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 计卓 2201

学　　号 U202215322

姓　　名 濮澍

指导教师 许贵平

报告日期 2023 年 04 月 20 日

计算机科学与技术学院

目　　录

1 基于顺序存储结构的线性表实现

为了强化相关数据结构知识, 学生将通过自主编程完成 ADT 的实现通过实验达到:

1. 加深对线性表的概念、基本运算的理解
2. 熟练掌握线性表的逻辑结构与物理结构的关系
3. 物理结构采用单链表, 熟练掌握线性表的基本运算的实现

1.1 问题描述

本实验将完成对顺序结-构线性表的基本操作与多线性链表的管理

1.2 系统设计

1.2.1 程序总设计

通过 CMakeLists.txt 将函数文件生成库, 将 main 文件与该库链接, 编译生成本实验采用了较为繁琐的每个函数设计一个文件，实际可通过类相关的进行优化

基于顺序存储结构的线性表的文件结构

- src - 包含了 main 文件 (即文件显示交互部分) 与 CMakeLists.txt
- Funcs - 包含了所有函数，通过 CMakeLists.txt 生成函数库放入 lib
- lib - 生成的函数库与头文件
- build - 构建文件夹
- bin - 可执行程序存放再此

1.2.2 数据结构设计

本实验需要设计两个数据元素

1. SQList
2. LISTS

其中 SQList 负责一个顺序表, 有长度 (length) 保存当前顺序表中元素个数, 大小 (size) 保存当前顺序表的最大存储个数,ElemType * elem 是指向该数据结构元素类型的指针，负责顺序存储相关数据对象类型的数据

而 LISTS 则是进行多顺序表管理的数据元素, 负责存储线性表, 通过 User 设定的名称管理多个线性表并进行删除, 添加。可通过遍历获得该集合的各个顺序表名字

Listing 1　Linear-List **重要结构体**

```
typedef struct SQList{
int length;
int size;
ElemType * elem = NULL;
}SQList;//顺序结构定义

//用线性表管理顺序表
typedef struct LISTS{
struct { char name[30];
SQList L;
} elem[10];
int length;
}LISTS;
```

1.2.3　相关算法设计

这里将对最小子列和与和为 k 的子列数两个操作进行算法描述最小子列和采用的是时间与空间复杂度最低的算法–动态规划 dp(i) 是下标 i 以前的子树列最大和, 则最终目标为 dp(n-1)

$$dp[i] = \max array[i], dp[i-1] + array[i]$$

这个的意思是如果某部分的连续数列和为负值，则丢弃不用此时的空间复杂度为 $O(n)$ 较大对此进行改良将数组压缩成 ElemType tmp, 当 tmp + array(i) < 0 时，因对最大子列和无贡献, 因此在 i 之前的全部删除, 而在此之前要保存最大值因此获得算法如下1.1

算法 1.1. 最大子列和算法

Input: An array $array$ of n integers

Output: The maximum subarray sum

```
procedure MaxSubArray(array)
    tmp ← 0
    max ← 0
    for i ← 0 → n − 1 do
        tmp ← tmp + array[i]
        if tmp < 0 then
            tmp ← 0
        end if
        if tmp > max then
            max ← tmp
        end if
    end for
    return max
end procedure
```

和为 k 的子列数用到了哈希表的思想先通过 sum 数组 – sum(i) 为 sum(i-1)+array(i)。所以如果有一段连续子数列和为 k 的话。假设为 (i–j) 为 k, 我们可以转化为 sum(j) - k == sum(i-1) 因此再建立一个哈希表, 记录 pre(i) 出现的次数即可 –> 记录 map(pre[i]-k) 的个数具体实现如下图1.2

算法 1.2. 连续子列和为 k 的个数

Input: An array $array$ of n integers

Output: The number of subarray sum is k

```
procedure SumKSubArray(array)
    map ← ∅
    map[0] ← 1
    sum ← 0
    ans ← 0
    for i ← 0 → n − 1 do
        sum ← sum + array[i]
```

if $map[sum-k] \neq \emptyset$ **then**

$ans \leftarrow ans + map[sum-k]$

end if

$map[sum] \leftarrow map[sum] + 1$

end for

return ans

end procedure

1.3 系统实现

- 初始化顺序表

```
status InitList(SQList &L);
```

- 销毁顺序表

```
status DestrList(SQList &L);
```

- 清空顺序表

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

- 初始条件是线性表 L 已存在; 操作结果是若 L 为空表则返回 TRUE, 否则返回 FALSE

```
status isEmpty(SQList L);
```

- 初始条件是线性表已存在; 操作结果是返回 L 中数据元素的个数;

```
status ListLength(SQList L);
```

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

```
status GetElem(SQList L,int i , int &e);
```

- 初始条件是线性表已存在; 操作结果是返回 L 中下标为 i 的 value

```
ElemType LocateElem(SQList L,int i);
```

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

```
ElemType PriorElem(SQList L,int e,int &pre);
```

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

```
ElemType NextElem(SQList L,int e,int &next);
```

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

```
status ListInsert(SQList &L,int i,int e);
```

- 初始条件是线性表 L 已存在且非空,1≤i≤ListLength(L); 操作结果: 删除 L 的第 i 个数据元素, 用 e 返回的值;

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

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

```
status ListTraverse(SQList L);
```

- 1. 需要设计文件数据记录格式，以高效保存线性表数据逻辑结构 (D,R) 的完整信息 2. 需要设计线性表文件保存和加载操作合理模式。附录 B 提供了文件存取的参考方法。

```
status SaveList(SQList L,char FileName[])
status LoadList(SQList &L, char FileName[])
```

- 设计相应的数据结构管理多个线性表的查找

```
status TraverseLISTS(LISTS Lists)
status InitLISTS(LISTS &L)
status AddList(LISTS &Lists,char ListName[])
status RemoveList(LISTS &Lists,char ListName[])
int LocateList(LISTS Lists,char ListName[],SQList &L)
```

- 初始条件是线性表 L 已存在; 操作结果是将 L 由小到大排序;

```
status SortList(SQList &L);
```

1.4 系统测试

在函数接口层面，可以利用返回值来处理出现的异常，在 lib SQList.h 文件中就定义了一组关于异常处理的宏定义：

```
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
```

- ClearList：当传入线性表为空时，返回 INFEASIBLE ，不为空时返回 OK 。
- LocateElem：当传入线性表为空时，返回 INFEASIBLE，找到该元素则返回 OK ，未找到则返回 ERROR 。
- PriorElem：当传入线性表为空时，返回 INFEASIBLE，当传入参数不是线性表中的一个元素或者是第一个元素时，返回 ERROR ，否则返回 OK 。
- NextElem：当传入线性表为空时，返回 INFEASIBLE，当传入参数不是线性表中的一个元素或者是最后一个元素时，返回 ERROR ，否则返回 OK 。
- ListInsert：当传入线性表为空时，返回 INFEASIBLE，当传入下标为负数或大于线性表长度时，返回 ERROR ，否则返回 OK 。
- ListDelete：当传入线性表为空时，返回 INFEASIBLE，当传入下标为负数或大于线性表长度减一时，返回 ERROR ，否则返回 OK 。

而在演示系统层面，则可以根据函数返回值来打印出更为详细的异常信息，对于不支持的操作也有错误的提示，详见图1-1。

```
case 1:
    //printf("\n----IntiList功能待实现! \n");
     if(InitList(L)==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(DestroyList(L) == OK) printf("线性表销毁成功! \n");
     else printf("无线性表，销毁失败\n");
     getchar();
     getchar();
     break;
case 3:
     if(ClearList(L) == OK)printf("线性表清空成功\n");
     else printf("无线性表，清空失败\n");
     getchar();
     getchar();
     break;
case 4:
   if(isEmpty(L) == INFEASIBLE)printf("线性表未初始化! \n");
   else if(isEmpty(L) == TRUE) printf("线性表为空\n");
   else printf("线性表非空! \n");
   getchar();
   getchar();
   break;
```

图 1-1　线性表系统测试

1.5　实验小结

在本节实验中，借助文件管理系统完成了对顺序存储线性表的操作，让我对顺序存储理解更加深刻，也让我独立思考，对曾经讲过的算法进行进一步的优化同时，在实验中实现了很多理论知识中不会提到的问题，在 debug 的途中也学习了许多东西最后，学习了伪代码是如何撰写的，并对个人代码进行优化，使得更加有可读性与规范性。

2 基于邻接表的图实现

为了强化相关数据结构知识, 学生将通过自主编程完成 ADT 的实现通过实验达到:

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，并重温链表运算
3. 物理结构采用邻接表, 熟练掌握线性表的基本运算来实现图

2.1 问题描述

本实验将完成对基于邻接表实现的无向图操作与多线性链表的管理

2.2 系统设计

2.2.1 程序总设计

通过 CMakeLists 构建项目，通过 main 函数调用测试函数，测试函数调用各个功能函数，实现对图的操作。

2.2.2 物理结构设计

本实验设计了两个数据结构 Graph 的邻接表存储结构与存储多表的线性表结构

Listing 2 Graph 重要结构体

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

typedef KeyType Map[MAX_KEY]; //关键字映射数组类型定义

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

```
typedef struct VNode{                              //头结点及其数组类型定义
          VertexType data;                 //顶点信息
          ArcNode *firstarc;                 //指向第一条弧
        } VNode,AdjList[MAX_VERTEX_NUM];
typedef  struct {  //邻接表的类型定义
        AdjList vertices;          //头结点数组
        int vexnum,arcnum;          //顶点数、弧数
        GraphKind  kind;          //图的类型
        } ALGraph;
typedef struct {
        struct{
                char name[20];  //图的名称
                ALGraph G;      //图的邻接表存储结构
        }elem[20]; //存储图的数组
        int length = 0; //图的数量
}GRAPHS;
```

2.2.3 部分算法设计

此部分将讲解部分算法的设计思路

1. 到结点小于 K 的节点
2. 求两点间最短路径
3. 求联通分量数

算法 1 2.1

算法 2.1. 到结点小于 K 的节点

Input: UG G, int k, vertex V

Output: An array of the length to V. Not reaching return -1

procedure VERTICESSETLESSTHANK(G, v, k)

$Path \leftarrow \emptyset$

$Path[i] \leftarrow -1$

$Path[v] \leftarrow 0$

$Queue \leftarrow \emptyset$

$Queue.push(v)$

while $Queue \neq \emptyset$ **do**

$u \leftarrow Queue.pop()$

for $w \in Adjacent(G, u)$ **do**
if $Path[w] = -1 || Path[w] \geq Path[u] + 1$ **then**
$Path[w] \leftarrow Path[u] + 1$
$Queue.push(w)$
end if
end for
end while
return $Path$
end procedure

算法 2 2.2

算法 2.2. 两节点间的最短路径

Input: UG G, vertex V1, vertex V2

Output: The shortest path from V1 to V2

procedure SHORTESTPATH($G, v1, v2$)
$Path \leftarrow \emptyset$
$Path[i] \leftarrow -1$
$Path[v1] \leftarrow 0$
$Queue \leftarrow \emptyset$
$Queue.push(v1)$
while $Queue \neq \emptyset$ **do**
$u \leftarrow Queue.pop()$
for $w \in Adjacent(G, u)$ **do**
if $Path[w] = -1 || Path[w] \geq Path[u] + 1$ **then**
$Path[w] \leftarrow Path[u] + 1$
$Queue.push(w)$
end if
end for
end while

return $Path[v2]$
end procedure

算法 3 2.3

算法 2.3. 求无向图联通分量个数

Input: UG G

Output: The number of connected components

procedure CONNECTEDCOMPONENTS(G)
$Count \leftarrow 0$
$Path \leftarrow \emptyset$
$Path[i] \leftarrow -1$
for $v \in V(G)$ **do**
if $Path[v] = -1$ **then**
$Count \leftarrow Count + 1$
$Path[v] \leftarrow Count$
$Queue \leftarrow \emptyset$
$Queue.push(v)$
while $Queue \neq \emptyset$ **do**
$u \leftarrow Queue.pop()$
for $w \in Adjacent(G, u)$ **do**
if $Path[w] = -1$ **then**
$Path[w] \leftarrow Count$
$Queue.push(w)$
end if
end for
end while
end if
end for
return $Count$

end procedure

2.3 系统实现

- 创建邻接表

```
status CreateGraph(ALGraph &G);
```

- 销毁邻接表

```
status DestroyGraph(ALGraph &G);
```

- 初始条件是图 G 存在; 操作结果是查找并返回顶点 v 在图 G 中的位置; 若图 G 中不存在顶点 v, 则返回-1

```
int LocateVex(ALGraph G,VertexType v);
```

- 初始条件是图 G 存在; 操作结果是更改 G 中顶点 v 的值为 value

```
status PutVex(ALGraph &G,VertexType v,VertexType value);
```

- 初始条件是图 G 存在; 操作是返回结点 v 的第一个邻接结点

```
int FirstAdjVex(ALGraph G,VertexType v);
```

- 初始条件是图 G 存在,v 是 G 中的一个顶点,w 是 v 的邻接顶点; 操作结果是返回 v 的 (相对于 w 的) 下一个邻接顶点

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

- 初始条件是图 G 存在, 操作为向图 G 中插入结点 v

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

- 初始条件是图 G 存在, 操作为从 G 中删除结点 v

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

- 初始条件是图 G 存在，v、w 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中增加弧 <v,w>，如果图 G 是无向图，还需要增加 <w,v>；
- 初始条件是图 G 存在，v、w 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中删除弧 <v,w>，如果图 G 是无向图，还需要删除 <w,v>；

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

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

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

- 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

```
int * VerticesSetLessThanK(ALGraph &G,KeyType v,int k);
```

- 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);// Using DFS
```

- 1. 需要设计文件数据记录格式，以高效保存邻接表数据逻辑结构的完整信息 2. 需要设计邻接表文件保存和加载操作合理模式。

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

- 设计相应的数据结构管理多个邻接表的查找

```
status Add_GRAPHS(GRAPHS &P,char GraphName[]);
status Remove_GRAPHS(GRAPHS &P,char GraphName[]);
status Locate_and_Modi_GRAPHS(GRAPHS &P,ALGraph &G,char GraphName[]);
status Traverse_GRAPHS(GRAPHS &P);
```

2.4 系统测试

在函数接口层面，可以利用返回值来处理出现的异常，在 Graph.h 中就定义了一组关于异常处理的宏

```
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define MAX_VERTEX_NUM 20
#define MAX_KEY 50
```

- CreateGraph: return FALSE 如果图为空，输入时不符合规范，或者图的顶点数超过了最大值
- InsertVex: return FALSE 如果没找到结点，或者图的顶点数超过了最大值
- DeleteArc: return FALSE 如果没找到结点，或者图的弧出现错误

图 2-1　图 main 的状态提示

2.5　实验小结

3 课程的收获和建议

通过本次实验，进行了对部分课上学习的重要数据结构的实现

3.1 基于顺序存储结构的线性表实现

对顺序表的部分，重要的操作基本与数组中元素移动相关而对应算法中，其实求最大顺序子数组和没有学过动态规划，更普通的方法是暴力求解，时间复杂度为 O(n2)

3.2 基于链式存储结构的线性表实现

链式表中销毁与删除操作并不相同，这在图中实现也很重要，因为为图的实现为 ADT

3.3 基于二叉链表的二叉树实现

二叉树实现让我对树数据结构更加熟悉。最复杂的部分是使用迭代代替递归实现遍历，很多时候都是通过自己模拟出一个栈来改递归为迭代。

3.4 基于邻接表的图实现

邻接表使用了类似数组加链表的结构，因此相对来说更加复杂。要实现的功能中 BFS 和 DFS 只要了解思想就可以较为容易地写出来，而计算最短路程就需要对图论算法有着基本的了解，因为没有负权边或者指向自己的边，所以我们可以直接使用 Dijkstra 算法，在算最短路径的同时记录下前驱节点，就可以通过回溯找到路径。

附录 A 基于顺序存储结构线性表实现的源程序

Listing 3　顺序线性表代码

```
                #include "SQList.h"
#include <cstdio>
#include <iostream>

status InitList(SQList &L){
        if (L.elem != NULL)
        return INFEASIBLE;
        int i = 0;
    L.length = 0;
    L.size = LIST_INIT_SIZE;
    L.elem = (ElemType *)malloc(L.size * sizeof(ElemType));
        ElemType e;
        std::cout << "请输入初始化的线性表，以0结束。";
        scanf("%d",&e);
        while(e && i<L.size){
        L.elem[i++] = e;
        L.length++;
        scanf("%d",&e);
  }
        return OK;
}

#include "SQList.h"
status DestroyList(SQList &L){
  if(L.elem == NULL)
        return INFEASIBLE;
    L.length = 0;
    L.size = 0;
    free(L.elem);
    L.elem = NULL;
    return OK;
}

#include "SQList.h"

status isEmpty(SQList L){
  if (L.elem == NULL) return INFEASIBLE;
    if(L.length == 0) return TRUE;
    return FALSE;
}
#include "SQList.h"


status AddList(LISTS &Lists,char ListName[]){
```

```
  if(Lists.length>=10) return ERROR;
    strcpy(Lists.elem[Lists.length].name,ListName);
    Lists.elem[Lists.length].L.elem = NULL;
    InitList(Lists.elem[Lists.length].L);
    Lists.length++;
    return OK;
}
#include "SQList.h"

status ClearList(SQList &L){
  if(L.elem == NULL)
        return INFEASIBLE;
    L.length = 0;
    free(L.elem);
    return OK;
}
#include "SQList.h"

status GetElem(SQList L,int i,ElemType &e){
 if(L.elem == NULL) return INFEASIBLE;
    if(i<1||i > L.length)  return ERROR;
    e = L.elem[i-1];
    return OK;
}
#include "SQList.h"

status InitLISTS(LISTS &L){
  int i;
  L.length = 0;
  for(int i = 0; i<10 ; i++)
        L.elem[i].name[i] = '\0';
  return OK;
}
#include "SQList.h"


status ListDelete(SQList &L,int i,ElemType &e){
  if(L.elem == NULL) return INFEASIBLE;
    if(i < 1 || i>L.length) return ERROR;
    e = L.elem[i-1];
    for (int j = i-1 ; j<L.length; j++){
        L.elem[j] = L.elem[j+1];
    }
    L.length--;
    return OK;
}
#include "SQList.h"


status ListInsert(SQList &L,int i,ElemType e){
```

```
  if(L.elem == NULL)return INFEASIBLE;
    if(i<=0|| i>L.length+1){
        return ERROR;}
    if(L.length>=L.size){
        L.elem = (ElemType *)realloc(L.elem,(LISTINCREMENT+L.size)*sizeof(ElemType));
        L.size+=LISTINCREMENT;
    }

    for (int j = L.length-1; j >= i-1 ; j--){
        L.elem[j+1] = L.elem[j];
    }
    L.elem[i-1] = e;
    L.length++;
    return OK;

}
#include "SQList.h"
status ListLength(SQList L)
// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(L.elem == NULL) return INFEASIBLE;
    return L.length;

    /********** End **********/
}
#include "SQList.h"


status ListTraverse(SQList L){
  if(L.elem == NULL) return INFEASIBLE;
    for ( int i = 0 ; i < L.length ; i++){
        printf("%d",L.elem[i]);
        printf((i == L.length - 1)?"" : " ");
    }
    return OK;
}
#include "SQList.h"


status LoadList(SQList &L,char FileName[]){
        if(L.elem) return INFEASIBLE;
    L.elem = (ElemType *) malloc(LISTINCREMENT * sizeof(ElemType));
    L.length = 0;
    L.size=LISTINCREMENT;
    FILE *fp;
    ElemType e;
    int i = 0;
    if((fp = fopen(FileName,"r")) == NULL) return INFEASIBLE;
```

```
    while(!feof(fp)){
        if(L.length >= L.size) {
            L.elem = (ElemType *)realloc(L.elem,(L.size+LISTINCREMENT) * sizeof(ElemType));
            L.size+=LISTINCREMENT;
        }
        fscanf(fp,"%d",&e);
        L.elem[L.length++] = e;
    }
        if(L.length == 0) return ERROR;
    L.length--;
    return OK;

}
#include "SQList.h"
int LocateElem(SQList L,ElemType e){
  if (L.elem == NULL) return INFEASIBLE;
    int i;
    for(i = 0; L.elem[i] != e && i<L.length ; i++);
    if(i == L.length) return ERROR;
    return i+1;
}

#include "SQList.h"


int LocateList(LISTS Lists,char ListName[],SQList &L){
        int i;
    for(i = 0 ; i<Lists.length ; i++){
        if(strcmp(ListName,Lists.elem[i].name) == 0){
                  L=Lists.elem[i].L;
                  break;
          }
    }
    if(i!=Lists.length){
        return i+1;
    }
    return 0;
}
#include "SQList.h"

ElemType MaxSubArray(SQList L){
  if(L.elem==NULL || L.length == 0) return INFEASIBLE;
  ElemType ThisSum = 0,MaxSum = 0;
  int index;
  for(index = 0 ; index < L.length ; index++){
        ThisSum+=L.elem[index];
        if(ThisSum < 0) ThisSum = 0;
        if(ThisSum > MaxSum) MaxSum = ThisSum;
  }
```

```
  return MaxSum;
}
#include "SQList.h"

status NextElem(SQList L,ElemType e,ElemType &next){
   if(L.elem == NULL) return INFEASIBLE;
    int i;
    for (i = 0 ; i<L.length ; i++){
        if(L.elem[i] == e) break;
    }
    if(i == L.length-1 || i == L.length) return ERROR;
    next = L.elem[i+1];
    return OK;
}
#include "SQList.h"

status PriorElem(SQList L,ElemType e,ElemType &pre){
   if(L.elem == NULL) return INFEASIBLE;
    int i = 0;
    while(i<L.length){
        if(L.elem[i]==e) break;
        i++;
    }
    if (i==0 || i == L.length) return ERROR;
    pre = L.elem[i-1];
    return OK;
}
#include "SQList.h"


status RemoveList(LISTS &Lists,char ListName[]){
// Lists中删除一个名称为ListName的线性表

    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    int flag = 0;
    int i;
    for(i = 0 ; i<Lists.length ; i++){
        if(strcmp(ListName,Lists.elem[i].name) == 0){
            DestroyList(Lists.elem[i].L);
            strcpy(Lists.elem[i].name,"\0");
            flag++;
            break;
        }
    }
    if(flag == 1){
        for(int j = i ; j<=Lists.length-1 ; j++){
            Lists.elem[j] = Lists.elem[j+1];
        }
                Lists.length--;
```

```
        return OK;
    }
    return ERROR;
    /********** End **********/
}
```

```
#include "SQList.h"


status  SaveList(SQList L,char FileName[]){
  if(L.elem == NULL) return INFEASIBLE;
    FILE *fp;
    fp = fopen(FileName,"w");
    for(int i = 0 ; i<L.length ; i++){
        fprintf(fp,"%d",L.elem[i]);
        fprintf(fp,(i==L.length-1)?"\n":" ");
    }
    fclose(fp);
    return OK;
}
#include "SQList.h"

status SortList(SQList &L){
  if(L.elem==NULL) return INFEASIBLE;
  ElemType min;
  for (int i = 0; i < L.length-1; i++) {
        min = i;
        for(int j = i+1 ; j<L.length ; j++){
          if(L.elem[min]>L.elem[j]) min = j;
        }
        if(i != min)
          swap(L.elem[i],L.elem[min]);
  }
  return OK;
}
#include "SQList.h"
#include <unordered_map>

int SubArrayNum(SQList L,ElemType k){
  int index = 0;
  ElemType sum = 0;
  if(L.elem == 0) return INFEASIBLE;
  //So elem[a...b] == k <==> hash_sum[b] - hash_sum[a-1] == k
  //hash_sum[b] - k == hash_sum[a-1]
  std::unordered_map <ElemType,ElemType> mp;
```

```
    mp[0] = 1;
    for(int i = 0 ; i < L.length ; i++){
          sum += L.elem[i];
          if (mp.find(sum - k) != mp.end()) {
                index += mp[sum - k];
      }
          //Mapping the sum to mp
          //If sum-k once appeared index can add it on
          mp[sum]++;
    }

    return index;
}
//Using Hash => maping the sum to the existence of sum
#include "SQList.h"

void swap(ElemType &x, ElemType &y){
    int tmp;
    tmp = x;
    x = y ;
    y = tmp;
}
#include "SQList.h"

status TraverseLISTS(LISTS Lists){
    if(!Lists.length) return INFEASIBLE;
      int i;
      for(i=0;i<Lists.length;i++)
      {
          printf("%s\n",Lists.elem[i].name);
      }
      return OK;
}
```

附录 B 基于链式存储结构线性表实现的源程序

```
                [language=c++,
                caption={链式线性表代码},
                label=code:Linklist_source,
                numbers=none,]
                #include "LinkList.h"
#include <cstddef>
#include <string>
#include <cstring>

status InitList(LinkList &L,int ElemNum)
// 线性表L不存在，构造一个空的线性表，返回OK，否则返回INFEASIBLE。
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(L) return INFEASIBLE;
    L = new LNode;
        L->next = NULL;
        LinkList tail = L;
        int cnt = 1 ;

        while (cnt <= ElemNum){
          LinkList P = new LNode;
          cin >> P->data;
          P->next = tail->next;
          tail->next = P;
          cnt++;
          tail = tail->next;
  }
    return OK;

    /********** End **********/
}

status DestroyList(LinkList &L)
// 如果线性表L存在，销毁线性表L，释放数据元素的空间，返回OK，否则返回INFEASIBLE。
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(!L) return INFEASIBLE;
    LinkList P = L->next;
    while (P) {
        L->next = P->next;
        delete P;
        P = L->next;
    }
    delete L;
```

```
    L=NULL;
    return OK;

    /********** End **********/
}


status ClearList(LinkList &L)
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(!L) return INFEASIBLE;
    LinkList P = L->next;
    while(P){
        L->next = P->next;
        delete P;
        P = L->next;
    }
    L->next = NULL;
    return OK;

    /********** End **********/
}

status ListEmpty(LinkList L)
// 如果线性表L存在，判断线性表L是否为空，空就返回TRUE，否则返回FALSE；如果线性表L不存在，返回INFEASIBLE。
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(!L) return INFEASIBLE;
    if(L->next) return FALSE;
    return TRUE;

    /********** End **********/
}

int ListLength(LinkList L)
// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(!L) return INFEASIBLE;
    int count = 0 ;
    LinkList P = L->next;
    while(P){
        P = P->next;
        count ++;
    }
    return count;
    /********** End **********/
```

```
}


status GetElem(LinkList L,int i,ElemType &e)
// 如果线性表L存在，获取线性表L的第i个元素，保存在e中，返回OK；如果i不合法，返回ERROR；如果线性表L不存在，返回
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(!L) return INFEASIBLE;
    if(i<1 || i > ListLength(L)) return ERROR;
    int cnt = 1;
    LinkList P = L->next;
    while(cnt < i){
        P = P->next;
        cnt++;
    }
    e = P->data;
    return OK;

    /********** End **********/
}

status LocateElem(LinkList L,ElemType e)
// 如果线性表L存在，查找元素e在线性表L中的位置序号；如果e不存在，返回ERROR；当线性表L不存在时，返回INFEASIBLE
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(!L) return INFEASIBLE;
    LinkList P = L->next;
    int cnt = 1;
    while(P){
        if(e == P->data) return cnt;
        cnt++;
        P = P->next;
    }
    return ERROR;

    /********** End **********/
}

status PriorElem(LinkList L,ElemType e,ElemType &pre)
// 如果线性表L存在，获取线性表L中元素e的前驱，保存在pre中，返回OK；如果没有前驱，返回ERROR；如果线性表L不存在
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(!L) return INFEASIBLE;
    LinkList Pre = L,P = L->next;
    while(P){
        if(P->data == e) break;
        Pre = P;
```

```
        P = P->next;
    }

    if(Pre == L || P == NULL)
        return ERROR;
    pre = Pre->data;
    return OK;
    /********** End **********/
}

status NextElem(LinkList L,ElemType e,ElemType &next)
// 如果线性表L存在，获取线性表L元素e的后继，保存在next中，返回OK；如果没有后继，返回ERROR；如果线性表L不存在，
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(!L) return INFEASIBLE;
    LinkList Next,P = L->next;
    while(P){
        Next = P->next;
        if(P->data == e) break;
        P = Next;
    }

    if(Next == NULL || P == NULL)
        return ERROR;
    next = Next->data;
    return OK;

    /********** End **********/
}

status ListInsert(LinkList &L,int i,ElemType e)
// 如果线性表L存在，将元素e插入到线性表L的第i个元素之前，返回OK；当插入位置不正确时，返回ERROR；如果线性表L不
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(!L) return INFEASIBLE;
    if(i<1) return ERROR;
    int cnt = 1;
    LinkList P = L->next,Pre = L;
    while(P && cnt < i){
        Pre = P;
        P = P->next;
        cnt++;
    }
    if(P == NULL && i != cnt) return ERROR;
    LinkList Q = new LNode;
    Q->data = e;
    Q->next = P;
    Pre->next  = Q;
```

```
    return OK;
    /********** End **********/
}

status ListDelete(LinkList &L,int i,ElemType &e)
// 如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回OK；当删除位置不正确时，返回ERROR；如果线性表L不
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(!L) return INFEASIBLE;
    if(i<1) return ERROR;
    LinkList P = L->next;
    LinkList Pre = L;
    int cnt = 1;
    while(P && cnt < i){
        Pre = P;
        P = P->next;
        cnt++;
    }
    if(P == NULL){
        return ERROR;
    }
    Pre -> next = P->next;
    e = P->data;
    delete P;
    return OK;
    /********** End **********/
}

status ListTraverse(LinkList L)
// 如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，返回OK；如果线性表L不存在，返回INFEASIBLE。
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(!L) return INFEASIBLE;
    LinkList P = L->next;
    while(P){
        printf("%d",P->data);
        putchar((P->next == NULL) ? '\n':' ');
        P=P->next;
    }
    return OK;
    /********** End **********/
}

status SaveList(LinkList L,const char FileName[])
// 如果线性表L存在，将线性表L的的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。
{
    // 请在这里补充代码，完成本关任务
    /********** Begin 1 *********/
```

```
    if(!L) return INFEASIBLE;
    FILE *fp;
    fp = fopen(FileName,"w");
    LinkList P = L->next;
    while(P){
        fprintf(fp,"%d",P->data);
        fputc((P->next == NULL)?'\n':' ',fp);
        P=P->next;
    }
    fclose(fp);
    return OK;

    /********** End 1 **********/
}

status LoadList(LinkList &L,const char FileName[]){
// 如果线性表L不存在，将FileName文件中的数据读入到线性表L中，返回OK，否则返回INFEASIBLE。 { 请在这里补充代码，
    if(L) return INFEASIBLE;
    InitList(L);
    LinkList P ,Pre = L;
        FILE *fp;
    fp= fopen(FileName,"r");
    while(!feof(fp)){
        P = new LNode;
        fscanf(fp,"%d",&P->data);
        Pre->next = P;
        Pre = Pre->next;
    }
    Pre->next = NULL;

    return OK;

    /********** End 2 **********/
}

status ReverseList(LinkList &L){
  if(!L) return INFEASIBLE;
  LinkList Pre = NULL;
  LinkList P = L->next;
  LinkList Next;
  while (P)
  {
        Next = P->next;
        P->next = Pre;
        Pre = P;
        P = Next;
  }
  L->next = Pre;
  return OK;
}
```

```
status RemoveNthFromEnd(LinkList &L,int n,ElemType &e){
  int cnt = 0;
  if(!L)return INFEASIBLE;
  if(ListLength(L) < n ) return FALSE;
  LinkList P = L->next,Pn = L->next,Pre = L;
  while(cnt < n){
        Pn = Pn->next;
        cnt++;
  }
  while(Pn){
        Pn = Pn->next;
        P = P->next;
        Pre = Pre->next;
  }
  Pre->next = P->next;
  e = P->data;
  delete P;
  return OK;
}
status SortList(LinkList &L){//升序
  if(!L) return INFEASIBLE;
  LinkList P=L->next,Q;
  for(;P->next;P = P->next){
        LinkList Min = P;
        for(Q = P->next;Q;Q = Q->next){
          if(Min->data > Q->data)
                Min = Q;
        }
        Swap(P,Min);
  }
  return OK;
}

void Swap(LinkList &L1,LinkList &L2){
  ElemType e = L1->data;
  L1->data = L2->data;
  L2->data = e;
}


status AddList(ArrList &Lists,char ListName[]){
  if(Lists.length>=20) return ERROR;
    strcpy(Lists.elem[Lists.length].Name,ListName);
    Lists.elem[Lists.length].L = NULL;
        int ElemNum;
        cout << "Please Input the ElemNum you wan't to have in this List, 0 means a nil List!\n" << endl;
        cin >> ElemNum;
        cout << "Now input the num you want to insert." << endl;
    InitList(Lists.elem[Lists.length].L,ElemNum);
        ListTraverse(Lists.elem[Lists.length].L);
```

```
    Lists.length++;
    return OK;
}


status RemoveList(ArrList &Lists,char ListName[],LinkList &L){
// Lists中删除一个名称为ListName的线性表

    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    int flag = 0;
    int i;
    for(i = 0 ; i<Lists.length ; i++){
        if(strcmp(ListName,Lists.elem[i].Name) == 0){
            DestroyList(Lists.elem[i].L);
            strcpy(Lists.elem[i].Name,"\0");
            flag++;
            break;
        }
    }
    if(flag == 1){
        for(int j = i ; j<=Lists.length-1 ; j++){
            Lists.elem[j] = Lists.elem[j+1];
        }
                Lists.length--;
                L = NULL;
        return OK;
    }
    return ERROR;
    /********** End **********/
}


int LocateList(ArrList Lists,char ListName[],LinkList &L){
        int i;
    for(i = 0 ; i<Lists.length ; i++){
        if(strcmp(ListName,Lists.elem[i].Name) == 0){
                  L=Lists.elem[i].L;
                  break;
          }
    }
    if(i!=Lists.length){
        return i+1;
    }
    return 0;
}

status Save2List(ArrList &Lists, char ListName[], LinkList L){

    strcpy(Lists.elem[Lists.length].Name,ListName);
```

```
    Lists.elem[Lists.length].L = L;
        Lists.length++;
        return OK;
}
```

附录 C 基于二叉链表二叉树实现的源程序

```
                [language=c++,
                caption={二叉树代码},
                label=code:BiTree_source,
                numbers=none,]
                #include "def.h"
using namespace std;
int cnt = 0;
status CreateBiTree(BiTree &T, TElemType definition[])
{
    for (int i = 0; definition[i].key != -1; i++)
    {
        for (int j = i + 1; definition[j].key != -1; j++)
        {
            if (definition[i].key > 0 && definition[i].key == definition[j].key)
                return ERROR;
        }
    }
    BiTree Root;
    CreateBiNode(Root, definition);
    T = Root;
    cnt = 0;
    return OK;
}
void CreateBiNode(BiTree &Root, TElemType definition[])
{
    if (definition[cnt].key == 0)
    {
        Root = NULL;
        cnt++;
        return;
    }
    else if (definition[cnt].key == -1)
    {
        return;
    }

    Root = new BiTNode;
    Root->data = definition[cnt++];
    CreateBiNode(Root->lchild, definition);
    CreateBiNode(Root->rchild, definition);
}
status ClearBiTree(BiTree &T)
{
    if (!T)
        return OK;
    if (ClearBiTree(T->lchild) && ClearBiTree(T->rchild))
```

```
    {
        delete T;
    }
    T = NULL;
    return OK;
}

status isEmpty(BiTree T)
{
    if (T == NULL)
        return TRUE;
    return FALSE;
}

int BiTreeDepth(BiTree T)
{
    if (!T)
        return 0;
    return 1 + max<int>(BiTreeDepth(T->lchild), BiTreeDepth(T->rchild));
}

int BiTreeHeight(BiTree T, BiTree p)
{
    //求结点到根结点的高度
    if (!T)
        return 0;
    int ans = 0;
    while (p != T)
    {
        ans++;
        p = Search(T, p);
    }
    return ans;
}
BiTNode *LocateNode(BiTree T, KeyType e)
{
    BiTree ans;
    if (T == NULL)
        return NULL;
    if (e == T->data.key)
    {
        return T;
    }
    else
        return (ans = LocateNode(T->lchild, e)) != NULL ? ans : LocateNode(T->rchild, e);
}
status PreOrderCheck(BiTree T1, BiTree T2)
{
    if (T1 == NULL)
        return OK;
```

```
    if (T2->data.key == T1->data.key && T2 != T1)
        return ERROR;
    return PreOrderCheck(T1->lchild, T2) && PreOrderCheck(T1->rchild, T2);
}
status Assign(BiTree &T, KeyType e, TElemType value)
{
    BiTree exc = LocateNode(T, e);
    if (!exc)
        return ERROR;
    exc->data = value;
    if (PreOrderCheck(T, exc) == ERROR)
        return ERROR;
    return OK;
}
BiTNode *GetSibling(BiTree T, KeyType e)
{
    if (T && T->data.key == e || !T->lchild || !T->rchild)
        return NULL;
    if (T->lchild->data.key == e)
        return T->rchild;
    if (T->rchild->data.key == e)
        return T->lchild;
    BiTree ans;
    return ((ans = GetSibling(T->lchild, e)) != NULL) ? ans : GetSibling(T->rchild, e);
}
status InsertNode(BiTree &T, KeyType e, int LR, TElemType c)
{
    if (LR == -1)
    {
        BiTree NewRoot = new BiTNode;
        NewRoot->data = c;
        NewRoot->lchild = NULL;
        NewRoot->rchild = T;
        T = NewRoot;
        return OK;
    }
    BiTree eNode = LocateNode(T, e);
    if (eNode == nullptr)
        return ERROR;
    BiTree NewNode = new BiTNode;
    NewNode->data = c;
    NewNode->lchild = nullptr;
    if (LR == 0)
    {
        NewNode->rchild = eNode->lchild;
        eNode->lchild = NewNode;
    }
    else
    {
        NewNode->rchild = eNode->rchild;
```

```
            eNode->rchild = NewNode;
        }
        if (PreOrderCheck(T, NewNode) == ERROR)
            return ERROR;
        return OK;
    }
    int flag = -1;
    BiTree Search(BiTree T, BiTree eNode)
    {
        if (!T)
            return NULL;
        if (T->lchild == eNode)
        {
            flag = 0;
            return T;
        }
        if (T->rchild == eNode)
        {
            flag = 1;
            return T;
        }
        BiTree ans;
        return (ans = Search(T->lchild, eNode)) != NULL ? ans : Search(T->rchild, eNode);
    }
    BiTree FindMax(BiTree T)
    {
        if (T->rchild == NULL)
            return T;
        return FindMax(T->rchild);
    }
    status DeleteNode(BiTree &T, KeyType e)
    {
        BiTree eNode = LocateNode(T, e);
        if (eNode == T)
        {
            if (!T->lchild && !T->rchild)
            {
                free(T);
                return OK;
            }
            BiTree Mem;
            if (!T->lchild)
            {
                Mem = T->rchild;
                free(T);
                T = Mem;
                return OK;
            }
            if (!T->rchild)
            {
```

```
            Mem = T->lchild;
            free(T);
            T = Mem;
            return OK;
        }
        Mem = T->lchild;
        BiTree Replacer = FindMax(T->lchild);
        Replacer->rchild = T->rchild;
        free(T);
        T = Mem;
        return OK;
    }
    if (eNode == NULL)
        return ERROR;
    BiTree Parents = Search(T, eNode);
    if (!eNode->lchild && !eNode->rchild)
    {
        if (flag == 0)
        {
            Parents->lchild = NULL;
            free(eNode);
        }
        if (flag == 1)
        {
            Parents->rchild = NULL;
            free(eNode);
        }
        return OK;
    }

    if (!eNode->lchild)
    {
        if (flag == 0)
        {
            Parents->lchild = eNode->rchild;
            free(eNode);
        }
        if (flag == 1)
        {
            Parents->rchild = eNode->rchild;
            free(eNode);
        }
        return OK;
    }
    if (!eNode->rchild)
    {
        if (flag == 0)
        {
            Parents->lchild = eNode->lchild;
            free(eNode);
```

```
        }
        if (flag == 1)
        {
            Parents->rchild = eNode->lchild;
            free(eNode);
        }
        return OK;
    }
    BiTree Replacer = FindMax(eNode->lchild);
    Replacer->rchild = eNode->rchild;
    BiTree eNodeLeft = eNode->lchild;
    free(eNode);
    if (flag == 0)
    {
        Parents->lchild = eNodeLeft;
    }
    if (flag == 1)
    {
        Parents->rchild = eNodeLeft;
    }

    return OK;
}
status PreOrderTraverse(BiTree T, void (*visit)(BiTree))
{
    if (T)
    {
        visit(T);
        PreOrderTraverse(T->lchild, visit);
        PreOrderTraverse(T->rchild, visit);
    }
    return OK;
}
status InOrderTraverse(BiTree T, void (*visit)(BiTree))
{
    if (T)
    {
        InOrderTraverse(T->lchild, visit);
        visit(T);
        InOrderTraverse(T->rchild, visit);
    }
    return OK;
}
status PostOrderTraverse(BiTree T, void (*visit)(BiTree))
{
    BiTree stack[1000];
    int top = 0;
    BiTree r = NULL;
    BiTree p = T;
    while (p || top > 0)
```

```
    {
        if (p)
        {
            stack[top++] = p;
            p = p->lchild;
        }
        else
        {
            p = stack[top - 1];
            if (p->rchild && p->rchild != r)
            {
                p = p->rchild;
            }
            else
            {
                p = stack[--top];
                visit(p);
                r = p;
                p = NULL;
            }
        }
    }
    return OK;
}
void visit(BiTree T)
{
    printf(" %d,%s", T->data.key, T->data.others);
}
BiTree queue[1000];
int head = 0, rear = 0;
status LevelOrderTraverse(BiTree T, void (*visit)(BiTree))
{
    visit(T);

    if (T->lchild)
        queue[rear++] = T->lchild;
    if (T->rchild)
        queue[rear++] = T->rchild;

    if (head < rear)
        LevelOrderTraverse(queue[head++], visit);
    return OK;
}
void SaveBiTreeRec(BiTree T, FILE *f)
{
    if (!T)
    {
        fprintf(f, "0 null\n");
    }
    else
```

```
    {
        fprintf(f, "%d %s\n", T->data.key, T->data.others);
        SaveBiTreeRec(T->lchild, f);
        SaveBiTreeRec(T->rchild, f);
    }
}
status SaveBiTree(BiTree T, char FileName[])
{
    FILE *f = fopen(FileName, "w");
    SaveBiTreeRec(T, f);
    fclose(f);
    return OK;
}
BiTree LoadBiTreeRec(FILE *f)
{
    BiTree root;
    TElemType e;
    fscanf(f, "%d %s", &e.key, e.others);
    if (e.key <= 0)
    {
        root = NULL;
    }
    else
    {
        root = (BiTree)malloc(sizeof(BiTNode));
        root->data = e;
        root->lchild = LoadBiTreeRec(f);
        root->rchild = LoadBiTreeRec(f);
    }
    return root;
}
status LoadBiTree(BiTree &T, char FileName[])
{
    FILE *f = fopen(FileName, "r");
    T = LoadBiTreeRec(f);
    fclose(f);
    return OK;
}
BiTree LowestCommonAncestor(BiTree T, BiTree e1, BiTree e2)
{
    int h1 = BiTreeHeight(T,e1);
    int h2 = BiTreeHeight(T,e2);
    int ht = 0;
    if (h1 == ht)
        return e1;
    if (h2 == ht)
        return e2;
    return TestAncestor(e1, h1, e2, h2, T, ht);
}
```

```
BiTree TestAncestor(BiTree T1, int h1, BiTree T2, int h2, BiTree T, int ht)
{
    if (h2 < h1)
    {
        BiTree TMP;
        int tmph;
        TMP = T1;
        T1 = T2;
        T2 = TMP;
        tmph = h1;
        h1 = h2;
        h2 = tmph;
    }
    if (h1 == ht)
        return T1;
    if (h1 == h2 && T1 == T2)
        return T1;
    if (h1 == h2)
        return TestAncestor(Search(T, T1), h1 - 1 , Search(T, T2), h2 - 1 , T, ht);
    return TestAncestor(T1, h1, Search(T, T2), h2 - 1, T, ht);
}
int TriMax(int i, int j, int k)
{
    return (i >= j ? i : j) >= k ? (i >= j ? i : j) : k;
}
int MaxPathLength(BiTree T)
{
    if (T == nullptr)
        return 0;
    T->Sum = T->data.key + TriMax(0, MaxPathLength(T->lchild), MaxPathLength(T->rchild));
    return T->Sum;
    //
}
KeyType MaxPathSum(BiTree T)
{
    KeyType solution = 0;
    dfs(T, solution);
    return solution;
}

KeyType dfs(BiTree T, KeyType &solution)
{
    if (!T)
        return 0;
    KeyType Left = dfs(T->lchild, solution);
    KeyType Right = dfs(T->rchild, solution);
    solution = max<KeyType>(solution, T->data.key + max<KeyType>(0, Left) + max<KeyType>(0, Right));
    // 更新结论为当前结点的最大路径值
    return T->data.key + max<KeyType>(Left, Right);
    // 传回单边最大
```

```
}
status InverseBiTree(BiTree &T)
{
   if(!T) return OK;
   BiTree TMP = T->lchild;
   T->lchild = T->rchild;
   T->rchild = TMP;
   InverseBiTree(T->lchild);
   InverseBiTree(T->rchild);
   return OK;
}

status Add_BiTree_to_Forest(BiForest &F, char FileName[], TElemType definitions[])
{
    if (F.length == 20)
        return ERROR;
    strcpy(F.elem[F.length].Name, FileName);
    if (CreateBiTree(F.elem[F.length].T, definitions) == ERROR)
        return ERROR;
    F.length++;
    return OK;
}
status Remove_BiTree_from_Forest(BiForest &F, char FileName[])
{
    if (F.length == 0)
        return ERROR;
    for (int i = 0; i < F.length; i++)
    {
        if (strcmp(FileName, F.elem[i].Name) == 0)
        {
            for (int j = i + 1; j < F.length; j++)
            {
                F.elem[j - 1] = F.elem[j];
            }
            F.length--;
            break;
        }
        if (i == F.length - 1)
            return ERROR;
    }
    return OK;
}
status Locate_And_Modify_a_Tree_in_Forest(BiForest F, char FileName[], BiTree &T)
{
    for (int i = 0; i < F.length; i++)
    {
        if (strcmp(FileName, F.elem[i].Name) == 0)
        {
            T = F.elem[i].T;
            return OK;
```

```
        }
    }
    return ERROR;
}

status Traverse_Forest_Names(BiForest F)
{
    for (int i = 0; i < F.length; i++)
    {
        printf("%s\n", F.elem[i].Name);
    }
    return OK;
}

status Auto_Add_a_BiTree(BiForest &F,BiTree T){
    //自动存储树到森林里，每次树名有自己的序号
    char FileName[20];
    sprintf(FileName,"Auto_Saved_Tree%d",F.length);
    F.elem[F.length].T=T;
    strcpy(F.elem[F.length].Name,FileName);
    F.length++;
    return OK;
}

int Index_of_Tree(BiForest F,char FileName[]){
    for(int i=0;i<F.length;i++){
        if(strcmp(F.elem[i].Name,FileName)==0){
            return i;
        }
    }
    return -1;
}
```

附录 D 基于邻接表图实现的源程序

```
                [language=c++,
                caption={图代码},
                label=code:Graph_source,
                numbers=none,]
                #include "Graph.h"

// Path: Graph.h

status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2])
    /*根据V和VR构造图T并返回OK，如果V和VR不正确，返回ERROR
如果有相同的关键字，返回ERROR。此题允许通过增加其它函数辅助实现本关任务*/
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    G.kind = DN;
    int i,k;
    int index;
    int flag1 = 0,flag2 = 0;
    for(i = 0; V[i].key != -1 ; i++){
        for(int j = i+1 ; V[j].key != -1 ; j++){
            if(V[i].key == V[j].key) return ERROR;
            if(strcmp(V[i].others,V[j].others) == 0) return ERROR;
        }
    }
    if(i > MAX_VERTEX_NUM) return ERROR;
    else G.vexnum = i;
    for(i = 0 ; VR[i][0] != -1 && VR[i][1] != -1; i++){
        if(VR[i][0] == VR[i][1]) return ERROR;
        for(int j = 0; V[j].key != -1 ; j++){
            if(VR[i][0] == V[j].key) flag1 = 1;
            if(VR[i][1] == V[j].key) flag2 = 1;
        }
        if(flag1 == 0 || flag2 == 0) return ERROR;
        flag1 = flag2 = 0;
        for(int k = i+1 ; VR[k][0]!= -1 && VR[k][1] != -1 ; k++){
            if(VR[i][0] == VR[k][0] && VR[i][1] == VR[k][1] || VR[i][0] == VR[k][1] && VR[i][1] == VR[k][0])
        }
    }//O(n^2)
    G.arcnum = i;
    if(G.vexnum == 0) return ERROR;
    for(i = 0 ; i < G.vexnum ; i++){
        G.vertices[i].data.key = V[i].key;
        //Map[V[i].key] = i;
        G.vertices[i].firstarc = NULL;
        strcpy(G.vertices[i].data.others,V[i].others);
    }
```

```
    for(i = 0 ; i<G.vexnum ; i++){
        for(int j = 0; j < G.arcnum ; j++){
            if(V[i].key == VR[j][0]){
                //search(G,VR[j][1])
                for(k = 0; k < G.vexnum ; k++){
                    if(G.vertices[k].data.key == VR[j][1]){
                        index = k;
                        break;
                    }
                }
                if(k == G.vexnum) continue;
                struct ArcNode * A = (struct ArcNode *)malloc(sizeof(struct ArcNode));
                A->adjvex = index;
                A->nextarc = G.vertices[i].firstarc;
                G.vertices[i].firstarc = A;
            }
            if(V[i].key == VR[j][1]){
                for(k = 0; k < G.vexnum ; k++){
                    if(G.vertices[k].data.key == VR[j][0]){
                        index = k;
                        break;
                    }
                }
                if(k == G.vexnum) continue;
                struct ArcNode * A = (struct ArcNode *)malloc(sizeof(struct ArcNode));
                A->adjvex = index;
                A->nextarc = G.vertices[i].firstarc;
                G.vertices[i].firstarc = A;
            }
        }
    }
    return OK;

    /********** End **********/
}


status DestroyGraph(ALGraph &G){
    int i;
    ArcNode *p,*q;
    for(i=0;i<G.vexnum;i++){
        p=G.vertices[i].firstarc;
        while(p){
            q=p->nextarc;
            delete p;
            p=q;
        }
    }
    G.vexnum=0;
    G.arcnum=0;
```

```
    return OK;
}

int LocateVex(ALGraph G,KeyType u){
    int i;
    for(i=0;i<G.vexnum;i++){
        if(G.vertices[i].data.key==u)
            return i;
    }
    return -1;
}

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



int FirstAdjVex(ALGraph G,KeyType u){
    int location;
    ArcNode *p;
    if((location = LocateVex(G,u)) == -1) return -1;
    p = G.vertices[location].firstarc;
    if(p) return p->adjvex;
    else return -1;
}

int NextAdjVex(ALGraph G,KeyType v,KeyType w)
//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
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    int location;
    if((location = LocateVex(G,v)) == -1) return -1;
    ArcNode * P = G.vertices[location].firstarc;
    while(P){
        if(G.vertices[P->adjvex].data.key == w)
            if(P->nextarc) return P->nextarc->adjvex;
            else return -1;

        P = P->nextarc;
    }
    return -1;
```

```
    /********** End **********/
}

status InsertVex(ALGraph &G,VertexType v)
//在图G中插入顶点v，成功返回OK,否则返回ERROR
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(G.vexnum == MAX_VERTEX_NUM) return ERROR;
    if(LocateVex(G,v.key) != -1) return ERROR;
    G.vertices[G.vexnum].firstarc = nullptr;
    G.vertices[G.vexnum++].data = v;
    return OK;
//可以初始化连线
    /********** End **********/
}


status DeleteVex(ALGraph &G,KeyType v)
//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
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    int location = LocateVex(G, v);
    if(location == -1) return ERROR;
    ArcNode *p = nullptr,*q = nullptr;
    p = G.vertices[location].firstarc;
    while(p){
        q = p->nextarc;
        free(p);
        p = q;
        G.arcnum--;
    }//
    for(int j = location+1 ; j < G.vexnum ; j++){
        G.vertices[j-1] = G.vertices[j];
    }
    G.vexnum--;//
    if(G.vexnum == 0) return ERROR;
    for (int k = 0 ; k < G.vexnum ; k++){
        p = G.vertices[k].firstarc;
        if(p && p->adjvex == location) {
            q = p->nextarc;
            free(p);
            G.vertices[k].firstarc = q;
            p = q;
        }
        q = p;
        while(q){
            if(q->adjvex > location){
                q->adjvex--;
```

```
                p = q;
                q = q->nextarc;
            }
            else if(q->adjvex < location){
                p = q;
                q = q->nextarc;
            }
            else{
                p->nextarc= q->nextarc;
                free(q);
                q = p->nextarc;
            }
        }
    }
    return OK;
    /********** End **********/
}

status InsertArc(ALGraph &G,KeyType v,KeyType w)
//在图G中增加弧<v,w>，成功返回OK,否则返回ERROR
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    int location1 = LocateVex(G,v);
    if(location1 == -1) return ERROR;
    int location2 = LocateVex(G,w);
    if(location2 == -1) return ERROR;
    ArcNode *P = G.vertices[location1].firstarc;
    while(P){
        if(P->adjvex == location2) return ERROR;
        P = P->nextarc;
    }
    ArcNode * P1 = (ArcNode *)malloc(sizeof(ArcNode)), *P2 = (ArcNode *)malloc(sizeof(ArcNode));
    P1->adjvex = location2;
    P1->nextarc = G.vertices[location1].firstarc;
    P2->adjvex = location1;
    P2->nextarc = G.vertices[location2].firstarc;
    G.vertices[location1].firstarc = P1;
    G.vertices[location2].firstarc = P2;
    G.arcnum++;
    return OK;
    /********** End **********/
}

status DeleteArc(ALGraph &G,KeyType v,KeyType w)
//在图G中删除弧<v,w>，成功返回OK,否则返回ERROR
{
    int location1 = LocateVex(G,v);
    if(location1 == -1) return ERROR;
    int location2 = LocateVex(G,w);
```

```
if(location2 == -1) return ERROR;
ArcNode *P = G.vertices[location1].firstarc;
int cnt = 0;
ArcNode * Pre;
if(P->adjvex == location2){
    cnt = 2;
}
else{
    while(P){
        if(P->nextarc && P->nextarc->adjvex == location2) {
            cnt = 1;
            Pre = P;
            P = P->nextarc;
            break;
        }
        P=P->nextarc;
    }
}
if(cnt == 0) return ERROR;
if(cnt == 1){
    Pre->nextarc = Pre->nextarc->nextarc;
    free(P);
}
if(cnt == 2){
    G.vertices[location1].firstarc = G.vertices[location1].firstarc->nextarc;
    free(P);
}

P = G.vertices[location2].firstarc;
cnt = 0;
Pre = P;
if(P->adjvex == location1){
    cnt = 2;
}
else{
    while(P){
        if(P->nextarc && P->nextarc->adjvex == location1) {
            cnt = 1;
            Pre = P;
            P = P->nextarc;
            break;
        }

        P=P->nextarc;
    }
}
if(cnt == 0) return ERROR;
if(cnt == 1){
    Pre->nextarc = Pre->nextarc->nextarc;
    free(P);
```

```
    }
    if(cnt == 2){
        G.vertices[location2].firstarc = G.vertices[location2].firstarc->nextarc;
        free(P);
    }

    G.arcnum--;
    return OK;

    /********** End **********/
}



status DFSTraverse(ALGraph &G,void (*visit)(VertexType))
//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
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    int i = 0;
    if(!G.vexnum) return ERROR;
    int visited[1000] = {0};
    for(; i < G.vexnum ; i++){
        if(visited[i] == 0){
            DFS(G,G.vertices[i],visited,i,visit);
        }
    }
    return OK;
    /********** End **********/
}

void DFS(ALGraph &G,VNode V, int (&visited)[1000], int i,void (*visit)(VertexType)){
    visited[i] = 1;
    visit(V.data);
    ArcNode * next = V.firstarc;
    while(next){
        if(!visited[next->adjvex])
            DFS(G,G.vertices[next->adjvex],visited,next->adjvex,visit);
            next = next->nextarc;
    }

}

status BFSTraverse(ALGraph &G,void (*visit)(VertexType))
//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
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    VNode VQueue[1000];
    int head = 0,rear = 0;
```

```
    bool isvisited[1000] = {false};
    VNode Vnow;
    for(int i = 0 ; i < G.vexnum ; i++){
        if(isvisited[i] == 0){
            isvisited[i] = 1;
            VQueue[rear++] = G.vertices[i];
        }
        while(head < rear){
            Vnow = VQueue[head++];
            visit(Vnow.data);
            ArcNode * next = Vnow.firstarc;
            while(next){
                if(!isvisited[next->adjvex]){
                    isvisited[next->adjvex] = true;
                    VQueue[rear++] = G.vertices[next->adjvex];
                }
                next = next->nextarc;
            }

        }
    }
    return OK;
    /********** End **********/
}

status SaveGraph(ALGraph G, char FileName[])
//将图的数据写入到文件FileName中
{
    // 请在这里补充代码，完成本关任务
    /********** Begin 1 *********/
    FILE * fp = fopen(FileName,"w");
    fprintf(fp,"%d %d %d ",G.vexnum,G.arcnum,G.kind);
    for(int i = 0 ; i < G.vexnum ; i++){
        fprintf(fp,"%d %s ",G.vertices[i].data.key,G.vertices[i].data.others);
        ArcNode * P = G.vertices[i].firstarc;
        while(P){
            fprintf(fp,"%d ",P->adjvex);
            P = P->nextarc;
        }
        fprintf(fp,"%d ",-1);
    }
    fclose(fp);
    return OK;
    /********** End 1 **********/
}

status LoadGraph(ALGraph &G, char FileName[])
//读入文件FileName的图数据，创建图的邻接表
{
    // 请在这里补充代码，完成本关任务
```

```
    /********** Begin 2 *********/
    FILE * fp = fopen(FileName,"r");
    if(!fp) return ERROR;
    fscanf(fp,"%d %d %d ",&G.vexnum,&G.arcnum,&G.kind);
    int i = 0;
    while(i<G.vexnum){
        fscanf(fp,"%d %s ", &G.vertices[i].data.key,G.vertices[i].data.others);
        G.vertices[i].firstarc = nullptr;
        int adj;
        ArcNode * P, * Q;
        fscanf(fp,"%d ",&adj);
        if(adj!=-1){
            Q = (ArcNode *)malloc(sizeof(ArcNode));
            Q->adjvex = adj;
            Q->nextarc = nullptr;
            G.vertices[i].firstarc = Q;
            while(fscanf(fp,"%d ",&adj)&&adj != -1){
                P = Q;
                Q = (ArcNode *)malloc(sizeof(ArcNode));
                Q->adjvex = adj;
                Q->nextarc = nullptr;
                P->nextarc = Q;
            }
        }
        i++;
    }
    fclose(fp);
    return OK;
    /********** End 2 **********/
}

int * Path_Length_to_v(ALGraph G,KeyType v){
    int * path = (int *)malloc(sizeof(int)*G.vexnum);
    for(int i = 0 ; i<G.vexnum; i++){
        path[i] = -1;
    }
    int location = LocateVex(G,v);
    path[location] = 0;
    DFS_LEN(G,location,0,path);
    return path;
}

void DFS_LEN(ALGraph G,int location,int len,int * &path){//通过DFS计算每个点到v的距离
//如果未访问或原长度过大，则重新分配
    ArcNode * P = G.vertices[location].firstarc;
    while(P){
        if(path[P->adjvex] == -1 || path[P->adjvex] > len+1){
            path[P->adjvex] = len+1;
            DFS_LEN(G,P->adjvex,len+1,path);
        }
```

```
        P = P->nextarc;
    }
}

int * VerticesSetLessThanK(ALGraph &G,KeyType v,int k){
    int location = LocateVex(G,v);
    if(location == -1) return nullptr;
    int * path = Path_Length_to_v(G,v);
    int * ans = (int *)malloc(sizeof(int)*G.vexnum);
    for(int i = 0 ; i < G.vexnum ; i++){
        if(path[i] < k && path[i] != -1){
            ans[i] = 1;
        }
        else ans[i] = 0;
    }
    return ans;
}


int ShortestPathLength(ALGraph G, KeyType v,KeyType w){//求无权图G中从顶点v到顶点w的最短路径长度
    if(G.kind == DG){
        int *path1 = Path_Length_to_v(G,v);
        int *path2 = Path_Length_to_v(G,w);
        int location1 = LocateVex(G,v);
        int location2 = LocateVex(G,w);
        return path1[location2] > path2[location1] ? path2[location1] : path1[location2];
    }
    else{
        int * path = Path_Length_to_v(G,v);
        int location = LocateVex(G,w);
        return path[location];
    }
}

int DFSCountCCP(ALGraph G){
//对图G进行深度优先搜索遍历 返回联通分量个数
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    int index = 0;
    int visited[1000] = {false};
    if(!G.vexnum) return ERROR;
    for(int i = 0 ; i<G.vexnum ; i++){
        if(visited[i] == false){
            DFSCount(G,G.vertices[i],visited,i);
            index++;
        }
    }
    return index;
    /********** End **********/
}
```

```
void DFSCount(ALGraph G,VNode V, int (&visited)[1000], int i){
    visited[i] = 1;
    ArcNode * next = V.firstarc;
    while(next){
        if(!visited[next->adjvex])
            DFSCount(G,G.vertices[next->adjvex],visited,next->adjvex);
            next = next->nextarc;
    }

}//返回树的数量

int ConnectedComponentsNums(ALGraph G){
    if(G.kind == DG) return Strongly_ConnectedComponentsNums(G);
    else{
        return DFSCountCCP(G);
    }
}

int Strongly_ConnectedComponentsNums(ALGraph G){
    //Use Two times DFS
    //DFS(G)
    //Reverse G
    //DFS(G^T) in decreasing order of finish time
    //Count the number of trees in DFS forest
    return 0;//undone
}

int * TopologicalSort(ALGraph G){
    if(G.kind == UDG) return nullptr;
    if(G.kind == DG) return nullptr;
    if(G.kind == UDN) return nullptr;
    if(G.kind == DN) {
        int * ans = (int *)malloc(sizeof(int)*G.vexnum);
        int * inDegree = (int *)malloc(sizeof(int)*G.vexnum);
        for(int i = 0 ; i < G.vexnum ; i++){
            inDegree[i] = 0;
        }
        for(int i = 0 ; i < G.vexnum ; i++){
            ArcNode * P = G.vertices[i].firstarc;
            while(P){
                inDegree[P->adjvex]++;
                P = P->nextarc;
            }
        }
        int index = 0;
        while(index < G.vexnum){
            for(int i = 0 ; i < G.vexnum ; i++){//Please Use Heap
                if(inDegree[i] == 0){
                    ans[index++] = i;
```

```
                    inDegree[i] = -1;
                    ArcNode * P = G.vertices[i].firstarc;
                    while(P){
                        inDegree[P->adjvex]--;
                        P = P->nextarc;
                    }
                }
            }
        }
        return ans;
    }
}

void visit(VertexType v)
{
    printf(" %d %s",v.key,v.others);
}

status Add_GRAPHS(GRAPHS &P,char FileName[]){
    if(P.length == 20) return ERROR;
    strcpy(P.elem[P.length].name,FileName);
    VertexType V[1000];
    KeyType VR[1000][2];
    int i = 0;
    cout << "Please Input the Vertexs' Information of the Graphs" << endl;
    while(1){
        KeyType key;
        char others[20];
        cin >> key;
        cin >> others;
        V[i].key = key;
        strcpy(V[i].others,others);
        if(key == -1) break;
        i++;
    }
    i = 0;
    cout << "Please Input the Edges of the Graphs" << endl;
    while(1){
        KeyType key1,key2;
        cin >> key1;
        cin >> key2;
        VR[i][0] = key1;
        VR[i][1] = key2;
        if(key1 == -1 || key2 == -1) break;
        i++;
    }
    if(CreateGraph(P.elem[P.length].G,V,VR)==ERROR) return ERROR;
    P.length++;
    return OK;
}
```

```
status Remove_GRAPHS(GRAPHS &P,char FileName[]){
    if(P.length == 0) return ERROR;
    for(int i = 0 ; i < P.length ; i++){
        if(strcmp(P.elem[i].name,FileName) == 0){
            for(int j = i ; j < P.length-1 ; j++){
                P.elem[j] = P.elem[j+1];
            }
            P.length--;
            return OK;
        }
    }
    return ERROR;
}

status Locate_and_Modi_GRAPHS(GRAPHS &P,ALGraph &G,char FileName[]){
    for(int i = 0 ; i <P.length ; i++){
        if(strcmp(P.elem[i].name,FileName) == 0){
            G = P.elem[i].G;
            return OK;
        }
    }
    return ERROR;
}

status Traverse_GRAPHS(GRAPHS &P){
    for(int i = 0 ; i < P.length ; i++){
        cout << P.elem[i].name << endl;
    }
    return OK;
}
```