;i++){
285         if(G.vertices[i].data.key == u){
286             for(j = 0 ; j < G.vexnum ;j++){
287                 if(G.vertices[j].data.key == value.key && i != j
                        ){
288                     return ERROR;
289                 }
290             }
291             G.vertices[i].data = value;
292             return OK;
293         }
294     }
295     return ERROR;
296 }
297
298 /*****************************************************
299  * 函数名称: FirstAdjVex
300  * 初始条件: 图G存在, v是G中的某个顶点
301  * 函数功能: 返回v的第一个邻接顶点, 如果v没有邻接顶点, 返回ERROR
302  *****************************************************/
```

```
303 int FirstAdjVex(ALGraph G,KeyType u)
304 {
305     int i;
306     ArcNode *p;
307     i = LocateVex(G, u);
308     if(i==-1){//如果没有找到该节点，返回ERROR
309         return ERROR;
310     }
311     p = G.vertices[i].firstarc;
312     if(p){
313         return p->adjvex;
314     }else{//如果没有邻接点，返回ERROR
315         return ERROR;
316     }
317 }
318 
319 /*************************************************
320  * 函数名称：NextAdjVex
321  * 初始条件：图G存在，v是G的一个顶点,w是v的邻接顶点
322  * 函数功能：返回v的（相对于w）下一个邻接顶点，
323              如果w是最后一个邻接顶点，返回ERROR
324  *************************************************/
325 int NextAdjVex(ALGraph G,KeyType v,KeyType w)
326 {
327     int i;
328     for(i = 0 ;i < G.vexnum ;i++){
329         if(G.vertices[i].data.key == v){//找到结点v
330             ArcNode *p = G.vertices[i].firstarc;
331             while(p){
332                 if(G.vertices[p->adjvex].data.key == w){//找到结
                        点w
333                     if(p->nextarc){//如果w不是最后一个邻接顶点
334                         return p->nextarc->adjvex;
335                     }else{
336                         return ERROR;
```

```
337                     }
338                 }
339                 p = p->nextarc;
340             }
341         }
342     }
343     return ERROR;
344 }
345
346
347 /************************************************
348  * 函数名称: InsertVex
349  * 初始条件: 图G存在, v和G中的顶点具有相同特征
350  * 函数功能: 在图G中增加新顶点v
351  * 返回值: 成功返回OK,若插入后有重复或插入后节点个数超了, 返回
        ERROR
352  ************************************************/
353
354 status InsertVex(ALGraph &G,VertexType v)
355 {
356     if(LocateVex(G,v.key)>=0) return ERROR;
357     if(G.vexnum==MAX_VERTEX_NUM) return ERROR;
358     G.vertices[G.vexnum].data=v;
359     G.vertices[G.vexnum].firstarc=NULL;
360     G.vexnum++;
361     return OK;
362 }
363
364 /*****************************************
365  * 函数名称: DeleteVex
366  * 初始条件: 图G存在, v是G的一个顶点
367  * 函数功能: 在图G中删除顶点v和与v相关的弧
368  *****************************************/
369 status DeleteVex(ALGraph &G,KeyType v) {
370         int i = 0, j;
```

```
371         ArcNode *p, *q;
372         if(G.vertices[0].data.key != -1 && G.vertices[1].data.key
                == -1){
373         return ERROR;
374     } //如果图中只有一个顶点，删除失败
375         j = LocateVex(G, v);
376         if(j == -1)
377                 return ERROR;
378         p = G.vertices[j].firstarc;
379         while(p) {//删除与顶点v相关的弧
380                 q = p;
381                 p = p->nextarc;
382                 free(q);
383                 G.arcnum--;
384                 i++;
385         }
386         G.vexnum--;//顶点数--
387         for(i = j; i < G.vexnum; i++)
388                 G.vertices[i] = G.vertices[i+1];
389         for(i = 0; i < G.vexnum; i++) {
390                 p = G.vertices[i].firstarc;
391                 while(p) {//删除与顶点v相关的弧
392                         if(p->adjvex == j) {
393                                 if(p == G.vertices[i].firstarc) {
394                                         G.vertices[i].firstarc =
                                            p->nextarc;
395                                         free(p);
396                                         p = G.vertices[i].
                                            firstarc;
397                                 } else {
398                                         q -> nextarc = p ->
                                            nextarc;
399                                         free(p);
400                                         p = q -> nextarc;
401                                 }
```

```
402                         } else {
403                                 if(p->adjvex > j)//如果顶点编号大
                                        于j，编号减一
404                                         p->adjvex--;
405                                 q = p;
406                                 p = p->nextarc;
407                         }
408                 }
409         }
410         return OK;
411 }
412
413
414
415 /*************************************************************
416  * 函数名称：InsertArc
417  * 初始条件:图G存在，v、w是G的顶点
418  * 函数功能：在图G中增加弧<v,w>，如果图G是无向图，还需要增加<w,v>
419  *************************************************************/
420 status InsertArc(ALGraph &G,KeyType v,KeyType w) {
421         ArcNode *p;
422         int  i, j;
423         i = LocateVex(G, v);
424         j = LocateVex(G, w);
425         if(i == -1 || j == -1)
426                 return ERROR;
427         p = G.vertices[i].firstarc;
428         while(p) {
429                 if( p -> adjvex == j)return ERROR;//如果已经存在
                        该边，返回ERROR
430                 p = p -> nextarc;
431         }
432         G.arcnum++;
433     //添加弧
434         p=(ArcNode*)malloc(sizeof(ArcNode));
```

```
435         p->adjvex = j;
436         p->nextarc = G.vertices[i].firstarc;
437         G.vertices[i].firstarc = p;
438 
439         p = (ArcNode*)malloc(sizeof(ArcNode));
440         p->adjvex = i;
441         p->nextarc = G.vertices[j].firstarc;
442         G.vertices[j].firstarc = p;
443         return OK;
444 }
445 
446 /*******************************************************
447  * 函数名称: DeleteArc
448  * 初始条件:图G存在, v、w是G的顶点
449  * 函数功能: 在图G中删除弧<v,w>, 如果图G是无向图, 还需要删除<w,v>
450  *******************************************************/
451 status DeleteArc(ALGraph &G, KeyType v, KeyType w) {
452         ArcNode *p, *q;
453         int i, j;
454         // 查找v和w在G中的位置
455         i = LocateVex(G, v);
456         j = LocateVex(G, w);
457         // 如果未找到v或w, 则删除操作失败
458         if(i < 0 || j < 0)return ERROR;
459         // 在顶点i的邻接表中查找指向顶点j的弧, 并记录其前驱q和当
                前指针p
460         p = G.vertices[i].firstarc;
461         while(p && p->adjvex != j) {
462                 q = p;
463                 p = p->nextarc;
464         }
465         // 如果找到弧, 则删除该弧
466         if(p && p->adjvex == j) {
467                 if(p == G.vertices[i].firstarc){
468                         G.vertices[i].firstarc = p->nextarc;
```

```
469                 }else{
470                         q->nextarc = p->nextarc;
471                         free(p);
472                         G.arcnum--;
473                 }
474         }else if(!p) return ERROR;      // 如果未找到弧，则删除操
              作失败
475
476         // 在顶点j的邻接表中查找指向顶点i的弧，并记录其前驱q和当
              前指针p
477         p = G.vertices[j].firstarc;
478         while(p && p->adjvex != i) {
479                 q = p;
480                 p = p->nextarc;
481         }
482         // 如果找到弧，则删除该弧
483         if(p && p->adjvex == i) {
484                 if(p == G.vertices[j].firstarc){
485                         G.vertices[j].firstarc=p->nextarc;
486                 }else{
487                         q->nextarc=p->nextarc;
488                         free(p);
489                 }
490         }
491         return OK;
492 }
493
494
495 /************************************************
496  * 函数名称：dfshelper
497  * 初始条件：图G存在
498  * 函数功能：深度优先遍历，服务于DFSTraverse
499  ************************************************/
500 void dfshelper(const ALGraph &G,int i ,int *visited,void (*visit)
      (VertexType)){
```

```
501     visited[i] = 1;
502     visit(G.vertices[i].data);
503     ArcNode *p = G.vertices[i].firstarc;
504     while(p){
505         if(!visited[p->adjvex]){
506             dfshelper(G,p->adjvex,visited,visit);
507         }
508         p = p->nextarc;
509     }
510     return;
511 }
512
513
514 /*************************************************************
515  * 函数名称: DFSTraverse
516  * 初始条件: 图G存在
517  * 函数功能: 进行深度优先搜索遍历,
518              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
519  *************************************************************/
520 status DFSTraverse(const ALGraph &G,void (*visit)(VertexType))
521 {
522     int k;
523     if(G.vexnum == 0) return ERROR;
524     int *visited;
525         visited = (int*)malloc(sizeof(int)*G.vexnum);
526     for(k = 0 ;k < G.vexnum ;k++){
527         visited[k] = 0;
528     }
529     for(int i=0;i<G.vexnum;i++){
530         if(!visited[i]){
531             dfshelper(G,i,visited,visit);
532         }
533     }
534     return OK;
```

```
535 }
536
537
538
539 /**************************************************************
540  * 函数名称: BFSTraverse
541  * 初始条件: 图G存在
542  * 函数功能: 进行广度优先搜索遍历,
543              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
544  **************************************************************/
545 status BFSTraverse(const ALGraph &G, void (*visit)(VertexType)) {
546     if (G.vexnum == 0) return ERROR;
547
548     int *visited;
549     visited = (int*)malloc(sizeof(int) * G.vexnum);
550     for (int i = 0; i < G.vexnum; i++) {
551         visited[i] = 0;
552     }
553
554     for (int i = 0; i < G.vexnum; i++) {
555         if (!visited[i]) {
556             queueptr q = (queueptr)malloc(sizeof(queue));
557             ArcNode *p;
558
559             initqueue(q);
560             visited[i] = 1;
561             visit(G.vertices[i].data);
562             enqueue(q, i);
563
564             while (!isempty(q)) {
565                 dequeue(q, i);
566                 p = G.vertices[i].firstarc;
567
568                 while (p) {
```

```
569                     if (!visited[p->adjvex]) {
570                         visited[p->adjvex] = 1;
571                         visit(G.vertices[p->adjvex].data);
572                         enqueue(q, p->adjvex);
573                     }
574                     p = p->nextarc;
575                 }
576             }
577 
578             free(q);
579         }
580     }
581 
582     free(visited);
583     return OK;
584 }
585 
586 
587 /*****************************
588  * 函数名称: visit
589  * 初始条件: 图G存在
590  * 函数功能: 访问节点一次
591  *****************************/
592 void visit(VertexType v) {
593         printf("                key = %d,others = %s\n",v.key,v.
                others);
594 }
595 
596 
597 
598 /*****************************
599  * 函数名称: SaveGraph
600  * 初始条件: 图G存在
601  * 函数功能: 保存图为文件
602  *****************************/
```

```
603 status SaveGraph(ALGraph G, char FileName[])
604 //将图的数据写入到文件FileName中
605 {
606     FILE *fp;
607     if(!G.vexnum)
608         return INFEASIBLE;
609     if (!(fp = fopen(FileName, "wb")))
610         return ERROR;
611     for (int i = 0; i < G.vexnum; i++)
612     { //写入每一个顶点
613         fprintf(fp, "%d %s ", G.vertices[i].data.key, G.vertices[
              i].data.others);
614         for (ArcNode *p = G.vertices[i].firstarc; p; p = p->
              nextarc)
615         { //顺序写入该顶点的每一个邻接点的位置
616             fprintf(fp, "%d ", p->adjvex);
617         }
618         fprintf(fp, "%d ", -1); //邻接点结尾处写上-1
619     }
620     fprintf(fp, "%d %s ", -1, "nil"); //顶点结尾处写上-1
621     fclose(fp);
622     return OK;
623 }
624
625 /*****************************
626  * 函数名称: LoadGraph
627  * 初始条件:文件存在
628  * 函数功能: 从文件中加载图
629  *****************************/
630 status LoadGraph(ALGraph &G, char FileName[])
631 //读入文件FileName的图数据, 创建图的邻接表
632 //本函数调用栈的数据结构及其操作函数
633 {
634     if (G.vexnum > 0)
635         return  INFEASIBLE;
```

```
636     FILE *fp;
637     if (!(fp = fopen(FileName, "rb")))
638         return ERROR;
639     G.vexnum = G.arcnum = 0;
640     KeyType key;            //存储从文件中读取顶点key值的临时变量
641     int ConnectVerLocate; //存储读取的顶点邻接点的位置的临时变量
642     char others[20];        //存储读取的顶点的data中others分量的临
        时变量
643     stack S;                //定义栈
644     iniStack(S);
645     fscanf(fp, "%d ", &key);
646     fscanf(fp, "%s ", others);
647     for (int i = 0; key != -1 && G.vexnum < MAX_VERTEX_NUM; i++)
648     { //创建顶点
649         G.vertices[i].data.key = key;
650         strcpy(G.vertices[i].data.others, others);
651         G.vertices[i].firstarc = NULL;
652         fscanf(fp, "%d ", &ConnectVerLocate);
653         for (; ConnectVerLocate != -1;)
654         { //建立顶点的邻接表
655             //通过入栈出栈把顺序倒置，确保读取后顶点邻接表顺序与
                原图相同
656             push(S, ConnectVerLocate);
657             fscanf(fp, "%d ", &ConnectVerLocate);
658         }
659         for (ArcNode *p; !isEmptyStack(S);)
660         {                                   //栈不为空时循环
661             ConnectVerLocate = pop(S); //出栈
662             //从邻接表头部插入邻接点结点
663             p = (ArcNode *)malloc(sizeof(ArcNode));
664             p->adjvex = ConnectVerLocate;
665             p->nextarc = G.vertices[i].firstarc;
666             G.vertices[i].firstarc = p;
667             G.arcnum++; //图的边数自增
668         }
```

```
669         G.vexnum++;                              //图的顶点数自增
670         fscanf(fp, "%d %s", &key, others); //读取下一顶点
671     }
672     G.arcnum/=2;
673     G.kind=UDG;
674     fclose(fp);
675     return OK;
676 }
677
678 /*附加功能*/
679 /*****************************
680  * 函数名称: VerticesSetLessThanK
681  * 初始条件:图G存在
682  * 函数功能: 返回与顶点v距离小于k的顶点集合result,并返回顶点个数
        count
683  * 返回值: 如果存在, 返回OK, 如果不存在, 返回ERROR
684  * 算法思路: BFS
685  *****************************/
686 status VerticesSetLessThanK(ALGraph G,int v,int k,int *result,int
        *count){
687     int dist[MAX_VERTEX_NUM];//dist[i]表示顶点v到顶点i的距离
688     int visited[MAX_VERTEX_NUM];//visited[i]表示顶点i是否被访问过
689     int i;
690     for(i=0; i<G.vexnum; i++){
691         dist[i] = -1;
692         visited[i] = 0;
693     }
694     //v是结点关键字, 要转换成数组索引
695     v = LocateVex(G,v);
696     if(v==-1)return ERROR;
697
698     dist[v] = 0;
699     visited[v] = 1;
700     int queue[MAX_VERTEX_NUM];
701     int front = 0,rear = 0;
```

```
    queue[rear++] = v;
    //BFS
    while(front < rear){
        int u = queue[front++];
        ArcNode *p = G.vertices[u].firstarc;
        while(p != NULL){
            int adjvex = p->adjvex;
            if(!visited[adjvex]){
                dist[adjvex] = dist[u] + 1;
                visited[adjvex] = 1;
                queue[rear++] = adjvex;
            }
            p = p->nextarc;
        }
    }
    *count = 0;
    for(i=0; i<G.vexnum; i++){
        if(dist[i]>=0 && dist[i]<k){
            result[(*count)++] = i;
        }
    }
    if(*count == 0){
        return ERROR;
    }
    return OK;
}

/*****************************
 * 函数名称：ShortestPathLength(G,v,w);
 * 初始条件:图G存在
 * 函数功能：返回顶点v与顶点w的最短路径的长度;
 * 算法思路：利用队列使用BFS，记录距离
 *****************************/

int ShortestPathLength(ALGraph G, KeyType v, KeyType w) {
```

```
737     int visited[MAX_VERTEX_NUM] = {0}; // 用于记录每个顶点是否被
            访问过
738     int distance[MAX_VERTEX_NUM] = {0}; // 用于记录起点到每个顶点
            的距离
739     int queue[MAX_VERTEX_NUM] = {0}; // 队列用于BFS
740     int front = -1, rear = -1;
741     int i, j;
742     ArcNode *p;
743
744     // 首先找到v所在的顶点下标i
745     for (i = 0; i < G.vexnum; i++) {
746         if (G.vertices[i].data.key == v) {
747             break;
748         }
749     }
750     if (i == G.vexnum) { // v不在图G中
751         return -1;
752     }
753
754     visited[i] = 1; // 标记i已经被访问
755     distance[i] = 0; // 起点到起点的距离为0
756     queue[++rear] = i; // 将i入队
757
758     while (front != rear) { // BFS主循环
759         i = queue[++front]; // 取出队首元素
760
761         // 遍历i的所有邻接点
762         p = G.vertices[i].firstarc;
763         while (p != NULL) {
764             j = p->adjvex; // 邻接点j的下标
765             if (!visited[j]) { // j没有被访问过
766                 visited[j] = 1; // 标记j已经被访问
767                 distance[j] = distance[i] + 1; // j到起点的距离比
                        i到起点的距离多1
768                 if (G.vertices[j].data.key == w) { // 找到了终点
```

```
769                     return distance[j];
770                 }
771                 queue[++rear] = j; // 将j入队
772             }
773             p = p->nextarc;
774         }
775     }
776 
777     return -1; // 没有找到从v到w的路径
778 }
779 
780 
781 void DFS(ALGraph G, int v, int visited[]) {
782     visited[v] = true;
783     ArcNode* p = G.vertices[v].firstarc;
784     while (p != NULL) {
785         int w = p->adjvex;
786         if (!visited[w]) {
787             DFS(G, w, visited);
788         }
789         p = p->nextarc;
790     }
791 }
792 /*****************************
793  * 函数名称: ConnectedComponentsNums
794  * 初始条件:图G存在
795  * 函数功能: 返回图G的所有连通分量的个数
796  * 算法思路: 使用DFS, 每次遍历就是找到了一个连通分量
797  *****************************/
798 int ConnectedComponentsNums(ALGraph G) {
799     int count = 0;
800     int visited[MAX_VERTEX_NUM];
801     for (int i = 0; i < G.vexnum; i++) {
802         visited[i] = 0;
803     }
```

```
804
805     for (int i = 0; i < G.vexnum; i++) {
806         if (!visited[i]) {
807             DFS(G, i, visited);
808             count++;
809         }
810     }
811     return count;
812 }
```

main 函数 u4.cpp 如下：

```
/*main function of graph*/
#include "func.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
        int op=1, i, i_num=1, key;
    VertexType e;//无向图中的节点
    ALGraph G[graphnum+1];//十个图,0号图不用
    VertexType V[MAX_VERTEX_NUM];
    KeyType VR[MAX_ARC_NUM][2];//用于创建图
    KeyType u;//用于查找顶点
        int result ,v1 ,v2 ,result1 ,result2 ,dis;
        char s[20];

        int arr[MAX_VERTEX_NUM];//用于附加功能中保存距离小于k的顶
            点
        int count;//用于附加功能中保存距离小于k的顶点的个数

    for (i = 0; i<=10; i++)
        {
        G[i].vexnum=0;
                G[i].arcnum=0;
                 G[i].kind = DG;//用来标记图的种类, DG说明未初始
                     化, UDG说明初始化成了无向图
        }
    while(op){
        system("cls");  //用于清屏
        printf("\n\n");
        printf("\t\t      Menu for Undirected Graph On Chain
            Structure \n");
        printf("\t
            **********************************************\n");
```

```
31          printf("\t\t\t1.  CreateGraph                2.
               DestroyGraph\n");
32          printf("\t\t\t3.  LocateVex                  4.  PutVex\n"
               );
33          printf("\t\t\t5.  FirstAdjVex               6.
               NextAdjVex\n");
34          printf("\t\t\t7.  InsertVex                  8.  DeleteVex
               \n");
35          printf("\t\t\t9.  InsertArc                  10. DeleteArc
               \n");
36          printf("\t\t\t11. DFSTraverse                12.
               BFSTraverse\n");
37          printf("\t\t\t13. SaveGraph                  14. LoadGraph
               \n");
38          printf("\t\t\t15. VerticesSetLessThanK       16.
               ShortestPathLength\n");
39          printf("\t\t\t17. ConnectedComponentsNums    18. choose\n"
               );
40      printf("\t\t\t0.Exit    \n");
41          printf("\t
               **************************************************\n")
               ;
42          printf("\t\t\t请选择你的操作[0-18]: ");
43          scanf("%d",&op);
44      switch(op)
45          {
46              case 1:
47         //创建图
48              if(G[i_num].kind==UDG){
49                          printf("\t\t\t该图已经创建过了!\n");
50                          getchar();getchar();
51                          break;
52              }
53               printf("\t\t\t请输入数据: \n") ;
54                   i=0;
```

```
55          do {
56              scanf("%d%s",&V[i].key,V[i].others);
57          } while(V[i++].key!=-1);
58          i=0;
59          do {
60              scanf("%d%d",&VR[i][0],&VR[i][1]);
61          } while(VR[i++][0]!=-1);
62
63          if (CreateGraph(G[i_num],V,VR)==OK) {
64                          printf("\t\t\t%d号图表创建成功! \n",i_num
                                );
65
66          }else {
67              printf("\t\t\t输入数据错, 无法创建\n");
68          }
69          getchar();getchar();
70          break;
71
72             case 2:
73         //销毁图
74             if(G[i_num].kind==DG){
75                  printf("\t\t\t该图还未创建!\n");
76                  getchar();getchar();
77          break;
78             }
79              if(DestroyGraph(G[i_num])==OK)
80                  printf("\t\t\t销毁无向图成功!\n");
81          else{
82                          printf("\t\t\t销毁无向图失败!\n");
83                  }
84
85                  getchar();getchar();
86                  break;
87
88             case 3:
```

```
89          //查找顶点
90              if(G[i_num].kind==DG){
91                    printf("\t\t\t该图还未创建!\n");
92                    getchar();getchar();
93           break;
94              }
95               printf("\t\t\t请输入需要查找结点的key值: ");
96           scanf("%d",&u);
97           i = LocateVex(G[i_num],u);
98           if(i != -1) {
99               printf("\t\t\t查找成功! 该结点信息为: \n");
100              visit(G[i_num].vertices[i].data);
101          } else printf("\t\t\t查找失败! \n");
102          getchar();
103          getchar();
104          break;
105
106             case 4:
107         //顶点赋值
108             if(G[i_num].kind==DG){
109                   printf("\t\t\t该图还未创建!\n");
110                   getchar();getchar();
111          break;
112             }
113                   printf("                    请输入需要赋值的结点key值:
                        ");
114          scanf("%d",&u);
115          printf("                  请输入赋值后的结点key值:");
116          scanf("%d",&e.key);
117          printf("                  请输入赋值后的结点others值:");
118          scanf("%s",e.others);
119          if(PutVex(G[i_num],u,e)==OK) {
120              printf("\t\t\t赋值成功! 现在该结点的信息为: \n");
121              visit(G[i_num].vertices[LocateVex(G[i_num],e.key)].
                    data);
```

```
122          }else printf("\t\t\t赋值失败! 赋值后有重复\n");
123          getchar(); getchar();
124          break;
125
126             case 5:
127         //获得第一邻接点
128             if(G[i_num].kind==DG){
129                   printf("\t\t\t该图还未创建!\n");
130                   getchar();getchar();
131          break;
132             }
133              printf("\t\t\t请输入要查找的顶点的key: ");
134              scanf("%d",&key);
135                   result=LocateVex(G[i_num],key);
136                   if(result==-1){
137                           printf("\t\t\t该顶点不存在!\n");
138                           getchar();getchar();
139                           break;
140                   }
141                   result=FirstAdjVex(G[i_num],key);
142                   if(result==ERROR){
143                           printf("\t\t\t该顶点没有邻接顶点!\n");
144                   }else{
145                           printf("\t\t\t该顶点的第一个邻接顶点为:\n
                                 ");
146                           visit(G[i_num].vertices[result].data);
147                   }
148              getchar();getchar();
149                   break;
150
151             case 6:
152         //获得下一邻接点
153             if(G[i_num].kind==DG){
154                   printf("\t\t\t该图还未创建!\n");
155                   getchar();getchar();
```

```
156         break;
157            }
158         printf("\t\t\t请输入要查找的顶点 v 的key: ");
159             scanf("%d",&key);
160                 printf("\t\t\t请输入和其相对的顶点 w 的key: ");
161                 scanf("%d",&i);
162                 result=NextAdjVex(G[i_num],key,i);
163             if(result==ERROR){
164                         printf("\t\t\t查找失败!\n");
165                 }else{
166                         printf("\t\t\t该顶点v相对于w的下一个邻接
                                顶点为:\n");
167                         visit(G[i_num].vertices[result].data);
168                 }
169                 getchar();getchar();
170                 break;
171
172            case 7:
173         //插入顶点
174            if(G[i_num].kind==DG){
175                 printf("\t\t\t该图还未创建!\n");
176                 getchar();getchar();
177          break;
178            }
179                 if(G[i_num].vexnum >= MAX_VERTEX_NUM){
180                         printf("\t\t\t顶点数已达到最大值!不支持插
                                入\n");
181                         getchar();getchar();
182                         break;
183                 }
184                 printf("\t\t\t请输入要插入的顶点的key值:\n");
185                 scanf("%d",&e.key);
186                 printf("\t\t\t请输入要插入的顶点的others值:\n");
187                 scanf("%s",e.others);
188                 if(InsertVex(G[i_num],e)==ERROR){
```

```
189                     printf("\t\t\t关键字重复插入失败!\n");
190             }else{
191                     printf("\t\t\t插入成功!\n");
192             }
193             getchar();getchar();
194             break;
195
196         case 8:
197       //删除顶点
198         if(G[i_num].kind==DG){
199             printf("\t\t\t该图还未创建!\n");
200             getchar();getchar();
201      break;
202         }
203             printf("\t\t\t请输入要删除的顶点的key值:");
204             scanf("%d",&key);
205             result=LocateVex(G[i_num],key);
206             if(result==-1){
207                     printf("\t\t\t该顶点不存在!\n");
208                     getchar();getchar();
209                     break;
210             }
211             if(DeleteVex(G[i_num],key)==ERROR){
212                     printf("\t\t\t删了就是空图了, 不许删>?<")
                            ;
213             }else{
214                     printf("\t\t\t删除成功!\n");
215             }
216             getchar();getchar();
217                     break;
218
219         case 9:
220       //插入弧
221         if(G[i_num].kind==DG){
222             printf("\t\t\t该图还未创建!\n");
```

```
223             getchar();getchar();
224         break;
225         }
226             printf("\t\t\t请输入边的两个节点的关键字(空格作为
                间隔,如a1 a2):");
227             scanf("%d %d" ,&v1,&v2);
228             result1=LocateVex(G[i_num],v1);
229             result2=LocateVex(G[i_num],v2);
230             if(result1==-1||result2==-1){
231                     printf("\t\t\t顶点不存在!\n");
232                     getchar();getchar();
233                     break;
234             }
235             if(InsertArc(G[i_num],v1,v2)==ERROR){
236                     printf("\t\t\t添加失败 ,它们之间已经有弧
                        !\n");
237             }else{
238                     printf("\t\t\t添加成功!\n");
239             }
240             getchar();getchar();
241             break;
242
243         case 10:
244     //删除弧
245         if(G[i_num].kind==DG){
246             printf("\t\t\t该图还未创建!\n");
247             getchar();getchar();
248         break;
249         }
250         printf("\t\t\t请输入边的两个节点的关键字(空格作为间隔,
                如a1 a2):");
251             scanf("%d %d" ,&v1,&v2);
252             result1=LocateVex(G[i_num],v1);
253             result2=LocateVex(G[i_num],v2);
254             if(result1==-1||result2==-1){
```

```
255                     printf("\t\t\t顶点不存在!\n");
256                     getchar();getchar();
257                     break;
258             }
259             if(DeleteArc(G[i_num],v1,v2)==ERROR){
260                     printf("\t\t\t删除失败 ,它们之间没有弧!\n
                           ");
261             }else{
262                     printf("\t\t\t删除成功!\n");
263             }
264             getchar();getchar();
265             break;
266
267         case 11:
268     //dfs
269         if(G[i_num].kind==DG){
270             printf("\t\t\t该图还未创建!\n");
271             getchar();getchar();
272      break;
273         }
274             printf("\t\t\t该无向图的深度优先搜索遍历为: \n");
275             DFSTraverse(G[i_num],visit);
276             printf("\n");
277             getchar();getchar();
278             break;
279
280
281         case 12:
282     //bfs
283         if(G[i_num].kind==DG){
284             printf("\t\t\t该图还未创建!\n");
285             getchar();getchar();
286      break;
287         }
288             printf("\t\t\t该无向图的广度优先搜索遍历为: \n");
```

```
289                 BFSTraverse(G[i_num],visit);
290                 printf("\n");
291                 getchar();getchar();
292                 break;
293
294             case 13:
295         //保存图
296             if(G[i_num].kind==DG){
297                 printf("\t\t\t该图还未创建!\n");
298                 getchar();getchar();
299          break;
300             }
301                 printf("\t\t\t请输入保存文件的文件名(如a.txt):\n"
                        );
302                 scanf("%s",s);
303
304              if(SaveGraph(G[i_num] ,s)==OK){
305                         printf("\t\t\t保存成功!\n");
306                 }else{
307                         printf("\t\t\t保存失败!\n");
308                 }
309                 getchar();getchar();
310                 break;
311
312
313             case 14:
314             //读取图
315             if(G[i_num].kind==UDG){
316                 printf("\t\t\t该图已存在!\n");
317                 getchar();getchar();
318          break;
319             }
320                 printf("\t\t\t请输入读取文件的文件名(如a.txt):\n"
                        );
321                 scanf("%s",s);
```

```
322         if(LoadGraph(G[i_num] ,s)==OK){
323                         printf("\t\t\t读取成功!\n");
324                 }else{
325                         printf("\t\t\t读取失败!\n");
326                 }
327                 getchar();getchar();
328                 break;
329
330             case 15:
331         //距离小于k的顶点集合
332             if(G[i_num].kind==DG){
333                 printf("\t\t\t该图还未创建!\n");
334                 getchar();getchar();
335          break;
336             }
337             printf("\t\t\t请输入要查找的顶点的key值:\n");
338             scanf("%d",&key);
339                 printf("\t\t\t你想找的距离小于多少的顶点集合?\n"
                        );
340                 scanf("%d",&dis);
341                 count=0;
342                 if(VerticesSetLessThanK(G[i_num],key,dis,arr,&
                        count)==ERROR){
343                         printf("\t\t\t一个也没有(?_?|||)\n");
344                 }else{
345                         printf("\t\t\t距离小于%d的顶点集合为:\n",
                                dis);
346                         for(i=0;i<count;i++){
347                                 visit(G[i_num].vertices[arr[i]].
                                        data);
348                         }
349                 }
350                 getchar();getchar();
351                 break;
352
```

```
353             case 16:
354         //顶点间最短路径和长度
355             if(G[i_num].kind==DG){
356                   printf("\t\t\t该图还未创建!\n");
357                   getchar();getchar();
358          break;
359             }
360             printf("\t\t\t请输入要查找的两个顶点的key值,空格分隔:\
                   n");
361             scanf("%d %d",&v1,&v2);
362                   result1=LocateVex(G[i_num],v1);
363                   result2=LocateVex(G[i_num],v2);
364                   if(result1==-1||result2==-1){
365                           printf("\t\t\t顶点不存在!\n");
366                           getchar();getchar();
367                           break;
368                   }
369                   result=ShortestPathLength(G[i_num],v1,v2);
370                   if(result==ERROR){
371                           printf("\t\t\t两个顶点之间没有路径!\n");
372                   }else{
373                           printf("\t\t\t两个顶点之间的最短路径长度
                                  为:%d\n",result);
374                   }
375                   getchar();getchar();
376                           break;
377
378          case 17:
379          //图的连通分量个数
380                   if(G[i_num].kind==DG){
381                   printf("\t\t\t该图还未创建!\n");
382                   getchar();getchar();
383          break;
384             }
385                   printf("\t\t\t连通分量个数为:%d\n",
```

```
                    ConnectedComponentsNums(G[i_num]));
386                 getchar();getchar();
387                         break;
388
389         case 18:
390         //choose
391         printf("\t\t请输入要在第几个图操作,只支持在%d个图进行操作
                : " ,graphnum);
392                 scanf("%d",&i_num);
393                 if(i_num<1||i_num>20)
394                 {
395                         printf("\t\t\t不支持在该图上进行操作,已默
                            认在第一个图!\n");
396                         i_num=1;
397                 }else{
398                         printf("\t\t\t已切换到第%d个图!\n",i_num)
                            ;
399                 }
400                 getchar(); getchar();
401                 break;
402
403             case 0:
404         break;
405         }//end of switch
406   }//end of while
407     printf("\n");
408         printf("\t\t\t欢迎下次使用本系统!\n\n");
409 }//end of main()
```