# 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** CS2104

**学　　号** U202115434

**姓　　名** 张家豪

**指导教师** 袁凌

**报告日期** 2021 年 6 月 10 日

**计算机科学与技术学院**

# 目　　录

# 1 基于顺序存储结构的线性表实现

## 1.1 实验目的

通过实验达到:

(1) 加深对线性表的概念、基本运算的理解;

(2) 熟练掌握线性表的逻辑结构与物理结构的关系;

(3) 物理结构采用顺序表, 熟练掌握顺序表基本运算的实现。

### 1.1.1 线性表抽象数据类型

依据最小完备性和常用性相结合的原则, 设计了线性表的数据对象和数据关系, 并以函数形式定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等 12 种基本运算, 具体运算功能定义如下:

(1) 初始化表:

InitList(L):

—-初始条件: 线性表 L 不存在;

—-操作结果: 构造一个空的线性表;

(2) 销毁表:

DestroyList(L);

—-初始条件: 线性表 L 已存在;

—-操作结果: 销毁线性表 L;

(3) 清空表:

ClearList(L);

—-初始条件: 线性表 L 已存在;

—-操作结果: 将 L 重置为空表;

(4) 判定空表:

ListEmpty(L);

—-初始条件: 线性表 L 已存在;

—-操作结果: 若 L 为空表则返回 TRUE, 否则返回 FALSE;

(5) 求表长:

ListLength(L);

—-初始条件: 线性表已存在;

—-操作结果: 返回 L 中数据元素的个数;

(6) 获得元素:

GetElem(L,i,e);

—-初始条件: 线性表已存在,1≤i≤ListLength(L);

—-操作结果: 用 e 返回 L 中第 i 个数据元素的值;

(7) 查找元素:

LocateElem(L,e,compare());

—-初始条件: 线性表已存在;

—-操作结果: 返回 L 中第 1 个与 e 满足关系 compare() 关系的数据元素的位序, 若这样的数据元素不存在, 则返回值为 0;

(8) 获得前驱:

PriorElem(L,cur_e,pre_e);

—-初始条件: 线性表 L 已存在;

—-操作结果: 若 cur_e 是 L 的数据元素, 且不是第一个, 则用 pre_e 返回它的前驱, 否则操作失败,pre_e 无定义;

(9) 获得后继:

NextElem(L,cur_e,next_e);

—-初始条件: 线性表 L 已存在;

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

(10) 插入元素:

ListInsert(L,i,e);

—-初始条件: 线性表 L 已存在,1≤i≤ListLength(L)+1;

—-操作结果: 在 L 的第 i 个位置之前插入新的数据元素 e。

(11) 删除元素:

ListDelete(L,i,e);

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

—-操作结果: 删除 L 的第 i 个数据元素, 用 e 返回其值;

(13) 遍历表:

ListTraverse(L,visit()),

—-初始条件: 线性表 L 已存在;

—-操作结果: 依次对 L 的每个数据元素调用函数 visit()。

附加功能:

(1) 最大连续子数组和:

MaxSubArray(L);

—-初始条件: 线性表 L 已存在且非空, 请找出一个具有最大和的连续子数组(子数组最少包含一个元素),

—-操作结果: 其最大和;

(2) 和为 K 的子数组:

SubArrayNum(L,k);

—-初始条件: 线性表 L 已存在且非空,

—-操作结果: 该数组中和为 k 的连续子数组的个数;

(3) 顺序表排序:

sortList(L);

—-初始条件: 线性表 L 已存在;

—-操作结果: 将 L 由小到大排序;

### 1.1.2 线性表的文件形式保存

1. 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. 设计了线性表文件保存和加载操作的合理模式。附录 B 提供了文件存取的具体方法。

### 1.1.3 实现多个线性表管理

在整个链表之外设计包装链表, 将已经存在的所有线性表用链式结构串起, 对每个线性表赋予独特 ID 标识, 在每次操作中通过 ID 选择要进行操作的链表。

## 1.2 系统设计

### 1.2.1 数据结构设计

1. 线性表的存储数据结构

结构体定义如下:

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

2. 多线性表的存储数据结构

结构体定义如下:

```
typedef struct
{ //线性表的集合类型定义
  struct
  {
    char name[30];
    SqList L;
  } elem[10];
  int length;
} LISTS;
```

在本程序中, 数据原子类型被定义为整型 int。

### 1.2.2 演示系统设计

包括用户操作界面和功能调用两个部分。

演示系统语言为英文, 所有操作和提示语言均为中文。

用户操作界面输出可选的线性表操作, 用户输入数字选择要进行的操作。系统提示用户输入参数。

功能调用部分则将用户输入的有关信息传递给线性数据结构的操作函数进行调用, 并对函数返回值进行处理判断输出相应提示信息。

## 1.3 系统实现

### 1.3.1 线性表运算算法实现

1.status InitList(SqList &L);

**功能:** 初始化线性表。

**算法实现:** 为线性表 L 的 data 申请空间, 申请失败返回 ERROR, 否则返回成功。

**时空效率:** 时间复杂度为 O(1), 空间复杂度为 O(1)。

2.status DestroyList(SqList &L);

**功能:** 销毁线性表。

**算法实现:** 如果线性表 L 存在,free 掉 data 空间并设置为 null, 将其他字段设置为 0, 返回 OK, 否则返回 INFEASIBLE。

**时空效率:** 时间复杂度为 O(1), 空间复杂度为 O(1)。

3.status ClearList(SqList &L);

**功能:** 清空线性表。

**算法实现:** 如果线性表 L 存在, 将 data 区域占有的内存空间释放并分配新内存空间, 返回 OK, 否则返回 INFEASIBLE。

**时空效率:** 时间复杂度为 O(1), 空间复杂度为 O(1)。

4.status ListEmpty(SqList L);

**功能:** 线性表判空。

**算法实现:** 如果线性表 L 存在, 判断 L.length 是否为 0, 空就返回 TRUE, 否则返回 FALSE; 如果线性表 L 不存在, 返回 INFEASIBLE。

**时空效率:** 时间复杂度为 O(1), 空间复杂度为 O(1)。

5.status ListLength(SqList L);

**功能:** 获得线性表长度。

**算法实现:** 如果线性表 L 存在, 返回线性表 L.length, 否则返回 INFEASIBLE。

**时空效率:** 时间复杂度为 O(1), 空间复杂度为 O(1)。

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

**功能:** 获得线性表指定位置数据。

**算法实现:** 如果线性表 L 不存在, 返回 INFEASIBLE。如果 i 不在 1 与 L.length 之间, 返回 ERROR。若上述情况未出现就把 L.elem[i-1] 的值赋给 e, 返回 ok。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

7.status LocateElem(SqList L, ElemType e); //简化过

**功能:** 寻找指定元素在线性表中位置。

**算法实现:** 如果线性表 L 存在, 遍历线性表数据, 若发现数据与 e 相等就返回 index+1; 如果 e 不存在, 返回 0; 当线性表 L 不存在时, 返回 INFEASIBLE(即-1)。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

8.status PriorElem(SqList L, ElemType i, ElemType &pre_e);

**功能:** 获得指定元素之前的一个元素。

**算法实现:** 如果线性表 L 存在, 遍历查找该元素, 将元素前一个位置元素赋值给 pre_e, 返回 OK; 如果没有前驱, 返回 ERROR; 如果线性表 L 不存在, 返回 INFEASIBLE。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

9.status NextElem(SqList L, ElemType i, ElemType &next_e);

**功能:** 获得指定元素之后的一个元素。

**算法实现:** 如果线性表 L 存在, 遍历查找该元素, 将元素前一个位置元素赋值给 next_e, 返回 OK; 如果没有后继, 返回 ERROR; 如果线性表 L 不存在, 返回 INFEASIBLE。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

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

**功能:** 插入元素。

**算法实现:** 如果线性表 L 存在, 判断线性表要插入位置是否在 1 和 L.length 之间, 不是则返回 ERROR, 判断线性表 L.length 是否等于 L.listsize, 相同则空间已

满, 分配新内存空间; 将制定位置后的元素全部后移一个位置, 新位置插入在指定位置, 返回 OK; 如果线性表 L 不存在, 返回 INFEASIBLE。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

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

**功能:** 删除元素。

**算法实现:** 如果线性表 L 存在, 判断线性表要插入位置是否在 1 和 L.length 之间, 不是则返回 ERROR, 把 L.elem[i-1] 的值赋给 e, 将之后的元素全部前移一个位置, 返回 OK; 当删除位置不正确时, 返回 ERROR; 如果线性表 L 不存在, 返回 INFEASIBLE。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

12.status ListTraverse(SqList L);

**功能:** 遍历并打印线性表。

**算法实现:** 如果线性表 L 存在, 直接遍历并打印线性表的所有元素, 返回 OK; 如果线性表 L 不存在, 返回 INFEASIBLE。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

### 1.3.2 文件存储实现算法

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

**功能:** 数据保存。

**算法实现:** 如果线性表 L 存在, 打开文件, 根据 L.length 的大小存入 L.elem 数据, 关闭文件, 返回 OK; 如果线性表 L 不存在, 返回 INFEASIBLE。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(n)。

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

**功能:** 读取文件。

**算法实现:** 如果线性表 L 存在, 初始化线性表, 打开文件, 读取数据直到所有数据已经被放入线性表中, 根据读取数据的大小改变 L.length 的值, 关闭文件, 返回 OK; 如果线性表 L 不存在, 返回 INFEASIBLE。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(n)。

### 1.3.3 多线性表存储实现算法

1.status AddList(LISTS &Lists, char ListName[]);

**功能:** 在多线性表集合里添加线性表。

**算法实现:** 检查过多线性表长度足够后, 往多线性表里多申请一块空间给新的线性表, 并把输入的 ListName 赋给新线性表的名字。

**时空效率:** 时间复杂度为 O(1), 空间复杂度为 O(1)。

2.status RemoveList(LISTS &Lists, char ListName[]);

**功能:** 在多线性表集合里删除线性表。

**算法实现:** 遍历多线性表寻找线性表, 寻找到后删除线性表并将后面的线性表向前移动一位。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

3.int LocateList(LISTS Lists, char ListName[]);

**功能:** 在多线性表集合里寻找名为 ListName 的线性表。

**算法实现:** 遍历多线性表寻找线性表, 寻找到后返回线性表的逻辑序号。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

## 1.4 系统测试

### 1.4.1 实验环境

实验环境为 manjaro linux 5.15.41-1, 编译器为 gcc 版本 12.1.0, 代码编写使用编辑器 VsCode。

文件说明:

def.h: 线性表库头文件

function.h: 线性表库实现

main.c: 演示系统实现

### 1.4.2 操作演示

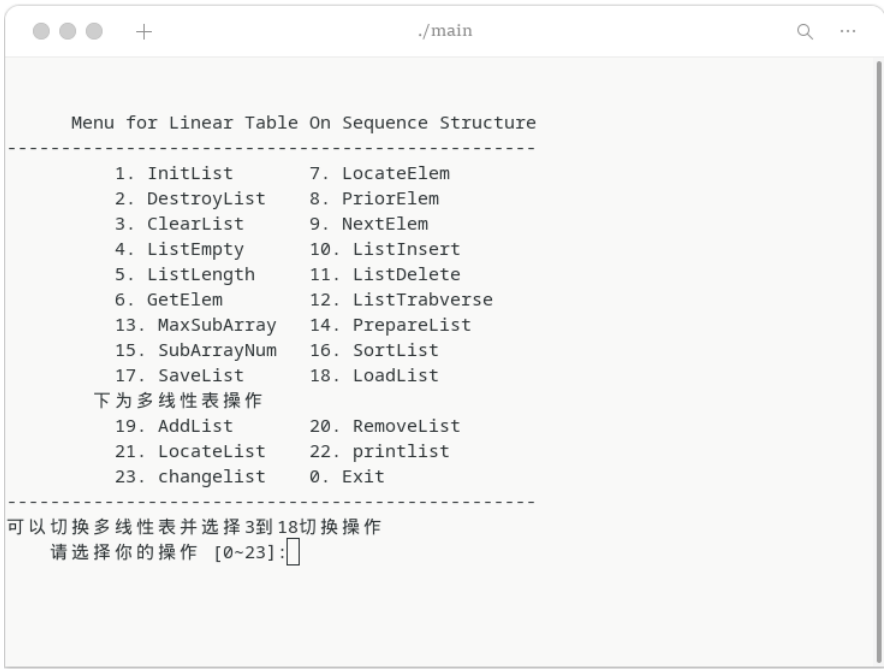


图 1-1 系统演示界面

## 1.5 实验小结

todo

# 2 基于邻接表的图实现

## 2.1 实验目的

通过实验达到

(1) 加深对图的概念、基本运算的理解;

(2) 熟练掌握图的逻辑结构与物理结构的关系;

(3) 以邻接表作为物理结构, 熟练掌握图基本运算的实现。

### 2.1.1 邻接表抽象数据类型

依据最小完备性和常用性相结合的原则, 以函数形式定义了创建图、销毁图、查找顶点、获得顶点值和顶点赋值等 13 种基本运算, 具体运算功能定义如下。

(1) 创建图

CreateCraph(&G,V,VR);

—-初始条件:V 是图的顶点集,VR 是图的关系集;

—-操作结果: 按 V 和 VR 的定义构造图 G。

(2) 销毁图:

DestroyBiTree(T);

—-初始条件: 图 G 已存在;

—-操作结果: 销毁图 G。

(3) 查找顶点

LocateVex(G,u);

—-初始条件: 图 G 存在,u 和 G 中的顶点具有相同特征;

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

(4) 获得顶点值

GetVex (G,v);

—-初始条件: 图 G 存在,v 是 G 中的某个顶点;

—-操作结果: 返回 v 的值。

(5) 顶点赋值

PutVex (G,v,value);

—-初始条件: 图 G 存在,v 是 G 中的某个顶点;

—-操作结果: 对 v 赋值 value。

(6) 获得第一邻接点

FirstAdjVex(&G, v);

—-初始条件: 图 G 存在,v 是 G 的一个顶点;

—-操作结果: 返回 v 的第一个邻接顶点, 如果 v 没有邻接顶点, 返回 “空”。

(7) 获得下一邻接点

NextAdjVex(&G, v, w);

—-初始条件: 图 G 存在,v 是 G 的一个顶点,w 是 v 的邻接顶点;

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

(8) 插入顶点

InsertVex(&G,v);

—-初始条件: 图 G 存在,v 和 G 中的顶点具有相同特征;

—-操作结果: 在图 G 中增加新顶点 v。

(9) 删除顶点

DeleteVex(&G,v);

—-初始条件: 图 G 存在,v 是 G 的一个顶点;

—-操作结果: 在图 G 中删除顶点 v 和与 v 相关的弧。

(10) 插入弧

InsertArc(&G,v,w);

—-初始条件: 图 G 存在,v、w 是 G 的顶点;

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

(11) 删除弧

DeleteArc(&G,v,w);

—-初始条件: 图 G 存在,v、w 是 G 的顶点;

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

(12) 深度优先搜索遍历

DFSTraverse(G,visit());

—-初始条件: 图 G 存在;

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

(13) 广深度优先搜索遍历

BFSTraverse(G,visit());

—-初始条件: 图 G 存在;

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

附加功能

(1) 距离小于 k 的顶点集合

VerticesSetLessThank(G,V,K);

—-初始条件: 图 G 存在;

—-操作结果: 返回与顶点 v 距离小于 k 的顶点集合;

(2) 顶点间最短路径和长度

ShortestPathLength(G,v,w);

初始条件: 图 G 存在;

操作结果: 返回顶点 v 与顶点 w 的最短路径的长度;

(3) 图的连通分量

ConnectedComponentsNums(G);

初始条件: 图 G 存在;

操作结果: 返回图 G 的所有连通分量的个数;

### 2.1.2 邻接表的文件形式管理

1. 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. 设计了线性表文件保存和加载操作的合理模式。附录 B 提供了文件存取的具体方法。

### 2.1.3 实现多个邻接表管理

在整个邻接表之外设计包装邻接表, 将已经存在的所有邻接表用链式结构串起, 对邻接表赋予独特 ID 标识, 通过 ID 选择要进行操作的链表。

## 2.2 系统设计

### 2.2.1 数据结构设计

1. 邻接表的存储数据结构

结构体定义如下:

```
typedef struct
{ //顶点类型定义
  KeyType key;
  char others[20];
} VertexType;

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

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

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

2. 多邻接表的存储数据结构:

结构体定义如下:

```
typedef struct
{ //邻接表的集合类型定义
  struct
  {
    char name[30];
    ALGraph L;
  } elem[10];
  int length;
} ALGraphs;
```

### 2.2.2 演示系统设计

包括用户操作界面和功能调用两个部分。

演示系统语言为英文, 所有操作和提示语言均为中文。

用户操作界面输出可选的邻接表操作, 用户输入数字选择要进行的操作。系统提示用户输入参数。

功能调用部分则将用户输入的有关信息传递给数据结构的操作函数进行调用, 并对函数返回值进行处理判断输出相应提示信息。

## 2.3 系统实现

主要说明各个主要函数的实现思想, 复杂函数可辅助流程图进行说明, 函数和系统实现的源代码放在附录中。

### 2.3.1 邻接表基础运算算法实现

1.status CreateGraph(ALGraph &G, VertexType V[], KeyType VR[][2]);

**功能:** 创建图

**算法实现:** 若 G 不存在, 初始化图, 用 V 创建顶点, 用 VR 创建图的边, 并遍历寻找顶点后用首插法加入邻接表。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

2.status DestroyGraph(ALGraph &G);

**功能:** 销毁无向图, 删除所有顶点和边。

**算法实现:** 遍历邻接图顶点数据, 删除顶点的所有的边后删除顶点。把顶点数和边数设置为 0。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

3.int LocateVex(ALGraph G, KeyType u);

**功能:** 寻找指定顶点在邻接表中位置。

**算法实现:** 遍历邻接表顶点, 寻找到指定元素则返回其逻辑序号。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

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

**功能:** 将顶点值修改为 value

**算法实现:** 遍历邻接表结点, 寻找顶点值为 u 的结点, 将 value 赋给此结点。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

5.int FirstAdjVex(ALGraph G, KeyType u);

**功能:** 寻找顶点 u 的第一邻接顶点位序

**算法实现:** 遍历邻接表结点, 寻找顶点值为 u 的结点, 返回其第一邻接顶点位序。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

6.int NextAdjVex(ALGraph G, KeyType v, KeyType w);

**功能:** 寻找顶点 w 在邻接表顶点 v 的邻接顶点中下一个邻接顶点位置。

**算法实现:** 遍历邻接表结点, 寻找顶点值为 u 的结点, 在其邻接顶点中寻找 v, 返回 v 的下一邻接顶点位序后返回 OK。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

7.status InsertVex(ALGraph &G, VertexType v);

**功能:** 插入顶点。

**算法实现:** 检查已有邻接表中是否存在与 v 的 key 相同的顶点, 邻接表是否不存在或顶点数大于 20, 是则返回 ERROR, 否则在邻接表最后加入 v 顶点并把该

位置的 firstarc 设置为 0, 邻接表顶点数加一后返回 OK。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

8.status DeleteVex(ALGraph &G, KeyType v);

**功能:** 删除顶点及相关的弧。

**算法实现:** 用 LocateVex 确定 v 位置并检查合法性, 无误则遍历邻接表各顶点删除该顶点及其邻接边数据 (每删除一条边则将邻接表总边数减 1), 将后续顶点前移。再遍历一次邻接表顶点, 在每个邻接顶点内对边遍历, 若值为 v 的位置序号则删除该边并将后续边向前移动一次, 对边再遍历一次, 若值大于 v 的位置序号则将值-1。将邻接表顶点数减 1 后返回 OK;

**时空效率:** 时间复杂度为 O($n^2$), 空间复杂度为 O(1)。

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

**功能:** 插入边。

**算法实现:** 检查已有邻接表 v 顶点中是否存在 w 邻接边或邻接表是否不存在, 是则返回 ERROR, 否则在邻接表 v 顶点处用首插法插入 w 的邻接边, 在邻接表 w 顶点处用首插法插入 v 的邻接边, 将边数加一后返回 OK。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

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

**功能:** 删除边。

**算法实现:** 检查已有邻接表 v 顶点中是否不存在 w 邻接边或邻接表是否不存在, 是则返回 ERROR, 否则在邻接表 v 顶点处删除 w 邻接边, 在邻接表 w 顶点处删除 v 邻接边, 将边数减一后返回 OK。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

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

**功能:** 深度优先搜索遍历邻接表。

**算法实现:** 建立大小为 G.vexnum 的数组 k, 初始化为 0, 从 G 第一个节点顺其邻接顶点递归访问, 对已经访问的节点在 k 内的相应位置标记为 1, 访问到最后一个邻接顶点时转到下一个顶点, 并检查 k 内相应位置的值是否为 1, 是则跳过,

上述循环直到最后一个顶点。返回 ok。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(n)。

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

**功能:** 广深度优先搜索遍历邻接表。

**算法实现:** 建立大小为 G.vexnum 的数组 k, 初始化为 0, 从 G 第一个节点顺其邻接顶点顺序访问, 对已经访问的节点在 k 内的相应位置标记为 1, 访问到最后一个邻接顶点时转到下一个顶点, 并检查 k 内相应位置的值是否为 1, 是则跳过, 上述循环直到最后一个顶点。返回 ok。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(n)。

附加功能:

1.status VerticesSetLessThank(ALGraph G, KeyType v, int k);

**功能:** 求距离小于 k 的顶点集合

**算法实现:** 建立大小为 G.vexnum 的数组 q, 初始化为 0, 寻找顶点 v 的位置, 递归访问 v 的邻接顶点并标记在 q 中直到栈调用达到 k 层。打印出所有在 q 中有标记的顶点, 返回 OK。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(n)。

2.int ShortestPathLength(ALGraph G, KeyType v, KeyType w);

**功能:** 计算顶点间最短路径和长度

**算法实现:** 如下图

**时空效率:** 时间复杂度为 O($n^2$), 空间复杂度为 O(n)。

3.int ConnectedComponentsNums(ALGraph G);

**功能:** 计算图的联通分量

**算法实现:** 建立大小为 G.vexnum 的数组 q, 初始化为 0, 设置变量 ans 为 1, 用 DFS 从第一个 G 的顶点递归并把沿途顶点标记, 之后访问下一个顶点, 若已被标记过就跳过, 未被标记就对该顶点进行像之前一样的 DFS 并对 ans+1, 最后返回 ans。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(n)。

```
int ShortestPathLength(ALGraph G, KeyType v, KeyType w)
{
    int i, m = -1, n = -1;
    for (int i = 0; i < G.vexnum; i++)//寻找v，w
    {
        if (G.vertices[i].data.key == v)
            m = i;
        if (G.vertices[i].data.key == w)
            n = i;
    }
    ArcNode *p;
    int s1=0,s2=0,x=0,sum = -1,see[20]={0};
    VXNode quene1[20];//这是一个为vnode搭建的栈结构
    quene1[s2].gg = G.vertices[m];
    quene1[s2++].length0 = 0;
    see[m] = 0;
    while (s1 != s2)
    {
        p = quene1[s1++].gg.firstarc;
        for (; p && p->adjvex >= 0; p = p->nextarc)
        {
            x = p->adjvex;
            if (see[x] == 0)
            {
                quene1[s2].gg = G.vertices[x];
                quene1[s2++].length0 = quene1[s1 - 1].length0 + 1;
                see[x] = 1;
            }
            if (x == n)
            {
                sum = quene1[s2 - 1].length0;
                return sum;
            }
        }
    }
    return sum;
}
```

图 2-1　ShortestPathLength 函数

### 2.3.2　文件存储算法实现

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

**功能:** 数据保存。

**算法实现:** 如果无向图 G 存在, 打开文件, 根据 G.vexnum 的大小存入数据, 关闭文件, 返回 OK; 如果无向图 G 不存在, 返回 INFEASIBLE。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(n)。

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

**功能:** 读取文件。

**算法实现:** 如果无向图 G 存在, 初始化无向图, 打开文件, 读取数据并使用 CreateGraph 函数将数据存入无向图, 关闭文件, 返回 OK; 如果无向图 G 不存在,

返回 INFEASIBLE。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(n)。

### 2.3.3 多文件存储算法实现

1.status AddGraph(ALGraphs &Graphs, char GraphName[], VertexType V[], KeyType VR[][2]);

**功能:** 在多无向图集合里添加无向图。

**算法实现:** 检查过多无向图长度足够后, 往多无向图里多申请一块空间给新的无向图, 并把输入的 GraphName 赋给新无向图的名字。

**时空效率:** 时间复杂度为 O(1), 空间复杂度为 O(1)。

2.status RemoveGraph(ALGraphs &Graphs, char GraphName[]);

**功能:** 在多无向图集合里删除无向图。

**算法实现:** 遍历多无向图寻找无向图, 寻找到后删除无向图并将后面的无向图向前移动一位。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

3.int LocateGraph(ALGraphs Graphs, char GraphName[]);

**功能:** 在无向图集合里寻找名为 GrpahName 的线性表。

**算法实现:** 遍历多无向图寻找无向图, 寻找到后返回无向图的逻辑序号。

**时空效率:** 时间复杂度为 O(n), 空间复杂度为 O(1)。

## 2.4 系统测试

主要说明针对各个函数正常和异常的测试用例及测试结果

### 2.4.1 实验环境

实验环境为 manjaro linux 5.15.41-1, 编译器为 gcc 版本 12.1.0, 代码编写使用编辑器 VsCode。

文件说明:

def.h: 线性表库头文件

function.h: 线性表库实现

main.c: 演示系统实现

### 2.4.2 操作演示

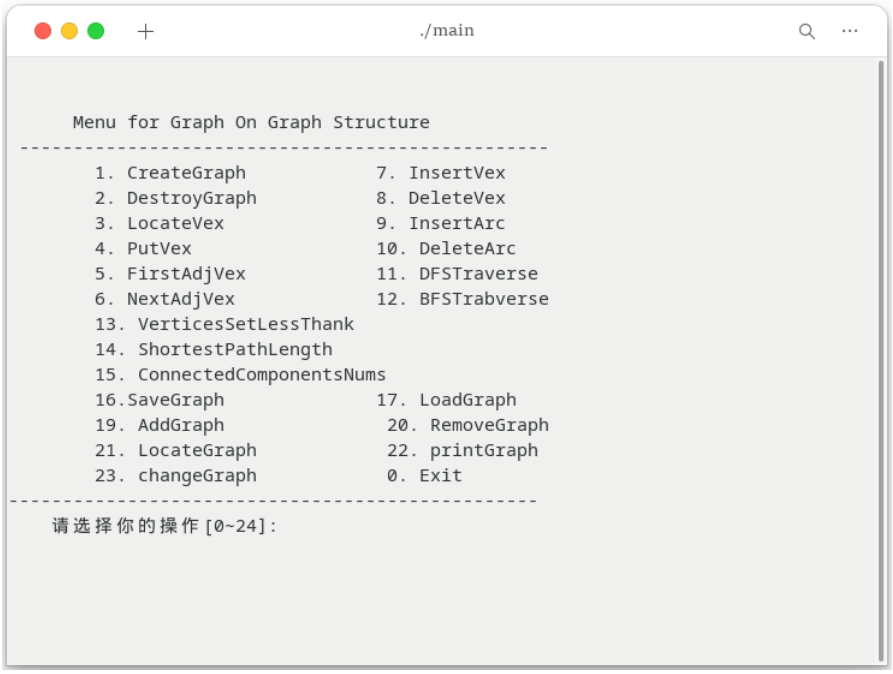


图 2-2 系统演示界面

## 2.5 实验小结

# 3 课程的收获和建议

描述通过学习该专题, 有何收获, 有何建议, 如某专题可适当减少讲授时间、某专题可适当增加讲授内容和时间等。描述通过学习该专题, 有何收获, 有何建议, 如某专题可适当减少讲授时间、某专题可适当增加讲授内容和时间等。描述通过学习该专题, 有何收获, 有何建议, 如某专题可适当减少讲授时间、某专题可适当增加讲授内容和时间等。描述通过学习该专题, 有何收获, 有何建议, 如某专题可适当减少讲授时间、某专题可适当增加讲授内容和时间等。

## 3.1 基于顺序存储结构的线性表实现

描述通过学习计算机基础知识专题, 有何收获, 有何建议, 如某专题可适当减少讲授时间、某专题可适当增加讲授内容和时间等。描述网页的设计和实现过程中遇到的问题及如何解决。描述网页的设计和实现过程中遇到的问题及如何解决。描述网页的设计和实现过程中遇到的问题及如何解决。描述网页的设计和实现过程中遇到的问题及如何解决。描述网页的设计和实现过程中遇到的问题及如何解决。描述网页的设计和实现过程中遇到的问题及如何解决。描述网页的设计和实现过程中遇到的问题及如何解决。描述网页的设计和实现过程中遇到的问题及如何解决。

## 3.2 基于链式存储结构的线性表实现

描述通过学习文档撰写工具 LaTeX 专题, 有何收获, 有何建议, 如某专题可适当减少讲授时间、某专题可适当增加讲授内容和时间等。描述通过学习文档撰写工具 LaTeX 专题, 有何收获, 有何建议, 如某专题可适当减少讲授时间、某专题可适当增加讲授内容和时间等。

## 3.3 基于二叉链表的二叉树实现

描述通过学习编程工具 Python 专题, 有何收获, 有何建议, 如某专题可适当减少讲授时间、某专题可适当增加讲授内容和时间等。描述通过学习编程工具 Python 专题, 有何收获, 有何建议, 如某专题可适当减少讲授时间、某专题可适当增加讲授内容和时间等。

## 3.4 基于二叉链表的二叉树实现

描述通过学习计算机基础知识专题, 有何收获, 有何建议, 如某专题可适当减少讲授时间、某专题可适当增加讲授内容和时间等。描述通过学习计算机基础知识专题, 有何收获, 有何建议, 如某专题可适当减少讲授时间、某专题可适当增加讲授内容和时间等。

# 参考文献

[1] CHEN J, LI Z, JIN Y, et al. Video Saliency Prediction via Spatio-Temporal Reasoning[J]. Neurocomputing, 2021, 462 : 59 – 68.

[2] CHEN J, LI Q, LING H, et al. Audiovisual Saliency Prediction via Deep Learning[J]. Neurocomputing, 2021, 428 : 248 – 258.

[3] MEHRABIAN A, RUSSELL J. An approach to environmental psychology[M]. [S.l.] : MIT, 1974.

[4] REZAEI M, KLETTE R. Look at the driver, look at the road: No distraction! no accident![C] // CVPR. 2014 : 129 – 136.

[5] RAMNATH K, KOTERBA S, XIAO J, et al. Multi-view AMM fitting and construction[J]. International Journal of Computer Vision, 2008, 76 : 183 – 204.

[6] BAFNA V, PEVZNER P A. Genome Rearrangements and Sorting by Reversals[J/OL]. SIAM J. Comput., 1996, 25(2) : 272 – 289. http://dx.doi.org/10.1137/S0097539793250627.

[7] SKINAZE. HUSTPaperTemp[EB/OL]. . https://github.com/skinaze/HUSTPaperTemp.

[8] 尹圣君, 钱尚达, 李永代, et al. [M]. [S.l.] : 机械工业出版社, 2015.

[9] ANON. GIEEE 802.21 Media Independent Handover (MIH)[S]. Washington University in St. Louis : IEEE, 2010.

[10] 戴维民. [M]. [S.l.] : 学林出版社, 2008.

[11] PRASAD N, KHOJASTEPOUR M A, JIANG M, et al. MU-MIMO: Demodulation at the Mobile Station[R]. 2009 : 1 – 11.

[12] PAULRAJ A J, Jr HEATH R W, SEBASTIAN P K, et al. Spatial Multiplexing in a Cellular Network[P]. 2000-5-23.

[13] 立陶宛进入欧元时代 [N]. –. –.

# 附录 A 基于顺序存储结构线性表实现的源程序

**def.h**

```
 /*
 * @Author: wizzz wizo.o@outlook.com
 * @Date: 2022-05-03 16:13:01
 * @LastEditors: wizzz wizo.o@outlook.com
 * @LastEditTime: 2022-06-13 22:54:39
 * @FilePath: /latex/home/rickeee/cprogram/line/def.h
 * @Description: a def.h document for wizzz's datastruct experimental
 * Copyright (c) 2022 by wiz wizo.o@outlook.com, All Rights Reserved.
 */
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <unordered_map>
#include <algorithm>
#include <iostream>
#include <string.h>

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

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

```

```
28 #define LIST_INIT_SIZE 100
29 #define LISTINCREMENT 10
30 typedef struct
31 { //顺序表（顺序结构）的定义
32       ElemType *elem;
33       int length;
34       int listsize;
35 } SqList;
36 typedef struct
37 { //线性表的集合类型定义
38       struct
39       {
40             char name[30];
41             SqList L;
42       } elem[10];
43       int length;
44 } LISTS;
45 status InitList(SqList &L);
46 status PrepareList(SqList &L);
47 status DestroyList(SqList &L);
48 status ClearList(SqList &L);
49 status ListEmpty(SqList L);
50 status ListLength(SqList L);
51 status GetElem(SqList L, int i, ElemType &e);
52 status LocateElem(SqList L, ElemType e);
53 status PriorElem(SqList L, ElemType i, ElemType &pre_e);
54 status NextElem(SqList L, ElemType i, ElemType &next_e);
55 status ListInsert(SqList &L, int i, ElemType e);
56 status ListDelete(SqList &L, int i, ElemType &e);
57 status ListTraverse(SqList L);
58 status MaxSubArray(SqList L);
```

```
59 status SubArrayNum(SqList L, int k);
60 status SortList(SqList &L);
61 status SaveList(SqList L, char FileName[]);
62 status LoadList(SqList &L, char FileName[]);
63 status AddList(LISTS &Lists, char ListName[]);
64 status RemoveList(LISTS &Lists, char ListName[]);
65 int LocateList(LISTS Lists, char ListName[]);
```

**function.h**

```
1   status InitList(SqList &L)
2  // 线性表L不存在，构造一个空的线性表，返回OK，否则返回INFEASIBLE。
3  {
4          if (L.elem)
5                  return INFEASIBLE;
6          L.elem = (ElemType *)malloc(LIST_INIT_SIZE);
7          L.length = 0;
8          L.listsize = LIST_INIT_SIZE;
9          return OK;
10 }
11 status PrepareList(SqList &L)
12 //一次准备一整个线性表
13 {
14         int n;
15         scanf("%d", &n);
16         if (!L.elem)
17                 return -1;
18         if (n + L.length >= L.listsize)
19         {
20
21                 L.elem = (ElemType *)realloc(L.elem, (n + L.listsize)
```

```
22                  L.listsize += LISTINCREMENT;
23          }
24          if (n < 1)
25                  return ERROR;
26          int len = L.length;
27          for (int j = len; j < n + len; j++)
28          {
29                  scanf("%d", &L.elem[j]);
30                  L.length++;
31          }
32          return OK;
33  }
34  status DestroyList(SqList &L)
35  // 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IN
36  {
37          if (L.elem)
38          {
39                  free(L.elem);
40                  L.elem = NULL;
41                  L.length = 0;
42                  L.listsize = 0;
43                  return OK;
44          }
45          return INFEASIBLE;
46  }
47  status ClearList(SqList &L)
48  // 如果线性表L存在，删除线性表L中的所有元素，返回OK，否则返回INFEASIBL
49  {
50          if (L.elem)
51          {
52                  free(L.elem);
```

```
53                  L.elem = (ElemType *)malloc(LIST_INIT_SIZE);
54                  L.length = 0;
55                  L.listsize = 0;
56                  return OK;
57          }
58          return INFEASIBLE;
59  }
60  status ListEmpty(SqList L)
61  // 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；
62  {
63          if (!L.elem)
64                  return INFEASIBLE;
65          if (L.length)
66                  return FALSE;
67          return TRUE;
68  }
69  status ListLength(SqList L)
70  // 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。
71  {
72          if (L.elem)
73                  return L.length;
74          return INFEASIBLE;
75  }
76  status GetElem(SqList L, int i, ElemType &e)
77  // 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
78  {
79          if (!L.elem)
80                  return INFEASIBLE;
81          if (i > L.length || i <= 0)
82                  return ERROR;
83          e = L.elem[i - 1];
```

```
84          return OK;
85  }
86  status LocateElem(SqList L, ElemType e)
87  // 如果线性表L存在，查找元素e在线性表L中的位置序号并返回该序号；如果e不
88  {
89          if (!L.elem)
90                  return -1;
91          for (int i = 0; i < L.length; i++)
92          {
93                  if (L.elem[i] == e)
94                          return i + 1;
95          }
96          return 0;
97  }
98  status PriorElem(SqList L, ElemType e, ElemType &pre)
99  // 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
100 {
101         if (!L.elem)
102                 return -1;
103         for (int i = 1; i < L.length; i++)
104         {
105                 if (L.elem[i] == e)
106                 {
107                         pre = L.elem[i - 1];
108                         return OK;
109                 }
110         }
111         return ERROR;
112 }
113 status NextElem(SqList L, ElemType e, ElemType &next)
114 // 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
```

```
115 {
116         if (!L.elem)
117                 return -1;
118         for (int i = 0; i < L.length - 1; i++)
119         {
120                 if (L.elem[i] == e)
121                 {
122                         next = L.elem[i + 1];
123                         return OK;
124                 }
125         }
126         return ERROR;
127 }
128 status ListInsert(SqList &L, int i, ElemType e)
129 // 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
130 {
131         if (!L.elem)
132                 return -1;
133         if (i < 1 || i > L.length + 1)
134                 return ERROR;
135         if (L.length == L.listsize)
136         {
137
138                 L.elem = (ElemType *)realloc(L.elem, (LISTINCREMENT +
139                 L.listsize += LISTINCREMENT;
140         }
141         if (i <= L.length)
142         {
143                 for (int j = L.length - 1; j >= i - 1; j--)
144                         L.elem[j + 1] = L.elem[j];
145         }
```

```
146         L.elem[i - 1] = e;
147         L.length++;
148         return OK;
149 }
150 status ListDelete(SqList &L, int i, ElemType &e)
151 // 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
152 {
153         if (!L.elem)
154                 return -1;
155         if (i < 1 || i > L.length)
156                 return ERROR;
157         e = L.elem[i - 1];
158         for (int j = i - 1; j < L.length - 1; j++)
159         {
160                 L.elem[j] = L.elem[j + 1];
161         }
162         L.length--;
163         return OK;
164 }
165 status ListTraverse(SqList L)
166 // 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;
167 {
168         int i;
169         printf("\n----------all elements ----------------------\n");
170         for (i = 0; i < L.length; i++)
171                 printf("%d ", L.elem[i]);
172         printf("\n------------------ end ----------------------\n");
173         return L.length;
174 }
175 status MaxSubArray(SqList L)
176 //最打连续子数组和
```

```
177 {
178         if (!L.elem)
179                 return -1;
180         if (L.length < 1)
181                 return ERROR;
182         int pre = 0, maxAns = L.elem[0];
183         for (int i = 0; i < L.length; i++)
184         {
185                 pre = pre + L.elem[i] > L.elem[i] ? pre + L.elem[i] : 
186                 maxAns = pre > maxAns ? pre : maxAns;
187         }
188         return maxAns;
189 }
190 
191 status SubArrayNum(SqList L, int k)
192 //和为k的子数组
193 {
194         int n;
195         using namespace std;
196         unordered_map<int, int> p;
197         p[0] = 1;
198         if (!L.elem)
199                 return -1;
200         if (L.length < 1)
201                 return ERROR;
202         int pre = 0, ans = 0;
203         for (int i = 0; i < L.length; i++)
204         {
205                 pre += L.elem[i];
206                 if (p.find(pre - k) != p.end())
207                         ans += p[pre - k];
```

```
208                     p[pre]++;
209             }
210             return ans + 1;
211     }
212     status SortList(SqList &L)
213     //顺序表排序
214     {
215             using namespace std;
216             if (!L.elem)
217                     return -1;
218             if (L.length < 1)
219                     return ERROR;
220             sort(L.elem, (L.elem + L.length));
221             return OK;
222     }
223     status SaveList(SqList L, char FileName[])
224     // 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
225     {
226             if (!L.elem)
227                     return -1;
228             FILE *pfile;
229             pfile = fopen(FileName, "w");
230             if (pfile != NULL)
231             {
232                     for (int i = 0; i < L.length; i++)
233                     {
234                             fprintf(pfile, "%d ", L.elem[i]);
235                     }
236             }
237             else
238             {
```

```
239                     fclose(pfile);
240                     return 0;
241             }
242             fclose(pfile);
243             return OK;
244     }
245     status LoadList(SqList &L, char FileName[])
246     // 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
247     {
248             if (!L.elem)
249                     return -1;
250             L.elem = (ElemType *)malloc(LIST_INIT_SIZE);
251             L.length = 0;
252             L.listsize = LIST_INIT_SIZE;
253             FILE *pfile;
254             pfile = fopen(FileName, "r+");
255             if (pfile != NULL)
256             {
257                     int i = 0;
258                     while (fscanf(pfile, "%d", &L.elem[i++]) != EOF)
259                             ;
260                     L.length = i - 1;
261             }
262             else
263             {
264                     fclose(pfile);
265                     return 0;
266             }
267             fclose(pfile);
268             return OK;
269     }
```

```
270
271 status AddList(LISTS &Lists, char ListName[])
272 // 只需要在Lists中增加一个名称为ListName的空线性表，线性表数据又后台测
273 {
274         if (Lists.length < 0 || Lists.length > 10)
275                 Lists.length = 0;
276         int len = Lists.length + 1;
277         strcpy(Lists.elem[len - 1].name, ListName);
278         Lists.elem[len - 1].L.elem = (ElemType *)malloc(LIST_INIT_SIZE
279         Lists.elem[len - 1].L.length = 0;
280         Lists.elem[len - 1].L.listsize = LIST_INIT_SIZE;
281         Lists.length = len;
282         return OK;
283 }
284 status RemoveList(LISTS &Lists, char ListName[])
285 // Lists中删除一个名称为ListName的线性表
286 {
287         for (int i = 0; i < Lists.length; i++)
288         {
289                 if (strcmp(ListName, Lists.elem[i].name) == 0)
290                 {
291                         DestroyList(Lists.elem[i].L);
292                         for (int j = i; j < Lists.length - 1; j++)
293                         {
294                                 Lists.elem[j] = Lists.elem[j + 1];
295                         }
296                         Lists.length--;
297                         return OK;
298                 }
299         }
300         return ERROR;
```

```
301 }
302 int LocateList(LISTS Lists, char ListName[])
303 // 在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，否则返回
304 {
305         for (int i = 0; i < Lists.length; i++)
306         {
307                 if (strcmp(ListName, Lists.elem[i].name) == 0)
308                 {
309                         return i + 1;
310                 }
311         }
312         return ERROR;
313 }
```

# 附录 B 基于链式存储结构线性表实现的源程序

# 附录 C 基于二叉链表二叉树实现的源程序

# 附录 D 基于邻接表图实现的源程序