

华中科技大学

课程实验报告

课程名称: 数据结构实验

专业班级 CS

学 号 塔塔开!

姓 名 Eren · Yeager

指导教师 Eren · Yeager

报告日期 2022 年 5 月 15 日

计算机科学与技术学院

目 录

1 基于链式存储结构的线性表实现.....	1
1.1 问题描述	1
1.2 系统设计	2
1.3 系统实现	3
1.4 系统测试	6
1.5 实验小结	15
2 基于二叉链表的二叉树实现	17
2.1 问题描述	17
2.2 系统设计	19
2.3 系统实现	20
2.4 系统测试	24
2.5 实验小结	32
3 课程的收获和建议	34
3.1 基于顺序存储结构的线性表实现	34
3.2 基于链式存储结构的线性表实现	34
3.3 基于二叉链表的二叉树实现	34
3.4 基于邻接表的图实现	35
参考文献	36
附录 A 基于顺序存储结构线性表实现的源程序	37
附录 B 基于链式存储结构线性表实现的源程序	57
附录 C 基于二叉链表存储结构实现二叉树的源程序	78
附录 D 基于邻接表图实现的源程序	100

1 基于链式存储结构的线性表实现

1.1 问题描述

为了加深对线性表的理解，我需要通过设计一个线性表管理系统 (采用链式存储)，了解线性表的逻辑结构与存储结构，实现线性表的基本运算 (任务1.1) 并且最终将所有基本运算封装成一个线性表管理系统。

1. 初始化表：函数名称是 `InitList(L)`；初始条件是线性表 `L` 不存在；操作结果是构造一个空的线性表
2. 销毁表：函数名称是 `DestroyList(L)`；初始条件是线性表 `L` 已存在；操作结果是销毁线性表 `L`
3. 清空表：函数名称是 `ClearList(L)`；初始条件是线性表 `L` 已存在；操作结果是将 `L` 重置为空表
4. 判定空表：函数名称是 `ListEmpty(L)`；初始条件是线性表 `L` 已存在；操作结果是若 `L` 为空表则返回 `TRUE`, 否则返回 `FALSE`
5. 求表长：函数名称是 `ListLength(L)`；初始条件是线性表已存在；操作结果是返回 `L` 中数据元素的个数
6. 获得元素：函数名称是 `GetElem(L,i,e)`；初始条件是线性表已存在， $1 \leq i \leq ListLength(L)$ ；操作结果是用 `e` 返回 `L` 中第 `i` 个数据元素的值
7. 查找元素：函数名称是 `LocateElem(L,e,compare())`；初始条件是线性表已存在；操作结果是返回 `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：函数名称是 `PriorElem(L,cur_e,pre_e)`；初始条件是线性表 `L` 已存在；操作结果是若 `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：函数名称是 `NextElem(L,cur_e,next_e)`；初始条件是线性表 `L` 已存在；操作结果是若 `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：函数名称是 `ListInsert(L,i,e)`；初始条件是线性表 `L` 已存在， $1 \leq i \leq ListLength(L) + 1$ ；操作结果是在 `L` 的第 `i` 个位置之前插入新的数据元素 `e`

11. 删除元素：函数名称是 `ListDelete(L,i,e)`；初始条件是线性表 `L` 已存在且非空， $1 \leq i \leq ListLength(L)$ ；操作结果：删除 `L` 的第 `i` 个数据元素，用 `e` 返回其值
12. 遍历表：函数名称是 `ListTraverse(L,visit())`，初始条件是线性表 `L` 已存在；操作结果是依次对 `L` 的每个数据元素调用函数 `visit()`
13. 链表翻转：函数名称是 `reverseList(L)`，初始条件是线性表 `L` 已存在；操作结果是将 `L` 翻转
14. 删除链表的倒数第 `n` 个结点：函数名称是 `RemoveNthFromEnd(L,n)`；初始条件是线性表 `L` 已存在且非空，操作结果是该链表中倒数第 `n` 个节点
15. 链表排序：函数名称是 `sortList(L)`，初始条件是线性表 `L` 已存在；操作结果是将 `L` 由小到大排序
16. 实现线性表的文件形式保存：其中，(1) 需要设计文件数据记录格式，以高效保存线性表数据逻辑结构 `(D,R)` 的完整信息；(2) 需要设计线性表文件保存和加载操作合理模式
17. 实现多个线性表管理：设计相应的数据结构管理多个线性表的查找、添加、移除等功能

1.2 系统设计

数据结构设计：

首先设计单张表的节点结构，其结构域中包含指向下一节点的指针 `next` 和该节点的数据信息 `data`。

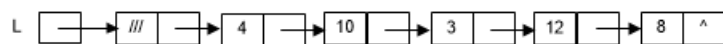


图 1-1 单个链表结构

其次设计多表管理系统的结构，其结构域中包含指向第一张表的指针 `start`，指向最后一张表的指针 `end` 以及表的数量 `length`。

为了使管理系统更干净方便地管理链表，我把单张表的各种操作封装在一个类 `LinkList` 中，该类的域中包含指向空表头的指针 `head`，指向表尾的指针 `tail` 和表长 `length`，同时使用一个指针 `down` 指向该链表的下一张。该类起着连接单表和管理系统的作用（即通过十字链表进行存储）。

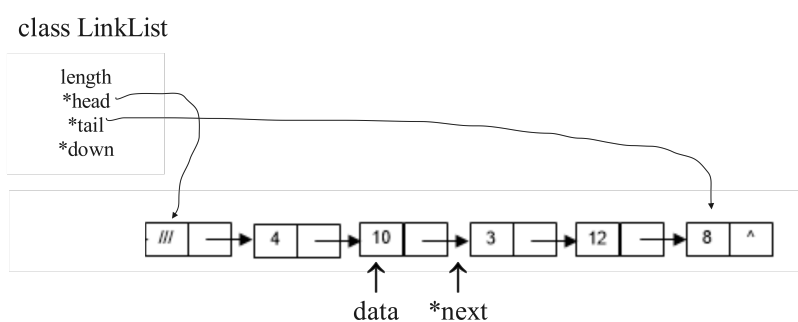


图 1-2 LinkList

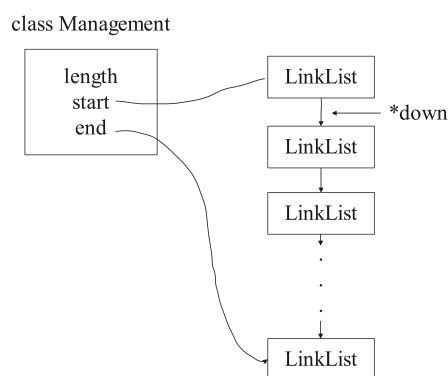


图 1-3 Management

整体系统结构设计：

整个系统分为两层。

最外层是管理系统 `Management`，负责新增表、删除表、进入某个表进行操作 (在这一层中，所有对单张表进行的操作都是被隐封装着的，只能通过接口 `single_list_op()` 进行访问)。

内层是对单张表的操作，需要先进入一张表然后操作。随时可以返回管理系统进行其他操作。

1.3 系统实现

单表操作：

1. `InitList(L)`: 用 `malloc` 函数为链表分配空间即可。注意判断链表是否已存在。
2. `DestroyList(L)`: 使用 `free` 函数将所有分配空间的节点依次释放即可。注意判空。
3. `ClearList(L)`: 用 `free` 函数释放除头节点的空间即可。注意判空。

4. ListEmpty(L): 由于前面及时用域 length 记录表长, 表长为 0 返回 false 反之返回 true。注意判空。
5. ListLength(L): 与 5 思路相同。
6. GetElem(L,i,e): 遍历线性表查找元素即可。注意判空。
7. LocateElem(L,i,e): 思路同 6。
8. PriorElem(L,cur_e,pre_e): 思路同 6。
9. NextElem(L,cur_e,next_e): 思路同 6。
10. ListInsert(L,i,e): 通过遍历找到位置后用 malloc 新分配一个节点空间即可。注意判空。
11. ListDelete(L,i,e): 通过遍历找到位置后调整指针域, 释放节点即可。注意判空。
12. ListTraverse(L,visit()): 遍历一遍输出即可。
13. reverseList(L):
14. RemoveNthFromEnd(L,n): 删除倒数第 n 个节点, 就是删除整数第 $ListLength - n + 1$ 个节点。
15. sortList(L): 这里可以用两种方法: 一是新建一个数组把链表的信息放到数组中, 借助数组进行排序, 但这样会使用额外的空间。二是可以使用冒泡排序的交换指针域形式, 单轮排序的形式如图。

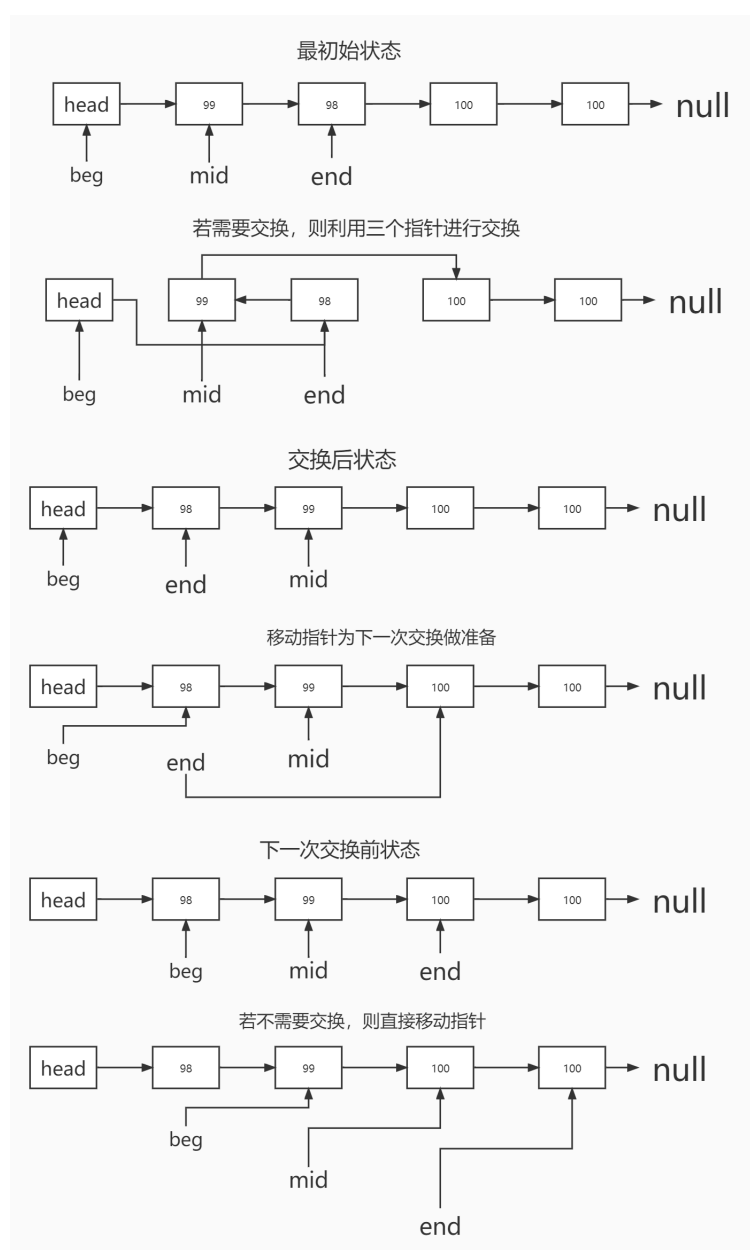


图 1-4 链表排序

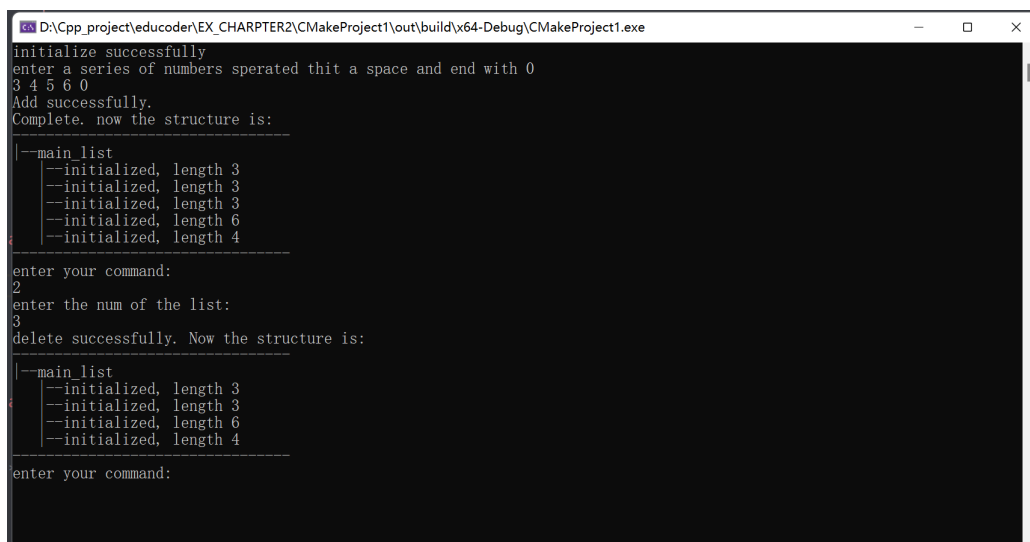
16. SaveList(L,char FileName[]): 遍历一遍输出到文件流中即可。注意判空。
17. LoadList(L,char FileName[]): 从文件流中读取数据, 边读取使用 malloc 新建节点。注意判断线性表非空。

管理系统操作:

1. NewList(): 在最后一张表 down 指针指向的位置为 class LinkList 开辟空间即可, 注意在构造函数中要将还没赋值的指针域置空。
2. DelList(): 将指定位置的表空间全部释放, 调节剩下的指针域即可。

一些自己写着玩的函数

1. `structure_disp(Manager M)`: 用于展示管理系统展示树形目录。由于表的数目和层次结构比较简单，只需要用迭代来打印目录即可。打印存储情况及状态如图



```
D:\Cpp_project\educoder\EX_CHARACTER2\CMakeProject1\out\build\x64-Debug\CMakeProject1.exe
Initialize successfully
enter a series of numbers sperated thit a space and end with 0
3 4 5 6 0
Add successfully.
Complete. now the structure is:
-----
--main list
--initialized, length 3
--initialized, length 3
--initialized, length 3
--initialized, length 6
--initialized, length 4
-----
enter your command:
2
enter the num of the list:
3
delete successfully. Now the structure is:
-----
--main list
--initialized, length 3
--initialized, length 3
--initialized, length 6
--initialized, length 4
-----
enter your command:
```

图 1-5 打印树形目录

1.4 系统测试

1. 新建一张线性表

函数名：NewList	
正常样例	1 2 -5 3 4 -7 -3 9 0
预期结果	创建成功并返回树形目录
异常样例	None
预期结果	None

实测结果：


```
enter your command:
1
initialize successfully
enter a series of numbers sperated thit a space and end with 0
1 2 -5 3 4 -7 -3 9 0
Add successfully.
Complete. now the structure is:
-----
|--main_list
  |--initialized, length 8
-----
```

图 1-6 测试结果-1

2. 删除一张线性表

函数名: DelList	
正常样例	1
预期结果	删除成功并返回树形目录
异常样例	3
预期结果	index out of range!

实测结果:

```
enter your command:
2
enter the num of the list:
1
delete successfully. Now the structure is:
-----
|--main_list
  |--empty
-----
```

(a) 正常样例

```
enter your command:
2
enter the num of the list:
3
index out of range.
```

(b) 异常样例

图 1-7 测试结果-2

3. 销毁已存在的线性表

函数名: ListEmpty	
正常样例	None
预期结果	not empty!
异常样例	None(空表或者未初始化的表)
预期结果	显示 empty list!

实测结果:

```
input your command:
4
not empty!
```

(a) 正常样例

```
input your command:
4
empty!
```

(b) 异常样例

图 1-10 测试结果-5

6. 求表长

函数名: ListLength	
正常样例	None
预期结果	返回表长
异常样例	None(未初始化的表)
预期结果	显示 empty list

实测结果:

```
input your command:
5
the length is 8
```

(a) 正常样例

```
input your command:
5
empty list
```

(b) 异常样例

图 1-11 测试结果-6

7. 获取指定位置元素

函数名: GetElem	
正常样例	元素位置
预期结果	返回对应元素
异常样例	元素位置 (未初始化的表)
预期结果	显示 empty list

实测结果:

```
input your command:
6
enter the index of the element:
2
the element is 2
```

(a) 正常样例

```
input your command:
6
enter the index of the element:
2
empty list
```

(b) 异常样例

图 1-12 测试结果-7

8. 获取元素位置

函数名: LocateElem	
正常样例	元素
预期结果	返回对应位置
异常样例	元素 (不在表中)
预期结果	显示找不到元素

实测结果:

```
input your command:
7
enter the num to be located:
-7
the element index is 6
```

(a) 正常样例

```
input your command:
7
enter the num to be located:
11
can't find the element.
```

(b) 异常样例

图 1-13 测试结果-8

9. 获取前驱

函数名: PriorElem	
正常样例	元素
预期结果	返回前驱
异常样例	元素 (不在表中或无后继)
预期结果	显示找不到元素

实测结果:

```
input your command:
8
enter the elem:
3
the prior element is -5
```

(a) 正常样例

```
input your command:
8
enter the elem:
10
can't find the element.
```

(b) 异常样例

图 1-14 测试结果-9

10. 获取后继

函数名: NextElem	
正常样例	元素
预期结果	返回前驱
异常样例	元素 (不在表中或无后继)
预期结果	显示找不到元素

实测结果:

```
input your command:
9
enter the elem:
3
the next element is 4
```

(a) 正常样例

```
input your command:
9
enter the elem:
0
can't find the element.
```

(b) 异常样例

图 1-15 测试结果-10

11. 插入元素

函数名: ListInsert	
正常样例	2 11
预期结果	显示插入后遍历结果
异常样例	错误的插入位置
预期结果	显示位置错误

实测结果:

```
input your command:
10
enter the location:
2
enter the elem:
11
insert successfully! traverse: 1 11 2 -5 3 4 -7 -3 9
```

(a) 正常样例

```
input your command:
10
enter the location:
0
enter the elem:
11
wrong position!.
```

(b) 异常样例

图 1-16 测试结果-11

12. 删除元素

函数名: ListDelete	
正常样例	2
预期结果	删除成功
异常样例	错误的删除位置
预期结果	显示位置错误

实测结果:

```
input your command:
11
enter the location:
2
delete successfully
```

(a) 正常样例

```
input your command:
11
enter the location:
-1
wrong position!.
```

(b) 异常样例

图 1-17 测试结果-12

13. 遍历表

函数名: ListTraverse	
正常样例	None
预期结果	遍历结果
异常样例	None
预期结果	None

实测结果:

```
input your command:
12
1 2 -5 3 4 -7 -3 9 traverse over.
```

图 1-18 测试结果-13

14. 反转链表

函数名: ListReverse	
正常样例	None
预期结果	反转后遍历结果
异常样例	None
预期结果	None

实测结果:

```
input your command:
13
reverse over. traverse: 9 -3 -7 4 3 -5 2 1
```

图 1-19 测试结果-14

15. 删除倒数第 n 个元素

函数名: RemoveNthfromEnd	
正常样例	1
预期结果	删除成功
异常样例	20
预期结果	显示位置错误

实测结果:


```
input your command:
14
enter the n:
1
delete successfully
```

(a) 正常样例

```
input your command:
14
enter the n:
20
wrong position!.
```

(b) 异常样例

图 1-20 测试结果-15

16. 链表排序

函数名: ListSort	
正常样例	None
预期结果	排序后遍历结果
异常样例	None
预期结果	None

实测结果:

```
input your command:
15
sort over. reverse: -7 -5 -3 1 2 3 4 9
```

图 1-21 测试结果-16

1.5 实验小结

内容收获:

1. 掌握了链式结构存储的一些实现方式。基本任务圆满完成。
2. 复习了 c++ 中类, STL, 文件 IO, 友元函数等的基本用法。
3. 学习了如何使用 VisualStudio + cmake 搭建跨平台项目。

一些典型错误:

1. 没有判断指针是否指向空导致 segmentation fault。
2. 文件忘记关闭, 这在向文件输入时会导致文件是空的, 输出时报错。
3. 一个引用的问题: 函数返回一个引用时要用引用来接受引用再进行操作。

经验:

1. 头文件和源文件分离的结构，头文件中存放相关的函数、类声明，源文件中存放相关的定义。
2. 对于一些小型课设，面向对象编程会造成一些不必要的冗余，还是使用面向过程编程更加方便 (这也在二叉树的报告中体现了)。
3. 能抽象成函数的部分尽量抽象成函数，重复的语句一定要用函数，否则后面调试维护时会很麻烦。
4. 指针不用时一定要置空，并且使用指针时一定要判空。

2 基于二叉链表的二叉树实现

2.1 问题描述

为熟练掌握二叉树的各种基本运算，了解其逻辑结构与物理结构，我需要设计一个二叉树管理系统，实现二叉树的管理及其基本运算的实现 (任务2.1)。

1. 创建二叉树：函数名称是 `CreateBiTree(T,definition)`；初始条件是 `definition` 给出二叉树 `T` 的定义，按照带空子树的二叉树前序遍历序列；操作结果是按 `definition` 构造二叉树 `T`
2. 销毁二叉树：函数名称是 `DestroyBiTree(T)`；初始条件是二叉树 `T` 已存在；操作结果是销毁二叉树 `T`
3. 清空二叉树：函数名称是 `ClearBiTree (T)`；初始条件是二叉树 `T` 存在；操作结果是将二叉树 `T` 清空
4. 判定空二叉树：函数名称是 `BiTreeEmpty(T)`；初始条件是二叉树 `T` 存在；操作结果是若 `T` 为空二叉树则返回 `TRUE`，否则返回 `FALSE`
5. 求二叉树深度：函数名称是 `BiTreeDepth(T)`；初始条件是二叉树 `T` 存在；操作结果是返回 `T` 的深度
6. 查找结点：函数名称是 `LocateNode(T,e)`；初始条件是二叉树 `T` 已存在，`e` 是和 `T` 中结点关键字类型相同的给定值；操作结果是返回查找到的结点指针，如无关键字为 `e` 的结点，返回 `NULL`
7. 结点赋值：函数名称是 `Assign(T,e,value)`；初始条件是二叉树 `T` 已存在，`e` 是和 `T` 中结点关键字类型相同的给定值；操作结果是关键字为 `e` 的结点赋值为 `value`
8. 获得兄弟结点：函数名称是 `GetSibling(T,e)`；初始条件是二叉树 `T` 存在，`e` 是和 `T` 中结点关键字类型相同的给定值；操作结果是返回关键字为 `e` 的结点的（左或右）兄弟结点指针。若关键字为 `e` 的结点无兄弟，则返回 `NULL`
9. 插入结点：函数名称是 `InsertNode(T,e,LR,c)`；初始条件是二叉树 `T` 存在，`e` 是和 `T` 中结点关键字类型相同的给定值，`LR` 为 0 或 1，`c` 是待插入结点；操作结果是根据 `LR` 为 0 或者 1，插入结点 `c` 到 `T` 中，作为关键字为 `e` 的结点的左或右孩子结点，结点 `e` 的原有左子树或右子树则为结点 `c` 的右子树。特殊情况：`c` 插入作为根结点？可以考虑 `LR` 为 -1 时，作为根结点插入，原根

结点作为 c 的右子树。

10. 删除结点：函数名称是 `DeleteNode(T,e)`；初始条件是二叉树 T 存在， e 是和 T 中结点关键字类型相同的给定值。操作结果是删除 T 中关键字为 e 的结点；同时，如果关键字为 e 的结点度为 0，删除即可；如关键字为 e 的结点度为 1，用关键字为 e 的结点孩子代替被删除的 e 位置；如关键字为 e 的结点度为 2，用 e 的左孩子代替被删除的 e 位置， e 的右子树作为 e 的左子树中最右结点的右子树
11. 前序遍历：函数名称是 `PreOrderTraverse(T,Visit())`；初始条件是二叉树 T 存在，`Visit` 是一个函数指针的形参（可使用该函数对结点操作）；操作结果：先序遍历，对每个结点调用函数 `Visit` 一次且一次，一旦调用失败，则操作失败。
12. 中序遍历：函数名称是 `InOrderTraverse(T,Visit())`；初始条件是二叉树 T 存在，`Visit` 是一个函数指针的形参（可使用该函数对结点操作）；操作结果是中序遍历 t ，对每个结点调用函数 `Visit` 一次且一次，一旦调用失败，则操作失败
13. 后序遍历：函数名称是 `PostOrderTraverse(T,Visit())`；初始条件是二叉树 T 存在，`Visit` 是一个函数指针的形参（可使用该函数对结点操作）；操作结果是后序遍历 t ，对每个结点调用函数 `Visit` 一次且一次，一旦调用失败，则操作失败
14. 按层遍历：函数名称是 `LevelOrderTraverse(T,Visit())`；初始条件是二叉树 T 存在，`Visit` 是对结点操作的应用函数；操作结果是层序遍历 t ，对每个结点调用函数 `Visit` 一次且一次，一旦调用失败，则操作失败
15. 最大路径和：函数名称是 `MaxPathSum(T)`，初始条件是二叉树 T 存在；操作结果是返回根节点到叶子结点的最大路径和
16. 最近公共祖先：函数名称是 `LowestCommonAncestor(T,e1,e2)`；初始条件是二叉树 T 存在；操作结果是该二叉树中 $e1$ 节点和 $e2$ 节点的最近公共祖先
17. 函数名称是 `InvertTree(T)`，初始条件是线性表 L 已存在；操作结果是将 T 翻转，使其所有节点的左右节点互换
18. 实现线性表的文件形式保存：其中，(1) 需要设计文件数据记录格式，以高效保存二叉树数据逻辑结构 (D,R) 的完整信息；(2) 需要设计二叉树文件保存和加载操作合理模式
19. 实现多个二叉树管理：可采用线性表的方式管理多个二叉树，线性表中的

每个数据元素为一个二叉树的基本属性，至少应包含有二叉树的名称。基于顺序表实现的二叉树管理

2.2 系统设计

数据结构设计：

首先是一棵二叉树的设计，通过二叉链表的方式实现，每个节点有指向孩子的指针 *lchild, *rchild 和节点的信息域 data。

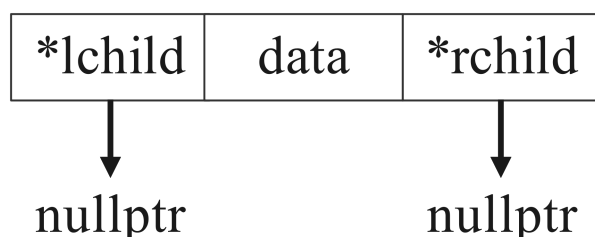


图 2-1 单个二叉树节点

其次是管理系统的设计，为方便，使用一个数组来线性存储二叉树。该数组与域 size、length 共同组成一个类 Manager。同时，为了便于操作，我为每棵二叉树分配了一个 name 域来标识二叉树。

整体结构如下：

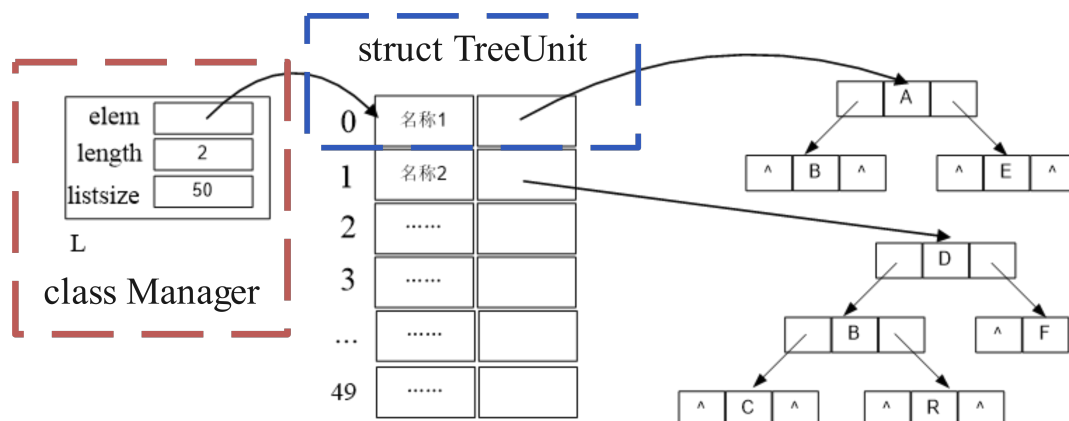


图 2-2 整体结构

整体系统结构设计设计：

按照项目规范，依然使用多文件编译进行架构。其中，CMakeLists.txt 是 cmake 构建文件；main.cpp 文件用于获取外部命令并执行相应的函数；def.h 存放引入的头文件、宏定义、以及结构体的定义；SingleTree 模块存放单棵树相关操作的函数声明与定义；Manager 模块存放类 Manager。除了用类进行多棵二叉树

的管理，程序主要采用面向过程编程的思想。

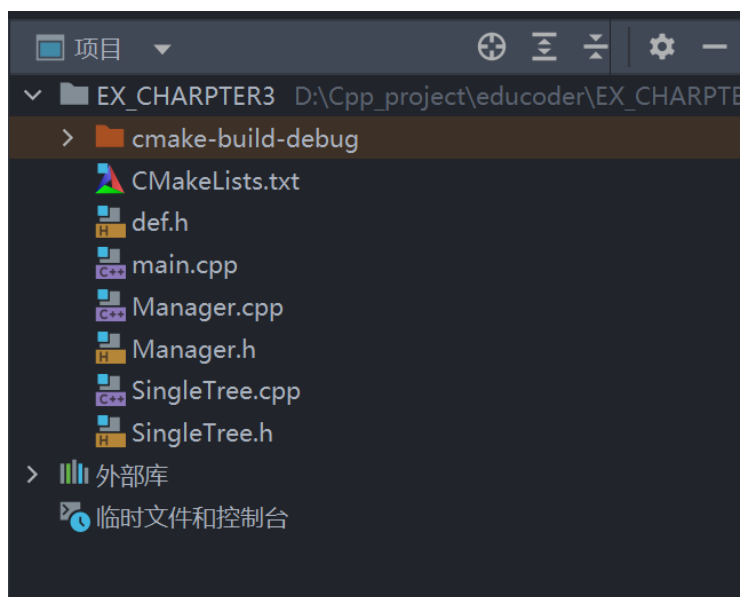


图 2-3 二叉树程序框架

2.3 系统实现

单棵二叉树操作

1. 创建二叉树：先用一个辅助函数 `GetData` 获取先序遍历数据 (带空节点), 然后使用一个递归函数 `build` 递归地、先序地构造一棵二叉树。
2. 销毁二叉树：用后序遍历的思想，先回收左节点，再回收右节点，最后回收自身。注意递归终止条件是碰到空节点。
3. 判空二叉树：判断传入的参数 `*T` 是否指向空即可。
4. 求二叉树深度：用后序遍历的思想，自身节点的深度等于左右孩子的较大者加一，即表达式 $depth_{cur} = \max(depth_{lc}, depth_{rc}) + 1$ 。
5. 查找节点：用先序遍历思想，若自身不是则去找左右孩子。递归终止条件是节点为空。伪代码如下：

算法 2.1. 查找节点

Input: root of the tree: `T`

Output: target tree node

procedure FIND(T)

if then T is *nullptr*

```
    return nullptr
end if
if then  $T$  is the target
    return  $T$ 
end if
 $l = find(T \rightarrow lchild)$ 
 $r = find(T \rightarrow rchild)$ 
    return  $l$  or  $r$  or  $nullptr$  according to the result
end procedure
```

6. 节点赋值：先找到节点，然后进行修改即可。
7. 获得兄弟节点：用先序遍历思想，先判断自身的左右孩子是否满足条件，再递归地判断左右孩子。注意递归终止条件是节点为空。
8. 插入节点：先找到插入位置，再用 `malloc` 分配一个新节点，调整原有的指针指向即可。
9. 删除节点：对查找函数进行略微改动(大体框架变成了一个带返回值的递归，节点的修改关键就在于递归！)，在查找到节点后进行指针域的调节，调节要分情况：

第一种情况：要删除的节点是叶子节点：直接 `free` 后返回 `nullptr` 即可。如图2-4

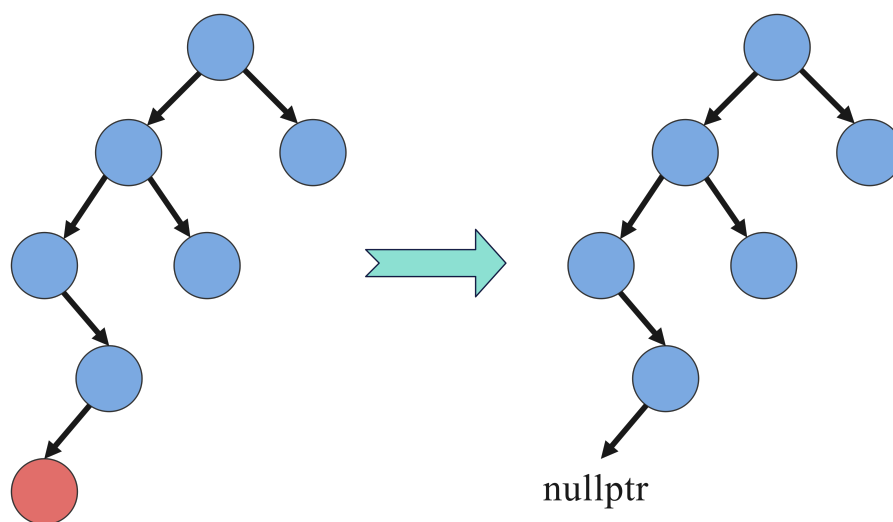


图 2-4 删除节点-情况 1

第二种情况：要删除的节点只有一个孩子节点：free 当前节点，将此孩子节点返回即可。如图2-5

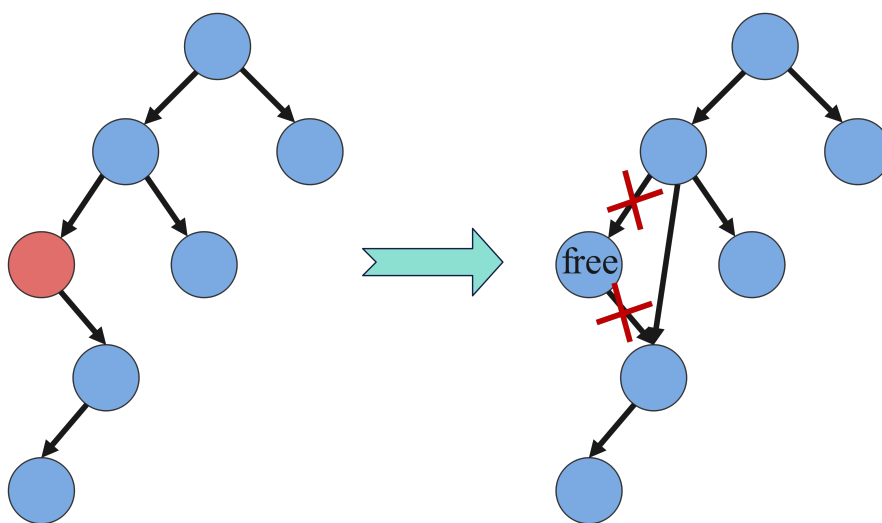


图 2-5 删除节点-情况 2

第三种情况：要删除的节点既有左孩子又有右孩子：将右子树挂到左子树的“最右侧”。如图2-6

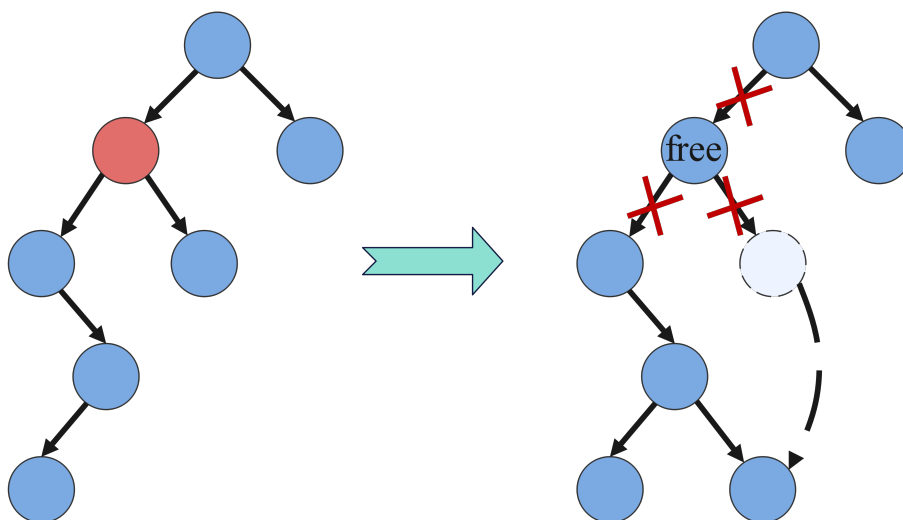


图 2-6 删除节点-情况 3

10. 前序遍历：这里采用非递归的方式实现。将节点按照中-右-左的顺序入栈，出栈时先访问，然后继续按照这样判断直到碰到空节点。
11. 中序遍历：递归，按照左-中-右的顺序即可。
12. 后序遍历：递归，按照左-右-中的顺序即可。
13. 层序遍历：类似于 bfs，每次将一个节点从队列 pop 出来就先访问，再将其孩子 push 入队列。

14. 最大路径和：使用后序遍历思想，从叶节点到自身节点的最大路径和为左孩子和右孩子路径的最大值加一。

即表达式 $Path_{cur} = \text{Max}(Path_{lc}, Path_{rc}) + 1$ 。

15. 最近公共祖先：使用后序遍历的思想，判断当前节点是否是左右孩子的最近公共祖先。
16. 反转二叉树：先序遍历思想，交换当前访问节点的左子树和右子树，在递归地反转左子树和右子树。递归终止条件是当前访问节点为空。伪代码如下：

算法 2.2. 反转二叉树

Input: root of the tree: T

Output: the tree root after invert

```
procedure INVERTTREE(T)
    if then T is nullptr
        return
    end if
    swap(T->lc, T->rc)
    InvertTree(T->lc)
    InvertTree(T->rc)
end procedure
```

管理系统操作

1. 添加一棵二叉树：与用户进行交互，获取树名、数据之后就调用 CreateTree 构建二叉树。
2. 删除一棵二叉树：获取用户输入的树名，根据索引表获取对应的 index，调用 ClearBiTree 即可。
3. GetCommand 函数：由于交互过程中要多次获取树名以及判断输入是否正确，特别编写了该函数，主要是用一个泛型指针接受树名对应的索引，如果指针指向 $name_index.end()$ 则说明输入出现问题。
4. 显示树形结构：和上一个实验中一样，使用迭代的方式打印目录即可。

2.4 系统测试

1. 新建一棵二叉树:

函数名: AddMember	
正常样例	1 a 2 b 4 d 0 null 0 null 5 e 0 null 0 null 3 c 0 null 0 null -1 nil
预期结果	创建成功并返回树形结构
异常样例	Node
预期结果	None

实测结果:

```

enter your choice:
1
enter a name for this tree:
湖北
enter the key-value pair(preorder) end with key '-1':
1 a 2 b 4 d 0 null 0 null 5 e 0 null 0 null 3 c 0 null 0 null -1 nil
read successfully!
add successfully. The preorder series is: 1,a 2,b 4,d 5,e 3,c
the structure now is:
-----
|-Manager
  |-湖北
-----
  
```

图 2-7 测试结果-1

2. 删除一棵二叉树:

函数名: DelMember	
正常样例	湖北
预期结果	删除成功并返回树形结构
异常样例	湖南
预期结果	显示找不到树

实测结果:

```
enter your command:
2
enter the name of the tree:
湖南
can't find the tree!
```

(a) 正常样例

```
enter your command:
2
enter the name of the tree:
湖北
delete successfully, the structure now is:
-----
|-Manager
  |-null
-----
```

(b) 异常样例

图 2-8 测试结果-2

3. 销毁二叉树

函数名: ClearBiTree	
正常样例	湖北
预期结果	显示清除成功并返回结构
异常样例	湖北 (空树)
预期结果	empty tree!

测试结果:

```
enter your command:
5
enter the name of the tree:
湖北
clear successfully! Now the structure is:
-----
|-Manager
  |-null
-----
```

(a) 正常样例

```
enter your command:
5
enter the name of the tree:
湖北
empty tree!
```

(b) 异常样例

图 2-9 测试结果-3

4. 判空二叉树

函数名: IsEmpty	
正常样例	湖北
预期结果	not empty
异常样例	湖北 (空树)
预期结果	empty!

测试结果:

```
enter your command:
21
enter the name of the tree:
湖北
not empty!
```

(a) 正常样例

```
enter your command:
21
enter the name of the tree:
湖北
empty!
```

(b) 异常样例

图 2-10 测试结果-4

5. 求二叉树深度

函数名：GetDepth	
正常样例	湖北
预期结果	3
异常样例	湖北 (空树)
预期结果	empty tree!

实测结果：

```
enter your command:
6
enter the name of the tree:
湖北
The depth is: 3
```

(a) 正常样例

```
enter your command:
6
enter the name of the tree:
湖北
empty tree!
```

(b) 异常样例

图 2-11 测试结果-5

6. 查找节点

函数名：FindNode	
正常样例	湖北 2
预期结果	2,b
异常样例	湖北 (空树)
预期结果	empty tree!

实测结果：

```
enter your command:
7
enter the name of the tree:
湖北
enter the key:
2
result: 2,b
```

(a) 正常样例

```
enter your command:
7
enter the name of the tree:
湖北
empty tree!
```

(b) 异常样例

图 2-12 测试结果-6

7. 节点赋值

函数名：AssignTreeNode	
正常样例	湖北 2 9 g
预期结果	显示成功并返回前序遍历
异常样例	湖北 2 5 e
预期结果	显示有重复键

实测结果：

```
enter your command:
8
enter the name of the tree:
湖北
enter the key and new key-value pair:
2 9 g
assign successfully! The preorder is:
1,a
2,e
3,c
```

(a) 正常样例

```
enter your command:
8
enter the name of the tree:
湖北
enter the key and new key-value pair:
2 5 e
duplicated key!
```

(b) 异常样例

图 2-13 测试结果-7

8. 获得兄弟节点

函数名：GetSibling	
正常样例	湖北 9
预期结果	显示成功并返回前序遍历
异常样例	湖北 2
预期结果	显示找不到兄弟节点

实测结果：

```
enter your command:
9
enter the name of the tree:
湖北
enter the key:
9
Sibling is 3,c
```

(a) 正常样例

```
enter your command:
9
enter the name of the tree:
湖北
enter the key:
2
can't find sibling!
```

(b) 异常样例

图 2-14 测试结果-8

9. 插入节点

函数名：InsertNode	
正常样例	湖北 5 2 b 0
预期结果	显示成功并返回前序遍历
异常样例	湖北 7 2 b 0
预期结果	显示找不到键

实测结果：

```

enter your command:
10
enter the name of the tree:
湖北
enter the key:
5
enter the new node info:
2 b
how to insert(enter -1 or 0 or 1):
0
insert over. The preorder traverse is:
1,a
9,g
4,d
5,e
2,b
3,c
        
```

```

enter your command:
10
enter the name of the tree:
湖北
enter the key:
7
enter the new node info:
2 b
how to insert(enter -1 or 0 or 1):
0
can't find the key!
        
```

(a) 正常样例

(b) 异常样例

图 2-15 测试结果-9

10. 删除节点

函数名：DeleteNode	
正常样例	湖北 2
预期结果	显示成功并返回前序遍历
异常样例	湖北 12
预期结果	显示找不到对应节点

实测结果：

```

enter your command:
11
enter the name of the tree:
湖北
enter the key:
2
delete over. The preorder is:
1,a
9,g
4,d
5,e
3,c
        
```

```

enter your command:
11
enter the name of the tree:
湖北
enter the key:
12
can't find the node!
        
```

(a) 正常样例

(b) 异常样例

图 2-16 测试结果-10

11. 前序遍历

函数名：PreOrder	
正常样例	湖北
预期结果	返回前序遍历
异常样例	None
预期结果	None

实测结果：

```
enter your command:
12
enter the name of the tree:
湖北
1,a
9,g
4,d
5,e
3,c
```

图 2-17 测试结果-11

12. 中序遍历

函数名：InOrder	
正常样例	湖北
预期结果	返回中序遍历
异常样例	None
预期结果	None

实测结果：

```
enter your command:
13
enter the name of the tree:
湖北
4,d
9,g
5,e
1,a
3,c
```

图 2-18 测试结果-12

13. 后序遍历

函数名: PostOrder	
正常样例	湖北
预期结果	返回后序遍历
异常样例	None
预期结果	None

实测结果:

```
enter your command:
14
enter the name of the tree:
湖北
4,d
5,e
9,g
3,c
1,a
```

图 2-19 测试结果-13

14. 层序遍历

函数名: LevelOrder	
正常样例	湖北
预期结果	返回层序遍历
异常样例	None
预期结果	None

实测结果:

```
enter your command:
15
enter the name of the tree:
湖北
1,a
9,g
3,c
4,d
5,e
```

图 2-20 测试结果-14

15. 最大路径和

函数名: MaxPathSum	
正常样例	湖北
预期结果	返回最大路径和
异常样例	None
预期结果	None

实测结果:

```
enter your command:
18
enter the name of the tree:
湖北
max sum is: 15
```

图 2-21 测试结果-15

16. 最近公共祖先

函数名: LCA	
正常样例	湖北 1 3
预期结果	1,a
异常样例	湖南
预期结果	显示找不到对应树

实测结果:

```
enter your command:
19
enter the name of the tree:
湖北
enter 2 child
1 3
the ancestor is: 1,a
```

(a) 正常样例

```
enter your command:
19
enter the name of the tree:
湖南
can't find the tree!
```

(b) 异常样例

图 2-22 测试结果-16

17. 翻转二叉树

函数名: InvertTree	
正常样例	湖北
预期结果	显示翻转后的前序遍历
异常样例	湖南
预期结果	显示找不到对应树

实测结果:

```
enter your command:
20
enter the name of the tree:
湖北
Invert over, the preorder is:
1,a
3,c
9,g
5,e
4,d
```

(a) 正常样例

```
enter your command:
20
enter the name of the tree:
湖南
can't find the tree!
```

(b) 异常样例

图 2-23 测试结果-16

2.5 实验小结

内容收获:

1. 掌握了用二叉链表实现二叉树以及二叉树的基本操作。基本任务圆满完成。
2. 复习了面向过程编程及一些代码规范
3. 学习了如何使用 clion 搭建跨平台项目。

一些典型错误:

1. 没有判断指针是否指向空导致 segmentation fault。
2. 经常忘记一些递归终止条件。

经验:

1. 递归三部曲: 基本情况, 递归函数, 终止条件。
2. 头文件和源文件分离的结构, 头文件中存放相关的函数、类声明, 源文件中存放相关的定义。
3. 对于一些小型课设, 上次说面向对象编程会造成一些不必要的冗余, 这次使用面向过程编程, 更加方便了。

4. 能抽象成函数的部分尽量抽象成函数，重复的语句一定要用函数，否则后面调试维护时会很麻烦。
5. 指针不用时一定要置空，并且使用指针时一定要判空。

3 课程的收获和建议

3.1 基于顺序存储结构的线性表实现

收获:

1. 掌握了线性表的数组存储结构以及基本操作。
2. 通过头歌的学习掌握了面向过程编程的一些知识。
3. 学会了使用 vscode+cmake 构建、调试跨平台项目

建议:

1. 可以多放一些数组相关的经典题目作为练习而适当减少过于基础的操作

3.2 基于链式存储结构的线性表实现

收获:

1. 掌握了线性表的链式存储结构以及基本操作。
2. 通过头歌的学习掌握了面向过程编程的一些知识。
3. 学会了使用 VS+cmake 构建、调试跨平台项目

建议:

1. 可以适当讲解更多的链式存储结构的知识

3.3 基于二叉链表的二叉树实现

收获:

1. 掌握了树的链式存储结构以及基本操作。
2. 通过头歌的学习掌握了面向过程编程的一些知识。
3. 学会了使用 clion+cmake 构建、调试跨平台项目

建议:

1. 很充实很全面花了很久时间写头歌的题目。谢谢你，让我巩固知识。
2. 可以适当补充数组存储的二叉树结构
3. 可以补充二叉搜索树以及 avl 树。

3.4 基于邻接表的图实现

收获:

1. 掌握了图的邻接表存储结构以及基本操作。
2. 通过头歌的学习掌握了面向过程编程的一些知识。
3. 学会了使用 clion+cmake 构建、调试跨平台项目。

建议:

1. 很充实很全面花了很久时间写头歌的题目。谢谢你，让我巩固知识。
2. 头歌里图的存储结构可以先从简单的邻接矩阵开始。邻接表难度感觉有点大。

参考文献

- [1] 严蔚敏, 吴伟民. 数据结构 [M]. [S.l.]: 清华大学出版社, 2019.
- [2] 严蔚敏, 吴伟民. 数据结构 [M]. [S.l.]: 清华大学出版社, 2019.

附录 A 基于顺序存储结构线性表实现的源程序

/* Linear Table On Sequence Structure */

main.cpp

```
#include "def.h"
#include "Management.h"
#include "SingleList.h"
#include "command_op.h"
using namespace std;

int main()
{
    int command;
    cout << "Initializing..." << "\n";
    List L;
    L.Init();
    cout << "A list(size 10) has been created" << "\n";
    cout << "Use the following command to operate the list.\n";
    main_menu_disp();
    cin >> command;
    while (command){
        response(L, command); // 获取线性表管理命令，跳转到处理语句
        cin >> command;
    }
    cout << "bye" << endl;
    system("pause");
    return 0;
}
```

def.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <algorithm>
#include "stdio.h"
#include "stdlib.h"
#include "stdio.h"
```

```
using namespace std;

#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

#define LIST_INIT_SIZE 100
#define LISTINCREMENT 10
typedef int ElemType;
```

SingleList.h

```
#include "def.h"
# pragma once

class SqList {
private:
    int* elem;
    int length;
    int listsize;
    string name;

public:
    SqList();

    //获取表名
    string GetName();

    //初始化
    status InitList(string name);

    //为刚刚初始化的表读入数据
    status ReadData();

    status DestroyList();
```



```
status ClearList();

//线性表判空
status ListEmpty();

//线性表长度
status ListLength();

//获取元素
status GetElem(int i, ElemType &e);

//查找元素
int LocateElem(ElemType e);

//获取前驱
status PriorElem(ElemType e, ElemType &pre);

//获取后继
status NextElem(ElemType e, ElemType &next);

//将元素e插入到线性表L的第i个元素之前
status ListInsert(int i, ElemType e);

//删除线性表L的第i个元素，保存在e中
status ListDelete(int i, ElemType &e);

//遍历线性表
status ListTraverse();

//最大连续子序列
status MaxSubArr();

//排序
void SortList();

//和为k的子数组
status SubArrNum(int k);

//保存表作为友元
//friend status List::Save(string fname);
};
```

SingleList.cpp

```
#include "SingleList.h"

SqlList::SqlList(){
    this->elem = nullptr;
}

string SqlList::GetName(){
    return this->name;
}

status SqlList::InitList(string name){
    if (this->elem != nullptr) return INFEASIBLE;
    else{
        this->elem = (int*)malloc(sizeof(int)*LIST_INIT_SIZE);
        this->length = 0;
        this->listsize = LIST_INIT_SIZE;
        this->name = name;
        return OK;
    }
}

status SqlList::ReadData(){
    if (this->elem == nullptr) return INFEASIBLE;
    if (this->length != 0) return ERROR;
    else{
        cout << "enter a series of numbers ended with 0, seperated with\n";
        a space:\n";
        int num;
        cin >> num;
        while (num){
            if (this->length == this->listsize){
                return OVERFLOW;
            }
            this->elem[this->length++] = num;
            // cout << this->length << endl;
            cin >> num;
        }
        // cout << "read:" << endl;
        // for(int i = 0; i < this->length; i++){
            // cout << this->elem[i] << " ";
        }
```

```
        // }
        //cout << "\n";
        return OK;
    }

}

status SqList::DestroyList(){
    if (this->elem == nullptr) return INFEASIBLE;
    else{
        ElemType* p = this->elem;
        this->elem = nullptr;
        delete p;
        return OK;
    }
}

status SqList::ClearList(){
    if (this->elem == nullptr) return INFEASIBLE;
    else{
        this->length = 0;
        return OK;
    }
}

status SqList::ListEmpty(){
    if (this->elem == nullptr) return INFEASIBLE;
    else{
        return this->length == 0 ? TRUE : FALSE;
    }
}

status SqList::ListLength(){
    if (this->elem) return this->length;
    else return INFEASIBLE;
}

status SqList::GetElem(int i, ElemType &e){
    if (this->elem == nullptr) return INFEASIBLE;
    else{
        if (i > this->length || i <= 0) return ERROR;
        else {
```

```
        e = this->elem[i-1];
        return OK;
    }
}

int SqList::LocateElem(ElemType e){
    if (this->elem == nullptr) return INFEASIBLE;
    else{
        for (int i = 0; i < this->length; i++){
            if (this->elem[i] == e) return i+1;
        }
        return 0;
    }
}

status SqList::PriorElem(ElemType e, ElemType &pre){
    if (this->elem == nullptr) return INFEASIBLE;
    if (this->length == 0 || this->elem[0] == e) return ERROR;
    for (int i = 0; i < this->length-1; i++){
        if (this->elem[i+1] == e){
            pre = this->elem[i];
            return OK;
        }
    }
    return ERROR;
}

status SqList::NextElem(ElemType e, ElemType &next){
    if (this->elem == nullptr) return INFEASIBLE;
    if (this->length <= 1 || this->elem[this->length-1] == e) return
    ERROR;
    for (int i = 0; i < this->length-1; i++){
        if (this->elem[i] == e){
            next = this->elem[i+1];
            return OK;
        }
    }
    return ERROR;
}
```

```
status Sqlist::ListInsert(int i, ElemType e){
    if (this->elem == nullptr) return INFEASIBLE;
    if (i <= 0) return ERROR;
    //还未用new重写
    this->elem = (int*)realloc(this->elem, sizeof(int)*100);
    if (i == this->length+1){
        this->elem[this->length++] = e;
        return OK;
    }
    if (i > this->length) return ERROR;

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

status Sqlist::ListDelete(int i, ElemType &e){
    if (this->elem == nullptr) return INFEASIBLE;
    if (i <= 0 || i > this->length) return ERROR;
    e = this->elem[i-1];
    for (int j = i-1; j < this->length-1; j++){
        this->elem[j] = this->elem[j+1];
    }
    this->length--;
    return OK;
}

status Sqlist::ListTraverse(){
    if (this->elem == nullptr) return INFEASIBLE;
    cout << this->GetName() << ": ";
    for(int i = 0; i < this->length; i++){
        cout << this->elem[i] << " ";
    }
    cout << "\n";
    return OK;
}

status Sqlist::MaxSubArr(){
```

```
if (!this->elem) return INFEASIBLE;
else if (this->length == 0) return INFEASIBLE;
else{
    vector<int> dp(this->length, 0);
    dp[0] = max(0, this->elem[0]);
    int ans = 0;
    for (int i = 1; i < this->length; i++){
        dp[i] = dp[i-1] + this->elem[i];
        if (dp[i] < 0) dp[i] = 0;
        ans = max(ans, dp[i]);
    }
    return ans;
}

}

void SqList::SortList(){
    sort(this->elem, this->elem+this->length);
}

status SqList::SubArrNum(int k){
    if (!this->elem) return INFEASIBLE;
    vector<int> presum(this->length+1, 0);
    presum[1] = this->elem[0];
    for (int i = 1; i < this->length; i++){
        presum[i+1] = presum[i] + this->elem[i];
    }
    int ans = 0;
    for (int i = 0; i < this->length; i++){
        for (int j = i+1; j <= this->length; j++){
            if (presum[j] - presum[i] == k) ans++;
        }
    }
    return ans;
}
```

Management.h

```
#include "def.h"
#include "SingleList.h"
#pragma once

class List {
```

```
private:
    int length;
    int listsize;
    vector<SqList> lists = vector<SqList>(10);
public:
    //初始化一张表
    status Init();

    //获取当前表长
    int GetLen();

    //插入一个新子表
    status AddList(string name);

    //移除第一个一个名字为name的子表
    status RemoveList(string name);

    //返回一个子表
    SqList& Query(int index);

    //查找第一个名为name的子表
    int LocateList(string name);

    //保存表
    status Save(string fname);

    //从文件读取数据使表初始化
    status Load(string fname);
};
```

Management.cpp

```
#include "Management.h"

status List::Init(){
    this->length = 0;
    this->listsize = 10;
    return OK;
}

int List::GetLen(){
    return this->length;
}
```

```
}

status List::AddList(string name){
    if (this->length == this->listsize) cout << "List has been full!"
    << endl;
    int state = this->lists[this->length++].InitList(name);
    if (state == OK) cout << "Add successfully\n" << endl;
    return OK;
}

status List::RemoveList(string name){
    for (int i = 0; i < this->length; i++){
        //printf("%d\n", i);
        if (this->lists[i].GetName() == name){
            this->lists[i].DestroyList();
            for (int j = i; j < this->length-1; j++){
                this->lists[j] = this->lists[j+1];
            }
            this->length--;
            //printf("%d\n", i);
            return OK;
        }
    }
    return ERROR;
}

SqlList& List::Query(int index){
    return this->lists[index-1];
}

status List::LocateList(string name){
    for (int i = 0; i < this->length; i++){
        if (this->lists[i].GetName() == name){
            return i+1;
        }
    }
    return 0;
}

status List::Save(string fname){
    ofstream ofile(fname, ios::app);
    cout << "creating file...";
}
```



```
if (ofile){
    cout << "done!" << endl;
    cout << "reading..." << endl;
    for (int i = 0; i < this->length; i++){
        SqList sublist = this->lists[i];
        ofile << sublist.GetName() << " ";
        int e;
        for (int j = 1; j <= sublist.ListLength(); j++){
            int state = sublist.GetElem(j, e);
            ofile << e << " ";
        }
        ofile << 0 << endl;
    }
    cout << "done!" << endl;
    return OK;
}

cout << "error!" << endl;
return ERROR;
}

status List::Load(string fname){
    ifstream infile(fname, ios::in);
    cout << "opening file...";
    if (infile){
        cout << "done!" << endl;
        string name;
        int i = 0; //当前在处理第几个列表
        while (infile >> name){
            this->AddList(name);
            SqList& sublist = this->lists[i];
            //sublist.InitList(name);
            int num;
            infile >> num;
            while (num){
                sublist.ListInsert(1, num);
                cout << "insert" << num << endl;
                infile >> num;
            }
            i++;
        }
        return OK;
    }
}
```

华中科技大学课程实验报告

```
    }  
    return ERROR;  
}
```

command_op.h

```
#include "def.h"  
#include "Management.h"  
#include "SingleList.h"  
#pragma once  
  
//线性表管理菜单  
void main_menu_disp();  
  
//单个表管理菜单  
void sub_menu_disp();  
  
//主表操作  
void response(List& L, int command);  
  
//子表操作  
void operate_single_list(Sqlist &SubL);
```

command_op.cpp

```
#include "command_op.h"  
  
void main_menu_disp(){  
    printf("      Menu for Linear Table On Sequence Structure  
      Management \n");  
    printf("-----\n");  
    printf("          1. add_a_list      4. goto_a_list\n");  
    printf("          2. remove_a_list   5. save_as_file\n");  
    printf("          3. locate_a_list   6. load_from_file\n");  
    printf("          0. quit\n");  
    printf("-----\n");  
}  
  
void sub_menu_disp(){  
    printf("      Menu for Linear Table On Single List \n");  
    printf("-----\n");  
    printf("          1. InitList      7. GetElem\n");
```

```
printf("          2. read_data      8. LocateElem\n");
printf("          3. DestroyList    9. PriorElem \n");
printf("          4. ClearList        10. NextElem\n");
printf("          5. ListEmpty        11. ListInsert\n");
printf("          6. ListLength       12. ListDelete\n");
printf("          13. ListTraverse    14. MaxSubArr\n");
printf("          15.SortList         16. SubArrSum\n");
printf("          0.quit\n");
printf("-----\n");

}

void response(List& L, int command){
    string name;
    if (command == 1){ // 初始化新建一个表
        cout << "input name:\n";
        cin >> name;
        while (name == "\n"){
            cout << "please enter a name:";
            cin >> name;
        }
        L.AddList(name);
    }
    else if (command == 2){ // 移除表
        cout << "enter the name of the list to be removed\n";
        cin >> name;
        int state = L.RemoveList(name);
        if (state == OK){
            cout << "remove successfully\n";
        }
        else{
            cout << "Can't find sublist named" << name << "!\n";
        }
    }
    else if (command == 3){ // 根据名字定位表
        cout << "enter the name:" << endl;
        cin >> name;
        int state = L.LocateList(name);
        if (state == 0){
            cout << "Can't find sublist named " << name << "!\n";
        }
    }
}
```

```
        else{
            cout << "The sublist named " << name << " locates in " <<
                state << \
                "th position.\n";
        }
    }
else if (command == 4){ // 进入单表中进行操作
    cout << "enter the name of the sublist:\n";
    cin >> name;
    int state = L.LocateList(name);
    if (state == 0){
        cout << "wrong name!\n";
    }
    else{
        SqList& sublist = L.Query(state); //这里如果不用引用来接受返回
        值，后面会出问题!!! 会导致本体并没有改变!!!
        operate_single_list(sublist);
    }
}

else if (command == 5){ // 保存
    cout << "file name?" << endl;
    string fname;
    cin >> fname;
    int state = L.Save(fname);
    if (state == OK) cout << "save successfully!" << endl;
    else cout << "error!" << endl;
}

else if (command == 6){ // 导入
    cout << "file name?" << endl;
    string fname;
    cin >> fname;
    int state = L.Load(fname);
    if (state == OK) cout << "load successfully!" << endl;
    else cout << "error!" << endl;
}

else{
    cout << "invalid command!" << endl;
}

cout << "your next command?" << endl;
}
```

```
//子表操作
void operate_single_list(SqlList& SubL){
    //system("cls");
    cout << "you have been entered list " << SubL.GetName() << endl;
    cout << "list command are as follows:" << endl;
    sub_menu_disp();
    int command;
    int state;
    string name;
    cout << "your command?\n";
    cin >> command;
    while (command){
        //初始化单个表
        if (command == 1){
            cout << "enter a name for this list:\n";
            cin >> name;
            state = SubL.InitList(name);
            if (state == INFEASIBLE){
                cout << "You can't initialize an existed list!\n";
            }
            else{
                cout << "initialize successfully!\n";
            }
        }
        //读取数据
        else if (command == 2){
            state = SubL.ReadData();
            if (state == INFEASIBLE){
                cout << "inexisted list!\n";
            }
            else if (state == OVERFLOW){
                cout << "too many numbers! some has been lost!" << "\n";
            }
            else if (state == ERROR){
                cout << "some elements have been exist!" << endl;
            }
            else{
                cout << "read successfully!\n";
            }
        }
    }
}
```

```
}  
  
//删除列表使之变为null  
else if (command == 3){  
    state = SubL.DestroyList();  
    if (state == INFEASIBLE){  
        cout << "You can't destroy an inexisted list!\n";  
    }  
    else{  
        cout << "destroy successfully!\n";  
    }  
}  
  
//清空列表,但表仍然存在  
else if (command == 4){  
    state = SubL.ClearList();  
    if (state == INFEASIBLE){  
        cout << "You can't clear an inexisted list!\n";  
    }  
    else{  
        cout << "clear successfully!\n";  
    }  
}  
  
//判断列表是否空  
else if (command == 5){  
    state = SubL.ListEmpty();  
    if (state == INFEASIBLE){  
        cout << "list inexists\n";  
    }  
    else{  
        if (state == 1) cout << "empty!" << endl;  
        else cout << "not empty!" << endl;  
    }  
}  
  
//获取列表长度  
else if (command == 6){  
    state = SubL.ListLength();  
    if (state == INFEASIBLE){  
        cout << "inexisted list!\n";  
    }  
    else{  
        cout << "the length is " << state << "\n";  
    }  
}
```

```
}
//获取指定位置元素
else if (command == 7){
    int index, e;
    cout << "please enter the index:\n";
    cin >> index;
    state = SubL.GetElem(index, e);
    if (state == INFEASIBLE){
        cout << "inexisted list!\n";
    }
    else if (state == ERROR){
        cout << "index out of range!\n";
    }
    else{
        cout << "The elem is " << e << "\n";
    }
}
//获取指定元素位置
else if (command == 8){
    int e;
    cout << "enter the element:\n";
    cin >> e;
    state = SubL.LocateElem(e);
    if (state == INFEASIBLE){
        cout << "inexisted list!\n";
    }
    else if (state == 0){
        cout << "Element " << e << " doesn't exist!\n" << endl;
    }
    else{
        cout << "position is" << state << "\n";
    }
}
//获取前驱
else if (command == 9){
    int e, pre;
    cout << "enter the element:\n";
    cin >> e;
    state = SubL.PriorElem(e, pre);
    if (state == INFEASIBLE){
        cout << "inexisted list!\n";
```

```
    }
    else if (state == 0){
        cout << "prior element doesn't exist!\n" << endl;
    }
    else{
        cout << "prior element is " << pre << "\n";
    }
}
//获取后继
else if (command == 10){
    int e, next;
    cout << "enter the element:\n";
    cin >> e;
    state = SubL.NextElem(e, next);
    if (state == INFEASIBLE){
        cout << "inexisted list!\n";
    }
    else if (state == 0){
        cout << "next element doesn't exist!\n" << endl;
    }
    else{
        cout << "next element is " << next << "\n";
    }
}
//插入元素
else if (command == 11){
    int e, i;
    cout << "enter the element:\n";
    cin >> e;
    cout << "enter the position:\n";
    cin >> i;
    state = SubL.ListInsert(i, e);
    if (state == INFEASIBLE){
        cout << "inexisted list!\n";
    }
    else if (state == 0){
        cout << "wrong insert position!\n" << endl;
    }
    else{
        cout << "insert successfully!\n";
    }
}
```



```
}  
//删除元素  
else if (command == 12){  
    int e, i;  
    cout << "enter the position to be deleted:\n";  
    cin >> i;  
    state = SubL.ListDelete(i, e);  
    if (state == INFEASIBLE){  
        cout << "inexisted list!\n";  
    }  
    else if (state == 0){  
        cout << "wrong delete position!\n" << endl;  
    }  
    else{  
        cout << "delete successfully!\n";  
    }  
}  
//遍历表  
else if (command == 13){  
    state = SubL.ListTraverse();  
    if (state == INFEASIBLE){  
        cout << "inexisted list!\n";  
    }  
    else{  
        cout << "traverse over\n";  
    }  
}  
else if (command == 14){  
    state = SubL.MaxSubArr();  
    if (state == INFEASIBLE){  
        cout << "inexisted list or the length less than 1!" <<  
            endl;  
    }  
    else{  
        cout << "the max sum is " << state << endl;  
    }  
}  
else if (command == 15){  
    SubL.SortList();  
}  
else if (command == 16){
```

```
        cout << "enter the sum k" << endl;
        int k;
        cin >> k;
        state = SubL.SubArrNum(k);
        if (state == INFEASIBLE){
            cout << "inexisted list" << endl;
        }
        else{
            cout << "the num sumed k is " << state << endl;
        }
    }
    else{
        cout << "invalid command!" << endl;
    }
    cout << "your command?\n";
    cin >> command;
}

//system("cls");
cout << "you have been backed to the manager.Choose your command."
<< endl;
main_menu_disp();
}
```

附录 B 基于链式存储结构线性表实现的源程序

/* Linear Table On LinkList Structure */

main.cpp

```
#include "def.h"
#include "LinkList.h"
#include "Management.h"
#include "command_op.h"

int main()
{
    cout << "Initializing..." << endl;
    Manager MainList;
    cout << "An empty list has been constructed." << endl;
    cout << "Use the following command to operate the list." << endl;
    int op, state;
    int flag = 1;
    main_menu();
    cout << "enter your command:" << endl;
    cin >> op;
    while (op) {
        switch (op) {
            case 1:
                state = MainList.NewList();
                if (state) {
                    cout << "Complete. now the structure is:" << endl;
                    structure_disp(MainList);
                }
            else {
                cout << "Memory error!" << endl;
                flag = 0;
            }
            break;
            case 2:
                int index;
                cout << "enter the num of the list:" << endl;
                cin >> index;
                state = MainList.DellList(index);
                if (state == OK) {
```

```
        cout << "delete successfully. Now the structure is:" << endl;
        structure_disp(MainList);
    }
    else if (state == INFEASIBLE) {
        cout << "empty table!" << endl;
    }
    else {
        cout << "index out of range." << endl;
        //flag = 0;
    }
    break;
case 3:
    /*单独使用一个函数来处理单链表操作*/
    state = single_list_op(MainList);
    if (state == ERROR) cout << "index out of range" << endl;
    cout << "now the structure is:" << endl;
    structure_disp(MainList);
    break;
case 4:
    structure_disp(MainList);
    break;
default:
    cout << "wrong command!" << endl;
}
if (flag == 0) break;
main_menu();
cout << "enter your command:" << endl;
cin >> op;
}
cout << "bye." << endl;
return 0;
}
```

def.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include "stdio.h"
```

```
#include "stdlib.h"
using namespace std;
#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
#define LIST_INIT_SIZE 100
#define LISTINCREMENT 10
typedef int ElemType;
typedef struct LNode { //单链表（链式结构）结点的定义
    ElemType data;
    struct LNode* next;
}LNode, * LinkList;
```

LinkList.h

```
#pragma once
#include "def.h"

class List {
private:
    int length;
    LinkList head; //空表头
    LinkList tail; //指向尾部的指针方便操作

public:
    List* down; //为线性表管理做准备，采用十字链表存储

    List();

    //获取头指针
    LinkList GetHead();

    //初始化表
    status InitList();

    //为表添加数据
    status AddData();
```

```
//销毁线性表
status DestroyList();

//清空线性表
status ClearList();

//线性表判空
status ListEmpty();

//返回线性表长度
int ListLength();

//获取第i个元素，保存在e中
status GetElem(int i, ElemType& e);

//查找元素e的位置序号
status LocateElem(ElemType e);

//获取线性表L中元素e的前驱，保存在pre中
status PriorElem(ElemType e, ElemType& pre);

//获取线性表L元素e的后继，保存在next中
status NextElem(ElemType e, ElemType& next);

//将元素e插入到线性表L的第i个元素之前
status ListInsert(int i, ElemType e);

//删除线性表L的第i个元素，并保存在e中
status ListDelete(int i, ElemType& e);

//遍历单链表
status ListTraverse();

//以文件形式保存表
status SaveList(char FileName[]);

//从文件加载表
status LoadList(char FileName[]);

//反转链表
```

```
status reverse();

//删除倒数第n个节点
status del_n();

//链表排序
status SortList();
};

LinkList _reverse(LinkList l, LinkList r);
```

LinkList.cpp

```
#include "LinkList.h"

List::List() {
    this->head = nullptr;
    this->tail = nullptr;
    this->length = 0;
    this->down = nullptr;
}

LinkList List::GetHead() {
    return this->head;
}

status List::AddData() {
    if (!this->head) return INFEASIBLE;
    cout << "enter a series of numbers sperated thit a space and end
    with 0" << endl;
    int num;
    cin >> num;
    while (num) {
        this->tail->next = (LinkList)malloc(sizeof(struct LNode));
        this->tail = this->tail->next;
        this->tail->next = nullptr;
        this->tail->data = num;
        this->length++;
        cin >> num;
    }
    cout << "Add successfully." << endl;
    return OK;
}
```

```
}

status List::InitList() {
    if (this->head) return INFEASIBLE;
    this->head = (LinkedList)malloc(sizeof(struct LNode));
    this->head->next = nullptr;
    this->tail = head;
    return OK;
}

status List::DestroyList() {
    if (!this->head) return INFEASIBLE;
    LinkedList p;
    LinkedList L = this->head;
    while (L) {
        p = L;
        L = L->next;
        free(p);
    }
    this->head = nullptr;
    this->tail = nullptr;
    this->length = 0;
    return OK;
}

status List::ClearList() {
    if (!this->head) return INFEASIBLE;
    LinkedList p;
    LinkedList L0 = this->head->next;
    while (L0) {
        p = L0;
        L0 = L0->next;
        free(p);
    }
    this->head->next = nullptr;
    this->length = 0;
    return OK;
}

status List::ListEmpty() {
    if (!this->head) return INFEASIBLE;
```



```
    if (this->length == 0) return TRUE;
    return FALSE;
}

int List::ListLength() {
    if (!this->head) return INFEASIBLE;
    return this->length;
}

status List::GetElem(int i, ElemType& e) {
    if (this->head == nullptr) return INFEASIBLE;
    int cnt = 1;
    LinkList L = this->head->next;
    while (L) {
        if (cnt == i) {
            e = L->data;
            return OK;
        }
        L = L->next;
        cnt++;
    }
    return ERROR;
}

status List::LocateElem(ElemType e) {
    if (this->head == nullptr) return INFEASIBLE;
    int cnt = 1;
    LinkList L = this->head->next;
    while (L) {
        if (L->data == e) return cnt;
        L = L->next;
        cnt++;
    }
    return ERROR;
}

status List::PriorElem(ElemType e, ElemType& pre) {
    if (this->head == nullptr) return INFEASIBLE;
    LinkList L = this->head;
    if (L->next == nullptr || L->next->data == e) return ERROR;
    L = L->next;
```

```
while (L) {
    if (L->next == nullptr) return ERROR;
    if (L->next->data == e) {
        pre = L->data;
        return OK;
    }
    L = L->next;
}

status List::NextElem(ElemType e, ElemType& next) {
    if (!this->head) return INFEASIBLE;
    LinkList L = this->head;
    if (!L->next || !L->next->next) return ERROR;
    L = L->next;
    while (L) {
        if (L->data == e) {
            if (L->next) {
                next = L->next->data;
                return OK;
            }
            else return ERROR;
        }
        L = L->next;
    }
    return ERROR;
}

status List::ListInsert(int i, ElemType e) {
    LinkList L = this->head;
    if (!this->head) return INFEASIBLE;
    int cnt = 1;
    while (L->next) {
        if (cnt == i) {
            if (L->next == nullptr) {
                L->next = new (struct LNode);
                L->next->data = e;
                L->next->next = nullptr;
                this->length++;
                //L = L0;
                return OK;
            }
        }
        L = L->next;
        cnt++;
    }
    return ERROR;
}
```

```
    }  
    LinkList new_node = new (struct LNode);  
    new_node->data = e;  
    new_node->next = L->next;  
    L->next = new_node;  
    //L = L0;  
    this->length++;  
    return OK;  
}  
L = L->next;  
cnt++;  
}  
return ERROR;  
}
```

```
status List::ListDelete(int i, ElemType& e) {  
    if (!this->head) return INFEASIBLE;  
    LinkList L = this->head;  
    int cnt = 1;  
    while (L->next) {  
        if (cnt == i) {  
            e = L->next->data;  
            LinkList cell = L->next;  
            L->next = L->next->next;  
            free(cell);  
            //L = L0;  
            this->length--;  
            return OK;  
        }  
        cnt++;  
        L = L->next;  
    }  
    //L = L0;  
    return ERROR;  
}
```

```
status List::ListTraverse() {  
    if (!this->head) return INFEASIBLE;  
    LinkList L = this->head;  
    if (L->next == nullptr) return OK;  
    L = L->next;
```

```
while (L) {
    printf("%d ", L->data);
    L = L->next;
}
return OK;
}

status List::SaveList(char FileName[]) {
    if (!this->head) return INFEASIBLE;
    FILE* fp = fopen(FileName, "w");
    LinkList L = this->head->next;
    while (L) {
        fprintf(fp, "%d ", L->data);
        L = L->next;
    }
    fclose(fp);
    return OK;
}

status List::LoadList(char FileName[]) {
    if (this->head) return INFEASIBLE;
    this->head = new (struct LNode);
    LinkList L0 = this->head;
    FILE* fp = fopen(FileName, "r");
    int num;
    while (fscanf(fp, "%d", &num) != EOF) {
        L0->next = new (struct LNode);
        L0 = L0->next;
        L0->data = num;
        this->length++;
    }
    L0->next = nullptr;
    fclose(fp);
    return OK;
}

status List::reverse() {
    if (!this->head) return INFEASIBLE;
    if (this->length <= 1) return OK;
    LinkList pre = this->head->next, cur = pre->next, next0 = cur->next;
    while (cur) {
```

```
        cur->next = pre;
        pre = cur;
        cur = next0;
        if (next0) next0 = next0->next;
    }
    this->head->next->next = nullptr;
    this->head->next = pre;
    return OK;
}

status List::del_n() {
    cout << "enter the n:" << endl;
    int n, e;
    cin >> n;
    n = this->length - n + 1;
    return this->ListDelete(n, e);
}

status List::SortList() {
    if (!this->head) return INFEASIBLE;
    if (this->length <= 1) return OK;

    vector<int> arr;
    LinkList cur = this->head->next;
    while (cur) {
        arr.push_back(cur->data);
        cur = cur->next;
    }
    cur = this->head->next;
    sort(arr.begin(), arr.end());
    for (int i = 0; i < arr.size(); i++) {
        cur->data = arr[i];
        cur = cur->next;
    }
    return OK;
}

LinkList _reverse(LinkList l, LinkList r) {
    if (l->next == r) {
        r->next = l;
        return l;
    }
}
```

华中科技大学课程实验报告

```
    }  
    l = _reverse(l->next, r->next);  
    return l;  
}
```

management.h

```
#pragma once  
#include "def.h"  
#include "LinkList.h"  
  
/*多线性表管理：用十字链表方式存储*/  
class Manager {  
    private:  
        int length;  
        List* start;  
        List* end;  
  
    public:  
        Manager();  
  
        //访问私有变量  
        List* GetStart();  
        List* GetEnd();  
        int GetLen();  
  
        //在最后添加一个表,并完成初始化以及数据读入  
        status NewList();  
  
        //不自动读入数据，为文件管理服务  
        status NewList(int mode);  
  
        //删除最后一张表  
        status DelList(int index);  
};
```

management.cpp

```
#include "Management.h"  
  
Manager::Manager() {  
    this->length = 0;  
    this->start = new List();  
}
```

```
        this->end = this->start;
    }

    List* Manager::GetStart() {
        return this->start;
    }

    List* Manager::GetEnd() {
        return this->end;
    }

    int Manager::GetLen() {
        return this->length;
    }

    status Manager::NewList() {
        this->end->down = new List();
        if (!this->end->down) return ERROR;
        this->end = this->end->down;
        //this->end->InitList();
        this->end->down = nullptr;
        this->length++;

        int state;
        state = this->end->InitList();
        if (state) {
            cout << "initialize successfully" << endl;
        }
        else return ERROR;
        this->end->AddData();
        return OK;
    }

    status Manager::NewList(int mode) {
        this->end->down = new List();
        if (!this->end->down) return ERROR;
        this->end = this->end->down;
        //this->end->InitList();
        this->end->down = nullptr;
        this->length++;
    }
}
```

```
int state;
state = this->end->InitList();
if (state) {
    cout << "initialize successfully" << endl;
}
else return ERROR;
}

status Manager::DellList(int index) {
    if (this->start == this->end) return INFEASIBLE;
    List* pre = this->start;
    List* L = pre->down;
    int cnt = 1;
    while (L && cnt < index) {
        pre = L;
        L = L->down;
        cnt++;
    }
    if (cnt == index && L) {
        if (pre->down == this->end) {
            this->end = pre;
        }
        pre->down = L->down;
        delete L;
        this->length--;
        return OK;
    }
    else {
        return OVERFLOW;
    }
}
```

command_op.h

```
#pragma once
#include "def.h"
#include "LinkList.h"
#include "Management.h"

//菜单输出
void main_menu();
void menu_disp();
```



```
//展示树形结构
void structure_disp(Manager M);

//对某一list操作
status single_list_op(Manager M);
```

command_op.cpp

```
#include "command_op.h"

void main_menu() {
    printf("          Menu for Linear Table Manager\n");
    printf("-----\n");
    printf("          1. NewList      2. Dellist\n");
    printf("          3. goto        4. disp_tree_structure\n");
    printf("          0. quit\n");
    printf("-----\n");
}

void menu_disp() {
    printf("          Menu for Single Linear \n");
    printf("-----\n");
    printf("          1. InitList      7. LocateElem\n");
    printf("          2. DestroyList   8. PriorElem\n");
    printf("          3. ClearList     9. NextElem \n");
    printf("          4. ListEmpty     10. ListInsert\n");
    printf("          5. ListLength    11. ListDelete\n");
    printf("          6. GetElem       12. ListTraverse\n");
    printf("          13. reverseList  14. RemoveNthFromEnd\n");
    printf("          15. SortList     16. AddData\n");
    printf("          17. SaveList     18. LoadList\n");
    printf("          0. Exit\n");
    printf("-----\n");
}

void structure_disp(Manager M) {
    cout << "-----" << endl;
    if (M.GetStart() == M.GetEnd()) {
        cout << "|--main_list" << endl;
        cout << "    |--empty" << endl;
    }
}
```

```
}
else {
    cout << "--main_list" << endl;
    List* sublist = M.GetStart()->down;
    for (int i = 1; i <= M.GetLen(); i++) {
        if (sublist->GetHead() == nullptr) {
            cout << "    --null" << endl;
        }
        else if (sublist->ListLength() == 0) {
            cout << "    --initialized, length 0" << endl;
        }
        else {
            cout << "    --initialized, length " <<
                sublist->ListLength() << endl;
        }
        sublist = sublist->down;
    }
}

cout << "-----" << endl;
}

status single_list_op(Manager M) {
    cout << "enter the index of the list:" << endl;
    int pos;
    cin >> pos;
    if (pos > M.GetLen()) return ERROR;

    cout << "operation for a single list:" << endl;
    menu_disp();

    int cnt = 1;
    List* L = M.GetStart()->down;
    while (cnt != pos) {
        L = L->down;
        cnt++;
    }

    cout << "input your command:" << endl;
    int op, state;
    int i, e, pre, next;
    char name[100];
```

```
cin >> op;
while (op) {
    switch (op) {
        case 1:
            state = L->InitList();
            if (state == INFEASIBLE) cout << "The list has existed!" <<
endl;
            else {
                cout << "initialize successfully! Now the structure
is:" << endl;
                structure_disp(M);
            }
            break;
        case 2:
            state = L->DestroyList();
            if (state == INFEASIBLE) cout << "The list doesn't
existed!" << endl;
            else {
                cout << "destroy successfully! now the structure is:"
<< endl;
                structure_disp(M);
            }
            break;
        //清空线性表, 只留下表头
        case 3:
            state = L->ClearList();
            if (state == INFEASIBLE) cout << "The list doesn't
existed!" << endl;
            else {
                cout << "clear successfully! now the structure is:" <<
endl;
                structure_disp(M);
            }
            break;
        //判断线性表是否空
        case 4:
            state = L->ListEmpty();
            if (state == INFEASIBLE) cout << "inexisted list!" << endl;
            else if (state == TRUE) cout << "empty!" << endl;
            else cout << "not empty!" << endl;
            break;
```

```
//返回表长度
case 5:
state = L->ListLength();
if (state == INFEASIBLE) cout << "empty list" << endl;
else {
    cout << "the length is " << state << endl;
}
break;
//获取第i个元素
case 6:
cout << "enter the index of the element:" << endl;
//int i, e;
cin >> i;
state = L->GetElem(i, e);
if (state == INFEASIBLE) cout << "empty list" << endl;
else if (state == ERROR) cout << "can't find the element."
<< endl;
else {
    cout << "the element is " << e << endl;
}
break;
//返回元素的索引
case 7:
cout << "enter the num to be located:" << endl;
//int e;
cin >> e;
state = L->LocateElem(e);
if (state == INFEASIBLE) cout << "empty list" << endl;
else if (state == ERROR) cout << "can't find the element."
<< endl;
else {
    cout << "the element index is " << state << endl;
}
break;
//返回前驱
case 8:
cout << "enter the elem:" << endl;
//int e, pre;
cin >> e;
state = L->PriorElem(e, pre);
if (state == INFEASIBLE) cout << "empty list" << endl;
```

```
else if (state == ERROR) cout << "can't find the element."
<< endl;
else {
    cout << "the prior element is " << pre << endl;
}
break;
//返回后继
case 9:
    cout << "enter the elem:" << endl;
    //int e, next;
    cin >> e;
    state = L->NextElem(e, next);
    if (state == INFEASIBLE) cout << "empty list" << endl;
    else if (state == ERROR) cout << "can't find the element."
<< endl;
    else {
        cout << "the next element is " << next << endl;
    }
    break;
//在位置i插入元素
case 10:
    //int i, e;
    cout << "enter the location:" << endl;
    cin >> i;
    cout << "enter the elem:" << endl;
    cin >> e;
    state = L->ListInsert(i, e);
    if (state == INFEASIBLE) cout << "empty list" << endl;
    else if (state == ERROR) cout << "wrong position!." << endl;
    else {
        cout << "insert successfully! traverse: ";
        L->ListTraverse();
    }
    break;
//删除位置i的元素
case 11:
    //int i, e;
    cout << "enter the location:" << endl;
    cin >> i;
    state = L->ListDelete(i, e);
    if (state == INFEASIBLE) cout << "empty list" << endl;
```

```
else if (state == ERROR) cout << "wrong position!." << endl;
else {
    cout << "delete successfully" << endl;
}
break;
//遍历表
case 12:
state = L->ListTraverse();
if (state == INFEASIBLE) cout << "empty list" << endl;
else {
    cout << "traverse over." << endl;
}
break;
//反转链表
case 13:
state = L->reverse();
if (state == INFEASIBLE) cout << "empty list" << endl;
else {
    cout << "reverse over. traverse: ";
    L->ListTraverse();
}
break;
//删除倒数第n个元素
case 14:
state = L->del_n();
if (state == INFEASIBLE) cout << "empty list" << endl;
else if (state == ERROR) cout << "wrong position!." << endl;
else {
    cout << "delete successfully" << endl;
}
break;
// 链表排序
case 15:
state = L->SortList();
if (state == INFEASIBLE) cout << "empty list" << endl;
else {
    cout << "sort over. reverse: ";
    L->ListTraverse();
    cout << endl;
}
break;
```

```
// 在末尾加入元素
case 16:
    state = L->AddData();
    if (state == INFEASIBLE) cout << "empty list" << endl;
    break;
// 保存
case 17:
    cout << "enter the filename:" << endl;
    scanf("%s", name);
    state = L->SaveList(name);
    if (state == INFEASIBLE) cout << "empty list" << endl;
    else {
        cout << "save successfully!" << endl;
    }
    break;
// 导入
case 18:
    cout << "enter the filename:" << endl;
    scanf("%s", name);
    state = L->LoadList(name);
    if (state == INFEASIBLE) cout << "list existed!" << endl;
    else {
        cout << "load successfully! Now the structure is:" <<
            endl;
        structure_disp(M);
    }
    break;
default:
    cout << "wrong command!" << endl;
}
menu_disp();
cout << "input your command:" << endl;
cin >> op;
}
cout << "you have been backed to the main menu:" << endl;
main_menu();
return OK;
}
```

附录 C 基于二叉链表存储结构实现二叉树的源程序

/* Binary Tree On Binary LinkList Structure */

main.cpp

```
#include "def.h"
#include "SingleTree.h"
#include "Manager.h"

int main() {
    int state, op, index, e;
    char filename[30];
    BiTree tmp = nullptr;
    TElemType value;
    Manager M;
    Menu();
    cout << "enter your choice:" << endl;
    cin >> op;
    while (op){
        switch (op) {
            case 1: // 添加成员
                state = M.AddMember();
                if (state == OK){
                    cout << "add successfully. The preorder series is: ";
                    traverse(M.member[M.length-1].T);
                    cout << endl;
                    cout << "the structure now is:" << endl;
                    M.DispStructure();
                    cout << '\n';
                }
            else{
                cout << "false order or duplicated key!" << endl;
            }
            break;
            case 2: // 删除成员
                state = M.DelMember();
                if (state == OK) {
                    cout << "delete successfully, the structure now is:" <<
                        endl;
                    M.DispStructure();
                }
        }
    }
}
```



```
else cout << "empty tree!" << endl;
break;
case 3: // 显示树形目录
M.DispStructure();
break;
case 4: // 初始化
index = M.GetCommand1();
if (index != -1){
    if (IsEmpty(M.member[index].T) == false){
        cout << "existed tree!" << endl;
    }
    else{
        TElemType definition[100];
        GetData(definition);
        state = CreateBiTree(M.member[index].T, definition);
        if (state == OK) {
            cout << "create successfully! Now the structure
                is: " << endl;
            M.DispStructure();
        }
        else cout << "error!" << endl;
    }
}
break;
case 5: // 清空二叉树
if ((index = M.GetCommand1()) != -1){
    if (IsEmpty(M.member[index].T) == true){
        cout << "empty tree!" << endl;
    }
    else{
        state = ClearBiTree(M.member[index].T);
        if (state == OK) {
            cout << "clear successfully! Now the structure
                is: " << endl;
            M.DispStructure();
        }
        else cout << "error!" << endl;
    }
}
break;
case 6: // 获取树深
```

```
if ((index = M.GetCommand1()) != -1){
    if (IsEmpty(M.member[index].T) == true){
        cout << "empty tree!" << endl;
    }
    else{
        cout << "The depth is: " <<
            GetDepth(M.member[index].T) << endl;
    }
}
break;
case 7: // 查找节点
if ((index = M.GetCommand1()) != -1){
    if (IsEmpty(M.member[index].T) == true){
        cout << "empty tree!" << endl;
    }
    else{
        cout << "enter the key:" << endl;
        cin >> e;
        tmp = find(M.member[index].T, e);
        if (tmp){
            cout << "result: " << tmp->data.key << ", " <<
                tmp->data.others << endl;
        }
        else cout << "no results." << endl;
    }
}
break;
case 8: // 节点赋值
if ((index = M.GetCommand1()) != -1){
    if (IsEmpty(M.member[index].T) == true){
        cout << "empty tree!" << endl;
    }
    else{
        cout << "enter the key and new key-value pair:" <<
            endl;
        int key;
        cin >> key >> value.key >> value.others;
        state = Assign(M.member[index].T, key, value);
        if (state == -1) cout << "duplicated key!" << endl;
        else if (state == ERROR) cout << "can't find the
            corresponding node!" << endl;
```

```
        else {
            cout << "assign successfully! The preorder is:
                " << endl;
            PreOrder(M.member[index].T);
        }
    }
}
break;
case 9: // 获取兄弟
if ((index = M.GetCommand1()) != -1){
    if (IsEmpty(M.member[index].T) == true){
        cout << "empty tree!" << endl;
    }
    else{
        cout << "enter the key:" << endl;
        cin >> e;
        tmp = GetSibling(M.member[index].T, e);
        if (!tmp) cout << "can't find sibling!" << endl;
        else{
            cout << "Sibling is " << tmp->data.key << ", "
                << tmp->data.others << endl;
        }
    }
}
break;
case 10: // 插入节点
if ((index = M.GetCommand1()) != -1){
    if (IsEmpty(M.member[index].T) == true){
        cout << "empty tree!" << endl;
    }
    else{
        cout << "enter the key:" << endl;
        cin >> e;
        cout << "enter the new node info: " << endl;
        cin >> value.key >> value.others;
        cout << "how to insert(enter -1 or 0 or 1): " <<
            endl;
        int lr;
        cin >> lr;
        state = InsertNode(M.member[index].T, e, lr, value);
        if (state == -1) cout << "duplicated key!" << endl;
```

```
        else if (state == ERROR) cout << "can't find the
            key!" << endl;
        else{
            cout << "insert over. The preorder traverse is:
                " << endl;
            PreOrder(M.member[index].T);
        }
    }
}
break;
case 11: // 删除节点
if ((index = M.GetCommand1()) != -1){
    if (IsEmpty(M.member[index].T) == true){
        cout << "empty tree!" << endl;
    }
    else{
        cout << "enter the key:" << endl;
        cin >> e;
        state = DeleteNode(M.member[index].T, e);
        if (state == ERROR) cout << "can't find the node!"
            << endl;
        else{
            cout << "delete over. The preorder is: " <<
                endl;
            PreOrder(M.member[index].T);
        }
    }
}
break;
case 12: // 前序遍历
if ((index = M.GetCommand1()) != -1){
    if (IsEmpty(M.member[index].T) == true){
        cout << "empty tree!" << endl;
    }
    else{
        PreOrder(M.member[index].T);
        cout << endl;
    }
}
break;
case 13: // 中序遍历
```

```
if ((index = M.GetCommand1()) != -1){
    if (IsEmpty(M.member[index].T) == true){
        cout << "empty tree!" << endl;
    }
    else{
        InOrder(M.member[index].T);
        cout << endl;
    }
}
break;
case 14: // 后序遍历
if ((index = M.GetCommand1()) != -1){
    if (IsEmpty(M.member[index].T) == true){
        cout << "empty tree!" << endl;
    }
    else{
        PostOrder(M.member[index].T);
        cout << endl;
    }
}
break;
case 15: // 层序遍历
if ((index = M.GetCommand1()) != -1){
    if (IsEmpty(M.member[index].T) == true){
        cout << "empty tree!" << endl;
    }
    else{
        LevelOrder(M.member[index].T);
        cout << endl;
    }
}
break;
case 16: // 保存文件
if ((index = M.GetCommand1()) != -1){
    if (IsEmpty(M.member[index].T) == true){
        cout << "empty tree!" << endl;
    }
    else{
        cout << "enter the filename:" << endl;
        cin >> filename;
        state = SaveBiTree(M.member[index].T, filename);
    }
}
```

```
        if (state == ERROR) cout << "IO Error!" << endl;
        else cout << "save successfully!" << endl;
    }
}
break;
case 17: // 载入文件
if ((index = M.GetCommand1()) != -1){
    if (IsEmpty(M.member[index].T) == false){
        cout << "tree existed!" << endl;
    }
    else{
        cout << "enter the filename:" << endl;
        cin >> filename;
        state = LoadBiTree(M.member[index].T, filename);
        if (state == ERROR) cout << "IO Error!" << endl;
        else cout << "load successfully!" << endl;
    }
}
break;
case 18: // 最大路径和
if ((index = M.GetCommand1()) != -1){
    cout << "max sum is: " << MaxPathSum(M.member[index].T)
    << endl;
}
break;
case 19: // lca问题
if ((index = M.GetCommand1()) != -1){
    cout << "enter 2 child" << endl;
    int e1, e2;
    cin >> e1 >> e2;
    tmp = LCA(M.member[index].T, e1, e2);
    cout << "the ancestor is: " << tmp->data.key << ", " <<
    tmp->data.others << endl;
}
break;
case 20: // 翻转二叉树
if ((index = M.GetCommand1()) != -1){
    InvertTree(M.member[index].T);
    cout << "Invert over, the preorder is:" << endl;
    PreOrder(M.member[index].T);
}
```

```
        break;
    default:
        cout << "wrong command!" << endl;
        break;
    case 21:
        if ((index = M.GetCommand1()) != -1){
            auto judge = IsEmpty(M.member[index].T);
            if (judge == true) cout << "empty!" << endl;
            else cout << "not empty!" << endl;
        }
        break;
    }
    Menu();
    cout << "enter your command:" << endl;
    cin >> op;
}
cout << "bye!" << endl;
return 0;
}
```

def.h

```
#pragma once
#include "cstdio"
#include "cstdlib"
#include <string>
#include <iostream>
#include <queue>
#include <map>
using namespace std;

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define MaxSize 50

typedef int status;
typedef int KeyType;
typedef struct TreeElem{
```

```
    KeyType key;
    char others[30];
} TElemType; //二叉树结点类型定义

typedef struct BiTNode{ //二叉链表结点的定义
    TElemType data;
    struct BiTNode *lchild,*rchild;
    BiTNode() : lchild(nullptr), rchild(nullptr){}
} BiTNode, *BiTree;

typedef struct TreeUnit{
    string name;
    BiTree T;
    TreeUnit() : name("unnamed"), T(nullptr){}
} TU;
```

SingleTree.h

```
#pragma once
#include "def.h"

//展示菜单
void Menu();

//判断是否是空树
status IsEmpty(BiTree T);

//获取先序初始化数组
void GetData(TElemType definition[]);

// 由带空节点的前序创建二叉树
status CreateBiTree(BiTree &T,TElemType definition[]);
BiTree build(BiTree &cur, TElemType definition[], int &cnt); // 辅助函数
void traverse(BiTree T); // 先序遍历判断创建是否成功

//清空二叉树
status ClearBiTree(BiTree &T);
void clear(BiTree T); // 辅助函数

//求二叉树深度
int GetDepth(BiTree T);
```



```
//查找给定关键词的节点
BiTree find(BiTree cur, KeyType e);

//节点赋值, 要求保证关键词唯一性
status Assign(BiTree &T, KeyType e, TElemType value);
int traverse(BiTree T, KeyType e, TElemType value);

//获取兄弟节点
BiTNode* GetSibling(BiTree T, KeyType e);

//插入节点
status InsertNode(BiTree &T, KeyType e, int LR, TElemType c);
int traverse1(BiTree T, KeyType e, TElemType value); // 辅助判断键唯一性

//删除节点
status DeleteNode(BiTree &T, KeyType e);
BiTree delete_(BiTree cur, KeyType e); // 删除辅助函数
int traverse2(BiTree T, KeyType e); // 判断是否有目标

//前序遍历
void PreOrder(BiTree T);

//中序遍历
void InOrder(BiTree T);

//后序遍历
void PostOrder(BiTree T);

//层序遍历
void LevelOrder(BiTree T);

//保存到文件
status SaveBiTree(BiTree T, char FileName[]);
void save_traverse(BiTree T, FILE* fp); // 保存文件辅助函数

//从文件读取
status LoadBiTree(BiTree &T, char FileName[]);
BiTree read(BiTree T, FILE* fp); //文件读取辅助函数

//最大路径和
```

华中科技大学课程实验报告

```
int MaxPathSum(BiTree T);

//最近公共祖先
BiTree LCA(BiTree T, KeyType e1, KeyType e2);
int FindChild(BiTree T, KeyType e);

//翻转二叉树
void InvertTree(BiTree &T);
```

SingleTree.cpp

```
#include "SingleTree.h"

void Menu(){
    cout << "                      Menu" << endl;
    cout << "-----" << endl;
    cout << "    1. NewTree                2. DelTree" << endl;
    cout << "    3. DispStructure          " << endl;
    cout << endl;
    cout << "    4. CreateBiTree           5. ClearBiTree" << endl;
    cout << "    6. GetDepth               7. FindNode" << endl;
    cout << "    8. TreeNodeAssign         9. GetSibling" << endl;
    cout << "   10. InsertNode             11. DeleteNode" << endl;
    cout << "   12. PreOrder               13. InOrder" << endl;
    cout << "   14. PostOrder              15. LevelOrder" << endl;
    cout << "   16. Save                   17. Load" << endl;
    cout << "   18. MaxPathSum             19. LCA" << endl;
    cout << "   20. InvertTree             21. IsEmpty" << endl;
    cout << "    0. quit" << endl;
    cout << "-----" << endl;
}

status IsEmpty(BiTree T){
    if (!T) return true;
    else return false;
}

void GetData(TElemType definition[]){
    cout << "enter the key-value pair(preorder) end with key \'-1\':"
    << endl;
```

```
int key, i = 0;
string value;
cin >> key;
while (key != -1){
    definition[i].key = key;
    scanf("%s", definition[i].others);
    i++;
    cin >> key;
}
definition[i].key = -1;
scanf("%s", definition[i].others);
cout << "read successfully!" << endl;
}

status CreateBiTree(BiTree &T,TElemType definition[])
/*根据带空枝的二叉树先根遍历序列definition构造一棵二叉树，将根节点指针赋值给T
并返回OK，
如果有相同的关键字，返回ERROR.*/
{
    int hash[200] = {0};
    int i = 0;
    while (definition[i].key != -1){
        if (hash[definition[i].key] == 1 && definition[i].key != 0){
            return ERROR;
        }
        hash[definition[i].key] = 1;
        i++;
    }
    int cnt = 0;
    T = build(T, definition, cnt);
    return OK;
}

BiTree build(BiTree &cur, TElemType definition[], int& cnt){
    if (definition[cnt].key == 0){
        cnt++;
        return nullptr;
    }
    cur = (BiTree)malloc(sizeof(struct BiTNode));
    cur->data = definition[cnt++];
```

```
        cur->lchild = build(cur->lchild, definition, cnt);
        cur->rchild = build(cur->rchild, definition, cnt);
        return cur;
    }

    void traverse(BiTree T){
        if (T == nullptr) return;
        printf("%d,%s  ", T->data.key, T->data.others);
        traverse(T->lchild);
        traverse(T->rchild);
    }

    status ClearBiTree(BiTree &T)
    //将二叉树设置成空，并删除所有结点，释放结点空间
    {
        // 请在这里补充代码，完成本关任务
        /***** Begin *****/
        if (T == nullptr) return ERROR;
        clear(T);
        T = nullptr;
        return OK;

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

    void clear(BiTree T){
        if (T == nullptr) return;
        clear(T->lchild);
        clear(T->rchild);
        free(T);
    }

    int GetDepth(BiTree T){
        if (T == nullptr) return 0;
        int l = GetDepth(T->lchild) + 1;
        int r = GetDepth(T->rchild) + 1;
        if (l >= r) return l;
        else return r;
    }
}
```

```
BiTree find(BiTree cur, KeyType e){
    if (cur == nullptr) return nullptr;
    if (cur->data.key == e) return cur;
    BiTree ans1 = find(cur->lchild, e);
    BiTree ansr = find(cur->rchild, e);
    if (ans1) return ans1;
    if (ansr) return ansr;
    return nullptr;
}

status Assign(BiTree &T, KeyType e, TElemType value){
    if (traverse(T, e, value) == 0) return -1; // -1表示键重复
    BiTree ans = find(T, e);
    if (!ans) return ERROR;
    else{
        ans->data = value;
        return OK;
    }
}

int traverse(BiTree T, KeyType e, TElemType value){
    if (T == nullptr) return 1;
    if (T->data.key != e && T->data.key == value.key) return 0;
    else return traverse(T->lchild, e, value) && traverse(T->rchild, e,
value);
}

BiTNode* GetSibling(BiTree T, KeyType e){
    if (T == nullptr) return nullptr;
    if (!(T->lchild && T->rchild)) return nullptr;
    if (T->lchild->data.key == e) return T->rchild;
    if (T->rchild->data.key == e) return T->lchild;
    BiTree l = GetSibling(T->lchild, e);
    if (l) return l;
    BiTree r = GetSibling(T->rchild, e);
    if (r) return r;
    return nullptr;
}
```

```
status InsertNode(BiTree &T,KeyType e,int LR,TElemType c){
    if (traverse1(T, e, c) == 0) return -1;
    BiTree ans = find(T, e);
    if (!ans) return ERROR;
    BiTree new_node = (BiTree)malloc(sizeof(struct BiTNode));
    if (LR == 0){
        new_node->rchild = ans->lchild;
        new_node->lchild = nullptr;
        new_node->data = c;
        ans->lchild = new_node;
    }
    else if (LR == -1){
        new_node->rchild = T;
        new_node->lchild = nullptr;
        new_node->data = c;
        T = new_node;
    }
    else{
        new_node->rchild = ans->rchild;
        new_node->lchild = nullptr;
        new_node->data = c;
        ans->rchild = new_node;
    }
    return OK;
}

int traverse1(BiTree T, KeyType e, TElemType value){
    if (T == nullptr) return 1;
    if (T->data.key == value.key) return 0;
    else return traverse(T->lchild, e, value) && traverse(T->rchild, e,
value);
}

status DeleteNode(BiTree &T,KeyType e){
    if (!traverse2(T, e)) return ERROR;
    T = delete_(T, e);
    return OK;
}
```

```
BiTree delete_(BiTree cur, KeyType e){
    if (!cur) return nullptr;
    if (cur->data.key != e){
        cur->lchild = delete_(cur->lchild, e);
        cur->rchild = delete_(cur->rchild, e);
    }
    else{
        if (!cur->lchild && !cur->rchild){
            free(cur);
            return nullptr;
        }
        else if (!cur->lchild){
            BiTree p = cur->rchild;
            free(cur);
            return p;
        }
        else if (!cur->rchild){
            BiTree p = cur->lchild;
            free(cur);
            return p;
        }
        else{
            BiTree p = cur->lchild;
            BiTree p0 = p;
            while (p0->rchild){
                p0 = p0->rchild;
            }
            p0->rchild = cur->rchild;
            free(cur);
            return p;
        }
    }
    return cur;
}

int traverse2(BiTree T, KeyType e){
    if (T == nullptr) return 0;
    if (T->data.key == e) return 1;
    return (traverse2(T->lchild, e) || traverse2(T->rchild, e));
}
```

```
void PreOrder(BiTree T){
    if (T == nullptr) return;
    cout << T->data.key << "," << T->data.others << " " << endl;
    PreOrder(T->lchild);
    PreOrder(T->rchild);
}

void InOrder(BiTree T){
    if (T == nullptr) return;
    InOrder(T->lchild);
    cout << T->data.key << "," << T->data.others << " " << endl;
    InOrder(T->rchild);
}

void PostOrder(BiTree T){
    if (T == nullptr) return;
    PostOrder(T->lchild);
    PostOrder(T->rchild);
    cout << T->data.key << "," << T->data.others << " " << endl;
}

void LevelOrder(BiTree T){
    queue<BiTree> q;
    BiTree tmp;
    q.push(T);
    while (!q.empty()){
        tmp = q.front();
        q.pop();
        cout << tmp->data.key << "," << tmp->data.others << " " << endl;
        if (tmp->lchild) q.push(tmp->lchild);
        if (tmp->rchild) q.push(tmp->rchild);
    }
}

status SaveBiTree(BiTree T, char FileName[])
```



```
{
    FILE* fp = fopen(fileName, "w");
    if (fp){
        save_traverse(T, fp);
        fclose(fp);
        return OK;
    }
    fclose(fp);
    return ERROR;
}

void save_traverse(BiTree T, FILE* fp){
    if (!T) {
        fprintf(fp, "%d\n", 0);
        return;
    }
    fprintf(fp, "%d %s\n", T->data.key, T->data.others);
    save_traverse(T->lchild, fp);
    save_traverse(T->rchild, fp);
}

status LoadBiTree(BiTree &T, char FileName[])
{
    FILE* fp = fopen(fileName, "r");
    if (fp){
        T = read(T, fp);
        fclose(fp);
        return OK;
    }
    fclose(fp);
    return ERROR;
}

BiTree read(BiTree T, FILE* fp){
    int key;
    fscanf(fp, "%d", &key);
    if (key == 0){
        return nullptr;
    }
    else{
```

```
T = (BiTree)malloc(sizeof(struct BiTNode));
T->data.key = key;
fscanf(fp, "%s", T->data.others);
T->lchild = read(T->lchild, fp);
T->rchild = read(T->rchild, fp);
return T;
}
}

int MaxPathSum(BiTree T){
    if (T == nullptr) return 0;
    int l = MaxPathSum(T->lchild);
    int r = MaxPathSum(T->rchild);
    return T->data.key + max(l, r);
}

BiTree LCA(BiTree T, KeyType e1, KeyType e2){
    int l1 = FindChild(T->lchild, e1);
    int l2 = FindChild(T->rchild, e1);
    int r1 = FindChild(T->lchild, e2);
    int r2 = FindChild(T->rchild, e2);
    if (l1 && r2 || l2 && r1) return T;
    if (T->data.key == e1 || T->data.key == e2) return T;
    if (l1 && r1) return LCA(T->lchild, e1, e2);
    else return LCA(T->rchild, e1, e2);
}

int FindChild(BiTree T, KeyType e){
    if (T == nullptr) return 0;
    if (T->data.key == e) return 1;
    return FindChild(T->lchild, e) || FindChild(T->rchild, e);
}

void InvertTree(BiTree &T){
    if (T == nullptr) return;
    BiTree tmp = T->lchild;
    T->lchild = T->rchild;
    T->rchild = tmp;
}
```

华中科技大学课程实验报告

```
InvertTree(T->lchild);  
InvertTree(T->rchild);  
}
```

Manager.h

```
#include "def.h"  
#include "SingleTree.h"  
  
class Manager {  
private:  
  
public:  
    TU member[MaxSize];  
    int length;  
    int size;  
    map<string, int> name_index;  
  
    Manager();  
  
    // 获取树名, 返回索引  
    int GetCommand1();  
  
    // 添加一棵树  
    status AddMember();  
  
    // 删除一棵树  
    status DelMember();  
  
    // 显示当前结构  
    void DispStructure();  
  
};
```

Manager.cpp

```
#include "Manager.h"  
  
Manager::Manager() : length(0), size(MaxSize){  
  
}  
  
int Manager::GetCommand1(){
```

```
string name;
cout << "enter the name of the tree:" << endl;
cin >> name;
auto it = this->name_index.find(name);
if (it == this->name_index.end()){
    cout << "can't find the tree!" << endl;
    return -1;
}
return it->second;
}

status Manager::AddMember() {
    if (this->length == size){
        return OVERFLOW;
    }
    string name;
    cout << "enter a name for this tree:" << endl;
    cin >> name;
    this->name_index.insert(pair<string, int>(name, this->length));
    this->member[this->length].name = name;
    TElemType definition[100];
    GetData(definition);

    int state = CreateBiTree(this->member[this->length].T, definition);
    if (state == ERROR) return ERROR;
    else{
        this->length++;
        return OK;
    }
}

status Manager::DelMember() {
    int index;
    index = this->GetCommand1();
    if (index != -1){
        int state = ClearBiTree(this->member[index].T);
        if (state == OK) {
            return OK;
        }
        else return ERROR;
    }
}
```

```
    return ERROR;
}

void Manager::DispStructure() {
    cout << "-----" << endl;
    cout << "|-Manager" << endl;
    for (int i = 0; i < this->length; i++){
        if (this->member[i].T == nullptr) cout << "    |-null" << endl;
        else cout << "    |-" << this->member[i].name << endl;
    }
    cout << "-----" << endl;
}
```

附录 D 基于邻接表图实现的源程序

/* Gragh on LinkTable Sturcture */

main.cpp

```
#include "def.h"
#include "SingleGraph.h"
#include "Manager.h"

int main() {
    int state;
    int op;
    int index;
    int key, key1, key2;
    char filename[30];
    VertexType v;
    Manager M;
    M.MenuDisp();
    cout << "enter your command:" << endl;
    cin >> op;
    while (op){
        switch(op){
            case 1: // 创建新表
                state = M.NewGraph();
                if (state == OK) {
                    cout << "create successfully. The structure is:" <<
                        endl;
                    M.DispStruc();
                }
                else cout << "wrong data set!" << endl;
                break;
            case 2: // 删除一张图
                if ((index = M.GetCommand1()) != -1){
                    M.DelGraph(index);
                    cout << "delete successfully! The structure is:" <<
                        endl;
                    M.DispStruc();
                    break;
                }
            case 3:
                M.DispStruc();
```

```
break;
case 4: // 创建图
if ((index = M.GetCommand2()) != -1){
    VertexType V[MAX_VERTEX_NUM];
    KeyType VR[100][2];
    ReadData(V, VR);
    state = CreateGraph(M.elem[index], V, VR);
    M.IfDataSetError(state);
}
break;
case 5: // 销毁图
if ((index = M.GetCommand1()) != -1){
    DestroyGraph(M.elem[index]);
    cout << "successfully." << endl;
}
break;
case 6: // 定位节点
if ((index = M.GetCommand1()) != -1){
    cout << "enter the key:" << endl;
    cin >> key;
    state = LocateVex(M.elem[index], key);
    if (state == -1){
        cout << "can't find vertex!" << endl;
    }
    else{
        cout << "the index and name is: " << state << ", " \
        << M.elem[index].vertices[state].data.others <<
        endl;
    }
}
break;
case 7: // 修改节点值
if ((index = M.GetCommand1()) != -1){
    M.GetKey(key);
    M.GetVertexValue(v);
    state = PutVex(M.elem[index], key, v);
    M.IfDataSetError(state);
}
break;
case 8: // 获取第一个邻接点
if ((index = M.GetCommand1()) != -1){
```

```
M.GetKey(key);
state = FirstAdjVex(M.elem[index], key);
if (state == -1){
    cout << "without firstadjvex!" << endl;
}
else{
    cout << "the first adjvex is: " <<
        M.elem[index].vertices[state].data.key << ", " \
        << M.elem[index].vertices[state].data.others <<
        endl;
}
}
break;
case 9: // 返回v的邻接点w的下一节点的位序
if ((index = M.GetCommand1()) != -1){
    cout << "node v: ";
    M.GetKey(key1);
    cout << "node w: ";
    M.GetKey(key2);
    state = NextAdjVex(M.elem[index], key1, key2);
    if (state == -1){
        cout << "without firstadjvex!" << endl;
    }
    else{
        cout << "the next adjvex is:" <<
            M.elem[index].vertices[state].data.key << ", " \
            << M.elem[index].vertices[state].data.others <<
            endl;
    }
}
break;
case 10: // 插入顶点
if ((index = M.GetCommand1()) != -1){
    M.GetVertexValue(v);
    state = InsertVex(M.elem[index], v);
    M.IfDataSetError(state);
}
break;
case 11: // 删除节点
if ((index = M.GetCommand1()) != -1){
    M.GetKey(key);
```



```
        state = DeleteVex(M.elem[index], key);
        M.IfDataSetError(state);
    }
    break;
case 12: // 插入弧
    if ((index = M.GetCommand1()) != -1){
        cout << "node v: ";
        M.GetKey(key1);
        cout << "node w: ";
        M.GetKey(key2);
        state = InsertArc(M.elem[index], key1, key2);
        M.IfDataSetError(state);
    }
    break;
case 13: // 删除弧
    if ((index = M.GetCommand1()) != -1){
        cout << "node v: ";
        M.GetKey(key1);
        cout << "node w: ";
        M.GetKey(key2);
        state = DeleteArc(M.elem[index], key1, key2);
        M.IfDataSetError(state);
    }
    break;
case 14: // DFS
    if ((index = M.GetCommand1()) != -1){
        DFSTraverse(M.elem[index], visit);
    }
    break;
case 15: // BFS
    if ((index = M.GetCommand1()) != -1){
        BFSTraverse(M.elem[index], visit);
    }
    break;
case 16: // 保存文件
    if ((index = M.GetCommand1()) != -1){
        cout << "enter the filename:" << endl;
        cin >> filename;
        state = SaveGraph(M.elem[index], filename);
        M.IfIoError(state);
    }
}
```

```
break;
case 17: // 加载文件
if ((index = M.GetCommand2()) != -1){
    cout << "enter the filename:" << endl;
    cin >> filename;
    state = LoadGraph(M.elem[index], filename);
    M.IfIoError(state);
}
break;
case 18: // 距离小于k的顶点集合
if ((index = M.GetCommand1()) != -1){
    M.GetKey(key);
    cout << "enter k:" << endl;
    int k;
    cin >> k;
    vector<int> LessThanK =
    VerticesSetLessThanK(M.elem[index], key, k);
    cout << "vertices are:" << endl;
    for (int i : LessThanK){
        cout << M.elem[index].vertices[i].data.key << "," \
        << M.elem[index].vertices[i].data.others << " ";
    }
}
break;
case 19: // 最短距离
if ((index = M.GetCommand1()) != -1){
    cout << "enter v:";
    M.GetKey(key1);
    cout << "enter w:";
    M.GetKey(key2);
    cout << "dist is: " <<
    ShortestPathLength(M.elem[index], key1, key2);
}
break;
case 20: // 连通分量数
if ((index = M.GetCommand1()) != -1){
    cout << "the num is:" <<
    ConnectedComponentsNums(M.elem[index]);
}
break;
default:
```

```
        cout << "wrong command!" << endl;
        break;

    }

    cout << endl;
    cout << endl;
    M.MenuDisp();
    cout << "enter your command:" << endl;
    cin >> op;

}

cout << "bye." << endl;

return 0;

}
```

def.h

```
#pragma once

#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <string>
#include <map>
#include <algorithm>
#include <vector>
#include <queue>
using namespace std;

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#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() : nextarc(nullptr){}
} 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() : firstarc(nullptr){}
} VNode, AdjList[MAX_VERTEX_NUM];

typedef struct LGraph{ //邻接表的类型定义
    string name;
    AdjList vertices; //头结点数组
    int vexnum, arcnum; //顶点数、弧数
    GraphKind kind; //图的类型
    LGraph() : vexnum(0), arcnum(0), kind(UDG), name("unnamed"){ }
} ALGraph;
```

SingleGraph.h

```
#pragma once
#include "def.h"

// 读取数据
void ReadData(VertexType V[], KeyType VR[][2]);

// 创建图
status CreateGraph(ALGraph &G, VertexType V[], KeyType VR[][2]);
int find_index(const ALGraph &G, int key); //
查找key对应顶点的下标-----通用函数
status AddVertex(ALGraph &G, int key1, int key2); // 添加边

// 销毁图
status DestroyGraph(ALGraph &G);

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

```
//根据u在图G中查找顶点, 查找成功将该顶点值修改成value, 返回OK;
//如果查找失败或关键字不唯一, 返回ERROR
status PutVex(ALGraph &G,KeyType u,VertexType value);

//根据u在图G中查找顶点, 查找成功返回顶点u的第一邻接顶点位序, 否则返回-1
int FirstAdjVex(const ALGraph &G,KeyType u);

//根据u在图G中查找顶点, 查找成功返回顶点v的邻接顶点相对于w的下一邻接顶点的位序, 查找失败返回-1
int NextAdjVex(const ALGraph &G,KeyType v,KeyType w);

//在图G中插入顶点v, 成功返回OK, 否则返回ERROR
status InsertVex(ALGraph &G,VertexType v);

//在图G中删除关键字v对应的顶点以及相关的弧, 成功返回OK, 否则返回ERROR
status DeleteVex(ALGraph &G,KeyType v);
int GetIndex(const ALGraph &G, KeyType v); //
和find_index用法相同-----通用

//在图G中增加弧<v,w>, 成功返回OK, 否则返回ERROR
status InsertArc(ALGraph &G,KeyType v,KeyType w);
int IfConflict(const ALGraph &G, KeyType v, KeyType w); // 防止同一条弧
被插两次

//在图G中删除弧<v,w>, 成功返回OK, 否则返回ERROR
status DeleteArc(ALGraph &G,KeyType v,KeyType w);

// dfs
status DFSTraverse(ALGraph &G,void (*visit)(VertexType));
void dfs(const ALGraph &G, int index, void (*visit)(VertexType), int
ifvisit[]);

//bfs
status BFSTraverse(ALGraph &G,void (*visit)(VertexType));
void visit(VertexType v); //-----
通用

//保存文件
status SaveGraph(ALGraph G, char FileName[]);

//加载文件
```

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

//与v距离小于k的顶点
vector<int> VerticesSetLessThanK(const ALGraph &G, KeyType v, int k);
vector<int> GetDist(const ALGraph &G, KeyType v); // 辅助函数: 返回各顶
点和v的距离-----通用

// v和w之间的最短距离
int ShortestPathLength(const ALGraph &G, KeyType v, KeyType w);

// 返回连通分量个数
int ConnectedComponentsNums(const ALGraph &G);
void dfs0(const ALGraph &G, int index, vector<int> &ifvisit); // 辅助函
数
```

SingleGraph.cpp

```
#include "def.h"
#include "SingleGraph.h"

void ReadData(VertexType V[], KeyType VR[][2]){
    // 获取V数组元素
    int key;
    char others[30];
    int i = 0;
    cout << "enter the key-value pair end with \"-1 nil\" << endl;
    cin >> key >> others;
    while (key != -1){
        V[i].key = key;
        strcpy(V[i].others, others);
        i++;
        cin >> key >> others;
    }
    V[i].key = key;
    strcpy(V[i].others, others);
    cout << "vertex read successfully. Enter the edge end with \"-1
        -1\" << endl;
    // 获取VR数组元素
    int key1, key2;
    i = 0;
    cin >> key1 >> key2;
    while (key1 != -1){
```

```
        VR[i][0] = key1;
        VR[i][1] = key2;
        i++;
        cin >> key1 >> key2;
    }
    VR[i][0] = key1;
    VR[i][1] = key2;
    cout << "read successfully!" << endl;
}

status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2])
{
    //判断key的唯一性
    int hash[100] = {0};
    int i = 0;
    while (V[i].key != -1){
        if (hash[V[i].key] == 1){
            return ERROR;
        }
        hash[V[i].key] = 1;
        i++;
    }
    //图的操作
    // 添加顶点
    i = 0;
    G.vexnum = 0;
    G.arcnum = 0;
    G.kind = UDG;
    while (i < 20 && V[i].key != -1){
        G.vertices[i].data = V[i];
        G.vertices[i].firstarc = nullptr;
        G.vexnum++;
        i++;
    }
    if (i == 20 && V[i].key != -1) return ERROR; // 加入的顶点数不能溢出
    // 添加边
    i = 0;
    while (VR[i][0] != -1){
        int state = AddVertex(G, VR[i][0], VR[i][1]);
        if (state == ERROR) return ERROR;
    }
}
```

```
        i++;
    }
    return OK;
}

status AddVertex(ALGraph &G, int key1, int key2) {
    int index1 = find_index(G, key1);
    if (index1 == -1) return ERROR;
    int index2 = find_index(G, key2);
    if (index2 == -1) return ERROR;

    VNode &Head1 = G.vertices[index1];
    ArcNode *NewNode1 = (ArcNode *) malloc(sizeof(struct ArcNode));
    NewNode1->adjvex = index2;
    NewNode1->nextarc = Head1.firstarc;
    Head1.firstarc = NewNode1;

    VNode &Head2 = G.vertices[index2];
    ArcNode *NewNode2 = (ArcNode *) malloc(sizeof(struct ArcNode));
    NewNode2->adjvex = index1;
    NewNode2->nextarc = Head2.firstarc;
    Head2.firstarc = NewNode2;

    G.arcnum++;
    return OK;
}

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

status DestroyGraph(ALGraph &G){
    for (int i = 0; i < G.vexnum; i++){
        ArcNode* p = G.vertices[i].firstarc;
        ArcNode* pre;
        while (p){
            pre = p;
```



```
        p = p->nextarc;
        free(pre);
    }
    G.vertices[i].firstarc = nullptr;
}
G.vexnum = 0;
G.arcnum = 0;
return OK;
}

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

status PutVex(ALGraph &G,KeyType u,VertexType value)
{
    int hash[100] = {0};
    int i;
    for (i = 0; i < G.vexnum; i++){
        hash[G.vertices[i].data.key] = 1;
    }
    if (hash[u] == 0 || u != value.key && hash[value.key] == 1) return
    ERROR;
    for (i = 0; i < G.vexnum; i++){
        if (u == G.vertices[i].data.key){
            G.vertices[i].data = value;
        }
    }
    return OK;
}

int FirstAdjVex(const ALGraph &G,KeyType u)
```

```
{
    int i;
    for (i = 0; i < G.vexnum; i++){
        if (G.vertices[i].data.key == u) break;
    }
    if (i == G.vexnum) return -1;
    if (G.vertices[i].firstarc->adjvex) return
    G.vertices[i].firstarc->adjvex;
    else return -1;
}

int NextAdjVex(const ALGraph &G,KeyType v,KeyType w)
{
    int i;
    for (i = 0; i < G.vexnum; i++){
        if (G.vertices[i].data.key == v) break;
    }
    if (i == G.vexnum) return -1;
    ArcNode* p = G.vertices[i].firstarc;
    while (p){
        if (G.vertices[p->adjvex].data.key == w){
            if (p->nextarc) return p->nextarc->adjvex;
            else return -1;
        }
        p = p->nextarc;
    }
    return -1;
}

status InsertVex(ALGraph &G,VertexType v)
{
    int hash[100] = {0};
    int i;
    for (i = 0; i < G.vexnum; i++){
        hash[G.vertices[i].data.key] = 1;
    }
    if (hash[v.key] == 1) return ERROR;
    if (G.vexnum == 20) return ERROR;
    G.vertices[G.vexnum++].data = v;
}
```

```
G.vertices[G.vexnum].firstarc = nullptr;
return OK;
}

status DeleteVex(ALGraph &G,KeyType v)
{
    int index = GetIndex(G, v);
    if (index == -1) return ERROR;
    if (G.vexnum == 1) return ERROR;
    // 删除节点
    ArcNode* p = G.vertices[index].firstarc;
    // 防止p本来就是null
    if (p != nullptr){
        ArcNode* next = p->nextarc;
        while (next){
            free(p);
            p = next;
            next = next->nextarc;
        }
        free(p);
        G.vertices[index].firstarc = nullptr;
    }
    for (int i = index; i < G.vexnum-1; i++){
        G.vertices[i] = G.vertices[i+1];
    }
    G.vexnum--;
    // 删除边
    for (int i = 0; i < G.vexnum; i++){
        p = G.vertices[i].firstarc;
        // 这里分类稍微麻烦,要看看第一个节点是否就是被删的节点
        if (p && p->adjvex == index) {
            G.vertices[i].firstarc = p->nextarc;
            G.arcnum--;
            free(p);
        }
        else{
            while (p && p->nextarc){
                if (p->nextarc->adjvex == index){
                    ArcNode* bin = p->nextarc;
                    p->nextarc = p->nextarc->nextarc;
                }
            }
        }
    }
}
```

```
        free(bin);
        G.arcnum--;
        break;
    }
    p = p->nextarc;
}
}
}
// 调整删除节点后adjvex的值
for (int i = 0; i < G.vexnum; i++){
    p = G.vertices[i].firstarc;
    while(p){
        if (p->adjvex > index) p->adjvex--;
        p = p->nextarc;
    }
}
return OK;
}

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

status InsertArc(ALGraph &G, KeyType v, KeyType w)
{
    int index1 = GetIndex(G, v);
    int index2 = GetIndex(G, w);
    if (index1 == -1 || index2 == -1) return ERROR;
    if (IfConflict(G, v, w)) return ERROR;

    ArcNode* p = G.vertices[index1].firstarc;
    ArcNode* newnode = (ArcNode*)malloc(sizeof(struct ArcNode));
    newnode->adjvex = index2;
    newnode->nextarc = p;
    G.vertices[index1].firstarc = newnode;
    G.arcnum++;
}
```

```
ArcNode* p0 = G.vertices[index2].firstarc;
ArcNode* newnode1 = (ArcNode*)malloc(sizeof(struct ArcNode));
newnode1->adjvex = index1;
newnode1->nextarc = p0;
G.vertices[index2].firstarc = newnode1;

return OK;
}

int IfConflict(const ALGraph &G, KeyType v, KeyType w){
    for (int i = 0; i < G.vexnum; i++){
        if (G.vertices[i].data.key == v){
            ArcNode* p = G.vertices[i].firstarc;
            int target = GetIndex(G, w);
            while (p){
                if (p->adjvex == target) return 1;
                p = p->nextarc;
            }
        }
    }
    return 0;
}

status DeleteArc(ALGraph &G, KeyType v, KeyType w)
{
    int index1 = GetIndex(G, v);
    int index2 = GetIndex(G, w);
    if (index1 == -1 || index2 == -1) return ERROR;
    int flag = 0;
    // 删弧
    ArcNode* p1 = G.vertices[index1].firstarc;
    if (p1 && p1->adjvex == index2) {
        flag = 1;
        G.vertices[index1].firstarc = p1->nextarc;
        G.arcnum--; // 弧计数在这里减去就行, 下面不用
        free(p1);
    }
    else{
        while (p1->nextarc){
            if (p1->nextarc->adjvex == index2){
```

```
        flag = 1; // 同样的, 这里找到弧设置一下就行, 下面不用
        ArcNode* bin = p1->nextarc;
        p1->nextarc = p1->nextarc->nextarc;
        free(bin);
        G.arcnum--;
        break;
    }
    p1 = p1->nextarc;
}
}
// 对称情况
ArcNode* p2 = G.vertices[index2].firstarc;
if (p2 && p2->adjvex == index1) {
    G.vertices[index2].firstarc = p2->nextarc;
    //G.arcnum--;
    free(p2);
}
else{
    while (p2->nextarc){
        if (p2->nextarc->adjvex == index1){
            ArcNode* bin = p2->nextarc;
            p2->nextarc = p2->nextarc->nextarc;
            free(bin);
            //G.arcnum--;
            break;
        }
        p2 = p2->nextarc;
    }
}

if (flag == 0) return ERROR;
return OK;
}

status DFSTraverse(ALGraph &G, void (*visit)(VertexType))
{
    int ifvisit[20] = {0};
    for (int i = 0; i < G.vexnum; i++){
        dfs(G, i, visit, ifvisit);
    }
}
```

```
    return OK;
}

void dfs(const ALGraph &G, int index, void (*visit)(VertexType), int
ifvisit[]){
    if (ifvisit[index] == 1) return;
    visit(G.vertices[index].data);
    ifvisit[index] = 1;
    ArcNode* p = G.vertices[index].firstarc;
    while (p){
        dfs(G, p->adjvex, visit, ifvisit);
        p = p->nextarc;
    }
}

status BFSTraverse(ALGraph &G, void (*visit)(VertexType))
{
    int q[100];
    int ifvisit[100];
    for (int i = 0; i < G.vexnum; i++){
        ifvisit[i] = 0;
    }
    int head = 0, tail = 0;
    for (int i = 0; i < G.vexnum; i++){
        if (ifvisit[i] == 0){
            q[tail++] = i;
            ifvisit[i] = 1;
        }
        while (head != tail){
            VNode cur = G.vertices[q[head++]];
            visit(cur.data);
            ArcNode* p = cur.firstarc;
            while (p){
                if (ifvisit[p->adjvex] == 0){
                    q[tail++] = p->adjvex;
                    ifvisit[p->adjvex] = 1;
                }
                p = p->nextarc;
            }
        }
    }
}
```

```
    }
    return OK;
}

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

status SaveGraph(ALGraph G, char FileName[])
{
    FILE* fp = fopen(FileName, "w");
    if (!fp) return ERROR;
    for (int i = 0; i < G.vexnum; i++){
        fprintf(fp, "%d %s\n", G.vertices[i].data.key,
            %%G.vertices[i].data.others);
    }
    char last[30] = "nil";
    fprintf(fp, "%d %s\n", -1, last);
    for (int i = 0; i < G.vexnum; i++){
        ArcNode* p = G.vertices[i].firstarc;
        while (p){
            fprintf(fp, "%d %d\n", G.vertices[i].data.key,
                %%G.vertices[p->adjvex].data.key);
            p = p->nextarc;
        }
    }
    fprintf(fp, "%d %d\n", -1, -1);
    fclose(fp);
    return OK;
}

status LoadGraph(ALGraph &G, char FileName[])
{
    FILE* fp = fopen(FileName, "r");
    if (!fp) return ERROR;
    int key;
    char others[30];
    fscanf(fp, "%d %s", &key, others);
```



```
while (key != -1){
    G.vertices[G.vexnum].data.key = key;
    strcpy(G.vertices[G.vexnum].data.others, others);
    G.vexnum++;
    fscanf(fp, "%d %s", &key, others);
}
KeyType key1, key2;
fscanf(fp, "%d %d", &key1, &key2);
while (key1 != -1){
    InsertArc(G, key1, key2);
    fscanf(fp, "%d %d", &key1, &key2);
}
fclose(fp);
return OK;
}

vector<int> VerticesSetLessThanK(const ALGraph &G, KeyType v, int k){
    vector<int> dist = GetDist(G, v);
    // 统计距离小于k的节点
    vector<int> LessThanK;
    for (int i = 0; i < G.vexnum; i++){
        if (dist[i] < k){
            LessThanK.push_back(i);
        }
    }
    return LessThanK;
}

vector<int> GetDist(const ALGraph &G, KeyType v){
    vector<int> ifvisited(G.vexnum, 0);
    vector<int> dist(G.vexnum, INT16_MAX);
    queue<int> q;
    int cur_size;
    int index;
    int cur_dist = 0;
    //bfs过程中记录距离
    int index0 = GetIndex(G, v);
    dist[index0] = 0;
    ifvisited[index0] = 0;
    q.push(index0);
```

```
cur_size = q.size();
while (!q.empty()){
    for (int i = 0; i < cur_size; i++){
        index = q.front();
        q.pop();
        dist[index] = cur_dist;
        // 加入与该点邻接的点
        ArcNode *p = G.vertices[index].firstarc;
        while (p){
            if (ifvisited[p->adjvex] == 0){
                q.push(p->adjvex);
                ifvisited[p->adjvex] = 1;
            }
            p = p->nextarc;
        }
    }
    cur_dist++;
    cur_size = q.size();
}
return dist;
}

int ShortestPathLength(const ALGraph &G, KeyType v, KeyType w){
    vector<int> dist = GetDist(G, v);
    return dist[GetIndex(G, w)];
}

int ConnectedComponentsNums(const ALGraph &G){
    vector<int> ifvisited(G.vexnum, 0);
    int num = 0;
    for (int i = 0; i < G.vexnum; i++){
        if (ifvisited[i] == 0){
            num++;
            dfs0(G, i, ifvisited);
        }
    }
    return num;
}
```

```
void dfs0(const ALGraph &G, int index, vector<int> &ifvisit){
    if (ifvisit[index] == 1) return;
    ifvisit[index] = 1;
    ArcNode* p = G.vertices[index].firstarc;
    while (p){
        dfs0(G, p->adjvex, ifvisit);
        p = p->nextarc;
    }
}
```

Manager.h

```
#include "def.h"
#include "SingleGraph.h"
#pragma once

class Manager {
private:

public:
    ALGraph elem[10];
    int length;
    int initial_size;
    map<string, int> name_index; // 图名索引

    Manager();

    //展示菜单
    void MenuDisp();

    // 交互判空函数，为空报错
    int GetCommand1();

    // 判不空交互函数，不为空报错
    int GetCommand2();

    // 数据集错误检查
    void IfDataSetError(int state);

    // 判断io错误
    void IfIoError(int state);
```

```
// 获取key
void GetKey(int &key);

// 获取一个节点值
void GetVertexValue(VertexType &v);

// // 定位节点位序检查, 找不到报错
// void IfFind(int state);

//创建一张图
status NewGraph();

//删除一张图
status DelGraph(int index);

//展示树形结构
void DispStruc();

};
```

Manager.cpp

```
#include "Manager.h"

Manager::Manager() : length(0), initial_size(10){}

void Manager::MenuDisp() {
    cout << "                Menu" << endl;
    cout << "-----" << endl;
    cout << "  1. NewGraph                2. DelGraph" << endl;
    cout << "  3. DispStruc                " << endl;
    cout << "  4. CreateGraph            5. DestroyGraph" << endl;
    cout << "  6. LocateVex              7. PutVex" << endl;
    cout << "  8. FirstAdjVex            9. NextAdjVex" << endl;
    cout << " 10. InsertVex              11. DeleteVex" << endl;
    cout << " 12. InsertArc              13. DeleteArc" << endl;
    cout << " 14. DFS                    15. BFS" << endl;
    cout << " 16. Save                   17. Load" << endl;
    cout << " 18. VerticesSetLessThanK   19. ShortestPathLength" <<
    endl;
    cout << " 20. ConnectedComponentsNums  0. quit" << endl;
    cout << "-----" << endl;
}
```

```
int Manager::GetCommand1(){
    string name;
    cout << "enter the name of the graph to be operated:" << endl;
    cin >> name;
    map<string, int>::iterator it = this->name_index.find(name);
    if (it == this->name_index.end()) {
        cout << "can't find the graph!" << endl;
        return -1;
    }
    else if (this->elem[it->second].vexnum == 0){
        cout << "empty graph!" << endl;
        return -1;
    }
    else return it->second;
}

int Manager::GetCommand2(){
    string name;
    cout << "enter the name:" << endl;
    cin >> name;
    map<string, int>::iterator it = this->name_index.find(name);
    if (it == this->name_index.end()) {
        cout << "can't find the graph!" << endl;
        return -1;
    }
    else if (this->elem[it->second].vexnum != 0){
        cout << "existed graph!" << endl;
        return -1;
    }
    else return it->second;
}

void Manager::IfDataSetError(int state) {
    if (state == ERROR){
        cout << "data set error!" << endl;
    }
    else{
        cout << "successfully." << endl;
    }
}
```

```
}

void Manager::IfIoError(int state) {
    if (state == ERROR){
        cout << "IO ERROR!" << endl;
    }
    else{
        cout << "successfully." << endl;
    }
}

void Manager::GetKey(int &key) {
    cout << "enter the key:" << endl;
    cin >> key;
}

void Manager::GetVertexValue(VertexType &v) {
    cout << "enter the key-value pair:" << endl;
    cin >> v.key >> v.others;
}

//void Manager::IfFind(int state) {
//    if (state == -1){
//        cout << "can't find vertex!" << endl;
//    }
//    else{
//        cout << ""
//    }
//}

status Manager::NewGraph() {
    VertexType V[MAX_VERTEX_NUM];
    KeyType VR[100][2];
    cout << "enter a name for this graph:" << endl;
    cin >> this->elem[this->length].name;
    // 读取数据
    ReadData(V, VR);
    // 创建图
    int state = CreateGraph(this->elem[this->length], V, VR);
    if (state == OK) {
        this->name_index.insert(pair<string
        ,int>(this->elem[this->length].name, this->length));
    }
}
```

```
        this->length++;
    }
    return state;
}

status Manager::DelGraph(int index) {
    return DestroyGraph(this->elem[index]);
}

void Manager::DispStruc() {
    cout << "|-Manager" << endl;
    for (int i = 0; i < this->length; i++){
        if (this->elem[i].vexnum == 0){
            cout << "    |-null" << endl;
        }
        else {
            cout << "    |-" << this->elem[i].name << endl;
        }
    }
}
```