華中科技大學

课 程 实 验 报 告

课程名称： 数据结构实验

专业班级	**计算机科学与技术**
学　　号	**U202414728**
姓　　名	**谈家能**
指导教师	**郝义学**
报告日期	**2025 年 6 月 15 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 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。

分析：需要了解链表的结构，同时演示系统要清晰明了。

1.2 系统设计

1.2.1 演示系统菜单的组织架构

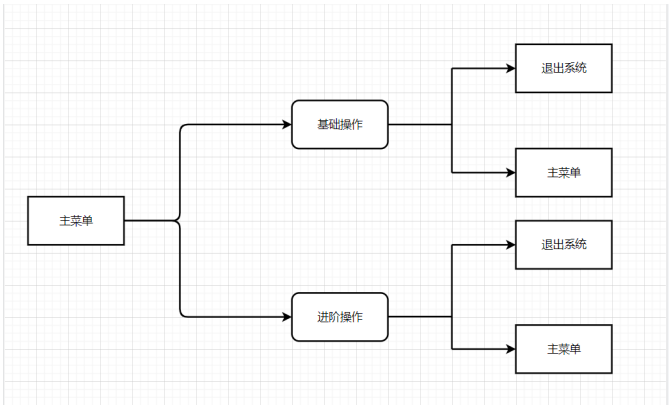


图 1-1 演示系统模块结构图

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，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。

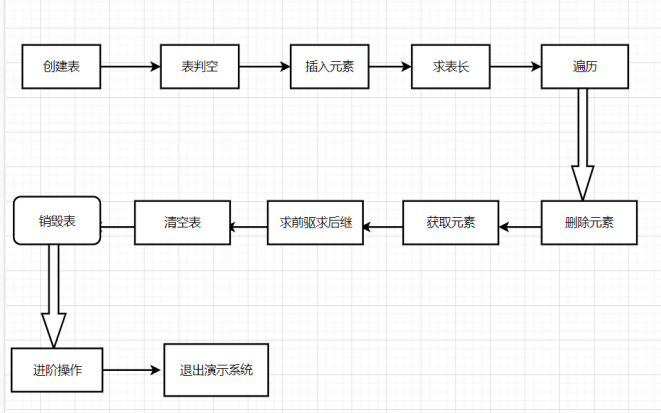


图 1-2　演示系统流程图

1.2.2　ADT 数据结构设计

对链表数据类型的定义如下

```
        typedef struct LNode {
    ElemType data;
    struct LNode* next;
}LNode, * LinkList;

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

对一些常量的定义如下

```
        #define TRUE 1
```

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

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

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: 若 L 为 NULL，返回 INFEASIBLE。用 while 循环依次释放所有节点的储存空间，再将 L 置为 NULL。

```
    status DestroyList(LinkList& L)
{
    if (L == NULL)
        return INFEASIBLE;
```

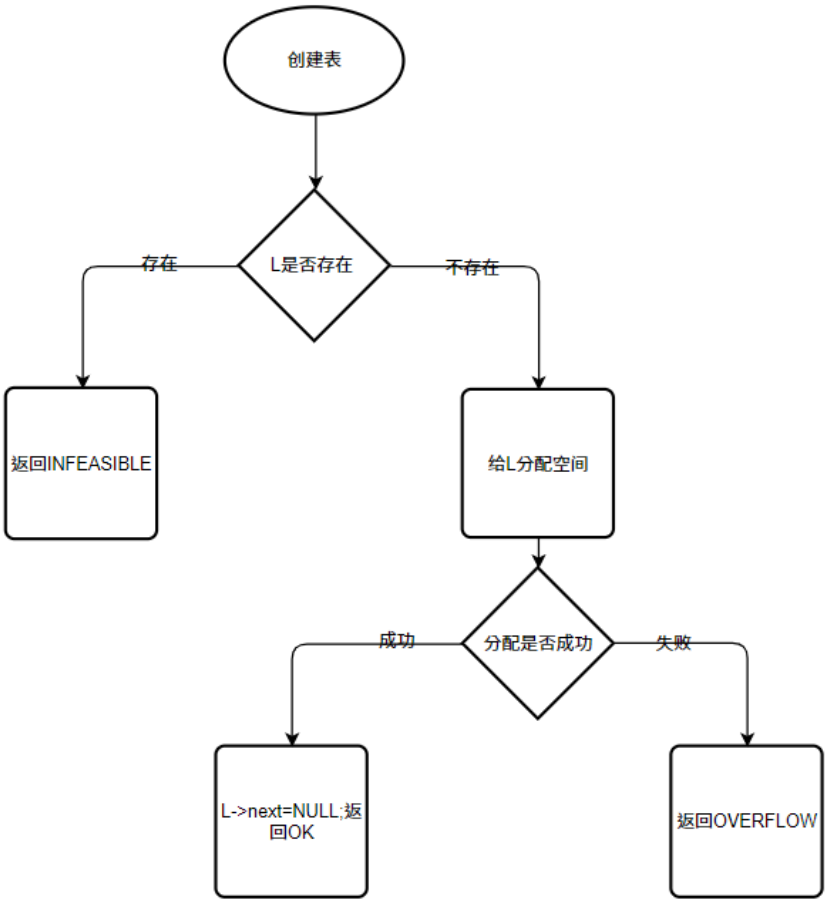


图 1-3　初始化表流程图

```
    else
    {
        LNode* p = NULL;
        while (L)
        {
            p = L;
            L = L->next;
            free(p);
        }
        L = NULL;
        return OK;
    }
}
```

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: 若 L 为 NULL，返回 INFEASIBLE。否则用 while 循环释放除头节点以外的所有节点的储存空间，再将 L->next 置为 NULL。

```
        status ClearList(LinkList& L)
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        LNode* p = L->next, * q = NULL;

        while (p)
        {
            q = p;
            p = p->next;
            free(q);
        }
        L->next = NULL;
        return OK;
    }
}
```

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: 若 L 为 NULL，返回 INFEASIBLE。否则判断 L->next 是否为 NULL，若为 NULL，则链表为空，否则链表不为空。

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

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

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) 若不为空则移动指针到下一个结点，长度加 1。
4) 返回长度的值。

时间复杂度：O(n)

空间复杂度：O(1)

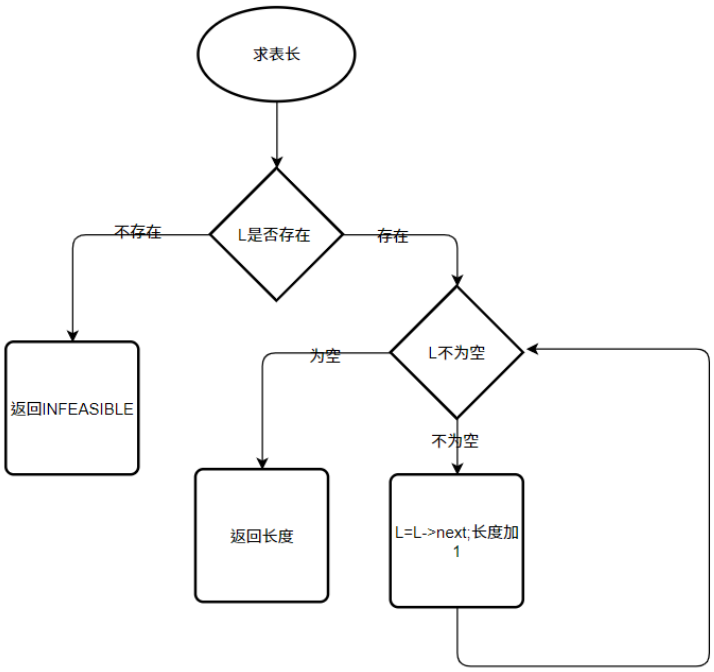


图 1-4　求表长流程图

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 L 为 NULL，返回 INFEASIBLE。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。

```
        status GetElem(LinkList L, int i, ElemType& e)
{
    int j = 1;
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        if (L->next == NULL)
            return ERROR;
        if (i < 1)
            return ERROR;
```

```
        for (LNode* p = L->next; j <= i; p = p->next, j++)
        {
            if (p == NULL)
                return ERROR;
            if (j == i)
            {
                e = p->data;
                return OK;
            }
        }
    }
}
```

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: 若 L 为 NULL，返回 INFEASIBLE。否则声明 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。

```
        status LocateElem(LinkList L, ElemType e, status(*CompareArr)
 (ElemType, ElemType))
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        LNode* p = L->next;
```

```
        for (int i = 1; p != NULL; p = p->next, i++)
            if (p->data == e)
                return i;
        return ERROR;
    }
}
```

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.

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)

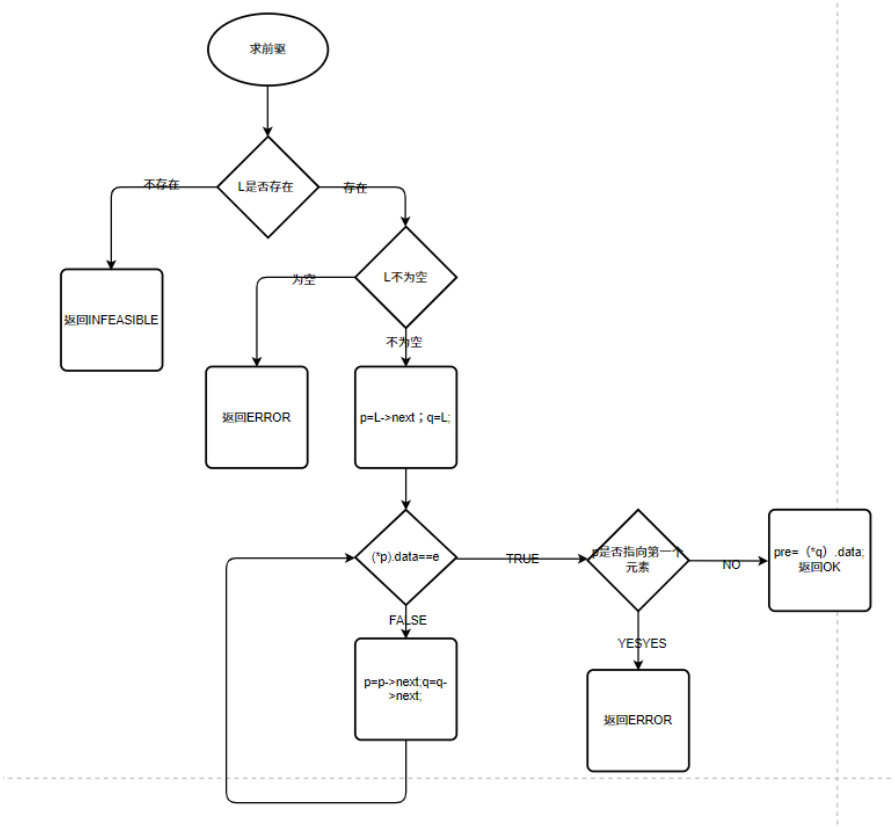


图 1-5 求前驱流程图

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: 若 L 为 NULL，返回 INFEASIBLE。否则链表查找值等于 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。

```
        status NextElem(LinkList L, ElemType e, ElemType& next)
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        if (L->next == NULL)
            return ERROR;
        LNode* p = L->next, * q = L->next->next;
        for (; q != NULL; p = p->next, q = q->next)
            if (p->data == e && q != NULL)
            {
                next = q->data;
                return OK;
            }
        return ERROR;
    }
}
```

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.

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, 判断 i 值是否符合要求, 不符合则返回 ERROR。
3) 如果 I 值合法，增加一个新的结点用于存储插入元素。
4) 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.
5) 返回 OK。

时间复杂度：O(n)

空间复杂度：O(1)

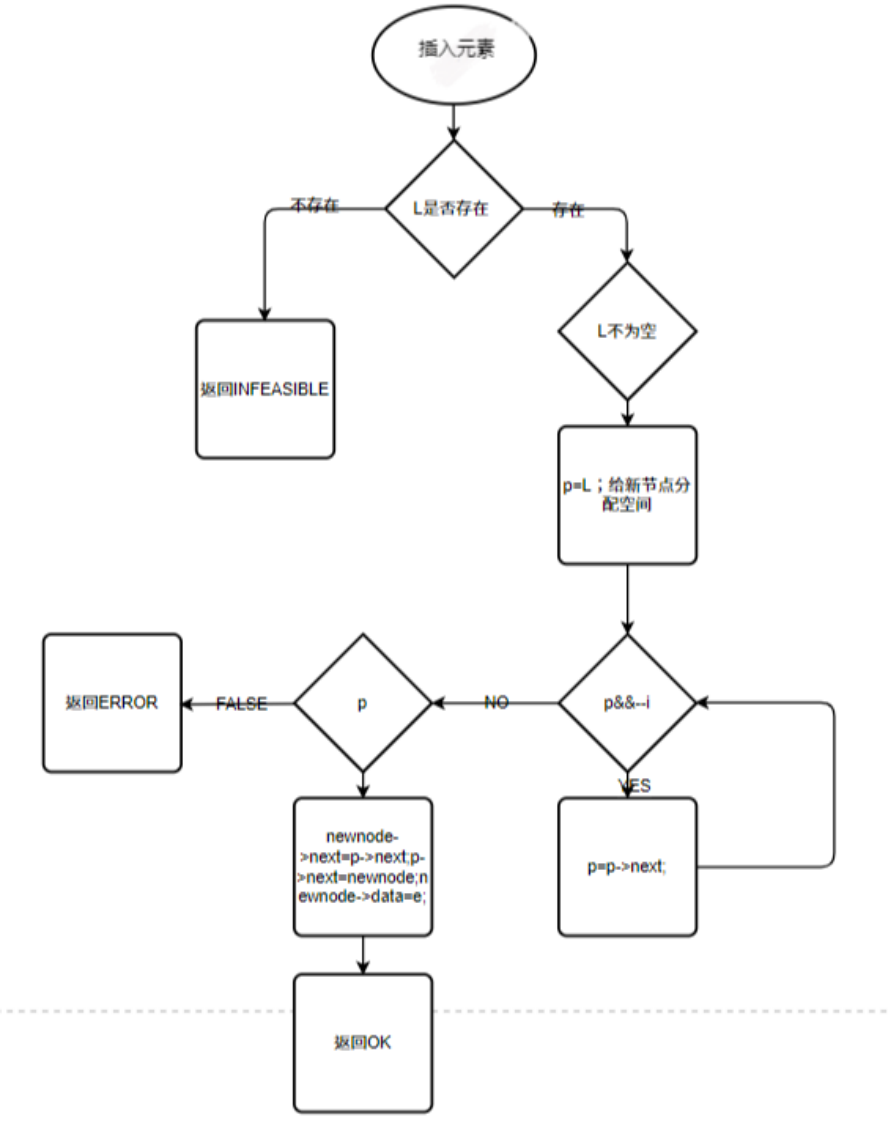


图 1-6　插入元素流程图

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 L 为 NULL，返回 INFEASIBLE。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。

```
        status ListDelete(LinkList& L, int i, ElemType& e)
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        int j = 1;
        for (LNode* p = L->next, *pre = L; p != NULL; p = p->next,
        pre = pre->next, j++)
            if (j == i)
            {
                e = p->data;
                if (p->next != NULL)
                    pre->next = p->next;
                else
                    pre->next = NULL;
                free(p);
                return OK;
            }
        return ERROR;
    }
}
```

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。

时间的复杂度:O(n).

空间的复杂度:O(1).

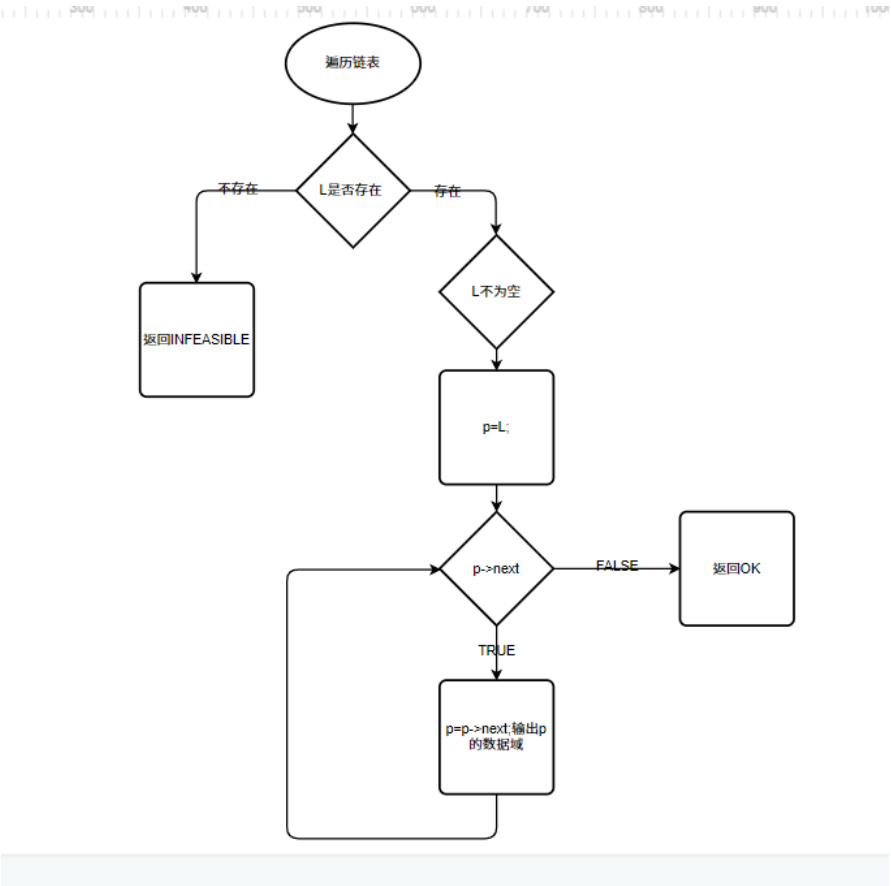


图 1-7 遍历链表流程图

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.

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。

```
        status RemoveNthFromEnd(LinkList& L, int n)
{
    if (L == NULL)
        return INFEASIBLE;
    else if (ListEmpty(L) == TRUE)
        return ERROR;
    else
    {
        int len, m, node;
        len = ListLength(L);
        if (n > len)
            return ERROR;
        m = len + 1 - n;
```

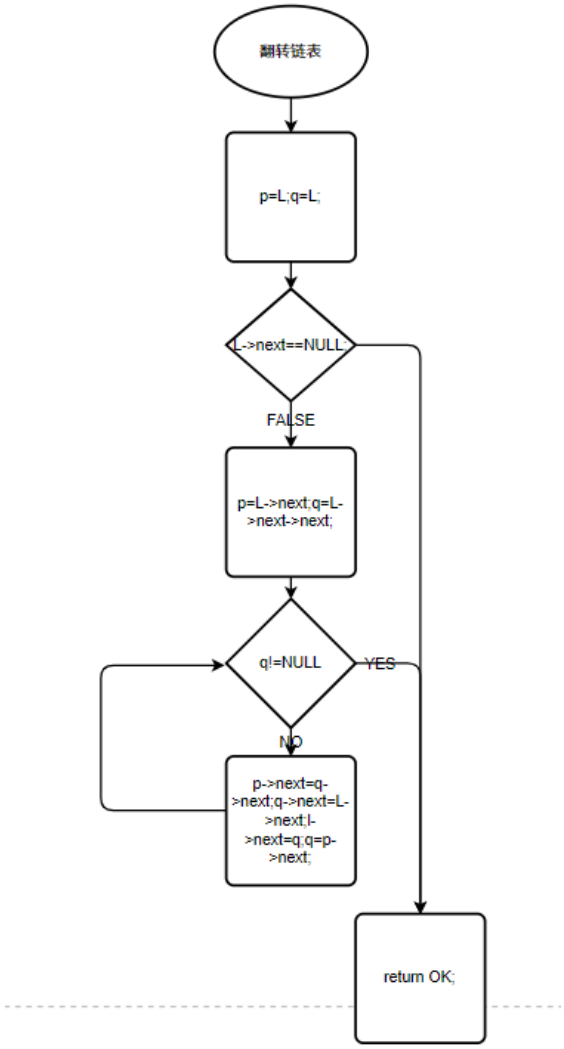


图 1-8　翻转链表流程图

```
        ListDelete(L, m, n);
        return n;
    }
}
```

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: 若 L 为 NULL，返回 INFEASIBLE。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。

```
        status LoadList(LinkList& L, char FileName[])
{
    if (L != NULL)
        return INFEASIBLE;
    else
    {
        int tmp;
        FILE* fp = fopen(FileName, "r");
        L = (LNode*)malloc(sizeof(LNode));
        L->next = NULL;
        LNode* q = L;
        while (fscanf(fp, "%d", &tmp) == 1)
        {
            LNode* p = (LNode*)malloc(sizeof(LNode));
            p->data = tmp;
            q->next = p;
            p->next = NULL;
            q = p;
        }
        fclose(fp);
        return OK;
    }
}
```

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: 若 L 为 NULL，返回 INFEASIBLE。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。

```
        status SaveList(LinkList L, char FileName[])
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        FILE* fp = fopen(FileName, "w");
        for (LNode* p = L->next; p != NULL; p = p->next)
            fprintf(fp, "%d␣", p->data);
        fclose(fp);
        return OK;
    }
}
```

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。

```
        status sortList(LinkList& L)
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        LNode* p, * q;
```

```
        for (p = L->next; p != NULL; p = p->next)
            for (q = p->next; q != NULL; q = q->next)
                if (q->data < p->data)
                {
                    int tmp = q->data;
                    q->data = p->data;
                    p->data = tmp;
                }
        return OK;
    }
}
```

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

PC 操作系统为 windows 操作系统使用语言为 C++。

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

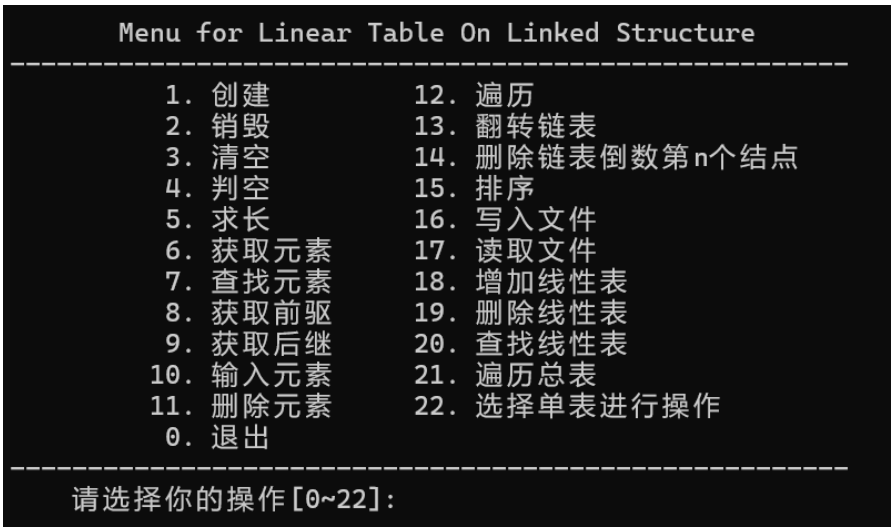


图 1-9　交互界面示例

1.4　系统测试

程序开发及实现环境：Win11 下使用 Dev C++ 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，比如在插入新结点时，必须先让新节点的指针域指向 p 指针所指元素的 next，然后才能让 p 指向新节点，反之则会大错特错。经过这次实验，我明白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。

分析：需要了解图的结构，同时演示系统要清晰明了。

2.2 系统设计

2.2.1 演示系统菜单的组织架构

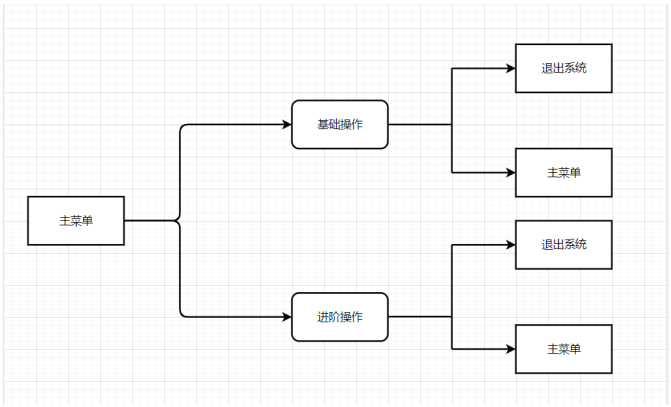


图 2-1 演示系统模块结构图

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。

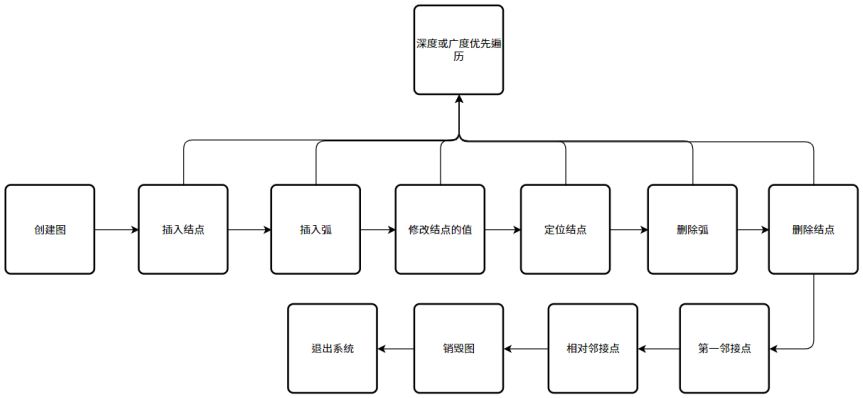


图 2-2　演示系统流程图

2.2.2　ADT 数据结构设计

对图数据类型的定义如下

```
typedef int status;
typedef int KeyType;
typedef enum { DG, DN, UDG, UDN } GraphKind;
typedef struct {
    KeyType key;
    char others[20];
} VertexType;


typedef struct ArcNode {
    int adjvex;
    struct ArcNode* nextarc;
} ArcNode;
```

```
typedef struct VNode {
    VertexType data;
    ArcNode* firstarc;
} VNode, AdjList[MAX_VERTEX_NUM];

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

对一些常量的定义如下

```
       #define TRUE 1     判定结果为正确
#define FALSE 0    判定结果为错误
#define OK 1       功能运行正常
#define ERROR 0    功能出现错误无法运行
#define INFEASIBLE -1操作对象不存在
#define OVERFLOW -2 内存溢出

#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

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

```
status CreateCraph(ALGraph& G, VertexType V[], KeyType VR[][2])
{
    if (V[0].key == -1)
        return ERROR;
    int a, i, j, flag1 = 0, flag2 = 0;
    KeyType keys[30];
    for (a = 0; a < 20; a++)
        keys[a] = 0;
    a = 0;
    for (i = 0; V[i].key != -1; i++)
    {
        if (i >= MAX_VERTEX_NUM)
            return ERROR;
        keys[a++] = V[i].key;
        for (j = i + 1; V[j].key != -1; j++)
            if (V[i].key == V[j].key)
                return ERROR;
    }
    for (i = 0; VR[i][0] != -1; i++)
    {
        flag1 = 0, flag2 = 0;
        for (a = 0; keys[a] != 0; a++)
        {
            if (VR[i][0] == keys[a])
                flag1 = 1;
            if (VR[i][1] == keys[a])
                flag2 = 1;
        }
```

```
    if (!flag1 || !flag2)
        return ERROR;
}
G.kind = UDG;
G.vexnum = 0;
G.arcnum = 0;
for (i = 0; V[i].key != -1 && i < MAX_VERTEX_NUM; i++)
{
    G.vertices[i].data = V[i];
    G.vertices[i].firstarc = NULL;
    G.vexnum++;
}
for (i = 0; VR[i][0] != -1; i++)
{
    flag1 = 0, flag2 = 0;
    for (a = 0; keys[a] != 0; a++)
    {
        if (VR[i][0] == keys[a])
            flag1 = a;
        if (VR[i][1] == keys[a])
            flag2 = a;
    }
    ArcNode* a1 = (ArcNode*)malloc(sizeof(ArcNode));
    a1->adjvex = flag2;
    a1->nextarc = G.vertices[flag1].firstarc;
    G.vertices[flag1].firstarc = a1;
    ArcNode* a2 = (ArcNode*)malloc(sizeof(ArcNode));
    a2->adjvex = flag1;
    a2->nextarc = G.vertices[flag2].firstarc;
    G.vertices[flag2].firstarc = a2;
    G.arcnum++;
```

```
    }
    return OK;
}
```

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

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。

```
int LocateVex(ALGraph G, KeyType u)
{
    int i;
    for (i = 0; i < G.vexnum; i++)
        if (u == G.vertices[i].data.key)
```

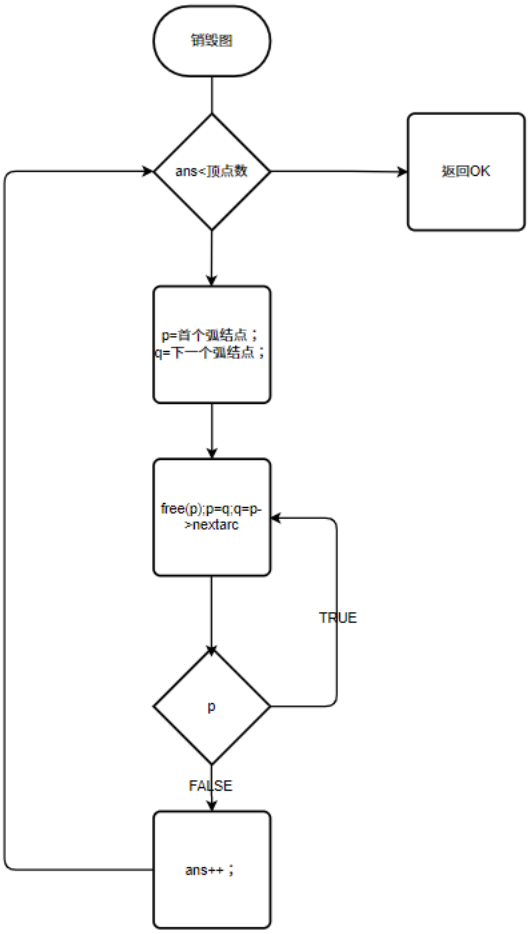


图 2-3　销毁表流程图

```
            return i;
    return -1;
}
```

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

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)

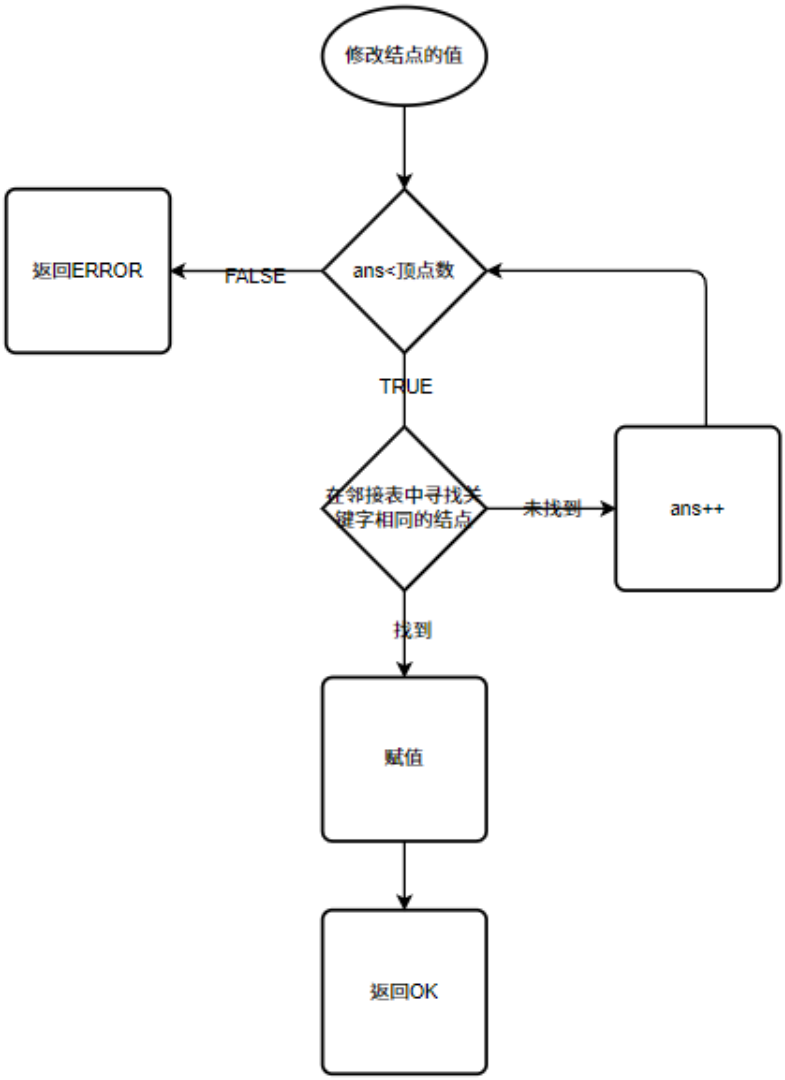


图 2-4　修改结点流程图

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 ||

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->adjvex。

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)

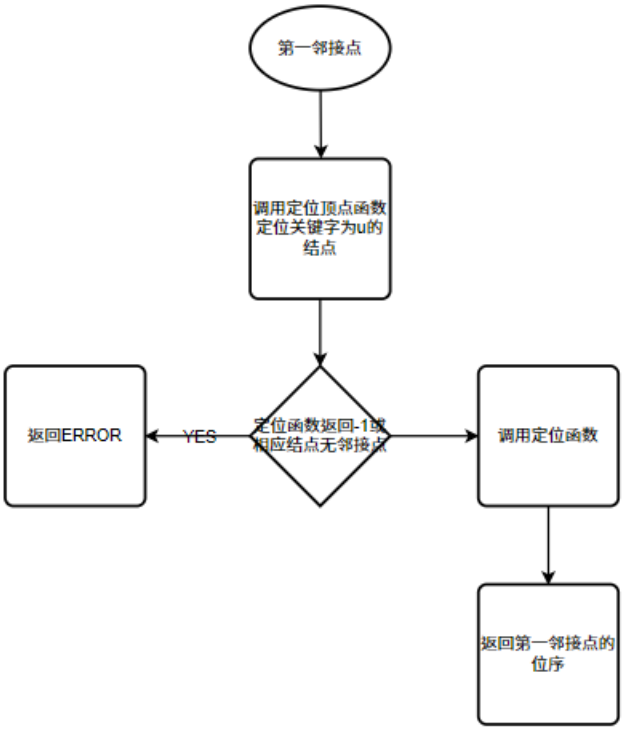


图 2-5　定位第一邻接点流程图

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，找到另个一顶点，用 p 指向该邻接点。如果 p == NULL || p->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->nextarc->adjvex。

```
int NextAdjVex(ALGraph G,KeyType v,KeyType w)
{
        int i=LocateVex(G,v),j=LocateVex(G,w);
        while(i!=-1&&j!=-1)
        {
                ArcNode *p=G.vertices[i].firstarc;
                while(p->adjvex!=j&&p)
                {
                        p=p->nextarc;
                }
                if(p->nextarc)
                {
                        return p->nextarc->adjvex;
                }
                else return -1;
        }
        return -1;

}
```

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。

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

```
{
    if (G.vexnum == MAX_VERTEX_NUM)
        return ERROR;
    int i;
    for (i = 0; i < G.vexnum; i++)
        if (v.key == G.vertices[i].data.key)
            return ERROR;
    G.vertices[G.vexnum].data = v;
    G.vertices[G.vexnum].firstarc = NULL;
    G.vexnum++;
    return OK;
}
```

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

```
        status DeleteVex(ALGraph& G, KeyType v)
{
    if (G.vexnum == 1)
        return ERROR;
    int i, j = 0, k;
    for (i = 0; i < G.vexnum; i++)
        if (v == G.vertices[i].data.key)
            break;
    if (i == G.vexnum)
        return ERROR;
```

```
ArcNode* a = G.vertices[i].firstarc, * pre = NULL;
while (a)
{
    pre = a;
    a = a->nextarc;
    ArcNode* b = G.vertices[pre->adjvex].firstarc, * preb = NULL;
    if (b->adjvex == i)
    {
        G.vertices[pre->adjvex].firstarc = b->nextarc;
        free(b);
    }
    else
        while (b)
        {
            preb = b;
            b = b->nextarc;
            if (b->adjvex == i)
            {
                preb->nextarc = b->nextarc;
                free(b);
                break;
            }
        }
    free(pre);
    G.arcnum--;
}
G.vertices[i].firstarc = NULL;
ArcNode* c = NULL;
for (j = 0; j < G.vexnum; j++)
{
    c = G.vertices[j].firstarc;
```

```
        while (c != NULL)
        {
            if (c->adjvex > i)
                c->adjvex--;
            c = c->nextarc;
        }
    }
    for (k = i + 1; k < G.vexnum; i++, k++)
    {
        G.vertices[i].data = G.vertices[k].data;
        G.vertices[i].firstarc = G.vertices[k].firstarc;
    }
    G.vexnum--;
    return OK;
}
```

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。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)

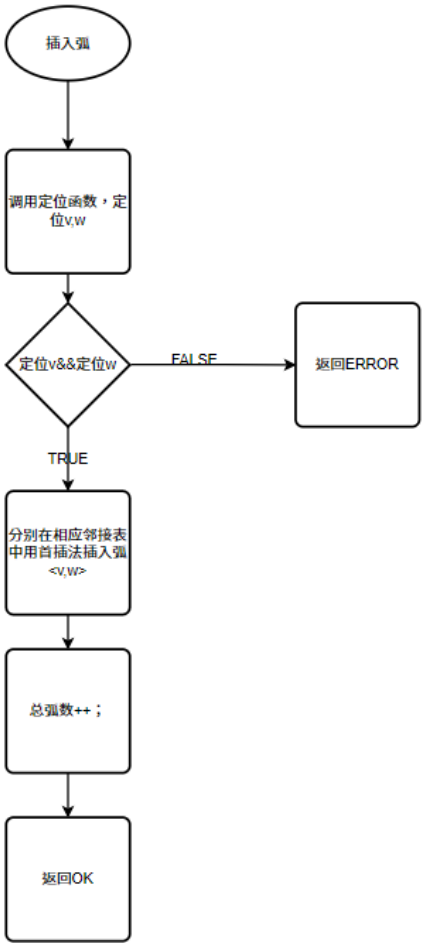


图 2-6 插入弧流程图

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。

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)

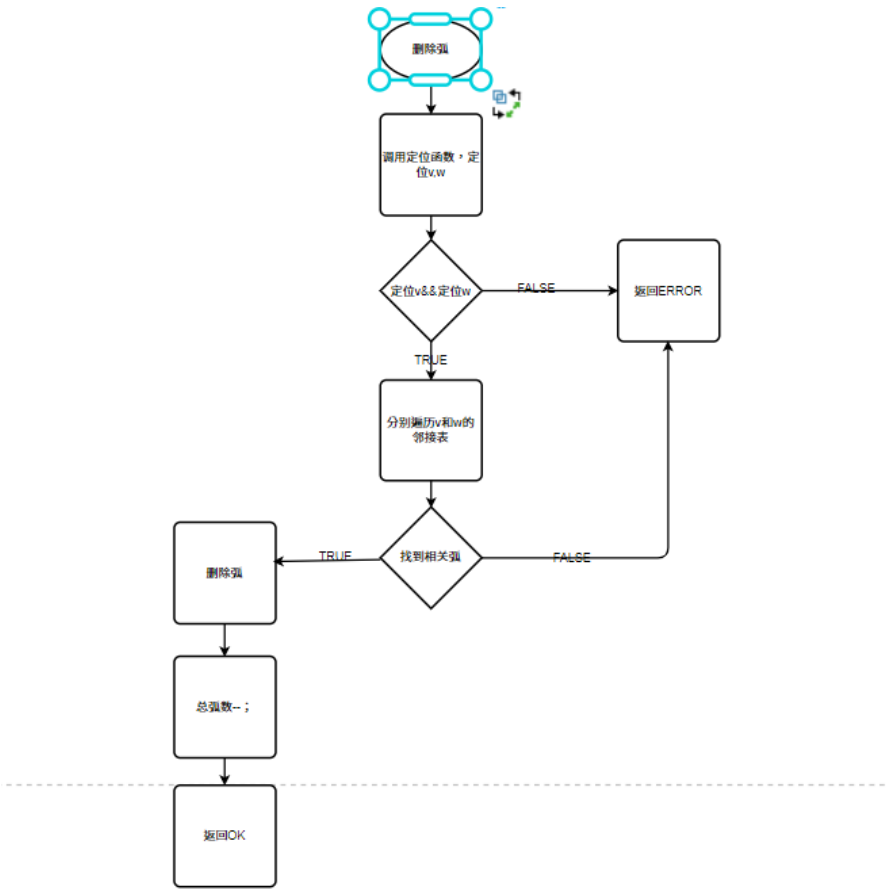


图 2-7 删除弧流程图

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: 如果 G.vexnum==0，即图不存在，返回 INFEASIBLE。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。。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 fps
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)

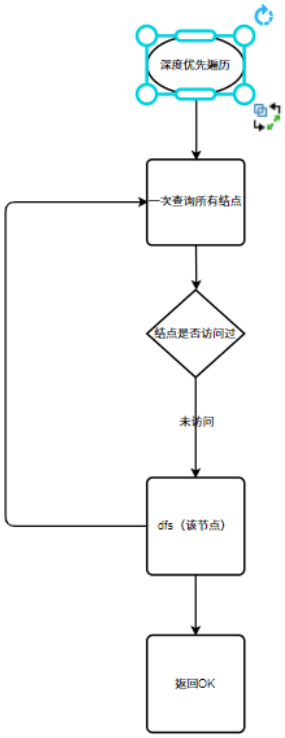


图 2-8 深度优先遍历流程图

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。

```
        status BFSTraverse(ALGraph& G, void (*visit)(VertexType))
{
    int v, u, w = -1;
    QUEUE Q;
    for (v = 0; v < G.vexnum; v++)
        visited[v] = 0;
    iniQueue(Q);
    for (v = 0; v < G.vexnum; v++)
        if (!visited[v])
        {
            visited[v] = 1;
            visit(G.vertices[v].data);
            enQueue(Q, v);
            while (Q.length)
            {
                deQueue(Q, u);
                for (w = FirstAdjVex(G, G.vertices[u].data.key);
                w >= 0; w = NextAdjVex(G, G.vertices[u].data.key,
G.vertices[w].data.key))
                    if (!visited[w])
                    {
                        visited[w] = 1;
                        visit(G.vertices[w].data);
                        enQueue(Q, w);
                    }
```

```
            }
        }
    return OK;
}
```

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

PC 操作系统为 windows 操作系统使用语言为 C++。

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

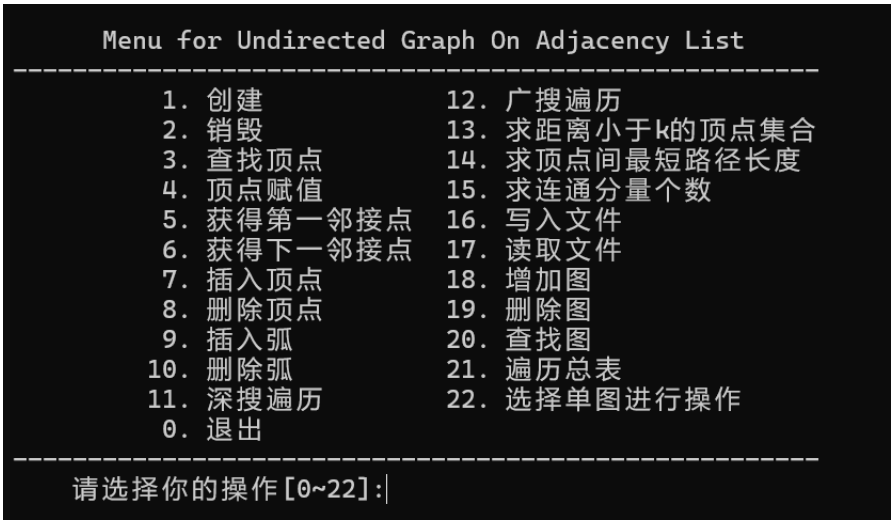


图 2-9 交互界面示例

2.4 系统测试

程序开发及实现环境：Win11 下使用 Dev C++ 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

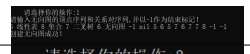
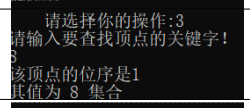
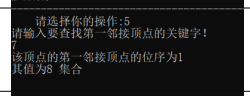
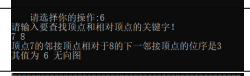
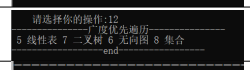
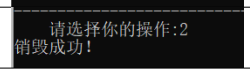
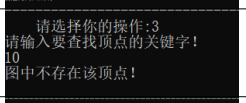
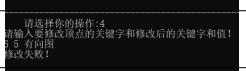
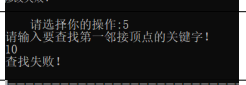
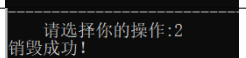
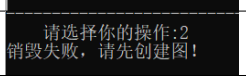
函数	输入	实际输出	预期结果
创建图	5..5 6 5 7 6 7 7 8 -1 -1		创建无向图成功
定位顶点	8		返回关键字为 8 的顶点的位序 1
修改顶点	6 9 有向图		修改成功
查找第一邻接点	7		该顶点的第一邻接顶点的位序为 1 其值为 8 集合
查找下一邻接点	7 8		顶点 7 的邻接顶点相对于 8 的下一邻接顶点的位序是 3 其值为 6 无向
深度优先遍历	11		5 线性表 7 二叉树 8 集合 6 无向图
广度优先遍历	12		5 线性表 7 二叉树 6 无向图 8 集合
销毁图	2		销毁成功

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		创建无向图成功
定位顶点	10		找不到关键字为 10 的顶点
修改顶点	6 5 有向图		关键字为 5 的结点已存在
查找第一邻接点	10		找不到关键字为 10 的顶点
销毁图	2		销毁成功
销毁图	2		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介的一大关键，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。

同时在实验过程中，尤其是在 debug 的过程中，我明白了看书看懂了并不代表自己就掌握了一种数据结构，实际上还差的很远，唯有动手实践，多用样例测试，才能了解与熟练运用它。

许许多多的细节问题不通过实践，是无法学到的。

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进了一步。

这门课程可以说是计算机专业的基础课程，也是核心课程。

经过一学期的学习，我希望未来数据结构课能够越来越好。

参考文献

附录 A 基于链式存储结构线性表实现的源程序

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

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

typedef int status;
typedef int ElemType;

#define LIST_INIT_SIZE 100
#define LISTINCREMENT 10

typedef struct LNode {
    ElemType data;
    struct LNode* next;
}LNode, * LinkList;

typedef struct {
    struct {
        char name[30];
```

```
        LinkList L;
    }elem[10];
    int length;
    int listsize;
}LISTS;
status Compare(ElemType e, ElemType elem);
void visit(ElemType elem);

status(*CompareArr)(ElemType e, ElemType elem) = Compare;
void (*visitArr)(ElemType elem) = visit;

status InitList(LinkList& L);
status DestroyList(LinkList& L);
status ClearList(LinkList& L);
status ListEmpty(LinkList L);
int ListLength(LinkList L);
status GetElem(LinkList L, int i, ElemType& e);
int LocateElem(LinkList L, ElemType e, status(*CompareArr)
(ElemType, ElemType));
status PriorElem(LinkList L, ElemType cur, ElemType& pre_e);
status NextElem(LinkList L, ElemType cur, ElemType& next_e);
status ListInsert(LinkList& L, int i, ElemType e);
status ListDelete(LinkList& L, int i, ElemType& e);
status ListTraverse(LinkList L, void (*visitArr)(ElemType));
status reverseList(LinkList& L);
int RemoveNthFromEnd(LinkList& L, int n);
status sortList(LinkList& L);
status SaveList(LinkList L, char FileName[]);
status LoadList(LinkList& L, char FileName[]);
status AddList(LISTS& Lists, char ListName[]);
status RemoveList(LISTS& Lists, char ListName[]);
```

```
int LocateList(LISTS Lists, char ListName[]);
status ListsTraverse(LISTS Lists);
LinkList* ChooseList(LinkList* L, LISTS& Lists, int i);



int main(void)
{
    定义单表并初始化
    LinkList L1 = NULL;
    LinkList* L = &L1;

    定义管理表并初始化
    LISTS Lists;
    Lists.length = 0, Lists.listsize = 0;

    int op = 1, len, flag, i, e, pre, next, num, k;
    char FileName[30], ListName[30];
    while (op)
    {
        system("cls");
        printf("\n\n");
        printf("请选择你的操作[0~22]:");
        scanf("%d", &op);
        switch (op)
        {
        case 1:创建
            if (InitList(*L) == OK)
                printf("线性表创建成功! \n这是一个与独立于多表的单表,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣你可以对它进行1~17操作! \n");
            else
```

```
            printf("线性表创建失败! \n");
        getchar();getchar();
        break;
    case 2:销毁
        if (DestroyList(*L) == OK)
            printf("成功销毁线性表并释放数据元素的空间! \n");
        else
            printf("不能对不存在的线性表进行销毁操作! \n");
        getchar();getchar();
        break;
    case 3:清空
        if (ClearList(*L) == OK)
            printf("成功删除线性表中所有的元素! \n");
        else
            printf("不能对不存在的线性表进行清空操作! \n");
        getchar();getchar();
        break;
    case 4:判空
        flag = ListEmpty(*L);
        if (flag == TRUE)
            printf("线性表为空! \n");
        else if (flag == FALSE)
            printf("线性表非空! \n");
        else
            printf("不能对不存在的线性表判空! \n");
        getchar();getchar();
        break;
    case 5:求长
        len = ListLength(*L);
        if (len >= 0)
            printf("线性表的长度为%d! \n", len);
```

```
        else
            printf("不能对不存在的线性表求长! \n");
        getchar();getchar();
        break;
    case 6:获取元素
        printf("请输入想要从线性表中获取的元素序号: ");
        scanf("%d", &i);
        getchar();
        flag = GetElem(*L, i, e);
        if (flag == OK)
            printf("线性表中第%d个元素为%d! \n", i, e);
        else if (flag == ERROR)
            printf("输入的序号不合法! \n");
        else
            printf("不能对不存在的线性表进行此操作! \n");
        getchar();getchar();
        break;
    case 7:查找元素
        printf("请输入想要在线性表中查找的元素: ");
        scanf("%d", &e);
        getchar();
        flag = LocateElem(*L, e, Compare);
        if (flag == 0)
            printf("线性表中不存在%d这个元素。\n", e);
        else if (flag == INFEASIBLE)
            printf("不能对不存在的线性表进行此操作! \n");
        else
            printf("元素%d在线性表中的序号为%d\n", e, flag);
        getchar();getchar();
        break;
    case 8:查找前驱
```

```
            printf("请输入想要在线性表中查找其前驱的元素: ");
            scanf("%d", &e);
            getchar();
            flag = PriorElem(*L, e, pre);
            if (flag == OK)
                printf("元素%d在线性表中的前驱为%d! \n", e, pre);
            else if (flag == ERROR)
                printf("元素%d在线性表中没有前驱! \n", e);
            else
                printf("不能对不存在的线性表进行此操作! \n");
            getchar();getchar();
            break;
        case 9:查找后继
            printf("请输入想要在线性表中查找其后继的元素: ");
            scanf("%d", &e);
            getchar();
            flag = NextElem(*L, e, next);
            if (flag == OK)
                printf("元素%d在线性表中的后继为%d! \n", e, next);
            else if (flag == ERROR)
                printf("元素%d在线性表中没有后继! \n", e);
            else
                printf("不能对不存在的线性表进行此操作! \n");
            getchar();getchar();
            break;
        case 10:输入元素
            printf("1.插入单个元素\n2.在表尾输入多个元素\n请选择将要进
␣␣␣␣␣␣␣␣␣␣␣␣行的操作: ");
            scanf("%d", &flag);
            getchar();
            switch (flag)
```

```
{
case 1:
    printf("请输入该元素: ");
    scanf("%d", &e);
    getchar();
    printf("请输入想要它作为第几个元素: ");
    scanf("%d", &i);
    getchar();
    flag = ListInsert(*L, i, e);
    if (flag == OK)
        printf("元素已插入线性表中! \n");
    else if (flag == ERROR)
        printf("位置插入非法! \n");
    else
        printf("不能对不存在的线性表进行插入! \n");
    break;
case 2:
    printf("请输入元素个数: ");
    scanf("%d", &num);
    getchar();
    printf("请输入这些元素, 以空格间隔: ");
    LNode* p = *L;
    for (; p->next != NULL; p = p->next)
        ;
    for (int i = 0; i < num; i++)
    {
        p->next = (LNode*)malloc(sizeof(LNode));
        p = p->next;
        scanf("%d", &p->data);
        p->next = NULL;
    }
```

```
            getchar();
            printf("元素已输入于线性表中! \n");
            break;
        }
        getchar();getchar();
        break;
    case 11:删除元素
        printf("请输入想要删除的元素序号: ");
        scanf("%d", &i);
        getchar();
        flag = ListDelete(*L, i, e);
        if (flag == OK)
            printf("序号为%d的元素%d已经从线性表中删除! \n", i, e);
        else if (flag == ERROR)
            printf("位置输入非法! \n");
        else
            printf("不能对不存在的线性表进行删除! \n");
        getchar();getchar();
        break;
    case 12:遍历
        if (ListTraverse(*L, visit) == INFEASIBLE)
            printf("不能对不存在的线性表进行遍历! \n");
        getchar();getchar();
        break;
    case 13:翻转链表
        flag = reverseList(*L);
        if (flag == OK)
            printf("当前链表已翻转! \n");
        else
            printf("不能对不存在的线性表进行翻转! \n");
        getchar();getchar();
```

```
            break;
        case 14:删除倒数第n个结点
            int n;
            printf("本功能实现删除链表倒数第n个结点, 请输入n: ");
            scanf("%d", &n);
            flag = RemoveNthFromEnd(*L, n);
            if (flag == INFEASIBLE)
                printf("不能对不存在的线性表进行删除操作! \n");
            else if (flag == ERROR)
                printf("不能对空线性表进行删除操作或n不合法! \n");
            else
                printf("链表倒数第%d个元素%d已被删除! \n", n, flag);
            getchar();getchar();
            break;
        case 15:排序
            flag = sortList(*L);
            if (flag == INFEASIBLE)
                printf("不能对不存在的线性表进行排序! \n");
            else
                printf("线性表已从小到大排序! \n");
            getchar();getchar();
            break;
        case 16:写入文件
            printf("请输入文件名称: ");
            scanf("%s", FileName);
            flag = SaveList(*L, FileName);
            if (flag == ERROR)
                printf("文件打开失败! \n");
            else if (flag == OK)
                printf("线性表的内容已经复制到名称为%s的文件中! \n",
                FileName);
```

```
            else if (flag == INFEASIBLE)
                printf("不能对不存在的线性表进行进行写文件操作! \n");
            getchar();getchar();
            break;
        case 17:读取文件
            printf("请输入文件名称: ");
            scanf("%s", FileName);
            flag = LoadList(*L, FileName);
            if (flag == ERROR)
                printf("文件打开失败! \n");
            else if (flag == OK)
                printf("名称为%s的文件中的内容已复制到线性表中! \n"
, FileName);
            else if (flag == INFEASIBLE)
                printf("不能对已存在的线性表进行进行读文件操作! 请先
销毁线性表! \n");
            else if (flag == OVERFLOW)
                printf("溢出! \n");
            getchar();getchar();
            break;
        case 18:新增线性表
            printf("请输入新增线性表的名称: ");
            scanf("%s", ListName);
            flag = AddList(Lists, ListName);
            if (flag == OK)
                printf("成功新增名称为%s的线性表! \n", ListName);
            else
                printf("新增失败! \n");
            getchar();getchar();
            break;
        case 19:删除线性表
```

```
        if (Lists.length)
        {
            for (int i = 0; i < Lists.length; i++)
                printf("%d␣%s\n", i + 1, Lists.elem[i].name);
        }
        else
        {
            printf("线性表的集合为空! 无法进行此操作! \n");
            getchar();getchar();
            break;
        }
        printf("请输入想要删除的线性表的名称: ");
        scanf("%s", ListName);
        flag = RemoveList(Lists, ListName);
        if (flag == OK)
            printf("成功删除名称为%s的线性表! \n", ListName);
        else if (flag == ERROR)
            printf("删除失败! \n");
        getchar();getchar();
        break;
    case 20:查找线性表
        if (!Lists.length)
        {
            printf("线性表的集合为空! 无法进行此操作! \n");
            getchar();getchar();
            break;
        }
        printf("请输入想要查找的线性表的名称: ");
        scanf("%s", ListName);
        i = LocateList(Lists, ListName);
        if (i == 0)
```

```
            printf("不存在名称为%s的线性表! \n", ListName);
        else printf("名称为%s的线性表的序号为%d! \n", ListName, i);
        getchar();getchar();
        break;
    case 21:遍历总表
        flag = ListsTraverse(Lists);
        if (flag == ERROR)
            printf("线性表的集合为空! \n");
        getchar();getchar();
        break;
    case 22:选择多表中的表进行单独操作
        if (Lists.length)
        {
            for (int i = 0; i < Lists.length; i++)
                printf("%d␣%s\n", i + 1, Lists.elem[i].name);
        }
        else
        {
            printf("线性表的集合为空! 无法进行此操作! \n");
            getchar();getchar();
            break;
        }
        printf("请输入想要进行操作的线性表的序号 (从1开始) : ");
        scanf("%d", &i);
        getchar();
        L = ChooseList(L, Lists, i);
        if (L == NULL)
            printf("输入的序号不合法! 单表已置空! \n");
        else
            printf("下面可对线性表集合中序号为␣%d␣的线性表进行操作!
\n", i);
```

```
            getchar();getchar();
            break;
        case 0:
            break;
        }
    }
    printf("欢迎下次再使用本系统! \n");
    return 0;
}
status Compare(ElemType e, ElemType elem)
{
    if (elem == e)
        return OK;
    return ERROR;
}

void visit(ElemType elem)
{
    printf("%d", elem);
}

status InitList(LinkList& L)
{
    if (L != NULL)
        return INFEASIBLE;
    else
    {
        L = (LNode*)malloc(sizeof(LNode));
        L->next = NULL;
        return OK;
    }
```

```
}

status DestroyList(LinkList& L)
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        LNode* p = NULL;
        while (L)
        {
            p = L;
            L = L->next;
            free(p);
        }
        L = NULL;
        return OK;
    }
}

status ClearList(LinkList& L)
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        LNode* p = L->next, * q = NULL;

        while (p)
        {
            q = p;
```

```
            p = p->next;
            free(q);
        }
        L->next = NULL;
        return OK;
    }
}

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

int ListLength(LinkList L)
{
    int num = 0;
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        for (LNode* p = L->next; p != NULL; p = p->next)
            num++;
        return num;
    }
}

status GetElem(LinkList L, int i, ElemType& e)
```

```
{
    int j = 1;
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        if (L->next == NULL)
            return ERROR;
        if (i < 1)
            return ERROR;
        for (LNode* p = L->next; j <= i; p = p->next, j++)
        {
            if (p == NULL)
                return ERROR;
            if (j == i)
            {
                e = p->data;
                return OK;
            }
        }
    }
}

int LocateElem(LinkList L, ElemType e, status(*CompareArr)
(ElemType, ElemType))
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        LNode* p = L->next;
```

```
        for (int i = 1; p != NULL; p = p->next, i++)
            if (p->data == e)
                return i;
        return ERROR;
    }
}

status PriorElem(LinkList L, ElemType e, ElemType& pre)
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        LNode* p = L->next, * q = L;
        for (; p != NULL; p = p->next, q = q->next)
            if (p->data == e && q != L)
            {
                pre = q->data;
                return OK;
            }
        return ERROR;
    }
}

status NextElem(LinkList L, ElemType e, ElemType& next)
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        if (L->next == NULL)
```

```
            return ERROR;
        LNode* p = L->next, * q = L->next->next;
        for (; q != NULL; p = p->next, q = q->next)
            if (p->data == e && q != NULL)
            {
                next = q->data;
                return OK;
            }
        return ERROR;
    }
}

status ListInsert(LinkList& L, int i, ElemType e)
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        int j = 1;
        for (LNode* p = L->next; p != NULL; p = p->next)
        {
            j++;
            if (j == i)
            {
                LNode* q = (LNode*)malloc(sizeof(LNode));
                q->data = e;
                q->next = p->next;
                p->next = q;
                return OK;
            }
        }
```

```
        return ERROR;
    }
}

status ListDelete(LinkList& L, int i, ElemType& e)
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        int j = 1;
        for (LNode* p = L->next, *pre = L; p != NULL; p = p->next,
        pre = pre->next, j++)
            if (j == i)
            {
                e = p->data;
                if (p->next != NULL)
                    pre->next = p->next;
                else
                    pre->next = NULL;
                free(p);
                return OK;
            }
        return ERROR;
    }
}

status ListTraverse(LinkList L, void (*visitArr) (ElemType))
{
    if (L == NULL)
        return INFEASIBLE;
```

```
    else
    {
       for (LNode* p = L->next; p != NULL; p = p->next)
           if (p != L)
               visit(p->data);
        return OK;
    }
}

status reverseList(LinkList& L)
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        LinkList p;
        LNode* q, * pnext = NULL;
        p = (LNode*)malloc(sizeof(LNode));
        p->next = pnext;
        for (q = L->next; q != NULL; q = q->next)
        {
            p->data = q->data;
            pnext = p;
            p = (LNode*)malloc(sizeof(LNode));
            p->next = pnext;
        }
        L = p;
        return OK;
    }
}
```

```
int RemoveNthFromEnd(LinkList& L, int n)
{
    if (L == NULL)
        return INFEASIBLE;
    else if (ListEmpty(L) == TRUE)
        return ERROR;
    else
    {
        int len, m, node;
        len = ListLength(L);
        if (n > len)
            return ERROR;
        m = len + 1 - n;
        ListDelete(L, m, n);
        return n;
    }
}

status sortList(LinkList& L)
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        LNode* p, * q;
        for (p = L->next; p != NULL; p = p->next)
            for (q = p->next; q != NULL; q = q->next)
                if (q->data < p->data)
                {
                    int tmp = q->data;
                    q->data = p->data;
```

```
                p->data = tmp;
            }
        return OK;
    }
}

status SaveList(LinkList L, char FileName[])
{
    if (L == NULL)
        return INFEASIBLE;
    else
    {
        FILE* fp = fopen(FileName, "w");
        for (LNode* p = L->next; p != NULL; p = p->next)
            fprintf(fp, "%d", p->data);
        fclose(fp);
        return OK;
    }
}

status LoadList(LinkList& L, char FileName[])
{
    if (L != NULL)
        return INFEASIBLE;
    else
    {
        int tmp;
        FILE* fp = fopen(FileName, "r");
        L = (LNode*)malloc(sizeof(LNode));
        L->next = NULL;
        LNode* q = L;
```

```
        while (fscanf(fp, "%d", &tmp) == 1)
        {
            LNode* p = (LNode*)malloc(sizeof(LNode));
            p->data = tmp;
            q->next = p;
            p->next = NULL;
            q = p;
        }
        fclose(fp);
        return OK;
    }
}

status AddList(LISTS& Lists, char ListName[])
{
    for (int i = 0; i < Lists.length; i++)
        if (!strcmp(Lists.elem[i].name, ListName))
        {
            printf("??????????????%s?????????\n", ListName);
            return ERROR;
        }
    strcpy(Lists.elem[Lists.length].name, ListName);
    Lists.elem[Lists.length].L = (LNode*)malloc(sizeof(LNode));
    Lists.elem[Lists.length].L->next = NULL;
    Lists.length++;
    return OK;
}

status RemoveList(LISTS& Lists, char ListName[])
{
    int i;
```

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

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

status ListsTraverse(LISTS Lists)
{
    if (Lists.length == 0)
        return ERROR;
    for (int n = 0; n < Lists.length; n++)
    {
        printf("?????%s??????§??", Lists.elem[n].name);
```

```
        ListTraverse(Lists.elem[n].L, visit);
        putchar('\n');
    }
    return OK;
}

LinkList* ChooseList(LinkList* L, LISTS& Lists, int i)
{
    if (i > Lists.length || i < 1)
        return NULL;
    else
    {
        L = &(Lists.elem[i - 1].L);
        return L;
    }
}
```

附录 B 基于邻接表图实现的源程序

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

#define TRUE 1     判定结果为正确
#define FALSE 0    判定结果为错误
#define OK 1       功能运行正常
#define ERROR 0    功能出现错误无法运行
#define INFEASIBLE -1操作对象不存在
#define OVERFLOW -2 内存溢出

#define MAX_VERTEX_NUM 20

typedef int status;
typedef int KeyType;
typedef enum { DG, DN, UDG, UDN } GraphKind; 图类型:有向图,有向网,无向图,无向网

顶点类型
typedef struct {
    KeyType key;      关键字
    char others[20];  内容
} VertexType;

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

```
 头结点及其数组类型
typedef struct VNode {
    VertexType data; 顶点信息
    ArcNode* firstarc; 指向第一条弧
} VNode, AdjList[MAX_VERTEX_NUM];

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

 图的管理表
typedef struct {
    struct {
        char name[30]; 图名称
        ALGraph G;
    }elem[10];
    int length;        管理表长度 (图数目)
    int listsize;      管理表容量
}LISTS;
 循环队列元素类型
typedef int ElemType;

 循环队列容量
#define MAXLENGTH 100

 循环队列结构
typedef struct QUEUE {
    ElemType elem[MAXLENGTH]; 循环队列元素
```

```
    int front, length;        循环队列定位用变量
} QUEUE;
 零件函数
void visit(VertexType v);
void DFS(ALGraph G, int v, void (*visit)(VertexType));

 队列操作函数
void iniQueue(QUEUE& Q);
status enQueue(QUEUE& Q, ElemType e);
status deQueue(QUEUE& Q, ElemType& e);

 单图基础操作函数
status CreateCraph(ALGraph& G, VertexType V[], KeyType VR[][2]);
status DestroyGraph(ALGraph& G);
int LocateVex(ALGraph G, KeyType u);
status PutVex(ALGraph& G, KeyType u, VertexType value);
int FirstAdjVex(ALGraph G, KeyType u);
int NextAdjVex(ALGraph G, KeyType v, KeyType w);
status InsertVex(ALGraph& G, VertexType v);
status DeleteVex(ALGraph& G, KeyType v);
status InsertArc(ALGraph& G, KeyType v, KeyType w);
status DeleteArc(ALGraph& G, KeyType v, KeyType w);
status DFSTraverse(ALGraph& G, void (*visit)(VertexType));
status BFSTraverse(ALGraph& G, void (*visit)(VertexType));

选做操作函数
VNode* VerticesSetLessThanK(ALGraph G, KeyType v, int k);
int ShortestPathLength(ALGraph G, KeyType v, KeyType w);
int ConnectedComponentsNums(ALGraph G);

文件读写函数
```

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

多图操作函数
status AddGragh(LISTS& Lists, char ListName[]);
status RemoveGragh(LISTS& Lists, char ListName[]);
int LocateGragh(LISTS Lists, char ListName[]);
status ListsTraverse(LISTS Lists);
ALGraph* ChooseGragh(ALGraph* L, LISTS& Lists, int i);

main函数
int main(void)
{
    定义单图并初始化
    ALGraph G1;
    ALGraph* G = &G1;
    G->arcnum = 0;
    G->vexnum = 0;
    memset(G->vertices, 0, MAX_VERTEX_NUM * sizeof(VNode));

    定义管理表并初始化
    LISTS Lists;
    Lists.length = 0, Lists.listsize = 10;

    op记录功能选择情况,flag记录功能运行情况,i与k存储int型
过程量,u与v存储关键字过程量
    int op = 1, flag = 0, i = 0, k = 0, u = 0, v = 0;
    char FileName[30], ListName[30]; 储存文件名与表名
    VertexType value; 存储顶点过程量
    VertexType V[30]; 存储顶点序列
    KeyType VR[100][2]; 存储关系对序列
```

```
VNode* VN = NULL; 存储顶点集合过程量，用于main函数中接收函数13的返回值

while (op)
{
    system("cls");
    printf("\n\n");

    printf("请选择你的操作[0~22]:");
    scanf("%d", &op);
    switch (op)
    {
    case 1:创建
        if (G->vexnum)
            printf("创建失败!图已存在!\n");
        else
        {
            i = 0;
            printf("顶点格式示例:␣1␣线性表␣3␣二叉树\n");
            printf("以-1␣nil作为结束标记\n");
            printf("空图不予创建\n");
            printf("请据此输入将建图中的顶点序列:");
            do {
                scanf("%d␣%s", &V[i].key, V[i].others);
            } while (V[i++].key != -1);
            i = 0;
            printf("关系对格式示例:␣1␣3\n");
            printf("以-1␣-1作为结束标记\n");
            printf("请据此输入将建图中的关系对序列:");
            do {
                scanf("%d␣%d", &VR[i][0], &VR[i][1]);
```

```
            } while (VR[i++][0] != -1);
            flag = CreateCraph(*G, V, VR);
            if (flag == OK)
                printf("无向图创建成功!\n这是一个与独立于多图的单图␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣,你可以对它进行1~17操作!\n");
            else
                printf("创建失败!顶点序列或关系对序列输入有误!\n");
        }
        getchar();getchar();
        break;
    case 2:销毁
        flag = DestroyGraph(*G);
        if (flag == OK)
            printf("成功销毁无向图并释放数据元素的空间!\n");
        else if (flag == INFEASIBLE)
            printf("不能对不存在的无向图进行销毁操作!\n");
        getchar();getchar();
        break;
    case 3:breaklines//breaklines查找顶点
        if (!G->vexnum)
            printf("不能对不存在的无向图进行查找操作!\n");
        else
        {
            printf("请输入想要查找的关键字:");
            scanf("%d", &u);
            flag = LocateVex(*G, u);
            if (flag == -1)
                printf("含关键字%d的顶点不存在!\n", u);
            else
                printf("含关键字%d的顶点位序为␣%d!", u, flag);
        }
```

```
            getchar();getchar();
            break;
        case 4:顶点赋值
            if (!G->vexnum)
                printf("不能对不存在的无向图进行赋值操作!\n");
            else
            {
                printf("请输入想要赋值的顶点的关键字用于查找:");
                scanf("%d", &u);
                printf("请输入将赋的关键字与内容:");
                scanf("%d%s", &value.key, value.others);
                flag = PutVex(*G, u, value);
                if (flag == OK)
                    printf("赋值成功!\n");
                else
                    printf("赋值失败!关键字不存在或将赋关键字不唯一
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣!\n");
            }
            getchar();getchar();
            break;
        case 5:获得第一邻接点
            if (!G->vexnum)
                printf("不能对不存在的无向图进行该操作!\n");
            else
            {
                printf("请输入想要求第一邻接点的顶点的关键字:");
                scanf("%d", &u);
                flag = FirstAdjVex(*G, u);
                if (flag == -1)
                    printf("操作失败!关键字不存在或该顶点不存在
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣邻接点!\n");
```

```
                else
                    printf("关键字为%d的顶点的第一邻接点是位序为%d
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣的%d%s!\n", u, flag, G->vertices[flag].da
                    ta.key, G->vertices[flag].data.others);
            }
            getchar();getchar();
            break;
        case 6:获得下一邻接点
            if (!G->vexnum)
                printf("不能对不存在的无向图进行该操作!\n");
            else
            {
                printf("请输入一个顶点和它的一个邻接点的关键字:");
                scanf("%d%d", &u, &v);
                flag = NextAdjVex(*G, u, v);
                if (flag == -1)
                    printf("操作失败!关键字不存在或该邻接点已是最后一
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣个!\n");
                else
                    printf("顶点%d的邻接点␣%d␣的下一邻接点是位序为␣%
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣d␣的␣%d␣%s!\n", u, v, flag, G->vertices[flag].data.key, G->ver
            }
            getchar();getchar();
            break;
        case 7:插入顶点
            if (!G->vexnum)
                printf("不能对不存在的无向图进行该操作!\n");
            else
            {
                printf("请输入将插入的关键字与内容:");
                scanf("%d%s", &value.key, value.others);
```

```
            flag = InsertVex(*G, value);
            if (flag == OK)
                printf("插入成功!\n");
            else
                printf("插入失败!顶点数目已达最大值或关键字已存在!\n");
        }
        getchar();getchar();
        break;
    case 8:删除顶点
        if (!G->vexnum)
            printf("不能对不存在的无向图进行该操作!\n");
        else
        {
            printf("请输入将删除的结点的关键字:");
            scanf("%d", &u);
            flag = DeleteVex(*G, u);
            if (flag == OK)
                printf("删除成功!与该顶点相关的弧已删除!\n");
            else
                printf("删除失败!关键字不存在或顶点数目为1!\n");
        }
        getchar();getchar();
        break;
    case 9:插入弧
        if (!G->vexnum)
            printf("不能对不存在的无向图进行该操作!\n");
        else
        {
            printf("请分别输入将插入的弧的两个顶点的关键字:");
            scanf("%d%d", &u, &v);
            flag = InsertArc(*G, u, v);
```

```
            if (flag == OK)
                printf("插入成功!\n");
            else
                printf("插入失败!关键字不存在或弧已存在!\n");
        }
        getchar();getchar();
        break;
    case 10:删除弧
        if (!G->vexnum)
            printf("不能对不存在的无向图进行该操作!\n");
        else
        {
            printf("请分别输入将删除的弧的两个顶点的关键字:");
            scanf("%d%d", &u, &v);
            flag = DeleteArc(*G, u, v);
            if (flag == OK)
                printf("删除成功!\n");
            else
                printf("删除失败!关键字不存在或弧不存在!\n");
        }
        getchar();getchar();
        break;
    case 11:深搜遍历
        if (!G->vexnum)
            printf("不能对不存在的无向图进行遍历操作!\n");
        else
            DFSTraverse(*G, visit);
        getchar();getchar();
        break;
    case 12:广搜遍历
        if (!G->vexnum)
```

```
            printf("不能对不存在的无向图进行遍历操作!\n");
        else
            BFSTraverse(*G, visit);
        getchar();getchar();
        break;
    case 13:求距离小于k的顶点集合
        if (!G->vexnum)
            printf("不能对不存在的无向图进行该操作!\n");
        else
        {
            printf("请输入一个顶点的关键字:");
            scanf("%d", &u);
            printf("请输入距离k:");
            scanf("%d", &k);
            VN = VerticesSetLessThanK(*G, u, k);
            if (VN == NULL)
                printf("操作失败!关键字输入有误!\n");
            else
            {
                if (VN[0].data.key == 0)
                    printf("集合为空!");
                else
                {
                    printf("与关键字为␣%d␣的顶点距离小于%d
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣的顶点集合为:", u, k);
                    for (i = 0; VN[i].data.key != 0; i++)
                        printf("%d␣%s␣", VN[i].data.key, VN[i].data.others);
                }
            }
        }
        getchar();getchar();
```

```
            break;
        case 14:求顶点间最短路径长度
            if (!G->vexnum)
                printf("不能对不存在的无向图进行该操作!\n");
            else
            {
                printf("请输入两个顶点的关键字:");
                scanf("%d%d", &u, &v);
                flag = ShortestPathLength(*G, u, v);
                if (flag == -1)
                    printf("操作失败!关键字不存在或两顶点间不存在路
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣径!\n");
                else
                    printf("关键字为%d和%d的顶点间最短路径长度为%d
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣!\n", u, v, flag);
            }
            getchar();getchar();
            break;
        case 15:求连通分量个数
            if (!G->vexnum)
                printf("不能对不存在的无向图进行该操作!\n");
            else
            {
                flag = ConnectedComponentsNums(*G);
                printf("\n该图的连通分量个数为%d!\n", flag);
            }
            getchar();getchar();
            break;
        case 16:breaklines//breaklines写入文件
            if (!G->vexnum)
                printf("不能对不存在的无向图进行该操作!\n");
```

```
            else
            {
                printf("请输入文件名称:");
                scanf("%s", FileName);
                flag = SaveGraph(*G, FileName);
                if (flag == ERROR)
                    printf("文件打开失败!\n");
                else
                    printf("无向图的内容已经复制到名称为%s的文
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣件中!\n", FileName);
            }
            getchar();getchar();
            break;
        case 17:读取文件
            if (G->vexnum)
                printf("不能对已存在的无向图进行读文件操作!请先
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣销毁无向图!\n");
            else
            {
                printf("请输入文件名称:");
                scanf("%s", FileName);
                flag = LoadGraph(*G, FileName);
                if (flag == ERROR)
                    printf("文件打开失败!\n");
                else
                    printf("名称为%s的文件中内容已复制到无向图中!\n", FileName);
            }
            getchar();getchar();
            break;
        case 18:breaklines//breaklines增加图
            printf("请输入新增无向图的名称:");
```

```
        scanf("%s", ListName);
        flag = AddGragh(Lists, ListName);
        if (flag == OK)
            printf("成功新增名称为%s的无向图!\n", ListName);
        else
            printf("新增失败!\n");
        getchar();getchar();
        break;
    case 19:breaklines//breaklines删除图
        if (Lists.length)
        {
            for (int i = 0; i < Lists.length; i++)
                printf("%d␣%s\n", i + 1, Lists.elem[i].
                name);
          }
        else
        {
            printf("无向图的集合为空!无法进行此操作!\n");
            getchar();getchar();
            break;
        }
        printf("请输入想要删除的无向图的名称:");
        scanf("%s", ListName);
        flag = RemoveGragh(Lists, ListName);
        if (flag == OK)
            printf("成功删除名称为%s的无向图!\n", ListName);
        else
            printf("删除失败!\n");
        getchar();getchar();
        break;
    case 20:breaklines//breaklines查找图
```

```
        if (!Lists.length)
        {
            printf("无向图的集合为空!无法进行此操作!\n");
            getchar();getchar();
            break;
        }
        printf("请输入想要查找的无向图的名称:");
        scanf("%s", ListName);
        flag = LocateGragh(Lists, ListName);
        if (!flag)
            printf("不存在名称为%s的无向图!\n", ListName);
        else
            printf("名称为%s的无向图的序号为␣%d!\n",
            ListName, flag);
        getchar();getchar();
        break;
    case 21:遍历总表
        if (ListsTraverse(Lists) == ERROR)
            printf("无向图的集合为空!\n");
        getchar();getchar();
        break;
    case 22:选择单图进行操作
        if (Lists.length)
        {
             for (int i = 0; i < Lists.length; i++)
                printf("%d%s\n", i + 1, Lists.elem[i].name);
        }
        else
        {
            printf("无向图的集合为空!无法进行此操作!\n");
            getchar();getchar();
```

```
            break;
        }
        printf("请输入想要操作的无向图的序号(从1开始):");
        scanf("%d", &flag);
        G = ChooseGragh(G, Lists, flag);
        if (G == NULL)
            printf("输入序号不合法!单图已置空!\n");
        else
            printf("下面可对无向图集合中序号为%d的无向图
进行操作!\n", flag);
        getchar();getchar();
        break;
    case 0:
        break;
    }
  }
  printf("欢迎下次再使用本系统! \n");
  return 0;
}
```

main函数

```
int visited[MAX_VERTEX_NUM]; 遍历时记录顶点是否已访问
int PathLength[MAX_VERTEX_NUM]; 记录所有顶点到某一顶点的最短路径长度
VNode vn[MAX_VERTEX_NUM];   存储顶点集合过程量,用于函数13中
```

功能5 获得第一邻接点

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

```
int FirstAdjVex(ALGraph G, KeyType u)
{
  int i;
  for (i = 0; i < G.vexnum; i++)
    if (u == G.vertices[i].data.key && G.vertices[i].
```

```
firstarc != NULL)
            return G.vertices[i].firstarc->adjvex;
    return -1;
}

 功能6 获得下一邻接点
 v对应G的一个顶点,w对应v的邻接顶点
 操作结果是返回v的(相对于w)下一个邻接顶点的位序
 如果w是最后一个邻接顶点,或v,w对应顶点不存在,则返回-1
int NextAdjVex(ALGraph G, KeyType v, KeyType w)
{
    int i, flag1 = -1, flag2 = -1;
    for (i = 0; i < G.vexnum; i++)
    {
        if (v == G.vertices[i].data.key)
            flag1 = i;
        if (w == G.vertices[i].data.key)
            flag2 = i;
    }
    if (flag1 == -1 || flag2 == -1)
        return -1;
    ArcNode* a = G.vertices[flag1].firstarc;
    while (a)
    {
        if (flag2 == a->adjvex && a->nextarc != NULL)
            return a->nextarc->adjvex;
        a = a->nextarc;
    }
    return -1;
}
```

```
 访问函数
void visit(VertexType v)
{
    printf("␣%d␣%s", v.key, v.others);
}

 深搜实现函数
void DFS(ALGraph G, int v, void (*visit)(VertexType))
{
    int w = -1;
    visited[v] = 1;
    visit(G.vertices[v].data);
    for (w = FirstAdjVex(G, G.vertices[v].data.key);
w >= 0; w = NextAdjVex(G, G.vertices[v].data.key, G.vertices[w].data.key))
        if (!visited[w])
            DFS(G, w, visit);
}




 初始化队列Q
void iniQueue(QUEUE& Q)
{
    Q.front = 0;
    Q.length = 0; breaklines//breaklines队列长度为breaklines0
}

 元素入队
status enQueue(QUEUE& Q, ElemType e)
{
```

```
    if (Q.length >= MAXLENGTH) 队列溢出
        return ERROR;
    Q.elem[(Q.front + Q.length++) % MAXLENGTH] = e;
    return OK;
}

 元素出队
 将Q队首元素出队，赋值给e。成功出队返回1，否则返回0
status deQueue(QUEUE& Q, ElemType& e)
{
    if (Q.length == 0) breaklines//breaklines队列为空
        return ERROR;
    e = Q.elem[Q.front];
    Q.front = (Q.front + 1) % MAXLENGTH;
    Q.length--;
    return OK;
}



功能1 创建
 根据V和VR构造图T并返回OK,如果V和VR不正确,返回ERROR,如果有相同
的关键字,返回ERROR
status CreateCraph(ALGraph& G, VertexType V[], KeyType
VR[][2])
{
    if (V[0].key == -1)
        return ERROR;
    int a, i, j, flag1 = 0, flag2 = 0;
    KeyType keys[30];
    for (a = 0; a < 20; a++)
```

```
    keys[a] = 0;
a = 0;
for (i = 0; V[i].key != -1; i++)
{
    if (i >= MAX_VERTEX_NUM)
        return ERROR;
    keys[a++] = V[i].key;
    for (j = i + 1; V[j].key != -1; j++)
        if (V[i].key == V[j].key)
            return ERROR;
}
for (i = 0; VR[i][0] != -1; i++)
{
    flag1 = 0, flag2 = 0;
    for (a = 0; keys[a] != 0; a++)
    {
        if (VR[i][0] == keys[a])
            flag1 = 1;
        if (VR[i][1] == keys[a])
            flag2 = 1;
    }
    if (!flag1 || !flag2)
        return ERROR;
}
G.kind = UDG;
G.vexnum = 0;
G.arcnum = 0;
for (i = 0; V[i].key != -1 && i < MAX_VERTEX_NUM; i++)
{
    G.vertices[i].data = V[i];
    G.vertices[i].firstarc = NULL;
```

```
        G.vexnum++;
    }
    for (i = 0; VR[i][0] != -1; i++)
    {
        flag1 = 0, flag2 = 0;
        for (a = 0; keys[a] != 0; a++)
        {
            if (VR[i][0] == keys[a])
                flag1 = a;
            if (VR[i][1] == keys[a])
                flag2 = a;
        }
        ArcNode* a1 = (ArcNode*)malloc(sizeof(ArcNode));
        a1->adjvex = flag2;
        a1->nextarc = G.vertices[flag1].firstarc;
        G.vertices[flag1].firstarc = a1;
        ArcNode* a2 = (ArcNode*)malloc(sizeof(ArcNode));
        a2->adjvex = flag1;
        a2->nextarc = G.vertices[flag2].firstarc;
        G.vertices[flag2].firstarc = a2;
        G.arcnum++;
    }
    return OK;
}

 功能2 销毁
 销毁无向图G,删除G的全部顶点和边
status DestroyGraph(ALGraph& G)
{
    if (!G.vexnum)
        return INFEASIBLE;
```

```
    int i;
    ArcNode* a = NULL, * pre = NULL;
    for (i = 0; i < G.vexnum; i++)
    {
        a = G.vertices[i].firstarc;
        while (a)
        {
            pre = a;
            a = a->nextarc;
            free(pre);
        }
        G.vertices[i].firstarc = NULL;
    }
    memset(G.vertices, 0, G.vexnum);
    G.arcnum = 0;
    G.vexnum = 0;
    return OK;
}

 功能3 查找顶点
 根据u在图G中查找顶点,查找成功返回位序,否则返回-1
int LocateVex(ALGraph G, KeyType u)
{
    int i;
    for (i = 0; i < G.vexnum; i++)
        if (u == G.vertices[i].data.key)
            return i;
    return -1;
}

 功能4 顶点赋值
```

```
 根据u在图G中查找顶点,查找成功将该顶点值修改成value,返回OK
 如果查找失败或关键字不唯一,返回ERROR
status PutVex(ALGraph& G, KeyType u, VertexType value)
{
    int i, j;
    for (i = 0; i < G.vexnum; i++)
        if (u == G.vertices[i].data.key)
            break;
    if (i == G.vexnum)
        return ERROR;
    for (j = 0; j < G.vexnum; j++)
        if (value.key == G.vertices[j].data.key && value.key != u)
            return ERROR;
    G.vertices[i].data = value;
    return OK;
}



 功能7 插入顶点
 在图G中插入顶点v,成功返回OK,否则返回ERROR
status InsertVex(ALGraph& G, VertexType v)
{
    if (G.vexnum == MAX_VERTEX_NUM)
        return ERROR;
    int i;
    for (i = 0; i < G.vexnum; i++)
        if (v.key == G.vertices[i].data.key)
            return ERROR;
    G.vertices[G.vexnum].data = v;
    G.vertices[G.vexnum].firstarc = NULL;
```

```
    G.vexnum++;
    return OK;
}

 功能8 删除顶点
 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
status DeleteVex(ALGraph& G, KeyType v)
{
    if (G.vexnum == 1)
        return ERROR;
    int i, j = 0, k;
    for (i = 0; i < G.vexnum; i++)
        if (v == G.vertices[i].data.key)
            break;
    if (i == G.vexnum)
        return ERROR;
    ArcNode* a = G.vertices[i].firstarc, * pre = NULL;
    while (a)
    {
        pre = a;
        a = a->nextarc;
        ArcNode* b = G.vertices[pre->adjvex].firstarc, * preb = NULL;
        if (b->adjvex == i)
        {
            G.vertices[pre->adjvex].firstarc = b->nextarc;
            free(b);
        }
        else
            while (b)
            {
                preb = b;
```

```
            b = b->nextarc;
            if (b->adjvex == i)
            {
                preb->nextarc = b->nextarc;
                free(b);
                break;
            }
        }
    free(pre);
    G.arcnum--;
}
G.vertices[i].firstarc = NULL;
ArcNode* c = NULL;
for (j = 0; j < G.vexnum; j++)
{
    c = G.vertices[j].firstarc;
    while (c != NULL)
    {
        if (c->adjvex > i)
            c->adjvex--;
        c = c->nextarc;
    }
}
for (k = i + 1; k < G.vexnum; i++, k++)
{
    G.vertices[i].data = G.vertices[k].data;
    G.vertices[i].firstarc = G.vertices[k].firstarc;
}
G.vexnum--;
return OK;
}
```

```
 功能9 插入弧
 在图G中增加弧<v,w>,成功返回OK,否则返回ERROR
status InsertArc(ALGraph& G, KeyType v, KeyType w)
{
    int i, flag1 = -1, flag2 = -1;
    for (i = 0; i < G.vexnum; i++)
    {
        if (v == G.vertices[i].data.key)
            flag1 = i;
        if (w == G.vertices[i].data.key)
            flag2 = i;
    }
    if (flag1 == -1 || flag2 == -1)
        return ERROR;

    ArcNode* a = G.vertices[flag1].firstarc;
    while (a)
    {
        if (a->adjvex == flag2)
            return ERROR;
        a = a->nextarc;
    }

    ArcNode* a1 = (ArcNode*)malloc(sizeof(ArcNode));
    a1->adjvex = flag2;
    a1->nextarc = G.vertices[flag1].firstarc;
    G.vertices[flag1].firstarc = a1;
    ArcNode* a2 = (ArcNode*)malloc(sizeof(ArcNode));
    a2->adjvex = flag1;
    a2->nextarc = G.vertices[flag2].firstarc;
```

```
    G.vertices[flag2].firstarc = a2;
    G.arcnum++;
    return OK;
}

 功能10 删除弧
 在图G中删除弧<v,w>,成功返回OK,否则返回ERROR
status DeleteArc(ALGraph& G, KeyType v, KeyType w)
{
    int i, flag1 = -1, flag2 = -1, flag = 0;
    for (i = 0; i < G.vexnum; i++)
    {
        if (v == G.vertices[i].data.key)
            flag1 = i;
        if (w == G.vertices[i].data.key)
            flag2 = i;
    }
    if (flag1 == -1 || flag2 == -1)
        return ERROR;

    ArcNode* a = G.vertices[flag1].firstarc;
    while (a)
    {
        if (a->adjvex == flag2)
            flag = 1;
        a = a->nextarc;
    }
    if (!flag)
        return ERROR;

    ArcNode* a1 = G.vertices[flag1].firstarc, * pre1 = NULL;
```

```
if (a1->adjvex == flag2)
{
    G.vertices[flag1].firstarc = a1->nextarc;
    free(a1);
}
else
    while (a1)
    {
        pre1 = a1;
        a1 = a1->nextarc;
        if (a1->adjvex == flag2)
        {
            pre1->nextarc = a1->nextarc;
            free(a1);
            break;
        }
    }

ArcNode* a2 = G.vertices[flag2].firstarc, * pre2 = NULL;
if (a2->adjvex == flag1)
{
    G.vertices[flag2].firstarc = a2->nextarc;
    free(a2);
}
else
    while (a2)
    {
        pre2 = a2;
        a2 = a2->nextarc;
        if (a2->adjvex == flag1)
        {
```

```
                pre2->nextarc = a2->nextarc;
                free(a2);
                break;
            }
        }
    G.arcnum--;
    return OK;
}
```

功能11 深搜遍历

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

```
status DFSTraverse(ALGraph& G, void (*visit)(VertexType))
{
    int v;
    for (v = 0; v < G.vexnum; v++)
        visited[v] = 0;
    for (v = 0; v < G.vexnum; v++)
        if (!visited[v])
            DFS(G, v, visit);
    return OK;
}
```

功能12 广搜遍历

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

```
status BFSTraverse(ALGraph& G, void (*visit)(VertexType))
{
    int v, u, w = -1;
    QUEUE Q;
    for (v = 0; v < G.vexnum; v++)
```

```
        visited[v] = 0;
    iniQueue(Q);
    for (v = 0; v < G.vexnum; v++)
        if (!visited[v])
        {
            visited[v] = 1;
            visit(G.vertices[v].data);
            enQueue(Q, v);
            while (Q.length)
            {
                deQueue(Q, u);
                for (w = FirstAdjVex(G, G.vertices[u].
data.key); w >= 0; w = NextAdjVex(G, G.vertices[u].data.key, G.vertices[w].data.ke
                    if (!visited[w])
                    {
                        visited[w] = 1;
                        visit(G.vertices[w].data);
                        enQueue(Q, w);
                    }
            }
        }
    return OK;
}

 附加功能13 求距离小于k的顶点集合
 初始条件是图G存在;操作结果是返回与顶点v距离小于k的顶点集合
VNode* VerticesSetLessThanK(ALGraph G, KeyType v, int k)
{
    int i, j, u, x = -1, len = 0, times = 0;
    int flag1 = -1, flag2 = -1;
    for (i = 0; i < MAX_VERTEX_NUM; i++)
```

```
        vn[i].data.key = 0;
    for (i = 0; i < G.vexnum; i++)
        if (v == G.vertices[i].data.key)
            flag1 = i;
    if (flag1 == -1)
        return NULL;
    int length[MAX_VERTEX_NUM];
    for (i = 0; i < MAX_VERTEX_NUM; i++)
        length[i] = 0;
    QUEUE Q;
    for (i = 0; i < G.vexnum; i++)
        visited[i] = 0;
    iniQueue(Q);
    for (i = flag1; times < G.vexnum; i++, times++)
    {
        if (!visited[i])
        {
            visited[i] = 1;
            length[i] = len;
            enQueue(Q, i);
            while (Q.length)
            {
                len++;
                deQueue(Q, u);
                for (x = FirstAdjVex(G, G.vertices[u].data
.key); x >= 0; x = NextAdjVex(G, G.vertices[u].data.key,
G.vertices[x].data.key))
                {
                    if (!visited[x])
                    {
                        visited[x] = 1;
```

```
                    length[x] = len;
                    enQueue(Q, x);
                }
            }
        }
        if (i + 1 == G.vexnum)
            i = -1;
        break;
    }
    for (i = 0, j = 0; i < G.vexnum; i++)
        if (length[i] > 0 && length[i] < k)
            vn[j++] = G.vertices[i];
    return vn;
}
```

附加功能14 求顶点间最短路径长度

初始条件是图G存在;操作结果是返回顶点v与顶点w的最短路径的长度

```
int ShortestPathLength(ALGraph G, KeyType v, KeyType w)
{
    int i, u, x = -1, len = 0, times = 0;
    int flag1 = -1, flag2 = -1;
    for (i = 0; i < G.vexnum; i++)
    {
        if (v == G.vertices[i].data.key)
            flag1 = i;
        if (w == G.vertices[i].data.key)
            flag2 = i;
    }
    if (flag1 == -1 || flag2 == -1)
        return -1;
```

```
    int length[MAX_VERTEX_NUM];
    for (i = 0; i < MAX_VERTEX_NUM; i++)
        length[i] = -1;
    QUEUE Q;
    for (i = 0; i < G.vexnum; i++)
        visited[i] = 0;
    iniQueue(Q);
    for (i = flag1; times < G.vexnum; i++, times++)
    {
        if (!visited[i])
        {
            visited[i] = 1;
            length[i] = len;
            enQueue(Q, i);
            while (Q.length)
            {
                len++;
                deQueue(Q, u);
                for (x = FirstAdjVex(G, G.vertices[u].
data.key);x >= 0; x = NextAdjVex(G, G.vertices[u].data.
key, G.vertices[x].data.key))
                {
                    if (!visited[x])
                    {
                        visited[x] = 1;
                        length[x] = len;
                        enQueue(Q, x);
                    }
                }
            }
        }
```

```
        if (i + 1 == G.vexnum)
            i = -1;
        break;
    }
    return length[flag2];
}

 附加功能15 求图连通分量个数
初始条件是图G存在;操作结果是返回图G的所有连通分量的个数
int ConnectedComponentsNums(ALGraph G)
{
    int i, nums = 0;
    for (i = 0; i < G.vexnum; i++)
        visited[i] = 0;
    for (i = 0; i < G.vexnum; i++)
        if (!visited[i])
        {
            nums++;
            DFS(G, i, visit);
        }
    return nums;
}

 附加功能16 写入文件
 将图的数据写入到文件FileName中
status SaveGraph(ALGraph G, char FileName[])
{
    FILE* fp = fopen(FileName, "w");

    for (int i = 0; i < G.vexnum; i++)
        fprintf(fp, "%d␣%s␣", G.vertices[i].data.key
```

```
,G.vertices[i].data.others);
    fprintf(fp, "%d␣%s␣", -1, "nil");
    int a1[30], a2[30];
    for (int i = 0; i < 30; i++)
        a1[i] = 0, a2[i] = 0;
    for (int i = 0; i < G.vexnum; i++)
    {
        KeyType k1 = G.vertices[i].data.key, k2 = -1;
        ArcNode* a = G.vertices[i].firstarc;
        int j = 0, flag = 0;
        while (a)
        {
            flag = 0;
            k2 = G.vertices[a->adjvex].data.key;
            a = a->nextarc;
            for (j = 0; a1[j] != 0; j++)
                if (k1 == a2[j] && k2 == a1[j])
                    flag = 1;
            if (flag)
                continue;
            else
            {
                a1[j] = k1;
                a2[j] = k2;
            }
        }
    }
    for (int i = 0; a1[i] != 0; i++)
        if (a2[i] < a1[i])
        {
            int tmp = a2[i];
```

```
            a2[i] = a1[i];
            a1[i] = tmp;
        }
    for (int i = 0; a1[i] != 0; i++)
    {
        for (int j = i + 1; a1[j] != 0; j++)
        {
            if ((a1[i] > a1[j]) || (a1[i] == a1[j] && a
2[i] > a2[j]))
            {
                int tmp1 = a1[i];
                a1[i] = a1[j];
                a1[j] = tmp1;
                int tmp2 = a2[i];
                a2[i] = a2[j];
                a2[j] = tmp2;
            }
        }
    }
    for (int i = 0; a1[i] != 0; i++)
        fprintf(fp, "%d␣%d␣", a1[i], a2[i]);
    fprintf(fp, "%d␣%d␣", -1, -1);
    fclose(fp);
    return OK;
}

 功能17 读取文件
 读入文件FileName的图数据，创建图的邻接表
status LoadGraph(ALGraph& G, char FileName[])
{
    FILE* fp = fopen(FileName, "r");
```

```
G.kind = UDG;
G.vexnum = 0;
G.arcnum = 0;
VertexType V[21];
KeyType VR[100][2], keys[30];
for (int k = 0; k < 30; k++)
    keys[k] = 0;
int i = 0, flag1 = -1, flag2 = -1;
do {
    fscanf(fp, "%d%s␣", &V[i].key, V[i].others);
    keys[i] = V[i].key;
} while (V[i++].key != -1);
i = 0;
do {
    fscanf(fp, "%d%d␣", &VR[i][0], &VR[i][1]);
} while (VR[i++][0] != -1);

for (i = 0; V[i].key != -1 && i < MAX_VERTEX_NUM; i++)
{
    G.vertices[i].data = V[i];
    G.vertices[i].firstarc = NULL;
    G.vexnum++;
}
for (i = 0; VR[i][0] != -1; i++)
{
    flag1 = 0, flag2 = 0;
    for (int a = 0; keys[a] != 0; a++)
    {
        if (VR[i][0] == keys[a])
            flag1 = a;
        if (VR[i][1] == keys[a])
```

```
                flag2 = a;
        }
        ArcNode* a1 = (ArcNode*)malloc(sizeof(ArcNode));
        a1->adjvex = flag2;
        a1->nextarc = G.vertices[flag1].firstarc;
        G.vertices[flag1].firstarc = a1;
        ArcNode* a2 = (ArcNode*)malloc(sizeof(ArcNode));
        a2->adjvex = flag1;
        a2->nextarc = G.vertices[flag2].firstarc;
        G.vertices[flag2].firstarc = a2;
        G.arcnum++;
    }
    fclose(fp);
    return OK;
}

 附加功能18 多无向图管理:增加新图
 在Lists中增加一个名称为GraphName的无向图
status AddGragh(LISTS& Lists, char ListName[])
{
    for (int i = 0; i < Lists.length; i++)
        if (!strcmp(Lists.elem[i].name, ListName))
        {
            printf("集合中已存在名称为%s的无向图!\n", ListName);
            return ERROR;
        }
    strcpy(Lists.elem[Lists.length].name, ListName);
    VertexType V1[30];
    KeyType VR1[100][2];
    int m = 0, flag1 = 0;
    printf("顶点格式示例:␣1␣线性表␣␣3␣二叉树\n");
```

```
    printf("以-1␣nil作为结束标记\n");
    printf("空图不予创建\n");
    printf("请据此输入将建图中的顶点序列:");
    do {
        scanf("%d%s", &V1[m].key, V1[m].others);
    } while (V1[m++].key != -1);
    m = 0;
    printf("关系对格式示例:␣1␣3\n");
    printf("以-1␣-1作为结束标记\n");
    printf("请据此输入将建图中的关系对序列:");
    do {
        scanf("%d%d", &VR1[m][0], &VR1[m][1]);
    } while (VR1[m++][0] != -1);
    flag1 = CreateCraph(Lists.elem[Lists.length].G, V1, VR1);
    if (flag1 == OK)
        printf("名称为␣%s␣的无向图创建成功!\n",
Lists.elem[Lists.length++].name);
    else
        printf("创建失败!顶点序列或关系对序列输入有误!\n");
    return OK;
}

 附加功能19 多无向图管理:删除图
status RemoveGragh(LISTS& Lists, char ListName[])
{
    int i;
    for (i = 0; i < Lists.length; i++)
        if (!strcmp(Lists.elem[i].name, ListName))
            break;
    if (i == Lists.length)
        return ERROR;
```

```
    else
    {
        DestroyGraph(Lists.elem[i].G);
        memset(Lists.elem[i].name, 0, strlen(Lists.elem[i].name));
        Lists.length--;
        for (int j = i; j < Lists.length; j++)
            Lists.elem[j] = Lists.elem[j + 1];
        return OK;
    }
}

 附加功能20 多无向图管理:查找图
int LocateGragh(LISTS Lists, char ListName[])
{
    for (int i = 0; i < Lists.length; i++)
        if (!strcmp(Lists.elem[i].name, ListName))
            return (i + 1);
    return ERROR;
}

 附加功能21 多无向图管理:遍历总表
status ListsTraverse(LISTS Lists)
{
    if (Lists.length == 0)
        return ERROR;
    for (int n = 0; n < Lists.length; n++)
    {
        printf("表名为%s的无向图的邻接表按深搜遍历得:\n", Lists.
elem[n].name);
        DFSTraverse(Lists.elem[n].G, visit);
    }
```

```
    return OK;
}

 附加功能22 多无向图管理:选择单图进行操作
ALGraph* ChooseGragh(ALGraph* G, LISTS& Lists, int i)
{
    if (i > Lists.length || i < 1)
        return NULL;
    else
    {
        G = &(Lists.elem[i - 1].G); 传递地址以同时改动单图与多图中的此图
        return G;
    }
}
```

附录 C 基于顺序存储结构的线性表实现

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

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

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

#define LIST_INIT_SIZE 100
#define LISTINCREMENT 10
#define MAX_NAME_LENGTH 20

typedef struct { 顺序表 (顺序结构) 的定义
    ElemType* elem;
    int length;
    int listsize;
    char name[MAX_NAME_LENGTH];
} SqList;

#define MAX_LISTS 100
typedef struct {
    SqList lists[MAX_LISTS];
```

```
    int count;
    int current;

} MultiListManager;
status InitList(SqList* L, const char* name);
status DestroyList(SqList* L);
status ClearList(SqList* L);
status ListEmpty(SqList L);
int ListLength(SqList L);
status GetElem(SqList L, int i, ElemType* e);
int LocateElem(SqList L, ElemType e); breaklines//breaklines简化过
status PriorElem(SqList L, ElemType e, ElemType* pre);
status NextElem(SqList L, ElemType e, ElemType* next);
status ListInsert(SqList* L, int i, ElemType e);
status ListDelete(SqList* L, int i, ElemType* e);
status ListTrabverse(SqList L); breaklines//breaklines简化过


int MaxSubArray(SqList L);
int SubArrayNum(SqList L, int k);
status sortList(SqList* L);
status saveListToFile(SqList L, const char* filename);
status loadListFromFile(SqList* L, const char* filename);
status initMultiListManager(MultiListManager* manager);
status addNewList(MultiListManager* manager, const char*
name);
status removeList(MultiListManager* manager, int index);
status switchToListByName(MultiListManager* manager, const
char* na
me);
int findListIndexByName(MultiListManager* manager, const char
```

```
* name
);
void printListNames(MultiListManager* manager);
void searchList(MultiListManager* manager);
breaklines/*breaklines----------------------------------------breaklines*/
void main(void) {
    MultiListManager manager;
    initMultiListManager(&manager);
    int op = 1,i;
    ElemType e, pre, next;
    int k, index;
    char filename[100];
    char listName[MAX_NAME_LENGTH];

    while (op) {
        system("cls"); printf("\n\n");
        if (manager.current >= 0 && manager.current < manager
.count) {
            printf("当前表:␣%s,␣表的个数有:␣%d\n", manager.lists[manager.current].na
        }
        else {
            printf("当前没有选中的表,␣表的个数有:␣%d\n", manager.
count);
        }
        printf("␣␣␣␣请选择你的操作[0~18]:");
        scanf("%d", &op);

        SqList* currentList = NULL;
        if (manager.current >= 0 && manager.current < manager.
count) {
            currentList = &manager.lists[manager.current];
```

```
    }

    switch (op) {
    case 1:
        printf("请输入线性表的名称:␣");
        scanf("%s", listName);
        if (addNewList(&manager, listName) == OK) {
            printf("线性表创建成功! \n");
        }
        else {
            printf("线性表创建失败! \n");
        }
        getchar();getchar();
        break;
    case 2:
        if (manager.count == 0) {
            printf("没有线性表可销毁! \n");
        }
        else {
            if (removeList(&manager, manager.current) == OK) {
                printf("线性表销毁成功! \n");
            }
            else {
                printf("线性表销毁失败! \n");
            }
        }
        getchar();getchar();
        break;
    case 3:
        if (currentList == NULL) {
            printf("当前没有选中的表! \n");
```

```
        }
        else {
            if (ClearList(currentList) == OK) {
                printf("线性表所有元素删除成功! \n");
            }
            else {
                printf("线性表所有元素删除失败! \n");
            }
        }
        getchar();getchar();
        break;
    case 4:
        if (currentList == NULL) {
            printf("当前没有选中的表! \n");
        }
        else {
            if (ListEmpty(*currentList) == TRUE) {
                printf("线性表为空! \n");
            }
            else {
                printf("线性表不为空! \n");
            }
        }
        getchar();getchar();
        break;
    case 5:
        if (currentList == NULL) {
            printf("当前没有选中的表! \n");
        }
        else {
            if (ListLength(*currentList) > 0) {
```

```
            printf("线性表长度为%d\n", ListLength(*currentList)
            );
        }
        else {
            printf("线性表为空\n");
        }
    }
    getchar();getchar();
    break;
case 6:
    if (currentList == NULL) {
        printf("当前没有选中的表! \n");
    }
    else {
        printf("请输入你要获取的线性表的第几个元素: ");
        int i;
        scanf("%d", &i);
        if (GetElem(*currentList, i, &e) == OK) {
            printf("\n线性表第%d个元素为%d\n", i, e);
        }
        else if (GetElem(*currentList, i, &e) == ERROR) {
            printf("\n线性表长度小于%d\n", i);
        }
        else {
            printf("\n线性表不存在");
        }
    }
    getchar();getchar();
    break;
case 7:
    if (currentList == NULL) {
```

```
            printf("当前没有选中的表! \n");
        }
        else {
            printf("请输入你想查找哪一个元素的位置: ");
            scanf("%d", &e);
            if (LocateElem(*currentList, e) > 0) {
                printf("\n元素%d在线性表的第%d个位置\n", e,
LocateElem(*currentList, e));
            }
            else if (LocateElem(*currentList, e) == 0) {
                printf("该元素不存在\n");
            }
            else {
                printf("线性表不存在");
            }
        }
        getchar();getchar();
        break;
    case 8:
        if (currentList == NULL) {
            printf("当前没有选中的表! \n");
        }
        else {
            printf("请输入你想查找哪一个元素的前驱: ");
            scanf("%d", &e);
            if (PriorElem(*currentList, e, &pre) == OK) {
                printf("\n元素%d在线性表的前驱是%d\n", e, pre);
            }
            else if (PriorElem(*currentList, e, &pre) == ERROR) {
                printf("没有前驱\n");
            }
```

```
            else {
                printf("线性表不存在");
            }
        }
        getchar();getchar();
        break;
    case 9:
        if (currentList == NULL) {
            printf("当前没有选中的表! \n");
        }
        else {
            printf("请输入你想查找哪一个元素的后继: ");
            scanf("%d", &e);
            if (NextElem(*currentList, e, &next) == OK) {
                printf("\n元素%d在线性表的后继是%d\n", e, next);
            }
            else if (NextElem(*currentList, e, &next) == ERROR) {
                printf("没有后继\n");
            }
            else {
                printf("线性表不存在");
            }
        }
        getchar();getchar();
        break;
    case 10:
        if (currentList == NULL) {
            printf("当前没有选中的表! \n");
        }
        else {
            printf("请输入元素e的值: ");
```

```
                scanf("%d", &e);
                printf("\n请输入你想把这个元素插入到线性表的第几个元
素之前: ");
                scanf("%d", &i);
                if (ListInsert(currentList, i, e) == OK) {
                    printf("\n插入成功! \n");
                }
                else if (ListInsert(currentList, i, e) == ERROR) {
                    printf("插入位置不正确\n");
                }
                else {
                    printf("线性表不存在");
                }
            }
            getchar();getchar();
            break;
        case 11:
            if (currentList == NULL) {
                printf("当前没有选中的表! \n");
            }
            else {
                printf("\n请输入你想删除线性表的第几个元素: ");
                scanf("%d", &i);
                if (ListDelete(currentList, i, &e) == OK) {
                    printf("\n删除成功! \n");
                }
                else if (ListDelete(currentList, i, &e) == ERROR) {
                    printf("删除位置不正确\n");
                }
                else {
                    printf("线性表不存在");
```

```
            }
        }
        getchar();getchar();
        break;
    case 12:
        if (currentList == NULL) {
            printf("当前没有选中的表! \n");
        }
        else {
            if (!ListTrabverse(*currentList)) {
                printf("线性表是空表! \n");
            }
        }
        getchar();getchar();
        break;
    case 13:
        if (currentList == NULL) {
            printf("当前没有选中的表! \n");
        }
        else if (ListLength(*currentList) <= 0) {
            printf("线性表不存在或为空! \n");
        }
        else {
            printf("最大连续子数组和为: %d\n",
MaxSubArray(*currentList));
        }
        getchar();getchar();
        break;
    case 14:
        if (currentList == NULL) {
            printf("当前没有选中的表! \n");
```

```
            }
            else if (ListLength(*currentList) <= 0) {
                printf("线性表不存在或为空! \n");
            }
            else {
                printf("请输入k的值: ");
                scanf("%d", &k);
                printf("和为%d的子数组个数为: %d\n", k,
SubArrayNum(*currentList, k));
            }
            getchar();getchar();
            break;
        case 15:
            if (currentList == NULL) {
                printf("当前没有选中的表! \n");
            }
            else {
                if (sortList(currentList) == OK) {
                    printf("线性表排序成功! \n");
                }
                else {
                    printf("线性表排序失败! \n");
                }
            }
            getchar();getchar();
            break;
        case 16:
            if (currentList == NULL) {
                printf("当前没有选中的表! \n");
            }
            else {
```

```
                printf("请输入文件名: ");
                scanf("%s", filename);
                if (saveListToFile(*currentList, filename)
== OK) {
                    printf("线性表保存成功! \n");
                }
                else {
                    printf("线性表保存失败! \n");
                }
            }
            getchar();getchar();
            break;
        case 17:
            if (currentList == NULL) {
                printf("当前没有选中的表! \n");
            }
            else {
                printf("请输入文件名: ");
                scanf("%s", filename);
                if (loadListFromFile(currentList, filename) == OK) {
                    printf("线性表加载成功! \n");
                }
                else {
                    printf("线性表加载失败! \n");
                }
            }
            getchar();getchar();
            break;
        case 18:
            while (op) {
                system("cls");
```

```
            printf("当前表的个数有:%d\n", manager.count);
            if (manager.current >= 0 && manager.current < manager.count) {
                printf("当前表:␣%s\n", manager.lists[manager.current].name);
            }
            printf("␣␣␣␣请选择你的操作[0~4]:");
            scanf("%d", &op);
            switch (op) {
            case 1:
                printf("请输入新线性表的名称:␣");
                scanf("%s", listName);
                if (addNewList(&manager, listName) == OK) {
                    printf("新线性表创建成功! 当前共有%d个线性表\n",
manager.count);
                }
                else {
                    printf("创建新线性表失败! \n");
                }
                getchar();getchar();
                break;
            case 2:
                printListNames(&manager);
                printf("请输入要删除的线性表名称: ");
                scanf("%s", listName);
                index = findListIndexByName(&manager,
listName);
          if (index != -1) {
                if (removeList(&manager, index) == OK) {
                    printf("线性表删除成功! 当前共有%d个线性表
\n", manager.count);
                    }
                    else {
```

```
                printf("删除线性表失败! \n");
            }
        }
        else {
            printf("找不到名为'%s'的线性表! \n", listName);
        }
        getchar();getchar();
        break;
    case 3:
        printListNames(&manager);
        printf("请输入要切换到的线性表名称: ");
        scanf("%s", listName);
        if (switchToListByName(&manager, listName) == OK)
        {
            printf("已切换到线性表'%s'! \n", listName);
        }
        else {
            printf("切换线性表失败! 找不到名为'%s'的线性表\n", listName);
        }
        getchar();getchar();
        break;
    case 4:
        printListNames(&manager);
        getchar();getchar();
        break;
    case 5:
        searchList(&manager);
        getchar();getchar();
        break;
    case 0:
        break;
```

```
                }
            }
            op = 1;
            break;
        case 0:
            break;
        }
    }
    printf("欢迎下次再使用本系统! \n");
}
status InitList(SqList* L, const char* name) {
    L->elem = (ElemType*)malloc(LIST_INIT_SIZE * sizeof(ElemType));
    if (!L->elem) exit(OVERFLOW);
    L->length = 0;
    L->listsize = LIST_INIT_SIZE;
    strncpy(L->name, name, MAX_NAME_LENGTH - 1);
    L->name[MAX_NAME_LENGTH - 1] = '\0';
    return OK;
}



status DestroyList(SqList* L) {
    if (L->elem == NULL)
        return INFEASTABLE;
    else {
        free(L->elem);
        L->elem = NULL;
        L->length = 0;
        L->listsize = 0;
        memset(L->name, 0, MAX_NAME_LENGTH);
        return OK;
```

```
    }
}
status ClearList(SqList* L) {
    if (L->elem == NULL)
        return INFEASTABLE;
    else {
        L->length = 0;
        return OK;
    }
}

status ListEmpty(SqList L) {
    if (L.elem == NULL)
        return INFEASTABLE;
    if (L.length == 0)
        return TRUE;
    else return FALSE;
}

int ListLength(SqList L) {
    if (L.elem == NULL)
        return INFEASTABLE;
    else {
        return L.length;
    }
}

status GetElem(SqList L, int i, ElemType* e) {
    if (L.elem == NULL)
        return INFEASTABLE;
    if (i > L.length || i <= 0)
```

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

int LocateElem(SqList L, ElemType e) {
    if (L.elem == NULL)
        return INFEASTABLE;
    for (int i = 0; i < L.length; i++) {
        if (L.elem[i] == e) {
            return i + 1;
        }
    }
    return 0;
}

status PriorElem(SqList L, ElemType e, ElemType* pre) {
    if (L.elem == NULL)
        return INFEASTABLE;
    for (int i = 1; i < L.length; i++) {
        if (L.elem[i] == e) {
            *pre = L.elem[i - 1];
            return OK;
        }
    }
    return ERROR;
}

status NextElem(SqList L, ElemType e, ElemType* next) {
    if (L.elem == NULL)
        return INFEASTABLE;
```

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

status ListInsert(SqList* L, int i, ElemType e) {
    if (L->elem == NULL)
        return INFEASTABLE;
    if (i < 1 || i > L->length + 1) {
        return ERROR;
    }
    if (L->length >= L->listsize) {
        ElemType* newbase = (ElemType*)realloc(L->elem, sizeof(ElemType) * (L->list
        if (!newbase) {
            return OVERFLOW;
        }
        L->elem = newbase;
        L->listsize += LISTINCREMENT;
    }
    for (int j = L->length; j >= i; j--) {
        L->elem[j] = L->elem[j - 1];
    }
    L->elem[i - 1] = e;
    L->length++;
    return OK;
}
```

```
status ListDelete(SqList* L, int i, ElemType* e) {
    if (L->elem == NULL) {
        return INFEASTABLE;
    }
    if (i < 1 || i > L->length) {
        return ERROR;
    }
    *e = L->elem[i - 1];
    for (int j = i; j < L->length; j++) {
        L->elem[j - 1] = L->elem[j];
    }
    L->length--;
    return OK;
}

status ListTrabverse(SqList L) {
    int i;
    printf("线性表'%s'的元素:\n", L.name);
    for (i = 0; i < L.length; i++) printf("%d␣", L.
elem[i]);
    return L.length;
}

int MaxSubArray(SqList L) {
    if (L.elem == NULL || L.length == 0) return INFEASTABLE;

    int maxSum = L.elem[0];
    int currentSum = L.elem[0];

    for (int i = 1; i < L.length; i++) {
        currentSum = (currentSum + L.elem[i]) > L.elem[i] ? (
```

```
        currentSum + L.elem[i]) : L.elem[i];
        maxSum = currentSum > maxSum ? currentSum : maxSum;
    }

    return maxSum;
}

int SubArrayNum(SqList L, int k) {
    if (L.elem == NULL || L.length == 0) return INFEASTABLE;

    int count = 0;

    for (int i = 0; i < L.length; i++) {
        int sum = 0;
        for (int j = i; j < L.length; j++) {
            sum += L.elem[j];
            if (sum == k) {
                count++;
            }
        }
    }

    return count;
}

status sortList(SqList* L) {
    if (L->elem == NULL) return INFEASTABLE;
    for (int i = 0; i < L->length - 1; i++) {
        for (int j = 0; j < L->length - i - 1; j++) {
            if (L->elem[j] > L->elem[j + 1]) {
                ElemType temp = L->elem[j];
```

```
                L->elem[j] = L->elem[j + 1];
                L->elem[j + 1] = temp;
            }
        }
    }

    return OK;
}

status saveListToFile(SqList L, const char* filename) {
    if (L.elem == NULL) return INFEASTABLE;

    FILE* file = fopen(filename, "wb");
    if (file == NULL) return ERROR;
    fwrite(&L.length, sizeof(int), 1, file);
    fwrite(L.elem, sizeof(ElemType), L.length, file);
    fclose(file);
    return OK;
}

status loadListFromFile(SqList* L, const char* filename) {
    FILE* file = fopen(filename, "rb");
    if (file == NULL) return ERROR;
    if (L->elem != NULL) {
        free(L->elem);
        L->elem = NULL;
    }
    int length;
    fread(&length, sizeof(int), 1, file);
    L->elem = (ElemType*)malloc(length * sizeof(ElemType));
    if (L->elem == NULL) {
```

```
        fclose(file);
        return OVERFLOW;
    }
    fread(L->elem, sizeof(ElemType), length, file);
    L->length = length;
    L->listsize = length;
    fclose(file);
    return OK;
}

status initMultiListManager(MultiListManager* manager) {
    manager->count = 0;
    manager->current = -1;
    return OK;
}

status addNewList(MultiListManager* manager, const char* name) {
    if (manager->count >= MAX_LISTS) return ERROR;
    SqList* newList = &manager->lists[manager->count];

    if (InitList(newList, name) == OK) {
        manager->count++;
        manager->current = manager->count - 1;
        return OK;
    }
    else {
        return ERROR;
    }
}

status removeList(MultiListManager* manager, int index) {
```

```
    if (index < 0 || index >= manager->count) return ERROR;
    DestroyList(&manager->lists[index]);
    for (int i = index; i < manager->count - 1; i++) {
        manager->lists[i] = manager->lists[i + 1];
    }
    memset(&manager->lists[manager->count - 1], 0, sizeof(SqList));
    manager->count--;

    if (manager->current >= manager->count) {
        manager->current = manager->count - 1;
        if (manager->current < 0) manager->current = -1;
    }

    return OK;
}

int findListIndexByName(MultiListManager* manager, const char*
name) {
    for (int i = 0; i < manager->count; i++) {
        if (strcmp(manager->lists[i].name, name) == 0) {
            return i;
        }
    }
    return -1; 没找到
}

status switchToListByName(MultiListManager* manager, const char*
name) {
    int index = findListIndexByName(manager, name);
    if (index == -1) return ERROR;
```

```
    manager->current = index;
    return OK;
}

void printListNames(MultiListManager* manager) {
    printf("\n所有线性表名称: \n");
    for (int i = 0; i < manager->count; i++) {
        printf("%d. %s\n", i + 1, manager->lists[i].name);
    }
    printf("\n");
}
void searchList(MultiListManager* manager) {
    char listName[MAX_NAME_LENGTH];
    printf("请输入要搜索的表的名称: ");
    scanf("%s", listName);
    int index = findListIndexByName(manager, listName);
    if (index != -1) {
        printf("找到名为'%s'的表, 其索引为 %d\n", listName,
index+1);
    }
    else {
        printf("未找到名为'%s'的表\n", listName);
    }
}
```

附录 D 基于二叉链表的二叉树实现

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

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

typedef int status;
typedef int KeyType;

typedef struct {
    KeyType key;
    char others[20];
}TElemType;

typedef struct BiTNode {
    TElemType data;
    struct BiTNode* lchild, * rchild;
}BiTNode, * BiTree;

typedef BiTNode* ElemType;

#define MAXLENGTH 100
```

```
typedef struct QUEUE {
    ElemType elem[MAXLENGTH];
    int front, length;
}QUEUE;

二叉树的管理表
typedef struct {
    struct {
        char name[30];
        BiTree T;
    }elem[10];
    int length;
    int listsize;
}LISTS;



static int flag_create = 0, k = 0;
static BiTNode* p = NULL;
static int flag1 = 0;

status Compare(KeyType e, TElemType elem);
void visit(BiTree elem);
int max(int x, int y);

status(*CompareArr)(KeyType e, TElemType elem) = Compare;
void (*visitArr)(BiTree elem) = visit;

void iniQueue(QUEUE& Q);
status enQueue(QUEUE& Q, ElemType e);
status deQueue(QUEUE& Q, ElemType& e);
```

```
status CreateBiTree(BiTree& T, TElemType definition[]);
status ClearBiTree(BiTree& T);
status DestroyBiTree(BiTree& T);
status ClearBiTree(BiTree& T);
status BiTreeEmpty(BiTree T);
int BiTreeDepth(BiTree T);
BiTNode* LocateNode(BiTree T, KeyType e);
status Assign(BiTree& T, KeyType e, TElemType value);
BiTNode* GetSibling(BiTree T, KeyType e);
status InsertNode(BiTree& T, KeyType e, int LR, TElemType c);
status DeleteNode(BiTree& T, KeyType e);
status PreOrderTraverse(BiTree T, void (*visit)(BiTree));
status InOrderTraverse(BiTree T, void (*visit)(BiTree));
status PostOrderTraverse(BiTree T, void (*visit)(BiTree));
status LevelOrderTraverse(BiTree T, void (*visit)(BiTree));

int MaxPathSum(BiTree T);
BiTNode* LowestCommonAncestor(BiTree T, KeyType e1, KeyType e2);
status InvertTree(BiTree& T);

status SaveBiTree(BiTree T, char FileName[]);
status LoadBiTree(BiTree& T, char FileName[]);

status AddTree(LISTS& Trees, char ListName[]);
status RemoveTree(LISTS& Trees, char ListName[]);
int LocateTree(LISTS Trees, char ListName[]);
status ListsTraverse(LISTS Trees);
BiTree* ChooseTree(BiTree* T, LISTS& Trees, int i);

int main(void)
{
```

```
BiTree T1 = NULL;
BiTree* T = &T1;

LISTS Trees;
Trees.length = 0, Trees.listsize = 10;

int op = 1, flag = 0, j = 0;
char FileName[30], ListName[30];
BiTNode* BTN = NULL;
TElemType definition[100];
memset(definition, 0, 100 * sizeof(TElemType));
while (op)
{
    system("cls");
    printf("\n\n");
    printf("请选择你的操作[0~24]:");
    scanf("%d", &op);
    switch (op)
    {
    case 1:
        if (*T != NULL)
            printf("二叉树已存在! 无法再创建! \n");
        else
        {
            flag_create = 0, k = 0;
            printf("请输入前序遍历序列: ");
            do
            {
                scanf("%d␣%s", &definition[j].key,
                definition[j].others);
            } while (definition[j++].key != -1);
```

```
            flag = CreateBiTree(*T, definition);
            if (flag == OK)
                printf("二叉树创建成功! \n这是一个与独立于多树的单
树, 你可以对它进行1~19操作! \n");
            else if (flag == ERROR)
                printf("关键词不唯一! 二叉树创建失败! \n");
        }
        getchar();getchar();
        break;
    case 2:销毁
        flag = DestroyBiTree(*T);
        if (flag == OK)
            printf("成功销毁二叉树并释放数据元素的空间! \n");
        else
            printf("不能对不存在的二叉树进行销毁操作! \n");
        getchar();getchar();
        break;
    case 3:清空
        flag = ClearBiTree(*T);
        if (flag == OK)
            printf("成功清空二叉树! \n");
        else
            printf("不能对不存在的二叉树进行清空操作! \n");
        getchar();getchar();
        break;
    case 4:判空
        flag = BiTreeEmpty(*T);
        if (flag == TRUE)
            printf("二叉树为空! \n");
        else if (flag == FALSE)
            printf("二叉树不为空! \n");
```

```
            else
                printf("不能对不存在的二叉树进行判空操作! \n");
            getchar();getchar();
            break;
        case 5:求深
            if(*T == NULL)
            printf("不能对不存在的二叉树进行求深操作! \n");
             else
        {
            flag = BiTreeDepth(*T);
            printf("二叉树的深度为␣%d! \n", flag);
        }
        getchar();getchar();
        break;
        case 6:查找结点
            if (*T == NULL)
                printf("不能对不存在的二叉树进行查找结点操作! \n");
            else
            {
                int e = 0;存储关键字
                BTN = NULL;存储所求结点
                printf("请输入想要查找的结点的关键字: ");
                scanf("%d", &e);
                BTN = LocateNode(*T, e);
                if (BTN == NULL)
                    printf("查找失败! 不存在关键字为␣%d␣的结点! \n"
, e);
                else
                    printf("查找成功! 该结点关键字为␣%d, 内容为␣%s!
\n",
                    BTN->data.key, BTN->data.others);
```

```
        }
        getchar();getchar();
        break;
    case 7:结点赋值
        if (*T == NULL)
            printf("不能对不存在的二叉树进行结点赋值操作! \n");
        else
        {

            int e = 0;存储关键字
            TElemType value[5];存储新关键字与内容
            memset(value, 0, 5 * sizeof(TElemType));初始化
            printf("请输入想要赋值的结点的关键字: ");
            scanf("%d", &e);
            printf("请输入将赋的关键字与内容: ");
            scanf("%d␣%s", &value->key, value->others);
            flag = Assign(*T, e, *value);
            if (flag == OK)
                printf("赋值成功! \n");
            else
                printf("赋值失败! 关键词已存在! \n");
        }
        getchar();getchar();
        break;
    case 8:获取兄弟
        if (*T == NULL)
            printf("不能对不存在的二叉树进行获取兄弟操作! \n");
        else
        {
            p = NULL;重置static变量
            int e = 0;存储关键字
```

```
                BTN = NULL;存储所求结点
                printf("请输入想要获取其兄弟结点的结点的关键字: ");
                scanf("%d", &e);
                BTN = GetSibling(*T, e);
                if (BTN == NULL)
                    printf("获取失败! 不存在关键字为%d的结点或该结
点没有兄弟! \n", e);
                else
                    printf("获取成功! 该结点的兄弟结点的关键字为%d
内容为␣%s! \n", BTN->data.key, BTN->data.others);
            }
            getchar();getchar();
            break;
        case 9:插入结点
            if (*T == NULL)
                printf("不能对不存在的二叉树进行插入结点操作! \n");
            else
            {
                int e = 0, LR = 0;存储关键字与插入方式
                printf("请输入想要插入的结点的父结点的关键字: ");
                scanf("%d", &e);
                printf("LR␣=␣0␣作为左孩子\nLR␣=␣1␣作为右孩子
\nLR␣=␣-1␣作为根结点\n");
                printf("请输入插入方式: ");
                scanf("%d", &LR);
                if (LR == -1 || LR == 0 || LR == 1)
                {
                    TElemType c[5];存储新关键字与内容
                    memset(c, 0, 5 * sizeof(TElemType));初始化
                    printf("请输入将插入的关键字与内容: ");
                    scanf("%d␣%s", &c[0].key, c[0].others);
```

```
            flag = InsertNode(*T, e, LR, c[0]);
            if (flag == OK)
                printf("插入成功! \n");
            else
                printf("插入失败! 关键词已存在! \n");
        }
        else
            printf("插入失败! 插入方式输入有误! \n");
    }
    getchar();getchar();
    break;
case 10:删除结点
    if (*T == NULL)
        printf("不能对不存在的二叉树进行插入结点操作! \n");
    else
    {
        flag1 = 0;重置static变量
        int e = 0;存储关键字
        printf("请输入想删除的结点的关键字: ");
        scanf("%d", &e);
        flag = DeleteNode(*T, e);
        if (flag == OK)
            printf("删除成功! \n");
        else
            printf("删除失败! \n");
    }
    getchar();getchar();
    break;
case 11:前序遍历
    if (*T == NULL)
        printf("不能对不存在的二叉树进行前序遍历操作! \n");
```

```
        else
        {
            PreOrderTraverse(*T, visit);
        }
        getchar();getchar();
        break;
    case 12:中序遍历
        if (*T == NULL)
            printf("不能对不存在的二叉树进行中序遍历操作! \n");
        else
        {
        InOrderTraverse(*T, visit);
        }
        getchar();getchar();
        break;
    case 13:后序遍历
        if (*T == NULL)
            printf("不能对不存在的二叉树进行后序遍历操作! \n");
        else
        {
             PostOrderTraverse(*T, visit);
        }
        getchar();getchar();
        break;
    case 14:层序遍历
        if (*T == NULL)
            printf("不能对不存在的二叉树进行层序遍历操作! \n");
        else
        {
        LevelOrderTraverse(*T, visit);
        }
```

```
            getchar();getchar();
            break;
        case 15:求最大路径和
            if (*T == NULL)
                printf("不能对不存在的二叉树进行该操作! \n");
            else
            {
                flag = MaxPathSum(*T);
                printf("该二叉树最大路径和为␣%d! ", flag);
            }
            getchar();getchar();
            break;
        case 16:求最近公共祖先
            if (*T == NULL)
                printf("不能对不存在的二叉树进行该操作! \n");
            else
            {
                int e1 = 0, e2 = 0;breaklines//breaklines存储关键字
                BTN = NULL;breaklines//breaklines存储所求结点
                printf("请输入二叉树中两个结点的关键字: \n");
                scanf("%d%d", &e1, &e2);
                BTN = LowestCommonAncestor(*T, e1, e2);
                if (BTN == NULL)
                    printf("操作失败! 关键字输入有误! \n");
                else
                    printf("关键字为%d和%d的两个结点的最近公共祖先为%d
%s! \n", e1, e2, BTN->data.key, BTN->data.others);
            }
            getchar();getchar();
            break;
        case 17:翻转二叉树
```

```
            if (*T == NULL)
                printf("不能对不存在的二叉树进行该操作! \n");
            else
                if (InvertTree(*T) == OK)
                    printf("二叉树翻转成功! ");
            getchar();getchar();
            break;
        case 18:写入文件
            if (*T == NULL)
                printf("不能对不存在的二叉树进行该操作! \n");
            else
            {
                printf("请输入文件名称: ");
                scanf("%s", FileName);
                flag = SaveBiTree(*T, FileName);
                if (flag == ERROR)
                    printf("文件打开失败! \n");
                else if (flag == OK)
                    printf("二叉树的内容已经复制到名称为%s的文件
中! \n", FileName);
            }
            getchar();getchar();
            break;
        case 19:读取文件
            printf("请输入文件名称: ");
            scanf("%s", FileName);
            flag = LoadBiTree(*T, FileName);
            if (flag == ERROR)
                printf("文件打开失败! \n");
            else if (flag == OK)
                printf("名称为␣%s␣的文件中的内容已复制到二叉树中! \n",
```

```
FileName);
            else if (flag == INFEASIBLE)
                printf("不能对已存在的二叉树进行进行读文件操作! 请先销
毁二叉树! \n");
            else if (flag == OVERFLOW)
                printf("溢出! \n");
            getchar();getchar();
            break;
        case 20:增加二叉树
            flag_create = 0, k = 0;
            printf("请输入新增二叉树的名称: ");
            scanf("%s", ListName);
            flag = AddTree(Trees, ListName);
            if (flag == OK)
                printf("成功新增名称为␣%s␣的二叉树! \n", ListName);
            else
                printf("新增失败! \n");
            getchar();getchar();
            break;
        case 21:删除二叉树
            if (Trees.length)
            {
                for (int i = 0; i < Trees.length; i++)
                    printf("%d␣%s\n", i + 1, Trees.elem[i].name);
            }
            else
            {
                printf("二叉树的集合为空! 无法进行此操作! \n");
                getchar();getchar();
                break;
            }
```

```
        printf("请输入想要删除的二叉树的名称: ");
        scanf("%s", ListName);
        flag = RemoveTree(Trees, ListName);
        if (flag == OK)
            printf("成功删除名称为␣%s␣的二叉树! \n", ListName);
        else if (flag == ERROR)
            printf("删除失败! \n");
        getchar();getchar();
        break;
    case 22:查找二叉树
    {
        if (!Trees.length)
        {
            printf("二叉树的集合为空! 无法进行此操作! \n");
            getchar();getchar();
            break;
        }
        int loc = 0;
        printf("请输入想要查找的二叉树的名称: ");
        scanf("%s", ListName);
        loc = LocateTree(Trees, ListName);
        if (loc == 0)
            printf("不存在名称为%s的二叉树! \n", ListName);
        else
            printf("名称为%s的二叉树的序号为%d! \n",ListName,loc);
        getchar();getchar();
        break;
    }
    case 23:遍历总表
        flag = ListsTraverse(Trees);
        if (flag == ERROR)
```

```
            printf("二叉树的集合为空! \n");
        getchar();getchar();
        break;
    case 24:选择单树进行操作
    {
        if (Trees.length)
        {
           for (int i = 0; i < Trees.length; i++)
               printf("%d␣%s\n", i + 1, Trees.elem[i].name);
         }
        else
        {
            printf("二叉树的集合为空! 无法进行此操作! \n");
            getchar();getchar();
            break;
        }
        int e = 0;
        printf("请输入想要进行操作的二叉树的序号 (从1开始) : ");
        scanf("%d", &e);
        getchar();
        T = ChooseTree(T, Trees, e);
        if (T == NULL)
            printf("输入的序号不合法! 单表已置空! \n");
        else
            printf("下面可对二叉树集合中序号为␣%d␣的二叉树进行操作! \n", e);
        getchar();getchar();
        break;
    }
    case 0:
        break;
    }
```

```
    }
    printf("欢迎下次再使用本系统! \n");
    return 0;
}

排序函数
status Compare(KeyType e, TElemType elem)
{
    if (elem.key == e)
        return OK;
    return ERROR;
}

访问函数
void visit(BiTree elem)
{
    printf("| %d %s |", elem->data.key, elem->data.others);
}

求最大值函数
int max(int x, int y)
{
    return (x >= y) ? x : y;
}

初始化队列Q
void iniQueue(QUEUE& Q)
{
    Q.front = 0;
    Q.length = 0; 队列长度为0
}
```

```
 元素入队
status enQueue(QUEUE& Q, ElemType e)
{
    if (Q.length >= MAXLENGTH) 队列溢出
        return ERROR;
    Q.elem[(Q.front + Q.length++) % MAXLENGTH] = e;
    return OK;
}

元素出队
将Q队首元素出队，赋值给e。成功出队返回1，否则返回0
status deQueue(QUEUE& Q, ElemType& e)
{
    if (Q.length == 0) breaklines//breaklines队列为空
        return ERROR;
    e = Q.elem[Q.front];
    Q.front = (Q.front + 1) % MAXLENGTH;
    Q.length--;
    return OK;
}

 功能1 创建
 根据带空枝的二叉树先根遍历序列definition构造一棵二叉树，将根节点
指针赋值给T并返回OK，如果有相同的关键字，返回ERROR。
status CreateBiTree(BiTree& T, TElemType definition[])
{
    if (flag_create == 0)
    {
        int i = 0, j = 0; 关键字检查循环用变量
        for (i = 0; definition[i].key != -1; i++)
```

```
    {
        if (definition[i].key == 0)
            continue;
        for (j = i + 1;definition[j].key != -1;j++)
            if (definition[j].key != 0)
                if (definition[i].key == definition[j].key)
                    return ERROR;
    }
    flag_create = 1;  标记已经进行关键字检查，后续递归无需再次检查
}

if (flag_create == 1)
{
    if (definition[k].key == 0) 关键字为0时赋NULL
    {
        T = NULL;
        k++;   definition向前推进至下一结点
    }
    else   关键字不为0时生成结点
    {
        if (!(T = (BiTNode*)malloc(sizeof(BiTNode))))
            return OVERFLOW;
        T->data = definition[k];
        k++;   definition向前推进至下一结点
        CreateBiTree(T->lchild, definition);递归生成左子树
        CreateBiTree(T->rchild, definition);递归生成右子树
    }
    return OK;
}

添加默认返回值，处理 flag_create 不等于 0 或 1 的情况
```

```
    return ERROR;
}

功能2 销毁
初始条件是二叉树T已存在; 操作结果是销毁二叉树T。
status DestroyBiTree(BiTree& T)
{
    if (T)
    {
        DestroyBiTree(T->lchild);递归销毁左子树
        DestroyBiTree(T->rchild);递归销毁右子树
        free(T);   销毁分配的空间
        T = NULL;  必须置空, free后指针仍指向原处
        return OK;
    }
    return ERROR;
}

功能3 清空
初始条件是二叉树T存在; 操作结果是将二叉树T清空。
status ClearBiTree(BiTree& T)
{
    if (T)
    {
        ClearBiTree(T->lchild);递归清空左子树
        ClearBiTree(T->rchild);递归清空右子树
        memset(T, 0, sizeof(BiTNode));清空不销毁
        return OK;
    }
    return ERROR;
}
```

```
功能4 判空
初始条件是二叉树T存在; 操作结果是若T为空二叉树则返回TRUE,
否则返回FALSE。
status BiTreeEmpty(BiTree T)
{
    if (T == NULL) 二叉树不存在
        return INFEASIBLE;
    else if (T->data.key == 0)
        return TRUE;
    return FALSE;
}

功能5 求深
初始条件是二叉树T存在; 操作结果是返回T的深度。
int BiTreeDepth(BiTree T)
{
    int depthL = 0, depthR = 0;
    if (T == NULL)
        return 0;
    depthL = BiTreeDepth(T->lchild);递归求左子树深度
    depthR = BiTreeDepth(T->rchild);递归求右子树深度
    if (depthL > depthR)
        return (depthL + 1);
    else
        return (depthR + 1);
}

功能6 查找结点
初始条件是二叉树T已存在, e是和T中结点关键字类型相同的给定值。
操作结果是返回查找到的结点指针, 如无关键字为e的结点, 返回NULL。
```

```
BiTNode* LocateNode(BiTree T, KeyType e)
{
    BiTNode* p = NULL, * q = NULL;保存左子树与右子树中查找到的结点
    if (T == NULL)
        return NULL;
    if (T->data.key == e)
        return T;
    p = LocateNode(T->lchild, e);递归查找左子树
    q = LocateNode(T->rchild, e);递归查找右子树
    if (p != NULL)所查结点在左子树中
        return p;
    if (q != NULL)所查结点在右子树中
        return q;
    return NULL;
}

功能7 结点赋值
初始条件是二叉树T已存在, e是和T中结点关键字类型相同的给定值。
操作结果是关键字为e的结点赋值为value。
status Assign(BiTree& T, KeyType e, TElemType value)
{
    static BiTNode* pt = NULL;
    static int flag3 = 0;
    if (T == NULL)
        return OK;
    if (T->data.key == value.key && T->data.key != e) 关键词冲突
    {
        pt = NULL;
        flag3 = 1;
        return ERROR;
    }
```

```
    if (T->data.key == e && !flag3)
    {
        pt = T;
        return OK;
    }
    Assign(T->lchild, e, value); 递归查找左子树
    Assign(T->rchild, e, value); 递归查找右子树
    if (pt != NULL)
    {
        pt->data.key = value.key;
        strcpy(pt->data.others, value.others);
        return OK;
    }
    else
        return ERROR;
}

功能8 获取兄弟
初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值。
操作结果是返回关键字为e的结点的（左或右）兄弟结点指针。若关键字
为e的结点无兄弟，则返回NULL。
BiTNode* GetSibling(BiTree T, KeyType e)
{

    if (T == NULL)
        return NULL;
    if (T->data.key == e)
        return T;
    if (GetSibling(T->lchild, e) != NULL && !p)
        p = T->rchild;
    if (GetSibling(T->rchild, e) != NULL && !p)
```

```
        p = T->lchild;
    return p;
}

功能9 插入结点
初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值，LR为0或
1，c是待插入结点。
操作结果是根据LR为0或者1，插入结点c到T中，作为关键字为e的结点的左
或右孩子结点，结点e的原有左子树或右子树则为结点c的右子树；
LR为-1时，作为根结点插入，原根结点作为c的右子树。
status InsertNode(BiTree& T, KeyType e, int LR, TElemType c)
{
    static BiTNode* p = NULL;
    if (LR == -1) 作为根结点插入
    {
        p = (BiTNode*)malloc(sizeof(BiTNode));
        p->data = c;
        p->lchild = NULL;
        p->rchild = T;
        T = p;
        return OK;
    }
    if (T == NULL)
        return OK;
    if (T->data.key == c.key)
        return ERROR;
    if (T->data.key == e)
    {
        p = (BiTNode*)malloc(sizeof(BiTNode));
        p->data = c;
        if (!LR) 作为左子树插入
```

```
        {
            p->rchild = T->lchild;
            T->lchild = p;
            p->lchild = NULL;
        }
        else 作为右子树插入
        {
            p->rchild = T->rchild;
            T->rchild = p;
            p->lchild = NULL;
        }
        return OK;
    }
    InsertNode(T->lchild, e, LR, c);
    InsertNode(T->rchild, e, LR, c);
    if (p != NULL)
        return OK;
    else
        return ERROR;
}

功能10 删除结点
初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值。
操作结果是删除T中关键字为e的结点；同时，如果关键字为e的结点度为0，
删除即可。
如关键字为e的结点度为1，用关键字为e的结点孩子代替被删除的e位置。
如关键字为e的结点度为2，用e的左孩子代替被删除的e位置，e的右子树作为
e的左子树中最右结点的右子树。
status DeleteNode(BiTree& T, KeyType e)
{
    if (T == NULL)
```

```
        return OK;
    if (T->data.key == e)
    {
        flag1 = 1;
        if (T->lchild == NULL && T->rchild == NULL) 度为0, 直接删除
        {
            free(T);
            T = NULL;
            return OK;
        }
        else if (T->lchild != NULL && T->rchild == NULL) 度为1, 用孩子代替e
        {
            BiTNode* p = T->lchild;
            free(T);
            T = p;
            return OK;
        }
        else if (T->lchild == NULL && T->rchild != NULL)
        {
            BiTNode* p = T->rchild;
            free(T);
            T = p;
            return OK;
        }
        else 度为2, 用左孩子代替e, 右子树作为左子树中最右结点的右子树
        {
            BiTNode* p = T->lchild;
            BiTNode* q = T->rchild;
            free(T);
            T = p;
            BiTNode* tmp = T;
```

```
            for (; tmp->rchild != NULL; tmp = tmp->rchild)
                ;
            tmp->rchild = q;
            return OK;
        }
    }
    DeleteNode(T->lchild, e);
    DeleteNode(T->rchild, e);
    if (!flag1)
        return ERROR;
    else
        return OK;
}
```

功能11 前序遍历

初始条件是二叉树T存在，Visit是一个函数指针的形参（可使用该函数对结点操作）。

操作结果：先序遍历，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

```
status PreOrderTraverse(BiTree T, void (*visit)(BiTree))
{
    if (T)
    {
        visit(T);访问当前结点
        PreOrderTraverse(T->lchild, visit);递归遍历左子树
        PreOrderTraverse(T->rchild, visit);递归遍历右子树
    }
    return OK;
}
```

功能12 中序遍历

初始条件是二叉树T存在，Visit是一个函数指针的形参（可使用该函数对结点操作）。
操作结果是中序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

```
status InOrderTraverse(BiTree T, void (*visit)(BiTree))
{
    if (T)
    {
        InOrderTraverse(T->lchild, visit);递归遍历左子树
        visit(T);访问当前结点
        InOrderTraverse(T->rchild, visit);递归遍历右子树
        return OK;
    }
    else
        return OK;
}
```

功能13 后序遍历
初始条件是二叉树T存在，Visit是一个函数指针的形参（可使用该函数对结点操作）。
操作结果是后序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败

```
status PostOrderTraverse(BiTree T, void (*visit)(BiTree))
{
    if (T)
    {
        PostOrderTraverse(T->lchild, visit);递归遍历左子树
        PostOrderTraverse(T->rchild, visit);递归遍历右子树
        visit(T);breaklines//breaklines访问当前结点
        return OK;
    }
```

```
    else
        return OK;
}

功能14 层序遍历
初始条件是二叉树T存在，Visit是对结点操作的应用函数。
操作结果是层序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败
，则操作失败。
status LevelOrderTraverse(BiTree T, void (*visit)(BiTree))
{
    BiTNode* p = NULL;
    QUEUE qu;
    iniQueue(qu);
    enQueue(qu, T);
    while (qu.length)
    {
        deQueue(qu, p);
        visit(p);
        if (p->lchild != NULL)
            enQueue(qu, p->lchild);左孩子进队
        if (p->rchild != NULL)
            enQueue(qu, p->rchild);右孩子进队
    }
    return OK;
}

附加功能15 求最大路径和
初始条件是二叉树T存在；操作结果是返回根节点到叶子结点的最大路径和。
int MaxPathSum(BiTree T)
{
    if (T == NULL)
```

```
        return 0;
    else
        return max(T->data.key + MaxPathSum(T->lchild), T-
>data.key + MaxPathSum(T->rchild));
}
```

附加功能16 求最近公共祖先

初始条件是二叉树T存在；操作结果是该二叉树中e1节点和e2节点的最近公共祖先。

```
BiTNode* LowestCommonAncestor(BiTree T, KeyType e1, KeyType
e2)
{
    if (T == NULL)
        return NULL;
    if (T->data.key == e1 || T->data.key == e2)
        return T;
    BiTNode* left = LowestCommonAncestor(T->lchild, e1, e2);
    BiTNode* right = LowestCommonAncestor(T->rchild, e1, e2);
    if (left == NULL)
        return right;
    if (right == NULL)
        return left;
    return T;
}
```

附加功能17 翻转二叉树

初始条件是线性表L已存在；操作结果是将T翻转，使其所有节点的左右节点互换。

```
status InvertTree(BiTree& T)
{
    if (T == NULL)
        return OK;
```

```
    InvertTree(T->lchild);递归翻转左子树
    InvertTree(T->rchild);递归翻转右子树
    BiTree tmp = NULL;交换左右结点时的临时存储结点
    tmp = T->lchild;
    T->lchild = T->rchild;
    T->rchild = tmp;
    return OK;
}

 附加功能18 写入文件
 如果二叉树T存在，将二叉树T的的元素写到FileName文件中，返回OK。
status SaveBiTree(BiTree T, char FileName[])
{
    static FILE* fp1 = fopen(FileName, "w");
    if (T == NULL)
    {
        fprintf(fp1, "%d%s", 0, "null");
        return OK;
    }
    else
    {
        fprintf(fp1, "%d%s", T->data.key, T->data.others);
        SaveBiTree(T->lchild, FileName);
        SaveBiTree(T->rchild, FileName);
    }
    if (T->data.key == 1)
        fclose(fp1);
    return OK;
}
```

```
status LoadBiTree(BiTree& T, char FileName[])
{
    static FILE* fp2 = fopen(FileName, "r");
    int t = 0;
    char s[20];
    memset(s, 0, 20);
    if (feof(fp2))
        return OK;
    fscanf(fp2, "%d%s", &t, s);
    if (t == 0)
        T = NULL;
    else
    {
        T = (BiTNode*)malloc(sizeof(BiTNode));
        T->data.key = t;
        strcpy(T->data.others, s);
        LoadBiTree(T->lchild, FileName);
        LoadBiTree(T->rchild, FileName);
    }
    if (feof(fp2))
        fclose(fp2);
    return OK;
}

status AddTree(LISTS& Trees, char ListName[])
{
    for (int i = 0; i < Trees.length; i++)
        if (!strcmp(Trees.elem[i].name, ListName))
        {
            printf("集合中已存在名称为%s的二叉树! \n", ListName);
            return ERROR;
```

```
        }
    strcpy(Trees.elem[Trees.length].name, ListName);
    TElemType defi[100];
    memset(defi, 0, 100 * sizeof(TElemType));
    int j = 0, flag3 = 0;
    printf("请输入前序遍历序列: ");
    do
    {
        scanf("%d%s", &defi[j].key, defi[j].others);
    } while (defi[j++].key != -1);
    flag3 = CreateBiTree(Trees.elem[Trees.length].T, defi);
    if (flag3 == OK)
    {
        printf("名称为%s的二叉树创建成功! \n",
Trees.elem[Trees.length].name);
        Trees.length++;
    }
    else if (flag3 == ERROR)
        printf("关键词不唯一! 二叉树创建失败! \n");
    return OK;
}

status RemoveTree(LISTS& Trees, char ListName[])
{
    int i;
    for (i = 0; i < Trees.length; i++)
        if (!strcmp(Trees.elem[i].name, ListName))
            break;
    if (i == Trees.length)
        return ERROR;
    else
```

```
    {
        DestroyBiTree(Trees.elem[i].T);
        memset(Trees.elem[i].name, 0, strlen(Trees.elem[i].name));
        Trees.length--;
        for (int j = i; j < Trees.length; j++)
            Trees.elem[j] = Trees.elem[j + 1];
        return OK;
    }
}

int LocateTree(LISTS Trees, char ListName[])
{
    for (int i = 0; i < Trees.length; i++)
        if (!strcmp(Trees.elem[i].name, ListName))
            return (i + 1);
    return ERROR;
}

status ListsTraverse(LISTS Trees)
{
    if (Trees.length == 0)
        return ERROR;
    for (int n = 0; n < Trees.length; n++)
    {
        printf("表名为␣%s␣的元素有: ", Trees.elem[n].name);
        PreOrderTraverse(Trees.elem[n].T, visit);
        putchar('\n');
    }
    return OK;
}
```

```
BiTree* ChooseTree(BiTree* T, LISTS& Trees, int i)
{
    if (i > Trees.length || i < 1)
        return NULL;
    else
    {
        T = &(Trees.elem[i - 1].T);
        return T;
    }
}
```