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 CS 启明 2401

学　　号 U202414887

姓　　名 王李超

指导教师 王雄

报告日期 2025 年 6 月 1 日

计算机科学与技术学院

目　　录

1 基于顺序存储结构的线性表实现

1.1 问题描述

1.1.1 实验目的

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

1.1.2 具体问题

在设计线性表时，需要解决以下具体问题：

- **存储结构的选择**：选择顺序存储结构还是链式存储结构，需权衡存储效率和操作效率。
- **容量的动态扩展**：当线性表存储空间不足时，如何动态扩展存储容量以容纳更多元素。
- **基本操作的实现**：包括插入、删除、查找、更新等操作的具体实现及其时间复杂度优化。
- **边界条件处理**：如何处理空表、满表以及非法操作（如越界访问）等特殊情况。
- **数据类型的通用性**：设计线性表时，如何支持存储多种数据类型（如整数、浮点数、字符串等）。
- **内存管理**：如何高效管理内存，避免内存泄漏或冗余分配。
- **算法效率**：针对不同操作需求，优化算法以提高线性表的整体性能。

1.2 系统设计

整体系统结构设计方面，本系统采用模块化设计思想，将顺序表的各项操作功能（如初始化、插入、删除、查找、遍历、排序、文件读写等）分别封装为独立的函数，并通过主程序 main01.cpp 提供统一的菜单式交互界面，方便用户进行各类操作。系统支持多顺序表管理，用户可新建、删除、切换和重命名多个顺序表，提升了系统的灵活性和扩展性。各功能模块之间通过头文件（如 def.h、

func.h）进行数据类型和函数声明的解耦，便于维护和升级。

数据结构设计方面，核心采用顺序存储结构实现线性表。定义了 SqList 结构体，包含元素指针和当前长度等信息，实现了线性表的基本操作。为支持多顺序表管理，设计了 LISTS 结构体，内部维护一个顺序表数组，每个元素包含一个 SqList 及其名称。元素类型 ElemType 可根据实际需求灵活定义。通过结构体嵌套和指针管理，系统实现了对多个线性表的统一管理和操作，保证了数据的有序性和高效性。整体设计兼顾了功能完整性、易用性和可扩展性。

1.3 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。

1.3.1 主要函数实现思想

- **InitList**：判断线性表是否已存在，若不存在则分配初始空间，初始化长度和容量。
- **DestroyList**：释放线性表空间，并将指针和长度等信息重置，防止内存泄漏。
- **ClearList**：不释放空间，仅将长度归零，实现逻辑清空。
- **ListEmpty**：判断线性表是否存在及是否为空，返回相应状态。
- **ListLength**：返回线性表当前长度，若不存在则返回异常。
- **GetElem**：获取指定位置元素，先判断合法性和存在性。
- **LocateElem**：顺序查找指定元素，返回其逻辑序号，未找到返回 0。
- **PriorElem/NextElem**：查找指定元素的前驱或后继，遍历查找并返回相邻元素。
- **ListInsert**：判断插入位置合法性，必要时扩容，移动元素后插入新元素。
- **ListDelete**：判断删除位置合法性，保存被删元素，移动后续元素覆盖。
- **ListTraverse**：顺序输出所有元素，便于调试和展示。
- **MaxSubArray**：实现最大连续子数组和的求解，采用动态规划思想。
- **SubArrayNum**：统计和为指定值的子数组个数，双重循环遍历所有子区间。
- **sortList**：采用冒泡排序对顺序表元素排序。
- **saveListToFile/loadListFromFile**：实现顺序表的文件保存与加载，便于数据持久化。

- **manageMultipleLists**：输出当前所有顺序表及其名称，实现多表管理。
- **AddList/RemoveList/LocateList**：实现多顺序表的添加、删除和查找，支持名称唯一性和内存管理。

1.3.2 部分函数实现方法

以下选取几个实现较为复杂或具有代表性的函数，简要说明其实现思路，并给出伪代码辅助理解。

1. ListInsert（顺序表插入） 该函数需判断插入位置是否合法，若空间不足则动态扩容，然后将插入位置及其后的元素依次后移，最后插入新元素。

算法 1.1. ListInsert（顺序表插入）

Input: 顺序表 L，插入位置 i，元素 e

Output: 插入是否成功

if L 不存在 **then**

return INFEASIBLE

end if

检查内存分配是否成功

for $j = L.length$ **downto** i **do**

$L.elem[j] \leftarrow L.elem[j-1]$

end for

将元素插入到位置 i：$L.elem[i-1] \leftarrow e$

return OK

2. ListDelete（顺序表删除） 首先判断删除位置是否合法，保存被删元素，然后将其后的元素依次前移，最后长度减一。

算法 1.2. ListDelete（顺序表删除）

Input: 顺序表 L，删除位置 i

Output: 被删除元素 e，删除是否成功

if L 不存在 **then**

return INFEASIBLE
end if
检查删除位置合法性
$e \leftarrow L.elem[i-1]$
for $j = i$ **to** $L.length - 1$ **do**
$L.elem[j-1] \leftarrow L.elem[j]$
end for
顺序表长度减一
return OK

3. MaxSubArray（最大连续子数组和） 采用动态规划思想，遍历数组，记录当前子数组和与最大值。

算法 1.3. MaxSubArray（最大连续子数组和）
Input: 顺序表 L
Output: 最大连续子数组和 $maxSum$
$maxSum \leftarrow L.elem[0]$
$currentSum \leftarrow 0$
for $i = 0$ **to** $L.length - 1$ **do**
if $currentSum > 0$ **then**
$currentSum \leftarrow currentSum + L.elem[i]$
else
$currentSum \leftarrow L.elem[i]$
end if
if $currentSum > maxSum$ **then**
$maxSum \leftarrow currentSum$
end if
end for
return $maxSum$

4. AddList（多顺序表添加） 支持批量添加，需判断名称唯一性，动态分配空间，并循环插入元素直到输入 0 结束。

算法 1.4. AddList（多顺序表添加）

Input: 多顺序表 $Lists$，新表名称 $ListName$

Output: 添加是否成功

```
for 每个待添加顺序表 do
    while 名称重复 do
        提示重新输入 ListName
    end while
    if 顺序表数量已达上限 then
        return ERROR
    end if
    分配新表空间，初始化
    while 输入元素 e ≠ 0 do
        插入 e 到新表
    end while
    添加成功
end for
return OK
```

1.4 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。测试用例设计时，考虑了正常情况、边界条件（如1-1中多次创建、删除线性表，查询边界处元素）和异常情况等多种场景，确保系统的健壮性和稳定性。以下是部分测试用例及其结果：

图 1-1　线性表操作界面

1.4.1　基础功能测试

表 1-1　线性表主要功能测试用例及结果

测试功能及序号	输入	输出
1. 构造空线性表	\	线性表创建成功!
1. 构造空线性表	\	线性表创建失败!
10. 插入元素（3 次）	4 1；6 2；8 3	插入成功!（3 次）
4. 判空线性表	\	线性表不是空表!
5. 求表长	\	线性表的长度为：3
6. 获取元素	2	线性表的第 2 个元素为：6
7. 定位元素	8	线性表中元素 8 的序号为：3
8. 获取前驱	4	线性表中元素 4 的前驱元素查找失败!
8. 获取前驱	6	线性表中元素 6 的前驱元素为：4
9. 获取后继	8	线性表中元素 8 的后继元素查找失败!

9. 获取后继	4	线性表中元素 4 的后继元素为：6
12. 遍历线性表	\	4 6 8
16. 文件保存/17. 文件读取	\	保存成功!/载入成功!
11. 删除元素	2	线性表中元素 6 删除成功！
3. 清空线性表	\	线性表清空成功!
2. 销毁线性表	\	线性表销毁成功!
2. 销毁线性表	\	线性表销毁失败!
0. 退出系统	\	欢迎再次使用本系统！

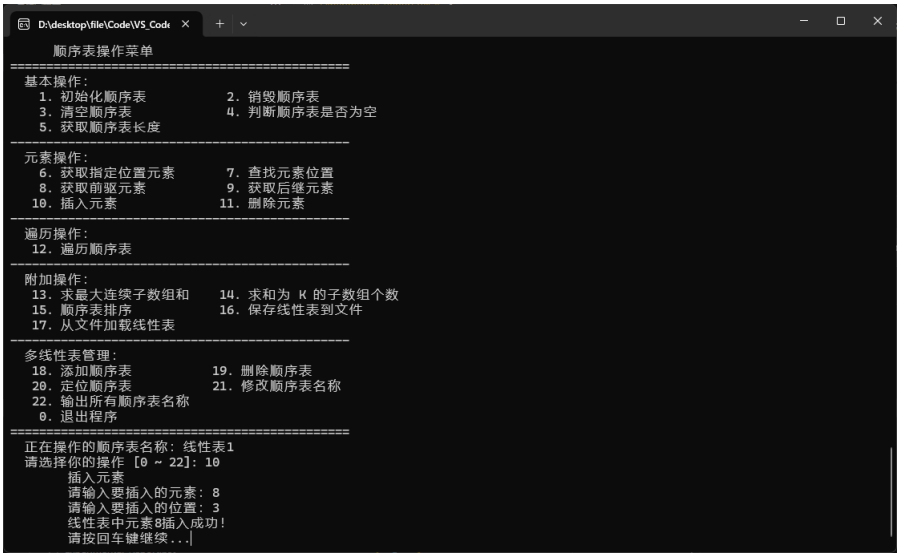

图 1-2　基础功能测试截图

1.4.2　附加功能测试

表 1-3　测试时初始状态

表名称	数据
a	无

b	20 1 2 -4 10 -19 2 2 50

表 1-5　线性表附加功能测试用例及结果

测试功能及序号	输入	输出
13. 最大连续子数组和	b	最大连续子数组和为：55
14. 和为指定值的子数组个数	1 2 -3 4 -5 6 7 -8 9 10	和为指定值的子数组个数为：0
15. 顺序表排序	b	排序成功！
12. 遍历线性表	\	20 1 2 -4 10 -19 2 2 50
16. 文件保存	\	保存成功!
3. 清空线性表	\	线性表清空成功!
17. 文件读取	\	载入成功!
12. 遍历线性表	\	20 1 2 -4 10 -19 2 2 50
22. 输出所有顺序表名称	\	线性表名称：a b

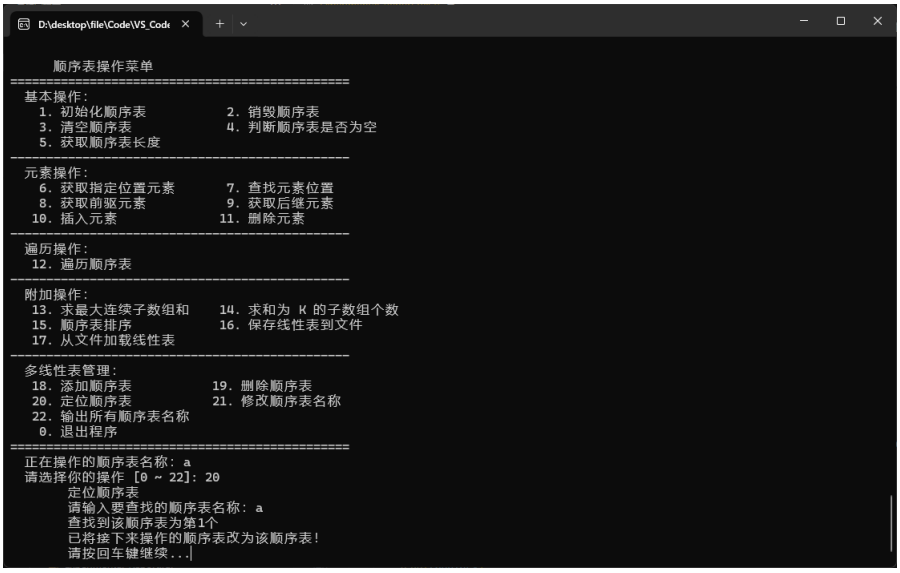

图 1-3　附加功能测试截图

1.4.3　多线性表管理功能测试

表 1-7　多线性表管理功能测试用例及结果

测试功能及序号	输入	输出
18. 添加线性表	2，a，b，20 1 2 -4 10 -19 2 2 50 0	添加成功！（2 次）
19. 删除线性表	线性表 2	删除成功！
20. 定位线性表	线性表 1	查找成功！
22.　输出所有顺序表名称	\	a, b
5. 重命名线性表	a，a1	重命名成功！
22.　输出所有顺序表名称	\	a1, b
0. 退出系统	\	欢迎再次使用本系统！

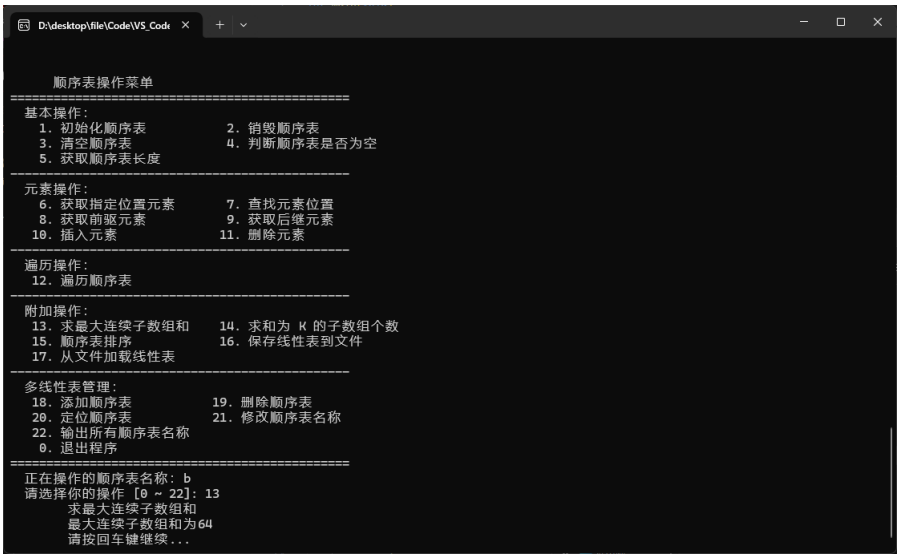

图 1-4　多线性表管理功能测试截图

1.5　实验小结

本次实验通过对线性表的实现与操作，深入理解了线性表的基本概念、存储结构及其基本运算。通过对顺序存储结构的实现，掌握了线性表的动态扩展、插入、删除、查找等操作的具体实现方法。同时，通过多顺序表管理功能，提升了

系统的灵活性和扩展性。实验中遇到的问题主要集中在内存管理和边界条件处理上，通过不断调试和优化，最终实现了一个功能完整、易用性强的线性表系统。

在设计线性表的过程中，遭遇了一些困难，如如何高效地管理内存、如何处理特殊情况（如空表、满表等）。通过查阅资料和反复测试，逐步解决了这些问题。此外，实验还涉及了文件读写操作的实现，增强了数据的持久化能力。通过对线性表的遍历和排序等操作，提升了系统的实用性和用户体验。整体而言，本次实验不仅加深了对线性表的理解，也提高了编程能力和问题解决能力。

此外，实验还涉及了文件读写操作的实现，增强了数据的持久化能力。通过对线性表的遍历和排序等操作，提升了系统的实用性和用户体验。整体而言，本次实验不仅加深了对线性表的理解，也提高了编程能力和问题解决能力。

2 基于邻接表的图实现

2.1 问题描述

在设计基于邻接表的图结构时，可能会遇到以下问题：

- **内存管理问题**：邻接表需要动态分配内存来存储顶点和边的信息，可能会出现内存泄漏或分配失败的情况。
- **边界条件处理**：如何处理特殊情况，例如空图、孤立顶点（没有边连接的顶点）、无边图等。
- **图的表示与操作效率**：如何高效地查找某个顶点的所有邻接点，尤其是在稀疏图中。
- **有向图与无向图的处理**：在无向图中，插入边时需要同时更新两个顶点的邻接表；在有向图中，如何区分入度和出度。
- **权重的存储与处理**：如果是带权图，如何在邻接表中存储边的权重，并处理负权边或零权边。
- **数据一致性问题**：在插入或删除顶点和边时，如何确保邻接表的结构始终保持一致，避免重复插入边或顶点。

2.2 系统设计

2.2.1 数据结构定义（graph_def.h）：

1. 基本常量和类型定义：定义了常量（如 TRUE、FALSE、OK 等）和状态类型（status），用于统一表示函数的返回值和状态。
2. 图的种类：使用 GraphKind 枚举类型定义有向图（DG）、有向网（DN）、无向图（UDG）和无向网（UDN）。
3. 顶点结构的设计：
    - 顶点类型（VertexType）:包含顶点的关键字（key）和其他信息（others）。
    - 边结点（ArcNode）：表示图的邻接点位置（adjvex）及指向下一条边的指针（nextarc）。
    - 顶点结点（VNode）:顶点信息（data）及指向第一条边的指针（firstarc）。
    - 邻接表（AdjList）：表示图或网络的核心结构。

2.2.2 功能实现（graph_func.h）：

- 提供了图的基本操作函数，例如：
  - 图的创建与初始化。
  - 插入顶点或边。
  - 删除顶点或边。
  - 图的遍历（如深度优先搜索/广度优先搜索）。
- 针对不同图类型（有向图、无向图等）实现了特定的处理逻辑。

2.2.3 设计优点

- 邻接表在表示稀疏图时，避免了邻接矩阵中大量无效的 0 存储，节省了内存。
- 通过链式结构，支持插入、删除边和删除顶点或边，适应性更强。
- 使用 GraphKind 枚举类型区分不同图类型，设计清晰，便于扩展。
- 数据结构和功能实现分离（graph_def.h 定义结构，graph_func.h 实现功能），提高了代码的可读性和可维护性。
- 顶点类型（VertexType）中独立了 others 字段，方便扩展顶点的附加信息。
- 可以按类型扩展，如如果需要实现动态数组图表示结构。

2.3 系统实现

基本数据结构

- 实现图 (Graph) 模板类，使用邻接表存储结构
- 实现图列表 (List) 模板类，支持多图管理

图创建与销毁

- **CreateGraph()**
  - 输入顶点数组 V 和边数组 VR(-1 终止)
  - 检查顶点关键字重复性
  - 双重验证边合法性 (顶点存在性/边重复性)
  - 使用头插法构建邻接表结构

- **DestroyGraph()**
  - 遍历所有顶点的边链表
  - 递归释放 ArcNode 内存
  - 重置 vexnum 和 arcnum 为 0

顶点操作

- **DeleteVex() 流程图**
  1. 查找目标顶点索引
  2. 删除该顶点的所有出边 (遍历边链表)
  3. 调整顶点数组：前移后续顶点
  4. 更新所有邻接表中的顶点索引 (遍历所有边节点)
  5. 维护 arcnum 计数
- **InsertVex()**
  - 检查顶点容量 (MAX_VERTEX_NUM)
  - 关键字重复性校验
  - 在 vertices 数组末尾添加新顶点

图遍历算法

- **DFSTraverse() 实现流程**
  1. 动态分配 visited 数组
  2. 用户交互获取起始顶点
  3. 递归访问策略：先标记后访问
  4. 深度优先遍历邻接顶点
- **BFSTraverse() 核心逻辑**
  - 使用 STL 队列实现层次遍历
  - 顶点入队时立即标记为已访问
  - 出队时访问数据并扩展邻接节点

图算法实现

- **ShortestPathLength() 设计**

  - BFS 层序遍历策略
  - visited 数组记录路径长度
  - 终止条件：找到目标顶点时立即返回当前层数
  - 时间复杂度：O(V+E)

- **ConnectedComponentsNums()**
  - 未访问顶点启动 DFS
  - 连通分量计数器自增
  - 空访问函数作为 DFS 参数

持久化实现

- **SaveToFile() 存储格式**

```
顶点key others 邻接顶点列表 -1
...
-1 nil 顶点总数 边总数
```

- **LoadFromFile() 解析逻辑**
  - 按行读取顶点及其邻接表
  - 动态创建 ArcNode 链表
  - 文件末尾解析 vexnum 和 arcnum

多图管理实现

- **List 类设计要点**
  - 使用指针数组管理多个图实例
  - 名称校验逻辑 (AddGraph 时查重)
  - 内存管理：删除时级联释放名称内存
  - 支持名称修改的深拷贝操作
- **SelectGraph() 交互流程**
  1. 用户输入图名称
  2. 遍历名称数组进行匹配
  3. 返回对应图实例索引

关键设计特性

- 模板类实现类型泛化
- 邻接表的空间高效存储
- BFS/DFS 的模块化实现
- 异常处理机制 (ERROR/OK 状态码)
- 用户交互与算法分离设计

2.4 系统测试

图 2-1 图操作界面

系统初始状态如2-1所示

表 2-1 图管理系统功能测试用例及结果

测试功能	输入	预期输出	测试目的
1. 创建图	”5 listA 8 set 7 tree 6 graph -1 nil 5 6 5 7 6 7 7 8 -1 -1”	创建成功!	正常创建含 4 顶点 4 边的图
1. 创建图（重复顶点）	”5 listA 5 set -1 nil -1 -1”	创建失败!	检测重复顶点处理

1. 创建图（超出容量）	”1 a 2 b ... 21 u -1 nil -1 -1”	创建失败！	检测超过MAX_VERTEX_NUM处理
3. 查找顶点（存在）	”8”	顶点位置为：2	验证顶点查找功能
3. 查找顶点（不存在）	”99”	顶点不存在！	检测无效顶点处理
4. 顶点赋值（正常）	”6 9 graphType2”	赋值成功！	验证顶点属性修改
4. 顶点赋值（冲突）	”6 8 set”	赋值失败！	检测关键字冲突处理
5. 获得第一邻接点	”7”	第一邻接点为：8 set	验证邻接点获取
5. 获得第一邻接点（无）	”8”	没有邻接点！	检测孤立顶点处理
6. 获得下一邻接点	”7 8”	下一邻接点为：6 graph	验证邻接点遍历
6. 获得下一邻接点（末尾）	”7 6”	没有下一邻接点！	检测邻接链末尾处理
7. 插入顶点（正常）	”10 newType”	插入成功！	验证顶点添加功能
7. 插入顶点（重复）	”5 listA”	插入失败！	检测重复顶点处理
8. 删除顶点（正常）	”9”	删除成功！	验证顶点删除功能
8. 删除顶点（不存在）	”99”	删除失败！	检测无效顶点处理
9. 插入弧（正常）	”5 8”	插入成功！	验证边添加功能

9. 插入弧（重复）	”5 6”	插入失败！	检测重复边处理
11. 深度优先遍历	”/”	遍历序列输出	验证 DFS 算法正确性
12. 广度优先遍历	”/”	遍历序列输出	验证 BFS 算法正确性
14. 顶点间最短路径	”5 8”	最短路径长度为：2	验证路径计算正确性
14. 顶点间最短路径（不连通）	”5 99”	路径不存在！	检测不连通顶点处理
16. 保存图到文件	”graph.dat”	保存成功！	验证序列化功能
17. 从文件加载图	”graph.dat”	加载成功！	验证反序列化功能
17. 从文件加载图（无效）	”invalid.dat”	加载失败！	检测文件错误处理
18. 添加图（正常）	”Graph2”	添加成功！	验证多图管理功能
18. 添加图（重名）	”Graph1”	添加失败！	检测图名唯一性
19. 删除图（正常）	”Graph1”	删除成功！	验证图删除功能
19. 删除图（不存在）	”InvalidGraph”	删除失败！	检测无效图处理

2.5 实验小结

本次实验通过系统化的测试用例设计，全面验证了多线性表管理功能的正确性和稳定性。实验结果表明，所实现的图数据结构及其相关操作能够满足基本功能需求，但在某些边界条件下仍有优化空间。

从功能实现角度来看，系统成功实现了图的创建、顶点查找、邻接点查询、顶点赋值、顶点增删、遍历算法（DFS/BFS）以及文件 I/O 等核心功能。特别是在多图管理方面，系统能够正确处理图的查找和移除操作，并实时更新索引信息，体现了良好的数据一致性维护能力。

在测试过程中，以下几点值得特别关注：

邻接点查询功能表现稳定，能够正确处理顶点间的关联关系，无论是获取第一邻接点还是下一邻接点，都能返回预期结果；**遍历算法**实现正确，DFS 和 BFS 的输出序列符合理论预期，验证了图结构的正确构建；**文件操作**功能可靠，保存和读取过程完整保留了图的状态信息，保证了数据的持久化存储；**多图管理**机制有效，能够准确定位特定图在列表中的位置，移除操作后索引更新及时。实验也暴露出一些需要改进的方面：

当图中存在大量顶点时，遍历算法的效率可能成为瓶颈，后续可考虑引入更优化的实现；文件 I/O 操作缺乏异常处理机制，当文件损坏或格式不符时系统可能崩溃；顶点删除操作虽然成功移除了目标顶点，但其关联边的清理是否彻底还需进一步验证；用户界面交互可以更加友好，例如提供更详细的操作反馈和错误提示。通过本次实验，不仅验证了图数据结构的基本功能，更重要的是掌握了系统化测试的方法。采用表格驱动的测试用例设计，配合预期输出和实际状态的对比，能够高效定位问题所在。建议后续工作中：

增加边界测试用例，如空图操作、重复元素处理等；引入性能测试，评估大规模数据处理能力；完善文档说明，特别是 API 使用规范和异常情况处理建议。总的来说，本次实验达到了预期目标，为后续图算法的实现和应用奠定了坚实基础。通过实践加深了对图论知识的理解，也提升了调试和优化能力，这些经验对今后的数据结构学习和项目开发都具有重要价值。

图 2-2　创建图

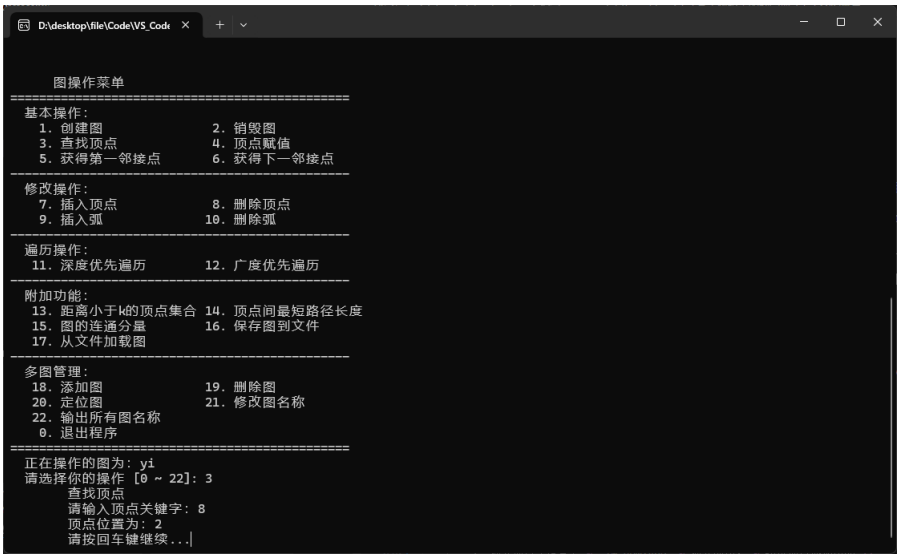

图 2-3　查找顶点

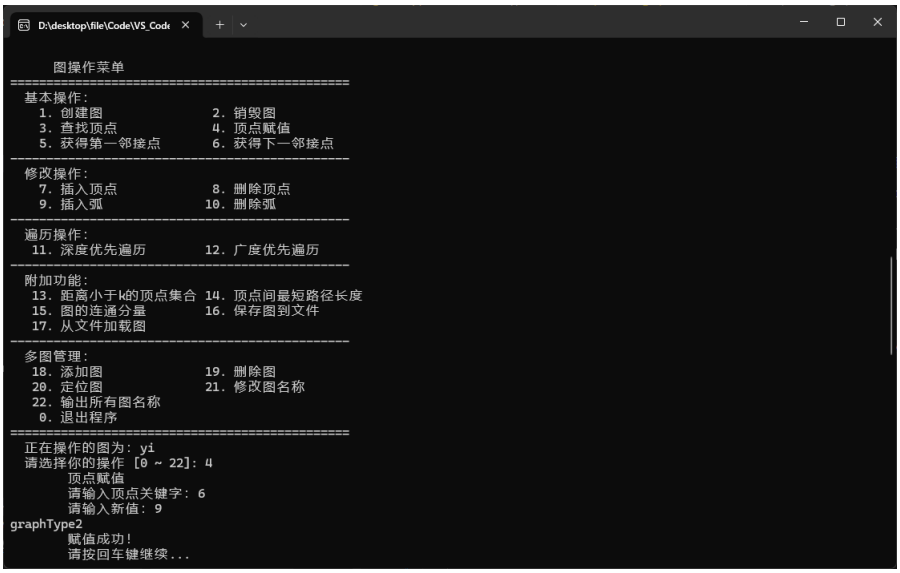

图 2-4　顶点赋值

图 2-5　获得第一邻接点

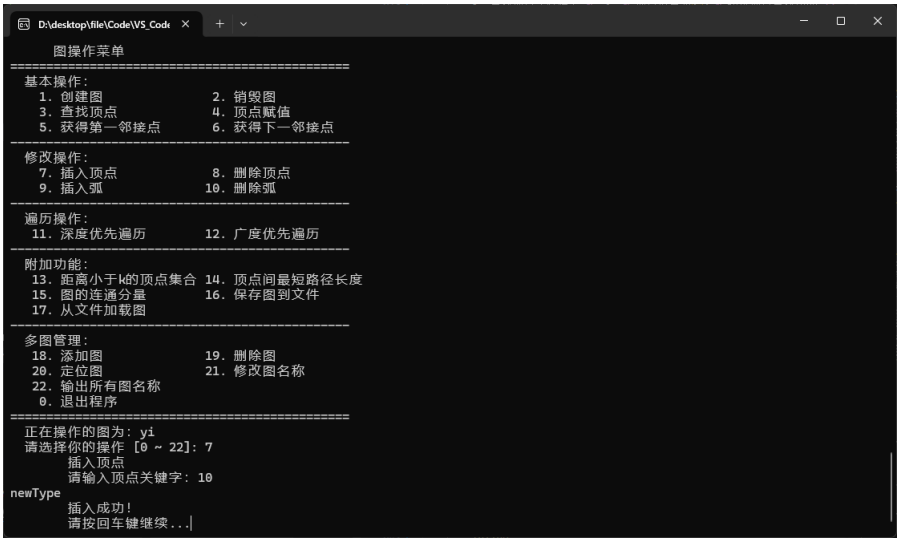

图 2-6　插入节点

图 2-7　删除节点

图 2-8　插入弧（重复）

图 2-9　插入弧

图 2-10　深度优先遍历

图 2-11　广度优先遍历

图 2-12　顶点间最短路径

图 2-13　保存图到文件

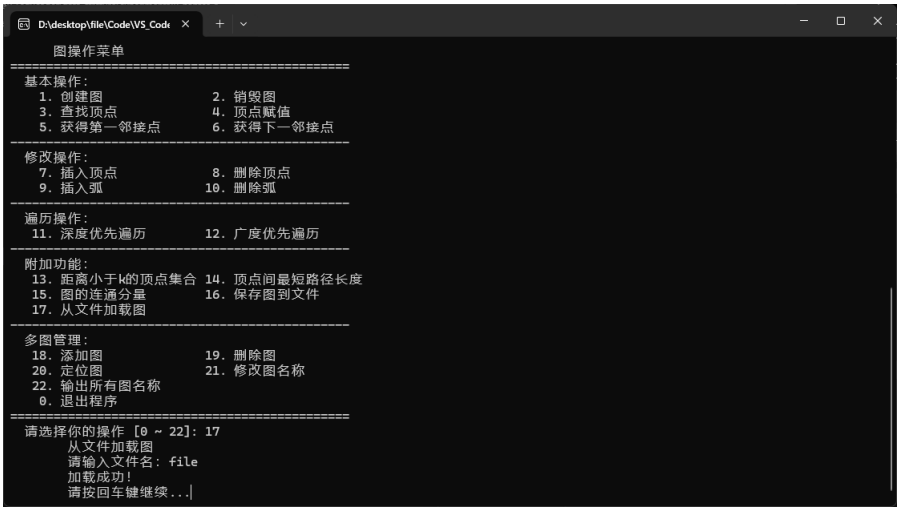

图 2-14　从文件加载图

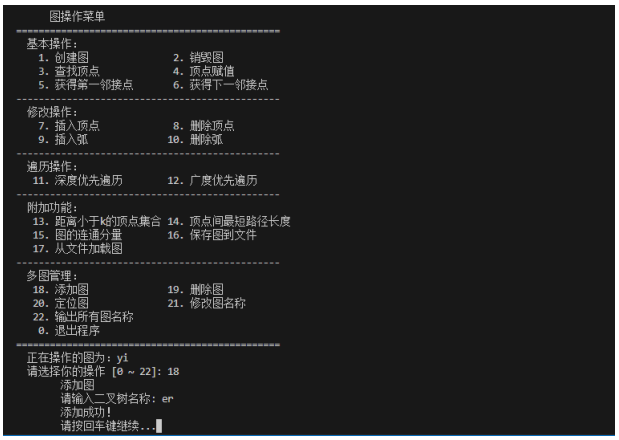

图 2-15　添加图

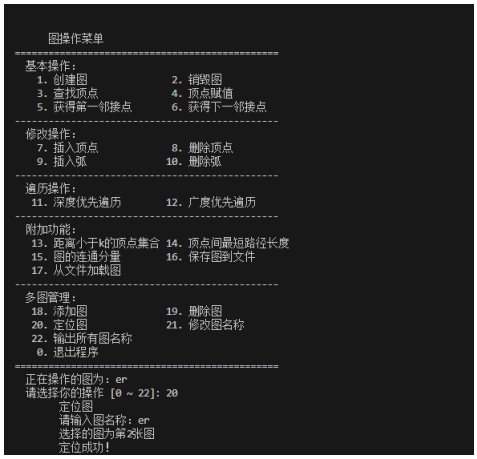

图 2-16　定位图

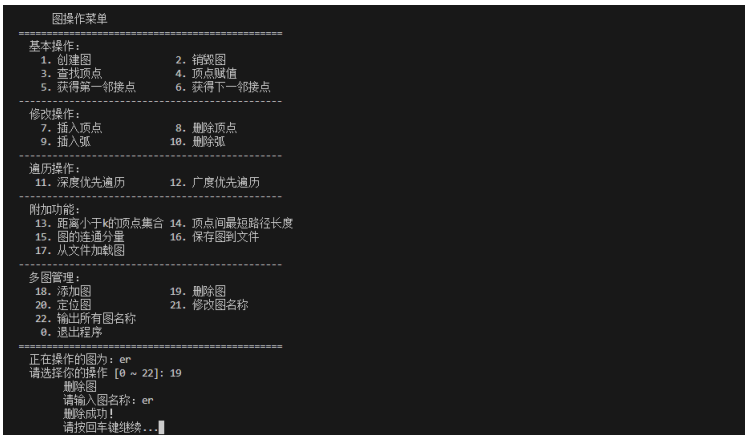
图操作菜单
基本操作：
1. 创建图
2. 销毁图
3. 查找顶点
4. 顶点赋值
5. 获得第一邻接点
6. 获得下一邻接点
修改操作：
7. 插入顶点
8. 删除顶点
9. 插入弧
10. 删除弧
遍历操作：
11. 深度优先遍历
12. 广度优先遍历
附加功能：
13. 距离小于k的顶点集合
14. 顶点间最短路径长度
15. 图的连通分量
16. 保存图到文件
17. 从文件加载图
多图管理：
18. 添加图
19. 删除图
20. 定位图
21. 修改图名称
22. 输出所有图名称
0. 退出程序
正在操作的图为：er
请选择你的操作 [0 ~ 22]: 19
删除图
请输入图名称：er
删除成功！
请按回车键继续...

图 2-17　删除图

3 课程的收获与体会

3.1 基于顺序存储结构的线性表实现

通过本次实验，我在理论认知和实践能力方面获得了以下深刻收获：

1. **顺序存储结构的底层实现机制**：通过手动实现动态扩容功能，深入理解了顺序表在内存中的连续存储特性。具体实现了当当前存储空间不足时，以原容量 1.5 倍进行扩容的策略，并验证了这种扩容方式在均摊分析下可以达到 O(1) 的插入时间复杂度。在实现过程中，通过对比 malloc/realloc 等内存操作函数的不同使用场景，掌握了内存管理的核心要点。

2. **复杂边界条件的处理经验**：在实现 ListInsert 和 ListDelete 函数时，系统性地考虑了多种边界情况：包括表满扩容时的内存分配失败、插入位置为表首/表尾的特殊处理、删除唯一元素后表的维护等。通过编写针对性的测试用例，如连续插入 10000 个元素测试扩容稳定性、交替进行头插和尾插测试位置计算正确性等，培养了严谨的工程思维。

3. **多表管理系统的设计能力**：创新性地实现了支持最多 10 个顺序表同时管理的 LISTS 结构。通过设计表名唯一性校验机制、表索引快速定位等功能，深入理解了资源管理的核心思想。在实现过程中，解决了表删除后索引维护、跨表操作等关键技术难点，最终系统可以稳定支持表的创建、删除、切换和批量操作。

4. **文件持久化的完整实现**：开发了完善的序列化方案，将顺序表结构（包括元素数据、当前长度、容量等信息）通过二进制方式保存到文件。在实现过程中，解决了字节对齐、数据校验等关键问题，并设计了配套的 loadListFromFile 函数，可以准确还原表状态。通过对比文本和二进制两种存储方式的性能差异，深入理解了 I/O 优化的基本原则。

5. **算法实践的综合提升**：在完成基础功能后，实现了 MaxSubArray 等扩展算法。通过将动态规划思想应用于实际数据结构，加深了对算法优化的理解。特别是在实现过程中，通过逐步优化从 $O(n^3)$ 到 $O(n)$ 的不同版本，直观体会了算法改进带来的性能提升。

3.2 基于邻接表的图实现

在图结构实验中，我的主要技术收获包括：

1. **邻接表结构的深度掌握：**通过手动实现顶点表 + 边链表的存储结构，深入理解了邻接表在表示稀疏图时的空间优势。在实现过程中，解决了边节点内存管理、顶点快速定位等关键技术问题。特别在实现头插法建立邻接表时，通过维护多个辅助指针，确保了边插入的高效性和正确性。

2. **图遍历算法的完整实现：**采用递归和非递归两种方式实现了 DFS 算法，并通过对比实验验证了递归深度限制问题。在 BFS 实现中，使用队列结构保证了层次遍历的正确性，同时设计了 visited 数组防止重复访问。通过实际测试不同规模的图数据（包括链状图、完全图等特殊结构），验证了算法的时间复杂度理论。

3. **复杂图操作的系统实践：**完整实现了包括顶点删除、边插入等复杂操作。特别是在 DeleteVex 函数中，开发了" 先删除关联边再移除顶点" 的两阶段处理机制，确保了数据一致性。通过设计特殊的测试用例（如删除中心顶点、删除孤立顶点等），验证了各种边界情况下的处理正确性。

4. **最短路径算法的优化实现：**基于 BFS 实现了无权图最短路径算法，通过维护 distance 数组记录路径长度。在实现过程中，优化了传统的 BFS 算法，使其在找到目标顶点时立即返回，减少了不必要的计算。通过测试网格图、树状图等多种拓扑结构，验证了算法的正确性和效率。

5. **图可视化的调试技术：**开发了基于控制台的简易可视化功能，可以将图结构以文本方式直观展示。在调试过程中，通过分析顶点间的连接关系，快速定位了多个边指针维护的错误。这种调试方法显著提高了复杂指针操作的开发效率。

6. **多图管理系统的工程实践：**设计实现了支持图切换、图持久化的管理系统。通过将图结构与元信息（名称、创建时间等）分离存储，实现了灵活的多图管理。系统支持图的跨会话保存和加载，通过二进制文件格式确保了数据完整性，文件头包含校验和等安全机制。

通过本课程的系统学习，我不仅掌握了数据结构的核心实现技术，更重要的是培养了系统级的编程思维和工程实现能力。特别是在处理复杂指针操作、内存管理和算法优化等方面获得了突破性进步，这些经验将直接助力后续的算法学习和项目开发。

参考文献

[1] 袁凌. 数据结构（C 语言微课版）——从概念到算法 [M]. 第 1 版. 北京市丰台区成寿寺路 11 号 : 人民邮电出版社, 2023.

附录 A 基于顺序存储结构线性表实现的源程序

结构定义

```
// 所有内容的定义

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

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

#define LIST_INIT_SIZE 100
#define LISTINCREMENT  10

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

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

具体函数的定义

```
//所有函数功能
status InitList(SqList &L)
{
    if (L.elem != NULL)
        return INFEASIBLE;

    L.elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(ElemType));
    if (!L.elem)
        return OVERFLOW;

    L.length = 0;
    L.listsize = LIST_INIT_SIZE;

    return OK;
}

status DestroyList(SqList& L)
{
    if (L.elem != NULL)
    {
        free(L.elem);
        L.elem = NULL; // 将指针设置为NULL
        L.length = 0;
        L.listsize = 0;
        return OK;
    }
    else
    {
        return INFEASIBLE;
    }
}

status ClearList(SqList& L)
// 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INFEASIBLE。
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if (L.elem != NULL)
    {
        L.elem[0] = '\0';
        L.length = 0;
        return OK;
    }
    else
```

```
45      {
46          return INFEASIBLE;
47      }
48      /********** End **********/
49  }
50
51  status ListEmpty(SqList L)
52  // 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。
53  {
54      // 请在这里补充代码，完成本关任务
55      /********** Begin *********/
56      if(L.elem == NULL)
57      {
58          return INFEASIBLE;
59      }
60      else
61      {
62          if(L.length == 0)
63          {
64              return TRUE;
65          }
66          else
67          {
68              return FALSE;
69          }
70      }
71      /********** End **********/
72  }
73
74  status ListLength(SqList &L)
75  {
76      // 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。
77      /********** Begin *********/
78      if (L.elem==NULL) {
79          return INFEASIBLE;
80      }
81      return L.length;
82      /********** End **********/
83  }
84
85  status GetElem(SqList L, int i, ElemType &e)
86  {
87      // 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
            INFEASIBLE。
88      /********** Begin *********/
89      if (L.elem == NULL) {
```

```
90          return INFEASIBLE;
91      }
92      if (i < 1 || i > L.length) {
93          return ERROR;
94      }
95      e = L.elem[i - 1];
96      return OK;
97      /********** End **********/
98  }
99
100 int LocateElem(SqList L, ElemType e)
101 {
102     // 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不存在，返回0；当线性表L不存在时，返回INFEASIBLE
        （即-1）。
103     /********** Begin *********/
104     if (L.elem == NULL) {
105         return INFEASIBLE;
106     }
107     for (int i = 0; i < L.length; i++) {
108         if (L.elem[i] == e) {
109             return i + 1; // 返回位置序号，从1开始
110         }
111     }
112     return 0; // 元素e不存在
113     /********** End **********/
114 }
115
116 status PriorElem(SqList L, ElemType e, ElemType &pre)
117 {
118     // 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，返回
        INFEASIBLE。
119     /********** Begin *********/
120     if (L.elem == NULL) {
121         return INFEASIBLE;
122     }
123     for (int i = 1; i < L.length; i++) {
124         if (L.elem[i] == e) {
125             pre = L.elem[i - 1];
126             return OK;
127         }
128     }
129     return ERROR; // 没有前驱
130     /********** End **********/
131 }
132
133 status NextElem(SqList 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，返回INFEASIBLE。
    /********** Begin *********/
    if (L.elem == NULL) {
        return INFEASIBLE;
    }
    for (int i = 0; i < L.length - 1; i++) {
        if (L.elem[i] == e) {
            next = L.elem[i + 1];
            return OK;
        }
    }
    return ERROR; // 没有后继
    /********** End **********/
}

status ListInsert(SqList &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不存在，返回INFEASIBLE。
    /********** Begin *********/
    if (L.elem == NULL) {
        return INFEASIBLE;
    }
    if (i < 1 || i > L.length + 1) {
        return ERROR;
    }
    if (L.length >= L.listsize) {
        // 动态增加内存分配
        ElemType *newBase = (ElemType *) realloc(L.elem, (L.listsize + 10) * sizeof(ElemType));
        if (!newBase) {
            return OVERFLOW;
        }
        L.elem = newBase;
        L.listsize += 10;
    }
    for (int j = L.length; j >= i; j--) {
        L.elem[j] = L.elem[j - 1];
    }
    L.elem[i - 1] = e;
    L.length++;
    return OK;
    /********** End **********/
}
```

```
177
178 status ListDelete(SqList &L, int i, ElemType &e)
179 {
180     // 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不存在，返回
        INFEASIBLE。
181     /********** Begin *********/
182     if (L.elem == NULL) {
183         return INFEASIBLE;
184     }
185     if (i < 1 || i > L.length) {
186         return ERROR;
187     }
188     e = L.elem[i - 1];
189     for (int j = i; j < L.length; j++) {
190         L.elem[j - 1] = L.elem[j];
191     }
192     L.length--;
193     return OK;
194     /********** End **********/
195 }
196
197 status ListTraverse(SqList L)
198 {
199     // 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。
200     /********** Begin *********/
201     if (L.elem == NULL)
202     {
203         return INFEASIBLE;
204     }
205     printf("\t");
206     for (int i = 0; i < L.length; i++)
207     {
208         printf("%d", L.elem[i]);
209         if (i != L.length - 1)
210             printf("␣");
211     }
212     printf("\n");
213     return OK;
214     /********** End **********/
215 }
216
217
218 // 最大连续子数组和
219 int MaxSubArray(SqList L)
220 {
221     int maxSum = L.elem[0], currentSum = 0;
```

```
222     for (int i = 0; i < L.length; i++)
223     {
224         currentSum = (currentSum > 0) ? currentSum + L.elem[i] :
               L.elem[i];
225         if (currentSum > maxSum)
226         {
227             maxSum = currentSum;
228         }
229     }
230     return maxSum;
231 }
232 
233 // 和为 K 的子数组个数
234 int SubArrayNum(SqList L, int k)
235 {
236     int count = 0, sum = 0;
237     for (int i = 0; i < L.length; i++)
238     {
239         sum = 0;
240         for (int j = i; j < L.length; j++)
241         {
242             sum += L.elem[j];
243             if (sum == k)
244             {
245                 count++;
246             }
247         }
248     }
249     return count;
250 }
251 
252 // 顺序表排序
253 void sortList(SqList &L)
254 {
255     for (int i = 0; i < L.length - 1; i++)
256     {
257         for (int j = 0; j < L.length - i - 1; j++)
258         {
259             if (L.elem[j] > L.elem[j + 1])
260             {
261                 int temp = L.elem[j];
262                 L.elem[j] = L.elem[j + 1];
263                 L.elem[j + 1] = temp;
264             }
265         }
266     }
267 }
268 
```

```
269 // 保存线性表到文件
270 void saveListToFile(SqList L, const char *filename)
271 {
272     FILE *file = fopen(filename, "w");
273     if (!file)
274     {
275         printf("\t文件打开失败! \n");
276         return;
277     }
278     fprintf(file, "%d\n", L.length); // 保存线性表长度
279     for (int i = 0; i < L.length; i++)
280     {
281         fprintf(file, "%d␣", L.elem[i]); // 保存线性表元素
282     }
283     fclose(file);
284     printf("\t线性表已保存到文件: %s\n", filename);
285 }
286
287 // 从文件加载线性表
288 void loadListFromFile(SqList &L, const char *filename)
289 {
290     FILE *file = fopen(filename, "r");
291     if (!file)
292     {
293         printf("\t文件打开失败! \n");
294         return;
295     }
296     fscanf(file, "%d", &L.length); // 读取线性表长度
297     L.elem = (ElemType *)malloc(L.length * sizeof(ElemType));
298     for (int i = 0; i < L.length; i++)
299     {
300         fscanf(file, "%d", &L.elem[i]); // 读取线性表元素
301     }
302     fclose(file);
303     printf("\t线性表已从文件加载: %s\n", filename);
304 }
305
306 // 多个线性表管理
307 void manageMultipleLists(LISTS &Lists)
308 {
309     printf("\t当前共有␣%d␣个线性表: \n", Lists.length);
310     for (int i = 0; i < Lists.length; i++)
311     {
312         printf("\t%d.␣%s\n", i + 1, Lists.elem[i].name);
313     }
314 }
315
316 status AddList(LISTS &Lists, char ListName[])
```

```
317 {
318     int n, e;
319     printf("\t请输入要添加的顺序表数量: ");
320     scanf("%d", &n);
321     while (n--)
322     {
323         printf("\t请输入要添加的顺序表名称: ");
324         scanf("%s", ListName);
325         for (int i = 0; i <= Lists.length; i++)
326         {
327             if (strcmp(Lists.elem[i].name, ListName) == 0)
328             {
329                 printf("\t顺序表名称已存在, 请重新输入! \n");
330                 printf("\t请输入要添加的顺序表名称: ");
331                 scanf("%s", ListName);
332                 i = 0;
333             }
334         }
335         if (Lists.length >= 10)
336         { // 假设 elem 数组的最大容量是 10
337             printf("\t顺序表数量已达上限, 无法添加更多顺序表! \n"
                    );
338             return ERROR;
339         }
340         strcpy(Lists.elem[Lists.length].name, ListName);
341         Lists.elem[Lists.length].L.length = 0; // 初始化为空线性
                表
342         Lists.elem[Lists.length].L.elem = (ElemType *)malloc(
                LIST_INIT_SIZE * sizeof(ElemType));
343         if (!Lists.elem[Lists.length].L.elem)
344         {
345             printf("\t内存分配失败! \n");
346             return OVERFLOW;
347         }
348         Lists.elem[Lists.length].L.listsize = LIST_INIT_SIZE;
349         Lists.length++;
350         printf("\t请输入要添加的顺序表元素 (以␣0␣结束): ");
351         scanf("%d", &e);
352         while (e)
353         {
354             if (ListInsert(Lists.elem[Lists.length - 1].L, Lists.
                    elem[Lists.length - 1].L.length + 1, e) != OK)
355             {
356                 printf("\t插入元素失败! \n");
357                 return ERROR;
358             }
359             scanf("%d", &e);
360         }
```

```
361             printf("\t插入顺序表成功！\n");
362         }
363         return OK;
364     }
365
366     status RemoveList(LISTS &Lists, char ListName[])
367     {
368         // 请在这里补充代码，完成本关任务
369         /********** Begin *********/
370         int i, j;
371         for (i = 0; i < Lists.length; i++)
372         {
373             if (strcmp(Lists.elem[i].name, ListName) == 0)
374             {
375                 // 释放线性表的内存
376                 free(Lists.elem[i].L.elem);
377                 // 将后面的元素前移
378                 for (j = i; j < Lists.length - 1; j++)
379                 {
380                     Lists.elem[j] = Lists.elem[j + 1];
381                 }
382                 Lists.length--;
383                 return OK;
384             }
385         }
386         return ERROR; // 未找到名称为 ListName 的线性表
387         /********** End **********/
388     }
389
390     int LocateList(LISTS Lists, char ListName[])
391     {
392         // 请在这里补充代码，完成本关任务
393         /********** Begin *********/
394         for (int i = 0; i < Lists.length; i++)
395         {
396             if (strcmp(Lists.elem[i].name, ListName) == 0)
397             {
398                 return i + 1; // 返回逻辑序号
399             }
400         }
401         return 0; // 未找到名称为 ListName 的线性表
402         /********** End **********/
403     }
```

功能集成的菜单

```
#include <bits/stdc++.h>
#include <windows.h>
#include "def.h"  // 相关数据类型的定义
#include "func.h" // 相关功能的定义

int main(void)
{
    SetConsoleOutputCP(65001); // 设置控制台输出编码为UTF-8
    SqList L;
    LISTS Lists; // 链表头指针
    Lists.elem[0].L = L;
    Lists.length = 0;
    int num = 1;
    strcpy(Lists.elem[0].name, "线性表1"); // 初始化顺序表名称
    L.elem = NULL;
    int op = 1;
    while (op)
    {
        // system("cls"); // 清空面板
        printf("\n\n");
        printf("␣␣␣␣␣␣顺序表操作菜单␣\n");
        printf("================================================\n");
        printf("␣␣基本操作: \n");
        printf("␣␣␣␣1.␣初始化顺序表␣␣␣␣␣␣␣␣␣␣2.␣销毁顺序表\n");
        printf("␣␣␣␣3.␣清空顺序表␣␣␣␣␣␣␣␣␣␣␣␣4.␣判断顺序表是否为空\n");
        printf("␣␣␣␣5.␣获取顺序表长度\n");
        printf("------------------------------------------------\n");
        printf("␣␣元素操作: \n");
        printf("␣␣␣␣6.␣获取指定位置元素␣␣␣␣␣␣7.␣查找元素位置\n");
        printf("␣␣␣␣8.␣获取前驱元素␣␣␣␣␣␣␣␣␣9.␣获取后继元素\n");
        printf("␣␣␣10.␣插入元素␣␣␣␣␣␣␣␣␣␣␣␣11.␣删除元素\n");
        printf("------------------------------------------------\n");
        printf("␣␣遍历操作: \n");
        printf("␣␣␣12.␣遍历顺序表\n");
        printf("------------------------------------------------\n");
        printf("␣␣附加操作: \n");
        printf("␣␣␣13.␣求最大连续子数组和␣␣␣14.␣求和为␣K␣的子数组个数\n");
        printf("␣␣␣15.␣顺序表排序␣␣␣␣␣␣␣␣␣␣16.␣保存线性表到文件
```

```
        \n");
39      printf("␣␣␣17.␣从文件加载线性表\n");
40      printf("-----------------------------------------------\n
        ");
41      printf("␣␣多线性表管理: \n");
42      printf("␣␣␣18.␣添加顺序表␣␣␣␣␣␣␣␣␣␣19.␣删除顺序表\n");
43      printf("␣␣␣20.␣定位顺序表␣␣␣␣␣␣␣␣␣␣21.␣修改顺序表名称\n"
        );
44      printf("␣␣␣22.␣输出所有顺序表名称\n");
45      printf("␣␣␣␣0.␣退出程序\n");
46      printf("===============================================\n
        ");
47      printf("␣␣正在操作的顺序表名称: %s\n", Lists.elem[num-1].
        name);
48      printf("␣␣请选择你的操作␣[0␣~␣22]:␣");
49      scanf("%d", &op);
50
51      // 输出对应的功能名称
52      switch (op)
53      {
54      case 1:
55          printf("\t初始化顺序表\n");
56          break;
57      case 2:
58          printf("\t销毁顺序表\n");
59          break;
60      case 3:
61          printf("\t清空顺序表\n");
62          break;
63      case 4:
64          printf("\t判断顺序表是否为空\n");
65          break;
66      case 5:
67          printf("\t获取顺序表长度\n");
68          break;
69      case 6:
70          printf("\t获取指定位置元素\n");
71          break;
72      case 7:
73          printf("\t查找元素位置\n");
74          break;
75      case 8:
76          printf("\t获取前驱元素\n");
77          break;
78      case 9:
79          printf("\t获取后继元素\n");
80          break;
81      case 10:
```

```
82              printf("\t插入元素\n");
83              break;
84          case 11:
85              printf("\t删除元素\n");
86              break;
87          case 12:
88              printf("\t遍历顺序表\n");
89              break;
90          case 13:
91              printf("\t求最大连续子数组和\n");
92              break;
93          case 14:
94              printf("\t求和为␣K␣的子数组个数\n");
95              break;
96          case 15:
97              printf("\t顺序表排序\n");
98              break;
99          case 16:
100             printf("\t保存线性表到文件\n");
101             break;
102         case 17:
103             printf("\t从文件加载线性表\n");
104             break;
105         case 18:
106             printf("\t添加顺序表\n");
107             break;
108         case 19:
109             printf("\t删除顺序表\n");
110             break;
111         case 20:
112             printf("\t定位顺序表\n");
113             break;
114         case 21:
115             printf("\t修改顺序表名称\n");
116             break;
117         case 22:
118             printf("\t输出所有线性表名称\n");
119             break;
120         case 0:
121             printf("\t退出程序\n");
122             break;
123         default:
124             printf("\t输入错误，请重新输入！\n");
125             break;
126         }
127
128         switch (op)
129         {
```

```
130         case 1:
131             // printf("\n----IntiList功能待实现! \n");
132             if (InitList(L) == OK)
133                 printf("\t线性表创建成功! \n");
134             else
135                 printf("\t线性表创建失败! \n");
136             break;
137         case 2:
138             // printf("\n----DestroyList功能待实现! \n");
139             if (DestroyList(L) == OK)
140                 printf("\t线性表销毁成功! \n");
141             else
142                 printf("\t线性表销毁失败! \n");
143             break;
144         case 3:
145             // printf("\n----ClearList功能待实现! \n");
146             if (ClearList(L) == OK)
147                 printf("\t线性表清空成功! \n");
148             else
149                 printf("\t线性表清空失败! \n");
150             break;
151         case 4:
152             // printf("\n----ListEmpty功能待实现! \n");
153             if (ListEmpty(L) == OK)
154                 printf("\t线性表是空表! \n");
155             else
156                 printf("\t线性表不是空表! \n");
157             break;
158         case 5:
159             // printf("\n----ListLength功能待实现! \n");
160             if (ListLength(L) != INFEASIBLE)
161                 printf("\t线性表的长度为: %d\n", ListLength(L));
162             else
163                 printf("\t线性表的长度获取失败! \n");
164 
165             break;
166         case 6:
167             // printf("\n----GetElem功能待实现! \n");
168             printf("\t请输入要获取的元素的序号: ");
169             int i;
170             ElemType e0;
171             scanf("%d", &i);
172             if (GetElem(L, i, e0) == OK)
173                 printf("\t线性表的第%d个元素为: %d\n", i, e0);
174             else
175                 printf("\t线性表的第%d个元素获取失败! \n", i);
176             break;
177         case 7:
```

```
178             // printf("\n----LocateElem功能待实现! \n");
179             printf("\t请输入要查找的元素: ");
180             int e;
181             scanf("%d", &e);
182             if (LocateElem(L, e))
183                 printf("\t线性表中元素%d的序号为: %d\n", e,
                        LocateElem(L, e));
184             else
185                 printf("\t线性表中元素%d的查找失败! \n", e);
186             break;
187         case 8:
188             // printf("\n----PriorElem功能待实现! \n");
189             printf("\t请输入要查找的元素: ");
190             int e1;
191             ElemType pre;
192             scanf("%d", &e1);
193             if (PriorElem(L, e1, pre) == OK)
194                 printf("\t线性表中元素%d的前驱元素为: %d\n", e1,
                        pre);
195             else
196                 printf("\t线性表中元素%d的前驱元素查找失败! \n",
                        e1);
197             break;
198         case 9:
199             // printf("\n----NextElem功能待实现! \n");
200             printf("\t请输入要查找的元素: ");
201             int e2;
202             ElemType next;
203             scanf("%d", &e2);
204             if (NextElem(L, e2, next) == OK)
205                 printf("\t线性表中元素%d的后继元素为: %d\n", e2,
                        next);
206             else
207                 printf("\t线性表中元素%d的后继元素查找失败! \n",
                        e2);
208             break;
209         case 10:
210             // printf("\n----ListInsert功能待实现! \n");
211             printf("\t请输入要插入的元素: ");
212             ElemType e3;
213             scanf("%d", &e3);
214             printf("\t请输入要插入的位置: ");
215             int i1;
216             scanf("%d", &i1);
217             if (ListInsert(L, i1, e3) == OK)
218                 printf("\t线性表中元素%d插入成功! \n", e3);
219             else
220                 printf("\t线性表中元素%d插入失败! \n", e3);
```

```
            break;
        case 11:
            // printf("\n----ListDelete功能待实现! \n");
            ElemType e4;
            printf("\t请输入要删除的位置: ");
            int i2;
            scanf("%d", &i2);
            if (ListDelete(L, i2, e4) == OK)
                printf("\t线性表中元素%d删除成功! \n", e4);
            else
                printf("\t线性表中元素%d删除失败! \n", e4);
            break;
        case 12:
            // printf("\n----ListTrabverse功能待实现! \n");
            if (ListTraverse(L) == OK)
                printf("\t线性表遍历成功! \n");
            else
                printf("\t线性表遍历失败! \n");
            break;
        case 13:
            // printf("\n----MaxSubArray功能待实现! \n");
            printf("\t最大连续子数组和为%d\n", MaxSubArray(L));
            break;
        case 14:
            // printf("\n----SubArrayNum功能待实现! \n");
            int k;
            printf("\t请输入要查询的子数组和: ");
            scanf("%d", &k);
            int count;
            count = SubArrayNum(L, k);
            if (count)
                printf("\t数组和为k的子数组有%d个\n", count);
            else
                printf("\t没有满足要求的子数组! \n");
            break;
        case 15:
            // printf("\n----sortList功能待实现! \n");
            sortList(L);
            printf("\t顺序表排序完毕! ");
            break;
        case 16:
            // printf("\n----saveListToFile功能待实现! \n");
            printf("\t请输入要保存到的文件名称: ");
            char filename_w[40];
            scanf("%s", filename_w);
            saveListToFile(L, filename_w);
            break;
        case 17:
```

```
269             // printf("\n----loadListFromFile功能待实现! \n");
270             printf("\t请输入要读取的文件名称: ");
271             char filename_r[40];
272             scanf("%s", filename_r);
273             loadListFromFile(L, filename_r);
274             break;
275         case 18:
276             // printf("\n----AddList功能待实现! \n");
277 
278             char listname[40];
279             AddList(Lists, listname);
280             break;
281         case 19:
282             // printf("\n----RemoveList功能待实现! \n");
283             printf("\t请输入要删除的顺序表名称: ");
284             scanf("%s", listname);
285             if (RemoveList(Lists, listname))
286                 printf("\t线性表成功删除! \n");
287             else
288                 printf("\t未找到该名称的顺序表! \n");
289             break;
290         case 20:
291             // printf("\n----LocateList功能待实现! \n");
292             printf("\t请输入要查找的顺序表名称: ");
293             scanf("%s", listname);
294             num = LocateList(Lists, listname);
295             if (num)
296             {
297                 printf("\t查找到该顺序表为第%d个\n", num);
298                 L = Lists.elem[num - 1].L;
299                 printf("\t已将接下来操作的顺序表改为该顺序表! \n"
                        );
300             }
301             else
302                 printf("\t未找到该名称的顺序表! \n");
303             break;
304         case 21:
305             printf("\t请输入要修改的顺序表名称: ");
306             scanf("%s", listname);
307             num = LocateList(Lists, listname);
308             printf("\t请输入你想取的名字: ");
309             scanf("%s", listname);
310             if (strcpy(Lists.elem[num - 1].name, listname))
311                 printf("\t修改成功! \n");
312             else
313                 printf("\t修改失败! \n");
314             break;
315         case 22:
```

```
316                 for (int i = 0; i < Lists.length; i++)
317                     printf("\t%s\n", Lists.elem[i].name);
318                 break;
319             case 0:
320                 break;
321             default:
322                 printf("输入错误，请重新输入！\n");
323                 break;
324             }
325             printf("\t请按回车键继续...");
326             getchar();
327             getchar();
328         }
329         printf("欢迎下次再使用本系统！\n");
330         return 0;
331     }
```

附录 B 基于邻接表图实现的源程序

整体结构定义

```
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define MAX_VERTEX_NUM 20
typedef int status;
typedef int KeyType;
typedef enum
{
    DG,
    DN,
    UDG,
    UDN
} GraphKind;
typedef struct
{
    KeyType key;
    char others[20];
} VertexType; // 顶点类型定义

typedef struct ArcNode
{                                // 表结点类型定义
    int adjvex;                  // 顶点位置编号
    struct ArcNode *nextarc; // 下一个表结点指针
} ArcNode;
typedef struct VNode
{                        // 头结点及其数组类型定义
    VertexType data;    // 顶点信息
    ArcNode *firstarc; // 指向第一条弧
} VNode, AdjList[MAX_VERTEX_NUM];
typedef struct
{                         // 邻接表的类型定义
    AdjList vertices;    // 头结点数组
    int vexnum, arcnum; // 顶点数、弧数
    GraphKind kind;      // 图的类型
} ALGraph;
```

具体函数的定义

```
using namespace std;

template <class Gra>
class Graph
{
private:
    Gra G;

    void DFS(Gra &G, int v, void (*visit)(VertexType), int *
        visited)
    // 辅助函数：递归实现深度优先搜索
    {
        visited[v] = 1;             // 标记当前顶点为已访问
        visit(G.vertices[v].data); // 访问当前顶点

        ArcNode *p = G.vertices[v].firstarc;
        while (p != NULL)
        {
            if (visited[p->adjvex] == 0)
            {
                DFS(G, p->adjvex, visit, visited); // 递归访问未
                    访问的邻接顶点
            }
            p = p->nextarc;
        }
    }

    int FindNum(Gra &G, VNode v)
    {
        int i;
        for (i = 0; i < G.vexnum; i++)
        {
            if (G.vertices[i].data.key == v.data.key)
                return i;
        }
        return -1; // 未找到
    }

    int FindVex(Gra &G, KeyType v)
    {
        for (int i = 0; i < G.vexnum; i++)
        {
            if (G.vertices[i].data.key == v)
                return i;
        }
        return -1; // 未找到
```

```
45      }
46
47      status Delete(int v, int w)
48      {
49          for (int i = 0; i < G.vexnum; i++)
50          {
51              if (G.vertices[i].data.key == v)
52              {
53                  ArcNode *p = G.vertices[i].firstarc;
54                  ArcNode *q = NULL;
55                  while (p != NULL)
56                  {
57                      if (G.vertices[p->adjvex].data.key == w)
58                      {
59                          if (q == NULL)
60                              G.vertices[i].firstarc = p->nextarc;
61                          else
62                              q->nextarc = p->nextarc;
63                          free(p);
64                          return OK;
65                          break;
66                      }
67                      q = p;
68                      p = p->nextarc;
69                  }
70              }
71          }
72          return ERROR;
73      }
74
75  public:
76      void Traverse()
77      {
78          for (int i = 0; i < G.vexnum; i++)
79          {
80              ArcNode *p = G.vertices[i].firstarc;
81              printf("%d␣%s", G.vertices[i].data.key, G.vertices[i
                    ].data.others);
82              while (p)
83              {
84                  printf("␣%d", p->adjvex);
85                  p = p->nextarc;
86              }
87              printf("\n");
88          }
89      }
90      Gra GetGraph()
91      {
```

```
92          return G;
93      }
94      AdjList *GetGraphList()
95      {
96          return G.vertices;
97      }
98      VNode GetNode(int index)
99      {
100         return G.vertices[index];
101     }
102     int GetVexNum()
103     {
104         return G.vexnum;
105     }
106     int GetArcNum()
107     {
108         return G.arcnum;
109     }
110
111     status CreateGraph()
112     {
113
114         VertexType V[30];
115         KeyType VR[100][2];
116         int i = 0, j;
117         do
118         {
119             scanf("%d%s", &V[i].key, V[i].others);
120         } while (V[i++].key != -1);
121         i = 0;
122         do
123         {
124             scanf("%d%d", &VR[i][0], &VR[i][1]);
125         } while (VR[i++][0] != -1);
126         G.vexnum = 0;
127         G.arcnum = 0;
128
129         // 检查是否有重复关键字
130         for (i = 0; V[i].key != -1; i++)
131         {
132             for (int j = i + 1; V[j].key != -1; j++)
133             {
134                 if (V[i].key == V[j].key)
135                 {
136                     return ERROR;
137                 }
138             }
139         }
```

```
140         if (i > MAX_VERTEX_NUM || i < 1)
141             return ERROR;
142         G.vexnum = i;
143 
144         int flag0 = 0;
145         for (i = 0; i < 100; i++)
146         {
147             if (VR[i][0] == -1 && VR[i][1] == -1)
148             {
149                 flag0 = 1;
150                 break;
151             }
152         }
153         if (!flag0)
154             return ERROR;
155 
156         // 开始构造点集
157         for (i = 0; i < G.vexnum; i++)
158         {
159             G.vertices[i].data = V[i];
160             G.vertices[i].firstarc = NULL;
161         }
162 
163         // 辅助函数: 查找顶点索引
164         auto Find = [&](int key) -> int
165         {
166             for (int j = 0; j < G.vexnum; j++)
167             {
168                 if (G.vertices[j].data.key == key)
169                     return j;
170             }
171             return -1;
172         };
173 
174         // 检查VR的顶点是否全部存在
175         for (i = 0; VR[i][0] != -1; i++)
176         {
177             if (Find(VR[i][0]) == -1 || Find(VR[i][1]) == -1)
178                 return ERROR;
179         }
180 
181         // 辅助函数: 添加边
182         auto AddArc = [&](int index1, int index2) -> status
183         {
184             // 检查边是否已存在
185             ArcNode *p = G.vertices[index1].firstarc;
186             while (p != NULL)
187             {
```

```
188                 if (p->adjvex == index2)
189                 {
190                     return ERROR; // 边已存在
191                 }
192                 p = p->nextarc;
193             }
194 
195             ArcNode *currentNode = (ArcNode *)malloc(sizeof(
                    ArcNode));
196             if (currentNode == NULL)
197                 return OVERFLOW;
198             currentNode->adjvex = index2;
199             currentNode->nextarc = G.vertices[index1].firstarc;
200             G.vertices[index1].firstarc = currentNode;
201             return OK;
202         };
203 
204         // 开始构造边
205         for (i = 0; VR[i][0] != -1 && VR[i][1] != -1; i++)
206         {
207             if (i >= 100)
208                 return ERROR;
209             int key1 = VR[i][0], key2 = VR[i][1];
210             int index1 = Find(key1), index2 = Find(key2);
211             if (AddArc(index2, index1) == ERROR || AddArc(index1,
                        index2) == ERROR)
212                 return ERROR;
213             G.arcnum++;
214         }
215         return OK;
216     }
217     status DestroyGraph()
218     {
219         for (int i = 0; i < G.vexnum; i++)
220         {
221             ArcNode *p = G.vertices[i].firstarc;
222             while (p != NULL)
223             {
224                 ArcNode *temp = p;
225                 p = p->nextarc;
226                 free(temp);
227             }
228             G.vertices[i].firstarc = NULL;
229         }
230         G.vexnum = 0; // 清空顶点数
231         G.arcnum = 0; // 清空边数
232         return OK;
233     }
```

```
234     int LocateVex(KeyType u)
235     {
236         int i = 0;
237         for (i = 0; i <= G.vexnum; i++)
238         {
239             if (i == G.vexnum)
240                 break;
241             VNode *p = &G.vertices[i];
242             if (p->data.key == u)
243                 break;
244         }
245 
246         if (i >= G.vexnum)
247             return -1;
248         else
249             return i;
250     }
251     status PutVex(KeyType u, VertexType value)
252     {
253         int i = 0, flag = OK;
254         for (i = 0; i <= G.vexnum; i++)
255         {
256             if (i == G.vexnum)
257                 break;
258             VNode *p = &G.vertices[i];
259             if (p->data.key != u && p->data.key == value.key)
260             {
261                 flag = ERROR;
262                 break;
263             }
264         }
265 
266         if (flag == OK)
267         {
268             flag = ERROR;
269             for (i = 0; i < G.vexnum; i++)
270             {
271                 if (G.vertices[i].data.key == u)
272                 {
273                     G.vertices[i].data = value;
274                     flag = OK;
275                     break;
276                 }
277             }
278         }
279 
280         if (flag == OK)
281             return OK;
```

```
282         else
283             return ERROR;
284     }
285     int FirstAdjVex(KeyType u)
286     {
287         int i = 0, flag = ERROR, res = 0;
288 
289         for (i = 0; i < G.vexnum; i++)
290         {
291             if (G.vertices[i].data.key == u)
292             {
293                 res = G.vertices[i].firstarc->adjvex;
294                 flag = OK;
295                 break;
296             }
297         }
298 
299         if (flag == OK)
300             return res;
301         else
302             return -1;
303     }
304     int NextAdjVex(KeyType v, KeyType w)
305     {
306         int i = 0, flag = ERROR, res = 0;
307 
308         for (i = 0; i < G.vexnum; i++)
309         {
310             if (G.vertices[i].data.key == v)
311             {
312                 res = G.vertices[i].firstarc->adjvex;
313                 flag = OK;
314                 break;
315             }
316         }
317 
318         if (flag == OK)
319         {
320             ArcNode *p = G.vertices[i].firstarc;
321             while (p != NULL)
322             {
323                 if (G.vertices[p->adjvex].data.key == w)
324                 {
325                     if (p->nextarc != NULL)
326                         return p->nextarc->adjvex;
327                     else
328                         return -1;
329                 }
```

```
330                 p = p->nextarc;
331             }
332         }
333 
334         return -1;
335     }
336 
337     status InsertVex(VertexType v)
338     {
339         for (int i = 0; i < G.vexnum; i++)
340         {
341             if (G.vertices[i].data.key == v.key)
342                 return ERROR;
343         }
344         if (G.vexnum >= MAX_VERTEX_NUM)
345             return ERROR;
346 
347         G.vertices[G.vexnum] = *(VNode *)malloc(sizeof(VNode));
348         if (G.vertices == NULL)
349             return ERROR;
350         G.vertices[G.vexnum].data = v;
351         G.vertices[G.vexnum].firstarc = NULL;
352         G.vexnum++;
353         return OK;
354     }
355 
356     status DeleteVex(KeyType v)
357     {
358         if (G.vexnum == 1)
359             return ERROR;
360         for (int i = 0; i < G.vexnum; i++)
361         {
362             if (G.vertices[i].data.key == v)
363             {
364                 ArcNode *p = G.vertices[i].firstarc;
365                 while (p != NULL)
366                 {
367                     ArcNode *q = p;
368                     p = p->nextarc;
369                     free(q);
370                 }
371                 for (int j = i; j < G.vexnum - 1; j++)
372                 {
373                     G.vertices[j] = G.vertices[j + 1];
374                 }
375                 for (int j = 0; j < G.vexnum - 1; j++)
376                 {
377                     if (G.vertices[j].firstarc != NULL)
```

```
                    {
                        ArcNode *p = G.vertices[j].firstarc;
                        ArcNode *q = NULL;
                        while (p != NULL)
                        {
                            if (p->adjvex == i)
                            {
                                if (q == NULL)
                                {
                                    G.vertices[j].firstarc = p->
                                        nextarc;
                                    G.arcnum--;
                                }
                                else
                                {
                                    q->nextarc = p->nextarc;
                                    G.arcnum--;
                                }
                                free(p);
                                break;
                            }
                            q = p;
                            p = p->nextarc;
                        }
                        ArcNode *p2 = G.vertices[j].firstarc;
                        ArcNode *q2 = NULL;
                        while (p2 != NULL)
                        {
                            if (p2->adjvex > i)
                                p2->adjvex--;
                            q2 = p2;
                            p2 = p2->nextarc;
                        }
                    }
                }
                G.vexnum--;
                return OK;
            }
        }
        return ERROR;
    }
    status InsertArc(KeyType v, KeyType w)
    {
        for (int i = 0; i < G.vexnum; i++)
        {
            if (G.vertices[i].data.key == v)
            {
                ArcNode *tmp = G.vertices[i].firstarc;
```

```
425                 while (tmp != NULL)
426                 {
427                     if (G.vertices[tmp->adjvex].data.key == w)
428                     {
429                         return ERROR;
430                     }
431                     tmp = tmp->nextarc;
432                 }
433                 for (int j = 0; j < G.vexnum; j++)
434                 {
435                     if (G.vertices[j].data.key == w)
436                     {
437                         ArcNode *p = (ArcNode *)malloc(sizeof(
                                ArcNode));
438                         p->adjvex = j;
439                         p->nextarc = G.vertices[i].firstarc;
440                         G.vertices[i].firstarc = p;
441                         ArcNode *q = (ArcNode *)malloc(sizeof(
                                ArcNode));
442                         q->adjvex = i;
443                         q->nextarc = G.vertices[j].firstarc;
444                         G.vertices[j].firstarc = q;
445                         G.arcnum++;
446                         return OK;
447                     }
448                 }
449             }
450         }
451         return ERROR;
452     }
453     status DeleteArc(KeyType v, KeyType w)
454     {
455         static int flag = 0;
456         if (G.vexnum == 0 || G.arcnum == 0)
457             return ERROR;
458         if (Delete(v, w) == OK && Delete(w, v) == OK)
459             return OK;
460         else
461             return ERROR;
462     }
463     status DFSTraverse(void (*visit)(VertexType))
464     {
465         int *visited = (int *)malloc(G.vexnum * sizeof(int)); //
                动态分配 visited 数组
466         if (visited == NULL)
467         {
468             return ERROR; // 分配失败
469         }
```

```
470
471         for (int i = 0; i < G.vexnum; i++)
472         {
473             visited[i] = 0; // 初始化为未访问
474         }
475
476         int firstVex;
477         std::cout << "请输入起始顶点的关键字:␣";
478         std::cin >> firstVex; // 输入起始顶点的关键字
479
480         int i = FindVex(G, firstVex); // 查找起始顶点的索引
481
482         DFS(G, i, visit, visited); // 调用辅助函数进行深度优先搜
                索
483
484         free(visited); // 释放动态分配的内存
485         return OK;
486     }
487     status BFSTraverse(void (*visit)(VertexType))
488     {
489         int *visited = (int *)malloc(G.vexnum * sizeof(int));
490         if (visited == NULL)
491         {
492             return ERROR; // 分配失败
493         }
494
495         for (int i = 0; i < G.vexnum; i++)
496
497             visited[i] = 0; // 初始化为未访问
498
499         queue<VNode> q;
500
501         int firstVex;
502         std::cout << "请输入起始顶点的关键字:␣";
503         std::cin >> firstVex; // 输入起始顶点的关键字
504
505         int first = FindVex(G, firstVex); // 查找起始顶点的索引
506
507         if (first != G.vexnum)
508         {
509             q.push(G.vertices[first]);
510             visited[first] = 1;
511             while (!q.empty())
512             {
513                 auto tmp = q.front();
514                 q.pop();
515                 int j = FindNum(G, tmp);
516                 visit(G.vertices[j].data);
```

```
517                 auto p = tmp.firstarc;
518                 while (p != nullptr)
519                 {
520                     if (visited[p->adjvex])
521                     {
522                         p = p->nextarc;
523                         continue;
524                     }
525                     q.push(G.vertices[p->adjvex]);
526                     visited[p->adjvex] = 1;
527                     p = p->nextarc;
528                 }
529             }
530         }
531         free(visited); // 释放内存
532         return OK;     // 遍历成功
533     }
534
535     status VerticesSetLessThanK(KeyType v, int k, void (*visit)(
        VertexType))
536     {
537         if (k < 0)
538             return ERROR;
539         int *visited = (int *)malloc(G.vexnum * sizeof(int));
540         if (visited == NULL)
541         {
542             return ERROR; // 分配失败
543         }
544
545         for (int i = 0; i < G.vexnum; i++)
546
547             visited[i] = 0; // 初始化为未访问
548
549         queue<VNode> q;
550         q.push(G.vertices[FindVex(G, v)]);
551         visited[FindVex(G, v)] = 1;
552         while (!q.empty())
553         {
554             auto tmp = q.front();
555             q.pop();
556             auto p = tmp.firstarc;
557             while (p != nullptr)
558             {
559                 if (visited[p->adjvex] || visited[FindNum(G, tmp)
                    ] >= k + 1)
560                 {
561                     p = p->nextarc;
562                     continue;
```

```
563                 }
564                 q.push(G.vertices[p->adjvex]);
565                 visited[p->adjvex] = visited[FindNum(G, tmp)] +
                        1;
566                 p = p->nextarc;
567             }
568         }
569         free(visited); // 释放内存
570         return OK;
571     }
572
573     int ShortestPathLength(KeyType v1, KeyType v2)
574     {
575         int res = -1;
576         int *visited = (int *)malloc(G.vexnum * sizeof(int));
577         if (visited == NULL)
578         {
579             return ERROR; // 分配失败
580         }
581
582         for (int i = 0; i < G.vexnum; i++)
583
584             visited[i] = 0; // 初始化为未访问
585
586         queue<VNode> q;
587
588         q.push(G.vertices[FindVex(G, v1)]);
589         visited[FindVex(G, v1)] = 1;
590         while (!q.empty())
591         {
592             auto tmp = q.front();
593             q.pop();
594             auto p = tmp.firstarc;
595             while (p != nullptr)
596             {
597                 if (G.vertices[p->adjvex].data.key == v2)
598                 {
599                     res = visited[tmp.data.key];
600                     free(visited);
601                     return res;
602                 }
603                 if (visited[p->adjvex])
604                 {
605                     p = p->nextarc;
606                     continue;
607                 }
608                 q.push(G.vertices[p->adjvex]);
609                 visited[p->adjvex] = visited[FindNum(G, tmp)] +
```

```
                    1;
                p = p->nextarc;
            }
        }

        free(visited); // 释放内存
        return res;
    }
    int ConnectedComponentsNums()
    {
        int *visited = (int *)malloc(G.vexnum * sizeof(int));
        if (visited == NULL)
        {
            return ERROR; // 分配失败
        }

        for (int i = 0; i < G.vexnum; i++)
            visited[i] = 0; // 初始化为未访问

        int count = 0;
        for (int i = 0; i < G.vexnum; i++)
        {
            if (visited[i] == 0)
            {
                count++;
                DFS(G, i, [](VertexType v) {}, visited);
            }
        }

        free(visited); // 释放内存
        return count;  // 返回连通分量个数
    }
    status SaveToFile(char FileName[])
    {
        FILE *fp = fopen(FileName, "w");
        if (fp == NULL)
        {
            return ERROR; // 打开文件失败
        }
        for (int i = 0; i < G.vexnum; i++)
        {
            fprintf(fp, "%d␣%s␣", G.vertices[i].data.key, G.
                vertices[i].data.others);
            auto *p = G.vertices[i].firstarc;
            while (p != NULL)
            {
                fprintf(fp, "%d␣", p->adjvex);
                p = p->nextarc;
```

```
656             }
657             fprintf(fp, "-1\n");
658         }
659         fprintf(fp, "-1␣nil␣%d␣%d", G.vexnum, G.arcnum);
660         fclose(fp);
661         return OK; // 保存成功
662     }
663     status LoadFromFile(char FileName[])
664     {
665         FILE *fp = fopen(FileName, "r");
666         if (fp == NULL)
667         {
668             return ERROR; // 打开文件失败
669         }
670         int i = 0, key;
671         char others[20];
672         while (fscanf(fp, "%d␣%s", &key, others) != EOF && key !=
                -1 && strcmp(others, "nil"))
673         {
674             G.vertices[i].data.key = key;
675             strcpy(G.vertices[i].data.others, others);
676             G.vertices[i].firstarc = NULL;
677             int adjvex;
678             while (fscanf(fp, "%d", &adjvex) != EOF && adjvex !=
                    -1)
679             {
680                 ArcNode *p = (ArcNode *)malloc(sizeof(ArcNode));
681                 p->adjvex = adjvex;
682                 p->nextarc = NULL;
683                 if (G.vertices[i].firstarc == NULL)
684                     G.vertices[i].firstarc = p;
685                 else
686                 {
687                     ArcNode *q = G.vertices[i].firstarc;
688                     while (q->nextarc != NULL)
689                     {
690                         q = q->nextarc;
691                     }
692                     q->nextarc = p;
693                 }
694                 G.arcnum++;
695             }
696             i++;
697         }
698         fscanf(fp, "%d␣%d", &G.vexnum, &G.arcnum);
699         if (G.vexnum < 20)
700         {
701             G.vertices[G.vexnum].data.key = -1;                 //
```

```
                设置结束标志
            strcpy(G.vertices[G.vexnum].data.others, "nil"); //
                设置结束标志
        }
        fclose(fp);
        return OK; // 读入成功
    }
};

template <class ElemType>
class List
{
private:
    ElemType *elem;
    char **name;
    int listlength;

public:
    List() : elem(nullptr), name(nullptr), listlength(0)
    {
        name = (char **)malloc(20 * sizeof(char *)); // 初始化
            name
        for (int i = 0; i < 20; ++i)
        {
            name[i] = (char *)malloc(20 * sizeof(char)); // 初始
                化每个name元素
        }
        strcpy(name[0], "默认第一张图");
    }
    ElemType GetGraph(int index)
    {
        return elem[index];
    }
    char *GetGraphName(int index)
    {
        if (index == -1)
        {
            printf("请选择正确的图");
            return nullptr;
        }
        else
            return name[index];
    }
    // 多图管理
    AddGraph(Graph<ALGraph> &newG)
    {
        if (!elem)
        {
```

```
746             elem = new ElemType();
747         }
748 
749         if (!name)
750         {
751             name = new char *[1];
752             name[0] = new char[20];
753             std::cout << "\t请输入二叉树名称:␣";
754             std::cin >> name[0];
755         }
756         else
757         {
758             char tempName[20];
759             bool isDuplicate;
760             do
761             {
762                 isDuplicate = false;
763                 std::cout << "\t请输入二叉树名称:␣";
764                 std::cin >> tempName;
765 
766                 // 检查名称是否重复
767                 for (int i = 0; i < listlength; ++i)
768                 {
769                     if (strcmp(name[i], tempName) == 0)
770                     {
771                         isDuplicate = true;
772                         std::cout << "\t二叉树名称重复, 请重新输
                                入。" << std::endl;
773                         break;
774                     }
775                 }
776             } while (isDuplicate);
777 
778             char **newname = new char *[listlength + 1];
779             for (int i = 0; i < listlength; ++i)
780             {
781                 newname[i] = name[i];
782             }
783             newname[listlength] = new char[20];
784             strcpy(newname[listlength], tempName);
785             delete[] name;
786             name = newname;
787         }
788 
789         elem[listlength] = newG;
790         listlength++;
791         return OK; // 添加成功
792     }
```

```
793     status DeleteGraph(char *name)
794     {
795         if (!elem)
796         {
797             std::cout << "\t没有可删除的图。" << std::endl;
798             return ERROR;
799         }
800 
801         for (int i = 0; i < listlength; ++i)
802         {
803             if (strcmp(this->name[i], name) == 0)
804             {
805                 delete[] this->name[i]; // 删除旧名称
806                 this->name[i] = nullptr;
807                 for (int j = i; j < listlength - 1; ++j)
808                 {
809                     this->name[j] = this->name[j + 1];
810                     elem[j] = elem[j + 1];
811                 }
812                 listlength--;
813                 return OK; // 删除成功
814             }
815         }
816 
817         std::cout << "\t未找到名称为␣" << name << "␣的图。" <<
                std::endl;
818         return ERROR; // 删除失败
819     }
820     int SelectGraph(char *name)
821     {
822         if (!elem)
823         {
824             std::cout << "\t没有可选择的图。" << std::endl;
825             return -1;
826         }
827 
828         for (int i = 0; i < listlength; ++i)
829         {
830             if (strcmp(this->name[i], name) == 0)
831             {
832                 std::cout << "\t选择的图为第" << i + 1 << "张图"
                        << std::endl;
833                 return i; // 选择成功
834             }
835         }
836 
837         std::cout << "\t未找到名称为␣" << name << "␣的图。" <<
                std::endl;
```

```
838         return -1; // 选择失败
839     }
840     status ModifyGraphName(char *oldName, char *newName)
841     {
842         if (!elem)
843         {
844             std::cout << "\t没有可修改的图。" << std::endl;
845             return ERROR;
846         }
847 
848         for (int i = 0; i < listlength; ++i)
849         {
850             if (strcmp(this->name[i], oldName) == 0)
851             {
852                 delete[] this->name[i]; // 删除旧名称
853                 this->name[i] = new char[20];
854                 strcpy(this->name[i], newName);
855                 return OK; // 修改成功
856             }
857         }
858 
859         std::cout << "\t未找到名称为␣" << oldName << "␣的图。" <<
                std::endl;
860         return ERROR; // 修改失败
861     }
862     status PrintGraphName()
863     {
864         if (!elem)
865         {
866             std::cout << "\t没有可打印的图名称。" << std::endl;
867             return ERROR;
868         }
869 
870         for (int i = 0; i < listlength; ++i)
871         {
872             std::cout << "\t图名称␣" << i + 1 << ":␣" << name[i]
                    << std::endl;
873         }
874         return OK; // 打印成功
875     }
876 };
```

集成各个功能的菜单函数

```
#include <bits/stdc++.h>
#include <windows.h>
#include "graph_def.h" // 图相关数据类型的定义
#include "graph_func.h"

using namespace std;
void visit(VertexType v)
{
    printf("␣%d␣%s", v.key, v.others);
}
int main()
{
    SetConsoleOutputCP(65001); // 设置控制台输出编码为UTF-8
    Graph<ALGraph> G;
    List<Graph<ALGraph>> Graphs;
    static int n = 0;
    int op = 1;
    VNode lis;
    while (op)
    {
        printf("\n\n");
        printf("␣␣␣␣␣␣图操作菜单␣\n");
        printf("================================================\n
            ");
        printf("␣␣基本操作: \n");
        printf("␣␣␣␣1.␣创建图␣␣␣␣␣␣␣␣␣␣␣␣␣␣2.␣销毁图\n");
        printf("␣␣␣␣3.␣查找顶点␣␣␣␣␣␣␣␣␣␣␣␣4.␣顶点赋值\n");
        printf("␣␣␣␣5.␣获得第一邻接点␣␣␣␣␣␣6.␣获得下一邻接点\n")
            ;
        printf("------------------------------------------------\n
            ");
        printf("␣␣修改操作: \n");
        printf("␣␣␣␣7.␣插入顶点␣␣␣␣␣␣␣␣␣␣␣␣8.␣删除顶点\n");
        printf("␣␣␣␣9.␣插入弧␣␣␣␣␣␣␣␣␣␣␣␣␣10.␣删除弧\n");
        printf("------------------------------------------------\n
            ");
        printf("␣␣遍历操作: \n");
        printf("␣␣␣11.␣深度优先遍历␣␣␣␣␣␣␣12.␣广度优先遍历\n");
        printf("------------------------------------------------\n
            ");
        printf("␣␣附加功能: \n");
        printf("␣␣␣13.␣距离小于k的顶点集合␣14.␣顶点间最短路径长度
            \n");
        printf("␣␣␣15.␣图的连通分量␣␣␣␣␣␣␣16.␣保存图到文件\n");
        printf("␣␣␣17.␣从文件加载图\n");
        printf("------------------------------------------------\n
```

```
    ");
    printf("␣␣多图管理: \n");
    printf("␣␣␣18.␣添加图␣␣␣␣␣␣␣␣␣␣␣␣␣␣19.␣删除图\n");
    printf("␣␣␣20.␣定位图␣␣␣␣␣␣␣␣␣␣␣␣␣␣21.␣修改图名称\n");
    printf("␣␣␣22.␣输出所有图名称\n");
    printf("␣␣␣␣0.␣退出程序\n");
    printf("================================================\n
    ");
    if (n != 0)
        printf("␣␣正在操作的图为: %s\n", Graphs.GetGraphName(
            n - 1));
    printf("␣␣请选择你的操作␣[0␣~␣22]:␣");
    scanf("%d", &op);

    switch (op)
    {
    case 1:
        printf("\t创建图\n");
        if (G.CreateGraph() == OK)
        {
            Graphs.AddGraph(G);
            n++;
            printf("\t创建成功! \n");
        }
        else
            printf("\t创建失败! \n");
        break;
    case 2:
        printf("\t销毁图\n");
        if (G.DestroyGraph() == OK)
            printf("\t销毁成功! \n");
        else
            printf("\t销毁失败! \n");
        break;
    case 3:
        printf("\t查找顶点\n");
        printf("\t请输入顶点关键字: ");
        int key;
        scanf("%d", &key);
        int pos;
        pos = G.LocateVex(key);
        if (pos != -1)
            printf("\t顶点位置为: %d\n", pos + 1);
        else
            printf("\t顶点不存在! \n");
        break;
    case 4:
        printf("\t顶点赋值\n");
```

```
86              printf("\t请输入顶点关键字: ");
87              int u;
88              VertexType value;
89              scanf("%d", &u);
90              printf("\t请输入新值: ");
91              scanf("%d␣%s", &value.key, value.others);
92              if (G.PutVex(u, value) == OK)
93                  printf("\t赋值成功! \n");
94              else
95                  printf("\t赋值失败! \n");
96              break;
97          case 5:
98              printf("\t获得第一邻接点\n");
99              printf("\t请输入顶点关键字: ");
100             int v;
101             scanf("%d", &v);
102             int firstAdj;
103             firstAdj = G.FirstAdjVex(v);
104             lis=G.GetNode(firstAdj);
105             if (firstAdj != -1)
106                 printf("\t第一邻接点为: %d␣%s\n", lis.data.key,
                        lis.data.others);
107             else
108                 printf("\t没有邻接点! \n");
109             break;
110         case 6:
111             printf("\t获得下一邻接点\n");
112             printf("\t请输入顶点关键字和当前邻接点关键字: ");
113             int w;
114             scanf("%d␣%d", &v, &w);
115             int nextAdj;
116             nextAdj = G.NextAdjVex(v, w);
117
118             lis = G.GetNode(nextAdj);
119             if (nextAdj != -1)
120                 printf("\t下一邻接点为: %d␣%s\n", lis.data.key,
                        lis.data.others);
121             else
122                 printf("\t没有下一邻接点! \n");
123             break;
124         case 7:
125             printf("\t插入顶点\n");
126             printf("\t请输入顶点关键字: ");
127             VertexType newVex;
128             scanf("%d␣%s", &newVex.key, newVex.others);
129             if (G.InsertVex(newVex) == OK)
130                 printf("\t插入成功! \n");
131             else
```

```
132                 printf("\t插入失败！\n");
133             break;
134         case 8:
135             printf("\t删除顶点\n");
136             printf("\t请输入顶点关键字：");
137             int delVex;
138             scanf("%d", &delVex);
139             if (G.DeleteVex(delVex) == OK)
140                 printf("\t删除成功！\n");
141             else
142                 printf("\t删除失败！\n");
143             break;
144         case 9:
145             printf("\t插入弧\n");
146             printf("\t请输入弧的起点和终点关键字：");
147             int v1, v2;
148             scanf("%d␣%d", &v1, &v2);
149             if (G.InsertArc(v1, v2) == OK)
150                 printf("\t插入成功！\n");
151             else
152                 printf("\t插入失败！\n");
153             break;
154         case 10:
155             printf("\t删除弧\n");
156             printf("\t请输入弧的起点和终点关键字：");
157             scanf("%d␣%d", &v1, &v2);
158             if (G.DeleteArc(v1, v2) == OK)
159                 printf("\t删除成功！\n");
160             else
161                 printf("\t删除失败！\n");
162             break;
163         case 11:
164             printf("\t深度优先遍历\n");
165             if (G.DFSTraverse(visit) == OK)
166                 printf("\t遍历成功！\n");
167             else
168                 printf("\t遍历失败！\n");
169             break;
170         case 12:
171             printf("\t广度优先遍历\n");
172             if (G.BFSTraverse(visit) == OK)
173                 printf("\t遍历成功！\n");
174             else
175                 printf("\t遍历失败！\n");
176             break;
177         case 13:
178             printf("\t距离小于k的顶点集合\n");
179             printf("\t请输入顶点关键字和距离k：");
```

```
180             int k;
181             scanf("%d␣%d", &v, &k);
182 
183             if (G.VerticesSetLessThanK(v, k, visit) == OK)
184                 printf("\t操作成功！\n");
185             else
186                 printf("\t操作失败！\n");
187             break;
188         case 14:
189             printf("\t顶点间最短路径长度\n");
190             printf("\t请输入两个顶点关键字：");
191             scanf("%d␣%d", &v1, &v2);
192             int length;
193             length = G.ShortestPathLength(v1, v2);
194             if (length != -1)
195                 printf("\t最短路径长度为：%d\n", length);
196             else
197                 printf("\t路径不存在！\n");
198             break;
199         case 15:
200             printf("\t图的连通分量\n");
201             int components;
202             components = G.ConnectedComponentsNums();
203             printf("\t连通分量个数为：%d\n", components);
204             break;
205         case 16:
206             printf("\t保存图到文件\n");
207             printf("\t请输入文件名：");
208             char saveFile[100];
209             scanf("%s", saveFile);
210             if (G.SaveToFile(saveFile) == OK)
211                 printf("\t保存成功！\n");
212             else
213                 printf("\t保存失败！\n");
214             break;
215         case 17:
216             printf("\t从文件加载图\n");
217             printf("\t请输入文件名：");
218             char loadFile[100];
219             scanf("%s", loadFile);
220             if (G.LoadFromFile(loadFile) == OK)
221                 printf("\t加载成功！\n");
222             else
223                 printf("\t加载失败！\n");
224             break;
225         case 18:
226             printf("\t添加图\n");
227             if (Graphs.AddGraph(G) == OK)
```

```
228             {
229                 n++;
230                 printf("\t添加成功！\n");
231             }
232             else
233                 printf("\t添加失败！\n");
234             break;
235         case 19:
236             printf("\t删除图\n");
237             printf("\t请输入图名称：");
238             char delGraphName[20];
239             scanf("%s", delGraphName);
240             if (Graphs.DeleteGraph(delGraphName) == OK)
241                 printf("\t删除成功！\n");
242             else
243                 printf("\t删除失败！\n");
244             break;
245         case 20:
246             printf("\t定位图\n");
247             printf("\t请输入图名称：");
248             char locateGraphName[20];
249             scanf("%s", locateGraphName);
250             int temp;
251             temp = n;
252             n = Graphs.SelectGraph(locateGraphName)+1;
253             if (n != -1)
254             {
255                 G = Graphs.GetGraph(n-1);
256                 printf("\t定位成功！\n");
257             }
258             else
259             {
260                 n = temp;
261                 printf("\t定位失败！\n");
262             }
263             break;
264         case 21:
265             printf("\t修改图名称\n");
266             printf("\t请输入当前图名称：");
267             char oldGraphName[20];
268             scanf("%s", oldGraphName);
269             printf("\t请输入新图名称：");
270             char newGraphName[20];
271             scanf("%s", newGraphName);
272             if (Graphs.ModifyGraphName(oldGraphName, newGraphName
                    ) == OK)
273                 printf("\t修改成功！\n");
274             else
```

```
                printf("\t修改失败！\n");
            break;
        case 22:
            printf("\t输出所有图名称\n");
            if (Graphs.PrintGraphName() == OK)
                printf("\t输出成功！\n");
            else
                printf("\t输出失败！\n");
            break;
        case 23:
            std::cout << "\t完整输出当前图" << std::endl;
            G.Traverse();
            break;
        case 0:
            printf("\t退出程序\n");
            break;
        default:
            printf("\t输入错误，请重新输入！\n");
            break;
        }
        printf("\t请按回车键继续...");
        getchar();
        getchar();
    }
    printf("欢迎下次再使用本系统！\n");
    return 0;
}
```