

华中科技大学

课程实验报告

课程名称: 数据结构实验

专业班级 CS2208

学 号 U202215595

姓 名 郭凯锐

指导教师 袁凌

报告日期 2023 年 5 月 28 日

计算机科学与技术学院

目 录

1	基于链式存储结构的线性表实现.....	1
1.1	问题描述	1
1.2	系统设计	1
1.3	系统实现.....	19
1.4	系统测试.....	20
1.5	实验小结.....	23
2	基于邻接表的图实现	24
2.1	问题描述	24
2.2	系统设计	24
2.3	系统实现.....	49
2.4	系统测试.....	50
2.5	实验小结.....	51
3	课程的收获和建议	52
3.1	基于链式存储结构的线性表实现	52
3.2	基于邻接表的图实现.....	52
4	附录 A 基于顺序存储结构线性表实现的源程序	53
5	附录 B 基于链式存储结构线性表实现的源程序.....	78
6	附录 C 基于二叉链表二叉树实现的源程序	103
7	附录 D 基于邻接表图实现的源程序.....	130

1 基于链式存储结构的线性表实现

1.1 问题描述

实验要求：

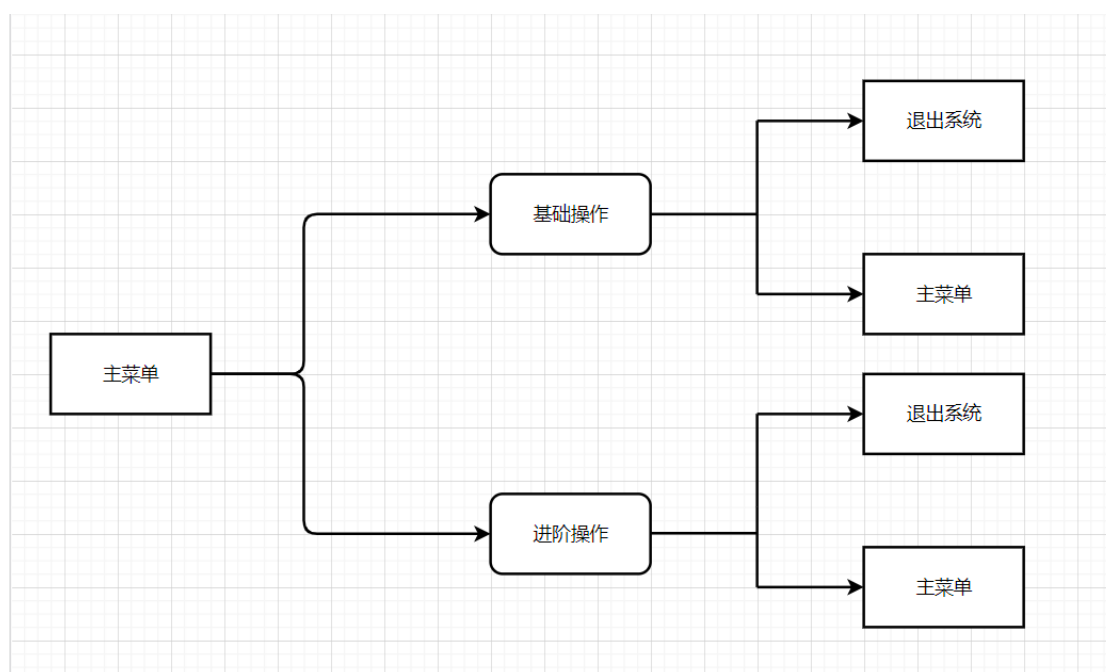
1. 实现对链表的基本操作；
2. 选择性实践对链表的进阶操作；
3. 设计演示系统。

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. 物理结构采用单链表, 熟练掌握线性表的基本运算的实现。

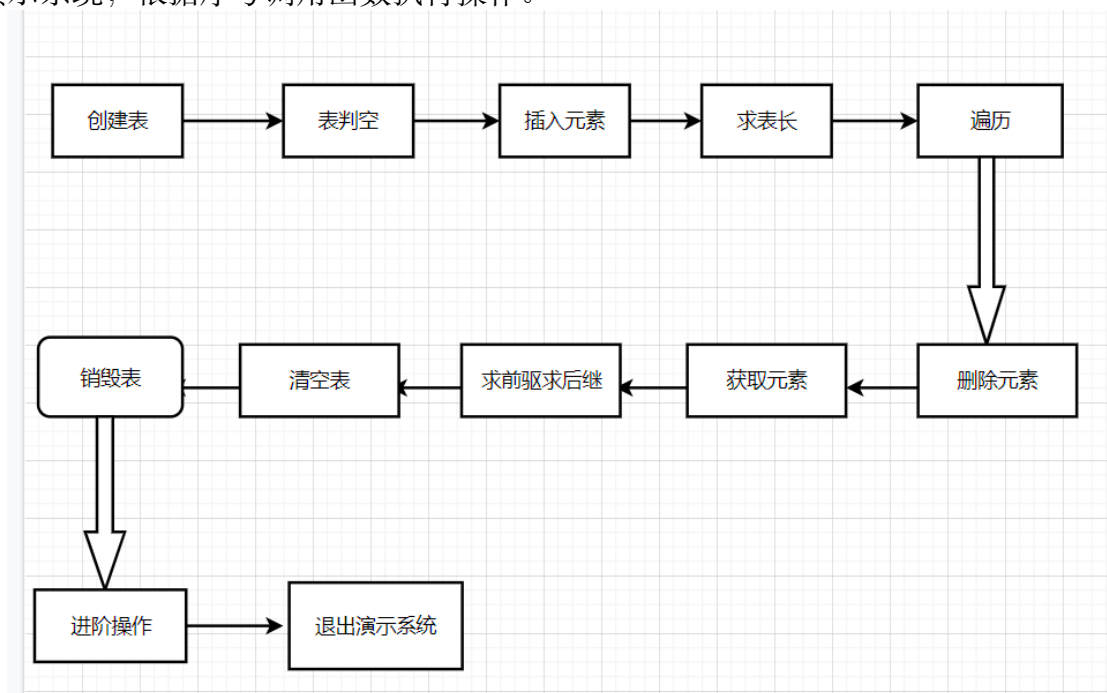
1.2 系统设计

1.2.1 演示系统菜单的组织架构



华中科技大学课程实验报告

演示系统分为基本操作和进阶操作两部分，以菜单为主界面，以序号 1 至 12 表示创建，销毁，清空，插入，删除，求表长等基本操作，以序号 13 至 17 表示保存为文件，加载文件，翻转链表，对链表排序等进阶操作，以序号 0 表示退出演示系统，根据序号调用函数执行操作。



1.2.2 ADT 数据结构设计

对链表数据类型的定义如下：

```
1 typedef int status ;
2 typedef int ElemType; //type of element
3
4
5 typedef struct LNode{
6     ElemType data;
7     struct LNode *next;
8 }LNode, *LinkList;
```

对一些常量的定义如下：

```
1 #define TRUE 1
2 #define FALSE 0
3 #define OK 1
4 #define ERROR 0
5 #define INFEASIBLE -1
6 #define OVERFLOW -2
7 #define MAX_NUM 10
8 #define LIST_INIT_SIZE 100
9 #define LISTINCREMENT 10
```

1.2.3 初始化表

输入：顺序表 (未知状态)

输出：函数执行状态

算法的思想描述：为 L 结点分配存储空间, 将 L->next 结点置空

```
1 status InitList (LinkList *L)
2 // initiate a list
3 {
4     *L = (LinkList)malloc( sizeof (LNode));//malloc a node
```

```
5     if(*L == NULL)
6     {
7         exit(OVERFLOW);//malloc failed
8     }
9     (*L)->data = 0;
10    (*L)->next = NULL;//the list is empty
11    return OK;
12 }
```

算法处理步骤:

- 1) 如果顺序表 L 不存在, 则为 L 分配存储空间。
- 2) 如果分配成功, 将 L->next 置空返回状态 OK。
- 3) 如果分配失败, 返回状态 OVERFLOW 并推出该程序块。
- 4) 如果顺序表存在, 返回状态 INFEASIBLE 以指出无法创建。

时间复杂度: $O(1)$

空间复杂度: $O(1)$

1.2.4 销毁表

输入: 顺序表 (未知状态)

输出: 函数执行状态

算法的思想描述: 用 while 循环依次释放所有节点的储存空间, 再将 L 置为 NULL。

```
1 status DestroyList(LinkList *L)
2 //destroy a list
3 {
4     LinkList p, q;
5     p = *L;
6     while(p)
7     {
8         q = p->next;
9         free(p);
```

```
10         p = q;
11     }
12     *L = NULL;
13     return OK;
14 }
```

时间复杂度: $O(n)$

空间复杂度: $O(1)$

1.2.5 清空表

输入: 顺序表 (未知状态)

输出: 函数执行状态

算法的思想描述: 用 while 循环释放除头节点以外的所有节点的储存空间, 再将 $L \rightarrow next$ 置为 NULL。

```
1  status ClearList (LinkList *L)
2  //clear a list
3  {
4      LinkList p, q;
5      p = (*L) -> next; //point to the first node
6      while(p)
7      {
8          q = p -> next;
9          free(p);
10         p = q;
11     }
12     (*L) -> next = NULL;
13     return OK;
14 }
```

时间复杂度: $O(n)$

空间复杂度: $O(1)$

1.2.6 表判空

输入: 顺序表 (未知状态)

输出: 函数执行状态

算法的思想描述: 判断 $L \rightarrow \text{next}$ 是否为 NULL, 若为 NULL, 则链表为空, 否则链表不为空。

```
1 status ListEmpty(LinkList L)
2 //judge if the list is empty
3 {
4     if(L->next)
5         return FALSE;
6     else
7         return TRUE;
8 }
```

时间复杂度: $O(1)$

空间复杂度: $O(1)$

1.2.7 求表长

输入: 顺序表 L

输出: 顺序表的长度

算法思想描述: 遍历整个顺序表

```
1 int ListLength(LinkList L)
2 //get the length of the list
3 {
4     int i = 0;
5     LinkList p = L->next;
6     while(p)
7     {
8         i++;
9         p = p->next;
10    }
```



```
11     return i;  
12 }
```

算法处理步骤:

- 1) 如果 L 存在, 从头开始遍历。
- 2) 若不为空则移动指针到下一个结点, 长度加 1。
- 3) 返回长度的值。

时间复杂度: $O(n)$

空间复杂度: $O(1)$

1.2.8 获取元素

输入: 顺序表 (未知状态)

输出: L 中第 i 个数据元素的值 e

算法的思想描述: 若 i 小于 1 或大于链表的长度, 则该序号非法, 返回 ERROR。否则遍历至第 i 个元素, 返回其数据域的值。

```
1  status GetElem(LinkList L, int i, ElemType *e)  
2  //get the element of the No.i node  
3  {  
4      int j = 1;  
5      LinkList p;  
6      p = L->next;  
7      while(p && j < i)  
8      {  
9          p = p->next;  
10         ++j;  
11     }  
12     if(!p || j > i)  
13         return ERROR;  
14     *e = p->data;  
15     return OK;  
16 }
```

时间复杂度: $O(n)$

空间复杂度: $O(1)$

1.2.9 定位元素

输入: 顺序表 (未知状态)

输出: L 中第 1 个与 e 相等的元素的序号, 若这样的数据元素不存在, 则值为 0

算法的思想描述: 声明 $i=0$, 遍历链表查找是否有值与 e 相同的元素, 每进入下一个节点时 i 自增, 若找到相应元素则返回 i, 否则返回 ERROR。

```
1  int LocateElem(LinkList L, ElemType e, status (*compare)(ElemType a, ElemType b))
2  //get the position of the element
3  {
4      int i = 0;
5      LinkList p = L->next;
6      while(p)
7      {
8          i++;
9          if((*compare)(p->data, e))
10             return i;
11             p = p->next;
12     }
13     return 0;
14 }
15
16 status compare(ElemType a, ElemType b)
17 {
18     if(a == b)
19         return TRUE;
20     else
21         return FALSE;
22 }
```

时间复杂度: $O(n)$

空间复杂度: $O(1)$

1.2.10 获得前驱

输入: 顺序表 L, 元素 e, 引用参数 pre。

输出: 函数的执行状态。

算法思想的描述: 遍历顺序表。

```
13 status PriorElem(LinkList L, ElemType cur_e, ElemType *pre_e)
14 //get the element before the cur_e
15 {
16     LinkList p = L->next;
17     if(p->data==cur_e)
18         return ERROR;
19     while(p->next != NULL && p->next->data != cur_e)
20         p = p->next;
21
22     if(p->next == NULL)
23         return OVERFLOW;
24
25     *pre_e = p->data;
26     return OK;
27 }
```

算法处理步骤:

- 1) 如果顺序表 L 不存在, 则输出 INFEASIBLE。
- 2) 如果 L 存在, 从头开始遍历。
- 3) 若当前节点的后继值为 e, 则将当前节点的值赋给 pre, 输出 OK。
- 4) 若为找到, 则返回 INFEASIBLE。

时间复杂度: $O(n)$

空间复杂度: $O(1)$

1.2.11 求后继

输入: 顺序表 (未知状态)

输出: 函数执行状态

算法的思想描述: 链表查找值等于 e 的元素, 若找到则将该节点后继节点的数据域赋值给 $next$, 否则返回 ERROR。

```
1 status NextElem(LinkList L, ElemType cur_e, ElemType *next_e)
2 //get the element after the cur_e
3 {
4     LinkList p = L->next;
5     while(p->next != NULL && p->data != cur_e)
6         p = p->next;
7
8     if(p->next == NULL && p->data != cur_e)
9         return ERROR;
10    if(p->next == NULL && p->data == cur_e)
11        return OVERFLOW;
12    *next_e = p->next->data;
13    return OK;
14 }
```

时间复杂度: $O(n)$

空间复杂度: $O(1)$

1.2.12 插入元素

输入: 顺序表 L , 插入位置 I , 插入元素 e 。

输出: 函数的执行状态。

算法的思想描述: 查找元素和移动之后的结点。

```
28 status ListInsert (LinkList *L, int i, ElemType e)
29 // insert a node before the No.i node
30 {
31     int j = 1;
```

```
32     LinkList p, q;
33     p = *L;
34     while(p && j < i)
35     {
36         p = p->next;
37         ++j;
38     }
39     if(!p || j > i)
40         return ERROR;
41
42     q = (LinkList)malloc(sizeof(LNode));
43     if(q == NULL)
44         exit(OVERFLOW);
45
46     q->data = e;
47     q->next = p->next;
48     p->next = q;
49     return OK;
50 }
```

算法处理步骤:

- 1) 如果 L 存在, 判断 i 值是否符合要求, 不符合则返回 ERROR。
- 2) 如果 I 值合法, 增加一个新的结点用于存储插入元素。
- 3) 将插入位置前的结点指向新插入的节点, 将新节点指向插入位置后一个节点.
- 4) 返回 OK。

时间复杂度: $O(n)$

空间复杂度: $O(1)$

1.2.13 删除元素

输入: 顺序表 (未知状态)

输出: 函数执行状态

算法的思想描述: 若 i 小于 1 或大于链表长度则返回 ERROR。否则遍历至链表第 $i-1$ 个元素, 将其指针域置为其后继的后继, 数据赋给 e , 最后释放其后继节点。

```
1 status ListDelete (LinkList *L, int i, ElemType *e)
2 // delete the No.i node
3 {
4     int j = 1;
5     LinkList p, q;
6     p = *L;
7     while(p->next && j < i)
8     {
9         p = p->next;
10        ++j;
11    }
12    if (!(p->next) || j > i)
13        return ERROR;
14
15    q = p->next;
16    p->next = q->next;
17    *e = q->data;
18    free(q);
19
20    return OK;
21 }
```

时间复杂度: $O(n)$

空间复杂度: $O(1)$

1.2.14 遍历链表

输入: 顺序表 L.

输出: 函数的执行状态

算法思想描述: 遍历链表并输出每一个元素的值.

- 1) 如果顺序表 L 不存在, 则输出 INFEASIBLE。
- 2) 如果 L 存在, 从头开始遍历并输出。
- 3) 返回 OK。

```
51 status ListTraverse (LinkList L)
52 // traverse the list
53 {
54     LinkList p = L;
55     if (!p)
56         return INFEASIBLE;
57     else if (!p->next)
58         return ERROR;
59     else
60     {
61         while(p)
62         {
63             if (p->data!=0)
64                 printf ("%d ",p->data);
65             p = p->next;
66         }
67         return OK;
68     }
69 }
```

时间的复杂度: $O(n)$.

空间的复杂度: $O(1)$.

1.2.15 链表的翻转

输入: 顺序表 L

输出: 链表翻转

算法的思想描述: 利用栈这一数据结构, 遍历让所有元素入栈再出栈, 即可得到翻转的结果.

```
70 status ReverseList(LinkList *L)
71 // reverse the list
72 {
73     if(L)
74     {
75         LinkList prev=NULL;
76         LinkList cur=*L;
77         LinkList next=NULL;
78         while(cur)
79         {
80             next=cur->next;
81             cur->next=prev;
82             prev=cur;
83             cur=next;
84         }
85         *L=prev;
86         return OK;
87     }
88     else
89         return INFEASIBLE;
90 }
```

算法处理的步骤:

- 1) 如果 L 不存在则为空表, 输出 ERROR 并退出.
- 2) 建立一个新的节点, 遍历所有节点以栈的形式存储
- 3) 改变头节点为栈的头节点

时间复杂度: $O(n)$

空间复杂度: $O(n)$

1.2.16 删除链表的倒数第 n 个节点

输入: 顺序表 (未知状态)

输出: 函数执行状态

算法的思想描述: 如果 L 为 NULL, 返回 INFEASIBLE。否则调用求表长函数和删除节点函数, 通过数学运算来实现倒数第 n 个元素的删除。

```
1 status RemoveNthFromEnd(LinkList &L, int n, ElemType& e)
2 // delete the nth node from the end of the list
3 {
4     if (!L)
5         return INFEASIBLE;
6     else
7     {
8         int k, j;
9         if (L->next == NULL)
10             return ERROR;
11         k = ListLength(L);
12         j = k - n + 1;
13         ListDelete(&L, j, &e);
14         return OK;
15     }
16 }
```

时间复杂度: $O(n)$

空间复杂度: $O(1)$

1.2.17 读入文件

输入: 顺序表 (未知状态)

输出: 函数执行状态

算法的思想描述: 打开文件, 从文件中每读入一个数据创建一个节点, 并置为上一个结点的后继, 直到读取完所有数据, 关闭文件。

```
1 status LoadList(LinkList *L, char *filename)
```

```
2 //load the list from a file
3 {
4     int i = 1, length = 0, listsize ;
5     ElemType e;
6     if ((fp = fopen(filename, "r")) == NULL)
7     {
8         printf ("File open error !\n");
9         return ERROR;
10    }
11    fscanf(fp, "%d ", &length);
12    fscanf(fp, "%d ", &listsize );
13    fscanf(fp, "%d ", &e);
14    while(i<=length)
15    {
16        ListInsert (L,i,e);
17        fscanf(fp, "%d ", &e);
18        i++;
19    }
20    fclose(fp);
21    return OK;
22 }
```

时间复杂度: $O(n)$

空间复杂度: $O(1)$

1.2.18 写入文件

输入: 顺序表 (未知状态)

输出: 函数执行状态

算法的思想描述: 打开文件, 遍历链表将所有元素的数据域写入文件, 关闭文件。

```
1 status SaveList(LinkList L, char* filename)
```

```
2 //save the list to a file
3 {
4     LinkedList p = L->next;
5     int listsize =LIST_INIT_SIZE;
6     if ((fp = fopen(filename, "w")) == NULL)
7     {
8         printf ("File open error !\n");
9         return ERROR;
10    }
11    fprintf (fp, "%d ", ListLength(L));
12    fprintf (fp, "%d ", listsize );
13    while(p)
14    {
15        fprintf (fp, "%d ", p->data);
16        p = p->next;
17    }
18    fclose (fp);
19    return OK;
20 }
```

时间复杂度: $O(n)$

空间复杂度: $O(1)$

1.2.19 链表排序

输入: 顺序表 (未知状态)

输出: 函数执行状态

算法的思想描述: 采用插入排序的方法对链表内元素进行大小排序。

```
1 status SortList (LinkedList L)
2 // sort the list
3 {
4     LinkedList p, q;
```

```
5  if(!L)
6      return INFEASIBLE;
7  else
8      {
9      if(L->next==NULL)
10         return ERROR;
11     int n = ListLength(L);
12     int i, j, temp;
13     for(i = 0, p = L -> next; i < n-1; i++, p = p -> next)
14         for(j = i + 1, q = p -> next; j < n; j++, q = q -> next)
15             if(p -> data > q -> data)
16                 {
17                     temp = p -> data;
18                     p -> data = q -> data;
19                     q -> data = temp;
20                 }
21
22     return OK;
23 }
24 }
```

时间复杂度: $O(n)$

空间复杂度: $O(1)$

1.3 系统实现

1.3.1 程序开发环境与语言

程序开发及实现环境：Win11 下使用 VScode+gcc 进行编译和调试，开发语言为 C 语言。

1.3.2 代码的组织结构

演示系统以一个菜单作为交互界面，用户通过输入命令对应的编号来调用相应的函数来实现创建表，销毁表，清空表，插入元素，删除元素，求表长，判空表，求前驱，求后继，遍历链表等基本操作，以及保存为文件，加载文件，翻转链表，对链表排序等进阶操作。

程序主函数为一个 switch 结构，根据输入的数字，执行不同的语句，进而调用不同的函数，宏定义函数返回值 ERROR 为 0, INFEASIBLE 为-1, OVERFLOW 为-2, OK 为 1.

交互界面如下图：

```
Menu for Linear Table On LinkedList
-----
1.InitList      2.DestroyList
3.ClearList    4.ListEmpty
5.ListLength   6.GetElem
7.LocateElem   8.PriorElem
9. NextElem    10.ListInsert
11.ListDelete  12.ListTraverse
13.SaveList    14.LoadList
-----
15.ReverseList 16.RemoveNthFromEnd
17.SortList    0. Exit
The max number of lists is 10.
-----
Choose your operation[0--15]:
```

1.4 系统测试

程序开发及实现环境：Win11 下使用 VScode+gcc 进行编译和调试，开发语言为 C 语言。

表1-1为正常样例测试的输入，预期结果与实际输出。

表1-2为异常样例的输入，实际输出和对输出结果的分析

通过异常用例可以看出，演示系统对表不存在时对除创建表以外的操作和对查找表中没有的元素的前驱、后继或对无后继、无前驱的元素要求获取后继、前驱的操作，以及在表存在时要求读取文件的操作的判定能力，可见演示系统能够识别异常样例。

表 1-1 正常样例测试

函数	输入	实际输出	预期结果
初始化表	1	1 线性表创建成功	线性表创建成功
销毁表	2	2 线性表销毁成功	线性表销毁成功
销毁表	2	2 线性表不存在	线性表不存在
初始化表	1	1 线性表创建成功	线性表创建成功
线性表判空	4	4 线性表为空	线性表为空
插入元素	10 1 1	10 1 1 插入成功	插入成功
插入元素	10 2 5	10 2 5 插入成功	插入成功
插入元素	10 3 7	10 3 7 插入成功	插入成功
求表长	5	5 线性表长度为3	线性表长度为 3
获取元素	6 2	6 想要获取第几个元素: 2 获取元素为5	获取元素为 5
查找元素	7 7	7 请输入元素: 7 元素位置为3	元素位置为 3
查找前驱	8 5	8 请输入元素: 5 元素的前驱为1	元素的前驱为 1
查找后继	9 5	9 请输入元素: 5 元素的后继为7	元素的后继为 7
遍历	12	12 1 5 7 遍历完成	1 5 7 遍历完成
删除元素	11 2	11 想要删除第几个元素: 2 删除成功, 被删除的元素为: 5	删除元素 5
保存文件	13 123	13 请输入文件名: 123 保存文件成功	保存文件成功
销毁表	2	2 线性表销毁成功	线性表销毁成功
读取文件	17 F" :/shiyan	14 请输入文件名:123 读入文件成功	读入文件成功
遍历链表	12	12 1 7 遍历完成	1 7 遍历完成

表 1-2 异常样例测试

异常样例	输入	实际输出	结果分析
销毁表	2	2 线性表不存在	线性表不存在
清空表	3	3 线性表不存在	线性表不存在
获取元素	6	6 线性表不存在	线性表不存在
查找后继	9 3	9 请输入元素: 3 查找后继元素失败	元素 3 没有后继元素
插入元素	10 5 5	10 5 5 插入位置不合法	插入元素的位置不合法
读取文件	17	14 该线性表已存在, 不可读入	线性表已存在

1.5 实验小结

本次实验让我对基于链式存储结构的线性表有更进一步的了解。

演示系统的搭建，让我体会到了如何搭建一个可以调用不同模块的系统。

在编写插入和删除乃至翻转链表的函数时，如何有条理地更改指针的指向是一大难点。通过这次实验，我深刻的感受到了赋值顺序对程序的巨大影响。

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

2 基于邻接表的图实现

2.1 问题描述

实验要求：

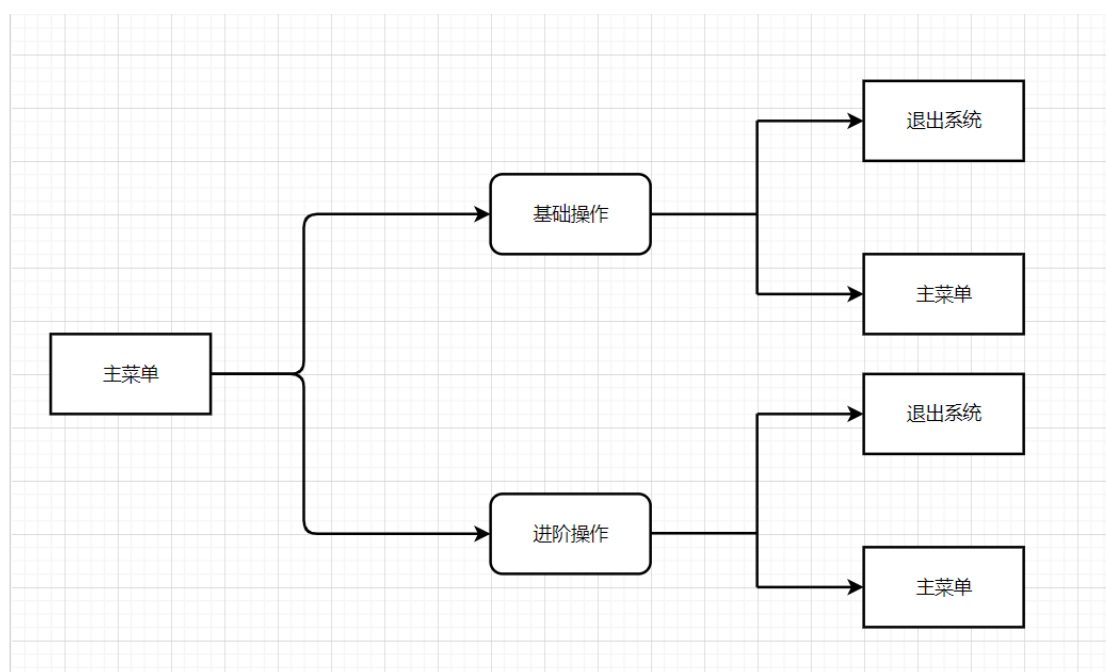
1. 实现对图的基本操作；
2. 选择性实践对图的进阶操作；
3. 设计演示系统。

通过实验达到：

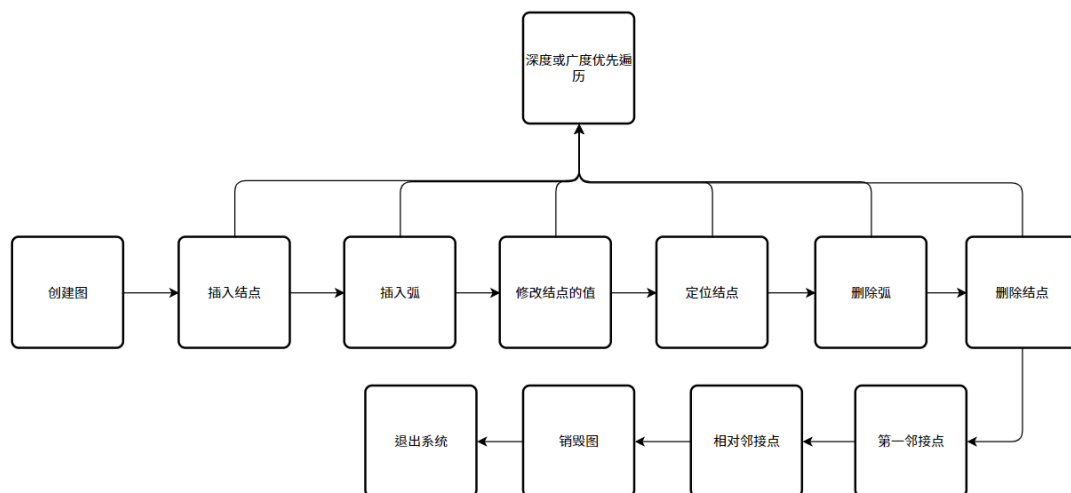
1. 加深对图的概念、基本运算的理解；
2. 熟练掌握图的逻辑结构与物理结构的关系；
3. 以邻接表作为物理结构，熟练掌握图基本运算的实现。

2.2 系统设计

2.2.1 演示系统菜单的组织架构



演示系统分为基本操作和进阶操作两部分，以一个菜单为主界面，以序号 1 至 12 表示创建，销毁，清空，插入，删除，遍历等基本操作，以序号 13 至 17 表示保存为文件，求最短通路，求距某顶点距离小于 d 的顶点，对顶点进行修改等进阶操作，以序号 0 表示退出演示系统，根据序号调用函数执行操作。



2.2.2 ADT 数据结构设计

对图数据类型的定义如下：

```
1 typedef int status ;
2 typedef int KeyType;
3 typedef enum {DG,DN,UDG,UDN} GraphKind;
4
5 typedef struct {
6     KeyType key;
7     char others [20];
8 } VertexType; //顶点类型定义
9
10 typedef struct ArcNode { //表结点类型定义
11     int adjvex; //顶点位置编号
12     struct ArcNode *nextarc; //下一个表结点指针
13 } ArcNode;
14
```

```
15 typedef struct VNode{//头结点及其数组类型定义
16     VertexType data; //顶点信息
17     ArcNode *firstarc; //指向第一条弧
18 } VNode,AdjList[MAX_VERTEX_NUM];
19
20 typedef struct { //邻接表的类型定义
21     AdjList vertices; //头结点数组
22     int vexnum,arcnum; //顶点数、弧数
23     GraphKind kind; //图的类型
24 } ALGraph;
```

对一些常量的定义如下：

```
1 #define TRUE 1
2 #define FALSE 0
3 #define OK 1
4 #define ERROR 0
5 #define INFEASIBLE -1
6 #define OVERFLOW -2
7 #define MAX_VERTEX_NUM 20
```

2.2.3 创建图

输入：图 G（未知状态），顶点集 V[]，边集 VR[]

输出：函数执行状态

算法的思想描述：定义 vexnum=0,arcnum=0, 分别记录顶点和边的数目。如若当前顶点序列的关键字不为-1 执行 while 循环：如果当前关键字未出现则更新标记数组并继续，否则返回 ERROR, 在邻接表添加新顶点，令表头结点为 NULL, 更新顶点数，检查是否超过最大数目 MAXVERTEXNUM，超过则返回 ERROR。循环结束如果 vexnum=0, 即没有顶点，则返回 ERROR, 否则令 G.vexnum=vexnum。当当前关系序列不为 (-1,-1) 时执行 while 循环：用 for 循环遍历邻接表，查找关系序列相应顶点

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

```
2  /*根据V和VR构造图T并返回OK，如果V和VR不正确，返回ERROR
3  如果有相同的关键字，返回ERROR。此题允许通过增加其它函数辅助实现本关任务*/
4  {
5      int i=0,j=0;
6      while(V[i].key!=-1)
7      {
8          for( int k=0;k<i;k++)
9              if(V[i].key==V[k].key)
10                 return ERROR;
11
12         if(i>=MAX_VERTEX_NUM)
13             return ERROR;
14         G.vertices[i].data=V[i]; //赋值
15         G.vertices[i].firstarc=NULL;
16         i++;
17     }
18     if(i==0)
19         return ERROR;
20     G.vexnum=i; //顶点数
21
22     while(VR[j][0]!=-1)
23     {
24         int a=Locate(G,VR[j][0]),b=Locate(G,VR[j][1]);
25         if(!(a>=0&&b>=0))
26             return ERROR;
27         ArcNode *q=(ArcNode*)malloc(sizeof(ArcNode));
28         q->adjvex=b;
29         q->nextarc=G.vertices[a].firstarc ; //头插法
30         G.vertices[a].firstarc=q;
31         q=(ArcNode*)malloc(sizeof(ArcNode));
32         q->adjvex=a;
```

```
33     q->nextarc=G.vertices[b].firstarc ;//头插法
34     G.vertices[b].firstarc =q;
35     j++;
36 }
37 G.arcnum=j;
38 return OK;
39 }
```

时间复杂度: $O(n)$

空间复杂度: $O(n)$

2.2.4 销毁图

输入: 图 G

输出: 函数执行状态

算法的思想描述: 遍历所有弧结点

```
92 status DestroyGraph(ALGraph &G)
93 /*销毁无向图G,删除G的全部顶点和边*/
94 {
95     ArcNode *p=NULL,*q=NULL;
96     for(int k=0;k<G.vexnum;k++)
97     {
98         if(G.vertices[k].firstarc !=NULL)//释放边
99         {
100             p=G.vertices[k].firstarc ;
101             while(p!=NULL)
102             {
103                 q=p->nextarc;//释放
104                 free(p);
105                 p=q;
106             }
107             G.vertices[k].firstarc =NULL;
```

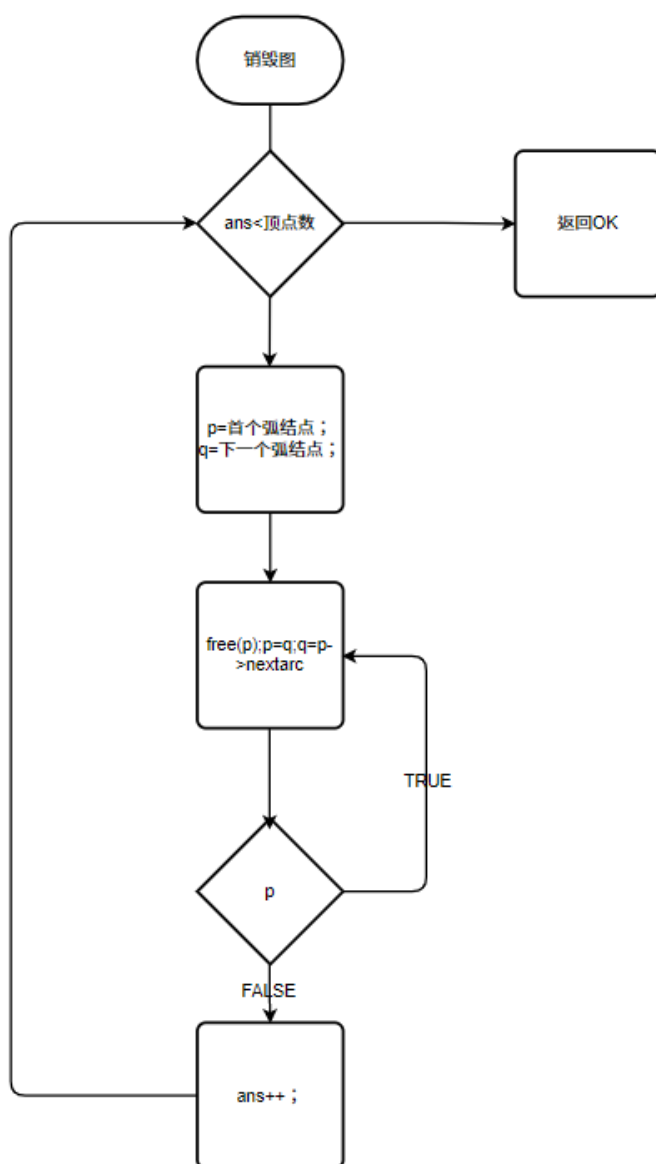
```
108     }  
109 }  
110 G.arcnum=0;  
111     G.vexnum=0;  
112     return OK;  
113 }
```

算法处理步骤：

- 1) 用两个临时变量分别记录首节点与下一结点。
- 2) 删除首个顶点然后两个结点开始移动，遍历完一个顶点的所有邻接点。
- 3) 重复直至所有结点都遍历过。
- 4) 删除成功，返回 OK。

时间复杂度： $O(n)$

空间复杂度： $O(1)$



2.2.5 定位顶点

输入: 图 G, 要查找顶点的关键字

输出: 要查找顶点的位序

算法的思想描述: 用 for 循环遍历邻接表, 如果当前顶点关键字与所找关键字相等, 则返回当前顶点序号。如果没找到, 返回-1。

```

1  int LocateVex(ALGraph G,KeyType u)
2  //根据u在图G中查找顶点, 查找成功返回位序, 否则返回-1;
3  {
    
```



```
4     int i=0;
5     for(i;i<G.vexnum;i++)
6         if(G.vertices[i].data.key==u) //找到
7             return i;
8
9     return -1;
10 }
```

时间复杂度: $O(n)$

空间复杂度: $O(1)$

2.2.6 修改顶点

输入: 图 G , 要修改结点的关键字, 以及要修改成的值 $value$

输出: 函数执行状态

算法的思想描述: 寻找赋值顶点并赋值

```
114 status PutVex(ALGraph &G,KeyType u,VertexType value)
115 //根据u在图G中查找顶点, 查找成功将该顶点值修改成value, 返回OK;
116 //如果查找失败或关键字不唯一, 返回ERROR
117 {
118     int i=0;
119     for (; i<G.vexnum;i++)
120         if(G.vertices[i].data.key==u)
121         {
122             for(int k=0;k<G.vexnum;k++)
123                 // if(G.vertices[k].data.key==value.key) //找到
124                 // return ERROR;
125
126             G.vertices[i].data=value;
127             return OK;
128         }
129 }
```

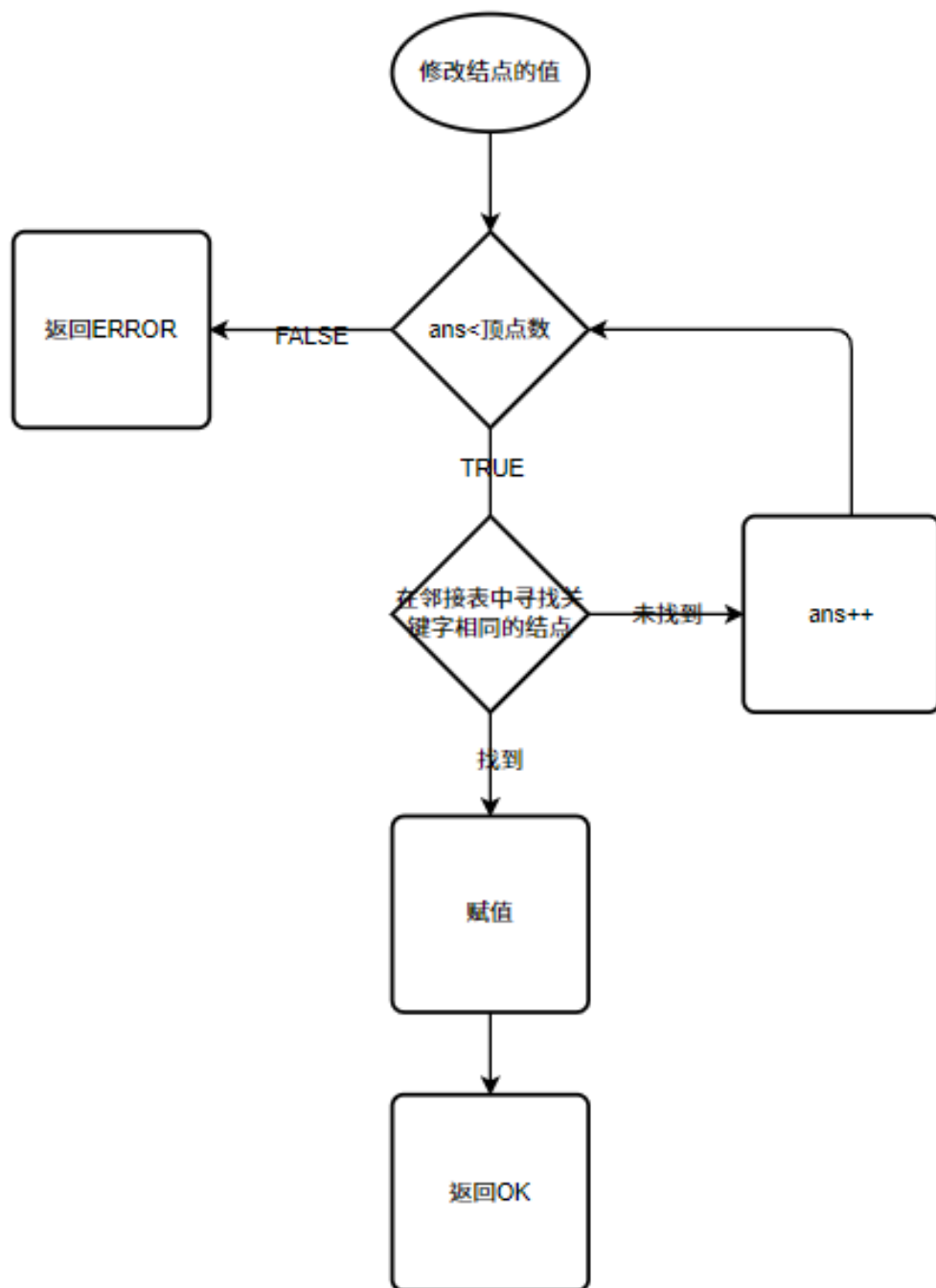
```
130     return ERROR;  
131 }
```

算法处理步骤：

- 1) 定位需要赋值的顶点的位置。
- 2) 将该顶点的关键字和值域重新赋值。
- 3) 修改成功，返回 OK。

时间复杂度： $O(n)$

空间复杂度： $O(1)$



2.2.7 第一邻接点

输入: 图 G, 要查找邻接点的顶点的关键字

输出: 第一邻接点的位序

算法思想描述: 调用定位函数查找关键字为 u 的结点。如果 $i == G.vexnum \parallel G.vertices[i].firstarc == NULL$, 即没找到该点或该点无邻接点, 则返回-1。否则返回 $G.vertices[i].firstarc \rightarrow adjvex$ 。

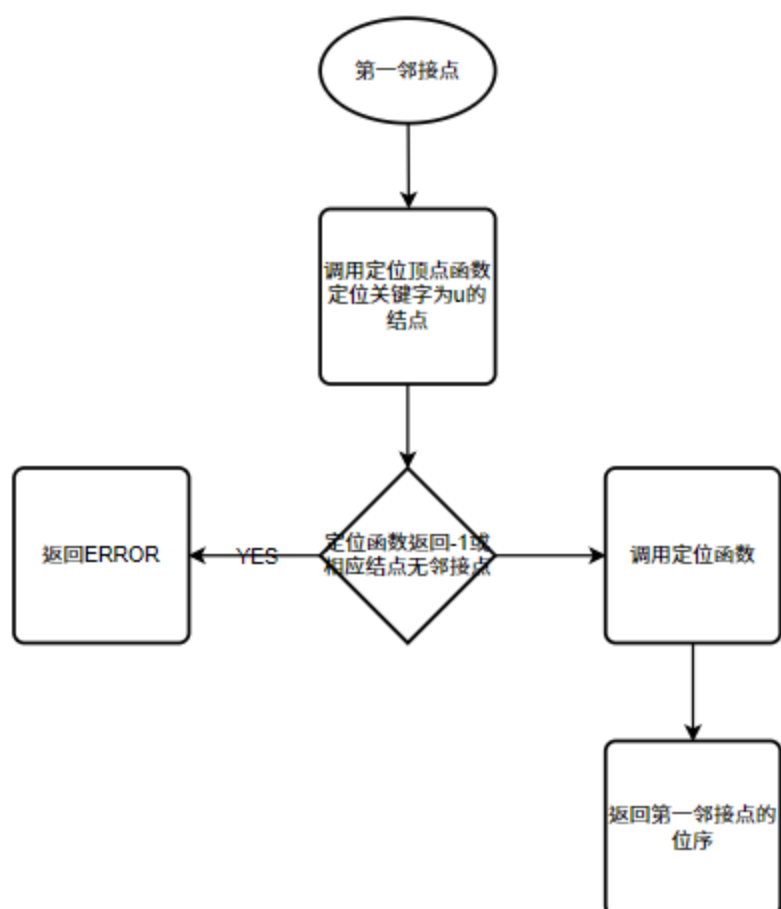
```
132 int FirstAdjVex(ALGraph G,KeyType u)
133 //根据u在图G中查找顶点, 查找成功返回顶点u的第一邻接顶点位序, 否则返回-1;
134 {
135     int i=0;
136     for (; i<G.vexnum;i++)
137         if (G.vertices[i].data.key==u&&G.vertices[i].firstarc )
138             return G.vertices[i].firstarc ->adjvex; //找到
139     return -1;
140 }
```

算法处理步骤:

- 1) 定位关键字为 v 的顶点的位置.
- 2) 若下一个弧结点不为空则返回它否则返回-1。

时间复杂度: $O(n)$

空间复杂度: $O(1)$



2.2.8 下一邻接点

输入: 图 G , 要查找邻接点的顶点的关键字

输出: 下一邻接点的位序

算法的思想描述: 调用定位函数查找关键字为 u 的结点。如果 $i == G.vexnum$, 返回 ERROR。用 while 循环遍历该顶点的所有邻接点, 找到另一个顶点, 用 p 指向该邻接点。如果 $p == NULL \parallel p \rightarrow nextarc == NULL$, 即 v 与 w 不相邻, 或 v 相对 w 无下一邻接点, 返回 -1。否则返回 $return p \rightarrow nextarc \rightarrow adjvex$ 。

```

1 int NextAdjVex(ALGraph G,KeyType v,KeyType w)
2 //v对应G的一个顶点,w对应v的邻接顶点; 操作结果是返回v的 (相对于w) 下一个邻接顶点的位序
3 {
    
```

```
4   int i=LocateVex(G,v);
5   int j=LocateVex(G,w);
6   while(i!=-1&& j!=-1)
7   {
8       if(G.vertices[i].firstarc ==NULL)
9           return -1;
10      else
11      {
12          ArcNode *p=G.vertices[i].firstarc ;//找到
13          while(p->adjvex!=j&& p)
14              p=p->nextarc;
15
16          if(p->nextarc)
17              return p->nextarc->adjvex;
18          else
19              return -1;
20      }
21  }
22  return -1;
23 }
```

时间复杂度: $O(n)$

空间复杂度: $O(1)$

2.2.9 插入顶点

输入: 图 G, 要插入顶点的关键字

输出: 函数执行状态

算法的思想描述: 如果 $G.vexnum == MAXVERTEXNUM$, 返回 ERROR。如若不然使用 LocateVex 查找要插入结点的关键字。如果要插入顶点的关键字已出现, 则返回 ERROR。否则插入新顶点, 更新顶点数, 返回 OK。

```
1 status InsertVex (ALGraph *G,VertexType v)
```

```
2 //在图G中插入顶点v, 成功返回OK, 否则返回ERROR
3 {
4     if (LocateVex(*G, v.key) >= 0)
5         return ERROR;
6     if ((*G).vexnum == MAX_VERTEX_NUM)
7         return ERROR;
8     (*G).vertices[(*G).vexnum].data = v; //赋值
9     (*G).vertices[(*G).vexnum].firstarc = NULL;
10    (*G).vexnum++;
11    return OK;
12 }
```

时间复杂度: $O(n)$

空间复杂度: $O(1)$

2.2.10 删除顶点

输入: 图 G, 要删除的顶点的关键字

输出: 函数的执行状态.

算法思想的描述: 寻找所要删除的顶点, 记录它所连接的所有弧结点, 在其他顶点中将所有的弧结点删除

```
1 status DeleteVex(ALGraph &G, KeyType v)
2 //在图G中删除关键字v对应的顶点以及相关的弧, 成功返回OK, 否则返回ERROR
3 {
4     int i = LocateVex(G, v), j = -1;
5     if (i == -1)
6         return ERROR;
7     if (G.vexnum == 1 || G.vexnum == 0)
8         return ERROR;
9     ArcNode *p = G.vertices[i].firstarc;
10    ArcNode *q = NULL;
11    ArcNode *temp = NULL;
```

```
12 while(p)
13 {
14     j=p->adjvex;
15     q=G.vertices[j].firstarc;
16     if(q->adjvex==i)
17     {
18         temp=q;
19         G.vertices[j].firstarc=q->nextarc;
20         free(temp);
21     }
22     else
23     {
24         while(q->nextarc->adjvex!=i)
25             q=q->nextarc;
26
27         temp=q->nextarc;
28         q->nextarc=temp->nextarc; //删除
29         free(temp);
30     }
31     temp=p->nextarc;
32     free(p);
33     p=temp;
34     G.arcnum--;
35 }
36 for(int k=i; k<G.vexnum; k++)
37     G.vertices[k]=G.vertices[k+1]; //删除
38
39 G.vexnum--;
40 for(int k=0; k<G.vexnum; k++)
41 {
42     p=G.vertices[k].firstarc;
```



```
43         while(p!=NULL)
44         {
45             if(p->adjvex>i)
46                 p->adjvex--;
47             p=p->nextarc;
48         }
49     }
50     return OK;
51 }
```

时间复杂度: $O(n*n)$

空间复杂度: $O(1)$

2.2.11 插入弧

输入: 图 G , 和顶点关键字类型相同的给定值 v, w

输出: 函数执行状态

算法的思想描述: 查找 v, w 的关键字, 如果其中任意一个未找到, 则返回 ERROR, 否则分别在这两个关键字后的链表中添加相应弧, 返回 OK。

```
142 status InsertArc (ALGraph &G,KeyType v,KeyType w)
143 //在图G中增加弧<v,w>, 成功返回OK,否则返回ERROR
144 {
145     int i=LocateVex(G,v);
146     int j=LocateVex(G,w);
147     if(i==-1||j==-1)
148         return ERROR;
149     if(LocateArc(G,v,w)!=ERROR)
150         return ERROR;
151     ArcNode *p=(ArcNode*)malloc(sizeof(ArcNode));//插入<v,w>
152     p->adjvex=j;
153     p->nextarc=G.vertices[i].firstarc ;
154     G.vertices[i].firstarc =p;
```

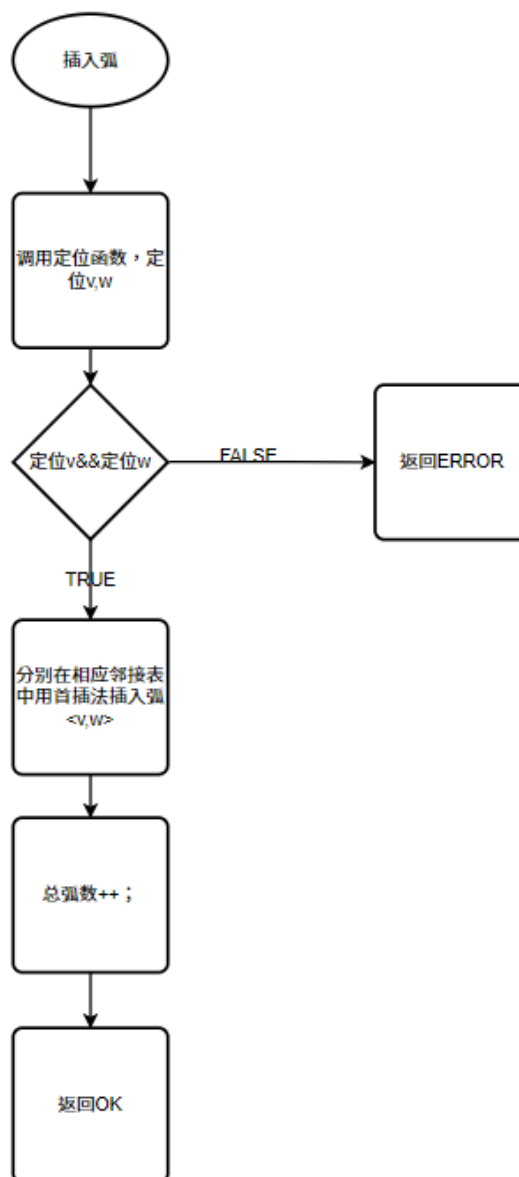
```
155     p=(ArcNode*)malloc(sizeof(ArcNode));
156     p->adjvex=i;
157     p->nextarc=G.vertices[j].firstarc ;
158     G.vertices[j].firstarc =p;
159     G.arcnum++;
160     return OK;
161 }
```

算法处理步骤:

- 1) 寻找所要插入的结点 v, w
- 2) 在 v 和 w 中分别用首插法
- 3) 总弧数加 1。

时间复杂度: $O(n)$

空间复杂度: $O(1)$



2.2.12 删除弧

输入: 图 G ，和顶点关键字类型相同的给定值 v, w

输出: 函数的执行状态。

算法的思想描述: 调用函数查找 v, w 的关键字，如果其中任意一个未找到，则返回 ERROR，否则分别在这两个关键字后的链表中寻找并删除对应结点的弧，返回 OK。

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

```
163 //在图G中删除弧<v,w>, 成功返回OK,否则返回ERROR
164 {
165     if (LocateArc(G,v,w)==ERROR)
166         return ERROR;
167     int i=LocateVex(G,v);
168     int j=LocateVex(G,w);//删除<v,w>
169     ArcNode *p=G.vertices[i]. firstarc ;//删除<v,w>
170     ArcNode *temp=NULL;
171     if (p->adjvex==j)
172     {
173         temp=p;
174         G.vertices [ i ]. firstarc =p->nextarc; //删除<v,w>
175         free (temp);
176     }
177     else
178     {
179         while(p->nextarc->adjvex!=j)
180             p=p->nextarc;
181
182         temp=p->nextarc;
183         p->nextarc=p->nextarc->nextarc; //删除<v,w>
184         free (temp);
185     }
186     p=G.vertices [ j ]. firstarc ;//删除<v,w>的对称弧<w,v>
187     temp=NULL; //删除<v,w>的对称弧<w,v>
188     if (p->adjvex==i)
189     {
190         temp=p;
191         G.vertices [ j ]. firstarc =p->nextarc;
192         free (temp);
193     }
```

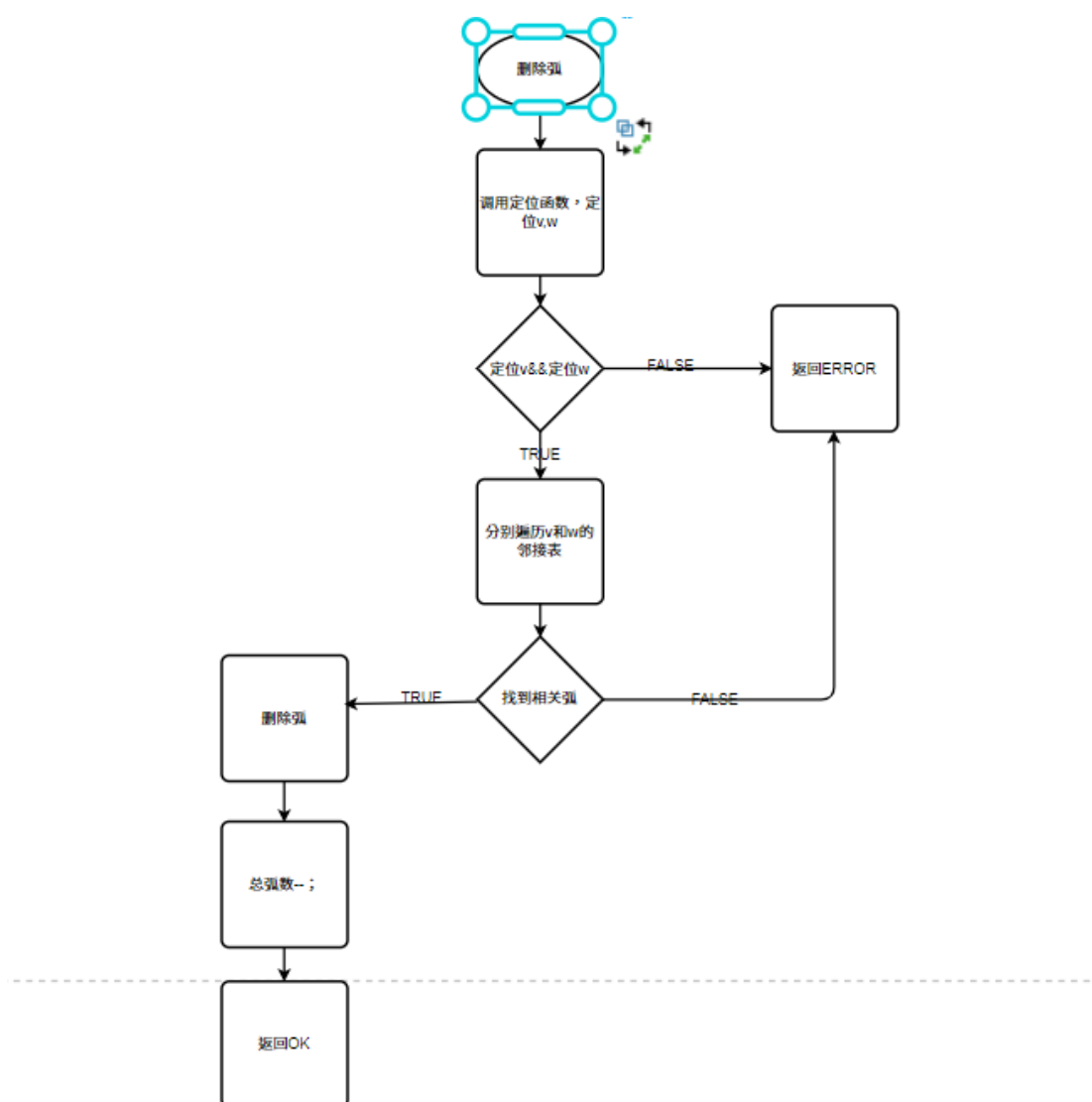
```
194     else
195     {
196         while(p->nextarc->adjvex!=i)
197             p=p->nextarc;
198
199         temp=p->nextarc;
200         p->nextarc=p->nextarc->nextarc;
201         free(temp);
202     }
203     G.arcnum--;
204     return OK;
205 }
```

算法处理步骤:

- 1) 寻找所要删除的弧的信息。
- 2) 分别在 v 中删除 w 和在 w 中删除 v 。
- 3) 若删除成功则返回 OK。

时间复杂度: $O(n)$

空间复杂度: $O(1)$



2.2.13 深度优先遍历

输入: 图 G

输出: 函数执行状态

算法的思想描述: 定义标记数组用于标记顶点是否已访问; 递归地从第一个顶点开始访问其余顶点, 每次访问时更新标记数组, 以确保每个节点只被访问一次; 返回 OK。

```

206 void dfs(ALGraph &G,KeyType v,void(*visit)(VertexType))
207 {
208     visited[v]=TRUE;
    
```

```
209     int x=LocateVex(G,v);
210     visit (G.vertices [x].data ); //访问顶点v
211     for( int w=FirstAdjVex(G,v);w>=0;w=NextAdjVex(G,v,G.vertices[w].data.key))
212         if(! visited [G.vertices [w].data .key])
213             dfs(G,G.vertices [w].data .key, visit ); //递归调用
214 }
215
216
217 status  DFSTraverse(ALGraph &G,void(*visit)(VertexType))
218 {
219     int i;
220     for(i=0;i<G.vexnum;i++)
221         visited [G.vertices [i].data .key]=FALSE; //标记数组记录关键字
222
223     for(i=0;i<G.vexnum;i++)
224         if(! visited [G.vertices [i].data .key]) //若未访问
225             dfs(G,G.vertices [i].data .key, visit );
226
227     return OK;
228 }
```

算法处理步骤:

- 1) 构造一个对一个顶点所有邻接结点的递归函数 dfs
- 2) 依次读取每个顶点并用标记数组记录已读取的顶点
- 3) 用 visit 函数将深度遍历的结点信息逐个打印。

时间复杂度: $O(n)$

空间复杂度: $O(1)$

2.2.14 广度优先遍历

输入: 图 G

输出: 函数的执行状态

算法思想描述: 利用队列这一数据结构, 保存每一次广搜的所有弧结点并将他们在下一次广搜时访问所有未被读取的首节点。

```
1  status  InitQueue(Linkqueue &Q)
2  {
3      Q.front=Q.rear=(QNode *)malloc(sizeof(QNode));
4      if (!Q.front)
5          return ERROR;
6      Q.front->next=NULL; //头结点指针域置空
7      return OK;
8  }
9
10
11 status  QueueEmpty(Linkqueue Q)
12 {
13     if (Q.front==Q.rear)
14         return TRUE; //队列为空
15     else
16         return FALSE;
17 }
18
19
20 status  enqueue(Linkqueue &Q, VertexType value)
21 {
22     Queue p=(Queue)malloc(sizeof(QNode));
23     if (!p)
24         return ERROR;
25     p->data=value;
26     p->next=NULL; //新结点指针域置空
27     Q.rear->next=p;
28     Q.rear=p;
29     return OK;
```



```
30 }
31
32
33 status deQueue(Linkqueue &Q,VertexType &value)
34 {
35     if(Q.front==Q.rear)
36         return ERROR;
37     Queue p=Q.front->next;
38     value=p->data;
39     Q.front->next=p->next;
40     if(Q.rear==p)
41         Q.rear=Q.front;
42     free(p);
43     return OK;
44 }
45
46
47 status BFSTraverse(ALGraph &G,void (*visit)(VertexType))
48 //对图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次
49 {
50     int i=0,j;
51     VertexType value;
52     Linkqueue Q;
53     InitQueue(Q);
54     for(i=0;i<G.vexnum;i++)
55         visited[G.vertices[i].data.key]=FALSE; //标记数组记录关键字
56
57     for(i=0;i<G.vexnum;i++)
58         if(! visited[G.vertices[i].data.key])
59         {
60             visited[G.vertices[i].data.key]=TRUE; //标记已访问
```

```
61     visit (G. vertices [ i ]. data );
62     enqueue(Q,G.vertices[ i ]. data );
63     while(! QueueEmpty(Q))
64     {
65         dequeue(Q,value);
66         for( int w=FirstAdjVex(G,value.key); w>=0;w=NextAdjVex(G,value.key,G.vertices[w].da
67             if (! visited [G. vertices [w].data .key]) //如果未访问过
68             {
69                 visited [G. vertices [w].data .key]=TRUE;//标记已访问
70                 visit (G. vertices [w].data );
71                 enqueue(Q,G.vertices[w].data );
72             }
73
74     }
75 }
76
77 return OK;
78 }
```

时间的复杂度: $O(n)$.

空间的复杂度: $O(n)$.

2.3 系统实现

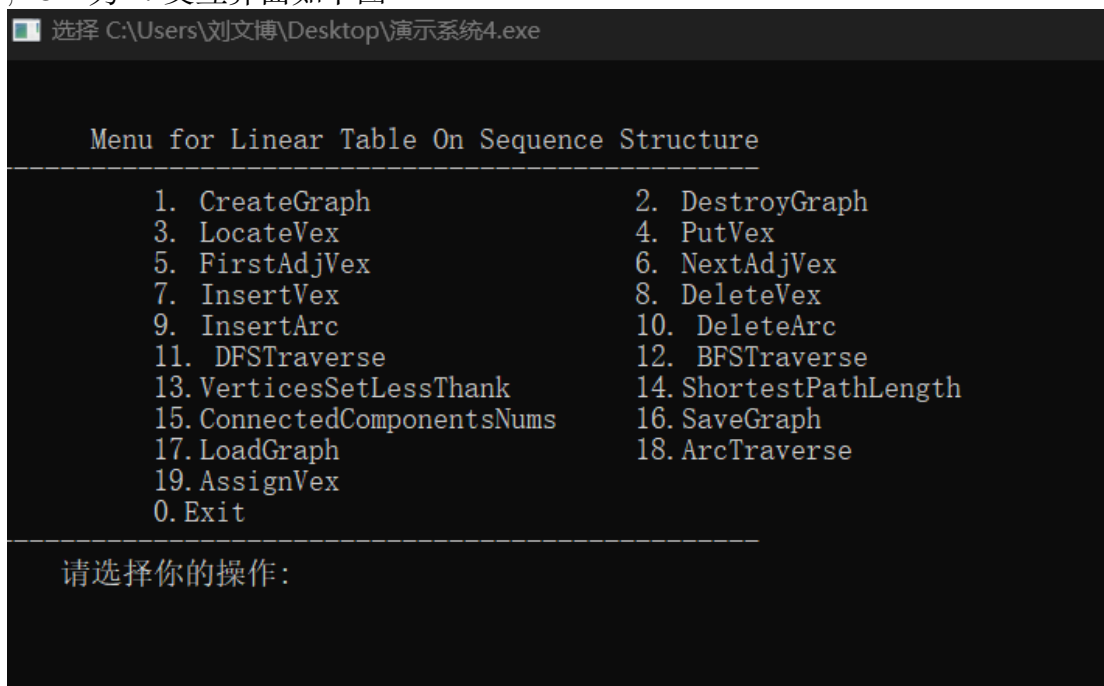
2.3.1 程序开发环境与语言

程序开发及实现环境：Win11 下使用 VScode+gcc 进行编译和调试，开发语言为 C 语言。

2.3.2 代码的组织结构

演示系统以一个菜单作为交互界面，用户通过输入命令对应的编号来调用相应的函数来实现创建图，销毁图，清空图，插入顶点，删除顶点，遍历等基本操作，以及保存为文件，求最短通路，求距某顶点距离小于 d 的顶点，对顶点进行修改等进阶操作。

程序主函数为一个 switch 结构，根据输入的数字，执行不同的语句，进而调用不同的函数，宏定义函数返回值 ERROR 为 0, INFEASIBLE 为 -1, OVERFLOW 为 -2, OK 为 1. 交互界面如下图



```
选择 C:\Users\刘文博\Desktop\演示系统4.exe

Menu for Linear Table On Sequence Structure
-----
1. CreateGraph          2. DestroyGraph
3. LocateVex            4. PutVex
5. FirstAdjVex          6. NextAdjVex
7. InsertVex            8. DeleteVex
9. InsertArc            10. DeleteArc
11. DFSTraverse          12. BFSTraverse
13. VerticesSetLessThan 14. ShortestPathLength
15. ConnectedComponentsNums 16. SaveGraph
17. LoadGraph           18. ArcTraverse
19. AssignVex
0. Exit
-----

请选择你的操作:
```

2.4 系统测试

程序开发及实现环境：Win11 下使用 VScode+gcc 进行编译和调试，开发语言为 C 语言。

表2-1为正常样例测试的输入，预期结果与实际输出

表 2-1 正常样例测试

函数	输入	实际输出	预期结果
创建图	5..5 6 5 7 6 7 7 8 -1 -1	请选择你的操作:1 请输入无向图的顶点数和边的权值,并以-1作为结束标志! 顶点数:5 权值:5 无向图:5 成功! 创建无向图成功!	创建无向图成功
定位顶点	8	请选择你的操作:3 请输入要查找顶点的关键字! 8 该顶点的位序是1 其值为8 集合	返回关键字为 8 的顶点的位序 1
修改顶点	6 9 有向图	请选择你的操作:4 请输入要修改顶点的关键字和修改后的关键字和值! 6 9 有向图 修改成功!	修改成功
查找第一邻接点	7	请选择你的操作:5 请输入要查找第一邻接顶点的关键字! 7 该顶点的第一邻接顶点的位序为1 其值为8 集合	该顶点的第一邻接顶点的位序为 1 其值为 8 集合
查找下一邻接点	7 8	请选择你的操作:6 请输入要查找顶点和相对顶点的关键字! 7 8 顶点7的邻接顶点相对于8的下一邻接顶点的位序是3 其值为6 无向图	顶点 7 的邻接顶点相对于 8 的下一邻接顶点的位序是 3 其值为 6 无向
深度优先遍历	11	请选择你的操作:11 深度优先遍历 5 线性表 7 二叉树 8 集合 6 无向图 end	5 线性表 7 二叉树 8 集合 6 无向图
广度优先遍历	12	请选择你的操作:12 广度优先遍历 5 线性表 7 二叉树 6 无向图 8 集合 end	5 线性表 7 二叉树 6 无向图 8 集合
销毁图	2	请选择你的操作:2 销毁成功!	销毁成功

表2-2为异常样例的输入，实际输出和对输出结果的分析

表 2-2 异常样例测试

异常样例	输入	实际输出	结果分析
创建图	5 (略) 5 6 5 7 6 7 7 8 -1 -1	请选择你的操作:1 请输入无向图的顶点数和边的权值,并以-1作为结束标志! 顶点数:5 权值:5 无向图:5 成功! 创建无向图成功!	创建无向图成功
定位顶点	10	请选择你的操作:3 请输入要查找顶点的关键字! 10 图中不存在该顶点!	找不到关键字为 10 的顶点
修改顶点	6 5 有向图	请选择你的操作:4 请输入要修改顶点的关键字和修改后的关键字和值! 6 5 有向图 修改失败!	关键字为 5 的结点已存在
查找第一邻接点	10	请选择你的操作:5 请输入要查找第一邻接顶点的关键字! 10 查找失败!	找不到关键字为 10 的顶点
销毁图	2	请选择你的操作:2 销毁成功!	销毁成功
销毁图	2	请选择你的操作:2 销毁失败, 请先创建图!	图还未创建, 无法销毁

通过异常用例可以看出，演示系统对要求插入与已有顶点关键字相同的顶点，和定位不包含在图中的顶点以及销毁还未创建的图等操作的判定能力，可见演示系统能够识别异常样例。

2.5 实验小结

本次实验让我对基于邻接表的图实现的了解更进了一步。演示系统的搭建，让我体会到了主函数和子函数的关系，以及如何搭建一个可以调用不同模块的系统。

在编写插入和删除弧的函数时，如何有效地根据关键字找到相应结点是使程序变得简洁的一大关键，在编写插入与删除弧的函数时，如果能够在之前定义定位顶点的函数并调用，将极大地省去冗杂的代码，这次实验让我对函数的工具性，模块性有了直观的感受。

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

相比之前几次的实验，图的系统实现更为复杂，进阶操作的实现需要了解一些经典的算法，本次实验让我有了较大的进步。

3 课程的收获和建议

3.1 基于链式存储结构的线性表实现

通过对基于链式存储结构的线性表实现的演示系统练习，我基本掌握了线性表的基本操作，能够根据需要调用不同的模块来灵活地使用线性表这一数据结构。

3.2 基于邻接表的图实现

数据结构这门课里最复杂的数据结构就是图，通过对基于邻接表的图实现的实验学习，我对数据结构的认识更加深入。

4 附录 A 基于顺序存储结构线性表实现的源程序

```
1  /*SequenceList.cpp*/
2
3  #include "SequenceListFunc.h"
4
5  // main function
6  int main(void)
7  {
8      SqList L;
9      L.elem=NULL; // Initialize the Sequence List
10     int op=1;
11     int flag=0;
12
13     while(op)
14     {
15         system("cls"); // Clear the screen
16
17         // Menu
18         printf("\n\n");
19         printf(" Menu for Linear Table On Sequence Structure \n");
20         printf("-----\n");
21         printf("    1. InitList      7. LocateElem\n");
22         printf("    2. DestroyList   8. PriorElem\n");
23         printf("    3. ClearList     9. NextElem \n");
24         printf("    4. ListEmpty     10. ListInsert \n");
25         printf("    5. ListLength    11. ListDelete \n");
26         printf("    6. GetElem       12. ListTraverse \n");
27         printf("    13. MaxSubArray  14. SubArrayNum\n");
28         printf("    15. SortList     16. SaveList \n");
29         printf("    17. LoadList\n");
```

```
30
31     printf ( "    0. Exit\n");
32     printf ( "-----\n");
33     printf ( "Choose your operation [0~17]:\n");
34     printf ( "Press Enter to continue after a certain operation ...\n");
35
36
37     // input the operation number
38     scanf( "%d",&op);
39
40
41     // Switch the operation
42     switch(op)
43     {
44         case 1: // InitList
45             {
46                 if( InitList (L)==OK)
47                     printf ( "Success!\n");
48                 else
49                     printf ( "Failed!\n");
50
51                 getchar (); // pause
52                 getchar ();
53                 break;
54             }
55
56         case 2: // DestroyList
57             {
58                 if( DestroyList (L)==OK)
59                     printf ( "Success!\n");
60                 else
```



```
61         printf ( "Failed!\n");
62
63         getchar (); // pause
64         getchar ();
65         break;
66     }
67
68     case 3: // ClearList
69     {
70         if ( ClearList (L)==OK)
71             printf ( "Success!\n");
72         else
73             printf ( "Failed!\n");
74
75         getchar (); // pause
76         getchar ();
77         break;
78     }
79
80     case 4: // ListEmpty
81     {
82         if (ListEmpty(L)==TRUE)
83             printf ( "The Sequence List is empty!\n");
84         else if (ListEmpty(L)==FALSE)
85             printf ( "The Sequence List is not empty!\n");
86         else
87             printf ( "Failed!\n");
88
89         getchar (); // pause
90         getchar ();
91         break;
```

```
92         }
93
94     case 5: // ListLength
95     {
96         if(ListLength(L)!=INFEASIBLE)
97         {
98             int Length=0;
99             Length=ListLength(L);
100             printf ("Success!");
101             printf ("\n The length of the Sequence List is %d .\n",Length);
102         }
103         else
104             printf ("Failed!\n");
105
106         getchar (); // pause
107         getchar ();
108         break;
109     }
110
111     case 6: // GetElem
112     {
113         int i=0;
114         ElemType e=0;
115
116         printf ("Which element(->position) do you want to get?\n");
117         scanf ("%d",&i);
118
119         if(GetElem(L,i,e)==ERROR)
120             printf ("The value of i is illegal !\n");
121         else if(GetElem(L,i,e)==INFEASIBLE)
122             printf ("Failed to get the element!\n");
```

```
123         else if(GetElem(L,i,e)==OK)
124         {
125             printf ("Success!\n");
126             printf ("The element is %d.\n",e);
127         }
128
129         getchar (); // pause
130         getchar ();
131         break;
132     }
133
134     case 7: // LocateElem
135     {
136         int i=0;
137         ElemType e=0;
138
139         printf ("Which element do you want to locate?\n");
140         scanf ("%d",&e);
141
142         i=LocateElem(L,e);
143
144         if(i==INFEASIBLE)
145             printf ("The Sequence List is not exist!\n");
146         if(i==0)
147             printf ("The element is not exist.\n");
148         if(i!=INFEASIBLE&& i!=0)
149             printf ("The element is in the No.%d position.\n",i);
150
151         getchar (); // pause
152         getchar ();
153         break;
```

```
154     }
155
156     case 8: // PriorElem
157     {
158         ElemType e=0;
159         printf ( "Which element do you wanna get the prior element?\n");
160         scanf( "%d",&e);
161         ElemType pre_e=0;
162
163         if(PriorElem(L,e,pre_e)==ERROR)
164             printf ( "The element has no prior element!\n");
165         if(PriorElem(L,e,pre_e)==INFEASIBLE)
166             printf ( "The Sequence List is not exist !\n");
167         if(PriorElem(L,e,pre_e)==OK)
168         {
169             printf ( "Success!\n");
170             printf ( "The prior element is %d .\n",pre_e);
171         }
172
173         getchar (); // pause
174         getchar ();
175         break;
176     }
177
178     case 9: // NextElem
179     {
180         ElemType e=0;
181         printf ( "Which element(->value) do you want to get the next element?\n");
182         scanf( "%d",&e);
183         ElemType next_e=0;
184         if(NextElem(L,e,next_e)==ERROR)
```

```
185         printf ( "The element has no next element !\n");
186     if (NextElem(L,e,next_e)==INFEASIBLE)
187         printf ( "The Sequence List is not exist !\n");
188     if (NextElem(L,e,next_e)==OK)
189     {
190         printf ( "Success !\n");
191         printf ( "The next element is %d .\n",next_e);
192     }
193
194     getchar (); // pause
195     getchar ();
196     break;
197 }
198
199 case 10: // ListInsert
200 {
201     int i;
202     ElemType e;
203     printf ( "format: [ serial number] [element]\n");
204     scanf ( "%d%d", &i, &e);
205     if ( ListInsert (L, i, e)==OK)
206         printf ( "Success !\n");
207     else if ( ListInsert (L, i, e)==INFEASIBLE)
208         printf ( "Sequence is empty! Initialize first !\n");
209     else
210         printf ( "Insertion failed ! Check your input !\n");
211
212     getchar (); // pause
213     getchar ();
214     break;
215 }
```

```
216
217     case 11: // ListDelete
218     {
219         int i=0,j=0;
220         ElemType e=0;
221         printf ( "Which element(->serial number) do you want to delete ?\n");
222         scanf( "%d",&i);
223         j=ListDelete (L,i ,e);
224         if(j==ERROR)
225             printf ( "The position is illegal !\n");
226         if(j==INFEASIBLE)
227             printf ( "The Sequence List is not exist !\n");
228         if(j==OK)
229         {
230             printf ( "Success !\n");
231             printf ( "The deleted element is %d .\n",e);
232         }
233
234         getchar (); // pause
235         getchar ();
236         break;
237     }
238
239     case 12: // ListTraverse
240     {
241         if(! ListTraverse (L))
242             printf ( "The Sequence List is not exist !\n");
243
244         getchar (); // pause
245         getchar ();
246         break;
```

```
247     }
248
249     case 13: // MaxSubArray
250     {
251         MaxSubArray(L);
252         printf ( "The maximum sum of the subarray is %d .\n",ans );
253
254         getchar (); // pause
255         getchar ();
256         break;
257     }
258
259     case 14: // SubArrayNum
260     {
261         int k=0;
262         printf ( "Please input the sum of the subarray .\n" );
263         scanf ( "%d",&k );
264         SubArrayNum(L,k);
265         printf ( "The number of the subarray whose sum is %d is %d .\n",k,ans );
266
267         getchar (); // pause
268         getchar ();
269         break;
270     }
271
272     case 15: // SortList
273     {
274         printf ( "The Sequence List is being sorted .\n\n" );
275         SortList (L);
276
277         getchar (); // pause
```

```
278         getchar ();
279         break;
280     }
281
282     case 16: // SaveList
283     {
284         char FileName[100];
285         printf ( "Please input the file name.\n");
286         scanf( "%s",&FileName);
287         if(SaveList(L,FileName))
288             printf ( "successfully !\n");
289         else
290             printf ( "Failed !\n");
291
292         getchar (); // pause
293         getchar ();
294         break;
295     }
296
297     case 17: // LoadList
298     {
299         if(!L.elem)
300         {
301             char FileName[100];
302             printf ( "Please input the file name.\n");
303             scanf( "%s",&FileName);
304             if(LoadList(L,FileName))
305                 printf ( "The Sequence List has been loaded successfully !\n");
306             else
307                 printf ( "Failed !\n");
308         }
```



```
309         getchar (); // pause
310         getchar ();
311         break;
312     }
313     else
314     {
315         printf ( "The Sequence List had existed !\n" );
316
317         getchar (); // pause
318         getchar ();
319         break;
320     }
321 }
322
323 case 0: // Exit
324     break;
325 }
326 }
327 printf ( "Bye!\n" );
328 return 0;
329 }
```

```
1  //SequenceListFunc.h
2
3  #include <stdio.h>
4  #include <malloc.h>
5  #include <stdlib.h>
6
7
8  #define TRUE 1
9  #define FALSE 0
10 #define OK 1
11 #define ERROR 0
12 #define INFEASIBLE -1
13 #define OVERFLOW -2
14 #define LIST_INIT_SIZE 100
15 #define LISTINCREMENT 10
16
17 typedef int status ;
18 typedef int ElemType;
19
20 typedef struct {
21     ElemType * elem;
22     int length ;
23     int listsize ;
24 } SqList; //type of Sequence List
25
26
27 // Function declaration
28 status InitList (SqList& L);
29 status DestroyList (SqList& L);
30 status ClearList (SqList&L);
31 status ListEmpty(SqList L);
```

```
32 int ListLength(SqList L);
33 status GetElem(SqList L, int i, ElemType& e);
34 status LocateElem(SqList L, ElemType e);
35 status PriorElem(SqList L, ElemType cur, ElemType &pre_e);
36 status NextElem(SqList L, ElemType cur, ElemType &next_e);
37 status ListInsert (SqList &L, int i, ElemType e);
38 status ListDelete (SqList&L, int i, ElemType& e);
39 status ListTraverse (SqList L);
40 status MaxSubArray(SqList L);
41 status SubArrayNum(SqList L, int k);
42 status SortList (SqList &L);
43 status SaveList(SqList L, char Filename []);
44 status LoadList(SqList &L, char Filename []);
45
46 int ans=0, sum=0;
47
48 // function definition
49
50 status InitList (SqList& L)
51 // 线性表L不存在，构造一个空的线性表，
52 //返回OK，否则返回INFEASIBLE。
53 {
54     if(L.elem==NULL)
55     {
56         L.elem=(ElemType*)malloc(LIST_INIT_SIZE*sizeof(ElemType));
57         L.length=0;
58         L.listsize =LIST_INIT_SIZE;
59         return OK;
60     }
61     else
62         return INFEASIBLE;
```

```
63     }
64
65
66     status DestroyList(SqList &L)
67     // 如果线性表L存在, 销毁线性表L, 释放数据元素的空间,
68     // 返回OK, 否则返回INFEASIBLE。
69     {
70         if(L.elem!=NULL)
71         {
72             free(L.elem);
73             L.elem=NULL;
74             L.length=0;
75             L.listsize =0;
76             return OK;
77         }
78         else
79             return INFEASIBLE;
80     }
81
82
83     status ClearList(SqList& L)
84     // 如果线性表L存在, 删除线性表L中的所有元素,
85     // 返回OK, 否则返回INFEASIBLE。
86     {
87         if(L.elem!=NULL)
88         {
89             int length = L.length;
90             int i;
91
92             for (i = 0; i < length; i++)
93                 L.elem[i] = 0;
```

```
94
95     L.length = 0;
96     return OK;
97 }
98 else
99     return INFEASIBLE;
100 }
101
102
103
104 status ListEmpty(SqList L)
105 // 如果线性表L存在，判断线性表L是否为空，空就返回TRUE，
106 // 否则返回FALSE；如果线性表L不存在，返回INFEASIBLE。
107 {
108     if(L.elem==NULL)
109         return INFEASIBLE;
110     else
111         if(L.length==0)
112             return TRUE;
113         else
114             return FALSE;
115 }
116
117
118 status ListLength(SqList L)
119 // 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。
120 {
121     if(L.elem!=NULL)
122         return L.length;
123     else
124         return INFEASIBLE;
```

```
125     }
126
127
128     status GetElem(SqList L, int i, ElemType &e)
129     // 如果线性表L存在，获取线性表L的第i个元素，保存在e中，返回OK；
130     //如果i不合法，返回ERROR；如果线性表L不存在，返回INFEASIBLE。
131     {
132         if (L.elem==NULL)
133             return INFEASIBLE;
134         else if (L.elem!=NULL&&(i<1||i>L.length))
135             return ERROR;
136         e=L.elem[i-1];
137         return OK;
138     }
139
140
141     int LocateElem(SqList L,ElemType e)
142     // 如果线性表L存在，查找元素e在线性表L中的位置序号并返回该序号；
143     //如果e不存在，返回0；当线性表L不存在时，返回INFEASIBLE（即-1）。
144     {
145         if (L.elem!=NULL)
146         {
147             int i=1;
148             while (L.elem[i-1]!=e&&(i-1)<=L.length-1)
149                 i++;
150
151             if (i>=1&&i<=L.length)
152                 return i;
153             else
154                 return 0;
155         }
```

```
156     else
157         return INFEASIBLE;
158     }
159
160
161     status PriorElem(SqList L,ElemType e,ElemType &pre)
162     // 如果线性表L存在，获取线性表L中元素e的前驱，保存在pre中，返回OK；
163     //如果没有前驱，返回ERROR；如果线性表L不存在，返回INFEASIBLE。
164     {
165         if(L.elem!=NULL)
166         {
167             int i=0;
168             if(L.elem[i]==e)
169                 return ERROR;
170             else
171             {
172                 i=1;
173                 while(i<L.length)
174                 {
175                     if(L.elem[i]==e)
176                     {
177                         pre=L.elem[i-1];
178                         return OK;
179                     }
180                     else
181                         i++;
182                 }
183                 return ERROR;
184             }
185         }
186         else
```

```
187         return INFEASIBLE;
188     }
189
190
191     status NextElem(SqList L,ElemType e,ElemType &next)
192     // 如果线性表L存在, 获取线性表L元素e的后继, 保存在next中, 返回OK;
193     //如果没有后继, 返回ERROR; 如果线性表L不存在, 返回INFEASIBLE。
194     {
195         if(L.elem!=NULL)
196         {
197             int i=L.length-1;
198             if(L.elem[i]==e)
199                 return ERROR;
200             else
201             {
202                 i=i-1;
203                 while(i>=0)
204                 {
205                     if(L.elem[i]==e)
206                     {
207                         next=L.elem[i+1];
208                         return OK;
209                     }
210                     i--;
211                 }
212                 if(i<0)
213                     return ERROR;
214             }
215         }
216         else
217             return INFEASIBLE;
```



```
218     return ERROR;
219 }
220
221
222 status ListInsert (SqList &L,int i,ElemType e)
223 // 如果线性表L存在，将元素e插入到线性表L的第i个元素之前，返回OK；
224 // 当插入位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。
225 {
226     if(L.elem==NULL)
227         return INFEASIBLE;
228     if(L.length>=L.listsize )
229     {
230         ElemType *newbase=(ElemType*)realloc(L.elem,(L.listsize+LISTINCREMENT)*sizeof(ElemType));
231         L.elem=newbase;
232         L.listsize +=LISTINCREMENT;
233     }
234     if(L.elem!=NULL)
235     {
236         if(L.length==0)
237         {
238             L.elem[0]=e;
239             L.length++;
240             return OK;
241         }
242         if(i>=1&& i<=L.length+1)
243         {
244             for( int j=L.length; j>=i; j--)
245                 L.elem[j]=L.elem[j-1];
246
247             L.elem[i-1]=e;
248             L.length++;
```

```
249         return OK;
250     }
251 }
252 return ERROR;
253 }
254
255
256 status ListDelete (SqList &L,int i,ElemType &e)
257 // 如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回OK；
258 // 当删除位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。
259 {
260     if(L.elem)
261     {
262         if((i < 1)|| ( i>L.length ))
263             return ERROR;
264
265         int *p,*q;
266
267         p=&(L.elem[i-1]);
268         e=*p;
269         q=L.elem+L.length-1;
270
271         for(++p;p<=q;++p)
272             *(p-1)=*p;
273
274         L.length--;
275         return OK;
276     }
277     else
278         return INFEASIBLE;
279 }
```

```
280
281
282     status ListTraverse (SqList L)
283     // 如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，
284     // 返回OK；如果线性表L不存在，返回INFEASIBLE。
285     {
286         if(L.elem==NULL)
287             return INFEASIBLE;
288         if(L.length>=L. listsize )
289         {
290             ElemType *newbase=(ElemType*)realloc(L.elem,(L. listsize+LISTINCREMENT)*sizeof(ElemType));
291             L.elem=newbase;
292             L. listsize +=LISTINCREMENT;
293         }
294         if(L.elem)
295         {
296             if(L.length)
297             {
298                 printf ( "=====Elements in Sequence List=====\n\n");
299                 for ( int i=0;i<L.length;i++)
300                 {
301                     if(i != L.length-1)
302                         printf ( "%d ",L.elem[i ]);
303                     else
304                         printf ( "%d\n\n",L.elem[i ]);
305                 }
306                 printf ( "=====\n");
307             }
308             return OK;
309         }
310         return L.length ;
```

```
311     }
312
313
314     status MaxSubArray(SqList L)
315     // 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
316     // （子数组最少包含一个元素），操作结果是其最大和；
317     {
318         sum=0;
319         ans=0;
320         if (!L.elem)
321             return ERROR;
322         for (int i=0; i<L.length; i++)
323         {
324             sum+=L.elem[i];
325             if (ans<sum)
326                 ans=sum;
327         }
328         return OK;
329     }
330
331
332     status SubArrayNum(SqList L, int k)
333     // 初始条件是线性表L已存在且非空，
334     // 请找出一个具有和为k的连续子数组的个数，操作结果是其个数；
335     {
336         ans=0;
337         if (!L.elem)
338             return ERROR;
339         for (int i=0; i<L.length; i++)
340         {
341             sum=0;
```

```
342         for( int j=i;j<L.length;j++)
343         {
344             sum+=L.elem[j];
345             if(sum==k)
346                 ans++;
347         }
348     }
349     return OK;
350 }
351
352
353 status SortList (SqList &L)
354 // 如果线性表L存在，将线性表L中的元素按从小到大的顺序排列，返回OK;
355 //如果线性表L不存在，返回INFEASIBLE。
356 {
357     if(!L.elem)
358         return ERROR;
359     int temp=0;
360
361     for( int i=0;i<L.length-1;i++)
362         for( int j=0;j<L.length-1-i;j++)
363             if(L.elem[j]>L.elem[j+1])
364                 {
365                     temp=L.elem[j];
366                     L.elem[j]=L.elem[j+1];
367                     L.elem[j+1]=temp;
368                 }
369     ListTraverse (L);
370     return OK;
371 }
372
```

```
373
374     status  SaveList(Sqlist L,char FileName[])
375     // 如果线性表L存在，将线性表L的元素写到FileName文件中，
376     //返回OK，否则返回INFEASIBLE。
377     {
378         if(L.elem==NULL)
379             return INFEASIBLE;
380         else
381         {
382             FILE *fp=fopen(FileName,"w");
383             if(!fp)
384                 return ERROR;
385
386             fprintf (fp, "%d\n",L.length);
387
388             for( int i=0;i<L.length;i++)
389                 fprintf (fp, "%d ",L.elem[i]);
390
391             fclose (fp);
392             return OK;
393         }
394     }
395
396
397     status  LoadList(Sqlist &L,char FileName[])
398     // 如果线性表L不存在，将FileName文件中的数据读入到线性表L中，
399     //返回OK，否则返回INFEASIBLE。
400     {
401
402         if(L.elem!=NULL)
403             return INFEASIBLE;
```

```
404     else
405     {
406         L.elem=(ElemType*)malloc(LIST_INIT_SIZE*sizeof(ElemType));
407         L.length=0;
408         L.listsize =LIST_INIT_SIZE;
409         FILE *fp=fopen(FileName, "r");
410         if (!fp)
411             return ERROR;
412         fscanf( fp, "%d",&L.length);
413         for( int i=0;i<L.length; i++)
414             fscanf( fp, "%d",&L.elem[i]);
415
416         fclose ( fp );
417         return OK;
418     }
419 }
```

5 附录 B 基于链式存储结构线性表实现的源程序

```
1 // LinkedList.cpp
2
3 #include "LinkedListFunc.h"
4
5 //main function
6 int main()
7 {
8     char filename[40];
9     int op=1;
10     int i, flag=0, i_num=0;
11     LinkList L[MAX_NUM];
12
13
14     for (i = 0; i<MAX_NUM; i++)
15         L[i]=NULL;
16
17     ElemType e, cur_e, pre_e, next_e;
18
19
20     while(op)
21     {
22         system("cls");
23
24         printf("\n\n");
25         printf("          Menu for Linear Table On LinkedList \n");
26         printf("          1. InitList          2. DestroyList \n");
27         printf("          3. ClearList         4. ListEmpty \n");
28         printf("          5. ListLength        6. GetElem \n");
29         printf("          7. LocateElem        8. PriorElem \n");
```



```
30 printf ( "          9. NextElem      10. ListInsert \n");
31 printf ( "          11. ListDelete   12. ListTraverse \n");
32 printf ( "          13. SaveList      14. LoadList \n");
33 printf ( "          15. ReverseList   16. RemoveNthFromEnd \n \n");
34 printf ( "          17. SortList        0. Exit \n \n");
35 printf ( "          The max number of lists is %d. \n", MAX_NUM);
36
37 printf ( "Choose your operation[0--15]: \n");
38 scanf( "%d",&op);
39
40
41 switch(op)
42 {
43 case 1: // InitList
44 if( InitList (&L[i_num])==OK)
45 {
46 printf ( "Input the filename to save the list : \n");
47 scanf( "%s", filename);
48 printf ( "Success ! \n");
49 }
50 else
51 printf ( "Failed ! \n");
52
53 getchar ();
54 getchar ();
55 break;
56
57 case 2: // DestroyList
58 if(L[i_num] == NULL)
59 {
60 printf ( "The list doesn't exist ! \n");
```

```
61
62 getchar ();
63 getchar ();
64 break;
65 }
66 if( DestroyList(&L[i_num])==OK)
67     printf ( "Success!\n");
68 else
69     printf ( "Failed!\n");
70
71 getchar ();
72 getchar ();
73 break;
74
75 case 3: // ClearList
76     if(L[i_num] == NULL)
77     {
78         printf ( "The list doesn't exist!\n");
79
80         getchar ();
81         getchar ();
82         break;
83     }
84     if( ClearList (&L[i_num])==OK)
85         printf ( "Success!\n");
86     else
87         printf ( "Failed!\n");
88
89     getchar ();
90     getchar ();
91     break;
```

```
92
93 case 4: //ListEmpty
94 if (L[i_num] == NULL)
95 {
96     printf ("The list doesn't exist !\n");
97
98     getchar ();
99     getchar ();
100 break;
101 }
102 if (ListEmpty(L[i_num]) == TRUE)
103     printf ("The file is empty!\n");
104
105 else
106     printf ("The file is not empty!\n");
107
108     getchar ();
109     getchar ();
110 break;
111
112 case 5: //ListLength
113 if (L[i_num] == NULL)
114 {
115     printf ("The list doesn't exist !\n");
116
117     getchar ();
118     getchar ();
119 break;
120 }
121 printf ("The length of the list is %d\n", ListLength(L[i_num]));
122
```

```
123  getchar ();
124  getchar ();
125  break;
126
127  case 6: //GetElem
128  if(L[i_num] == NULL)
129  {
130  printf ("The list doesn't exist !\n");
131
132  getchar ();
133  getchar ();
134  break;
135  }
136  printf ("Input the position of the element you wanna get:\n");
137  scanf ("%d",&i);
138  if(GetElem(L[i_num],i,&e)==OK)
139  printf ("The element of the position %d is %d\n",i,e);
140  else
141  printf ("The position is invalid !\n");
142
143  getchar ();
144  getchar ();
145  break;
146
147  case 7: //LocateElem
148  if(L[i_num] == NULL)
149  {
150  printf ("The list doesn't exist !\n");
151
152  getchar ();
153  getchar ();
```

```
154 break;
155 }
156 printf ( "Input the element you wanna locate:\n");
157
158 scanf( "%d",&e);
159
160 if (i=LocateElem(L[i_num],e,compare))
161     printf ( "The element %d is in the position %d\n",e,i);
162 else
163     printf ( "The element doesn't exist !\n");
164
165 getchar ();
166 getchar ();
167 break;
168
169 case 8: //PriorElem
170     if (L[i_num] == NULL)
171     {
172         printf ( "The list doesn't exist !\n");
173
174         getchar ();
175         getchar ();
176         break;
177     }
178     printf ( "Input the element you wanna get its prior element:\n");
179
180     scanf( "%d",&cur_e);
181
182     PriorElem(L[i_num],cur_e,&pre_e);
183
184     if (PriorElem(L[i_num],cur_e,&pre_e)==OK)
```

```
185 printf ( "The prior element of %d is %d\n",cur_e,pre_e);
186 else if(PriorElem(L[i_num],cur_e,&pre_e)==OVERFLOW)
187 printf ( "The element doesn't exist !\n");
188 else
189 printf ( "The element doesn't have a prior element !\n");
190
191 getchar ();
192 getchar ();
193 break;
194
195
196 case 9: //NextElem
197 if (L[i_num] == NULL)
198 {
199 printf ( "The list doesn't exist !\n");
200
201 getchar ();
202 getchar ();
203 break;
204 }
205 printf ( "Input the element you wanna get its next element:\n");
206
207 scanf( "%d",&cur_e);
208
209 if (NextElem(L[i_num],cur_e,&next_e)==OK)
210 printf ( "The next element of %d is %d\n",cur_e,next_e);
211 else if (NextElem(L[i_num],cur_e,&pre_e)==ERROR)
212 printf ( "The element doesn't exist !\n");
213 else
214 printf ( "The element doesn't have a next element !\n");
215
```

```
216 getchar ();
217 getchar ();
218 break;
219
220 case 10: // ListInsert
221 if(L[i_num] == NULL)
222 {
223 printf ("The list doesn't exist !\n");
224
225 getchar ();
226 getchar ();
227 break;
228 }
229 printf ("Input the element and its position :\n");
230 printf ("format:[ position ] [element]\n");
231
232 scanf ("%d %d",&i,&e);
233
234 if ( ListInsert (&L[i_num],i,e)==OK)
235 printf ("Success!\n");
236 else
237 printf ("Failed!\n");
238
239 getchar ();
240 getchar ();
241 break;
242
243 case 11: // ListDelete
244 if(L[i_num] == NULL)
245 {
246 printf ("The list doesn't exist !\n");
```

```
247
248 getchar ();
249 getchar ();
250 break;
251 }
252 printf ( "Input the position of the element you wanna delete:\n");
253 scanf( "%d",&i);
254 if( ListDelete(&L[i_num],i,&e)==OK)
255     printf ( "Success!\n");
256 else
257     printf ( "Failed!\n");
258
259 getchar ();
260 getchar ();
261 break;
262
263 case 12: // ListTraverse
264     if(L[i_num] == NULL)
265     {
266         printf ( "The list doesn't exist!\n");
267
268         getchar ();
269         getchar ();
270         break;
271     }
272     if( ListTraverse (L[i_num])==0)
273         printf ( "The list is empty!\n");
274
275     getchar ();
276     getchar ();
277     break;
```



```
278
279 case 13: // SaveList
280 if (L[i_num] == NULL)
281 {
282     printf ("The list doesn't exist !\n");
283
284     getchar ();
285     getchar ();
286     break;
287 }
288 if (SaveList(L[i_num], filename)==OK)
289     printf ("file named %s has been saved!\n", filename);
290
291     getchar ();
292     getchar ();
293     break;
294
295 case 14: // LoadList
296     InitList (&L[i_num]);
297
298     printf ("Input the filename you wanna load:\n");
299
300     scanf ("%s", filename);
301
302     if (LoadList(&L[i_num], filename)==OK)
303         printf ("file named %s has been loaded!\n", filename);
304     getchar ();
305     getchar ();
306     break;
307
308 case 15: // ReverseList
```

```
309 flag=ReverseList(&L[i_num]);
310 if( flag== INFEASIBLE)
311     printf ( "The list doesn't exist !\n");
312 else if( flag == ERROR)
313     printf ( "The list is empty!\n");
314 else
315     printf ( "Success!\n");
316
317 getchar ();
318 getchar ();
319 break;
320
321
322 case 16: //RemoveNthFromEnd
323     printf ( "Input the position of the element you wanna delete:\n");
324     scanf( "%d",&i);
325     flag=RemoveNthFromEnd(L[i_num],i,e);
326     if( flag== INFEASIBLE)
327         printf ( "The list doesn't exist !\n");
328     else if( flag == ERROR)
329         printf ( "The list is empty!\n");
330     else
331         printf ( "Success!\n");
332
333     getchar ();
334     getchar ();
335     break;
336
337
338 case 17: // SortList
339     flag=SortList (L[i_num]);
```

```
340 if( flag== INFEASIBLE)
341     printf ( "The list doesn't exist !\n");
342 else if( flag == ERROR)
343     printf ( "The list is empty!\n");
344 else
345     printf ( "Success!\n");
346
347 getchar ();
348 getchar ();
349 break;
350
351 case 0: // Exit
352     break;
353 }
354 }
355 printf ( "Bye!\n");
356 }
```

```
1 //LinkedListFunc.h
2
3 #include <stdio.h>
4 #include <malloc.h>
5 #include <stdlib.h>
6
7
8 #define TRUE 1
9 #define FALSE 0
10 #define OK 1
11 #define ERROR 0
12 #define INFEASIBLE -1
13 #define OVERFLOW -2
14 #define MAX_NUM 10
15 #define LIST_INIT_SIZE 100
16 #define LISTINCREMENT 10
17
18
19 typedef int status ;
20 typedef int ElemType; //type of element
21
22
23 typedef struct LNode{
24     ElemType data;
25     struct LNode *next;
26 }LNode, *LinkList;
27
28
29 FILE *fp; // file pointer
30
31
```

```
32  // initiate the functions
33  status InitList (LinkList *L);
34  status DestroyList (LinkList *L);
35  status ClearList (LinkList *L);
36  status ListEmpty(LinkList L);
37  int ListLength (LinkList L);
38  status GetElem(LinkList L, int i, ElemType *e);
39  int LocateElem(LinkList L, ElemType e, status (*compare)(ElemType a, ElemType b));
40  status compare(ElemType a, ElemType b);
41  status PriorElem(LinkList L, ElemType cur_e, ElemType *pre_e);
42  status NextElem(LinkList L, ElemType cur_e, ElemType *next_e);
43  status ListInsert (LinkList *L, int i, ElemType e);
44  status ListDelete (LinkList *L, int i, ElemType *e);
45  status ListTraverse (LinkList L);
46  status SaveList(LinkList L, char* filename);
47  status LoadList(LinkList *L, char *filename);
48  status ReverseList (LinkList *L);
49
50  status InitList (LinkList *L)
51  // initiate a list
52  {
53      *L = (LinkList)malloc( sizeof(LNode)); //malloc a node
54      if(*L == NULL)
55      {
56          exit (OVERFLOW); //malloc failed
57      }
58      (*L)->data = 0;
59      (*L)->next = NULL; //the list is empty
60      return OK;
61  }
62
```

```
63
64
65 status DestroyList(LinkList *L)
66     // destroy a list
67     {
68     LinkList p, q;
69     p = *L;
70     while(p)
71     {
72     q = p->next;
73     free(p);
74     p = q;
75     }
76     *L = NULL;
77     return OK;
78     }
79
80
81 status ClearList(LinkList *L)
82     // clear a list
83     {
84     LinkList p, q;
85     p = (*L)->next;//point to the first node
86     while(p)
87     {
88     q = p->next;
89     free(p);
90     p = q;
91     }
92     (*L)->next = NULL;
93     return OK;
```

```
94     }
95
96
97 status ListEmpty(LinkList L)
98     //judge if the list is empty
99     {
100     if(L->next)
101     return FALSE;
102     else
103     return TRUE;
104     }
105
106
107 int ListLength(LinkList L)
108     //get the length of the list
109     {
110     int i = 0;
111     LinkList p = L->next;
112     while(p)
113     {
114     i++;
115     p = p->next;
116     }
117     return i;
118     }
119
120
121
122 status GetElem(LinkList L, int i, ElemType *e)
123     //get the element of the No.i node
124     {
```

```
125 int j = 1;
126 LinkList p;
127 p = L->next;
128 while(p && j < i)
129 {
130 p = p->next;
131 ++j;
132 }
133 if(!p || j > i)
134 return ERROR;
135 *e = p->data;
136 return OK;
137 }
138
139
140 int LocateElem(LinkList L, ElemType e, status (*compare)(ElemType a, ElemType b))
141 //get the position of the element
142 {
143 int i = 0;
144 LinkList p = L->next;
145 while(p)
146 {
147 i++;
148 if((*compare)(p->data, e))
149 return i;
150 p = p->next;
151 }
152 return 0;
153 }
154
155 status compare(ElemType a, ElemType b)
```



```
156 {
157 if(a == b)
158 return TRUE;
159 else
160 return FALSE;
161 }
162
163
164
165 status PriorElem(LinkList L, ElemType cur_e, ElemType *pre_e)
166 //get the element before the cur_e
167 {
168 LinkList p = L->next;
169 if(p->data==cur_e)
170 return ERROR;
171 while(p->next != NULL && p->next->data != cur_e)
172 p = p->next;
173
174 if(p->next == NULL)
175 return OVERFLOW;
176
177 *pre_e = p->data;
178 return OK;
179 }
180
181
182
183 status NextElem(LinkList L, ElemType cur_e, ElemType *next_e)
184 //get the element after the cur_e
185 {
186 LinkList p = L->next;
```

```
187 while(p->next != NULL && p->data != cur_e)
188     p = p->next;
189
190 if(p->next == NULL && p->data != cur_e)
191     return ERROR;
192 if(p->next == NULL && p->data == cur_e)
193     return OVERFLOW;
194     *next_e = p->next->data;
195     return OK;
196 }
197
198
199
200 status ListInsert (LinkedList *L, int i, ElemType e)
201     // insert a node before the No.i node
202     {
203     int j = 1;
204     LinkedList p, q;
205     p = *L;
206     while(p && j < i)
207     {
208     p = p->next;
209     ++j;
210     }
211     if(!p || j > i)
212     return ERROR;
213
214     q = (LinkedList)malloc( sizeof(LNode));
215     if(q == NULL)
216     exit (OVERFLOW);
217
```

```
218 q->data = e;
219 q->next = p->next;
220 p->next = q;
221 return OK;
222 }
223
224
225
226 status ListDelete (LinkList *L, int i, ElemType *e)
227     // delete the No.i node
228     {
229     int j = 1;
230     LinkList p, q;
231     p = *L;
232     while(p->next && j < i)
233     {
234     p = p->next;
235     ++j;
236     }
237     if (!(p->next) || j > i)
238     return ERROR;
239
240     q = p->next;
241     p->next = q->next;
242     *e = q->data;
243     free(q);
244
245     return OK;
246     }
247
248
```

```
249
250 status ListTraverse (LinkedList L)
251     // traverse the list
252     {
253         LinkedList p = L;
254         if (!p)
255             return INFEASIBLE;
256         else if (!p->next)
257             return ERROR;
258         else
259             {
260                 while(p)
261                 {
262                     if(p->data!=0)
263                         printf ( "%d ",p->data);
264                     p = p->next;
265                 }
266                 return OK;
267             }
268     }
269
270
271
272 status SaveList(LinkedList L, char* filename)
273     //save the list to a file
274     {
275         LinkedList p = L->next;
276         int listsize =LIST_INIT_SIZE;
277         if ((fp = fopen(filename, "w")) == NULL)
278             {
279                 printf ( "File open error !\n");
```

```
280 return ERROR;
281 }
282 fprintf (fp, "%d ", ListLength(L));
283 fprintf (fp, "%d ", listsize );
284 while(p)
285     {
286     fprintf (fp, "%d ", p->data);
287     p = p->next;
288     }
289 fclose (fp);
290 return OK;
291 }
292
293
294
295 status LoadList(LinkList *L, char *filename)
296     //load the list from a file
297     {
298     int i = 1, length = 0, listsize ;
299     ElemType e;
300     if ((fp = fopen(filename, "r")) == NULL)
301     {
302     printf ("File open error !\n");
303     return ERROR;
304     }
305     fscanf(fp, "%d ", &length);
306     fscanf(fp, "%d ", &listsize );
307     fscanf(fp, "%d ", &e);
308     while(i<=length)
309     {
310     ListInsert (L,i,e);
```

```
311 fscanf(fp, "%d ", &e);
312 i++;
313     }
314 fclose(fp);
315 return OK;
316 }
317
318
319 status ReverseList(LinkList *L)
320     // reverse the list
321 {
322     if(L)
323     {
324         LinkList prev=NULL;
325         LinkList cur=*L;
326         LinkList next=NULL;
327         while(cur)
328         {
329             next=cur->next;
330             cur->next=prev;
331             prev=cur;
332             cur=next;
333         }
334         *L=prev;
335         return OK;
336     }
337     else
338         return INFEASIBLE;
339 }
340
341
```

```
342 status RemoveNthFromEnd(LinkList &L, int n, ElemType& e)
343     // delete the nth node from the end of the list
344     {
345         if(!L)
346             return INFEASIBLE;
347         else
348         {
349             int k,j;
350             if(L->next==NULL)
351                 return ERROR;
352             k = ListLength(L);
353             j = k - n + 1;
354             ListDelete(&L, j, &e);
355             return OK;
356         }
357     }
358
359
360 status SortList (LinkList L)
361     // sort the list
362     {
363         LinkList p, q;
364         if(!L)
365             return INFEASIBLE;
366         else
367         {
368             if(L->next==NULL)
369                 return ERROR;
370             int n = ListLength(L);
371             int i, j, temp;
372             for(i = 0,p = L->next; i < n-1; i++,p = p->next)
```

```
373         for(j = i + 1,q = p -> next; j < n; j++,q = q -> next)
374             if(p -> data > q -> data)
375         {
376             temp = p -> data;
377             p -> data = q -> data;
378             q -> data = temp;
379         }
380
381     return OK;
382 }
383 }
```


6 附录 C 基于二叉链表二叉树实现的源程序

```

1  // BinaryTree.cpp
2  #include "BinaryTreeFunc.h"
3
4
5  int main()
6  {
7      int op=1,flag=0,i,lr;
8      KeyType key,key1,key2;
9
10     BiTree T=NULL;
11     BiTNode* t;
12     char FileName[30],TreeName[30];
13     T=NULL;
14
15     while(op)
16     {
17         system("cls");
18
19         printf("\n\n");
20         printf("      Menu for Linear Table On Sequence Structure \n");
21         printf("      1. CreateBiTree      2. DestroyBiTree\n");
22         printf("      3. ClearBiTree      4. BiTreeEmpty\n");
23         printf("      5. BiTreeDepth      6. LocateNode\n");
24         printf("      7. Assign            8. GetSibling\n");
25         printf("      9. InsertNode        10.DeleteNode\n");
26         printf("      11.PreOrderTraverse  12.InOrderTraverse\n");
27         printf("      13.PostOrderTraverse 14.LevelOrderTraverse\n");
28         printf("      15.MaxPathSum        16.LowestCommonAncestor\n");
29         printf("      17. InvertTree       18.SaveBiTree\n");

```

```
30 printf ( "          19.LoadBiTree          0. Exit \n");
31 printf ( "Choose your operation [0~19]:\n");
32
33
34 scanf( "%d",&op);
35
36
37 switch(op)
38 {
39 case 1: //CreateBiTree
40 if(T)
41 printf ( "The tree has been existed !\n");
42 else
43 {
44 printf ( "Please input the definition of the tree with empty node:\n");
45 i=0;
46 do
47 {
48 scanf( "%d%s",&definition[i].key, definition [ i ]. others ); //输入结点的值
49 } while ( definition [ i++].key!=-1);
50 flag=CreateBiTree(T, definition ); //创建二叉树
51 if( flag==1)
52 printf ( "The tree has been created successfully !\n");
53 else if( flag==0)
54 printf ( "The tree has been created failed !\n");
55 }
56 getchar ();
57 getchar ();
58 break;
59
60
```

```
61 case 2: //DestroyBiTree
62 flag=DestroyBiTree(&T);//销毁二叉树
63 if (flag==1)
64 printf ("The tree has been destroyed successfully !\n");
65 else
66 printf ("The tree has been destroyed failed !\n");
67 getchar ();
68 getchar ();
69 break;
70
71
72 case 3: //ClearBiTree
73 if (!T)
74 printf ("The tree has not been created !\n");
75 else
76 {
77 flag=ClearBiTree(T); //清空二叉树
78 if (flag==OK)
79 printf ("The tree has been cleared successfully !\n");
80 else
81 printf ("The tree has been cleared failed !\n");
82 }
83 getchar ();
84 getchar ();
85 break;
86
87
88 case 4: //BiTreeEmpty
89 flag=BiTreeEmpty(T);
90 if (flag==1)
91 printf ("The tree is empty!\n");
```

```
92  else if( flag==0)
93  printf ( "The tree is not empty!\n");
94  else
95  printf ( "The tree has not been created!\n");
96
97  getchar ();
98  printf ( "Press Enter to continue ... " );
99  getchar ();
100 break;
101
102
103 case 5: //BiTreeDepth
104 flag=BiTreeDepth(T); //求二叉树的深度
105 if( flag !=-1)
106 printf ( "The depth of the tree is %d.\n",flag );
107 else
108 printf ( "The tree has not been created!\n");
109 getchar ();
110 getchar ();
111 break;
112
113
114 case 6: //LocateNode
115 if (!T)
116 printf ( "The tree has not been created!\n");
117 else
118 {
119 printf ( "Please input the key of the node:" );
120 scanf( "%d",&key);
121 t=LocateNode(T,key);
122 if(t)
```

```
123 printf ( "The node is (%d,%s).\n", t->data.key, t->data.others );
124 else
125 printf ( "The node is not existed !\n");
126 }
127 getchar ();
128 getchar ();
129 break;
130
131
132 case 7: //Assign
133 if (!T)
134 printf ( "The tree has not been created !\n");
135 else
136 {
137 printf ( "Please input as follows :\n");
138 printf ( "[key] [new key] [new data]\n");
139 scanf( "%d%d%s", &key, &val.key, val.others);
140 flag=Assign(T, key, val );
141 if( flag==1)
142 printf ( "The node has been assigned successfully !\n");
143 else
144 printf ( "The node has been assigned failed !\n");
145 }
146 getchar ();
147 getchar ();
148 break;
149
150
151 case 8: //GetSibling
152 if (!T)
153 printf ( "The tree has not been created !\n");
```

```
154 else
155 {
156 printf ( "Please input the key of the node:" );
157 scanf( "%d", &key);
158 t=GetSibling(T, key);
159 if (t)
160 printf ( "The sibling of the node is (%d,%s).\n", t->data.key, t->data.others );
161 else
162 printf ( "The node is not existed or the node has no sibling !\n" );
163 }
164 getchar ();
165 getchar ();
166 break;
167
168
169 case 9: //InsertNode
170 if (!T)
171 printf ( "The tree has not been created !\n" );
172 else
173 {
174 printf ( "Please input as below:\n" );
175 printf ( "[key] [0: insert left child / 1: insert right child] [new key] [new data]\n" );
176 scanf( "%d%d%d%s", &key, &lr, &val.key, val.others);
177 flag=InsertNode(T, key, lr, val );
178 if ( flag==1)
179 printf ( "The node has been inserted successfully !\n" );
180 else
181 printf ( "The node has been inserted failed !\n" );
182 }
183 getchar ();
184 getchar ();
```

```
185 break;
186
187
188 case 10: //DeleteNode
189 if (!T)
190 printf ("The tree has not been created !\n");
191 else
192 {
193 printf ("Please input the key of the node:");
194 scanf ("%d",&key);
195 flag=DeleteNode(T,key);
196 if ( flag==1)
197 printf ("The node has been deleted successfully !\n");
198 else
199 printf ("The node has been deleted failed !\n");
200 }
201 getchar ();
202 getchar ();
203 break;
204
205
206 case 11: //PreOrderTraverse
207 if (!T)
208 printf ("The tree has not been created !\n");
209 else
210 {
211 printf ("PreOrderTraverse:\n-----\n\n");
212 PreOrderTraverse(T, Visit );
213 printf ("\n\n-----\nEND\n");
214 }
215 getchar ();
```

```
216  getchar ();
217  break;
218
219
220  case 12: //InOrderTraverse
221  if (!T)
222  printf ( "The tree has not been created !\n");
223  else
224  {
225  printf ( "InOrderTraverse:\n-----\n\n");
226  InOrderTraverse(T, Visit );
227  printf ( "\n\n-----\nEND\n");
228  }
229  getchar ();
230  getchar ();
231  break;
232
233
234  case 13: //PostOrderTraverse
235  if (!T)
236  printf ( "The tree has not been created !\n");
237  else
238  {
239  printf ( "PostOrderTraverse:\n-----\n\n");
240  PostOrderTraverse(T, Visit );
241  printf ( "\n\n-----\nEND\n");
242  }
243  getchar ();
244  getchar ();
245  break;
246
```



```
247
248 case 14: //LevelOrderTraverse
249 if (!T)
250     printf ("The tree has not been created !\n");
251 else
252 {
253     printf ("LevelOrderTraverse:\n-----\n\n");
254     LevelOrderTraverse(T, Visit );
255     printf ("\n\n-----\nEND\n");
256 }
257 getchar ();
258 getchar ();
259 break;
260
261
262 case 15: //MaxPathSum
263 if (!T)
264     printf ("The tree has not been created !\n");
265 else
266     printf ("The max path sum is %d.\n",MaxPathSum(T));
267
268 getchar ();
269 getchar ();
270 break;
271
272
273 case 16: //LowestCommonAncestor
274 if (!T)
275     printf ("The tree has not been created !\n");
276 else
277 {
```

```
278 printf ( "Please input the key of the two nodes:\n");
279 scanf( "%d%d",&key1,&key2);
280 t=LowestCommonAncestor(T,key1,key2);
281 if(t)
282     printf ( "(%d,%s).\n",t->data.key,t->data.others );
283 else
284     printf ( "The two nodes are not existed or the tree is empty!\n");
285 }
286 getchar ();
287 getchar ();
288 break;
289
290
291 case 17: // InvertTree
292     if (!T)
293         printf ( "The tree has not been created !\n");
294     else
295     {
296         flag=InvertTree (T);
297         if ( flag==1)
298             printf ( "The tree has been inverted successfully !\n");
299         else
300             printf ( "The tree has been inverted failed !\n");
301     }
302     getchar ();
303     getchar ();
304     break;
305
306
307 case 18: // SaveBiTree
308     if (!T)
```

```
309 printf ("The tree has not been created !\n");
310 else
311 {
312     printf ("Please input the file name:");
313     scanf ("%s",FileName);
314     flag=SaveBiTree(T,FileName);
315     if ( flag==1)
316         printf ("The tree has been saved successfully !\n");
317     else
318         printf ("The tree has been saved failed !\n");
319 }
320 getchar ();
321 getchar ();
322 break;
323
324
325 case 19: //LoadBiTree
326     if(T)
327         printf ("The tree has been created !\n");
328     else
329     {
330         printf ("Please input the file name:");
331         scanf ("%s",FileName);
332         flag=LoadBiTree(T,FileName);
333         if ( flag==1)
334             printf ("The tree has been loaded successfully !\n");
335         else
336             printf ("The tree has been loaded failed !\n");
337     }
338     getchar ();
339     getchar ();
```

```
340 break;
341
342
343 case 0:
344 break;
345
346
347 }
348 }
349 getchar ();
350 getchar ();
351 }
```

```
1 //BinaryTreeFunc.h
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5
6
7 #define TRUE 1
8 #define FALSE 0
9 #define OK 1
10 #define ERROR 0
11 #define INFEASIBLE -1
12 #define OVERFLOW -2
13
14
15 typedef int status ;
16 typedef int KeyType;
17 typedef struct {
18     KeyType key;
19     char others [20];
20 } TElemType;
21
22 typedef struct BiTNode{
23     TElemType data;
24     struct BiTNode *lchild,*rchild;
25 } BiTNode, *BiTree;
26
27
28 status CreateBiTree(BiTree &T,TElemType definition []);
29 status DestroyBiTree(BiTree *T);
30 status ClearBiTree(BiTree &T);
31 status BiTreeEmpty(BiTree &T);
```

```
32 int BiTreeDepth(BiTree T);
33 BiTNode* LocateNode(BiTree T,KeyType e);
34 status Assign(BiTree &T,KeyType e,TElemType value);
35 BiTNode* GetSibling(BiTree T,KeyType e);
36 status InsertNode(BiTree &T,KeyType e,int LR,TElemType c);
37 status DeleteNode(BiTree &T,KeyType e);
38 status PreOrderTraverse(BiTree T,void (* visit )(BiTree ));
39 status InOrderTraverse(BiTree T,void (* visit )(BiTree ));
40 status PostOrderTraverse(BiTree T,void (* visit )(BiTree ));
41 status LevelOrderTraverse(BiTree T,void (* visit )(BiTree ));
42 status MaxPathSum(BiTree T);
43 BiTNode* LowestCommonAncestor(BiTree T,KeyType e1,KeyType e2);
44 status InvertTree (BiTree &T);
45 status SaveBiTree(BiTree T, char FileName[]);
46 status LoadBiTree(BiTree &T, char FileName[]);
47
48
49
50 int max(int x, int y)
51 {
52     return x>y?x:y;
53 }
54
55
56 void Visit (BiTree T)
57 {
58     printf ( " ");
59     printf ( " (%d,%s)",T->data.key,T->data.others );
60 }
61
62
```

```
63 TElemType val, definition [100];
64
65
66 status CreateBiTree(BiTree &T,TElemType definition[])
67 /*根据带空枝的二叉树先根遍历序列definition构造一棵二叉树,
68 将根节点指针赋值给T并返回OK,
69 如果有相同的关键字, 返回ERROR。此题允许通过增加其它函数辅助实现本关任务*/
70 {
71     static TElemType *p=definition;
72     if (p->key==-1)//空树
73     {
74         T=NULL;
75         return OK;
76     }
77
78     if (!p->key)//空结点
79     {
80         T=NULL;
81         p++;
82         return OK;
83     }
84     else
85     {
86         TElemType *q=definition;
87         while(q<p)//判断是否有相同的关键字
88         {
89             if (q->key==p->key) //有相同的关键字
90                 return ERROR;
91             else
92                 q++;
93         }
```

```
94 T=(BiTree)malloc( sizeof( BiTNode)); //生成根结点
95 T->data=*p; //赋值
96 p++;
97 if( CreateBiTree(T->lchild, definition )&&CreateBiTree(T->rchild, definition ))
98 //递归构造左右子树
99 return OK;
100 }
101 }
102
103
104 status ClearBiTree(BiTree &T)
105 //将二叉树设置成空，并删除所有结点，释放结点空间
106 {
107 if (!T) //空树
108 return OK;
109 else
110 {
111 ClearBiTree(T->lchild); //递归删除左子树
112 ClearBiTree(T->rchild); //递归删除右子树
113 free (T); //释放根结点
114 T=NULL;
115 return OK;
116 }
117 }
118
119
120 status DestroyBiTree(BiTree *T)
121 {
122 if (*T) //非空树
123 {
124 if ((*T)->lchild) //递归删除左子树
```



```
125 DestroyBiTree(&((*T)->lchild));
126 if ((*T)->rchild) //递归删除右子树
127 DestroyBiTree(&((*T)->rchild));
128 free(*T); //释放根结点
129 (*T) = NULL;
130 }
131 return OK;
132 }
133
134
135 status BiTreeEmpty(BiTree &T)
136 {
137 if (!T) //空树
138 return -1;
139 if (T==NULL) //空树
140 return 1;
141 else //非空树
142 return 0;
143 }
144
145
146 int BiTreeDepth(BiTree T)
147 //求二叉树T的深度
148 {
149 if (!T) //空树
150 return 0;
151 int d=0,dr=0,dl=0;
152 dl=BiTreeDepth(T->lchild); //递归求左子树深度
153 dr=BiTreeDepth(T->rchild); //递归求右子树深度
154 d=dl>dr?dl:dr; //取左右子树深度的最大值
155 return d+1;
```

```
156 }
157
158
159 BiTNode* LocateNode(BiTree T,KeyType e)
160 //查找结点
161 {
162 if(!T) //空树
163 return NULL;
164 if(T->data.key==e) //找到结点
165 return T;
166 BiTNode *p=NULL;
167 p=LocateNode(T->lchild,e); //递归查找左子树
168 if(p)
169 return p;
170 p=LocateNode(T->rchild,e); //递归查找右子树
171 if(p) //找到结点
172 return p;
173 return NULL;
174 }
175
176
177 status Assign(BiTree &T,KeyType e,TElemType value)
178 //实现结点赋值。此题允许通过增加其它函数辅助实现本关任务
179 {
180 if(LocateNode(T,value.key)&&(value.key!=e)) //有相同的关键字
181 return ERROR;
182 BiTree p=LocateNode(T,e); //查找结点
183 if(p)
184 {
185 p->data=value; //赋值
186 return OK;
```

```
187 }
188 return ERROR;
189 }
190
191
192 BiTNode* GetSibling(BiTree T,KeyType e)
193 //实现获得兄弟结点
194 {
195
196 if (!(T->lchild&&T->rchild)) //空树或者叶子结点
197 return NULL;
198 if ((T->lchild)->data.key==e) //左子树为e
199 return T->rchild;
200 if ((T->rchild)->data.key==e) //右子树为e
201 return T->lchild;
202
203 BiTree p=NULL;
204 p=GetSibling(T->lchild,e); //递归查找左子树
205 if(p)
206 return p;
207 p=GetSibling(T->rchild,e); //递归查找右子树
208 if(p)
209 return p;
210 return NULL;
211 }
212
213
214 status InsertNode(BiTree &T,KeyType e,int LR,TElemType c)
215 //插入结点。此题允许通过增加其它函数辅助实现本关任务
216 {
217 BiTree root=LocateNode(T,e); //查找结点
```

```
218 if(! root) //没有找到结点
219 return ERROR;
220 switch(LR)//插入结点
221 {
222 case 0: //左子树
223 {
224 BiTNode *temp=root->lchild;//原来的左子树
225 root->lchild=(BiTNode*)malloc(sizeof(BiTNode));//生成结点
226 if(! root->lchild) //生成结点失败
227 return ERROR;
228 root->lchild->data=c;//赋值
229 root->lchild->lchild=NULL;//左子树为空
230 root->lchild->rchild=temp;//右子树为原来的左子树
231 break;
232 }
233
234
235 case 1: //右子树
236 {
237 BiTNode *temp=root->rchild;//原来的右子树
238 root->rchild=(BiTNode*)malloc(sizeof(BiTNode));//生成结点
239 if(! root->rchild) //生成结点失败
240 return ERROR;
241 root->rchild->data=c;//赋值
242 root->rchild->lchild=NULL;//左子树为空
243 root->rchild->rchild=temp;//右子树为原来的右子树
244 break;
245 }
246
247
248 case -1: //根结点
```

```
249 {
250 BiTNode *temp=root;//原来的根结点
251 root=(BiTNode*)malloc(sizeof(BiTNode));//生成结点
252 if(! root) //生成结点失败
253 return ERROR;
254 root->data=c;//赋值
255 root->lchild=NULL;//左子树为空
256 root->rchild=temp;//右子树为原来的根结点
257 break;
258 }
259 default :
260 return ERROR;
261 }
262 return OK;
263 }
264
265
266 status DeleteNode(BiTree &T,KeyType e)
267 //删除结点
268 {
269 if(! T)
270 return 0;
271 if(T->data.key==e)//找到结点
272 {
273 BiTree p=T;
274 if(! T->lchild&&!T->rchild)//叶子结点
275 T=NULL;
276 else if(! T->rchild) //只有左子树
277 T=T->lchild;
278 else if(! T->lchild) //只有右子树
279 T=T->rchild;
```

```
280 else
281 {
282 T=T->lchild; //左子树
283 BiTree t=T;
284 while(t->rchild) //找到左子树的最右结点
285 t=t->rchild;
286 t->rchild=p->rchild; //将原来的右子树连接到左子树的最右结点
287 }
288 free(p); //释放结点
289 return OK;
290 }
291 return DeleteNode(T->lchild,e) || DeleteNode(T->rchild,e); //递归查找
292 }
293
294
295 status PreOrderTraverse(BiTree T,void (* visit )(BiTree))
296 //先序遍历二叉树T
297 {
298 BiTree a[100];
299 int i=-1;
300 a[0]=T;
301 BiTree p=T;
302 while(p || i!=-1) //栈不为空
303 {
304 while(p) //遍历左子树
305 {
306 visit(p); //访问结点
307 a[++i]=p; //入栈
308 p=p->lchild; //左子树
309 }
310 if(i!=-1) //栈不为空
```

```
311 {
312 p=a[i--]; //出栈
313 p=p->rchild; //右子树
314 }
315 }
316 }
317
318
319 status InOrderTraverse(BiTree T,void (* visit )(BiTree))
320 //中序遍历二叉树T
321 {
322
323 if (!T)
324 return ERROR;
325 InOrderTraverse(T->lchild, visit ); //递归遍历左子树
326 visit (T); //访问结点
327 InOrderTraverse(T->rchild, visit ); //递归遍历右子树
328 return OK;
329 }
330
331
332 status PostOrderTraverse(BiTree T,void (* visit )(BiTree))
333 //后序遍历二叉树T
334 {
335 if (!T)
336 return ERROR;
337 PostOrderTraverse(T->lchild, visit ); //递归遍历左子树
338 PostOrderTraverse(T->rchild, visit ); //递归遍历右子树
339 visit (T);
340 return OK;
341 }
```

```
342
343
344 status LevelOrderTraverse(BiTree T,void (* visit )(BiTree))
345 //按层遍历二叉树T
346 {
347 BiTree a[100];
348 int come=0,go=0;
349 a[come++]=T;//根结点入队
350 while(come>go)//队不为空
351 {
352
353 if(a[go]) //结点不为空
354 {
355 visit (a[go]); //访问结点
356 a[come++]=a[go]->lchild;//左子树入队
357 a[come++]=a[go]->rchild;//右子树入队
358 }
359 go++;
360 }
361 }
362
363
364 void fpr(BiTree T,FILE* fout)
365 {
366 if(!T)
367 {
368 fprintf ( fout , "0 null "); //空结点
369 return ;
370 }
371 fprintf ( fout , "%d %s ",T->data.key,T->data.others); //结点数据
372 fpr (T->lchild, fout ); //左子树
```



```
373 fpr(T->rchild, fout); //右子树
374 return ;
375 }
376
377
378 void fread(BiTree& T, FILE* fin)
379 {
380 int key;
381 char s[20];
382 fscanf(fin, "%d %s", &key, s); //读入结点数据
383 if(key==0) //空结点
384 {
385 T=NULL;
386 return ;
387 }
388 if(key==1) //根结点
389 return ;
390 T=(BiTree)malloc(sizeof(BiTNode)); //生成结点
391 T->data.key=key; //赋值
392 strcpy(T->data.others, s); //赋值
393 fread(T->lchild, fin); //左子树
394 fread(T->rchild, fin); //右子树
395 return ;
396 }
397
398
399 status SaveBiTree(BiTree T, char FileName[])
400 //将二叉树的结点数据写入到文件FileName中
401 {
402 FILE* fout=fopen(FileName, "w"); //打开文件
403 fpr(T, fout); //写入数据
```

```
404  fprintf ( fout , "-1 null"); //根结点
405  fclose ( fout ); //关闭文件
406  return OK;
407  }
408
409
410  status LoadBiTree(BiTree &T, char FileName[])
411  //读入文件FileName的结点数据，创建二叉树
412  {
413  FILE* fin=fopen(FileName,"r");//打开文件
414  if ( fin==NULL)
415  return 0;
416  fread (T, fin ); //读入数据
417  fclose ( fin ); //关闭文件
418  return OK;
419  }
420
421
422  status MaxPathSum(BiTree T)
423  //最大路径和
424  {
425  if (!T)
426  return 0;
427  int l=MaxPathSum(T->lchild); //左子树
428  int r=MaxPathSum(T->rchild); //右子树
429  return max(l,r)+T->data.key; //返回最大值
430  }
431
432
433  BiTNode* LowestCommonAncestor(BiTree T,KeyType e1,KeyType e2)
434  //最近公共祖先
```

```
435 {
436 if(LocateNode(T->lchild,e1)) // 左子树
437 {
438 if(LocateNode(T->lchild,e2)) // 左子树
439 return LowestCommonAncestor(T->lchild,e1,e2); // 递归
440 return T;
441 }
442 else
443 {
444 if(LocateNode(T->rchild,e2))
445 return LowestCommonAncestor(T->rchild,e1,e2);
446 return T;
447 }
448 }
449
450
451 status InvertTree (BiTree &T)
452 // 翻转二叉树
453 {
454 if (!T)
455 return OK;
456 BiTNode*t;
457 t=T->lchild; // 交换左右子树
458 T->lchild=T->rchild; // 交换左右子树
459 T->rchild=t; // 交换左右子树
460 InvertTree (T->lchild); // 左子树
461 InvertTree (T->rchild); // 右子树
462 return 1;
463 }
```

7 附录 D 基于邻接表图实现的源程序

```

1  //Graph.cpp
2  #include "GraphFunc.h"
3
4  int main()
5  {
6  //定义变量
7  int op=1,flag,x;
8  ALGraph G;
9  G.arcnum=0,G.vexnum=0;
10 VertexType V[30];
11 VertexType value;
12 KeyType VR[100][2];
13 int i=0,j,e,v,w,k;
14 char filename[20];
15 char name[20];
16
17 while(op)
18 {
19 system("cls");
20 printf("\n");
21
22 printf("Menu for Linear Table On Sequence Structure\n");
23 printf("1. CreateGraph          2. DestroyGraph\n");
24 printf("3. LocateVex              4. PutVex\n");
25 printf("5. FirstAdjVex            6. NextAdjVex\n");
26 printf("7. InsertVex              8. DeleteVex\n");
27 printf("9. InsertArc              10.DeleteArc\n");
28 printf("11.DFS Traverse           12.BFS Traverse\n");
29 printf("13.VerticesSetLessThanK   14.ShortestPathLength\n");

```

```
30 printf ( "15.ConnectedComponentsNums   16.SaveGraph\n");
31 printf ( "17.LoadGraph                   0. Exit\n\n");
32 printf ( "Choose your operation [0~17]:\n");
33
34 scanf( "%d",&op);
35
36 switch(op)
37 {
38 case 1: //InitGraph
39 i=0;
40 if(pd) //判断是否已经创建图
41 {
42 printf ( "The graph has been created.\n");
43 printf ( "Please destroy it first !\n");
44 }
45 else
46 {
47 printf ( "Please enter the vertex sequence,\n");
48 printf ( "the following pair sequence of the undirected graph,\n");
49 printf ( "with -1 as the ending mark!\n\n");
50
51 do{
52 scanf( "%d %s",&V[i].key,V[i].others );
53 }while(V[i++].key!=-1); //输入顶点序列
54
55 i=0;
56 do{
57 scanf( "%d%d",&VR[i][0],&VR[i][1]);
58 }while(VR[i++][0]!=-1); //输入边序列
59
60 if (CreateCraph(G,V,VR)==ERROR)
```

```
61 printf ( "Invalid input !\n"); //创建图
62
63 else
64 {
65 if (G.arcnum!=i-1)
66 {
67 printf ( "The number of arcs is wrong!\n");
68 return 0;
69 }
70 printf ( "\n");
71 for (j=0;j<G.vexnum;j++)//输出邻接表
72 {
73 ArcNode *p=G.vertices[j]. firstarc ;
74 printf ( "(%d , %s)",G.vertices [j ]. data .key,G.vertices [j ]. data . others );
75 while (p)
76 {
77 printf ( " %d",p->adjvex);
78 p=p->nextarc;
79 }
80 printf ( "\n");
81 }
82 }
83 }
84 getchar ();
85 getchar ();
86 break;
87
88
89 case 2: //DestroyGraph
90 if (G.vexnum==0)//判断是否已经创建图
91 printf ( "Fail to destroy !\n");
```

```
92  else
93  {
94  if (DestroyGraph(G)==OK)
95  printf ( "Success to destroy !\n");
96  else
97  printf ( "Fail to destroy !\n");
98  }
99  getchar ();
100 getchar ();
101 break;
102
103
104 case 3: //LocateVex
105 if (G.vexnum==0)
106 printf ( "The graph does not exist !\n");
107 else
108 {
109 printf ( "Please enter the key of the vertex !\n");
110 scanf( "%d",&e);
111 if (LocateVex(G,e)==-1)
112 printf ( "The vertex does not exist !\n");
113 else
114 {
115 x=LocateVex(G,e); //定位顶点
116 printf ( "The vertex is in the No.%d position !\n",x);
117 printf ( "Its value is (%d , %s)\n",G.vertices [x].data.key,G.vertices [x].data.others );
118 }
119 }
120 getchar ();
121 getchar ();
122 break;
```

```
123
124
125 case 4: //PutVex
126 if(G.vexnum==0)
127     printf("The graph does not exist !\n");
128 else
129 {
130     printf("Please input as follows :\n");
131     printf("[key] [newkey] [newothers]\n");
132     scanf("%d%d%s",&j,&value.key,value.others);//输入新的顶点值
133
134     if(LocateVex(G,j)==-1 || LocateVex(G,j)==-2)
135         printf("The vertex does not exist !\n");
136     else if(PutVex(G,j,value)==OK)
137         printf("Success to modify !\n");
138     else
139         printf("Fail to modify !\n");//修改顶点值
140 }
141 getchar();
142 getchar();
143 break;
144
145
146 case 5: //FirstAdjVex
147 if(G.vexnum==0)
148     printf("The graph does not exist !\n");
149 else
150 {
151     printf("Please enter the key of the vertex !\n");
152     scanf("%d",&e);//输入顶点值
153
```



```

154 if (FirstAdjVex(G,e)==-1)
155     printf ( "Fail to find !\n");
156 else
157 {
158     x=FirstAdjVex(G,e); //查找第一个邻接点
159     printf ( "The FirstAdjVex is in the No.%d position !\n",x);
160     printf ( "Its value is (%d , %s)\n",G.vertices [x].data .key,G.vertices [x].data .others );
161     //输出第一个邻接点的值
162 }
163 }
164 getchar ();
165 getchar ();
166 break;
167
168
169 case 6: //NextAdjVex
170 if (G.vexnum==0)
171     printf ( "The graph does not exist !\n");
172 else
173 {
174     printf ( "The format is as follows :\n");
175     printf ( "[MainVertex Key] [FirstAdjVex Key]\n");
176     scanf( "%d%d",&v,&w); //输入顶点值
177
178     if (NextAdjVex(G,v,w)==-1)
179         printf ( "Fail to find !\n");
180     else
181     {
182         x=NextAdjVex(G,v,w); //查找下一个邻接点
183         printf ( "According to [%d] vertex, the [%d] vertex's nextadjvex is in No.%d position\n",w,v,x);
184         printf ( "Its value is (%d , %s)\n",G.vertices [x].data .key,G.vertices [x].data .others );

```

```
185 }
186 }
187 getchar ();
188 getchar ();
189 break;
190
191
192 case 7: // InsertVex
193     printf ("Please input as follows :\n");
194     printf ("[key] [others] \n");
195     scanf ("%d%s",&value.key,value.others); //输入新的顶点值
196
197     if (InsertVex(&G,value)==OK)
198         printf ("Success to insert !\n");
199     else
200         printf ("Fail to insert !\n");
201     getchar ();
202     getchar ();
203     break;
204
205
206 case 8: // DeleteVex
207     if (G.vexnum==0)
208         printf ("The graph does not exist !\n");
209     else
210     {
211         printf ("Please enter the key of the vertex !\n");
212         scanf ("%d",&v); //输入顶点值
213
214         if (DeleteVex(G,v)==OK)
215             printf ("Success to delete !\n");
```

```
216 else
217     printf ("Fail to delete !\n");
218 }
219 getchar ();
220 getchar ();
221 break;
222
223
224 case 9: // InsertArc
225     if(G.vexnum==0)
226         printf ("The graph does not exist !\n");
227     else
228     {
229         printf ("Please input the two vertexs of the arc !\n");
230         scanf ("%d%d",&v,&w); //输入弧尾和弧头
231
232         if( InsertArc (G,v,w)==OK)
233             printf ("Insert the arc successfully !\n");
234         else
235             printf ("Fail to insert !\n");
236     }
237     getchar ();
238     getchar ();
239     break;
240
241
242 case 10: //DeleteArc
243     if(G.vexnum==0)
244         printf ("The graph does not exist !\n");
245     else
246     {
```

```
247 printf ( "Please input the two vertexs of the arc !\n" );
248 scanf ( "%d%d", &v, &w);
249 if (DeleteArc(G, v, w) == OK)
250 printf ( "Success to delete !\n" );
251 else
252 printf ( "Fail to delete !\n" );
253 }
254 getchar ();
255 getchar ();
256 break;
257
258
259 case 11: //DFSTraverse
260 if (G.vexnum == 0)
261 printf ( "The graph does not exist !\n" );
262 else
263 {
264 printf ( "DFS Traverse:\n-----\n" );
265 DFSTraverse(G, visit );
266 printf ( "\n-----\nEND\n" );
267 }
268 getchar ();
269 getchar ();
270 break;
271
272
273 case 12: //BFSTraverse
274 if (G.vexnum == 0)
275 printf ( "The graph does not exist !\n" );
276 else
277 {
```

```
278 printf ( "BFS Traverse:\n-----\n");
279 BFSTraverse(G, visit );
280 printf ( "\n-----\nEND\n");
281 }
282 getchar ();
283 getchar ();
284 break;
285
286
287 case 13: // VerticesSetLessThanK
288 printf ( "Please input the key of the vertex and the distance !\n");
289 scanf( "%d%d", &v, &k);
290 flag=VerticesSetLessThank(G,v,k);
291 if (! flag )
292 printf ( "Fail to find !\n");
293 else
294 printf ( "The vertices set less than %d is:%d\n", k, flag ); // 错误处理
295 getchar ();
296 getchar ();
297 break;
298
299
300 case 14: // ShortestPathLength
301 printf ( "Please input the two vertexs of the arc !\n");
302 scanf( "%d%d", &v, &w);
303 k=ShortestPathLength(G,v,w);
304 if (k==-1)
305 printf ( "The two vertexs are not connected !\n");
306 else
307 printf ( "The shortest path length is %d\n", k);
308 getchar ();
```

```
309 getchar ();
310 break;
311
312
313 case 15: //ConnectedComponentsNums
314     printf ( "The number of connected components is %d\n",ConnectedComponentsNums(G));
315
316     getchar ();
317     getchar ();
318     break;
319
320
321 case 16: //SaveGraph
322     if (G.vexnum==0)
323         printf ( "The graph does not exist !\n");
324     else
325     {
326         printf ( "Please input the file name!\n");
327         scanf( "%s",filename);
328         if (SaveGraph(G,filename)==OK)
329             printf ( "Save the graph successfully !\n");
330         else
331             printf ( "Fail to save !\n");
332     }
333     getchar ();
334     getchar ();
335     break;
336
337
338 case 17: //LoadGraph
339     printf ( "Please input the file name!\n");
```

```
340 scanf( "%s",filename);
341 if(LoadGraph(G,filename)==OK)
342     printf ( "Load the graph successfully !\n");
343 else
344     printf ( "Fail to load !\n");
345 getchar ();
346 getchar ();
347 break;
348
349
350
351 case 0: // Exit
352     exit (0);
353 break;
354 }
355 }
356 }
```

```
1 //GraphFunc.h
2 #include<stdio.h>
3 #include<stdlib.h>
4 #include<string.h>
5
6
7 #define TRUE 1
8 #define FALSE 0
9 #define OK 1
10 #define ERROR 0
11 #define INFEASIBLE -1
12 #define OVERFLOW -2
13 #define MAX_VERTEX_NUM 20
14
15
16 typedef int status ;
17 typedef int KeyType;
18 typedef enum {DG,DN,UDG,UDN} GraphKind;
19
20 typedef struct {
21     KeyType key;
22     char others [20];
23 } VertexType; //顶点类型定义
24
25 typedef struct ArcNode { //表结点类型定义
26     int adjvex; //顶点位置编号
27     struct ArcNode *nextarc; //下一个表结点指针
28 } ArcNode;
29
30 typedef struct VNode { //头结点及其数组类型定义
31     VertexType data; //顶点信息
```



```
32 ArcNode *firstarc ; //指向第一条弧
33 } VNode,AdjList[MAX_VERTEX_NUM];
34
35 typedef struct { //邻接表的类型定义
36 AdjList vertices ; //头结点数组
37 int vexnum,arcnum; //顶点数、弧数
38 GraphKind kind; //图的类型
39 } ALGraph;
40
41 struct ptr {
42 void *pused[100],*pfree[100];
43 int len_used, len_free ;
44 } pm;
45
46 typedef struct QNode{
47 VertexType data ;
48 struct QNode *next;
49 }QNode,*Queue;
50
51 typedef struct {
52 Queue front ;
53 Queue rear ;
54 }Linkqueue;
55
56 void free0(void *p)
57 {
58 pm.pfree[pm.len_free++]=p;
59 memset(p,0,sizeof(ArcNode));
60 free (p);
61 }
62
```

```
63 bool visited [MAX_VERTEX_NUM];
64 int pd;
65
66 void visit (VertexType v)
67 {
68     printf ( " (%d , %s)",v.key,v. others );
69 }
70
71
72 status issame(VertexType V[])//判断顶点重复
73 {
74     int i=0;
75     int mark[100000]={0};
76     while(V[i].key!=-1)
77     {
78         if(mark[V[i].key]>0)
79             return 1;
80
81         mark[V[i].key]++;//标记
82         i++;
83     }
84     return 0;
85 }
86
87
88 int Locate (ALGraph G,KeyType VR)
89 {
90     int i=0;
91     for (; i<G.vexnum;i++)
92         if(G.vertices [ i ]. data .key==VR)
93             return i;
```

```
94
95 return -1;
96 }
97
98
99 status CreateCraph(ALGraph &G,VertexType V[],KeyType VR[][2])
100 /*根据V和VR构造图T并返回OK, 如果V和VR不正确, 返回ERROR
101 如果有相同的关键字, 返回ERROR。此题允许通过增加其它函数辅助实现本关任务*/
102 {
103     int i=0,j=0;
104     while(V[i].key!=-1)
105     {
106         for (int k=0;k<i;k++)
107             if (V[i].key==V[k].key)
108                 return ERROR;
109
110         if(i>=MAX_VERTEX_NUM)
111             return ERROR;
112         G.vertices[i].data=V[i]; //赋值
113         G.vertices[i].firstarc =NULL;
114         i++;
115     }
116     if(i==0)
117         return ERROR;
118     G.vexnum=i;//顶点数
119
120     while(VR[j][0]!=-1)
121     {
122         int a=Locate(G,VR[j][0]), b=Locate(G,VR[j][1]);
123         if (!(a>=0&&b>=0))
124             return ERROR;
```

```
125 ArcNode *q=(ArcNode*)malloc(sizeof(ArcNode));
126 q->adjvex=b;
127 q->nextarc=G.vertices[a]. firstarc ;//头插法
128 G.vertices[a]. firstarc =q;
129 q=(ArcNode*)malloc(sizeof(ArcNode));
130 q->adjvex=a;
131 q->nextarc=G.vertices[b]. firstarc ;//头插法
132 G.vertices[b]. firstarc =q;
133 j++;
134 }
135 G.arcnum=j;
136 return OK;
137 }
138
139
140
141 status DestroyGraph(ALGraph &G)
142 /*销毁无向图G,删除G的全部顶点和边*/
143 {
144 ArcNode *p=NULL,*q=NULL;
145 for( int k=0;k<G.vexnum;k++)
146 {
147 if(G.vertices[k]. firstarc !=NULL)//释放边
148 {
149 p=G.vertices[k]. firstarc ;
150 while(p!=NULL)
151 {
152 q=p->nextarc;//释放
153 free(p);
154 p=q;
155 }
```

```
156 G.vertices[k].firstarc=NULL;
157 }
158 }
159 G.arcnum=0;
160 G.vexnum=0;
161 return OK;
162 }
163
164
165 int LocateVex(ALGraph G,KeyType u)
166 //根据u在图G中查找顶点，查找成功返回位序，否则返回-1;
167 {
168     int i=0;
169     for(i;i<G.vexnum;i++)
170         if(G.vertices[i].data.key==u) //找到
171             return i;
172
173     return -1;
174 }
175
176
177 status PutVex(ALGraph &G,KeyType u,VertexType value)
178 //根据u在图G中查找顶点，查找成功将该顶点值修改成value，返回OK;
179 //如果查找失败或关键字不唯一，返回ERROR
180 {
181     int i=0;
182     for(; i<G.vexnum;i++)
183         if(G.vertices[i].data.key==u)
184             {
185                 for(int k=0;k<G.vexnum;k++)
186                     // if(G.vertices[k].data.key==value.key) //找到
```

```
187 //    return ERROR;
188
189 G.vertices [ i ]. data=value;
190 return OK;
191 }
192
193 return ERROR;
194 }
195
196
197 int FirstAdjVex(ALGraph G,KeyType u)
198 //根据u在图G中查找顶点，查找成功返回顶点u的第一邻接顶点位序，否则返回-1;
199 {
200 int i=0;
201 for (; i<G.vexnum;i++)
202 if(G.vertices [ i ]. data.key==u&&G.vertices[i]. firstarc )
203 return G.vertices [ i ]. firstarc ->adjvex;//找到
204 return -1;
205 }
206
207
208 int NextAdjVex(ALGraph G,KeyType v,KeyType w)
209 //v对应G的一个顶点,w对应v的邻接顶点；操作结果是返回v的（相对于w）
210 //下一个邻接顶点的位序；如果w是最后一个邻接顶点，
211 //或v、w对应顶点不存在，则返回-1。
212 {
213 int i=LocateVex(G,v);
214 int j=LocateVex(G,w);
215 while(i!=-1&&j!=-1)
216 {
217 if(G.vertices [ i ]. firstarc ==NULL)
```

```
218 return -1;
219 else
220 {
221 ArcNode *p=G.vertices[i]. firstarc ;//找到
222 while(p->adjvex!=j&& p)
223 p=p->nextarc;
224
225 if(p->nextarc)
226 return p->nextarc->adjvex;
227 else
228 return -1;
229 }
230 }
231 return -1;
232 }
233
234
235 status InsertVex (ALGraph *G,VertexType v)
236 //在图G中插入顶点v, 成功返回OK,否则返回ERROR
237 {
238 if(LocateVex(*G,v.key)>=0)
239 return ERROR;
240 if((*G).vexnum==MAX_VERTEX_NUM)
241 return ERROR;
242 (*G). vertices [(*G).vexnum].data=v;//赋值
243 (*G). vertices [(*G).vexnum]. firstarc =NULL;
244 (*G).vexnum++;
245 return OK;
246 }
247
248
```

```
249 status DeleteVex(ALGraph &G,KeyType v)
250 //在图G中删除关键字v对应的顶点以及相关的弧,成功返回OK,否则返回ERROR
251 {
252     int i=LocateVex(G,v),j=-1;
253     if(i==-1)
254         return ERROR;
255     if(G.vexnum==1 || G.vexnum==0)
256         return ERROR;
257     ArcNode *p=G.vertices[i]. firstarc ;
258     ArcNode *q=NULL;
259     ArcNode *temp=NULL;
260     while(p)
261     {
262         j=p->adjvex;
263         q=G.vertices[j]. firstarc ;
264         if(q->adjvex==i)
265         {
266             temp=q;
267             G.vertices[j]. firstarc =q->nextarc;
268             free(temp);
269         }
270         else
271         {
272             while(q->nextarc->adjvex!=i)
273                 q=q->nextarc;
274
275             temp=q->nextarc;
276             q->nextarc=temp->nextarc;//删除
277             free(temp);
278         }
279         temp=p->nextarc;
```



```
280 free(p);
281 p=temp;
282 G.arcnum--;
283 }
284 for(int k=i;k<G.vexnum;k++)
285 G.vertices[k]=G.vertices[k+1]; //删除
286
287 G.vexnum--;
288 for(int k=0;k<G.vexnum;k++)
289 {
290 p=G.vertices[k].firstarc;
291 while(p!=NULL)
292 {
293 if(p->adjvex>i)
294 p->adjvex--;
295 p=p->nextarc;
296 }
297 }
298 return OK;
299 }
300
301
302 status Locatearc(ALGraph G,KeyType v,KeyType w)
303 {
304 ArcNode *p=G.vertices[v].firstarc;
305 while(p!=NULL)
306 {
307 if(p->adjvex==w)//查找<v,w>
308 return OK;
309 p=p->nextarc;
310 }
```

```
311 return ERROR;
312 }
313
314
315 int LocateArc(ALGraph G,KeyType v,KeyType w)
316 {
317     int i=LocateVex(G,v);
318     int j=LocateVex(G,w);
319     if(i==-1||j==-1)
320         return ERROR;
321     ArcNode *p=G.vertices[i]. firstarc ;//查找<v,w>
322     while(p!=NULL)
323     {
324         if(p->adjvex==j)
325             return OK;
326         p=p->nextarc;
327     }
328     return ERROR;
329 }
330
331
332 status InsertArc (ALGraph &G,KeyType v,KeyType w)
333 //在图G中增加弧<v,w>, 成功返回OK,否则返回ERROR
334 {
335     int i=LocateVex(G,v);
336     int j=LocateVex(G,w);
337     if(i==-1||j==-1)
338         return ERROR;
339     if(LocateArc(G,v,w)!=ERROR)
340         return ERROR;
341     ArcNode *p=(ArcNode*)malloc(sizeof(ArcNode));//插入<v,w>
```

```
342 p->adjvex=j;
343 p->nextarc=G.vertices[i].firstarc;
344 G.vertices[i].firstarc=p;
345 p=(ArcNode*)malloc(sizeof(ArcNode));
346 p->adjvex=i;
347 p->nextarc=G.vertices[j].firstarc;
348 G.vertices[j].firstarc=p;
349 G.arcnum++;
350 return OK;
351 }
352
353
354 status DeleteArc(ALGraph &G,KeyType v,KeyType w)
355 //在图G中删除弧<v,w>, 成功返回OK, 否则返回ERROR
356 {
357 if(LocateArc(G,v,w)==ERROR)
358 return ERROR;
359 int i=LocateVex(G,v);
360 int j=LocateVex(G,w); //删除<v,w>
361 ArcNode *p=G.vertices[i].firstarc; //删除<v,w>
362 ArcNode *temp=NULL;
363 if(p->adjvex==j)
364 {
365 temp=p;
366 G.vertices[i].firstarc=p->nextarc; //删除<v,w>
367 free(temp);
368 }
369 else
370 {
371 while(p->nextarc->adjvex!=j)
372 p=p->nextarc;
```

```
373
374 temp=p->nextarc;
375 p->nextarc=p->nextarc->nextarc; //删除<v,w>
376 free(temp);
377 }
378 p=G.vertices[j].firstarc; //删除<v,w>的对称弧<w,v>
379 temp=NULL; //删除<v,w>的对称弧<w,v>
380 if(p->adjvex==i)
381 {
382     temp=p;
383     G.vertices[j].firstarc=p->nextarc;
384     free(temp);
385 }
386 else
387 {
388     while(p->nextarc->adjvex!=i)
389         p=p->nextarc;
390
391     temp=p->nextarc;
392     p->nextarc=p->nextarc->nextarc;
393     free(temp);
394 }
395 G.arcnum--;
396 return OK;
397 }
398
399
400 void dfs(ALGraph &G,KeyType v,void(*visit)(VertexType))
401 {
402     visited[v]=TRUE;
403     int x=LocateVex(G,v);
```

```
404 visit (G. vertices [x]. data ); //访问顶点v
405 for ( int w=FirstAdjVex(G,v);w>=0;w=NextAdjVex(G,v,G.vertices[w].data.key))
406 if (! visited [G. vertices [w]. data .key])
407 dfs(G,G. vertices [w]. data .key, visit ); //递归调用
408 }
409
410
411 status DFSTraverse(ALGraph &G,void(*visit)(VertexType))
412 {
413 int i;
414 for ( i=0;i<G.vexnum;i++)
415 visited [G. vertices [ i ]. data .key]=FALSE;//标记数组记录关键字
416
417 for ( i=0;i<G.vexnum;i++)
418 if (! visited [G. vertices [ i ]. data .key]) //若未访问
419 dfs (G,G. vertices [ i ]. data .key, visit );
420
421 return OK;
422 }
423
424
425 status InitQueue(Linkqueue &Q)
426 {
427 Q.front=Q.rear=(QNode *)malloc(sizeof(QNode));
428 if (!Q.front )
429 return ERROR;
430 Q.front->next=NULL;//头结点指针域置空
431 return OK;
432 }
433
434
```

```
435 status QueueEmpty(Linkqueue Q)
436 {
437     if(Q.front==Q.rear)
438         return TRUE;//队列为空
439     else
440         return FALSE;
441 }
442
443
444 status enQueue(Linkqueue &Q,VertexType value)
445 {
446     Queue p=(Queue)malloc(sizeof(QNode));
447     if (!p)
448         return ERROR;
449     p->data=value;
450     p->next=NULL;//新结点指针域置空
451     Q.rear->next=p;
452     Q.rear=p;
453     return OK;
454 }
455
456
457 status deQueue(Linkqueue &Q,VertexType &value)
458 {
459     if(Q.front==Q.rear)
460         return ERROR;
461     Queue p=Q.front->next;
462     value=p->data;
463     Q.front->next=p->next;
464     if(Q.rear==p)
465         Q.rear=Q.front;
```

```
466 free(p);
467 return OK;
468 }
469
470
471 status BFSTraverse(ALGraph &G,void (*visit)(VertexType))
472 //对图G进行广度优先搜索遍历,
473 //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
474 {
475     int i=0,j;
476     VertexType value;
477     Linkqueue Q;
478     InitQueue(Q);
479     for(i=0;i<G.vexnum;i++)
480         visited[G.vertices[i].data.key]=FALSE; //标记数组记录关键字
481
482     for(i=0;i<G.vexnum;i++)
483         if(! visited[G.vertices[i].data.key])
484         {
485             visited[G.vertices[i].data.key]=TRUE; //标记已访问
486             visit(G.vertices[i].data);
487             enQueue(Q,G.vertices[i].data);
488             while(!QueueEmpty(Q))
489             {
490                 deQueue(Q,value);
491                 for(int w=FirstAdjVex(G,value.key); w>=0; w=NextAdjVex(G,value.key,G.vertices[w].data.key))
492                     if(! visited[G.vertices[w].data.key]) //如果未访问过
493                     {
494                         visited[G.vertices[w].data.key]=TRUE; //标记已访问
495                         visit(G.vertices[w].data);
496                         enQueue(Q,G.vertices[w].data);
```

```
497 }
498
499 }
500 }
501
502 return OK;
503 }
504
505
506 status SaveGraph(ALGraph G, char FileName[])
507 //将图的数据写入到文件FileName中
508 {
509 FILE *fp=fopen(FileName,"w");
510 fprintf ( fp, "%d %d ",G.vexnum,G.arcnum);//输出顶点数和边数
511
512 int i=0;
513 for ( i=0;i<G.vexnum;i++)
514 {
515 fprintf ( fp, "%d,%s ",G.vertices[ i ]. data .key,G. vertices [ i ]. data . others );
516 ArcNode *p=G.vertices[i]. firstarc ;
517 while(p!=NULL)
518 {
519 fprintf ( fp, "%d ",p->adjvex);//输出邻接点
520 p=p->nextarc;
521 }
522 fprintf ( fp, "%d ",-1);
523 }
524 fclose ( fp );
525 return OK;
526 }
527
```



```
528
529 status LoadGraph(ALGraph &G, char FileName[])
530 //读入文件FileName的图数据，创建图的邻接表
531 {
532 FILE *fp=fopen(FileName, "r");
533 char c,d;
534 char other [20];
535 int i,j,k;
536 fscanf(fp, "%d%d", &G.vexnum, &G.arcnum); //读入顶点数和边数
537
538 for(i=0; i<G.vexnum; i++)
539 {
540 fscanf(fp, "%d%c%s", &k, &c, other); //c用来接收逗号
541 G.vertices[i].data.key=k;
542 strcpy(G.vertices[i].data.others, other); //字符串复制函数
543 j=0;
544 ArcNode *rear=G.vertices[i].firstarc=NULL;
545 while(j!=-1)
546 {
547 fscanf(fp, "%d", &j);
548 if(j!=-1)
549 {
550 ArcNode *p=(ArcNode *)malloc(sizeof(ArcNode));
551 p->adjvex=j;
552 p->nextarc=NULL;
553 if(G.vertices[i].firstarc==NULL)
554 {
555 G.vertices[i].firstarc=p; //头指针指向第一个结点
556 rear=G.vertices[i].firstarc;
557 }
558 else
```

```
559 {
560 rear->nextarc=p;
561 rear=p;
562 }
563 }
564 }
565 rear=NULL;
566 }
567 fclose (fp);
568 return OK;
569 }
570
571
572
573 int ShortestPathLength (ALGraph G,KeyType v,KeyType w)
574 { //返回顶点v与顶点w的最短路径的长度
575 int x,y;
576 x=LocateVex(G,v);
577 y=LocateVex(G,w);
578 if(x==-1||y==-1||x==y)
579 return -1;
580 int vexcheck[21], vexdistance [21];
581 for ( int i=0;i<G.vexnum;i++)
582 {
583 vexcheck[i]=0;
584 vexdistance [ i]=0;
585 }
586 ArcNode *p;
587 int line [21], front=0,end=0;
588 line [0]=x;
589 end++;
```

```
590 vexcheck[x]=1;
591 while( front<end)
592 {
593 p=G.vertices [ line [ front ]]. firstarc ;
594 while(p!=NULL)
595 {
596 if(vexcheck[p->adjvex]==0)
597 {
598 line [end]=p->adjvex;
599 vexcheck[p->adjvex]=1;
600 vexdistance [p->adjvex]=vexdistance[ line [ front ]]+1;
601 end++;
602 }
603 p=p->nextarc;
604 }
605 front ++;
606 }
607 if(vexcheck[y]==0)
608 return -1;
609 else
610 return vexdistance [y];
611 }
612 status VerticesSetLessThank (ALGraph G,KeyType v,int k)
613 { //输出与顶点v距离小于k的顶点
614 int vexcheck[21], vexdistance [21];
615 for( int i=0;i<G.vexnum;i++)
616 {
617 vexcheck[i]=0;
618 vexdistance [ i ]=0;
619 }
620 int groud;
```

```
621 ArcNode *p;
622 groud=LocateVex(G,v);
623 if(groud==-1)
624     return ERROR;
625 int line [21], front=0,end=0;
626 line [0]=groud;
627 end++;
628 vexcheck[groud]=1;
629 while( front < end)
630 {
631     p=G.vertices [ line [ front ]]. firstarc ;
632     while(p!=NULL)
633     {
634         if(vexcheck[p->adjvex]==0)
635         {
636             line [end]=p->adjvex;
637             vexcheck[p->adjvex]=1;
638             vexdistance [p->adjvex]=vexdistance[ line [ front ]]+1;
639             end++;
640         }
641         p=p->nextarc;
642     }
643     front ++;
644 }
645 int sum=0;
646 for( int i=0;i<G.vexnum;i++)
647 {
648     if( i !=groud&&vexcheck[i]==1&&vexdistance[i]<k)
649     {
650         printf ( " %d %s\n",G.vertices[ i ]. data .key,G.vertices [ i ]. data . others );
651         sum++;
```

```
652 }
653 }
654 if(sum==0)
655 return ERROR;
656 return OK;
657 }
658
659 bool mark[20];
660 void dfs(ALGraph &G,KeyType v)
661 {
662 mark[v]=TRUE;
663 for (int w=FirstAdjVex(G,v);w>=0;w=NextAdjVex(G,v,G.vertices[w].data.key))
664 if (! mark[G.vertices [w].data .key]) //如果没有访问过
665 dfs(G,G.vertices [w].data .key);
666
667 }
668 status ConnectedComponentsNums(ALGraph G)//求图中连通分量个数
669 {
670 int count=0,i;
671 for (i=0;i<G.vexnum;i++)
672 mark[G.vertices [i].data .key]=FALSE;//标记数组记录关键字
673 for (i=0;i<G.vexnum;i++)//遍历所有顶点
674 if (! mark[G.vertices [i].data .key])
675 {
676 dfs(G,G.vertices [i].data .key);
677 count++;
678 }
679
680 return count;
681 }
```