

华中科技大学

课程实验报告

课程名称: 数据结构实验

专业班级 CS2307

学 号 U202315618

姓 名 陈华宇

指导教师 周全

报告日期 2024 年 6 月 10 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
1.1 问题描述	1
1.2 系统设计	1
1.3 系统实现	4
1.4 系统测试	13
1.5 实验小结	15
2 基于邻接表的图实现	16
2.1 问题描述	16
2.2 系统设计	16
2.3 系统实现	18
2.4 系统测试	30
2.5 实验小结	32
3 课程的收获和建议	33
3.1 基于顺序存储结构的线性表实现	33
3.2 基于链式存储结构的线性表实现	34
3.3 基于二叉链表的二叉树的实现	34
3.4 基于邻接表的图实现	35
4 参考文献	37
A 附录 A 基于顺序存储结构线性表实现的源程序	38
B 附录 B 基于链式存储结构线性表实现的源程序	65
C 附录 C 基于二叉链表二叉树实现的源程序	87
D 附录 D 基于邻接表图实现的源程序	110

1 基于顺序存储结构的线性表实现

1.1 问题描述

线性表是最常用而且最简单的一种数据结构。简言之，一个线性表是 n 个数据元素的有限序列。线性表的存储结构分为顺序存储和链式存储。其中本实验采取顺序存储的方式实现线性表的功能。

目的：构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。附录 A 提供了简易菜单的框架。程序实现线性表的初始化、销毁线性表、清空线性表、线性表判空、求线性表表长、获得元素等基本功能，以及最大连续子数组和、和为 K 的子数组、顺序表排序等附加功能。可以选择以文件的形式进行存储和加载，将生成的线性表存入到相应的文件中，也可以从文件中获取线性表进行操作。同时实现多线性表管理，完成多线性表的添加、删除、选择等操作。

1.2 系统设计

1.2.1 数据结构设计

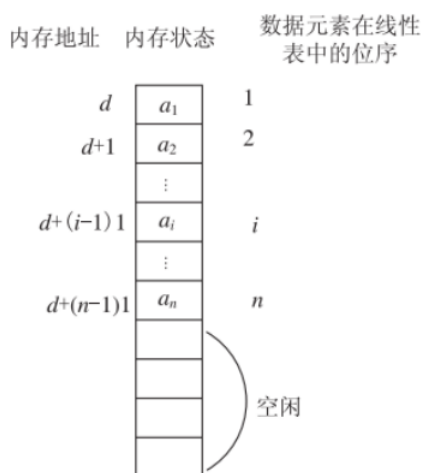


图 1-1 线性表的储存结构

线性表的逻辑结构定义如下：ADT List{

数据对象： $D = \{a_i | a_i \in \text{ElemSet}, i = 1, 2, \dots, n, n \geq 0\}$

数据关系： $R1 = \{ \langle a_{i-1}, a_i \rangle | a_{i-1}, a_i \in D, i = 2, \dots, n \}$ }

1.2.2 基本的功能函数

根据线性表的逻辑结构设计基本的功能函数，用来完成要求实现的功能, 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, 同时需要满足 $1 \leq i \leq \text{ListLength}(L)$; 操作结果是用 `e` 返回 `L` 中第 `i` 个数据元素的值;
7. 查找元素: 函数名称是 `LocateElem(L,e,compare())`; 初始条件是线性表已存在; 操作结果是返回 `L` 中第 1 个与 `e` 满足关系 `compare ()` 关系的数据元素的位序, 若这样的数据元素不存在, 则返回值为 0;
8. 获得前驱: 函数名称是 `PriorElem(L,cur_e,pre_e)`; 初始条件是线性表 `L` 已存在; 操作结果是若 `cur_e` 是 `L` 的数据元素, 且不是第一个, 则用 `pre_e` 返回它的前驱, 否则操作失败, `pre_e` 无定义;
9. 获得后继: 函数名称是 `NextElem(L,cur_e,next_e)`; 初始条件是线性表 `L` 已存在; 操作结果是若 `cur_e` 是 `L` 的数据元素, 且不是最后一个, 则用 `next_e` 返回它的后继, 否则操作失败, `next_e` 无定义;
10. 插入元素: 函数名称是 `ListInsert(L,i,e)`; 初始条件是线性表 `L` 已存在, 同时需要满足 $1 \leq i \leq \text{ListLength}(L)+1$; 操作结果是在 `L` 的第 `i` 个位置之前插入新的数据元素 `e`。
11. 删除元素: 函数名称是 `ListDelete(L,i,e)`; 初始条件是线性表 `L` 已存在且非空, $1 \leq i \leq \text{ListLength}(L)$; 操作结果: 删除 `L` 的第 `i` 个数据元素, 用 `e` 返回其值;

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

1.2.3 附加功能函数

1. 最大连续子数组和：函数名称是 MaxSubArray(L); 初始条件是线性表 L 已存在且非空，请找出一个具有最大和的连续子数组（子数组最少包含一个元素），操作结果是其最大和；
2. 和为 K 的子数组：函数名称是 SubArrayNum(L,k); 初始条件是线性表 L 已存在且非空, 操作结果是该数组中和为 k 的连续子数组的个数；
3. 顺序表排序：函数名称是 sortList(L); 初始条件是线性表 L 已存在；操作结果是将 L 由小到大排序；
4. 实现线性表的文件形式保存：其中，1 需要设计文件数据记录格式，以高效保存线性表数据逻辑结构 (D,R) 的完整信息；2 需要设计线性表文件保存和加载操作合理模式。
 - (a) 文件写入：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 L 的元素写到名称为 FileName 的文件中。
 - (b) 文件读出：函数名称是 LoadList(L,FileName); 初始条件是线性表 L 不存在；操作结果是将文件 FileName 中的元素读到表 L 中。
5. 实现多个线性表管理：设计相应的数据结构管理多个线性表的查找、添加、移除等功能。
 - (a) 增加线性表：函数名称是 AddList(Lists, ListName); 初始条件是名称为 ListName 的线性表不存在于线性表集合中；操作结果是在 Lists 中创建一个名称为 ListName 的初始化好的线性表。
 - (b) 移除线性表：函数名称是 RemoveList(Lists, ListName); 初始条件是名称为 ListName 的线性表存在于线性表集合中；操作结果是将该线性表移除。
 - (c) 查找线性表：函数名称是 LocateList(Lists, ListName); 初始条件是名称为 ListName 的线性表存在于线性表集合中；操作结果是返回该线性表在 Lists 中的逻辑索引。
 - (d) 选择表：函数名称是 SwitchList (Lists, i); 初始条件是 Lists 已存在, $1 \leq i \leq \text{Lists.Length}+1$ ；操作结果是将 Lists 中逻辑索引为 i 的线性表选择为当前处理的线性表，

以便后续再调用其他函数对该表进行操作。

1.2.4 菜单的设计

通过菜单的演示循环选择数字，通过数字的不同，进入相应的条件判断，实现相应的功能，其中选择 0 为退出该系统。

1.3 系统实现

1.3.1 菜单的演示系统

系统通过 while 语句多次输入 op 的值，通过 switch 语句具体实现用户选择的功能，并通过 system("cls") 清屏，提高用户的使用体验。(其中 op==0 时，退出该系统)

1.3.2 数据结构设计

1. 线性表: SqList:elem 储存每个元素，listsize 表示最大的大小,length 储存线性表的元素。
2. 多个线性表的管理: Lists:elem 包括一个 SqList 和表示线性表名字的 name, listsize 与 length 同上 SqList 的作用。

1.3.3 函数功能的实现

1. 初始化表

status InitList (SqList &L)

输入：线性表（引用参数）

输出：函数运行状态

该函数为空线性表的初始化,在函数中，首先使用 malloc 函数为线性表分配 LISTSIZE 大小的连续内存空间，将首地址赋值给 L.elem，由于线性表的长度为 0，将 L.length 初始化为 0，即完成了线性表的初始化。

2. 销毁线性表

status DestroyList(SqList &L);

初始条件是线性表 L 已存在；操作结果是销毁线性表 L；该函数设计思想是，若 L 非空，将 L 置空；若 L 已空，返回异常即可。(利用 free 释放 elem 的首空间，再将 length 变为 0。

3. 清空线性表

```
status ClearList(SqList &L);
```

初始条件是线性表 L 已经存在，若 L 已经存在，就将 L 设置为空表；

并将 length 变为 0，此处与函数二的不同点在于 ClearList 并不用释放线性表 elem 首空间，只需清空元素。

4. 判定空表

```
status ListEmpty(SqList L)
```

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

请参考算法1.1

算法 1.1. 判定空表

```
procedure ListEmpty(L)
  if L.elem == NULL then
    return INFEASIBLE
  end if
  if L.length != 0 then
    return FALSE
  end if
  return TRUE
end procedure
```

经分析，该算法的时间复杂度为 $O(1)$ 。

5. 求表长

```
int ListLength(SqList L);
```

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

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

6. 获得元素

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

初始条件是线性表已存在, $1 \leq i \leq \text{ListLength}(L)$; 操作结果是用 e 返回 L 中第 i 个数据元素的值;

请参考算法1.2

算法 1.2. 获得元素

Input: : Linear Sequence: L,i

Output: : status, e

```
procedure GetElem(L, i)
    if L.elem = NULL then
        return Infeasible
    end if
    if  $i < 1$  or  $i > \text{length}$  then
        return Error
    end if
     $e \leftarrow L.\text{elem}[i - 1]$ 
    return True, e
end procedure
```

7. 查找元素

status LocateElem (SqList L, ElemType e, int (*compare)(SqList, int, ElemType));

初始条件是线性表已存在; 操作结果是返回 L 中第 1 个与 e 满足关系 compare() 关系的数据元素的位序, 若这样的数据元素不存在, 则返回值为 0; (由于元素是整形, compare 函数用等于号代替)

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

请参考算法1.3

算法 1.3. 查找元素

Input: : Linear Sequence: L, e

Output: : i

```
procedure LocateElem( $L, e$ )
    if  $L.\text{elem} = \text{NULL}$  then
        return Infeasible
    end if
     $i \leftarrow 1$ 
    while  $i < L.\text{length}$  do
        if Compare( $L.\text{elem}[i - 1], e$ ) then
            return  $i$ 
        end if
         $i \leftarrow i + 1$ 
    end while
    return 0
end procedure
```

经分析，该算法的时间复杂度为 $O(n)$ 。

8. 获得前驱

$\text{status PriorElem (SqList } L, \text{ ElemType } e, \text{ ElemType } \&\text{pre});$

初始条件是线性表 L 已存在；操作结果是若 e 是 L 的数据元素，且不是第一个，则用 pre 返回它的前驱，否则操作失败， pre 无定义；

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

相关算法与上述的算法类似，简单拼凑即可。

9. 获得后继

status NextElem (SqList L, ElemType e, ElemType &next);

初始条件是线性表 L 已存在；操作结果是若 cure 是 L 的数据元素，且不是最后一个，则用 next 返回它的后继，否则操作失败，next 无定义；

算法的设计思想和获得前驱类似。

10. 插入元素

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

初始条件是线性表 L 已存在， $1 \leq i \leq \text{ListLength}(L)+1$ ；操作结果是在 L 的第 i 个位置之前插入新的数据元素 e。

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

请参考算法1.4

算法 1.4. 插入元素

Input: : Linear Sequence: L,i,e

Output: : Status

```
procedure ListInsert(L, i, e)
    if L.elem = NULL then
        return Infeasible
    end if
    if number(i) illegal then
        return Error
    end if
     $j \leftarrow L.length - 1$ 
    while  $j > i$  do
         $L.elem[j + 1] \leftarrow L.elem[j]$ 
         $j \leftarrow j - 1$ 
    end while
```

```
L.elem[i - 1] ← e  
return OK  
end procedure
```

11. 删除元素

```
status ListDelete (SqList L, int i, ElemType &e);
```

初始条件是线性表 L 已存在且非空, $1 \leq i \leq \text{ListLength}(L)$; 操作结果: 删除 L 的第 i 个数据元素, 将被删除的元素赋值给 e , 然后将位序为 i 后的元素向前移动一个内存单元, 返回 OK;

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

12. 遍历表

```
status ListTraverse (SqList L, int (*visit)(int));
```

初始条件是线性表 L 已存在; 操作结果是依次对 L 的每个数据元素调用函数 $\text{visit}()$ 。

该算法的设计思想是, 依次对 L 的每个元素调用 $\text{visit}()$ 即可。

13. 最大连续子数组和

```
status MaxSubArray(SqList L)
```

思想一 (暴力枚举法): 计算前缀和 $\text{sum}[1..n]$, 而后枚举 $i, j (i \leq j)$, 计算 $\text{sum}[j] - \text{sum}[i]$ 并取最大值, 并输出该最大值即可。具体算法附如下: 算法输入: 线性表 L 算法输出: 最大值 算法流程: 1. 计算前缀和数组 sum , $\text{sum}[0] = L.\text{elem}[0]$, 对于 $i \neq 0$ 时, $\text{sum}[i] = \text{sum}[i-1] + L.\text{elem}[i]$; 2. 枚举 i, j , 记录 $\text{sum}[j] - \text{sum}[i]$ 的最大值。算法时间复杂度: 因为我们枚举了 i 和 j , 故其时间复杂度为 $O(n^2)$ 。

显然上述的方法是最容易想到, 也是最简单易懂的, 但是在规模很大的数组中运用此方法时, 显然会导致所消耗的时间过多, 于是经过我的学习和查找资料, 发现了可以使用动态规划的方法, 时间复杂度将大大减小。

思想二 (动态规划): 定义状态: 首先定义问题的状态。在这个问题中, 可以定义状态为 $\text{dp}[i]$, 表示以第 i 个元素结尾的最大连续子数组和。

找到状态转移方程：接下来找到状态之间的转移关系，即状态转移方程。对于每个位置 i ， $dp[i]$ 可以由前一个状态 $dp[i-1]$ 推导得出。

$$dp[i] = \max(dp[i-1] + \text{nums}[i], \text{nums}[i])$$

这个方程的意思是，要么加上当前元素 $\text{nums}[i]$ ，要么从当前元素开始重新计算。最终结果就是这两者的最大值。

确定初始状态：确定初始状态。在这个问题中，初始状态可以定义 $dp[0] = \text{nums}[0]$ ，即第一个元素本身就构成一个子数组。

实现状态转移：根据状态转移方程，利用循环实现状态的转移，计算出所有状态 $dp[i]$ 的值。

找出最优解：最终的结果即为所有状态中的最大值 $\max(dp[i])$

请参考算法1.5

算法 1.5. 最大连续子数组的和

```
function MaxSubArray(nums)  
    if not nums then  
        return 0  
    end if  
     $n \leftarrow \text{length of } \text{nums}$   
     $dp[0] \leftarrow \text{nums}[0]$   
     $\text{max\_sum} \leftarrow \text{nums}[0]$   
    for  $i \leftarrow 1$  to  $n - 1$  do  
         $dp[i] \leftarrow \max(dp[i - 1] + \text{nums}[i], \text{nums}[i])$   
         $\text{max\_sum} \leftarrow \max(\text{max\_sum}, dp[i])$   
    end for  
    return  $\text{max\_sum}$   
end function
```

经过分析，上述算法的时间复杂度为 $O(n)$ 。

14. 和为 K 的子数组

```
status SubArrayNum(Sqlist L, int k);
```

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

算法思想: 依次枚举从 i 到 j 的子元素和, 如果他的和等于 k, 那么 count 自增, 最后返回 count 即为个数。

由相似的分析可得, 其时间复杂度为 $O(n^2)$ \square

15. 顺序表排序

status sortList(SqList& L)

初始条件是线性表 L 已存在; 操作结果是将 L 由小到大排序;

本实验一开始, 我本来是准备用冒泡排序去解决问题的, 但随着课程的深入, 我逐渐地了解了各种不同的算法, 并选择了快速排序, 它具有时间复杂度更小的优点。

利用快速排序, 可将线性表中的元素快速排序。快速排序是一种高效的排序算法, 基于分治策略。其核心思想是通过选择一个基准值, 将待排序数组分割成两个子数组, 其中一个子数组的元素都小于基准值, 另一个子数组的元素都大于等于基准值。然后对这两个子数组分别递归地应用快速排序算法, 直到子数组长度为 1 或 0。最后, 将所有子数组的排序结果合并起来即可得到最终的排序结果。快速排序的关键在于快速地划分数组, 通常使用双指针实现。

它具有平均时间复杂度 $O(n \log n)$ 、最坏情况下 $O(n^2)$ 的特点, 且原地排序, 不需要额外的空间。

快速排序可参考下面的流程图:

16. 存至文件

status SaveList (SqList L, char FileName []);

将保存线性表为文件写成函数, 函数的参数是结构体类型变量 L 和文件名称 FileName。首先判断线性表是否存在, 如果存在, 然后判断文件内是否有内容, 文件为空或不存在时则打开或创建文件, 然后调用 fprintf 函数将表中的所有元素写入该文件中, 之后关闭文件。

17. 读取文件

status LoadList (SqList L, char FileName []);

将读取文件中的线性表写成函数, 函数的参数是结构体类型变量 L 和文件名称 FileName。首先判断 L 是否存在, 如果线性表 L 存在, 表示 L 中已经有数

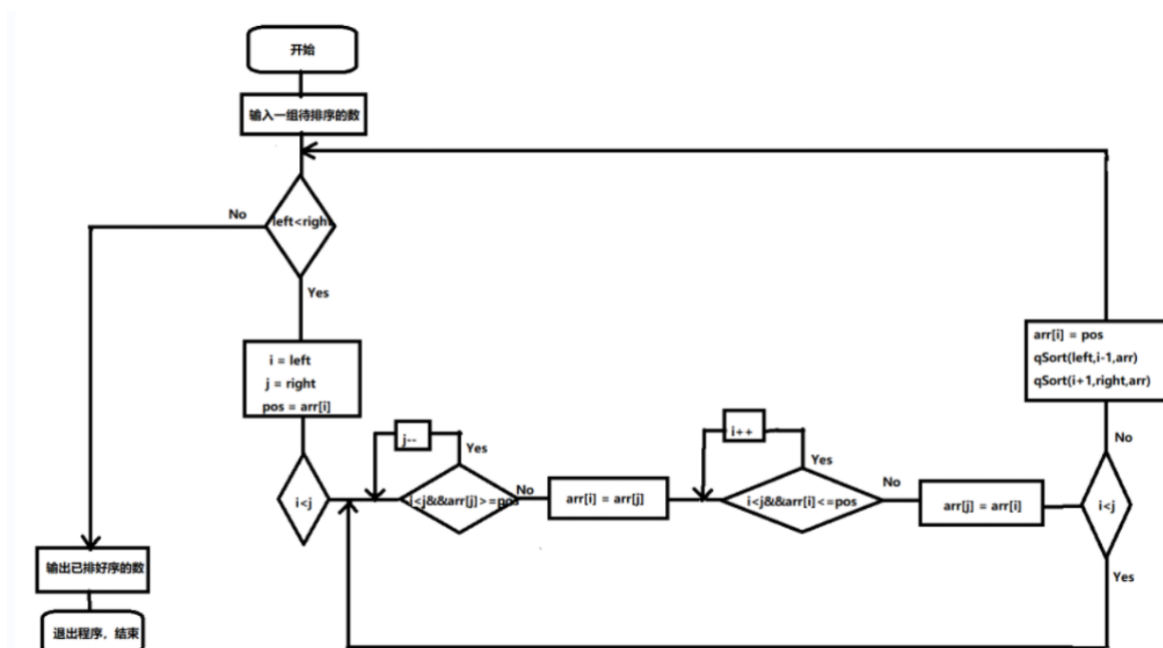


图 1-2 快速排序流程图

据，读入数据会覆盖原数据造成数据丢失，故只有 L 不存在时才可以继续操作。然后打开文件，调用 fscanf 函数将所有元素写入表中，之后关闭文件。

18. 多文件操作

```

status AddList (LISTS Lists, char ListName []);
status RemoveList (LISTS List, char ListName []);
int LocateList (LISTS Lists, char ListName []);
SqList* SwitchList(LISTS Lists, int i);
    
```

上述的函数用来实现多线性表管理的增删改查的操作, 因为函数实现的功能与上述的线性表算法类似, 因此不再赘述。

1.4 系统测试

本次实验使用 VSCode 进行编写。语言使用为 C/C++。

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

测试功能 及其对应序号	输入	预期输出	当前操作完成后 线性表状态
1.构造空线性表	\	线性表创建成功!	空表
1.构造空线性表	\	线性表创建失败!	空表,已存在的表无法再次构造
10.插入元素 (3 次)	1 4; 2 6; 3 8; (前者为插入位置,后者 为元素值)	插入成功! (3 次)	(3 个元素) 4 6 8
4.判空线性表	\	线性表不为空!	同上
5.求表长	\	线性表的长度是 3	同上
6.获取元素	2 (要获取的元素的位置)	第 2 个元素是 6	同上
7.定位元素	8 (要定位的元素)	该数字在第 3 位	同上
8.获取前驱	4 (要获取前驱的元素)	这个元素没有前驱!	同上
	6 (要获取前驱的元素)	前驱是 4	同上
9.获取后继	8 (要获取后继的元素)	这个元素没有后继!	同上
	4 (要获取后继的元素)	后继是 6	同上
12.遍历线性表	\	4 6 8	同上
13.文件保存/ 文件读取	\	保存成功! 载入成功!	同上(将线性表中的 数据保存到 test.txt 中后再从文件中读 取数据到线性表中)

图 1-3 系统测试

测试功能 及其对应序号	要管理的 线性表序号	输入	预期输出	当前操作完成后 多线性表状态
14.添加线性表 (2次)	1和2	湖北 河南 (表名)	插入成功! (2次)	多表内有两个空 表,表名分别为 湖北和河南
10.插入元素	1和2	1;1 2 3 (湖北) 2;4 5 6 (河南)	插入成功 (6次)	湖北: 1 2 3 河南: 4 5 6
16.查找线性表	1和2	河南	位置在 2	同上
15.移除线性表	1	湖北	移除成功!	河南: 4 5 6

图 1-4 系统测试

通过随机构造一组数据,测试附加功能,附加功能测试如下:

附加功能测试样例: -2, 1, -3, 4, -1, 2, 1, -5, 4

- 1、最大连续子数组和结果为 6
- 2、和为-1 的连续子数组个数为 5 个
- 3、从小到大排序结果为-5, -3, -2, -1, 1, 1, 2, 4, 4

图 1-5 系统测试

菜单如下:

```

Menu for Linear Table On Sequence Structure
-----
1. InitList      10. ListInsert
2. DestroyList  11. ListDelete
3. ClearList    12. ListTraverse
4. ListEmpty     13. SaveList
5. ListLength   14. LoadList
6. GetElem      15. AddList
7. LocateElem   16. RemoveList
8. PriorElem    17. LocateList
9. NextElem     18. SortList
19. MaxSubarray 20. SubarrayK
0.exit-----
请选择你的操作[0~20]:
    
```

图 1-6 菜单

综合上述的实验结果可知吗,实验结果基本符合预期,达到目的。

1.5 实验小结

本次实验让我加深了对线性表的概念、基本运算的理解，掌握了线性表的基本运算的实现，熟练了线性表的逻辑结构和物理结构的关系。

在编写程序和测试的过程中，遇到了诸多问题，例如如何设计多线性表操作，如何保证能够单独对某一线性表进行基本操作。解决方案是将其完整赋值给主函数中的全局变量，保证了集合中表的独立性，可以不受主函数中操作的影响，表之间可以分立进行。同时在进行顺序表的增删改查时，需要额外注意顺序表的边界问题。

更加理解了顺序存储结构的优缺点：顺序存储结构使用数组来存储数据，具有随机访问的优点，但在插入和删除操作时需要移动大量元素，效率较低。同时我更加注意代码的效率，这让我开始思考如何在特定场景下选择合适的存储结构以优化算法性能。

同时在附加功能的撰写上，我也收益颇丰，在解决最大连续子数组和的时候，通过自己的学习和研究使用了动态规划的方法，这让我对线性表和部分算法的理解更加深刻了，同时也锻炼了我解决问题的能力。

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。

2 基于邻接表的图实现

2.1 问题描述

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

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

2.2 系统设计

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

1. 创建图: 函数名称是 $CreateGraph(G,V,VR)$; 初始条件是 V 是图的顶点集, VR 是图的关系集; 操作结果是按 V 和 VR 的定义构造图 G ;
2. 销毁图: 函数名称是 $DestroyGraph(G)$; 初始条件图 G 已存在; 操作结果是销毁图 G ;
3. 查找顶点: 函数名称是 $LocateVex(G,u)$; 初始条件是图 G 存在, u 是和 G 中顶点关键字类型相同的给定值; 操作结果是若 u 在图 G 中存在, 返回关键字为 u 的顶点位置序号 (简称位序), 否则返回其它表示 “不存在” 的信息;
4. 顶点赋值: 函数名称是 $PutVex(G,u,value)$; 初始条件是图 G 存在, u 是和 G 中顶点关键字类型相同的给定值; 操作结果是对关键字为 u 的点赋值 $value$;
5. 获得第一邻接点: 函数名称是 $FirstAdjVex(G,u)$; 初始条件是图 G 存在, u 是 G 中顶点的位序; 操作结果是返回 u 对应顶点的第一个邻接顶点位序, 如果 u 的顶点没有邻接顶点, 否则返回其它表示 “不存在” 的信息;
6. 获得下一邻接点: 函数名称是 $NextAdjVex(G,v,w)$; 初始条件是图 G 存在, v 和 w 是 G 中两个顶点的位序, v 对应 G 的一个顶点, w 对应 v 的邻接顶点; 操作结果是返回 v 的 (相对于 w) 下一个邻接顶点的位序, 如果 w 是最后一个邻接顶点, 返回其它表示 “不存在” 的信息;

7. 插入顶点：函数名称是 `InsertVex(G,v)`；初始条件是图 G 存在， v 和 G 中的顶点具有相同特征；操作结果是在图 G 中增加新顶点 v 。（在这里也保持顶点关键字的唯一性）；
8. 删除顶点：函数名称是 `DeleteVex(G,v)`；初始条件是图 G 存在， v 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中删除关键字 v 对应的顶点以及相关的弧；
9. 插入弧：函数名称是 `InsertArc(G,v,w)`；初始条件是图 G 存在， v 、 w 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中增加弧 $\langle v,w \rangle$ ，如果图 G 是无向图，还需要增加 $\langle w,v \rangle$ ；
10. 删除弧：函数名称是 `DeleteArc(G,v,w)`；初始条件是图 G 存在， v 、 w 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中删除弧 $\langle v,w \rangle$ ，如果图 G 是无向图，还需要删除 $\langle w,v \rangle$ ；
11. 深度优先搜索遍历：函数名称是 `DFSTraverse(G,visit())`；初始条件是图 G 存在；操作结果是图 G 进行深度优先搜索遍历，依次对图中的每一个顶点使用函数 `visit` 访问一次，且仅访问一次；
12. 广度优先搜索遍历：函数名称是 `BFSTraverse(G,visit())`；初始条件是图 G 存在；操作结果是图 G 进行广度优先搜索遍历，依次对图中的每一个顶点使用函数 `visit` 访问一次，且仅访问一次。

附加功能如下：

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；
4. 实现图的文件形式保存：其中，需要设计文件数据记录格式，以高效保存图的数据逻辑结构 (D,R) 的完整信息；
5. 实现多个图管理：设计相应的数据结构管理多个图的查找、添加、移除等功能。

2.3 系统实现

2.3.1 菜单的演示系统

系统通过 while 语句多次输入 op 的值，通过 switch 语句具体实现用户选择的功能，并通过 system("cls") 清屏，提高用户的使用体验。(其中 op==0 时，退出该系统)

2.3.2 数据结构设计

1. 图：包括顶点集，vexnum,arcnum 和图的类别
 - (a) 顶点：包括顶点信息和指向邻近的第一条弧
 - (b) 边：包括顶点的位置编号和指向下一条弧的指针
2. 多图操作：elem 由 ALGraph 和 name 组成，还有 length 和 size。

2.3.3 函数功能的实现

1. 创建图

status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2]); 初始条件是 V 是图的顶点集，VR 是图的关系集；操作结果是按 V 和 VR 的定义构造图 G；

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

请参考算法2.1

算法 2.1. 创建图

Input: : Sequence of (V, E)

Output: : Initialized graph

procedure CreateGraph(*G*)

 CheckIsDuplicate(*V*)

for *i* ← 1 to *n* **do**

 Initialize(*V*[*i*])

```
end for
for all distinct pairs  $(i, j)$  where  $i \neq j$  do
    if  $V[i] = V[j]$  then
        return ERROR
    end if
end for
for  $k \leftarrow 1$  to  $e$  do
    AddEdge( $V[u], V[v]$ )
    AddEdge( $V[v], V[u]$ )
end for
 $G.vexnum \leftarrow n$ 
 $G.arcnum \leftarrow e$ 
return OK
end procedure
```

经分析，该算法空间复杂度为 $O(n + e)$ ，时间复杂度为 $O(n + e)$ 。

2. 销毁图

status DestroyGraph(ALGraph &G);

初始条件图 G 已存在；操作结果是销毁图 G；

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

流程图请参考图 2-1。

该算法空间复杂度为 $O(n + e)$ ，时间复杂度为 $O(1)$ 。

3. 查找节点

int LocateVex(ALGraph G,KeyType u);

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

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

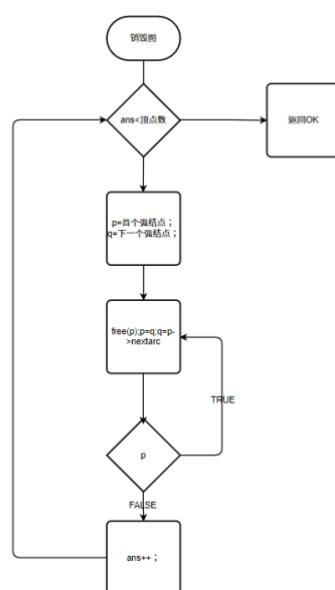


图 2-1 销毁图流程图

请参考算法2.2

算法 2.2. 查找节点

Input: : Sequence of (V, E)

Output: : int

procedure LocateVex(G, u)

for $i \leftarrow 1$ to n **do**

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

return i

end if

end for

return -1

end procedure

该算法空间复杂度为 $O(n)$ ，时间复杂度为 $O(1)$ 。

4. 顶点赋值

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

初始条件是图 G 存在， u 是和 G 中顶点关键字类型相同的给定值；操作结

果是对关键字为 u 的顶点赋值 $value$;

该算法的设计思想是,先利用查找结点找到该结点,而后将该结点的顶点值进行修改即可。

该算法空间复杂度为 $O(1)$,时间复杂度为 $O(n)$ 。

5. 获得第一邻接点

```
int FirstAdjVex(ALGraph G,KeyType u);
```

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

该算法的设计思想是,先找到该结点,随后直接返回该点的第一邻接点即可。(firstarc 的顶点位序)

该算法空间复杂度为 $O(1)$,时间复杂度为 $O(n)$ 。

6. 获得下一邻接点

```
int NextAdjVex(ALGraph G,KeyType v,KeyType w);
```

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

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

请参考算法2.3

算法 2.3. 获得下一邻接点

Input: : Sequence of (V, E)

Output: : int

procedure NextAdjVex(G, v, w)

$u \leftarrow \text{LocateVex}(G, v)$

for $i \leftarrow 1$ to $V[u].\text{arcnum}$ **do**

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

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

end if

```
end for
return -1
end procedure
```

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

7. 插入顶点

```
status InsertVex(ALGraph &G,VertexType v);
```

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

该算法的设计思想是, 直接在最末处添加结点, 并将顶点个数增加一即可。

该算法空间复杂度为 $O(1)$, 时间复杂度为 $O(1)$.

8. 删除结点

```
void DeleteVex_A_Node(ALGraph &G,VNode &Node,int index);
```

上述的函数是用来删除以 $Node$ 为起点的包括 $G.vertices[index]$ 的弧, 依次遍历, 分为 $Node$ 是不是 $G.vertices[index]$ 两种情况, 是的话所有有关的弧都要删除, 不是只需要删除与 $G.vertices[index]$ 相关的弧。

```
status DeleteVex(ALGraph &G,KeyType v);
```

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

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

该算法空间复杂度为 $O(1)$, 时间复杂度为 $O(n + e)$ 。

9. 插入弧

```
status InsertArc(ALGraph &G,KeyType v,KeyType w);
```

初始条件是图 G 存在, v 、 w 是和 G 中顶点关键字类型相同的给定值; 操作结果是在图 G 中增加弧 $\langle v,w \rangle$, 如果图 G 是无向图, 还需要增加 $\langle w,v \rangle$, 再 $++G.arcnum$;

该算法的设计思想是，将顶点 v 和顶点 w 找到，并且进行加边操作。加边操作采用头插法。

该算法空间复杂度为 $O(1)$ ，时间复杂度为 $O(n)$ 。

10. 删除弧

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

初始条件是图 G 存在， v 、 w 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中删除弧 $\langle v,w \rangle$ ，如果图 G 是无向图，还需要删除 $\langle w,v \rangle$ ；

请参考算法2.4

算法 2.4. 删除弧

Input: : v,w

Output: : ALGraph G'

procedure DeleteArc(G, v, w)

$w_found \leftarrow \text{false}$

$v_found \leftarrow \text{false}$

$u \leftarrow \text{LocateVex}(G, w)$ ▷ Find node w

if $u = -1$ **then**

return ERROR ▷ Node w not found

end if

$i \leftarrow 1$

while $i \leq V[u].arcnum$ **do** ▷ Traverse the arc list of node w

if $V[u][i] = v$ **then**

Remove $V[u][i]$ from $V[u]$ ▷ Remove if found

$w_found \leftarrow \text{true}$

end if

$i \leftarrow i + 1$

end while

if $w_found = \text{false}$ **then**

return ERROR ▷ Node v not found in the arc list of node w

end if

```

     $v\_found \leftarrow \text{true}$ 
     $u \leftarrow \text{LocateVex}(G, v)$  ▷ Find node  $v$ 
    if  $u = -1$  then
        return ERROR ▷ Node  $v$  not found
    end if
     $i \leftarrow 1$ 
    while  $i \leq V[u].\text{arcnum}$  do ▷ Traverse the arc list of node  $v$ 
        if  $V[u][i] = w$  then
            Remove  $V[u][i]$  from  $V[u]$  ▷ Remove if found
             $v\_found \leftarrow \text{true}$ 
        end if
         $i \leftarrow i + 1$ 
    end while
    if  $v\_found = \text{false}$  then
        return ERROR ▷ Node  $w$  not found in the arc list of node  $v$ 
    end if
    return OK ▷ Nodes  $v$  and  $w$  removed successfully
end procedure

```

11. 深度优先遍历

status DFSTraverse(ALGraph &G, void (*visit)(VertexType));

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；

具体的方法是，建立一个 $\text{visited}[]$ 数组以保存结点是否以访问的信息，以保证每个结点只被访问一次。如果我们正在访问一个结点，并且如果和它相邻的结点没有被访问，那么我们将跳转至下一结点进行进一步的搜索。

算法2.5如下：

算法 2.5. 深度优先遍历

Input: : Sequence of (V, E)

Output: : Visit of Each one

procedure DFS($G, v, visited$)

$visited[v] \leftarrow \mathbf{true}$

visit v

for all u adjacent to v **do**

if $\neg visited[u]$ **then**

DFS($G, u, visited$)

end if

end for

end procedure

因为每一结点仅被访问一次，故深度优先搜索的时间复杂度为 $O(n)$ 。

12. 广度优先遍历

status BFSTraverse(ALGraph &G,void(*visit)(VertexType));

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

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

具体的方法是，建立一个 $visited[]$ 数组以保存结点是否以访问的信息，以保证每个结点只被访问一次。如果我们正在访问一个结点，那么我们将和它相邻的并且和未被访问的结点压入访问队列中。每次在访问队列中去队首元素进行访问，并在 $visited$ 数组中标记其已被访问。

算法如下2.6

算法 2.6. 广度优先遍历

Input: : Sequence of (V, E)

Output: : Visit of Each one

procedure BFS(G, s)

$Q \leftarrow$ empty queue

```
visited[s] ← true
enqueue s into Q
while Q is not empty do
    v ← dequeue from Q
    print v
    for all u adjacent to v do
        if  $\neg$ visited[u] then
            visited[u] ← true
            enqueue u into Q
        end if
    end for
end while
end procedure
```

因为每一结点仅被访问一次，故深度优先搜索的时间复杂度为 $O(n)$ 。

13. 距离小于 k 的顶点集合

status VerticesSetLessThanK(ALGraph G,KeyType v,int k);

初始条件是图 G 存在；操作结果是返回与顶点 v 距离小于 k 的顶点集合；

算法思想：因为所有邻接点之间的距离都是 1，所以可以根据邻接矩阵去求解这个问题。

第一步：现将邻接表矩阵化，一次遍历每一条边，并将边对应的矩阵位置的元素设置为 1；

第二步：对连通矩阵进行 k 次方，如果非零即为距离小于 k 的顶点。（或者通过连通矩阵进行 k 次遍历，能到的节点标记为一，则可以得到该集合）

14. 顶点间最短路径和长度

int ShortestPathLength(ALGraph G,KeyType v,KeyType 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；

思想一：同样的根据该无向图的特殊性，可以根据该图的邻接矩阵的 n 次方判断，当矩阵相应位置的元素为 1 且 n 最小时，可以求出最小的距离即为 n 。

思想二：通过离散数学和数据结构课本的介绍，最短路径还可以采取迪杰斯特拉算法和弗洛伊德算法实现，这里可以使用迪杰斯特拉算法，本质上这是一种贪心算法，算法如下2.7

算法 2.7. 迪杰斯特拉算法

```
procedure Dijkstra( $G, s$ )  
   $dist[s] \leftarrow 0$   
  for all vertices  $v$  in  $G$  do  
    if  $v \neq s$  then  
       $dist[v] \leftarrow \infty$   
    end if  
     $visited[v] \leftarrow \text{false}$   
  end for  
   $Q \leftarrow$  empty priority queue  
  enqueue ( $s, 0$ ) into  $Q$   
  while  $Q$  is not empty do  
     $(v, d) \leftarrow$  dequeue from  $Q$   
    if  $visited[v]$  then  
      continue  
    end if  
     $visited[v] \leftarrow \text{true}$   
    for all neighbors  $u$  of  $v$  do  
       $alt \leftarrow dist[v] + \text{weight}(v, u)$   
      if  $alt < dist[u]$  then  
         $dist[u] \leftarrow alt$  ▷ Update tentative distance  
        enqueue ( $u, alt$ ) into  $Q$   
      end if  
    end for  
  end while  
end procedure
```

迪杰斯特拉算法中，每个结点和边仅被搜索一次，可以得到该算法时间复杂度为 $O((V + E) / \log V)$ 。

15. 图的连通分量

`int ConnectedComponentsNums(ALGraph &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；

通过从不同的节点进行深度优先遍历，如果能一次遍历完所有节点，连通分量即为 1，以此类推。

具体算法如下2.8

算法 2.8. 图的连通分量

Input: : Sequence of (V, E)

Output: : cnt

Initialize array *book* with all elements set to 0

$cnt \leftarrow 0$

if $G.vertices$ is **NULL** **then**

return 0

end if

for $i \leftarrow 0$ **to** 99 **do** ▷ Assuming the maximum size of *book* is 100

$book[i] \leftarrow 0$

end for

for $i \leftarrow 0$ **to** $G.vexnum - 1$ **do**

if $book[i] == 0$ **then**

$cnt \leftarrow cnt + 1$

$book[i] \leftarrow 1$

 BFS(G, i)

end if

end for

return cnt

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

16. 文件的保存和读取

```
status SaveGraph(ALGraph G, char FileName[]);
```

```
status LoadGraph(ALGraph &G, char FileName[]);
```

需要设计文件数据记录格式以高效保存图的数据逻辑结构 (D,R) 的完整信息;

我设计的储存结构是, 将其顶点依次存下, 在将其边存下即可。(即存储 V 和 VR 的信息)。

17. 实现多个图管理

```
status AddALGraph(ALGraphs &GS, char ListName[]);
```

```
status LocateALGraph(ALGraphs &GS, char ListName[]);
```

```
status RemoveALGraph(ALGraphs &GS, char ListName[]);
```

设计相应的数据结构管理多个图的查找、添加、移除等功能。

只需创建多图结构数组, 后将其实现增加删除创建等函数即可。(与顺序表类似)。

2.4 系统测试

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

菜单如下：

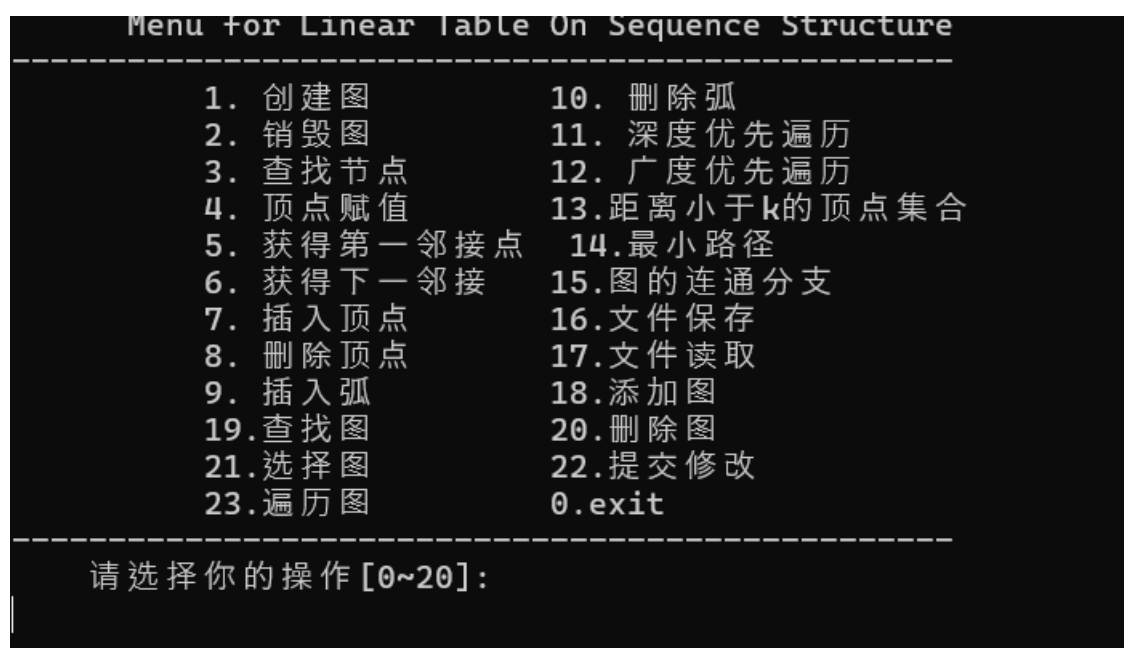


图 2-2 图实验的菜单

2.4.1 实际测试

1. 创建图如下：实验结果：创建成功！

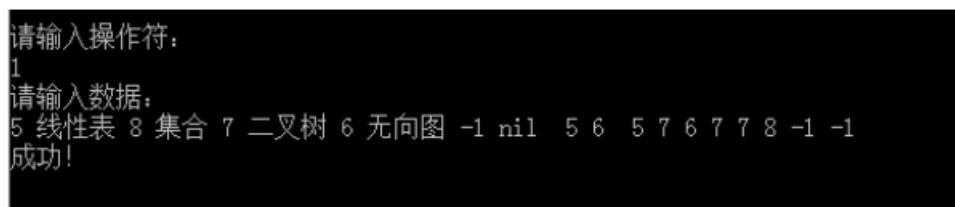


图 2-3 创建图

2. 查找节点：关键字为 6 的节点位序为 3，正确！
3. 插入新节点：关键词为 9，名称为 “456”，插入成功！
4. 该图连通分量为 2，是正确 的答案。(连通分量的测试)

5. 插入弧 (8,9)，插入成功

6. 遍历结果如下：



```
请输入操作符:  
11  
5, 线性表 7, 二叉树 8, 集合 9, 456 6, 无向图
```

图 2-4 遍历结果

该结果说明：上述的插入弧和节点的操作均成功执行且遍历没有问题。

7. 附加功能：距离小于 k 的顶点个数：

如下图：



```
请输入关键词和距离:  
5 1  
##0
```

图 2-5 距离小于 k 功能测试

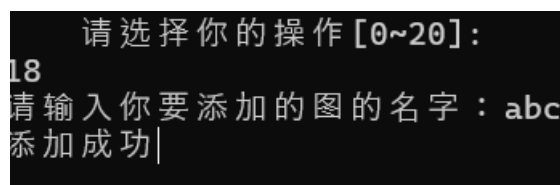
有实验结果可知：基本可行。

8. 文件操作：

读取文件操作：成功，从名为“tounge.txt”的文件中成功读取并创建了图。

9. 多图操作测试：

添加图，名为“abc”：



```
请选择你的操作[0~20]:  
18  
请输入你要添加的图的名字: abc  
添加成功|
```

图 2-6 添加图

查找图“abc”：显示结果为 1，说明位序正确。

删除图和选择图的操作也均没有问题：说明多文件操作基本符合要求。

综上所述：本次实验的所实现的功能基本符合要求。

2.5 实验小结

通过本次实验，我深入学习了图的基本概念以及如何使用邻接表来实现图的表示和操作。以下是我在实验中的主要收获：

1. 理解邻接表的优势：相比邻接矩阵，邻接表更适合表示稀疏图，因为它只存储每个顶点的邻居节点，节省了空间。我了解到邻接表在处理大型图时具有更好的性能。

2. 在掌握邻接表的过程中，在某些问题的解决上，邻接矩阵更具有优势，比如距离小于 k 的顶点集合，因此我便尝试了把邻接表转换成邻接矩阵，二者的相互转化有利于我们采取更合适的手段去合适地解决问题。

2. 掌握图的基本操作：在实验中，我学会了如何使用邻接表来实现图的基本操作，包括添加顶点、添加边、查找邻居节点等。这些操作是对图进行构建和遍历时必不可少的。

4. 思考时间复杂度：在使用邻接表表示图时，我开始思考不同操作的时间复杂度。例如，在邻接表中查找邻居节点的时间复杂度取决于顶点的度数，通常为 $O(d)$ ，其中 (d) 是顶点的度数。

5. 实践图算法：通过实现邻接表的图，我有机会实践图算法，例如深度优先搜索（DFS）和广度优先搜索（BFS）。这些算法是解决图相关问题的重要工具。

6. 加强编程能力：通过编写代码实现图的基本操作，我加深了对数据结构和算法的理解，并提高了编程能力。我学会了如何将理论知识转化为实际的程序。

3 课程的收获和建议

1. 学习收获

在理论课上，我系统地学习了数据结构的基本概念、算法复杂度分析以及不同数据结构在解决问题时的优缺点。通过课堂讲解和理论练习，我深入了解了线性结构如数组、链表，以及非线性结构如树、图等特性和应用场景。理论课程的收获不仅包括对每种数据结构操作的理解，还涉及如何选择合适的数据结构来优化算法性能。我学会了使用数学模型和算法分析工具（如时间复杂度和空间复杂度）来评估和比较不同算法和数据结构的效率，这为我后续的编程和算法设计打下了坚实的基础。

实践课则为我提供了将理论知识应用到实际中的机会。通过编程实现各种数据结构和算法，我不仅加深了对理论概念的理解，还锻炼了自己的编程能力和解决问题的实际能力。在实践课程中，我学会了如何在编程中灵活运用数组、链表、树和图等数据结构，解决现实生活中复杂的问题。例如，在实现线性表时，我理解了数据的插入、删除、查找操作对应的数据结构操作和算法设计思路；在实现图的算法时，我掌握了深度优先搜索（DFS）、广度优先搜索（BFS）等基础算法的具体实现方法。

通过学习数据结构实验，我深刻理解了不同数据结构的不同具体实现，并加深了对这四种数据结构的认识。

3.1 基于顺序存储结构的线性表实现

理解线性表的概念：通过实验，我深入理解了线性表作为一种基本的数据结构，它由相同类型的数据元素组成，并且元素之间存在线性关系。

掌握顺序存储结构：实验中我学习了如何使用数组来表示线性表，并实现了顺序存储结构。这包括初始化、插入、删除、查找和销毁等基本操作。

操作实现：

边界条件处理：在实现操作时，我学会了如何处理边界条件，例如插入和删除操作时对位置的有效性检查。

时间复杂度分析：通过实验，我理解了顺序表操作的时间复杂度，例如插入和删除操作在最坏情况下需要移动多个元素，时间复杂度为 $O(n)$ 。

通过多种方式实现相同的功能：比如用枚举法和动态规划实现相同的功能，显然动态规划的时间复杂度更低，这样的方式拓展了我的思维，加深了我对不同算法的理解；再比如用快速排序和冒泡排序解决同一个问题，二者虽然实现的功能是一样的，但是在时间复杂度上却具有巨大的差别，这也是我在数据结构理论课上学到的东西，通过此次的实践，我更加加深了对排序算法的理解，并加强了编程的能力。

数据结构的应用：通过实现线性表，我了解了数据结构在实际编程中的应用，以及如何根据实际需求选择合适的数据结构。

问题解决能力：在实验过程中，我遇到了各种问题，如内存分配、数组越界等，通过解决这些问题，我们提高了问题解决能力。

通过这次实验，我不仅掌握了线性表顺序存储结构的设计与实现方法，还提高了编程实践能力和问题解决能力，为今后的编程学习和工作打下了坚实的基础。

3.2 基于链式存储结构的线性表实现

理解了链式存储结构的基本概念和特点，包括单链表、双链表、循环链表和静态链表等类型。

熟悉了链表的基本操作，包括插入、删除、查找、遍历、逆置等。

了解了链式存储结构的优点和缺点，如不要求大片连续空间，改变容量方便，删除添加更简单，但不可随机存取，需要耗费一定空间存放指针。

通过实验，加深了对线性表链式存储结构设计 with 基本操作的实现的理解。

学会了如何使用链式存储结构解决实际问题，如实现链表的逆置、合并等操作。

线性表总共通过顺序结构和链式结构两种方式实现，两项对比之下，我更加理解了二者各自具有的优势与缺点，理解了在哪种情况下该使用哪一种形式。

通过实验，加深了对数据结构与算法分析课程内容的理解和掌握。

3.3 基于二叉链表的二叉树的实现

理解二叉树的基本概念：通过实现二叉链表，我深入理解了二叉树的结构和基本概念，包括树的节点、父子关系以及二叉树的性质。

掌握二叉链表的实现方式：我学会了如何使用指针来表示二叉树节点，并通过指针链接节点来构建二叉树的结构。这种链表方式使得插入和删除操作更加灵活和高效。

实现基本操作：在实验中，我实现了二叉树的基本操作，如插入节点、删除节点、搜索节点以及树的遍历（前序、中序、后序遍对基本操作的实现也更加夯实了我的基础。

深入了解递归算法：二叉树的许多操作（如遍历求深度等）都可以通过递归算法来实现。在实验过程中，我加深了对递归算法在树结构中应用的理解，并学会了如何编写和调试递归函数，写递归代码在我初学时就是一件全凭感觉的事，而在多次的实践过后，我也略微找到了一些门路，摸清了一些方法。

应用到实际问题：通过实现二叉链表，我能够将学到的知识应用到解决实际的问题中。例如，树的搜索和遍历算法可以用于查找树中的特定元素或生成树的不同表示形式。

3.4 基于邻接表的图实现

理解图的概念：通过实验，我深入理解了图作为一种数据结构，它由顶点和边组成，顶点表示对象，边表示对象之间的关系。

掌握邻接表存储结构：实验中我学习了如何使用邻接表来表示图，并实现了邻接表的创建、插入边、删除边、查找顶点等基本操作。

掌握不同储存结构的优劣势：在解决不同的问题时，可采取不一样的结构，比如邻接矩阵在一些问题的解决上有妙用，邻接表处理稀疏图更好等等。

操作实现：

时间复杂度分析：通过实验，我理解了邻接表操作的时间复杂度，例如插入和删除边操作的时间复杂度为 $O(1)$ ，查找顶点操作的时间复杂度为 $O(n)$ 。

数据结构的应用：通过实现图，我了解了数据结构在实际编程中的应用，以及如何根据实际需求选择合适的数据结构。

通过多种方式实现相同的功能：比如用矩阵的乘法和迪杰斯特拉算法求最小路径，二者方法不同，所实现的功能却是相同的，这样的操作加深了我对算法的理解，加深了对图的理解。

问题解决能力：在实验过程中，我遇到了各种问题，如内存分配、链表操作等，通过解决这些问题，我们提高了问题解决能力。

通过这次实验，更好地掌握了图邻接表存储结构的设计与实现方法。

4 参考文献

- [1] 严蔚敏等. 数据结构 (C 语言版). 清华大学出版社
- [2] 袁凌, 祝建华等. 数据结构, 人民邮电出版社

A 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 <stdlib.h>

/*—————page 10 on textbook —————*/

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

1  #include <stdio.h>
2  #include <Windows.h>
3  #include <string.h>
4
5  #define TRUE 1
6  #define FALSE 0
7  #define OK 1
8  #define ERROR 0
9  #define INFEASIBLE -1
10 #define OVERFLOW -2
11
12 typedef int status;
13 typedef int ElemType; //数据元素类型定义
14
15
16 #define LIST_INIT_SIZE 100
17 #define LISTINCREMENT 10
18
```



```
19 typedef int ElemType;
20
21
22 typedef struct { //顺序表（顺序结构）的定义
23     ElemType* elem;
24     int length;
25     int listsize;
26 } SqList;
27
28 typedef struct { //线性表的管理表定义
29     struct {
30         char name[30];
31         SqList L;
32     } elem[10];
33     int length;
34     int listsize;
35 } LISTS;
36
37
38
39
40 void sort_list(SqList L);
41
42 int cmp(const void* a, const void* b);
43
44 status InitList(SqList& L);
45
46 status DestroyList(SqList& L);
47
48 status ClearList(SqList& L);
49
50 status ListEmpty(SqList L);
51
52 int ListLength(SqList L);
53
```

```
54 status GetElem(SqList L, int i, ElemType& e);
55
56 status LocateElem(SqList L, ElemType e);
57
58 status PriorElem(SqList L, ElemType cur, ElemType& pre_e);
59
60 status NextElem(SqList L, ElemType cur, ElemType& next_e);
61
62 status ListInsert(SqList& L, int i, ElemType e);
63
64 status ListDelete(SqList& L, int i, ElemType& e);
65 status ListTraverse(SqList L);
66
67 bool is_same(char* str1, char* str2);
68
69 status SaveList(SqList L, const char FileName[]);
70
71 status LoadList(SqList& L, const char FileName[]);
72
73 status AddList(LISTS& Lists, const char ListName[]);
74
75 status RemoveList(LISTS& Lists, char ListName[]);
76
77 int LocateList(LISTS Lists, const char ListName[]);
78 #include <stdint.h>
79
80 int max(int a, int b){
81     return a>b? a:b;
82 }
83 int cmp(const void* a, const void* b)
84 {
85     return *(int*)a - *(int*)b;
86 }
87
88 int max_sub_array(SqList L)
```

```
89 {
90     if (L.length == 0)
91     {
92         return INT64_MIN;
93     }
94     int result = 0;
95     int* dp = (int*)malloc(sizeof(int) * L.length);
96     dp[0] = L.elem[0];
97     for (int i = 1; i < L.length; i++)
98     {
99         dp[i] = max(dp[i - 1] + L.elem[i], L.elem[i]);
100         result = max(result, dp[i]);
101     }
102     free(dp);
103     return result;
104 }
105 void sort_list(SqList L)
106 {
107     qsort(L.elem, L.length, sizeof(int), cmp);
108     return;
109 }
110
111
112
113 status InitList(SqList& L)
114 {
115
116     // 线性表L不存在，构造一个空的线性表，返回OK，
117     // 否则返回INFEASIBLE。
118
119     // 请在这里补充代码，完成本关任务
120     //***** Begin *****/
121     //如果线性表不存在
122     if (L.elem == NULL)
123     {
```

```
123         L.elem = (ElemType*)malloc(sizeof(int)
           * LIST_INIT_SIZE);    //为elem数组
           开辟空间
124         L.length = 0;    //线性表的初始长度设
           置为0
125         L.listsize = LIST_INIT_SIZE;    //线性
           表的最大长度
126         return OK;
127     }
128     //如果线性表存在, 返回INFEASIBLE
129     return INFEASIBLE;
130     /***** End *****/
131
132 }
133
134
135
136 status DestroyList(SqList& L)
137 {
138     if (L.elem != NULL)
139     {
140         L.elem = NULL;
141         free(L.elem);
142         return OK;
143     }
144
145     return INFEASIBLE;
146 }
147
148
149 status ClearList(SqList& L)
150 {
151     if (L.elem != NULL)
152     {
153         //删除线性表中的元素
```

```
154             L.length = 0;
155
156             return OK;
157         }
158         //如果线性表不存在, 返回INFEASIBLE
159         return INFEASIBLE;
160     }
161
162
163
164     status ListEmpty(SqList L)
165     // 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。
166     {
167         // 请在这里补充代码, 完成本关任务
168         /***** Begin *****/
169         //如果线性表不存在
170         if (L.elem == NULL)
171         {
172             return INFEASIBLE;
173         }
174         else
175         {
176
177             if (L.length == 0)
178             {
179                 return true;
180             }
181             else
182             {
183                 return false;
184             }
185         }
186         /***** End *****/
187     }
```

```
188
189
190 status ListLength(SqList L)
191 // 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。
192 {
193     // 请在这里补充代码，完成本关任务
194     /***** Begin *****/
195     if (L.elem == NULL)
196     {
197         return INFEASIBLE;
198     }
199     return L.length;
200
201     /***** End *****/
202 }
203
204 status GetElem(SqList L, int i, ElemType& e)
205 // 如果线性表L存在，获取线性表L的第i个元素，保存在e中，返回OK
    ; 如果i不合法，返回ERROR；如果线性表L不存在，返回INFEASIBLE
    。
206 {
207     // 请在这里补充代码，完成本关任务
208     /***** Begin *****/
209     if (L.elem == NULL)
210     {
211         return INFEASIBLE;
212     }
213     if (i > L.length || i < 1)
214     {
215         return ERROR;
216     }
217     e = L.elem[i - 1];
218     return OK;
219     /***** End *****/
220 }
```

```
221
222 int LocateElem(SqList L, ElemType e)
223 // 如果线性表L存在，查找元素e在线性表L中的位置序号并返回该序
    号；如果e不存在，返回0；当线性表L不存在时，返回INFEASIBLE
    (即-1)。
224 {
225     // 请在这里补充代码，完成本关任务
226     /***** Begin *****/
227     if (L.elem == NULL)
228     {
229         return INFEASIBLE;
230     }
231     for (int i = 0; i < L.length; i++)
232     {
233         if (L.elem[i] == e)
234         {
235             return i + 1;
236         }
237     }
238     return 0;
239
240     /***** End *****/
241 }
242
243 status PriorElem(SqList L, ElemType e, ElemType& pre)
244 // 如果线性表L存在，获取线性表L中元素e的前驱，保存在pre中，返
    回OK；如果没有前驱，返回ERROR；如果线性表L不存在，返回
    INFEASIBLE。
245 {
246     // 请在这里补充代码，完成本关任务
247     /***** Begin *****/
248     if (L.elem == NULL)
249     {
250         return INFEASIBLE;
251     }
```

```
252
253     for (int i = 0; i < L.length; i++)
254     {
255         if (L.elem[i] == e)
256         {
257             if (i == 0)
258             {
259                 return ERROR;
260             }
261             pre = L.elem[i - 1];
262             return OK;
263         }
264     }
265     return ERROR;
266     /***** End *****/
267 }
268
269 status NextElem(SqList L, ElemType e, ElemType& next)
270 // 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。
271 {
272     // 请在这里补充代码, 完成本关任务
273     /***** Begin *****/
274     if (L.elem == NULL)
275     {
276         return INFEASIBLE;
277     }
278
279     for (int i = 0; i < L.length; i++)
280     {
281         if (L.elem[i] == e)
282         {
283             if (i == L.length - 1)
284             {
```



```
285                                     return ERROR;
286                                     }
287                                     next = L.elem[i + 1];
288                                     return OK;
289                                     }
290     }
291     return ERROR;
292     ***** End *****
293 }
294
295 status ListInsert(SqList& L, int i, ElemType e)
296 // 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。
297 {
298     // 请在这里补充代码，完成本关任务
299     ***** Begin *****
300     if (L.elem == NULL)
301     {
302         return INFEASIBLE;
303     }
304
305     if (i <= 0 || i > L.length + 1)
306     {
307         return ERROR;
308     }
309     L.length++;
310     L.listsize++;
311     int index = L.length - 1;
312     for (index; index >= i; index--)
313     {
314         L.elem[index] = L.elem[index - 1];
315     }
316     L.elem[index] = e;
317     return OK;
```

```
318          ***** End *****
319  }
320
321  status ListDelete(SqList& L, int i, ElemType& e)
322  // 如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回
    OK；当删除位置不正确时，返回ERROR；如果线性表L不存在，返回
    INFEASIBLE。
323  {
324      // 请在这里补充代码，完成本关任务
325      ***** Begin *****
326      if (L.elem == NULL)
327      {
328          return INFEASIBLE;
329      }
330      if (i <= 0 || i > L.length)
331      {
332          return ERROR;
333      }
334      e = L.elem[i - 1];
335      for (int j = i - 1; j < L.length; j++)
336      {
337          L.elem[j] = L.elem[j + 1];
338      }
339      L.length--;
340      return OK;
341      ***** End *****
342  }
343
344  status ListTraverse(SqList L)
345  // 如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，
    返回OK；如果线性表L不存在，返回INFEASIBLE。
346  {
347      // 请在这里补充代码，完成本关任务
348      ***** Begin *****
349      if (L.elem == NULL)
```

```
350     {
351         return INFEASIBLE;
352     }
353     for (int i = 0; i < L.length; i++)
354     {
355         if (i == L.length - 1)
356         {
357             printf("%d", L.elem[i]);
358         }
359         else
360         {
361             printf("%d ", L.elem[i]);
362         }
363     }
364 }
365 return OK;
366 ***** End *****
367 }
368
369
370
371 status SaveList(SqList L, const char FileName[])
372 // 如果线性表L存在，将线性表L的元素写到FileName文件中，返回
OK，否则返回INFEASIBLE。
373 {
374     // 请在这里补充代码，完成本关任务
375     ***** Begin *****
376     if (L.elem == NULL)
377     {
378         return INFEASIBLE;
379     }
380     FILE* stream;
381     stream = fopen(FileName, "w");
382     for (int i = 0; i < L.length; i++)
383     {
```

华中科技大学课程实验报告

```
384             fprintf(stream, "%d ", L.elem[i]);
385         }
386         fclose(stream);
387         return OK;
388         ***** End *****/
389     }
390     status LoadList(SqList& L, const char FileName[])
391     // 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。
392     {
393         // 请在这里补充代码, 完成本关任务
394         ***** Begin *****/
395         if (L.elem != NULL)
396         {
397             return INFEASIBLE;
398         }
399         L.elem = (ElemType*)malloc(sizeof(ElemType) *
400                                     LIST_INIT_SIZE);
401         if (L.elem == NULL)
402         {
403             return INFEASIBLE;
404         }
405         L.length = 0;
406         FILE* stream;
407         stream = fopen(FileName, "r");
408         while (fscanf(stream, "%d", &L.elem[L.length++]) !=
409                 EOF);
410         L.length--;
411         fclose(stream);
412         return OK;
413         ***** End *****/
414     }
415     status AddList(LISTS& Lists, const char ListName[])
416     // 只需要在Lists中增加一个名称为ListName的空线性表, 线性表数据
```

又后台测试程序插入。

```
416 {
417     // 请在这里补充代码，完成本关任务
418     /***** Begin *****/
419     int i = 0;
420     for (i; ListName[i] != '\0'; i++)
421     {
422         Lists.elem[Lists.length].name[i] = ListName[i];
423     }
424     Lists.elem[Lists.length].L.elem = (ElemType*)malloc(
425         sizeof(ElemType) * 10);
426     Lists.elem[Lists.length].L.listsize = 10;
427     Lists.elem[Lists.length].L.length = 0;
428     Lists.elem[Lists.length].name[i] = '\0';
429     Lists.length++;
430     return OK;
431     /***** End *****/
432 }
433
434
435
436 status RemoveList(LISTS& Lists, char ListName[])
437 // Lists 中删除一个名称为 ListName 的线性表
438 {
439     // 请在这里补充代码，完成本关任务
440     /***** Begin *****/
441     for (int i = 0; i < Lists.length; i++)
442     {
443         if (is_same(Lists.elem[i].name, ListName))
444         {
445             for (int j = i; j < Lists.length - 1;
446                 j++)
```

```
447             Lists.elem[i] = Lists.elem[i +
                                     1];
448         }
449         Lists.length--;
450         return OK;
451     }
452 }
453 return false;
454
455     /******* End *****/
456 }
457
458
459 bool is_same(char* str1, char* str2)
460 {
461     if (strlen(str1) != strlen(str2))
462     {
463         return false;
464     }
465     int n = strlen(str1);
466     for (int i = 0; i < n; i++)
467     {
468         if (str1[i] != str2[i])
469         {
470             return false;
471         }
472     }
473     return true;
474 }
475
476 int LocateList(LISTS Lists, const char ListName[])
477 // 在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，
    否则返回0
478 {
479     // 请在这里补充代码，完成本关任务
```

```
480      /****** Begin *****/
481      for (int i = 0; i < Lists.length; i++)
482      {
483          if (strcmp(Lists.elem[i].name, ListName) == 0)
484          {
485              return i + 1;
486          }
487      }
488      return 0;
489
490      /****** End *****/
491  }
492  int SubarrayK(int K, SqList L) {
493      int count=0;
494      for(int i=0;i<L.length;++i){
495          int sum=0;
496          for(int j=i;j<L.length;++j){
497              sum+=L.elem[j];
498              if(sum==K) count++;
499          }
500      }
501      return count;
502  }
503  ElemType SubArrayNum(SqList L, int k) {
504      if (L.length == 0&&L.elem) return ERROR; // 空表直接返回0
505      else if(L.elem==NULL){
506          return INFEASIBLE;
507      }
508      int count = 0;
509      int prefixSum = 0;
510      int prefixSumCounts[20001] = {0};
511      prefixSumCounts[0] = 1; // 初始化前缀和为0的次数为1
512
513      for (int i = 0; i < L.length; i++) {
514          prefixSum += L.elem[i];
```

```
515      // 如果 prefixSum - k 在之前出现过, 则找到了一个和为 k 的子
      数组
516      if (prefixSum - k >= 0 && prefixSumCounts[prefixSum -
          k] > 0) {
517          count += prefixSumCounts[prefixSum - k];
518      }
519      // 更新当前前缀和出现的次数
520      if (prefixSum >= 0 && prefixSum < 20001) { // 确保不会
          越界
521          prefixSumCounts[prefixSum]++;
522      }
523  }
524  return count;
525 }
526 int main(void)
527 {
528     SqList L;
529     LIST* lists;
530     lists.length = 0;
531     lists.listsize = 10;
532     L.elem = NULL;
533     L.listsize = LIST_INIT_SIZE;
534     L.length = 0;
535     int op = 1;
536     int i;
537     int e;
538     int x;
539     int state;
540     char name[100];
541     while (op)
542     {
543         system("cls");
544         printf("\n\n");
545         printf("      Menu for Linear Table On
          Sequence Structure \n");
```



```
546         printf("
           -----|
           n");
547         printf("          1. InitList          10.
           ListInsert\n");
548         printf("          2. DestroyList       11.
           ListDelete\n");
549         printf("          3. ClearList          12.
           ListTraverse\n");
550         printf("          4. ListEmpty          13.
           SaveList\n");
551         printf("          5. ListLength         14.
           LoadList\n");
552         printf("          6. GetElem            15. AddList
           \n");
553         printf("          7. LocateElem         16.
           RemoveList\n");
554         printf("          8. PriorElem          17.
           LocateList\n");
555         printf("          9. NextElem           18.
           SortList\n");
556         printf("         19. MaxSubarray        20.
           SubarrayK\n");
557         printf("         0. exit");
558         printf("
           -----|
           n");
559         printf("      请选择你的操作[0~20]:\n");
560         scanf("%d", &op);
561         switch (op)
562         {
563         case 1:
564             state = InitList(L);
565             if (state == INFEASIBLE)
566             {
```

```
567             printf("线性表创建失败。");
568         }
569         else
570         {
571             printf("线性表创建成功。");
572         }
573         break;
574     case 2:
575         state = DestroyList(L);
576         if (state == OK)
577         {
578             printf("销毁成功");
579         }
580         else
581         {
582             printf("线性表不存在。");
583         }
584         break;
585     case 3:
586         state = ClearList(L);
587         if (state == -1)
588         {
589             printf("线性表不存在。\\n");
590         }
591         else
592         {
593             printf("删除成功。\\n");
594         }
595         break;
596     case 4:
597         if (ListEmpty(L) == TRUE)
598         {
599             printf("线性表是空的。\\n");
600         }
601         else if (ListEmpty(L) == INFEASIBLE)
```

```
602         {
603             printf("线性表不存在。\\n");
604         }
605         else
606         {
607             printf("线性表不是空的。\\n");
608         }
609         break;
610     case 5:
611         state = ListLength(L);
612         if (state == INFEASIBLE)
613         {
614             printf("线性表不存在。\\n");
615         }
616         else
617         {
618             printf("线性表的长度为:%d\\n",
619                 state);
620         }
621         break;
622     case 6:
623         if (L.elem == NULL)
624         {
625             printf("线性表不存在。\\n");
626             break;
627         }
628         printf("请输入你想获取元素的序号: ");
629         scanf("%d", &i);
630         state = GetElem(L, i, e);
631         if (state == -1)
632         {
633             printf("线性表不存在。\\n");
634         }
635         else if (state == 0)
```

```
636
637             printf("%d不合法。\\n", i);
638         }
639         else
640         {
641             printf("序号为%d的元素为: %d。\\n", i, e);
642         }
643
644         break;
645     case 7:
646         if (L.elem == NULL)
647         {
648             printf("线性表不存在。\\n");
649             break;
650         }
651         printf("请输入你想查找的元素: \\n");
652         scanf("%d", &e);
653         state = LocateElem(L, e);
654         if (state == -1)
655         {
656             printf("线性表不存在。\\n");
657         }
658         if (state == 0)
659         {
660             printf("元素%d不存在。\\n", e);
661         }
662         if (state != 0 && state != -1)
663         {
664             printf("%d的序号为%d。\\n", e,
665                 state);
666         }
667         break;
668     case 8:
669         if (L.elem == NULL)
```

```
669         {
670             printf("线性表不存在。\\n");
671             break;
672         }
673     printf("请输入目标元素: \\n");
674     scanf("%d", &e);
675     state = PriorElem(L, e, x);
676     if (state == INFEASIBLE)
677     {
678         printf("线性表不存在。\\n");
679     }
680     else if (state)
681     {
682
683         printf("%d的前驱元素为%d。\\n",
684             e, x);
685     }
686     else
687     {
688         if (LocateElem(L, e) == 0)
689         {
690             printf("%d不存在。", e);
691             break;
692         }
693         printf("%d无前驱元素\\n", e);
694     }
695     break;
696 case 9:
697     if (L.elem == NULL)
698     {
699         printf("线性表不存在。\\n");
700         break;
701     }
702     printf("请输入你想获得后继元素的元素:
```

```

    ");
702     scanf("%d", &e);
703     state = NextElem(L, e, x);
704     if (state == INFEASIBLE)
705     {
706         printf("线性表不存在。\\n");
707     }
708     else if (state)
709     {
710         printf("%d的后继元素为: %d\\n",
711             e, x);
712     }
713     else
714     {
715         if (LocateElem(L, e) == 0)
716         {
717             printf("%d不存在。\\n",
718                 e);
719             break;
720         }
721         printf("%d无后继元素。\\n", e);
722     }
723     break;
724 case 10:
725     if (L.elem == NULL)
726     {
727         printf("线性表不存在。\\n");
728         break;
729     }
730     printf("请输入你想插入的序号和元素: \\n");
731     scanf("%d %d", &i, &e);
732     state = ListInsert(L, i, e);
733     if (state == -1)
```

```
732         {
733             printf("线性表不存在。\\n");
734         }
735         else if (state)
736         {
737             printf("插入成功。\\n");
738         }
739         else
740         {
741             printf("输入的序号无效。\\n");
742         }
743         break;
744     case 11:
745         if (L.elem == NULL)
746         {
747             printf("线性表不存在。\\n");
748             break;
749         }
750         printf("请输入你想删除元素的序号：");
751         scanf("%d", &i);
752         state = ListDelete(L, i, e);
753         if (state == INFEASIBLE)
754         {
755             printf("线性表不存在。\\n");
756         }
757         else if (state)
758         {
759             printf("已成功删除元素%d。\\n",
760                 e);
761         }
762         else
763         {
764             printf("输入的序号无效。\\n");
765         }
766         break;
```

```
766         case 12:
767             if (L.elem == NULL)
768             {
769                 printf("线性表不存在。\\n");
770                 break;
771             }
772             ListTraverse(L);
773             break;
774         case 13:
775             state = SaveList(L, "C:\\Users\\LENOVO
776                 \\Desktop\\edcoder.txt");
777             if (state == -1)
778             {
779                 printf("线性表不存在。\\n");
780             }
781             else
782             {
783                 printf("保存完成。\\n");
784             }
785             break;
786         case 14:
787             state = LoadList(L, "C:\\Users\\LENOVO
788                 \\Desktop\\edcoder.txt");
789             if (state == -1)
790             {
791                 printf("线性表已存在，无法读
792                     入。\\n");
793             }
794             else
795             {
796                 printf("已成功读入。\\n");
797             }
798             break;
799         case 15:
800             printf("请输入要添加线性表的名称：");
```



```
798         scanf("%s", name, 100);
799         state = AddList(lists, name);
800         if (state == 1)
801         {
802             printf("线性表添加成功.");
803         }
804         else
805         {
806             printf("添加失败.");
807         }
808         break;
809     case 16:
810         printf("请输入要移除线性表的名称: ");
811         scanf("%s", name, 100);
812         state = RemoveList(lists, name);
813         if (state == 1)
814         {
815             printf("移除成功.");
816         }
817         else
818         {
819             printf("移除失败");
820         }
821         break;
822     case 17:
823         printf("请输入要查找线性表的名称: ");
824         scanf("%s", name);
825         state = LocateList(lists, name);
826         if (state == 0)
827         {
828             printf("该线性表不存在.");
829         }
830         else
831         {
832             printf("该线性表的序号为: %d",
```

```

state);
833         }
834         break;
835     case 18:
836         sort_list(L);
837         printf("排序成功");
838         break;
839     case 19:
840         state=max_sub_array(L);
841         printf("最大的连续子数组和为%d",state)
            ;
842         break;
843     case 20:
844         int K;
845         printf("请输入K的值:");
846         scanf("%d",&K);
847         printf("连续子数组和为%d的个数为%d",K,
            SubarrayK(K,L));
848         break;
849     case 0:
850         break;
851
852     }
853     Sleep(1000);
854 }
855 return 0;
856 }
```

B 附录 B 基于链式存储结构线性表实现的源程序

```
1      #include "stdio.h"
2      #include "stdlib.h"
3      #include <windows.h>
4      #define TRUE 1
5      #define FALSE 0
6      #define OK 1
7      #define ERROR 0
8      #define INFEASIBLE -1
9      #define OVERFLOW -2
10
11     typedef int status;
12     typedef int ElemType; //数据元素类型定义
13
14     #define LIST_INIT_SIZE 100
15     #define LISTINCREMENT 10
16     typedef int ElemType;
17     typedef struct LNode{ //单链表（链式结构）结点的定义
18         ElemType data;
19         struct LNode *next;
20     }LNode,*LinkList;
21
22     typedef struct { //多个线性表管理
23         struct {
24             char name[30];
25             LinkList L;
26         } elem[10];
27         int length;
28         int listsize;
29     }LISTS;
30     //函数一
31     status InitList(LinkList &L)
32     // 线性表L不存在，构造一个空的线性表，返回OK，否则返回
        INFEASIBLE。
```

```
33 {
34     // 请在这里补充代码，完成本关任务
35     /***** Begin *****/
36     if(!L){
37         L=(LinkedList)malloc(sizeof(LNode));
38         L->next=NULL;
39         return OK;
40     }
41     return INFEASIBLE;
42     /***** End *****/
43 }
44 // 函数二
45 status DestroyList(LinkedList &L)
46 // 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。
47 {
48     // 请在这里补充代码，完成本关任务
49     /***** Begin *****/
50     if(!L) return INFEASIBLE;
51     while(L->next!=NULL){
52         LNode *p =L->next;
53         L->next=L->next->next;
54         free(p);
55     }
56     free(L);
57     L=NULL;
58     return OK;
59     /***** End *****/
60 }
61 // 函数三
62 status ClearList(LinkedList &L)
63 // 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。
64 {
65     // 请在这里补充代码，完成本关任务
```

```
66      /****** Begin *****/
67      if(!L) return INFEASIBLE;
68      while(L->next!=NULL){
69          LNode *p =L->next;
70          L->next=L->next->next;
71          free(p);
72      }
73      return OK;
74      /****** End *****/
75  }
76  // 函数四
77  status ListEmpty(LinkList L)
78  // 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。
79  {
80      // 请在这里补充代码, 完成本关任务
81      /****** Begin *****/
82      if(!L) return INFEASIBLE;
83      if(L->next==NULL) return OK;
84      else return FALSE;
85      /****** End *****/
86  }
87  // 函数五
88  int ListLength(LinkList L)
89  // 如果线性表L存在, 返回线性表L的长度, 否则返回INFEASIBLE。
90  {
91      // 请在这里补充代码, 完成本关任务
92      /****** Begin *****/
93      if(!L) return INFEASIBLE;
94      int length=0;
95      LinkList current=L->next;
96      while(current!=NULL){
97          current=current->next;
98          length++;
99      }
```

```
100     return length;
101     /***** End *****/
102 }
103 //函数六
104 status GetElem(LinkList L,int i,ElemType &e)
105 // 如果线性表L存在，获取线性表L的第i个元素，保存在e中，返回OK
    ; 如果i不合法，返回ERROR；如果线性表L不存在，返回INFEASIBLE
    。
106 {
107     // 请在这里补充代码，完成本关任务
108     /***** Begin *****/
109     if(!L) return INFEASIBLE;
110     int cur_pos=1;
111     LinkList cur = L->next;
112     while( cur!=NULL){
113         if( cur_pos==i){
114             e=cur->data;
115             return OK;
116         }
117         cur_pos++;
118         cur=cur->next;
119     }
120     return ERROR;
121     /***** End *****/
122 }
123 //函数七
124 status LocateElem(LinkList L,ElemType e)
125 // 如果线性表L存在，查找元素e在线性表L中的位置序号；如果e不存在，
    返回ERROR；当线性表L不存在时，返回INFEASIBLE。
126 {
127     // 请在这里补充代码，完成本关任务
128     /***** Begin *****/
129     if(!L) return INFEASIBLE;
130     LinkList cur;
131     if(L->next)
```

```
132     cur=L->next;
133     else return ERROR;
134     int postion = 1;
135     while( cur!=NULL){
136         if( cur->data==e){
137             return postion;
138         }
139         cur=cur->next;
140         postion++;
141     }
142     return ERROR;
143     /***** End *****/
144 }
145 // 函数八
146 status PriorElem( LinkList L, ElemType e, ElemType &pre )
147 // 如果线性表L存在，获取线性表L中元素e的前驱，保存在pre中，返回OK；如果没有前驱，返回ERROR；如果线性表L不存在，返回INFEASIBLE。
148 {
149     // 请在这里补充代码，完成本关任务
150     /***** Begin *****/
151     if(!L) return INFEASIBLE;
152     if(L->next)
153         L=L->next;
154     else return ERROR;
155     while( L->next!=NULL){
156         if( L->next->data==e){
157             pre = L->data;
158             return OK;
159         }
160         L=L->next;
161     }
162     return ERROR;
163     /***** End *****/
164 }
```

```
165 //函数九
166 status NextElem(LinkList L,ElemType e,ElemType &next)
167 // 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。
168 {
169     // 请在这里补充代码, 完成本关任务
170     /***** Begin *****/
171     if(!L)return INFEASIBLE;
172     if(L->next)
173         L=L->next;
174     else return ERROR;
175     while(L->next!=NULL){
176         if(L->data==e){
177             next = L->next->data;
178             return OK;
179         }
180         L=L->next;
181     }
182     return ERROR;
183     /***** End *****/
184 }
185 //函数十
186 status ListInsert(LinkList &L,int i,ElemType e)
187 // 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。
188 {
189     // 请在这里补充代码, 完成本关任务
190     /***** Begin *****/
191     if(!L)return INFEASIBLE;
192     LinkList cur=L->next;
193     int postion=1;
194     if(i==1){
195         LinkList NewNode = (LinkList)malloc(sizeof(LNode));
```



```
196         NewNode->data=e;
197         NewNode->next=L->next;
198         L->next=NewNode;
199         return OK;
200     }
201     while (cur!=NULL) {
202         if (postion==i-1) {
203             LinkList NewNode = (LinkList) malloc (sizeof(LNode))
                ;
204             NewNode->data=e;
205             NewNode->next=cur->next;
206             cur->next=NewNode;
207             return OK;
208         }
209         postion++;
210         cur=cur->next;
211     }
212     return ERROR;
213     ***** End *****
214 }
215 // 函数十一
216 status ListDelete (LinkList &L, int i, ElemType &e)
217 // 如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回
OK；当删除位置不正确时，返回ERROR；如果线性表L不存在，返回
INFEASIBLE。
218 {
219     // 请在这里补充代码，完成本关任务
220     ***** Begin *****
221     if (!L) return INFEASIBLE;
222     LinkList cur=L->next;
223     LinkList pre=L;
224     int postion=1;
225     if (i==1) {
226         e=L->next->data;
227         LinkList node = L->next;
```

```
228         L->next=L->next->next;
229         free(node);
230         return OK;
231     }
232     while(cur!=NULL){
233         if( postion==i-1&&cur->next!=NULL){
234             e=cur->next->data;
235             LinkList node = cur->next;
236             cur->next=cur->next->next;
237             free(node);
238             return OK;
239         }
240         postion++;
241         cur=cur->next;
242     }
243     return ERROR;
244     /***** End *****/
245 }
246 //函数十二
247 status ListTraverse(LinkList L)
248 // 如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，
    返回OK；如果线性表L不存在，返回INFEASIBLE。
249 {
250     // 请在这里补充代码，完成本关任务
251     /***** Begin *****/
252     if(!L) return INFEASIBLE;
253     LinkList cur = L->next;
254     while(cur!=NULL){
255         printf("%d ",cur->data);
256         cur=cur->next;
257     }
258     return OK;
259     /***** End *****/
260 }
261 //函数十三
```

```
262 status reverseList(LinkList &L){
263     if(!L) return INFEASIBLE;
264     LinkList pre=NULL, cur=L->next;
265     while( cur!=NULL){
266         LinkList next=cur->next;
267         cur->next=pre;
268         pre=cur;
269         cur=next;
270     }
271     L->next=pre;
272     return OK;
273 }
274 // 函数十四
275 status RemoveNthFromEnd(LinkList &L, int i){
276     int length=ListLength(L);
277     int postion=1;
278     i=length-i+1;
279     if(!L) return INFEASIBLE;
280     LinkList cur=L->next;
281     if(i==1){
282         LinkList node = L->next;
283         L->next=L->next->next;
284         free(node);
285         return OK;
286     }
287     while( cur!=NULL){
288         if(postion==i-1&&cur->next!=NULL){
289             LinkList node = cur->next;
290             cur->next=cur->next->next;
291             free(node);
292             return OK;
293         }
294         postion++;
295         cur=cur->next;
296     }
```

```
297     return ERROR;
298 }
299 //函数十五
300 //排序
301 status sortList(LinkList &L){
302     if(!L) return INFEASIBLE;
303     LNode *cur = L->next;
304     while (cur != NULL) {
305         LNode *minNode=cur;
306         LNode *temp=cur->next;
307         // 交换当前节点和最小节点的值
308         while (temp!=NULL) {
309             if (temp->data < minNode->data) {
310                 minNode=temp;
311             }
312             temp=temp->next;
313         }
314         int tmp=minNode->data;
315         cur->data=minNode->data;
316         minNode->data=tmp;
317     }
318     return OK;
319 }
320
321 //函数十六           以文件的形式保存
322 status SaveList(LinkList L, const char FileName[])
323 // 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。
324 {
325     // 请在这里补充代码，完成本关任务
326     /***** Begin I *****/
327     if (L == NULL) {
328         return INFEASIBLE;
329     }
330     FILE* stream;
```

```
331     stream = fopen(FileName, "w");
332     LinkList temp = L->next;
333     while (temp){
334         fprintf(stream, "%d ", temp->data);
335         temp = temp->next;
336     }
337     fprintf(stream, "%d", 0);
338     fclose(stream);
339     return OK;
340
341     /****** End 1 *****/
342 }
343
344 //函数十七 将文件的数据导出
345 status LoadList(LinkList &L, const char FileName[])
346 // 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。
347 {
348     // 请在这里补充代码, 完成本关任务
349     /****** Begin 2 *****/
350     // if (L) return INFEASIBLE;
351     L = (LinkList)malloc(sizeof(LNode));
352     FILE *fp = fopen(FileName, "r");
353     LinkList p = L;
354     int e;
355     fscanf(fp, "%d", &e);
356     while (e) {
357         p->next = (LinkList)malloc(sizeof(LNode));
358         p = p->next;
359         p->data = e;
360         fscanf(fp, "%d", &e);
361     }
362     p->next = NULL;
363     fclose(fp);
364     return OK;
```

```
365      /***** End 2 *****/
366  }
367
368  //函数十八 add
369  //附加功能18 多线性表管理：增加新线性表
370  status AddList(LISTS &Lists, char ListName[])
371  // 在Lists中增加一个名称为ListName的线性表。
372  {
373      for(int i=0; i<Lists.length; i++)
374          if (strcmp(Lists.elem[i].name, ListName) == 0)
375          {
376              printf("多线性表中已存在名称为 %s 的线性表! \n",
377                  ListName);
378              return ERROR;
379          }
380          strcpy(Lists.elem[Lists.length].name, ListName);
381          //接下来对新线性表初始化
382          Lists.elem[Lists.length].L=NULL;
383          InitList(Lists.elem[Lists.length].L);
384          Lists.length++;
385          return OK;
386  }
387  //函数十九 remove
388  //附加功能19 多线性表管理：删除线性表
389  status RemoveList(LISTS &Lists, char ListName[])
390  // Lists中删除一个名称为ListName的线性表
391  {
392      for(int i = 0; i < Lists.length; ++i){
393          if(!strcmp(ListName, Lists.elem[i].name)){
394              for(int j = i; j < Lists.length - 1; ++j){
395                  Lists.elem[j] = Lists.elem[j + 1];
396              }
397              Lists.length--;
398              return OK;
399          }
400      }
```

```
399     }
400     return ERROR;
401 }
402 //函数二十 Locate
403 //附加功能20 多线性表管理：查找线性表
404 int LocateList(LISTS Lists, char ListName[])
405 // 在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，
    否则返回0
406 {
407     for(int i = 0; i < Lists.length; ++i){
408         if(!strcmp(ListName, Lists.elem[i].name))
409             return i + 1;
410     }
411     return 0;
412 }
413
414 //附加功能21 多线性表管理：遍历多线性表
415 status ListsTraverse(LISTS Lists)
416 {
417     if (Lists.length == 0)
418         return ERROR;
419     for (int num = 0; num < Lists.length; num++)
420     {
421         printf("当前表名: %s ", Lists.elem[num].name);
422         ListTraverse(Lists.elem[num].L);
423         putchar('\n');
424     }
425     return OK;
426 }
427
428 //附加功能22 多线性表管理：选择一个线性表，输入其序号，后续可
    对其进行操作
429 LinkList switchList(LinkList L, LISTS &Lists, int i)
430 {
431     if (i > Lists.length || i < 1)
```

```
432         return NULL;
433     else
434     {
435         L = Lists.elem[i - 1].L;  //传递地址以改动表
436         L->next = Lists.elem[i - 1].L->next;
437         L->data = Lists.elem[i - 1].L->data;
438         return L;
439     }
440 }
441
442 int main() {
443     int op=1;
444     int state;
445     int e,i;
446     LISTS Lists;
447     Lists.length=0,Lists.listsize=10;
448     LinkList L=NULL;
449     while (op)
450     {
451         system("cls");
452         printf("\n\n");
453         printf("          Menu for Linear Table On
454                Sequence Structure \n");
455         printf("-----\n");
456         printf("          1. InitList          10.
457                ListInsert\n");
458         printf("          2. DestroyList       11.
459                ListDelete\n");
460         printf("          3. ClearList         12.
461                ListTraverse\n");
462         printf("          4. ListEmpty         13.
463                reverseList\n");
464         printf("          5. ListLength        14.
```



```

        RemoveNthFromEnd\n");
460     printf("          6. GetElem          15.
        sortList\n");
461     printf("          7. LocateElem        16.
        SaveList\n");
462     printf("          8. PriorElem         17.
        LoadList\n");
463     printf("          9. NextElem          18. AddList
        \n");
464     printf("         19. RemoveList        20.
        LocateList\n");
465     printf("        21. ListsTraverse 22.
        SwitchList\n");
466     printf("          0. exit\n");
467     printf("
        -----|
        n");
468     printf("      请选择你的操作[0~20]:\n");
469     scanf("%d", &op);
470     switch (op){
471     case 1:
472         state=InitList(L);
473         if( state==OK){
474             printf("线性表创建成功");
475         } else if( state==INFEASIBLE){
476             printf("线性表创建失败");
477         }
478         break;
479
480     case 2:
481         state=DestroyList(L);
482         if( state==OK) printf("线性表销毁成功");
483         else if( state==INFEASIBLE) printf("线性表不存在
        ");
484         break;

```

```
485
486         case 3:
487             state=ClearList(L);
488             if( state==OK) printf("线性表清空成功");
489             else if( state==INFEASIBLE) printf("线性表不存在");
490             break;
491
492         case 4:
493             state=ListEmpty(L);
494             if( state==OK) printf("线性表为空");
495             else if( state==FALSE) printf("线性表不为空");
496             else if( state==INFEASIBLE) printf("线性表不存在");
497             break;
498
499         case 5:
500             state=ListLength(L);
501             if( state==INFEASIBLE) printf("线性表不存在");
502             else {
503                 printf("线性表的长度是%d",state);
504             }
505             break;
506
507         case 6:
508             printf("请输入你想要获取的位置:");
509             scanf("%d",&i);
510             state=GetElem(L,i,e);
511             if( state==INFEASIBLE) printf("线性表不存在");
512             else if( state==ERROR) printf("i 不合法");
513             else if( state==OK) {
514                 printf("%d号位置的元素是%d",i,e);
515             }
516             break;
517
```

```
518         case 7:
519             printf("请输入你要查找的元素:");
520             scanf("%d",&e);
521             state=LocateElem(L,e);
522             if( state==INFEASIBLE) printf("线性表不存在");
523             else if( state==ERROR) printf("%d不存在",e);
524             else printf("%d在表中的位置是%d",e, state);
525             break;
526
527         case 8:
528             int pre;
529             printf("请输入你要查找的元素的前驱结点:");
530             scanf("%d",&e);
531             state=PriorElem(L,e,pre);
532             if( state==INFEASIBLE) printf("线性表不存在");
533             else if( state==OK) printf("%d的前驱结点元素是%d",e,pre);
534             else if( state==ERROR) printf("该节点没有前驱");
535             break;
536
537         case 9:
538             int next;
539             printf("请输入你要查找的元素的后驱结点:");
540             scanf("%d",&e);
541             state=NextElem(L,e,next);
542             if( state==INFEASIBLE) printf("线性表不存在");
543             else if( state==OK) printf("%d的后驱结点元素是%d",e,next);
544             else if( state==ERROR) printf("该节点没有后驱");
545             break;
546
547         case 10:
548             printf("请输入你想要插入的位置和元素:");
549             scanf("%d%d",&i,&e);
550             state=ListInsert(L,i,e);
```

```
551         if( state==INFEASIBLE) printf("线性表不存在");
552         else if( state==OK) printf("插入成功");
553         else if( state==ERROR) printf("插入位置有问题");
554         break;
555
556     case 11:
557         printf("请输入想删除的位置:");
558         scanf("%d",&i);
559         state=ListDelete(L,i,e);
560         if( state==INFEASIBLE) printf("线性表不存在");
561         else if( state==OK) printf("删除成功");
562         else if( state==ERROR) printf("删除位置有问题");
563         break;
564
565     case 12:
566         state=ListTraverse(L);
567         if( state==INFEASIBLE) printf("线性表不存在");
568         break;
569
570     case 13:
571         state=reverseList(L);
572         if( state==INFEASIBLE) printf("线性表不存在");
573         else if( state==OK) printf("翻转成功");
574         break;
575
576     case 14:
577         printf("请输入倒数第几个: ");
578         scanf("%d",&i);
579         state=RemoveNthFromEnd(L,i);
580         if( state==INFEASIBLE) printf("线性表不存在");
581         else if( state ==OK) printf("删除成功");
582         break;
583
584     case 15:
585         state=sortList(L);
```

华中科技大学课程实验报告

```
586         if( state==INFEASIBLE) printf("线性表不存在");
587         else if( state ==OK) printf("排序成功");
588         break;
589
590     case 16:
591         state = SaveList(L, "D:\\tounge.txt");
592         if ( state == -1){
593             printf("线性表不存在。\\n")
594             ;
595         }
596         else {
597             printf("线性表已经成功复制
598             在路径为%s的文件中。\\n"
599             , "D:\\tounge.txt");
600         }
601         break;
602
603     case 17:
604         state=LoadList(L, "D:\\tounge.txt");
605         printf("已成功读入。\\n");
606         break;
607
608     case 18:
609         char ListName[100];
610         printf("请输入你要添加的线性表的名字: ");
611         scanf("%s", ListName);
612         state = AddList(Lists, ListName);
613         if ( state == OK)
614             printf("成功新增名称为 %s 的线性表! \\n",
615             ListName);
616         else
617             printf("新增失败! \\n");
618         break;
619
620     case 19:
```

```
617         if(!Lists.length)
618         {
619             printf("线性表的集合为空! 无法进行此操作!
620                 \n");
621             break;
622         }
623         printf("请输入想要删除的线性表的名称: ");
624         scanf("%s", ListName);
625         state = RemoveList(Lists, ListName);
626         if (state == OK)
627             printf("成功删除名称为 %s 的线性表! \n",
628                 ListName);
629         else if (state == ERROR)
630             printf("删除失败! \n");
631         break;
632
633 case 20:
634         if(!Lists.length)
635         {
636             printf("线性表的集合为空! 无法进行此操作!
637                 \n");
638             break;
639         }
640         printf("请输入想要查找的线性表的名称: ");
641         scanf("%s", ListName);
642         i = LocateList(Lists, ListName);
643         if (i == 0)
644             printf("不存在名称为 %s 的线性表! \n",
645                 ListName);
646         else printf("名称为 %s 的线性表的序号为 %d ! \n", ListName, i);
647         break;
648
649 case 21:
650         state = ListsTraverse(Lists);
```

```
647         if (state== ERROR)
648             printf("线性表的集合为空! \n");
649         break;
650
651     case 22:
652         if(Lists.length)
653         {
654             printf("\n----- All lists
655             -----\n");
656             for(int i = 0; i < Lists.length; i++)
657                 printf("%d %s\n", i + 1, Lists.elem[i
658                     ].name);
659             printf("----- end
660             -----\n");
661         }
662     else
663     {
664         printf("线性表的集合为空! 无法进行此操作!
665             \n");
666         getchar();getchar();
667         break;
668     }
669     printf("请输入想要进行操作的线性表的序号 (从1
670         开始) : ");
671     scanf("%d", &i);
672     L = switchList(L, Lists , i);
673     if (L == NULL)
674         printf("输入的序号不合法! 单表已置空! \n")
675         ;
676     else
677         printf("下面可对线性表集合中序号为 %d 的线
678             性表进行操作! \n", i);
679     break;
680
681 case 0:
```

```
675             op=0;
676             break;
677
678         }
679         Sleep(1000);
680     }
681 }
```

C 附录 C 基于二叉链表二叉树实现的源程序

```
1      #ifndef DEF_H_INCLUDED
2      #define DEF_H_INCLUDED
3
4      #include "stdio.h"
5      #include "stdlib.h"
6
7      #define TRUE 1
8      #define FALSE 0
9      #define OK 1
10     #define ERROR 0
11     #define INFEASIBLE -1
12     #define OVERFLOW -2
13
14     typedef int status;
15     typedef int KeyType;
16     typedef struct {
17         KeyType key;
18         char others[20];
19     } TElemType; //二叉树结点类型定义
20
21
22     typedef struct BiTNode{ //二叉链表结点的定义
23         TElemType data;
24         struct BiTNode *lchild,*rchild;
25     } BiTNode, *BiTree;
26
27     typedef struct Trees{
28         struct {
29             char name[30];
30             BiTNode* T;
31         } elem[10];
32         int length;
33         int listsize;
```

```
34 } Trees;
35 #endif
36 #ifndef OPT_H_INCLUDED
37 #define OPT_H_INCLUDED
38
39 #include <cstring>
40 #include <set>
41 #include <queue>
42 #include <stdio.h>
43 #include <string.h>
44 #include "def.h"
45 #define INF 0x3f3f3f3f
46
47 using namespace std;
48
49 set<int> s;
50
51 int n = 0, k = 0;
52
53 void visit(BiTree T)
54 {
55     printf(" %d,%s", T->data.key, T->data.others);
56 }
57
58 bool isDuplicate(TElemType definition[]) {
59     int Hash[1000] = {0};
60     for(int i = 0; definition[i].key != -1; ++i) {
61         Hash[definition[i].key]++;
62     }
63     for(int j = 1; j < 10; ++j) {
64         if(Hash[j] > 1) return TRUE; //表示有重复的
65     }
66     return FALSE;
67 }
68
```

```
69 status CreateBiTree(BiTree &T,TElemType definition[])
70 /*根据带空枝的二叉树先根遍历序列 definition 构造一棵二叉树，将根
   节点指针赋值给 T 并返回 OK，
71 如果有相同的关键字，返回 ERROR。此题允许通过增加其它函数辅助实
   现本关任务*/
72 {
73     // 请在这里补充代码，完成本关任务
74     /****** Begin *****/
75     if(isDuplicate(definition))return ERROR; // 重复元素
76     static int index=0;
77     if(definition==nullptr || definition[index].key==0){
78         T=nullptr;
79         return OK;
80     }
81     T=(BiTree) malloc(sizeof(BiTNode));
82     T->data.key=definition[index].key;
83     strcpy(T->data.others,definition[index].others);
84     // 左
85     index++;
86     CreateBiTree(T->lchild,definition);
87     // 右
88     index++;
89     CreateBiTree(T->rchild,definition);
90     return OK;
91     /****** End *****/
92 }
93 // 不太确定的功能
94 status DestroyBiTree(BiTree &T){
95     if(!T)return ERROR;
96     free(T);
97     return OK;
98 }
99
100 status ClearBiTree(BiTree &T){
101     // 将二叉树设置成空，并删除所有结点，释放结点空间
```

```
102     if(T==nullptr) return OK;
103     ClearBiTree(T->lchild);
104     ClearBiTree(T->rchild);
105     free(T);
106     T=nullptr;
107     return OK;
108 }
109
110 int BiTreeDepth(BiTree T){
111     //求二叉树的深度
112     if(T==nullptr) return 0;
113     int dep1=0,dep2=0;
114     dep1=BiTreeDepth(T->lchild);
115     dep2=BiTreeDepth(T->rchild);
116     return dep1>dep2? dep1+1: dep2+1;
117 }
118
119 BiTNode* LocateNode(BiTree T,KeyType e)
120 //查找结点
121 {
122     // 请在这里补充代码，完成本关任务
123     /***** Begin *****/
124     if(T==nullptr || T->data.key==e) return T;
125     BiTNode *lnode = LocateNode(T->lchild,e);
126     if(lnode) return lnode;
127     BiTNode *rnode =LocateNode(T->rchild,e);
128     return rnode;
129 }
130     /***** End *****/
131
132 status Assign(BiTree &T,KeyType e,TElemType value)
133 //实现结点赋值。此题允许通过增加其它函数辅助实现本关任务
134 {
135     // 请在这里补充代码，完成本关任务
136     /***** Begin *****/
```

```
137     BiTNode* node=LocateNode(T, e);
138     if (node!= NULL) {
139         if (LocateNode(T, value.key)!=NULL&&LocateNode(T, value.
            key)!=node) {
140             return ERROR;
141         }
142         node->data = value;
143         return OK;
144     }
145     return ERROR;
146     ***** End *****
147 }
148
149 BiTNode* GetSibling(BiTree T,KeyType e)
150 //实现获得兄弟结点
151 {
152     // 请在这里补充代码，完成本关任务
153     ***** Begin *****
154     if (T==nullptr) {
155         return nullptr; // 如果当前节点为空，直接返回 nullptr
156     }
157     if (T->lchild!=nullptr&&T->lchild->data.key==e&&T->rchild
        != nullptr) {
158         return T->rchild; // 如果目标节点是左子树节点且右子树
            存在，则返回右子树节点
159     }
160     if (T->rchild!=nullptr&&T->rchild->data.key==e&&T->lchild
        != nullptr) {
161         return T->lchild; // 如果目标节点是右子树节点且左子树
            存在，则返回左子树节点
162     }
163     BiTNode* leftSibling=GetSibling(T->lchild, e);
164     if (leftSibling != nullptr) {
165         return leftSibling; //如果在左子树中找到兄弟节点，直接
            返回结果
```

```
166     }
167     BiTNode* rightSibling=GetSibling(T->rchild , e);
168     if (rightSibling!=nullptr) {
169         return rightSibling; // 如果在右子树中找到兄弟节点，直接返回结果
170     }
171     return nullptr;
172     /***** End *****/
173 }
174
175 bool BiTreeEmpty(BiTree T){
176     if(T==nullptr) return true;
177     return false;
178 }
179
180 void insert(BiTNode*T, int LR, BiTNode*newNode) {
181     if(LR==0){
182         if(T->lchild==NULL){
183             T->lchild=newNode;
184         }
185         else {
186             newNode->rchild=T->lchild;
187             T->lchild=newNode;
188         }
189     }
190     if(LR==1){
191         if(T->rchild==NULL){
192             T->rchild=newNode;
193         }
194         else {
195             newNode->rchild=T->rchild;
196             T->rchild=newNode;
197         }
198     }
199 }
```

```
200 status InsertNode(BiTree &T,KeyType e,int LR,TElemType c)
201 //插入结点。此题允许通过增加其它函数辅助实现本关任务
202 {
203     // 请在这里补充代码，完成本关任务
204     /***** Begin *****/
205     BiTree newNode = (BiTree)malloc(sizeof(BiTNode));
206     newNode->data=c;
207     newNode->lchild=NULL;
208     newNode->rchild=NULL;
209     if(LR==-1){
210         newNode->rchild=T;
211         T=newNode;
212         return OK;
213     }
214     BiTNode* node=LocateNode(T,e); //找到目标的节点
215     if(node==NULL) return ERROR; //没有目标的节点
216     if(LocateNode(T,c.key)!=NULL) return ERROR; //关键词重复
217     insert(node,LR,newNode);
218     return OK;
219     /***** End *****/
220 }
221
222 status DeleteNode(BiTree &T,KeyType e){
223     //删除结点。此题允许通过增加其它函数辅助实现本关任务
224     if(!T) return ERROR;
225
226     int flag=0;
227     if(T->data.key==e) flag=1;
228     if(flag){
229         int deg=0;
230         if(T->lchild) deg+=1;
231         if(T->rchild) deg+=2;
232         if(deg==0){
233             free(T);
234             T=NULLptr;
```

```
235     }
236     if (deg==1){
237         BiTNode*tmp=T;
238         T=T->lchild;
239         free(tmp);
240         tmp=NULLPTR;
241     }
242     if (deg==2){
243         BiTree tmp=T;
244         T=T->rchild;
245         free(tmp);
246         tmp=NULLPTR;
247     }
248     if (deg==3){
249         BiTree tmp=T;
250         BiTree LC=T->lchild;
251         BiTree RC=T->rchild;
252         T=LC;
253         while(LC->rchild){
254             LC=LC->rchild;
255         }
256         LC->rchild=RC;
257         free(tmp);
258         tmp=NULLPTR;
259     }
260     return OK;
261 }
262 else{
263     status state1, state2;
264     state1=DeleteNode(T->lchild, e);
265     state2=DeleteNode(T->rchild, e);
266     if (state1==OK || state2==OK) return OK;
267     return ERROR;
268 }
269 }
```



```
270
271 typedef struct _stack {
272     BiTNode* stack[100];
273     int top;
274 } Stack;
275 int isEmpty(Stack s) {
276     return s.top==0? 1:0;
277 }
278 BiTNode* Pop(Stack *s) {
279     return s->stack[--(s->top)];
280 }
281 void push(Stack *s, BiTNode* node) {
282     s->stack[s->top++]=node;
283 }
284 status PreOrderTraverse(BiTree T, void (*visit)(BiTree))
285 //先序遍历二叉树T
286 {
287     // 请在这里补充代码，完成本关任务
288     /***** Begin *****/
289     Stack nodes;
290     nodes.top=0;
291     push(&nodes, T);
292     while(!isEmpty(nodes)) {
293         BiTNode* current=Pop(&nodes);
294         if(current!=NULL) {
295             visit(current);
296             push(&nodes, current->rchild);
297             push(&nodes, current->lchild);
298         }
299     }
300     return OK;
301     /***** End *****/
302 }
303
304 status InOrderTraverse(BiTree T, void (*visit)(BiTree))
```

```
305 //中序遍历二叉树T
306 {
307     // 请在这里补充代码，完成本关任务
308     /***** Begin *****/
309     Stack nodes;
310     nodes.top=0;
311     BiTNode *node=T;
312     while(!isEmpty(nodes)||node!=NULL){
313         while(node!=NULL){
314             push(&nodes,node);
315             node=node->lchild;
316         }
317         node=Pop(&nodes);
318         visit(node);
319         node=node->rchild;
320     }
321     return OK;
322     /***** End *****/
323 }
324
325 status PostOrderTraverse(BiTree T,void (*visit)(BiTree))
326 //后序遍历二叉树T
327 {
328     // 请在这里补充代码，完成本关任务
329     /***** Begin *****/
330     if(T==nullptr) return OK;
331     PostOrderTraverse(T->lchild,visit);
332     PostOrderTraverse(T->rchild,visit);
333     visit(T);
334     /***** End *****/
335 }
336
337 status LevelOrderTraverse(BiTree T,void (*visit)(BiTree))
338 //按层遍历二叉树T
339 {
```

```
340 // 请在这里补充代码，完成本关任务
341 /***** Begin *****/
342 BiTNode* Queue[100];
343 int rear, front;
344 rear=front=0;
345 Queue[rear++]=T;
346 while( front<rear){
347     BiTNode* node=Queue[front++];
348     visit(node);
349     if(node->lchild!=NULL){
350         Queue[rear++]=node->lchild;
351     } if(node->rchild!=NULL){
352         Queue[rear++]=node->rchild;
353     }
354 }
355 return OK;
356 /***** End *****/
357 }
358
359 typedef struct {
360     int pos;
361     TElemType data;
362 } HE;
363
364 int get_index(BiTree T,BiTree child,int idx)
365 {
366     if(T==NULL) return 0;
367     if(T->data.key==child->data.key){
368         return idx;
369     }
370     int gll = get_index(T->lchild,child,idx*2);
371     int grr = get_index(T->rchild,child,idx*2+1);
372     if(grr>gll) return grr;
373     return gll;
374 }
```

```
375
376 status CreateBiTree1(BiTree &T, HE definition[])
377 {
378     int i=0, j;
379     static BiTNode *st[100];
380     while (j=definition[i].pos)
381     {
382         st[j]=(BiTNode *)malloc(sizeof(BiTNode));
383         st[j]->data=definition[i].data;
384         st[j]->lchild=NULL;
385         st[j]->rchild=NULL;
386         if (j!=1)
387             if (j%2) st[j/2]->rchild=st[j];
388             else st[j/2]->lchild=st[j];
389         i++;
390     }
391     T=st[1];
392     return OK;
393 }
394
395 int MaxPathSum(BiTree T){
396     if(!T) return ERROR;
397     if(!T->lchild&&!T->rchild) return T->data.key;
398     int ll, rr;
399     ll=(T->lchild?MaxPathSum(T->lchild):-INF);
400     rr=(T->rchild?MaxPathSum(T->rchild):-INF);
401     return std::max(ll, rr)+T->data.key;
402 }
403
404 BiTree LowestCommonAncestor(BiTree T, KeyType e1, KeyType e2){
405     if(T->data.key==e1 || T->data.key==e2 || T==nullptr) return T;
406     BiTree left=LowestCommonAncestor(T->lchild, e1, e2);
407     BiTree right=LowestCommonAncestor(T->rchild, e1, e2);
408     if (left && right) return T;
409     else if (!left) return right;
```

```
410     else if (!right) return left;
411     else return T;
412 }
413
414 void InvertTree(BiTree &T){
415     if(T->lchild) InvertTree(T->lchild);
416     if(T->rchild) InvertTree(T->rchild);
417     BiTree tmp=T->lchild;
418     T->lchild=T->rchild;
419     T->rchild=tmp;
420 }
421
422 status SaveBiTree(BiTree T, char FileName[])
423 //将二叉树的结点数据写入到文件FileName中
424 {
425     // 请在这里补充代码，完成本关任务
426     /***** Begin I *****/
427     if(T!=NULL)
428     {
429         FILE *fp;
430         int index=1;
431         fp=fopen(FileName, "w");
432         queue<BiTree> Q;
433         Q.push(T);
434         BiTree cur;
435         while(Q.size()){
436             cur=Q.front();
437             Q.pop();
438             if(cur!=NULL){
439                 fprintf(fp, "%d %d %s ", get_index(T, cur, 1), cur
                        ->data.key, cur->data.others);
440             }
441             index++;
442             if(cur->lchild!=NULL) Q.push(cur->lchild);
443             if(cur->rchild!=NULL) Q.push(cur->rchild);
```

```
444     }
445     fprintf(fp, "0 0 null");
446     fclose(fp);
447 }
448 return OK;
449 }
450
451 status LoadBiTree(BiTree &T, char FileName[])
452 // 读入文件FileName的结点数据, 创建二叉树
453 {
454     int i=0;
455     HE definition[100];
456     FILE *fp;
457     fp=fopen(FileName, "r");
458     do {
459         fscanf(fp, "%d %d %s", &definition[i].pos, &definition[i].
            data.key, definition[i].data.others);
460     } while (definition[i++].pos);
461
462     CreateBiTree1(T, definition);
463     fclose(fp);
464     return OK;
465 }
466
467 status AddTree(Trees &trees, char ListName[]) {
468     for(int i=0; i<trees.length; i++)
469         if (strcmp(trees.elem[i].name, ListName) == 0)
470             {
471                 printf("多线性表中已存在名称为 %s 的线性表! \n",
                    ListName);
472                 return ERROR;
473             }
474     strcpy(trees.elem[trees.length].name, ListName);
475     trees.elem[trees.length].T=NULLPTR;
476     trees.elem[trees.length].T=(BiTNode*)malloc(sizeof(BiTNode
```

```
        ));
477     trees.length++;
478     return OK;
479 }
480
481 status LocateTree(Trees &trees, char ListName[]) {
482     for(int i=0; i<trees.length; ++i) {
483         if(strcmp(trees.elem[i].name, ListName)==0) {
484             return 1+i;
485         }
486     }
487     return 0;
488 }
489
490 status RemoveTree(Trees &trees, char ListName[]) {
491     for(int i = 0; i < trees.length; ++i) {
492         if(!strcmp(ListName, trees.elem[i].name)) {
493             for(int j = i; j < trees.length - 1; ++j) {
494                 trees.elem[j] = trees.elem[j + 1];
495             }
496             trees.length--;
497             return OK;
498         }
499     }
500     return ERROR;
501 }
502
503 BiTree SwitchTree(Trees &trees, int i) {
504     if (i > trees.length || i < 1)
505         return NULL;
506     else
507     {
508         return trees.elem[i-1].T;
509     }
510 }
```

```
511
512 BiTNode *findLCA(BiTNode *root, int n1, int n2){
513     if (root == nullptr) return nullptr;
514
515     if (root->data.key == n1 || root->data.key == n2)
516         return root;
517
518     BiTNode* leftLCA = findLCA(root->lchild, n1, n2);
519     BiTNode* rightLCA = findLCA(root->rchild, n1, n2);
520
521     // 如果在左子树和右子树中分别找到了n1和n2
522     if (leftLCA && rightLCA) return root;
523
524     // 只有左子树或只有右子树中有n1和n2
525     return (leftLCA != nullptr) ? leftLCA : rightLCA;
526 }
527
528 #endif
529 #include <iostream>
530 #include <sstream>
531 #include <cstdio>
532 #include <cstdlib>
533 #include <cstring>
534 #include <string>
535 #include <windows.h>
536 #include "def.h"
537 #include "opt.h"
538
539 using namespace std;
540 BiTree T;
541 Trees trees;
542
543 int main(){
544     int op=1;
545     int state;
```



```

546     trees.length=0,trees.listsize=10;
547     T=NULL;
548     TElemType value;
549     BiTNode *node;
550     int e,LR,i;
551     char str[100];
552     while(op){
553         system("cls");
554         printf("\n\n");
555         printf("          Menu for Linear Table On
          Sequence Structure \n");
556         printf("
          -----|
          n");
557         printf("          1. 创建二叉树          10. 删除
          节点\n");
558         printf("          2. 销毁二叉树          11. 前序
          遍历\n");
559         printf("          3. 清空二叉树          12. 中序
          遍历\n");
560         printf("          4. 判空二叉树          13. 后序
          遍历\n");
561         printf("          5. 二叉树深度          14. 按层
          遍历\n");
562         printf("          6. 查找节点          15. 最大
          路径和\n");
563         printf("          7. 节点赋值          16. 最近
          公共祖先\n");
564         printf("          8. 兄弟节点          17. 翻转
          二叉树\n");
565         printf("          9. 插入节点          18. 文件
          保存\n");
566         printf("          19.文件读取          20. 添加
          树\n");
567         printf("          21.查找树          23. 删除

```

华中科技大学课程实验报告

```
树\n"); //1 a 2 b 0 null 0 null 3 c 4 d 0
null 0 null 5 e 0 null 0 null -1 null
568 printf("                22.选中树                0.exit\n");
569 printf("
-----|
n");
570 printf("    请选择你的操作[0~20]:\n");
571 scanf("%d", &op);
572 switch (op)
573 {
574 case 1:
575     i=0;
576     printf("请依次输入关键词和名字(以-1结尾)\n");
577     TElemType definitions[100];
578     do{
579         scanf("%d%s",&definitions[i].key,definitions[i]
            ].others);
580     }while(definitions[i++].key!=-1);
581     state=CreateBiTree(T,definitions);
582     if(state==OK) printf("创建成功\n");
583     else if(state==ERROR) printf("创建失败\n");
584     break;
585
586 case 2:
587     state=DestroyBiTree(T);
588     if(state==OK) printf("销毁成功\n");
589     else if(state==ERROR) printf("销毁失败, 树为空\n");
590     break;
591
592 case 3:
593     state=ClearBiTree(T);
594     if(state==OK) printf("清空成功!\n");
595     else if(state==ERROR) printf("失败, 树为空\n");
596     break;
597
```

```
598         case 4:
599             state=BiTreeDepth(T);
600             if( state==0)printf( "树为空\n");
601             else printf( "树非空");
602             break;
603
604         case 5:
605             state=BiTreeDepth(T);
606             if( state==0)printf( "树为空");
607             else printf( "该二叉树的深度为:%d\n",state);
608             break;
609
610         case 6:
611             printf( "请输入该结点关键词: ");
612             scanf( "%d",&e);
613             node=LocateNode(T,e);
614             if(node==NULL)printf( "该节点不存在");
615             else printf( "节点名: %s\n节点数值: %d\n",node->
                    data.others ,node->data.key);
616             break;
617
618         case 7:
619             printf( "请输入该节点的关键词: ");
620             scanf( "%d",&e);
621             printf( "请输入该节点之新关键词和新名称:");
622             scanf( "%d%s",&value.key ,value.others);
623             state=Assign(T,e,value);
624             if( state==ERROR)printf( "未找到该节点");
625             if( state==OK)printf( "更行成功");
626             break;
627
628         case 8:
629             printf( "请输入节点的关键词:");
630             scanf( "%d",&e);
631             node=GetSibling(T,e);
```

```
632         if(node==NULL) printf("不存在兄弟节点");
633     else
634         printf("兄弟节点的名: %s, 兄弟节点数值: %d\n",
                node->data.others, node->data.key);
635     break;
636
637     case 9:
638         printf("请输入节点的关键词: ");
639         scanf("%d",&e);
640         printf("请输入LR值: ");
641         scanf("%d",&LR);
642         printf("请输入插入节点的关键词和名字: ");
643         scanf("%d%s",&value.key, value.others);
644         state=InsertNode(T,e,LR,value);
645         if(state==ERROR) printf("更新失败(关键词重复或未找到该节点)");
646         if(state==OK) printf("更新成功");
647         break;
648
649     case 10:
650         printf("请输入节点的关键词: ");
651         scanf("%d",&e);
652         state=DeleteNode(T,e);
653         if(state==OK) printf("删除成功");
654         else if(state==ERROR) printf("删除失败, 节点不存在");
655         break;
656
657     case 11:
658         printf("前序遍历结果如下: \n");
659         PreOrderTraverse(T,visit);
660         break;
661
662     case 12:
663         printf("中序遍历结果如下: \n");
```

```
664         InOrderTraverse(T, visit);
665         break;
666
667     case 13:
668         printf("后序遍历结果如下: \n");
669         PostOrderTraverse(T, visit);
670         break;
671
672     case 14:
673         printf("层序遍历结果如下: \n");
674         LevelOrderTraverse(T, visit);
675         break;
676
677     case 15:
678         state=MaxPathSum(T);
679         printf("该二叉树的最大路径和为: %d", state);
680         break;
681
682     case 16:
683         int e1, e2;
684         printf("请输入节点一和二的关键字: ");
685         scanf("%d%d", &e1, &e2);
686         node=findLCA(T, e1, e2);
687         if(node==NULL) printf("找不到");
688         else printf("节点名: %s\n节点关键字: %d", node->
            data.others, node->data.key);
689         break;
690
691     case 17:
692         InvertTree(T);
693         printf("更新成功");
694         break;
695
696     case 18:
697         SaveBiTree(T, "D:\\tounge.txt");
```

```
698         printf("保存成功");
699         break;
700
701     case 19:
702         LoadBiTree(T, "D:\\tounge.txt");
703         printf("读取成功");
704         break;
705
706     case 20:
707         printf("请输入新表的名字: ");
708         scanf("%s", str);
709         state=AddTree(trees, str);
710         if( state==OK) printf("添加成功");
711         else if( state==ERROR) printf("已经存在, 无法添加");
712         break;
713
714     case 21:
715         printf("请输入你要查找的表名: ");
716         scanf("%s", str);
717         state=LocateTree(trees, str);
718         if( state==0) printf("不存在");
719         else printf("位置是:%d", state);
720         break;
721
722     case 22:
723         if(trees.length){
724             printf("\n----- All lists\n");
725             for( int i=0; i<trees.length; ++i) {
726                 printf("%d\t%s\n", i+1, trees.elem[i].name);
727             }
728             printf("\n----- end\n");
729         }
```

华中科技大学课程实验报告

```
730         printf("请输入想要进行操作的树的序号 (从1开始) : "
731                );
732         scanf("%d", &i);
733         T=SwitchTree(trees , i);
734         if (T == NULL)
735             printf("输入的序号不合法! 单表已置空! \n");
736         else
737             printf("下面可对线性表集合中序号为%d的树进行操作! \n", i);
738         break;
739     case 23:
740         printf("请输入你要删除的表名");
741         scanf("%s", str);
742         state=RemoveTree(trees , str);
743         if( state==OK) printf("删除成功");
744         break;
745
746     case 0:
747         op=0;
748         break;
749
750     default:
751         break;
752     }
753     Sleep(2000);
754 }
755 }
```

D 附录 D 基于邻接表图实现的源程序

```
1      #ifndef DEF_H_INCLUDED
2  #define DEF_H_INCLUDED
3
4      #include "stdio.h"
5      #include "stdlib.h"
6      #define TRUE 1
7      #define FALSE 0
8      #define OK 1
9      #define ERROR 0
10     #define INFEASIBLE -1
11     #define OVERFLOW -2
12     #define MAX_VERTEX_NUM 20
13     typedef int status;
14     typedef int KeyType;
15     typedef enum {DG,DN,UDG,UDN} GraphKind;
16     typedef struct {
17         KeyType key;
18         char others[20];
19     } VertexType; //顶点类型定义
20
21
22     typedef struct ArcNode {           //表结点类型定义
23         int adjvex;                     //顶点位置编号
24         struct ArcNode *nextarc;       //下一个表结点指针
25     } ArcNode;
26     typedef struct VNode{              //头结
27         VertexType data;                //顶点信息
28         ArcNode *firstarc;              //指向第一条弧
29     } VNode, AdjList[MAX_VERTEX_NUM];
30     typedef struct { //邻接表的类型定义
31         AdjList vertices;               //头结点数组
32         int vexnum, arcnum;             //顶点数、弧数
```



```
33         GraphKind  kind;           //图的类型
34     } ALGraph;
35     typedef struct {
36         struct {
37             char name[30];
38             ALGraph G;
39         } elem[10];
40         int length;
41         int listsize;
42     } ALGraphs;
43
44 #endif // DEF_H_INCLUDED
45
46 #ifndef OPT_H_INCLUDED
47 #define OPT_H_INCLUDED
48
49
50 #include <queue>
51 #include <cstring>
52 #include "def.h"
53
54 using namespace std;
55
56 bool CheckDuplicateKeys(VertexType V[], int vernum) {
57     for (int i = 0; i < vernum - 1; i++) {
58         for (int j = i + 1; j < vernum; j++) {
59             if (V[i].key == V[j].key) {
60                 return true;
61             }
62         }
63     }
64     return false;
65 }
66
67 status CreateGraph(ALGraph &G, VertexType V[], KeyType VR[][2])
68 /*根据V和VR构造图T并返回OK, 如果V和VR不正确, 返回ERROR
```

```
68 如果有相同的关键字，返回ERROR。此题允许通过增加其它函数辅助实
    现本关任务*/
69 {
70     // 请在这里补充代码，完成本关任务
71     /***** Begin *****/
72     G.vexnum=0;
73     G.arcnum=0;
74     G.kind=UDG;
75     //初始化头部节点
76     for (int i = 0; i<MAX_VERTEX_NUM; i++){
77         G.vertices[i].data.key = -1;
78         G.vertices[i].firstarc = NULL;
79     }
80     // 遍历顶点序列和关系对序列，创建图的头结点
81     while(V[G.vexnum].key != -1){
82         G.vertices[G.vexnum].data = V[G.vexnum];
83         G.vertices[G.vexnum].firstarc=NULL;
84         G.vexnum++;
85         if(G.vexnum>MAX_VERTEX_NUM) return ERROR;
86     }
87     if(G.vexnum==0) return ERROR; //判断是否有顶点
88
89     if(CheckDuplicateKeys(V, G.vexnum)) return ERROR; //检查是否
        有重复节点
90     // 遍历关系对序列，构建邻接表
91     for(int i = 0; VR[i][0]!=-1 && VR[i][1] != -1; i++){
92         KeyType v1=VR[i][0];
93         KeyType v2=VR[i][1];
94
95         // 找到 v1 和 v2 对应的顶点位置
96         int pos_v1 = -1, pos_v2 = -1;
97         for(int j=0;j<G.vexnum; j++){
98             if(G.vertices[j].data.key == v1)
99                 pos_v1 = j;
100             else if(G.vertices[j].data.key == v2)
```

```
101         pos_v2 =j;
102     }
103     // 无效顶点
104     if(pos_v1 == -1||pos_v2 == -1){
105         return ERROR;
106     }
107     // 创建表结点并插入邻接表中
108     ArcNode *arc1=(ArcNode*)malloc(sizeof(ArcNode));
109     arc1->adjvex=pos_v2;
110     arc1->nextarc=G.vertices[pos_v1].firstarc;
111     G.vertices[pos_v1].firstarc=arc1;
112
113     ArcNode *arc2 = (ArcNode*)malloc(sizeof(ArcNode));
114     arc2->adjvex = pos_v1;
115     arc2->nextarc = G.vertices[pos_v2].firstarc;
116     G.vertices[pos_v2].firstarc = arc2;
117
118     G.arcnum++;
119 }
120 return OK;
121 }
122 /***** End *****/
123
124 status DestroyGraph(ALGraph &G)
125 /*销毁无向图G,删除G的全部顶点和边*/
126 {
127     // 请在这里补充代码，完成本关任务
128     /***** Begin *****/
129     //删除边
130     for(int i=0;i<G.vexnum;++i){
131         ArcNode *p= G.vertices[i].firstarc;
132         while(p!=NULL){
133             ArcNode *current=p;
134             p=p->nextarc;
135             free(current);
```

```
136         current=NULL;
137     }
138 }
139 G.vexnum=0;
140 G.arcnum=0;
141 return OK;
142 ***** End *****/
143 }
144
145 int LocateVex(ALGraph G,KeyType u)
146 //根据u在图G中查找顶点，查找成功返回位序，否则返回-1;
147 {
148     // 请在这里补充代码，完成本关任务
149     ***** Begin *****/
150     for (int i=0;i<G.vexnum;++i) {
151         if(G.vertices[i].data.key==u) return i;
152     }
153     return -1;
154     ***** End *****/
155 }
156
157 status PutVex(ALGraph &G,KeyType u,VertexType value)
158 //根据u在图G中查找顶点，查找成功将该顶点值修改成value，返回OK
159 ;
160 //如果查找失败或关键字不唯一，返回ERROR
161 {
162     // 请在这里补充代码，完成本关任务
163     ***** Begin *****/
164     for (int i=0;i<G.vexnum-1;++i) {
165         if(G.vertices[i].data.key==value.key)
166             return ERROR;
167     } //检查是否有重复的节点
168     for (int i=0;i<G.vexnum;++i) {
169         if(G.vertices[i].data.key==u) {
170             G.vertices[i].data=value;
```

```
170         return OK;
171     }
172 }
173 return ERROR;
174 /****** End *****/
175 }
176
177 int FirstAdjVex(ALGraph G,KeyType u)
178 //根据u在图G中查找顶点，查找成功返回顶点u的第一邻接顶点位序，
否则返回-1；
179 {
180     // 请在这里补充代码，完成本关任务
181     /****** Begin *****/
182     int pos_1=LocateVex(G,u);
183     if(pos_1==-1)return -1;
184     ArcNode *p=G.vertices[pos_1].firstarc;
185     return p->adjvex;
186     /****** End *****/
187 }
188
189 int NextAdjVex(ALGraph G,KeyType v,KeyType w)
190 //v对应G的一个顶点,w对应v的邻接顶点；操作结果是返回v的（相对于
w）下一个邻接顶点的位序；
191 //如果w是最后一个邻接顶点，或v、w对应顶点不存在，则返回-1。
192 {
193     // 请在这里补充代码，完成本关任务
194     /****** Begin *****/
195     int pos_1=LocateVex(G,v);
196     int pos_2=LocateVex(G,w);
197     if(pos_1==-1||pos_2==-1)return -1;
198
199     ArcNode* p=G.vertices[pos_1].firstarc;
200     while(p&&p->nextarc){
201         if(p->adjvex==pos_2)return p->nextarc->adjvex;
202         p=p->nextarc;
```

```
203     }
204     return -1;
205     /***** End *****/
206 }
207
208 status InsertVex(ALGraph &G, VertexType v)
209 // 在图G中插入顶点v, 成功返回OK, 否则返回ERROR
210 {
211     // 请在这里补充代码, 完成本关任务
212     /***** Begin *****/
213     // 先判断是否重复
214     for (int i=0; i<G.vexnum; ++i) {
215         if (G.vertices[i].data.key==v.key) return ERROR;
216     }
217     // 判断是否已满
218     if (G.vexnum==20) return ERROR;
219     VNode newNode;
220     newNode.data=v;
221     newNode.firstarc=NULL;
222     G.vertices[G.vexnum++]=newNode;
223     return OK;
224     /***** End *****/
225 }
226
227 void DeleteVex_A_Node(ALGraph &G, VNode &Node, int index)
228 {
229     ArcNode * p =Node.firstarc;
230     while (Node.firstarc!=NULL && Node.firstarc->adjvex==index)
231     {
232         G.arcnum--;
233         Node.firstarc=Node.firstarc->nextarc;
234         free(p);
235         p=Node.firstarc;
236     }
237     if (Node.firstarc==NULL) return;
```

```
238     ArcNode * q = Node.firstarc ->nextarc;
239     for (; q!=NULL;)
240     {
241         if (q->adjvex==index)
242         {
243             G.arcnum--;
244             p->nextarc=q->nextarc;
245             free(q);
246             q=p->nextarc;
247         }
248         else
249         {
250             p=p->nextarc;
251             q=q->nextarc;
252         }
253     }
254 }
255 status DestroyList(ArcNode * f)
256 {
257     if (f==NULL) return INFEASIBLE;
258     ArcNode * p=f;
259     while(p)
260     {
261         f=f->nextarc;
262         free(p);
263         p=f;
264     }
265     return OK;
266 }
267 status DeleteVex(ALGraph &G,KeyType v)
268 // 在图G中删除关键字v对应的顶点以及相关的弧, 成功返回OK, 否则返
    回ERROR
269 {
270     // 请在这里补充代码, 完成本关任务
271     /***** Begin *****/
```

```
272     int loc=LocateVex(G,v);
273     if (loc==-1||G.vexnum==1) return ERROR;
274     for(int i=0;i<G.vexnum;i++)
275     {
276         DeleteVex_A_Node(G,G.vertices[i],loc);
277     }
278     for(int i=0;i<G.vexnum;i++)
279     {
280         ArcNode * f=G.vertices[i].firstarc;
281         for(;f!=NULL;f=f->nextarc)
282         {
283             if(f->adjvex>loc)
284             {
285                 (f->adjvex)--;
286             }
287         }
288     }
289     DestroyList(G.vertices[loc].firstarc);
290     for (int i=loc;i<G.vexnum-1;++i)
291         G.vertices[i] = G.vertices[i+1];
292     --G.vexnum;
293     return OK;
294     /****** End *****/
295 }
296 status InsertArc(ALGraph &G,KeyType v,KeyType w)
297 // 在图G中增加弧<v,w>, 成功返回OK, 否则返回ERROR
298 {
299     // 请在这里补充代码, 完成本关任务
300     /****** Begin *****/
301     int pos_1=LocateVex(G,v);
302     int pos_2=LocateVex(G,w);
303     if(pos_1==-1||pos_2==-1)return ERROR;
304     //检查是否加入了重复的边
305     ArcNode *p=G.vertices[pos_1].firstarc;
306     while(p){
```



```
307         if(p->adjvex==pos_2) return ERROR;
308         p=p->nextarc;
309     }
310
311     ArcNode *v1=(ArcNode*)malloc(sizeof(ArcNode));
312     v1->adjvex=pos_2;
313     v1->nextarc=G.vertices[pos_1].firstarc;
314     G.vertices[pos_1].firstarc=v1;
315
316     ArcNode *v2=(ArcNode*)malloc(sizeof(ArcNode));
317     v2->adjvex=pos_1;
318     v2->nextarc=G.vertices[pos_2].firstarc;
319     G.vertices[pos_2].firstarc=v2;
320     ++G.arcnum;
321     return OK;
322     ***** End *****
323 }
324
325 status DeleteArc(ALGraph &G,KeyType v,KeyType w)
326 // 在图G中删除弧<v,w>, 成功返回OK, 否则返回ERROR
327 {
328     // 请在这里补充代码, 完成本关任务
329     ***** Begin *****
330     int pos_1=LocateVex(G,v);
331     int pos_2=LocateVex(G,w);
332     if(pos_1==-1||pos_2==-1) return ERROR;
333
334     ArcNode *p1 = G.vertices[pos_1].firstarc;
335     ArcNode *prev_p1 = nullptr;
336     while (p1) {
337         if (p1->adjvex == pos_2) {
338             if (prev_p1) {
339                 prev_p1->nextarc = p1->nextarc;
340             } else {
341                 G.vertices[pos_1].firstarc = p1->nextarc;
```

```
342         }
343         free(p1);
344         //p1=(prev_p1? prev_p1->nextarc:G.vertices[pos_1].
           firstarc);
345         break;
346     }
347     prev_p1 = p1;
348     p1 = p1->nextarc;
349 }
350 if (!p1)
351     return ERROR;
352
353 ArcNode *p2 = G.vertices[pos_2].firstarc;
354 ArcNode *prev_p2 = nullptr;
355 while (p2) {
356     if (p2->adjvex == pos_1) {
357         if (prev_p2) {
358             prev_p2->nextarc = p2->nextarc;
359         } else {
360             G.vertices[pos_2].firstarc = p2->nextarc;
361         }
362         free(p2);
363         //p2=(prev_p2? prev_p2->nextarc:G.vertices[pos_2].
           firstarc);
364         break;
365     }
366     prev_p2 = p2;
367     p2 = p2->nextarc;
368 }
369 G.arcnum--;
370 return OK;
371 /***** End *****/
372 }
373
374 int visited[25];
```

```
375 void dfs(ALGraph G, int n, void (*visit)(VertexType)){
376     visited[n]=1;
377     visit(G.vertices[n].data);
378     ArcNode *p=G.vertices[n].firstarc;
379     while(p){
380         if(visited[p->adjvex]==0)dfs(G,p->adjvex, visit);
381         p=p->nextarc;
382     }
383 }
384 status DFSTraverse(ALGraph &G, void (*visit)(VertexType)){
385     for(int i=0; i<25; ++i) visited[i]=0;
386     for(int i=0; i<G.vexnum; ++i){
387         if(visited[i]==0)dfs(G, i, visit);
388     }
389 }
390
391 status BFSTraverse(ALGraph &G, void(*visit)(VertexType)){
392     for(int i=0; i<25; ++i) visited[i]=0;
393     queue<int>Queue;
394     for(int i=0; i<G.vexnum; ++i){
395         if(visited[i]==0){
396             Queue.push(i);
397             visited[i]=1;
398         }
399         while(!Queue.empty()){
400             int cur = Queue.front();
401             Queue.pop();
402             visit(G.vertices[cur].data);
403             ArcNode*p=G.vertices[cur].firstarc;
404             while(p){
405                 if(visited[p->adjvex]==0){Queue.push(p->adjvex
406                     );
407                     visited[p->adjvex]=1;}
408                 p=p->nextarc;
409             }
410         }
411     }
```

```
409     }
410 }
411     return OK;
412 }
413
414 status SaveGraph(ALGraph G, char FileName[])
415 //将图的数据写入到文件FileName中
416 {
417     FILE*fp1;
418     fp1=fopen(FileName,"w");
419     int i,j,tt=-1;
420     int nnn[30][2];
421     i=0;
422     while(i<G.vexnum)
423     {
424         fprintf(fp1,"%d %s ",G.vertices[i].data.key,G.vertices
            [i].data.others);
425         ++i;
426     }
427     i=0;
428     fprintf(fp1,"%d null ",tt);
429     while(i<G.vexnum)
430     {
431         for(j=0;j<30;++j)
432         {
433             nnn[j][0]=-1;
434             nnn[j][1]=-1;
435         }
436         j=0;
437         if(G.vertices[i].firstarc!=NULL)
438         {
439             // if(G.vertices[i].data.key<G.vertices[G.vertices
            [i].firstarc->adjvex].data.key)
440             // {
441             //     nnn[j][0]=G.vertices[i].data.key;
```

```
442          //      nnn[j][1]=G.vertices[G.vertices[i].firstarc
              ->adjvex].data.key;
443          //      ++j;
444          // }
445          ArcNode*p=G.vertices[i].firstarc;
446          while(p!=NULL)
447          {
448              if(G.vertices[i].data.key<G.vertices[p->adjvex
                  ].data.key)
449              {
450                  nnn[j][0]=G.vertices[i].data.key;
451                  nnn[j][1]=G.vertices[p->adjvex].data.key;
452                  ++j;
453              }
454              p=p->nextarc;
455          }
456      }
457      --j;
458      while(j>=0)
459      {
460          fprintf(fp1, "%d %d ", nnn[j][0], nnn[j][1]);
461          --j;
462      }
463      ++i;
464  }
465  fprintf(fp1, "%d %d ", tt, tt);
466  fclose(fp1);
467  return OK;
468
469 }
470
471 status LoadGraph(ALGraph &G, char FileName[]) {
472     FILE *fp=fopen(FileName, "r+");
473     VertexType V[30];
474     int VR[100][2];
```

```
475     int i=0;
476     do{
477         fscanf(fp, "%d %s ", &V[i].key, V[i].others);
478     } while (V[i++].key != -1); // 将点的数据取出
479     i=0;
480     do{
481         fscanf(fp, "%d %d ", &VR[i][0], &VR[i][1]);
482     } while (VR[i++][0] != -1);
483     fclose(fp);
484     CreateGraph(G, V, VR);
485 }
486
487 status AddALGraph(ALGraphs &GS, char ListName[]) {
488     for (int i=0; i<GS.length; i++)
489         if (strcmp(GS.elem[i].name, ListName) == 0)
490             {
491                 printf("多表中已存在名称为 %s 的图! \n", ListName)
492                 ;
493                 return ERROR;
494             }
495     strcpy(GS.elem[GS.length].name, ListName);
496     GS.elem[GS.length].G.vexnum=0;
497     GS.elem[GS.length].G.arcnum=0;
498     GS.length++;
499     return OK;
500 }
501
502 status LocateALGraph(ALGraphs &GS, char ListName[]) {
503     for (int i=0; i<GS.length; ++i) {
504         if (strcmp(GS.elem[i].name, ListName) == 0)
505             return i+1;
506     }
507     return 0;
508 }
```

```
509 status RemoveALGraph(ALGraphs &GS, char ListName[]) {
510     for(int i=0; i<GS.length; ++i) {
511         if(strcmp(GS.elem[i].name, ListName)==0) {
512             for(int j=i; j<GS.length-1; ++j) {
513                 GS.elem[j]=GS.elem[j+1];
514             }
515             GS.length--;
516             return OK;
517         }
518     }
519     return ERROR;
520 }
521
522 void ALGraphTraverse(ALGraph &G) {
523     for(int i=0; i<G.vexnum; ++i) {
524         printf("%d %s", G.vertices[i].data.key, G.vertices[i].
525             data.others);
526         ArcNode *p=G.vertices[i].firstarc;
527         while(p!=nullptr) {
528             printf(" %d", p->adjvex);
529             p=p->nextarc;
530         }
531         printf("\n");
532     }
533 #endif
534
535 #include "def.h"
536 #include "opt.h"
537 #include <stdio.h>
538 #include <string.h>
539 #include <windows.h>
540 #include <queue>
541
542 void visit(VertexType v)
```

```
543 {
544     printf( " %d,%s ",v.key,v.others );
545 }
546
547 int g[40][40];
548
549 void MatrixTransformer( ALGraph G) {
550     for( int i=0;i<40;++i )
551         for( int j=0;j<40;++j )
552             g[i][j]=0;
553     for( int i=0;i<G.vexnum;++i ) {
554         ArcNode *p=G.vertices[i].firstarc ;
555         while(p) {
556             g[i][p->adjvex]=1;
557             p=p->nextarc ;
558         }
559     }
560
561 }
562
563 int ans[20],anss[20];
564 status VerticesSetLessThanK( ALGraph G,KeyType v,int k) {
565     MatrixTransformer(G); // 矩阵化
566     int p = LocateVex(G,v);
567     for( int i=0;i<20;++i ) ans[i]=0;
568     if(p==-1) return ERROR; // 节点不存在
569     ans[p]=1;
570     for( int i=1;i<k;++i ) {
571         for( int h=0;h<20;++h ) anss[h]=ans[h];
572         for( int j=0;j<20;++j ) {
573             if( anss[j]==1 ) {
574                 for( int k=0;k<G.vexnum;++k )
575                     if( g[j][k]==1 ) ans[k]=1; // 利用邻接矩阵，两
576                                     点之间有边即可以标记
577             }
578         }
579     }
580 }
```



```
577         }
578     }
579     return OK;
580 }
581
582 int tm[40][40];
583 void MatrixMul() {
584     for (int i=0; i<20; ++i) {
585         for (int j=0; j<20; ++j) {
586             tm[i][j]=0;
587         }
588     }
589     for (int i=0; i<20; ++i) {
590         for (int j=0; j<20; ++j) {
591             for (int k=0; k<20; ++k) {
592                 tm[i][j]+=g[i][k]*g[k][j];
593             }
594         }
595     }
596     for (int i=0; i<20; ++i) {
597         for (int j=0; j<20; ++j) {
598             g[i][j]=tm[i][j];
599         }
600     }
601 } //利用矩阵的乘法判断可达性
602
603 int ShortestPathLength(ALGraph G, KeyType v, KeyType w) {
604     int s1=LocateVex(G, v), s2=LocateVex(G, w);
605     if (s1==-1||s2==-1) return -1;
606     MatrixTransformer(G);
607     if (g[s1][s2]) return 1;
608     for (int i=0; i<G.vexnum; ++i) {
609         MatrixMul();
610         if (g[s1][s2]) return i+2;
611     }
```

```
612     return -1;
613     /******/
614 }
615
616 int shortest2 (ALGraph G,KeyType v,KeyType w){
617     int i=0,j=0,pos_1=-1,pos_2=-1,minid=0,min=1;
618     int inf=1<<30;
619     int dist[G.vexnum]={0}; // 距离数组
620     int book[G.vexnum]={0}; // 标记数组
621     pos_1=LocateVex (G,v);
622     pos_2=LocateVex (G,w);
623     if(pos_1==-1||pos_2==-1) return -1;
624     book[pos_1]++;
625     for (int i=0;i<G.vexnum;++i) dist[i]=inf;
626     ArcNode *p=G.vertices[pos_1].firstarc;
627     dist[pos_1]=0;
628     while(p!=NULL){
629         dist[p->adjvex]=1;
630         p=p->nextarc;
631     } // 首节点
632     for (int i=0;i<G.vexnum-1;++i){
633         min=inf;
634         for (int j=0;j<G.vexnum;++j){
635             if(book[j]==0&&min>dist[j]){
636                 minid=j;
637                 min=dist[j];
638             }
639         } // 找到距离最近的节点
640         book[minid]++;
641         p=G.vertices[minid].firstarc;
642         while(p!=NULL){
643             if (dist[p->adjvex]>1+dist[minid]){
644                 dist[p->adjvex]=1+dist[minid];
645             }
646             p=p->nextarc;
```

```
647     }
648 }
649 if( dist[pos_2]==inf) return -1;
650 return dist[pos_2];
651 }
652 int book[100];
653 void bfs(ALGraph G,int startnode){
654     std::queue<int>Q;
655     Q.push(startnode);
656     while(!Q.empty()){
657         int p=Q.front();
658         Q.pop();
659         book[p]=1;
660         ArcNode *pp=G.vertices[p].firstarc;
661         while(pp!=NULL){
662             if(book[pp->adjvex]==0){
663                 Q.push(pp->adjvex);
664             }
665             pp=pp->nextarc;
666         }
667     }
668 }
669 int ConnectedComponentsNums(ALGraph &G){ //连通分支
670     for(int i=0;i<100;++i) book[i]=0;
671     int cnt=0;
672     if(G.vertices==NULL) return 0;
673     for(int i=0;i<G.vexnum;++i){
674         if(book[i]==0){
675             cnt++;
676             book[i]++;
677             bfs(G,i);
678         }
679     }
680     return cnt;
681 }
```

```

682
683 int main() {
684     int op=1;
685     int state;
686     ALGraphs GS;
687     GS.length=0;
688     GS.listsize=10;
689     ALGraph G;
690     while(op){
691         system("cls");
692         printf("\n\n");
693         printf("          Menu for Linear Table On
694                Sequence Structure \n");
695         printf("
        -----|
                n");
696         printf("          1. 创建图          10. 删除弧
                \n");
697         printf("          2. 销毁图          11. 深度优
                先遍历\n"); //5 xxb 8 jh 7 ecs 6 wxt -1 nil
                5 6 5 7 6 7 7 8 -1 -1
698         printf("          3. 查找节点          12. 广度优
                先遍历\n"); //5 线性表 8 集合 7 二叉树 6 无
                向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1
699         printf("          4. 顶点赋值          13. 距离小
                于k的顶点集合\n");
700         printf("          5. 获得第一邻接点  14. 最小路
                径\n");
701         printf("          6. 获得下一邻接  15. 图的连
                通分支\n");
702         printf("          7. 插入顶点          16. 文件保
                存\n");
703         printf("          8. 删除顶点          17. 文件读
                取\n");
704         printf("          9. 插入弧          18. 添加图\

```

```

        n");
704         printf("          19. 查找图          20. 删除图 \
        n");
705         printf("          21. 选择图          22. 提交修
        改\n");
706     printf("          23. 遍历图          0. exit\n");
707     printf("
        -----\
        n");
708     printf("    请选择你的操作 [0~20]:\n");
709     scanf("%d",&op);
710     if(op==1){
711         printf("请输入数据: ");
712         VertexType V[30];
713         KeyType VR[100][2];
714         int i=0;
715         do {
716             scanf("%d%s",&V[i].key,V[i].others);
717         } while(V[i++].key!=-1);
718         i=0;
719         do {
720             scanf("%d%d",&VR[i][0],&VR[i][1]);
721         } while(VR[i++][0]!=-1);
722         if (CreateGraph(G,V,VR)==ERROR) printf("输入数据
        错, 无法创建\n");
723         else printf("创建成功! \n");
724     }
725     if(op==2){
726         state=DestroyGraph(G);
727         if(state==OK) printf("销毁成功");
728     }
729     if(op==3){
730         printf("请输入关键词:");
731         int key;
732         scanf("%d",&key);
    
```

```
733         state=LocateVex(G,key);
734         if( state==-1)printf("不存在该节点");
735         else printf("位序为%d\n",state);
736     }
737     if(op==4){
738         printf("请输入关键词和所赋的新值:(所赋的新值包括
              两个)");
739         int key;
740         VertexType value;
741         scanf("%d%d%s",&key,&value.key,value.others);
742         state=PutVex(G,key,value);
743         if( state==ERROR)printf("赋值失败");
744         else printf("赋值成功");
745     }
746     if(op==5){
747         printf("请输入关键词:");
748         int key;
749         scanf("%d",&key);
750         state=FirstAdjVex(G,key);
751         if( state==-1)printf("不存在");
752         else printf("第一邻接点为%d,%s",G.vertices[state].
              data.key,G.vertices[state].data.others);
753     }
754     if(op==6){
755         printf("请输入2个关键词:");
756         int key1;
757         int key2;
758         scanf("%d%d",&key1,&key2);
759         state=NextAdjVex(G,key1,key2);
760         if( state==-1)printf("不存在");
761         else printf("下一个邻接点为%d,%s",G.vertices[state].
              data.key,G.vertices[state].data.others);
762     }
763     if(op==7){
764         printf("请输入关键词(包括字符串):");
```

```
765         VertexType value;
766         scanf( "%d%s",&value.key,value.others);
767         state=InsertVex(G,value);
768         if( state==ERROR) printf( "插入失败");
769         else printf( "插入成功");
770     }
771     if( op==8){
772         printf( "请输入关键词(不包括字符串): ");
773         int key;
774         scanf( "%d",&key);
775         state=DeleteVex(G,key);
776         if( state==ERROR) printf( "删除失败");
777         else printf( "删除成功");
778     }
779     if( op==9){
780         printf( "请输入你想加边的两个点:");
781         int k1,k2;
782         scanf( "%d%d",&k1,&k2);
783         state=InsertArc(G,k1,k2);
784         if( state==ERROR) printf( "插入失败");
785         else printf( "插入成功");
786     }
787     if( op==10){
788         printf( "请输入你想要删除的边的两个点: ");
789         int k1,k2;
790         scanf( "%d%d",&k1,&k2);
791         state=DeleteArc(G,k1,k2);
792         if( state==ERROR) printf( "删除失败");
793         else printf( "删除成功");
794     }
795     if( op==11){
796         printf( "深度优先遍历结果如下: \n");
797         DFSTraverse(G,visit);
798     }
799     if( op==12){
```

```
800         printf("广度优先遍历结果如下: \n");
801         BFSTraverse(G, visit);
802     }
803     if (op==13){
804         printf("请输入关键词和距离: ");
805         int k1,k2;
806         scanf("%d%d",&k1,&k2);
807         state=VerticesSetLessThanK(G,k1,k2);
808         if (state==ERROR) printf("失败");
809         else {
810             printf("节点如下\n");
811             for (int i=0;i<G.vexnum;i++) if (ans[i]) printf("%d %s\n",G.vertices[i].data.key,G.vertices[i].data.others);
812         }
813     }
814     if (op==14){
815         printf("请输入两个关键词: ");
816         int k1,k2;
817         scanf("%d%d",&k1,&k2);
818         state=ShortestPathLength(G,k1,k2);
819         if (state==-1) printf("不连通");
820         else {
821             printf("最短距离为: %d",state);
822         }
823     }
824     if (op==15){
825         state=ConnectedComponentsNums(G);
826         printf("分支数为: %d",state);
827     }
828     if (op==16){
829         SaveGraph(G,"D:\\tougex.txt");
830         printf("保存成功");
831     }
832     if (op==17){
```



```
833         LoadGraph(G, "D:\\tounge.txt");
834         printf("读取成功");
835     }
836     if(op==18){
837         char ListName[20];
838         printf("请输入你要添加的图的名字: ");
839         scanf("%s", ListName);
840         state=AddALGraph(GS, ListName);
841         printf("添加成功");
842     }
843     if(op==19){
844         char ListName[20];
845         printf("请输入你要查找的图的名字: ");
846         scanf("%s", ListName);
847         state=LocateALGraph(GS, ListName);
848         if(state==0) printf("没有该图");
849         else printf("该图的位置是%d", state);
850     }
851     if(op==20){
852         char ListName[20];
853         printf("请输入你要删除的图的名字: ");
854         scanf("%s", ListName);
855         state=RemoveALGraph(GS, ListName);
856         if(state==ERROR) printf("没有该图");
857         else printf("删除成功");
858     }
859     if(op==21){
860         if(GS.length){
861             printf("\n----- All lists\n");
862             for(int i = 0; i < GS.length; i++){
863                 printf("%d %s\n", i + 1, GS.elem[i].name);
864             }
865             printf("----- end\n");
866         }
867         int key;
```

```
866         printf("请选择上述的序号之一进行操作:");
867         scanf("%d",&key);
868         G=GS.elem[key-1].G;
869         printf("现在你可以对该序号的图进行操作!");
870     }
871     else printf("为空,无法操作");
872     printf("请选择其中之一进行操作:");
873
874 }
875 if(op==22){
876     int key;
877     printf("请输入你想要修改的图的序号(先前选择的图):"
878         );
879     scanf("%d",&key);
880     GS.elem[key-1].G=G;
881     printf("修改成功");
882 }
883 if(op==23){
884     if(G.vexnum==0)printf("图为空");
885     else {
886         printf("遍历的结果如下\n");
887         ALGraphTraverse(G);
888     }
889 }
890 Sleep(2000);
891 }
892 }
```
