数据结构课程任务书

□ 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。

□ 设计要求

（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。

□ 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

1 引言

1.1 课题背景与意义

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。平衡二叉树支持动态的查找和插入，将其时间复杂度控制在 O（logn），效率十分高。在本课题中将其应用到实际生活中，查询社交网络中人与人之间的关系，具有实用意义。

1.2 国内外研究现状

AVL 作为最早的平衡二叉树，在应用上较其他数据结构较少，国内外研究更多的是更为复杂的结构，例如红黑树，B 树，B+树，Trie 树。linux 进程调度 Completely Fair Scheduler，用红黑树管理进程控制块；epoll 在内核中的实现，用红黑树管理事件块；Java 中 Treemap 的实现等；B 树，B+树主要用于文件系统以及数据库中做索引等；Trie 树用于前缀匹配等。

1.3 课程设计的主要研究工作

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。

在 AVL 树的基本操作中，稍微复杂的是 AVL 树的删除操作，维护其动态平衡。在做课程设计的过程中，通过查找文献，实现 AVL 树的删除，这是整个课程设计的难点，也是主要研究工作。

2 系统需求分析与总体设计

2.1 系统需求分析

构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出相应操作提示显示。

演示系统可同时实现平衡二叉树的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存平衡二叉树数据逻辑结构(D,{R})的完整信息；②需要设计平衡二叉树文件保存和加载操作合理模式。

以二叉链表为存储结构，设计 AVL 的 6 种基本运算，以 AVL 树表示集合，实现集合抽象类型及其 11 中运算，以集合表示个人微博或社交网络中好友集，关注集，粉丝集等的初始化，增删改查，随机生成数据集，实现共同关注，共同喜好，二度好友等功能。

2.2 系统总体设计

演示系统由用户操作界面与功能调用部分组成。

用户操作界面输出可选的操作项、操作的结果及错误信息，基于 c 语言的标准输入输出库(stdio.h)及控制台的 cls 命令实现。

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

演示结构模块图如下图所示(如图 2-1 所示)

其中数据保存分为四个模块，前两个模块用于保存人物信息，后两个模块用于保存关系信息，如一个人的好友，关注，粉丝，爱好。

Show tree 模块能输出图化平衡二叉树，显示树的形状。

Create at random 随机生成大数据集，便于后面的功能测试。

Insert name 模块用于插入新成员的时候插入姓名。

Options for friends，fans，following，likes 分别是对成员的好友集，粉丝集，关注集，爱好集执行初始化，增删改查等操作。

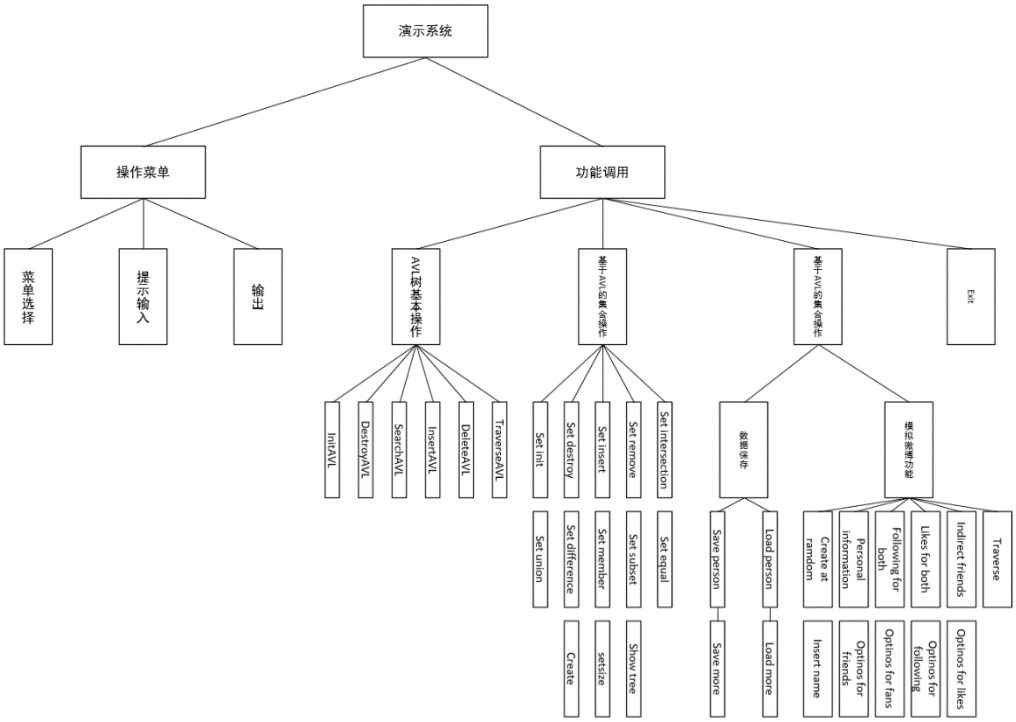


图 2-1 演示系统模块结构图

3 系统详细设计

3.1 有关数据结构的定义

系统要处理二叉树节点信息，以及个人信息。个人信息包含个人关键字，姓，名，爱好，朋友节点，关注节点，粉丝节点，爱好节点。二叉树节点信息包括个人信息数据结构，二叉树状态，高度，左孩子节点，右孩子节点。（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
信息结构体data	Info	名firstname	Char[]
二叉树状态bflag	Int	爱好like	Char[]
高度h	Int	朋友节点friends	BATNode *
左孩子节点lchild	BATNode *	粉丝节点fans	BATNode *
右孩子节点rchild	BATNode *	关注节点following	BATNode *
关键字key	Int	爱好节点likes	BATNode *
姓lastname	Char[]		

多种数据间的关系如下图所示（如图3-1）

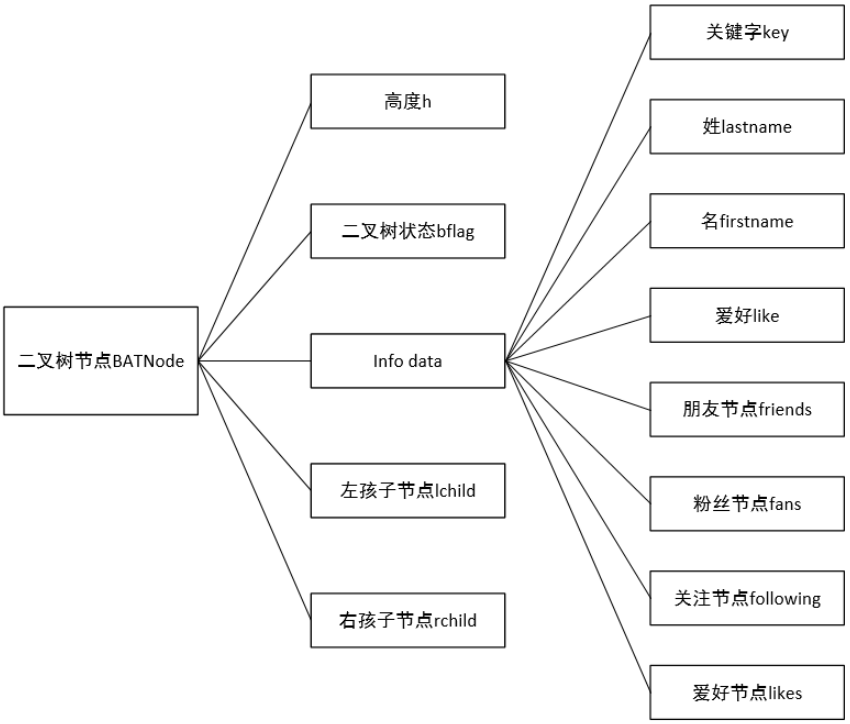


图3-1 多种数据之间的关系图

3.2 主要算法设计

1.L_Rotate(*p)算法思想：创建二叉树节点 temp，让 temp 指向 p 的右子树根节点，temp 左子树作为 p 的右子树，temp 左子树指向*p，*p 指向 temp。

时间复杂度：O（1）

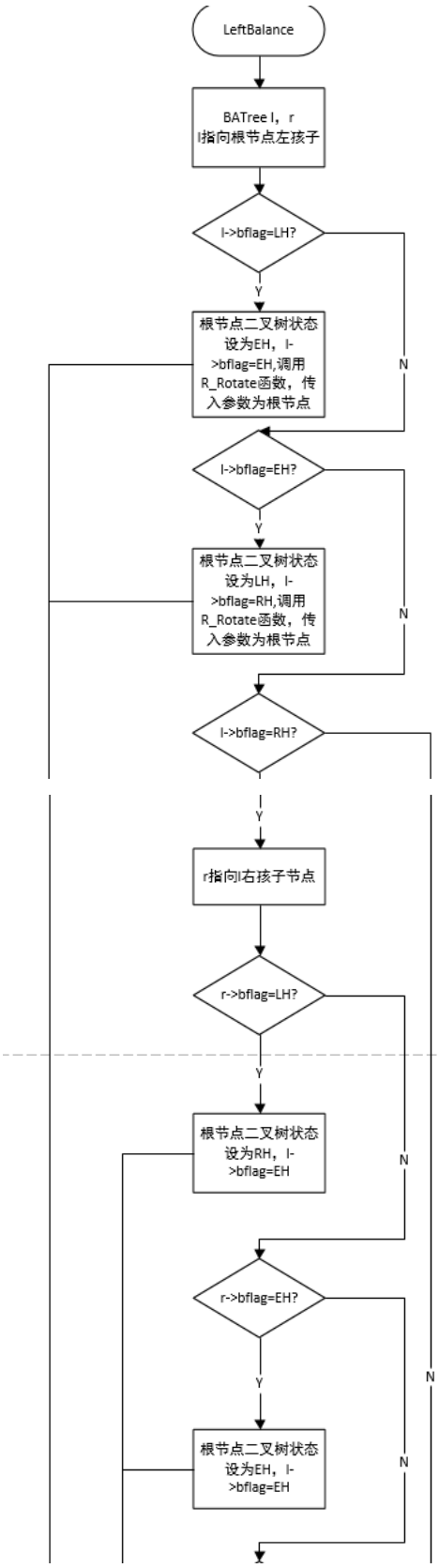
空间复杂度：O（1）

2.R_Rotate(*p)算法思想：创建二叉树节点 temp，让 temp 指向 p 的左子树根节点，temp 右子树作为 p 的左子树，temp 右子树指向*p，*p 指向 temp。

时间复杂度：O（1）

空间复杂度：O（1）

3.LeftBalance(*T)算法流程图：


LeftBalance
BATree l，r
l指向根节点左孩子
l->bflag=LH?
Y
N
根节点二叉树状态设为EH，l->bflag=EH,调用R_Rotate函数，传入参数为根节点
l->bflag=EH?
Y
N
根节点二叉树状态设为LH，l->bflag=RH,调用R_Rotate函数，传入参数为根节点
l->bflag=RH?
Y
N
r指向l右孩子节点
r->bflag=LH?
Y
N
根节点二叉树状态设为RH，l->bflag=EH
r->bflag=EH?
Y
N
根节点二叉树状态设为EH，l->bflag=EH
N

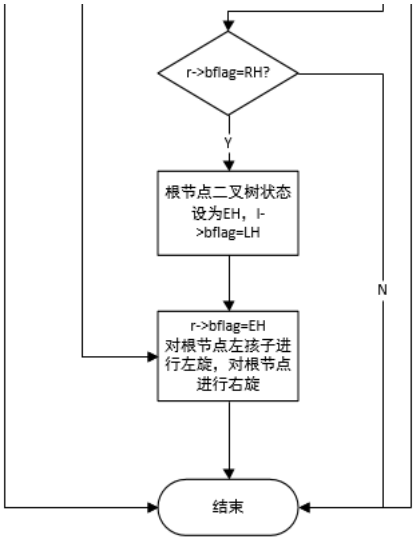
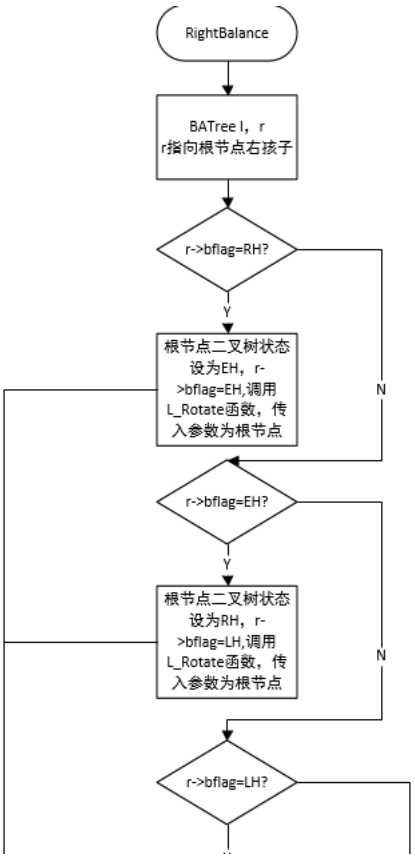


图 3-2 LeftBalance 函数流程图

时间复杂度：O（1）

空间复杂度：O（1）

4.RightBalance(*T)算法流程图：



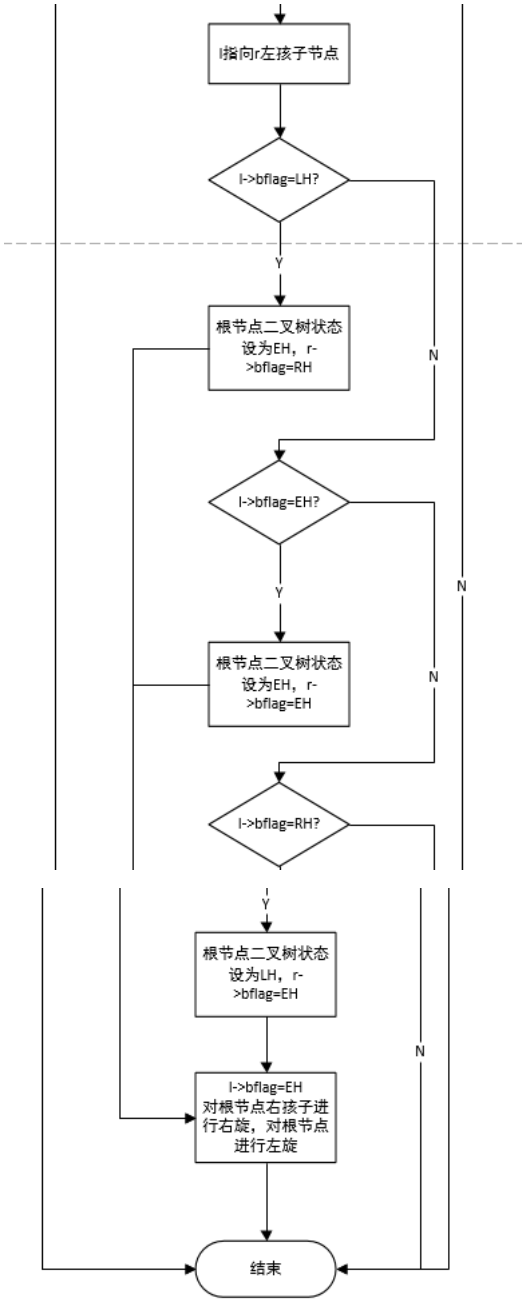


图 3-3 RightBalance 函数流程图

时间复杂度：O（1）

空间复杂度：O（1）

5.set_init(*T)算法思想：将*T 置空。

时间复杂度：O（1）

空间复杂度：O（1）

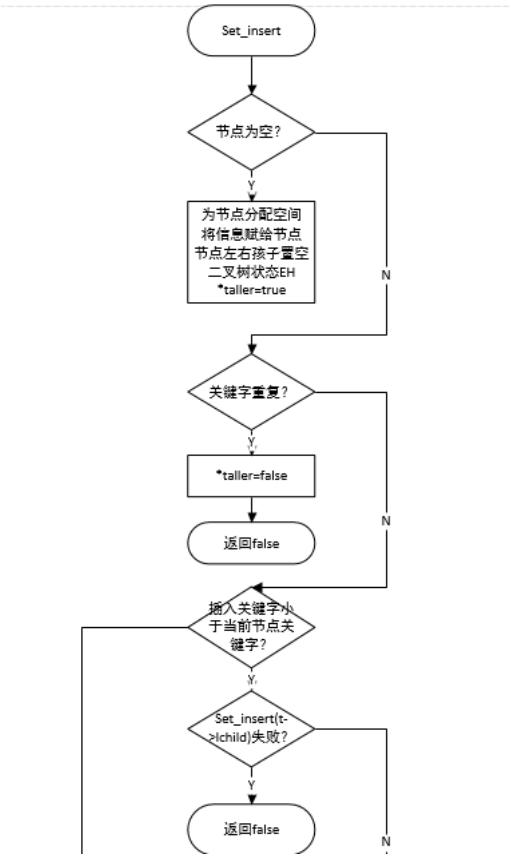
6.set_destroy(*T)算法思想：如果*T 不为空，则调用自身销毁左子树，调用

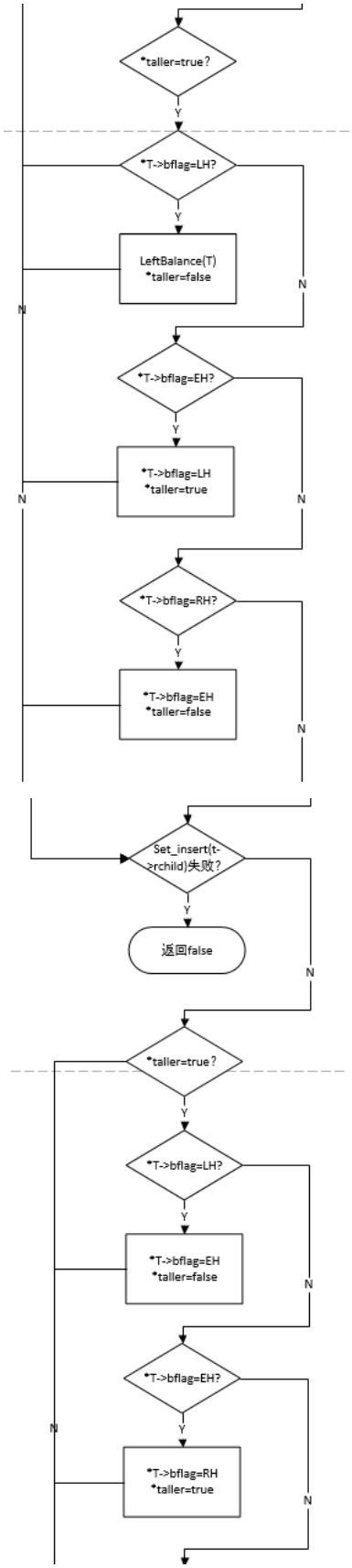
自身销毁右子树，释放*T 的空间。

时间复杂度：O（n）

空间复杂度：O（n）

7.set_insert (*T，e，*taller)算法流程图：




*taller=true?
Y
*T->bflag=LH?
Y
LeftBalance(T)
*taller=false
N
N
*T->bflag=EH?
Y
*T->bflag=LH
*taller=true
N
N
*T->bflag=RH?
Y
*T->bflag=EH
*taller=false
N
Set_insert(t->rchild)失败?
Y
返回false
N
*taller=true?
Y
*T->bflag=LH?
Y
*T->bflag=EH
*taller=false
N
*T->bflag=EH?
Y
*T->bflag=RH
*taller=true
N
N

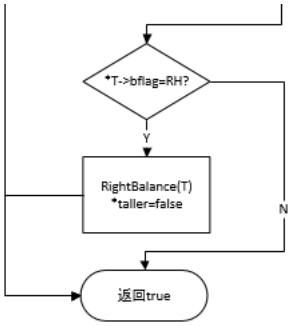
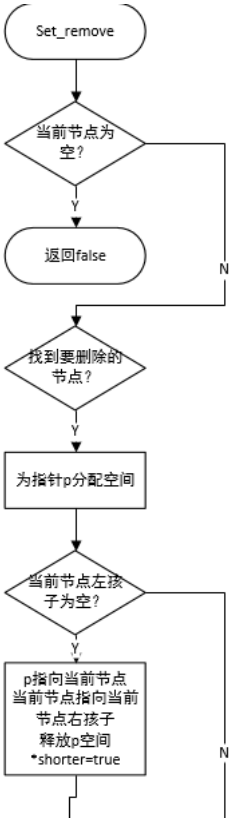


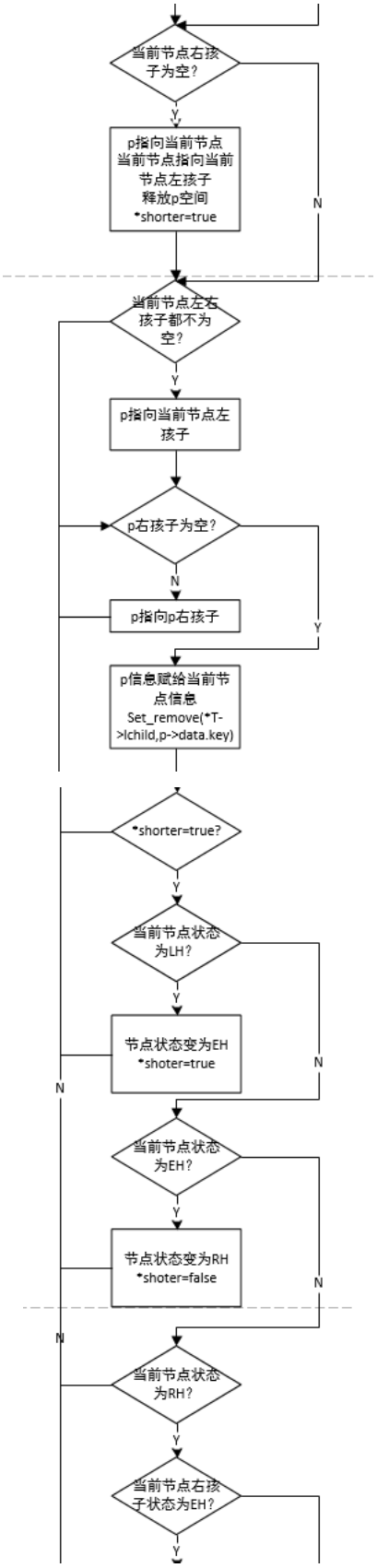
图 3-4 set_insert 函数流程图

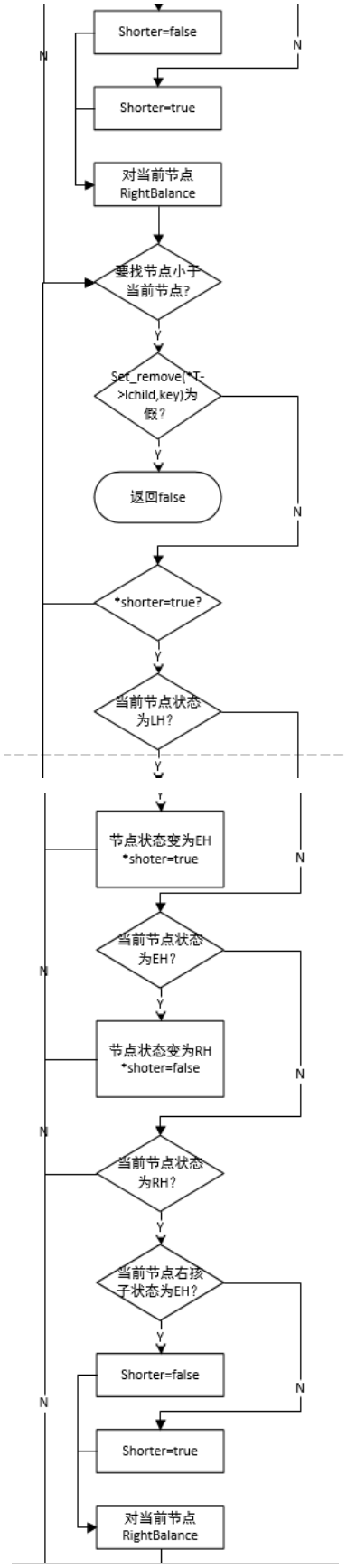
时间复杂度：O（logn）

空间复杂度：O（n）

8.set_remove(*T，key，*shorter)算法流程图：




当前节点右孩子为空?
Y
N
p指向当前节点
当前节点指向当前节点左孩子
释放p空间
*shorter=true
当前节点左右孩子都不为空?
Y
p指向当前节点左孩子
p右孩子为空?
N
Y
p指向p右孩子
p信息赋给当前节点信息
Set_remove(*T->lchild,p->data.key)
*shorter=true?
Y
N
当前节点状态为LH?
Y
N
节点状态变为EH
*shoter=true
当前节点状态为EH?
Y
N
节点状态变为RH
*shoter=false
当前节点状态为RH?
Y
N
当前节点右孩子状态为EH?
Y


Shorter=false
Shorter=true
对当前节点
RightBalance
要找节点小于
当前节点?
Set_remove(*T-
>lchild,key)为
假?
返回false
*shorter=true?
当前节点状态
为LH?
节点状态变为EH
*shoter=true
当前节点状态
为EH?
节点状态变为RH
*shoter=false
当前节点状态
为RH?
当前节点右孩
子状态为EH?
Shorter=false
Shorter=true
对当前节点
RightBalance
Y
N

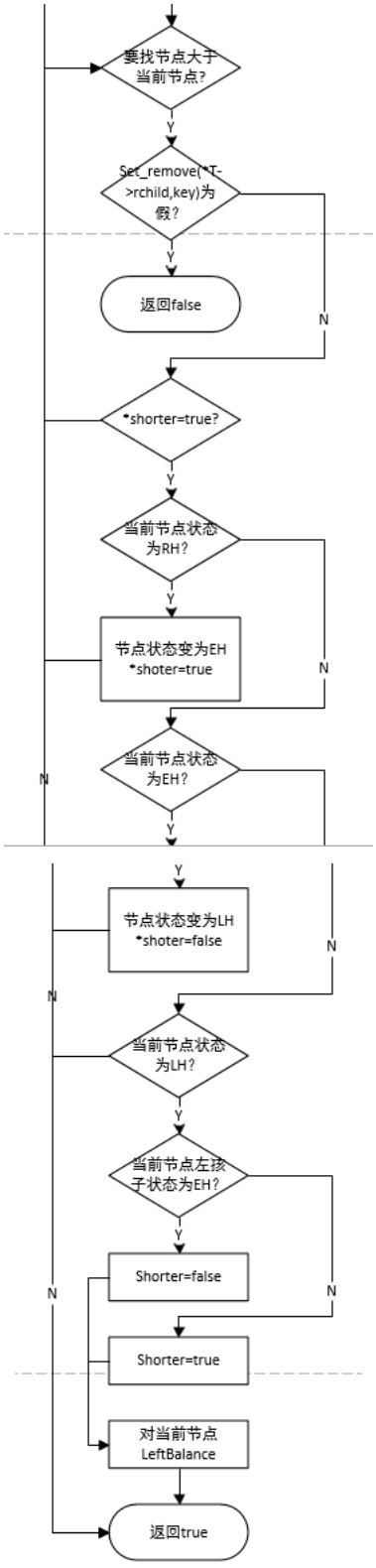


图 3-5 set_remove 函数流程图

时间复杂度：O（logn）

空间复杂度：O（n）

9.set_intersection(T，T1，*T0)算法思想：如果 T1 为空，则返回；如果 T1 是 T 中的成员，则将 T1 节点插入 T0 中；对 T1 左孩子递归调用自身函数，对 T1 右孩子递归调用自身函数。

时间复杂度：O（logn）

空间复杂度：O（n）

10.set_union(*T，T1)算法思想：如果 T1 为空，返回；将 T1 插入树 T 中，对 T1 左孩子递归调用自身函数，对 T1 右孩子递归调用自身函数。

时间复杂度：O（logn）

空间复杂度：O（n）

11.set_difference(*T，T1)算法思想：如果*T 为空或 T1 为空，返回；将 T1 节点从 T 中删除，对 T1 左孩子递归调用自身函数，对 T1 右孩子递归调用自身函数。

时间复杂度：O（logn）

空间复杂度：O（n）

12.set_size(T1，T)算法思想：如果 T1 为空，返回；如果 T1 是 T 中的成员，count（计数器）加 1，对 T1 左孩子递归调用自身函数，对 T1 右孩子递归调用自身函数。

时间复杂度：O（logn）

空间复杂度：O（n）

13.set_member(T，key，*Tsub)算法思想：如果*T 为空，返回 false；如果找到要找的节点，让 Tsub 指向 T，返回 true；否则当 key 小于当前节点的 key 时，对 T 的左子树调用自身函数，如果找到返回 true；再否则对 T 的右子树调用自身函数，如果找到返回 true；返回 false。

时间复杂度：O（logn）

空间复杂度：O（n）

14.set_subset(T，T1)算法思想：如果 T1 为空，返回 true；如果 T1 是 T 中的成员，对 T1 左孩子调用自身函数，如果为假，返回 false；对 T1 右孩子调用自身函数，如果为假，返回 false；返回 true。否则返回 false。

时间复杂度：O（logn）

空间复杂度：O（n）

15.set_equal(T,T1)算法思想：如果 T1 不是 T 的子集，返回 false；如果 T 不是 T1 的子集，返回 false；返回 true。

时间复杂度：O（logn）

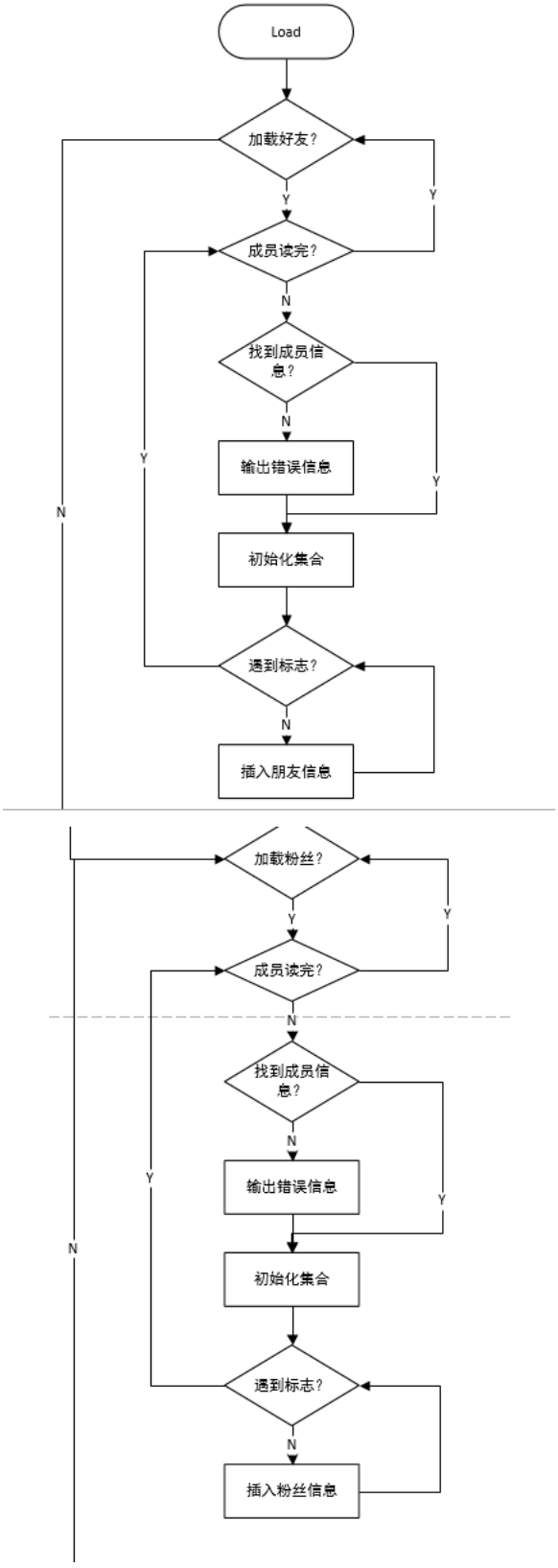
空间复杂度：O（n）

16.load_data(*T,*filename)算法思想：打开 filename，当没读到文件尾时，为 p 分配空间，读文件，将 p 节点插入到 T 中；关闭文件指针，计算 T 高度。

时间复杂度：O（nlogn）

空间复杂度：O（n）

17.load(*T)算法流程图：


Load
加载好友?
Y
成员读完?
N
找到成员信息?
N
输出错误信息
初始化集合
遇到标志?
N
插入朋友信息
N
Y
Y
Y
加载粉丝?
Y
成员读完?
N
找到成员信息?
N
输出错误信息
初始化集合
遇到标志?
N
插入粉丝信息
N
Y
Y
Y

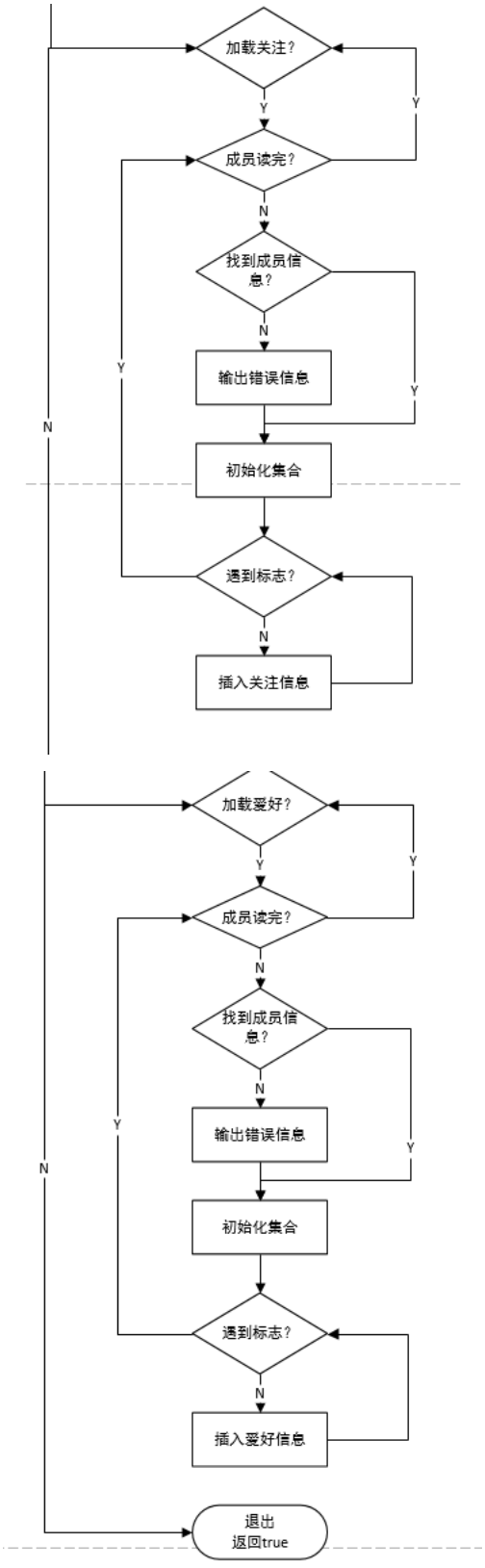


图 3-6 load 函数流程图

时间复杂度：O（nlogn）

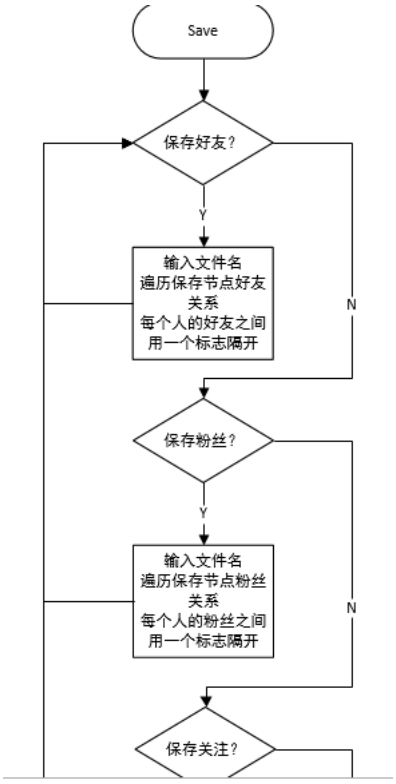
空间复杂度：O（n）

18.save_data(T，*fp)算法思想：如果 T 不为空，将 T 中信息写入文件；否则返回 true；对 T 左子树递归调用自身函数，不为真返回 false，对 T 右子树递归调用自身函数，不为真返回 false；返回 true。

时间复杂度：O（n）

空间复杂度：O（n）

19.save(T)算法流程图：



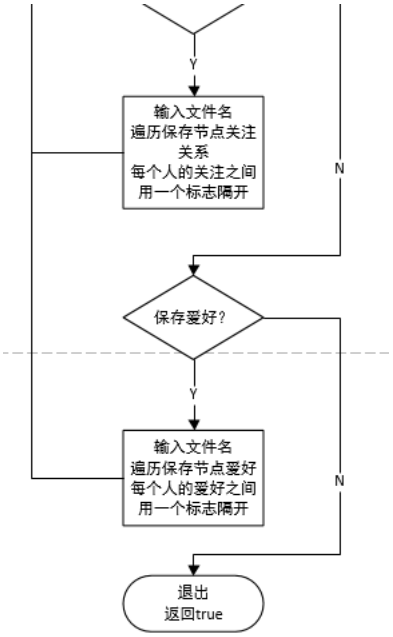


图 3-7 save 函数流程图

时间复杂度：O（n）

空间复杂度：O（n）

20.create(*T)算法思想：为信息 e 分配空间，输入关键字，插入到 T 中，之后给出提示信息，若输入 y 则继续插入，若输入 n 则停止插入，创建成功，返回。

时间复杂度：O（nlogn）

空间复杂度：O（n）

21.showtree(T，x，y，output_t)算法思想：如果 T 不为空，跳到指定位置输出关键字，如果左孩子存在，根据一定的数学运算，跳到指定位置输出关键字，如果右孩子存在，根据一定的数学运算，跳到指定位置输出关键字，递归调用函数本身，直到所有的节点都被访问到。

时间复杂度：O（n）

空间复杂度：O（n）

22.create at random(*T)算法思想：打开姓名文件，顺序读取人名，并按照顺序编号，将其信息初始化，节点插入到 T 中，打开爱好文件，将其中爱好插入到树 L 中。随机生成一个人的社交关系，递归实现，利用随机函数生成一个人

的好友集，粉丝集，关注集，爱好集。完善集合，保证好友是双向的，关注人的粉丝里有自己，粉丝的关注集里有自己，自己的关系中不出现自己。

时间复杂度：O（nlogn）

空间复杂度：O（n）

23.mutual following(T1，T2)算法思想：对 T1，T2 的关注集取交集，输出交集。

时间复杂度：O（logn）

空间复杂度：O（n）

24.mutual likes(T1，T2)算法思想：对 T1，T2 的爱好集取交集，输出交集。

时间复杂度：O（logn）

空间复杂度：O（n）

25.indirect friends(*T0，T，T1)算法思想：将要查询的成员 T1 的好友的所有好友都插入到 T0 中，求 T0 与 T1 好友集的差，最后删除 T1，输出 T0。

时间复杂度：O（nlogn）

空间复杂度：O（n）

26.InitAVL(*T)算法思想：同 set_init 在此不赘述。

时间复杂度：O（1）

空间复杂度：O（1）

27.DestroyAVL(*T)算法思想：同 set_destroy 在此不赘述。

时间复杂度：O（n）

空间复杂度：O（n）

28.SearchAVL(T)算法思想：同 set_member 在此不赘述。

时间复杂度：O（logn）

空间复杂度：O（1）

29.InsertAVL(*T)算法思想：同 set_insert 在此不赘述。

时间复杂度：O（logn）

空间复杂度：O（n）

30.DeleteAVL(*T)算法思想：同 set_remove 在此不赘述。

时间复杂度：O（logn）

空间复杂度：O（n）

31.TraverseAVL(T)算法思想：使用递归进行中序遍历。

时间复杂度：O（n）

空间复杂度：O（n）

32.options for friends(*T)算法思想：包含初始化，增删改查等功能；初始化将指定人员的好友集清空；插入好友，选择一位不是自己的好友插入，若好友存在提示错误信息，若好友不存在则插入好友，同时将这位好友的好友集中插入自己；删除好友，选择一位不是自己的好友删除，若好友存在则删除好友，同时将这位好友的好友集中删除自己，若好友不存在提示错误信息；查找好友判断要找的人员是不是自己的好友；遍历好友输出自己好友集所有好友。

时间复杂度：O（nlogn）

空间复杂度：O（n）

33.options for fans(*T)算法思想：包含初始化，增删改查等功能；初始化将指定人员的粉丝集清空；插入粉丝，选择一位不是自己的粉丝插入，若粉丝存在提示错误信息，若粉丝不存在则插入粉丝，同时将这位粉丝的关注集中插入自己；删除粉丝，选择一位不是自己的粉丝删除，若粉丝存在则删除粉丝，同时将这位好友的关注集中删除自己，若粉丝不存在提示错误信息；查找粉丝判断要找的人员是不是自己的粉丝；遍历粉丝输出自己粉丝集所有粉丝。

时间复杂度：O（nlogn）

空间复杂度：O（n）

34.options for following(*T)算法思想：包含初始化，增删改查等功能；初始化将指定人员的关注集清空；插入关注，选择一位不是自己的关注插入，若关注存在提示错误信息，若关注不存在则插入关注，同时将这位关注的粉丝集中插入自己；删除关注，选择一位不是自己的关注删除，若关注存在则删除关注，同时将这位关注的粉丝集中删除自己，若关注不存在提示错误信息；查找关注判断要找的人员是不是自己的关注；遍历关注输出自己关注集所有关注。

时间复杂度：O（nlogn）

空间复杂度：O（n）

35.options for likes(*T)算法思想：包含初始化，增删改查等功能；初始化将指定人员的爱好集清空；插入爱好，选择一个爱好插入，若爱好存在提示错误信息，若爱好不存在则插入爱好；删除爱好，选择一个爱好删除，若爱好存在则删除爱好，若爱好不存在提示错误信息；查找爱好判断要找的爱好是不是自己的爱好；遍历爱好输出自己爱好集所有爱好。

时间复杂度：O（nlogn）

空间复杂度：O（n）

4 系统实现与测试

4.1 系统实现

测试环境：Microsoft VisualStudio 2017 Community

数据类型的定义：

```
typedef struct Info{
    int key;//关键字
    char lastname[3];
    char firstname[5];
    char like[5];
    struct BATNode *friends;
    struct BATNode *fans;
    struct BATNode *following;
    struct BATNode *likes;
}Info;
typedef struct BATNode {
    Info data;
    int bflag;//二叉树状态
    int h;//高度
    struct BATNode *lchild, *rchild;
}BATNode, *BATree;
```

系统函数简要说明：

```
void L_Rotate(BATree *p);//左旋函数
void R_Rotate(BATree *p);//右旋函数
void LeftBalance(BATree *T);//对节点实现左平衡
void RightBalance(BATree *T);//对节点实现右平衡
void avl_init(BATree *T);//AVL树初始化
void avl_destroy(BATree *T);//AVL树销毁
```

```
bool avl_insert(BATree *T, Info e, bool *taller);//AVL树插入
bool avl_delete(BATree *T, int key, bool *shorter);//AVL树删除
void avl_traverse(BATree T);//AVL树遍历
bool avl_search(BATree T, int key, BATree *Tsub);//AVL树搜索
void set_init(BATree *T);//集合初始化
void set_destory(BATree *T);//集合销毁
bool set_insert(BATree *T, Info e, bool *taller);//集合插入
bool set_remove(BATree *T, int key, bool *shorter);//集合删除
void set_intersection(BATree T, BATree T1, BATree *T0);//求集合交集
void set_union(BATree *T, BATree T1);//求集合并集
void set_difference(BATree *T, BATree T1);//求集合差集
void set_size(BATree T1, BATree T);//求集合元素个数
bool set_member(BATree T, int key, BATree *Tsub);//判断是否为集合中成员
bool set_subset(BATree Tsub, BATree T1);//判断一个集合是否是另一个集合
的子集
bool set_equal(BATree T, BATree T1);//判断两个集合是否相等
bool load_data(BATree *T, char *filename);//加载主要的树
bool save_data(BATree T, FILE *fp);//保存主要的树
void input_key(Info *data);//输入关键字
void output_key(BATree T);//输出关键字
void create(BATree *T);//创建树
void height(BATree T, int i);//求树高度
void gotoxy(int x, int y);//锁定位置
void showtree(BATree T, int x, int y, visit fp);//显示树图
void InOrderTraverse(BATree T);//中序遍历
void output_relation(BATree T1, BATree T);//输出关系
void output_likes(BATree T1, BATree L);//输出爱好
bool input_pinfo(BATree *T, int p_size);//随机输入人员信息
bool input_likes(BATree *L, int *l_size);//随机生成爱好树
```

void input_relation(BATree *T, int p_size, int l_size);//输入关系信息

void input_information(BATree *T, int size, int max, int min);//输入个人信息

void improve_sets(BATree T, BATree T0);//完善随机生成的各种集合

void improve_friends(BATree T, BATree T1, BATree T2);//完善好友集

void improve_fans(BATree T, BATree T1, BATree T2);//完善粉丝集

void improve_following(BATree T, BATree T1, BATree T2);//完善关注集

void indirect_friends_traverse(BATree T, BATree T1, BATree *T0);//二度好友遍历

void indirect_friends_insert(BATree T, BATree *T0);//二度好友插入

void set_init_fors(BATree *T);//对一个人的好友集，粉丝集，关注集，爱好集等

bool insert_name(BATree T);//为一个节点赋名

bool init_friends(BATree T);//初始化好友集

bool insert_friends(BATree T);//插入好友

bool remove_friends(BATree T);//删除好友

bool search_friends(BATree T);//搜索好友

bool traverse_friends(BATree T);//遍历好友

bool init_fans(BATree T);//初始化粉丝集

bool insert_fans(BATree T);//插入粉丝

bool remove_fans(BATree T);//删除粉丝

bool search_fans(BATree T);//搜索粉丝

bool traverse_fans(BATree T);//遍历粉丝

bool init_following(BATree T);//初始化关注集

bool insert_following(BATree T);//插入关注

bool remove_following(BATree T);//删除关注

bool search_following(BATree T);//搜索关注

bool traverse_following(BATree T);//遍历关注

bool init_likes(BATree T,BATree L);//初始化爱好集

bool insert_likes(BATree T, BATree L);//插入爱好

bool remove_likes(BATree T, BATree L);//删除爱好

bool search_likes(BATree T, BATree L);//搜索爱好

bool traverse_likes(BATree T, BATree L);//遍历爱好

bool load(BATree *T);//加载更多信息

bool traverse_relation(BATree T, FILE *fp);//遍历人与人之间的关系

bool save(BATree T);//保存更多信息

bool save_relation(BATree T, FILE *fp, int i);//保存人与人之间的关系

利用函数实现各模块的功能：

在这里只叙述应用层的实现方法，其他都是函数名即功能。共同关注，共同爱好都是使用求交集的功能，将两个人对应的集合求交集即可。二度好友则是使用了求差集的功能，将一个人的好友所有好友形成的树减去这个人的好友树即可得二度好友。

函数间的调用关系：

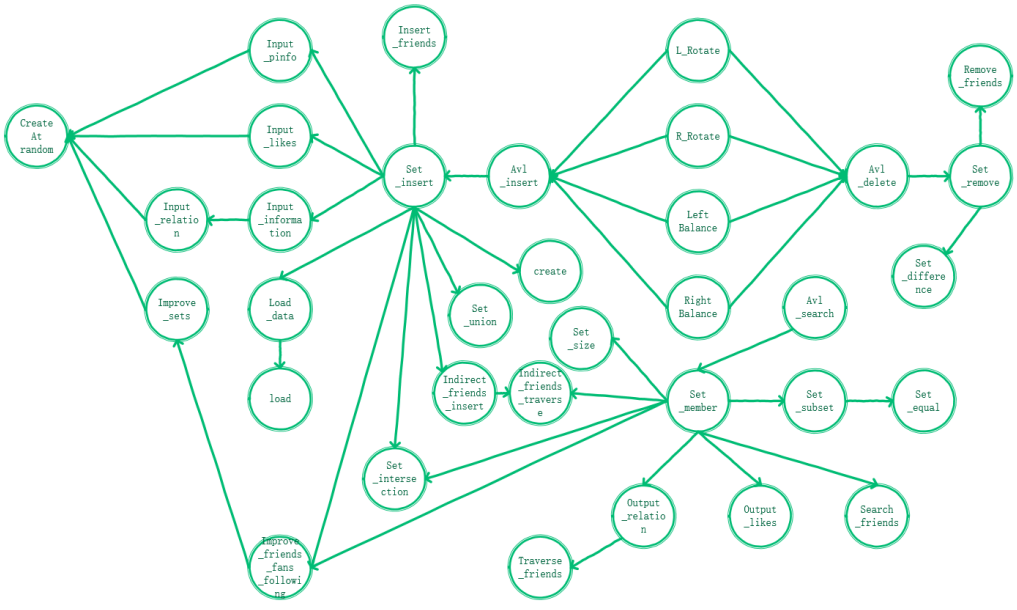


图 4-1 函数调用关系图

4.2 系统测试

4.2.1 测试计划

对系统的主要功能按以下计划测试：

检测功能	操作顺序	操作指令	测试输入	预计输出
set init	1	1	无	初始化成功！
set insert	2	3	11,26,16,36,46,21,32,29,56,7,10	插入成功！
showtree	3	13	无	输出正确的平衡二叉树
set remove	4	4	21,29,32,7,36,56,46	删除成功！
showtree	5	13	无	输出正确的平衡二叉树
set member	6	8	26	是成员！
set member	7	8	21	不是成员！
set intersection	8	5	两棵树：1,2,4,7,9,11和1,2,4,6,8,9	1,2,4,9
set union	9	6	两棵树：1,2,4,7,9,11和1,2,4,6,8,9	1,2,4,6,7,8,9,11
set differnce	10	7	两棵树：1,2,4,7,9,11和1,2,4,6,8,9	7,11
set subset	11	9	两棵树：1,2,4,7,9,11和1,2,4,6,8,9	不是子集！
set subset	12	9	两棵树：1,2,4和1,2,4,6,8,9	是子集！
set equal	13	10	两棵树：1,2,4,7,9,11和1,2,4,6,8,9	不相等！
set equal	14	10	两棵树：1,2,4,7,9,11和1,2,4,7,9,11	相等！
set size	15	11	1,2,4,7,9,11	6
create at random	16	16	无	创建成功！
mutual following	17	18	2,6	输出两人的共同关注
mutual likes	18	19	3,59	输出两人的共同爱好
options for friends	19	23	0,1,2,3,4	分别输出对应的操作结果
indirect friends	20	20	1	输出1的二度好友
save person	21	14	tree6	保存成功！
save more	22	28	1,2,3,4,0	分别输出对应的操作结果
load person	23	15	tree6	加载成功！
load more	24	27	1,2,3,4,0	分别输出对应的操作结果

图 4-2 测试计划

在测试主要的插入删除功能时，要考虑到插入，删除能遇到的所有情况，这样才能体现出完善性。这时就要求测试数据要包含所有可能出现的情况。

4.2.2 测试结果

1.执行 1-set init：初始化成功！

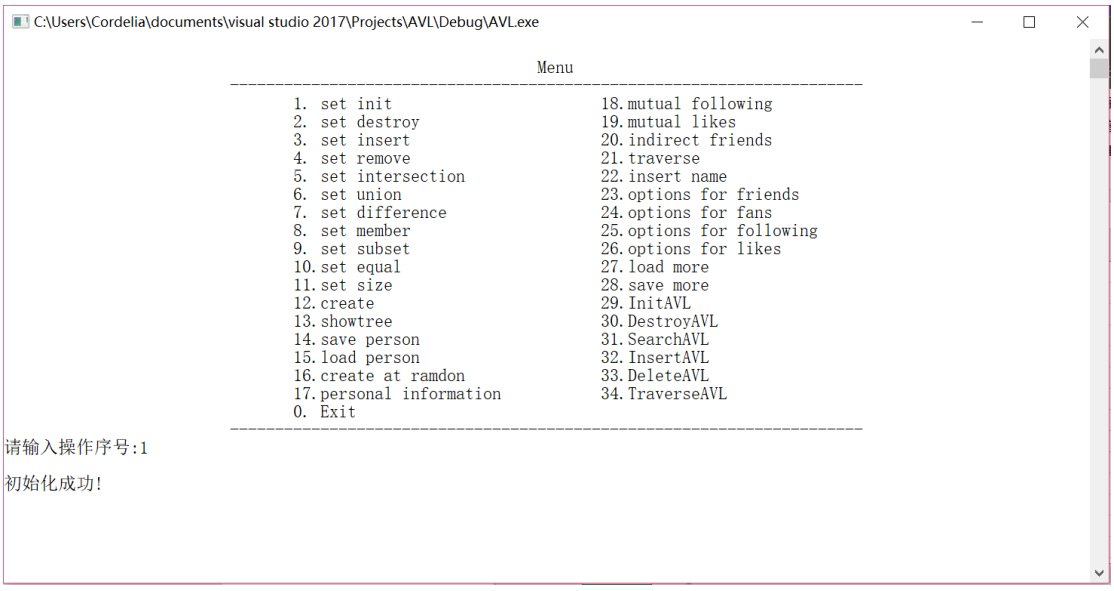


图 4-3 初始化集合

2.执行 3-set insert：插入成功！（输入 11,26,16,36,46,21,32,29,56,7,10）

执行 13-showtree：下面选取三种插入后的二叉树的图，预测结果：（中序）

11,16,26,36

11,16,21,26,32,36,46

7,10,11,16,21,26,29,32,36,46,56

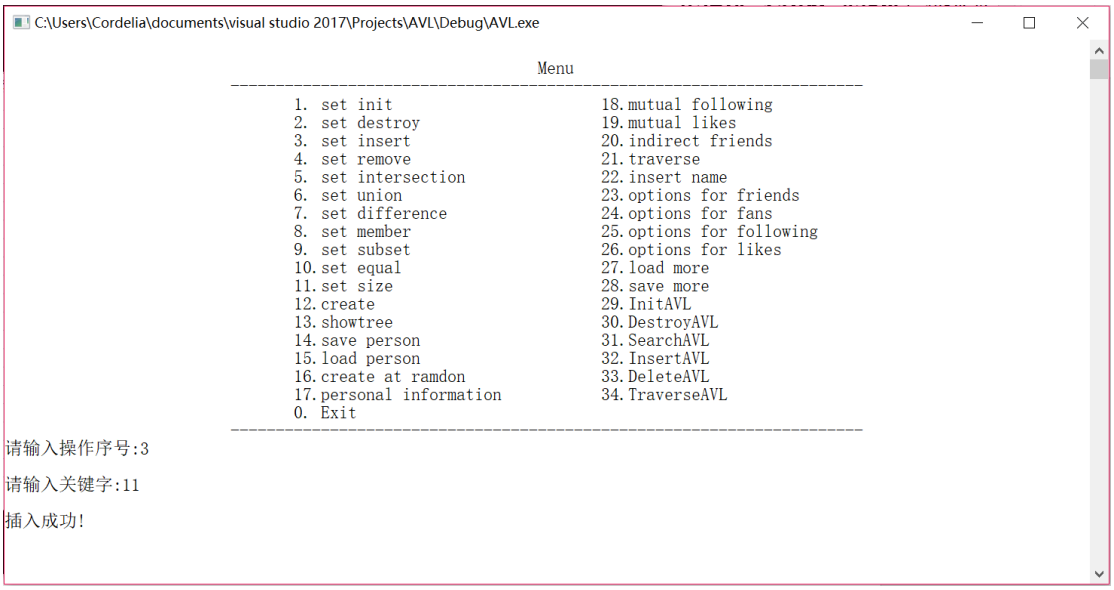


图 4-4 执行 set insert

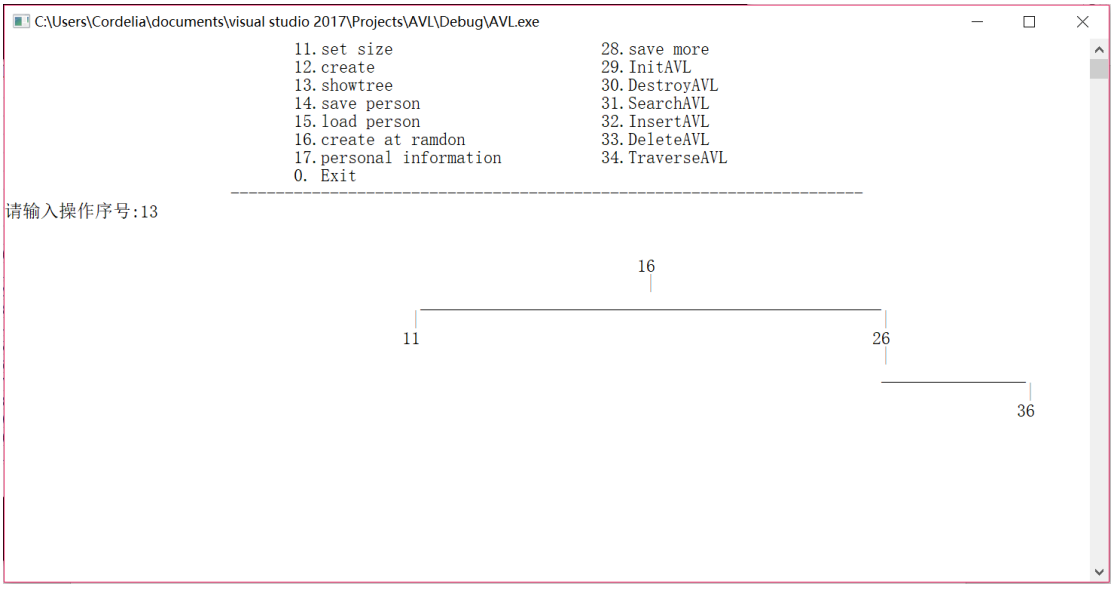


图 4-5 执行 showtree

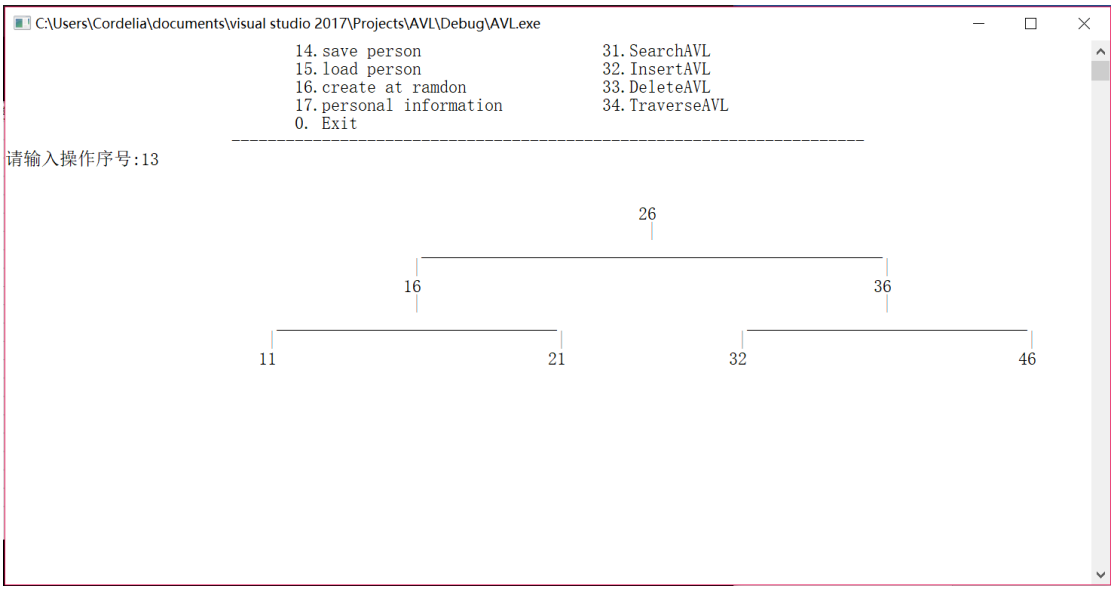


图 4-6 执行 showtree

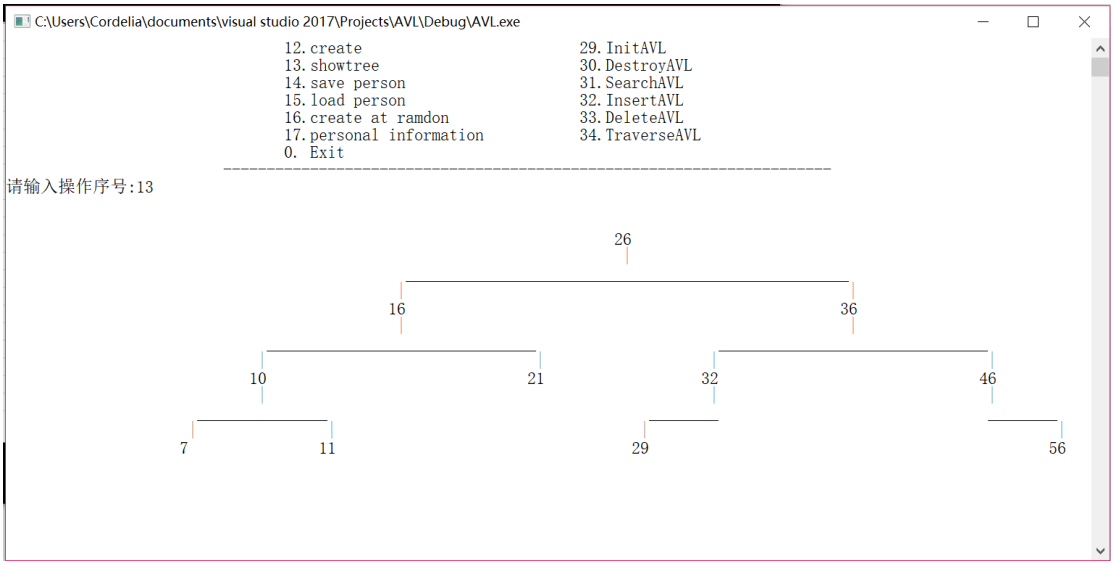


图 4-7 执行 showtree

3.执行 4-set remove：删除成功！(输入 one ABC##DE###F#G##)

执行 13-showtree：下面选取后的二叉树的图，预测结果：（中序）

7,10,11,16,26,29,32,36,46,56

7,10,11,16,26,36,46,56

10,11,16,26,36,46,56

10,11,16,26

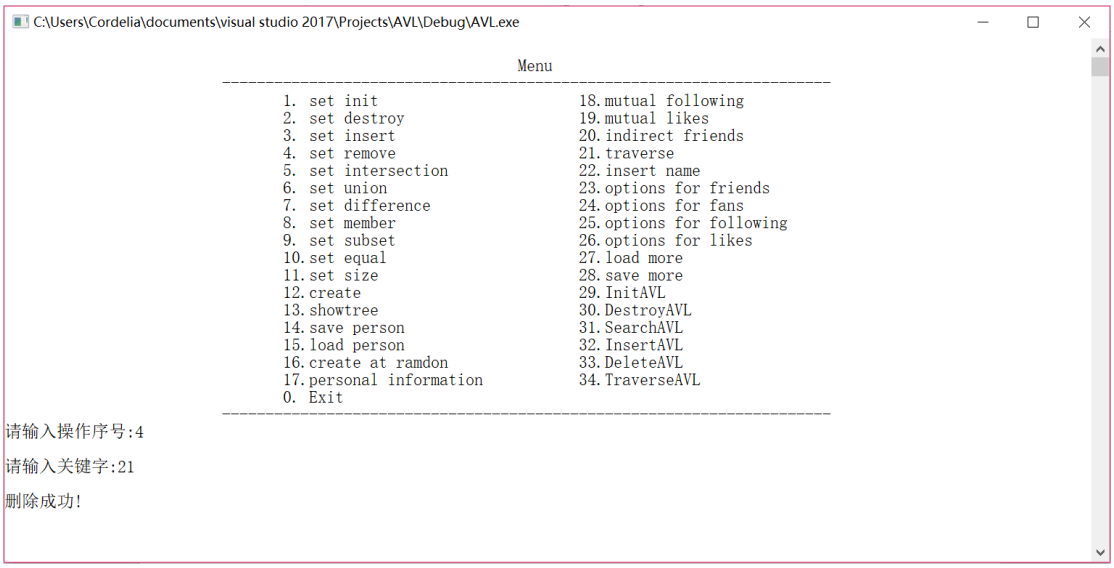


图 4-8 执行 set remove

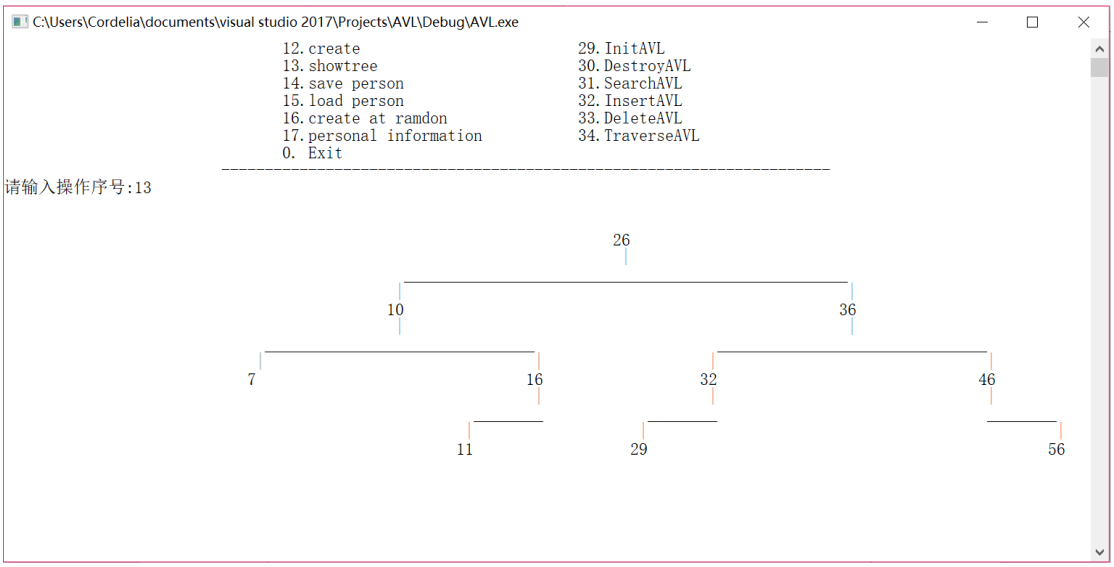


图 4-9 执行 showtree

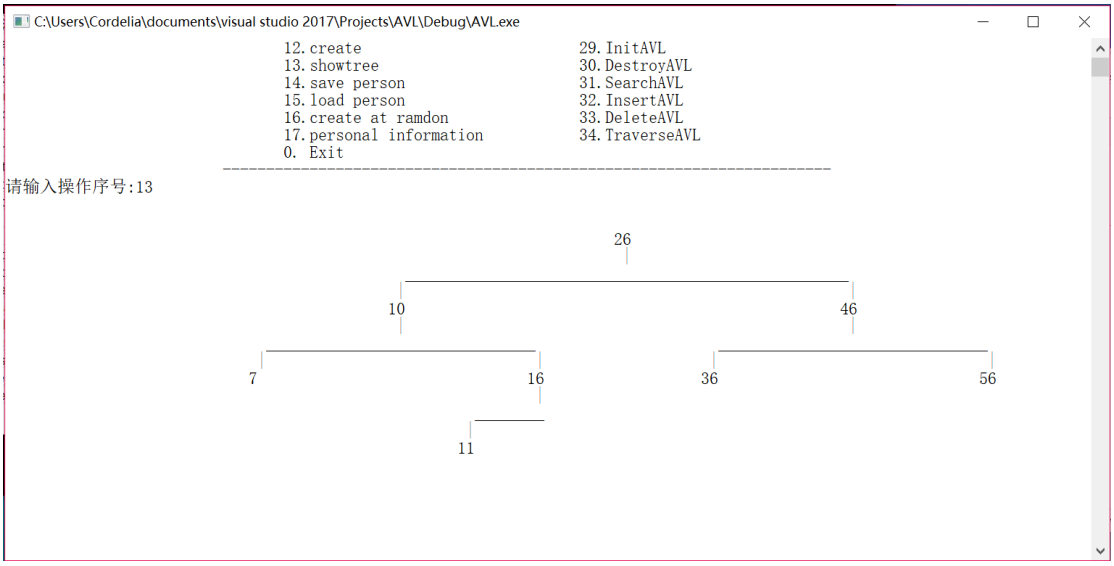


图 4-10 执行 showtree

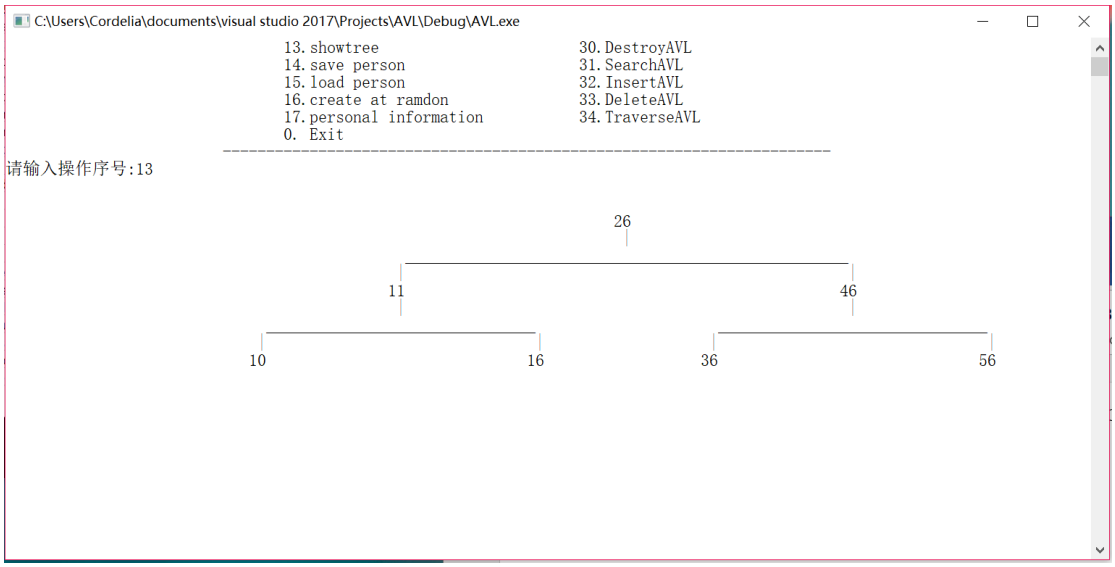


图 4-11 执行 showtree

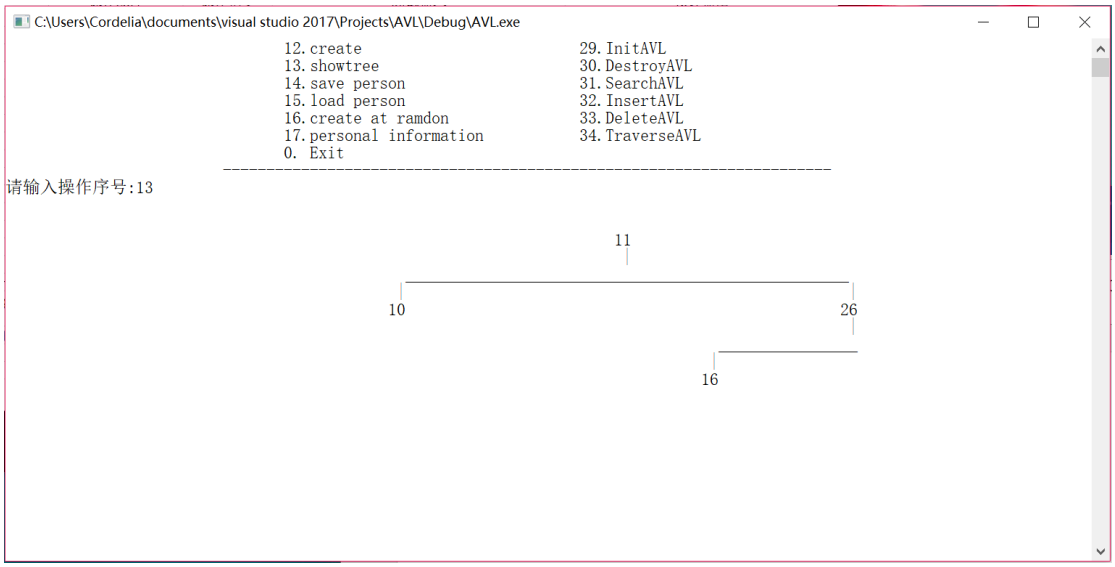


图 4-12 执行 showtree

4.执行 8-set member：是成员！（输入 26）

执行 8-set member：不是成员！（输入 21）

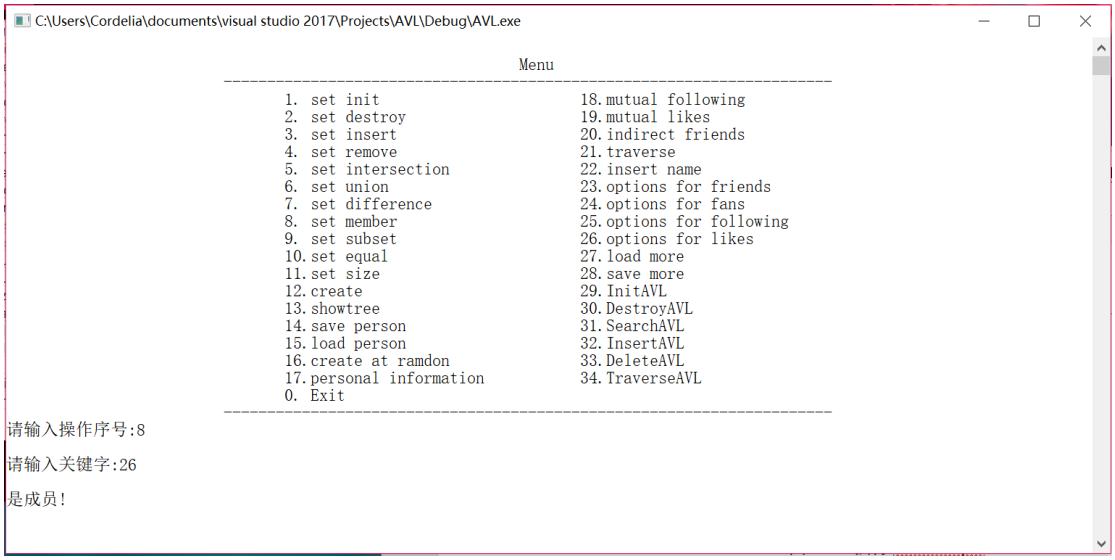


图 4-13 执行 set member

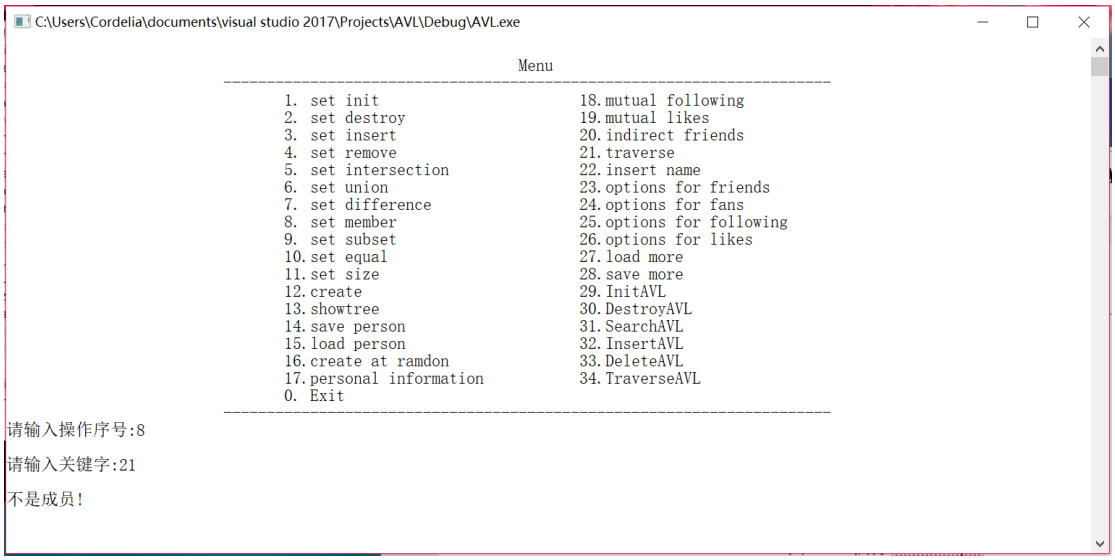


图 4-14 执行 set member

5.执行 5-set intersection：1,2,4,9（输入树为 1,2,4,7,9,11 和 1,2,4,6,8,9）

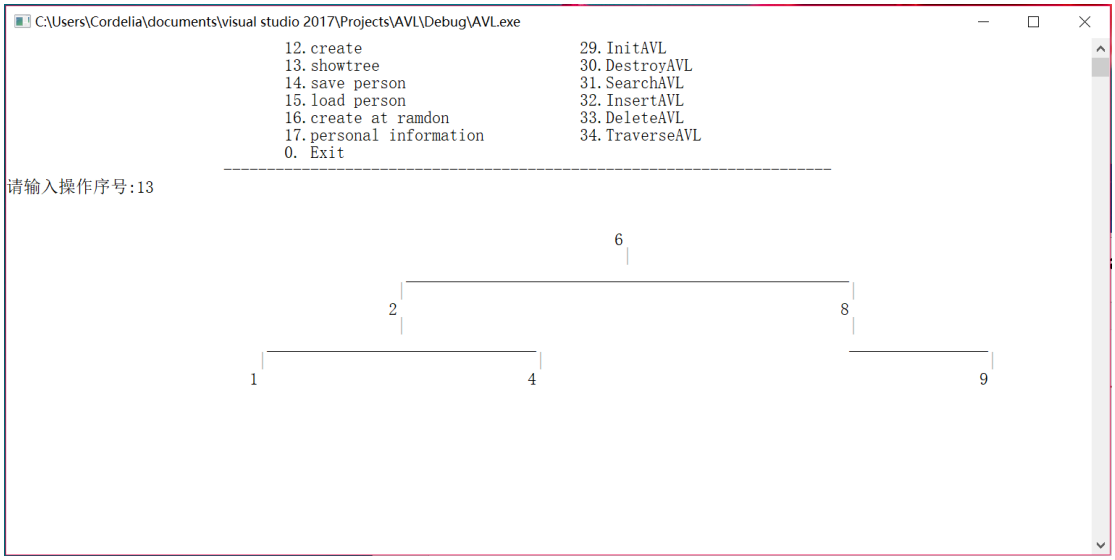


图 4-15 其中一棵树

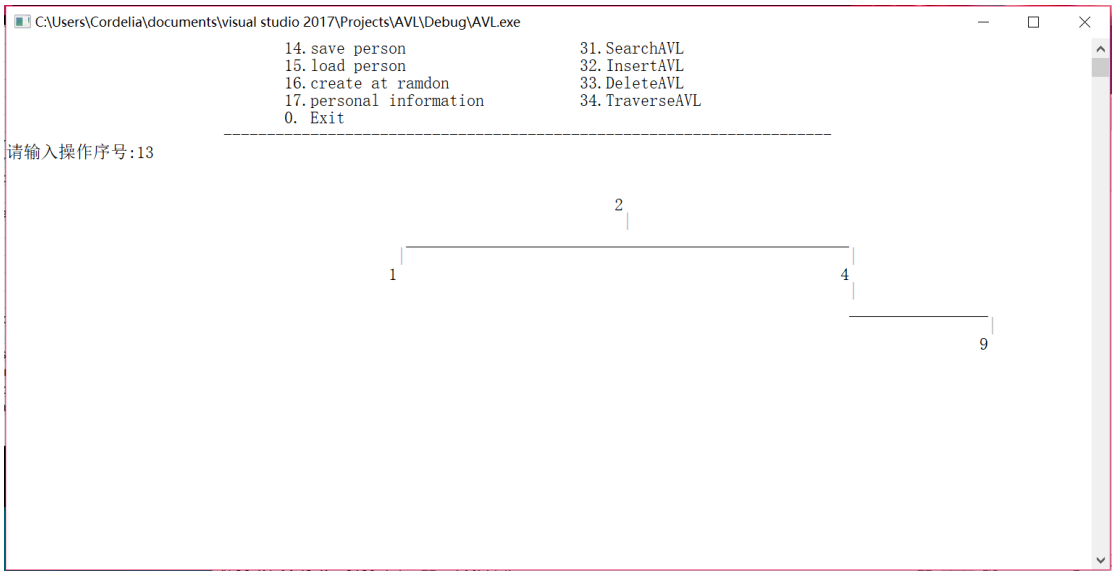


图 4-16 执行 set intersection

6.执行 6-set union：1,2,4,6,7,8,9,11（输入树为 1,2,4,7,9,11 和 1,2,4,6,8,9）

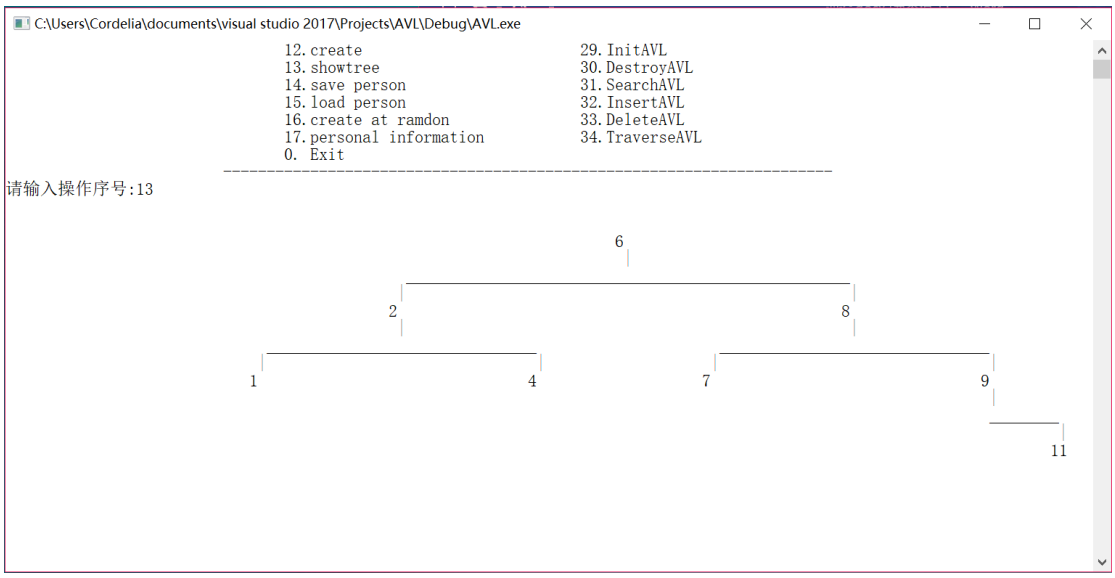


图 4-17 执行 set union

7.执行 7-set difference：7,11（求 1,2,4,7,9,11 减去 1,2,4,6,8,9 的集合）

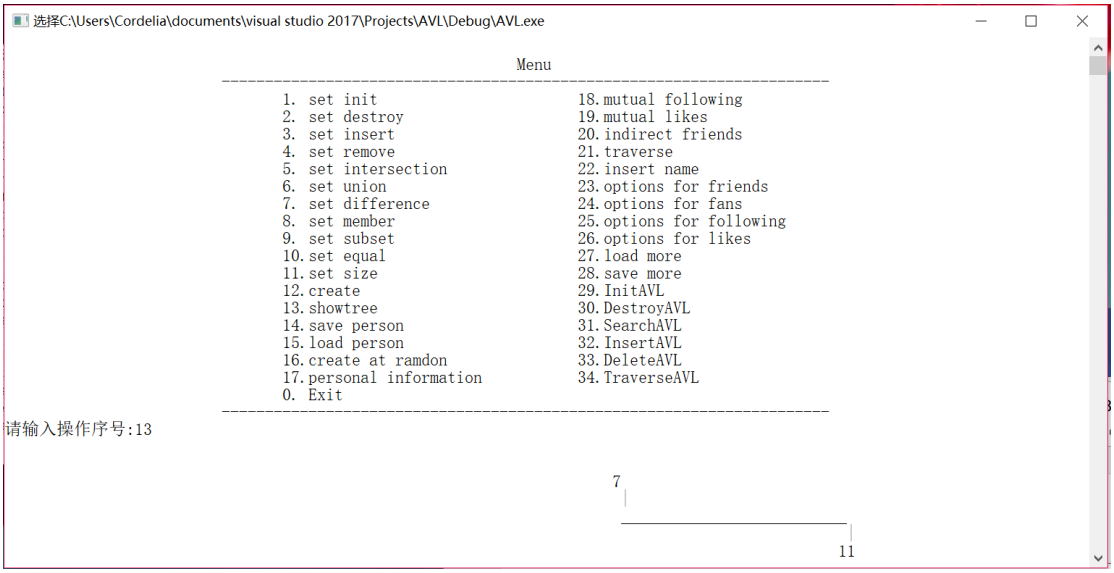


图 4-18 执行 set difference

8.执行 9-set subset：不是子集！（判断 1,2,4,6,8,9 是否是 1,2,4,7,9,11 的子集）

执行 9-set subset：是子集！（判断 1,2,4 是否是 1,2,4,6,8,9 的子集）

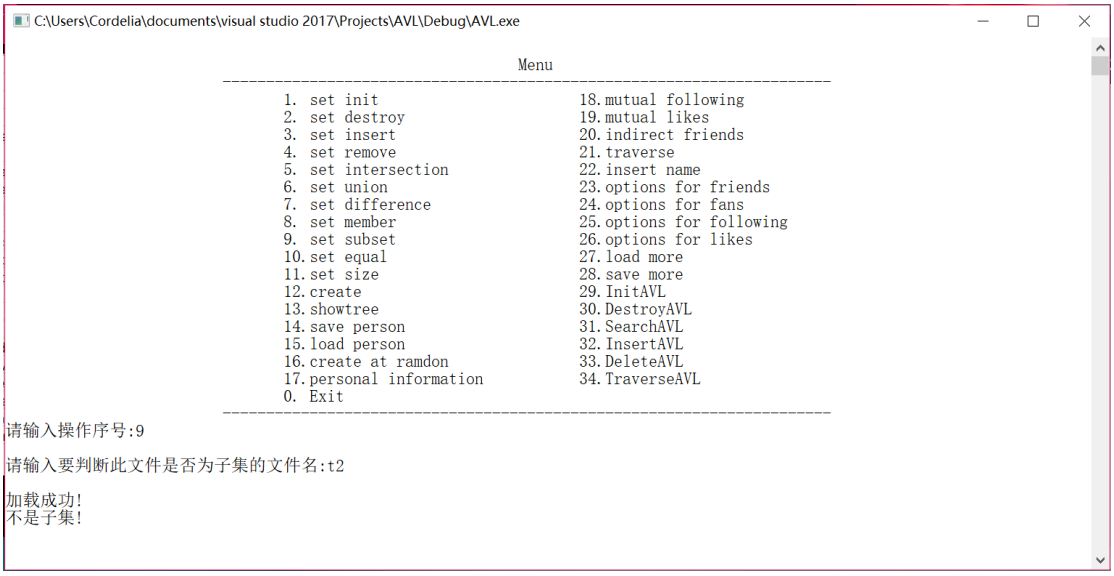


图 4-19 执行 set subset

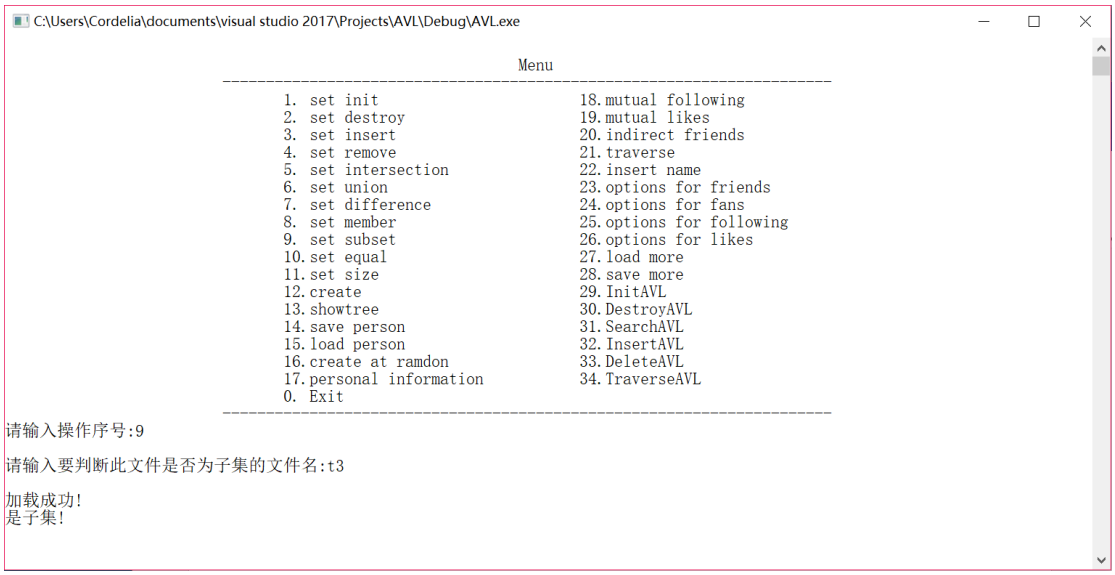


图 4-20 执行 set subset

9.执行 10-set equal：不相等！（判断 1,2,4,7,9,11 和 1,2,4,6,8,9 是否相等）

执行 10-set equal：相等！（判断 1,2,4,7,9,11 和 1,2,4,7,9,11 是否相等）

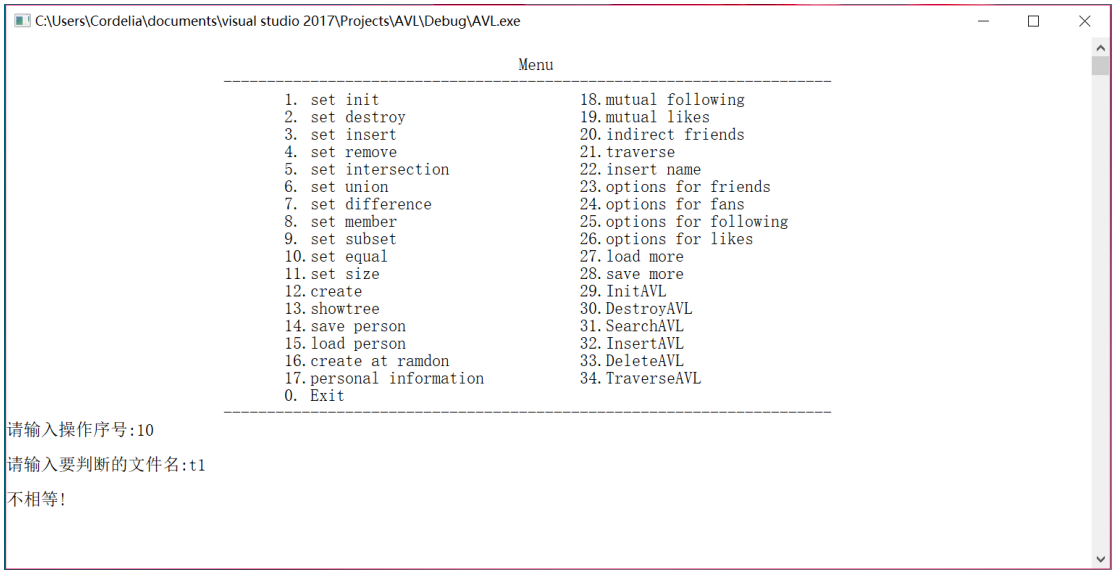


图 4-21 执行 set equal

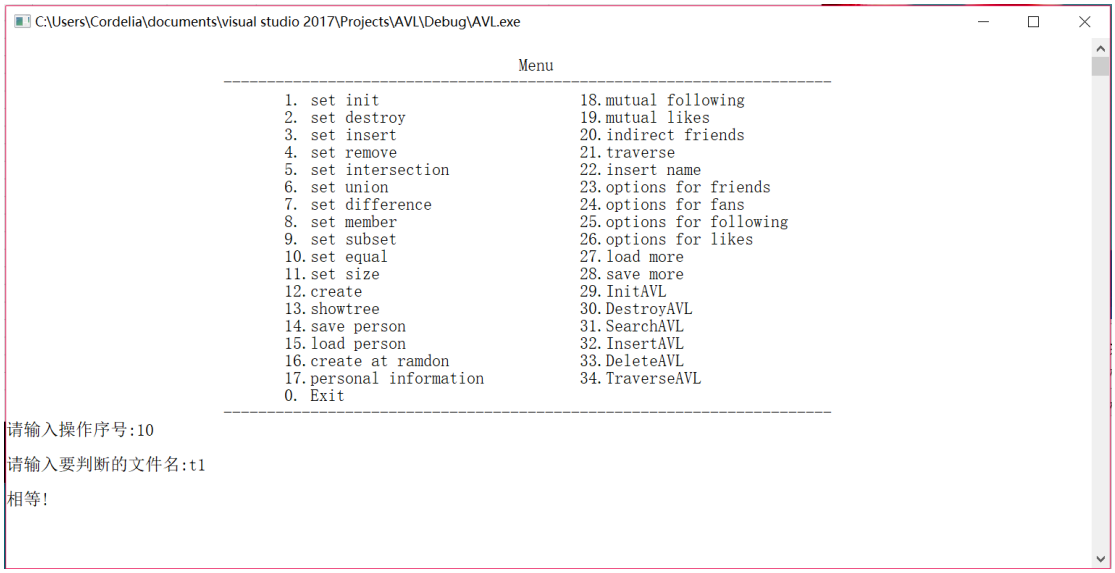


图 4-22 执行 set equal

10.执行 11-set size：6（输入树为 1,2,4,7,9,11）

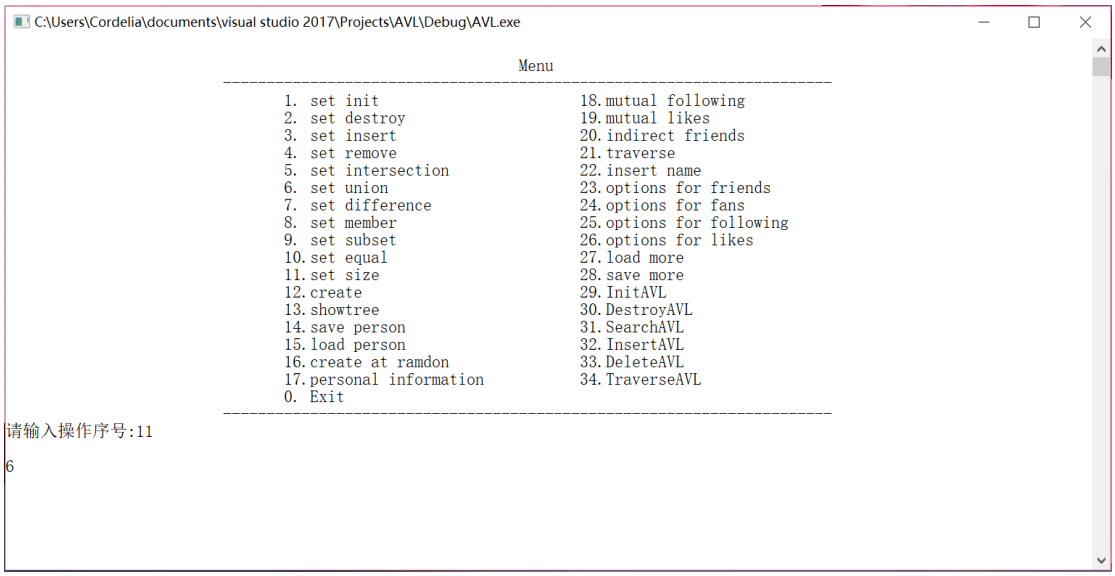


图 4-23 执行 set size

11.执行 18-mutual following：71,166,171,173,183（输入 2,6）

图 4-24 执行 mutual following

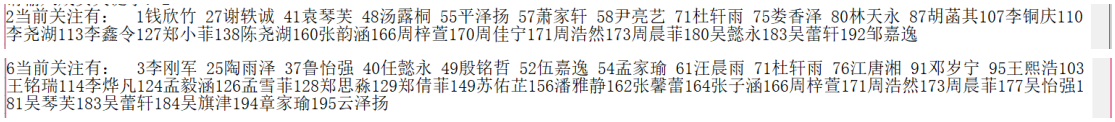


图 4-25 2 和 6 的关注

12.执行 19-mutual likes：0,16（输入 3,59）

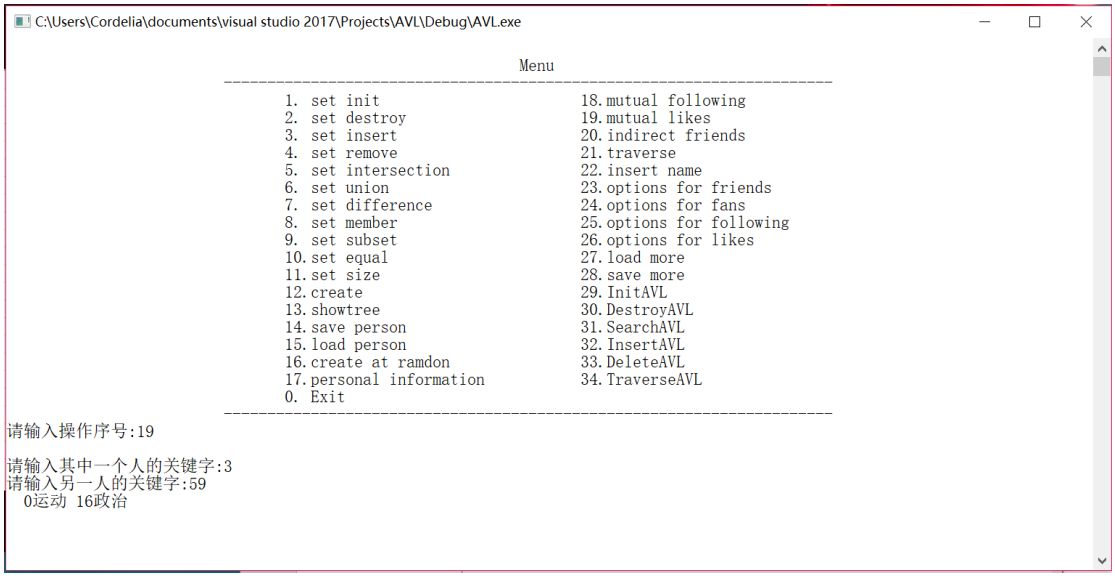


图 4-26 执行 mutual likes

图 4-27 3 和 59 的爱好

13.执行 23-options for friends：（options for fans，following，likes 都差不多，在此不测试）

选择 0：初始化成功！（并且使其他人的好友集不含自己）

选择 1:1 和 2 互相成为好友！（输入 2）

选择 2:1 与 2 互删成功！（输入 2）

选择 3:2 是 1 的好友！（输入 2）

选择 4:1 当前的好友有：2

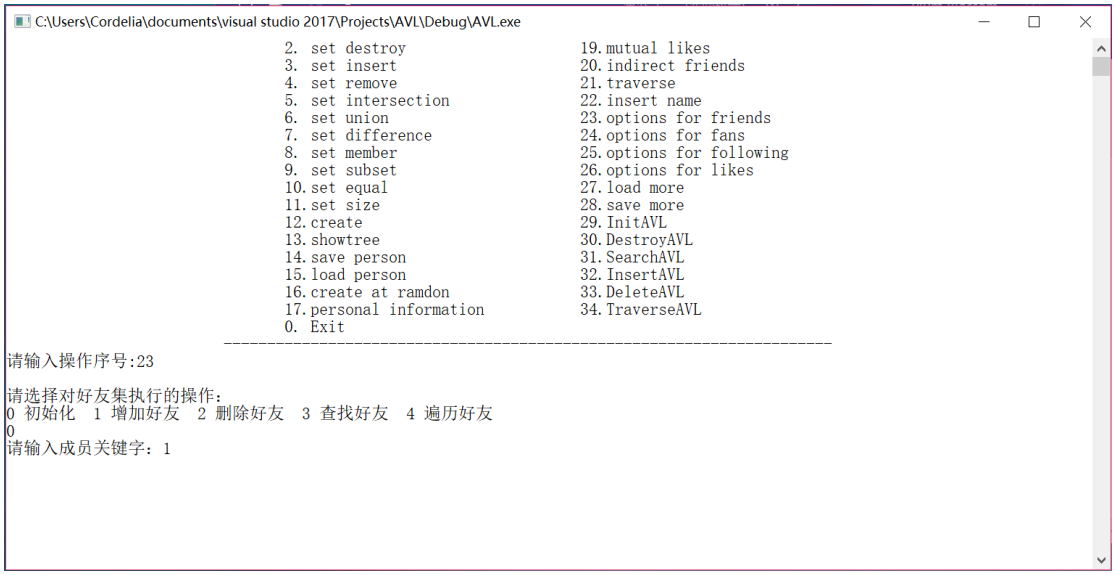


图 4-28 好友集初始化

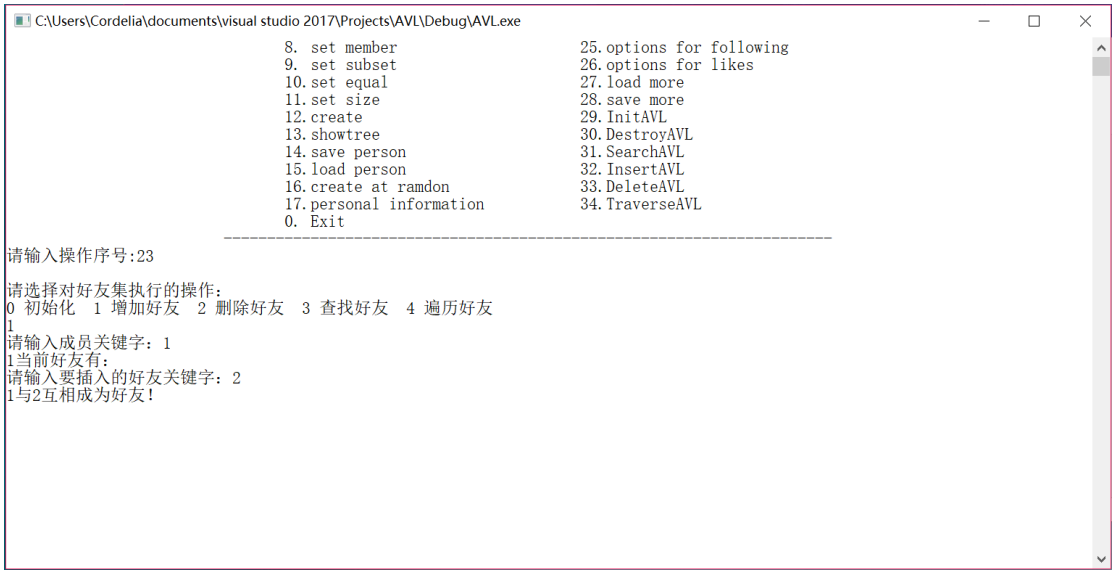


图 4-29 增加好友

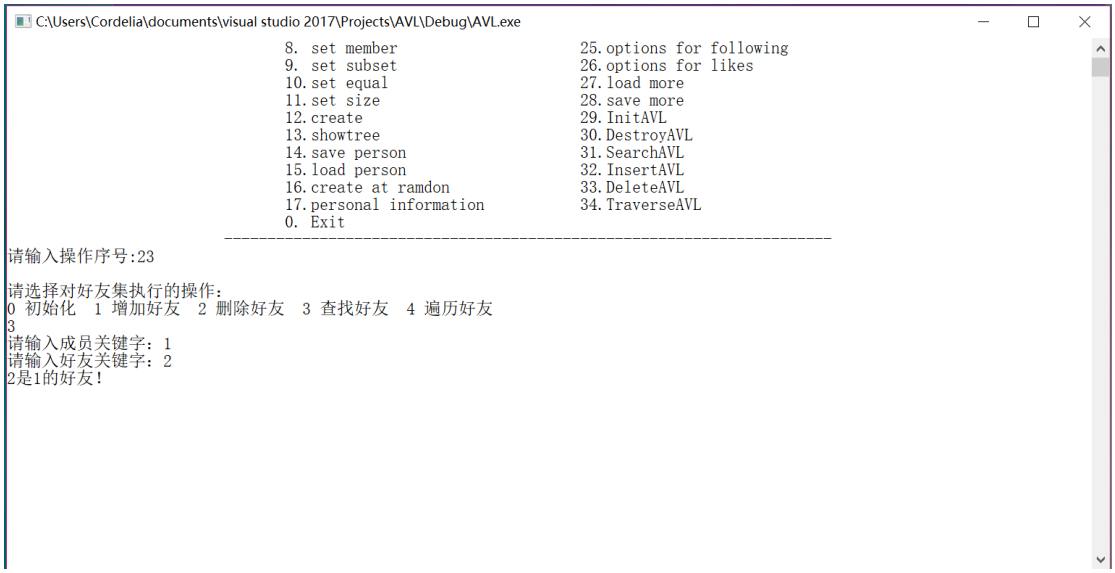


图 4-30 查找好友

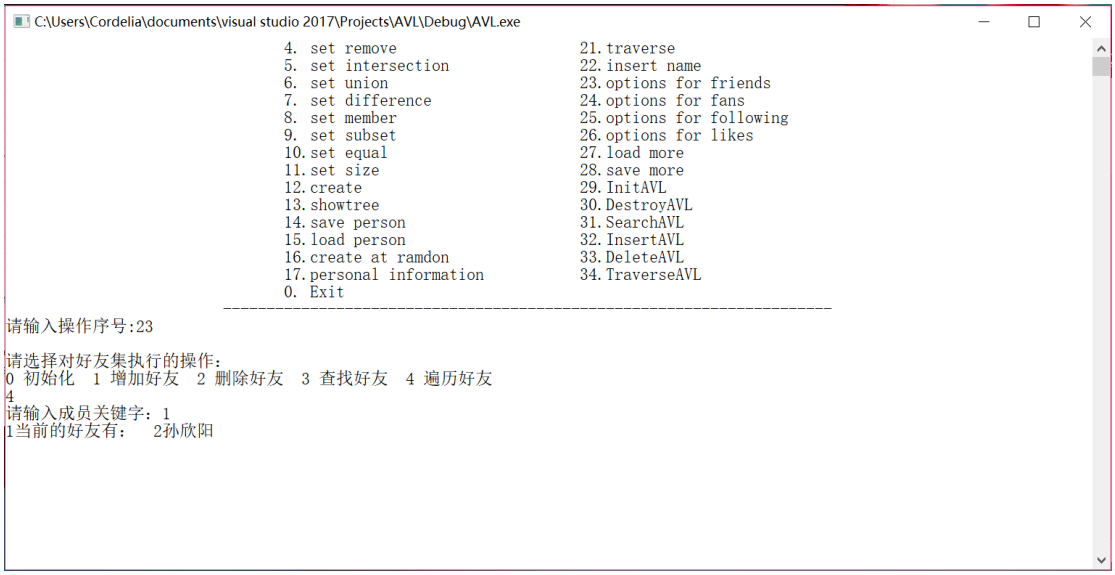


图 4-31 遍历好友

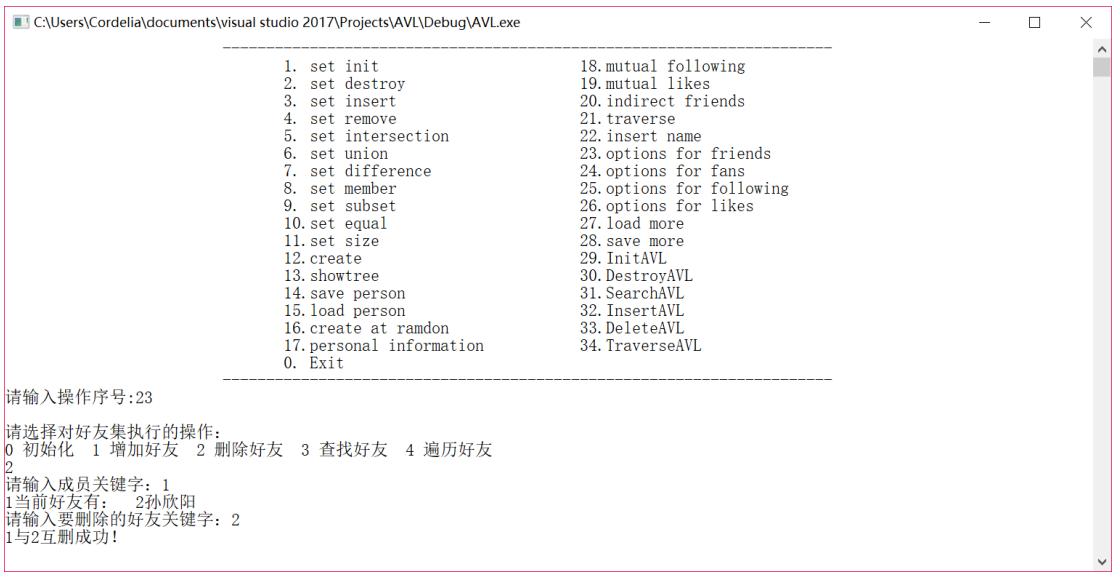


图 4-32 删除好友

14.执行 14-save person：保存成功！（输入 tree6）

执行 28-save more：保存成功！

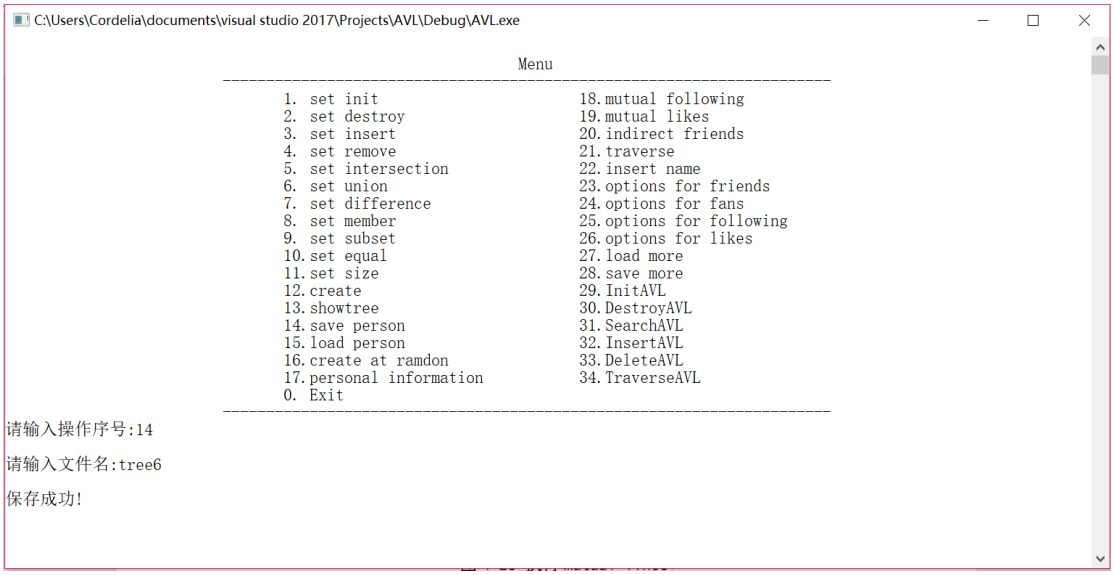


图 4-33 执行 save person

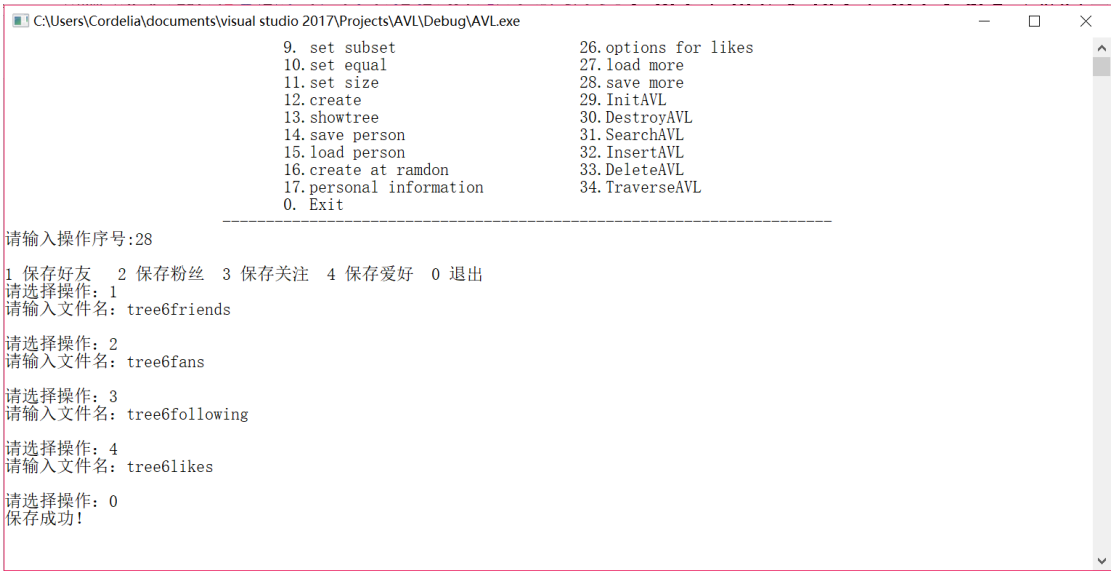


图 4-34 执行 save more

15.执行 15-load person：加载成功！（输入 tree6）

执行 27-load more：加载成功！

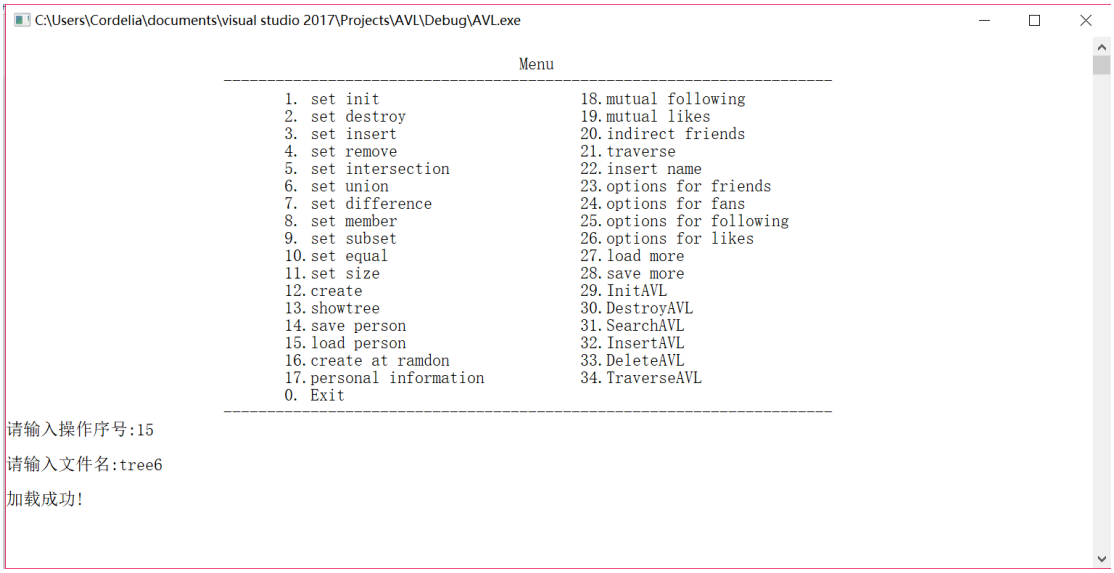


图 4-35 执行 load person

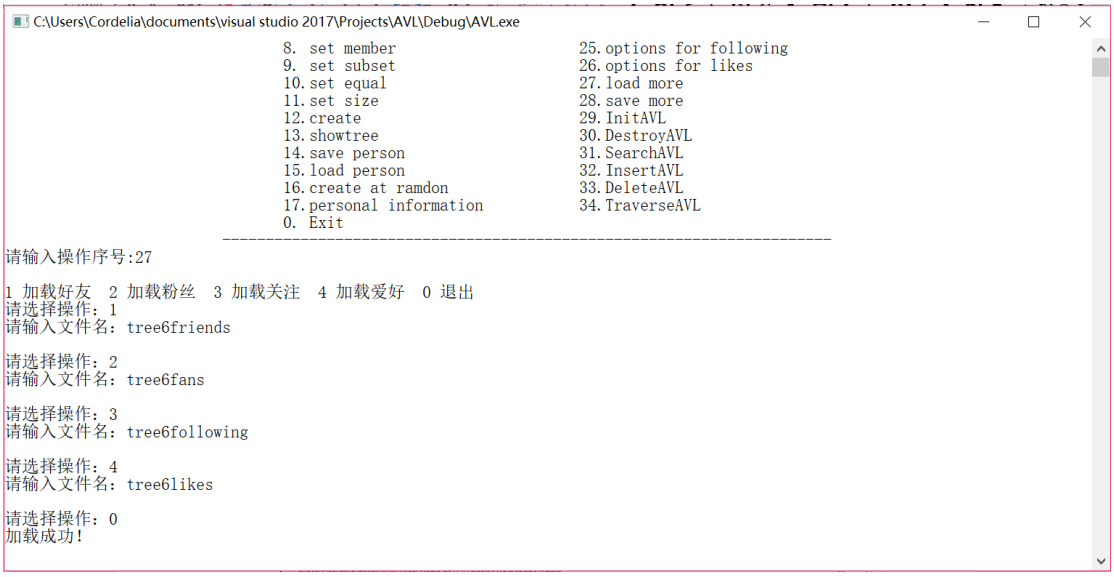


图 4-36 执行 load more

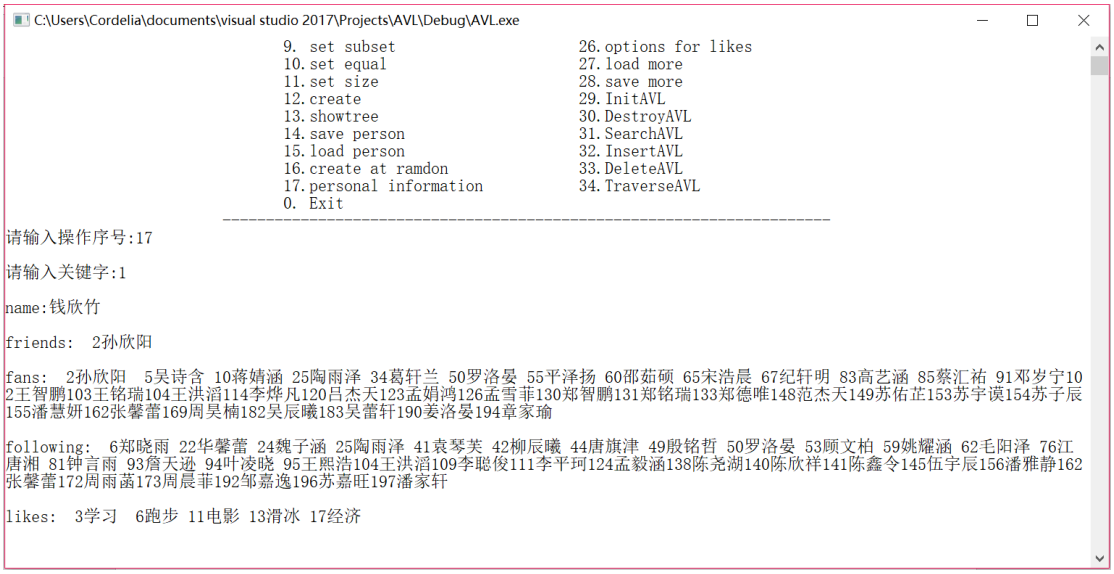


图 4-37 查看人物信息

16.执行 20-indirect friends：

4,5（输入 1）

```
1当前的好友有:  2钱二二  3孙三三

2当前的好友有:  1赵一一  4李四四

3当前的好友有:  1赵一一  5吴五五
```

图 4-38 1, 2, 3 当前好友

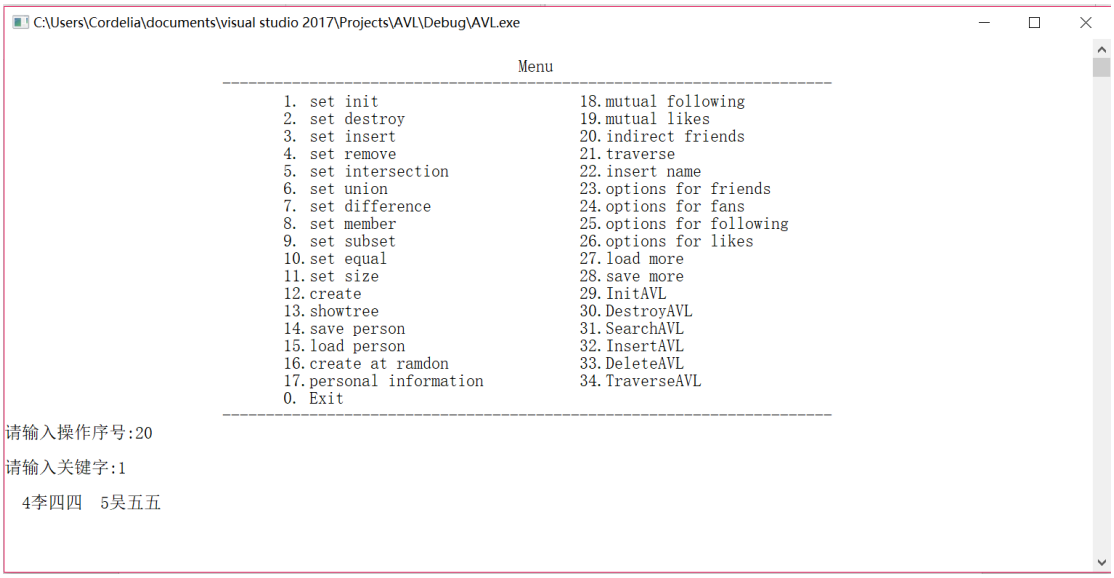


图 4-39 执行 indirect friends

5 总结与展望

5.1 全文总结

所做工作如下：

1.积累了个人开发小项目的实践经验

2.通过查找资料来实现一些比较困难的算法

3.初步熟悉了较为复杂的平衡二叉树的基本算法

4.进一步了解了递归的思想

5.通过平衡二叉树的基本操作实现了模拟社交网络的功能，上升到应用层

6.提升了自己写程序的熟练度

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.美化界面，有一定设计

2.使得操作更加易懂，面向使用者

3.完善一些应用的功能，弥补其残缺

4.思考其他实现平衡二叉树插入删除的方法

5.不止局限于平衡二叉树，多学习其他更加实用的数据结构，如红黑树等

6 体　会

本次课程设计是数据结构课程结束后的一次大作业。

我选择的题目是基于 AVL 实现 ADT 应用层功能，其中主要是 AVL 树的插入，删除，查找等基本操作。

首先说一下本次数据结构课程设计的界面，由于对界面实现的不熟悉，本次我没有选择复杂的界面，当然这也是一个遗憾，一个好的应用程序除了功能强大之外，还要有优秀的 UI 设计，在这次课程设计过后，我准备对界面实现方面的知识做一些了解。

接着就是本次课程设计的基础，AVL 树的基本操作如初始化，插入，删除，搜索，遍历，销毁等。这些是后面功能实现的基础。其中初始爱，搜索，遍历，销毁操作比较简单。在实现插入，删除操作上是一个新的挑战，这两个操作的完成是我基于已有文献的基础上实现的。在经过考虑后，我选择了引入平衡因子这个概念，插入删除需要调用的函数有 L_Rotate,R_Rotate,LeftBalance 和 RightBalance 这几个函数，其作用分别是左旋，右旋，左平衡，右平衡。这些概念我就不在这里一一叙述了，相比较删除算法来说，插入算法比较简单。插入算法如果在左子树插入有三种情况，根据平衡因子的不同采取不同的措施，在右子树插入也有三种情况，同左子树采取类似的措施即可完成插入操作。删除算法可以说是这里面最复杂的算法了，在查找了很多资料的情况下，我才能自己慢慢地理解算法，并实现删除功能。在要找节点的左子树或者右子树都为空时，都比较好操作，当左右子树都不为空时，声明一个指针指向当前节点左子树的最右子树，再向左子树递归删除节点，删除完之后，再对现有树做一次平衡调整。当输入的关键字小于当前节点的关键字时，在左子树中继续查找节点，删除节点后对树做一次平衡处理。当输入的关键字大于当前节点关键字时，在右子树中进行类似操作。

在完成了 AVL 树的基本操作后，就可以调用这些函数来完成一些集合的基本操作，比如集合的交集，用查找以及插入就可以实现求交集的操作，集合的并集，差集等等都是如此，在这一层面上还是比较简单的。

接着就上升到应用层了，一些应用比如共同关注共同爱好都是使用集合的交集来实现的，这些都比较简单，二度好友则稍微复杂一点，它要求先将一个人的

好友的所有好友插入到一棵树上，再求这棵树和这个人的好友树的差集，最后删除自己，即可完成二度好友的功能。最后还比较复杂的就是对一个人好友集，粉丝集，关注集等的各种操作。初始化，插入，删除，遍历这些都要考虑一些边缘情况，要求你认真考虑实际应用中的各种情况，本质上也是通过 AVL 树的基本操作来实现应用层功能。

最后就是文件保存加载功能了，关于这个功能我写的比较复杂了，我先保存了所有人的一棵树，然后分别保存了这些人的人际关系树，通过保存这些关键字即可达到保存信息的功能。同理，加载也是这样。

在完成这个项目的过程中，我查阅了许多资料，这让我感觉到即使通过一个学期的学习，我还是对数据结构这门课程缺乏深入的了解，仅仅通过这么短时间的学习是不够的。书本上的理论知识是比较浅的，但即使是这些知识，要熟练掌握也需要大量的练习，这时平时的实践就显得尤为重要了，只有在实践中，我们才能加深对这些数据结构的理解。而且数据结构也并非是有限的，要在现有基础上创造出什么样的更好的数据结构，是我们要思考的问题。在掌握这些基本的数据结构后，我们是否能自己创造出更实用，效率更高的数据结构。

理论要和实践结合，才能更加有说服力，计算机科学是一门理论加实践的学科，在今后的学习中，不仅仅要把专业知识掌握牢固，还要增强自己的动手能力，多付出时间，才能收获自己想要的结果。

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

附录

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<conio.h>
#include<windows.h>
#include<time.h>
#include<stddef.h>
#include<stdbool.h>

#define LH 1      //左边高
#define EH 0      //等高
#define RH -1     //右边高

typedef struct Info{
    int key;//关键字
    char lastname[3];
    char firstname[5];
    char like[5];
    struct BATNode *friends;
    struct BATNode *fans;
    struct BATNode *following;
    struct BATNode *likes;
}Info;
typedef struct BATNode {
    Info data;
    int bflag;//二叉树状态
    int h;//高度
    struct BATNode *lchild, *rchild;
}BATNode, *BATree;

typedef void(*visit)(BATree T);
void L_Rotate(BATree *p);//左旋函数
void R_Rotate(BATree *p);//右旋函数
void LeftBalance(BATree *T);//对节点实现左平衡
void RightBalance(BATree *T);//对节点实现右平衡
void avl_init(BATree *T);//AVL 树初始化
void avl_destroy(BATree *T);//AVL 树销毁
bool avl_insert(BATree *T, Info e, bool *taller);//AVL 树插入
bool avl_delete(BATree *T, int key, bool *shorter);//AVL 树删除
void avl_traverse(BATree T);//AVL 树遍历
bool avl_search(BATree T, int key, BATree *Tsub);//AVL 树搜索
void set_init(BATree *T);//集合初始化
```

```
void set_destory(BATree *T);//集合销毁
bool set_insert(BATree *T, Info e, bool *taller);//集合插入
bool set_remove(BATree *T, int key, bool *shorter);//集合删除
void set_intersection(BATree T, BATree T1, BATree *T0);//求集合交集
void set_union(BATree *T, BATree T1);//求集合并集
void set_difference(BATree *T, BATree T1);//求集合差集
void set_size(BATree T1, BATree T);//求集合元素个数
bool set_member(BATree T, int key, BATree *Tsub);//判断是否为集合中成员
bool set_subset(BATree Tsub, BATree T1);//判断一个集合是否是另一个集合的子集
bool set_equal(BATree T, BATree T1);//判断两个集合是否相等
bool load_data(BATree *T, char *filename);//加载主要的树
bool save_data(BATree T, FILE *fp);//保存主要的树
void input_key(Info *data);//输入关键字
void output_key(BATree T);//输出关键字
void create(BATree *T);//创建树
void height(BATree T, int i);//求树高度
void gotoxy(int x, int y);//锁定位置
void showtree(BATree T, int x, int y, visit fp);//显示树图
void InOrderTraverse(BATree T);//中序遍历
void output_relation(BATree T1, BATree T);//输出关系
void output_likes(BATree T1, BATree L);//输出爱好
bool input_pinfo(BATree *T, int p_size);//随机输入人员信息
bool input_likes(BATree *L, int *l_size);//随机生成爱好树
void input_relation(BATree *T, int p_size, int l_size);//输入关系信息
void input_information(BATree *T, int size, int max, int min);//输入个人信息
void improve_sets(BATree T, BATree T0);//完善随机生成的各种集合
void improve_friends(BATree T, BATree T1, BATree T2);//完善好友集
void improve_fans(BATree T, BATree T1, BATree T2);//完善粉丝集
void improve_following(BATree T, BATree T1, BATree T2);//完善关注集
void indirect_friends_traverse(BATree T, BATree T1, BATree *T0);//二度好友遍历
void indirect_friends_insert(BATree T, BATree *T0);//二度好友插入
void set_init_fors(BATree *T);//对一个人的好友集，粉丝集，关注集，爱好集等
bool insert_name(BATree T);//为一个节点赋名
bool init_friends(BATree T);//初始化好友集
bool insert_friends(BATree T);//插入好友
bool remove_friends(BATree T);//删除好友
bool search_friends(BATree T);//搜索好友
bool traverse_friends(BATree T);//遍历好友
bool init_fans(BATree T);//初始化粉丝集
```

```
bool insert_fans(BATree T);//插入粉丝
bool remove_fans(BATree T);//删除粉丝
bool search_fans(BATree T);//搜索粉丝
bool traverse_fans(BATree T);//遍历粉丝
bool init_following(BATree T);//初始化关注集
bool insert_following(BATree T);//插入关注
bool remove_following(BATree T);//删除关注
bool search_following(BATree T);//搜索关注
bool traverse_following(BATree T);//遍历关注
bool init_likes(BATree T,BATree L);//初始化爱好集
bool insert_likes(BATree T, BATree L);//插入爱好
bool remove_likes(BATree T, BATree L);//删除爱好
bool search_likes(BATree T, BATree L);//搜索爱好
bool traverse_likes(BATree T, BATree L);//遍历爱好
bool load(BATree *T);//加载更多信息
bool traverse_relation(BATree T, FILE *fp);//遍历人与人之间的关系
bool save(BATree T);//保存更多信息
bool save_relation(BATree T, FILE *fp, int i);//保存人与人之间的关系

bool taller, shorter;
int x_move = 70, y_move= 25;
int count = 0;
BATree Tsub = NULL;

int main(){
    int i;
    int p_size = 200, l_size=10;
    BATree T = NULL;
    BATree L = NULL;
    srand((unsigned)time(NULL));
    system("color f0");
    do {
        printf("\n");
        printf("
Menu\n");
        printf("
-------------------------------------------------------------------
-\n");
        printf("                                                1. set init
18.mutual following\n");
        printf("                                              2. set destroy
19.mutual likes\n");
        printf("                                               3. set insert
20.indirect friends\n");
```

```
        printf("                                          4. set remove
21.traverse\n");
        printf("                                    5. set intersection
22.insert name\n");
        printf("                                          6. set union
23.options for friends\n");
        printf("                                   7. set difference
24.options for fans\n");
        printf("                                          8. set member
25.options for following\n");
        printf("                                          9. set subset
26.options for likes\n");
        printf("                                         10.set equal
27.load more\n");
        printf("                                          11.set size
28.save more\n");
        printf("                                            12.create
29.InitAVL\n");
        printf("                                          13.showtree
30.DestroyAVL\n");
        printf("                                        14.save person
31.SearchAVL\n");
        printf("                                        15.load person
32.InsertAVL\n");
        printf("                                    16.create at ramdon
33.DeleteAVL\n");
        printf("                                          17.personal
information          34.TraverseAVL\n");
        printf("                                0. Exit\n");
        printf("
-------------------------------------------------------------------
-\n");
        printf("请输入操作序号:");
        scanf("%d", &i);
        getchar();
        printf("\n");
        switch (i) {
            case 0:
                break;
            case 1: {
                set_init(&T);
                break;
            }
            case 2: {
```

```
        set_destory(&T);
        printf("销毁成功!\n");
        break;
    }
    case 3: {
        Info e;
        input_key(&e);
        if (set_insert(&T, e, &taller)) {
            printf("插入成功!\n");
            height(T, 0);
        }
        else printf("插入失败!\n");
        break;
    }
    case 4: {
        int key;
        printf("请输入关键字:");
        scanf("%d", &key);
        getchar();
        printf("\n");
        if (set_remove(&T, key, &shorter)) {
            printf("删除成功!\n");
            height(T, 0);
        }
        else printf("删除失败!\n");
        break;
    }
    case 5: {
        char filename[20];
        BATree T1 = NULL;
        BATree T0 = NULL;
        printf("请输入文件名:");
        scanf("%s", filename);
        getchar();
        printf("\n");
        if (load_data(&T1, filename)) {
            printf("加载成功!\n");
            set_intersection(T, T1, &T0);
            set_init(&T);
            T = T0;
            height(T, 0);
            printf("取交集成功!\n");
        }
        else printf("加载失败!\n");
```

```
        break;
    }
    case 6: {
        char filename[20];
        BATree T1 = NULL;
        printf("请输入文件名:");
        scanf("%s", filename);
        getchar();
        printf("\n");
        if (load_data(&T1, filename)) {
            printf("加载成功!\n");
            set_union(&T, T1);
            height(T, 0);
            printf("取并集成功!\n");
        }
        else printf("加载失败!\n");
        break;
    }
    case 7: {
        char filename[20];
        BATree T1 = NULL;
        printf("请输入文件名:");
        scanf("%s", filename);
        getchar();
        printf("\n");
        if (load_data(&T1, filename)) {
            printf("加载成功!\n");
            set_difference(&T, T1);
            height(T, 0);
            printf("取差成功!\n");
        }
        else printf("加载失败!\n");
        break;
    }
    case 8: {
        int key;
        printf("请输入关键字:");
        scanf("%d", &key);
        getchar();
        printf("\n");
        if (set_member(T, key, &Tsub)) printf("是成员!\n");
        else printf("不是成员!\n");
        break;
    }
```

```
case 9: {
    char filename[20];
    BATree T1 = NULL;
    printf("请输入要判断此文件是否为子集的文件名:");
    scanf("%s", filename);
    getchar();
    printf("\n");
    if (load_data(&T1, filename)) {
        printf("加载成功!\n");
        if (set_subset(T, T1)) printf("是子集!\n");
        else printf("不是子集!\n");
    }
    else printf("加载失败!\n");
    break;
}
case 10: {
    char filename[20];
    BATree T1 = NULL;
    printf("请输入要判断的文件名:");
    scanf("%s", filename);
    getchar();
    printf("\n");
    if (load_data(&T1, filename)) {
        if (set_equal(T, T1)) printf("相等!\n");
        else printf("不相等!\n");
    }
    else printf("加载失败!\n");
    break;
}
case 11:
{
    count = 0;
    set_size(T, T, &count);
    printf("%d\n", count);
    break;
}
case 12: {
    create(&T);
    break;
}
case 13: {
    visit fp = &output_key;
    showtree(T, 0, 0, fp);
    break;
```

```
            }
            case 14: {
                FILE *fp;
                char filename[20];
                printf("请输入文件名:");
                scanf("%s", filename);
                getchar();
                printf("\n");
                if ((fp = fopen(filename, "wb")) == NULL) {
                    printf("文件打开失败!\n");
                    break;
                }
                if (save_data(T, fp)) printf("保存成功!\n");
                else printf("保存失败!\n");
                fclose(fp);
                break;
            }
            case 15: {
                char filename[20];
                printf("请输入文件名:");
                scanf("%s", filename);
                getchar();
                printf("\n");
                if (load_data(&T, filename)) printf("加载成功!\n");
                break;
            }
            case 16:{
                BATree T0;
                if (input_pinfo(&T, p_size)) printf("成功输入个人信
息!\n");
                else {
                    printf("输入个人信息失败!\n");
                    break;
                }
                 if (input_likes(&L, &l_size)) printf("成功输入爱好信
息!\n");
                else {
                     printf("输入爱好信息失败!\n");
                     break;
                }
                 input_relation(&T, p_size, l_size);
                 T0 = T;
                 improve_sets(T, T0);
                break;
```

```
            }
            case 17: {
                BATree T1;
                int key;
                printf("请输入关键字:");
                scanf("%d", &key);
                getchar();
                if (!set_member(T, key, &T1)) {
                    printf("没有找到此人!\n");
                    break;
                }
                printf("\n");
                printf("name:%s%s\n",                T1->data.lastname,
T1->data.firstname);
                printf("\n");
                printf("friends:");
                output_relation(T1->data.friends, T);
                printf("\n\n");
                printf("fans:");
                output_relation(T1->data.fans, T);
                printf("\n\n");
                printf("following:");
                output_relation(T1->data.following, T);
                printf("\n\n");
                printf("likes:");
                output_likes(T1->data.likes, L);
                printf("\n\n");
                break;
            }
            case 18: {
                int key1, key2;
                BATree T1 = NULL, T2 = NULL, T0 = NULL;
                printf("请输入其中一个人的关键字:");
                scanf("%d", &key1);
                getchar();
                printf("请输入另一人的关键字:");
                scanf("%d", &key2);
                getchar();
                if (!set_member(T, key1, &T1)) {
                    printf("没有找到此人!\n");
                    break;
                }
                if (!set_member(T, key2, &T2)) {
                    printf("没有找到此人!\n");
```

```
                break;
            }
           set_intersection(T1->data.following, T2->data.following,
&T0);
            if (T0) {
                output_relation(T0, T);
                break;
            }
            else {
                printf("没有共同关注!\n");
                break;
            }
        }
        case 19: {
            int key1, key2;
            BATree T1 = NULL, T2 = NULL, T0 = NULL;
            printf("请输入其中一个人的关键字:");
            scanf("%d", &key1);
            getchar();
            printf("请输入另一人的关键字:");
            scanf("%d", &key2);
            getchar();
            if (!set_member(T, key1, &T1)) {
                printf("没有找到此人!\n");
                break;
            }
            if (!set_member(T, key2, &T2)) {
                printf("没有找到此人!\n");
                break;
            }
            set_intersection(T1->data.likes, T2->data.likes, &T0);
            if (T0) {
                output_likes(T0, L);
                break;
            }
            else {
                printf("没有共同爱好!\n");
                break;
            }
        }
        case 20: {
            int key;
            BATree T1 = NULL, T0 = NULL;
            printf("请输入关键字:");
```

```
            scanf("%d", &key);
            getchar();
            printf("\n");
            if (!set_member(T, key, &T1)) {
                printf("没有找到此人!\n");
                break;
            }
            indirect_friends_traverse(T, T1->data.friends, &T0);
            set_difference(&T0, T1->data.friends);
            set_remove(&T0, T1->data.key, &shorter);
            output_relation(T0, T);
            break;
        }
        case 21: {
            InOrderTraverse(T);
            break;
        }
        case 22: {
            insert_name(T);
            getchar();
            break;
        }
        case 23: {
            printf("请选择对好友集执行的操作：\n");
            printf("0 初始化  1 增加好友  2 删除好友  3 查找好友  4
遍历好友\n");
            int i;
            scanf("%d", &i);
            switch (i)
            {
            case 0:init_friends(T); break;
            case 1:insert_friends(T); break;
            case 2:remove_friends(T); break;
            case 3:search_friends(T); break;
            case 4:traverse_friends(T); break;
            default:printf("错误输入！");
            }
            getchar();
            break;
        }
        case 24: {
            printf("请选择对粉丝集执行的操作：\n");
            printf("0 初始化  1 增加粉丝  2 删除粉丝  3 查找粉丝  4
遍历粉丝\n");
```

```
            int i;
            scanf("%d", &i);
            switch (i)
            {
            case 0:init_fans(T); break;
            case 1:insert_fans(T); break;
            case 2:remove_fans(T); break;
            case 3:search_fans(T); break;
            case 4:traverse_fans(T); break;
            default:printf("错误输入！");
            }
            getchar();
            break;
        }
        case 25: {
            printf("请选择对关注集执行的操作：\n");
            printf("0 初始化  1 增加关注  2 删除关注  3 查找关注  4
遍历关注\n");
            int i;
            scanf("%d", &i);
            switch (i)
            {
            case 0:init_following(T); break;
            case 1:insert_following(T); break;
            case 2:remove_following(T); break;
            case 3:search_following(T); break;
            case 4:traverse_following(T); break;
            default:printf("请输入正确的操作符号！");
            }
            getchar();
            break;
        }
        case 26: {
            printf("请选择对爱好集执行的操作：\n");
            printf("0 初始化  1 增加爱好  2 删除爱好  3 查找爱好  4
遍历爱好\n");
            int i;
            scanf("%d", &i);
            switch (i)
            {
            case 0:init_likes(T,L); break;
            case 1:insert_likes(T,L); break;
            case 2:remove_likes(T,L); break;
            case 3:search_likes(T,L); break;
```

```
        case 4:traverse_likes(T,L); break;
        default:printf("请输入正确的操作符号！");
        }
        getchar();
        break;
    }
    case 27: {
        input_likes(&L, &l_size);
        if (load(&T))
            printf("加载成功！\n");
        else
            printf("加载失败！\n");
        getchar();
        break;
    }
    case 28: {
        if (save(T))
            printf("保存成功！\n");
        else
            printf("保存失败！\n");
        getchar();
        break;
    }
    case 29: {
        avl_init(&T);
        break;
    }
    case 30: {
        avl_destroy(&T);
        printf("销毁成功!\n");
        break;
    }
    case 31: {
        int key;
        printf("请输入关键字:");
        scanf("%d", &key);
        getchar();
        printf("\n");
        if (avl_search(T, key, &Tsub)) printf("元素在树中!\n");
        else printf("元素不在树中!\n");
        break;
    }
    case 32: {
        Info e;
```

```
            input_key(&e);
            if (avl_insert(&T, e, &taller)) {
                printf("插入成功!\n");
                height(T, 0);
            }
            else printf("插入失败!\n");
            break;
        }
        case 33: {
            int key;
            printf("请输入关键字:");
            scanf("%d", &key);
            getchar();
            printf("\n");
            if (avl_delete(&T, key, &shorter)) {
                printf("删除成功!\n");
                height(T, 0);
            }
            else printf("删除失败!\n");
            break;
        }
        case 34: {
            avl_traverse(T);
            break;
        }
        default:
            printf("输入错误!\n");
         }
        getchar();
        system("cls");
        }while (i);
    printf("\n--------------------          欢 迎 下 次 使 用 !
--------------------\n");
    return 0;
}
```

```
/**
* 函数名称：L_Rotate
* 函数参数：二叉树的根*p
* 函数功能：对以*p 为根的二叉树作左旋处理
* 返回值：新的树根结点
```

```
 **/
void L_Rotate(BATree *p) {
    BATree temp = NULL;
    temp = (*p)->rchild;//temp 指向 p 的右子树根节点
    (*p)->rchild = temp->lchild;//temp 左子树作为 p 的右子树
    temp->lchild = (*p);
    (*p) = temp;//*p 指向新的根节点
}


/**
* 函数名称：R_Rotate
* 函数参数：二叉树的根*p
* 函数功能：对以*p 为根的二叉树作右旋处理
* 返回值：新的树根结点
**/
void R_Rotate(BATree *p) {
    BATree temp = NULL;
    temp = (*p)->lchild;//temp 指向 p 的左子树根节点
    (*p)->lchild = temp->rchild;//temp 右子树作为 p 的左子树
    temp->rchild = (*p);
    (*p) = temp;//*p 指向新的根节点
}


/**
* 函数名称：LeftBalance
* 函数参数：二叉树*T
* 函数功能：二叉树左子树过高，对其做左平衡
* 返回值：新二叉树*T
**/
void LeftBalance(BATree *T) {
    BATree l = NULL, r = NULL;
    l = (*T)->lchild;
    switch (l->bflag){
    case LH://LL
        (*T)->bflag = EH;
        l->bflag = EH;
        R_Rotate(T);//右旋 T
        break;
    case EH:
        (*T)->bflag = LH;
        l->bflag = RH;
        R_Rotate(T);//右旋 T
```

```
        break;
    case RH://LR
        r = l->rchild;
        switch (r->bflag){
        case LH:
            (*T)->bflag = RH;
            l->bflag = EH;
            break;
        case EH:
            (*T)->bflag = EH;
            l->bflag = EH;
            break;
        case RH:
            (*T)->bflag = EH;
            l->bflag = LH;
            break;
        }
        r->bflag = EH;
        L_Rotate(&((*T)->lchild));//左旋根左孩子
        R_Rotate(T);//右旋 T
        break;
    }
}


/**
* 函数名称：RightBalance
* 函数参数：二叉树*T
* 函数功能：二叉树右子树过高，对其做右平衡
* 返回值：新二叉树*T
**/
void RightBalance(BATree *T) {
    BATree r = NULL, l = NULL;
    r = (*T)->rchild;
    switch (r->bflag) {
    case LH://RL
        l = r->lchild;
        switch (l->bflag)
        {
        case LH:
            (*T)->bflag = EH;
            r->bflag = RH;
            break;
        case EH:
```

```
            (*T)->bflag = EH;
            r->bflag = EH;
            break;
        case RH:
            (*T)->bflag = LH;
            r->bflag = EH;
            break;
        }
        l->bflag = EH;
        R_Rotate(&((*T)->rchild));//右旋根右孩子
        L_Rotate(T);//左旋根
        break;
    case EH:
        (*T)->bflag = RH;
        r->bflag = LH;
        L_Rotate(T);//左旋根
        break;
    case RH://RR
        (*T)->bflag = EH;
        r->bflag = EH;
        L_Rotate(T);//左旋根
        break;
    }
}


/**
* 函数名称：set_init
* 函数参数：二叉树*T
* 函数功能：初始化二叉树
* 返回值：初始化后的二叉树
**/
void set_init(BATree *T) {
     *T = NULL;
     printf("初始化成功!\n");
}

void set_init_fors(BATree *T)
{
    (*T) = (BATree)malloc(sizeof(BATNode));
    (*T)->lchild = NULL;
    (*T)->rchild = NULL;
    (*T)->bflag = EH;
    (*T)->h = 0;
```

```
}


/**
* 函数名称：set_destory
* 函数参数：二叉树*T
* 函数功能：销毁二叉树
* 返回值：无
**/
void set_destory(BATree *T) {
    if ((*T) != NULL) {
        set_destory(&((*T)->lchild));
        set_destory(&((*T)->rchild));
        free(*T);
    }
}


/**
* 函数名称：set_insert
* 函数参数：二叉树*T，信息 e，bool 型变量 taller(二叉树是否增高)
* 函数功能：在平衡二叉树中插入节点 e，并保持二叉树平衡
* 返回值：成功返回 true，否则返回 false
**/
bool set_insert(BATree *T, Info e, bool *taller) {
    if ((*T)==NULL)
    {
        (*T) = (BATree)malloc(sizeof(BATNode));//分配空间
        (*T)->data = e;
        (*T)->lchild = NULL;
        (*T)->rchild = NULL;
        (*T)->bflag = EH;
        *taller = true;//变高为 true
    }
    else {
        if (e.key == (*T)->data.key) {//含已有关键字
            *taller = false;
            return false;
        }
        if (e.key < (*T)->data.key) {//左子树插入
            if (!set_insert(&((*T)->lchild), e, taller))return false;//
插入失败
            if (*taller)//插入成功
                switch ((*T)->bflag) {
```

```
            case LH://左边高
                LeftBalance(T);
                *taller = false;
                break;
            case EH://等高
                (*T)->bflag = LH;
                *taller = true;
                break;
            case RH://右边高
                (*T)->bflag = EH;
                *taller = false;
                break;
            }
        }
        else {//右子树中插入
            if (!set_insert(&((*T)->rchild), e, taller))return false;//
插入失败
            if (*taller)//插入成功
                switch ((*T)->bflag) {
                case LH://左边高
                    (*T)->bflag = EH;
                    *taller = false;
                    break;
                case EH://等高
                    (*T)->bflag = RH;
                    *taller = true;
                    break;
                case RH://右边高
                    RightBalance(T);
                    *taller = false;
                    break;
                }
        }
    }
    return true;
}


/**
* 函数名称：set_remove
* 函数参数：二叉树*T，关键字 key，bool 型变量 shorter(二叉树是否变矮)
* 函数功能：在平衡二叉树中删除节点 e，并保持二叉树平衡
* 返回值：成功返回 true，否则返回 false
**/
```

```
bool set_remove(BATree *T, int key, bool *shorter) {
    if ((*T) == NULL)return false;//不存在
    else if (key == (*T)->data.key) {
        BATree p;
        if ((*T)->lchild == NULL) {//左子树空
            p = (*T);
            (*T) = (*T)->rchild;
            free(p);
            *shorter = true;
        }
        else if ((*T)->rchild == NULL) {//右子树空
            p = (*T);
            (*T) = (*T)->lchild;
            free(p);
            *shorter = true;
        }
        else {//左右子树都不为空
            p = (*T)->lchild;
            while (p->rchild) {
                p = p->rchild;
            }
            (*T)->data = p->data;
            set_remove(&((*T)->lchild), p->data.key, shorter);//左子树
递归删除前驱节点
            if (*shorter) {
                switch ((*T)->bflag) {
                case LH:
                    (*T)->bflag = EH;
                    *shorter = true;
                    break;
                case EH:
                    (*T)->bflag = RH;
                    *shorter = false;
                    break;
                case RH:
                    if ((*T)->rchild->bflag == EH) *shorter = false;//当
前节点右孩子状态为EH
                    else *shorter = true;
                    RightBalance(T);//右平衡
                    break;
                }
            }
        }
    }
```

```
    else if (key<(*T)->data.key) {//左子树中继续找节点
        if (!set_remove(&((*T)->lchild), key, shorter))return false;
        if (*shorter) {
            switch ((*T)->bflag) {
            case LH:
                (*T)->bflag = EH;
                *shorter = true;
                break;
            case EH:
                (*T)->bflag = RH;
                *shorter = false;
                break;
            case RH:
                if ((*T)->rchild->bflag == EH) *shorter = false;//节点右
孩子状态 EH
                else *shorter = true;
                RightBalance(T);//右平衡
                break;
            }
        }
    }
    else {//右子树中继续找节点
        if (!set_remove(&((*T)->rchild), key, shorter))return false;
        if (*shorter) {
            switch ((*T)->bflag) {
            case LH:
                if ((*T)->lchild->bflag == EH) *shorter = false;//节点左
孩子状态为 EH
                else *shorter = true;
                LeftBalance(T);//左平衡
                break;
            case EH:
                (*T)->bflag = LH;
                *shorter = false;
                break;
            case RH:
                (*T)->bflag = EH;
                *shorter = true;
                break;
            }
        }
    }
    return true;
}
```

```
/**
* 函数名称：set_intersection
* 函数参数：二叉树 T，二叉树 T1，二叉树*T0
* 函数功能：求 T 与 T1 的交集
* 返回值：交集 T0
**/
void set_intersection(BATree T, BATree T1, BATree *T0) {
    if (T1 == NULL)return;
    if (set_member(T, T1->data.key, &Tsub)) set_insert(T0, T1->data,
&taller);//若元素为交集里的元素
    set_intersection(T, T1->lchild, T0);
    set_intersection(T, T1->rchild, T0);
}


/**
* 函数名称：set_union
* 函数参数：二叉树*T，二叉树 T1
* 函数功能：求 T 与 T1 的并集
* 返回值：并集 T
**/
void set_union(BATree *T, BATree T1) {
    if (T1 == NULL)return;
    set_insert(T, T1->data, &taller);//插入
    set_union(T, T1->lchild);
    set_union(T, T1->rchild);
}


/**
* 函数名称：set_difference
* 函数参数：二叉树*T，二叉树 T1
* 函数功能：求 T 与 T1 的差
* 返回值：差 T
**/
void set_difference(BATree *T, BATree T1) {
    if (*T == NULL || T1 == NULL)return;
    set_remove(T, T1->data.key, &shorter);//删除 T1
    set_difference(T, T1->lchild);
    set_difference(T, T1->rchild);
}
```

```
/**
* 函数名称：set_size
* 函数参数：二叉树 T1，二叉树 T
* 函数功能：求 T 的个数
* 返回值：无
**/
void set_size(BATree T1, BATree T) {
    if (T1 == NULL)return;
    set_size(T1->lchild, T);
    if (set_member(T, T1->data.key, &Tsub))count++;//数目加 1
    set_size(T1->rchild, T);
}


/**
* 函数名称：set_member
* 函数参数：二叉树 T，关键字 key,二叉树*Tsub(指向要找的节点)
* 函数功能：判断关键字为 key 的节点是否是 T 中的成员
* 返回值：是返回 true，否则返回 false
**/
bool set_member(BATree T, int key, BATree *Tsub) {
    if (T == NULL)return false;
    if (T->data.key == key) {
        *Tsub = T;
        return true;
    }
    else if (key<T->data.key) {//左子树查找
        if (set_member(T->lchild, key, Tsub))return true;
    }
    else {if (set_member(T->rchild, key, Tsub))return true; }//右子树
查找
    return false;
}


/**
* 函数名称：set_subset
* 函数参数：二叉树 T，二叉树 T1
* 函数功能：判断 T1 是否是 T 的子集
* 返回值：是返回 true，否则返回 false
**/
bool set_subset(BATree T, BATree T1) {
    if (T1 == NULL)return true;
```

```
    if (set_member(T, T1->data.key, &Tsub)) {
        if (!set_subset(T, T1->lchild)) return false;//有成员不被 T 包含
        if (!set_subset(T, T1->rchild)) return false;//有成员不被 T 包含
        return true;
    }
    else return false;
}


/**
* 函数名称：set_equal
* 函数参数：二叉树 T，二叉树 T1
* 函数功能：判断 T1 是否与 T 相等
* 返回值：是返回 true，否则返回 false
**/
bool set_equal(BATree T, BATree T1) {
    if (!set_subset(T, T1))return false;//T 是否是 T1 子集
    if (!set_subset(T1, T))return false;//T1 是否是 T 的子集
    return true;
}


/**
* 函数名称：load_data
* 函数参数：二叉树*T，文件名 filename
* 函数功能：加载数据
* 返回值：成功返回 true，否则返回 false
**/
bool load_data(BATree *T, char *filename) {
    BATree p;
    FILE *fp;
    if ((fp = fopen(filename, "rb")) == NULL) {
        printf("打开文件失败!\n");
        return false;
    }
    while (!feof(fp)) {
        if ((p = (BATree)malloc(sizeof(BATNode))) == NULL) {//分配空间
            printf("储存空间不足!\n");
            fclose(fp);
            return false;
        }
        if (fread(p, sizeof(BATNode), 1, fp) != 1) {
            free(p);//读完
            break;
```

```
        }
        set_insert(T, p->data, &taller);//插入
    }
    fclose(fp);
    height(*T, 0);
    return true;
}


/**
* 函数名称：save_data
* 函数参数：二叉树 T，文件指针 fp
* 函数功能：存储数据
* 返回值：成功返回 true，否则返回 false
**/
bool save_data(BATree T, FILE *fp) {
    if (T) {
        if (fwrite(T, sizeof(BATNode), 1, fp) != 1) {//写入文件
            printf("保存失败!\n");
            fclose(fp);
            return false;
        }
    }
    else return true;
    if (!save_data(T->lchild, fp))return false;//递归调用
    if (!save_data(T->rchild, fp))return false;
    return true;
}


/**
* 函数名称：input_key
* 函数参数：数据指针 data
* 函数功能：输入数据的关键字
* 返回值：无
**/
void input_key(Info *data) {
    printf("请输入关键字:");
    scanf("%d", &(data->key));//输入关键字
    getchar();
    printf("\n");
}
```

```
/**
* 函数名称：height
* 函数参数：二叉树 T，变量 i
* 函数功能：计算二叉树高
* 返回值：二叉树高
**/
void height(BATree T, int i) {
    if(T) T->h = i + 1;
    else return;
    height(T->lchild, T->h);//递归求高度
    height(T->rchild, T->h);
}


/**
* 函数名称：output_key
* 函数参数：二叉树 T
* 函数功能：输出关键字
* 返回值：无
**/
void output_key(BATree T) {
    printf("%d", T->data.key);//输出关键字
}


/**
* 函数名称：create
* 函数参数：二叉树*T
* 函数功能：创建二叉树
* 返回值：无
**/
void create(BATree *T) {
    char s;
    Info *e;
    if ((e = (Info *)malloc(sizeof(Info))) == NULL) {//分配空间
        printf("存储空间不足!\n");
        return;
    }
    input_key(e);
    if (set_insert(T, *e, &taller)) {//插入
        printf("创建成功，是否要继续?(y/n):");
        while (1) {
            s = getchar();
            getchar();
```

```
            printf("\n");
            switch (s) {
            case 'y':
                create(T);//递归调用
                height(*T, 0);
                return;
            case 'n':
                printf("完成创建!\n");
                return;
            default:
                printf("输入错误，请重新输入:");
            }
        }
    }
    else printf("创建失败!\n");
    return;
}


/**
* 函数名称：gotoxy
* 函数参数：变量 x，y
* 函数功能：设置光标位置
* 返回值：无
**/
void gotoxy(int x, int y) {
    COORD coord = { x,y };
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}


/**
* 函数名称：showtree
* 函数参数：二叉树 T，变量 x，y，output_t
* 函数功能：显示树图
* 返回值：无
**/
void showtree(BATree T, int x, int y, visit output_t) {
    if (T) {
        gotoxy(x + x_move, y + y_move);
        output_t(T);//输出
    }
    else return;
    if (T->lchild || T->rchild) {//左右孩子都有
```

```
        gotoxy(x + x_move, y + y_move + 1);
        printf(" | ");
        if (T->lchild) {//左孩子存在
            int i,j;
            i = 80 / (pow(2, T->h) + 1);
            j = i;
            gotoxy(x + x_move - j, y + y_move + 3);
            printf(" | ");
            gotoxy(x + x_move - i + 2, y + y_move + 2);
            while (i) {
                printf("_");
                i--;
            }
            showtree(T->lchild, x - j, y + 4, output_t);//递归调用
        }
        if (T->rchild) {//右孩子存在
            int i, j;
            i = 80 / (pow(2, T->h) + 1);
            j = i;
            gotoxy(x + x_move + 1, y + y_move + 2);
            while (i) {
                printf("_");
                i--;
            }
            gotoxy(x + x_move + j, y + y_move + 3);
            printf(" | ");
            showtree(T->rchild, x + j, y + 4, output_t);//递归调用
        }
    }
}


/**
* 函数名称：InOrderTraverse
* 函数参数：二叉树 T
* 函数功能：中序遍历
* 返回值：无
**/
void InOrderTraverse(BATree T){
    if (T == NULL)return;
    InOrderTraverse(T->lchild);
    printf("%3d%s%s       ",       T->data.key,       T->data.lastname,
T->data.firstname);//输出姓名信息
    InOrderTraverse(T->rchild);
```

```
}


/**
* 函数名称：output_relation
* 函数参数：二叉树 T1,二叉树 T
* 函数功能：输出关系
* 返回值：无
**/
void output_relation(BATree T1, BATree T) {
    if (T1 == NULL)return;
    output_relation(T1->lchild, T);
    if          (set_member(T,          T1->data.key,          &Tsub))
printf("%3d%s%s",Tsub->data.key,Tsub->data.lastname,Tsub->data.firstn
ame);//输出信息
    output_relation(T1->rchild,T);
}


/**
* 函数名称：output_likes
* 函数参数：二叉树 T1,二叉树 T
* 函数功能：输出爱好
* 返回值：无
**/
void output_likes(BATree T1, BATree T) {
    if (T1 == NULL)return;
    output_likes(T1->lchild, T);
    if    (set_member(T,    T1->data.key,    &Tsub))    printf("%3d%s",
Tsub->data.key, Tsub->data.like);//输出信息
    output_likes(T1->rchild, T);
}


/**
* 函数名称：input_pinfo
* 函数参数：二叉树*T,变量 p_size
* 函数功能：输入人信息
* 返回值：无
**/
bool input_pinfo(BATree *T, int p_size) {
    Info *e;
    FILE *fp1, *fp2;
    int num;
```

```
if ((fp1 = fopen("lastname.txt", "r")) == NULL) {
    printf("打开 lastname 失败!\n");
    return false;
}
if ((fp2 = fopen("firstname.txt", "r")) == NULL) {
    printf("打开 firstname 失败!\n");
    return false;
}
for (num = 0; num <= p_size; num++) {//随机生成大数据集
    if ((e = (Info *)malloc(sizeof(Info))) == NULL) {
        printf("存储空间不足!\n");
        fclose(fp1);//关闭指针
        fclose(fp2);
        return false;
    }
    if (feof(fp1)) {
        fclose(fp1);
        if ((fp1 = fopen("lastname.txt", "r")) == NULL) {
            printf("打开 lastname 失败!\n");
            return false;
        }
    }
    else {
        if (fgets(e->lastname, 3, fp1) == NULL) {//读取姓
            free(e);
            break;
        }
    }
    if (feof(fp2)) {
        fclose(fp2);
        if ((fp2 = fopen("firstname.txt", "r")) == NULL){
            printf("打开 firstname 失败!\n");
            return false;
        }
    }
    else {
        if (fgets(e->firstname, 5, fp2) == NULL) {//读取名
            free(e);
            break;
        }
    }
    e->key = num;//关键字分配
    e->friends = NULL;//置空
    e->fans = NULL;
```

```
        e->following = NULL;
        e->likes = NULL;
        set_insert(T, *e, &taller);//插入
    }
    fclose(fp1);
    fclose(fp2);
    height(*T, 0);
    return true;
}


/**
* 函数名称：input_likes
* 函数参数：二叉树 T,变量 l_size
* 函数功能：输入爱好
* 返回值：无
**/
bool input_likes(BATree *T, int *l_size) {
    Info *e;
    FILE *fp;
    int num = 0;
    if ((fp = fopen("likes.txt", "r")) == NULL) {
        printf("打开 likes 失败!\n");
        return false;
    }
    while (!feof(fp)) {
        if ((e = (Info *)malloc(sizeof(Info))) == NULL) {//分配空间
            printf("存储空间不足!\n");
            fclose(fp);
            return false;
        }
        if (fgets(e->like, 5, fp) == NULL) {//读取爱好
            free(e);
            break;
        }
        e->key = num++;//分配关键字
        e->friends = NULL;
        e->fans = NULL;
        e->following = NULL;
        e->likes = NULL;
        set_insert(T, *e, &taller);//插入
    }
    *l_size = num;
    fclose(fp);//关闭文件指针
```

```
    height(*T, 0);
    return true;
}


/**
* 函数名称：input_relation
* 函数参数：二叉树 T,变量 p_size,l_size
* 函数功能：输入关系
* 返回值：无
**/
void input_relation(BATree *T, int p_size, int l_size) {
    if (*T == NULL)return;
    input_information(&((*T)->data.friends), p_size, 80, 20);//随机生成好友
    input_information(&((*T)->data.fans), p_size, 60, 10);//随机生成粉丝
    input_information(&((*T)->data.following), p_size, 30, 5);//随机生成关注
    input_information(&((*T)->data.likes), l_size, 8, 2);//随机生成爱好
    input_relation(&((*T)->lchild), p_size, l_size);//递归
    input_relation(&((*T)->rchild), p_size, l_size);
}


/**
* 函数名称：input_information
* 函数参数：二叉树 T，变量 size，max，min
* 函数功能：输入 key
* 返回值：无
**/
void input_information(BATree *T, int size, int max, int min) {
    int i;
    Info *e;
    for (i = 0; i <= rand() % max + min; i++) {//随机分配
        if ((e = (Info *)malloc(sizeof(Info))) == NULL) {
            printf("存储空间不足!\n");
            break;
        }
        e->key = rand() % size;
        e->friends = NULL;//置空
        e->fans = NULL;
        e->following = NULL;
```

```
        e->likes = NULL;
        set_insert(T, *e, &taller);//插入信息
    }
}


/**
* 函数名称：improve_sets
* 函数参数：二叉树 T,二叉树 T0
* 函数功能：完善集合
* 返回值：无
**/
void improve_sets(BATree T, BATree T0) {
    if (T0 == NULL)return;
    improve_sets(T, T0->lchild);//递归
    improve_friends(T, T0, T0->data.friends);//完善好友
    improve_fans(T, T0, T0->data.fans);//完善粉丝
    improve_following(T, T0, T0->data.following);//完善关注
    set_remove(&T0->data.friends, T0->data.key, &shorter);//删除自己
    set_remove(&T0->data.fans, T0->data.key, &shorter);//删除自己
    set_remove(&T0->data.following, T0->data.key, &shorter);//删除自己
    improve_sets(T, T0->rchild);
}


/**
* 函数名称：improve_friends
* 函数参数：二叉树 T1,二叉树 T1，二叉树 T2
* 函数功能：完善好友集，完善好友与好友之间的关系
* 返回值：无
**/
void improve_friends(BATree T, BATree T1, BATree T2) {
    if (T2 == NULL)return;
    BATree T3;
    set_member(T, T2->data.key, &Tsub);
    if     (!set_member(Tsub->data.friends,     T1->data.key,     &T3))
set_insert(&Tsub->data.friends, T1->data, &taller);//使得好友是双向的
    improve_friends(T, T1, T2->lchild);//递归
    improve_friends(T, T1, T2->rchild);
}


/**
* 函数名称：improve_fans
```

```
* 函数参数：二叉树 T1,二叉树 T1，二叉树 T2
* 函数功能：完善粉丝的关注集
* 返回值：无
**/
void improve_fans(BATree T, BATree T1, BATree T2) {
    if (T2 == NULL)return;
    BATree T3;
    set_member(T, T2->data.key, &Tsub);
    if    (!set_member(Tsub->data.following,    T1->data.key,    &T3))
set_insert(&Tsub->data.following, T1->data, &taller);//自己没有在粉丝
的关注人集里
    improve_fans(T, T1, T2->lchild);//递归
    improve_fans(T, T1, T2->rchild);
}


/**
* 函数名称：improve_following
* 函数参数：二叉树 T1,二叉树 T1，二叉树 T2
* 函数功能：完善关注的粉丝集
* 返回值：无
**/
void improve_following(BATree T, BATree T1, BATree T2) {
    if (T2 == NULL)return;
    BATree T3;
    set_member(T, T2->data.key, &Tsub);
    if      (!set_member(Tsub->data.fans,      T1->data.key,      &T3))
set_insert(&Tsub->data.fans, T1->data, &taller);//自己没有在关注人的粉
丝集里
    improve_following(T, T1, T2->lchild);//递归
    improve_following(T, T1, T2->rchild);
}


/**
* 函数名称：indirect_friends_traverse
* 函数参数：二叉树 T1,二叉树 T1，二叉树*T0
* 函数功能：二度好友遍历
* 返回值：无
**/
void indirect_friends_traverse(BATree T, BATree T1, BATree *T0) {
    if (T1 == NULL)return;
    set_member(T, T1->data.key, &Tsub);
    indirect_friends_insert(Tsub->data.friends, T0);//插入所有连接好友
```

```
    indirect_friends_traverse(T, T1->lchild, T0);//遍历
    indirect_friends_traverse(T, T1->rchild, T0);
}


/**
* 函数名称：indirect_friends_insert
* 函数参数：二叉树 T,二叉树*T0
* 函数功能：二度好友输入
* 返回值：无
**/
void indirect_friends_insert(BATree T, BATree *T0) {
    if (T == NULL)return;
    set_insert(T0, T->data, &taller);//插入
    indirect_friends_insert(T->lchild, T0);
    indirect_friends_insert(T->rchild, T0);
}


/**
* 函数名称：insert_name
* 函数参数：二叉树 T
* 函数功能：插入姓名
* 返回值：无
**/
bool insert_name(BATree T) {
    int key;
    BATree T1;
    printf("请输入成员关键字：");
    scanf("%d", &key);
    if (!set_member(T, key, &T1))
    {
        printf("该成员不存在！");
        return false;
    }
    printf("请输入姓：");
    scanf("%s", T1->data.lastname);
    printf("请输入名：");
    scanf("%s", T1->data.firstname);
    printf("插入成功！");
    return true;
}
```

```
/**
* 函数名称：traverse_del_friends
* 函数参数：二叉树 T，T1
* 函数功能：完善初始化好友集
* 返回值：无
**/
bool traverse_del_friends(BATree T, BATree T1) {
    if (T)
    {
        if (set_member(T->data.friends, T1->data.key, &Tsub))
        {
            set_remove(&(T->data.friends), T1->data.key, &shorter);
        }
        traverse_del_friends(T->lchild, T1);
        traverse_del_friends(T->rchild, T1);
    }
    return true;
}


/**
* 函数名称：init_friends
* 函数参数：二叉树 T
* 函数功能：输出化好友集
* 返回值：无
**/
bool init_friends(BATree T) {
    int key;
    BATree T1;
    printf("请输入成员关键字：");
    scanf("%d", &key);
    if (!set_member(T, key, &T1))
    {
        printf("该成员不存在！");
        return false;
    }
    set_init_fors(&(T1->data.friends));//初始化好友集
    traverse_del_friends(T, T1);//将其他
}


/**
* 函数名称：insert_friends
* 函数参数：二叉树 T
```

```
* 函数功能：插入好友
* 返回值：无
**/
bool insert_friends(BATree T) {
    int key1, key2;
    BATree T1, T2, T0;
    printf("请输入成员关键字：");
    scanf("%d", &key1);
    if (!set_member(T, key1, &T1))
    {
        printf("该成员不存在！\n");
        return false;
    }
    if (T1->data.friends == NULL)//未初始化
    {
        printf("%d 的好友集未初始化！\n", key1);
        return false;
    }
    if (T1->data.friends == NULL)
        printf("%d 当前无好友！\n", key1);
    else
    {
        printf("%d 当前好友有：", key1);
        output_relation(T1->data.friends, T);//输出好友
        printf("\n");
    }
    printf("请输入要插入的好友关键字：");
    scanf("%d", &key2);
    if (!set_member(T, key2, &T2))
    {
        printf("该成员不存在！");
        return false;
    }
    if (key1 == key2)//不能输入自己
    {
        printf("不能输入自己！");
        return false;
    }
    if (T2->data.friends == NULL)
    {
        printf("%d 的好友集未初始化！\n", T2->data.key);
        return false;
    }
    if (!set_member(T1->data.friends, key2, &T0))//不是好友
```

```
    {
        Info data1, data2;
        data1.key = key2;
        data2.key = key1;
        set_insert(&(T1->data.friends), data1, &taller);//T1 好友插入 T2
        set_insert(&(T2->data.friends), data2, &taller);//T2 好友插入 T1
        printf("%d 与%d 互相成为好友！\n", key1, key2);
        return true;
    }
    printf("该好友已存在！");
    return false;
}


/**
* 函数名称：remove_friends
* 函数参数：二叉树 T
* 函数功能：删除好友
* 返回值：无
**/
bool remove_friends(BATree T) {
    int key1, key2;
    BATree T1, T2, T0;
    printf("请输入成员关键字：");
    scanf("%d", &key1);
    if (!set_member(T, key1, &T1))
    {
        printf("该成员不存在！");
        return false;
    }
    if (T1->data.friends == NULL)//未初始化
    {
        printf("%d 的好友集未初始化！\n", key1);
        return false;
    }
    if (T1->data.friends == NULL)
        printf("%d 当前无好友！\n", key1);
    else
    {
        printf("%d 当前好友有：", key1);
        output_relation(T1->data.friends, T);//输出好友
        printf("\n");
    }
    printf("请输入要删除的好友关键字：");
```

```
    scanf("%d", &key2);
    if (!set_member(T, key2, &T2))
    {
        printf("该成员不存在！");
        return false;
    }
    if (T1->data.friends == NULL)//好友本就不存在
    {
        printf("该好友不存在！");
        return false;
    }
    if (key1 == key2)//不能输入自己
    {
        printf("不能输入自己！");
        return false;
    }
    if (set_member(T1->data.friends, key2, &T0))//T2 是 T1 好友
    {
        set_remove(&(T1->data.friends), key2, &shorter);//T1 好友集删除
T2
        set_remove(&(T2->data.friends), key1, &shorter);//T2 好友集删除
T1
        printf("%d 与%d 互删成功！\n", key1, key2);
        return true;
    }
    printf("该好友不存在！");
    return false;
}


/**
* 函数名称：search_friends
* 函数参数：二叉树 T
* 函数功能：搜索好友
* 返回值：无
**/
bool search_friends(BATree T) {
    int key1, key2;
    BATree T1, T2, T0;
    printf("请输入成员关键字：");
    scanf("%d", &key1);
    if (!set_member(T, key1, &T1))
    {
        printf("该成员不存在！");
```

```
        return false;
    }
    printf("请输入好友关键字：");
    scanf("%d", &key2);
    if (!set_member(T, key2, &T2))//成员不存在
    {
        printf("该成员不存在！");
        return false;
    }
    if (T1->data.friends == NULL)//无好友
    {
        printf("%d 无好友！\n", key1);
        return false;
    }
    if (key1 == key2)//不能输入自己
    {
        printf("不能输入自己！");
        return false;
    }
    if (set_member(T1->data.friends, key2, &T0))//是好友
    {
        printf("%d 是%d 的好友！\n", key2, key1);
        return true;
    }
    printf("%d 不是%d 的好友！\n", key2, key1);
    return false;
}


/**
* 函数名称：traverse_friends
* 函数参数：二叉树 T
* 函数功能：遍历好友
* 返回值：无
**/
bool traverse_friends(BATree T) {
    int key;
    BATree T1;
    printf("请输入成员关键字：");
    scanf("%d", &key);
    if (!set_member(T, key, &T1))
    {
        printf("该成员不存在！");
        return false;
```

```
    }
    if (T1->data.friends == NULL)//未初始化
    {
        printf("%d 的好友集未初始化！\n", T1->data.key);
        return false;
    }
    if (T1->data.friends == NULL)
        printf("%d 当前无好友！\n", key);
    else
    {
        printf("%d 当前的好友有：", key);
        output_relation(T1->data.friends,T);//输出好友
        printf("\n");
    }
}


/**
* 函数名称：traverse_del_following
* 函数参数：二叉树 T，T1
* 函数功能：完善初始化粉丝集
* 返回值：无
**/
bool traverse_del_following(BATree T, BATree T1) {
    if (T)
    {
        if (set_member(T->data.following, T1->data.key, &Tsub))
        {
            set_remove(&(T->data.following), T1->data.key, &shorter);
        }
        traverse_del_following(T->lchild, T1);
        traverse_del_following(T->rchild, T1);
    }
    return true;
}


/**
* 函数名称：init_fans
* 函数参数：二叉树 T
* 函数功能：初始化粉丝集
* 返回值：无
**/
bool init_fans(BATree T) {
```

```
    int key;
    BATree T1;
    printf("请输入成员关键字：");
    scanf("%d", &key);
    if (!set_member(T, key, &T1))
    {
        printf("该成员不存在！\n");
        return false;
    }
    set_init_fors(&(T1->data.fans));//初始化
    traverse_del_following(T, T1);
}


/**
* 函数名称：insert_fans
* 函数参数：二叉树 T
* 函数功能：插入粉丝
* 返回值：无
**/
bool insert_fans(BATree T) {
    int key1, key2;
    BATree T1, T2, T0;
    printf("请输入成员关键字：");
    scanf("%d", &key1);
    if (!set_member(T, key1, &T1))
    {
        printf("该成员不存在！\n");
        return false;
    }
    if (T1->data.fans == NULL)//未初始化
    {
        printf("%d 的粉丝集未初始化！\n", key1);
        return false;
    }
    if (T1->data.fans == NULL) printf("%d 当前无粉丝！\n", key1);
    else
    {
        printf("%d 当前粉丝有：", key1);
        output_relation(T1->data.fans,T);//输出粉丝
        printf("\n");
    }
    printf("请输入要插入的粉丝关键字：");
    scanf("%d", &key2);
```

```
    if (!set_member(T, key2, &T2))
    {
        printf("该成员不存在！\n");
        return false;
    }
    if (key1 == key2)//不能输入自己
    {
        printf("不能输入自己！\n");
        return false;
    }
    if (!set_member(T1->data.fans, key2, &T0))//T2 不是 T1 粉丝
    {
        Info data1;
        Info data2;
        data1.key = key2;
        data2.key = key1;
        set_insert(&(T1->data.fans), data1, &taller);//T2 插入 T1 粉丝集
        set_insert(&(T2->data.following), data2, &taller);//T1 插入 T2
关注集
        printf("插入成功！\n");
        return true;
    }
    printf("该粉丝已存在！\n");
    return false;
}


/**
* 函数名称：remove_fans
* 函数参数：二叉树 T
* 函数功能：删除粉丝
* 返回值：无
**/
bool remove_fans(BATree T) {
    int key1, key2;
    BATree T1, T2, T0;
    printf("请输入成员关键字：");
    scanf("%d", &key1);
    if (!set_member(T, key1, &T1))
    {
        printf("该成员不存在！\n");
        return false;
    }
    if (T1->data.fans == NULL)//未初始化
```

```
    {
        printf("%d 的粉丝集未初始化！\n", key1);
        return false;
    }
    if (T1->data.fans == NULL)
        printf("%d 当前无粉丝！\n", key1);
    else
    {
        printf("%d 当前的粉丝有：", key1);
        output_relation(T1->data.fans, T);//输出粉丝
        printf("\n");
    }
    printf("请输入要删除的粉丝关键字：");
    scanf("%d", &key2);
    if (!set_member(T, key2, &T2))
    {
        printf("该成员不存在！");
        return false;
    }
    if (key1 == key2)//不能输入自己
    {
        printf("不能输入自己！");
        return false;
    }
    if (set_member(T1->data.fans, key2, &T0))
    {
        set_remove(&(T1->data.fans), key2,&shorter);//将 T2 从 T1 粉丝集
里删除
        set_remove(&(T2->data.following), key1, &shorter);//将 T1 从 T2
关注集里删除
        printf("删除成功！\n");
        return true;
    }
    printf("该粉丝不存在！\n");
    return false;
}


/**
* 函数名称：search_fans
* 函数参数：二叉树 T
* 函数功能：搜索粉丝
* 返回值：无
**/
```

```
bool search_fans(BATree T) {
    int key1, key2;
    BATree T1, T2, T0;
    printf("请输入成员关键字：");
    scanf("%d", &key1);
    if (!set_member(T, key1, &T1))
    {
        printf("该成员不存在！\n");
        return false;
    }
    if (T1->data.fans == NULL)//无粉丝
    {
        printf("%d 无粉丝！\n", key1);
        return false;
    }
    printf("请输入要查找的粉丝关键字：");
    scanf("%d", &key2);
    if (!set_member(T, key2, &T2))
    {
        printf("该成员不存在！");
        return false;
    }
    if (key1 == key2)//不能输入自己
    {
        printf("不能输入自己！");
        return false;
    }
    if (set_member(T1->data.fans, key2, &T0))//是成员
    {
        printf("%d 是%d 的粉丝！\n", key2, key1);
        return true;
    }
    printf("%d 不是%d 的粉丝！\n", key2, key1);
    return false;
}


/**
* 函数名称：traverse_fans
* 函数参数：二叉树 T
* 函数功能：遍历粉丝
* 返回值：无
**/
bool traverse_fans(BATree T) {
```

```
    int key;
    BATree T1;
    printf("请输入成员编号：");
    scanf("%d", &key);
    if (!set_member(T, key, &T1))
    {
        printf("该成员不存在！");
        return false;
    }
    if (T1->data.fans == NULL)//未初始化
    {
        printf("%d 的粉丝集未初始化！\n", T1->data.key);
        return false;
    }
    if (T1->data.fans == NULL)
        printf("%d 当前无粉丝！\n", key);
    else
    {
        printf("%d 当前粉丝有：", key);
        output_relation(T1->data.fans, T);//输出粉丝
        printf("\n");
    }
}


/**
* 函数名称：traverse_del_fans
* 函数参数：二叉树 T，T1
* 函数功能：完善初始化关注集
* 返回值：无
**/
bool traverse_del_fans(BATree T, BATree T1) {
    if (T)
    {
        if (set_member(T->data.fans, T1->data.key, &Tsub))
        {
            set_remove(&(T->data.fans), T1->data.key, &shorter);
        }
        traverse_del_fans(T->lchild, T1);
        traverse_del_fans(T->rchild, T1);
    }
    return true;
}
```

```
/**
* 函数名称：init_following
* 函数参数：二叉树 T
* 函数功能：初始化关注集
* 返回值：无
**/
bool init_following(BATree T) {
    int key;
    BATree T1;
    printf("请输入成员关键字：");
    scanf("%d", &key);
    if (!set_member(T, key, &T1))
    {
        printf("该成员不存在！");
        return false;
    }
    set_init_fors(&(T1->data.following));//初始化
    traverse_del_fans(T, T1);
}


/**
* 函数名称：insert_following
* 函数参数：二叉树 T
* 函数功能：插入关注
* 返回值：无
**/
bool insert_following(BATree T) {
    int key1, key2;
    BATree T1, T2, T0;
    printf("请输入成员关键字：");
    scanf("%d", &key1);
    if (!set_member(T, key1, &T1))
    {
        printf("该成员不存在！");
        return false;
    }
    if (T1->data.following == NULL)//未初始化
    {
        printf("%d 的粉丝集未初始化！\n", key1);
        return false;
    }
    if (T1->data.following == NULL) printf("%d 当前无关注！\n", key1);
```

```
    else
    {
        printf("%d 当前关注有：", key1);
        output_relation(T1->data.following,T);//输出关注
        printf("\n");
    }
    printf("请输入要插入的关注关键字：");
    scanf("%d", &key2);
    if (!set_member(T, key2, &T2))
    {
        printf("该成员不存在！");
        return false;
    }
    if (key1 == key2)//不能输入自己
    {
        printf("不能输入自己！");
        return false;
    }
    if (!set_member(T1->data.following, key2, &T0))//不是成员
    {
        Info data1;
        Info data2;
        data1.key = key2;
        data2.key = key1;
        set_insert(&(T1->data.following), data1,&taller);//将 T2 插入 T1
的关注集
        set_insert(&(T2->data.fans), data2, &taller);//将 T1 插入 T2 的粉
丝集
        printf("插入成功！");
        return true;
    }
    printf("该关注已存在！");
    return false;
}


/**
* 函数名称：remove_following
* 函数参数：二叉树 T
* 函数功能：删除关注
* 返回值：无
**/
bool remove_following(BATree T) {
    int key1, key2;
```

```
    BATree T1, T2, T0;
    printf("请输入成员关键字：");
    scanf("%d", &key1);
    if (!set_member(T, key1, &T1))
    {
        printf("该成员不存在！");
        return false;
    }
    if (T1->data.following == NULL)//未初始化
    {
        printf("%d 的粉丝集未初始化！\n", key1);
        return false;
    }
    if (T1->data.following == NULL)
        printf("%d 当前无关注！\n", key1);
    else
    {
        printf("%d 当前关注有：", key1);
        output_relation(T1->data.following,T);//输出关注
        printf("\n");
    }
    printf("请输入要取消关注的成员关键字：");
    scanf("%d", &key2);
    if (!set_member(T, key2, &T2))
    {
        printf("该成员不存在！");
        return false;
    }
    if (key1 == key2)//不能输入自己
    {
        printf("不能输入自己！");
        return false;
    }
    if (set_member(T1->data.following, key2, &T0))
    {
        set_remove(&(T1->data.following), key2,&shorter);//将 T2 从 T1
关注集删除
        set_remove(&(T2->data.fans), key1, &shorter);//将 T1 从 T2 粉丝集
删除
        printf("删除成功！");
        return true;
    }
    printf("该关注不存在！");
    return false;
```

```
}


/**
* 函数名称：search_following
* 函数参数：二叉树 T
* 函数功能：搜索关注
* 返回值：无
**/
bool search_following(BATree T) {
    int key1, key2;
    BATree T1, T2, T0;
    printf("请输入成员关键字：");
    scanf("%d", &key1);
    if (!set_member(T, key1, &T1))
    {
        printf("该成员不存在！");
        return false;
    }
    if (T1->data.following == NULL)//无关注
    {
        printf("%d 无关注！\n", key1);
        return false;
    }
    printf("请输入要查找的关注关键字：");
    scanf("%d", &key2);
    if (!set_member(T, key2, &T2))
    {
        printf("该成员不存在！");
        return false;
    }
    if (key1 == key2)//不能输入自己
    {
        printf("不能输入自己！");
        return false;
    }
    if (set_member(T1->data.following, key2, &T0))//是成员
    {
        printf("%d 是%d 的关注！\n", key2, key1);
        return true;
    }
    printf("%d 不是%d 的关注！\n", key2, key1);
    return false;
}
```

```
/**
* 函数名称：traverse_following
* 函数参数：二叉树 T
* 函数功能：遍历关注
* 返回值：无
**/
bool traverse_following(BATree T) {
    int key;
    BATree T1;
    printf("请输入成员关键字：");
    scanf("%d", &key);
    if (!set_member(T, key, &T1))
    {
        printf("该成员不存在！");
        return false;
    }
    if (T1->data.following == NULL)//未初始化
    {
        printf("%d 的粉丝集未初始化！\n", key);
        return false;
    }
    if (T1->data.following == NULL)//无关注
        printf("%d 当前无关注！\n", key);
    else
    {
        printf("%d 当前关注有：", key);
        output_relation(T1->data.following,T);//输出关注
        printf("\n");
    }
}


/**
* 函数名称：init_likes
* 函数参数：二叉树 T，二叉树 L
* 函数功能：初始化爱好集
* 返回值：无
**/
bool init_likes(BATree T, BATree L) {
    int key;
    BATree T1, T0;
    printf("请输入成员关键字：");
```

```
    scanf("%d", &key);
    if (!set_member(T, key, &T1))
    {
        printf("该成员不存在！");
        return false;
    }
    set_init_fors(&(T1->data.likes));//初始化
    T1->data.likes->h = 0;
}


/**
* 函数名称：insert_likes
* 函数参数：二叉树 T，二叉树 L
* 函数功能：插入爱好
* 返回值：无
**/
bool insert_likes(BATree T, BATree L) {
    int key;
    BATree T1, T0;
    printf("请输入成员关键字：");
    scanf("%d", &key);
    if (!set_member(T, key, &T1))
    {
        printf("该成员不存在！");
        return false;
    }
    if (T1->data.likes == NULL)//未初始化
    {
        printf("%d 的爱好集未初始化！\n", key);
        return false;
    }
    if (T1->data.likes == NULL)
        printf("%d 当前无爱好！\n", key);
    else
    {
        printf("%d 当前爱好有：", key);
        output_likes(T1->data.likes, L);//输出爱好
        printf("\n");
    }
    int key2;
    printf("请输入要插入的爱好关键字：");
    scanf("%d", &key2);
    if (!set_member(L, key2, &T0))
```

```
    {
        printf("该爱好不存在！");
        return false;
    }
    if (!set_member(T1->data.likes, key2, &T0))//爱好没有存在
    {
        Info data1;
        data1.key = key2;
        set_insert(&(T1->data.likes), data1, &taller);//插入爱好
        printf("插入成功！");
        return true;
    }
    printf("该爱好已存在！");
    getchar();
    return false;
}


/**
* 函数名称：remove_likes
* 函数参数：二叉树 T，二叉树 L
* 函数功能：删除爱好
* 返回值：无
**/
bool remove_likes(BATree T, BATree L) {
    int key;
    BATree T1, T0;
    printf("请输入成员关键字：");
    scanf("%d", &key);
    if (!set_member(T, key, &T1))
    {
        printf("该成员不存在！");
        return false;
    }
    if (T1->data.likes == NULL)//未初始化
    {
        printf("%d 的爱好集未初始化！\n", T1->data.key);
        return false;
    }
    if (T1->data.likes == NULL)
        printf("%d 当前无爱好！\n", key);
    else
    {
        printf("%d 当前爱好有：", key);
```

```
        output_likes(T1->data.likes, L);//输出爱好
        printf("\n");
    }
    int key2;
    printf("请输入删除爱好的关键字：");
    scanf("%d", &key2);
    if (!set_member(L, key2, &T0))//爱好不存在
    {
        printf("该爱好不存在！");
        return false;
    }
    if (set_member(T1->data.likes, key2, &T0))//存在
    {
        set_remove(&(T1->data.likes), key2, &shorter);//删除
        printf("删除成功！");
        return true;
    }
    printf("该爱好不存在！");

    return false;
}


/**
* 函数名称：search_likes
* 函数参数：二叉树 T，二叉树 L
* 函数功能：搜索爱好
* 返回值：无
**/
bool search_likes(BATree T, BATree L) {
    int key;
    BATree T1, T0;
    printf("请输入成员关键字：");
    scanf("%d", &key);
    if (!set_member(T, key, &T1))
    {
        printf("该成员不存在！");
        return false;
    }
    if (T1->data.likes == NULL)//未初始化
    {
        printf("%d 的爱好集未初始化！\n", T1->data.key);
        return false;
    }
```

```
    int key2;
    printf("请输入要查找的爱好关键字：");
    scanf("%d", &key2);
    if (set_member(T1->data.likes, key2, &T0))//是爱好
    {
        printf("%d 是爱好！\n", key2);
        return true;
    }
    printf("%d 不是爱好！\n", key2);
    return false;
}


/**
* 函数名称：traverse_likes
* 函数参数：二叉树 T，二叉树 L
* 函数功能：遍历爱好
* 返回值：无
**/
bool traverse_likes(BATree T, BATree L) {
    int key;
    BATree T1;
    printf("请输入成员关键字：");
    scanf("%d", &key);
    if (!set_member(T, key, &T1))
    {
        printf("该成员不存在！");
        return false;
    }
    if (T1->data.likes == NULL)//未初始化
    {
        printf("%d 的爱好集未初始化！\n", T1->data.key);
        return false;
    }
    if (T1->data.likes == NULL)
        printf("%d 当前无爱好！\n", key);
    else
    {
        printf("%d 当前的爱好有：", key);
        output_likes(T1->data.likes,L);//输出爱好
        printf("\n");
    }
}
```

```
/**
* 函数名称：load
* 函数参数：二叉树*T
* 函数功能：加载更多信息
* 返回值：无
**/
bool load(BATree *T) {
    char filename[20];
    FILE *fp;
    Info data;
    char s[10000];
    int i = 1;
    int key;
    printf("1 加载好友  2 加载粉丝  3 加载关注  4 加载爱好  0 退出
\n");
    while (i)
    {
        printf("请选择操作：");
        scanf("%d", &i);
        if ((i != 0) && (i != 1) && (i != 2) && (i != 3) && (i != 4))
continue;//输入错误继续
        if (i == 0) break;
        printf("请输入文件名：");
        scanf("%s", filename);
        getchar();
        printf("\n");
        if ((fp = fopen(filename, "r")) == NULL)//打开文件
        {
            printf("文件打开失败！\n");
            return false;
        }
        switch (i)
        {
        case 1:while (fscanf(fp, "%d", &key) != EOF)
        {
            BATree Ts;
            if (!set_member(*T, key, &Ts))//找不到成员
            {
                printf("信息有误！\n");
                break;
            }
            set_init_fors(&(Ts->data.friends));//初始化朋友集
```

```
    while (1)
    {
        fscanf(fp, "%d", &key);
        if (key == 666) break;//下一位
        data.key = key;
        data.friends = NULL;
        data.fans = NULL;
        data.following = NULL;
        data.likes = NULL;
        set_insert(&(Ts->data.friends), data,&taller);//插入
    }
}
       break;
case 2:while (fscanf(fp, "%d", &key) != EOF)
{
    BATree Ts;
    if (!set_member(*T, key, &Ts))//找不到成员
    {
        printf("信息有误! \n");
        break;
    }
    set_init_fors(&(Ts->data.fans));//初始化
    while (1)
    {
        fscanf(fp, "%d", &key);
        if (key == 666) break;//下一个
        data.key = key;
        data.friends = NULL;
        data.fans = NULL;
        data.following = NULL;
        data.likes = NULL;
        set_insert(&(Ts->data.fans), data,&taller);//插入
    }
}
       break;
case 3:while (fscanf(fp, "%d", &key) != EOF)
{
    BATree Ts;
    if (!set_member(*T, key, &Ts))//找不到成员
    {
        printf("信息有误! \n");
        break;
    }
    set_init_fors(&(Ts->data.following));//初始化
```

```
            while (1)
            {
                fscanf(fp, "%d", &key);
                if (key == 666) break;//下一个
                data.key = key;
                data.friends = NULL;
                data.fans = NULL;
                data.following = NULL;
                data.likes = NULL;
                set_insert(&(Ts->data.following), data,&taller);//插入
            }
        }
               break;
        case 4:while (fscanf(fp, "%d", &key) != EOF)
        {
            BATree Ts;
            if (!set_member(*T, key, &Ts))//找不到成员
            {
                printf("信息有误! \n");
                break;
            }
            set_init_fors(&(Ts->data.likes));//初始化
            while (1)
            {
                fscanf(fp, "%d", &key);
                if (key == 666) break;//下一个
                data.key = key;
                data.friends = NULL;
                data.fans = NULL;
                data.following = NULL;
                data.likes = NULL;
                set_insert(&(Ts->data.likes), data, &taller);//插入
            }
        }
               break;
               fclose(fp);
        }
    }
    return true;
}
```

```
/**
* 函数名称：save
* 函数参数：二叉树 T
* 函数功能：保存更多信息
* 返回值：无
**/
bool save(BATree T) {
    FILE *fp;
    char filename[20];
    int i = 1;
    printf("1 保存好友   2 保存粉丝  3 保存关注  4 保存爱好  0 退出
\n");
    while (i)
    {
        printf("请选择操作：");
        scanf("%d", &i);
        if ((i != 0) && (i != 1) && (i != 2) && (i != 3) && (i != 4))
continue;//输入错误继续输入
        if (i == 0) break;
        printf("请输入文件名：");
        scanf("%s", filename);
        getchar();
        printf("\n");
        if ((fp = fopen(filename, "w")) == NULL)//打开文件
        {
            printf("文件打开失败！\n");
            return false;
        }
        save_relation(T, fp, i);//保存关系
        fclose(fp);
    }
    return true;
}


/**
* 函数名称：save_relation
* 函数参数：二叉树 T,文件指针*fp
* 函数功能：保存更多信息
* 返回值：无
**/
bool save_relation(BATree T, FILE *fp, int i) {
    if (T != NULL)
    {
```

```
        fprintf(fp, "%d ", T->data.key);
        switch (i)
        {
        case 1:
            traverse_relation(T->data.friends, fp);//保存朋友关系
            fprintf(fp, "%d ", 666);//标志
            break;
        case 2:
            traverse_relation(T->data.fans, fp);//保存粉丝关系
            fprintf(fp, "%d ", 666);
            break;
        case 3:
            traverse_relation(T->data.following, fp);//保存关注关系
            fprintf(fp, "%d ", 666);
            break;
        case 4:
            traverse_relation(T->data.likes, fp);//保存爱好
            fprintf(fp, "%d ", 666);
            break;
        }
        save_relation(T->lchild, fp, i);//递归
        save_relation(T->rchild, fp, i);
    }
}


/**
* 函数名称：traverse_relation
* 函数参数：二叉树 T,文件指针*fp
* 函数功能：遍历保存
* 返回值：无
**/
bool traverse_relation(BATree T, FILE *fp) {
    if (T)
    {
        fprintf(fp, "%d ", T->data.key);
        traverse_relation(T->lchild, fp);
        traverse_relation(T->rchild, fp);
    }
    return true;
}


/**
```

```
* 函数名称：avl_init
* 函数参数：二叉树*T
* 函数功能：初始化二叉树
* 返回值：初始化后的二叉树
**/
void avl_init(BATree *T) {
    *T = NULL;
    printf("初始化成功!\n");
}


/**
* 函数名称：avl_destroy
* 函数参数：二叉树*T
* 函数功能：销毁二叉树
* 返回值：无
**/
void avl_destroy(BATree *T) {
    if ((*T) != NULL) {
        avl_destroy(&((*T)->lchild));
        avl_destroy(&((*T)->rchild));
        free(*T);
    }
}


/**
* 函数名称：avl_insert
* 函数参数：二叉树*T，信息 e，bool 型变量 taller(二叉树是否增高)
* 函数功能：在平衡二叉树中插入节点 e，并保持二叉树平衡
* 返回值：成功返回 true，否则返回 false
**/
bool avl_insert(BATree *T, Info e, bool *taller) {
    if ((*T) == NULL)
    {
        (*T) = (BATree)malloc(sizeof(BATNode));//分配空间
        (*T)->data = e;
        (*T)->lchild = NULL;
        (*T)->rchild = NULL;
        (*T)->bflag = EH;
        *taller = true;
    }
    else {
        if (e.key == (*T)->data.key) {//含已有关键字
```

```
            *taller = false;
            return false;
        }
        if (e.key < (*T)->data.key) {//左子树插入
            if (!avl_insert(&((*T)->lchild), e, taller))return false;//
插入失败
            if (*taller)//插入成功
                switch ((*T)->bflag) {
                case LH://左边高
                    LeftBalance(T);
                    *taller = false;
                    break;
                case EH://等高
                    (*T)->bflag = LH;
                    *taller = true;
                    break;
                case RH://右边高
                    (*T)->bflag = EH;
                    *taller = false;
                    break;
                }
        }
        else {//右子树中插入
            if (!avl_insert(&((*T)->rchild), e, taller))return false;//
插入失败
            if (*taller)//插入成功
                switch ((*T)->bflag) {
                case LH://左边高
                    (*T)->bflag = EH;
                    *taller = false;
                    break;
                case EH://等高
                    (*T)->bflag = RH;
                    *taller = true;
                    break;
                case RH://右边高
                    RightBalance(T);
                    *taller = false;
                    break;
                }
        }
    }
    return true;
}
```

```
/**
* 函数名称：avl_delete
* 函数参数：二叉树*T，关键字 key，bool 型变量 shorter(二叉树是否变矮)
* 函数功能：在平衡二叉树中删除节点 e，并保持二叉树平衡
* 返回值：成功返回 true，否则返回 false
**/
bool avl_delete(BATree *T, int key, bool *shorter) {
    if ((*T) == NULL)return false;//不存在
    else if (key == (*T)->data.key) {
        BATree p;
        if ((*T)->lchild == NULL) {//左子树空
            p = (*T);
            (*T) = (*T)->rchild;
            free(p);
            *shorter = true;
        }
        else if ((*T)->rchild == NULL) {//右子树空
            p = (*T);
            (*T) = (*T)->lchild;
            free(p);
            *shorter = true;
        }
        else {//左右子树都不为空
            p = (*T)->lchild;
            while (p->rchild) {
                p = p->rchild;
            }
            (*T)->data = p->data;
            avl_delete(&((*T)->lchild), p->data.key, shorter);//左子树
递归删除前驱节点
            if (*shorter) {
                switch ((*T)->bflag) {
                case LH:
                    (*T)->bflag = EH;
                    *shorter = true;
                    break;
                case EH:
                    (*T)->bflag = RH;
                    *shorter = false;
                    break;
                case RH:
                    if ((*T)->rchild->bflag == EH) *shorter = false;//右
```

```
孩子为EH
                else *shorter = true;
                RightBalance(T);//右平衡
                break;
            }
        }
    }
}
else if (key<(*T)->data.key) {//左子树中继续找节点
    if (!avl_delete(&((*T)->lchild), key, shorter))return false;
    if (*shorter) {
        switch ((*T)->bflag) {
        case LH:
            (*T)->bflag = EH;
            *shorter = true;
            break;
        case EH:
            (*T)->bflag = RH;
            *shorter = false;
            break;
        case RH:
            if ((*T)->rchild->bflag == EH) *shorter = false;//右孩子
为EH
            else *shorter = true;
            RightBalance(T);//右平衡
            break;
        }
    }
}
else {//右子树中继续找节点
    if (!avl_delete(&((*T)->rchild), key, shorter))return false;
    if (*shorter) {
        switch ((*T)->bflag) {
        case LH:
            if ((*T)->lchild->bflag == EH) *shorter = false;//左孩子
为EH
            else *shorter = true;
            LeftBalance(T);//左平衡
            break;
        case EH:
            (*T)->bflag = LH;
            *shorter = false;
            break;
        case RH:
```

```
                (*T)->bflag = EH;
                *shorter = true;
                break;
            }
        }
    }
    return true;
}


/**
* 函数名称：avl_traverse
* 函数参数：二叉树 T
* 函数功能：中序遍历
* 返回值：无
**/
void avl_traverse(BATree T) {
    if (T == NULL)return;
    avl_traverse(T->lchild);
    printf("%3d", T->data.key);
    avl_traverse(T->rchild);
}


/**
* 函数名称：avl_search
* 函数参数：二叉树 T，关键字 key,二叉树*Tsub(指向要找的节点)
* 函数功能：判断关键字为 key 的节点是否是 T 中的成员
* 返回值：是返回 true，否则返回 false
**/
bool avl_search(BATree T, int key, BATree *Tsub) {
    if (T == NULL)return false;
    if (T->data.key == key) {
        *Tsub = T;
        return true;
    }
    else if (key<T->data.key) {//左子树查找
        if (avl_search(T->lchild, key, Tsub))return true;
    }
    else { if (avl_search(T->rchild, key, Tsub))return true; }//右子树
查找
    return false;
}
```