

**课 程 设 计 报 告**

**题目：基于AVL树表示的集合的实现与应用**

**课程名称： 数据结构课程设计**

**专业班级： CS1606**

**学 号： U201614683**

**姓 名： 薛旭升**

**指导教师： 周时阳**

**报告日期： 2018.3.27**

**计算机科学与技术学院**

# 任务书

**·**设计目的

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

**·**设计内容

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

主要数据对象：好友集、粉丝集、关注人集等。

主要数据关系：

（1）抽象层面AVL可以表示数据元素之间层次关系或一对多关系。

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

**·**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。

# 目录

[任务书 1](#_Toc16966)

[目录 2](#_Toc24512)

[1引言 3](#_Toc10064)

[1.1课题背景与意义 3](#_Toc28120)

[1.2国内外研究现状 3](#_Toc2826)

[1.3课程设计的主要研究工作 3](#_Toc18374)

[2系统需求分析与总体设计 5](#_Toc24715)

[2.1系统需求分析 5](#_Toc17791)

[2.2系统总体设计 5](#_Toc29767)

[3系统详细设计 7](#_Toc19636)

[3.1有关数据结构的定义 7](#_Toc23567)

[3.2主要算法设计 8](#_Toc20299)

[4系统实现 14](#_Toc4127)

[4.1系统实现 14](#_Toc17078)

[4.2系统测试 17](#_Toc15783)

[5总结与展望 41](#_Toc32486)

[5.1全文总结 41](#_Toc9827)

[6体会与改进 42](#_Toc10731)

[参考文献 43](#_Toc15945)

[附录 44](#_Toc15743)

# 1引言

## 1.1课题背景与意义

平衡二叉树（AVL 树）是数据结构中的重要知识点，在实际应用中多用于在内存中组织数据，对于平衡二叉树最大的应用就是来查找数据，因为它的查找的时间效率为 logn，因此就存在要创建平衡二叉树、对其进行插入、删除这三种基本操作。同时因它在动态查找表中的查找效率非常高，在地理信息处理、医学 模型处理以及快速成形等技术中都有广泛应用。

## 1.2国内外研究现状

当前对平衡二叉树的研究都基本上是对其的扩展和改进，像比较复杂的 B-

树、B+树、键树、2-3 树等。

## 1.3课程设计的主要研究工作

任务：

实现平衡二叉树的插入、删除、查找等基本功能，并基于平衡二叉树的基本功能实现集合的交、并、差等功能，最后利用集合实现应用层中共同关注、二度好友以及好友集、粉丝集等的增删查改。

分析：

平衡二叉树又称 AVL 树。它或者是一棵空树，或者是具 有下列性质的二叉树：它的左子树和右子树都是平衡 二叉树，且二叉树上的所有结点的平衡因子绝对值不超过 1。平衡二叉树的操作主要难点在于插入、删除功能的实现，每次插入、删除时需要不断判断左右子树高度，当二叉树失衡时，需要通过单左旋、单右旋及双旋进行平衡处理。

本次实验中集合使用二叉树表示，集合的交集功能可通过查找功能遍历得到，而并集功能只需将一棵二叉树插入另一棵二叉树即可，差集功能只需调用删除函数从一棵二叉树中将另一棵二叉树的成员删除即可。其他功能直接调用二叉树函数即可实现。

应用层功能大部分功能调用集合函数实现，只有随机生成数据功能需要使用库函数rand随机生成一个数，再根据文件内容生成信息。

# 2系统需求分析与总体设计

## 2.1系统需求分析

本系统最主要的需求为应用层功能，二叉树与集合为实现的方法。本系统类似于微博、qq等的好友系统，每个用户有好友集、粉丝集、关注集、喜好集四个集，系统需要能对这四个集的成员进行增、删、查、改四个功能，并能求取

共同关注、二度好友。

## 2.2系统总体设计

本程序主要由6个模块组成：

1. 主程序模块：

void main（）{

初始化；

do{

接受命令；

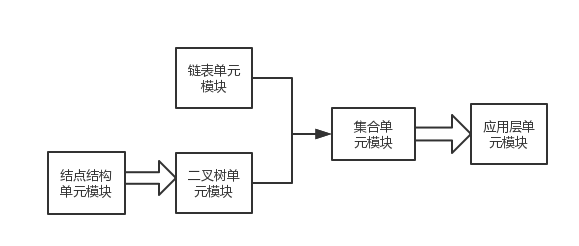
处理命令；

}while（“命令”=“退出”）；

}

1. 结点结构单元模块-----------定义二叉树的结点结构
2. 二叉树单元模块-----------实现二叉树的抽象数据类型
3. 集合单元模块-----------实现集合的抽象数据类型
4. 链表单元模块-----------实现多集合操作，使用户可以单独操作每个集合
5. 应用层单元模块------------基于集合实现本系统最终要求功能

系统结构模块图如图2-1：



**图2-1 系统结构模块图**

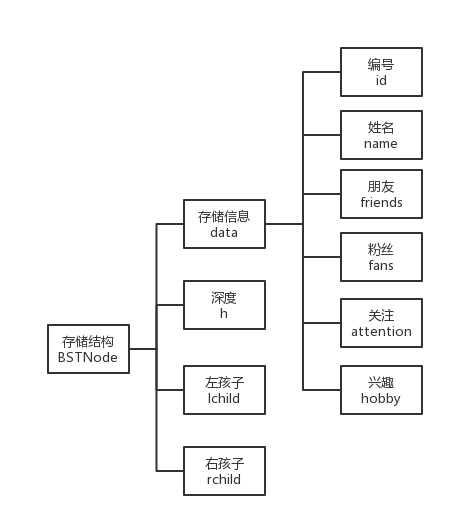
# 3系统详细设计

## 3.1有关数据结构的定义

该系统要处理的数据有二叉树与集合，两者基于相同的结点结构。其各数据项及数据项类型如表3-1，数据间关系如图3-1。

**表 3-1 数据项及类型**

|  |  |  |  |
| --- | --- | --- | --- |
| 数据项 | 数据类型 | 数据项 | 数据类型 |
| 存储结构BSTNode | BSTNode | 编号id | int |
| 存储信息data | ElemType | 姓名name | char[] |
| 深度h | int | 朋友friends | BSTree |
| 左孩子lchild | BSTree | 粉丝fans | BSTree |
| 右孩子rchild | BSTree | 关注attention | BSTree |
|  |  | 兴趣hobby | BSTree |



**图3-1 数据间关系**

## 3.2主要算法设计

1.左右平衡函数LeftBalance与RightBalance

左平衡处理函数和右平衡处理函数相似，故只该出右平衡处理函数流程图，左平衡处理函数调用了左旋函数与右旋函数，同样只给出左旋函数流程图。左旋函数流程图如图3-2，右平衡处理函数人如图3-3。



图3-2 左旋函数流程图

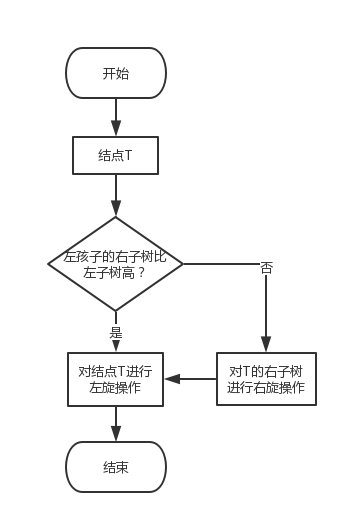


图3-3右平衡处理函数

1. 插入函数InsertAVL

插入函数使用递归法，先通过比较，不断遍历二叉树，直至到叶结点，插入后在层层递归回去，然后对每层结点深度比较，当结点失衡时进行平衡处理，一直返回到根结点结束。

由于是平衡二叉树，故若现在共有n个结点，则二叉树的高度为log2（n），由于每个结点还需计算深度，所以插入成功的时间复杂度为O（log(n)^2）。插入失败时，可计算其平均时间复杂度，第一层有1个结点，需要1步；第二层有2个结点，需要2步，……，第log2（n）层有[n/2]个结点，需要log2（n）步，其平均时间复杂度为O（nlog（n））。

流程图如图3-4。

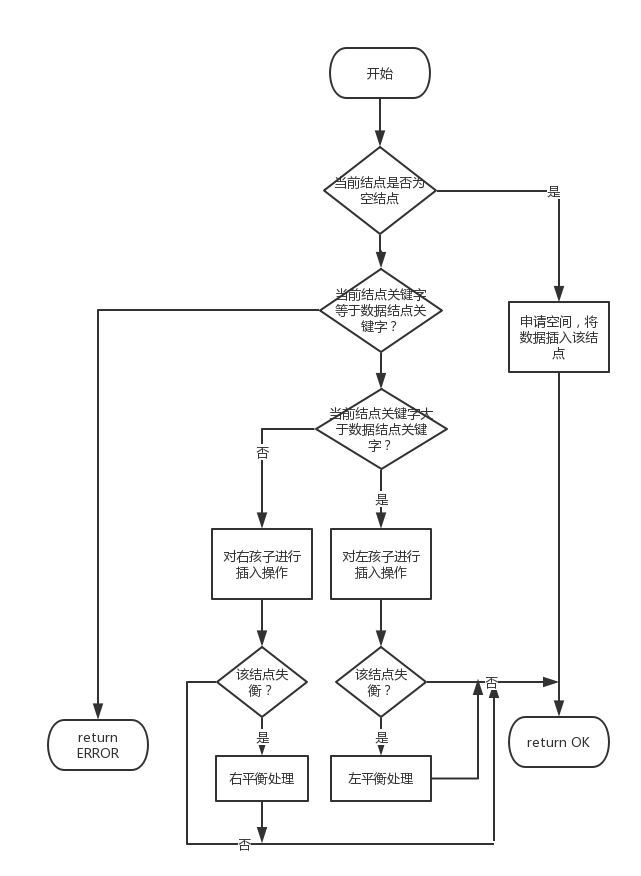


图3-4 插入函数流程图

1. 删除函数DeleteAVL

删除函数与插入函数平衡处理方式相同，不同的是对结点的处理方式，删除函数中删除结点的情况还分三种：删除的结点无孩子、只有一个孩子、有两个孩子。前两种情况只需删除结点即可，最后一种情况中若左子树更高，则用左子树的最右孩子替代结点，然后遍历结点的左子树，删除最右孩子；否则用右子树的最左孩子替代结点，然后遍历结点的右子树，删除最左孩子。

时间复杂度计算方法与插入函数相同，删除失败时时间复杂度为O（nlog（n）），删除成功时平均时间复杂度为O(log(n)^2)。

流程图如图3-5。

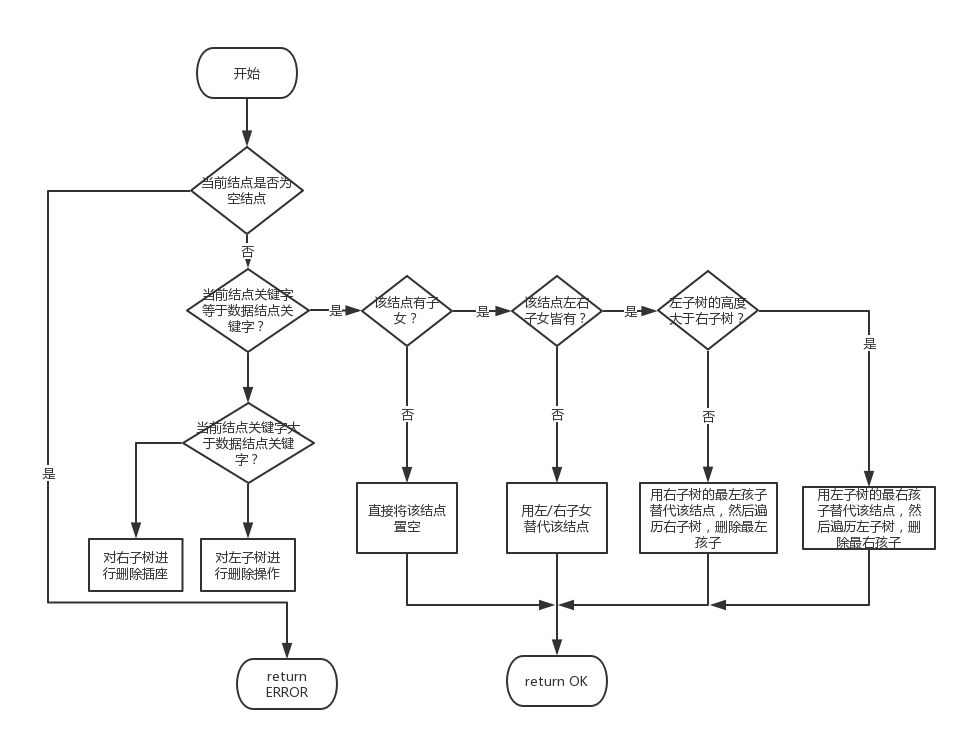


图3-5 删除函数流程图

1. 图形化输出函数TraverseAVL

图形化输出函数主要应用<windows.h>库中的SetConsoleCursorPosition函数，该函数可以使光标移动到任意位置。其具体流程图如图3-6。

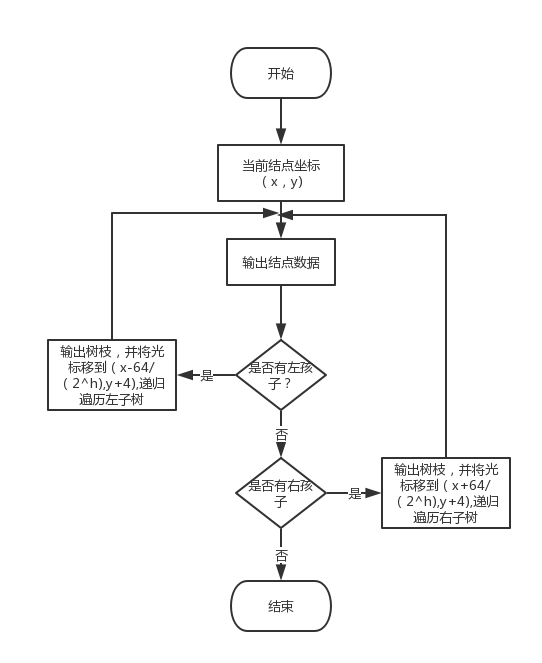


图 3-6 图形化输出函数

1. 交集函数set\_intersection、并集函数set\_union、差集函数set\_diffrence

这三个函数主要调用二叉树函数实现。交集函数需对二叉树T2的每个结点遍历，若当前结点在二叉树T1中查询成功，将该结点的数据插入一个新的二叉树，T2遍历结束后得到的新的二叉树即为所求交集。并集函数只需将T2的每个结点数据插入T1即可实现，同样差集函数只需将T2的每个结点数据从T1中删除即可。

1. 随机生成大数据

此功能通过三个函数实现，先通过Tree\_input函数生成一个主二叉树记录成员信息，再通过relation\_input函数生成关系，最后使用improve\_relation函数完善关系。其流程图如图3-7。

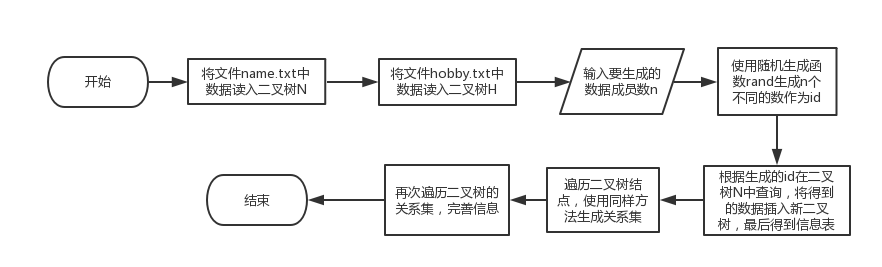


图3-7 随机生成大数据流程图

1. 求取二度好友函数second\_friend

流程图如图3-8.

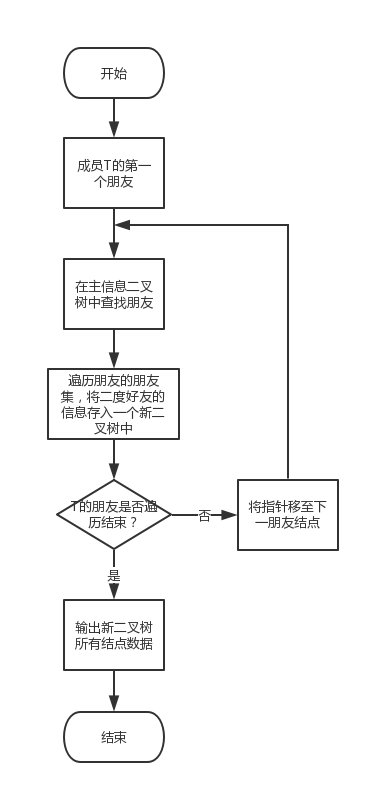


图3-8 求取二度好友函数

1. 链表的插入函数List\_insert

流程图如图3-9。

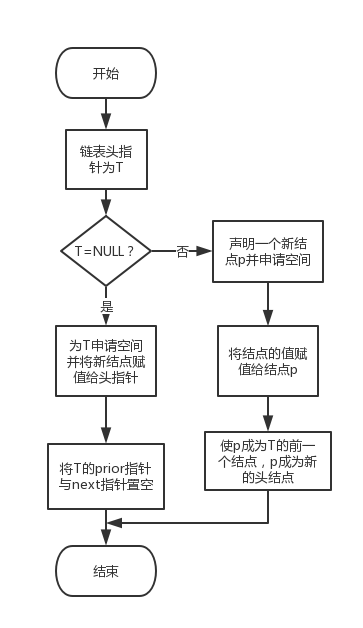


图3-9 链表插入函数流程图

# 4系统实现

## 4.1系统实现

程序能够实现对平衡二叉树的基本操作，并在此基础上数显对微博用户集合的常用操作。

编译环境为 Windows，编译软件为 Dev-C++。

结点的定义为

typedef struct ElemType{

int id;

char name[10]; //姓名

BSTNode\* friends; //好友集

BSTNode\* fans; //粉丝集

BSTNode\* attention; //关注人集

BSTNode\* hobby; //喜好

}ElemType;

typedef struct BSTNode{

ElemType data;

int h; //树的高度

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

}BSTNode,\*BSTree;

程序见附录。其主要函数如下：

**二叉树单元模块**：

status InitAVL(BSTree &T)；初始化平衡二叉排序树

status DestroyAVL(BSTree &T)；摧毁平衡二叉排序树

status SearchAVL(BSTree T,int e,BSTree &Ta)；在二叉树T中查找关键字为e的结点

void R\_Rotate(BSTree &p)；左旋函数

void L\_Rotate(BSTree &p)；右旋函数

status depth(BSTree T)；求取深度函数

void LeftBalance(BSTree &T)；左平衡处理函数

void RightBalance(BSTree &T)；右平衡处理函数

status InsertAVL(BSTree &T,ElemType e)；在二叉树T中插入结点e

status DeleteAVL(BSTree &T,int e)；删除二叉树T中结点id为e的结点

status PTraverseAVL(BSTree T)；将二叉树T以先序序列输出

void gotoxy(int x,int y)；移动光标，将光标移至坐标（x，y)

status height(BSTree &T,int i) ；计算二叉树T各结点的深度并赋值

status TraversAVL(int x,int y,BSTree T)；将二叉树图形化输出

status Save(BSTree T,FILE \*fp)；将二叉树T保存入fp指向的文件中

status Load(BSTree &T,char \*filename)；将fp文件名为filename的文件中的内容加载入二叉树T中

**集合单元模块：**

status set\_init(BSTree &T)；初始化集合

status set\_destroy(BSTree &T)；摧毁集合

status set\_member(BSTree T,int e,BSTree &Ta)；在集合T中查找关键字为e的结点

status set\_insert(BSTree &T,ElemType e)；在集合中插入结点e

status set\_remove(BSTree &T,int e)；删除集合T中id为e的成员

status set\_intersection(BSTree T1,BSTree T2,BSTree &T)；求集合T1与T2的交集，并赋值给T

status set\_union(BSTree &T,BSTree T1)；求集合T1与T的并集，并赋值给T

status set\_diffrence(BSTree &T,BSTree T1)；求二叉树T与T1的差集，并赋值给T

status set\_size(BSTree T)；计算集合成员个数

status set\_subset(BSTree T1,BSTree T2)；判断集合T2是否为T1的子集

status set\_equal(BSTree T1,BSTree T2)；判断集合T1与T2是否相等

status set\_traverse(BSTree T)；将集合以集合形式遍历输出

**链表单元模块：**

status TraverseLink(LinkList L)；输出链表L中所有集合的id

status search\_link(LinkList L,int id,LinkList &L1)；对链表L进行查询，若链表L中存在集合id为id的结点，将该结点赋值给L1，并返回TRUE，否则返回FALSE

status List\_insert(LinkList &L,Node e)；将数据e插入链表L中

status save\_list(LinkList L)；将链表信息保存入文件List.dat中

status save\_linklist(LinkList L)；将链表L的集合保存入集合名文件中

status load\_linklist(LinkList &L)；将文件List.dat的信息载入链表L中

status tree\_to\_link(LinkList &L,BSTree T)；将二叉树T存入链表L

void set\_menu(LinkList &L,BSTree &T,LinkList &buff)；对链表中各集合进行操作

**应用层单元模块：**

status common(BSTree T1,BSTree T2,BSTree &T)；求取数据项T1与T2的共同成员

status search\_friend(BSTree T,BSTree Ta,BSTree &Tb)；求取二度好友的辅助函数，二叉树Ta为T中某成员的朋友集，将朋友集中成员的朋友信息存入集合Tb

status second\_friend(BSTree T,int id)求取id为id的成员的二度好友

status Common\_attention(BSTree &T)；共同关注查找系统

status Common\_hobby(BSTree &T)；共同喜好查找系统

status hobby\_set\_input(BSTree &H,int &n)；读取文件hobby.txt,将其加载入二叉树H，n记录读取数量

status name\_set\_input(BSTree &N,int &n)；读取文件name.txt,将其加载入二叉树N，n记录读取数量

status name\_input(BSTree &T,int gross,int max,int min,BSTree N)；随机生成一棵二叉树，该二叉树结点id不超过gross，结点数为min-min+max

status relation\_input(BSTree &T,BSTree N,BSTree H,int a,int b)；随机生成关系

status Tree\_input(BSTree &T,BSTree N,int b；随机生成一个结点数量为b的成员信息

status improve\_friend(BSTree &T,BSTree Ta,ElemType e)；完善成员的朋友集信息

status improve\_attention(BSTree &T,BSTree Ta,ElemType e)；完善成员的关注集信息

status improve\_fans(BSTree &T,BSTree Ta,ElemType e)；完善成员的粉丝集信息,Ta为某成员关注集,e为该成员信息

status search\_relation(BSTree &T)；检查T的关系集中是否有自身，完善关系集status improve\_relation(BSTree &T,BSTree &T1)；完善成员关系集，使朋友互相存在，使成员的粉丝的关注中有该成员，防止成员的朋友集、粉丝集、关注集中有本人

status operator\_tree(BSTree &T,BSTree H)；对应用层中数据进行查找，输出其所有信息，随后进行删除、增加、修改等操作

status operator\_tree2(BSTree &T,BSTree H)；查找成员的单个关系信息，随后进行删除、增加、修改等操作

函数间调用关系如图4-1，其中箭头表示调用，A—>B表示函数B中调用了函数A，链表单元模块是集合单元模块的辅助函数，但只是使用户可以操作集合，两者函数间无调用关系。

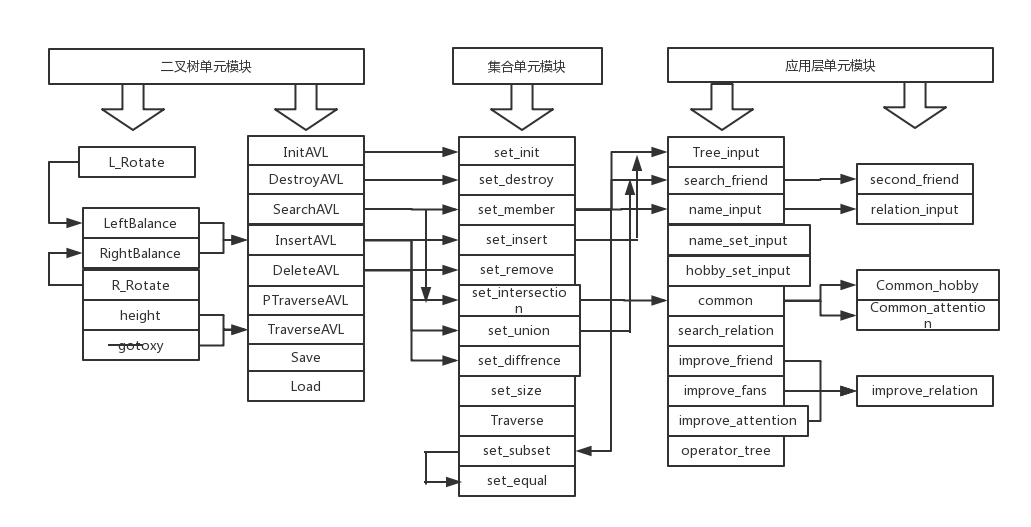


图4-1 函数间调用关系

## 4.2系统测试

系统整体目录如图4-2。

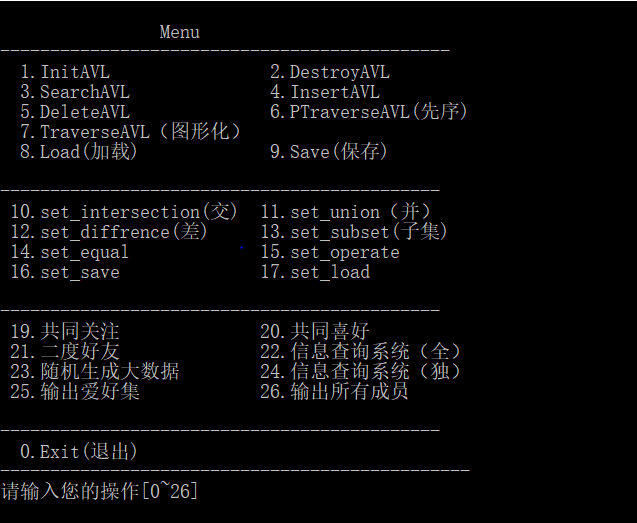


图4-2 目录

**二叉树部分：**

1. 先测试二叉树函数合法性，不初始化二叉树，运行函数，如图4-3。

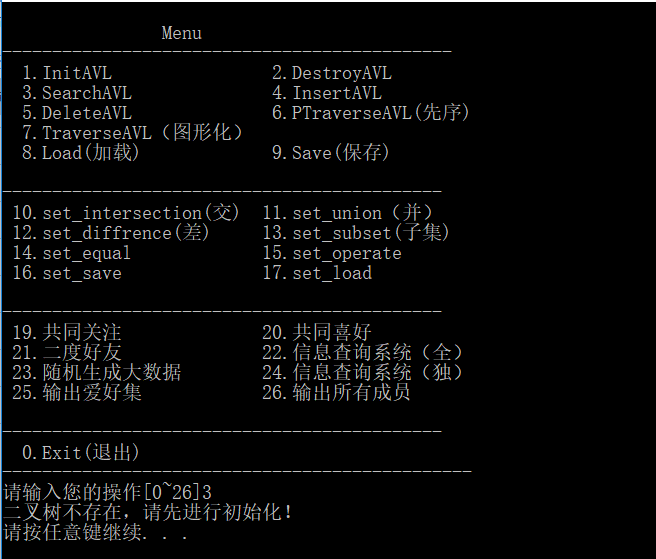


图4-3 函数合法性测试

1. 初始化函数测试，如图4-4。

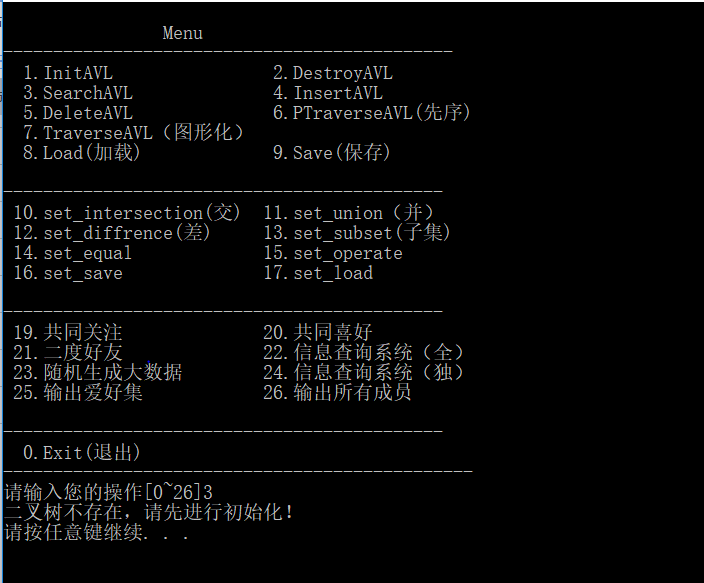


图4-4 初始化函数

1. 插入函数测试
2. 先插入4、5、6，测试单左旋函数并图形化输出，如图4-5、4-6。

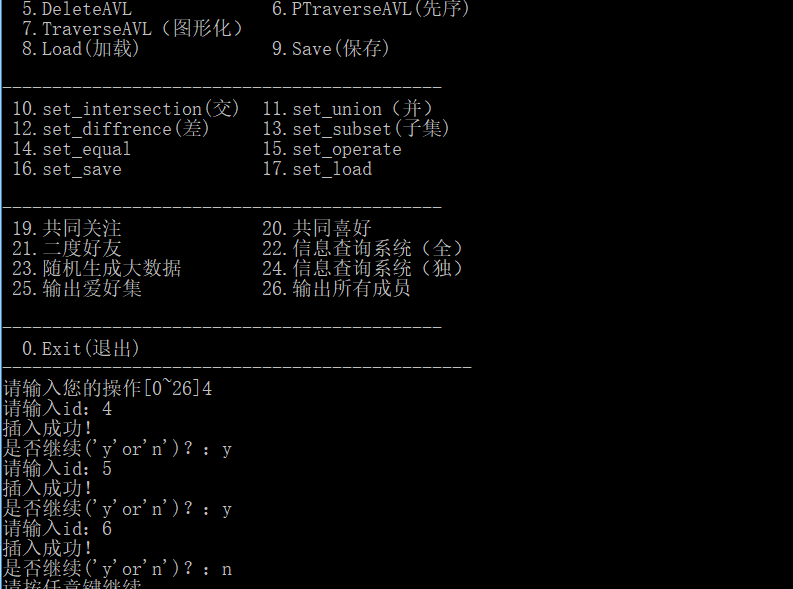


图4-5 插入函数1

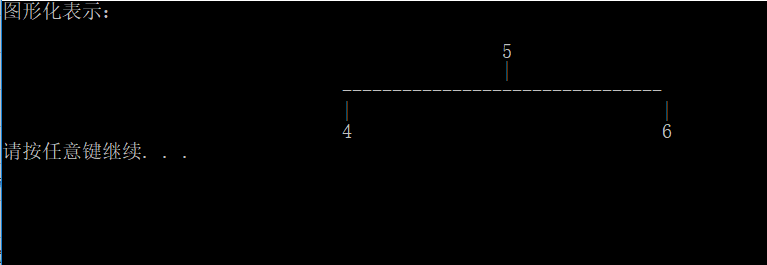


图4-6 图形化输出1

插入6后，头结点4失衡，对头结点进行单左旋操作，结点5成为新结点，单左旋函数实现。

1. 再插入1、0，测试单右旋函数，如图4-7、4-8。

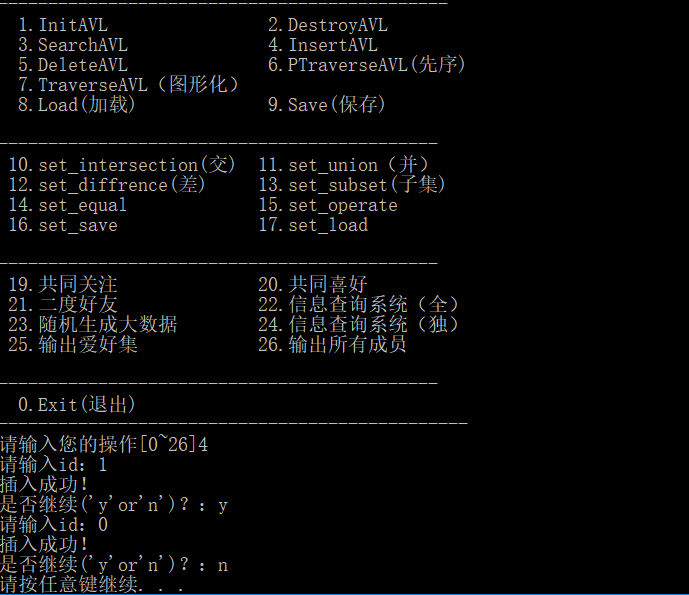


图4-7 插入函数2

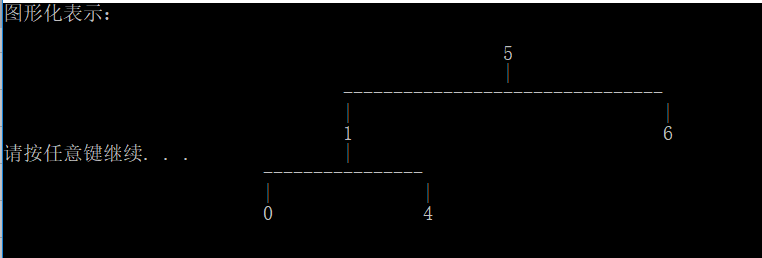


图4-8 图形化输出2

1与0都插入到左子树，使左子树失衡，在结点4处进行单右旋操作，单右旋函数实现。

1. 插入3，测试双旋函数，如图4-9、4-10。

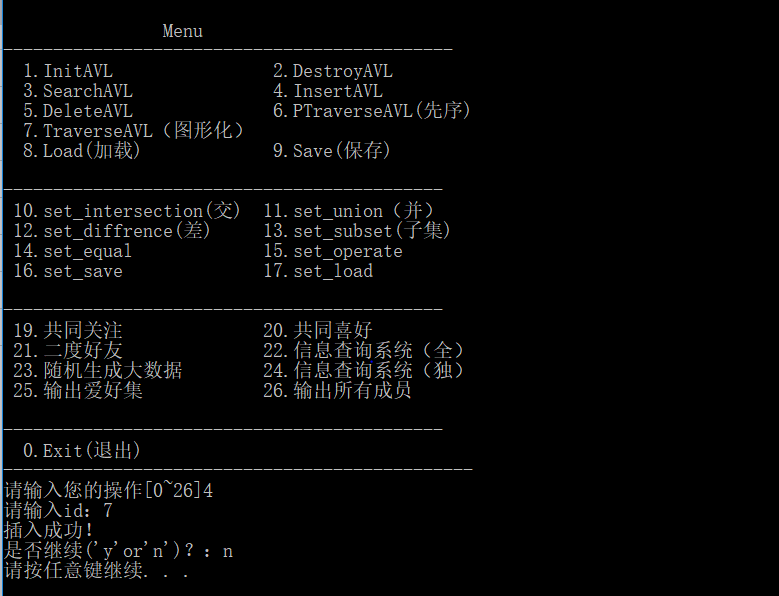


图4-9 插入函数3

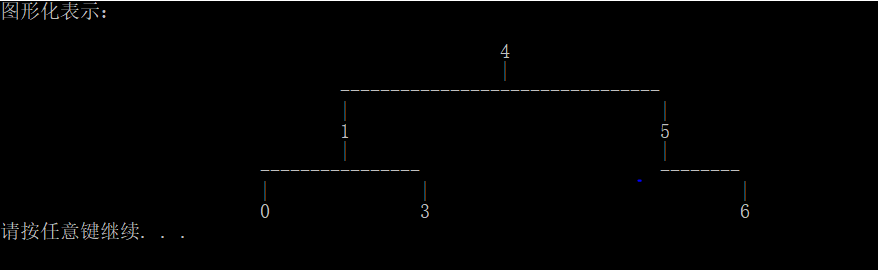


图4-10 图形化输出3

结点3应插入到结点4的左子树，结点5失衡，进行双旋操作，先对结点1进行左旋操作，再对结点5进行右旋操作，结果正确。

1. 插入7，再次单左旋。如图4-11、4-12。

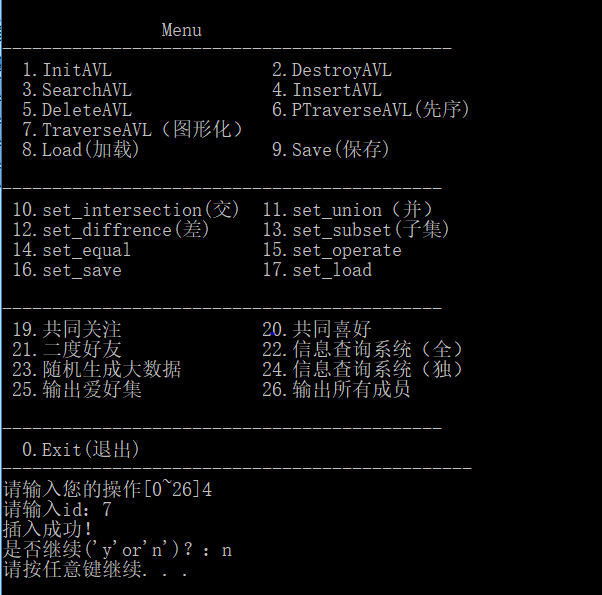


图4-11 插入函数4

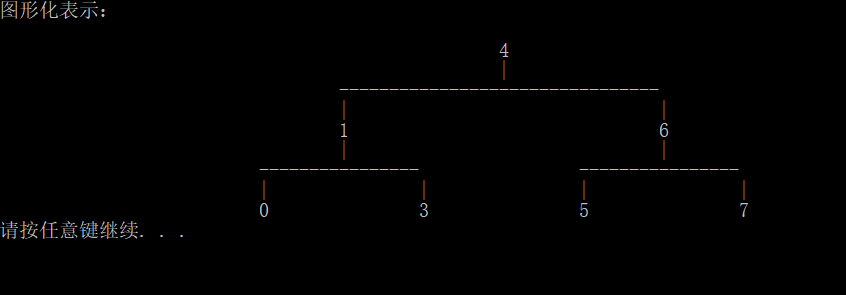


图4-12 图形化输出4

插入7后，结点5失衡，结点5左旋，结果正确。

1. 插入9、10，再次左旋，用于删除测试。如图4-13、4-14。

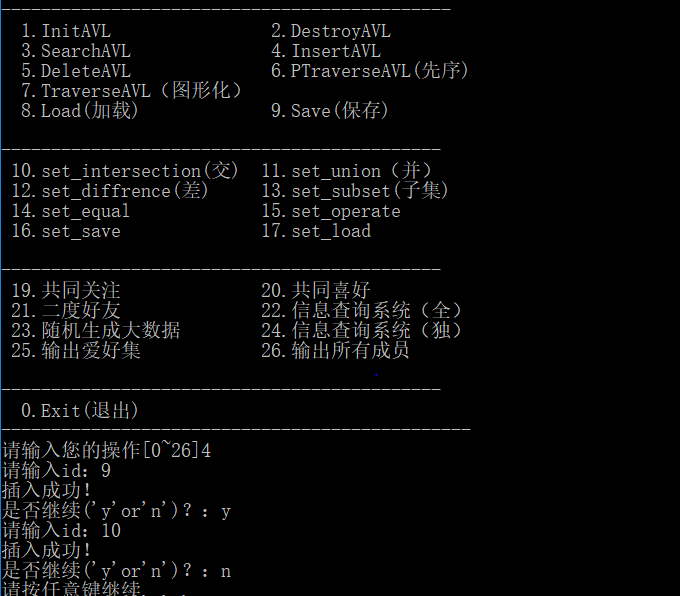


图4-13 插入函数5

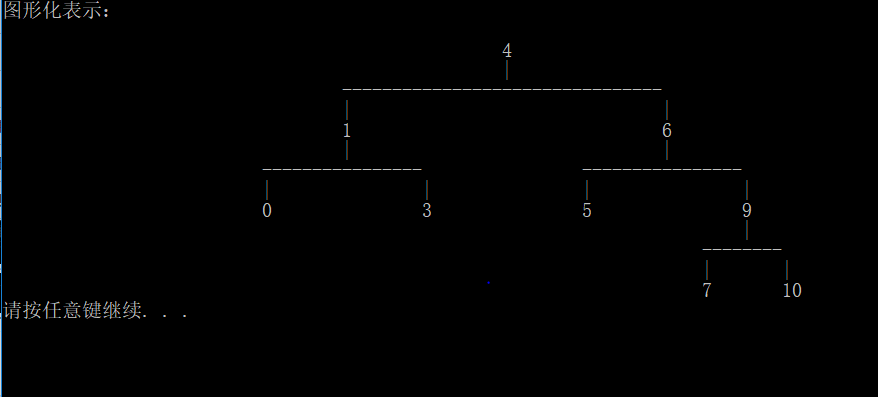


图4-14 图形化输出5

结点9、10插入到结点7的右子树，插入后结点7左旋。结果正确。

1. 删除函数

删除的结点分三种情况：无子女，子女有一个，子女都在，接下来每种情况都测试1遍。

1. 删除结点5，测试无子女情况与旋转情况，如图4-15、4-16。

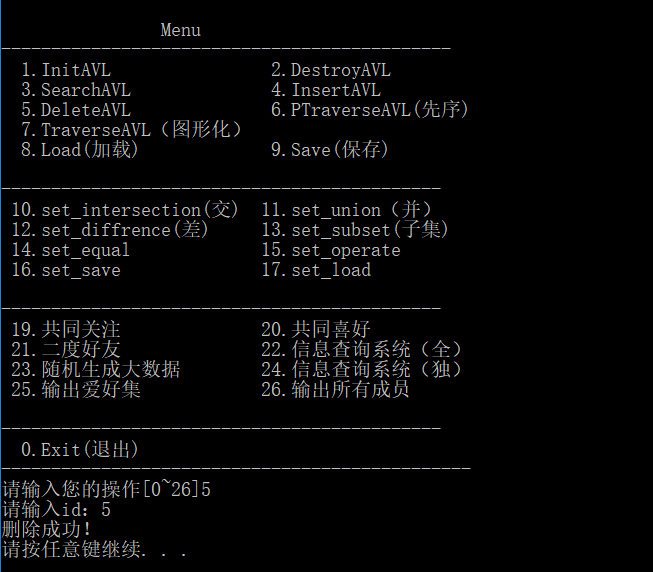


图4-15 删除函数1

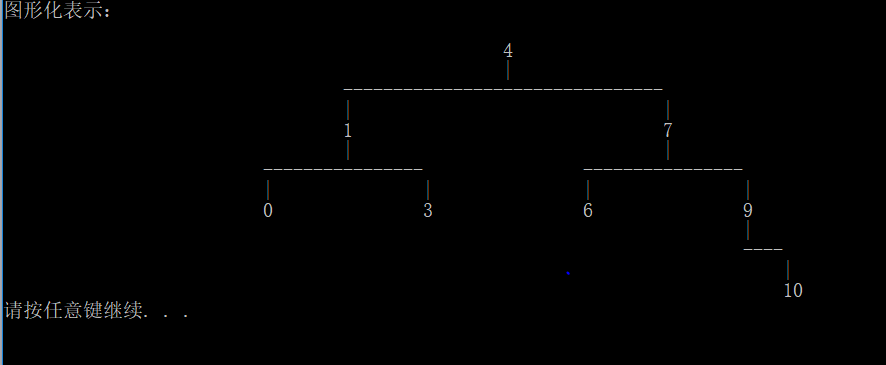


图4-16 图形化输出6

结点5无子女，直接置空，删除结点5后结点6失衡，进行右平衡双旋处理。

结果正确。

1. 删除结点9，测试有一个子女的情况。如图4-17、4-18。

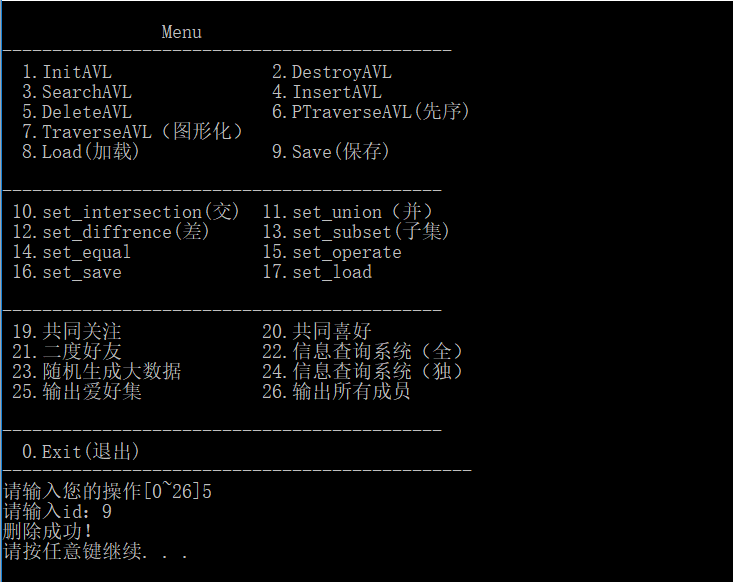


图4-17 删除函数2

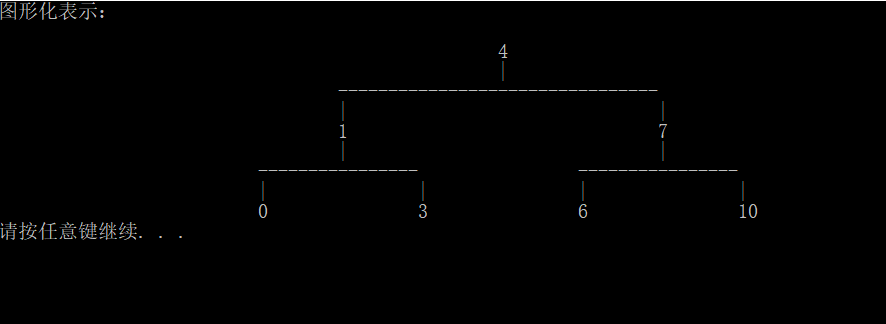


图4-18 图形化输出7

结点9存在右孩子，删除后直接将右孩子赋给结点9，结果正确。

1. 删除结点7，测试左右孩子都存在的情况。如图4-19、4-20。

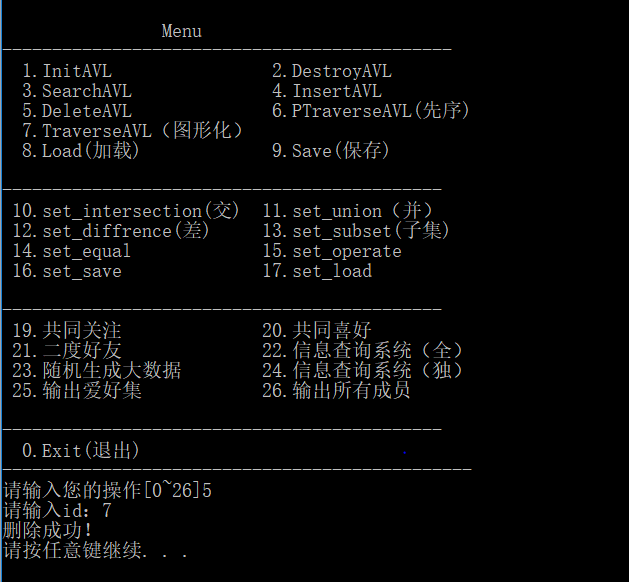


图4-19 删除函数3

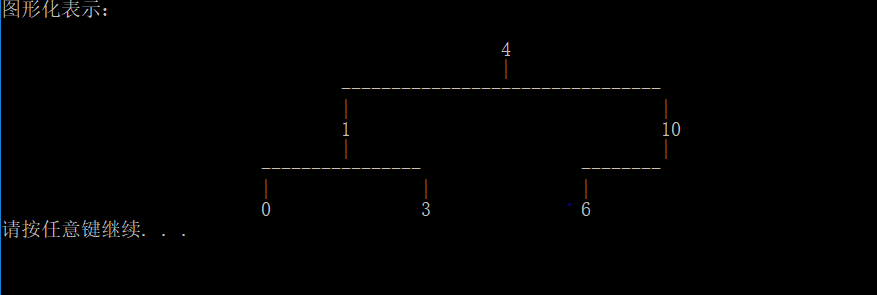


图4-20 图形化输出8

结点7左右子树都为1，用右子树的最左孩子取代结点7，然后在右子树中删除最左孩子，结果正确。

1. 查找函数测试，如图4-21、4-22。

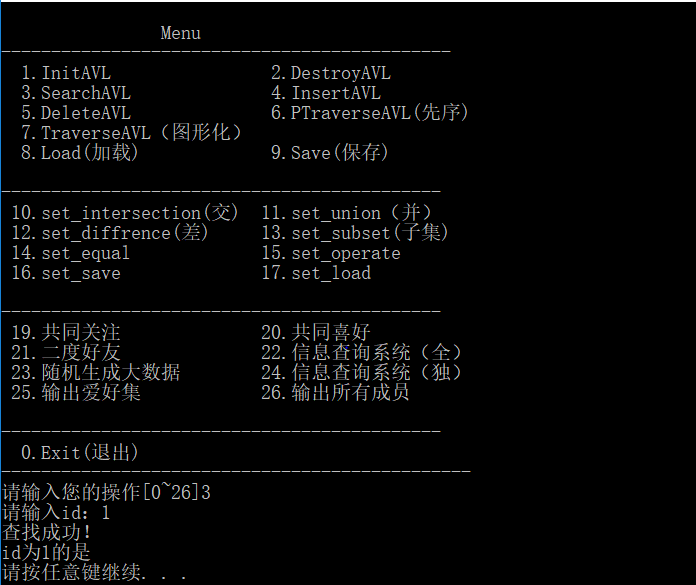


图4-21 查找成功截图

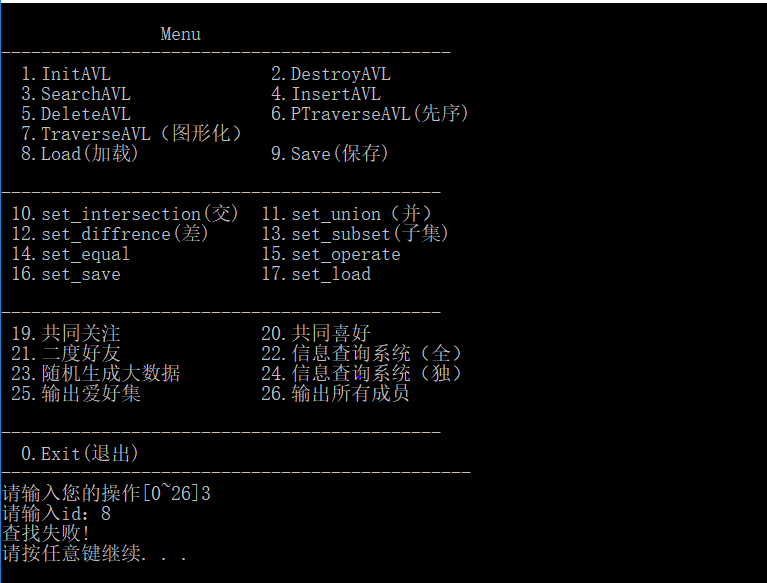


图4-22 查找越界截图

1. 先序输出函数测试，如图4-23。

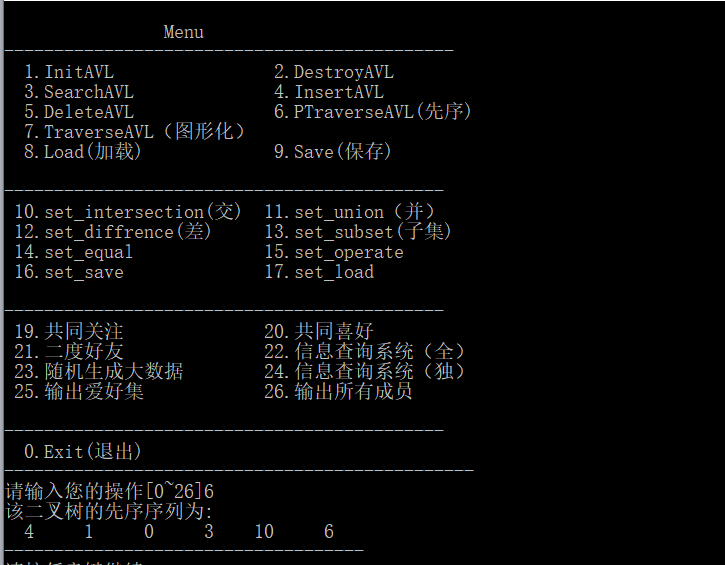


图4-23 先序输出截图

1. 保存函数测试，如图4-24、4-25。

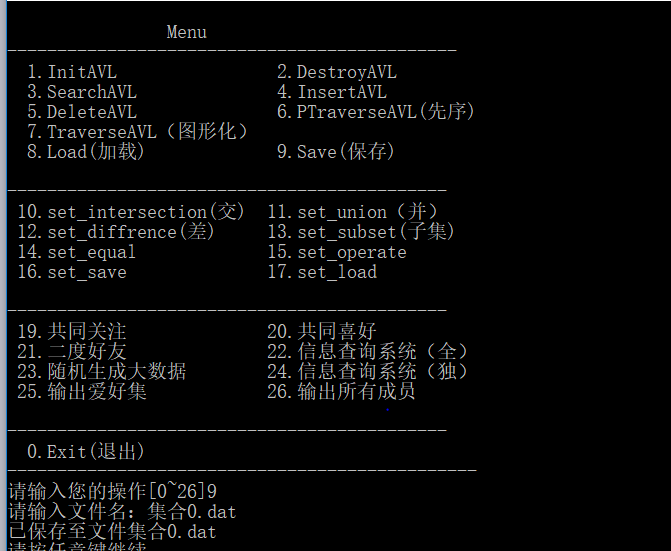


图4-24 保存界面



图4-25 保存文件截图

1. 摧毁函数测试，如图4-26、4-27。

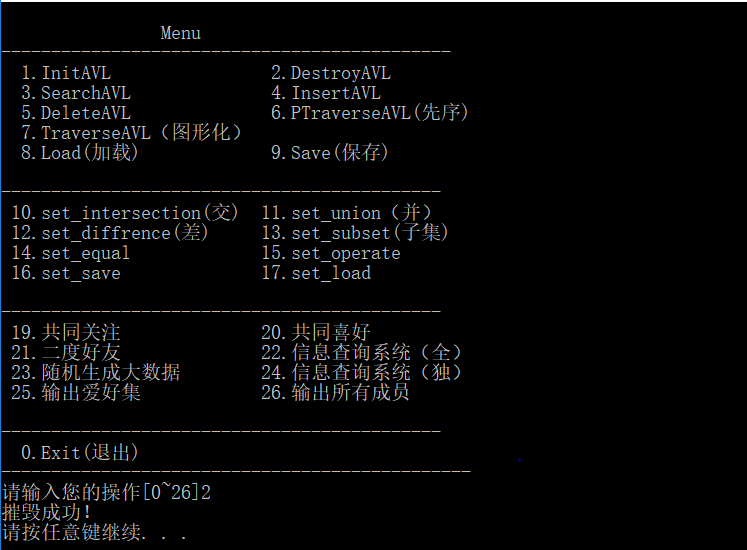


图4-26 摧毁函数

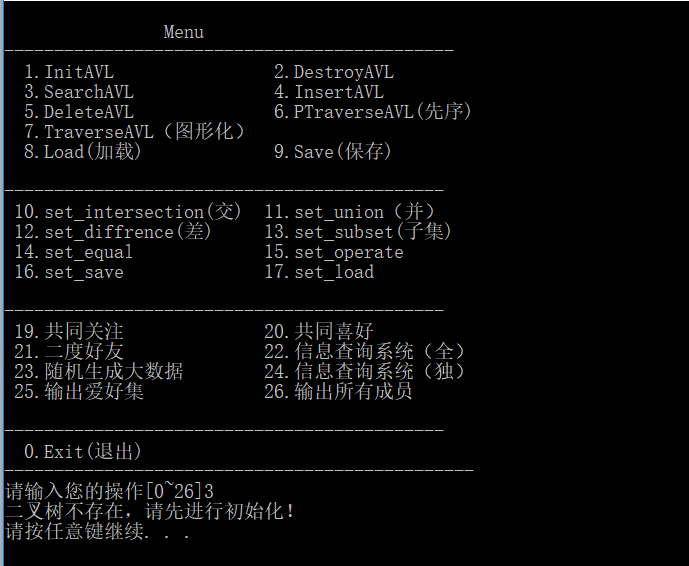


图4-27 摧毁结果测试

9.加载函数测试，如图4-28、4-29。

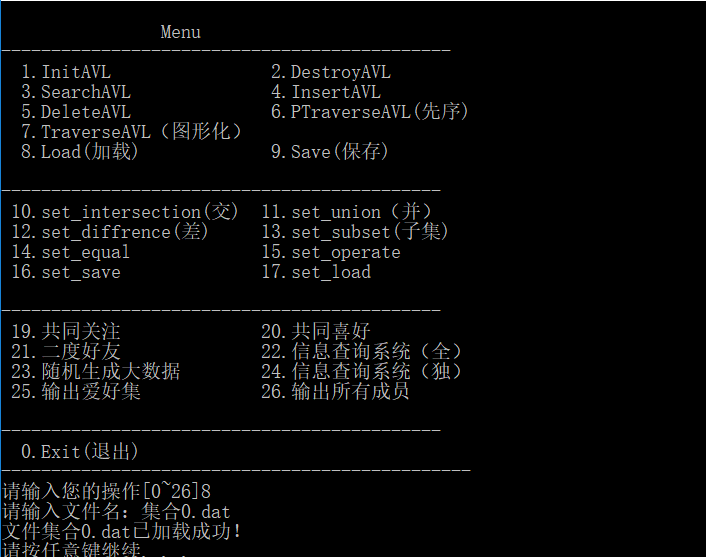


图4-28 加载界面

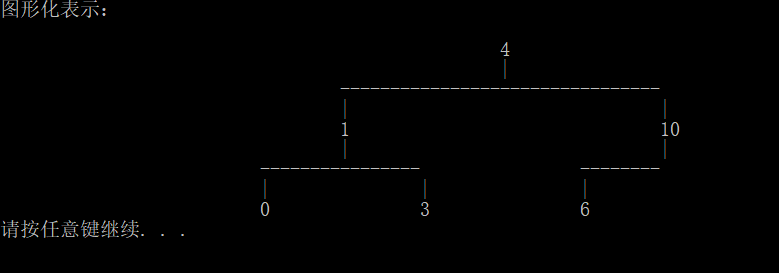


图4-29 加载验证

**集合：**

1. 进入集合操作，如图4-30

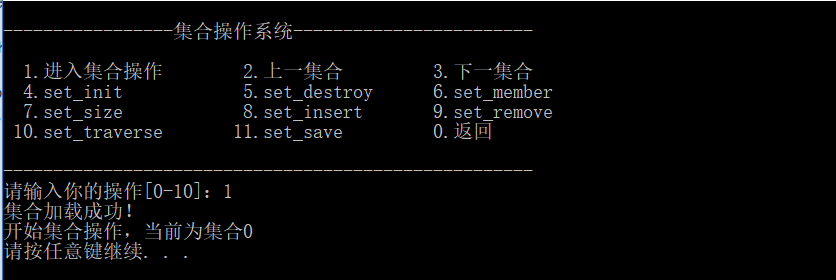


图4-30 进入集合操作

1. 依次遍历当前集合文件中所有集合，如图4-31、4-32、4-33。

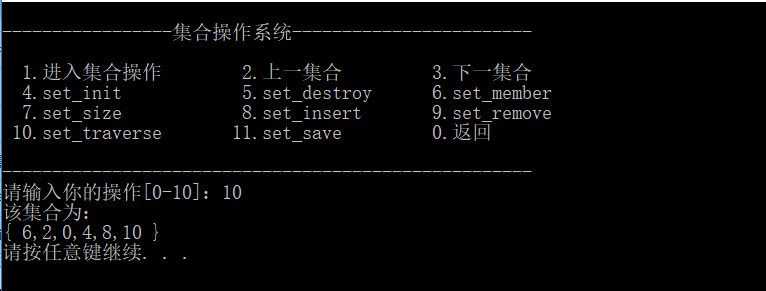


图4-31 遍历集合0

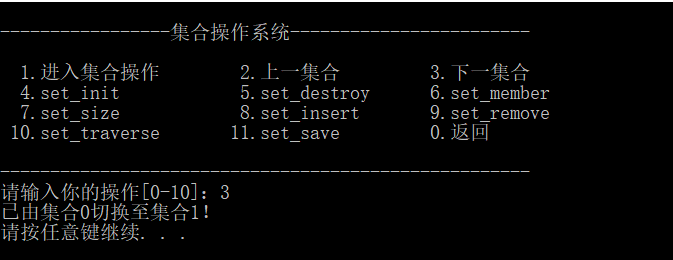


图4-32 切换至集合1

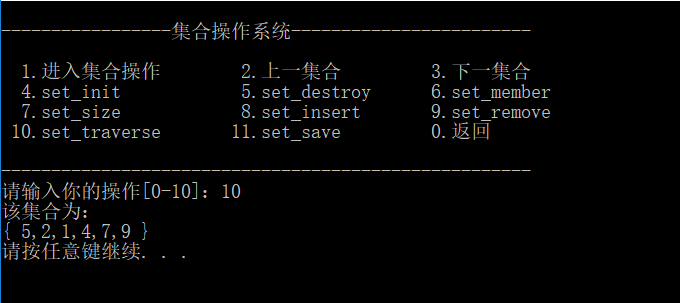


图4-33 遍历集合1

1. 加载集合T1、T2，如图4-34。

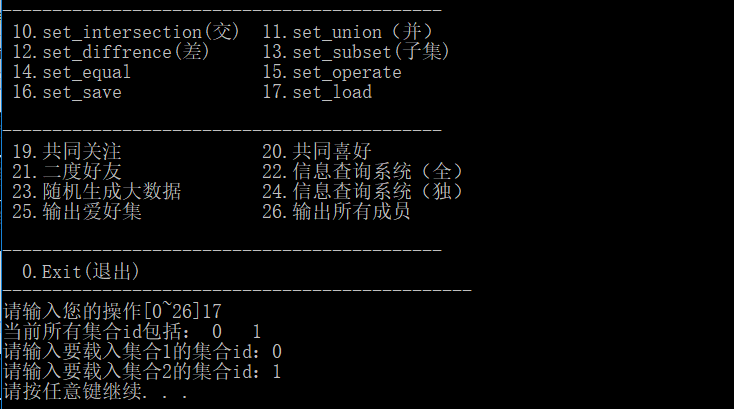


图4-34 加载集合

1. 交、并、差集函数测试，如图4-35、4-36、4-37。

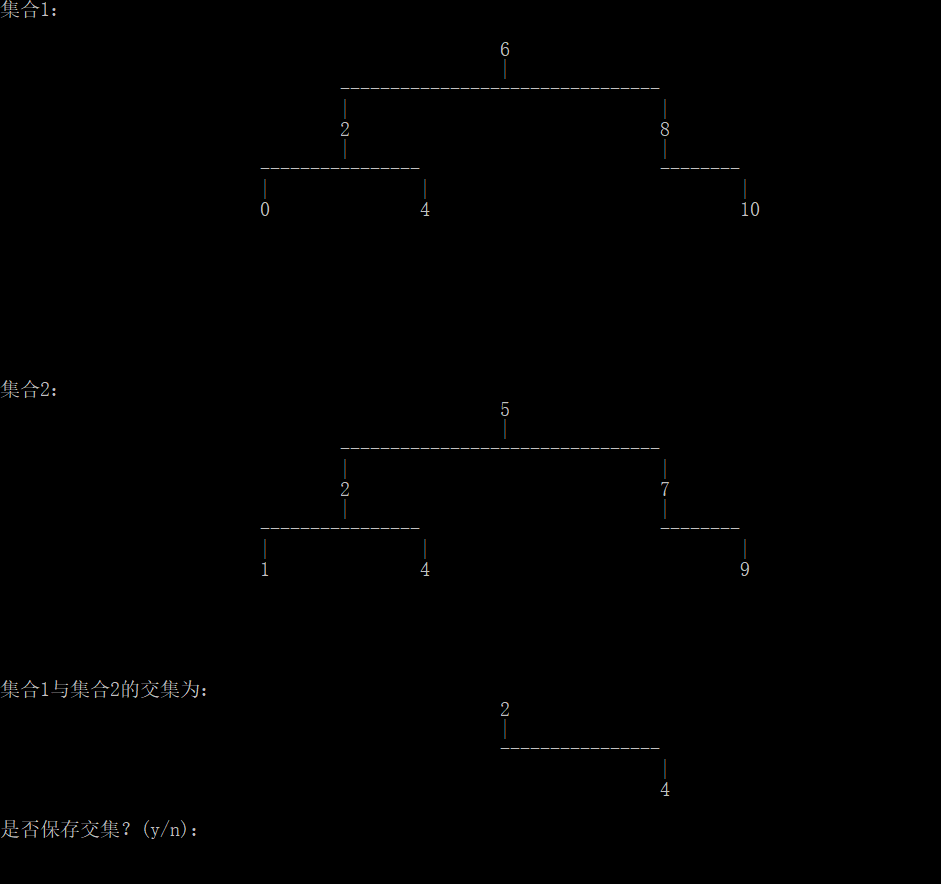


图4-35 交集函数

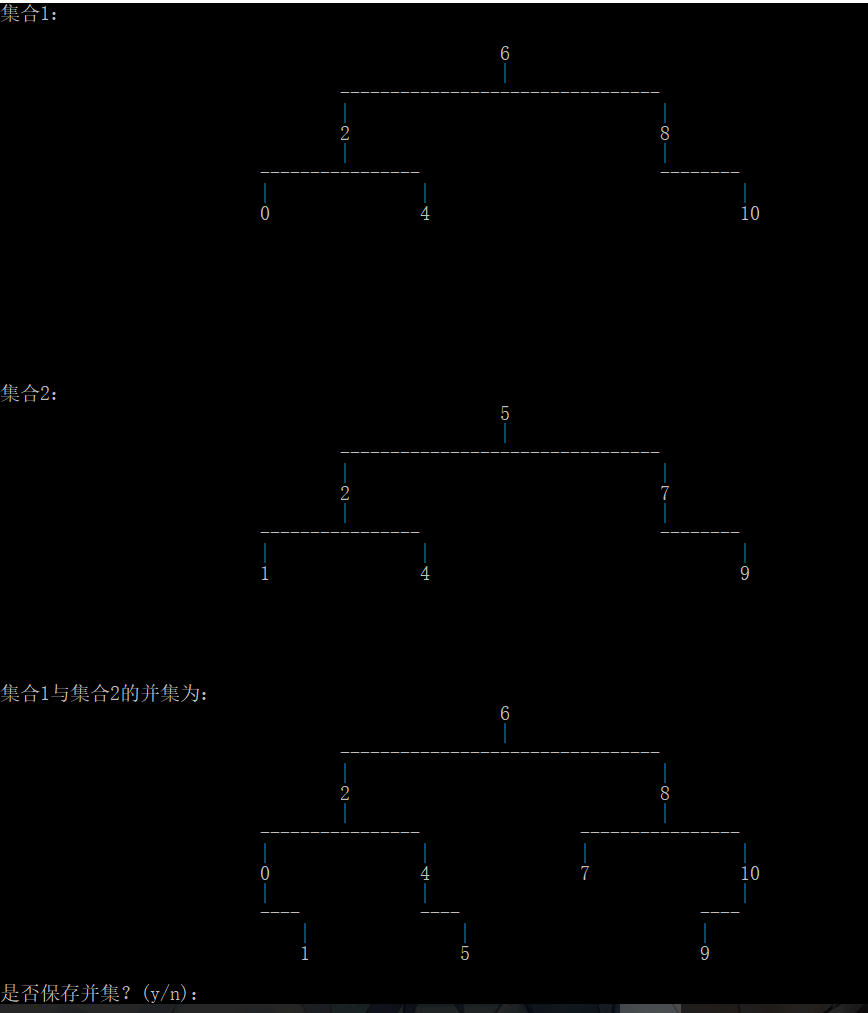


图4-36 并集函数

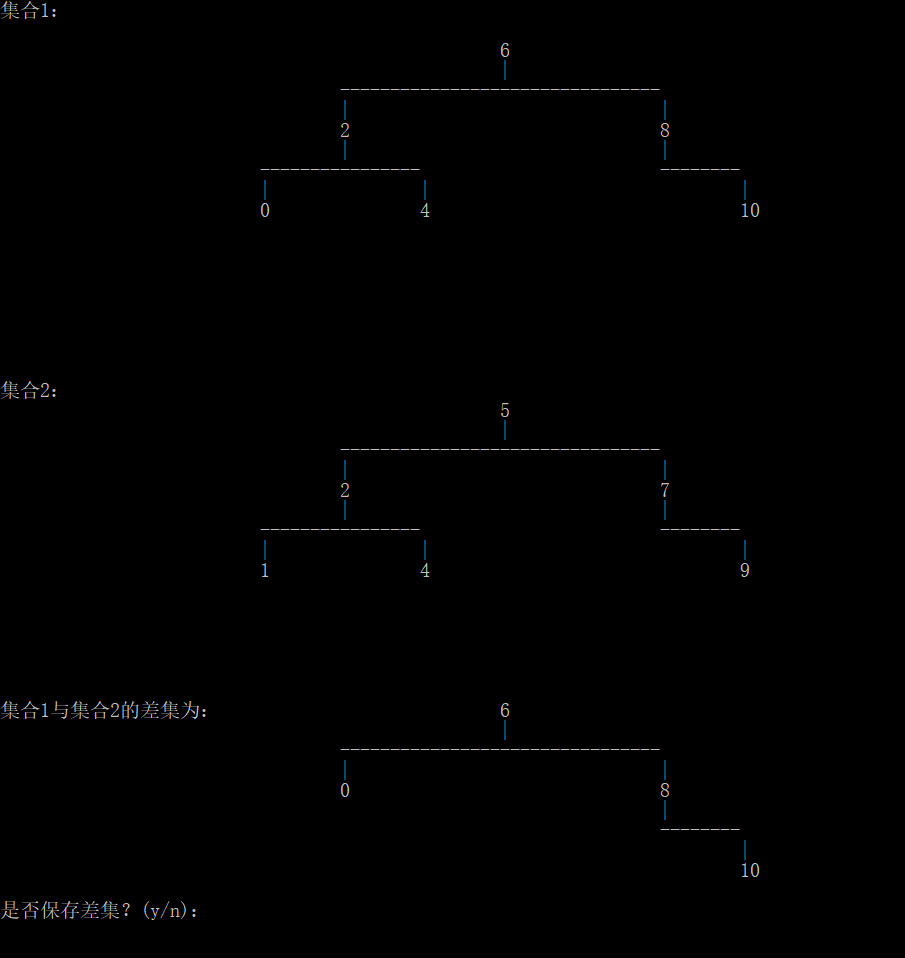


图4-37 差集函数

1. 将差集保存入链表，并遍历验证，如图4-38、4-39、4-40。

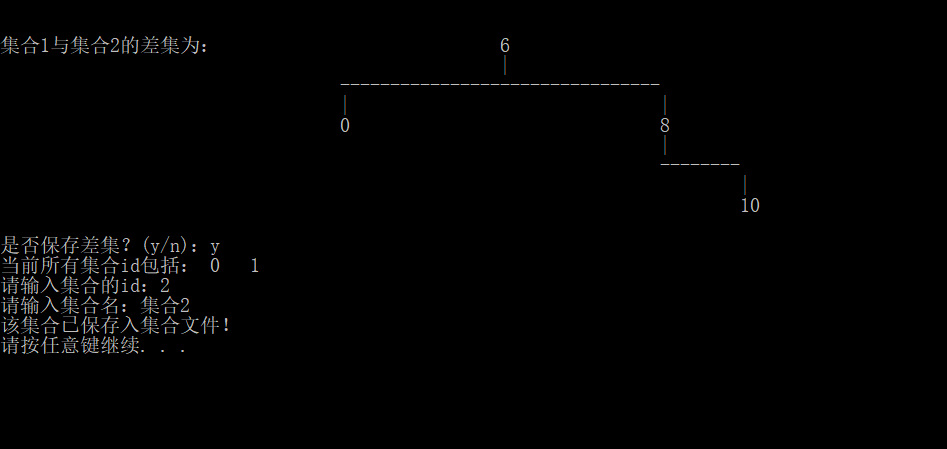


图4-38 保存差集

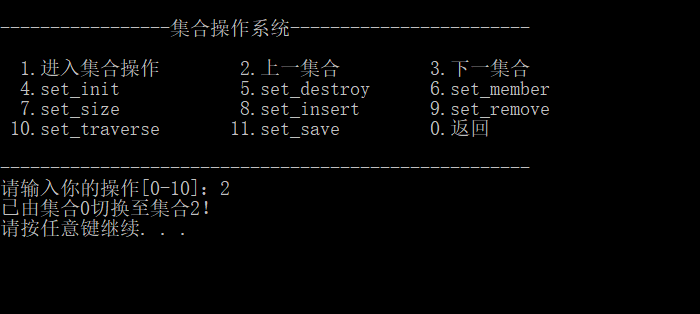


图4-39 切换至差集

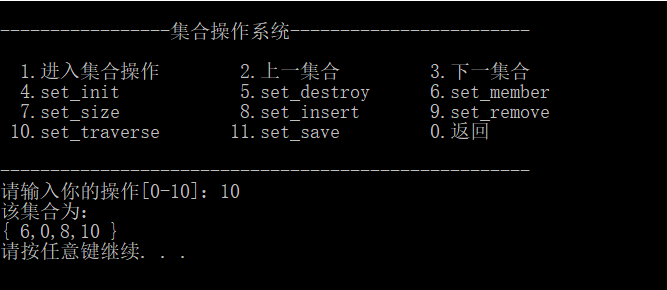


图4-40 遍历差集

集合2遍历出的结果与差集相同，结果正确。

1. 判断子集、集合相等函数测试，如图4-41、4-42。

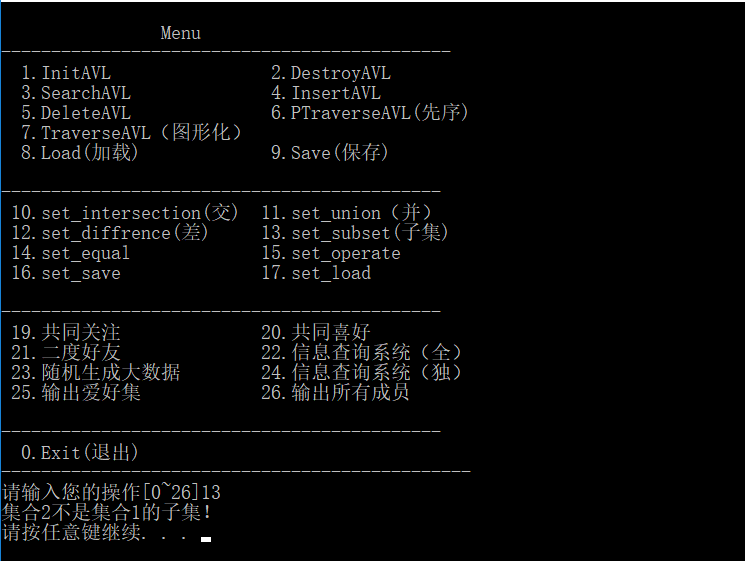


图4-41 判断子集函数1

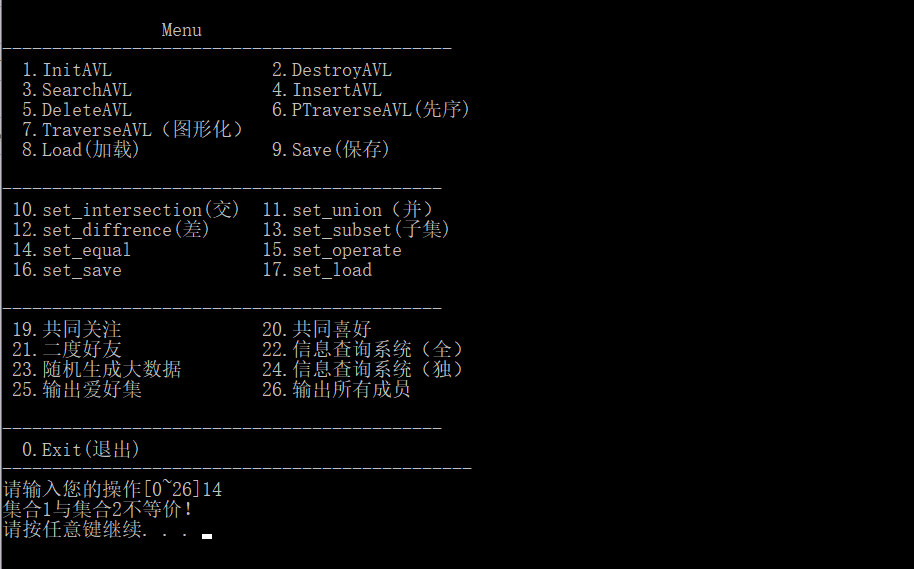


图4-42 判断集合相等函数1

显然T1与T2不是子集关系，故判断正确。

1. T1与T2加载相同集合，再次验证判断子集、集合相等函数，如图4-43、4-44。

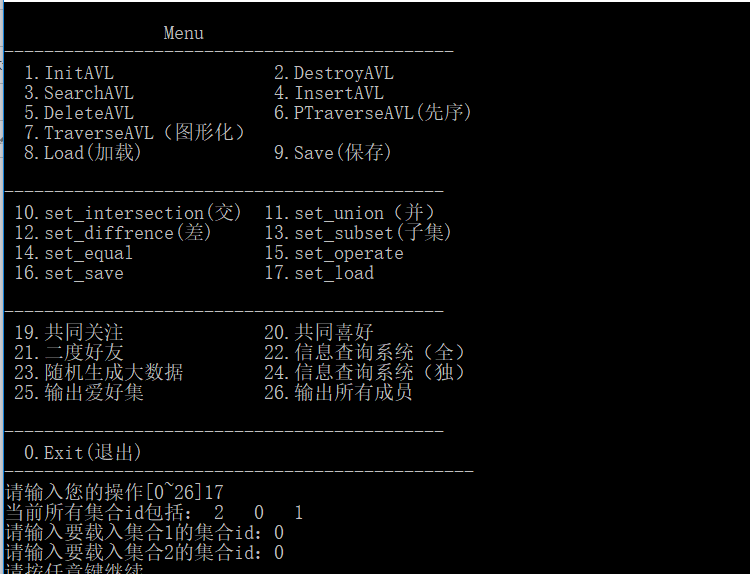


图4-43 重新加载

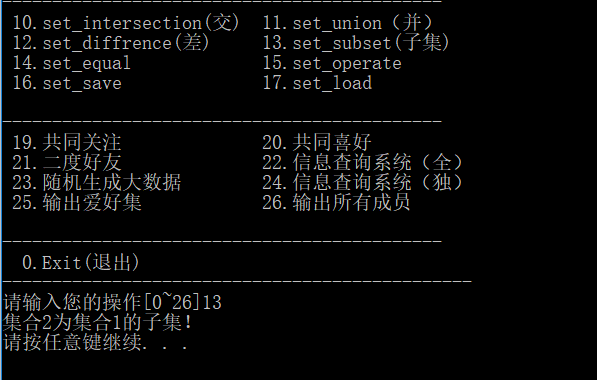


图4-44 判断子集函数2

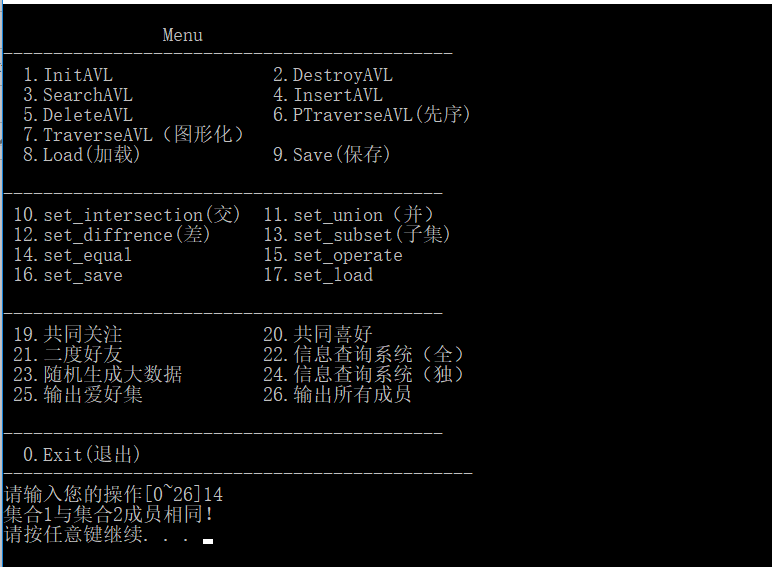


图4-45 判断集合相等函数2

T1与T2相同，故两功能判断结果都应该为“是”，结果正确。

**应用层：**

1. 随机生成数据，如图4-46。

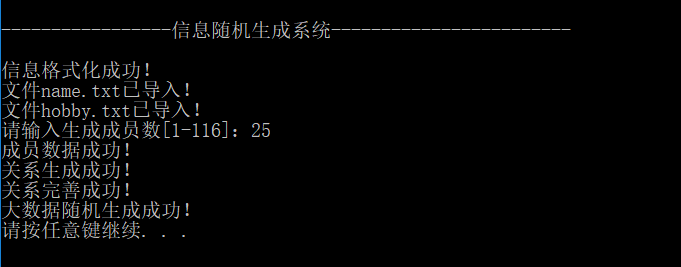


图4-46 随机生成数据

1. 遍历爱好集、姓名集，如图4-47、4-48。

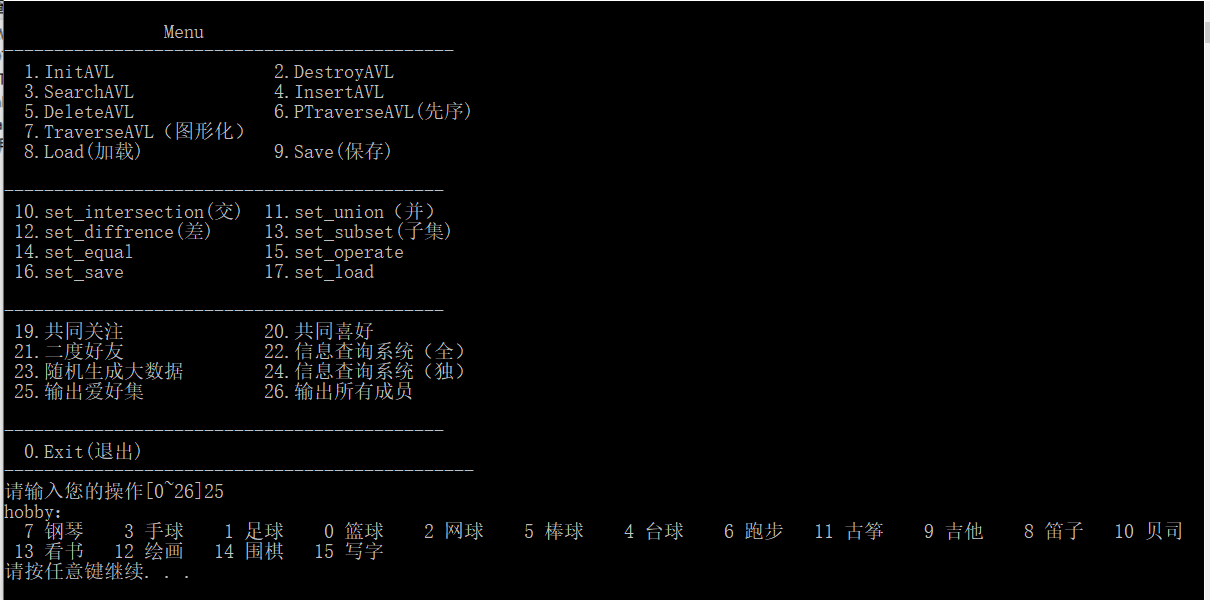


图4-47 遍历爱好集

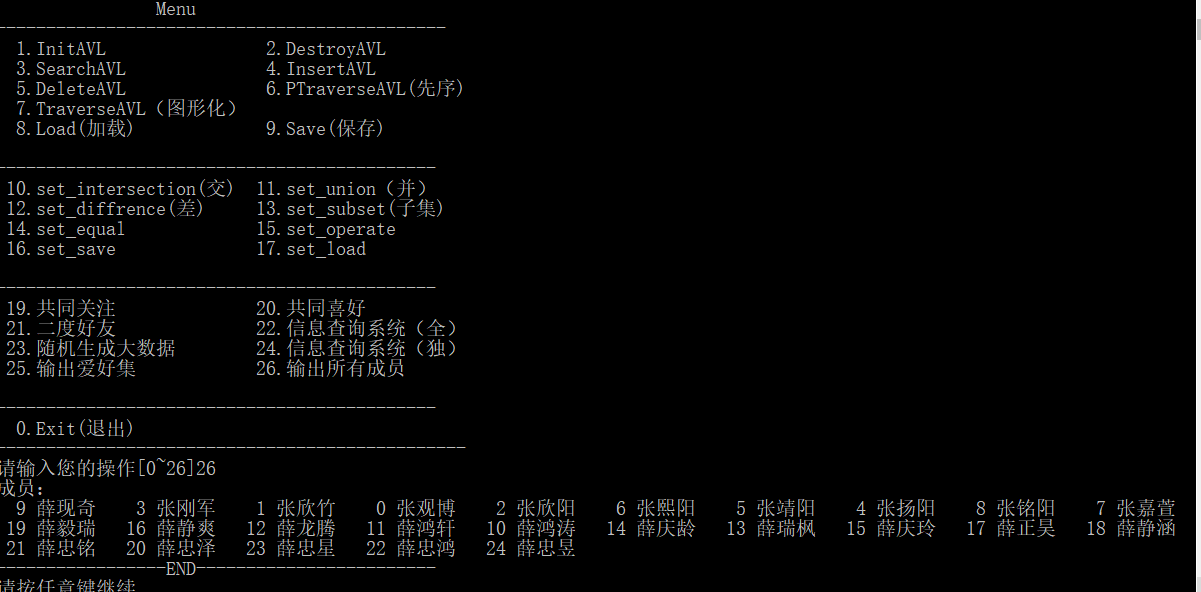


图4-48 遍历所有人

1. 测试关系集，由于粉丝集、朋友集、关注集操作类似，故只测试朋友集。

如图4-49、4-50。

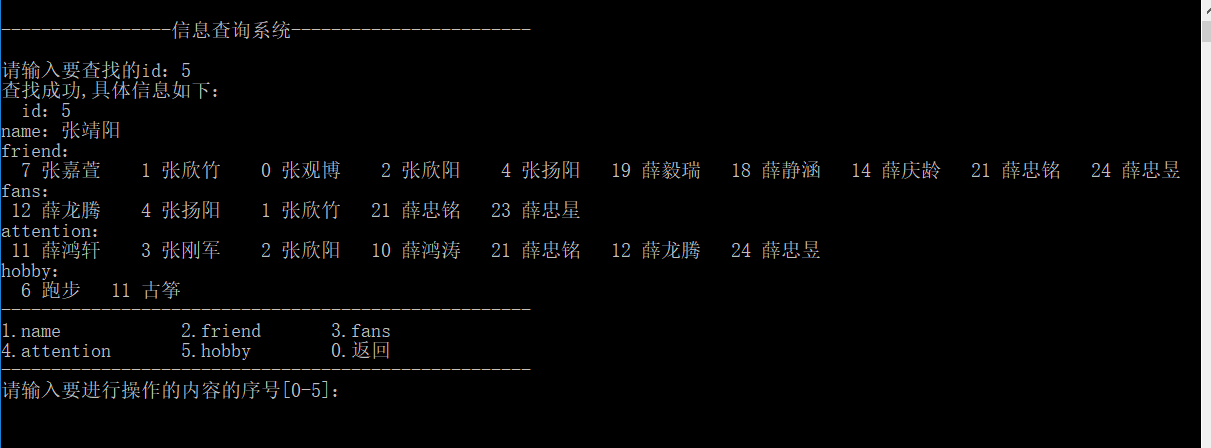


图4-49 查询系统1

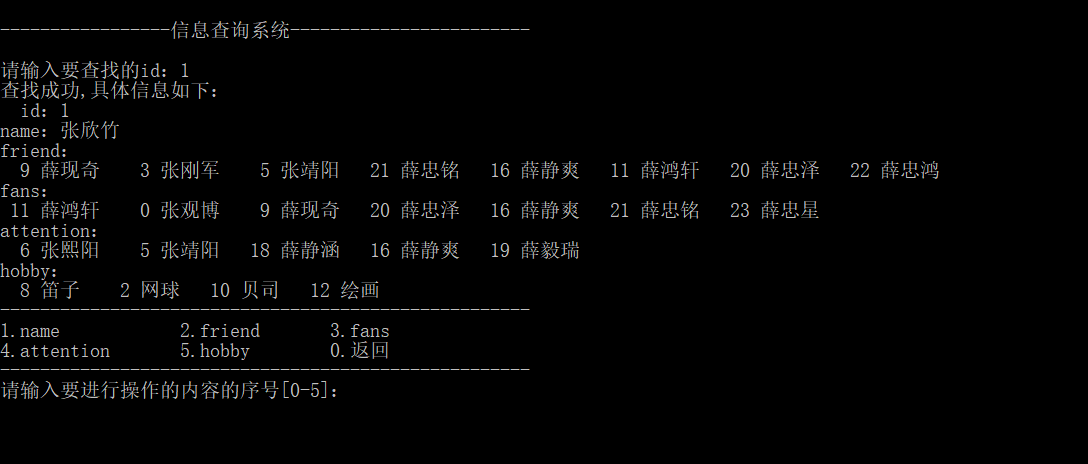


图4-50 查询系统2

由图4-49、4-50可知，张靖阳与张欣竹互为好友，成员信息完整输出并且朋友集完善，符合实验要求。

1. 测试好友删除功能，如图4-51、4-52、4-53。

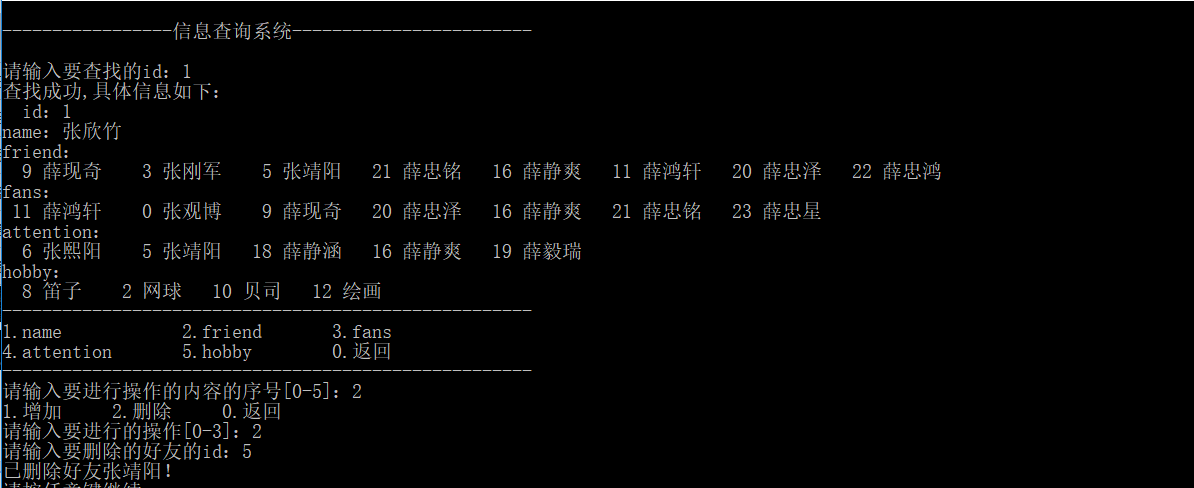


图4-51 删除好友

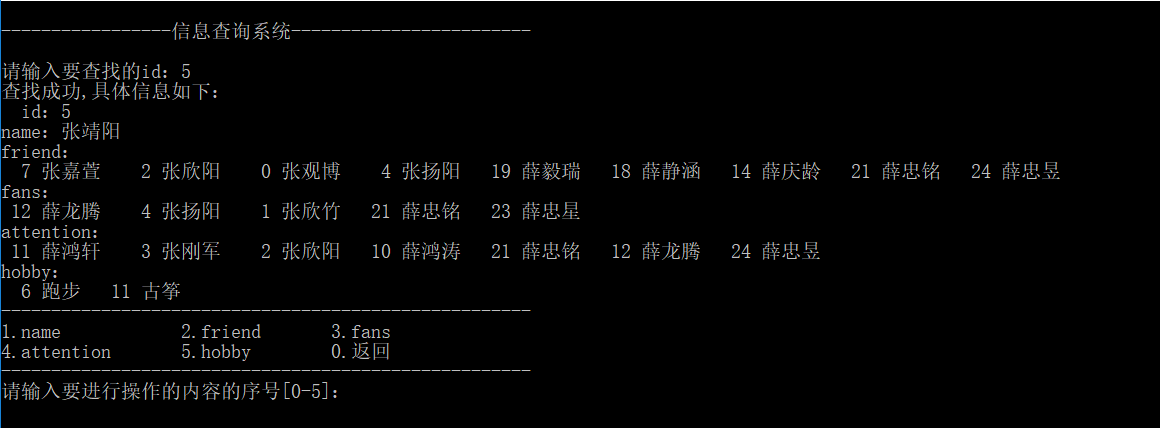


图4-52 删除验证1

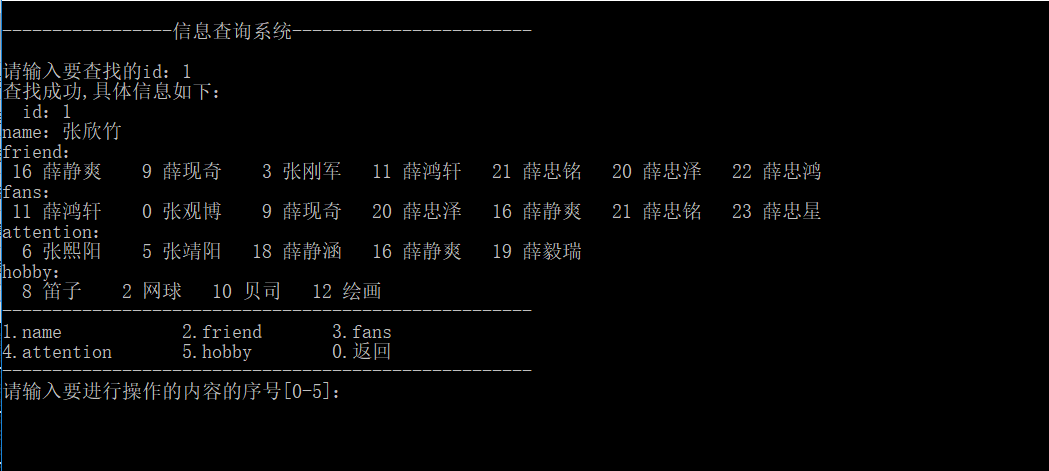


图4-52 删除验证2

由于互为好友，当一方删除时另一方也应该删除，根据图4-51、4-52，删除后，张靖阳与张欣竹的好友列表中无对方，删除功能实现。

1. 求取共同关注功能，如图4-53。

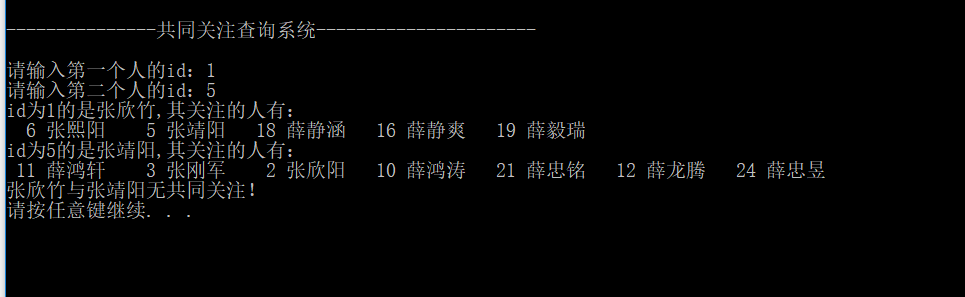


图4-53 共同关注

1. 求取共同喜好功能，如图4-54。

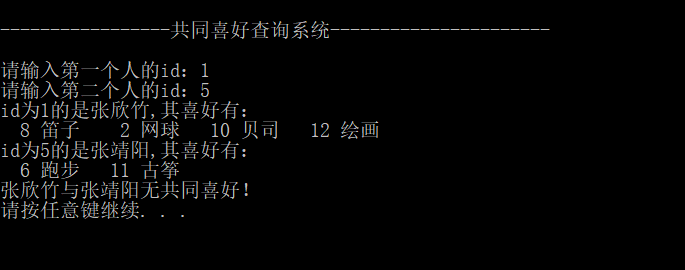


图4-54 共同喜好

1. 求取二度好友功能，如图4-55。

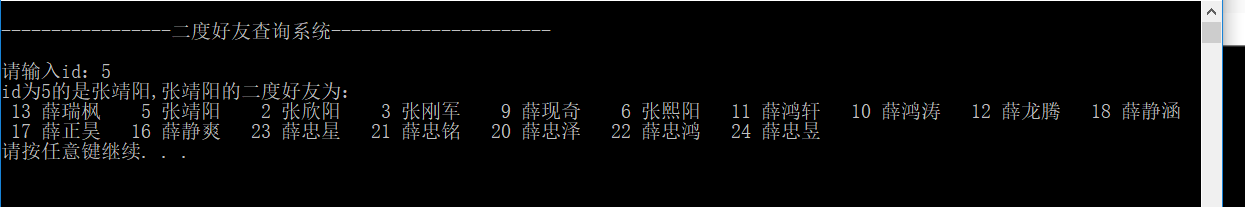


图4-55 二度好友

# 5总结与展望

5.1全文总结  
对自己的工作做个总结，主要工作如下：  
（1）对新数据结构类型定义进行了了解  
（2）熟悉了AVL的各种操作，尤其是较为复杂的插入删除函数等  
（3）学会了用AVL树的方式，表示集合，放大来看就是知道了以后可以用些巧妙的转换来完成一些我们需要完成的工作  
（4）学习了在网络上学习一些先进的算法，同时也综合自己的看法进行改进。此外，还学会了一些复杂情况下的调试情况  
（5）就课程设计而言，对AVL树的结构设计了许多函数，对其进行操作，模拟了微博上好友等关系的应用层关系，并且可以对之进行简单处理。  
  
5.2工作展望  
在今后的研究中，围绕着如下几个方面开展工作:  
（1）学习和研究更多的优秀的便捷的算法，充实自己的记忆，弥补经验的不足。  
（2）学习更多便捷和易于使用的数据结构，了解一些其他人的处理方法，综合自己的思考，得到自己容易掌握和使用的方法。  
（3）学习设计不同风格的界面，增强程序和使用者基于界面的交互感，面对对象服务是宗旨，所以应当使我们的设计在不伤害功能的前提下简化与人性化。  
（4）学习模块化编程，将程序分模块整理好，加上基本的注释，使之易于阅读，在和别人交流的时候会方便很多，在复杂情况下的debug和修改也可以避免很多麻烦的情况。

# 6体会与改进

本次课设是对上学期所学知识的一个简单总结，包含了数据结构课程的思想，虽然仍然是用C语言写的，但与C课设相比却有了显著的提升。通过数据结构这门课我们不仅学习了对数据的基本处理方式，而且规范了代码，使代码具有条理性，使代码更加易读。与C课设相比我的代码质量明显提升了一大截。

为了完成课设，我从网上借鉴了不少别人写的代码，学习了不少知识，比如为了图形化输出二叉树，我第一次了解到了<windows.h>库中的移动光标函数，除此之外，我也深刻体会到了二叉树的递归过程中，遇到空结点返回1与返回0的不同，对于不同函数，要认真考虑返回1还是0更好。

在检查了课设之后，通过老师的指导，我对自己的代码再次改进。具体改进内容如下：

1. 对于集合，增加了集合形式输出，而不是如二叉树般输出。二叉树利用图形化输出更形象，而集合运用集合形式反而更形象；
2. 完善了多集合操作，对于多集合操作，由于链表出了问题而未实现，之后解决了该问题；
3. 完善了应用层成员关系，考虑到了各关系集之间联系，使朋友的朋友集、关注集的粉丝集、粉丝集的关注集中有自己，同时防止了自己的关系集中有自己的情况的发生。
4. 增加了输出爱好集功能，用于用户查看。

# 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

[6] <http://www.taodocs.com/p-48256949.html>

[7] <http://blog.csdn.net/u010672692/article/details/45749071>

# 附录

**主函数部分：**

#include "Link.h"

void Menu();

void choice(int op,int &a,int &b);

BSTree T=NULL,T1=NULL,T2=NULL,T3=NULL,T4=NULL,T5=NULL;

//T用于二叉树操作，T1,T2用于集合交、并、补，T3用于多集合操作

//T4用于应用层操作

BSTree H=NULL,N=NULL; // H为爱好集，N为姓名集

LinkList L=NULL,buff=NULL; //用于多集合操作的链表

/\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/

int main() {

int op=1;

int b=0; //喜好集数量

int a=0; //姓名集数量

while(op){

Menu();

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

if(op==0) break;

choice(op,a,b);

}

system("cls");

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

printf("欢迎下次再使用本系统！\n");

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

}

void Menu(){

system("cls");

printf("\n Menu \n");

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

printf(" 1.InitAVL 2.DestroyAVL\n");

printf(" 3.SearchAVL 4.InsertAVL\n");

printf(" 5.DeleteAVL 6.PTraverseAVL(先序)\n");

printf(" 7.TraverseAVL（图形化）\n");

printf(" 8.Load(加载) 9.Save(保存)\n");

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

printf(" 10.set\_intersection(交) 11.set\_union（并）\n");

printf(" 12.set\_diffrence(差) 13.set\_subset(子集)\n");

printf(" 14.set\_equal 15.set\_operate\n");

printf(" 16.set\_save 17.set\_load\n");

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

printf(" 19.共同关注 20.共同喜好\n");

printf(" 21.二度好友 22.信息查询系统（全）\n");

printf(" 23.随机生成大数据 24.信息查询系统（独）\n");

printf(" 25.输出爱好集 26.输出所有成员\n");

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

printf(" 0.Exit(退出)\n");

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

printf("请输入您的操作[0~15]");

}

void choice(int op,int &a,int &b){

//实现对各函数的准备工作，输入各个参数，完善输入输出

int id=0;

BSTree Ta=NULL; //辅助二叉树

ElemType e; InitElem(e);

char c='y';

char filename[20];

FILE \*fp; //指向文件的指针，用于保存与加载

LinkList L1=NULL;

switch(op){

case 1:

if(T) printf("该树已存在，请先进行摧毁！\n");

else{

if(InitAVL(T)) printf("初始化成功！\n");

else printf("初始化失败！\n");

}

system("pause"); break;

case 2:

if(!T) printf("二叉树不存在，摧毁失败！\n");

else{

if(DestroyAVL(T)) printf("摧毁成功！\n");

else printf("摧毁失败！\n");

}

system("pause"); break;

case 3:

if(!T) printf("二叉树不存在，请先进行初始化！\n");

else if(!T->lchild) printf("该二叉树为空，请先插入数据！\n");

else{

printf("请输入id：");

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

if(SearchAVL(T->lchild,id,Ta)){

printf("查找成功！\n");

printf("id为%d的是%s\n",id,Ta->data.name);

}

else printf("查找失败!\n");

}

system("pause"); break;

case 4:printf("");

if(!T) printf("二叉树不存在，请先进行初始化！\n");

else{

while(c=='y'){

printf("请输入id："); scanf("%d",&e.id); getchar();

printf("请输入姓名："); scanf("%s",&e.name); getchar();

if(InsertAVL(T->lchild,e)){

printf("插入成功！\n");

height(T->lchild,1);

}

else printf("插入失败！\n");

printf("是否继续('y'or'n')？：");

scanf("%c",&c); getchar();

if(c!='y'&&c!='n'){

printf("请重新输入：");

scanf("%c",&c); getchar();

}

}

}

system("pause"); break;

case 5:

if(!T) printf("二叉树不存在，请先进行初始化！\n");

else if(!T->lchild) printf("该二叉树为空，请先插入数据！\n");

else{

printf("请输入id："); scanf("%d",&id);

if(DeleteAVL(T->lchild,id)) printf("删除成功！\n");

else printf("删除失败！");

}

system("pause"); break;

case 6:

if(!T) printf("二叉树不存在，请先进行初始化！\n");

else if(!T->lchild) printf("该二叉树为空，请先插入数据！\n");

else{

printf("该二叉树的先序序列为: \n");

PTraverseAVL(T->lchild);

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

}

system("pause"); break;

case 7:

if(!T) printf("二叉树不存在，请先进行初始化！\n");

else if(!T->lchild) printf("该二叉树为空，请先插入数据！\n");

else{

system("cls");

printf("图形化表示：\n");

TraverseAVL(50,2,T->lchild);

}

system("pause"); break;

case 8:

//先将T置为空树

if(T==NULL) InitAVL(T);

else { DestroyAVL(T); InitAVL(T);}

//进行加载

printf("请输入文件名：");

scanf("%s",filename);

if(Load(T->lchild,filename)) printf("文件%s已加载成功！\n",filename);

else{

printf("文件加载失败！\n");

DestroyAVL(T);

}

system("pause"); break;

case 9:

if(!T) printf("二叉树不存在，保存失败！\n");

else{

printf("请输入文件名：");

scanf("%s",filename);

if ((fp=fopen(filename,"wb"))==NULL){ //文件打开失败

printf("文件打开失败！\n ");

}

else{

if(Save(T->lchild,fp)) printf("已保存至文件%s\n",filename);

else printf("文件保存失败！\n");

}

}

fclose(fp);

system("pause"); break;

case 10:

if(!T1||!T2){

if(!T1) printf("集合1不存在，请先进行加载！\n");

if(!T2) printf("集合2不存在，请先进行加载！\n");

}

else if(!T1->lchild||!T2->lchild){

if(!T1->lchild) printf("集合1为空集，交集为空集！\n");

if(!T2->lchild) printf("集合2为空集，交集为空集！\n");

}

else{

system("cls");

set\_destroy(T3); set\_init(T3);

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

printf("集合1：\n");

TraverseAVL(50,2,T1->lchild);

printf("\n集合2：");

TraverseAVL(50,20,T2->lchild);

if(!T3->lchild){

printf("\n集合1与集合2的交集为空集！");

}

else{

printf("\n集合1与集合2的交集为：\n");

TraverseAVL(50,35,T3->lchild);

printf("\n");

}

c='0';

while(c!='y'||c!='n'){

printf("是否保存交集？(y/n)：");

scanf("%c",c);

}

if(c=='y') tree\_to\_link(L,T3);

}

else printf("求取交集出错！\n");

}

system("pause"); break;

case 11:

if(!T1||!T2){

if(!T1) printf("集合1不存在，请先进行加载！\n");

if(!T2) printf("集合2不存在，请先进行加载！\n");

}

else{

set\_destroy(T3); set\_init(T3);

if(!set\_union(T3->lchild,T1->lchild)) printf("求取并集出错!\n"); //分别将T1，T2与Ta合并

else if(!set\_union(T3->lchild,T2->lchild)) printf("求取并集出错！\n");

else{

system("cls");

printf("集合1：\n");

TraverseAVL(50,2,T1->lchild);

printf("\n集合2：");

TraverseAVL(50,20,T2->lchild);

if(!T3->lchild){

printf("\n集合1与集合2的并集为空集！\n");

}

else{

printf("\n集合1与集合2的并集为：\n");

TraverseAVL(50,35,T3->lchild);

printf("\n");

}

c='0';

while(c!='y'||c!='n'){

printf("是否保存并集？(y/n)：");

scanf("%c",c);

}

if(c=='y') tree\_to\_link(L,T3);

}

}

system("pause"); break;

case 12:

set\_destroy(T3); set\_init(T3);

if(!T1||!T2){

if(!T1) printf("集合1不存在，请先进行加载！\n");

if(!T2) printf("集合2不存在，请先进行加载！\n");

}

else if(!T1->lchild) printf("集合1为空集，差集为空集！\n");

else if(!set\_union(T3->lchild,T1->lchild)) printf("求取差集出错！\n");

else{

if(!set\_diffrence(T3->lchild,T2->lchild)) printf("求取差集出错！\n"); //分别将T1，T2与Ta合并

else{

system("cls");

printf("集合1：\n");

TraverseAVL(50,2,T1->lchild);

printf("\n集合2：");

TraverseAVL(50,20,T2->lchild);

if(!T3->lchild){

printf("\n集合1与集合2的差集为空集！\n");

}

else{

printf("\n集合1与集合2的差集为：\n");

TraverseAVL(50,35,T3->lchild);

printf("\n");

}

c='0';

while(c!='y'||c!='n'){

printf("是否保存差集？(y/n)：");

scanf("%c",c);

}

if(c=='y') tree\_to\_link(L,T3);

}

}

system("pause"); break;

case 13:

if(!T1||!T2){

if(!T1) printf("集合1不存在，请先进行加载！\n");

if(!T2) printf("集合2不存在，请先进行加载！\n");

}

else if(!T1->lchild&&!T2->lchild) printf("集合1与集合2都为空集，集合2为集合1的子集！\n");

else if(!T1->lchild) printf("集合1为空集，集合2不是集合1的子集！\n");

else{

if(set\_subset(T1->lchild,T2->lchild)) printf("集合2为集合1的子集！\n");

else printf("集合2不是集合1的子集！\n");

}

system("pause"); break;

case 14:

if(!T1||!T2){

if(!T1) printf("集合1不存在，请先进行加载！\n");

if(!T2) printf("集合2不存在，请先进行加载！\n");

}

else{

if(set\_equal(T1->lchild,T2->lchild)) printf("集合1与集合2成员相同！\n");

else printf("集合1与集合2不等价！\n");

}

system("pause"); break;

case 15:

InitAVL(T5);

set\_menu(L,T5,buff);

system("pause"); break;

case 16: //集合的保存

if(!L) printf("当前无集合！\n");

else{

if(save\_linklist(L)) printf("集合信息已保存入文件List.dat！\n");

else{

printf("集合保存失败！\n");

}

}

system("pause"); break;

case 17: //集合的加载，用于交、并、补运算

if(!L){

printf("还未进入集合操作，请先加载！\n");

}

if(T1){

set\_destroy(T1); set\_init(T1);

}

if(T2){

set\_destroy(T2); set\_init(T2);

}

printf("当前所有集合id包括："); TraverseLink(L);

printf("请输入要载入集合1的集合id：");

scanf("%d",id);

if(!search\_link(L,id,L1)){

printf("不存在该集合！\n");

set\_destroy(T1); set\_destroy(T2);

}

else{

T1->lchild=L1->data.T;

printf("请输入要载入集合2的集合id：");

scanf("%d",id);

if(!search\_link(L,id,L1)){

printf("不存在该集合！\n");

set\_destroy(T1); set\_destroy(T2);

}

else T2->lchild=L1->data.T;

}

system("pause"); break;

case 19:

if(!T4) printf("信息未导入,请先导入信息！\n");

else if(!T4->lchild) printf("当前无数据，请先添加数据！\n");

else{

system("cls");

printf("\n---------------共同关注查询系统----------------------\n\n");

Common\_attention(T4->lchild);

}

system("pause"); break;

case 20:

if(!T4) printf("信息未导入,请先导入信息！\n");

else if(!T4->lchild) printf("当前无数据，请先添加数据！\n");

else{

system("cls");

printf("\n-----------------共同喜好查询系统----------------------\n\n");

Common\_hobby(T->lchild);

}

system("pause"); break;

case 21:

if(!T4) printf("信息未导入,请先导入信息！\n");

else if(!T4->lchild) printf("当前无数据，请先添加数据！\n");

else{

system("cls");

printf("\n-----------------二度好友查询系统----------------------\n\n");

printf("请输入id：");

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

if(!second\_friend(T4->lchild,id)){

printf("二度好友求取失败！\n");

}

}

system("pause"); break;

case 22:

if(!T4) printf("信息未导入,请先导入信息！\n");

else if(!T4->lchild) printf("当前无数据，请先添加数据！\n");

else{

system("cls");

printf("\n-----------------信息查询系统------------------------\n\n");

operator\_tree(T4->lchild,H,a,b);

}

system("pause"); break;

case 23: //随机生成数据

if(!T4){

set\_destroy(T4); set\_init(T4);

}

system("cls");

printf("\n-----------------信息随机生成系统------------------------\n\n");

if(!H||!N){

set\_destroy(H); H=NULL;

set\_destroy(N); N=NULL;

printf("信息格式化成功！\n");

}

if(!name\_set\_input(N,a)){

printf("文件name.txt导入失败！\n");

}

else{

printf("文件name.txt已导入！\n");

if(!hobby\_set\_input(H,b)){

printf("文件hobby.txt导入失败！\n");

}

else{

printf("文件hobby.txt已导入！\n");

printf("请输入生成成员数[1-%d]：",a);

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

while(id<1||id>a){

printf("请重新输入：");

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

}

if(!Tree\_input(T4,N,id)){

printf("成员数据生成失败！\n");

set\_destroy(T4); set\_destroy(H); set\_destroy(N);

}

else{

printf("成员数据成功！\n");

if(!relation\_input(T4->lchild,N,H,id,b)){

printf("关系生成失败！\n");

set\_destroy(T4); set\_destroy(H); set\_destroy(N);

}

else{

printf("关系生成成功！\n");

if(!improve\_relation(T4->lchild,T4->lchild)){

printf("关系完善失败！\n");

set\_destroy(T4); set\_destroy(H); set\_destroy(N);

}

else{

printf("关系完善成功！\n");

printf("大数据随机生成成功！\n");

}

}

}

}

}

system("pause"); break;

case 24:

if(!T4) printf("信息未导入,请先导入信息！\n");

else if(!T4->lchild) printf("当前无数据，请先添加数据！\n");

else{

system("cls");

printf("\n-----------------信息查询系统------------------------\n\n");

operator\_tree2(T4->lchild,H,a,b);

}

system("pause"); break;

case 25:

if(!H) printf("爱好集还未加载！\n");

else{

printf("hobby：\n");

PTraverseAVL(H); printf("\n");

}

system("pause"); break;

case 26:

if(!T4) printf("信息未导入,请先导入信息！\n");

else if(!T4->lchild) printf("当前无数据，请先添加数据！\n");

else{

printf("成员：\n");

PTraverseAVL(T4->lchild);

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

}

system("pause"); break;

}

}

**二叉树部分：**

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#include <windows.h>

#include <math.h>

/\*---------------常量----------------\*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

const int middle=32; //用于图形化显示二叉树

//数据类型

struct BSTNode;

typedef int status;

typedef struct ElemType{

int id;

char name[10]; //姓名

//这四个二叉树集无头结点

BSTNode\* friends; //好友集

BSTNode\* fans; //粉丝集

BSTNode\* attention; //关注人集

BSTNode\* hobby; //喜好

}ElemType;

//平衡二叉树结点类型定义

typedef struct BSTNode{

ElemType data;

int h; //树的高度

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

}BSTNode,\*BSTree;

void R\_Rotate(BSTree &p){

//对以\*p为根的二叉排序树进行右旋处理，处理后p指向新的树根结点，

//即旋转之前的左子树的根结点

BSTree lc=p->lchild;

p->lchild=lc->rchild;

lc->rchild=p;

p=lc;

}

void L\_Rotate(BSTree &p){

//对以\*p为根的二叉排序树进行左旋处理，处理后p指向新的树根结点，

//即旋转之前的右子树的根结点

BSTree lc=p->rchild;

p->rchild=lc->lchild;

lc->lchild=p;

p=lc;

}

status depth(BSTree T){

//计算并返回T的深度

if(!T) return ERROR;

int a=depth(T->lchild), b=depth(T->rchild);

return (a>b)? a+1 : b+1;

}

void LeftBalance(BSTree &T){

//对以指针T所指结点为根的二叉树作左平衡旋转处理，

//本算法结束时，指针T指向新的根结点

BSTree lc=T->lchild;

//新结点插在\*T的左孩子的左子树上，要作单右旋处理

if(depth(lc->lchild)>depth(lc->rchild)) R\_Rotate(T);

//新结点插在\*T的左孩子的右子树上，要作双旋处理

else{

L\_Rotate(T->lchild);

R\_Rotate(T);

}

}

void RightBalance(BSTree &T){

//对以指针T所指结点为根的二叉树作右平衡旋转处理，

//本算法结束时，指针T指向新的根结点

BSTree rc=T->rchild;

//新结点插在\*T的右孩子的右子树上，要作单左旋处理

if(depth(rc->rchild)>depth(rc->lchild)) L\_Rotate(T);

else{

R\_Rotate(T->rchild);

L\_Rotate(T);

}

}

status InsertAVL(BSTree &T,ElemType e){

//若在平衡的二叉排序树T中不存在和e有相同关键字的结点，则插入一个数据元素

if(!T){//插入新结点

T=(BSTree)malloc(sizeof(BSTNode)); T->data=e;

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

}

else{

if(e.id==T->data.id) return ERROR; //结点T的关键字与e相同

else if(e.id<T->data.id){

if(!InsertAVL(T->lchild,e)) return ERROR; //未插入

if(depth(T->lchild)-depth(T->rchild)==2) LeftBalance(T);

}

else{

if(!InsertAVL(T->rchild,e)) return ERROR; //未插入

if(depth(T->lchild)-depth(T->rchild)==-2) RightBalance(T);

}

}

return OK;

}

status DeleteAVL(BSTree &T,int e){

//删除二叉树T中结点id为e的结点

if(!T) return ERROR; //二叉树不变

BSTree pre=NULL;

if(T->data.id==e){ //找到结点

if(T->lchild==NULL&&T->rchild==NULL){ //该结点无子节点

free(T); T=NULL;

}

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

pre=T; T=T->lchild;

free(pre); pre=NULL;

}

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

pre=T; T=T->rchild;

free(pre); pre=NULL;

}

else{

if(depth(T->lchild)>depth(T->rchild)){ //左子树比右子树高

pre=T->lchild;

while(pre->rchild) pre=pre->rchild; //寻找左子树的最右子树

T->data=pre->data;

DeleteAVL(T->lchild,pre->data.id);

}

else{ //左子树比右子树低

pre=T->rchild;

while(pre->lchild) pre=pre->lchild; //寻找右子树的最左子树

T->data=pre->data;

DeleteAVL(T->rchild,pre->data.id);

}

}

return OK;

}

else if(T->data.id>e){

if(!DeleteAVL(T->lchild,e)) return ERROR;

if(depth(T->lchild)-depth(T->rchild)==-2) RightBalance(T);

}

else{

if(!DeleteAVL(T->rchild,e)) return ERROR;

if(depth(T->lchild)-depth(T->rchild)==2) LeftBalance(T);

}

return OK;

}

void InitElem(ElemType &e){

e.id=0; e.name[0]='\0';

e.attention=NULL; e.fans=NULL;

e.friends=NULL; e.hobby=NULL;

}

status InitAVL(BSTree &T){

//初始条件：平衡二叉排序树不存在

//操作结果：初始化创建一个新的平衡二叉排序树

if(T) return ERROR;

T=(BSTree)malloc(sizeof(BSTNode));

if(!T) return ERROR;

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

InitElem(T->data);

return OK;

}

status DestroyAVL(BSTree &T){

//初始条件：平衡二叉排序树T存在

//操作结果：摧毁平衡二叉排序树T

if(!T) return ERROR;

DestroyAVL(T->lchild); DestroyAVL(T->rchild);

free(T); T=NULL;

return OK;

}

status PTraverseAVL(BSTree T){

//将二叉树T以先序序列输出

if(!T) return ERROR;

printf("%3d ",T->data.id);

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

PTraverseAVL(T->lchild); PTraverseAVL(T->rchild);

return OK;

}

void gotoxy(int x,int y){

//移动光标，将光标移至坐标（x，y)

COORD coord={x,y};

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE),coord);

}

int power(int x){

//计算并返回2的x次幂，用于二叉树图形化输出

int y=1;

for(int i=0;i<x;i++){

y\*=2;

}

return y;

}

status height(BSTree &T,int i){

//计算二叉树T各结点的深度并赋值

if(!T) return ERROR;

T->h=i;

height(T->lchild,i+1); height(T->rchild,i+1);

return OK;

}

status TraversAVL(int x,int y,BSTree T){

//图形化输出二叉树

if(!T) return ERROR;

gotoxy(x,y); //根结点存在，输出根结点id

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

int j=middle/power(T->h);

if((!T->lchild)&&(!T->rchild)) return OK; //该结点无左右子树，结束函数

gotoxy(x,y+1); printf("|");

if(T->lchild){ //输出左子树id

gotoxy(x-j,y+3); printf("|");

gotoxy(x-j,y+2);

for(int i=0;i<j;i++) printf("-");

TraversAVL(x-j,y+4,T->lchild);

}

if(T->rchild){ //输出右子树

gotoxy(x+j,y+3); printf("|");

gotoxy(x,y+2);

for(int i=0;i<j;i++) printf("-");

TraversAVL(x+j,y+4,T->rchild);

}

return OK;

}

status TraverseAVL(int x,int y,BSTree T){

height(T,1);

TraversAVL(x,y,T);

printf("\n");

}

status SearchAVL(BSTree T,int e,BSTree &Ta){

//在二叉树T中查找关键字为e的结点，若存在该结点，返回TRUE,

//并使Ta指向该结点,否则返回FALSE

if(!T) return FALSE;

if(e==T->data.id){

Ta=T;

return TRUE;

}

else if(e<T->data.id){

if(SearchAVL(T->lchild,e,Ta)) return TRUE;

}

else {

if(SearchAVL(T->rchild,e,Ta)) return TRUE;

}

return FALSE;

}

status Save(BSTree T,FILE \*fp){

//将二叉树T保存入fp指向的文件中

if(!T) return OK;

else{

if(!fwrite(&(T->data),sizeof(ElemType),1,fp))

return ERROR;

}

if(!Save(T->lchild,fp)) return ERROR;

if(!Save(T->rchild,fp)) return ERROR;

return OK;

}

status Load(BSTree &T,char \*filename){

//将fp文件名为filename的文件中的内容加载入二叉树T中

FILE \*fp;

if ((fp=fopen(filename,"rb"))==NULL){ //文件打开失败

printf("文件打开失败！\n ");

return ERROR;

}

ElemType e; InitElem(e);

while(fread(&e,sizeof(ElemType),1,fp)){

if(!InsertAVL(T,e)) return ERROR;

}

//检查无法读取时是否到文件尾

if(!feof(fp)) return ERROR;

height(T,1);

fclose(fp);

return OK;

}

集合部分：

#include "BSTree.h"

//以下几个函数直接引用二叉树函数

status set\_init(BSTree &T){

return InitAVL(T);

}

status set\_destroy(BSTree &T){

return DestroyAVL(T);

}

status set\_member(BSTree T,int e,BSTree &Ta){

return SearchAVL(T,e,Ta);

}

status set\_insert(BSTree &T,ElemType e){

return InsertAVL(T,e);

}

status set\_remove(BSTree &T,int e){

return DeleteAVL(T,e);

}

//以下函数为二叉树函数的组合引用

status set\_intersection(BSTree T1,BSTree T2,BSTree &T){

//求集合T1与T2的交集，并赋值给T

if(!T1) return OK;

BSTree Ta=NULL;

if(SearchAVL(T2,T1->data.id,Ta)==TRUE) //如果结点T1在二叉树T2中存在，将结点T1插入二叉树T

if(InsertAVL(T,T1->data)==ERROR) return ERROR;

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

if(!set\_intersection(T1->rchild,T2,T)) return ERROR; //分别遍历T1左子树与右子树

return OK;

}

status set\_union(BSTree &T,BSTree T1){

//求集合T1与T的并集，并赋值给T

if(!T1) return OK;

InsertAVL(T,T1->data); //将T1插入T

if(!set\_union(T,T1->lchild)) return ERROR;

if(!set\_union(T,T1->rchild)) return ERROR;

//将T1的左右子树插入T

return OK;

}

status set\_diffrence(BSTree &T,BSTree T1){

//求集合T与T1的差集，并赋值给T

if(!T1) return OK;

DeleteAVL(T,T1->data.id); //将T1从T中删除

if(!set\_diffrence(T,T1->lchild)) return ERROR; //将T1的左右子树从T中删除

if(!set\_diffrence(T,T1->rchild)) return ERROR;

return OK;

}

status set\_size(BSTree T){

//计算集合中结点个数

if(!T) return ERROR;

int a=set\_size(T->lchild),b=set\_size(T->rchild);

return a+b+1;

}

status set\_subset(BSTree T1,BSTree T2){

//判断T2是否为T1的子集，若是子集返回TRUE，否则返回FALSE

if(T2==NULL) return TRUE;

BSTree Ta=NULL;

if(set\_member(T1,T2->data.id,Ta)){

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

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

return TRUE;

}

else return FALSE;

}

status set\_equal(BSTree T1,BSTree T2){

//判断集合T1与T2是否相等，若相等返回TRUE，否则返回FALSE

if(!set\_subset(T1,T2)) return FALSE;

if(!set\_subset(T2,T1)) return FALSE;

return TRUE;

}

void Traverse(BSTree T){

//将集合以集合形式遍历输出

if(T){

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

if(T->lchild){

printf(","); Traverse(T->lchild);

}

if(T->rchild){

printf(","); Traverse(T->rchild);

}

}

}

status set\_traverse(BSTree T){

if(!T) return ERROR;

printf("{ ");

Traverse(T);

printf(" }\n");

return OK;

}

**应用层部分：**

#include "ADT.h"

//共同关注、共同好友

status common(BSTree T1,BSTree T2,BSTree &T){

//求取数据项T1与T2的共同成员

if(!set\_intersection(T1,T2,T)) return ERROR;

return OK;

}

//二度好友

status search\_friend(BSTree T,BSTree Ta,BSTree &Tb){

//二度好友的辅助函数，二叉树Ta为T中某成员的朋友集

//将朋友集中成员的朋友信息存入集合Tb

BSTree Tc=NULL; //指向各朋友的指针

if(!Ta) return OK;

if(!set\_member(T,Ta->data.id,Tc)) return ERROR;

//通过并集操作将朋友的朋友集赋值入二叉树Tb

if(!set\_union(Tb,Tc->data.friends)) return ERROR;

if(!search\_friend(T,Ta->lchild,Tb)) return ERROR;

return OK;

}

status second\_friend(BSTree T,int id){

//将二叉树中id为id的成员的二度好友

if(!T) return ERROR;

BSTree Ta=NULL; //一度朋友集

BSTree Tb=NULL; //二度朋友集

if(!set\_member(T,id,Ta)){

printf("此人不存在！\n");

return ERROR;

}

if(!(Ta->data.friends)){

printf("id为%d的是%s,该成员无朋友！\n",Ta->data.id,Ta->data.name);

return ERROR;

}

if(!search\_friend(T,Ta->data.friends,Tb)) return ERROR;

if(!Tb){

printf("id为%d的是%s,该成员无二度好友！\n",Ta->data.id,Ta->data.name);

}

else{

printf("id为%d的是%s,%s的二度好友为：\n",Ta->data.id,Ta->data.name,Ta->data.name);

PTraverseAVL(Tb); printf("\n");

}

return OK;

}

status Common\_attention(BSTree &T){

//共同关注查找系统

int id=0;

BSTree Ta=NULL,Tb=NULL,Tc=NULL; //辅助函数

printf("请输入第一个人的id：");

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

if(!set\_member(T,id,Ta)){

printf("此人不存在！\n");

return ERROR;

}

printf("请输入第二个人的id：");

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

if(!set\_member(T,id,Tb)){

printf("此人不存在！\n");

return ERROR;

}

if(!common(Ta->data.attention,Tb->data.attention,Tc)){

printf("求取共同关注失败！\n"); return ERROR;

}

if(!(Ta->data.attention)){

printf("id为%d的是%s,其无关注人！\n",Ta->data.id,Ta->data.name);

return ERROR;

}

if(!(Tb->data.attention)){

printf("id为%d的是%s,其无关注人！\n",Tb->data.id,Tb->data.name);

return ERROR;

}

printf("id为%d的是%s,其关注的人有：\n",Ta->data.id,Ta->data.name);

PTraverseAVL(Ta->data.attention); printf("\n");

printf("id为%d的是%s,其关注的人有：\n",Tb->data.id,Tb->data.name);

PTraverseAVL(Tb->data.attention); printf("\n");

if(!Tc){

printf("%s与%s无共同关注！\n",Ta->data.name,Tb->data.name);

return OK;

}

else{

printf("共同关注有：\n");

PTraverseAVL(Tc); printf("\n");

}

return OK;

}

status Common\_hobby(BSTree &T){

//共同喜好查找系统

int id=0;

BSTree Ta=NULL,Tb=NULL,Tc=NULL; //辅助函数

printf("请输入第一个人的id：");

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

if(!set\_member(T,id,Ta)){

printf("此人不存在！\n");

return ERROR;

}

printf("请输入第二个人的id：");

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

if(!set\_member(T,id,Tb)){

printf("此人不存在！\n");

return ERROR;

}

if(!common(Ta->data.hobby,Tb->data.hobby,Tc)){

printf("求取共同喜好失败！\n"); return ERROR;

}

if(!(Ta->data.hobby)){

printf("id为%d的是%s,其无喜好！\n",Ta->data.id,Ta->data.name);

return ERROR;

}

if(!(Tb->data.hobby)){

printf("id为%d的是%s,其无喜好！\n",Tb->data.id,Tb->data.name);

return ERROR;

}

printf("id为%d的是%s,其喜好有：\n",Ta->data.id,Ta->data.name);

PTraverseAVL(Ta->data.hobby); printf("\n");

printf("id为%d的是%s,其喜好有：\n",Tb->data.id,Tb->data.name);

PTraverseAVL(Tb->data.hobby); printf("\n");

if(!Tc){

printf("%s与%s无共同喜好！\n",Ta->data.name,Tb->data.name);

return OK;

}

else{

printf("共同喜好有：\n");

PTraverseAVL(Tc); printf("\n");

}

return OK;

}

status hobby\_set\_input(BSTree &H,int &n){

//读取文件hobby.txt,将其加载入二叉树H,

//n记录读取数量

FILE \*fp;

ElemType e; InitElem(e);

int number=0;

if(H){ //清空二叉树

set\_destroy(H); H=NULL;

}

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

printf("打开文件hobby.txt失败!\n");

return ERROR;

}

while(fgets(e.name,5,fp)){

e.id=number++;

e.attention=NULL;

e.fans=NULL;

e.friends=NULL;

e.hobby=NULL;

if(!set\_insert(H,e)) return ERROR;

}

if(!feof(fp)){

fclose(fp); return ERROR;

}

n=number;

fclose(fp);

height(H,1);

return OK;

}

status name\_set\_input(BSTree &N,int &n){

//读取文件name.txt,将其加载入二叉树N

FILE \*fp;

ElemType e;

int number=0;

if(N){ //清空二叉树

set\_destroy(N); N=NULL;

}

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

printf("打开文件name.txt失败!\n");

return ERROR;

}

while(fgets(e.name,7,fp)){

e.id=number++;

e.attention=NULL;

e.fans=NULL;

e.friends=NULL;

e.hobby=NULL;

if(!set\_insert(N,e)) return ERROR;

}

if(!feof(fp)){

fclose(fp); return ERROR;

}

n=number;

fclose(fp);

height(N,1);

return OK;

}

status name\_input(BSTree &T,int gross,int max,int min,BSTree N){

//随机生成一棵二叉树，该二叉树结点id不超过gross，结点数为min-min+max

int j=0;

BSTree Ta=NULL;

for(int i=0;i<=rand()%max+min;){

j=rand()%gross;

if(!set\_member(N,j,Ta)) return ERROR;

if(set\_insert(T,Ta->data)) i++;

}

return OK;

}

status relation\_input(BSTree &T,BSTree N,BSTree H,int a,int b){

//随机生成关系，N为姓名集，H为喜好集

//a为姓名集的最大数,b为喜好集的最大数

if(!T) return OK;

if(!name\_input(T->data.friends,a,5,3,N)) return ERROR;

if(!name\_input(T->data.attention,a,5,1,N)) return ERROR;

if(!name\_input(T->data.hobby,b,3,1,H)) return ERROR;

if(!name\_input(T->data.fans,a,5,1,N)) return ERROR;

if(!relation\_input(T->lchild,N,H,a,b)) return ERROR;

if(!relation\_input(T->rchild,N,H,a,b)) return ERROR;

return OK;

}

status Tree\_input(BSTree &T,BSTree N,int b){

//随机生成一个结点数量为b的成员信息

set\_destroy(T); set\_init(T);

int j=0;

BSTree Ta=NULL;

for(int i=0;i<b;){

j=rand()%b;

if(!set\_member(N,j,Ta)) return ERROR;

if(set\_insert(T->lchild,Ta->data)) i++;

}

return OK;

}

status improve\_friend(BSTree &T,BSTree Ta,ElemType e){

//完善成员的朋友集信息,Ta为某成员朋友集,e为该成员信息

BSTree Tb=NULL;

if(!Ta) return OK;

else if(set\_member(T,Ta->data.id,Tb)){

set\_insert(Tb->data.friends,e);

}

else return ERROR;

if(!improve\_friend(T,Ta->lchild,e)) return ERROR;

if(!improve\_friend(T,Ta->lchild,e)) return ERROR;

return OK;

}

status improve\_attention(BSTree &T,BSTree Ta,ElemType e){

//完善成员的关注集信息,Ta为某成员粉丝集,e为该成员信息

BSTree Tb=NULL;

if(!Ta) return OK;

else if(set\_member(T,Ta->data.id,Tb)){

set\_insert(Tb->data.attention,e);

}

else return ERROR;

if(!improve\_attention(T,Ta->lchild,e)) return ERROR;

if(!improve\_attention(T,Ta->lchild,e)) return ERROR;

return OK;

}

status improve\_fans(BSTree &T,BSTree Ta,ElemType e){

//完善成员的粉丝集信息,Ta为某成员关注集,e为该成员信息

BSTree Tb=NULL;

if(!Ta) return OK;

else if(set\_member(T,Ta->data.id,Tb)){

set\_insert(Tb->data.fans,e);

}

else return ERROR;

if(!improve\_fans(T,Ta->lchild,e)) return ERROR;

if(!improve\_fans(T,Ta->lchild,e)) return ERROR;

return OK;

}

status search\_relation(BSTree &T){

//检查T的关系集中是否有自己，如果有，删除

BSTree Ta=NULL;

if(!T) return OK;

//检查朋友集

if(set\_member(T->data.friends,T->data.id,Ta)){

if(!set\_remove(T->data.friends,T->data.id)) return ERROR;

}

//检查粉丝集

if(set\_member(T->data.fans,T->data.id,Ta)){

if(!set\_remove(T->data.fans,T->data.id)) return ERROR;

}

//检查关注集

if(set\_member(T->data.attention,T->data.id,Ta)){

if(!set\_remove(T->data.attention,T->data.id)) return ERROR;

}

return OK;

}

status improve\_relation(BSTree &T,BSTree &T1){

//完善成员关系集，使朋友互相存在，使成员的粉丝的关注中有该成员

//防止成员的朋友集、粉丝集、关注集中有本人

if(!T1) return OK;

if(!improve\_friend(T,T1->data.friends,T1->data)) return ERROR;

if(!improve\_attention(T,T1->data.fans,T1->data)) return ERROR;

if(!improve\_fans(T,T1->data.attention,T1->data)) return ERROR;

if(!search\_relation(T1)) return ERROR;

if(!improve\_relation(T,T1->lchild)) return ERROR;

if(!improve\_relation(T,T1->rchild)) return ERROR;

return OK;

}

status operator\_tree(BSTree &T,BSTree H,int a,int b){

//对应用层中数据进行查找，随后进行删除、增加、修改等操作

BSTree Ta=NULL,Tb=NULL;

int id=0, i=0,j=0;

printf("请输入要查找的id：");

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

if(!set\_member(T,id,Ta)){

printf("查找失败！\n"); return ERROR;

}

else{

printf("查找成功,具体信息如下：\n");

printf(" id："); printf("%d\n",Ta->data.id);

printf("name："); printf("%s\n",Ta->data.name);

printf("friend：\n"); PTraverseAVL(Ta->data.friends); printf("\n");

printf("fans：\n"); PTraverseAVL(Ta->data.fans); printf("\n");

printf("attention：\n"); PTraverseAVL(Ta->data.attention); printf("\n");

printf("hobby：\n"); PTraverseAVL(Ta->data.hobby); printf("\n");

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

printf("1.name 2.friend 3.fans\n");

printf("4.attention 5.hobby 0.返回\n");

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

printf("请输入要进行操作的内容的序号[0-5]："); scanf("%d",&j); getchar();

while(j<0||j>5){

printf("请重新输入：");

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

}

switch(j){

case 0: break;

case 1:

printf("请输入新的姓名：");

scanf("%s",&Ta->data.name);

printf("修改成功！\n");

break;

case 2: //好友集操作

printf("1.增加 2.删除 0.返回\n");

printf("请输入要进行的操作[0-3]："); scanf("%d",&i); getchar();

while(i<0||i>3){

printf("请重新输入：");

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

}

switch(i){

case 0: break;

case 1:

id=0;

printf("请输入要插入的好友的id：");

scanf("%d",&id);

if(set\_member(T,id,Tb)){ //好友信息存在

if(set\_insert(Ta->data.friends,Tb->data)){

if(set\_insert(Tb->data.friends,Ta->data))

printf("插入成功！\n"); break;

}

printf("插入失败！\n"); break;

}

else{ //好友信息不存在

printf("该好友不存在，插入失败！\n");

}

break;

case 2:

id=0;

printf("请输入要删除的好友的id：");

scanf("%d",&id);

if(set\_member(Ta->data.friends,id,Tb)){ //好友信息存在

if(set\_remove(Ta->data.friends,id)){

if(set\_member(T,id,Tb)){

if(set\_remove(Tb->data.friends,Ta->data.id))

printf("已删除好友%s！\n",Tb->data.name); break;

}

}

printf("删除失败！\n"); break;

}

else{ //好友信息不存在

printf("该好友不存在，删除失败！\n");

}

break;

}

break;

case 3: //粉丝集操作

printf("1.增加 2.删除 0.返回\n");

printf("请输入要进行的操作[0-3]："); scanf("%d",&i); getchar();

while(i<0||i>3){

printf("请重新输入：");

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

}

switch(i){

case 0: break;

case 1:

id=0;

printf("请输入要插入的粉丝的id：");

scanf("%d",&id);

if(set\_member(T,id,Tb)){ //好友信息存在

if(set\_insert(Ta->data.fans,Tb->data)){

if(set\_insert(Tb->data.attention,Ta->data))

printf("插入成功！\n"); break;

}

printf("插入失败！\n"); break;

}

else{ //好友信息不存在

printf("该粉丝不存在，插入失败！\n");

}

break;

case 2:

id=0;

printf("请输入要删除的粉丝的id：");

scanf("%d",&id);

if(set\_member(Ta->data.fans,id,Tb)){ //好友信息存在

if(set\_remove(Ta->data.fans,id)){

if(set\_member(T,id,Tb)){

if(set\_remove(Tb->data.attention,Ta->data.id))

printf("已删除粉丝%s！\n",Tb->data.name); break;

}

}

printf("删除失败！\n"); break;

}

else{ //好友信息不存在

printf("该粉丝不存在，删除失败！\n");

}

break;

}

break;

case 4: //关注集操作

printf("1.增加 2.删除 0.返回\n");

printf("请输入要进行的操作[0-3]："); scanf("%d",&i); getchar();

while(i<0||i>3){

printf("请重新输入：");

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

}

switch(i){

case 0: break;

case 1:

id=0;

printf("请输入要关注的人的id：");

scanf("%d",&id);

if(set\_member(T,id,Tb)){ //好友信息存在

if(set\_insert(Ta->data.attention,Tb->data)){

if(set\_insert(Tb->data.fans,Ta->data))

printf("关注成功！\n"); break;

}

printf("关注失败！\n"); break;

}

else{ //好友信息不存在

printf("此人不存在，关注失败！\n");

}

break;

case 2:

id=0;

printf("请输入要取消关注的人的id：");

scanf("%d",&id);

if(set\_member(Ta->data.attention,id,Tb)){ //好友信息存在

if(set\_remove(Ta->data.attention,id)){

if(set\_member(T,id,Tb)){

if(set\_remove(Tb->data.fans,Ta->data.id))

printf("已取消关注%s！\n",Tb->data.name); break;

}

}

printf("删除失败！\n"); break;

}

else{ //好友信息不存在

printf("该好友不存在，删除失败！\n");

}

break;

}

break;

case 5: //爱好集操作

printf("1.增加 2.删除 0.返回\n");

printf("请输入要进行的操作[0-3]："); scanf("%d",&i); getchar();

while(i<0||i>3){

printf("请重新输入：");

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

}

switch(i){

case 0: break;

case 1:

id=0;

printf("请输入要插入的喜好的id[0-%d]：",b);

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

if(set\_member(H,id,Tb)){

printf("id为%d的喜好为%s!\n",id,Tb->data.name);

if(set\_insert(Ta->data.hobby,Tb->data)){

printf("插入成功！\n"); break;

}

else{

printf("插入失败！\n"); break;

}

}

else{

printf("该喜好不存在，插入失败！\n");

}

break;

case 2:

id=0;

printf("请输入要删除的喜好的id：");

scanf("%d",&id);

if(set\_member(Ta->data.hobby,id,Tb)){

if(set\_remove(Ta->data.hobby,id)){

printf("已删除喜好%s！\n",Tb->data.name);

}

else{

printf("删除失败！\n");

}

}

else{

printf("该成员无此喜好，删除失败！\n");

}

break;

}

break;

}

}

}

status operator\_tree2(BSTree &T,BSTree H,int a,int b){

//对应用层中数据进行查找，随后进行删除、增加、修改等操作

BSTree Ta=NULL,Tb=NULL;

int id=0, i=0,j=0;

printf("请输入要查找的id：");

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

if(!set\_member(T,id,Ta)){

printf("查找失败！\n"); return ERROR;

}

else{

printf("查找成功,id为%d的是%s!：\n",id,Ta->data.name);

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

printf("1.friend 2.fans 3.attention \n");

printf("4.hobby 0.返回\n");

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

printf("请输入要查询的内容的序号[0-4]："); scanf("%d",&j); getchar();

while(j<0||j>4){

printf("请重新输入：");

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

}

switch(j){

case 0: break;

case 1: //好友集操作

PTraverseAVL(Ta->data.friends);

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

printf("1.增加 2.删除 0.返回\n");

printf("请输入要进行的操作[0-3]："); scanf("%d",&i); getchar();

while(i<0||i>3){

printf("请重新输入：");

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

}

switch(i){

case 0: break;

case 1:

id=0;

printf("请输入要插入的好友的id：");

scanf("%d",&id);

if(set\_member(T,id,Tb)){ //好友信息存在

if(set\_insert(Ta->data.friends,Tb->data)){

if(set\_insert(Tb->data.friends,Ta->data))

printf("插入成功！\n"); break;

}

printf("插入失败！\n"); break;

}

else{ //好友信息不存在

printf("该好友不存在，插入失败！\n");

}

break;

case 2:

id=0;

printf("请输入要删除的好友的id：");

scanf("%d",&id);

if(set\_member(Ta->data.friends,id,Tb)){ //好友信息存在

if(set\_remove(Ta->data.friends,id)){

if(set\_member(T,id,Tb)){

if(set\_remove(Tb->data.friends,Ta->data.id))

printf("已删除好友%s！\n",Tb->data.name); break;

}

}

printf("删除失败！\n"); break;

}

else{ //好友信息不存在

printf("该好友不存在，删除失败！\n");

}

break;

}

break;

case 2: //粉丝集操作

PTraverseAVL(Ta->data.fans);

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

printf("1.增加 2.删除 0.返回\n");

printf("请输入要进行的操作[0-3]："); scanf("%d",&i); getchar();

while(i<0||i>3){

printf("请重新输入：");

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

}

switch(i){

case 0: break;

case 1:

id=0;

printf("请输入要插入的粉丝的id：");

scanf("%d",&id);

if(set\_member(T,id,Tb)){ //好友信息存在

if(set\_insert(Ta->data.fans,Tb->data)){

if(set\_insert(Tb->data.attention,Ta->data))

printf("插入成功！\n"); break;

}

printf("插入失败！\n"); break;

}

else{ //好友信息不存在

printf("该粉丝不存在，插入失败！\n");

}

break;

case 2:

id=0;

printf("请输入要删除的粉丝的id：");

scanf("%d",&id);

if(set\_member(Ta->data.fans,id,Tb)){ //好友信息存在

if(set\_remove(Ta->data.fans,id)){

if(set\_member(T,id,Tb)){

if(set\_remove(Tb->data.attention,Ta->data.id))

printf("已删除粉丝%s！\n",Tb->data.name); break;

}

}

printf("删除失败！\n"); break;

}

else{ //好友信息不存在

printf("该粉丝不存在，删除失败！\n");

}

break;

}

break;

case 3: //关注集操作

PTraverseAVL(Ta->data.attention);

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

printf("1.增加 2.删除 0.返回\n");

printf("请输入要进行的操作[0-3]："); scanf("%d",&i); getchar();

while(i<0||i>3){

printf("请重新输入：");

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

}

switch(i){

case 0: break;

case 1:

id=0;

printf("请输入要关注的人的id：");

scanf("%d",&id);

if(set\_member(T,id,Tb)){ //好友信息存在

if(set\_insert(Ta->data.attention,Tb->data)){

if(set\_insert(Tb->data.fans,Ta->data))

printf("关注成功！\n"); break;

}

printf("关注失败！\n"); break;

}

else{ //好友信息不存在

printf("此人不存在，关注失败！\n");

}

break;

case 2:

id=0;

printf("请输入要取消关注的人的id：");

scanf("%d",&id);

if(set\_member(Ta->data.attention,id,Tb)){ //好友信息存在

if(set\_remove(Ta->data.attention,id)){

if(set\_member(T,id,Tb)){

if(set\_remove(Tb->data.fans,Ta->data.id))

printf("已取消关注%s！\n",Tb->data.name); break;

}

}

printf("删除失败！\n"); break;

}

else{ //好友信息不存在

printf("该好友不存在，删除失败！\n");

}

break;

}

break;

case 4: //爱好集操作

PTraverseAVL(Ta->data.hobby);

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

printf("1.增加 2.删除 0.结束\n");

printf("请输入要进行的操作[0-3]："); scanf("%d",&i); getchar();

while(i<0||i>5){

printf("请重新输入：");

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

}

switch(i){

case 0: break;

case 1:

id=0;

printf("请输入要插入的喜好的id[0-%d]：",b);

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

if(set\_member(H,id,Tb)){

printf("id为%d的喜好为%s!\n",id,Tb->data.name);

if(set\_insert(Ta->data.hobby,Tb->data)){

printf("插入成功！\n"); break;

}

else{

printf("插入失败！\n"); break;

}

}

else{

printf("该喜好不存在，插入失败！\n");

}

break;

case 2:

id=0;

printf("请输入要删除的喜好的id：");

scanf("%d",&id);

if(set\_member(Ta->data.hobby,id,Tb)){

if(set\_remove(Ta->data.hobby,id)){

printf("已删除喜好%s！\n",Tb->data.name);

}

else{

printf("删除失败！\n");

}

}

else{

printf("该成员无此喜好，删除失败！\n");

}

break;

}

break;

}

}

}

**链表部分：**

#include "应用层.h"

typedef struct Node{

int id;

char name[10]; //集合名称

BSTree T; //集合头指针

}Node;

typedef struct List{

Node data;

struct List \*prior;

struct List \*next;

}List,\*LinkList;

status TraverseLink(LinkList L){

//输出链表L中所有集合的id

if(!L) return ERROR;

LinkList q=L;

while(q){

printf("%2d ",q->data.id);

q=q->next;

}

printf("\n");

return OK;

}

status search\_link(LinkList L,int id,LinkList &L1){

//对链表L进行查询，若链表L中存在集合id为id的结点，将该结点赋值给L1，

//并返回TRUE，否则返回FALSE

if(!L) return ERROR;

LinkList q=L;

while(q){

if(q->data.id==id){

L1=q; return TRUE;

}

else q=q->next;

}

return FALSE;

}

status List\_insert(LinkList &L,Node e){

//将数据e插入链表L中

if(!L){

L=(LinkList)malloc(sizeof(List));

if(!L) return ERROR;

L->data=e;

L->prior=NULL;

L->next=NULL;

}

else{

LinkList q=NULL;

LinkList t=L;

q=(LinkList)malloc(sizeof(List));

if(!q) return ERROR;

q->data=e;

q->prior=NULL;

q->next=t;

L=q;

}

return OK;

}

status save\_list(LinkList L){

//将链表信息保存入文件List.dat中

if(!L) return ERROR;

FILE \*fp;

LinkList q=L;

if((fp=fopen("List.dat","wb"))==NULL){ //文件打开失败

printf("文件List.dat打开失败！\n ");

return ERROR;

}

while(q){

if(!fwrite(&(q->data),sizeof(Node),1,fp)) {

fclose(fp); return ERROR;

}

q=q->next;

}

fclose(fp);

if(!L) return ERROR;

return OK;

}

status save\_linklist(LinkList L){

//将链表L的集合保存入集合名文件中

if(!L) return ERROR;

FILE \*fp;

LinkList q=L;

if(!save\_list(L)) return ERROR;

while(q){

if ((fp=fopen(q->data.name,"wb"))==NULL){ //文件打开失败

return ERROR;

}

if(!Save(q->data.T,fp)) {

fclose(fp); return ERROR;

}

fclose(fp);

q=q->next;

}

return OK;

}

status load\_linklist(LinkList &L){

//将文件List.dat的信息载入链表L中

L=NULL;

FILE \*fp;

LinkList q=NULL;

if((fp=fopen("List.dat","rb"))==NULL){ //文件打开失败

printf("文件List.dat打开失败！\n ");

return ERROR;

}

while(!feof(fp)){

q=(LinkList)malloc(sizeof(List));

if(!q){

printf("分配空间出错！\n"); return ERROR;

}

if(!fread(&(q->data),sizeof(Node),1,fp)) break;

if(!List\_insert(L,q->data)) return ERROR;

}

fclose(fp);

q=L;

while(q){

q->data.T=NULL;

if(!Load(q->data.T,q->data.name))return ERROR;

q=q->next;

}

return OK;

}

status tree\_to\_link(LinkList &L,BSTree T){

//将二叉树T存入链表L

LinkList L1=NULL;

Node q;

int a=0;

char setname[10];

if(!L) printf("当前集合文件中无集合！\n");

else{

printf("当前所有集合id包括：");

TraverseLink(L);

}

printf("请输入集合的id：");

scanf("%d",&a);

while(search\_link(L,a,L1)){

printf("此id已存在，请重新输入：");

scanf("%d",a);

}

printf("请输入集合名：");

scanf("%s",setname);

q.id=a; strcpy(q.name,setname);

q.T=T;

if(List\_insert(L,q)){

printf("该集合已保存入集合文件！\n"); return OK;

}

else{

printf("保存失败！\n"); return ERROR;

}

}

void set\_menu(LinkList &L,BSTree &T,LinkList &buff){

int i=1;

int id=0;

BSTree Ta=NULL; //辅助二叉树

char c='y';

LinkList L1=NULL;

Node e;

ElemType a;

while(i){

system("cls");

printf("\n-----------------集合操作系统------------------------\n\n");

printf(" 1.进入集合操作 2.上一集合 3.下一集合\n");

printf(" 4.set\_init 5.set\_destroy 6.set\_member\n");

printf(" 7.set\_size 8.set\_insert 9.set\_remove\n");

printf(" 10.set\_traverse 11.set\_save 0.返回\n");

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

printf("请输入你的操作[0-10]：");

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

switch(i){

case 0: break;

case 1:

if(L){

printf("已进入集合操作！\n");

system("pause"); break;

}

if(!load\_linklist(L)){

printf("加载集合失败！\n");

}

else{

printf("集合加载成功！\n");

buff=L;

T->lchild=L->data.T;

printf("开始集合操作，当前为集合%d\n",L->data.id);

}

system("pause"); break;

case 2:

if(buff->prior==NULL){

printf("已至第一个集合！\n");

}

else{

buff=buff->prior;

T->lchild=buff->data.T;

printf("已由集合%d切换至集合%d！\n",L1->data.id,L1->prior->data.id);

}

system("pause"); break;

case 3:

if(buff->next==NULL){

printf("已至最后一个集合！\n");

}

else{

buff=buff->next;

T->lchild=buff->data.T;

printf("已由集合%d切换至集合%d！\n",L1->data.id,L1->next->data.id);

}

system("pause"); break;

case 4:

if(T) printf("该集合已存在，请先进行摧毁！\n");

else{

if(set\_init(T)) printf("初始化成功！\n");

else printf("初始化失败！\n");

}

system("pause"); break;

case 5:

if(!T) printf("集合不存在，摧毁失败！\n");

else{

if(set\_destroy(T)) printf("摧毁成功！\n");

else printf("摧毁失败！\n");

}

system("pause"); break;

case 6:

if(!T) printf("集合不存在，请先进行初始化！\n");

else if(!T->lchild) printf("该集合为空，请先插入数据！\n");

else{

printf("请输入id：");

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

if(set\_member(T->lchild,id,Ta)){

printf("查找成功！\n");

printf("id为%d的是%s\n",id,Ta->data.name);

}

else printf("查找失败!\n");

}

system("pause"); break;

case 7:

if(!T) printf("集合不存在，请先进行初始化！\n");

else if(!T->lchild) printf("该集合为空，请先插入数据！\n");

else{

printf("当前集合为%d，其成员个数为%d!\n",buff->data.id,set\_size(T->lchild));

}

system("pause"); break;

case 8:

if(!T) printf("集合不存在，请先进行初始化！\n");

else{

while(c=='y'){

printf("请输入id："); scanf("%d",&a.id); getchar();

printf("请输入姓名："); scanf("%s",&a.name); getchar();

if(set\_insert(T->lchild,a)){

printf("插入成功！\n");

height(T->lchild,1);

}

else printf("插入失败！\n");

printf("是否继续('y'or'n')？：");

scanf("%c",&c); getchar();

if(c!='y'&&c!='n'){

printf("请重新输入：");

scanf("%c",&c); getchar();

}

}

}

system("pause"); break;

case 9:

if(!T) printf("集合不存在，请先进行初始化！\n");

else if(!T->lchild) printf("该集合为空，请先插入数据！\n");

else{

printf("请输入id："); scanf("%d",&id);

if(set\_remove(T->lchild,id)) printf("删除成功！\n");

else printf("删除失败！");

}

system("pause"); break;

case 10:

if(!T) printf("集合不存在，请先进行初始化！\n");

else if(!T->lchild) printf("该集合为空，请先插入数据！\n");

else{

printf("该集合为：\n");

Traverse(T->lchild);

}

system("pause"); break;

case 11:

if(!T) printf("集合不存在，保存失败！\n");

tree\_to\_link(L,T);

system("pause"); break;

}

}

}