

**课 程 设 计 报 告**

**题目： 基于AVL树表示**

**的集合ADT实现与应用**

**课程名称：**

**专业班级：**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期： 2018年3月21日**

**计算机科学与技术学院**

# 任 务 书

* **设计目的**

平衡二叉树(AVL)作为一种重要的查找表结构，能有效地支持数据的并行处理。本设计使学生牢固掌握AVL树及其实现方法，并应用该结构实现集合抽象数据类型，提升学生对数据结构与数据抽象的认识，提高学生的综合实践与应用能力。

* **设计内容**

本设计分为三个层次：（1）以二叉链表为存储结构，设计与实现AVL树-动态查找表及其6种基本运算；（2）以AVL树表示集合，实现集合抽象数据类型及其10种基本运算；（3）以集合表示个人微博或社交网络中好友集、粉丝集、关注人集，实现共同关注、共同喜好、二度好友等查询功能。

* **主要数据对象：**好友集、粉丝集、关注人集等。
* **主要数据关系：**

**（1）**抽象层面AVL可以表示数据元素之间层次关系或一对多关系。

**（2）**实际应用层面，所讨论的人物关系为集合内元素间的关系。立足于集合建立数据的逻辑模型。

* **主要运算与功能要求：**

**（1）**交互式操作界面(并非一定指图形式界面)；

**（2）**AVL树的6种基本运算：InitAVL、DestroyAVL、SearchAVL、InsertAVL、DeleteAVL、TraverseAVL；

**（3）**基于AVL表示及调用其6种基本运算实现集合ADT的基本运算：初始化set\_init，销毁set\_destroy，插入set\_insert，删除set\_remove，交set\_intersection，并set\_union，差set\_diffrence，成员个数set\_size，判断元素是否为集合成员的查找set\_member，判断是否为子集set\_subset，判断集合是否相等set\_equal；

**（4）**基于集合ADT实现应用层功能：好友集、粉丝集、关注人集等的初始化与对成员的增删改查，实现共同关注、共同喜好、二度好友等查询；

**（5）**主要数据对象的数据文件组织与存储。

* **设计提示**

（1）参考有关文献，实现AVL树的删除操作，维护其动态平衡，这可能是设计中较为复杂的算法；要求提供关键算法的时间与空间复杂度分析。

（2）要求从互联网上获取测试数据集或随机生成测试数据集，数据集的大小具有一定规模；数据与结果以文件保存。

* **参考文献**

[1] 严蔚敏, 吴伟民. 数据结构（C语言版）. 北京: 清华大学出版社,1997

[2] 严蔚敏, 吴伟民, 米宁. 数据结构题集（C语言版）. 北京: 清华大学出版社,1999

[3] Lin Chen. O(1) space complexity deletion for AVL trees, Information Processing Letters, 1986, 22(3)：147-149

[4] S.H. Zweben, M. A. McDonald.**An optimal method for deletion in one-sided height-balanced trees,** Communications of the ACM, 1978, 21(6): 441-445

[5] Guy Blelloch. Principles of Parallel Algorithms and Programming, CMU, 2014

# 目 录

[任务书 I](#_Toc509086181)

[1 引言 1](#_Toc509086182)

[1.1 课题背景与意义 1](#_Toc509086183)

[1.2 国内外研究现状 1](#_Toc509086184)

[1.3 课程设计的主要研究工作 1](#_Toc509086185)

[2 系统需求分析与总体设计 2](#_Toc509086186)

[2.1 系统需求分析 2](#_Toc509086187)

[2.2 系统总体设计 2](#_Toc509086188)

[3 系统详细设计 3](#_Toc509086189)

[3.1 有关数据结构的定义 3](#_Toc509086190)

[3.2 主要算法设计 4](#_Toc509086191)

[4 系统实现与测试 14](#_Toc509086192)

[4.1 系统实现 14](#_Toc509086193)

[4.1.1 实验环境 14](#_Toc509086194)

[4.1.2 演示系统操作 14](#_Toc509086195)

[4.1.3 定义的各种数据类型 15](#_Toc509086196)

[4.2 系统测试 18](#_Toc509086197)

[5 总结与展望 23](#_Toc509086198)

[5.1 全文总结 38](#_Toc509086199)

[5.2 工作展望 38](#_Toc509086200)

[6 体会 39](#_Toc509086201)

[参考文献 40](#_Toc509086202)

[附录 41](#_Toc509086203)

# 1 引言

## 1.1 课题背景与意义

AVL树即平衡二叉树，其左子树和右子树都是平衡二叉树，且左子树和右子树的深度之差的绝对值不超过1。AVL树是一种重要的数据结构，支持动态的查找和删除，且查询时间复杂度可以保证是O（log n）。如windows对进程地址空间的管理就用到了AVL树。

## 1.2 国内外研究现状

AVL树是最先发明的自平衡二叉查找树，随后延伸出许多相关数据结构，如B-树、B+树、键树等等。

## 1.3 课程设计的主要研究工作

实现对AVL树的初始化、销毁、插入、删除、查找等基础功能，并在其基础上实现对基于AVL树的集合的操作，并最终以集合表示社交网络中好友集、粉丝集、关注人集，实现应用层上的一些功能。

2 系统需求分析与总体设计

## 2.1 系统需求分析

以基于链表实现的AVL树作为存储结构，能随机生成一定量数据并实现数据的保存和读取，以此模拟社交网络中的人际关系，并实现应用层对集合的插入、删除、修改、遍历等操作，以及对二度好友、共同爱好等查询功能。

## 2.2 系统总体设计

图2-1 系统总体设计图

3 系统详细设计

## 3.1 有关数据结构的定义

系统所要处理的数据类型如表3-1所示：

表3-1 数据项及数据类型

|  |  |  |  |
| --- | --- | --- | --- |
| 数据项 | 数据类型 | 数据项 | 数据类型 |
| 存储结构Forest | Forest | 存储结构BSTNode | BSTNode |
| 存储结构TElemType | TElemType | 树名name | char[20] |
| 存储数据data | TElemType | 平衡因子BF | Int |
| 左孩子lchild | BSTNode | 右孩子rchild | BSTNode |
| 身份证号id | char[20] | 姓名name | char[20] |
| 粉丝集 | BSTNode | 好友集 | BSTNode |
| 关注人集 | BSTNode | 兴趣爱好集 | BSTNode |

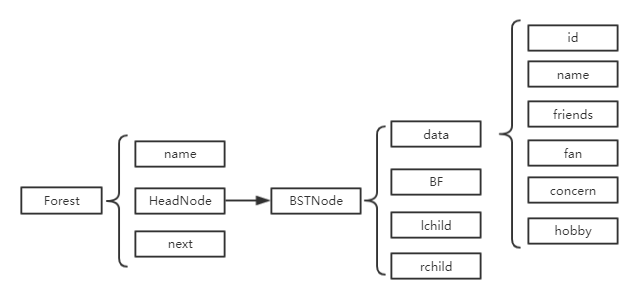
以上数据项间的关系如下图3-1所示：

图3-1 数据间的关系

多树结构的具体定义方式如下：

typedef struct Forest {

char name[20];

struct BSTNode \*HeadNode; //二叉树头结点

struct Forest \*next;

}Forest;

平衡二叉树的结构的具体定义方式如下：

typedef struct BSTNode {

struct TElemType data; //数据信息

int BF; //平衡因子

struct BSTNode \*lchild, \*rchild; //左右孩子指针

}BSTNode;

数据项的具体定义方式如下：

typedef struct TElemType {

char id[20]; //身份证号

char name[20]; //姓名

struct BSTNode \*friends; //好友集

struct BSTNode \*fan; //粉丝集

struct BSTNode \*concern; //关注人集

struct BSTNode \*hobby; //兴趣爱好集

}TElemType;

## 3.2 主要算法设计

1. InsertAVL(T1, data)：

**算法思想：** 算法流程图如3-2所示:

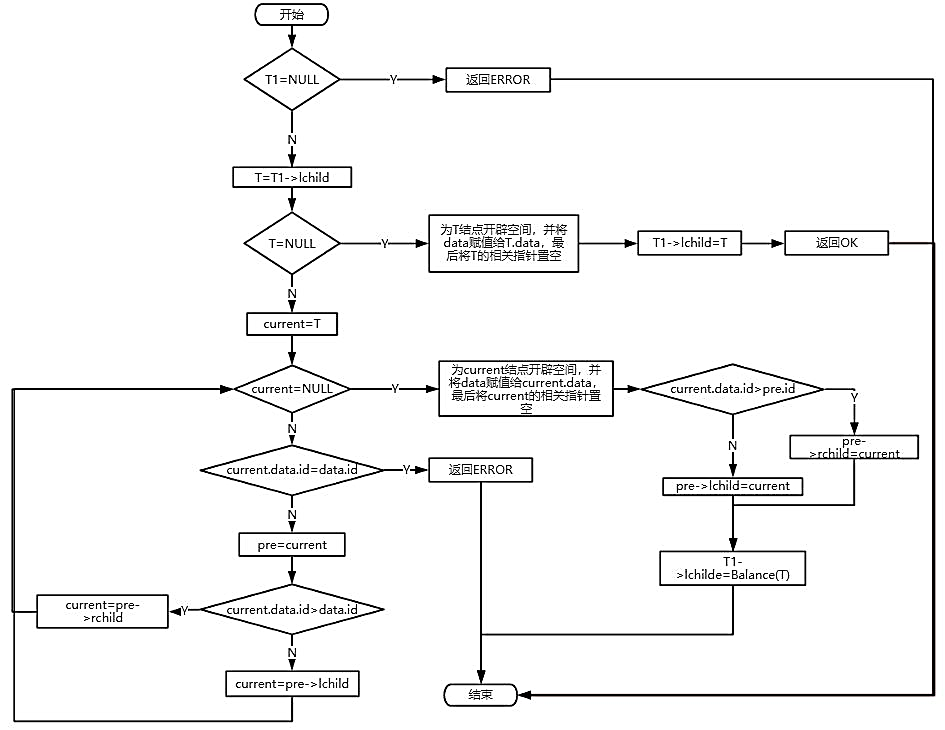


图3-2 插入函数的算法流程图

**时间复杂度分析：** 在查找到当前节点时使用了循环，而在最后调整至平衡时调用了Balance函数，而Balance函数使用递归算法，所以最终插入函数的时间复杂度为0(n)。

**空间复杂度分析：** 插入结点时，需开辟新的空间建立新的结点，所以插入函数的空间复杂度为0(1)。

2. DeleteAVL (T1, id)：

**算法思想：** T1为AVL树的头结点，首先判断当前AVL树是否初始化。T=T1->lchild，即二叉树的根结点。调用SearchAVL函数找到AVL树中id为参数id的结点，并赋值给node。若node和T不同时非空，则返回ERROR。

（1）若node的左右孩子结点都为空，则判断node结点是否为根结点，是则直接将根结点置空，否则找到node结点的双亲结点，将其对应孩子指针置空。

（2）若node的左孩子为空而右孩子不为空，则判断node结点是否为根结点，是则直接将根结点更改为原根结点的右孩子结点，否则找到node结点的双亲结点，将其对应孩子指针置空。

（3）若node的右孩子为空而左孩子不为空，则判断node结点是否为根结点，是则直接将根结点更改为原根结点的左孩子结点，否则找到node结点的双亲结点，将其对应孩子指针置空。

（4）若node的左右孩子结点都非空，则以node的左孩子的最右结点作为新的根结点。

最后同插入函数一样，令T1->lchild=Balance(T)，返回OK。

**时间复杂度分析：** 在四种情况中查找到当前节点时都使用了循环，并在三种情况中需要调用Parent函数以返回参数的双亲结点，Parent函数使用递归算法，同时在最后调整至平衡时调用了Balance函数，Balance函数使用递归算法，所以最终插入函数的时间复杂度为0(n)。

**空间复杂度分析：** 删除结点时需要释放一个结点的空间，所以删除函数的空间复杂度为0(1)。

3. Balance (T)：

**算法思想：** 算法运用递归的算法思想，流程图如图3-3所示:

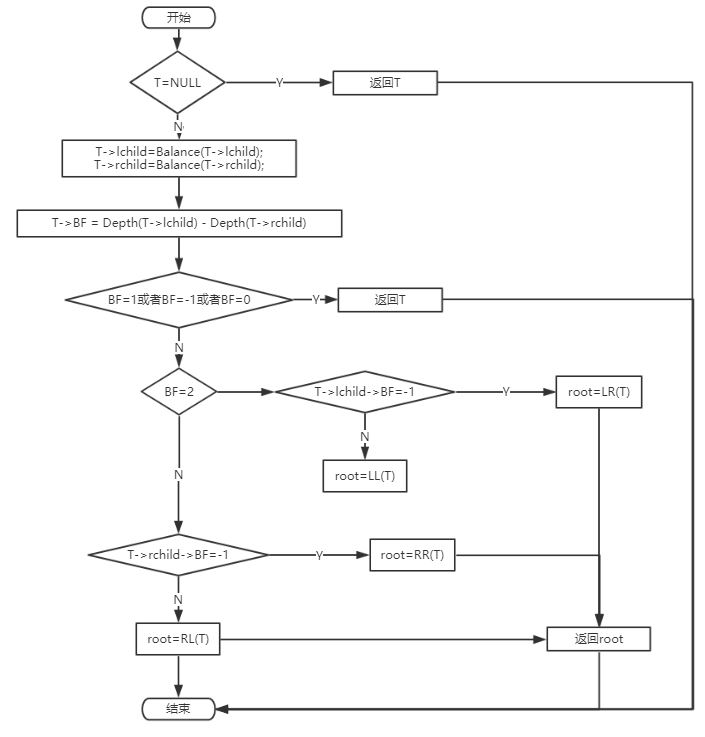


图3-3 调整函数的算法流程图

**时间复杂度分析：** 运用递归的算法思想，并每次要调用两次Depth函数，Depth函数运用递归算法，之后还调用了LL、LR、RR、RL函数，所以调整函数的时间复杂度为0(n2)。

**空间复杂度分析：** 在四种旋转函数中声明临时变量，函数的空间复杂度为0(1)。

4. SearchAVL (T,name,choice)：

**算法思想：** 算法运用递归的算法思想，根据choice的值选择根据名称进行查找（可能有多个结果）或身份证号进行查找（结果唯一），流程图如3-4所示:

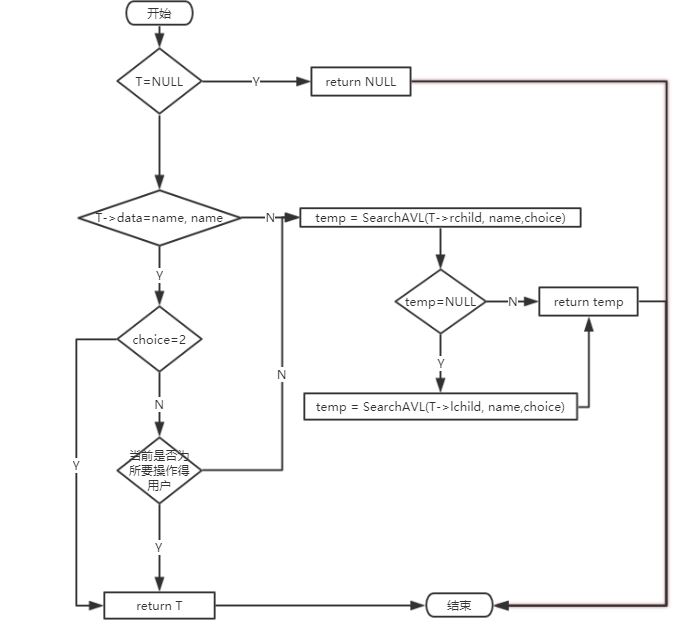


图3-4 查找函数的算法流程图

**时间复杂度分析：** 运用递归的算法思想，查找函数的时间复杂度为0(n)。

**空间复杂度分析：**声明临时变量temp，函数的空间复杂度为0(1)。

5. set\_diffrence（T1,T2,T3）：

**算法思想：** 运用递归遍历T3，将存在于T3而不存在于T2的节点利用InsertAVL函数插入到T1中。两次调用该函数，先将存在于A集合而不存在于B集合的节点插入到T，再将存在于B集合而不存在于A集合的节点插入T，以此求得交集T，且T是一个平衡二叉树。

**时间复杂度分析：**函数的时间复杂度为0(n)。

**空间复杂度分析：**函数的空间复杂度为0(1)。

6. set\_union（T1,T2）：

**算法思想：** 运用递归遍历T2，将存在于T2而不存在于T1的节点利用InsertAVL函数插入到T1中。首先调用copy函数，同样运用递归函数复制一个A集合到T中，再调用该函数，将存在于B而不存在于T的节点插入T，以此求得并集T，且T时一个平衡二叉树。

**时间复杂度分析：**函数的时间复杂度为0(n)。

**空间复杂度分析：**函数的空间复杂度为0(1)。

7. intersection（T1,T2,T3）：

**算法思想：** 运用递归遍历T3，将存在于T3且存在于T2的节点利用InsertAVL函数插入到T1中，以此求得交集T，且T时一个平衡二叉树。

**时间复杂度分析：**函数的时间复杂度为0(n)。

**空间复杂度分析：**函数的空间复杂度为0(1)。

8. second\_friends（T1,T）：

**算法思想：** 算法运用递归的算法思想，通过调用求交集函数，并不生成一颗平衡二叉树而是直接输出是二度好友的用户的信息，流程图如3-5所示:

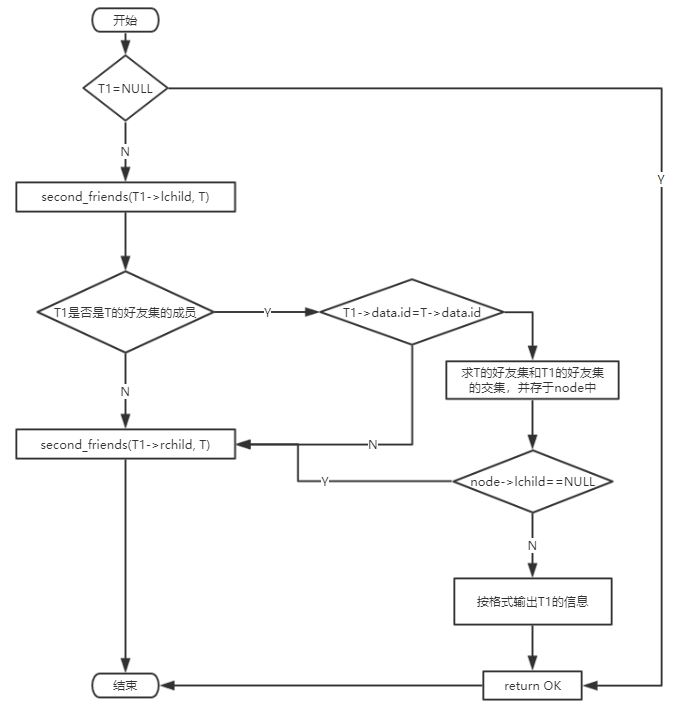


图3-5 查找二度好友的算法流程图

**时间复杂度分析：** 运用递归的算法思想，查找函数的时间复杂度为0(n2)。

**空间复杂度分析：**声明临时变量temp，函数的空间复杂度为0(1)。

9. random(T,gross)：

**算法思想：** 调用已有的name文件库，读入到二维数组中，按照随机生成的数字，依次插入数据中对应名称的节点到成员的平衡二叉树中，并同样的方式处理hobby文件库。之后调用random1函数，运用递归的算法思想，对每个节点依次对好友集、粉丝机、关注人集、兴趣爱好集进行随机生成。前三者在随机生成时以在一定范围内随机生成的数字为身份证号，插入到对应的平衡二叉树中，最后的兴趣爱好集以随机生成的数字选择数据中对应的兴趣爱好，并插入到对应的集合中。

**时间复杂度分析：**函数的时间复杂度为0(n\*gross)。

**空间复杂度分析：**其中存在复杂的调用关系，调用函数包括random1、aggregate\_init、aggregate\_insert，函数的空间复杂度为0(n\*gross)。

10. LL(root)：

**算法思想：** 插入结点的左孩子的左子树而导致失衡的情况需要进行LL旋转，如下图3-6所示：

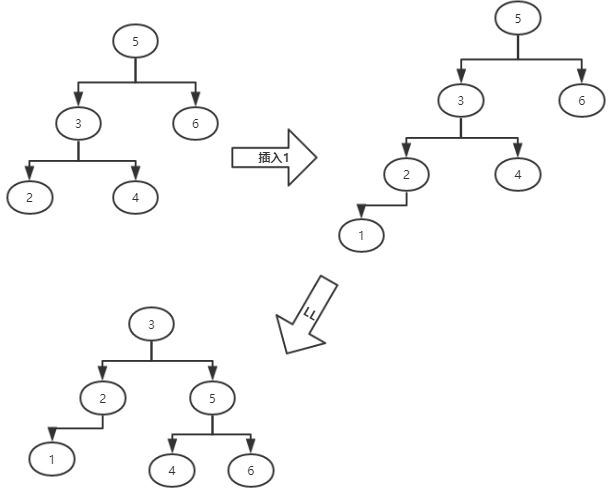


图3-6 LL的图示过程

**时间复杂度分析：**函数的时间复杂度为0(1)。

**空间复杂度分析：**函数的空间复杂度为0(1)。

11. RR(root)：

**算法思想：** 插入结点的右孩子的右子树而导致失衡的情况需要进行RR旋转，如下图3-7所示：

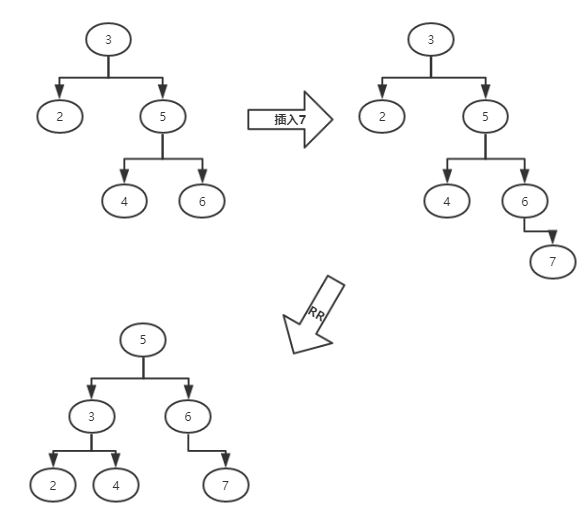


图3-7 RR的图示过程

**时间复杂度分析：**函数的时间复杂度为0(1)。

**空间复杂度分析：**函数的空间复杂度为0(1)。

12. LR(root)：

**算法思想：** 插入结点的左孩子的右子树而导致失衡的情况需要进行LR旋转，如下图3-8、3-9所示：

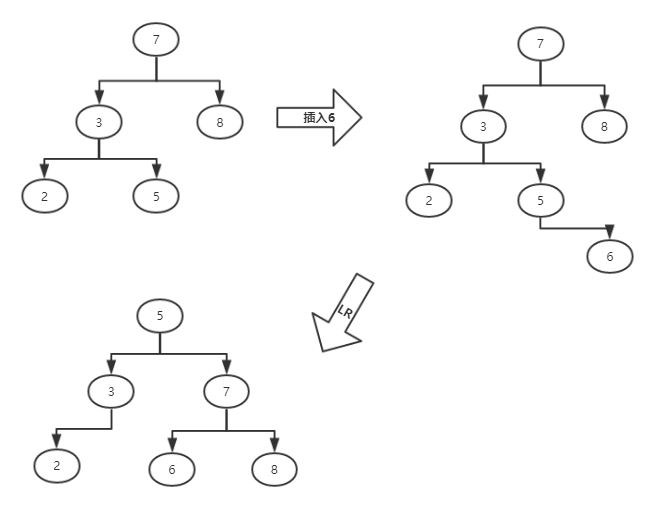


图3-8 LR的图示过程

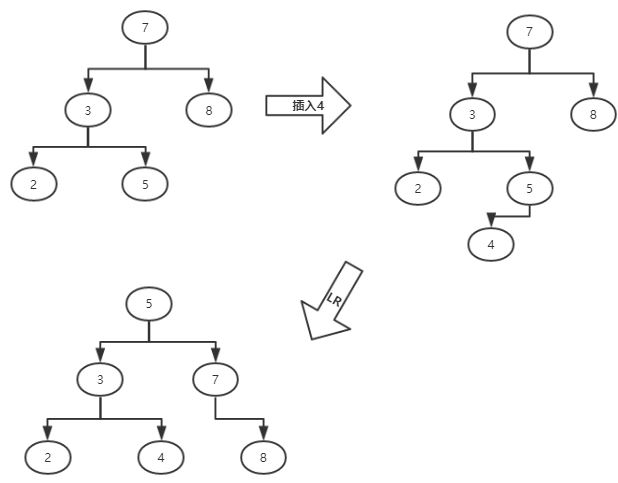


图3-9 LR的图示过程

**时间复杂度分析：**函数的时间复杂度为0(1)。

**空间复杂度分析：**函数的空间复杂度为0(1)。

13. RL(root)：

**算法思想：** 插入结点的右孩子的左子树而导致失衡的情况需要进行LR旋转，如下图3-9、3-10所示：

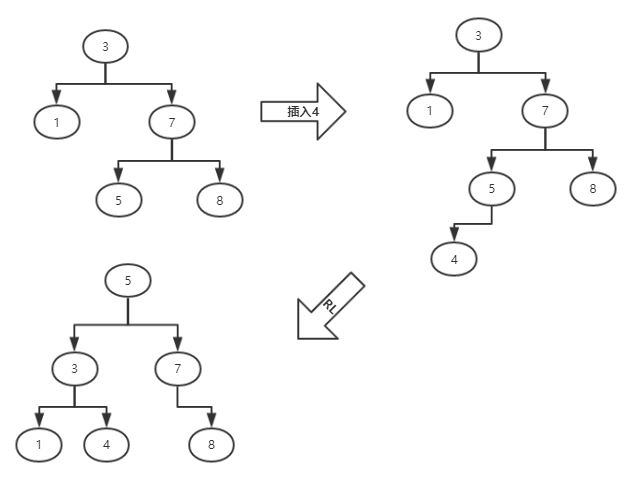


图3-9 RL的图示过程

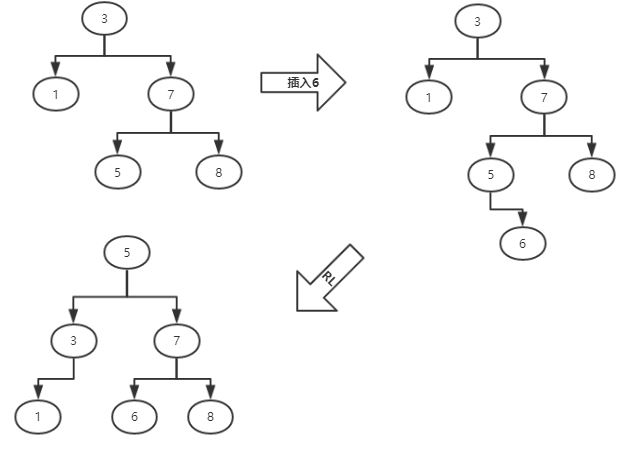


图3-10 RL的图示过程

**时间复杂度分析：**函数的时间复杂度为0(1)。

**空间复杂度分析：**函数的空间复杂度为0(1)。

4 系统实现与测试

## 4.1 系统实现

### 4.1.1 实验环境

实验代码使用visual studio 2017 community编写与编译。

### 4.1.2 演示系统操作

演示系统采用键盘操作，控制台输出操作提示与结果，上面显示可用的操作项，下面显示当前操作树的情况，输入对应的操作指令即可执行，输入非0~25作号会提示错误。

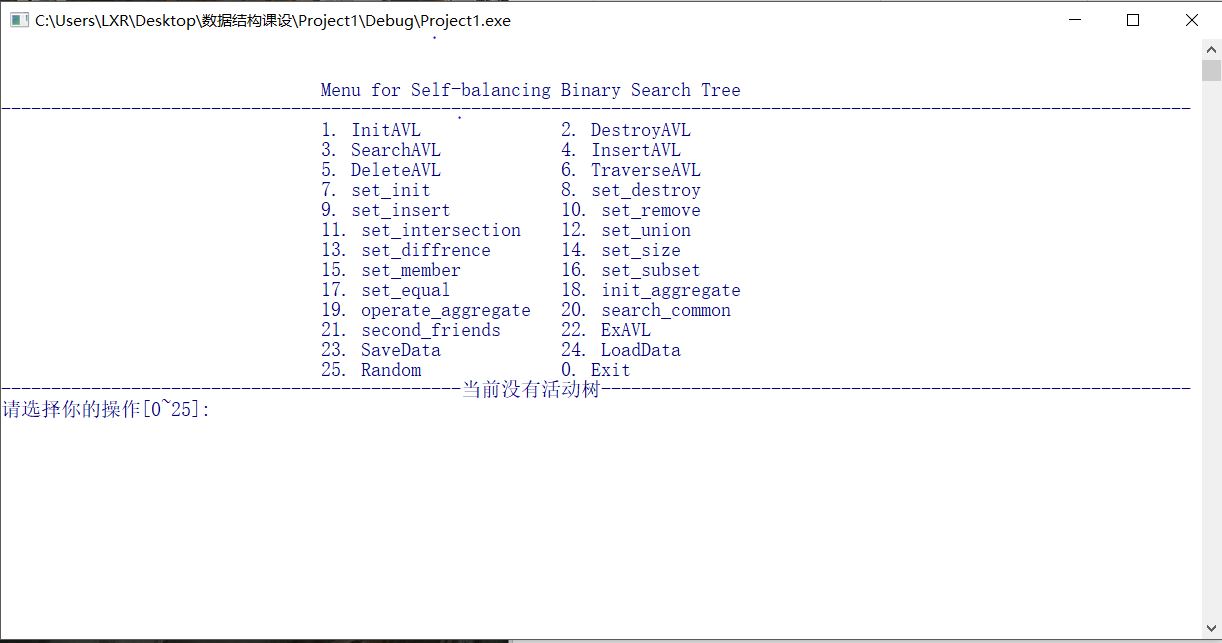


图4-1 启动系统后的操作菜单

输入操作指令0，系统退出。

图4-2 输入0后程序退出

### 4.1.3 函数的定义

void show(Forest \*head); //显示菜单

void show1(); //显示集合操作菜单

int InitAVL(Forest \*head, Forest \*\*F); //初始化

int DestroyAVL(BSTNode\*T); //销毁

int DestroyAVL1(Forest \*head, Forest \*F); //递归销毁

BSTNode\*SearchAVL(BSTNode\*T, char name[20],int choice);//查找

int InsertAVL(BSTNode\*T, TElemType data); //插入

int DeleteAVL(BSTNode\*T, char name[20]); //删除

BSTNode \*Balance(BSTNode \*T); //调整二叉树至平衡二叉树

int Depth(BSTNode \*T); //求深度

BSTNode\*Parent(BSTNode \*T, char id[20]); //求双亲结点

BSTNode\*LL(BSTNode\*root); //LL型调整函数

BSTNode\*RR(BSTNode\*root); //RR型调整函数

BSTNode\*LR(BSTNode\*root); //LR型调整函数

BSTNode\*RL(BSTNode\*root); //RL调整函数

int set\_size(BSTNode\*T); //求规模

int set\_member(BSTNode\*T, char id[20]); //求是否为成员

int set\_intersection(BSTNode\*T1, BSTNode\*T2,BSTNode\*T3);//参数为根结点

int set\_intersection1(BSTNode\*T1, BSTNode\*T2);//求交集，参数为头结点

void copy(BSTNode\*T1, BSTNode\*T2); //复制一棵树

int set\_union(BSTNode\*T1, BSTNode\*T2); //求并集，参数为根结点

int set\_union1(BSTNode\*T1, BSTNode\*T2); //求并集，参数为头结点

int set\_diffrence(BSTNode\*T1, BSTNode\*T2, BSTNode\*T3);//参数为根结点

int set\_diffrence1(BSTNode\*T1, BSTNode\*T2); //求差集，参数为头结点

int set\_subset(BSTNode\*T1, BSTNode\*T2); //判断是否为子集

int set\_equal(BSTNode\*T1, BSTNode\*T2); //判断是否相等

int search\_common(BSTNode\*T1, BSTNode\*T2);//查找相同成员

int second\_friends(BSTNode\*T1, BSTNode\*T2); //查找二度好友

BSTNode \*aggregate(BSTNode \*T, int choice); //选择集合

int aggregate\_init(BSTNode\*\*T, int choice); //对集合进行初始化

int Delete(BSTNode\*F, char id[20], int choice); //反向删除

int aggregate\_insert(BSTNode\*T1, BSTNode\*T, int choice, char id[20]);//集合插入

int InsertAVL2(BSTNode\*T1, int choice, TElemType data);//集合初始化

int aggregate\_remove(BSTNode\*T1, BSTNode\*T, int choice, char id[20]);//删除

int operate\_aggregate(BSTNode\*root, BSTNode\*T, int choice);//对集合进行操作

void TraverseAVL(BSTNode\*T); //中序遍历

void TraverseAVL2(BSTNode\*T); //前序遍历

void Traverse(Forest \*head); //遍历整个森林

Forest \*ExAVL(Forest \*head, char name[20]);//切换AVL树

int SaveData(Forest\*F); //保存数据

int SaveData1(FILE \*fp, BSTNode\*T, int choice);//递归保存数据

int SaveData2(FILE \*fp, BSTNode\*T);//递归保存数据

int LoadData(Forest\*F); //加载数据

int random(BSTNode \*T, int gross); //随机生成

int random1(BSTNode \*F,BSTNode \*T, int gross,char hobby[60][20]);//递归

### 4.1.4 函数的调用关系

函数的调用关系如图4-3所示：

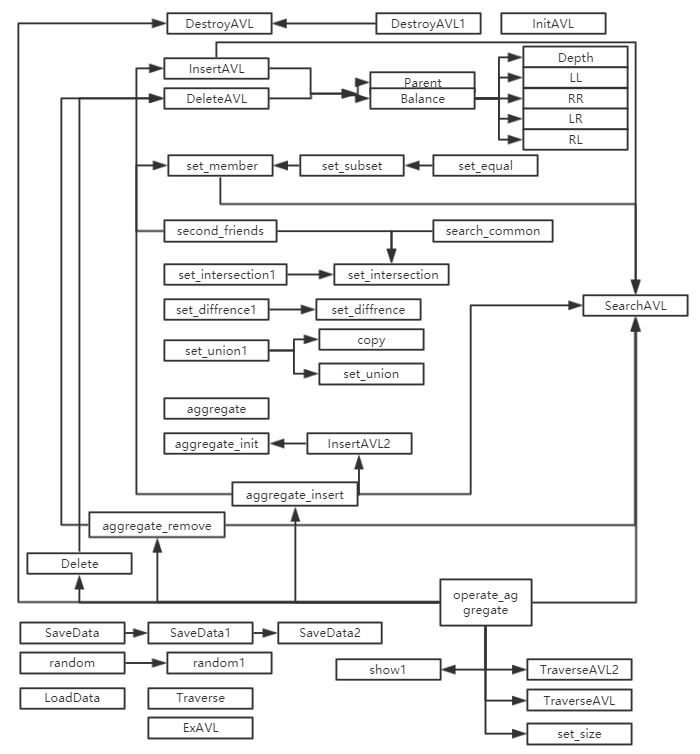


图4-3 函数间的调用关系图

## 4.2 系统测试

### 4.2.1 常用的软件测试方法

根据程序内部结构可分为白盒测试和黑盒测试：

（1）白盒测试：白盒测试是一种[测试用例设计](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B%E8%AE%BE%E8%AE%A1)方法，又称结构测试、透明盒测试、逻辑驱动测试或基于[代码](https://baike.baidu.com/item/%E4%BB%A3%E7%A0%81)的测试。白盒法是穷举路径测试。在使用这一方案时，测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据。

（2）黑盒测试：黑盒测试也称[功能测试](https://baike.baidu.com/item/%E5%8A%9F%E8%83%BD%E6%B5%8B%E8%AF%95)，它是通过测试来检测每个功能是否都能正常使用。在完全不考虑程序内部结构和内部特性的情况下，在[程序接口](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E6%8E%A5%E5%8F%A3)进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息。

根据是否运行程序分为静态测试和动态测试：

（1）静态测试：静态方法是指不运行被测程序本身，仅通过分析或检查[源程序](https://baike.baidu.com/item/%E6%BA%90%E7%A8%8B%E5%BA%8F)的语法、结构、过程、接口等来检查程序的正确性。对需求规格说明书、[软件设计](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E8%AE%BE%E8%AE%A1)说明书、[源程序](https://baike.baidu.com/item/%E6%BA%90%E7%A8%8B%E5%BA%8F)做[结构分析](https://baike.baidu.com/item/%E7%BB%93%E6%9E%84%E5%88%86%E6%9E%90)、[流程图](https://baike.baidu.com/item/%E6%B5%81%E7%A8%8B%E5%9B%BE)分析、符号执行来找错。静态方法通过程序静态特性的分析，找出欠缺和可疑之处，例如不匹配的参数、不适当的[循环嵌套](https://baike.baidu.com/item/%E5%BE%AA%E7%8E%AF%E5%B5%8C%E5%A5%97)和分支嵌套、不允许的递归、未使用过的[变量](https://baike.baidu.com/item/%E5%8F%98%E9%87%8F)、空指针的引用和可疑的计算等。

（2）动态测试：动态测试方法是指通过运行被测程序，检查运行结果与[预期结果](https://baike.baidu.com/item/%E9%A2%84%E6%9C%9F%E7%BB%93%E6%9E%9C)的差异，并分析[运行效率](https://baike.baidu.com/item/%E8%BF%90%E8%A1%8C%E6%95%88%E7%8E%87)、正确性和[健壮性](https://baike.baidu.com/item/%E5%81%A5%E5%A3%AE%E6%80%A7)等性能。这种方法由三部分组成：构造[测试用例](https://baike.baidu.com/item/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B)、[执行程序](https://baike.baidu.com/item/%E6%89%A7%E8%A1%8C%E7%A8%8B%E5%BA%8F)、分析程序的输出结果。

详细内容可见文献[5]。

### 4.2.2 测试计划

首先测试AVL树的6种基本运算：InitAVL、DestroyAVL、SearchAVL、InsertAVL、DeleteAVL、TraverseAVL，测试计划一如下表4-1所示：

表4-1 测试计划一

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试功能 | 测试顺序 | 测试输入 | 预计输出 | AVL树状态 |
| InitAVL | 3 | ZXC | 3：初始化成功 | 空树 |
| DestroyAVL | 1 |  | 1：销毁失败 | 未初始化 |
| SearchAVL | 2  4  12--15 | 12:2 1  13/14/15:2 Z | 2/4/15：查找失败  12/13/14:查找成功 | 2：同1  4：同3  12-15：同10 |
| InsertAVL | 5-10 | 1 Z 2 X 3 C 4 V 5 Z 1 M | 5-9:插入成功  10：插入失败 | 10：含有1 Z、2 X、3 C、4 V、5 Z的平衡二叉树 |
| DeleteAVL | 16 | 1 9 | 1删除成功  9删除失败 | 16：含有2 X、3 C、4 V、5 Z的平衡二叉树 |
| TraverseAVL | 11  17 |  |  | 11：同10  17：同16 |

之后测试基于AVL表示及调用其6中基本运算实现集合ADT的基本运算，因为初始化set\_init，销毁set\_destroy，插入set\_insert，删除set\_remove与InitAVL、DestroyAVL、InsertAVL相同，所以测试计划二的测试函数包括：交set\_intersection，并set\_union，差set\_diffrence，成员个数set\_size，判断元素是否为集合成员的查找set\_member，判断是否为子集set\_subset，判断集合是否相等set\_equal以及方便检查所需的文件存取和随机生成数据。测试计划二如表4-2所示：

表4-2 测试计划二

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试功能 | 测试顺序 | 测试输入 | 预计输出 | AVL树状态 |
| set\_intersection | 3  4 | 3：1 1  10 1  4：10 2 1  2 1 | 3：空集  4：1 朱 7 尤 | 同1 |
| set\_union | 5 | 1 1  10 1 | 1 10 2 7 9 | 同1 |
| set\_diffrence | 6 | 1 1  10 1 | 1 10 2 7 9 | 同1 |
| set\_size | 7 |  | 10 | 同1 |
| set\_member | 8  9 | 1  11 | 8：是  9：不是 | 同1 |
| set\_subset | 10  11 | 10：5 1 7 1  11：5 1 2 1 | 10：不是  11：是 | 同1 |
| set\_equal | 12 | 1 1 2 1 | 不相同 | 同1 |
| SaveData | 2 |  | 保存成功 | 同1 |
| Random | 1 | 10 | 随机生成成功 | 可查看文件 |

然后对基于集合ADT实现的应用层功能进行测试，测试计划三如表4-3所示：

表4-3 测试计划三

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试功能 | 测试顺序 | 测试输入 | 预计输出 | AVL树状态 |
| LoadData | 1 2 |  | 1：成功 2：失败 | 可查看文件 |
| init\_aggregate | 3 4 | 4 1  5 1 | 3：初始化成功  4：初始化失败 | 同1 |
| operate\_aggregate | 5 6  8 9  10 11  12 | 5-6：4 1 插入8并遍历  8：4 1 删除8 9 10  9：9 1 修改10为1  10-11：9 1 查找1 2  12：1 1 1 | 5：插入成功  6：遍历成功  8：8删除成功  9删除失败  10删除失败  9：修改成功  10：1查找成功  11：2查找失败  12：销毁成功 | 可查看文件 |
| search\_common | 16 | 1 2 | 输出相同信息 | 同14 |
| second\_friends | 17 | 2 | 6 8 9 | 同14 |
| set\_equal | 7 | 4 1 5 1 | 集合相同 | 同6 |
| SaveData | 10 13 15 |  |  | 10：同9  13：同12  15：同14 |
| set\_remove | 14 | 10 | 删除成功 | 可查看文件 |

最后对多数管理功能进行测试，测试计划四如表4-4所示：

表4-4 测试计划四

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试功能 | 测试顺序 | 测试输入 | 预计输出 | AVL树状态 |
| LoadData | 2 |  | 2：成功 | 2：同1 |
| InitAVL | 1 4 5 | 1/4：ZXC  5：X | 1/5：成功 4：失败 | 1:名为ZXC  4：同1  5：名为X |
| TraverseAVL | 3 6 8 |  |  | 3/8：同1  6：同5 |
| ExAVL | 7 9 | 7：ZXC  9：X | 7：切换成功  9：切换失败 | 7/9：同1 |

### 4.2.3 测试结果与分析

测试计划一：

对AVL树的六种基本运算的检验，对应表4-1。

测试结果：

（1）销毁二叉树，因为当前没有初始化二叉树，所以会提示当前没有活动树。



图4-4 销毁二叉树

（2）查找相应成员，因为当前没有初始化二叉树，所以会提示当前没有活动树。



图4-5 查找成员

（3）初始化一个名字为ZXC的二叉树，初始化成功。



图4-6 初始化一个平衡二叉树

（4）查找对应成员，因为当前平衡二叉树为空，不能进行查找，所以查找失败。



图4-7 查找成员

（5）依次插入成员Z（1）、X（2）、C（3）、V（4）、Z（5）、M（1），前面为名称，括号内为身份证号。由于名称可重复而身份证号不可重复，所以前五个插入成功，最后一个插入失败。

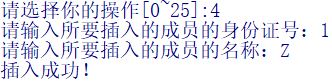


图4-8 插入身份证号为1，名称为Z的成员，插入成功

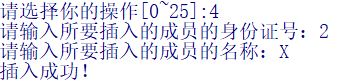


图4-9 插入身份证号为2，名称为X的成员，插入成功

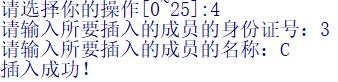


图4-10 插入身份证号为3，名称为C的成员，插入成功

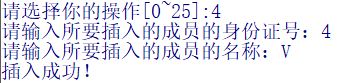


图4-11 插入身份证号为4，名称为V的成员，插入成功

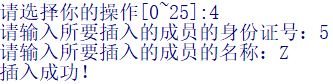


图4-12 插入身份证号为5，名称为Z的成员，插入成功

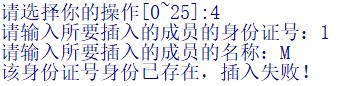


图4-13 插入身份证号为1，名称为M的成员,插入失败

（6）遍历当前活动树，检验插入结果，得插入操作无误。

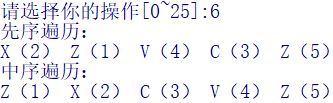
****

图4-13 遍历AVL树

（7）查找成员，通过身份证号查询，可获得唯一结果。

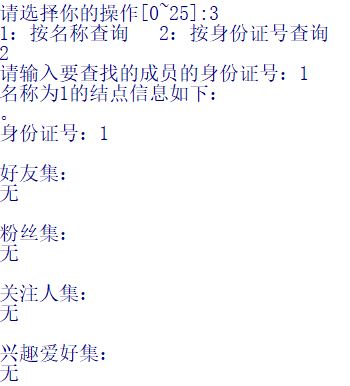
****

图4-12 用身份证号查找成员

（8）查找成员，通过名称查找，结果不唯一，依次查找名称为Z的成员，结果如下。

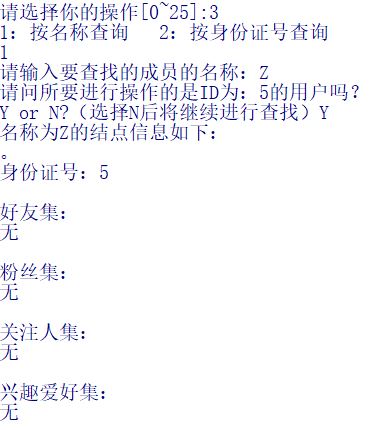


图4-13 用名称查找成员

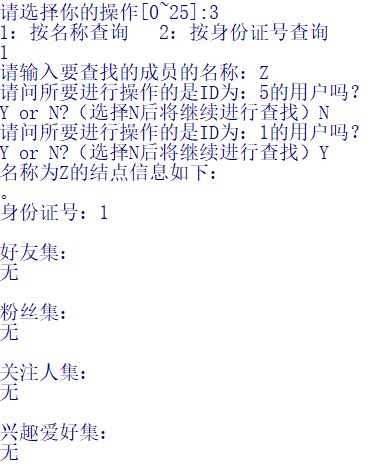


图4-14 用名称查找成员

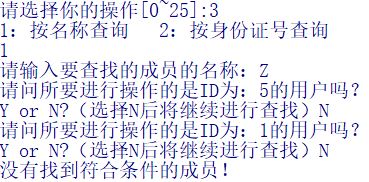


图4-15 用名称查找成员

（9）删除成员，支持多成员同时删除，同时删除身份证号为1和9的成员，前者删除成功但身份证号为9的成员不存在所以删除失败。

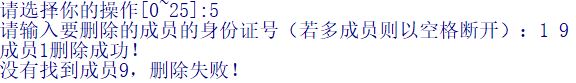
****

图4-16 删除存在的节点

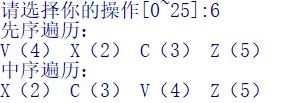
****

图4-17 删除不存在的节点

（10）退出系统。

测试计划二：

对基于AVL表示及调用其6种基本运算实现集合ADT的基本运算的检验，对应表4-2。

测试结果：

（1）初始化一个名为ZXC的树并随机生成规模为10的数据。

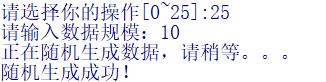


图4-18 随机生成规模为10的数据

（2）将数据存入文件中，并通过文件显示随机生成的相关数据。newname文件中存入的是成员信息，格式为身份证号 姓名。newfriends文件存的是成员的好友集信息，第一个是成员信息，之后是其好友信息，每行显示二十个好友，“#”为终止符。newfan文件中存的是成员的粉丝集信息，第一个是成员信息，之后是其粉丝信息，每行显示二十个粉丝，“#”为终止符。newconcern文件中存的是成员的关注人集合，第一个是成员信息，之后是其关注的人的信息，每行显示二十个关注人，“#”为终止符。newhobby文件存的是成员的兴趣爱好集信息，第一个是成员信息，之后是兴趣爱好信息，为统一格式，将兴趣爱好所对应的名称都设置为0。

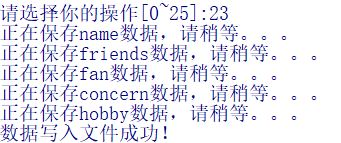


图4-19 将数据存入文件

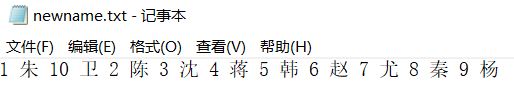


图4-20 newname文件

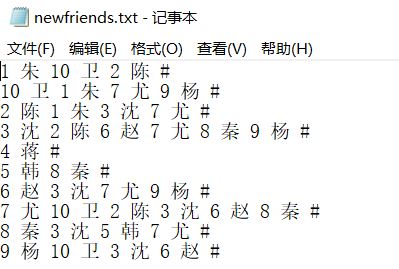


图4-21 newfriends文件

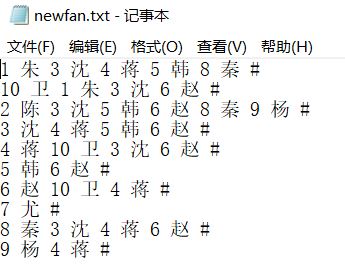


图4-22 newfan文件

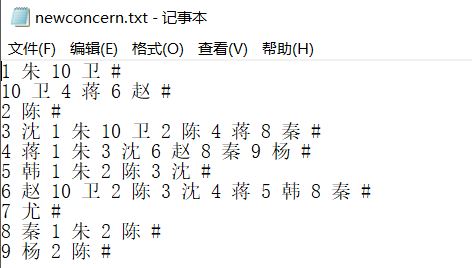


图4-23 newconcern文件

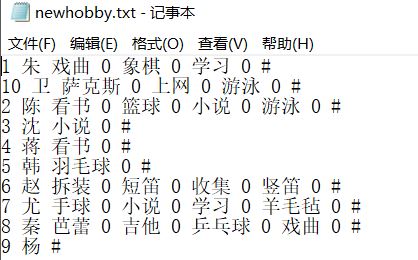


图4-24 newhobby文件

（3）求成员1和成员10的好友集的交集，根据文件显然为空。

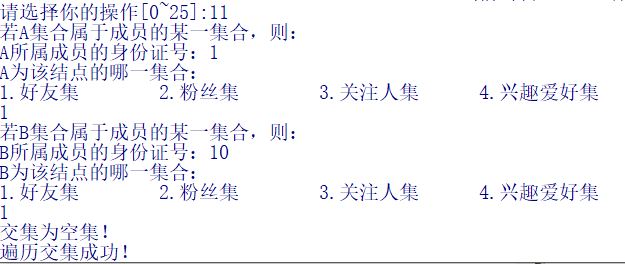


图4-25 求交集

（4）求成员10和成员2的好友集的交集，由文件显然得为1和7。

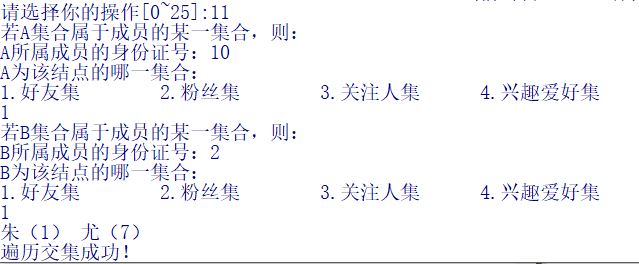


图4-26 求交集

（5）求成员1和成员10的好友集的并集。

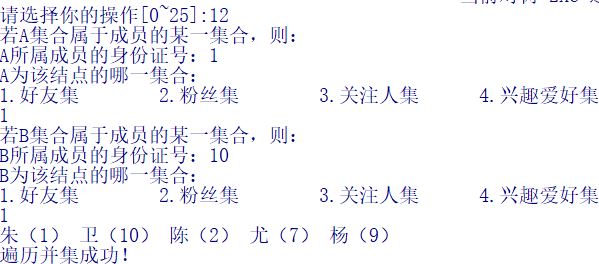


图4-27 求并集

（6）求成员1和成员10的好友集的差集。

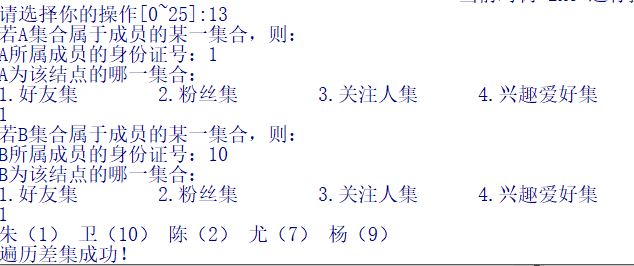


图4-28 求差集

（7）求当前平衡二叉树的规模，即成员的个数。



图4-29 求成员个数

（8）求1和11是否是当前成员集合的成员之一，显然1是而11不是。

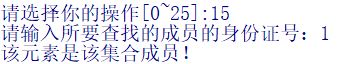


图4-30 1是集合成员

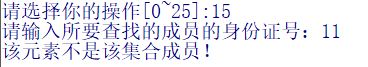


图4-31 11不是集合成员

（9）判断5的好友集是否是7的好友集的子集，显然是。判断5的好友集是否是2的好友集的子集，显然不是。

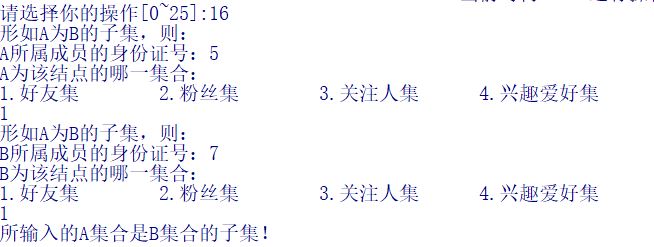


图4-32 是子集

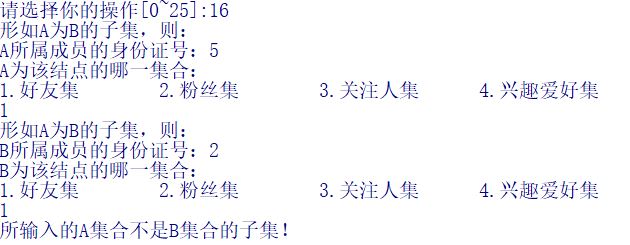


图4-33 不是子集

（10）判断1的好友集是否和2的好友集相同，显然不相同。

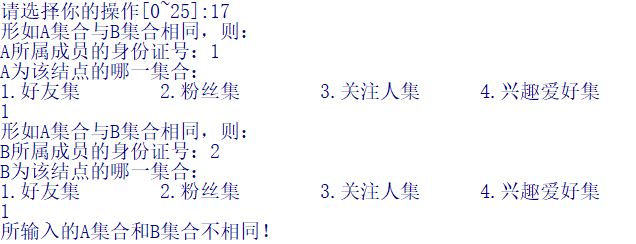


图4-34 集合不相同

（11）退出系统。

测试计划三：

对基于集合ADT实现应用层功能的检验，对应表4-3。

测试结果：

（1）初始化一个名为ZXC的树并连续两次从文件中读取数据，文件内容为测试计划二的内容。第一次读取数据成功，第二次读取失败。

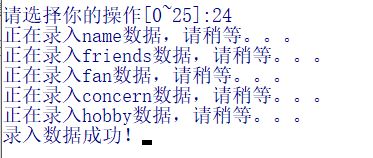


图4-35 录入文件中数据



图4-36 录入文件中数据

（2）初始化4和5的好友集合。4的好友集合初始化成功，5的好友集合已经初始化所以初始化失败。

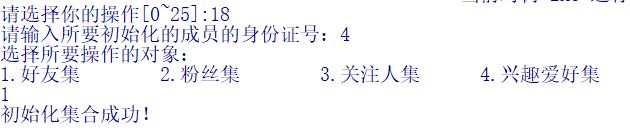


图4-37 初始化4的好友集

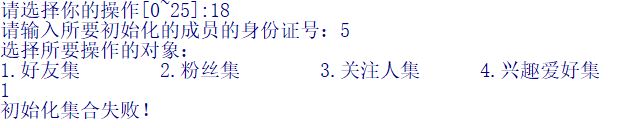


图4-38 初始化5的好友集

（3）在4的好友集中添加成员8，并通过遍历判断是否添加成功。

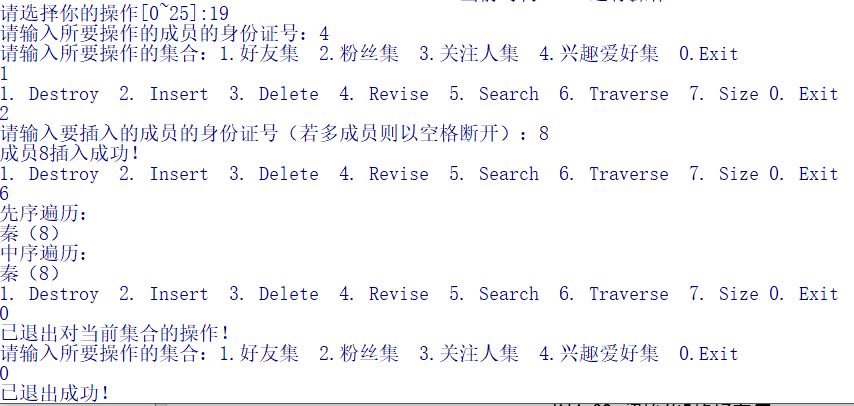


图4-39 添加集合成员并遍历集合

（4）求4的好友集是否和5的好友集相同。

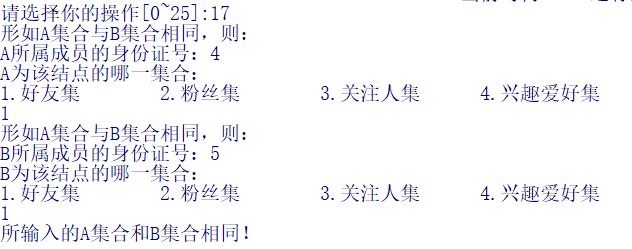


图4-40 集合相同

（5）在4的好友集合同时删除8 9 10，删除8成功，但9和10不是集合成员所以删除失败。

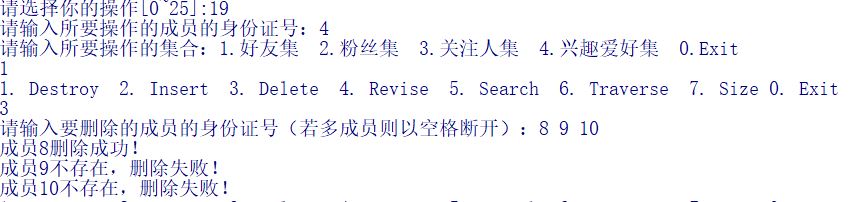


图4-41 删除集合成员

（6）修改9的好友集中的10为1，并保存文件，通过文件显示双向修改，删除10好友集中的9，在1的好友集添加9，在9的好友集添加1。

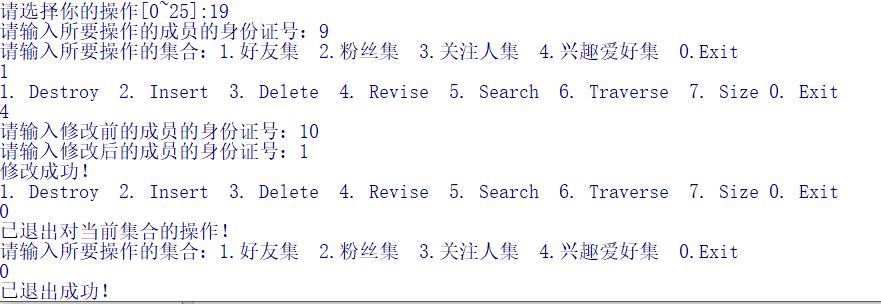


图4-42 修改集合成员

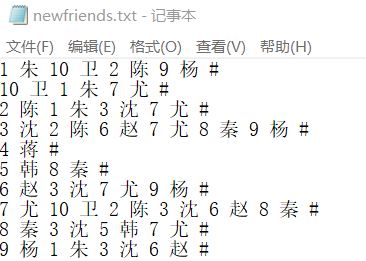
****

图4-43 newfriends文件

（7）查找1和2是否是9的好友集的成员，显然1是而2不是。

****

图4-44 查找集合成员

（8）销毁1的好友集合，并将数据保存在文件中，通过查看文件看出反向删除。

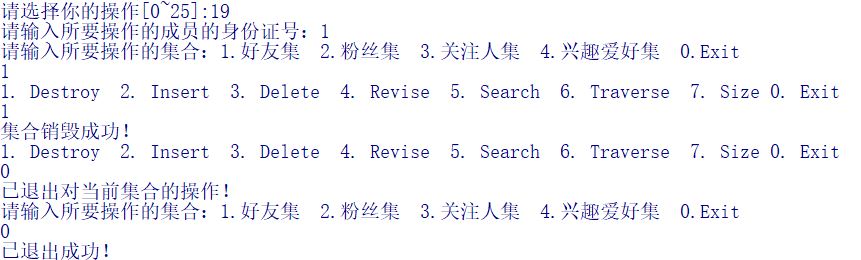


图4-45 销毁集合

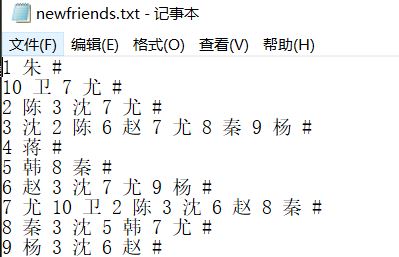


图4-46 newfriends文件

（9）删除成员10，并将数据保存在文件中，通过查看文件可看出所有10的内容都已删除。

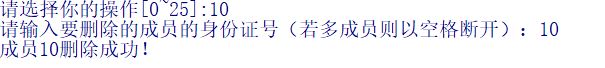


图4-47 删除成员

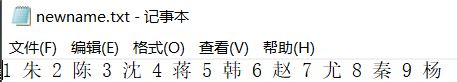


图4-48 newname文件

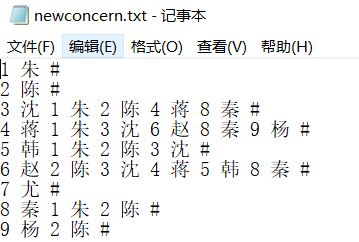


图4-49 newconcern文件

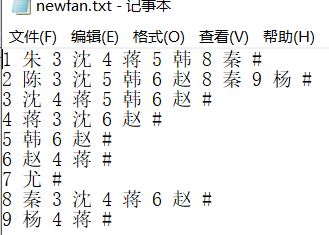


图4-50 newfan文件

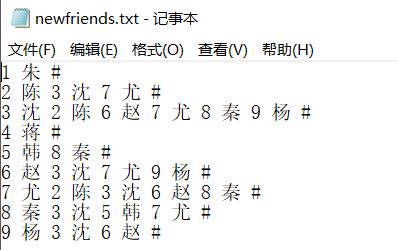


图4-51 newfriends文件

（10）查找1和2的共同好友、共同粉丝、共同观众、共同爱好。

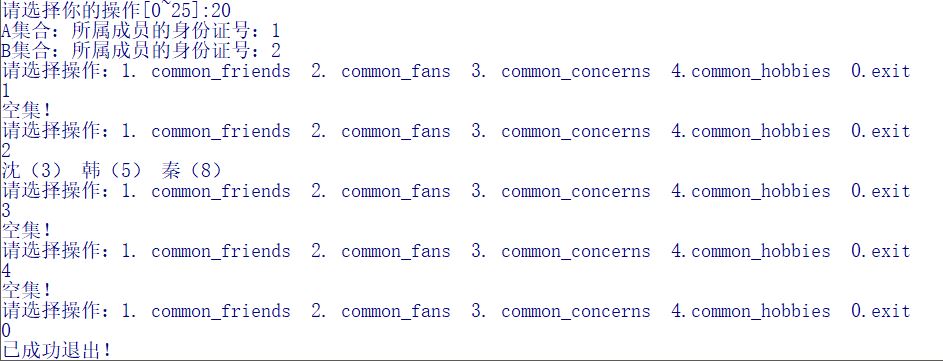


图4-52 查找共同信息

（11）查找2的二度好友。

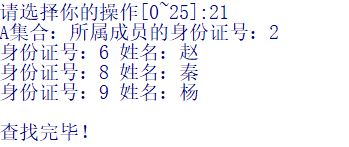


图4-53 查找二度好友

（12）退出系统。

测试计划四：

对多树管理功能的检验，对应表4-4。

测试结果：

1. 初始化一个名为ZXC的树，之后录入文件中的数据并遍历。



图4-54 初始化ZXC

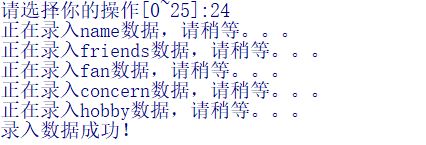


图4-55 录入数据

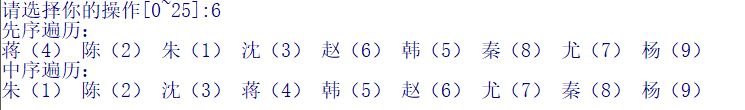


图4-56 遍历ZXC

1. 再初始化一个名为ZXC的树。



图4-57 初始化ZXC

1. 初始化一个名为X的树，并遍历。

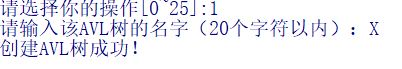


图4-58 初始化X



图4-59 遍历X

1. 将活动树切换为ZXC，并遍历。



图4-60 切换活动树为ZXC

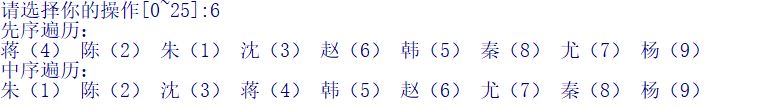


图4-61 遍历活动树

1. 将活动树切换为Z，因为没有名为Z的二叉树所以切换失败。

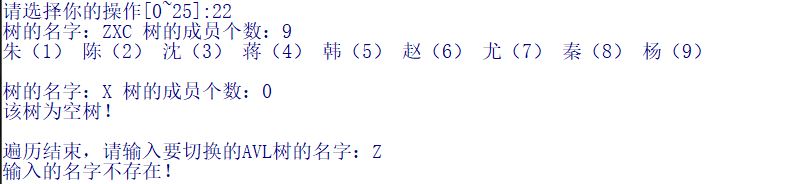


图4-62 切换活动树为Z

（6）退出系统。

5 总结与展望

## 5.1 全文总结

（1）掌握了在插入删除等操作中维持AVL树平衡的方法，这是本次实验的重点内容之一。

（2）深一步熟悉了递归算法。

## 5.2 工作展望

在今后的研究中，围绕着如下几个方面开展工作

1. 学习更优秀的算法，降低时间复杂度和空间复杂度。
2. 学会简化代码，避免大量重复等情况出现。
3. 学习对界面的处理。

（4）设计人性化界面，如学习以树状直接输出二叉树，而非通过先序遍历和中序遍历人工构造二叉树，后者对于大量数据时较难操作。

（5）多尝试使用复杂的数据结构，用实际操作更深入地了解掌握数据结构。

6 体会

本次课程设计，首先是我对AVL树的数据结构有了更深入的理解，其次是克服了平衡二叉树相关运算的难点，总体来说收获颇多。

事实上本次课程设计最大的问题就是自己对题目的理解比较奇怪，完全往着错误的方向理解，并且没有真正认识到什么时基础操作，什么时应用层操作，以至于在初次检查后对程序进行一次大改。吃一堑长一智，这无非是警示自己以后在写程序时不可心急，必须是正确理解题意后再开始写，否则只会越错越多。

在最初选题时一方面担心自己能力不足，一方面是时间原因，最终选择了第三道课程设计题目。在最初着手写程序时，就在插入功能处停滞了很久。之后我又重新看了一遍《数据结构》中对平衡二叉树的描述，并在网上查阅了不少关于平衡二叉树的操作解读，最终克服了插入时的问题。之后是删除功能，删除功能较之插入功能要更复杂一些，多种情况下选择删除功能所要进行的操作不同，这就使得在构思时应考虑仔细全面，这一方面我并没有做好，在最后检查课设程序中又遇到了在删除时程序崩溃的情况，在老师的理解下我又以错误为入口检查时哪种情况并未考虑周全，最终修改完整程序。这是一次教训，对我而言也是一种经验，想必在以后会更加认真全面一些。这也同时显示出了上机实习的重要性，在学习的时候只觉得纸上画来画去算不上什么难事，只有在实际操作时才能意识到平时平时考虑事情的时候想得总是太简单。

通过这次课程设计，也让我重温了一部分《数据结构》。“温故而知新”，其实不管是否能达到“知新”得地步，“温故”都是不可或缺的。学习如此，学习写代码更是如此。

参考文献

[1] 严蔚敏, 吴伟民. 数据结构（C语言版）. 北京: 清华大学出版社,1997

[2] 严蔚敏, 吴伟民, 米宁. 数据结构题集（C语言版）. 北京: 清华大学出版社,1999

[3]URL: <http://www.cnblogs.com/abatei/archive/2008/11/17/1335031.html>

[4][URL: https://baike.baidu.com](URL:%20https://baike.baidu.com)

[5][URL:](URL:https://baike.baidu.com) https://zhidao.baidu.com/question/492182095.html

# 附录

#define \_CRT\_SECURE\_NO\_DEPRECATE

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#include<string.h>

#include<time.h>

#define OK 1

#define ERROR 0

typedef struct TElemType {

char id[20]; //身份证号

char name[20]; //姓名

struct BSTNode \*friends; //好友集

struct BSTNode \*fan; //粉丝集

struct BSTNode \*concern; //关注人集

struct BSTNode \*hobby; //兴趣爱好集

}TElemType;

typedef struct BSTNode {

struct TElemType data; //数据信息

int BF; //平衡因子

struct BSTNode \*lchild, \*rchild; //左右孩子指针

}BSTNode;

typedef struct Forest {

char name[20];

struct BSTNode \*HeadNode; //二叉树头结点

struct Forest \*next;

}Forest;

void show(Forest \*head); //显示菜单

void show1(); //显示集合操作菜单

int InitAVL(Forest \*head, Forest \*\*F);

int DestroyAVL(BSTNode\*T);

int DestroyAVL1(Forest \*head, Forest \*F);

BSTNode\*SearchAVL(BSTNode\*T, char name[20],int choice);

int InsertAVL(BSTNode\*T, TElemType data);

int DeleteAVL(BSTNode\*T, char name[20]);

BSTNode \*Balance(BSTNode \*T);

int Depth(BSTNode \*T);

BSTNode\*Parent(BSTNode \*T, char id[20]);

BSTNode\*LL(BSTNode\*root);

BSTNode\*RR(BSTNode\*root);

BSTNode\*LR(BSTNode\*root);

BSTNode\*RL(BSTNode\*root);

int set\_size(BSTNode\*T);

int set\_member(BSTNode\*T, char id[20]);

int set\_intersection(BSTNode\*T1, BSTNode\*T2,BSTNode\*T3);

int set\_intersection1(BSTNode\*T1, BSTNode\*T2);

void copy(BSTNode\*T1, BSTNode\*T2);

int set\_union(BSTNode\*T1, BSTNode\*T2);

int set\_union1(BSTNode\*T1, BSTNode\*T2);

int set\_diffrence(BSTNode\*T1, BSTNode\*T2, BSTNode\*T3);

int set\_diffrence1(BSTNode\*T1, BSTNode\*T2);

int set\_subset(BSTNode\*T1, BSTNode\*T2);

int set\_equal(BSTNode\*T1, BSTNode\*T2);

int search\_common(BSTNode\*T1, BSTNode\*T2);

int second\_friends(BSTNode\*T1, BSTNode\*T2);

BSTNode \*aggregate(BSTNode \*T, int choice);

int aggregate\_init(BSTNode\*\*T, int choice);

int aggregate\_destroy(BSTNode\*T, int choice);

int Delete(BSTNode\*F, char id[20], int choice);

int aggregate\_insert(BSTNode\*T1, BSTNode\*T, int choice, char id[20]);

int InsertAVL2(BSTNode\*T1, int choice, TElemType data);

int aggregate\_remove(BSTNode\*T1, BSTNode\*T, int choice, char id[20]);

int operate\_aggregate(BSTNode\*root, BSTNode\*T, int choice);

void TraverseAVL(BSTNode\*T); //中序遍历

void TraverseAVL2(BSTNode\*T); //前序遍历

void Traverse(Forest \*head); //遍历整个森林

Forest \*ExAVL(Forest \*head, char name[20]);//切换AVL树

int SaveData(Forest\*F);

int SaveData1(FILE \*fp, BSTNode\*T, int choice);

int SaveData2(FILE \*fp, BSTNode\*T);

int LoadData(Forest\*F);

int random(BSTNode \*T, int gross);

int random1(BSTNode \*F,BSTNode \*T, int gross,char hobby[60][20]);

int j=0;//格式化输出

int main()

{

system("color F1");//改变颜色

Forest \*head = (Forest\*)malloc(sizeof(Forest));

head->HeadNode = NULL; memset(head->name, '0', 20); head->next = NULL;

Forest \*F = head->next;

BSTNode \*T = (BSTNode\*)malloc(sizeof(BSTNode));

BSTNode \*T1 = (BSTNode\*)malloc(sizeof(BSTNode));

BSTNode \*agg1 = NULL, \*agg2 = NULL;

Forest \*F1 = (Forest\*)malloc(sizeof(Forest));

int op = 1;

int choice=0,choice2=0,choice3=0,re=0;

char id[20], name[20]; char s[100];

TElemType data;

while (op)

{

show(F);

scanf("%d", &op); getchar();

switch (op)

{

case 0:

printf("欢迎下次再使用本系统！\n");

getchar(); break;

case 1:case 7:

if (InitAVL(head, &F) == OK)

printf("创建AVL树成功！\n");

else

printf("创建AVL树失败！\n");

getchar();

break;

case 2:case 8:

if (F == NULL)

printf("当前没有活动树！\n");

else if(DestroyAVL1(head, F) == OK)

{

F = NULL;

printf("销毁当前AVL树成功！\n");

}

else

printf("销毁当前AVL树失败！\n");

getchar();

break;

case 3:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("该活动树为空树，查找失败！");

else

{

re = 0;

while (re == 0)

{

printf("1：按名称查询\t2：按身份证号查询\n"); scanf("%d", &choice3); getchar();

switch (choice3)

{

case 1:re = 1; printf("请输入要查找的成员的名称："); scanf("%s", name); getchar(); break;

case 2:re = 1; printf("请输入要查找的成员的身份证号："); scanf("%s", name); getchar(); break;

default:printf("输入错误，请重新选择");

}

}

T = SearchAVL(F->HeadNode->lchild, name,choice3);

if (T != NULL)

{

printf("名称为%s的结点信息如下：\n。", name);

printf("\n身份证号：%s", T->data.id);

printf("\n\n好友集：\n");

j = 0;

if (T->data.friends == NULL|| T->data.friends->lchild==NULL) printf("无");

else TraverseAVL(T->data.friends->lchild);

printf("\n\n粉丝集：\n");

j = 0;

if (T->data.fan == NULL|| T->data.fan->lchild == NULL) printf("无");

else TraverseAVL(T->data.fan->lchild);

printf("\n\n关注人集：\n");

j = 0;

if (T->data.concern == NULL|| T->data.concern->lchild == NULL) printf("无");

else TraverseAVL(T->data.concern->lchild);

printf("\n\n兴趣爱好集：\n");

j = 0;

if (T->data.hobby == NULL|| T->data.hobby->lchild == NULL) printf("无");

else TraverseAVL(T->data.hobby->lchild);

}

else

printf("没有找到符合条件的成员！");

}

getchar();

break;

case 4:case 9:

if (F == NULL)

printf("当前没有活动树！\n");

else

{

printf("请输入所要插入的成员的身份证号："); scanf("%s", data.id); getchar();

printf("请输入所要插入的成员的名称："); scanf("%s", data.name); getchar();

data.concern = NULL; data.fan = NULL; data.friends = NULL; data.hobby = NULL;

if (InsertAVL(F->HeadNode, data) == OK)

printf("插入成功！");

else

printf("该身份证号身份已存在，插入失败！");

}

getchar();

break;

case 5:case 10:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("该活动树为空树，删除失败！");

else

{

printf("请输入要删除的成员的身份证号（若多成员则以空格断开）："); fgets(s, 100, stdin);

for (int i = 0, j = 0; i < 100; i++)

{

if (s[i] == ' ' || s[i] == '\n')

{

id[j] = '\0'; j = 0;

T = SearchAVL(F->HeadNode->lchild, id, 2);

if (DeleteAVL(F->HeadNode, id) == OK)

{

Delete(F->HeadNode->lchild, id, 1);

Delete(F->HeadNode->lchild, id, 2);

Delete(F->HeadNode->lchild, id, 3);

printf("成员%s删除成功！\n", id);

}

else

printf("没有找到成员%s，删除失败！\n",id);

}

else

{

id[j] = s[i]; j++;

}

}

}

getchar();

break;

case 6:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("当前活动树为空树！\n");

else

{

j = 0;

printf("先序遍历：\n");

TraverseAVL2(F->HeadNode->lchild);

j = 0;

printf("\n中序遍历：\n");

TraverseAVL(F->HeadNode->lchild);

}

getchar();

break;

case 11:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("该活动树为空树，判断失败！");

else

{

printf("若A集合属于成员的某一集合，则：\nA所属成员的身份证号："); scanf("%s", id); getchar();

T = SearchAVL(F->HeadNode->lchild, id, 2);

if (T == NULL)

{

printf("没有找到成员%s！", id);

getchar(); break;

}

re = 0;

while (re == 0)

{

printf("A为该结点的哪一集合：\n1.好友集\t2.粉丝集\t3.关注人集\t4.兴趣爱好集\n"); scanf("%d", &choice); getchar();

switch (choice)

{

case 1:case 2:case 3:case 4:re = 1; agg1 = aggregate(T, choice); break;

default:printf("输入错误，请重新输入");

}

}

if (agg1 == NULL)

{

printf("集合A未初始化，无法进行操作！");

getchar();

break;

}

printf("若B集合属于成员的某一集合，则：\nB所属成员的身份证号："); scanf("%s", id); getchar();

T1 = SearchAVL(F->HeadNode->lchild, id, 2);

if (T1 == NULL)

{

printf("没有找到成员%s！", id);

getchar(); break;

}

re = 0;

while (re == 0)

{

printf("B为该结点的哪一集合：\n1.好友集\t2.粉丝集\t3.关注人集\t4.兴趣爱好集\n"); scanf("%d", &choice2); getchar();

switch (choice2)

{

case 1:case 2:case 3:case 4:re = 1; agg2 = aggregate(T1, choice2); break;

default:printf("输入错误，请重新输入");

}

}

if (agg2 == NULL)

{

printf("集合B未初始化，无法进行操作！");

getchar();

break;

}

if (set\_intersection1(agg1, agg2) == OK)

printf("\n遍历交集成功！");

}

getchar();

break;

case 12:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("该活动树为空树，判断失败！");

else

{

printf("若A集合属于成员的某一集合，则：\nA所属成员的身份证号："); scanf("%s", id); getchar();

T = SearchAVL(F->HeadNode->lchild, id, 2);

if (T == NULL)

{

printf("没有找到成员%s！", id);

getchar(); break;

}

re = 0;

while (re == 0)

{

printf("A为该结点的哪一集合：\n1.好友集\t2.粉丝集\t3.关注人集\t4.兴趣爱好集\n"); scanf("%d", &choice); getchar();

switch (choice)

{

case 1:case 2:case 3:case 4:re = 1; agg1 = aggregate(T, choice); break;

default:printf("输入错误，请重新输入");

}

}

printf("若B集合属于成员的某一集合，则：\nB所属成员的身份证号："); scanf("%s", id); getchar();

T1 = SearchAVL(F->HeadNode->lchild, id, 2);

if (T1 == NULL)

{

printf("没有找到成员%s！", id);

getchar(); break;

}

re = 0;

while (re == 0)

{

printf("B为该结点的哪一集合：\n1.好友集\t2.粉丝集\t3.关注人集\t4.兴趣爱好集\n"); scanf("%d", &choice2); getchar();

switch (choice2)

{

case 1:case 2:case 3:case 4:re = 1; agg2 = aggregate(T1, choice2); break;

default:printf("输入错误，请重新输入");

}

}

if (agg1 == NULL)

printf("集合A未初始化，无法进行判断！");

else if (agg2 == NULL)

printf("集合B未初始化，无法进行判断！");

if (set\_union1(agg1, agg2) == OK)

printf("\n遍历并集成功！");

}

getchar();

break;

case 13:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("该活动树为空树，判断失败！");

else

{

printf("若A集合属于成员的某一集合，则：\nA所属成员的身份证号："); scanf("%s", id); getchar();

T = SearchAVL(F->HeadNode->lchild, id, 2);

if (T == NULL)

{

printf("没有找到成员%s！", id);

getchar(); break;

}

re = 0;

while (re == 0)

{

printf("A为该结点的哪一集合：\n1.好友集\t2.粉丝集\t3.关注人集\t4.兴趣爱好集\n"); scanf("%d", &choice); getchar();

switch (choice)

{

case 1:case 2:case 3:case 4:re = 1; agg1 = aggregate(T, choice); break;

default:printf("输入错误，请重新输入");

}

}

printf("若B集合属于成员的某一集合，则：\nB所属成员的身份证号："); scanf("%s", id); getchar();

T1 = SearchAVL(F->HeadNode->lchild, id, 2);

if (T1 == NULL)

{

printf("没有找到成员%s！", id);

getchar(); break;

}

re = 0;

while (re == 0)

{

printf("B为该结点的哪一集合：\n1.好友集\t2.粉丝集\t3.关注人集\t4.兴趣爱好集\n"); scanf("%d", &choice2); getchar();

switch (choice2)

{

case 1:case 2:case 3:case 4:re = 1; agg2 = aggregate(T1, choice2); break;

default:printf("输入错误，请重新输入");

}

}

if (agg1 == NULL)

printf("集合A未初始化，无法进行判断！");

else if (agg2 == NULL)

printf("集合B未初始化，无法进行判断！");

if (set\_diffrence1(agg1, agg2) == OK)

printf("\n遍历差集成功！");

}

getchar();

break;

case 14:

if (F == NULL)

printf("当前没有活动树！\n");

else

printf("该集合中成员个数为：%d。", set\_size(F->HeadNode->lchild));

getchar();

break;

case 15:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("该活动树为空树，判断失败！");

else

{

printf("请输入所要查找的成员的身份证号："); scanf("%s", id); getchar();

if (set\_member(F->HeadNode->lchild, id) == OK)

printf("该元素是该集合成员！");

else

printf("该元素不是该集合成员！");

}

getchar();

break;

case 16:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("该活动树为空树，判断失败！");

else

{

printf("形如A为B的子集，则：\nA所属成员的身份证号："); scanf("%s", id); getchar();

T = SearchAVL(F->HeadNode->lchild, id, 2);

if (T == NULL)

{

printf("没有找到成员%s！", id);

getchar(); break;

}

re = 0;

while (re == 0)

{

printf("A为该结点的哪一集合：\n1.好友集\t2.粉丝集\t3.关注人集\t4.兴趣爱好集\n"); scanf("%d", &choice); getchar();

switch (choice)

{

case 1:case 2:case 3:case 4:re = 1; agg1 = aggregate(T, choice); break;

default:printf("输入错误，请重新输入");

}

}

printf("形如A为B的子集，则：\nB所属成员的身份证号："); scanf("%s", id); getchar();

T1 = SearchAVL(F->HeadNode->lchild, id, 2);

if (T1 == NULL)

{

printf("没有找到成员%s！", id);

getchar(); break;

}

re = 0;

while (re == 0)

{

printf("B为该结点的哪一集合：\n1.好友集\t2.粉丝集\t3.关注人集\t4.兴趣爱好集\n"); scanf("%d", &choice2); getchar();

switch (choice2)

{

case 1:case 2:case 3:case 4:re = 1; agg2 = aggregate(T1, choice2); break;

default:printf("输入错误，请重新输入");

}

}

if (agg1 == NULL)

printf("集合A未初始化，无法进行判断！");

else if (agg2 == NULL)

printf("集合B未初始化，无法进行判断！");

if (set\_subset(agg1->lchild, agg2->lchild) == OK)

printf("所输入的A集合是B集合的子集！");

else

printf("所输入的A集合不是B集合的子集！");

}

getchar();

break;

case 17:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("该活动树为空树，判断失败！");

else

{

printf("形如A集合与B集合相同，则：\nA所属成员的身份证号："); scanf("%s", id); getchar();

T = SearchAVL(F->HeadNode->lchild, id, 2);

if (T == NULL)

{

printf("没有找到成员%s！", id);

getchar(); break;

}

re = 0;

while (re == 0)

{

printf("A为该结点的哪一集合：\n1.好友集\t2.粉丝集\t3.关注人集\t4.兴趣爱好集\n"); scanf("%d", &choice); getchar();

switch (choice)

{

case 1:case 2:case 3:case 4:re = 1; agg1 = aggregate(T, choice); break;

default:printf("输入错误，请重新输入");

}

}

printf("形如A集合与B集合相同，则：\nB所属成员的身份证号："); scanf("%s", id); getchar();

T1 = SearchAVL(F->HeadNode->lchild, id, 2);

if (T1 == NULL)

{

printf("没有找到成员%s！", id);

getchar(); break;

}

re = 0;

while (re == 0)

{

printf("B为该结点的哪一集合：\n1.好友集\t2.粉丝集\t3.关注人集\t4.兴趣爱好集\n"); scanf("%d", &choice2); getchar();

switch (choice2)

{

case 1:case 2:case 3:case 4:re = 1; agg2 = aggregate(T1, choice2); break;

default:printf("输入错误，请重新输入");

}

}

if (agg1 == NULL)

printf("集合A未初始化，无法进行判断！");

else if (agg2 == NULL)

printf("集合B未初始化，无法进行判断！");

if (set\_equal(agg1->lchild, agg2->lchild) == OK)

printf("所输入的A集合和B集合相同！");

else

printf("所输入的A集合和B集合不相同！");

}

getchar();

break;

case 18:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("该活动树为空树，初始化失败！");

else

{

printf("请输入所要初始化的成员的身份证号："); scanf("%s", id); getchar();

T = SearchAVL(F->HeadNode->lchild, id, 2);

if (T == NULL)

printf("没有找到成员%s！", id);

else

{

re = 0;

while (re == 0)

{

printf("选择所要操作的对象：\n1.好友集\t2.粉丝集\t3.关注人集\t4.兴趣爱好集\n"); scanf("%d", &choice); getchar();

switch (choice)

{

case 1:case 2:case 3:case 4:re = 1; break;

default:printf("输入错误，请重新");

}

}

if (aggregate\_init(&T, choice) == OK)

printf("初始化集合成功！");

else

printf("初始化集合失败！");

}

}

getchar();

break;

case 19:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("该活动树为空树，判断失败！");

else

{

printf("请输入所要操作的成员的身份证号："); scanf("%s", id); getchar();

T = SearchAVL(F->HeadNode->lchild, id, 2);

if (T == NULL)

{

printf("没有找到成员%s！", id);

getchar(); break;

}

re = 0;

while (re == 0)

{

printf("请输入所要操作的集合：1.好友集 2.粉丝集 3.关注人集 4.兴趣爱好集 0.Exit\n"); scanf("%d", &choice); getchar();

switch (choice)

{

case 1:case 2:case 3:case 4:

agg1 = aggregate(T, choice);

if (agg1 == NULL)

{

printf("集合未初始化，不能进行操作！");

re = 1; break;

}

operate\_aggregate(F->HeadNode->lchild,T, choice);

break;

case 0:re = 1; printf("已退出成功！"); break;

default:printf("输入错误，");

}

}

}

getchar();

break;

case 20:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("该活动树为空树，判断失败！");

else

{

printf("A集合：所属成员的身份证号："); scanf("%s", id); getchar();

T = SearchAVL(F->HeadNode->lchild, id, 2);

if (T == NULL)

{

printf("没有找到成员%s！",id);

getchar(); break;

}

printf("B集合：所属成员的身份证号："); scanf("%s", id); getchar();

T1 = SearchAVL(F->HeadNode->lchild, id, 2);

if (T1 == NULL)

{

printf("没有找到成员%s！", id);

getchar(); break;

}

re = 0;

while (re == 0)

{

printf("请选择操作：1. common\_friends 2. common\_fans 3. common\_concerns 4.common\_hobbies 0.exit\n"); scanf("%d", &choice); getchar();

switch (choice)

{

case 1: case 2:case 3:case 4:

agg1 = aggregate(T, choice); agg2 = aggregate(T1, choice);

search\_common(agg1, agg2); printf("\n");

break;

case 0:re = 1; printf("已成功退出！"); break;

default:printf("输入错误，");

}

}

}

getchar();

break;

case 21:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("该活动树为空树，判断失败！");

else

{

printf("A集合：所属成员的身份证号："); scanf("%s", id); getchar();

T = SearchAVL(F->HeadNode->lchild, id,2);

if (T == NULL)

printf("没有找到成员%s，查找二度好友失败！",id);

else if (T->data.friends == NULL)

printf("好友集合未初始化，查找二度好友失败！");

else

{

second\_friends(F->HeadNode->lchild, T);

printf("\n查找完毕！");

}

}

getchar();

break;

case 22:

if (head == NULL)

printf("当前没有活动树！\n");

else

{

j = 0;

Traverse(head);

printf("请输入要切换的AVL树的名字："); scanf("%s", name); getchar();

F1 = ExAVL(head, name);

if (F1 == NULL)

printf("输入的名字不存在，切换失败！\n");

else

{

F = F1;

printf("切换成功！");

}

}

getchar();

break;

case 23:

if (SaveData(F) == OK)

printf("数据写入文件成功！");

else

printf("数据写入文件失败！");

getchar();

break;

case 24:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild != NULL)

printf("当前活动树中已存有数据，不能录入数据！");

else if (LoadData(F) == OK)

printf("录入数据成功！");

else

printf("录入数据失败！");

getchar();

break;

case 25:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild != NULL)

printf("当前活动树中已存有数据，不能随机生成数据！");

else

{

printf("请输入数据规模："); scanf("%d", &choice); getchar();

if (random(F->HeadNode, choice) == OK)

printf("随机生成成功！");

else

printf("随机生成失败！");

}

getchar();

break;

default:

printf("功能选择是无效的输入！\n");

getchar();

break;

}

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：random

函数参数：当前AVL树的头指针，数据规模gross

函数功能：随机生成AVL树的成员

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int random(BSTNode \*T, int gross)

{

srand(time(NULL));

int i=0,id;

TElemType data;

FILE \*fp=NULL;

printf("正在随机生成数据，请稍等。。。\n");

char name[1000][20];//存储从文件里读入的名称

if ((fp = fopen("name.txt", "r")) == NULL)

{

printf("name.txt打开失败！");

return ERROR;

}

while (!feof(fp))

{

fscanf(fp, "%s", name[i]);

if (strcmp(name[i], "#") == 0)break;

i++;

}

for (; i < 1000; i++) strcpy(name[i], "#");

for (i = 1; i <= gross; i++)

{

id = rand() % gross + 1;

sprintf(data.id, "%d", id);

strcpy(data.name,name[i-1]);

data.friends = data.concern= data.fan =data.hobby = NULL;

if (InsertAVL(T, data) == ERROR)i--;

}

i = 0;

if ((fp = fopen("hobby.txt", "r")) == NULL)

{

printf("hobby.txt打开失败！");

return ERROR;

}

char hobby[60][20];

while (!feof(fp))

{

fscanf(fp, "%s", hobby[i]);

if (strcmp(hobby[i], "#") == 0)break;

i++; if (i >= 60)break;

}

for (; i < 60; i++) strcpy(hobby[i], "#");

random1(T, T->lchild, gross, hobby);

fclose(fp);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：random1

函数参数：当前AVL树的遍历指针F，当前结点T，数据规模gross，爱好集合hobby

函数功能：随机生成AVL树的成员

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int random1(BSTNode \*F,BSTNode \*T, int gross,char hobby[60][20])

{

if (T == NULL)

return OK;

int j,k; char id[20];

k = rand() % (gross/2);

if (T->data.friends == NULL)

aggregate\_init(&T, 1);

for (int i = 1; i <=k; i++)

{

j = rand() % gross + 1;

sprintf(id, "%d", j);

aggregate\_insert(F->lchild,T->data.friends,1,id);

}

k = rand() % (gross/2);

if (T->data.fan == NULL)

aggregate\_init(&T, 2);

for (int i = 1; i <= k; i++)

{

j = rand() % gross + 1;

sprintf(id, "%d", j);

aggregate\_insert(F->lchild, T->data.fan, 2, id);

}

k = rand() % (gross/2);

if (T->data.concern == NULL)

aggregate\_init(&T, 3);

for (int i = 1; i <= k; i++)

{

j = rand() % gross + 1;

sprintf(id, "%d", j);

aggregate\_insert(F->lchild, T->data.concern, 3, id);

}

k = rand() % (gross/2);

if (T->data.hobby == NULL)

aggregate\_init(&T, 4);

if (k > 60)k= 60;

for (int i = 1; i <= k; i++)

{

j = rand() % 60;

strcpy(id, hobby[j]);

aggregate\_insert(F->lchild, T->data.hobby, 4, id);

}

random1(F,T->lchild, gross,hobby);

random1(F,T->rchild, gross,hobby);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：LoadData

函数参数：文件指针fp，当前AVL树的遍历指针F

函数功能：将指定文件中的内容读入当前AVL树中

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int LoadData(Forest\*F)

{

FILE\*fp1 = NULL, \*fp2 = NULL, \*fp3 = NULL, \*fp4 = NULL, \*fp5 = NULL;

printf("正在录入name数据，请稍等。。。\n");

if ((fp1 = fopen("newname.txt", "r")) == NULL)

{

printf("newname.txt打开失败！");

return ERROR;

}

char id[20]; char name[20];

fscanf(fp1, "%s", id);

fscanf(fp1, "%s", name);

while (!feof(fp1))

{

TElemType data;

strcpy(data.name, name);strcpy(data.id, id);

data.concern = data.fan = data.friends = data.hobby = NULL;

InsertAVL(F->HeadNode, data);

fscanf(fp1, "%s", id);

fscanf(fp1, "%s", name);

}

fclose(fp1);

int num = set\_size(F->HeadNode->lchild);

printf("正在录入friends数据，请稍等。。。\n");

if ((fp2 = fopen("newfriends.txt", "r")) == NULL)

{

printf("newfriends.txt打开失败！");

return ERROR;

}

BSTNode\*T = NULL;

for (int i = 1; i <= num; i++)

{

fscanf(fp2, "%s", id);

fscanf(fp2, "%s", name);

T = SearchAVL(F->HeadNode->lchild, id,2);

while (1)

{

fscanf(fp2, "%s", id);if(strcmp(id, "#") == 0)break;

fscanf(fp2, "%s", name);

aggregate\_init(&T,1);

aggregate\_insert(F->HeadNode->lchild, T->data.friends, 1, id);

}

}

fclose(fp2);

printf("正在录入fan数据，请稍等。。。\n");

if ((fp3 = fopen("newfan.txt", "r")) == NULL)

{

printf("newfan.txt打开失败！");

return ERROR;

}

for (int i = 1; i <= num; i++)

{

fscanf(fp3, "%s", id);

fscanf(fp3, "%s", name);

T = SearchAVL(F->HeadNode->lchild, id, 2);

while (1)

{

fscanf(fp3, "%s", id); if (strcmp(id, "#") == 0)break;

fscanf(fp3, "%s", name);

aggregate\_init(&T,2);

aggregate\_insert(F->HeadNode->lchild, T->data.fan, 2, id);

}

}

fclose(fp3);

printf("正在录入concern数据，请稍等。。。\n");

if ((fp4 = fopen("newconcern.txt", "r")) == NULL)

{

printf("newconcern.txt打开失败！");

return ERROR;

}

for (int i = 1; i <= num; i++)

{

fscanf(fp4, "%s", id);

fscanf(fp4, "%s", name);

T = SearchAVL(F->HeadNode->lchild, id, 2);

while (1)

{

fscanf(fp4, "%s", id); if (strcmp(id, "#") == 0)break;

fscanf(fp4, "%s", name);

aggregate\_init(&T,3);

aggregate\_insert(F->HeadNode->lchild, T->data.concern, 3, id);

}

}

fclose(fp4);

printf("正在录入hobby数据，请稍等。。。\n");

if ((fp5 = fopen("newhobby.txt", "r")) == NULL)

{

printf("newhobby.txt打开失败！");

return ERROR;

}

for (int i = 1; i <= num; i++)

{

fscanf(fp5, "%s", id);

fscanf(fp5, "%s", name);

T = SearchAVL(F->HeadNode->lchild, id, 2);

while (1)

{

fscanf(fp5, "%s", id);if (strcmp(id, "#") == 0)break;

fscanf(fp5, "%s", name);

aggregate\_init(&T,4);

aggregate\_insert(F->HeadNode->lchild, T->data.hobby, 4, id);

}

}

fclose(fp5);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：SaveData2

函数参数：文件指针fp，当前AVL树的结点T

函数功能：将当前AVL树的内容存入指定文件

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int SaveData2(FILE \*fp, BSTNode\*T)

{

if (T) {

SaveData2(fp, T->lchild);

fprintf(fp, "%s %s",T->data.id,T->data.name);

j++; if (j % 20 == 0)fprintf(fp, "\n");else fprintf(fp, " ");

SaveData2(fp, T->rchild);

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：SaveData1

函数参数：文件指针fp，当前AVL树的结点T，集合选项choice

函数功能：将当前AVL树的内容存入指定文件

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int SaveData1(FILE \*fp, BSTNode\*T,int choice)

{

if (T)

{

SaveData1(fp, T->lchild, choice);

fprintf(fp, "%s %s ", T->data.id, T->data.name);

switch (choice)

{

case 1:j = 0;

if (T->data.friends != NULL)

SaveData2(fp, T->data.friends->lchild);

break;

case 2:j = 0;

if (T->data.fan != NULL)

SaveData2(fp, T->data.fan->lchild);

break;

case 3:j = 0;

if (T->data.concern != NULL)

SaveData2(fp, T->data.concern->lchild);

break;

case 4:j = 0;

if (T->data.hobby != NULL)

SaveData2(fp, T->data.hobby->lchild);

break;

}

fprintf(fp, "#\n");

SaveData1(fp,T->rchild,choice);

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：SaveData

函数参数：文件指针fp，当前AVL树的遍历指针F

函数功能：将当前AVL树的内容存入指定文件

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int SaveData(Forest\*F)

{

FILE\*fp1=NULL,\*fp2=NULL, \*fp3=NULL, \*fp4=NULL, \*fp5=NULL;

BSTNode\*T = NULL;

printf("正在保存name数据，请稍等。。。\n");

j = 0;

if ((fp1 = fopen("newname.txt", "w")) == NULL)

{

printf("newname.txt打开失败！");

return ERROR;

}

if(F==NULL)fprintf(fp1, "");

else

{

T = F->HeadNode->lchild;

if (T == NULL)fprintf(fp1, "");

else

{

if (SaveData2(fp1, T) == ERROR)

{

fclose(fp1);

return ERROR;

}

}

}

fclose(fp1);

printf("正在保存friends数据，请稍等。。。\n");

if ((fp2 = fopen("newfriends.txt", "w")) == NULL)

{

printf("newfriends.txt打开失败！");

return ERROR;

}

if (F == NULL)fprintf(fp2, "");

else

{

T = F->HeadNode->lchild;

if (T == NULL)fprintf(fp2, "");

else

{

if (SaveData1(fp2, T, 1) == ERROR)

{

fclose(fp2);

return ERROR;

}

}

}

fclose(fp2);

printf("正在保存fan数据，请稍等。。。\n");

if ((fp3 = fopen("newfan.txt", "w")) == NULL)

{

printf("newfan.txt打开失败！");

return ERROR;

}

if (F == NULL)fprintf(fp2, "");

else

{

T = F->HeadNode->lchild;

if (T == NULL)fprintf(fp3, "");

else

{

if (SaveData1(fp3, T, 2) == ERROR)

{

fclose(fp3);

return ERROR;

}

}

}

fclose(fp3);

printf("正在保存concern数据，请稍等。。。\n");

if ((fp4 = fopen("newconcern.txt", "w")) == NULL)

{

printf("newconcern.txt打开失败！");

return ERROR;

}

if (F == NULL)fprintf(fp2, "");

else

{

T = F->HeadNode->lchild;

if (T == NULL)fprintf(fp4, "");

else

{

if (SaveData1(fp4, T, 3) == ERROR)

{

fclose(fp4);

return ERROR;

}

}

}

fclose(fp4);

printf("正在保存hobby数据，请稍等。。。\n");

if ((fp5 = fopen("newhobby.txt", "w")) == NULL)

{

printf("newhobby.txt打开失败！");

return ERROR;

}

if (F == NULL)fprintf(fp2, "");

else

{

T = F->HeadNode->lchild;

if (T == NULL)fprintf(fp5, "");

else

{

if (SaveData1(fp5, T, 4) == ERROR)

{

fclose(fp5);

return ERROR;

}

}

}

fclose(fp5);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：Traverse

函数参数：森林的头结点head

函数功能：对所有AVL树进行中序遍历（同时输出序号和名称，形如1（李））

返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Traverse(Forest \*head)

{

Forest \*p = head->next;

if (p == NULL)

printf("当前无树存在！\n");

while (p)

{

printf("树的名字：%s 树的成员个数：%d\n", p->name, set\_size(p->HeadNode->lchild));

if (p->HeadNode->lchild == NULL)

printf("该树为空树！");

else

TraverseAVL(p->HeadNode->lchild);

printf("\n\n");

p = p->next;

}

printf("遍历结束，");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：ExAVL

函数参数：森林的头结点head，名称name

函数功能：将活动树切换为名称为name的AVL树

返回值：成功返回指向切换后的树的遍历指针，失败返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Forest \*ExAVL(Forest \*head, char name[20])

{

Forest \*q = head->next;

while (q)

{

if (strcmp(q->name, name) == 0)

break;

q = q->next;

}

return q;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：operate\_aggregate

函数参数：AVL树的根结点root，集合所属结点T，集合选项choice

函数功能：对指定集合进行销毁、增添、删除、修改、查找、遍历、统计规模的操作

返回值：返回OK

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int operate\_aggregate(BSTNode\*root, BSTNode\*T, int choice)

{

BSTNode\*node = NULL,\*T1=NULL;

char s[100];

switch (choice)

{

case 1:T1 = T->data.friends; break;

case 2:T1 = T->data.fan; break;

case 3:T1 = T->data.concern; break;

case 4:T1 = T->data.hobby; break;

}

int op = 1, re = 0; char id[20], id2[20];

while (op)

{

show1();

scanf("%d", &op); getchar();

switch (op)

{

case 0:

printf("已退出对当前集合的操作！\n");

break;

case 1:

if (DestroyAVL(T1->lchild) == OK)

{

switch (choice)

{

case 1:T->data.friends = NULL; break;

case 2:T1 = T->data.fan = NULL; break;

case 3:T1 = T->data.concern = NULL; break;

case 4:T1 = T->data.hobby = NULL; break;

}

Delete(root, T->data.id, choice);

printf("集合销毁成功！\n");

}

else

printf("销毁失败！\n");

break;

case 2:

printf("请输入要插入的成员的身份证号（若多成员则以空格断开）：");

fgets(s, 100, stdin);

for (int i = 0, j = 0; i < 100; i++)

{

if (s[i] == ' ' || s[i] == '\n')

{

id[j] = '\0'; j = 0;

if (aggregate\_insert(root, T1, choice, id) == OK)

printf("成员%s插入成功！\n", id);

else

printf("成员%s插入失败！\n", id);

}

else

{

id[j] = s[i]; j++;

}

}

break;

case 3:

printf("请输入要删除的成员的身份证号（若多成员则以空格断开）："); fgets(s, 100, stdin);

for (int i = 0, j = 0; i < 100; i++)

{

if (s[i] == ' ' || s[i] == '\n')

{

id[j] = '\0'; j = 0;

if (aggregate\_remove(root, T1, choice, id) == OK)

printf("成员%s删除成功！\n",id);

else

printf("成员%s不存在，删除失败！\n",id);

}

else

{

id[j] = s[i]; j++;

}

}

break;

case 4:

printf("请输入修改前的成员的身份证号："); scanf("%s", id); getchar();

if (SearchAVL(T1->lchild, id, 2) != NULL)

{

printf("请输入修改后的成员的身份证号："); scanf("%s", id2); getchar();

if (SearchAVL(T1->lchild, id, 2) != NULL)

{

if (aggregate\_remove(root, T1, choice, id) == OK)

{

aggregate\_insert(root, T1, choice, id2);printf("修改成功！\n");

}

}

else

printf("不存在身份证号为%s的成员，修改失败！\n", id2);

}

else

printf("不存在身份证号为%s的成员，修改失败！\n", id);

break;

case 5:

printf("请输入要查找的成员的身份证号："); scanf("%s", id); getchar();

node = SearchAVL(T1->lchild, id, 2);

if (node == NULL)

printf("不存在身份证号为%s的成员，查找失败！\n", id);

else

printf("查找成功！\n身份证号：%s 名称：%s\n", node->data.id, node->data.name);

break;

case 6:

if (T1->lchild == NULL)

printf("集合为空集！");

else

{

j = 0;

printf("先序遍历：\n");

TraverseAVL2(T1->lchild);

j = 0;

printf("\n中序遍历：\n");

TraverseAVL(T1->lchild);

}

printf("\n");

break;

case 7:

printf("该集合中成员个数为：%d。\n", set\_size(T1->lchild));

break;

default:

printf("功能选择是无效的输入！\n");

break;

}

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：show1

函数参数：无

函数功能：显示关于集合操作的伪菜单界面

返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void show1()

{

printf("1. Destroy 2. Insert 3. Delete 4. Revise 5. Search 6. Traverse 7. Size 0. Exit\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：aggregate\_remove

函数参数：当前AVL树的根结点T，所要进行删除的集合所属于的成员结点，集合选项choice，身份证号id

函数功能：将对应集合删除一个身份证号id的成员

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int aggregate\_remove(BSTNode\*T1, BSTNode\*T, int choice, char id[20])

{

BSTNode\*node = NULL;

if (choice == 4)

node = SearchAVL(T->lchild, id, 2);

else

node = SearchAVL(T1, id, 2);

if (node == NULL)

{

printf("没有找到符合条件的结点，");

return ERROR;

}

if (T == NULL)

{

printf("该集合未初始化，");

return ERROR;

}

switch (choice)

{

case 1:

if (DeleteAVL(T, id) == OK)

{

DeleteAVL(node->data.friends, T->data.id);

return OK;

}

else return ERROR;

case 2:

if (DeleteAVL(T, id) == OK)

{

DeleteAVL(node->data.concern, T->data.id);

return OK;

}

else return ERROR;

case 3:

if (DeleteAVL(T, id) == OK)

{

DeleteAVL(node->data.fan, T->data.id);

return OK;

}

else return ERROR;

case 4://删除T的爱好集合中身份证号为id的结点，不需要反向删除

if (DeleteAVL(T, id) == OK)return OK;

}

return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：aggregate\_insert

函数参数：AVL树的根结点，所要进行插入的集合所属于的成员结点，集合选项choice，身份证号id

函数功能：将对应集合插入一个身份证号id的成员

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int aggregate\_insert(BSTNode\*T1, BSTNode\*T, int choice, char id[20])

{

if (strcmp(T->data.id, id) == 0)return ERROR;

BSTNode\*node = NULL;

if (choice == 4)//如果插入的时爱好集合，则将name存为0，在id中存入爱好

{

node = (BSTNode\*)malloc(sizeof(BSTNode));

strcpy(node->data.id, id); strcpy(node->data.name, "0");

node->data.concern = NULL; node->data.friends = NULL;

node->data.fan = NULL; node->data.hobby = NULL;

node->lchild = node->rchild = NULL;

}

else//如果不是，则在AVL树中查找到身份证号为id的结点，以便实现反向添加

{

node = SearchAVL(T1, id, 2);

if (node == NULL)

{

printf("没有找到符合条件的结点，");

return ERROR;

}

}

if (T == NULL)

{

printf("该集合未初始化，");

return ERROR;

}

else if (SearchAVL(T->lchild, id, 2) != NULL)

return ERROR;

//在T的好友集合中插入node.data并在node的好友集合插入T.data

if (InsertAVL(T, node->data) == OK&&InsertAVL2(node, choice, T->data) == OK)

return OK;

return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：InsertAVL2

函数参数：AVL树的头结点，集合选项choice，数据结构data

函数功能：插入一个数据域为data的结点（仅用于完善集合，即在集合未初始化时，自行初始化集合）

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int InsertAVL2(BSTNode\*T1, int choice, TElemType data)

{

BSTNode\*T = NULL;

switch (choice)

{

case 1:

if (T1->data.friends == NULL)//若集合没有初始化，则先初始化

aggregate\_init(&T1, 1);

T = T1->data.friends->lchild;

if (T == NULL)

{

T = (struct BSTNode\*)malloc(sizeof(struct BSTNode));

T->BF = 0; T->data = data;

T->lchild = NULL; T->rchild = NULL;

T1->data.friends->lchild = T;

return OK;

}

break;

case 2:

if (T1->data.concern == NULL)//若集合没有初始化，则先初始化

aggregate\_init(&T1, 3);

T = T1->data.concern->lchild;

if (T == NULL)

{

T = (struct BSTNode\*)malloc(sizeof(struct BSTNode));

T->BF = 0; T->data = data;

T->lchild = NULL; T->rchild = NULL;

T1->data.concern->lchild = T;

return OK;

}

break;

case 3:

if (T1->data.fan == NULL)//若集合没有初始化，则先初始化

aggregate\_init(&T1, 2);

T = T1->data.fan->lchild;

if (T == NULL)

{

T = (struct BSTNode\*)malloc(sizeof(struct BSTNode));

T->BF = 0; T->data = data;

T->lchild = NULL; T->rchild = NULL;

T1->data.fan->lchild = T;

return OK;

}

break;

case 4:return OK;

}

BSTNode\*pre = NULL, \*current = T;

while (current != NULL)

{

if (strcmp(current->data.id, data.id) == 0)

return ERROR;

pre = current;

current = (strcmp(data.id, pre->data.id)>0) ? pre->rchild : pre->lchild;

}

current = (struct BSTNode\*)malloc(sizeof(struct BSTNode)); current->BF = 0; current->data = data;

current->lchild = NULL; current->rchild = NULL;

if (strcmp(data.id, pre->data.id)>0) pre->rchild = current;

else pre->lchild = current;

switch (choice)

{

case 1:

T1->data.friends->lchild = Balance(T);

if (T1->data.friends->lchild != NULL)

return OK;

case 2:

T1->data.concern->lchild = Balance(T);

if (T1->data.concern->lchild != NULL)

return OK;

case 3:

T1->data.fan->lchild = Balance(T);

if (T1->data.fan->lchild != NULL)

return OK;

}

return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：aggregate\_destroy

函数参数：当前进行操作的集合所属的成员结点，集合选项choice

函数功能：销毁当前AVL树

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int aggregate\_destroy(BSTNode\*T, int choice)

{

switch (choice)

{

case 1:

if (T->data.friends == NULL)

return ERROR;

DestroyAVL(T->data.friends->lchild);

T->data.friends = NULL;

break;

case 2:

if (T->data.fan == NULL)

return ERROR;

DestroyAVL(T->data.fan->lchild);

T->data.fan = NULL;

break;

case 3:

if (T->data.concern == NULL)

return ERROR;

DestroyAVL(T->data.concern->lchild);

T->data.concern = NULL;

break;

case 4:

if (T->data.hobby == NULL)

return ERROR;

DestroyAVL(T->data.hobby->lchild);

T->data.hobby = NULL;

break;

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：Delete

函数参数：当前进行操作的AVL树的根结点，集合所属的成员身份证号，集合选项choice

函数功能：删除掉原来集合中所销毁的

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Delete(BSTNode\*F, char id[20], int choice)

{

if (F == NULL)

return OK;

BSTNode\*node = NULL;

switch (choice)

{

case 1://反向删除F结点好友集中的id为T.data.id的成员

if (F->data.friends == NULL)return OK;

DeleteAVL(F->data.friends, id);

break;

case 2://反向删除F结点关注人集中的id为T.data.id的成员

if (F->data.concern == NULL)return OK;

DeleteAVL(F->data.concern, id); break;

case 3://反向删除F结点粉丝集中的id为T.data.id的成员

if (F->data.fan == NULL)return OK;

DeleteAVL(F->data.fan, id); break;

case 4:return OK;//爱好集合不需要反向删除

}

Delete(F->lchild, id, choice); Delete(F->rchild, id, choice);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：aggregate\_init

函数参数：当前进行操作的集合所属的成员结点，集合选项choice

函数功能：初始化集合

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int aggregate\_init(BSTNode\*\*T, int choice)

{

switch (choice)

{

case 1:

if ((\*T)->data.friends == NULL)//初始化好友集合

{

(\*T)->data.friends = (BSTNode\*)malloc(sizeof(BSTNode));

(\*T)->data.friends->BF = 0; (\*T)->data.friends->lchild = NULL; (\*T)->data.friends->rchild = NULL;

strcpy((\*T)->data.friends->data.id, (\*T)->data.id); strcpy((\*T)->data.friends->data.name, (\*T)->data.name);

(\*T)->data.friends->data.hobby = NULL; (\*T)->data.friends->data.friends = NULL;

(\*T)->data.friends->data.fan = NULL; (\*T)->data.friends->data.concern = NULL;

return OK;

}

case 2:

if ((\*T)->data.fan == NULL)//初始化粉丝集合

{

(\*T)->data.fan = (BSTNode\*)malloc(sizeof(BSTNode));

(\*T)->data.fan->BF = 0; (\*T)->data.fan->lchild = NULL; (\*T)->data.fan->rchild = NULL;

strcpy((\*T)->data.fan->data.id, (\*T)->data.id); strcpy((\*T)->data.fan->data.name, (\*T)->data.name);

(\*T)->data.fan->data.hobby = NULL; (\*T)->data.fan->data.friends = NULL;

(\*T)->data.fan->data.fan = NULL; (\*T)->data.fan->data.concern = NULL;

return OK;

}

case 3:

if ((\*T)->data.concern == NULL)//初始化关注人集合

{

(\*T)->data.concern = (BSTNode\*)malloc(sizeof(BSTNode));

(\*T)->data.concern->BF = 0; (\*T)->data.concern->lchild = NULL; (\*T)->data.concern->rchild = NULL;

strcpy((\*T)->data.concern->data.id, (\*T)->data.id); strcpy((\*T)->data.concern->data.name, (\*T)->data.name);

(\*T)->data.concern->data.hobby = NULL; (\*T)->data.concern->data.friends = NULL;

(\*T)->data.concern->data.fan = NULL; (\*T)->data.concern->data.concern = NULL;

return OK;

}

case 4:

if ((\*T)->data.hobby == NULL)//初始化好友集合

{

(\*T)->data.hobby = (BSTNode\*)malloc(sizeof(BSTNode));

(\*T)->data.hobby->BF = 0; (\*T)->data.hobby->lchild = NULL; (\*T)->data.hobby->rchild = NULL;

strcpy((\*T)->data.hobby->data.id, (\*T)->data.id); strcpy((\*T)->data.hobby->data.name, (\*T)->data.name);

(\*T)->data.hobby->data.hobby = NULL; (\*T)->data.hobby->data.friends = NULL;

(\*T)->data.hobby->data.fan = NULL; (\*T)->data.hobby->data.concern = NULL;

return OK;

}

}

return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：aggregate

函数参数：成员结点T和集合选项choice

函数功能：返回所需集合的头指针

返回值：所需集合的头指针

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BSTNode\*aggregate(BSTNode \*T, int choice)

{

switch (choice)

{

case 1:return T->data.friends; break;

case 2:return T->data.fan; break;

case 3:return T->data.concern; break;

case 4:return T->data.hobby; break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：search\_common

函数参数：两个集合的头指针T1、T2

函数功能：输出T1和T2的交集

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int search\_common(BSTNode\*T1, BSTNode\*T2)

{

if (T1 == NULL || T2 == NULL)

{

printf("空集！");

return OK;

}

BSTNode\*T = (struct BSTNode\*)malloc(sizeof(struct BSTNode));

T->BF = 0; T->lchild = NULL; T->rchild = NULL;

//求T1和T2的集合的交集，并生成AVL树，T为AVL树的头结点

set\_intersection(T, T1->lchild, T2->lchild);

j = 0;

if (T->lchild == NULL)

printf("空集！");

else//若集合的交集非空，则遍历输出交集T

TraverseAVL(T->lchild);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：second\_friends

函数参数：遍历指针T1和所要查找的结点T

函数功能：输出和T是二度好友的结点的信息

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int second\_friends(BSTNode\*T1, BSTNode\*T)

{

if (T1 == NULL)

return OK;

second\_friends(T1->lchild, T);

if (set\_member(T->data.friends->lchild, T1->data.id));

else if (T1->data.friends == NULL);

else if (strcmp(T1->data.id, T->data.id) != 0)

{

BSTNode\*node = (struct BSTNode\*)malloc(sizeof(struct BSTNode));

node->BF = 0; node->lchild = NULL; node->rchild = NULL;

//求T1和T2的好友集合的交集，并生成AVL树，T为AVL树的头结点

set\_intersection(node, T->data.friends->lchild, T1->data.friends->lchild);

j = 0;

if (node->lchild != NULL)//若非空则T1为T的二度好友，输出T1的相关信息

printf("身份证号：%s 姓名：%s\n", T1->data.id, T1->data.name);

}

second\_friends(T1->rchild, T);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：set\_subset

函数参数：AVL树的根结点T1、T2

函数功能：T1是否和T2相同

返回值：是返回OK，不是返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int set\_equal(BSTNode\*T1, BSTNode\*T2)

{

//如果T1是T2的子集且T2是T1的子集，则T1与T2相同

if (set\_subset(T1, T2) == OK&&set\_subset(T2, T1) == OK)

return OK;

else return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：set\_subset

函数参数：AVL树的根结点T1、T2

函数功能：求T1是否是T2的子集

返回值：是返回OK，不是返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int set\_subset(BSTNode\*T1, BSTNode\*T2)

{

if (T1 == NULL)return OK;

if (set\_member(T2, T1->data.id)==OK)

{

if (!set\_subset(T1->lchild, T2))return ERROR;

if (!set\_subset(T1->rchild, T2))return ERROR;

return OK;

}

else return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：set\_member

函数参数：AVL树的根结点T，身份证号id

函数功能：求身份证号为id的成员是否输入结点T的某集合

返回值：是返回OK，不是返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int set\_member(BSTNode\*T, char id[20])

{

if(T==NULL)

return ERROR;

if (SearchAVL(T, id,2) != NULL)return OK;

return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：set\_intersection

函数参数：所要进行操作的两个集合所属于的两个成员结点T1、T2、T3

函数功能：将存在于T3且存在于T2的结点添加到T1中

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int set\_intersection(BSTNode\*T1, BSTNode\*T2, BSTNode\*T3)

{

if (T3 == NULL||T2==NULL)

return OK;

if (SearchAVL(T2, T3->data.id,2) != NULL)

{

TElemType data;

strcpy(data.id,T3->data.id); strcpy(data.name, T3->data.name);

data.concern = NULL; data.fan = NULL; data.friends = NULL; data.hobby = NULL;

InsertAVL(T1, data);

}

set\_intersection(T1, T2, T3->lchild);

set\_intersection(T1, T2, T3->rchild);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：set\_intersection1

函数参数：所要进行操作的两个集合头指针T1、T2

函数功能：求两集合的交集

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int set\_intersection1(BSTNode\*T1, BSTNode\*T2)

{

BSTNode\*T = (struct BSTNode\*)malloc(sizeof(struct BSTNode));

T->BF = 0; T->lchild = NULL; T->rchild = NULL;

if (T1 == NULL || T2 == NULL)return ERROR;

set\_intersection(T, T1->lchild, T2->lchild);

j = 0;

if (T->lchild == NULL)

printf("交集为空集！");

else

TraverseAVL(T->lchild);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：set\_union

函数参数：所要进行操作的两个集合所属于的两个成员结点T1、T2

函数功能：将存在于T2而不存在于T1的结点添加到T1中

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int set\_union(BSTNode\*T1, BSTNode\*T2)

{

if (T2 == NULL)

return OK;

if (SearchAVL(T1, T2->data.id,2) == NULL)

{

TElemType data;

strcpy(data.id, T2->data.id); strcpy(data.name, T2->data.name);

data.concern = NULL; data.fan = NULL; data.friends = NULL; data.hobby = NULL;

InsertAVL(T1, data);

}

set\_union(T1, T2->lchild);

set\_union(T1, T2->rchild);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：copy

函数参数：所要进行操作的两个集合所属于的两个成员结点T1、T2

函数功能：将存在于T2的结点添加到T1中

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void copy(BSTNode\*T1, BSTNode\*T2)

{

if (T2 == NULL)

return;

TElemType data;

strcpy(data.id,T2->data.id); strcpy(data.name, T2->data.name);

data.concern = NULL; data.fan = NULL; data.friends = NULL; data.hobby = NULL;

InsertAVL(T1, data);

copy(T1, T2->lchild);

copy(T1, T2->rchild);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：set\_union1

函数参数：所要进行操作的两个集合头指针T1、T2

函数功能：求两集合的并集

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int set\_union1(BSTNode\*T1, BSTNode\*T2)

{

BSTNode\*T = (struct BSTNode\*)malloc(sizeof(struct BSTNode));

T->BF = 0; T->lchild = NULL; T->rchild = NULL;

if (T1 == NULL || T2 == NULL)return ERROR;

copy(T, T1->lchild);

set\_union(T, T2->lchild);

j = 0;

if (T->lchild == NULL)

printf("并集为空集！");

else

TraverseAVL(T->lchild);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：set\_diffrence

函数参数：所要进行操作的两个集合所属于的两个成员结点T1、T2、T3

函数功能：将存在于T3而不存在于T2的结点添加到T1中

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int set\_diffrence(BSTNode\*T1, BSTNode\*T2, BSTNode\*T3)

{

if (T3 == NULL || T2 == NULL)

return OK;

if (SearchAVL(T2, T3->data.id,2) == NULL)

{

TElemType data;

strcpy(data.id, T3->data.id); strcpy(data.name, T3->data.name);

data.concern = NULL; data.fan = NULL; data.friends = NULL; data.hobby = NULL;

InsertAVL(T1, data);//将data插入到T1中

}

set\_diffrence(T1, T2, T3->lchild);

set\_diffrence(T1, T2, T3->rchild);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：set\_diffrence1

函数参数：所要进行操作的两个集合的头指针T1、T2

函数功能：求两集合的差集

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int set\_diffrence1(BSTNode\*T1, BSTNode\*T2)

{

BSTNode\*T = (struct BSTNode\*)malloc(sizeof(struct BSTNode));

T->BF = 0; T->lchild = NULL; T->rchild = NULL;

if (T1 == NULL || T2 == NULL)return ERROR;

set\_diffrence(T, T1->lchild, T2->lchild); set\_diffrence(T, T2->lchild, T1->lchild);

j = 0;

if (T->lchild == NULL)

printf("差集为空集！");

else

TraverseAVL(T->lchild);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：set\_size

函数参数：当前AVL树的根结点

函数功能：求集合的规模

返回值：成功返回集合规模

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int set\_size(BSTNode\*T)

{

if (T == NULL)

return 0;

else

return set\_size(T->lchild) + set\_size(T->rchild) + 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：TraverseAVL

函数参数：AVL树的根结点

函数功能：对当前AVL树进行中序遍历（同时输出序号和名称，形如1（李））

返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TraverseAVL(BSTNode\*T)

{

if (T)

{

TraverseAVL(T->lchild);

printf("%s（%s）", T->data.name, T->data.id);

j++; if(j % 10 == 0)printf("\n"); else printf(" ");

TraverseAVL(T->rchild);

}

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：TraverseAVL2

函数参数：AVL树的根结点

函数功能：对当前AVL树进行先序遍历（同时输出序号和名称，形如1（李））

返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TraverseAVL2(BSTNode\*T)

{

if (T)

{

printf("%s（%s）", T->data.name, T->data.id);

j++; if(j % 10 == 0 )printf("\n"); else printf(" ");

TraverseAVL2(T->lchild);

TraverseAVL2(T->rchild);

}

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：LL

函数参数：AVL树的根结点

函数功能：插入旋转根的左孩子的左子树而导致失衡的情况需要进行RR旋转

返回值：旋转后新的根结点

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BSTNode\*LL(BSTNode\*root)

{

BSTNode\*newroot = root->lchild;

root->lchild = newroot->rchild;

newroot->rchild = root;

if (newroot->BF == 1)

{

root->BF = 0;

newroot->BF = 0;

}

else

{

root->BF = 1;

newroot->BF = -1;

}

return newroot;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：RR

函数参数：AVL树的根结点

函数功能：插入旋转根的右孩子的右子树而导致失衡的情况需要进行RR旋转

返回值：旋转后新的根结点

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BSTNode\*RR(BSTNode\*root)

{

BSTNode\*newroot = root->rchild;

root->rchild = newroot->lchild;

newroot->lchild = root;

if (newroot->BF == -1)

{

root->BF = 0;

newroot->BF = 0;

}

else

{

root->BF = -1;

newroot->BF = 1;

}

return newroot;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：LR

函数参数：AVL树的根结点

函数功能：插入旋转根的左孩子的右子树而导致失衡的情况需要进行LR旋转

返回值：旋转后新的根结点

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BSTNode\*LR(BSTNode\*root)

{

BSTNode\*root1 = root->lchild;

BSTNode\*newroot = root1->rchild;

root->lchild = newroot->rchild;

root1->rchild = newroot->lchild;

newroot->lchild = root1;

newroot->rchild = root;

switch (newroot->BF) //改变平衡因子

{

case 0:

root->BF = 0;

root1->BF = 0;

break;

case 1:

root->BF = -1;

root1->BF = 0;

break;

case -1:

root->BF = 0;

root1->BF = 1;

break;

}

newroot->BF = 0;

return newroot;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：RL

函数参数：AVL树的根结点

函数功能：插入旋转根的右孩子的左子树而导致失衡的情况需要进行RL旋转

返回值：旋转后新的根结点

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BSTNode\*RL(BSTNode\*root)

{

BSTNode\*root1 = root->rchild;

BSTNode\*newroot = root1->lchild;

root->rchild = newroot->lchild;

root1->lchild = newroot->rchild;

newroot->rchild = root1;

newroot->lchild = root;

switch (newroot->BF)

{

case 0:

root->BF = 0;

root1->BF = 0;

break;

case 1:

root->BF = 0;

root1->BF = -1;

break;

case -1:

root->BF = 1;

root1->BF = 0;

break;

}

newroot->BF = 0;

return newroot;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：DeleteAVL

函数参数：AVL树的头结点，所要删除的结点的身份证号

函数功能：删除名称为name的结点

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int DeleteAVL(BSTNode\*T1, char id[20])

{

if (T1 == NULL)return ERROR;

BSTNode\*T = T1->lchild;

BSTNode\*node = NULL;

node = SearchAVL(T, id, 2);

if (node == NULL||T==NULL)//所要查找的结点不存在或者当前AVL树为空树则返回ERROR

{

return ERROR;

}

BSTNode\*temp1=NULL,\*temp2=NULL;

if (node->lchild != NULL&&node->rchild != NULL)

{

temp1 = node->lchild; temp2 = node;

while (temp1->rchild != NULL)

{

temp2 = temp1;

temp1 = temp1->rchild;

}

if (temp1->lchild == NULL)

{

if (temp2 != node)

{

temp1->lchild = node->lchild; temp1->rchild = node->rchild;

temp2->rchild = NULL;

}

else

temp1->rchild = node->rchild;

}

if (node == T)//如果要删除的结点为根结点，修改根结点

T = temp1;

else

{

BSTNode\*prenode = Parent(T, node->data.id);//调用函数用prenode存储所要删除结点的双亲结点

if (strcmp(temp1->data.id, prenode->data.id) > 0)

prenode->rchild = temp1;

else

prenode->lchild = temp1;

}

node->lchild = NULL; node->rchild = NULL;

free(node);

}

else if (node->lchild == NULL&&node->rchild == NULL)

{

if (node == T)

T = NULL;

else

{

BSTNode\*prenode = Parent(T, node->data.id);

if (strcmp(node->data.id, prenode->data.id) > 0)

prenode->rchild = NULL;

else

prenode->lchild = NULL;

}

}

else if (node->lchild == NULL&&node->rchild != NULL)

{

if (node == T)

T = T->rchild;

else

{

BSTNode\*prenode = Parent(T, node->data.id);

if (strcmp(node->data.id, prenode->data.id) > 0)

prenode->rchild = node->rchild;

else

prenode->lchild = node->rchild;

}

}

else if (node->lchild != NULL&&node->rchild == NULL)

{

if (node == T)

T = T->lchild;

else

{

BSTNode\*prenode = Parent(T, node->data.id);

if (strcmp(node->data.id, prenode->data.id) > 0)

prenode->rchild = node->lchild;

else

prenode->lchild = node->lchild;

}

}

T1->lchild=Balance(T);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：InsertAVL

函数参数：AVL树的头结点，数据结构data

函数功能：插入一个数据域为data的结点

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int InsertAVL(BSTNode\*T1, TElemType data)

{

if (T1 == NULL)

{

printf("未初始化，");

return ERROR;

}

BSTNode\*T = T1->lchild;

if (T == NULL)//插入前为空树

{

T = (struct BSTNode\*)malloc(sizeof(struct BSTNode));

T->BF = 0; T->data = data;

T->lchild = NULL; T->rchild = NULL;

T1->lchild = T;

return OK;

}

BSTNode\*pre = NULL, \*current = T;

while (current != NULL)//查找到所要插入的位置，并用pre保存所要插入位置的双亲结点

{

if (strcmp(current->data.id,data.id)==0)

return ERROR;

pre = current;

current = (strcmp(data.id, pre->data.id)>0) ? pre->rchild : pre->lchild;

}

current = (struct BSTNode\*)malloc(sizeof(struct BSTNode)); current->BF = 0; current->data = data;

current->lchild = NULL; current->rchild = NULL;

if (strcmp(data.id,pre->data.id)>0) pre->rchild = current;

else pre->lchild = current;

T1->lchild = Balance(T);

if (T1->lchild != NULL)

return OK;

else return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：Balance

函数参数：AVL树的根结点

函数功能：将插入新结点后的AVL树调整至平衡

返回值：成功返回新的根结点

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BSTNode \*Balance(BSTNode \*T)

{

BSTNode \*root = NULL;

if (T == NULL)return T;

T->lchild=Balance(T->lchild);//将T的左子树调整至AVL树

T->rchild=Balance(T->rchild);//将T的右子树调整至AVL树

T->BF = Depth(T->lchild) - Depth(T->rchild);//赋值平衡因子

if (T->BF == 0 || T->BF == -1 || T->BF == 1)

return T;

if (T->BF == 2)

{

int BF = T->lchild->BF;

if (BF == -1)

root = LR(T);

else if (BF == 1)

root = LL(T);

else //删除时选用

root = LL(T);

}

else if (T->BF == -2)

{

int BF = T->rchild->BF;

if (BF == -1)

root = RR(T);

else if (BF == 1)

root = RL(T);

else //删除时选用

root = RR(T);

}

return root;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：Depth

函数参数：AVL树的根结点

函数功能：返回AVL树的深度

返回值：返回AVL树的深度

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Depth(BSTNode \*T)

{

int deep = 0;

if (T)

{

int leftdeep = Depth(T->lchild);

int rightdeep = Depth(T->rchild);

deep = leftdeep >= rightdeep ? leftdeep + 1 : rightdeep + 1;

}

return deep;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：Parent

函数参数：AVL树的根结点，序号id

函数功能：寻找双亲结点

返回值：如果查找到返回结点指针，否则返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BSTNode\*Parent(BSTNode \*T, char id[20])

{

if (T == NULL)return NULL;

if ((T->lchild != NULL && strcmp(id, T->lchild->data.id) == 0) || (T->rchild != NULL && strcmp(id, T->rchild->data.id) == 0))

return T;

else {

BSTNode \* temp = NULL;

if (strcmp(id, T->data.id)>0)

temp = Parent(T->rchild, id);

if (strcmp(id, T->data.id)<0)

temp = Parent(T->lchild, id);

return temp;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：SearchAVL

函数参数：当前AVL树的根结点，所要查找的结点的名称，功能选项choice

函数功能：查找名称为name的结点或身份证号为name的结点

返回值：成功返回结点指针，失败返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BSTNode\*SearchAVL(BSTNode\*T, char name[20],int choice)

{

if (T == NULL)

return NULL;

if (choice == 2 && strcmp(T->data.id,name) == 0)//按照ID查找，唯一确定

return T;

if (choice == 1 && strcmp(T->data.name, name) == 0)//按照name查找，不唯一确定

{

printf("请问所要进行操作的是ID为：%s的用户吗？\n", T->data.id);

printf("Y or N?（选择N后将继续进行查找）");

char c; scanf("%c", &c); getchar(); int re = 0;

while (re == 0)

{

switch (c)

{

case 'Y':case 'y':

re = 1;return T; break;

case 'N':case 'n':

re = 1;break;

default:

re = 0;

printf("输入错误，请重新选择Y or N?（选择N后将继续进行查找）"); scanf("%c", &c); getchar();

}

}

}

BSTNode \* temp=NULL;

temp = SearchAVL(T->rchild, name,choice);

if (temp==NULL)

temp = SearchAVL(T->lchild, name,choice);

return temp;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：DestroyAVL

函数参数：当前AVL树的根结点

函数功能：递归销毁当前AVL树

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int DestroyAVL(BSTNode\*T)

{

if (T == NULL) {

return OK;

}

DestroyAVL(T->lchild);

DestroyAVL(T->rchild);

free(T); T = NULL;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：DestroyAVL1

函数参数：整个森林头指针head，遍历指针F

函数功能：销毁当前F所指向的AVL树

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int DestroyAVL1(Forest \*head, Forest \*F)

{

Forest \*p = head;

Forest \*q = p;

while (p != F)

{

q = p;

p = p->next;

}

q->next = p->next;

if (DestroyAVL(p->HeadNode->lchild) == ERROR)//递归销毁AVL树

return ERROR;

free(p->HeadNode);

p->HeadNode = NULL;

free(p);

F = NULL;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：InitAVL

函数参数：整个森林头指针head，遍历指针F

函数功能：初始化一颗空的AVL树

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int InitAVL(Forest \*head, Forest \*\*F)

{

Forest \*q = head;

char name[20];

printf("请输入该AVL树的名字（20个字符以内）："); scanf("%s", name); getchar();

while (q->next)

{

q = q->next;

if (strcmp(q->name, name) == 0)

break;

}

if (strcmp(q->name, name) == 0)

{

printf("该名字已被使用，");

return ERROR;

}

q->next = (Forest \*)malloc(sizeof(Forest));

(\*F) = q->next;

strcpy((\*F)->name, name);

(\*F)->HeadNode = (BSTNode\*)malloc(sizeof(BSTNode));//对当前指针F进行初始化，并令头结点的左孩子结点为AVL树的根结点，右孩子始终指向NULL

(\*F)->HeadNode->BF = 0; (\*F)->HeadNode->lchild = NULL; (\*F)->HeadNode->rchild = NULL;

memset((\*F)->HeadNode->data.id, '0', 20); memset((\*F)->HeadNode->data.name, '0', 20);

(\*F)->HeadNode->data.hobby = NULL; (\*F)->HeadNode->data.friends = NULL;

(\*F)->HeadNode->data.fan = NULL; (\*F)->HeadNode->data.concern = NULL;

(\*F)->next = NULL;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：show

函数参数：遍历指针F

函数功能：显示伪菜单界面

返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void show(Forest \*F)

{

system("cls");

printf("\n\n");

printf("\t\t\t\tMenu for Self-balancing Binary Search Tree\n");

printf("-----------------------------------------------------------------------------------------------------------------------\n");

printf("\t\t\t\t1. InitAVL\t\t2. DestroyAVL\n");

printf("\t\t\t\t3. SearchAVL\t\t4. InsertAVL\n");

printf("\t\t\t\t5. DeleteAVL\t\t6. TraverseAVL\n");

printf("\t\t\t\t7. set\_init\t\t8. set\_destroy\n");

printf("\t\t\t\t9. set\_insert\t\t10. set\_remove\n");

printf("\t\t\t\t11. set\_intersection\t12. set\_union\n");

printf("\t\t\t\t13. set\_diffrence\t14. set\_size\n");

printf("\t\t\t\t15. set\_member\t\t16. set\_subset\n");

printf("\t\t\t\t17. set\_equal\t\t18. init\_aggregate\n");

printf("\t\t\t\t19. operate\_aggregate\t20. search\_common\n");

printf("\t\t\t\t21. second\_friends\t22. ExAVL\n");

printf("\t\t\t\t23. SaveData\t\t24. LoadData\n");

printf("\t\t\t\t25. Random\t\t0. Exit\n");

if (F == NULL)

printf("----------------------------------------------当前没有活动树-----------------------------------------------------------");

else

printf("----------------------------------------------当前对树 %s 进行操作---------------------------------------------", F->name);

printf("\n请选择你的操作[0~25]:");

}