華中科技大學

课 程 实 验 报 告

课程名称： 数据结构实验

专业班级 CS2201

学　　号 U202215357

姓　　名 王文涛

指导教师 李剑军

报告日期 2023 年 5 月 29 日

计算机科学与技术学院

目　　录

1 基于顺序存储结构的线性表实现

1.1 问题描述

构造顺序表，呈现一个简易菜单的功能演示系统，该演示系统可选择实现多个线性表管理。需要在主程序中完成函数调用以及所需实参值和函数执行结果的输出。定义线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等函数，并给出适当的操作提示，并且可选择以文件的形式进行存储和加载，可以进行多线性表操作。依据最小完备性和常用性相结合的原则，以函数形式定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等 21 种功能，具体功能定义如下。

1) 初始化表：函数名称是 InitList(L)；初始条件是线性表 L 不存在；操作结果是构造一个空的线性表；
2) 销毁表：函数名称是 DestroyList(L)；初始条件是线性表 L 已存在；操作结果是销毁线性表 L；
3) 清空表：函数名称是 ClearList(L)；初始条件是线性表 L 已存在；操作结果是将 L 重置为空表；
4) 判定空表：函数名称是 ListEmpty(L)；初始条件是线性表 L 已存在；操作结果是若 L 为空表则返回 TRUE, 否则返回 FALSE；
5) 求表长：函数名称是 ListLength(L)；初始条件是线性表已存在；操作结果是返回 L 中数据元素的个数；
6) 获得元素：函数名称是 GetElem(L,i,e)；初始条件是线性表已存在，1≤i≤ListLength(L)；操作结果是用 e 返回 L 中第 i 个数据元素的值；
7) 查找元素：函数名称是 LocateElem(L,e,compare())；初始条件是线性表已存在；操作结果是返回 L 中第 1 个与 e 满足关系 compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为 0；
8) 获得前驱：函数名称是 PriorElem(L,cur_e,pre_e)；初始条件是线性表 L 已存在；操作结果是若 cur_e 是 L 的数据元素，且不是第一个，则用 pre_e 返回它的前驱，否则操作失败，pre_e 无定义；
9) 获得后继：函数名称是 NextElem(L,cur_e,next_e)；初始条件是线性表 L 已存在；操作结果是若 cur_e 是 L 的数据元素，且不是最后一个，则用 next_e

返回它的后继，否则操作失败，next_e 无定义；

10) 插入元素：函数名称是 ListInsert(L,i,e)；初始条件是线性表 L 已存在，1≤i≤ListLength(L)+1；操作结果是在 L 的第 i 个位置之前插入新的数据元素 e。
11) 删除元素：函数名称是 ListDelete(L,i,e)；初始条件是线性表 L 已存在且非空，1≤i≤ListLength(L)；操作结果：删除 L 的第 i 个数据元素，用 e 返回的值；
12) 遍历表：函数名称是 ListTraverse(L,visit())，初始条件是线性表 L 已存在；操作结果是依次对 L 的每个数据元素调用函数 visit()。

附加功能：

1) 最大连续子数组和：函数名称是 MaxSubArray(L)；初始条件是线性表 L 已存在且非空，请找出一个具有最大和的连续子数组（子数组最少包含一个元素），操作结果是其最大和；
2) 和为 K 的子数组：函数名称是 SubArrayNum(L,k)；初始条件是线性表 L 已存在且非空，操作结果是该数组中和为 k 的连续子数组的个数；
3) 顺序表排序：函数名称是 sortList(L)；初始条件是线性表 L 已存在；操作结果是将 L 由小到大排序；
4) 实现线性表的文件形式保存：其中，需要设计文件数据记录格式，以高效保存线性表数据逻辑结构 (D,R) 的完整信息；需要设计线性表文件保存和加载操作合理模式。附录 B 提供了文件存取的参考方法。
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。

1.1.1 实验目的

通过实验达到：

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

1.2 系统设计

1.2.1 系统总体设计

本系统提供一个基于顺序存储结构的多线性表的实现。

菜单可供选择的操作有：线性表的添加、线性表的销毁、线性表的查找、线性表的转换、初始化线性表、销毁表、清空表、判空表，求表长、得到某元素、查找元素、获得某元素的前驱、获得某元素的后继、插入元素、删除元素、遍历线性表、最大连续子数组和、和为 K 的子数组、顺序表排序、实现线性表的文件形式保存。

1.2.2 有关常量和类型定义

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>

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

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

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

```
}SqList;
```

1.2.3 算法设计

1) 函数名称：InitaList(L)；

初始条件：线性表 L 不存在已存在；

操作结果：是构造一个空的线性表；

算法思路：先分配存储空间后，将表长设为 0，再将线性表容量设为预定义的初始存储容量，图 1-1 为 InitaList 函数的流程图。

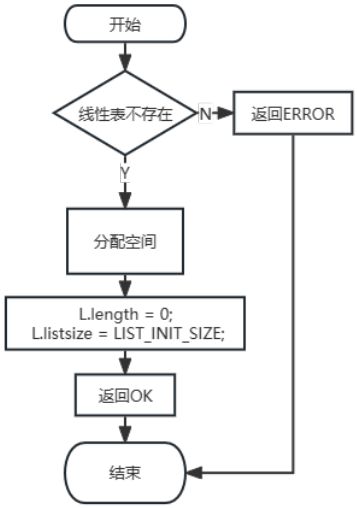

图 1-1 InitaList 流程图

2) 函数名称：DestroyList(L)；

初始条件：线性表 L 已存在；

操作结果：销毁线性表 L；

算法思路：释放内存并将各数据设置为初值，图 1-2 为 DestroyList 的流程图。

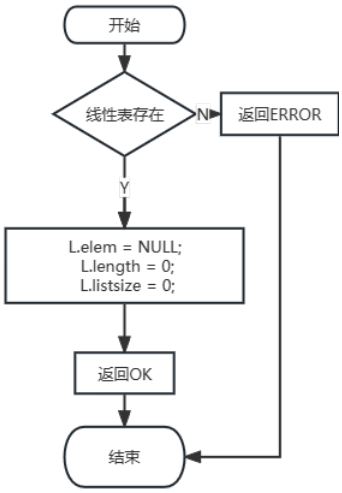

图 1-2　DestroyList 流程图

3) 函数名称：ClearList(L)；

初始条件：线性表 L 已存在；

操作结果：将 L 重置为空表；

算法思路：将表长设为 0 即可。

4) 函数名称：ListEmpty(L)；

初始条件：线性表 L 已存在；

操作结果：若 L 为空表则返回 TRUE, 否则返回 FALSE；

算法思路：表长为 0 则为空表，否则不是空表。，图 1-3 为 ListEmpty 的流程图。

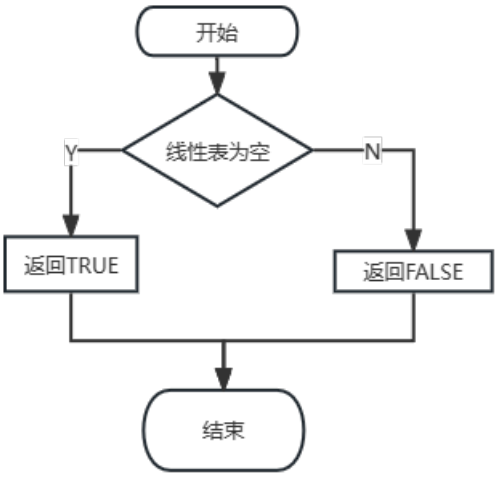

图 1-3　ListEmpty 流程图

5) 函数名称：ListLength(L)；
   初始条件：线性表已存在；
   操作结果：返回 L 中数据元素的个数；
   算法思路：返回线性表表长的值。
6) 函数名称：GetElem(L,i,e)；
   初始条件：线性表已存在，1≤i≤ListLength(L)；
   操作结果：用 e 返回 L 中第 i 个数据元素的值；
   算法思路：将线性表中第 i 个数据元素的值赋值给 e。图 1-4 为 GetElem 的流程图。

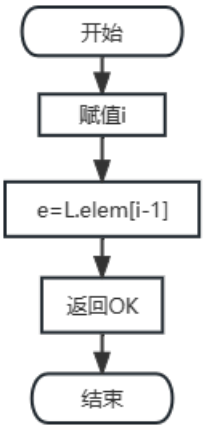

图 1-4　GetElem 流程图

7) 函数名称：LocateElem(L,e,compare())；

初始条件：线性表已存在；

操作结果：返回线性表中第 1 个与 e 相等的数据元素的位置，若这样的数据元素不存在，则返回值为 0；

算法思路：先遍历顺序表，将线性表中的数据元素依次与 e 进行比较，返回该元素的位序。

8) 函数名称：PriorElem(L,cur_e,pre_e)；

初始条件：线性表 L 已存在；

操作结果：若 cur_e 是 L 的数据元素并且不是第一个数据元素，用 pre_e 返回它的前驱，否则操作失败。

算法思路：首先遍历该顺序表。若找到该结点，并且该结点有前驱元素，则将前驱元素赋值给 pre_e。若未找到该结点，或者找到该结点但该结点不存在前驱元素，则返回 FALSE。图 1-5 为 PriorElem(L,cur_e,pre_e) 的流程图。

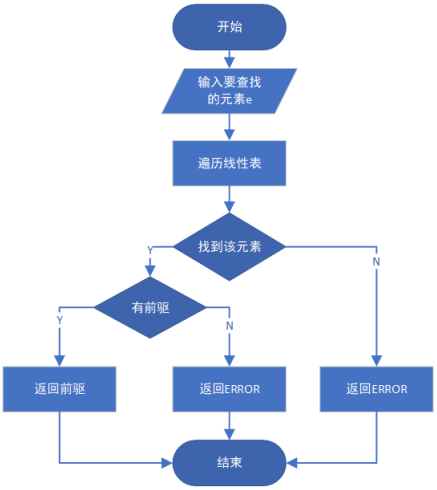

图 1-5　PriorElem 流程图

9) 函数名称：NextElem(L,cur_e,next_e)

初始条件：线性表 L 已存在；

操作结果：若 cur_e 是 L 的数据元素，且不是最后一个，则用 next_e 返回它的后继，否则操作失败，next_e 无定义。

算法思路：先遍历线性表，如果找到该结点并且该结点有后继元素，则将后继节点元素赋值给 next_e。若未找到该结点，或者找到该结点但该结点不存在后继元素，则返回 FALSE.

10) 函数名称：ListInsert(L,i,e)

初始条件：线性表 L 已存在且非空，1≤i≤ListLength(L)+1；

操作结果：在 L 的第 i 个位置之前插入新的数据元素 e。

算法思路：先遍历顺序表，若线性表 L 已存在且不为空，输入的 i 值不合法，则返回 ERROR。若满足线性表 L 已存在且 L 非空，并且 i 的值合法，则在线性表的第 i 个位置之前插入新的数据元素 e，返回 OK。图 1-6 为 ListInsert(L,i,e) 的流程图。

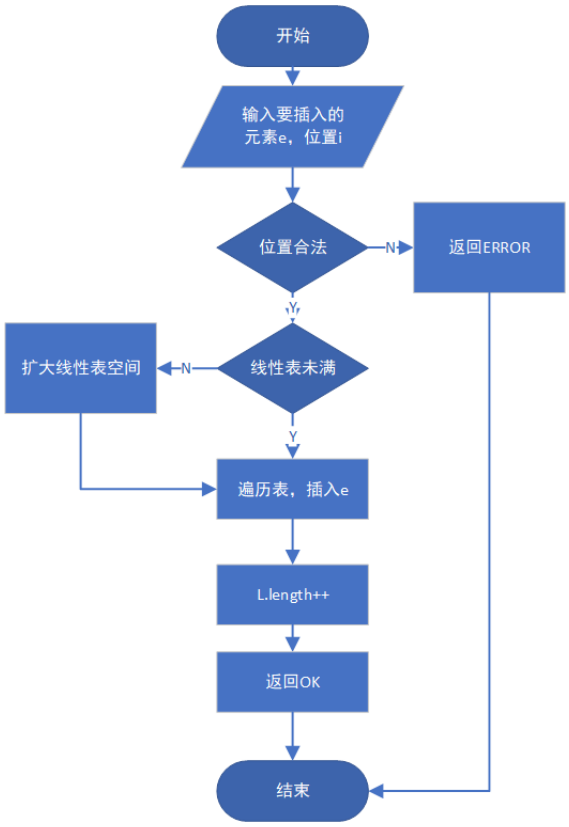

图 1-6　ListInsert 流程图

11) 函数名称：ListDelete(L,i,e)；

初始条件：线性表 L 已存在且非空，1≤i≤ListLength(L)；

操作结果：删除 L 的第 i 个数据元素，用 e 返回其值；

设计思想：先遍历顺序表，如果线性表 L 已存在且非空，并且输入的 i 值不合法，则返回 ERROR。若满足线性表 L 已存在且 L 非空，并且 i 的值合法，则删除线性表的第 i 个位置的数据元素，并用 e 返回其值，返回 OK。

12) 函数名称：ListTrabverse(L)；

初始条件：线性表 L 已存在；

操作结果：依次遍历 L 的每个数据元素；

设计思想：若线性表 L 存在，则遍历元素；否则返回 ERROR.

13) 函数名称：SaveList(L,filename)；

初始条件：线性表 L 已存在；

操作结果：将线性表 L 保存为文件形式；

算法思路：用 fprintf 保存为文件，图 1-7 为 SaveList(L,filename) 的流程图。

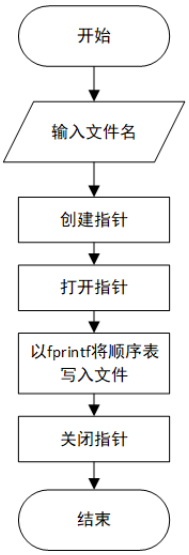

图 1-7　SaveList 流程图

14) 函数名称函数名称：LoadList (L);

初始条件：文件 filename 已存在；

操作结果：将线性表 L 以文件形式加载读取;

算法思路：用 fscanf 将文件读取顺序表。

1.3　系统实现

1.3.1　程序源代码

见《附录 A 基于顺序存储结构线性表实现的源程序》。

1.4　系统测试

程序采用简易界面，如图 1-8 所示，挑选 ListEmpty, GetElem, LocateElem, PriorElem, ListInsert, ListDelete, ListTrabverse 这些重要功能进行测试。

图 1-8　界面视图

1) ListEmpty 测试

测试用例	输入	理论结果	测试结果
{1，2，3}	7	线性表不为空!	线性表不为空!
{}	7	线性表为空!	线性表为空!
表不存在	7	线性表不存在!	线性表不存在!

表 1-1　ListEmpty 测试用例表

图 1-9　ListEmpty 测试截图

2) ListLength 测试

测试用例	输入	理论结果	测试结果
{1，2，3}	8	线性表的长度为 3	线性表的长度为 3
{}	8	线性表为空!	线性表为空!
表不存在	8	线性表不存在!	线性表不存在!

表 1-2　ListLength 测试用例表

图 1-10　ListLength 测试截图

3) GetElem 测试

测试用例	输入	理论结果	测试结果
{1，2，3}	9 2	这个元素是 2	这个元素是 2
{}	8	不存在该元素!	不存在该元素!
表不存在	8	线性表不存在!	线性表不存在!

表 1-3　GetElem 测试用例表

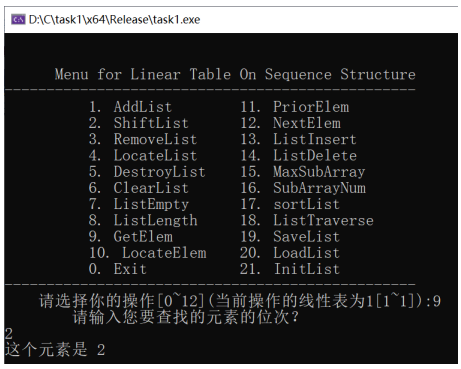

图 1-11　GetElem 测试截图 1

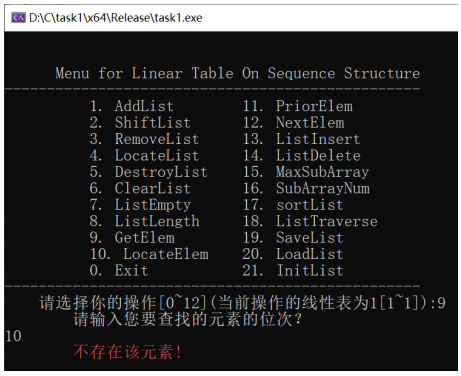

图 1-12　GetElem 测试截图 2

4) PriorElem 测试

测试用例	输入	理论结果	测试结果
{1，2，3}	11 1	该元素没有前驱!	该元素没有前驱!
{1，2，3}	11 2	该元素前驱为 1	该元素前驱为 1
表不存在	8	线性表不存在!	线性表不存在!

表 1-4　PriorElem 测试用例表

图 1-13　PriorElem 测试截图 1

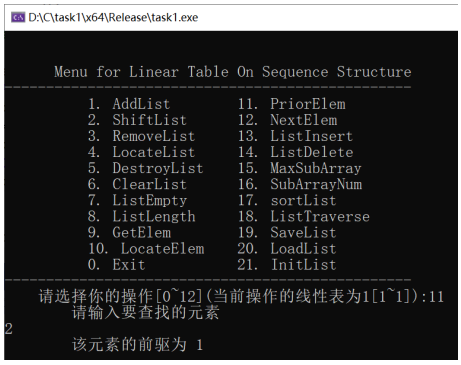

图 1-14　PriorElem 测试截图 2

5) ListInsert 测试

测试用例	输入	理论结果	测试结果
{1，2，3}	13 3 3	插入成功!	插入成功!
{1，2，3}	13 6 6	插入位置不正确! 1	插入位置不正确!
表不存在	8	线性表不存在!	线性表不存在!

表 1-5　ListInsert 测试用例表

图 1-15　ListInsert 测试截图 1

图 1-16　LListInsert 测试截图 2

6) ListTraverse 测试

测试用例	输入	理论结果	测试结果
q1{1，2，3}	18	q1 1 2 3	q1 1 2 3
表不存在	18	线性表不存在!	线性表不存在!

表 1-6　ListTraverse 测试用例表

图 1-17　ListTraverse 测试截图 1

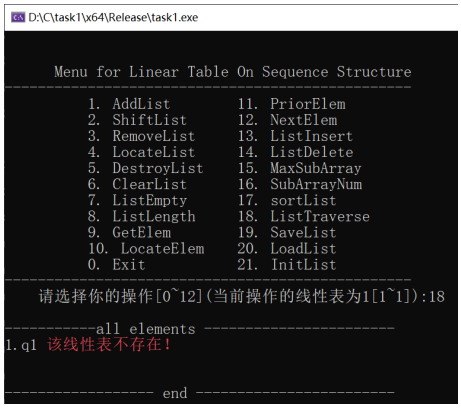

图 1-18　ListTraverse 测试截图 2

1.5 实验小结

本次实验要求我们在老师给出的框架上，完成一个处理线性表的系统，突出对课程内容的考查与训练。

在完成这次实验后，我受益匪浅。这是我第一次写一个几百行的程序，在过程中遇到了许多困难。比如说，如何处理输入流中的换行符，还有如何让输出停留在页面上。尤其是在完成附加功能时，为了实现文件的读与写，我研究了很长的时间，感觉自己对文件的写法有了更深的领悟。为了完成多线性表管理，更是耗费了我几天的时间，最后想到了用一个转换函数来对每个线性表单独操作。但是这次经历让我在处理相关的问题时，变得更加的熟练，在完成之后的实验就轻车熟路了。

这次实验也磨砺了我的意志，面对几十个 bug 也能慢慢修好，同时也锻炼了我查找信息，解决问题的能力。

本次实验锻炼了我们将理论与实践相结合的能力，更加深了我们对于数据结构这门课程的学习，从这次实验中，我学到了很多。最后，我由衷地感谢老师、助教和同学在本次实验中对我的帮助，帮助我解决实验中遇到的难题，谢谢！

2 基于邻接表的图实现

2.1 问题描述

物理结构为邻接表，创建一个简易菜单的功能演示界面。依据最小完备性和常用性相结合的原则，以函数形式定义了创建图、销毁图、查找顶点、获得顶点值和顶点赋值等 13 种基本运算，并给出了相应的操作提示。也可选择以文件的形式进行存储和加载，整个系统在主程序中完成函数调用。

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

1) 创建图：函数名称是 CreateCraph(&G,V,VR)；初始条件是 V 是图的顶点集，VR 是图的关系集；操作结果是按 V 和 VR 的定义构造图 G。
2) 销毁图：树函数名称是 DestroyBiTree(T)；初始条件图 G 已存在；操作结果是销毁图 G。
3) 查找顶点：函数名称是 LocateVex(G,u)；初始条件是图 G 存在，u 和 G 中的顶点具有相同特征；操作结果是若 u 在图 G 中存在，返回顶点 u 的位置信息，否则返回其它信息。
4) 获得顶点值：函数名称是 GetVex (G,v)；初始条件是图 G 存在，v 是 G 中的某个顶点；操作结果是返回 v 的值。
5) 顶点赋值：函数名称是 PutVex (G,v,value)；初始条件是图 G 存在，v 是 G 中的某个顶点；操作结果是对 v 赋值 value。
6) 获得第一邻接点：函数名称是 FirstAdjVex(&G, v)；初始条件是图 G 存在，v 是 G 的一个顶点；操作结果是返回 v 的第一个邻接顶点，如果 v 没有邻接顶点，返回“空”。
7) 获得下一邻接点：函数名称是 NextAdjVex(&G, v, w)；初始条件是图 G 存在，v 是 G 的一个顶点,w 是 v 的邻接顶点；操作结果是返回 v 的（相对于 w）下一个邻接顶点，如果 w 是最后一个邻接顶点，返回“空”。
8) 插入顶点：函数名称是 InsertVex(&G,v)；初始条件是图 G 存在，v 和 G 中的顶点具有相同特征；操作结果是在图 G 中增加新顶点 v。
9) 删除顶点：函数名称是 DeleteVex(&G,v)；初始条件是图 G 存在，v 是 G 的一个顶点；操作结果是在图 G 中删除顶点 v 和与 v 相关的弧。

10) 插入弧：函数名称是 InsertArc(&G,v,w)；初始条件是图 G 存在，v、w 是 G 的顶点；操作结果是在图 G 中增加弧 <v,w>，如果图 G 是无向图，还需要增加 <w,v>。
11) 删除弧：函数名称是 DeleteArc(&G,v,w)；初始条件是图 G 存在，v、w 是 G 的顶点；操作结果是在图 G 中删除弧 <v,w>，如果图 G 是无向图，还需要删除 <w,v>。
12) 深度优先搜索遍历：函数名称是 DFSTraverse(G,visit())；初始条件是图 G 存在；操作结果是图 G 进行深度优先搜索遍历，依次对图中的每一个顶点使用函数 visit 访问一次，且仅访问一次。
13) 广深度优先搜索遍历：函数名称是 BFSTraverse(G,visit())；初始条件是图 G 存在；操作结果是图 G 进行广度优先搜索遍历，依次对图中的每一个顶点使用函数 visit 访问一次，且仅访问一次。

附加功能：

1) 距离小于 k 的顶点集合：函数名称是 VerticesSetLessThanK(G,v,k)，初始条件是图 G 存在；操作结果是返回与顶点 v 距离小于 k 的顶点集合；
2) 顶点间最短路径和长度：函数名称是 ShortestPathLength(G,v,w); 初始条件是图 G 存在；操作结果是返回顶点 v 与顶点 w 的最短路径的长度；
3) 图的连通分量：函数名称是 ConnectedComponentsNums(G)，初始条件是图 G 存在；操作结果是返回图 G 的所有连通分量的个数；
4) 实现图的文件形式保存：其中，需要设计文件数据记录格式，以高效保存图的数据逻辑结构 (D,R) 的完整信息；需要设计图文件保存和加载操作合理模式。附录 B 提供了文件存取的方法；
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。

2.1.1 实验目的

通过实验达到：

1) 加深对图的概念、基本运算的理解；
2) 熟练掌握图的逻辑结构与物理结构的关系；
3) 以邻接表作为物理结构，熟练掌握图基本运算的实现。

2.2 系统设计

2.2.1 系统总体设计

物理结构为邻接表，创建一个简易菜单的功能演示界面。依据最小完备性和常用性相结合的原则，以函数形式定义了创建图、销毁图、查找顶点、获得顶点值和顶点赋值等 13 种基本运算，并给出了相应的操作提示。也可选择以文件的形式进行存储和加载，整个系统在主程序中完成函数调用，以及程序的退出。

2.2.2 有关常量和类型定义

```
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define MAX_VERTEX_NUM 20
typedef int status;
typedef int KeyType;
typedef enum { DG, DN, UDG, UDN } GraphKind;
typedef struct {
        KeyType  key;
        char others[20];
} VertexType; //顶点类型定义
typedef struct ArcNode {        //表结点类型定义
        int adjvex;             //顶点位置编号
        struct ArcNode* nextarc;        //下一个表结点指针
} ArcNode;
typedef struct VNode {           //头结点及其数组类型定义
        VertexType data;        //顶点信息
        ArcNode* firstarc;     //指向第一条弧
```

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

2.2.3 算法设计

1) 函数名称：CreateGraph(G)

操作结果：构造图 G

算法思路：采用邻接表作为物理结构，先输入顶点数和边数，再依次输入各顶点并赋值，最后将按边尾和边首表示的边输入并创建完无向图。

2) 函数名称：DestroyGraph(G)

初始条件：图 G 已存在

操作结果：销毁图 G

算法思路：遍历图，依次释放存储数据元素的空间，最后将图的几个顶点数、弧数和种类均置为零，完成销毁。图 2-1 为 DestroyGraph 的流程图

图 2-1　DestroyGraph 流程图

3) 函数名称：LocateVex(G,u)

初始条件：图 G 存在，u 和 G 中的顶点具有相同特征

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

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

4) 函数名称：GetVex (G,v)

初始条件：图 G 存在，v 是 G 中的某个顶点

操作结果：返回 v 的值

算法思路：返回该 key 所指向的顶点的值。

5) 函数名称：PutVex (G,v,value)

初始条件：图 G 存在，v 是 G 中的某个顶点

操作结果：对 v 赋值 value

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

图 2-2 为 PutVex 的流程图

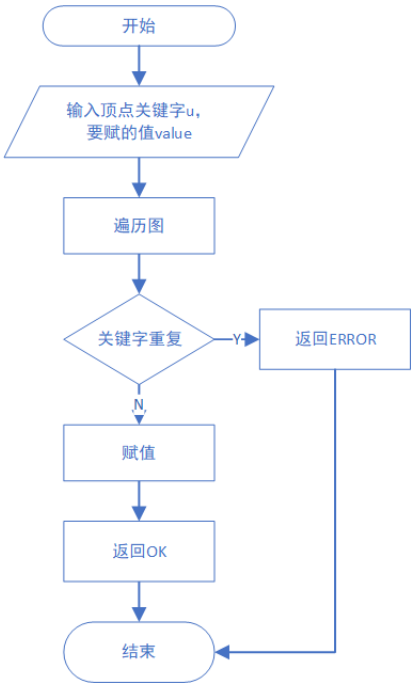

图 2-2　PutVex 流程图

6) 函数名称：FirstAdjVex(G, v)

初始条件：图 G 存在，v 是 G 的一个顶点

操作结果：返回 v 的第一个邻接顶点，如果 v 没有邻接顶点，返回“空”

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

7) 函数名称：NextAdjVex(G, v, w)

初始条件：图 G 存在，v 是 G 的一个顶点,w 是 v 的邻接顶点

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

算法思路：调用 LocateVex(G,u) 函数，并返回找到的顶点相对所给的邻接顶点的下一个邻接顶点。图 2-3 为 NextAdjVex 的流程图

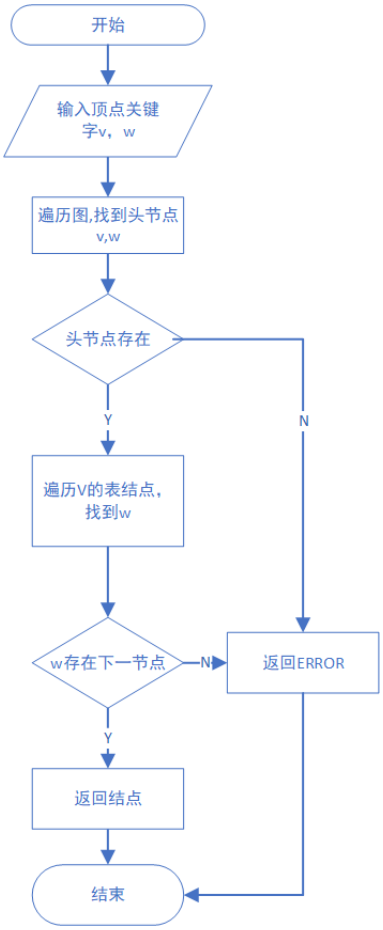

图 2-3　NextAdjVex 流程图

8) 函数名称：InsertVex(G,v)

初始条件：图 G 存在，v 和 G 中的顶点具有相同特征

操作结果：在图 G 中增加新顶点 v

算法思路：输入该顶点的值，插入新结点并对其赋值。

9) 函数名称：DeleteVex(G,v)

初始条件：图 G 存在，v 是 G 的一个顶点

操作结果：在图 G 中删除顶点 v 和与 v 相关的弧

算法思路：调用 LocateVex(G,u) 函数，找到该顶点，删除与该顶点相关的所

有边以及该顶点。图 2-4 为 LocateVex 的流程图

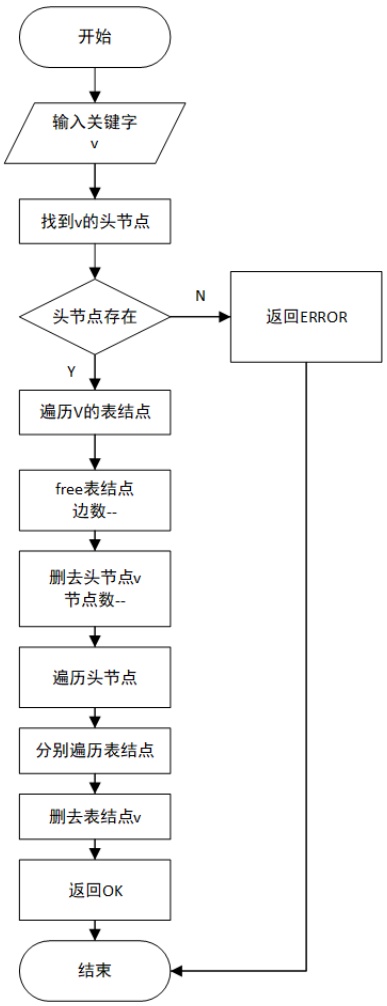

图 2-4　DeleteVex 流程图

10) 函数名称：InsertArc(G,v,w)

初始条件：图 G 存在，v、w 是 G 的顶点

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

算法思路：调用 LocateVex(G,u) 函数，找到相应顶点，增添对应的弧 <v,w> 以及 <w,v>。图 2-5 为 InsertArc 的流程图

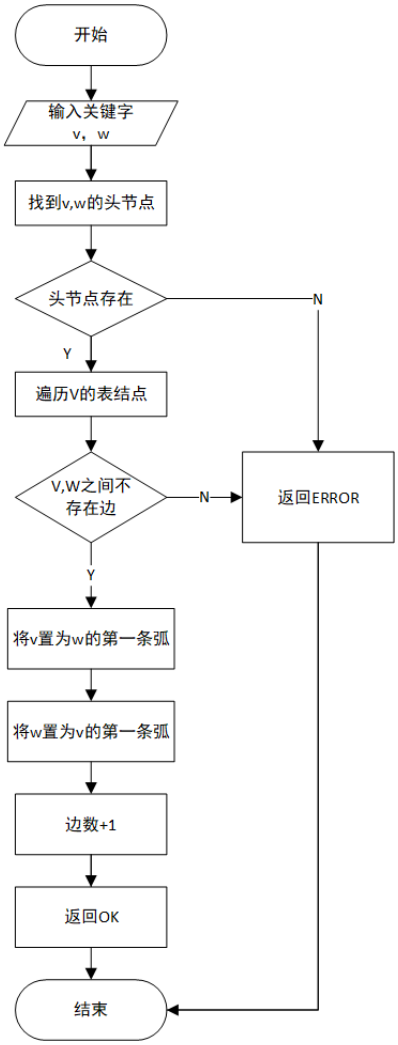

图 2-5　InsertArc 流程图

11) 函数名称：DeleteArc(G,v,w)

初始条件：是图 G 存在，v、w 是 G 的顶点

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

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

12) 函数名称：DFSTraverse(G,visit())

初始条件：图 G 存在

操作结果：进行深度优先搜索遍历，依次对图中的每一个顶点使用函数 visit 访问一次，且仅访问一次

算法思路：首先，访问出发顶点 v 并作访问标记；然后，依次从 v 的未访问过的邻接顶点 w 出发，进行深度优先遍历。图 2-6 为 DFSTraverse 流程图

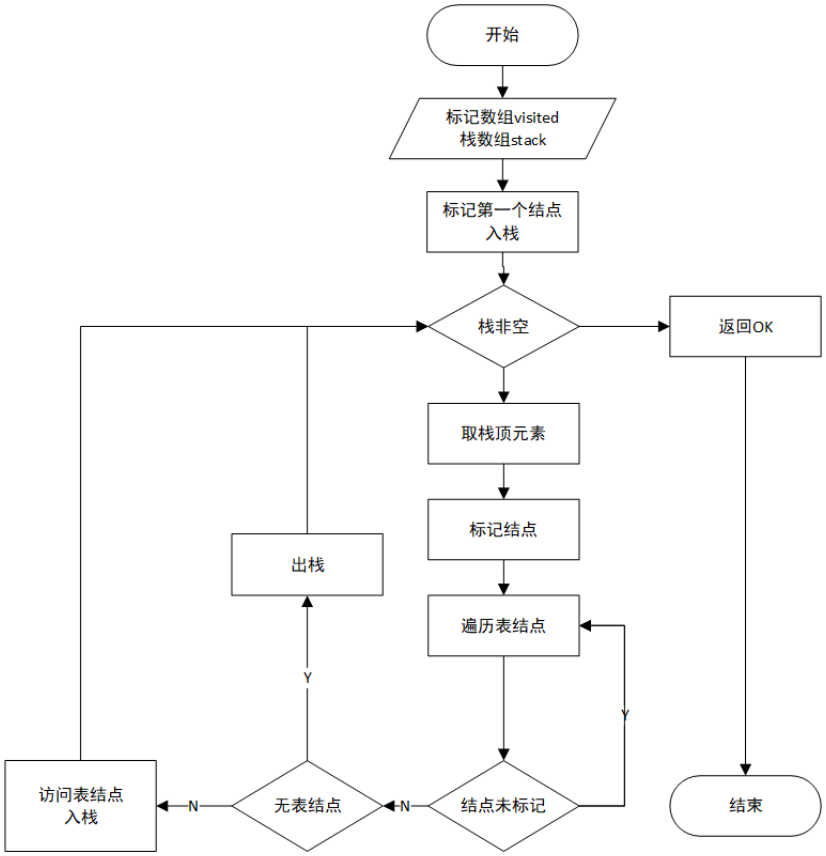

图 2-6　DFSTraverse 流程图

13) 函数名称：BFSTraverse(G,visit())

初始条件：图 G 存在

操作结果：进行广度优先搜索遍历，依次对图中的每一个顶点使用函数 visit 访问一次，且仅访问一次

算法思路：首先，访问出发顶点 v 并作访问标记；然后，依次访问 v 的所有未访问过的邻接顶点 w，再依次广度优先遍历 w 的所有未访问过的邻接顶点，直到出发顶点 v 的所有可达顶点均被访问过为止。图 2-7 为 BFSTraverse 流程图

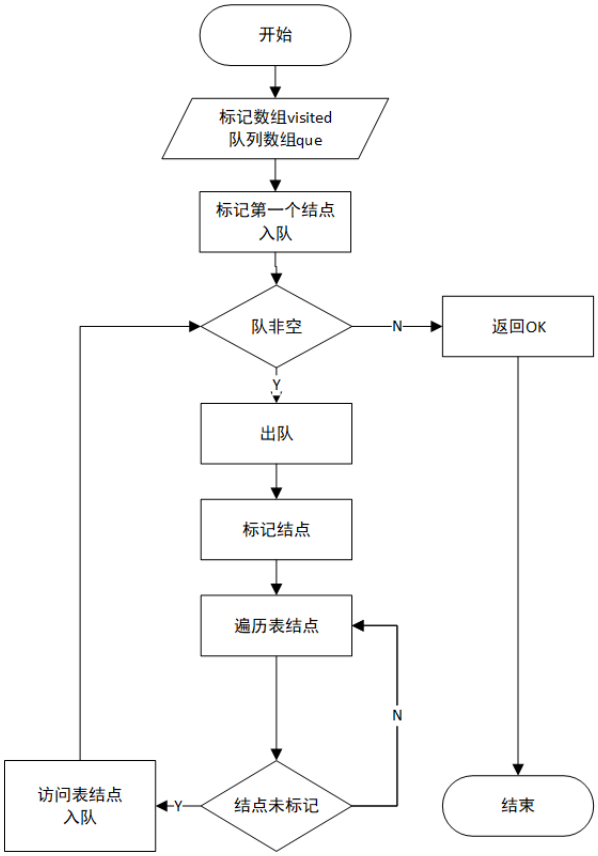

图 2-7　BFSTraverse 流程图

2.3　系统实现

2.3.1　程序源代码

见《附录 D 基于邻接表图实现的源程序》。

2.4　系统测试

本系统界面简洁舒适，如图 4-1 所示，挑选 PutVex,NextAdjVex, DeleteVex, InsertArc, DeleteArc, DFSTraverse, BFSTraverse 这些重要功能进行测试。

测试用例以文件形式输入，以下是测试用例 t1。

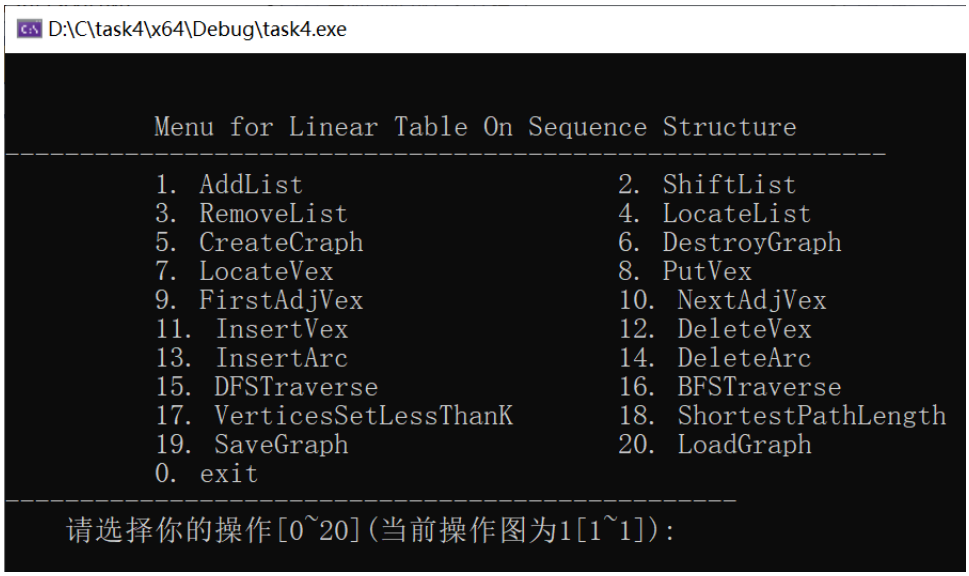

图 2-8　界面视图

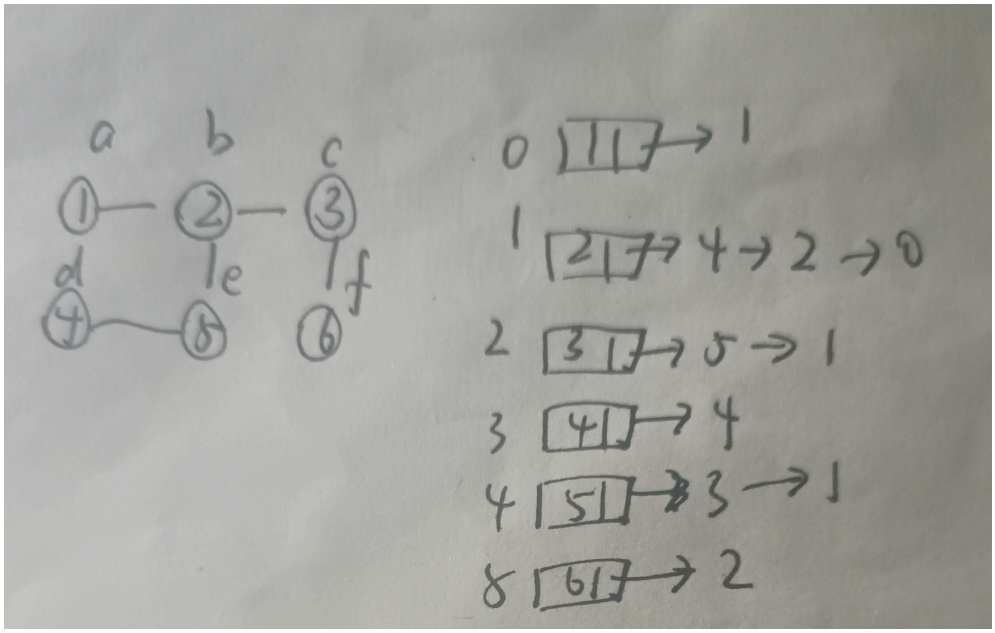

图 2-9　测试用例 t1

1) PutVex 测试

测试用例	输入	理论结果	测试结果
t1	7 2 11 ab	赋值成功!	赋值成功!
图不存在	7	图不存在!	图不存在!

表 2-1　PutVex 测试用例表

图 2-10　PutVex 测试截图

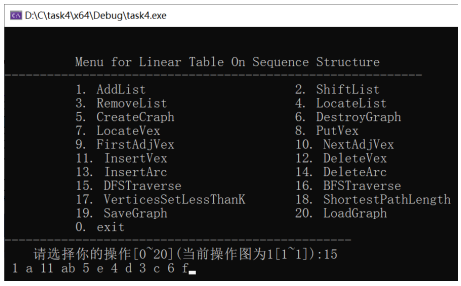

图 2-11　通过深度遍历验证

2) NextAdjVex 测试

测试用例	输入	理论结果	测试结果
t1	10 2 3	下一邻接点的位序为 0	下一邻接点的位序为 0
t1	10 1 2	不存在!	不存在!
图不存在	10	图不存在!	图不存在!

表 2-2　NextAdjVex 测试用例表

图 2-12　PutVex 测试截图 1

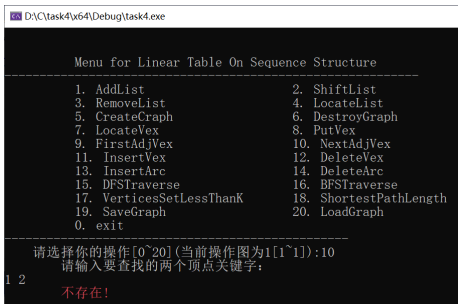

图 2-13　PutVex 测试截图 2

3) DeleteVex 测试

测试用例	输入	理论结果	测试结果
t1	12 4	删除成功!	删除成功!
t1	12 8	删除失败!	删除失败!
图不存在	12	图不存在!	图不存在!

表 2-3　DeleteVex 测试用例表

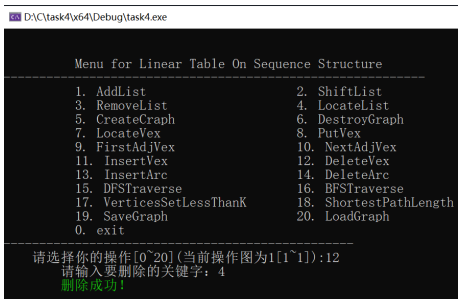

图 2-14　DeleteVex 测试截图 1

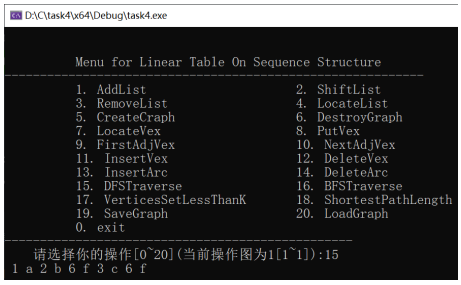

图 2-15　通过深度遍历验证

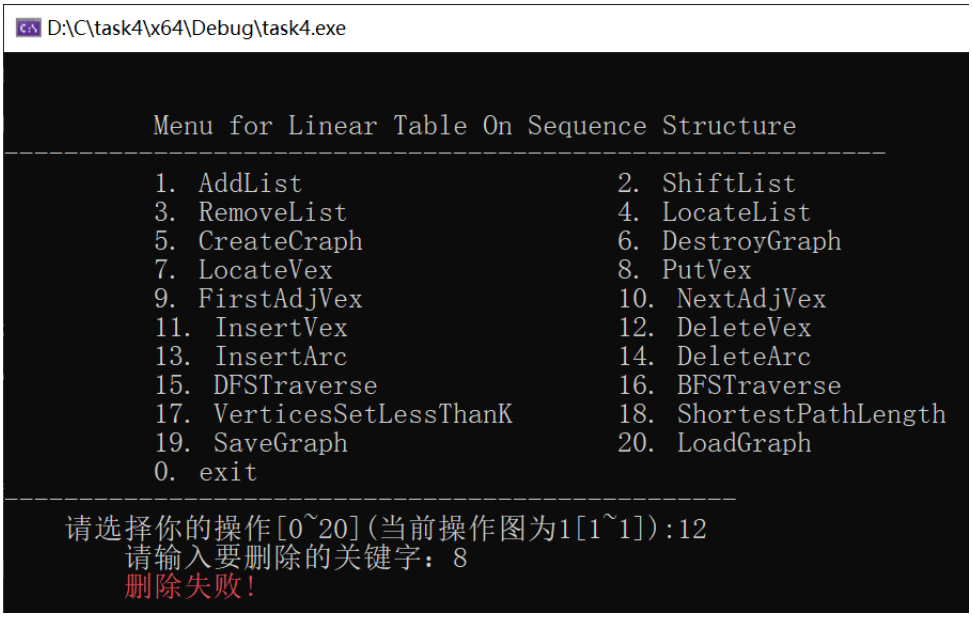

图 2-16　DeleteVex 测试截图 2

4) InsertArc 测试

测试用例	输入	理论结果	测试结果
t1	13 1 4	插入成功!	插入成功!
t1	13 1 2	插入失败!	插入失败!
图不存在	13	图不存在!	图不存在!

表 2-4　InsertArc 测试用例表

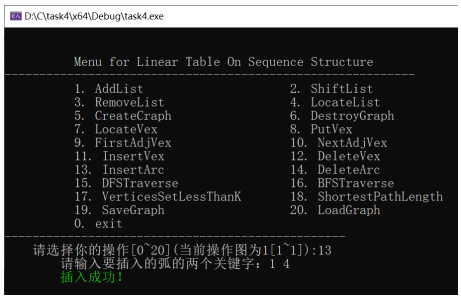

图 2-17　InsertArc 测试截图 1

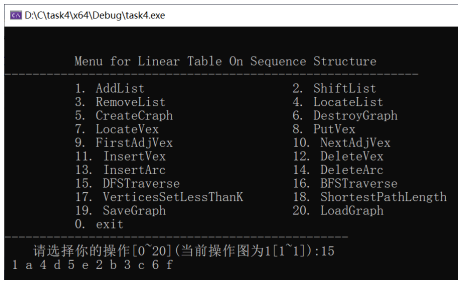

图 2-18　通过深度遍历验证

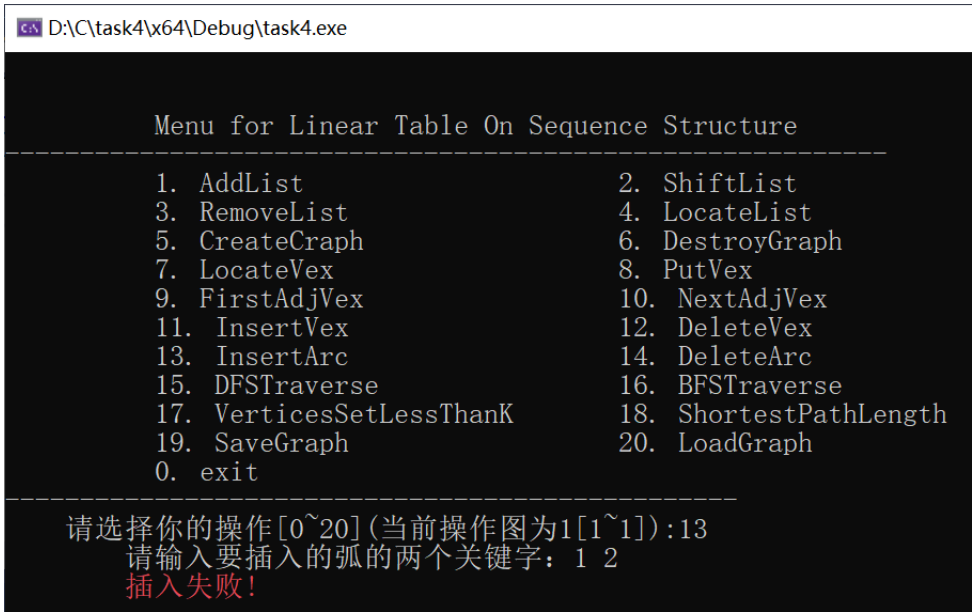

图 2-19　InsertArc 测试截图 2

5) DFSTraverse 测试

测试用例	输入	理论结果	测试结果
t1	15	1 a 2 b 5 e 4 d 3 c 6 f	1 a 2 b 5 e 4 d 3 c 6 f
图不存在	15	图不存在!	图不存在!

表 2-5　DFSTraverse 测试用例表

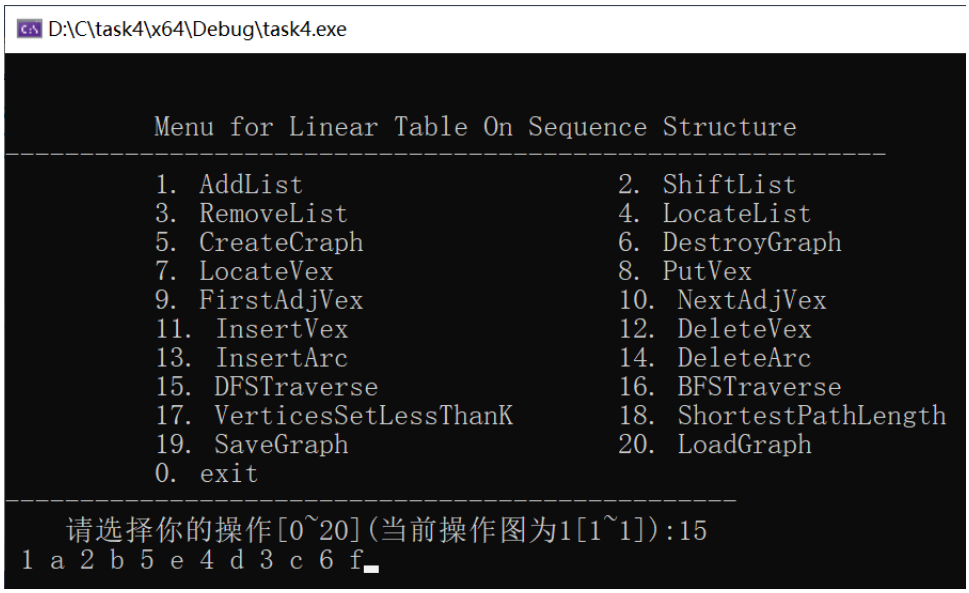

图 2-20　DFSTraverse 测试截图

6) BFSTraverse 测试

测试用例	输入	理论结果	测试结果
t1	16	1 a 2 b 5 e 3 c 4 d 6 f	1 a 2 b 5 e 3 c 4 d 6 f
图不存在	16	图不存在!	图不存在!

表 2-6　BFSTraverse 测试用例表

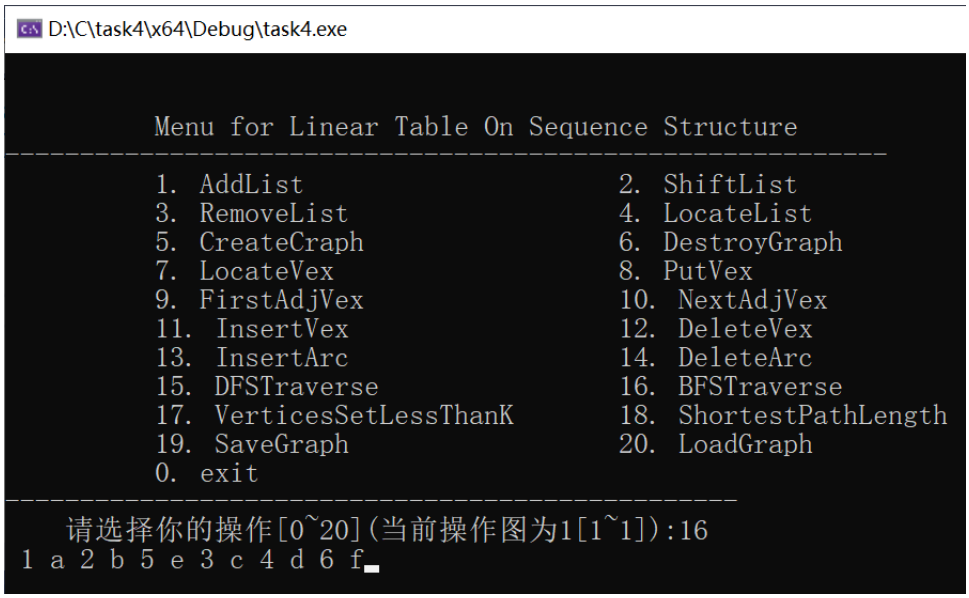

图 2-21　BFSTraverse 测试截图

2.5 实验小结

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

在写实验四时，我对表的深度优先遍历和广度优先遍历有了更深刻的理解，将自己的代码变得更加完善了。还有对图的输入，所有可能的图的情况都进行了研究，使代码能够包含各种情况。

实验四也花费了我很长时间，因为临近考试，时间很紧张，但是也使我收获颇丰。短暂的数据结构实验课程告一段落，期待寒假数据结构的课设题目，愿我可以在 0 和 1 的世界里找到乐趣。

3 课程的收获和建议

收获：经过一个学期的学习，我基本上熟练掌握了对线性表，树和图的各种操作。更重要的是，我学会了构建一个完备的系统，能让操作者很容易知道如何使用，并且能处理各种错误的输入。

建议：附录 B 文件读写的示例，我没看懂是什么意思。最后文件读写和多表储存都是我自己花了很长时间研究出来的。以后上课可以讲讲这部分的实现方法。

4 附录 A 基于顺序存储结构线性表实现的源程序

```
#include"def.h"
int  pre, next, sta, e;
int n;
SqList L;
LISTS Lists;
#include"fun.h"

int main()
{
    int op = 1;
    Lists.length = 0;
    L.elem = NULL;
    while (op)
    {
        system("cls");  printf("\n\n");
        printf("      Menu for Linear Table On Sequence Structure \n")
        printf("-------------------------------------------\n");
        printf("          1. AddList        11. PriorElem\n");
        printf("          2. ShiftList      12. NextElem\n");
        printf("          3. RemoveList     13. ListInsert\n");
        printf("          4. LocateList     14. ListDelete\n");
        printf("          5. DestroyList    15. MaxSubArray\n");
        printf("          6. ClearList      16. SubArrayNum\n");
        printf("          7. ListEmpty      17. sortList\n");
        printf("          8. ListLength     18. ListTraverse\n");
        printf("          9. GetElem        19. SaveList\n");
        printf("          10. LocateElem    20. LoadList\n");
        printf("          0. Exit           21. InitList\n");
```

```
printf("------------------------------------------------\n");
printf("     请选择你的操作[0~12](当前操作的线性表为%d[1~%d]):"
scanf("%d", &op);
switch (op)
{
case 1://AddList
        sta = AddList(Lists);
        if (sta == ERROR)
                printf_red("     添加失败, 线性表数量已达上限! "
        else
                printf_green("添加成功! \n");
        getchar(); getchar();
        break;
case 2://ShiftList
        sta = ShiftList();
        if (sta == ERROR)
                printf_red("     该线性表不存在! \n");
        else
                printf_green("   转换成功! \n");
        getchar(); getchar();
        break;
case 3://RemoveList
        sta = RemoveList(Lists);
        if (sta == OK)
                printf_green("   删除成功! \n");
        else
                printf_red("     未找到该线性表!\n");
        getchar(); getchar();
        break;
case 4://LocateList
        sta = LocateList(Lists);
```

```
            if (sta == 0)
                    printf_red("     未找到该线性表!\n");
            else
            {
                    printf("         这是第%d个线性表，内容如下：\n
                    printf("%s ", Lists.elem[sta - 1].name);
                    ListTraverse(Lists.elem[sta - 1].L);
                    putchar('\n');
            }
            getchar(); getchar();
            break;
    case 5://DestroyList
            sta = DestroyList(Lists.elem[n].L);
            if (sta == OK)
                    printf_green("   线性表已销毁!\n");
            else
                    printf_red("     线性表不存在! \n");
            getchar(); getchar();
            break;
    case 6://ClearList
            sta = ClearList(Lists.elem[n].L);
            if (sta == OK)
                    printf_green("   线性表已清空!\n");
            else
                    printf_red("     线性表不存在! \n");
            getchar(); getchar();
            break;
    case 7://ListEmpty
            sta = ListEmpty(Lists.elem[n].L);
            if (sta == INFEASIBLE)
                    printf_red("     线性表不存在!\n");
```

```
            else if (sta == TRUE)
                    printf_green("     线性表为空！\n");
            else
                    printf_yellow("     线性表不为空！\n");
            getchar(); getchar();
            break;
    case 8://ListLength
            sta = ListLength(Lists.elem[n].L);
            if (sta == INFEASIBLE)
                    printf_red("     线性表不存在!\n");
            else
                    printf("         线性表的长度为 %d", ListLength
            getchar(); getchar();
            break;
    case 9://GetElem
            e = 0;
            sta = GetElem(Lists.elem[n].L, e);
            if (sta == INFEASIBLE)
                    printf_red("     线性表不存在!\n");
            else if (sta == ERROR)
                    printf_red("     不存在该元素!\n");
            else
                    printf("这个元素是 %d\n", e);
            getchar(); getchar();
            break;
    case 10://LocateElem
            sta = LocateElem(Lists.elem[n].L);
            if (sta == INFEASIBLE)
                    printf_red("     线性表不存在!\n");
            else if (sta == ERROR)
                    printf_yellow(" 没有找到该元素!\n");
```

```
            else
                    printf("          这是第 %d 个元素\n", sta);
            getchar(); getchar();
            break;
        case 11://PriorElem
            pre = 0;
            sta = PriorElem(Lists.elem[n].L, pre);
            if (sta == INFEASIBLE)
                    printf_red("     线性表不存在!\n");
            else if (sta == ERROR)
                    printf_red("     没有找到前驱!\n");
            else
                    printf("          该元素的前驱为 %d \n", pre);
            getchar(); getchar();
            break;
        case 12://NextElem
            next = 0;
            sta = NextElem(Lists.elem[n].L, next);
            if (sta == INFEASIBLE)
                    printf_red("     线性表不存在!\n");
            else if (sta == ERROR)
                    printf_red("     没有找到后继!\n");
            else
                    printf("          该元素的后继为 %d \n", next);
            getchar(); getchar();
            break;
        case 13://ListInsert
            sta = ListInsert(Lists.elem[n].L);
            if (sta == INFEASIBLE)
                    printf_red("     线性表不存在!\n");
            else if (sta == ERROR)
```

```
                printf_yellow(" 插入位置不正确!\n");
            else
                printf_green("  插入成功!\n");
            getchar(); getchar();
            break;
        case 14://ListDelete
            sta = ListDelete(Lists.elem[n].L, e);
            if (sta == INFEASIBLE)
                printf_red("    线性表不存在!\n");
            else if (sta == ERROR)
                printf_yellow(" 删除位置不正确!\n");
            else
            {
                printf_green("  删除成功!\n");
                printf("        你删除的元素是 %d !\n", e);
            }
            getchar(); getchar();
            break;
        case 15://MaxSubArray
            sta = MaxSubArray(Lists.elem[n].L);
            if (sta == INFEASIBLE)
                printf_red("    线性表不存在!\n");
            else if (sta == ERROR)
                printf_yellow("    线性表为空! \n");
            else
                printf("    该数组最大连续子数组和为 %d \n", s
            getchar(); getchar();
            break;
        case 16://SubArrayNum
            sta = SubArrayNum(Lists.elem[n].L);
            if (sta == INFEASIBLE)
```

```
                printf_red("    线性表不存在!\n");
            else if (sta == 0)
                printf_yellow("    没有这样的子数组! \n");
            else
                printf("    这样的子数组个数为 %d \n", sta);
            getchar(); getchar();
            break;
        case 17://sortList
            sta = sortList(Lists.elem[n].L);
            if (sta == INFEASIBLE)
                printf_red("    线性表不存在!\n");
            else if (sta == ERROR)
                printf_yellow("    线性表为空! \n");
            else
                printf("    从小到大排序成功! \n");
            getchar(); getchar();
            break;
        case 18://ListTraverse
            printf("\n-----------all elements --------------------
            for (int i = 0; i < Lists.length; i++)
            {
                printf("%d.", i + 1);
                printf("%s ", Lists.elem[i].name);
                sta = ListTraverse(Lists.elem[i].L);
                if (sta == INFEASIBLE)
                    printf_red("该线性表不存在! \n");
                putchar('\n');
            }
            printf("\n------------------ end ---------------------
            getchar(); getchar();
            break;
```

```
case 19://SaveList
{
        char filename[100] = { 0 };
        char name[20] = { 0 };
        printf("          请输入要写入的文件: ");
        scanf("%s", name);
        sprintf(filename, "%s%s%s", "./", name, ".txt");
        sta = SaveList(Lists.elem[n].L, filename);
        if (sta == OK)
                printf_green("   写入成功! \n");
        else
                printf_red("    写入失败! \n");
        getchar(); getchar();
        break;
}
case 20://LoadList
{
        char filename[100] = { 0 };
        char name[20] = { 0 };
        printf("          请输入要读的文件: ");
        scanf("%s", name);
        sprintf(filename, "%s%s%s", "./", name, ".txt");
        sta = LoadList(Lists.elem[n].L, filename);
        if (sta == OK)
                printf_green("   读入成功! \n");
        else
                printf_red("    写入失败! \n");
        getchar(); getchar();
        break;
}
case 21://InitList
```

```
                if (InitList(Lists.elem[n].L) == OK)
                        printf_green("   线性表创建成功！\n");
                else
                        printf_red("     线性表创建失败！\n");
                getchar(); getchar();
                        break;
                case 0://exit
                        break;
                }//end of switch
        }//end of while
        printf("欢迎下次再使用本系统！\n");
}//end of main()
```

/*———”fun.h”———*/

```
#pragma once

status InitList(SqList& L) //1
{
        L.elem = (ElemType*)malloc(LIST_INIT_SIZE * sizeof(ElemType));
        if (!L.elem) exit(OVERFLOW);
        L.length = 0;
        L.listsize = LIST_INIT_SIZE;
        return OK;
}

status DestroyList(SqList& L)//2
{
        if (!L.elem)
                return INFEASIBLE;
        free(L.elem);
        L.elem = NULL;
```

```
        L.length = 0;
        L.listsize = 0;
        return OK;
}

status ClearList(SqList& L)//3
{
        if (!L.elem)
                return INFEASIBLE;
        L.length = 0;
        L.listsize = 0;
        return OK;
}

status ListEmpty(SqList L)//4
{
        if (!L.elem)
                return INFEASIBLE;
        if (L.length == 0)
                return TRUE;
        else
                return FALSE;
}

status ListLength(SqList L)//5
{
        if (!L.elem)
                return INFEASIBLE;
        return L.length;
}
```

```
status GetElem(SqList L, ElemType& e)//6
{
        int i = 0;
        printf("          请输入您要查找的元素的位次？\n");
        scanf("%d", &i);
        if (!L.elem)
                return INFEASIBLE;
        if (i > L.length || i < 1)
                return ERROR;
        e = L.elem[i - 1];
        return OK;
}

int LocateElem(SqList L)//7
{
        int e = 0;
        printf("          请输入您要查找的元素？\n");
        scanf("%d", &e);
        if (!L.elem)
                return INFEASIBLE;
        for (int i = 0; i < L.length; i++)
                if (e == L.elem[i])
                        return i + 1;
        return 0;
}

status PriorElem(SqList L, ElemType& pre)//8
{
        if (!L.elem)
                return INFEASIBLE;
        int e = 0;
```

```
        printf("          请输入要查找的元素\n");
        scanf("%d", &e);
        for (int i = 0; i < L.length; i++)
        {
                if (e == L.elem[i])
                {
                        if (i == 0 || i > L.length)
                                return ERROR;
                        pre = L.elem[i - 1];
                        return OK;
                }
        }
        return ERROR;
}

status NextElem(SqList L, ElemType& next)//9
{
        if (!L.elem)
                return INFEASIBLE;
        int e;
        printf("          请输入要查找的元素： ");
        scanf("%d", &e);
        for (int i = 0; i < L.length; i++)
        {
                if (e == L.elem[i])
                {
                        if (i > L.length - 2)
                                return ERROR;
                        next = L.elem[i + 1];
                        return OK;
                }
```

```
        }
        return ERROR;
}

status ListInsert(SqList& L)//10
{
        if (!L.elem)
                return INFEASIBLE;
        int e = 0, i = 0;
        printf("          请输入你要插入的元素\n");
        scanf("%d", &e);
        printf("          请输入你要插入的位次序号\n");
        scanf("%d", &i);
        if (i<1 || i>L.length + 1)
                return ERROR;
        if (L.length >= L.listsize)
        {
                L.elem = (ElemType*)realloc(L.elem, (L.listsize + LIST
                L.listsize += LIST_INIT_SIZE;
        }
        int j;
        if (L.length == 0)
        {
                L.length++;
                L.elem[0] = e;
                return OK;
        }
        for (j = L.length; j >= i; j--)
                L.elem[j] = L.elem[j - 1];
        L.elem[i - 1] = e;
        L.length += 1;
```

```
        return OK;
}

status ListDelete(SqList& L, ElemType& e)//11
{
        int i = 0;
        printf("          请输入你要删除的元素的位次序号\n");
        scanf("%d", &i);
        if (!L.elem)
                return INFEASIBLE;
        if (i<1 || i>L.length)
                return ERROR;
        e = L.elem[i - 1];
        for (int j = i - 1; j < L.length; j++)
                L.elem[j] = L.elem[j + 1];
        L.length--;
        return OK;
}

status MaxSubArray(SqList L)
{
        if (!L.elem)
                return INFEASIBLE;
        if (!L.length)
                return ERROR;
        int max_sum = 0, now_sum;
        for (int i = 0; i < L.length; i++)
        {
                now_sum = 0;
                for (int j = i; j < L.length; j++)
                {
```

```
                        now_sum += L.elem[j];
                        if (now_sum > max_sum)
                                max_sum = now_sum;
                }
        }
        return max_sum;
}

status SubArrayNum(SqList L)
{
        printf("          请输入您想要查找的连续子数组的和：");
        int e = 0;
        scanf("%d", &e);
        if (!L.elem)
                return INFEASIBLE;
        if (!L.length)
                return ERROR;
        int  now_sum, count = 0;
        for (int i = 0; i < L.length; i++)
        {
                now_sum = 0;
                for (int j = i; j < L.length; j++)
                {
                        now_sum += L.elem[j];
                        if (now_sum == e)
                                count++;
                }
        }
        return count;
}
```

```
status sortList(SqList& L)
{
        if (!L.elem)
                return INFEASIBLE;
        if (!L.length)
                return ERROR;
        qsort(L.elem, L.length, sizeof(ElemType), compare);
        return OK;
}

status ListTraverse(SqList L)//12
{
        if (!L.elem)
                return INFEASIBLE;
        int i;
        if (L.length == 0)
        {
                printf_yellow(" 线性表为空！\n");
                return OK;
        }
        for (i = 0; i < L.length - 1; i++)
                printf("%d ", L.elem[i]);
        printf("%d", L.elem[i]);
        return OK;
}

status  SaveList(SqList L, char FileName[])//13
// 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
{
        // 请在这里补充代码，完成本关任务
        /********** Begin *********/
```

```
        if (!L.elem)
                return INFEASIBLE;
        FILE* w = fopen(FileName, "w");
        if (!w)
                return ERROR;
        for (int i = 0; i < L.length; i++)
        {
                fprintf(w, "%d ", L.elem[i]);
        }
        fclose(w);
        return OK;
        /********** End **********/
}
status  LoadList(SqList& L, char FileName[])//14
{
        int t, i = 0;
        FILE* r = fopen(FileName, "r");
        if (!r)
                return ERROR;
        while (fscanf(r, "%d", &t) != EOF)
        {
                L.elem[i++] = t;
        }
        L.length = i;
        fclose(r);
        return OK;
}

status AddList(LISTS& Lists)//15
{
        if (Lists.length >= 10)
```

```
                return ERROR;
        char ListName[100];
        printf("请输入您想添加的线性表的名称：");
        scanf("%s", ListName);
        if (strlen(ListName) >= 20)
        {
                printf_red("名字过长！\n");
                return ERROR;
        }
        ElemType i;
        for (i = 0; ListName[i]; i++)
        {
                Lists.elem[Lists.length].name[i] = ListName[i];
        }
        Lists.elem[Lists.length].name[i] = ListName[i];
        Lists.elem[Lists.length].L.elem = NULL;
        InitList(Lists.elem[Lists.length].L);
        Lists.length++;
        return OK;
}

status RemoveList(LISTS& Lists)//16
{
        char ListName[20];
        printf("请输入您要删除的线性表的名称：");
        scanf("%s", ListName);
        int i, j;
        for (i = 0, j = 0; i < Lists.length; i++)
        {
                if (strcmp(ListName, Lists.elem[i].name) == 0)
                {
```

```
                        DestroyList(Lists.elem[i].L);
                        for (int j = i; j < Lists.length - 1; j++)
                        {
                                Lists.elem[j] = Lists.elem[j + 1];
                        }
                        Lists.length--;
                        return OK;
                }
        }
        return ERROR;
}

int LocateList(LISTS Lists)
{
        char ListName[20];
        printf("请输入您要查找的线性表的名称：");
        scanf("%s", ListName);
        int i, j, f = 1;
        for (i = 0; i < Lists.length; i++)
        {
                if (strcmp(ListName, Lists.elem[i].name) == 0)
                        return i + 1;
        }
        return 0;
}

status ShiftList()
{
        int i;
        printf("请输入您要操作的线性表的序号：");
        scanf("%d", &i);
```

```
        if (i > Lists.length)
                return ERROR;
        n = i - 1;
        return OK;
}

int compare(const void* a, const void* b)
{
        return *(int*)b - *(int*)a;
}

void printf_red(const char* s)
{
        printf("\033[1;31m%s\033[0m", s);
}

void printf_green(const char* s)
{
        printf("\033[1;32m%s\033[0m", s);
}

void printf_yellow(const char* s)
{
        printf("\033[1;33m%s\033[0m", s);
}
```

/*———”def.h” ———*/

```
#pragma once
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
```

```
#include <string.h>
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2

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

#define LIST_INIT_SIZE 100
#define LISTINCREMENT  10

typedef struct {  //顺序表（顺序结构）的定义
    ElemType* elem;
    int length;
    int listsize;
}SqList;

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

//函数的定义
status InitList(SqList& L);
```

```
status DestroyList(SqList& L);
status ClearList(SqList& L);
status ListEmpty(SqList L);
int ListLength(SqList L);
status GetElem(SqList L, ElemType& e);
status LocateElem(SqList L); //简化过
status PriorElem(SqList L, ElemType& pre_e);
status NextElem(SqList L, ElemType& next_e);
status ListInsert(SqList& L);
status ListDelete(SqList& L, ElemType& e);
status MaxSubArray(SqList L);
status SubArrayNum(SqList L);
status sortList(SqList& L);
status RemoveList(LISTS& Lists);
status ShiftList();
status ListTraverse(SqList L);
status AddList(LISTS& Lists);
int LocateList(LISTS Lists);
status  SaveList(SqList L, char FileName[]);
status  LoadList(SqList& L, char FileName[]);

int compare(const void* a, const void* b);
/*-------------------------------------------*/

void printf_red(const char* s);
void printf_green(const char* s);
void printf_yellow(const char* s);
```

5 附录 B 基于链式存储结构线性表实现的源程序

```
//"def.h"
#pragma once
#define _CRT_SECURE_NO_WARNINGS

#include "stdio.h"
#include "stdlib.h"
#include "string.h"

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define LIST_INIT_SIZE 100
#define LISTINCREMENT  10

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

typedef struct LNode {  //单链表（链式结构）结点的定义
    ElemType data;
    struct LNode* next;
}LNode, * LinkList;

typedef struct {  //线性表的管理表定义
    struct {
```

```
        char name[30];
        LinkList L;
    } elem[10];
    int length;
    int listsize;
}LISTS;

status InitList(LinkList&);//1
status DestroyList(LinkList&);//2
status ClearList(LinkList&);//3
status ListEmpty(LinkList);//4
int ListLength(LinkList);//5
status GetElem(LinkList, ElemType&);//6
status LocateElem(LinkList); //7
status PriorElem(LinkList, ElemType&);//8
status NextElem(LinkList, ElemType&);//9
status ListInsert(LinkList&);//10
status ListDelete(LinkList&, ElemType&);//11
status ListTraverse(LinkList);//12
status reverseList(LinkList&);//13
status RemoveNthFromEnd(LinkList&, ElemType);//14
status sortList(LinkList&);//15
status SaveList(LinkList);//16

int compare(const void* a, const void* b);
/*-----------------------------------------*/

void printf_red(const char* s);
void printf_green(const char* s);
void printf_yellow(const char* s);
```

```
//fun.h

#pragma once

extern int n;
extern LISTS Lists;

status InitList(LinkList& L)//1
{
        if (L)
                return INFEASIBLE;
        L = (LinkList)malloc(sizeof(LNode));
        L->next = NULL;
        return OK;
}

status DestroyList(LinkList& L)//2
{
        if (!L)
                return INFEASIBLE;
        LinkList p;
        while (L)
        {
                p = L->next;
                free(L);
                L = p;
        }
        return OK;
}
```

```
status ClearList(LinkList& L)//3
{
        if (!L)
                return INFEASIBLE;
        if (!L->next)
                return ERROR;
        LinkList p=L->next,q;
        while (p)
        {
                q = p->next;
                free(p);
                p = q;
        }
        L->next = NULL;
        return OK;
}

status ListEmpty(LinkList L)//4
{
        if (!L)
                return INFEASIBLE;
        if (L->next == NULL)
                return TRUE;
        else
                return FALSE;
}

int ListLength(LinkList L)//5
{
        if (!L)
                return INFEASIBLE;
```

```
        int count = 0;
        while (L->next)
        {
                count++;
                L = L->next;
        }
        return count;
}

status GetElem(LinkList L, ElemType& e)//6
{
        if (!L)
                return INFEASIBLE;
        int i = 0;
        printf("          请输入您要查找的元素的位次？\n");
        scanf("%d", &i);
        LinkList p = L->next;
        for (int j = 1; j < i && p; j++)
        {
                p = p->next;
        }
        if (i < 1 || !p)
                return ERROR;
        e = p->data;
        return OK;
}

status LocateElem(LinkList L)//7
{
        if (!L)
                return INFEASIBLE;
```

```
        int e = 0;
        printf("          请输入您要查找的元素？\n");
        scanf("%d", &e);
        LinkList p = L->next;
        for (int i = 1; p; i++)
        {
                if (p->data == e)
                        return i;
                p = p->next;
        }
        return ERROR;
}

status PriorElem(LinkList L,ElemType& pre)//8
{
        if (!L)
                return INFEASIBLE;
        int e = 0;
        printf("          请输入要查找的元素\n");
        scanf("%d", &e);
        LinkList p;
        if (L->next == NULL)
                return ERROR;
        p = L->next;
        for (; p->next && p->next->data != e; p = p->next);
        if (!L->next || !p->next)
                return ERROR;
        pre = p->data;
        return OK;
}
```

```
status NextElem(LinkList L, ElemType& next)//9
{
        if (!L)
                return INFEASIBLE;
        int e;
        printf("          请输入要查找的元素：");
        scanf("%d", &e);
        LinkList p;
        if (L->next == NULL)
                return ERROR;
        p = L;
        for (; p->next && p->data != e; p = p->next);
        if (!p->next)
                return ERROR;
        next = p->next->data;
        return OK;
}

status ListInsert(LinkList& L)//10
{
        if (!L)
                return INFEASIBLE;
        int e = 0, i = 0;
        printf("          请输入你要插入的元素\n");
        scanf("%d", &e);
        printf("          请输入你要插入的位次序号\n");
        scanf("%d", &i);
        LinkList p = L;
        for (int j = 0; j < i - 1 && p; j++, p = p->next);
        if (!p || i < 1)
                return ERROR;
```

```
        LinkList q = (LinkList)malloc(sizeof(LNode));
        q->data = e;
        q->next = p->next;
        p->next = q;
        return OK;
}

status ListDelete(LinkList& L, ElemType& e)//11
{
        if (!L)
                return INFEASIBLE;
        int i = 0;
        printf("          请输入你要删除的元素的位次序号\n");
        scanf("%d", &i);
        LinkList p = L;
        for (int j = 0; j < i - 1 && p; j++, p = p->next);
        if (!p->next || i < 1)
                return ERROR;
        LinkList q = p->next;
        e = q->data;
        p->next = q->next;
        free(q);
        return OK;
}

status ListTraverse(LinkList L)//12
{
        if (!L)
                return INFEASIBLE;
        LinkList p = L->next;
        while (p)
```

```
        {
                printf("%d ", p->data);
                p = p->next;
        }
        return OK;
}

status reverseList(LinkList& L)//13
{
        if (!L)
                return INFEASIBLE;
        if (!L->next)
                return ERROR;
        LinkList temp = NULL, new_head = NULL, p = L->next;
        while(p)
        {
                temp = p;
                p = p->next;
                temp->next = new_head;
                new_head = temp;
        }
        L->next = new_head;
        return OK;
}

status RemoveNthFromEnd(LinkList& L)//14
{
        if (!L)
                return INFEASIBLE;
        if (!L->next)
                return ERROR;
```

```
        printf("          您要删除的是该链表中倒数第几个节点？ ");
        int n;
        scanf("%d", &n);
        LinkList p, q;
        p = q = L;
        for (int i = 0; i < n; i++)
                q = q->next;
        while (q->next)
        {
                q = q->next;
                p = p->next;
        }
        p->next = p->next->next;
        return OK;
}

status sortList(LinkList& L)//15
{
        if (!L)
                return INFEASIBLE;
        if (!L->next)
                return ERROR;
        LinkList pre, p, q;
        p = L->next->next;
        L->next->next = NULL;
        while (p)
        {
                q = p->next;
                pre = L;
                while (pre->next && pre->next->data < p->data)
                        pre = pre->next;
```

```
                p->next = pre->next;
                pre->next = p;
                p = q;
        }
        return OK;
}

status SaveList(LinkList L)//16
{
        if (!L)
                return INFEASIBLE;
        char filename[20];
        printf("          您要写入的文件夹名称为：");
        scanf("%s", filename);
        LinkList p;
        p = L->next;
        FILE* w = fopen(filename, "w");
        while (p)
        {
                fprintf(w, "%d ", p->data);
                p = p->next;
        }
        fprintf(w, "\n");
        fclose(w);
        return OK;
}

status LoadList(LinkList& L)
{
        if (!L)
                return INFEASIBLE;
```

```
        if (L->next)
                return ERROR;
        char filename[20];
        printf("          您要读入的文件夹名称为：");
        scanf("%s", filename);
        LinkList p;
        L = (LinkList)malloc(sizeof(LNode));
        p = L;
        FILE* r = fopen(filename,"r");
        int t;
        while (fscanf(r, "%d", &t) != EOF)
        {
                p->next = (LinkList)malloc(sizeof(LNode));
                p = p->next;
                p->data = t;
        }
        p->next = NULL;
        fclose(r);
        return OK;
}

status AddList(LISTS& Lists)//15
{
        if (Lists.length >= 10)
                return ERROR;
        char ListName[100];
        printf("          请输入您想添加的线性表的名称：");
        scanf("%s", ListName);
        if (strlen(ListName) >= 20)
        {
                printf_red("名字过长！\n");
```

```
                return ERROR;
        }
        ElemType i;
        for (i = 0; ListName[i]; i++)
        {
                Lists.elem[Lists.length].name[i] = ListName[i];
        }
        Lists.elem[Lists.length].name[i] = ListName[i];
        InitList(Lists.elem[Lists.length].L);
        Lists.length++;
        return OK;
}

status ShiftList()//16
{
        int i;
        printf("          请输入您要操作的线性表的序号：");
        scanf("%d", &i);
        if (i > Lists.length)
                return ERROR;
        n = i - 1;
        return OK;
}

status RemoveList(LISTS& Lists)//17
{
        char ListName[20];
        printf("          请输入您要删除的线性表的名称：");
        scanf("%s", ListName);
        int i, j;
        for (i = 0, j = 0; i < Lists.length; i++)
```

```
        {
                if (strcmp(ListName, Lists.elem[i].name) == 0)
                {
                        DestroyList(Lists.elem[i].L);
                        for (int j = i; j < Lists.length - 1; j++)
                        {
                                Lists.elem[j] = Lists.elem[j + 1];
                        }
                        Lists.length--;
                        return OK;
                }
        }
        return ERROR;
}

int LocateList(LISTS Lists)//18
{
        char ListName[20];
        printf("          请输入您要查找的线性表的名称：");
        scanf("%s", ListName);
        int i, f = 1;
        for (i = 0; i < Lists.length; i++)
        {
                if (strcmp(ListName, Lists.elem[i].name) == 0)
                        return i + 1;
        }
        return 0;
}

//main.cpp
```

```
#include"def.h"
#include"fun.h"

int sta, n;
LISTS Lists;
LinkList L;
int main()
{
    int op = 1;
    Lists.length = 0;
    while (op)
    {
        system("cls");  printf("\n\n");
        printf(" Menu for Linear Table On Sequence Structure \n");
        printf("-------------------------------------\n");
        printf("     1. AddList          11. PriorElem\n");
        printf("     2. ShiftList        12. NextElem\n");
        printf("     3. RemoveList       13. ListInsert\n");
        printf("     4. LocateList       14. ListDelete\n");
        printf("     5. DestroyList      15. ListTraverse\n");
        printf("     6. ClearList        16. reverseList\n");
        printf("     7. ListEmpty        17. RemoveNthFromEnd\n");
        printf("     8. ListLength       18. sortList\n");
        printf("     9. GetElem          19. SaveList\n");
        printf("     10. LocateElem      20. LoadList\n");
        printf("     0. Exit             21. InitList\n");
        printf("-----------------------------------------------\n");
        printf("  请选择你的操作[0~12](当前操作线性表为%d):", n + 1);
        scanf("%d", &op);
        switch (op)
        {
```

```
case 1://AddList
    sta = AddList(Lists);
    if (sta == ERROR)
        printf_red("    添加失败，线性表数量已达上限！\n");
    else
        printf_green("  添加成功！\n");
    getchar(); getchar();
    break;
case 2://ShiftList
    sta = ShiftList();
    if (sta == ERROR)
        printf_red("    该线性表不存在！\n");
    else
        printf_green("  转换成功！\n");
    getchar(); getchar();
    break;
case 3://RemoveList
    sta = RemoveList(Lists);
    if (sta == OK)
        printf_green("  删除成功！\n");
    else
        printf_red("    未找到该线性表!\n");
    getchar(); getchar();
    break;
case 4://LocateList
    sta = LocateList(Lists);
    if (sta == 0)
        printf_red("    未找到该线性表!\n");
    else
    {
        printf("        这是第%d个线性表，内容如下：\n", sta);
```

```
            printf("%s ", Lists.elem[sta - 1].name);
            ListTraverse(Lists.elem[sta - 1].L);
            putchar('\n');
        }
        getchar(); getchar();
        break;
    case 5://DestroyList
            sta = DestroyList(Lists.elem[n].L);
            if (sta == OK)
                    printf_green("   线性表已销毁!\n");
            else
                    printf_red("    线性表不存在! \n");
            getchar(); getchar();
            break;
    case 6://ClearList
            sta = ClearList(Lists.elem[n].L);
            if (sta == OK)
                    printf_green("   线性表已清空!\n");
            else if (sta == ERROR)
                    printf_yellow(" 线性表已为空! \n");
            else
                    printf_red("    线性表不存在! \n");
            getchar(); getchar();
            break;
    case 7://ListEmpty
            sta = ListEmpty(Lists.elem[n].L);
            if (sta == INFEASIBLE)
                    printf_red("    线性表不存在!\n");
            else if (sta == TRUE)
                    printf_green("    线性表为空! \n");
            else
```

```
                printf_yellow("    线性表不为空！\n");
            getchar(); getchar();
            break;
        case 8://ListLength
            sta = ListLength(Lists.elem[n].L);
            if (sta == INFEASIBLE)
                printf_red("    线性表不存在!\n");
            else
                printf("        线性表的长度为 %d", ListLength(Lists.e
            getchar(); getchar();
            break;
        case 9://GetElem
        {
            ElemType e;
            sta = GetElem(Lists.elem[n].L, e);
            if (sta == INFEASIBLE)
                printf_red("    线性表不存在!\n");
            else if (sta == ERROR)
                printf_red("    不存在该元素!\n");
            else
                printf("这个元素是 %d\n", e);
            getchar(); getchar();
            break;
        }
        case 10://LocateElem
            sta = LocateElem(Lists.elem[n].L);
                if (sta == INFEASIBLE)
                        printf_red("    线性表不存在!\n");
                else if (sta == ERROR)
                        printf_yellow(" 没有找到该元素!\n");
                else
```

```
                printf("          这是第 %d 个元素\n", sta);
        getchar(); getchar();
        break;
case 11://PriorElem
{
    int pre = 0;
    sta = PriorElem(Lists.elem[n].L, pre);
    if (sta == INFEASIBLE)
        printf_red("    线性表不存在!\n");
    else if (sta == ERROR)
        printf_red("    没有找到前驱!\n");
    else
        printf("          该元素的前驱为 %d \n", pre);
    getchar(); getchar();
    break;
}
case 12://NextElem
{
        int next = 0;
        sta = NextElem(Lists.elem[n].L, next);
        if (sta == INFEASIBLE)
                printf_red("    线性表不存在!\n");
        else if (sta == ERROR)
                printf_red("    没有找到后继!\n");
        else
                printf("          该元素的后继为 %d \n", next);
        getchar(); getchar();
        break;
}
case 13://ListInsert
        sta = ListInsert(Lists.elem[n].L);
```

```
            if (sta == INFEASIBLE)
                    printf_red("    线性表不存在!\n");
            else if (sta == ERROR)
                    printf_yellow(" 插入位置不正确!\n");
            else
                    printf_green("  插入成功!\n");
            getchar(); getchar();
            break;
    case 14://ListDelete
    {
            int e;
            sta = ListDelete(Lists.elem[n].L,e);
            if (sta == INFEASIBLE)
                    printf_red("    线性表不存在!\n");
            else if (sta == ERROR)
                    printf_yellow(" 删除位置不正确!\n");
            else
            {
                    printf_green("  删除成功!\n");
                    printf("        你删除的元素是 %d !\n", e);
            }
            getchar(); getchar();
            break;
    }
    case 15://ListTraverse
            printf("\n-----------all elements --------------------
            for (int i = 0; i < Lists.length; i++)
            {
                    printf("%d.", i + 1);
                    printf("%s ", Lists.elem[i].name);
                    sta = ListTraverse(Lists.elem[i].L);
```

```
                if (sta == INFEASIBLE)
                        printf_red("该线性表不存在！\n");
                putchar('\n');
            }
            printf("\n------------------ end --------------------
            getchar(); getchar();
            break;
        case 16://reverseList
            sta = reverseList(Lists.elem[n].L);
            if (sta == INFEASIBLE)
                printf_red("    线性表不存在!\n");
            else if (sta == ERROR)
                printf_yellow("    线性表为空！\n");
            else
                printf_green("    翻转成功！\n");
            getchar(); getchar();
            break;
        case 17://RemoveNthFromEnd
            sta = RemoveNthFromEnd(Lists.elem[n].L);
            if (sta == INFEASIBLE)
                printf_red("    线性表不存在!\n");
            else if (sta == ERROR)
                printf_yellow("    线性表为空！\n");
            else
                printf_green("    删除成功！\n");
            getchar(); getchar();
            break;
        case 18://sortList
            sta = sortList(Lists.elem[n].L);
            if (sta == INFEASIBLE)
                printf_red("    线性表不存在!\n");
```

```
            else if (sta == ERROR)
                    printf_yellow("    线性表为空！\n");
            else
                    printf_green("    从小到大排序成功！\n");
            getchar(); getchar();
            break;
    case 19://SaveList
            sta = SaveList(Lists.elem[n].L);
            if (sta == INFEASIBLE)
                    printf_red("    线性表不存在!\n");
            else
                    printf_green("  写入成功！\n");
            getchar(); getchar();
            break;
    case 20://LoadList
            sta = LoadList(Lists.elem[n].L);
            if (sta == INFEASIBLE)
                    printf_red("    线性表不存在!\n");
            else if (sta == ERROR)
                    printf_red("    线性表已有数据！\n");
            else
                    printf_green("  读入成功！\n");
            getchar(); getchar();
            break;
    case 21://InitList
            if (InitList(Lists.elem[n].L) == OK)
                    printf_green("  线性表创建成功！\n");
            else
                    printf_red("    线性表创建失败！\n");
            getchar(); getchar();
            break;
```

```
        case 0://exit
                break;
        }//end of switch
    }//end of while
    printf("欢迎下次再使用本系统！\n");
}//end of main()


int compare(const void* a, const void* b)
{
        return *(int*)b - *(int*)a;
}

void printf_red(const char* s)
{
        printf("\033[1;31m%s\033[0m", s);
}

void printf_green(const char* s)
{
        printf("\033[1;32m%s\033[0m", s);
}

void printf_yellow(const char* s)
{
        printf("\033[1;33m%s\033[0m", s);
}
```

6 附录 C 基于二叉链表二叉树实现的源程序

```
//def.h
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2

typedef int status;
typedef int KeyType;
typedef struct {
        KeyType  key;
        char others[20];
} TElemType; //二叉树结点类型定义


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

typedef struct {  //线性表的管理表定义
    struct {
```

```
        char name[30];
        BiTree L;
    } elem[10];
    int length;
    int listsize;
}LISTS;

BiTNode* LocateNode(BiTree T, KeyType e);
BiTNode* LocateNodep(BiTree T, KeyType e);
//fun.h
#pragma once
void PT(BiTree T,int a[])
{
    if (T == NULL)
        return;
    a[T->data.key]++;
    PT(T->lchild,a);
    PT(T->rchild,a);
}

status CreateBiTree(BiTree& T, TElemType definition[])
{
    int a[9999] = { 0 };
    for (int k = 0; definition[k].key != -1; k++)
    {
        if (definition[k].key && a[definition[k].key] > 0)
            return ERROR;
        a[definition[k].key]++;
    }
    static int i = -1;
    i++;
```

```
    if (definition[i].key == 0)
        T = NULL;
    else
    {
        T = (BiTree)malloc(sizeof(BiTNode));
        T->data.key = definition[i].key; //生成根结点
        strcpy(T->data.others, definition[i].others); //生成根结点
        CreateBiTree(T->lchild, definition); //递归构造左子树
        CreateBiTree(T->rchild, definition); //递归构造右子树
    }
    return OK;
}
int compare(const void* a, const void* b)
{
    return *(int*)b - *(int*)a;
}
void printf_red(const char* s)
{
    printf("\033[1;31m%s\033[0m", s);
}
void printf_green(const char* s)
{
    printf("\033[1;32m%s\033[0m", s);
}
void printf_yellow(const char* s)
{
    printf("\033[1;33m%s\033[0m", s);
}
BiTNode* LocateNodep(BiTree T, KeyType e)
{
    if (!T)
```

```
        return NULL;
    if (T->lchild != NULL && T->lchild->data.key == e || T->rchild != 
        return T;
    BiTree temp;
    temp = LocateNodep(T->lchild, e);
    if (temp)
        return temp;
    else
        return LocateNodep(T->rchild, e);
}


/*--------------------------------------------------------*/
void DestroyBiTree(BiTree& T)
{
    if (T == NULL)
        return;
    DestroyBiTree(T->lchild);
    DestroyBiTree(T->rchild);
    free(T);
    T = NULL;
}
void ClearBiTree(BiTree& T)
{
    if (T == NULL)
        return;
    ClearBiTree(T->lchild);
    ClearBiTree(T->rchild);
    T = NULL;

}
```

```
status BiTreeEmpty(BiTree T)
{
    if (T == NULL)
        return OK;
    else
        return ERROR;
}
int BiTreeDepth(BiTree T)
{
    int lH = 0, rH = 0, maxH = 0;//左子树，右子树，最大深度
    if (T == NULL) //如果为空树
        return 0;
    else
    {
        lH = BiTreeDepth(T->lchild);//左子树深度
        rH = BiTreeDepth(T->rchild);//右子树深度
        maxH = lH > rH ? lH : rH;
        return maxH + 1;//二叉树深度=最大深度+1
    }
}
BiTNode* LocateNode(BiTree T, KeyType e)
{
    BiTree tT;
    if (T == NULL)
        return NULL;
    if (T->data.key == e)
        return T;
    tT = LocateNode(T->lchild, e);
    if (tT)
        return tT;
    tT = LocateNode(T->rchild, e);
```

```
    if (tT)
        return tT;
}
status Assign(BiTree& T)
{
    KeyType e;
    TElemType value;
    BiTree t = NULL;
    printf("请输入您要替换的关键字：");
    scanf("%d", &e);
    printf("请输入新的关键字与字符串：");
    scanf("%d %s", &value.key, value.others);
    t = LocateNode(T, e);
    int a[9999] = { 0 };
    PT(T, a);
    if(a[value.key])
        return ERROR;
    t->data.key = value.key;
    strcpy(t->data.others, value.others);
    return OK;
}
BiTNode* GetSibling(BiTree T, KeyType e)
{
    BiTNode* p=NULL, * q=NULL;
    p = LocateNodep(T, e);
    q = LocateNode(T, e);
    if (p == NULL || q == NULL)
        return NULL;
    if (q == p->lchild && p->rchild != NULL)
        return p->rchild;
    if(q == p->rchild && p->lchild != NULL)
```

```
        return p->lchild;
    return NULL;
}
status InsertNode(BiTree& T, KeyType e, int LR, TElemType c)
{
    if (c.key <= e)
        return ERROR;
    BiTree t=NULL;
    t = LocateNode(T, e);
    BiTree nw;
    nw = (BiTree)malloc(sizeof(BiTNode));
    nw->data.key = c.key;
    strcpy(nw->data.others, c.others);
    if (LR == 0)
    {
        nw->rchild = t->lchild;
        t->lchild = nw;
        nw->lchild = NULL;
    }
    else if (LR == 1)
    {
        nw->rchild = t->rchild;
        t->rchild = nw;
        nw->rchild = NULL;
    }
    else if (LR == -1)
    {
        nw->rchild = T;
        T = nw;
    }
    return OK;
```

```
}
status DeleteNode(BiTree& T, KeyType e)
{
    BiTree t=NULL,p=NULL, tp=NULL;
    t = LocateNode(T, e);
    tp = LocateNodep(T, e);
    if (!t)
        return ERROR;
    if (!tp)
    {
        if (!(t->lchild || t->rchild))
            free(T);
        else if (!t->lchild && t->rchild)
        {
            p = T;
            free(p);
            T = T->rchild;
        }
        else if (!t->rchild && t->lchild)
        {
            p = T;
            free(p);
            T = T->lchild;
        }
        else
        {
            p = t->lchild;
            while (p->rchild)
            {
                p = p->rchild;
            }
```

```
            p->rchild = t->rchild;
            p = t->lchild;
            free(t);
            T = p;
        }

    }
    else
    {
        if (!(t->lchild || t->rchild))
        {
            if (t == tp->lchild)
            {
                free(t);
                tp->lchild = NULL;
            }
            else
            {
                free(t);
                tp->rchild = NULL;
            }
        }
        else if (!t->lchild && t->rchild)
        {
            p = t->rchild;
            if (t == tp->lchild)
            {
                free(t);
                tp->lchild = p;
            }
            else
```

```
        {
            free(t);
            tp->rchild = p;
        }
    }
    else if (t->lchild && !t->rchild)
    {
        p = t->lchild;
        if (t == tp->lchild)
        {
            free(t);
            tp->lchild = p;
        }
        else
        {
            free(t);
            tp->rchild = p;
        }
    }
    else
    {
        p = t->lchild;
        while (p->rchild)
        {
            p = p->rchild;
        }
        p->rchild = t->rchild;
        p = t->lchild;
        free(t);
        if (t == tp->lchild)
            tp->lchild = p;
```

```
            else
                tp->rchild = p;
        }
    }
    return OK;
    /********** End **********/
}
void PreOrderTraverse(BiTree T)
{
    if (T == NULL)
        return;
    printf("%d %s ", T->data.key, T->data.others);
    PreOrderTraverse(T->lchild);
    PreOrderTraverse(T->rchild);
}
void InOrderTraverse(BiTree T)
{
    if (T == NULL)
        return;
    InOrderTraverse(T->lchild);
    printf("%d %s ", T->data.key, T->data.others);
    InOrderTraverse(T->rchild);
}
void PostOrderTraverse(BiTree T)
{
    BiTree  St[9999], pre;
    int flag, top = 0;
    do
    {
        while (T != NULL)
        {
```

```
            St[top++] = T;
            T = T->lchild;
        }
        pre = NULL;
        flag = 1;
        while (top && flag)
        {
            T = St[top - 1];
            if (T->rchild == pre)
            {
                printf("%d %s ", T->data.key, T->data.others);
                top--;
                pre = T;
            }
            else
            {
                T = T->rchild;
                flag = 0;
            }
        }
    } while (top);
    return ;
}
void LevelOrderTraverse(BiTree T)
{
    BiTree que[9999] = { 0 };
    BiTree p;
    int front = 0, rear = 0;
    que[rear++] = T;
    while (front != rear)
    {
```

```
        p = que[front];
        printf("%d %s ", que[front]->data.key, que[front]->data.others
        front++;
        if (p->lchild)
            que[rear++] = p->lchild;
        if (p->rchild)
            que[rear++] = p->rchild;
    }
    return ;
}
/*-------------------------------------------------------*/
int maxPathSum(BiTree T)
{
    if (!T)
        return 0;
    int left = max(0, maxPathSum(T->lchild));
    int right = max(0, maxPathSum(T->rchild));
    maxSum = max(maxSum, left + right + T->data.key);
    return T->data.key + max(left, right);
}
BiTree LowestCommonAncestor(BiTree T, KeyType pk, KeyType qk)
{
    BiTree q = NULL, p = NULL;
    q = LocateNode(T, pk);
    p = LocateNode(T, qk);
    if (T == q || T == p || T == NULL)
        return T;
    BiTree left = LowestCommonAncestor(T->lchild, pk, qk);
    BiTree right = LowestCommonAncestor(T->rchild, pk, qk);
    if (left != NULL && right != NULL)
        return T;
```

```
    if (left == NULL)
        return right;
    return left;
}
BiTree invertTree(BiTree T)
{
    if (T == NULL)
        return NULL;
    BiTree left = invertTree(T->lchild);
    BiTree right = invertTree(T->rchild);
    T->lchild = right;
    T->rchild = left;
    return T;
}
status SaveBiTree(BiTree T, char FileName[])
{
    static FILE* file = fopen(FileName, "w");
    if (!file)
        return ERROR;
    static int fs = 0;
    int ff = fs++;
    if (T == NULL)
    {
        fprintf(file, "0 null ");
        return OK;
    }
    fprintf(file, "%d %s ", T->data.key, T->data.others);
    SaveBiTree(T->lchild, FileName);
    SaveBiTree(T->rchild, FileName);
    if (ff == 0)
    {
```

```
        fprintf(file, "-1 null ");
        fclose(file);
    }
    return OK;
}
status LoadBiTree(BiTree& T, TElemType definition[])
{
    i++;
    if (definition[i].key == 0)
        T = NULL;
    else
    {
        T = (BiTree)malloc(sizeof(BiTNode));
        T->data.key = definition[i].key; //生成根结点
        strcpy(T->data.others, definition[i].others); //生成根结点
        LoadBiTree(T->lchild, definition);
        LoadBiTree(T->rchild, definition);
    }
    return OK;
}
/*-------------------------------------------------------*/
status AddList(LISTS& Lists)//15
{
    if (Lists.length >= 10)
        return ERROR;
    char ListName[100];
    printf("    请输入您想添加的线性表的名称：");
    scanf("%s", ListName);
    if (strlen(ListName) >= 20)
    {
        printf_red("    名字过长！\n");
```

```
        return ERROR;
    }
    strcpy(Lists.elem[Lists.length].name, ListName);
    TElemType definition[9999] = { 0 };
    int j = 0;
    printf("    请输入二叉树内容：");
    do {
        scanf("%d%s", &definition[j].key, definition[j].others);
    } while (definition[j++].key != -1);
    CreateBiTree(Lists.elem[Lists.length].L, definition);
    Lists.length++;
    return OK;
}
status ShiftList()//16
{
    int i;
    printf("    请输入您要操作的线性表的序号：");
    scanf("%d", &i);
    if (i > Lists.length)
        return ERROR;
    n = i - 1;
    return OK;
}
status RemoveList(LISTS& Lists)//17
{
    char ListName[20];
    printf("    请输入您要删除的线性表的名称：");
    scanf("%s", ListName);
    int i, j;
    for (i = 0, j = 0; i < Lists.length; i++)
    {
```

```
        if (strcmp(ListName, Lists.elem[i].name) == 0)
        {
            DestroyBiTree(Lists.elem[i].L);
            for (int j = i; j < Lists.length - 1; j++)
            {
                Lists.elem[j] = Lists.elem[j + 1];
            }
            Lists.length--;
            return OK;
        }
    }
    return ERROR;
}
int LocateList(LISTS Lists)//18
{
    char ListName[20];
    printf("    请输入您要查找的线性表的名称：");
    scanf("%s", ListName);
    int i, f = 1;
    for (i = 0; i < Lists.length; i++)
    {
        if (strcmp(ListName, Lists.elem[i].name) == 0)
            return i + 1;
    }
    return 0;
}
//main.cpp
#include"def.h"
#define max(a,b) ((a)>(b)?(a):(b))
LISTS Lists;
int n,sta,i;
```

```
int maxSum = 9999;
BiTree T;
#include "fun.h"
int main()
{
    int op = 1;
    Lists.length = 0;
    while (op)
    {
        system("cls");  printf("\n\n");
        printf("  Menu for Linear Table On Sequence Structure \n");
        printf("-------------------------------------------\n");
        printf("   1. AddList               2. ShiftList\n");
        printf("   3. RemoveList            4. LocateList\n");
        printf("   5. DestroyBiTree         6. ClearBiTree\n");
        printf("   7. BiTreeEmpty           8. BiTreeDepth\n");
        printf("   9. LocateNode            10. Assign\n");
        printf("   11. GetSibling           12. InsertNode\n");
        printf("   13. DeleteNode           14. PreOrderTraverse\n");
        printf("   15. InOrderTraverse      16. PostOrderTraverse\n");
        printf("   17. LevelOrderTraverse   18. maxPathSum\n");
        printf("   19. LowestCommonAncestor 20. invertTree\n");
        printf("   21. SaveBiTree           22. LoadBiTree\n");
        printf("   23. CreateBiTree         0. exit\n");
        printf("-------------------------------------------\n");
        printf("    请选择你的操作[0~23](当前操作二叉树为%d[1~%d]):",
        scanf("%d", &op);
        switch (op)
        {
        case 1://AddList
            sta = AddList(Lists);
```

```
            if (sta == ERROR)
                printf_red("    添加失败，二叉树数量已达上限！\n");
            else
                printf_green("  添加成功！\n");
            getchar(); getchar();
            break;
        case 2://ShiftList
            sta = ShiftList();
            if (sta == ERROR)
                printf_red("    该二叉树不存在！\n");
            else
                printf_green("  转换成功！\n");
            getchar(); getchar();
            break;
        case 3://RemoveList
            sta = RemoveList(Lists);
            if (sta == OK)
                printf_green("  删除成功！\n");
            else
                printf_red("    未找到该二叉树!\n");
            getchar(); getchar();
            break;
        case 4://LocateList
            sta = LocateList(Lists);
            if (sta == 0)
                printf_red("    未找到该二叉树!\n");
            else
                printf("        这是第%d个二叉树\n", sta);
            getchar(); getchar();
            break;
        case 5://DestroyBiTree
```

```
        if(Lists.elem[n].L==NULL)
            printf_red("    二叉树不存在！\n");
        else
        {
                DestroyBiTree(Lists.elem[n].L);
                    printf_green("  二叉树已销毁!\n");
        }
            getchar(); getchar();
            break;
    case 6://ClearBiTree
        if (Lists.elem[n].L == NULL)
            printf_red("    二叉树不存在！\n");
        else
        {
            ClearBiTree(Lists.elem[n].L);
            printf_green("  二叉树已清空!\n");
        }
        getchar(); getchar();
        break;
    case 7://BiTreeEmpty
        sta = BiTreeEmpty(Lists.elem[n].L);
        if (sta == OK)
            printf_green("    二叉树为空！\n");
        else
            printf_yellow("    二叉树不为空！\n");
        getchar(); getchar();
        break;
    case 8://BiTreeDepth
        sta = BiTreeDepth(Lists.elem[n].L);
        if (sta == 0)
            printf_red("    二叉树不存在!\n");
```

```
        else
            printf("          二叉树深度为 %d", sta);
        getchar(); getchar();
        break;
    case 9://LocateNode
    {
        BiTree temp = NULL;
        KeyType e;
        printf("您要查找的关键字为：");
        scanf("%d", &e);
        temp = LocateNode(Lists.elem[n].L,e);
        if (temp == NULL)
            printf_red("     二叉树不存在!\n");
        else
            printf("%d %s\n", temp->data.key, temp->data.others);
        getchar(); getchar();
        break;
    }
    case 10://Assign
        sta = Assign(Lists.elem[n].L);
        if (sta == ERROR)
            printf_red("     替换失败!\n");
        else
            printf_green("   替换成功！\n");
        getchar(); getchar();
        break;
    case 11://GetSibling
    {
        BiTree temp;
        KeyType e;
        printf("您要查找的关键字为：");
```

```
        scanf("%d", &e);
        temp = GetSibling(Lists.elem[n].L,e);
        if (temp == NULL)
            printf_red("     结点不存在!\n");
        else
            printf("          兄弟节点为:  %d %s\n", temp->data.key,
        getchar(); getchar();
        break;
    }
    case 12://InsertNode
    {
        KeyType e;
        int LR;
        TElemType c;
        printf("请输入待插入结点关键字: ");
        scanf("%d", &e);
        printf("请输入插入方式LR=");
        scanf("%d", &LR);
        printf("请输入插入的结点信息: ");
        scanf("%d %s", &c.key, c.others);
        sta = InsertNode(Lists.elem[n].L, e, LR, c);
        if (sta == ERROR)
            printf_red("     插入失败!\n");
        else
            printf_green("   插入成功\n");
        getchar(); getchar();
        break;
    }
    case 13://DeleteNode
    {
        KeyType e;
```

```
        printf("请输入您要删除的结点的关键字： ");
        scanf("%d", &e);
        sta = DeleteNode(Lists.elem[n].L,e);
        if (sta == ERROR)
            printf_red("    删除失败!\n");
        else
            printf_green("  删除成功!\n");
        getchar(); getchar();
        break;
    }
    case 14://PreOrderTraverse
    {
        if (!Lists.elem[n].L)
            printf_red("二叉树为空！ ");
        PreOrderTraverse(Lists.elem[n].L);
            getchar(); getchar();
            break;
    }
    case 15://InOrderTraverse
        if (!Lists.elem[n].L)
            printf_red("二叉树为空！ ");
        InOrderTraverse(Lists.elem[n].L);
        getchar(); getchar();
        break;
    case 16://PostOrderTraverse
        if (!Lists.elem[n].L)
            printf_red("二叉树为空！ ");
        PostOrderTraverse(Lists.elem[n].L);
        getchar(); getchar();
        break;
    case 17://LevelOrderTraverse
```

```
        if (!Lists.elem[n].L)
            printf_red("二叉树为空！ ");
        LevelOrderTraverse(Lists.elem[n].L);
        getchar(); getchar();
        break;
    case 18://maxPathSum
        sta = maxPathSum(Lists.elem[n].L);
        if (sta == 0)
            printf_red("    二叉树不存在!\n");
        else
            printf("    最大路径和为%d\n",sta);
        getchar(); getchar();
        break;
    case 19://LowestCommonAncestor
    {
        BiTree temp;
        KeyType pk, qk;
        printf("请输入查找的两个结点的关键字： ");
        scanf("%d %d", &pk, &qk);
        temp = LowestCommonAncestor(Lists.elem[n].L, pk, qk);
        if (temp == NULL)
            printf_red("不存在！ \n");
        else
            printf("该节点为： %d %s\n", temp->data.key, temp->data
        getchar(); getchar();
        break;
    }
    case 20://invertTree
        Lists.elem[n].L = invertTree(Lists.elem[n].L);
        printf_green("转置成功！ \n");
        getchar(); getchar();
```

```
        break;
    case 21://SaveBiTree
    {
        char filename[100] = { 0 };
        char name[20] = { 0 };
        printf("请输入要写入的文件：");
        scanf("%s", name);
        sprintf(filename, "%s%s%s", "./", name, ".txt");
        sta = SaveBiTree(Lists.elem[n].L, filename);
        if (sta == OK)
            printf_green("写入成功！\n");
        else
            printf_red("写入失败！\n");
        getchar(); getchar();
        break;
    }
    case 22://LoadBiTree
    {
        TElemType definition[1000] = { 0 };
        char filename[100] = { 0 };
        char name[20] = { 0 };
        printf("请输入要读的文件：");
        scanf("%s", name);
        sprintf(filename, "%s%s%s", "./", name, ".txt");
        FILE* file = fopen(filename, "r");
        int j = 0;
        while (fscanf(file, "%d %s\n", &definition[j].key, &defini
        {
            j++;
        }
        i = -1;
```

```
            sta = LoadBiTree(Lists.elem[n].L, definition);
            if (sta == OK)
                printf_green("读入成功！\n");
            else
                printf_red("写入失败！\n");
                getchar(); getchar();
                break;
        }
        case 23:
        {
            TElemType definition[9999];
            int i = 0;
            do {
                scanf("%d%s", &definition[i].key, definition[i].others
            } while (definition[i++].key != -1);
            sta = CreateBiTree(T, definition);
            if (sta == OK)
                printf("创建成功！\n");
            else
                printf("创建失败！\n");
        }
        case 0://exit
            break;
        }//end of switch
    }//end of while
    printf("欢迎下次再使用本系统！\n");
}//end of main()
```

7 附录 D 基于邻接表图实现的源程序

```
\\def.h
#pragma once
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define MAX_VERTEX_NUM 20
typedef int status;
typedef int KeyType;
typedef enum { DG, DN, UDG, UDN } GraphKind;
typedef struct {
        KeyType  key;
        char others[20];
} VertexType; //顶点类型定义

typedef struct ArcNode {          //表结点类型定义
        int adjvex;               //顶点位置编号
        struct ArcNode* nextarc;          //下一个表结点指针
} ArcNode;
typedef struct VNode {                            //头结点及其数组类型定
        VertexType data;         //顶点信息
        ArcNode* firstarc;                //指向第一条弧
} VNode, AdjList[MAX_VERTEX_NUM];
```

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

typedef struct {  //线性表的管理表定义
        struct {
                char name[30];
                ALGraph L;
        } elem[10];
        int length;
        int listsize;
}LISTS;
int LocateVex(ALGraph G, KeyType u);
void printf_red(const char* s)
{
        printf("\033[1;31m%s\033[0m", s);
}
void printf_green(const char* s)
{
        printf("\033[1;32m%s\033[0m", s);
}
void printf_yellow(const char* s)
{
        printf("\033[1;33m%s\033[0m", s);
}
void visit(VertexType v)
{
        printf(" %d %s", v.key, v.others);
}
```

```
//fun.h
#pragma once
status CreateCraph(ALGraph& G)
{
    VertexType V[MAX_VERTEX_NUM]; KeyType VR[MAX_VERTEX_NUM][2];
    printf("    请输入顶点序列和关系对序列：");
    int i = 0;
    do {
        scanf("%d %s", &V[i].key, V[i].others);
    } while (V[i++].key != -1);
    i = 0;
    do {
        scanf("%d %d", &VR[i][0], &VR[i][1]);
    } while (VR[i++][1] != -1);
    int j, k;
    int check[MAX_VERTEX_NUM] = {0};
    for (i = 0; V[i].key != -1; i++)
    {
        if (check[V[i].key] != 0)
            return ERROR;
        check[V[i].key]++;
    }
    if (i == 0 || i >= 21)
        return ERROR;
    G.vexnum = 0, G.arcnum = 0;
    for (i = 0; i < MAX_VERTEX_NUM; i++)
        G.vertices[i].firstarc = NULL;
    i = 0;
    while (V[i].key != -1)
    {
        G.vertices[i].data.key = V[i].key;
```

```
        strcpy(G.vertices[i].data.others, V[i].others);
        i++;
    }
    G.vexnum = i;
    for (k = 0; VR[k][0] != -1; k++)
    {
        i = LocateVex(G, VR[k][0]);
        j = LocateVex(G, VR[k][1]);
        if (i == -1 || j == -1)
            return ERROR;
        ArcNode* p = (ArcNode*)malloc(sizeof(ArcNode));
        p->adjvex = j, p->nextarc = G.vertices[i].firstarc, G.vertices
        p = (ArcNode*)malloc(sizeof(ArcNode));
        p->adjvex = i, p->nextarc = G.vertices[j].firstarc, G.vertices
        G.arcnum++;
    }
    return OK;
}
status DestroyGraph(ALGraph& G)
{
    int i;
    ArcNode* p, * q;
    for (i = 0; i < G.vexnum; i++) {
        p = G.vertices[i].firstarc;
        while (p) {
            q = p->nextarc;
            free(p);
            p = q;
        }
    }
    G.vexnum = 0;
```

```
    G.arcnum = 0;
    return OK;
}
int LocateVex(ALGraph G, KeyType u)
{
    int i;
    for (i = 0; i < G.vexnum; i++)
    {
        if (u == G.vertices[i].data.key)
            return i;
    }
    return -1;
}
status PutVex(ALGraph& G, KeyType u, VertexType value)
{
    int check[MAX_VERTEX_NUM] = { 0 };
    for (int i = 0; i < G.vexnum; i++)
        check[G.vertices[i].data.key]++;
    for (int i = 0; i < G.vexnum; i++) {
        if (G.vertices[i].data.key == u && !check[value.key])
        {
            G.vertices[i].data.key = value.key;
            strcpy(G.vertices[i].data.others, value.others);
            return OK;
        }
    }
    return ERROR;
}
int FirstAdjVex(ALGraph G, KeyType u)
{
    int i;
```

```
    for (i = 0; i < G.vexnum && G.vertices[i].data.key != u; i++);
    if (i == G.vexnum) {
        return -1;
    }
    ArcNode* p = G.vertices[i].firstarc;
    if (p) {
        return p->adjvex;
    }
    else {
        return -1;
    }
}
int NextAdjVex(ALGraph G, KeyType v, KeyType w)
{
    int i;
    for (i = 0; i < G.vexnum && G.vertices[i].data.key != v; i++);
    if (i == G.vexnum)
        return -1;
    ArcNode* p = G.vertices[i].firstarc;
    while (p && G.vertices[p->adjvex].data.key != w) {
        p = p->nextarc;
    }
    if (!p || !p->nextarc) {
        return -1;
    }
    return p->nextarc->adjvex;
}
status InsertVex(ALGraph& G, VertexType v)
{
    for (int i = 0; i < G.vexnum; i++)
    {
```

```
        if (G.vertices[i].data.key == v.key)
            return ERROR;
    }

    if (G.vexnum == MAX_VERTEX_NUM) return ERROR;
    G.vertices[G.vexnum].data.key = v.key;
    strcpy(G.vertices[G.vexnum].data.others, v.others);
    G.vertices[G.vexnum].firstarc = NULL;
    G.vexnum++;
    return OK;
}
status DeleteVex(ALGraph& G, KeyType v)
{
    int i, j, k;
    ArcNode* p, * q;
    for (i = 0; i < G.vexnum && G.vertices[i].data.key != v; i++);
    if (i == G.vexnum)
        return ERROR;
    p = G.vertices[i].firstarc;
    while (p)
    {
        q = p;
        p = p->nextarc;
        free(q);
        G.arcnum--;
    }
    if (!G.arcnum)
        return ERROR;
    for (j = i; j < G.vexnum - 1; j++)
    {
        G.vertices[j] = G.vertices[j + 1];
```

```
    }
    G.vexnum--;
    for (k = 0; k < G.vexnum; k++)
    {
        p = G.vertices[k].firstarc;
        ArcNode* pre = NULL;
        while (p)
        {
            if (p->adjvex == i)
            {
                if (pre == NULL)
                    G.vertices[k].firstarc = p->nextarc;
                else
                    pre->nextarc = p->nextarc;
                    q = p;
                    p = p->nextarc;
                    free(q);
            }
            else
            {
                pre = p;
                p = p->nextarc;
            }
        }
    }
    return OK;
}
status InsertArc(ALGraph& G, KeyType v, KeyType w)
{
    int i, j;
    ArcNode* p;
```

```
    i = LocateVex(G, v);
    j = LocateVex(G, w);
    if (i < 0 || j < 0)
        return ERROR;
    for (ArcNode* q = G.vertices[i].firstarc; q; q = q->nextarc)
    {
        if (q->adjvex == j)
            return ERROR;
    }
    p = (ArcNode*)malloc(sizeof(ArcNode));
    p->adjvex = j;
    p->nextarc = G.vertices[i].firstarc;
    G.vertices[i].firstarc = p;
    p = (ArcNode*)malloc(sizeof(ArcNode));
    p->adjvex = i;
    p->nextarc = G.vertices[j].firstarc;
    G.vertices[j].firstarc = p;
    G.arcnum++;
    return OK;
}
status DeleteArc(ALGraph& G, KeyType v, KeyType w)
{
    int i, j;
    ArcNode* p1, * p2, * q;
    i = LocateVex(G, v);
    j = LocateVex(G, w);
    if (i < 0 || j < 0)
        return ERROR;
    p1 = G.vertices[i].firstarc;
    q = NULL;
    while (p1 != NULL)
```

```
{
    if (p1->adjvex == j)
        break;
    q = p1;
    p1 = p1->nextarc;
}
if (p1 == NULL)
    return ERROR;
if (q == NULL)
    G.vertices[i].firstarc = p1->nextarc;
else
    q->nextarc = p1->nextarc;
free(p1);
p2 = G.vertices[j].firstarc;
q = NULL;
while (p2 != NULL)
{
    if (p2->adjvex == i)
        break;
    q = p2;
    p2 = p2->nextarc;
}
if (p2 == NULL)
    return ERROR;
if (q == NULL)
    G.vertices[j].firstarc = p2->nextarc;
else
    q->nextarc = p2->nextarc;
free(p2);
G.arcnum--;
return OK;
```

```
}
status DFSTraverse(ALGraph& G, void (*visit)(VertexType))
{
    int visited[MAX_VERTEX_NUM] = { 0 };
    int stack[MAX_VERTEX_NUM] = { 0 };
    int  top = 0, i = 0, pos;
    visit(G.vertices[i].data);
    stack[top++] = i;
    while (top > 0)
    {
        pos = stack[top - 1];
        visited[pos] = 1;
        ArcNode* e = G.vertices[pos].firstarc;
        while (e && visited[e->adjvex])
        {
            e = e->nextarc;
        }
        if (e == NULL)
            top--;
        else
        {
            visit(G.vertices[e->adjvex].data);
            stack[top++] = e->adjvex;
        }
    }
    return OK;
}
status BFSTraverse(ALGraph& G, void (*visit)(VertexType))
{
    int visited[MAX_VERTEX_NUM] = { 0 };
    int que[MAX_VERTEX_NUM] = { 0 };
```

```
    int head = 0, tail = 0, pos, i = 0;
    visit(G.vertices[i].data);
    que[tail++] = i;
    while (head < tail)
    {
        pos = que[head++];
        visited[pos] = 1;
        ArcNode* e = G.vertices[pos].firstarc;
        while (e)
        {
            if (!visited[e->adjvex])
            {
                que[tail++] = e->adjvex;
                visit(G.vertices[e->adjvex].data);
            }
            e = e->nextarc;
        }
    }
    return OK;
}
status VerticesSetLessThanK(ALGraph G, KeyType v, int k, void (*visit)
{
    int dis[MAX_VERTEX_NUM], que[MAX_VERTEX_NUM], head = 0, tail = 0;
    memset(dis, -1, sizeof(dis));
    int p=LocateVex(G, v),pos;
    dis[p] = 0;
    que[tail++] = p;
    while (head < tail)
    {
        int pos;
        pos = que[head++];
```

```
            ArcNode* e = G.vertices[pos].firstarc;
            while (e)
            {
                if (dis[e->adjvex] == -1)
                {
                    dis[e->adjvex] = dis[pos] + 1;
                    if(dis[e->adjvex]<k)
                        visit(G.vertices[e->adjvex].data);
                    que[tail++] = e->adjvex;
                }
                e = e->nextarc;
            }
        }
        return OK;
}
status ShortestPathLength(ALGraph G, KeyType v, KeyType w)
{
    int dis[MAX_VERTEX_NUM], que[MAX_VERTEX_NUM], head = 0, tail = 0;
    memset(dis, -1, sizeof(dis));
    int p = LocateVex(G, v), q = LocateVex(G, w), pos;
    dis[p] = 0;
    que[tail++] = p;
    while (head < tail)
    {
        int pos;
        pos = que[head++];
        ArcNode* e = G.vertices[pos].firstarc;
        while (e)
        {
            if (dis[e->adjvex] == -1)
            {
```

```
                dis[e->adjvex] = dis[pos] + 1;
                que[tail++] = e->adjvex;
            }
            e = e->nextarc;
        }
    }
    if (dis[q] < 0)
        return ERROR;
    printf("最短路径长度为%d\n",dis[q]);
    return OK;
}
status SaveGraph(ALGraph G, char FileName[])
{
    int ct = 0;
    FILE* fp = fopen(FileName, "w");
    if (fp == NULL)
        return ERROR;
    fprintf(fp, "%d %d\n", G.vexnum, G.arcnum);
    for (int i = 0; i < G.vexnum; i++)
    {
        fprintf(fp, "%d %s ", G.vertices[i].data.key, G.vertices[i].da
        ArcNode* p = G.vertices[i].firstarc;
        while (p)
        {
            fprintf(fp, "%d ", p->adjvex);
            p = p->nextarc;
        }
        fprintf(fp, "-1\n");
    }
    fclose(fp);
    return OK;
```

```
}
status LoadGraph(ALGraph& G, char FileName[])
{
    FILE* fp = fopen(FileName, "r");
    if (fp == NULL)
        return ERROR;
    ArcNode* p, * q;
    fscanf(fp, "%d %d", &G.vexnum, &G.arcnum);
    for (int i = 0; i < G.vexnum; i++)
    {
        fscanf(fp, "%d %s ", &G.vertices[i].data.key, G.vertices[i].da
        G.vertices[i].firstarc = NULL;
        int adjvex;
        fscanf(fp, "%d", &adjvex);
        q = NULL;
        while (adjvex != -1)
        {
            p = (ArcNode*)malloc(sizeof(ArcNode));
            p->adjvex = adjvex;
            p->nextarc = NULL;
            if (q == NULL)
                G.vertices[i].firstarc = p;
            else
                q->nextarc = p;
            q = p;
            fscanf(fp, "%d", &adjvex);
        }
    }
    fclose(fp);
    return OK;
}
```

```
status AddList(LISTS& Lists)//15
{
    if (Lists.length >= 10)
        return ERROR;
    char ListName[100];
    printf("    请输入您想添加的线性表的名称：");
    scanf("%s", ListName);
    if (strlen(ListName) >= 20)
    {
        printf_red("    名字过长！\n");
        return ERROR;
    }
    strcpy(Lists.elem[Lists.length].name, ListName);
    CreateCraph(Lists.elem[Lists.length].L);
    Lists.length++;
    return OK;
}
status ShiftList()//16
{
    int i;
    printf("    请输入您要操作的线性表的序号：");
    scanf("%d", &i);
    if (i > Lists.length)
        return ERROR;
    n = i - 1;
    return OK;
}
status RemoveList(LISTS& Lists)//17
{
    char ListName[20];
    printf("    请输入您要删除的线性表的名称：");
```

```
    scanf("%s", ListName);
    int i, j;
    for (i = 0, j = 0; i < Lists.length; i++)
    {
        if (strcmp(ListName, Lists.elem[i].name) == 0)
        {
            DestroyGraph(Lists.elem[i].L);
            for (int j = i; j < Lists.length - 1; j++)
            {
                Lists.elem[j] = Lists.elem[j + 1];
            }
            Lists.length--;
            return OK;
        }
    }
    return ERROR;
}
int LocateList(LISTS Lists)//18
{
    char ListName[20];
    printf("    请输入您要查找的线性表的名称：");
    scanf("%s", ListName);
    int i, f = 1;
    for (i = 0; i < Lists.length; i++)
    {
        if (strcmp(ListName, Lists.elem[i].name) == 0)
            return i + 1;
    }
    return 0;
}
//main.cpp
```

```
#include"def.h"
LISTS Lists;
int n;
#include"fun.h"
status sta;

int main()
{
    int op = 1;
    Lists.length = 0;
    while (op)
    {
        system("cls");  printf("\n\n");
        printf("           Menu for Linear Table On Sequence Structure
        printf("-------------------------------------------------------
        printf("   1. AddList                  2. ShiftList\n");
        printf("   3. RemoveList               4. LocateList\n");
        printf("   5. CreateCraph              6.DestroyGraph\n");
        printf("   7. LocateVex                8. PutVex\n");
        printf("   9. FirstAdjVex              10. NextAdjVex\n");
        printf("   11. InsertVex               12. DeleteVex\n");
        printf("   13. InsertArc               14. DeleteArc\n");
        printf("   15. DFSTraverse             16.BFSTraverse\n");
        printf("   17. VerticesSetLessThanK    18.ShortestPathLength\n"
        printf("   19. SaveGraph               20. LoadGraph\n");
        printf(" 0. exit\n");
        printf("-------------------------------------------------\n");
        printf("    请选择你的操作[0~20](当前操作图为%d[1~%d]):", n +
           scanf("%d", &op);
           switch (op)
           {
```

```
case 1://AddList
    sta = AddList(Lists);
    if (sta == ERROR)
        printf_red("    添加失败，图数量已达上限！\n")
    else
        printf_green("  添加成功！\n");
    getchar(); getchar();
    break;
case 2://ShiftList
    sta = ShiftList();
    if (sta == ERROR)
        printf_red("    该图不存在！\n");
    else
        printf_green("  转换成功！\n");
    getchar(); getchar();
    break;
case 3://RemoveList
    sta = RemoveList(Lists);
    if (sta == OK)
        printf_green("  删除成功！\n");
    else
        printf_red("    未找到该图!\n");
    getchar(); getchar();
    break;
case 4://LocateList
    sta = LocateList(Lists);
    if (sta == 0)
        printf_red("    未找到该图!\n");
    else
        printf("        这是第%d个图\n", sta);
    getchar(); getchar();
```

```
            break;
        case 5://CreateCraph
        {
            sta=CreateCraph(Lists.elem[n].L);
            if (sta == OK)
                    printf_green("   创建成功！\n");
            getchar(); getchar();
            break;
        }
        case 6://DestroyGraph
            if (!Lists.elem[n].L.vertices)
                    printf_red("     图不存在！\n");
            else
            {
                    DestroyGraph(Lists.elem[n].L);
                    printf_green("   图已销毁!\n");
            }
            getchar(); getchar();
            break;
        case 7://LocateVex
        {
            KeyType u;
            printf("          请输入要查找的顶点关键字：\n");
            scanf("%d", &u);
            sta=LocateVex(Lists.elem[n].L,u);
            if (sta == -1)
                    printf_red("     顶点不存在！\n");
            else
                    printf("          这是第%d个顶点\n",sta);
            getchar(); getchar();
            break;
```

```
}
case 8://PutVex
{
      KeyType u;
      VertexType value;
      printf("          请输入要操作的顶点关键字：\n");
      scanf("%d", &u);
      printf("          请输入要赋的值：\n");
      scanf("%d %s", &value.key, value.others);
      sta = PutVex(Lists.elem[n].L,u,value);
      if (sta == OK)
            printf_green("     赋值成功！\n");
      else
            printf_red("     赋值失败！\n");
      getchar(); getchar();
      break;
}
case 9://FirstAdjVex
{
      KeyType u;
      printf("          请输入要查找的顶点关键字：\n");
      scanf("%d", &u);
      sta = FirstAdjVex(Lists.elem[n].L,u);
      if (sta == -1)
         printf_red(" 不存在!\n");
      else
         printf("      第一个邻接顶点位序为 %d", sta);
      getchar(); getchar();
      break;
}
case 10://NextAdjVex
```

```
{
        KeyType v,w;
        printf("          请输入要查找的两个顶点关键字：\n");
        scanf("%d %d", &v, &w);
        sta = NextAdjVex(Lists.elem[n].L, v, w);
        if (sta == -1)
           printf_red(" 不存在!\n");
        else
           printf("       下一邻接点的位序为%d\n",sta);
        getchar(); getchar();
        break;
}
case 11://InsertVex
{
        VertexType v;
        printf("          请输入要增加的顶点信息：");
        scanf("%d %s", &v.key, v.others);
        sta = InsertVex(Lists.elem[n].L,v);
        if (sta == ERROR)
                printf_red("     增加失败!\n");
        else
                printf_green("   增加成功！\n");
        getchar(); getchar();
        break;
}
case 12://DeleteVex
{
        KeyType v;
        printf("          请输入要删除的关键字：");
        scanf("%d", &v);
        sta = DeleteVex(Lists.elem[n].L, v);
```

```
            if (sta == ERROR)
                    printf_red("    删除失败!\n");
            else
                    printf_green("  删除成功! \n");
            getchar(); getchar();
            break;
        }
        case 13://InsertArc
        {
            KeyType v,w;
            printf("          请输入要插入的弧的两个关键字: ");
            scanf("%d %d", &v, &w);
            sta = InsertArc(Lists.elem[n].L, v,w);
            if (sta == ERROR)
                    printf_red("    插入失败!\n");
            else
                    printf_green("  插入成功! \n");
            getchar(); getchar();
            break;
        }
        case 14://DeleteArc
        {
            KeyType v, w;
            printf("          请输入要删除的弧的两个关键字: ");
            scanf("%d %d", &v, &w);
            sta = DeleteArc(Lists.elem[n].L, v, w);
            if (sta == ERROR)
                    printf_red("    删除失败!\n");
            else
                    printf_green("  删除成功! \n");
            getchar(); getchar();
```

```
            break;
        }
        case 15://DFSTraverse
        {
            if (!Lists.elem[n].L.vexnum)
                    printf_red("    图为空！");
            DFSTraverse(Lists.elem[n].L, visit);
            getchar(); getchar();
            break;
        }
        case 16://BFSTraverse
            if (!Lists.elem[n].L.vexnum)
                    printf_red("    图为空！");
            BFSTraverse(Lists.elem[n].L,visit);
            getchar(); getchar();
            break;
        case 17://VerticesSetLessThanK
        {
            int v, k;
            printf("          请输入开始的顶点关键字：");
            scanf("%d", &v);
            printf("          请输入距离：");
            scanf("%d", &k);
            VerticesSetLessThanK(Lists.elem[n].L, v, k, visit);
            getchar(); getchar();
            break;
        }
        case 18://ShortestPathLength
        {
            int v, w;
            printf("          请输入要查找的两个顶点的关键字：");
```

```
            scanf("%d %d", &v, &w);
            if (ShortestPathLength(Lists.elem[n].L, v, w) == ERROR
                    printf_red("    两顶点不相连！");
        }
            getchar(); getchar();
            break;
        case 19://SaveGraph
        {
            char filename[100] = { 0 };
            char name[20] = { 0 };
            printf("         请输入要写入的文件：");
            scanf("%s", name);
            sprintf(filename, "%s%s%s", "./", name, ".txt");
            sta = SaveGraph(Lists.elem[n].L, filename);
            if (sta == OK)
                    printf_green("  写入成功！\n");
            else
                    printf_red("    写入失败！\n");
            getchar(); getchar();
            break;
        }
        case 20://LoadGraph
        {
            char filename[100] = { 0 };
            char name[20] = { 0 };
            printf("         请输入要读的文件：");
            scanf("%s", name);
            sprintf(filename, "%s%s%s", "./", name, ".txt");
            sta = LoadGraph(Lists.elem[n].L, filename);
            if (sta == OK)
                    printf_green("  读入成功！\n");
```

```
                else
                        printf_red("     写入失败！\n");
                getchar(); getchar();
                break;
            }
            case 0://exit
                break;
            }//end of switch
        }//end of while
        printf("          欢迎下次再使用本系统！\n");
}//end of main()
```