

---

华中科技大学

# 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：数据结构课程设计

专业班级：CS1601

学 号：U201614532

姓 名：吕鹏泽

指导教师：周时阳

报告日期：2018/03/23

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.....	II
1 引言.....	3
1.1 课题背景与意义 .....	3
1.2 国内外研究现状 .....	3
1.3 课程设计的主要研究工作 .....	3
2 系统需求分析与总体设计.....	4
2.1 系统需求分析 .....	4
2.2 系统总体设计 .....	4
2.2.1 存储结构设计.....	4
2.2.2 功能结构设计.....	5
3 系统详细设计.....	9
3.1 有关数据结构的定义 .....	9
3.1.1 平衡二叉树节点设计.....	9
3.1.2 节点数据域设计.....	9
3.1.3 相关常量定义.....	10
3.1.4 各数据项间关系.....	11
3.2 主要算法设计 .....	14
3.2.1 平衡调节函数设计.....	14
3.2.2 平衡二叉树基本运算函数设计.....	16
3.2.3 集合基本运算函数设计.....	22
3.2.4 集合演示函数设计.....	28
3.2.5 人际关系模拟演示函数设计.....	31
4 系统实现与测试.....	33
4.1 系统实现 .....	33
4.2 系统测试 .....	38

4.2.1	AVL 树插入删除测试.....	38
4.2.2	集合函数测试.....	41
4.2.3	人际关系模拟功能测试.....	47
5	总结与展望.....	55
5.1	全文总结 .....	55
5.1	工作展望 .....	55
6	体 会.....	56
	参考文献.....	58
	附录 A 程序头文件源代码 .....	59
	附录 B 程序 C 文件源代码 .....	64

# 1 引言

## 1.1 课题背景与意义

AVL 树是最早出现的数据结构之一，大大提高了访问数据的效率，也为以后数据结构的发展起着开拓性的作用。在计算机科学中，AVL 树是最先发明的自平衡二叉查找树。在 AVL 树中任何节点的两个子树的高度最大差别为一，所以它也被称为高度平衡树。AVL 树的查找、插入和删除在平均和最坏情况下都是  $O(\log n)$ 。增加和删除可能需要通过一次或多次树旋转来重新平衡这个树，因此它在动态查找表中的查找效率非常高，在地理信息处理、医学模型处理以及快速成形等技术中都有广泛应用。

## 1.2 国内外研究现状

AVL 树的理论实现已经成熟，现在国内外研究方向是基于 AVL 树的应用，如基于 AVL 树的 IP 地址自动分配算法、数据存储、磁盘管理等。

## 1.3 课程设计的主要研究工作

实现 AVL 树的插入、删除、查找等基本运算，并应用 AVL 树结构实现集合抽象数据类型，最后用集合模拟微博上的人际关系，提升对数据结构与数据抽象的认识，提高我的综合实践与应用能力。

## 2 系统需求分析与总体设计

### 2.1 系统需求分析

以二叉链表作为 AVL 树的物理结构，实现了基于 AVL 树的增删改查等基本操作，并且用 AVL 树表示集合，然后利用集合模拟表示微博中的各个用户及其朋友集、关注人集、粉丝集、爱好集，通过集合间的操作来模拟人际关系。

### 2.2 系统总体设计

#### 2.2.1 存储结构设计

如图 1-1，以二叉链表作为 AVL 树的物理结构，实现了基于 AVL 树的增删改查等基本操作，并且用 AVL 树表示集合，然后利用集合模拟表示微博中的各个用户及其朋友集、关注人集、粉丝集、爱好集，通过集合间的操作来模拟人际关系。

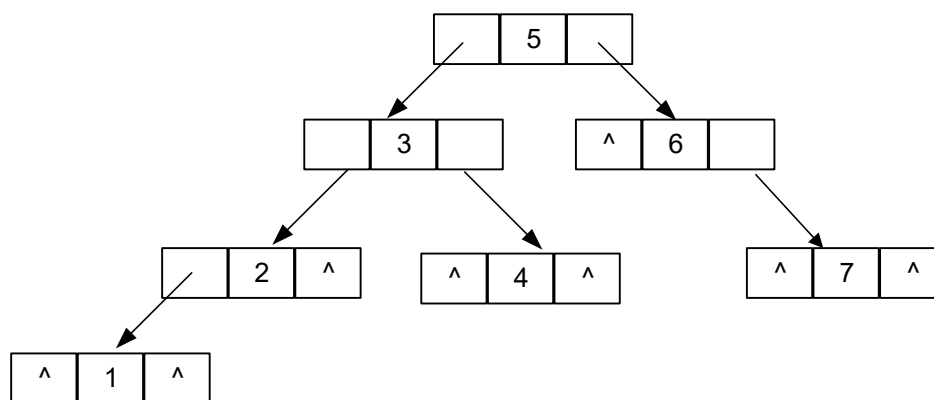


图 1-1 AVL 树物理结构示意图

在集合演示时，我采用了线性表对多集合进行管理，并对集合间运算产生的新集合进行保存，且支持数据加载和保存。多集合管理的物理结构如图 1-2 所示。

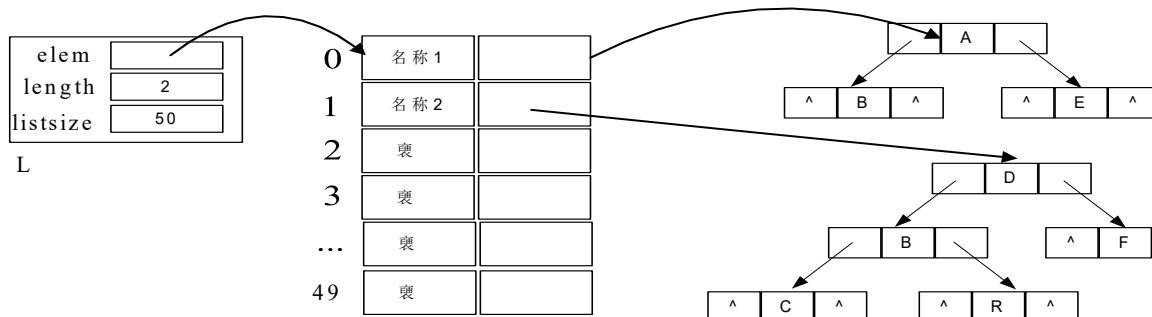


图 1-2 多集合管理物理结构

在实现人际关系模拟的功能中，设有账户集 Account，集合中一个元素记录着一位用户的账号、姓名及其个人关系，而用户的个人关系又由四个集合组成，分别为朋友集、粉丝集、关注人集、爱好集，这四个集合只记录了该用户某一关系对象的 id，在查询该用户信息时，通过在 Account 或 Hobby 集中查询 id 来显示对应 id 的名称。令外设有集合 Hobby，该集合记录了系统中所有的爱好的编号及名称。

## 2.2.2 功能结构设计

程序有三个演示系统，分别为 AVL 树演示系统、集合操作演示和人际关系模拟演示，通过键盘输入确认进入相应的演示系统。如图 1-4、图 1-5 和图 1-6 所示，分别为 AVL 树演示系统、集合操作演示界面和人际关系模拟演示界面，用户可通过键盘输入来选择演示的功能，输入 0 可回退到上一菜单。

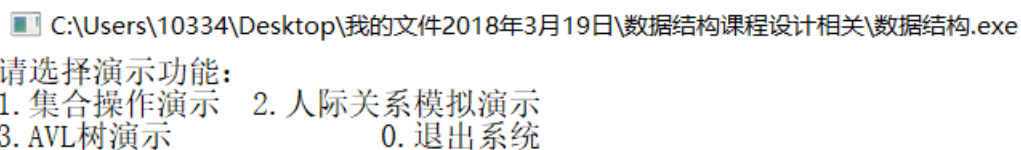


图 1-3 程序载入界面

其中：

(1) AVL 树操作演示包含六个模块，分别为：初始化 AVL 树、插入 AVL 树节点、删除 AVL 树节点、销毁 AVL 树、查询 AVL 树节点、遍历 AVL 树。

(2) 集合操作演示包含十四个模块，分别为：初始化集合、销毁集合、插入集合元素、删除集合元素、集合大小、查找集合元素、集合交、集合并、集合补、



集合子集判断、集合相等判断、集合遍历、集合数据加载、集合数据保存。

(3) 人际关系模拟演示包含十个模块，分别为：添加用户、维护用户信息查询用户信息、显示共同关注、显示共同爱好，显示二度好友、加载数据、保存数据、显示所有用户。

如图 1-4 所示，通过输入数字 0-6 可以选择演示的功能。

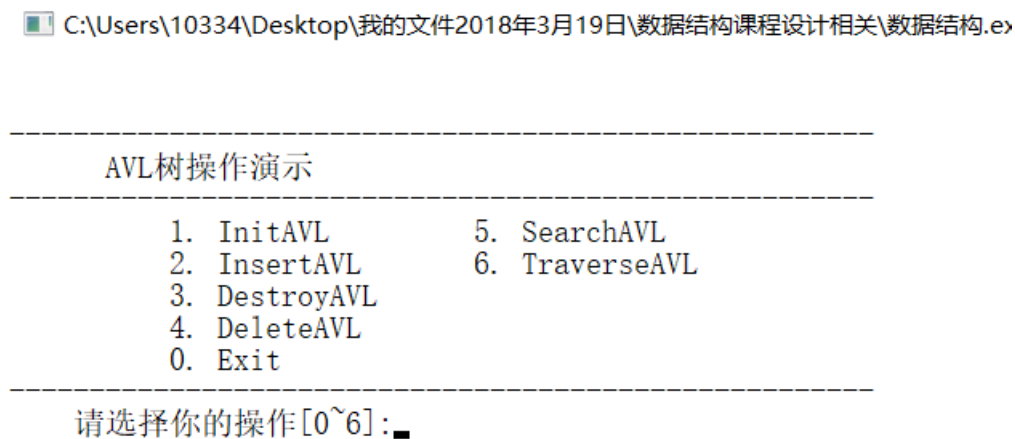


图 1-4 AVL 树演示

如图 1-5 所示，通过输入数字 1-14 可以选择演示的功能。

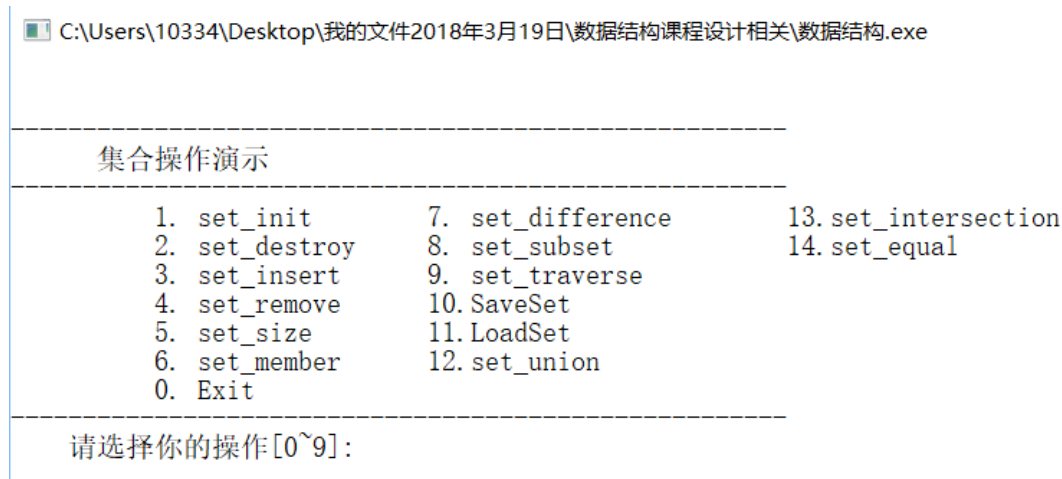


图 1-5 集合操作演示

如图 1-6 所示，通过输入数字 0-10 可以选择演示的功能。

C:\Users\10334\Desktop\我的文件2018年3月19日\数据结构课程设计相关\数据结构.exe

```
-----
人际关系模拟演示
-----
      1. 添加用户          2. 维护用户信息
      3. 查询用户信息      4. 显示共同关注
      5. 显示共同爱好      6. 显示共同好友
      7. 加载数据          8. 保存数据
      9. 显示所有用户      10. 显示二度好友
      0. Exit
-----
请选择你的操作[0~9]:
```

图 1-6 人际关系模拟演示

如图 1-7 所示，演示系统的层次关系可由流程图直观显示出来。

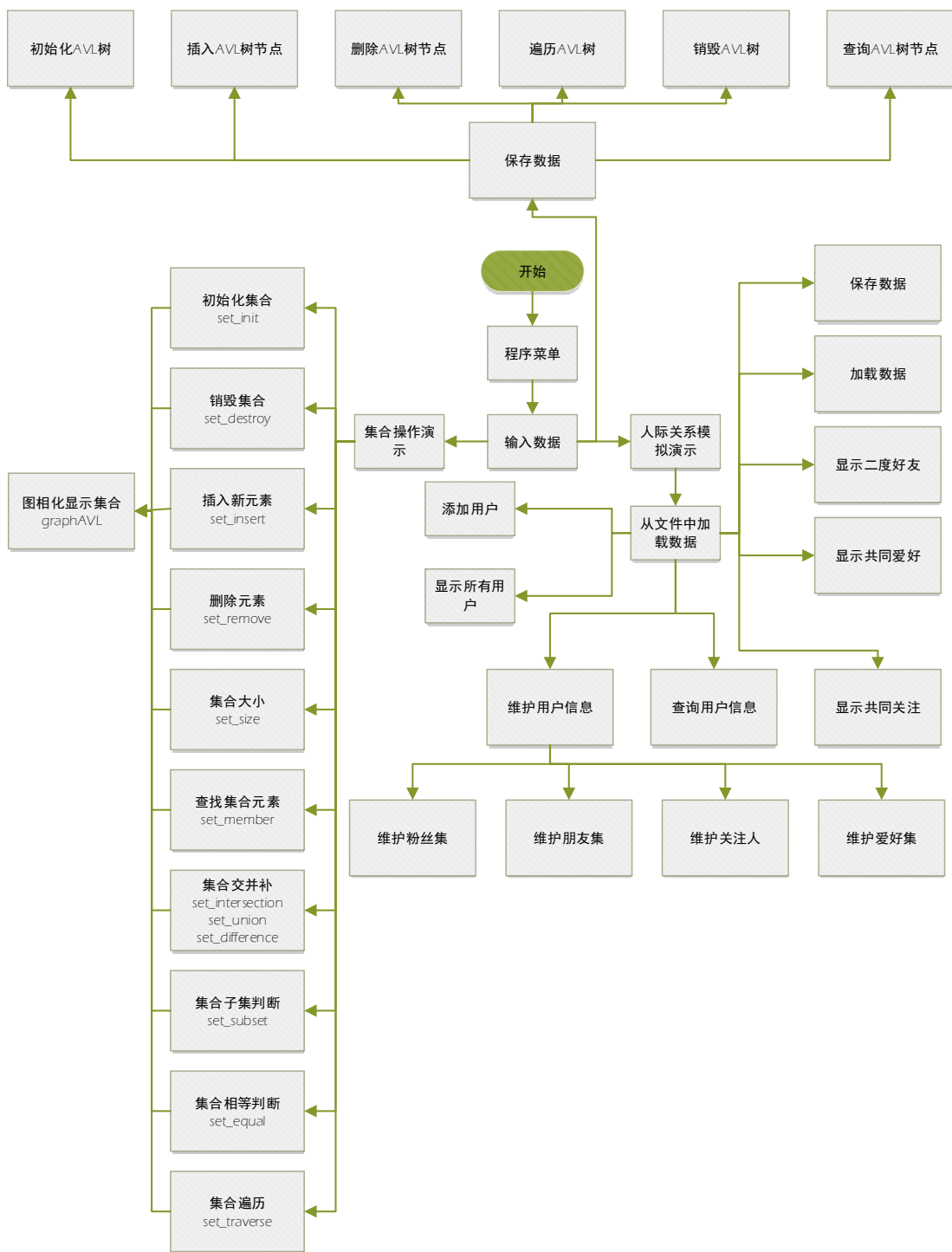


图 1-7 系统总体设计流程图

## 3 系统详细设计

### 3.1 有关数据结构的定义

#### 3.1.1 平衡二叉树节点设计

表 1-1 树节点数据项

数据项	数据类型	中文名称
BSTNode	BSTNode	存储节点(树节点)
BSTree	BSTNode *	存储节点指针(树节点指针)
data	Info	存储节点数据域
bf	int	平衡因子
lchild	BSTNode *	树节点左孩子
rchild	BSTNode *	树节点右孩子

一个存储节点保存一位用户的个人信息，根据用户账号(id)的大小关系建立平衡二叉树。AVL 树节点由数据域 data，指针域 lchild&rchild 以及平衡因子 bf 组成。

#### 3.1.2 节点数据域设计

在集合演示时，我采用了线性表对多集合进行管理，并对集合间运算产生的新集合进行保存，且支持数据加载和保存。线性表 SqList 包含表头指针 pT，表长 listlenth，表大小 listsize，表节点为 SetData 类型，包含集合名称 name，集合指针 pRoot。

而应用层中，数据域 data 存储了用户的账号 id、姓名 name、朋友集 friends、关注人集 follows、粉丝集 fans 及爱好集 hobby。

如表 1-2，列举了各数据项的类型及其表示内容。

表 1-2 节点数据域数据项

数据项	数据类型	中文名称
data	Info	存储节点数据域

name	char[]	姓名
id	int	用户账号
friends	BSTNode *	用户朋友集
follows	BSTNode *	用户关注人集
fans	BSTNode *	用户粉丝集
hobby	BSTNode *	用户爱好集
SqList	struct SqList	管理集合的线性表
SetData	struct setdata	线性表节点
listlenth	int	线性表长
listsize	int	线性表大小
pT	setdata *	线性表头指针
name	char *	集合名称
pRoot	BSTNode *	集合指针

### 3.1.3 相关常量定义

一方面，为了方便程序的维护，另一方面，为了提高程序代码的可读性，我将一些常量定义了出来，包括函数运行状态常量、数据文件名称常量以及其他常量。如表 1-3，列举出了程序中用到的常量。

表 1-3 相关常量定义

Status	int	函数执行状态
TRUE	int	函数正常执行
FALSE	int	函数执行失败
ACCOUNTDATA	char *	账户数据文件
RELATIONDATA	char *	关系数据文件
HOBBYDATA	char *	爱好数据文件
X0	int	AVL 树图形化显示时根节点的位置
LH	int	AVL 树左侧高
EH	int	AVL 树左右等高

RH	int	AVL 树右侧高
L_DATA_NAME	char *	线性表数据文件名称
T_DATA_NAME	char *	集合数据文件名称
OK	int	函数正常执行
OVERFLOW	int	未成功分配空间
FLAT	作为集合存储的结束符	int

函数运行成功返回 TRUE，失败返回 FALSE，xxxDATA 为数据文件名称。

### 3.1.4 各数据项间关系

如表 1-4，列举了程序中所使用的数据项，为了清楚的反应出各个数据项间的关系，我用结构图绘制出了他们的包含关系，如图 1-6 所示。

表 1-4 所有数据项汇总

数据项	数据类型	中文名称
BSTNode	BSTNode	存储节点(树节点)
BSTree	BSTNode *	存储节点指针(树节点指针)
data	Info	存储节点数据域
bf	int	平衡因子
name	char[]	姓名
id	int	用户账号
friends	BSTNode *	用户朋友集
follows	BSTNode *	用户关注人集
fans	BSTNode *	用户粉丝集
hobby	BSTNode *	用户爱好集
lchild	BSTNode *	树节点左孩子
rchild	BSTNode *	树节点有孩子
Status	int	函数执行状态
TRUE	int	函数正常执行

华中科技大学计算机学院数据结构课程设计实验报告

FALSE	int	函数执行失败
ACCOUNTDATA	char *	账户数据文件
RELATIONDATA	char *	关系数据文件
HOBBYDATA	char *	爱好数据文件
X0	int	AVL 树图形化显示时根节点的位置
SqList	struct SqList	管理集合的线性表
SetData	struct setdata	线性表节点
listlenth	int	线性表长
listsize	int	线性表大小
pT	setdata *	线性表头指针
name	char *	集合名称
pRoot	BSTNode *	集合指针
L_DATA_NAME	char *	线性表数据文件名称
T_DATA_NAME	char *	集合数据文件名称
OK	int	函数正常执行
OVERFLOAW	int	未成功分配空间
FLAT	作为集合存储的结束符	int

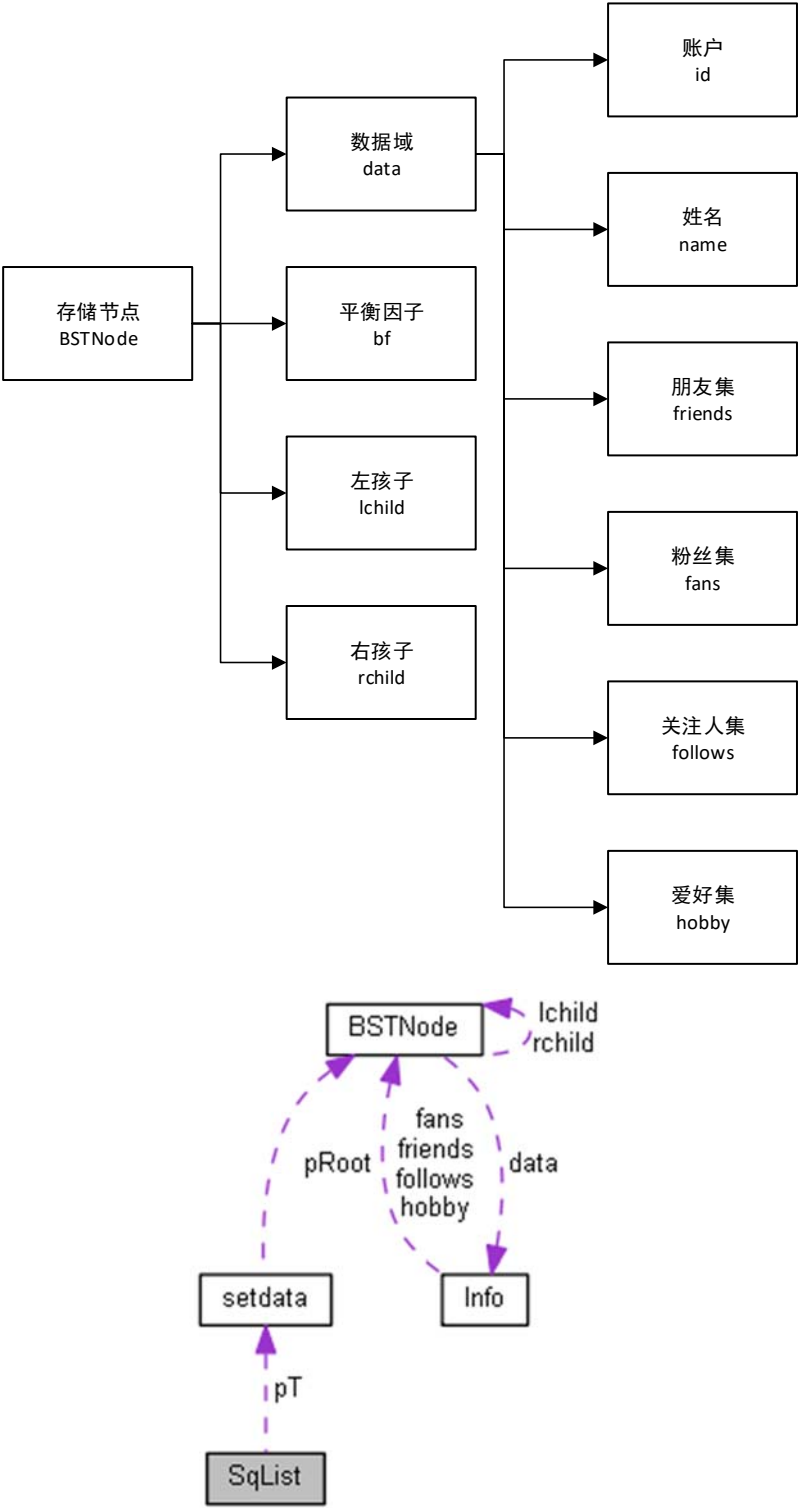


图 1-8 数据项间的关系



## 3.2 主要算法设计

### 3.2.1 平衡调节函数设计

在 AVL 树不平衡时需要对其进行平衡调节，右旋和右平衡调节与左旋和左平衡调节类似。

(1) L\_Rotate(BSTree &p)

初始条件：AVL 树 p 存在且不平衡

操作结果：对以 \*p 为跟的二叉排序树作左旋处理，处理之后 p 指向旋转之前右子树的根节点

算法思想：左旋左靠，右旋右靠，旋转后 \*p 指向原根节点左孩子

时间复杂度：O(1)

流程图：

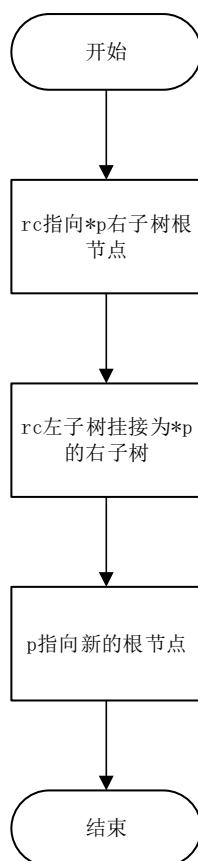


图 1-9 左旋函数流程图

(2) LeftBalance(BSTree &p)

初始条件：AVL 树 p 存在且不平衡

操作结果：对 AVL 进行左平衡旋转处理，结束后  $p$  指向新的根节点

算法思想：根据新插入节点的位置采取单项右旋处理或先左后右处理并修改平衡因子

时间复杂度： $O(1)$

流程图：

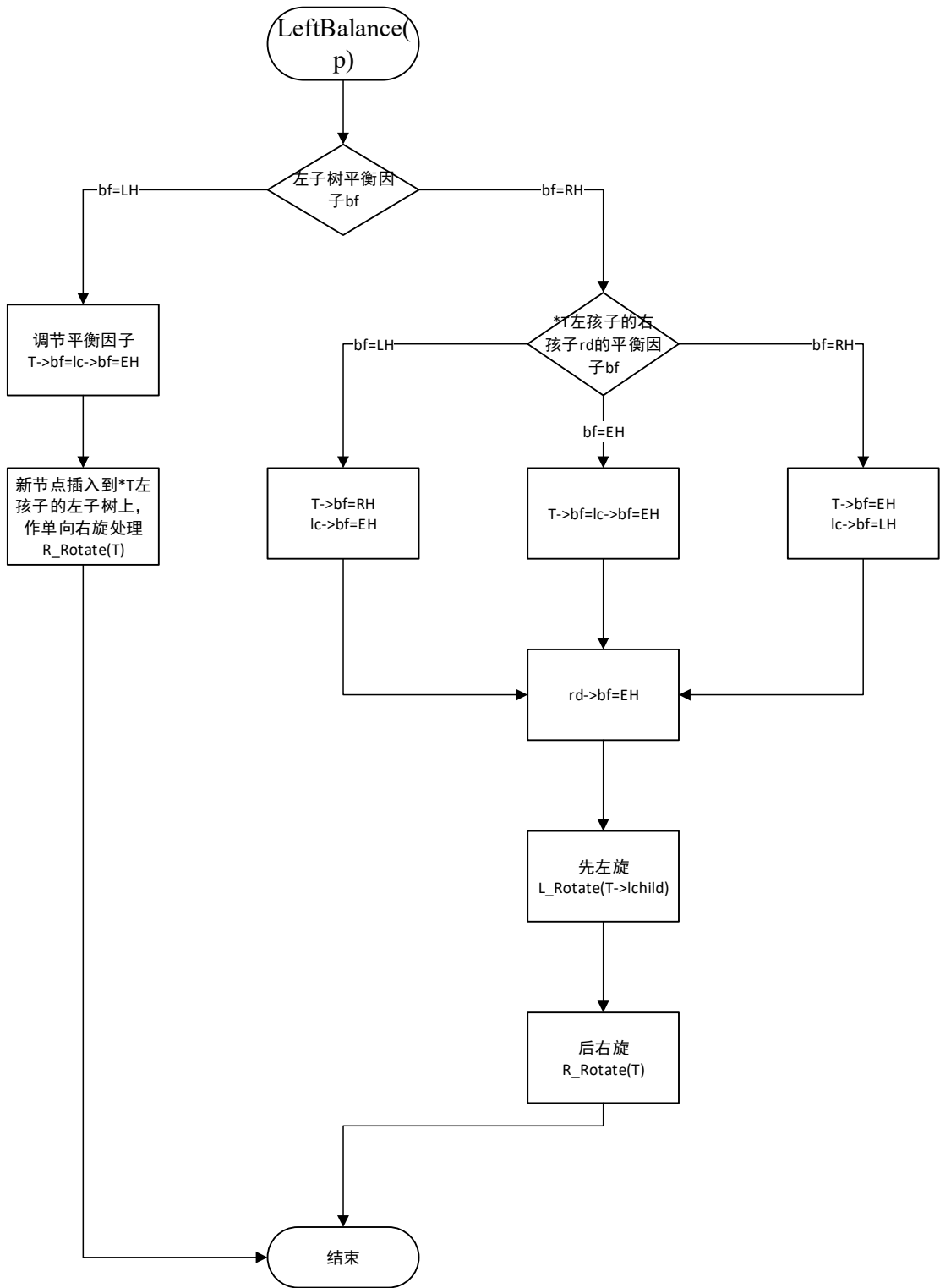


图 1-10 左平衡调节函数流程图

3.2.2 平衡二叉树基本运算函数设计

(1) InitAVL(BSTree &T)

初始条件：AVL 树 T 不存在

操作结果：创建空 AVL 树 T

算法思想：置空树节点指针 T

时间复杂度： $O(1)$

流程图：

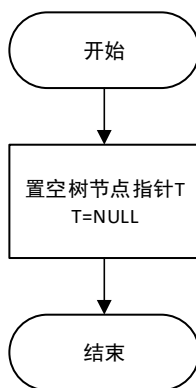


图 1-11 创建 AVL 树流程图

(2) InsertAVL(BSTree &T, Info e, bool &taller)

初始条件：AVL 树 T 存在

操作结果：若 T 不存在与 e 有相同关键字的节点，插入 e，返回 1，否则返回 0

算法思想：根据 e 的关键字在 T 中搜索，若不存在与 e 相同关键字的节点则在叶节点左或右孩子处插入 e，若插入后二叉排序树失去平衡，作平衡化处理

时间复杂度： $O(\log n)$

流程图：

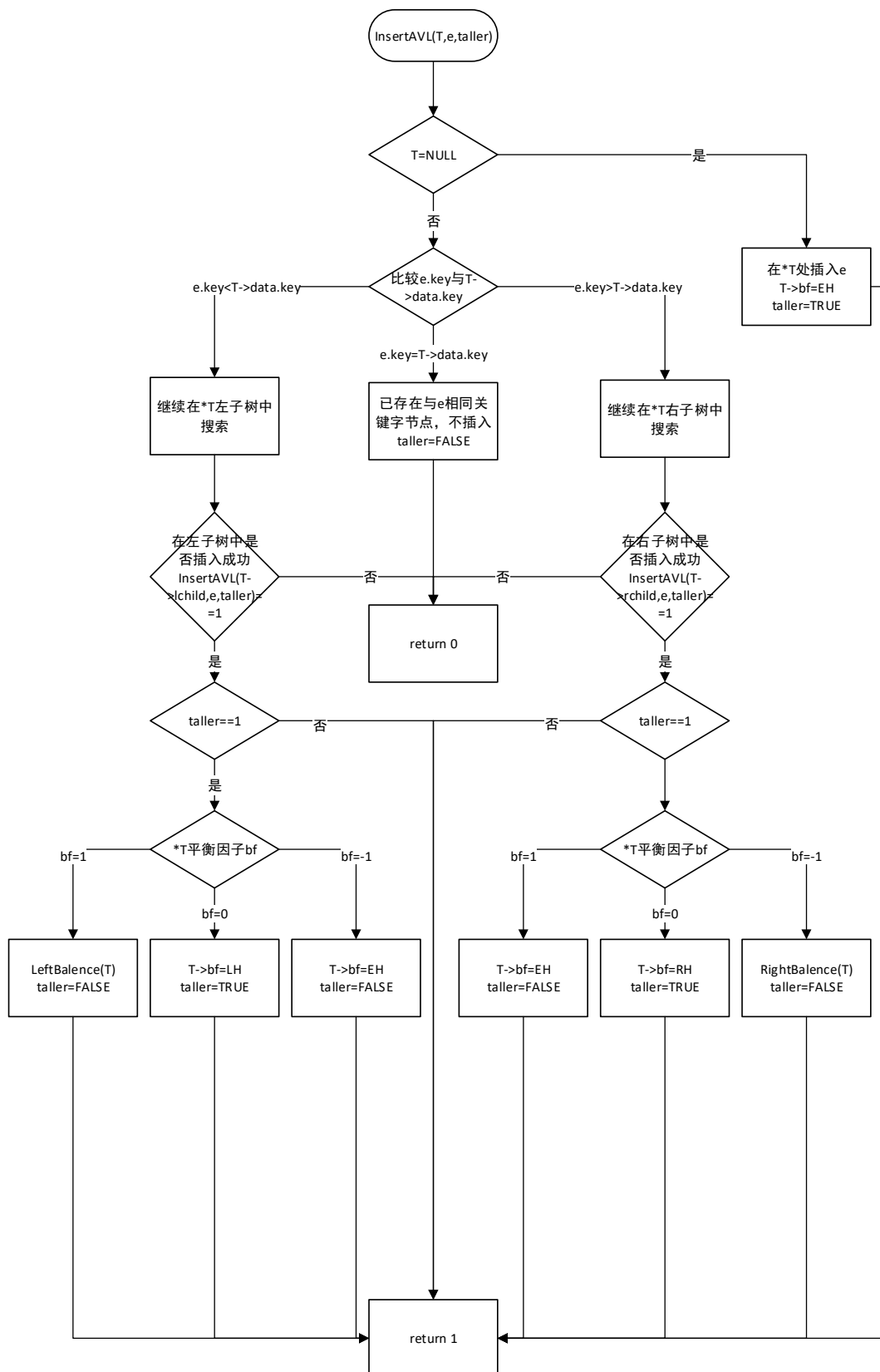


图 1-12 插入函数流程图

(3) DestroyAVL(BSTree &T)

初始条件：AVL 树  $p$  存在

操作结果：销毁 AVL 树  $T$

算法思想：1.若树非空，先销毁  $T$  的左子树，后销毁  $T$  的右子树，最后销毁根。2.对  $T$  的左右子树重复步骤 1。3.若  $T$  为空，则返回。

时间复杂度： $O(n)$

流程图：

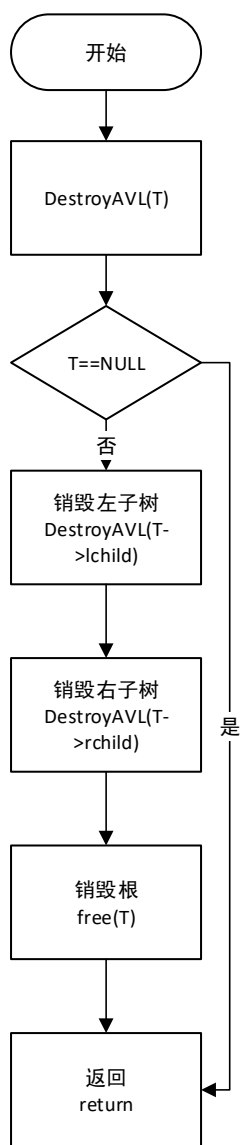


图 1-13 销毁函数流程图

(4) DeleteAVL(BSTree& T, int key, bool& shorter)

初始条件：AVL 树  $p$  存在

操作结果：若  $T$  存在与  $e$  有相同关键字的节点，删除  $e$ ，返回 1，否则返回

0

算法思想：若在平衡的二叉排序树  $t$  中存在和  $e$  有相同关键字的结点，则删除之并返回  $true$ ，否则返回  $false$ 。若因删除而使二叉排序树失去平衡，则作平衡旋转处理。

时间复杂度： $O(\log n)$

流程图：

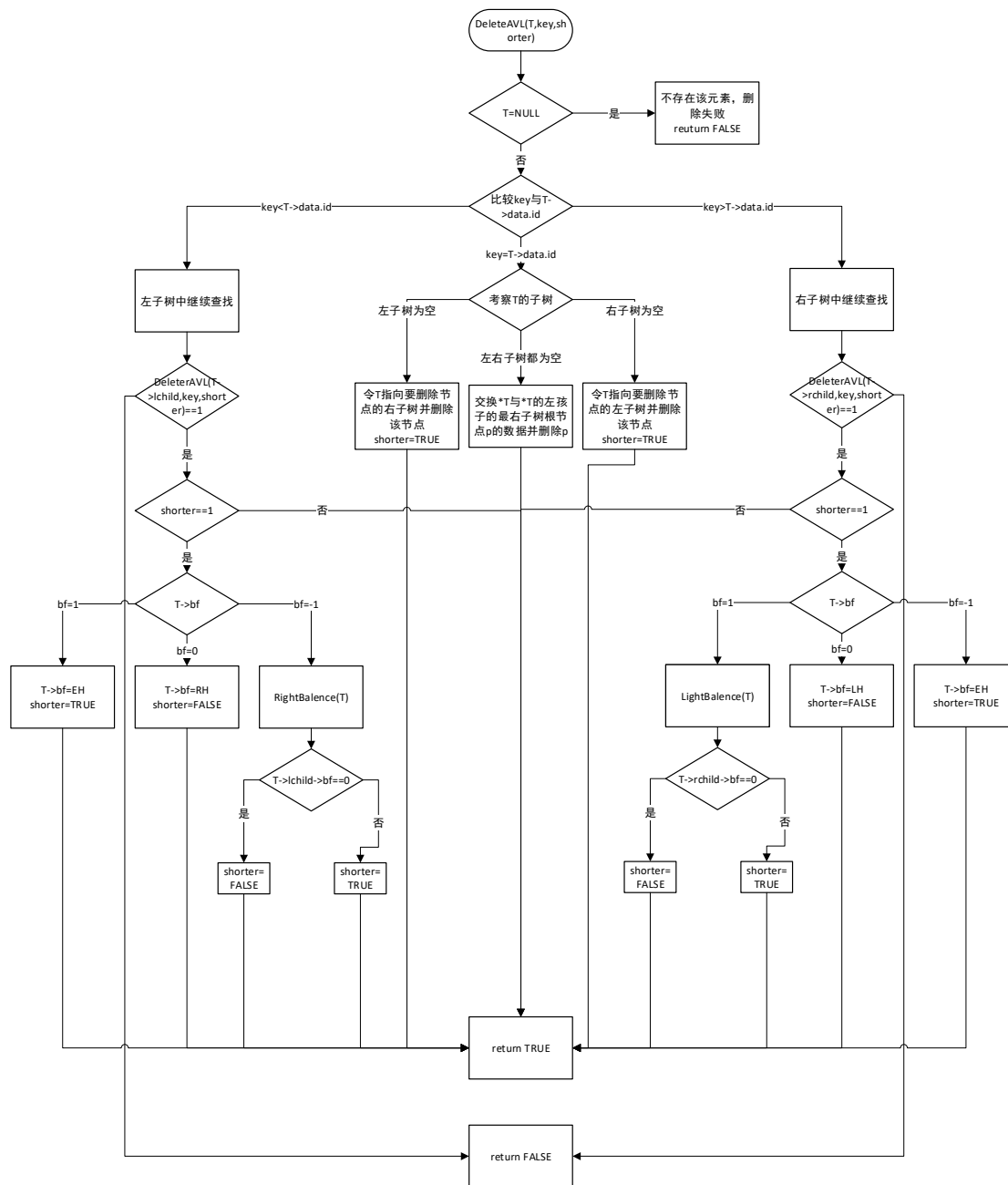


图 1-14 删除函数流程图

(5) SearchAVL(BSTree& T, int key)

初始条件：AVL 树  $p$  存在

操作结果：在 AVL 树  $T$  中查找关键字为  $key$  的节点，找到返回其地址，否则返回 NULL

算法思想：若  $T$  非空，比较  $key$  与  $*T$  的关键字，大于在  $T$  的右子树中搜索，小于在  $T$  的左子树中搜索，等于则返回  $T$ 。若  $T$  为空，则未找到。

时间复杂度： $O(\log n)$

流程图：

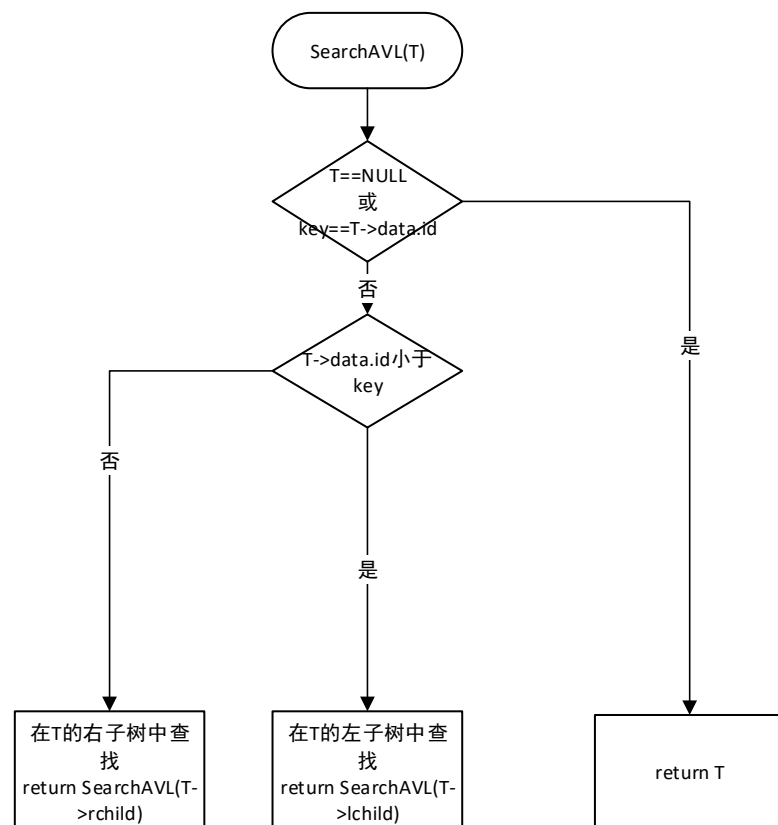


图 1-15 查找函数流程图

#### (6) InOrderTraverse(BSTree T, Status(\*visit)(Info e))

初始条件：AVL 树  $T$  存在， $visit$  是对结点操作的应用函数。

操作结果：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。

算法思想：1.先  $visit()$   $T$  的左子树，其次  $visit()$   $T$  的根，再  $visit()$   $T$  的右子树  
2.对于  $T$  的左右子树，重复步骤 1。

时间复杂度： $O(n)$

流程图：



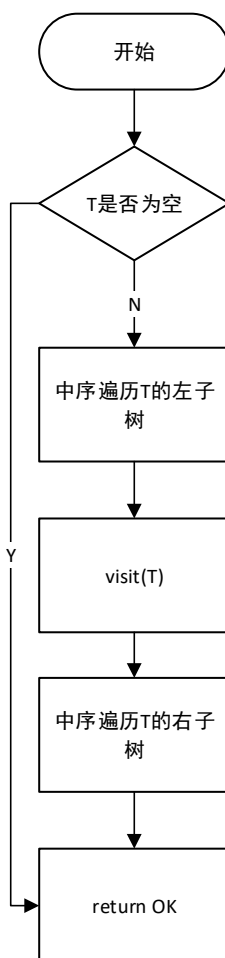


图 1-16 遍历函数流程图

### 3.2.3 集合基本运算函数设计

设集合 T1 大小为 m，T2 大小为 n。T 的大小为 n

(1) set\_intersection(BSTree T1, BSTree T2, BSTree &T3)

初始条件：集合 T1，T2 存在，集合 T3 为空集合

操作结果：T3=T1 ∩ T2

算法思想：遍历 T2 中所有元素，若 T1 中也有，则插入到 T3 中

时间复杂度：O(nlogm)

流程图：

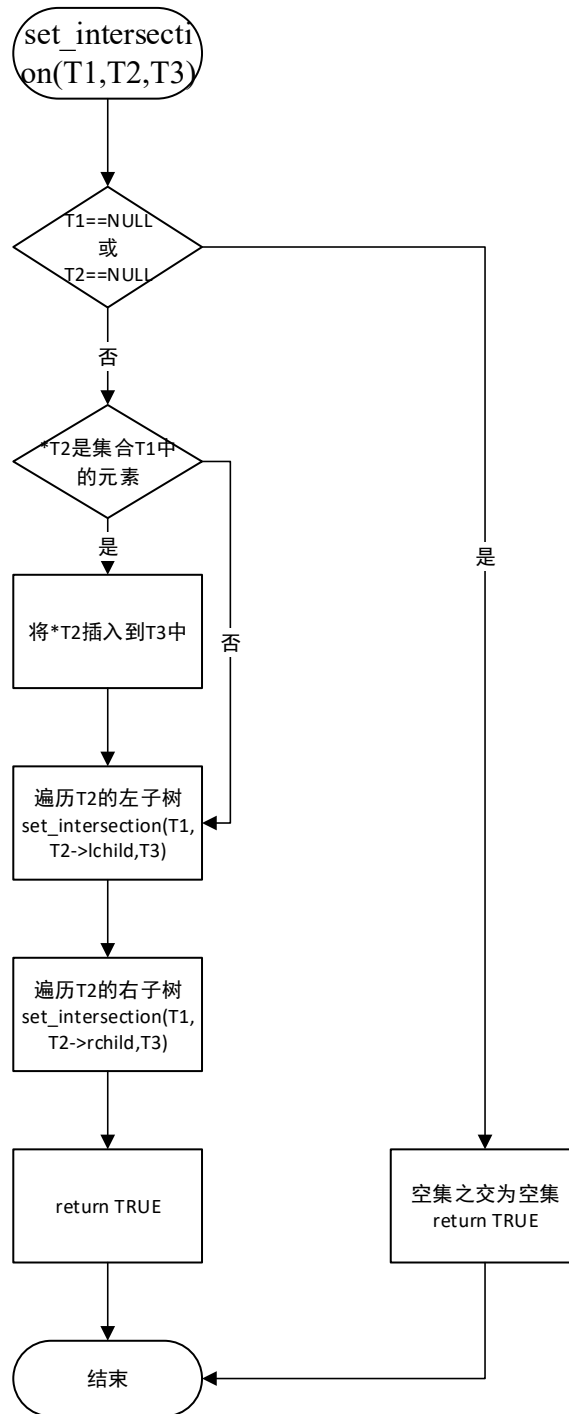


图 1-17 集合交集函数流程图

(2) set\_union(BSTree &T1, BSTree T2)

初始条件：集合 T1，T2 存在

操作结果：将 T2 中的元素并到 T1 中

算法思想：遍历 T2 中所有元素，在 T1 中查询，若不存在，则插入到集合 T1 中

时间复杂度:  $O(n \log m)$

流程图:

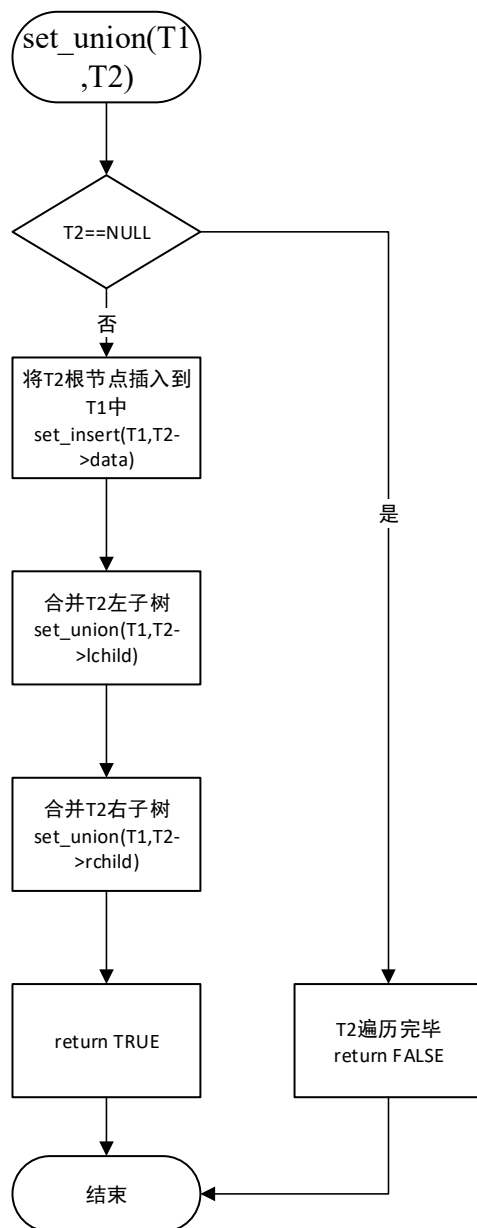


图 1-18 集合并集流程图

(3) `set_difference(BSTree T1, BSTree T2, BSTree &T3)`

初始条件: 集合 T1, T2 存在, 集合 T3 为空

操作结果:  $T3 = T1 - T2$

算法思想: 遍历集合 T1, 将不属于 T2 的元素插入到 T3 中

时间复杂度:  $O(m \log n)$

流程图:

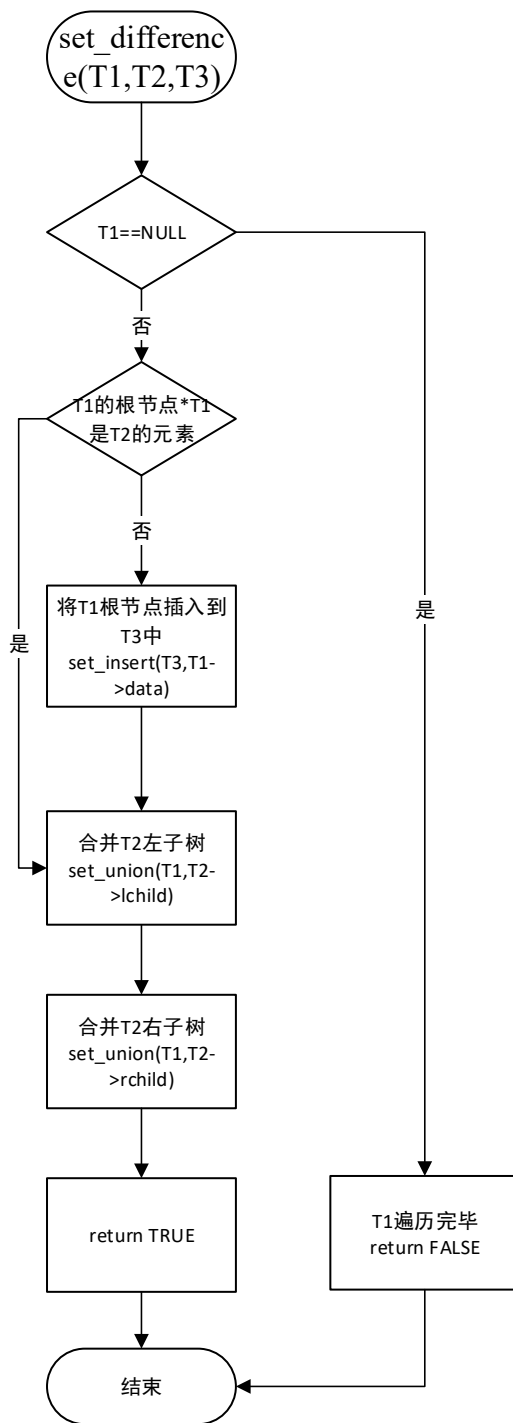


图 1-19 集合差集函数流程图

#### (4) set\_size(BSTree T)

初始条件：AVL 树 p 存在且不平衡

操作结果：对 AVL 进行左平衡旋转处理，结束后 p 指向新的根节点

算法思想：集合 T 的大小=1+T 的左子树大小+T 的右子树大小

时间复杂度：O(n)

流程图：

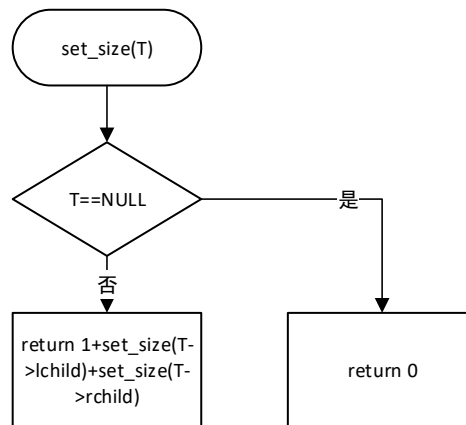


图 1-20 集合大小函数流程图

(5) set\_subset(BSTree T1, BSTree T2)

初始条件：集合 T1, T2 存在

操作结果：判断 T2 是否为 T1 的子集，是返回 TRUE，否返回 FALSE

算法思想：T2 是 T1 的子集当且仅当 T2 根节点是 T1 子集且 T2 左子树是 T1 子集且 T2 右子树是 T1 子集

时间复杂度：O(nlogm)

流程图：

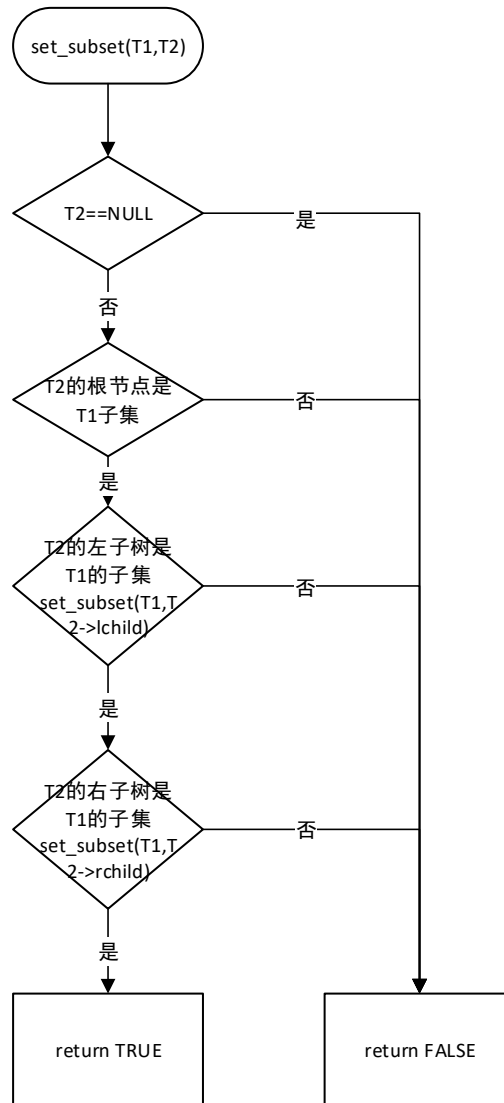


图 1-21 判断集合子集函数流程图

(6) set\_equal(BSTree T1, BSTree T2)

初始条件：集合 T1, T2 存在

操作结果：若 T1=T2 返回 1，否则返回 0

算法思想：T1==T2 当且仅当 T1、T2 互为子集

流程图：

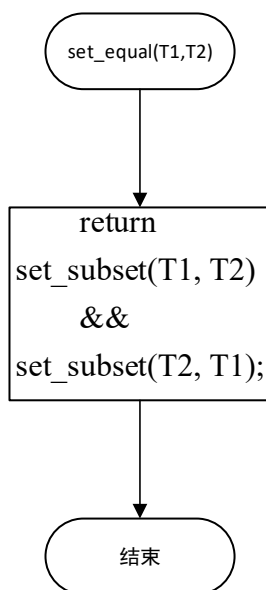


图 1-22 判断集合相等函数流程图

(7) set\_insert(BSTree &T, Info e)

算法思想：直接调用 InsertAVL(T,e,taller)

(8) set\_remove(BSTree T, int key)

算法思想：直接调用 DeleteAVL(T,key,shorter)

(9) set\_member(BSTree T, int key)

算法思想：直接调用 SearchAVL(T,key)

(10) set\_destroy(BSTree &T)

算法思想：直接调用 DestroyAVL(T)

(11) set\_init(BSTree &T)

算法思想：直接调用 InitAVL(T)

### 3.2.4 集合演示函数设计

(1) graphAVL(BSTree T, int x, int l, int y)

初始条件：AVL 树 T 存在

操作结果：在(x,y)处打印 AVL 树 T

算法思想：设屏幕上根节点位置为(x,y)，左右孩子间距为 l，则其左孩子位置为(x-l/2,y+1)，右孩子位置为(x+l/2,y+1)，其左右孩子的孩子的间距为 l/2.

流程图：



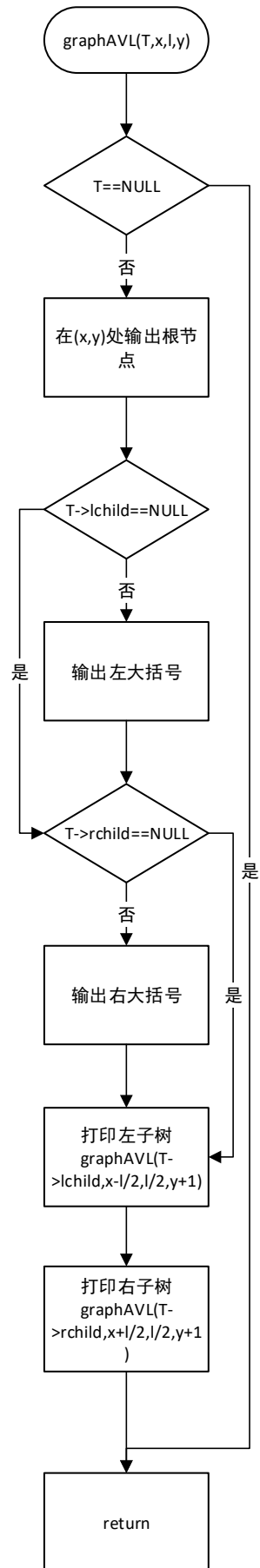


图 1-23 图形化显示 AVL 树函数流层图

### 3.2.5 人际关系模拟演示函数设计

#### (1) common\_friend(BST Account)

说明: common\_follow,common\_hobby 函数思想相同

初始条件: 账户集合 Account 存在。

操作结果: 输入两位用户 id, 给出两位用户的二度好友

算法思想: 求两位用户好友集的交集

时间复杂度:  $O(m\log n)$

流程图:

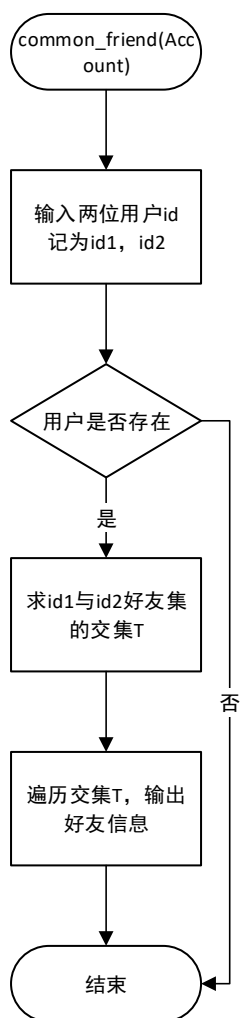


图 1-24 二度好友函数流程图

(2) maintain\_data(BST Account)

初始条件：账户集合 Account 存在。

操作结果：输入用户 id，选择维护的信息并修改

算法思想：查找用户并修改信息

流程图：

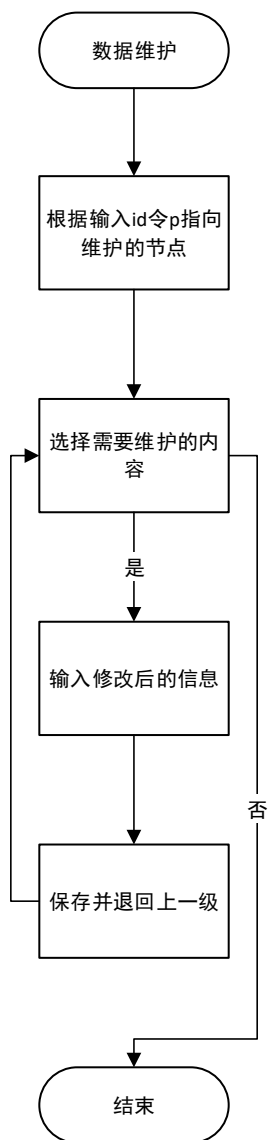


图 1-25 数据维护函数流程说明

## 4 系统实现与测试

### 4.1 系统实现

程序可在 VS017 编译环境下编译成功。系统实现了 AVL 树的基本运算，以 AVL 树结构实现了集合的基本运算，系统支持对多集合操作，支持对集合间运算产生的新集合进行保存，支持对所有集合的数据进行保存。最后，用 AVL 树表示的集合实现了对微博人际关系的模拟，可以查询出单个用户的好友、粉丝、关注人、爱好，也可以查询出两位用户的共同关注、二度好友、共同爱好、共同好友，且支持对这些信息的修改和存储。

程序数据结构的 C 语言实现如下所示：

```
#include<stdbool.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<windows.h>
#define ACCOUNTDATA "account_data"//账户集
#define RELATIONDATA "relation_data"//用户关系（粉丝、朋友、兴趣等）
#define HOBBYDATA "hobby_data"//爱好集
#define L_DATA_NAME "ListData"//多集合管理的线性表信息
#define T_DATA_NAME "SetData"//集合信息
#define LH 1 //左高
#define EH 0 //等高
#define RH -1 //右高
#define TRUE 1
#define FALSE 0
#define X0 50//AVL树图形化显示时根节点的位置
#define OK 1
#define OVERFLOW 0
#define FLAT -100//作为集合存储信息的结束符
#define LIST_INIT_SIZE 50//可管理的最大集合数量

typedef int Status;
typedef struct Info {
    int id;
    char name[20];
    struct BSTNode * friends;//朋友集
    struct BSTNode * fans;//粉丝集
    struct BSTNode * follows;//关注人集
```

```

    struct BSTNode * hobby;//爱好集
}Info;//data域
typedef struct BSTNode {
    Info data;
    int bf;//平衡因子
    struct BSTNode *lchild, *rchild;//左右孩子
}BSTNode, *BSTree;
typedef struct {
    struct setdata * pT;
    int listlenth;
    int listsize;
}SqList;//顺序表，用于管理多集合

typedef struct setdata {
    struct BSTNode * pRoot;
    char name[20];
}SetData;//集合信息(线性表的结点)
/*平衡调节*/
void L_Rotate(BSTree &p);//左旋
void R_Rotate(BSTree &p);//右旋
void LeftBalance(BSTree &T);//左平衡处理
void RightBalance(BSTree &T);//右平衡处理
/*AVL树的基本运算*/
Status InitAVL(BSTree &T);//创建AVL树
Status InsertAVL(BSTree &T, Info e, bool &taller);//插入结点
Status DestroyAVL(BSTree &T);//销毁AVL树
Status DeleteAVL(BSTree &T, int key, bool &shorter);//删除AVL树中结点
BSTNode * SearchAVL(BSTree &T, int key);//在AVL树中查找关键字为key的结点
Status InOrderTraverse(BSTree T, Status(*visit)(Info e));//中序遍历
/*集合的基本运算*/
Status set_init(BSTree &T);//初始化集合
Status set_destroy(BSTree &T);//销毁集合
Status set_insert(BSTree &T, Info e);//插入元素
Status set_remove(BSTree &T, int key);//删除元素
Status set_intersection(BSTree T1, BSTree T2, BSTree &T3);//求T1与T2的交，存在T3中
Status set_union(BSTree &T1, BSTree T2);//求T1与T2的并，结果存在T1中
Status set_difference(BSTree T1, BSTree T2, BSTree &T3);//求集合T1与集合T2的差(T1-T2)，存在集合T3中
int set_size(BSTree T);//集合代销
BSTNode * set_member(BSTree T, int key);//集合成员
Status set_subset(BSTree T1, BSTree T2);//集合子集
Status set_equal(BSTree T1, BSTree T2);//集合相等

```

```

Status set_traverse(BSTree T, Status(*visit)(Info e));//集合遍历
/*集合演示函数*/
void menu1(void);//集合操作演示菜单
void set_display(void);//集合演示
void printKey(BSTree p, int x, int y);//在命令行(x,y)处输出结点p的关键字
void graphAVL(BSTree T, int x, int l, int y);//将AVL树图形化显示 (x,y为根结点
坐标, l为根结点的左右孩子相距距离)
void setxy(short x, short y); //定位光标到 (x, y)
Status visit(Info e);
int SelectSet(SqList L);
int get_cursor_x(void);//获取当前光标横坐标
int get_cursor_y(void);//获取当前光标纵坐标
Status SaveSetData(SqList L);
Status LoadSetData(SqList &L);
Status SaveSet(BSTree T, FILE *fp);
Status LoadSet(BSTree &T, FILE *fp);
/*人际关系模拟函数*/
void menu2(void);//人际关系模拟演示菜单
void apply_display(void);//应用演示
Status destroy_accountset(BSTree &T);//删除账户集合
Status data_input(BSTree T, Info &in);//为集合T中插入的新节点输入data,成功返
回TRUE, 失败返回FALSE
int confirm_save(void);//确认是否保存, 是返回1, 否返回0
int select_display(void);//确认演示功能,集合演示返回1, 应用演示返回2, 退出返
回0
void maintain_data(BSTree &Account,BSTree Hobby);//维护用户信息
void common_follow(BSTree Account);//共同关注
void common_friend(BSTree Account);//共同好友
void second_friend(BSTree Account);//二度好友
void S_friend(BSTree Account, BSTree p, BSTree &T);//添加*p的二度好友, 结果
存在T中
void common_hobby(BSTree Account,BSTree Hobby);//共同爱好
Status save_account(BSTree T, FILE *fp);//保存账户信息
Status save_hobby(BSTree T,FILE *fp);//保存兴趣集合
Status save_relation(BSTree T, FILE *fp);//保存账户关系1
Status save_rela(BSTree T, FILE *fp);//保存账户关系2
Status load_account(BSTree &T, FILE *fp);//加载账户信息
Status load_relation(BSTree &T, FILE *fp);//加载关系信息
Status load_hobby(BSTree &Hobby,FILE *fp);//加载兴趣集合
Status load_data(BSTree &Account, BSTree &Hobby);//加载数据
Status save_data(BSTree Account, BSTree Hobby);//保存数据
void account_data(BSTree Account, BSTree Hobby);//显示用户的关注人、朋友、
粉丝、兴趣
void print_account_data(BSTree Account, BSTree T);//根据T中的用户id在Account

```

中查找相应的用户姓名并显示

```
void proc(void);
```

```
void AVL_display(void);
```

```
void menu3(void);
```

如图 1-26，显示了各函数之间的调用关系

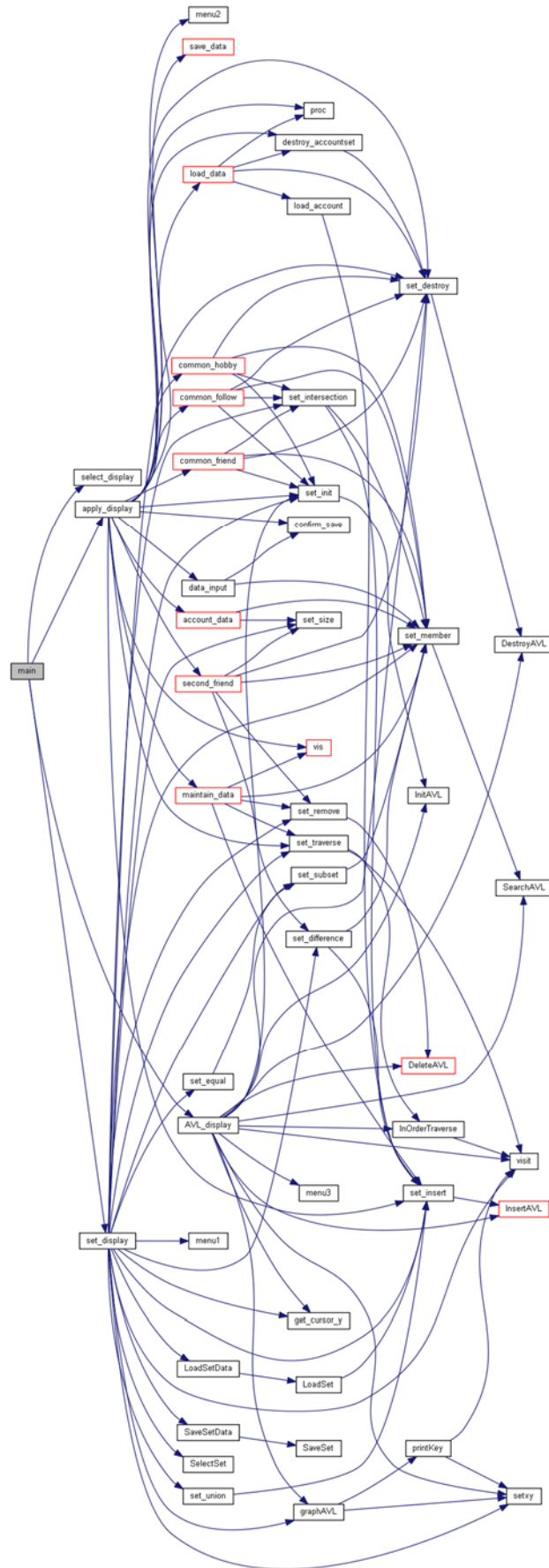




图 1-26 各函数调用关系图示

4.2 系统测试


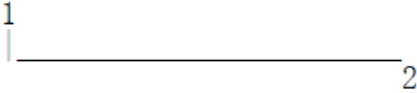
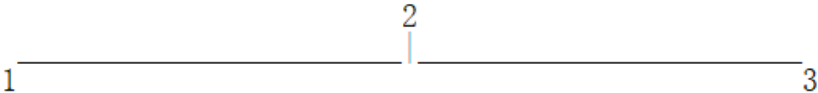
系统测试，英文是 System Testing。是对整个系统的测试，将硬件、软件、操作人员看作一个整体，检验它是否有不符合系统说明书的地方。这种测试可以发现系统分析和设计中的错误。



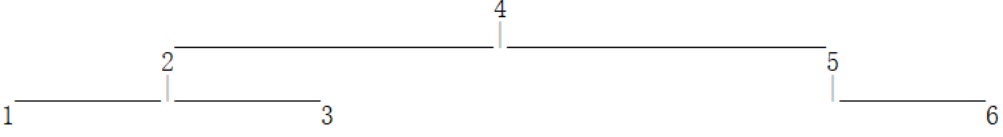
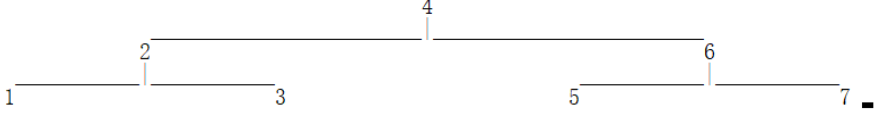
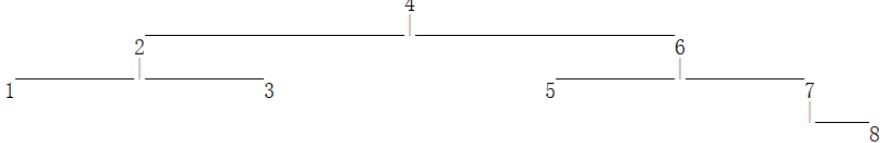
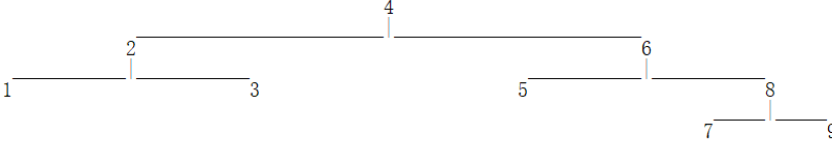
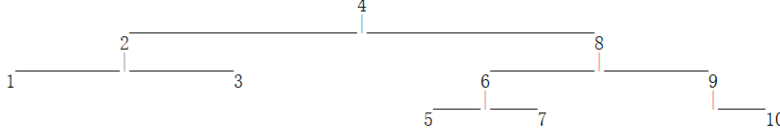
系统测试分为以下几个步骤：制定系统测试计划、设计系统测试用例、执行系统测试、缺陷管理与改错。软件测试工程师常用的测试方法有白盒测试、黑盒测试、自动化测试、静态测试、动态测试、单元测试、集成测试、系统测试、端到端、卸载测试、验收测试、性能测试、安全测试等 30 多种。

4.2.1 AVL 树插入删除测试

首先演示 AVL 树的创建，对一颗空的 AVL 树一次插入 7 个节点，观察其是创建过程。如表 1-5 所示：

表 1-5 AVL 树插入过程演示

插入 1	
插入 2	
插入 3	

插入 4	
插入 5	
插入 6	
插入 7	
插入 8	
插入 9	
插入 10	

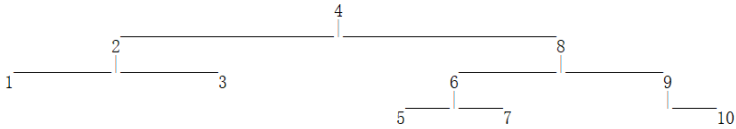
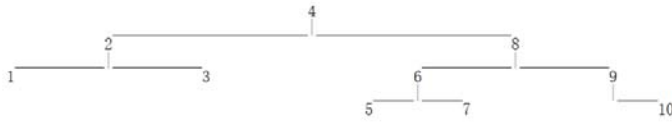
可以看出,创建的 AVL 树始终是平衡的,接下来对 AVL 树的删除进行测试。  
测试结果如表 1-6 所示, 删除过程中 AVL 树始终保持平衡。

表 1-6 AVL 树删除演示

未删除	
删除 10	
删除 9	
删除 4	
删除 1	
删除 2	
删除 3	
删除 5	

最后，对 AVL 树的查找功能进行测试，AVL 树中的元素为{1,2...10}：

表 1-7 查找测试

查找 3	<div>请输入节点 3 3节点在树中</div> 
查找 30	<div>请输入节点 30 节点不存在</div> 

4.2.2 集合函数测试

初始化我创建了 3 个集合，分别记为 T1、T2、T3,集合的元素如表 1-8 所示。

表 1-8 测试集合内容

集合	集合元素
T1	{1,2,3,4,5}
T2	{1,2,3,4,5,6,7,8,9,10}
T3	{8,9,10,11,12}

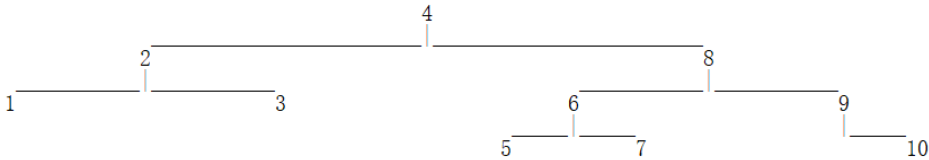
接下来，对集合 T1、T2、T3 进行运算，运算的结果可选择进行保存，可参与接下来的集合运算。如表 1-9 所示，为 T1、T2、T3 三个集合相互运算的理论结果。

表 1-9 测试理论结果

$T2 \cap T3$	$T2 \cup T3$	$T2 - T3$
$T4 = \{8,9,10\}$	$T5 = \{1,2,3,4,5,6,7,8,9,10,11,12\}$	$T6 = \{1,2,3,4,5,6,7\}$
$T1 \cap T3$	$T1 \cup T3$	$T1 - T3$
$T7 = \Phi$	$T8 = \{1,2,3,4,5,8,9,10,11,12\}$	$T9 = \{1,2,3,4,5\}$

6 组数据的测试结果如图 1-25 – 图 1-30 所示，测试结果符合正确。

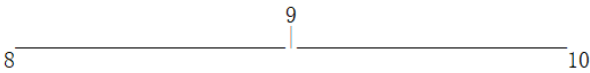
-----  
请选择你的操作[0~9]:13  
请选择第一个集合（运算结果为第一个集合  $\cap$  第二个集合）  
请选择集合(1-3)  
1. T1  
2. T2  
3. T3  
2  
请选择第二个集合请选择集合(1-3)  
1. T1  
2. T2  
3. T3  
3  
输入保存运算结果的集合的名称：  
T4  
集合T4创建成功！  
集合T2: 1 2 3 4 5 6 7 8 9 10



集合T3: 8 9 10 11 12



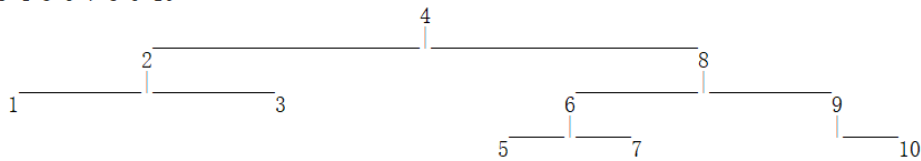
集合T2 $\cap$ 集合T3: 8 9 10



是否保存运算结果数据? 1. 是 0. 否  
1

图 1-25 集合 T2 $\cap$ 集合 T3

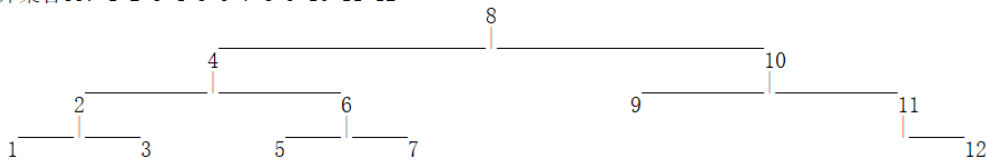
请选择你的操作[0~9]:12  
请选择第一个集合（运算结果为第一个集合 U 第二个集合）  
请选择集合(1-4)  
1. T1  
2. T2  
3. T3  
4. T4  
2  
请选择第二个集合请选择集合(1-4)  
1. T1  
2. T2  
3. T3  
4. T4  
3  
输入保存运算结果的集合的名称:  
T5  
集合T5创建成功!  
集合T2: 1 2 3 4 5 6 7 8 9 10



集合T3: 8 9 10 11 12



集合T2并集合T3: 1 2 3 4 5 6 7 8 9 10 11 12



是否保存运算结果数据? 1. 是 0. 否  
1

图 1-26 集合 T2 U 集合 T3

请选择第一个集合（运算结果为第一个集合 - 第二个集合）

请选择集合 (1-5)

- 1. T1
- 2. T2
- 3. T3
- 4. T4
- 5. T5

2

请选择第二个集合请选择集合 (1-5)

- 1. T1
- 2. T2
- 3. T3
- 4. T4
- 5. T5

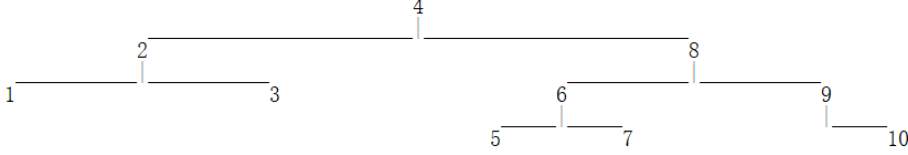
3

输入保存运算结果的集合的名称:

T6

集合T6创建成功!

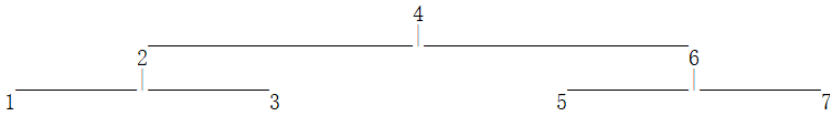
集合T2: 1 2 3 4 5 6 7 8 9 10



集合T3: 8 9 10 11 12



集合T2-集合T3: 1 2 3 4 5 6 7

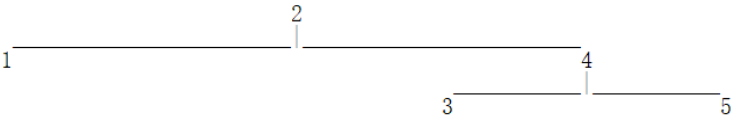


是否保存运算结果数据? 1. 是 0. 否

1

图 1-27 集合 T2-集合 T3

-----  
请选择你的操作[0`9]:13  
请选择第一个集合（运算结果为第一个集合  $\cap$  第二个集合）  
请选择集合(1-6)  
1. T1  
2. T2  
3. T3  
4. T4  
5. T5  
6. T6  
  
1  
请选择第二个集合请选择集合(1-6)  
1. T1  
2. T2  
3. T3  
4. T4  
5. T5  
6. T6  
  
3  
输入保存运算结果的集合的名称:  
T7  
集合T7创建成功!  
集合T1: 1 2 3 4 5



集合T3: 8 9 10 11 12



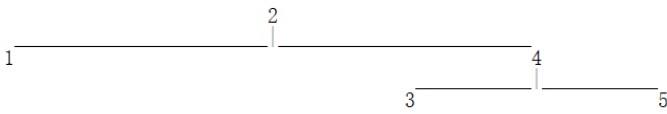
集合T1 $\cap$ 集合T3: 集合为空

是否保存运算结果数据? 1. 是 0. 否  
1

图 1-28 集合 T1 $\cap$ 集合 T3



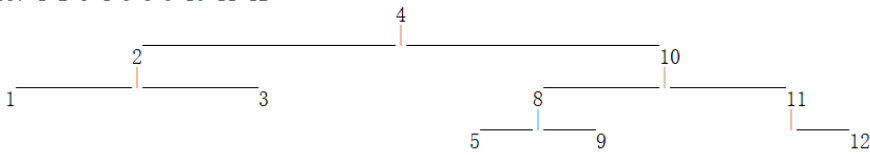
输入保存运算结果的集合的名称:  
T8  
集合T8创建成功!  
集合T1: 1 2 3 4 5



集合T3: 8 9 10 11 12



集合T1并集合T3: 1 2 3 4 5 8 9 10 11 12



是否保存运算结果数据? 1. 是 0. 否

图 1-29 集合 T1U集合 T3

输入保存运算结果的集合的名称:  
T9  
集合T9创建成功!  
集合T1: 1 2 3 4 5



集合T3: 8 9 10 11 12



集合T1-集合T3: 1 2 3 4 5



是否保存运算结果数据? 1. 是 0. 否  
1

图 1-30 集合 T1-集合 T3

经过运算之后各集合的元素如表 1-10 所示:

表 1-10 运算后各集合元素

集合	集合元素
----	------

T1	{1,2,3,4,5}
T2	{1,2,3,4,5,6,7,8,9,10}
T3	{8,9,10,11,12}
T4	{8,9,10}
T5	{1,2,3,4,5,6,7,8,9,10,11,12}
T6	{1,2,3,4,5,6,7,}
T7	$\Phi$
T8	{1,2,3,4,5,8,9,10,11,12}
T9	{1,2,3,4,5}

可以看到，集合间的运算符合理论结果。

接下来对集合子集和相等进行判断，测试结果如表 1-11 所示，测试结果符合理论结果。

表 1-11 集合关系判断

关系判断	理论结果	测试结果	是否正确
T1、T2	T1 是 T2 的子集	集合T1是集合T2的子集合	√
T1、T3	T1 不是 T3 的子集	集合T1不是集合T3的子集合	√
T1、T9	T1=T9	集合T1 = 集合T9	√
T1、T3	T1!=T3	集合T1 != 集合T3	√

#### 4.2.3 人际关系模拟功能测试

首先测试查询功能，选取 id=1 的账户进行测试，测试的理论结果如下表(数字表示相应关系人的 id)：

表 1-12 用户 1 信息查询理论结果

朋友	关注人	粉丝	爱好
10 11 21 24 26 31 38	7 26 28 32 35 48 73	12 16 18 28 37 43 44	27 38 48 65 66 72

华中科技大学计算机学院数据结构课程设计实验报告

39 43 50 53 57 67 75	79 80 86 134 139 143	47 52 63 67 74 76 78	
86 87 95 108 110 112	155 162 168 169 173	80 95 102 105 110	
113 116 118 120 122	174 178 183 200 201	112 130 133 136 144	
123 126 139 141 142	205 216 235 274 283	154 157 160 166 170	
148 150 151 161 169	309 332 344 351 359	192 194 202 207 210	
173 186 187 188 202	374 395 401 406 418	230 234 240 242 244	
203 204 207 209 210	431 439 457 463 466	247 249 261 263 272	
214 216 226 232 239	467 483 485 496 498	281 284 294 298 301	
246 254 261 269 270	8674 8691 8694 8704	310 314 318 322 324	
277 278 279 302 312	8705 8714 8735 8759	327 328 331 334 335	
315 319 322 326 331	8766 8767 8774 8775	347 359 369 372 376	
343 346 351 358 360	8783 8790 8795 8801	379 386 390 393 397	
363 366 367 381 385	8812 8813 8825 8828	400 408 410 418 432	
395 399 401 403 405	8860 8867 8870 8882	434 436 438 439 440	
406 412 419 420 431	8900 8901 8907 8916	443 444 445 460 462	
432 433 435 440 459	8927 8928 8941 8942	464 465 466 472 480	
463 465 469 471 476	8965 8973 8985 8988	481 483 488 497 499	
478 486 8668 8684	9003 9010 9022 9061	500 8670 8671 8674	
8688 8696 8699 8704	9063 9070 9091 9097	8675 8678 8698 8711	
8713 8721 8724 8731	9113 9127 9131 9139	8713 8716 8720 8722	
8732 8735 8743 8748	9149 9153	8723 8726 8731 8738	
8749 8758 8759 8761		8739 8746 8747 8754	
8764 8766 8770 8771		8757 8778 8781 8787	
8778 8780 8781 8782		8790 8801 8808 8809	
8791 8803 8805 8811		8815 8826 8831 8841	
8815 8828 8832 8837		8842 8845 8858 8864	
8838 8839 8844 8858		8866 8872 8875 8876	
8867 8872 8876 8877		8884 8886 8895 8901	
8886 8887 8889 8893		8904 8914 8915 8920	

# 华中科技大学计算机学院数据结构课程设计实验报告

8895 8899 8906 8907		8921 8925 8939 8940	
8909 8921 8923 8935		8941 8947 8956 8968	
8936 8938 8941 8947		8969 8983 8992 9008	
8949 8950 8951 8954		9025 9026 9035 9038	
8964 8966 8971 8988		9042 9048 9051 9066	
8993 8996 8997 9006		9068 9074 9076 9080	
9008 9018 9020 9023		9088 9091 9093 9109	
9026 9028 9029 9031		9110 9121 9124	
9032 9037 9042			

如图 1-28，用户 1 的信息查询结果符合预期。

1 “蔡艾冬”的朋友：(178位)

10 蔡辞	11 蔡从宏	21 蔡共捷	24 蔡光兴	26 蔡贵山	31 蔡红蕴	38 蔡兼宇	39 蔡建	
43 蔡敬先	50 蔡伶茜	53 蔡壮玟	57 蔡前阳	67 蔡圣泉	75 蔡文武	86 蔡星	87 蔡杏瑾	
95 蔡茵	108 蔡枝洁	110 蔡志帅	112 蔡致瑗	113 蔡忠羽	116 曹本发	118 曹宾	120 曹帛帝	
122 曹参辰	123 曹参君	126 曹翠	139 曹佳倩	141 曹佳贤	142 曹建洪	148 曹君	150 曹蓝	
151 曹蓝茹	161 曹美媛	169 曹千帝	173 曹庆洁	186 曹思苹	187 曹天志	188 曹彤	202 曹欣鸢	
203 曹兴	204 曹兴贤	207 曹亚盈	209 曹彦	210 曹瑶玲	214 曹意珠	216 曹应眉	226 曹镇国	
232 曾秉希	239 曾承茗	246 曾功林	254 曾辉	261 曾骄强	269 曾昆	270 曾乐华	277 曾农彤	
278 曾萍婷	279 曾萍娅	302 曾书思	312 曾贤菲	315 曾小健	319 曾秀姣	322 曾雪菊	326 曾彦	
331 曾游武	343 曾泽捷	346 曾芝	351 陈必林	358 陈秉远	360 陈辞	363 陈东	366 陈刚	
367 陈更材	381 陈兼修	385 陈骄发	395 陈敬轮	399 陈骏	401 陈利	403 陈利怡	405 陈连舜	
406 陈莲眉	412 陈鹏	419 陈庆书	420 陈秋利	431 陈世辰	432 陈世根	433 陈世明	435 陈世山	
440 陈祥	459 陈永	463 陈有迪	465 陈月璐	469 陈珍媚	471 陈致蓉	476 成帛原	478 成参刚	
486 成恩川	8668 郑敬辰	8684 郑庆振	8688 郑瑞	8696 郑淑娇	8699 郑文洪	8704 郑欣舒	8713 郑艳	
8721 郑原	8724 郑芝	8731 郑忠卿	8732 钟艾宏	8735 钟璨	8743 钟聪友	8748 钟蝶辰	8749 钟妃	
8758 钟红馨	8759 钟宏	8761 钟华恒	8764 钟欢霞	8766 钟继洪	8770 钟兼振	8771 钟建	8778 钟景芷	
8780 钟力	8781 钟利欢	8782 钟利兰	8791 钟其聪	8803 钟少兴	8805 钟仕刚	8811 钟香蓉	8815 钟小仪	
8828 钟亿鹏	8832 钟盈	8837 钟映友	8838 钟喻倩	8839 钟月丽	8844 钟芷	8858 周丹	8867 周姣	
8872 周敬	8876 周蓝灵	8877 周蓝芹	8886 周伶	8887 周茂军	8889 周玫芹	8893 周启业	8895 周润吟	
8899 周树原	8906 周天芷	8907 周苇	8909 周希	8921 周雅桂	8923 周亚时	8935 周远清	8936 周云	
8938 周韵蕴	8941 周致旺	8947 朱帛彤	8949 朱春花	8950 朱春霞	8951 朱聪员	8954 朱斗光	8964 朱虹	
8966 朱虹萱	8971 朱佳帅	8988 朱千	8993 朱任晨	8996 朱锐潇	8997 朱书婕	9006 朱伟冬	9008 朱苇彦	
9018 朱星伟	9020 朱秀珊	9023 朱雅雁	9026 朱娅茜	9028 朱颜棉	9029 朱艳榕	9031 朱瑶苹	9032 朱瑶英	
9037 朱宇舜	9042 朱再权							

a

## 华中科技大学计算机学院数据结构课程设计实验报告

1 “蔡艾冬”的关注人：（98位）												
7	蔡彩惠	26	蔡贵山	28	蔡海	32	蔡华	35	蔡怀莲	48	蔡力	73
												79 蔡熙株
80	蔡祥发	86	蔡星	134	曹关彤	139	曹佳倩	143	曹建岚	155	曹莉虹	162
												168 曹千
169	曹千帝	173	曹庆洁	174	曹庆莎	178	曹森进	183	曹诗	200	曹欣丹	201
												205 曹雪荃
216	曹应眉	235	曾材业	274	曾玖鹏	283	曾庆荃	309	曾希	332	曾宇	344
												351 陈必林
359	陈超	374	陈汉恒	395	陈敬轮	401	陈利	406	陈莲眉	418	陈庆	431
												439 陈穗
457	陈盈滢	463	陈有迪	466	陈珍静	467	陈珍玲	483	成迪杰	485	成斗康	496
												498 成捷清
8674	郑龙	8691	郑森靖	8694	郑仕兴	8704	郑欣舒	8705	郑星	8714	郑业平	8735
												8759 钟宏
8766	钟继洪	8767	钟继华	8774	钟姣	8775	钟婕芹	8783	钟利莎	8790	钟娜	8795
												8801 钟若嘉
8812	钟翔	8813	钟小欢	8825	钟怡遥	8828	钟亿鹏	8860	周奉贵	8867	周姣	8870
												8882 周良岚
8900	周顺书	8901	周思	8907	周苇	8916	周啸杰	8927	周亿贤	8928	周益	8941
												8942 周中兵
8965	朱虹婷	8973	朱婕	8985	朱飘慧	8988	朱千	9003	朱天骏	9010	朱熙香	9022
												9061 邹飞
9063	邹更江	9070	邹海莉	9091	邹金莲	9097	邹连林	9113	邹求臻	9127	邹仙琴	9131
												9139 邹怡音
9149	邹瑜玲	9153	邹占临									

### b

1 “蔡艾冬”的粉丝：（173位）												
12	蔡道奇	16	蔡妃	18	蔡富颖	28	蔡海	37	蔡佳芬	43	蔡敬先	44
												47 蔡蕾
52	蔡铭鹏	63	蔡秋琦	67	蔡圣泉	74	蔡尉莺	76	蔡惜瑾	78	蔡惜珠	80
												95 蔡茵
102	蔡泽娜	105	蔡珍瑗	110	蔡志帅	112	蔡致瑗	130	曹迪健	133	曹功川	136
												曹惠静
154	曹利	157	曹茂旺	160	曹美娟	166	曹萍棋	170	曹巧慧	192	曹贤菊	194
												曹晓琳
207	曹亚盈	210	曹瑶玲	230	曾本渊	234	曾帛员	240	曾大根	242	曾菲	244
												曾革南
249	曾虹爽	261	曾骄强	263	曾婕怡	272	曾茂贵	281	曾强	284	曾庆芝	294
												曾尚希
301	曾书芹	310	曾希苗	314	曾香莲	318	曾秀	322	曾雪菊	324	曾言璐	327
												曾燕帛
331	曾游武	334	曾玉芳	335	曾元昆	347	曾众冬	359	陈超	369	陈恭娥	372
												陈海
379	陈慧斐	386	陈婕苗	390	陈金静	393	陈进坚	397	陈菊香	400	陈蕾	408
												陈伶苇
418	陈庆	432	陈世根	434	陈世权	436	陈寿渊	438	陈素姬	439	陈穗	440
												陈祥
444	陈彦芝	445	陈艳蕾	460	陈永文	462	陈游东	464	陈喻蓉	465	陈月璐	466
												陈珍静
480	成德臻	481	成德贵	483	成迪杰	488	成芳	497	成建虎	499	成婕瑶	500
												成津希
8671	郑莲萧	8674	郑龙	8675	郑铭东	8678	郑奇	8698	郑添贞	8711	郑研刚	8713
												郑艳
8720	郑喻婷	8722	郑泽秋	8723	郑支友	8726	郑枝花	8731	郑忠卿	8738	钟春霏	8739
												钟春霞
8747	钟蝶帛	8754	钟固伟	8757	钟红	8778	钟景芷	8781	钟利欢	8787	钟龙	8790
												钟娜
8808	钟素君	8809	钟苇娇	8815	钟小仪	8826	钟乙	8831	钟意音	8841	钟月影	8842
												钟占飞
8858	周丹	8864	周吉	8866	周娇	8872	周敬	8875	周静瑶	8876	周蓝灵	8884
												周林
8895	周润吟	8901	周思	8904	周滕宏	8914	周孝庆	8915	周啸彪	8920	周雪蓝	8921
												周雅桂
8939	周在昆	8940	周泽南	8941	周致旺	8947	朱帛彤	8956	朱恩甫	8968	朱虹珠	8969
												朱洪
8992	朱荃	9008	朱苇彦	9025	朱亚娅	9026	朱娅茜	9035	朱咏苗	9038	朱玉荃	9042
												朱再权
9051	邹碧香	9066	邹关武	9068	邹海	9074	邹红	9076	邹华冲	9080	邹继泉	9088
												邹江刚
9093	邹敬庆	9109	邹苹	9110	邹启华	9121	邹淑玲	9124	邹顺南			

### c

1 “蔡艾冬”的爱好：（6个）												
27	舞蹈	38	小说	48	网球	65	滑雪	66	KTV	72	时尚	

### d

图 1-31 信息查询结果

接着对二度好友进行测试，由于共同关注、共同爱好与此类似，不再测试。  
如表 1-13，为用户 1 和用户 2 的二度好友理论测试结果

表 1-13 共同好友测试理论结果

用户 1 的好友	用户 2 的好友	用户 1、2 的二度好友
10 11 21 24 26 31 38 39 43	4 8 29 33 39 48 51 54 56 57	39 57 75 122 139 207 214
50 53 57 67 75 86 87 95 108	60 71 75 77 89 122 129 134	322 395 403 476 8684 8731
110 112 113 116 118 120 122	139 144 156 177 178 207 214	8766 8782 8867 8909 8949
123 126 139 141 142 148 150	218 228 237 241 242 245 251	8964 8997 9018 9032
151 161 169 173 186 187 188	259 265 285 289 305 309 314	
202 203 204 207 209 210 214	318 322 337 342 354 369 378	
216 226 232 239 246 254 261	395 396 403 408 409 410 423	
269 270 277 278 279 302 312	430 434 444 456 466 472 476	
315 319 322 326 331 343 346	479 480 494 8684 8690 8693	
351 358 360 363 366 367 381	8697 8717 8719 8722 8725	
385 395 399 401 403 405 406	8731 8734 8736 8755 8763	
412 419 420 431 432 433 435	8766 8782 8798 8812 8818	
440 459 463 465 469 471 476	8822 8823 8842 8850 8867	
478 486 8668 8684 8688	8874 8900 8903 8909 8911	
8696 8699 8704 8713 8721	8913 8919 8928 8933 8942	
8724 8731 8732 8735 8743	8943 8944 8949 8964 8978	
8748 8749 8758 8759 8761	8979 8997 9010 9018 9019	
8764 8766 8770 8771 8778	9027 9032 9038 9046 9047	
8780 8781 8782 8791 8803	9048 9051 9054 9057 9058	
8805 8811 8815 8828 8832	9067 9073 9076 9078 9086	
8837 8838 8839 8844 8858	9094 9095 9098 9123 9124	
8867 8872 8876 8877 8886	9129 9135 9140 9146 9152	
8887 8889 8893 8895 8899	9158	
8906 8907 8909 8921 8923		
8935 8936 8938 8941 8947		

8949 8950 8951 8954 8964		
8966 8971 8988 8993 8996		
8997 9006 9008 9018 9020		
9023 9026 9028 9029 9031		
9032 9037 9042		

如图 1-32，用户 1 和 2 的二度好友，测试结果符合预期。

```

请选择你的操作[0~9]:6
请分别输入两位用户id:
1
2
"蔡艾冬"和"蔡艾豪"的共同好友:
39  蔡建   |57  蔡前阳 |75  蔡文武 |122  曹参辰 |139  曹佳倩 |207  曹亚盈 |214  曹意珠 |322  曾雪菊 |
395  陈敬轮 |403  陈利怡 |476  成帛原 |8684  郑庆振 |8731  郑忠卿 |8766  钟继洪 |8782  钟利兰 |8867  周姣 |
8909  周希   |8949  朱春花 |8964  朱虹   |8997  朱书婵 |9018  朱星伟 |9032  朱瑶英 |
    
```

图 1-32 二度好友测试结果

然后，测试信息修改。首先，测试姓名修改。查询 1 号用户信息可知其姓名为“蔡艾冬”，修改姓名后再次查询结果为“太阳骑士”。修改过程如图 1-33 所示。

```

C:\Users\10334\source\repos\数据结构\Debug\数据结构.exe
-----1 蔡艾冬-----
1. 好友集                2. 关注人集
3. 粉丝集                4. 爱好集
5. 重新选择查询用户      0. 退出查询

C:\Users\10334\source\repos\数据结构\Debug\数据结构.exe
请选择维护用户"蔡艾冬"的内容
      1. 维护用户朋友集      2. 维护用户关注人集
      3. 维护用户爱好集      4. 移除用户粉丝
      5. 修改用户姓名
      0. 退出维护
5
请输入新的姓名
太阳骑士
姓名修改成功.
    
```

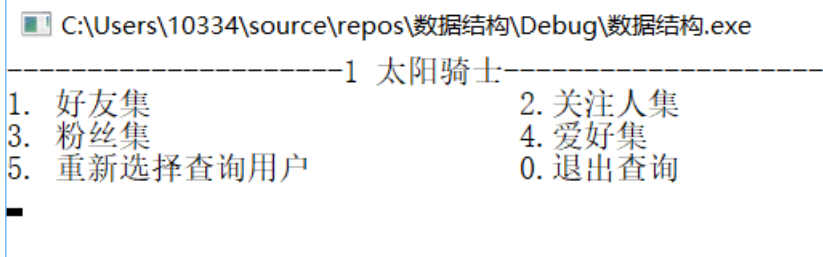


图 1-33 姓名修改

姓名修改之后 1 号用户姓名在其好友中也一同改变，这里选取用户 1 的一位朋友用户 10 来说明。



图 1-34 用户 10 的好友集

接着，对用户 1 的好友集进行增删操作。将用户 10 从用户 1 的好友集移除后用户 10 的好友集也无用户 1，同样用户的粉丝集、关注人集以及爱好集的删除操作也是如此。

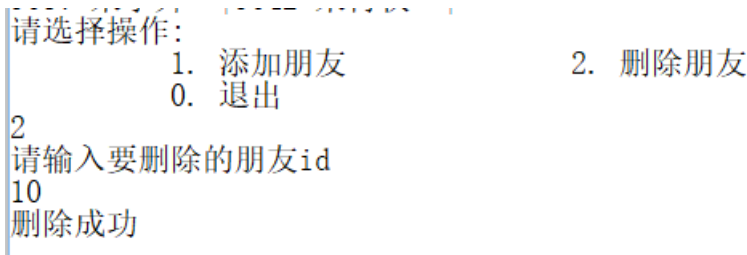
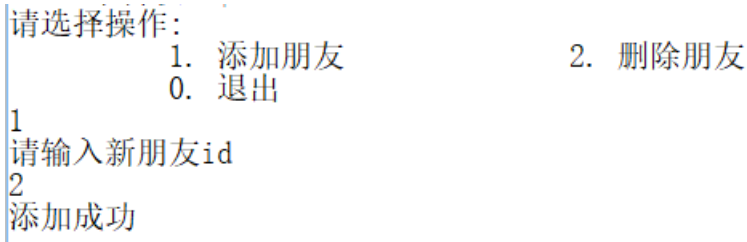


图 1-35 从用户 1 的好友集移除用户 10



图 1-36 移除后用户 10 的好友集

再次将 10 用户添加为 1 的好友后又可以重新查询到，这里将用户 2 和用户 10 添加为 1 的好友，之后查询用户 1 的好友集可以找到用户 2 和用户 10。操作结果如图 1-37 所示：





```
请选择操作:
      1. 添加朋友      2. 删除朋友
      0. 退出
1
请输入新朋友id
10
添加成功
```

b

图 1-37 添加好友操作

1	用户“太阳骑士”的朋友														
2	蔡艾豪	10	蔡辞	11	蔡从宏	21	蔡共捷	24	蔡光兴	26	蔡贵山	31	蔡红蕴	38	蔡兼宇
39	蔡建	43	蔡敬先	50	蔡伶俐	53	蔡壮玟	57	蔡前阳	67	蔡圣泉	75	蔡文武	86	蔡星

图 1-38 添加后用户 1 的好友

## 5 总结与展望

### 5.1 全文总结

- 1) 深刻理解并掌握了 AVL 树的创建、删除、查询算法
- 2) 对递归的了解更深了一步
- 3) 对实际问题的抽象有了了解，初步了解了将实际问题转化为合适的数据结构表示
- 4) 调试能力有了极大提高

### 5.1 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) 为实际应用选取合适的数据结构
- 2) 采用更高效的算法
- 3) 更加友好的用户交互

## 6 体 会

数据结构是计算机科学的一门非常重要的专业基础课,内容丰富,涉及面广,与 C 语言相比,它更加注重实践,注重与实际应用的结合。数据结构课程设计是计算机科学与技术专业学生的集中实践性环节之一,是学习“数据结构与算法”理论和实验课程后进行的一次全面的综合练习。

在这次课程设计当中,我了解到了我的不足,如算法的不完善、不细心等等。不细心的我在调试程序时,老是因为某个书写错误导致错误,比如在修改函数定义时忘了修改函数原型,从而因数据类型不匹配而造成编译错误,再比如大括号只少输了一个,导致程序大面积错误却不知道在哪里少输入了大括号;对这些错误,我不得不花大量的时间去更正,并且还要重复检查是否出现雷同的错误而导致程序不能运行。通过这次课程设计的训练,我的这些缺点有些改善,在写新的程序时,首先要考虑的深入一点、仔细一点,这样要修改程序的时间就会少很多。并且也不会因为自己不细心而导致的浪费时间的情况出现。

程序设计不同于普通的 C 语言程序,首先体现在代码量上,上千行的代码是必不可少的,因此,在进行程序设计时,要注意想好思路,而合理的函数变量命名时写好课程设计的基础,恰当模块名、变量名、常量名、子程序名等就显得十分重要了,只有做到条理清晰,才能使自己对程序的总体有清楚的构思。在书写程序时,要在合适的地方加上代码注释,将每个功能的模块,即函数名要清晰的表述出来,这样不仅是使用户能够一目了然此程序的功能,更重要的是方便了自己之后的调试和更改。当然,任何程序必须经过计算机的调试,在调试的过程中发现错误,一个个,一步步去解决,这样就能从错误中进步。此外,在调试过程中,最好是写完一个模块调试一个模块,不要积攒到最后一起调试,否则很难找到错误出现在哪一个环节

通过此次课程设计,我了解了编写应用软件的一般步骤,加强了我的动手能力,提升了自己查阅总结文献的能力,特别是怎样将理论与实践相结合。从拿到题目到完成整个编程,从理论到实践,在整整半个月的时间里,我学到了很多很多的东西,同时不仅可以巩固了以前所学过的知识,而且学到了很多在书本上所没有学到过的知识。最为重要的是,通过本次程序设计,我逐步具备了走向程

序员的基本素质，知道了如何在困难重重时一步一步细心发现问题，解决问题，知道了在软件设计中对界面和功能如何平衡，如何达到相对完美。

最后，在这次课程设计中，得到了好多同学的帮助以及老师的指导，在此要表达我真诚的谢意！

## 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] 杜薇薇, 张翼燕, 瞿春柳. 基于平衡因子的 AVL 树设计实现[J]. 计算机技术与发展, 2010, (3): 24-27, 31. DOI: 10.3969/j.issn.1673-629X.2010.03.007.
- [4] 孙权志. 用 C 语言实现的 AVL 树查找方法[J]. 微型机与应用, 1993, (8): 12-14.
- [5] 杜薇薇, 张翼燕, 瞿春柳. 基于平衡因子的 AVL 树设计实现[J]. 计算机技术与发展, 2010, (3): 24-27, 31. DOI: 10.3969/j.issn.1673-629X.2010.03.007.
- [6] CSDN 博客 [http://blog.csdn.net/sysu\\_arui/article/details/7897017](http://blog.csdn.net/sysu_arui/article/details/7897017)
- [7] C Primer Plus (第 6 版) 中文版 / (美) 普拉达 (Prata, S.) 著; 姜佑译. 北京: 人民邮电出版社, 2016.4

## 附录 A 程序头文件源代码

```
#include<stdbool.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<windows.h>

#define ACCOUNTDATA "account_data"//账户集

#define RELATIONDATA "relation_data"//用户关系（粉丝、朋友、兴趣等）

#define HOBBYDATA "hobby_data"//爱好集

#define L_DATA_NAME "ListData"//多集合管理的线性表信息

#define T_DATA_NAME "SetData"//集合信息

#define LH 1 //左高

#define EH 0 //等高

#define RH -1 //右高

#define TRUE 1

#define FALSE 0

#define X0 50//AVL 树图形化显示时根节点的位置

#define OK 1

#define OVERFLOW 0

#define FLAT -100//作为集合存储信息的结束符

#define LIST_INIT_SIZE 50//可管理的最大集合数量


typedef int Status;

typedef struct Info {

    int id;

    char name[20];

    struct BSTNode * friends;//朋友集
```

```

    struct BSTNode * fans;//粉丝集

    struct BSTNode * follows;//关注人集

    struct BSTNode * hobby;//爱好集
}Info;//data 域
typedef struct BSTNode {
    Info data;
    int bf;//平衡因子
    struct BSTNode *lchild, *rchild;//左右孩子
}BSTNode, *BSTree;

typedef struct {
    struct setdata * pT;
    int listlenth;
    int listsize;
}SqList;//顺序表，用于管理多集合

typedef struct setdata {
    struct BSTNode * pRoot;
    char  name[20];
}SetData;//集合信息(线性表的结点)
/*平衡调节*/
void L_Rotate(BSTree &p);//左旋
void R_Rotate(BSTree &p);//右旋
void LeftBalance(BSTree &T);//左平衡处理
void RightBalance(BSTree &T);//右平衡处理
/*AVL 树的基本运算*/
Status InitAVL(BSTree &T);//创建 AVL 树
Status InsertAVL(BSTree &T, Info e, bool &taller);//插入结点
Status DestroyAVL(BSTree &T);//销毁 AVL 树
Status DeleteAVL(BSTree& T, int key, bool& shorter);//删除 AVL 树中结点

```

```

BSTNode * SearchAVL(BSTree& T, int key);//在 AVL 树中查找关键字为 key 的结
点
Status InOrderTraverse(BSTree T, Status(*visit)(Info e));//中序遍历
/*集合的基本运算*/
Status set_init(BSTree &T);//初始化集合
Status set_destroy(BSTree &T);//销毁集合
Status set_insert(BSTree &T, Info e);//插入元素
Status set_remove(BSTree &T, int key);//删除元素
Status set_intersection(BSTree T1, BSTree T2, BSTree &T3);//求 T1 与 T2 的交，存
在 T3 中
Status set_union(BSTree &T1, BSTree T2);//求 T1 与 T2 的并，结果存在 T1 中
Status set_difference(BSTree T1, BSTree T2, BSTree &T3);//求集合 T1 与集合 T2 的
差(T1-T2)，存在集合 T 中
int set_size(BSTree T);//集合代销
BSTNode * set_member(BSTree T, int key);//集合成员
Status set_subset(BSTree T1, BSTree T2);//集合子集
Status set_equal(BSTree T1, BSTree T2);//集合相等
Status set_traverse(BSTree T, Status(*visit)(Info e));//集合遍历
/*集合演示函数*/
void menu1(void);//集合操作演示菜单
void set_display(void);//集合演示
void printKey(BSTree p, int x, int y);//在命令行(x,y)处输出结点 p 的关键字
void graphAVL(BSTree T, int x, int l, int y);//将 AVL 树图形化显示（x,y 为根结点
坐标，l 为根结点的左右孩子相距距离）
void setxy(short x, short y); //定位光标到（x，y）
Status visit(Info e);
int SelectSet(SqList L);
int get_cursor_x(void);//获取当前光标横坐标
int get_cursor_y(void);//获取当前光标纵坐标
    
```



```

Status SaveSetData(SqlList L);
Status LoadSetData(SqlList &L);
Status SaveSet(BSTree T, FILE *fp);
Status LoadSet(BSTree &T, FILE *fp);
/*人际关系模拟函数*/
void menu2(void);//人际关系模拟演示菜单
void apply_display(void);//应用演示
Status destroy_accountset(BSTree &T);//删除账户集合
Status data_input(BSTree T, Info &in);//为集合 T 中插入的新节点输入 data,成功返回 TRUE, 失败返回 FALSE
int confirm_save(void);//确认是否保存, 是返回 1, 否返回 0
int select_display(void);//确认演示功能,集合演示返回 1, 应用演示返回 2, 退出返回 0
void maintain_data(BSTree &Account,BSTree Hobby);//维护用户信息
void common_follow(BSTree Account);//共同关注
void common_friend(BSTree Account);//共同好友
void second_friend(BSTree Account);//二度好友
void S_friend(BSTree Account, BSTree p, BSTree &T);//添加*p 的二度好友, 结果存在 T 中
void common_hobby(BSTree Account,BSTree Hobby);//共同爱好
Status save_account(BSTree T, FILE *fp);//保存账户信息
Status save_hobby(BSTree T,FILE *fp);//保存兴趣集合
Status save_relation(BSTree T, FILE *fp);//保存账户关系 1
Status save_rela(BSTree T, FILE *fp);//保存账户关系 2
Status load_account(BSTree &T, FILE *fp);//加载账户信息
Status load_relation(BSTree &T, FILE *fp);//加载关系信息
Status load_hobby(BSTree &Hobby,FILE *fp);//加载兴趣集合
Status load_data(BSTree &Account, BSTree &Hobby);//加载数据
Status save_data(BSTree Account, BSTree Hobby);//保存数据

```

void account\_data(BSTree Account, BSTree Hobby);//显示用户的关注人、朋友、粉丝、兴趣

void print\_account\_data(BSTree Account, BSTree T);//根据 T 中的用户 id 在 Account 中查找相应的用户姓名并显示

void proc(void);

void AVL\_display(void);

void menu3(void);

## 附录 B 程序 C 文件源代码

```
#include"head.h"

int space;//调整输出格式


int main(void) {
    //system("mode con cols=200 lines=40  ");
    system("color F0\n");
    while (1)
    {
        system("cls");
        int op = select_display();
        if (op == 1) { set_display(); }
        else if (op == 2) { apply_display(); }
        else if (op == 3) { AVL_display(); }
        else break;
    }
    printf("欢迎下次使用\n");
    getchar();
    return 0;

} //end of main()


/*AVL 树旋转操作*/
void L_Rotate(BSTree &p) {
    //对以*p 为根的二叉排序树作左旋处理，处理之后 p 指向新的数根节点，即
    //旋转处理之前的右子树的根节点
    BSTNode *rc=NULL;
    rc = p->rchild;//rc 指向*p 的右孩子
```

```

    p->rchild = rc->lchild;//rc 的左子树挂接为*p 的右子树
    rc->lchild = p; p = rc;//p 指向新的根节点
}

void R_Rotate(BSTree &p)
{
    //类似于 L_Rotate
    BSTNode *lc=NULL;
    lc = p->lchild;
    p->lchild = lc->rchild;
    lc->rchild = p; p = lc;
}

void LeftBalance(BSTree &T)
{
    {
        BSTNode* lc = NULL;
        BSTNode* rd = NULL;
        lc = T->lchild;
        switch (lc->bf)
        {
            case LH:                //LL 旋转
                T->bf = EH;
                lc->bf = EH;
                R_Rotate(T);
                break;

            case EH:                //deleteAVL 需要，insertAVL 用不着
                T->bf = LH;
                lc->bf = RH;
                R_Rotate(T);
                break;
        }
    }
}

```

```

        case RH:                                //LR 旋转

            rd = lc->rchild;
            switch (rd->bf)
            {
                case LH:
                    T->bf = RH;
                    lc->bf = EH;
                    break;
                case EH:
                    T->bf = EH;
                    lc->bf = EH;
                    break;
                case RH:
                    T->bf = EH;
                    lc->bf = LH;
                    break;
            }
            rd->bf = EH;
            L_Rotate(T->lchild);//不能写 L_Rotate(lc);采用的是引用参数
            R_Rotate(T);
            break;
        }
    }
}

void RightBalance(BSTree &T)
{
    //对以二叉树 T 作右平衡处理，并使 T 指向新的根节点
    BSTNode* rc = NULL;

```

```

BSTNode *ld = NULL;

rc = T->rchild;
switch (rc->bf)
{
case LH:                //RL 旋转
    ld = rc->lchild;
    switch (ld->bf)
    {
case LH:
        T->bf = EH;
        rc->bf = RH;
        break;
case EH:
        T->bf = EH;
        rc->bf = EH;
        break;
case RH:
        T->bf = LH;
        rc->bf = EH;
        break;
    }
    ld->bf = EH;
    R_Rotate(T->rchild);
    L_Rotate(T);
    break;

case EH:
    T->bf = RH;

```

```

        rc->bf = LH;
        L_Rotate(T);
        break;

    case RH:                //RR 旋转
        T->bf = EH;
        rc->bf = EH;
        L_Rotate(T);
        break;
    }
}

/*AVL 树的基本运算*/
Status InitAVL(BSTree &T)
{
    T = NULL;
    return TRUE;
}

Status InsertAVL(BSTree &T, Info e, bool &taller)
{
    //若 T 不存在与 e 有相同关键字的节点，插入 e，返回 1，否则返回 0
    //若插入 e 后二叉排序树失去平衡，作平衡化处理，taller 反应 T 长高与否

    if (!T) { //插入新节点，树长高，taller 为 TRUE
        T = (BSTNode *)malloc(sizeof(BSTNode)); T->data = e;
        T->rchild = T->lchild = NULL; T->bf = EH; taller = TRUE;
    }
    else
    {
        if (e.id == T->data.id) //树中已存在和 e 有相同关键字的结点

```

```

{
    taller = FALSE; return 0;//不再插入
}
if (e.id < T->data.id)//在 T 的左子树中搜索
{
    if (!InsertAVL(T->lchild, e, taller)) return 0;//未插入
    if (taller)//已插入到*T 的左子树中且左子树长高
        switch (T->bf)//检查*T 的平衡度
        {
            case LH://原本左子树比右子树高，需要作左平衡处理
                LeftBalance(T); taller = FALSE; break;//
            case EH://原本左右子树等高，现因左子树增高而使树增高
                T->bf = LH; taller = TRUE; break;
            case RH://原本右子树比左子树高，现左右子树等高
                T->bf = EH; taller = FALSE; break;
        }
    }
else//在 T 的右子树中搜索
{
    if (!InsertAVL(T->rchild, e, taller)) return 0;//未插入
    if (taller)//已插入到*T 的右子树中且右子树长高
        switch (T->bf)//检查*T 的平衡度
        {
            case LH://原本左子树比右子树高，现左右子树等高
                T->bf = EH; taller = FALSE; break;
            case EH://原本左右子树等高，现因右子树增高而使树增高
                T->bf = RH; taller = TRUE; break;
            case RH://原本右子树比左子树高，需要作右平衡处理
                RightBalance(T); taller = FALSE; break;
        }
    }
}

```



```

        }

    }

}

return TRUE;
}

Status DestroyAVL(BSTree &T)
{//销毁 AVL 树 T
    if (!T) return FALSE;//树为空
    else
    {
        DestroyAVL(T->lchild);//销毁左子树
        DestroyAVL(T->rchild);//销毁右子树
        free(T);//释放节点
        T = NULL;//根节点指针置空
        return TRUE;
    }
}

BSTNode * SearchAVL(BSTree& T, int key)
{//在 AVL 树 T 中查找关键字为 key 的节点，找到返回其地址，否则返回 NULL
    if ((T == NULL) || (key == T->data.id))
        return T;
    else if (key < (T->data.id))//在左子树中继续查找
        return SearchAVL(T->lchild, key);
    else
        return SearchAVL(T->rchild, key);//在右子树中继续查找
}

Status DeleteAVL(BSTree& T, int key, bool& shorter)
{
    //若在平衡的二叉排序树 t 中存在和 e 有相同关键字的结点，则删除之

```

```

//并返回 true，否则返回 false。若因删除而使二叉排序树
//失去平衡，则作平衡旋转处理，布尔变量 shorter 反映 t 变矮与否
if (T == NULL)//不存在该元素
{
    return FALSE;//删除失败
}
else if (key == T->data.id)//找到元素结点
{
    BSTNode * q = NULL;
    if (T->lchild == NULL)//左子树为空,令 T 指向要删除结点的右子树，然后
删除该结点(包含左右子树都为空的情况)
    {
        q = T;
        T = T->rchild;
        free(q);
        shorter = TRUE;
    }
    else if (T->rchild == NULL)//右子树为空，令 T 指向要删除结点的左子树，
然后删除该结点
    {
        q = T;
        T = T->lchild;
        free(q);
        shorter = TRUE;
    }
    else//左右子树都存在,交换结点数据后删除结点
    {
        q = T->lchild;

```

```

while (q->rchild)
{
    q = q->rchild;
} //q 定位到要删除结点的左子树的最右子树根节点
T->data = q->data; //将 q 的数据置换到 T 结点
//graphAVL(T, X0, X0, get_cursor_y());
DeleteAVL(T->lchild, q->data.id, shorter); //在左子树中递归删除 q 结点
if (shorter)
{
    switch (T->bf)
    {
        case LH: //原来左高，现左变矮，故等高
            T->bf = EH;
            shorter = TRUE;
            break;
        case EH: //原等高，现左变矮，故右高
            T->bf = RH;
            shorter = FALSE;
            break;
        case RH: //原右高，现左矮，故右更高，不平衡，需进行右平衡调
                节

            if (T->rchild->bf == EH)
                shorter = FALSE;
            else
                shorter = TRUE;
            RightBalance(T); //右平衡处理
            break;
    }
}

```

```

        }
    }
}
else if (key < T->data.id)//左子树中继续查找
{
    if (!DeleteAVL(T->lchild, key, shorter))//删除失败
    {
        return FALSE;
    }
    if (shorter)//左子树变矮，调节 T 平衡因子
    {
        switch (T->bf)
        {
            case LH://原来左高，现左变矮，故等高
                T->bf = EH;
                shorter = TRUE;
                break;
            case EH://原等高，现左变矮，故右高
                T->bf = RH;
                shorter = FALSE;
                break;
            case RH://原右高，现左矮，故右更高，不平衡，需进行右平衡调节

                if (T->rchild->bf == EH)
                    shorter = FALSE;
                else
                    shorter = TRUE;
                RightBalance(T);    //右平衡处理
                break;
            }
        }
    }
}

```

```

        }
    }
}
else //在右子树中继续查找，操作同左子
树
{
    if (!DeleteAVL(T->rchild, key, shorter))
    {
        return FALSE;
    }
    if (shorter)
    {
        switch (T->bf)
        {
            case LH:

                if (T->lchild->bf == EH)
                    shorter = FALSE;
                else
                    shorter = TRUE;
                LeftBalance(T);    //左平衡处理
                break;
            case EH:
                T->bf = LH;
                shorter = FALSE;
                break;
            case RH:
                T->bf = EH;
                shorter = TRUE;

```

```

        break;
    }
}
}
return true;

}

Status InOrderTraverse(BSTree T, Status(*visit)(Info e))
{
    if (T)//T 非空
    {

        InOrderTraverse(T->lchild, visit);
        if (!visit(T->data)) return FALSE;
        InOrderTraverse(T->rchild, visit);
    }
    return TRUE;
}

/*集合的基本运算*/
Status set_init(BSTree &T)
{
    return InitAVL(T);
}

Status set_destroy(BSTree &T)
{
    return DestroyAVL(T);
}

Status set_insert(BSTree &T, Info e)
{

```

```

    bool taller;

    return InsertAVL(T, e, taller);
}

Status set_remove(BSTree &T, int key)
{
    bool shorter;

    return DeleteAVL(T, key, shorter);
}

Status set_intersection(BSTree T1, BSTree T2, BSTree &T3)
{
    //遍历 T2 中所有元素，若 T1 中也有，则插入到 T3 中
    if (!T2 || !T1) return TRUE;//T1 或 T2 为空
    if (set_member(T1, T2->data.id))
        set_insert(T3, T2->data);
    set_intersection(T1, T2->lchild, T3);//遍历 T2 的左子树
    set_intersection(T1, T2->rchild, T3);//遍历 T2 的右子树
    return TRUE;
}

Status set_union(BSTree &T1, BSTree T2)
{
    //将 T2 中的元素并到 T1 中
    if (!T2) return FALSE;//T2 为空
    else
    {
        set_insert(T1, T2->data);
        set_union(T1, T2->lchild);//合并 T2 的左子树
        set_union(T1, T2->rchild);//合并 T2 的右子树
        return TRUE;
    }
}

Status set_difference(BSTree T1, BSTree T2, BSTree &T3)

```

```

{
    if (!T1) return FALSE;//T1 为空,则 T1-T2 为空
    if (!set_member(T2, T1->data.id))
        set_insert(T3, T1->data);//将属于 T1, 不属于 T2 的元素插入到 T 中
    set_difference(T1->lchild, T2, T3);//遍历 T1 的左子树
    set_difference(T1->rchild, T2, T3);//遍历 T1 的右子树
    return TRUE;
}

int set_size(BSTree T)
{
    if (!T) return 0;//T 为空
    else
    {
        return 1 + set_size(T->lchild) + set_size(T->rchild);//T 元素个数=1+左子树
        元素个数+右子树元素个数
    }
}

BSTNode * set_member(BSTree T, int key)
{
    //找到返回节点地址, 未找到返回 NULL
    return SearchAVL(T, key);
}

Status set_subset(BSTree T1, BSTree T2)
{
    //判断 T2 是否为 T1 的子集, 是返回 TRUE, 否返回 FALSE
    if (T2 == NULL) return TRUE;//T2 为空集
    if (set_member(T1, T2->data.id))//T2 根节点是 T1 的元素
    {
        if (!set_subset(T1, T2->lchild))//判断 T2 的左子树是否为 T1 的子集
            return FALSE;
        if (!set_subset(T1, T2->rchild))//判断 T2 的右子树是否为 T1 的子集

```



```

        return FALSE;

    return TRUE;

}

else return FALSE;//T2 的根结点不是 T1 的子集
}

Status set_equal(BSTree T1, BSTree T2)
{
    //集合 T1=T2 返回 1， 否则返回 0

    return set_subset(T1, T2) && set_subset(T2, T1);//T1 T2 互为子集则相等
}

Status set_traverse(BSTree T, Status(*visit)(Info e))
{
    space = 0;

    if (T == NULL) { printf("集合为空\n"); return FALSE; }
    else { InOrderTraverse(T, visit); return TRUE; }
}

/*集合演示函数*/
void menu1(void)
{
    printf("\n\n");
    printf("-----\n");
    printf("      集合操作演示  \n");
    printf("-----\n");
    printf("      1. set_init      7. set_difference      13.set_intersection\n");
    printf("\n");
    printf("      2. set_destroy   8. set_subset          14.set_equal\n");
    printf("      3. set_insert    9. set_traverse        \n");
    printf("      4. set_remove    10.SaveSet\n");
    printf("      5. set_size      11.LoadSet\n");
}

```

```

printf("      6. set_member      12.set_union\n");
printf("      0. Exit\n");
printf("-----\n");
printf("      请选择你的操作[0~9]:");
}

void set_display(void)
{
    SqList L;
    L.pT = (SetData *)malloc(LIST_INIT_SIZE * sizeof(SetData));//线性表最多可
管理 LIST_INIT_SIZE 棵树
    L.listsize = LIST_INIT_SIZE;
    L.listlenth = 0;

    int myop = 1, key;
    while (myop) {
        system("cls");
        menu1();
        scanf("%d", &myop);
        if (L.listlenth == 0 && (myop != 1 && myop != 11))
        {
            printf("集合不存在，请先初始化集合\n");
            getchar(); getchar();
            continue;
        }
        switch (myop) {
        case 1:
        {

            if (L.listlenth >= L.listsize)

```

```

    {
        printf("空间不足，无法创建新集合\n");
        break;
    }

    set_init(L.pT[L.listlenth].pRoot); //在线性表表尾创建集合
    printf("输入集合的名称:\n");
    scanf_s("%s", L.pT[L.listlenth].name, 20);
    printf("集合%s 创建成功! \n", L.pT[L.listlenth].name);
    L.listlenth++; //线性表同步创建一集合

    getchar(); getchar();
    break;
}
case 2:
{
    printf("请选择销毁的集合\n");
    int op = SelectSet(L);
    set_destroy(L.pT[op].pRoot);
    printf("集合 T 销毁成功\n");
    printf("集合%s 销毁成功! \n", L.pT[op].name);

    for (int i = op; i < L.listlenth - 1; i++)
    {
        L.pT[i] = L.pT[i + 1];
    }
    L.listlenth--; //线性表同步删除一棵树
    getchar(); getchar();
    break;
}

```

```

graphAVL(L.pT[op].pRoot, X0, X0, get_cursor_y());
getchar(); getchar();
break;
}
case 3:
{
printf("请选择要初始化的集合\n");
int op = SelectSet(L);
int N;
Info a;
printf("请输入数据的组数\n");
scanf("%d", &N);
while (N--)
{
scanf("%d", &a.id);
set_insert(L.pT[op].pRoot, a);
}
graphAVL(L.pT[op].pRoot, X0, X0, get_cursor_y());
getchar(); getchar();
break;
}
case 4:
{
printf("请选择需要删除元素的集合\n");
int op = SelectSet(L);
printf("输入要删除元素的关键字\n");
graphAVL(L.pT[op].pRoot, X0, X0, get_cursor_y());
putchar('\n');
putchar('\n');

```

```

scanf("%d", &key);
putchar('\n'); putchar('\n');
if (set_remove(L.pT[op].pRoot, key)) printf("删除成功\n");
else printf("元素不存在\n");
graphAVL(L.pT[op].pRoot, X0, X0, get_cursor_y());
getchar(); getchar();
break;
}
case 5:
{
    printf("请选择需要显示大小的集合\n");
    int op = SelectSet(L);
    printf("集合%s 大小%d\n", L.pT[op].name, set_size(L.pT[op].pRoot));
    getchar(); getchar();
    break;
}
case 6:
{
    printf("请选择需要查询元素的集合\n");
    int op = SelectSet(L);
    printf("输入关键字\n");
    int key;
    scanf("%d", &key);
    if (set_member(L.pT[op].pRoot, key)) printf("%d 在集合%s 中\n", key,
L.pT[op].name);
    else
    {
        printf("%d 不在集合%s 中\n", key, L.pT[op].name);
    }
}

```

```

graphAVL(L.pT[op].pRoot, X0, X0, get_cursor_y());
getchar(); getchar();
break;
}
case 7:
{
printf("请选择第一个集合（运算结果为第一个集合 - 第二个集合）
\n");

int op1 = SelectSet(L);
printf("请选择第二个集合");
int op2 = SelectSet(L);

set_init(L.pT[L.listlenth].pRoot);//在线性表表尾创建集合
printf("输入保存运算结果的集合的名称:\n");
scanf_s("%s", L.pT[L.listlenth].name, 20);
printf("集合%s 创建成功! \n", L.pT[L.listlenth].name);

printf("集合%s: ", L.pT[op1].name); set_traverse(L.pT[op1].pRoot, visit);
putchar('\n');

graphAVL(L.pT[op1].pRoot, X0, X0, get_cursor_y());
setxy(0, get_cursor_y() + 4);//重新定位光标在新的一行

printf("集合%s: ", L.pT[op2].name); set_traverse(L.pT[op2].pRoot, visit);
putchar('\n');

graphAVL(L.pT[op2].pRoot, X0, X0, get_cursor_y());
setxy(0, get_cursor_y() + 4);

set_difference(L.pT[op1].pRoot, L.pT[op2].pRoot,
L.pT[L.listlenth].pRoot);//T1-T2

```

```

printf("集合%s-集合%s: ", L.pT[op1].name, L.pT[op2].name);
set_traverse(L.pT[L.listlenth].pRoot, visit);
putchar('\n');
graphAVL(L.pT[L.listlenth].pRoot, X0, X0, get_cursor_y());
setxy(0, get_cursor_y() + 4);
printf("是否保存运算结果数据? 1.是 0.否\n");
int c;
scanf("%d", &c);
if (c == 0)
{
    set_destroy(L.pT[L.listlenth].pRoot);
}
else L.listlenth++;
getchar(); getchar();
break;
}
case 8:
{
    printf("请选择原集合（判断子集合是否为原集合的子集）\n");
    int op1 = SelectSet(L);
    printf("请选择子集合\n");
    int op2 = SelectSet(L);

    printf("集合%s: ", L.pT[op1].name); set_traverse(L.pT[op1].pRoot, visit);
    putchar('\n');

    graphAVL(L.pT[op1].pRoot, X0, X0, get_cursor_y());
    setxy(0, get_cursor_y() + 4); //重新定位光标在新的一行

    printf("集合%s: ", L.pT[op2].name); set_traverse(L.pT[op2].pRoot, visit);

```

```

putchar('\n');

    graphAVL(L.pT[op2].pRoot, X0, X0, get_cursor_y());
    setxy(0, get_cursor_y() + 4);

    if (set_subset(L.pT[op1].pRoot, L.pT[op2].pRoot))
    {
        printf("集合 %s 是集合 %s 的子集合\n", L.pT[op2].name,
L.pT[op1].name);
    }
    else printf("集合 %s 不是集合 %s 的子集合\n", L.pT[op2].name,
L.pT[op1].name);

    getchar(); getchar();
    break;
}
case 9:
{
    printf("请选择需要遍历的集合\n");
    int op = SelectSet(L);
    set_traverse(L.pT[op].pRoot, visit);
    graphAVL(L.pT[op].pRoot, X0, X0, get_cursor_y() + 1);
    getchar(); getchar();
    break;
}
case 10:
{
    SaveSetData(L);
    printf("数据保存成功\n");
    getchar(); getchar();
    break;
}

```



```

    }
case 11:
{
    int c=1;
    if (L.listlenth)
    {
        printf("重新加载数据会丢失当前数据，是否继续? \n");
        printf("1.是 0.否\n");
        scanf("%d", &c);
        if (c == 0)
        {
            printf("未加载数据\n");
            getchar(); getchar();
            break;
        }
        else
        {
            for (int i = 0; i < L.listlenth; i++)
                set_destroy(L.pT[i].pRoot); //销毁所有集合，重新加载
            L.listlenth = 0;
        }
    }
    if (c == 0)
    {
        printf("未加载数据\n");
        getchar(); getchar();
        break;
    }
    if (LoadSetData(L))

```

```

        printf("数据加载成功\n");
    else printf("数据加载失败\n");
    getchar(); getchar();
    break;
}
case 12:
{
    printf("请选择第一个集合（运算结果为第一个集合  $\cup$  第二个集合）\n");

    int op1 = SelectSet(L);
    printf("请选择第二个集合");
    int op2 = SelectSet(L);

    set_init(L.pT[L.listlenth].pRoot); //在线性表表尾创建集合
    printf("输入保存运算结果的集合的名称:\n");
    scanf_s("%s", L.pT[L.listlenth].name, 20);
    printf("集合%s 创建成功! \n", L.pT[L.listlenth].name);

    printf("集合%s: ", L.pT[op1].name); set_traverse(L.pT[op1].pRoot, visit);
    putchar('\n');

    graphAVL(L.pT[op1].pRoot, X0, X0, get_cursor_y());
    setxy(0, get_cursor_y() + 4); //重新定位光标在新的一行

    printf("集合%s: ", L.pT[op2].name); set_traverse(L.pT[op2].pRoot, visit);
    putchar('\n');

    graphAVL(L.pT[op2].pRoot, X0, X0, get_cursor_y());
    setxy(0, get_cursor_y() + 4);

    set_union(L.pT[L.listlenth].pRoot, L.pT[op1].pRoot);

```

```

set_union(L.pT[L.listlenth].pRoot, L.pT[op2].pRoot);//并集
printf("集合%s 并集合%s: ", L.pT[op1].name, L.pT[op2].name);
set_traverse(L.pT[L.listlenth].pRoot, visit);
putchar('\n');
graphAVL(L.pT[L.listlenth].pRoot, X0, X0, get_cursor_y());
setxy(0, get_cursor_y() + 4);
printf("是否保存运算结果数据? 1.是 0.否\n");
int c;
scanf("%d", &c);
if (c == 0)//不保存
{
    set_destroy(L.pT[L.listlenth].pRoot);
}
else L.listlenth++;
getchar(); getchar();
break;
}
case 13:
{
    printf("请选择第一个集合（运算结果为第一个集合  $\cap$  第二个集合）
\n");

    int op1 = SelectSet(L);
    printf("请选择第二个集合");
    int op2 = SelectSet(L);

    set_init(L.pT[L.listlenth].pRoot);//在线性表表尾创建集合
    printf("输入保存运算结果的集合的名称:\n");
    scanf_s("%s", L.pT[L.listlenth].name, 20);
    printf("集合%s 创建成功! \n", L.pT[L.listlenth].name);

```

```

printf("集合%s: ", L.pT[op1].name); set_traverse(L.pT[op1].pRoot, visit);
putchar('\n');

graphAVL(L.pT[op1].pRoot, X0, X0, get_cursor_y());
setxy(0, get_cursor_y() + 4); //重新定位光标在新的一行

printf("集合%s: ", L.pT[op2].name); set_traverse(L.pT[op2].pRoot, visit);
putchar('\n');

graphAVL(L.pT[op2].pRoot, X0, X0, get_cursor_y());
setxy(0, get_cursor_y() + 4);

set_intersection(L.pT[op1].pRoot, L.pT[op2].pRoot,
L.pT[L.listlenth].pRoot); //  $T_1 \cap T_2$ 

printf("集合%s  $\cap$  集合%s: ", L.pT[op1].name, L.pT[op2].name);
set_traverse(L.pT[L.listlenth].pRoot, visit);
putchar('\n');
graphAVL(L.pT[L.listlenth].pRoot, X0, X0, get_cursor_y());
setxy(0, get_cursor_y() + 4);
printf("是否保存运算结果数据? 1.是 0.否\n");
int c;
scanf("%d", &c);
if (c == 0)
{
    set_destroy(L.pT[L.listlenth].pRoot);
}
else L.listlenth++;
getchar(); getchar();
break;
}

```

```

case 14:
{
    printf("请选择第一个集合(判断第一个集合是否和第二个集合相
等)\n");

    int op1 = SelectSet(L);
    printf("请选择第二个集合\n");
    int op2 = SelectSet(L);

    printf("集合%s: ", L.pT[op1].name); set_traverse(L.pT[op1].pRoot, visit);
    putchar('\n');

    graphAVL(L.pT[op1].pRoot, X0, X0, get_cursor_y());
    setxy(0, get_cursor_y() + 4); //重新定位光标在新的一行

    printf("集合%s: ", L.pT[op2].name); set_traverse(L.pT[op2].pRoot, visit);
    putchar('\n');

    graphAVL(L.pT[op2].pRoot, X0, X0, get_cursor_y());
    setxy(0, get_cursor_y() + 4);

    if (set_equal(L.pT[op1].pRoot, L.pT[op2].pRoot))
    {
        printf("集合%s = 集合%s\n", L.pT[op1].name, L.pT[op2].name);
    }
    else printf("集合%s != 集合%s\n", L.pT[op1].name, L.pT[op2].name);
    getchar(); getchar();
    break;
}

case 0:
{
    for (int i = 0; i < L.listlenth; i++)

```

```

        set_destroy(L.pT[i].pRoot); //销毁所有集合，重新加载
        free(L.pT);
        break;
    }
} //end of switch
} //end of while
}

void printKey(BSTree p, int x, int y)
{
    setxy(x, y);
    visit(p->data); //printf("(%d)", p->bf);
}

void graphAVL(BSTree T, int x, int l, int y)
{
    if (!T) return; //T 为空
    printKey(T, x, y); //输出根
    if (T->lchild || T->rchild) { setxy(x, ++y); putchar('|'); } //如果根有孩子，在根正
    下方输出大括号箭头
    if (T->lchild)
    { //有左孩子，输出大括号左半部分
        setxy(x - l / 2 + 1, y);
        int i;
        for (i = 0; i < l / 2 - 1; i++)
            putchar('_');
    }
    if (T->rchild)
    { //有右孩子，输出大括号右半部分
        setxy(x + 1, y);
        int i;

```

```

        for (i = 0; i < l / 2 - 1; i++)
            putchar('_');
    }
    graphAVL(T->lchild, x - l / 2, l / 2, y + 1); //输出左子树
    graphAVL(T->rchild, x + l / 2, l / 2, y + 1); //输出右子树
}

int get_cursor_x(void)
{
    HANDLE      gh_std_out;
    gh_std_out = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_SCREEN_BUFFER_INFO    bInfo;
    GetConsoleScreenBufferInfo(gh_std_out, &bInfo);
    return bInfo.dwCursorPosition.X; //返回当前光标的横坐标
}

int get_cursor_y(void)
{
    HANDLE      gh_std_out;
    gh_std_out = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_SCREEN_BUFFER_INFO    bInfo;
    GetConsoleScreenBufferInfo(gh_std_out, &bInfo);
    return bInfo.dwCursorPosition.Y; //返回当前光标的纵坐标
}

void setxy(short x, short y)
{ //将光标定位到(x,y)处
    COORD coord = { x, y };
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

Status visit(Info e)
{

```

```

    printf("%d ", e.id);
    return TRUE;
}

int SelectSet(SqList L)
{
    int i, op;
    printf("请选择集合(1-%d)\n", L.listlenth);
    for (i = 1; i <= L.listlenth; i++)
    {
        printf("      %d. %s\n", i, L.pT[i - 1].name);
    }
    scanf("%d", &op);
    while (op < 1 || op > L.listlenth)
    {
        printf("输入错误，请重新输入\n");
        getchar(); getchar();
        scanf("%d", &op);
    }
    return op - 1;
}

Status SaveSetData(SqList L)
{
    FILE *fp;
    /*保存线性表信息*/
    if ((fp = fopen(L_DATA_NAME, "w")) == NULL)//打开文件失败
    {
        printf("File open error\n ");
        return FALSE;
    }
}

```



```

    fwrite(L.pT, sizeof(SetData), L.listlenth, fp); //线性表数据一次性写入到文件中
    fclose(fp);
    /*保存树结点信息*/
    fp = fopen(T_DATA_NAME, "w");
    int i;
    for (i = 0; i < L.listlenth; i++)
    {
        SaveSet(L.pT[i].pRoot, fp);
        fprintf(fp, " %d", FLAT);
        fputc('\n', fp);
    }
    fclose(fp);

    return TRUE;
}

Status SaveSet(BSTree T, FILE *fp)
{
    if (T)
    {
        SaveSet(T->lchild, fp);
        fprintf(fp, "%d ", T->data.id);
        SaveSet(T->rchild, fp);
    }
    return OK;
}

Status LoadSetData(SqList &L)
{
    /*写入线性表数据*/
    FILE *fp;

```

```

if ((fp = fopen(L_DATA_NAME, "r")) == NULL)//文件打开失败
{
    printf("File open error\n ");
    return FALSE;
}

while (fread(&L.pT[L.listlenth], sizeof(SetData), 1, fp))
{
    L.pT[L.listlenth].pRoot = NULL;
    L.listlenth++;//准备写入下一个数据
}
fclose(fp);

/*写入二叉树数据*/
if ((fp = fopen(T_DATA_NAME, "r")) == NULL)//文件打开失败
{
    printf("File open error\n ");
    return FALSE;
}
int i;
for (i = 0; i < L.listlenth; i++)
{
    LoadSet(L.pT[i].pRoot, fp);
}
return TRUE;//写入完毕
}

Status LoadSet(BSTree &T, FILE *fp)
{
    Info temp;

```

```

while (fscanf(fp, "%d", &temp.id) != EOF)
{
    if (temp.id == FLAT) return TRUE;

    temp.fans = temp.follows = temp.friends = temp.hobby = NULL;

    set_insert(T, temp);
}

return TRUE;
}

/* 人际关系模拟函数 */
Status vis(Info e)
{
    printf("%-4d %-8s|", e.id, e.name); space++; //space 调整输出格式，一行输出 8
    个信息

    if (space % 8 == 0)
    {
        int x = get_cursor_x();
        putchar('\n');

        for (int i = 0; i < x; i++) putchar('-');
        putchar('\n');
    }

    return TRUE;
}

void menu2(void)
{
    printf("\n\n");
    printf("-----\n");
    printf("      人际关系模拟演示 \n");
    printf("-----\n");
    printf("      1. 添加用户          2. 维护用户信息 \n");

```

```

printf("      3. 查询用户信息          4. 显示共同关注\n");
printf("      5. 显示共同爱好          6. 显示共同好友\n");
printf("      7. 加载数据                8. 保存数据\n");
printf("      9. 显示所有用户          10. 显示二度好友\n");
printf("      0. Exit\n");
printf("-----\n");
printf("      请选择你的操作[0~9]:");
}

void apply_display(void)
{
    BSTree Account, Hobby;//总用户集，总兴趣集

    set_init(Account);
    set_init(Hobby);

    load_data(Account, Hobby);
    system("pause");
    int op = 1;

    while (op) {
        system("cls");
        menu2();
        scanf("%d", &op);
        switch (op) {
            case 1:
                {
                    Info in;
                    if (data_input(Account, in)) { set_insert(Account, in); printf("新用户添
加成功\n"); }

```

```
else printf("新用户添加失败\n");

    getchar(); getchar();
    break;
}
case 2:
{
    maintain_data(Account,Hobby);
    getchar(); getchar();
    break;
}
case 3:
{
    account_data(Account, Hobby);
    getchar(); getchar();
    break;
}
case 4:
{
    common_follow(Account);
    getchar(); getchar();
    break;
}
case 5:
{
    common_hobby(Account,Hobby);
    getchar(); getchar();
    break;
}
```

case 6:

```
{
    common_friend(Account);
    getchar(); getchar();
    break;
}
```

case 7:

```
{
    load_data(Account,Hobby);
    getchar(); getchar();
    break;
}
```

case 8:

```
{
    save_data(Account,Hobby);
    proc();
    getchar(); getchar();
    break;
}
```

case 9:

```
{
    printf("用户: \n");
    set_traverse(Account, vis);
    putchar('\n');
    /*printf("爱好: \n");
    set_traverse(Hobby, vis);*/
    getchar(); getchar();
    break;
}
```

```

    case 10:
    {
        second_friend(Account);
        getchar(); getchar();
        break;
    }
    case 0:
        if (confirm_save()) { save_data(Account, Hobby); proc(); }
        destroy_accountset(Account);
        set_destroy(Hobby);
        getchar(); getchar();
        break;
    } //end of switch
} //end of while
}

void common_follow(BSTree Account)
{
    int id1, id2;
    printf("请分别输入两位用户 id: \n");
    scanf("%d %d", &id1, &id2);
    if (!set_member(Account, id1) || !set_member(Account, id2))
        printf("用户不存在\n");
    else {
        BSTree common_follows;
        set_init(common_follows);
        BSTree p1, p2;
        p1 = set_member(Account, id1); p2 = set_member(Account, id2); //定位两位
        用户
        set_intersection(p1->data.follows, p2->data.follows, common_follows); // 两

```

位用户关注人集的交集

```

    printf("\"%s\"和\"%s\"的共同关注:\n", p1->data.name, p2->data.name);
    printf("-----\n");
    space = 0;
    print_account_data(Account, common_follows);
    set_destroy(common_follows);

    {
        printf("\n\n\n");
        system("pause");
        printf("\"%s\"关注人:\n", p1->data.name);
        space = 0;
        print_account_data(Account, p1->data.follows);
        printf("\n\n\"%s\"关注人:\n", p2->data.name);
        space = 0;
        print_account_data(Account, p2->data.follows);
    }
}

void common_friend(BSTree Account)
{
    int id1, id2;
    printf("请分别输入两位用户 id: \n");
    scanf("%d %d", &id1, &id2);
    if (!set_member(Account, id1) || !set_member(Account, id2))
        printf("用户不存在\n");
    else
    {
        BSTree common_friend;

```



```

    set_init(common_friend);

    BSTree p1, p2;

    p1 = set_member(Account, id1); p2 = set_member(Account, id2); //定位两位
    用户

    set_intersection(p1->data.friends, p2->data.friends, common_friend); //两位
    用户朋友集的交集

    //set_traverse(common_friend, visit);

    printf("\n%s"和"%s"的共同好友:\n", p1->data.name, p2->data.name);
    printf("-----\n");

    space = 0;

    print_account_data(Account, common_friend);

    set_destroy(common_friend);

    {
        printf("\n\n"); system("pause");
        printf("\n\n");
        printf("\n%s"好友:\n", p1->data.name);
        space = 0;
        print_account_data(Account, p1->data.friends);

        printf("\n\n%s"好友:\n", p2->data.name);
        space = 0;
        print_account_data(Account, p2->data.friends);
    }
}

void common_hobby(BSTree Account, BSTree Hobby)
{
    int id1, id2;

    printf("请分别输入两位用户 id: \n");

```

```

scanf("%d %d", &id1, &id2);
if (!set_member(Account, id1) || !set_member(Account, id2))
    printf("用户不存在\n");
else
{
    BSTree common_hobby;
    set_init(common_hobby);
    BSTree p1, p2;
    p1 = set_member(Account, id1); p2 = set_member(Account, id2); //定位两位
    用户
    set_intersection(p1->data.hobby, p2->data.hobby, common_hobby);
    printf("\n%s"和"%s"的共同爱好:\n", p1->data.name, p2->data.name); //两
    位用户爱好集的交集
    printf("-----\n");
    space = 0;
    print_account_data(Hobby, common_hobby);
    set_destroy(common_hobby);
    {
        printf("\n\n"); system("pause");
        printf("\n%s"爱好:\n", p1->data.name);
        space = 0;
        print_account_data(Hobby, p1->data.hobby);
        printf("\n\n%s"爱好:\n", p2->data.name);
        space = 0;
        print_account_data(Hobby, p2->data.hobby);
    }
}
}

void second_friend(BSTree Account)

```

```

{
    int id;
    printf("请输入用户 id\n");
    scanf("%d", &id);
    BSTree p; BSTree T = NULL;
    p = set_member(Account, id);
    if (!p)
    {
        printf("用户不存在\n"); return;
    }

    int op = 1;
    while (op)
    {
        system("cls");
        printf("%s\n", p->data.name);
        printf("1. 查询\"%s\"关于某个好友的二度好友      2.查询\"%s\"所有二
度好友\n", p->data.name, p->data.name);
        printf("0. 退出\n");
        scanf("%d", &op);
        switch (op)
        {
        case 1:
        {
            int friend_id;
            printf("\"%s\"的好友:\n", p->data.name);
            space = 0;
            print_account_data(Account, p->data.friends);
            printf("\n 请输入用户好友 id\n");

```

```

scanf("%d", &friend_id);
while(!set_member(p->data.friends, friend_id))
{
    printf("好友不存在，请重新输入\n");
    scanf("%d", &friend_id);
}
BSTree pfriend; BSTree T = NULL;
pfriend = set_member(Account, friend_id);
set_difference(pfriend->data.friends, p->data.friends, T); //求二度好友
set_remove(T, id); //二度好友除去自己
space = 0;
print_account_data(Account, T);
set_destroy(T); //销毁集合
getchar(); getchar(); break;

}

case 2:
{
    S_friend(Account, p->data.friends, T); //获得好友的好友
    BSTree T1 = NULL;
    set_difference(T, p->data.friends, T1); //二度好友中除去自己的好友
    set_remove(T1, p->data.id); //二度好友中除去自己
    space = 0;
    printf("\'%s\'的二度好友: (%d 位) \n", p->data.name, set_size(T1)); //
    两位用户爱好集的交集

    print_account_data(Account, T1); //输出二度好友
    set_destroy(T); set_destroy(T1); //销毁集合
    getchar(); getchar(); break;
}

```

```

    }
    default: break;
    }
}

void S_friend(BSTree Account, BSTree p, BSTree &T)
{//p 为某个用户的好友集，求该用户好友的好友的并集于 T 中
    if (p)
    {
        BSTree p_friend = set_member(Account, p->data.id); //定位好友节点
        set_union(T, p_friend->data.friends); //添加该好友的好友于 T 中
        S_friend(Account, p->lchild, T); //在左子树中查找并添加二度好友
        S_friend(Account, p->rchild, T); //在右子树中查找并添加二度好友
    }

}

Status destroy_accountset(BSTree &T)
{
    if (T)
    {
        destroy_accountset(T->lchild);
        destroy_accountset(T->rchild);
        set_destroy(T->data.fans);           set_destroy(T->data.follows);
        set_destroy(T->data.friends); set_destroy(T->data.hobby); //销毁个人关系集
        free(T); //释放节点
        T = NULL;
    }
    return TRUE;
}

```

```

Status data_input(BSTree T, Info &in)
{
    Info temp;
    printf("请输入新用户 id:\n");
    scanf("%d", &temp.id);
    while (set_member(T, temp.id)) { printf("用户%d 已存在，请重新输入(输入 0
退出)\n", temp.id); scanf("%d", &temp.id); if (temp.id== 0 )return FALSE; }
    printf("请输入用户姓名:\n");
    scanf("%s", &temp.name);
    temp.follows = temp.friends = temp.fans = temp.hobby = NULL;//新用户无粉丝
    好友等信息
    if (confirm_save()) { in = temp; return TRUE; }
    else return FALSE;
}

int confirm_save(void)
{
    printf("是否保存?\n");
    printf("1.是  0.否\n");
    int c;
    scanf("%d", &c);
    while (c != 0 && c != 1)
    {
        printf("输入错误，请重新输入\n");
        scanf("%d", &c);
    }
    return c;
}

int select_display(void)
{

```

```

printf("请选择演示功能： \n");
printf("1.集合操作演示  2.人际关系模拟演示\n");
printf("3.AVL 树演示      0.退出系统\n");
int c;
scanf("%d", &c);
getchar();
while (c != 1 && c != 2 && c != 0&&c!=3)
{
    printf("输入错误，请重新输入\n");
    scanf("%d", &c);
    getchar();
}
return c;
}

void maintain_data(BSTree &Account,BSTree Hobby)
{
    int id;
    printf("请输入用户 id\n");
    scanf("%d", &id);
    BSTNode *pt;
    pt = set_member(Account, id);
    if (!pt)
    {
        printf("用户不存在\n");
    }
    else
    {
        int op = 1;
        while (op)

```

```

    {
        system("cls");
        printf("请选择维护用户\"%s\"的内容\n", pt->data.name);
        printf("          1. 维护用户朋友集          2. 维护用户关注
人集 \n");
        printf("          3. 维护用户爱好集          4. 移除用户粉丝
\n");

        printf("          5. 修改用户姓名\n");
        printf("          0. 退出维护\n");

        scanf("%d", &op);
        switch (op)
        {
        case 1:
            {
                int o;
                printf("用户\"%s\"的朋友\n", pt->data.name);
                space = 0; print_account_data(Account, pt->data.friends);
                printf("\n 请选择操作:\n");
                printf("          1. 添加朋友          2. 删除朋友 \n");
                printf("          0. 退出 \n");
                scanf("%d", &o);
                if (o == 1)
                {
                    Info new_friend;
                    printf("请输入新朋友 id\n");
                    scanf("%d", &new_friend.id);
                    if (!set_member(Account, new_friend.id)) printf("用户%d 不存
在\n", new_friend.id);

```



```

        else
        {
            if (set_insert(pt->data.friends, new_friend))//插入到用户
*pt 的朋友集中
            {
                BSTNode *p = set_member(Account, new_friend.id);
                set_insert(p->data.friends, pt->data);//朋友是相互的

                printf("添加成功\n");
            }
            else printf("用户%d 已经是%s 的朋友", new_friend.id,
pt->data.name);//元素插入失败
        }
    }
    else if (o == 2)
    {
        int friend_id;
        printf("请输入要删除的朋友 id\n");
        scanf("%d", &friend_id);
        if (set_member(pt->data.friends, friend_id))
        {
            set_remove(pt->data.friends, friend_id);
            BSTNode *p = set_member(Account, friend_id);
            set_remove(p->data.friends, id);//移除朋友是相互的
            printf("删除成功\n");
        }
        else printf("%d 不是\"%s\"的朋友\n", friend_id, pt->data.name);
    }
    getchar(); getchar();

```

```

        break;
    }
case 2:
{
    int o;
    printf("\'%s\'的关注人:\n", pt->data.name);
    space = 0; print_account_data(Account, pt->data.follows);
    printf("\n 请选择操作:\n");
    printf("          1. 添加关注人          2. 移除关注人
\n");

    printf("          0. 退出 \n");
    scanf("%d", &o);
    if (o == 1)
    {
        Info new_follow;
        printf("请输入新关注人 id\n");
        scanf("%d", &new_follow.id);
        if (!set_member(Account, new_follow.id)) printf("用户%d 不存
在\n", new_follow.id);
        else
        {
            if (set_insert(pt->data.follows, new_follow))
            {//为关注人添加粉丝
                printf("添加成功\n");
                BSTNode *p;
                Info new_fan;
                new_fan.id = id;
                p = set_member(Account, new_follow.id);//p 指向新关
注人

```

```

        set_insert(p->data.fans, new_fan); //添加粉丝
    }
    else printf("用户 %d 已经是 \"%s\" 的关注人\n",
new_follow.id, pt->data.name);
    }
}
else if (o == 2)
{
    int follow_id;
    printf("请输入要删除的关注人 id\n");
    scanf("%d", &follow_id);
    if (set_member(pt->data.follows, follow_id))
    {
        set_remove(pt->data.follows, follow_id); //移除关注人
        set_remove(set_member(Account, follow_id)->data.fans,
id); //移除被关注人的粉丝
        printf("删除成功\n");
    }
    else printf("关注人%d 不存在\n", follow_id);
}
getchar(); getchar();
break;
}
case 3:
{
    int o;
    printf("\"%s\" 的爱好:\n", pt->data.name);
    space = 0; print_account_data(Hobby, pt->data.hobby);
    printf("\n 请选择操作:\n");

```

```

printf("          1. 添加爱好          2. 移除爱好 \n");
printf("          0. 退出 \n");
scanf("%d", &o);
if (o == 1)
{
    Info new_hobby;
    printf("请输入新爱好 id\n");
    space = 0; set_traverse(Hobby, vis); putchar('\n');
    scanf("%d", &new_hobby.id);
    if (!set_member(Hobby, new_hobby.id)) printf("爱好%d 不存在\n", new_hobby.id);
    else
    {
        if (set_insert(pt->data.hobby, new_hobby))//插入到用户*pt
            的爱好集中

            printf("添加成功\n");
            else printf("爱好%d 已存在\n", new_hobby.id);
    }
}
else if (o == 2)
{
    int hobby_id;
    printf("请输入要删除的爱好 id\n");
    scanf("%d", &hobby_id);
    if          (set_member(pt->data.hobby,          hobby_id))
{ set_remove(pt->data.hobby, hobby_id); printf("爱好移除成功\n"); }
    else printf("%d 不是\"%s\"的爱好\n", hobby_id, pt->data.name);
}
getchar(); getchar();

```

```
        break;
    }
    case 4:
    {
        printf("\"%s\" 的粉丝:\n", pt->data.name);
        space = 0; print_account_data(Account, pt->data.fans);
        putchar('\n');
        int fan_id;
        printf("请输入要删除的粉丝 id\n");
        scanf("%d", &fan_id);
        if (set_member(pt->data.fans, fan_id)) { set_remove(pt->data.fans,
fan_id); printf("粉丝移除成功\n"); }
        else printf("%d 不是 \"%s\" 的粉丝\n", fan_id, pt->data.name);
        getchar(); getchar();
        break;
    }
    case 5:
    {
        printf("请输入新的姓名\n");
        char new_name[20];
        scanf("%s", new_name);
        strcpy(pt->data.name, new_name);
        printf("姓名修改成功.\n");
        getchar(); getchar();
        break;
    }
    default:
        break;
}
```

```

        }
    }

}

Status save_account(BSTree T, FILE *fp)
{
    if (T)//T 非空
    {

        save_account(T->lchild, fp);

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

        save_account(T->rchild, fp);

    }

    return TRUE;
}

Status save_hobby(BSTree T, FILE *fp)
{
    if (T)//T 非空
    {

        save_hobby(T->lchild, fp);

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

        save_hobby(T->rchild, fp);

    }

    return TRUE;
}

Status save_relation(BSTree T, FILE *fp)
{
    if (T)

```

```

    {
        save_relation(T->lchild, fp); //保存左子树信息
        fprintf(fp, "%d\n", T->data.id); //保存节点 id
        save_rela(T->data.fans, fp); fprintf(fp, " 0\n");
        save_rela(T->data.follows, fp); fprintf(fp, " 0\n");
        save_rela(T->data.friends, fp); fprintf(fp, " 0\n");
        save_rela(T->data.hobby, fp); fprintf(fp, " 0\n\n"); //存储节点的关系信息,以
0 作为结束标记

        save_relation(T->rchild, fp); //保存右子树信息
    }
    return TRUE;
}

Status save_rela(BSTree T, FILE *fp)
{
    if (T)
    {

        save_rela(T->lchild, fp);
        fprintf(fp, "%d ", T->data.id);
        save_rela(T->rchild, fp);
    }
    return TRUE;
}

Status load_account(BSTree &T, FILE *fp)
{
    Info temp;
    while (fscanf(fp, "%d %s", &temp.id, temp.name) != EOF)
    {
        temp.fans = temp.follows = temp.friends = temp.hobby = NULL;
    }
}

```

```

        set_insert(T, temp);
    }
    return TRUE;
}

Status load_hobby(BSTree &Hobby, FILE *fp)
{
    Info temp;
    while (fscanf(fp, "%d %s", &temp.id, temp.name) != EOF)
    {
        set_insert(Hobby, temp);
    }
    return TRUE;
}

Status load_relation(BSTree &T, FILE *fp)
{
    Info temp;
    BSTNode *pt;
    int id1, id2;
    while (fscanf(fp, " %d", &id1) != EOF) //读取一个用户
    {
        pt = set_member(T, id1);
        fscanf(fp, "%d", &id2);
        while (id2 != 0) //加载用户粉丝集
        {
            temp.id = id2;
            set_insert(pt->data.fans, temp);
            fscanf(fp, "%d", &id2);
        }
    }
}

```



```

        fscanf(fp, "%d", &id2);
        while (id2 != 0)//加载用户关注人集
        {
            temp.id = id2;
            set_insert(pt->data.follows, temp);
            fscanf(fp, "%d", &id2);
        }

        fscanf(fp, "%d", &id2);
        while (id2 != 0)//加载用户朋友集
        {
            temp.id = id2;

            set_insert(pt->data.friends, temp);
            fscanf(fp, "%d", &id2);
        }

        fscanf(fp, "%d", &id2);
        while (id2 != 0)//加载用户兴趣集
        {
            temp.id = id2;
            set_insert(pt->data.hobby, temp);
            fscanf(fp, "%d", &id2);
        }

    }

    return TRUE;
}

Status load_data(BSTree &Account, BSTree &Hobby)

```

```
{

FILE *fp;

fp = fopen(ACCOUNTDATA, "rb");
if (!fp) { printf("账户数据加载失败\n"); } //文件打开失败
else {
    destroy_accountset(Account); //销毁已存在的集合，重新加载
    load_account(Account, fp);
    printf("账户数据加载成功\n");
    fclose(fp);

    fp = fopen(RELATIONDATA, "r");
    if (!fp) { printf("关系数据加载失败\n"); }
    else {

        load_relation(Account, fp);
        fclose(fp);
        printf("关系数据加载成功\n");
    }
}

fp = fopen(HOBBYDATA, "r");
if (!fp) { printf("兴趣数据加载失败\n"); } //文件打开失败
else {
    set_destroy(Hobby);
    load_hobby(Hobby, fp);
    printf("兴趣数据加载成功\n");
    fclose(fp);
}
```

```

    proc();
    return TRUE;
}
Status save_data(BSTree Account,BSTree Hobby)
{
    FILE *fp;
    fp = fopen(ACCOUNTDATA, "wb");
    save_account(Account, fp);
    fclose(fp);

    fp = fopen(HOBBYDATA, "w");
    save_hobby(Hobby,fp);
    fclose(fp);

    fp = fopen(RELATIONDATA, "w");
    save_relation(Account, fp);
    fclose(fp);
    printf("数据保存成功\n");

    return TRUE;
}
void account_data(BSTree Account, BSTree Hobby)
{
    system("cls");
    int id;
    printf("请输入查询的用户 id（输入 0 退出查询）\n");
    scanf("%d", &id);
    while (id)
    {

```

```

BSTNode *T = set_member(Account, id); //定位用户
if (!T) printf("用户%d 不存在\n", id);
else
{
    int op = 1;
    while (op!=5)
    {
        system("cls");
        printf("-----%d    %s-----\n", T->data.id,
T->data.name);

        printf("1. 好友集          2.关注人集\n");
        printf("3. 粉丝集          4.爱好集\n");
        printf("5. 重新选择查询用户    0.退出查询\n");

        scanf("%d", &op);
        if (op == 0) return;
        switch (op)
        {
            case 1:
            {
                printf("%d \"%s\"的朋友: (%d 位)\n", T->data.id, T->data.name,
set_size(T->data.friends)); space = 0;

                print_account_data(Account, T->data.friends); space = 0;
                putchar('\n'); //调整输入格式

                putchar('\n'); putchar('\n');
                getchar(); getchar();
                break;

```

```

    }
    case 2:
    {
        printf("%d \\\'%s\\\' 的关注人：( %d 位 ) \n", T->data.id,
T->data.name, set_size(T->data.follows));
        print_account_data(Account, T->data.follows); space = 0;
putchar('\n');

        putchar('\n'); putchar('\n');
        getchar(); getchar();
        break;
    }
    case 3:
    {
        printf("%d \\\\'%s\\\' 的粉丝：( %d 位 ) \n", T->data.id, T->data.name,
set_size(T->data.fans));
        print_account_data(Account, T->data.fans); space = 0;
putchar('\n');

        putchar('\n'); putchar('\n');
        getchar(); getchar();
        break;
    }

    case 4:
    {
        printf("%d \\\\'%s\\\' 的爱好：( %d 个 ) \n", T->data.id, T->data.name,
set_size(T->data.hobby));
        print_account_data(Hobby, T->data.hobby); space = 0;

```

```

putchar('\n');

                putchar('\n'); putchar('\n');
                getchar(); getchar();
                break;
            }
            default:
                break;
        }

    }
}

printf("请重新输入用户 id:(输入 0 退出查询)");
scanf("%d", &id);
printf("切换成功\n");
}
}

void print_account_data(BSTree Account, BSTree T)
{
    if (T)
    {
        print_account_data(Account, T->lchild);
        BSTree p = set_member(Account, T->data.id);
        if(p)
            vis(p->data);
        print_account_data(Account, T->rchild);
    }
}

void proc(void)
{

```

```

char buf[103];
memset(buf, ' ', sizeof(buf));
buf[0] = '[';
buf[101] = ']';
buf[102] = '\0';
int i = 0;
char index[6] = "-\\|/^0";
while (i <= 100)
{
    buf[i] = '=';
    printf("%s [%d%%][%c]r", buf, i, index[i % 4]);
    fflush(stdout);//刷新缓冲区
    Sleep(5);
    i++;
}

printf("\n");
}

void menu3(void)
{
    printf("\n\n");
    printf("-----\n");
    printf("      AVL 树操作演示 \n");
    printf("-----\n");
    printf("      1. InitAVL          5. SearchAVL\n");
    printf("      2. InsertAVL       6. TraverseAVL\n");
    printf("      3. DestroyAVL     \n");
    printf("      4. DeleteAVL\n");
}

```

```

    printf("      0. Exit\n");
    printf("-----\n");
    printf("    请选择你的操作[0~6]:");
}
void AVL_display(void)
{
    BSTree T;
    InitAVL(T);
    int op = 1, key;
    while (op) {
        system("cls");
        menu3();
        scanf("%d", &op);
        switch (op) {
        case 1:
            {
                set_init(T);
                printf("AVL 树 T 初始化成功\n");

                getchar(); getchar();
                break;
            }
        case 2:
            {
                int N;
                Info a;
                bool taller;
                printf("请输入数据的组数\n");
                scanf("%d", &N);
            }
        }
    }
}

```



```

while (N--)
{
    scanf("%d", &a.id);
    InsertAVL(T, a,taller);
}

graphAVL(T, X0, X0, get_cursor_y());
getchar(); getchar();
break;
}

case 3:
{
    DestroyAVL(T);
    printf("AVL 树销毁成功\n");
    getchar(); getchar();
    break;
}

case 4:
{
    bool shorter;
    graphAVL(T, X0, X0, get_cursor_y());
    setxy(0, get_cursor_y() + 2);
    printf("输入要删除的节点\n");
    scanf("%d", &key);
    if (DeleteAVL(T, key,shorter)) printf("删除成功\n");
    else printf("节点不存在\n");
    graphAVL(T, X0, X0, get_cursor_y());
    getchar(); getchar();
    break;
}

```

```

case 5:
{
    printf("请输入节点\n");
    int key;
    scanf("%d", &key);
    if (SearchAVL(T, key)) printf("%d 节点在树中", key);
    else printf("节点不存在\n");
    graphAVL(T, X0, X0, get_cursor_y());
    getchar(); getchar();
    break;
}

case 6:
{
    InOrderTraverse(T, visit);
    graphAVL(T, X0, X0, get_cursor_y());
    getchar(); getchar();
    break;
}

case 0:
    set_destroy(T); //退出前销毁集合 T
    break;
} //end of switch
} //end of while
}

```