

**课 程 设 计 报 告**

**题目 :基于AVL树表示的集合ADT实现与应用**

**课程名称： 数据结构课程设计**

**专业班级： 物联网1601**

**学 号： U201614887**

**姓 名： 林子涵**

**指导教师： 李剑军**

**报告日期： 2018年2月9日**

**计算机科学与技术学院**

# 任务书

**题目三 基于AVL树表示的集合ADT实现与应用**

* **设计目的**

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

* **设计内容**

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

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

**（1）**抽象层面AVL可以表示数据元素之间层次关系或一对多关系。

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

* **主要运算与功能要求：**

**（1）**交互式操作界面(并非一定指图形式界面)；

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

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

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

**（5）**主要数据对象的数据文件组织与存储。

* **设计提示**

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

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

**目录**

[任务书 1](#_Toc508010952)

[引言 2](#_Toc508010953)

[1.1课题背景与意义 2](#_Toc508010954)

[1.2问题描述与技术现状分析 2](#_Toc508010955)

[2 系统总体设计 3](#_Toc508010956)

[2.1 有关数据结构的定义 3](#_Toc508010957)

[2.2 系统模块结构图 4](#_Toc508010958)

[3 数据结构与算法详细设计 4](#_Toc508010959)

[3.1 底层数据结构设计 4](#_Toc508010960)

[3.2集合抽象数据结构设计 7](#_Toc508010961)

[3.3 应用层结构设计 11](#_Toc508010962)

[4 系统实现 13](#_Toc508010963)

[4.1开发环境与相关支持 13](#_Toc508010964)

[4.2主要函数说明与复杂度分析 13](#_Toc508010965)

[5 系统测试与结果 15](#_Toc508010966)

[6 使用说明 21](#_Toc508010967)

[7 特色与不足 21](#_Toc508010968)

[8 感悟与体会 21](#_Toc508010969)

[9主要参考文献 22](#_Toc508010970)

[附录 23](#_Toc508010971)

# 引言

## 1.1课题背景与意义

在现今网络中，每个人都扮演着复杂的角色，人与人之间的关系错综复杂，尤其随着各类社交网站的出现，在扩大了人们交际圈的同时也为维护人与人之间复杂关系的表达带来了挑战。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。同时因为其高效的数据查找、修改与删除功能非常适合作为抽象数据结构ADT的底层实现。本课题意图利用基于AVL的抽象集合数据结构来实现对个人粉丝集、好友集、关注集等集合的管理。

通过本课题，我们可以深入理解抽线结构与其底层实现之间的关系，同时体会到高效的数据结构对于数据维护的重要性，进一步锻炼自己的代码能力。

## 1.2问题描述

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。

## 1.3国内外技术现状分析

近几年随着网络社交的兴起，越来越多的人在网络建立自己的人脉圈，随之而来的关系处理与集合运算优化已经成为各大互联网公司研究的重点，其中如facebook、新浪等为代表的企业在这方面均有所突破，其中应用最广泛的便是数据库技术，各个公司均在mysql基础上建立了自己的独特的数据库系统，而其核心便是集合的运算，传统数据库中已经实现了简单的集合代数运算，对其稍加封装即可维护复杂的集合关系。我们的课题与其类似，都是处理集合关系，但我们采用平衡树的数据结构来构建集合关系。

现今平衡树的各类算法已经比较成熟，其各种实现均能够在网络上轻易找到，又因为其出色的稳定性，平衡树的应用范围十分广泛。通过查阅相关资料得知，基于平衡树的各类集合操作有强大的函数库实现，其中出名的就是C++的STL中的set库，这个库是用红黑树实现的抽象结构，并采用了各种优化，系统效率十分出色。而本课题要求我们自己实现一个类似set库的数据结构，我们可以参考它的思路，将底层结构迁移到AVL上即可。

## 1.4 系统需求分析

在本系统中，我们需要在应用层维护社交网络中的粉丝集、关注集、好友集等集合关系，需要对各个集合提供合理的抽象数据结构，对于交并补差等基本集合运算提供接口，并在此基础上实现共同好友，共同关注，二度好友，等集合间关系的运算。

同时，因为我们需要AVL作为底层实现数据结构，还需要将AVL树的增删改查作为向上层提供的数据接口。

在用户界面模块，需要提供用户可操作感强，体验良好的交互接口，基本的文本菜单界面。

# 2 系统总体设计

## 2.1 有关数据结构的定义

根据题目要求与个人理解，我将用到的数据结构分为三类：底层结构、抽象层结构、应用层结构。下面给出各种数据的定义：

**底层结构：**

需要操作的最小元数据项为一个人的基本信息，包括姓名（字符串），年龄（整数），性别（整数），等。

AVL树节点结构：包括节点的高度（整数），节点的数据项（struct）、节点的左右孩子（指针）。

**抽象层结构：**

需要抽象出一个集合的数据结构，包括：存放数据的AVL树的根节点指针（指针）、此集合中元素的个数（整数）

**应用层结构：**

在社交网络中一个人的基本信息所形成的数据结构，包括：此人的基本信息（struct）、此人的好友集指针（指针）、此人的粉丝集指针（指针）、此人的关注集指针（指针）。

三层结构为依次依赖关系，抽象层是以底层为基础的，而应用层是根据实际应用需要对抽象层的一此封装。

## 2.2 系统模块结构

本系统是一个应用层系统，其应主要包含一下几个模块：

数据维护模块：利用AVL树维护数据之间的组织关系。

数据查询模块：实现集合的抽象以及在此抽象上数据的结构化查询。

用户UI模块：作为与用户命令交互的窗口，显示查询模块数据，

功能拓展模块：提供一些接口方便进行系统功能的拓展。

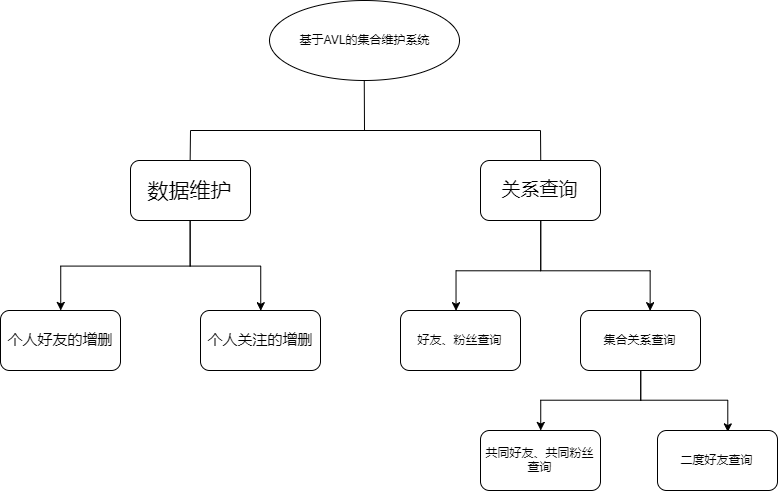


图2-1模块结构图

# 3 数据结构与算法详细设计

## 3.1 底层数据结构设计

**3.1.1 AVL相关函数定义与参数**

void InitAVL(BitTree \*root);

void DestroyAVL(BitTree \*root);

info SearchAVL(BitTree root,char\* x);

int InsertAVL(BitTree \*root,info x);

int DeleteAVL(BitTree \*root,char\* x);

void TraverseAVL(BitTree root);

int getHeight(BitTree root);

void DestroyNode(BitTree \*root);

int Max(int a,int b);

int namebigger(char \*a,char \*b);

int namesmaller(char \*a,char \*b);

void zig(BitTree \*x);

void zig(BitTree \*x);

void zigzag(BitTree \*x);

void zagzig(BitTree \*x);

**3.2.2 主要功能流程图**

添加操作（InsertAVL）：

算法思想：

采用递归的思想，从根节点开始逐层寻找可插入的位置，找到位置后插入节点，同时调整左右子树的树高，利用左旋右旋操作维持树的平衡。

时间复杂度： O（logn）

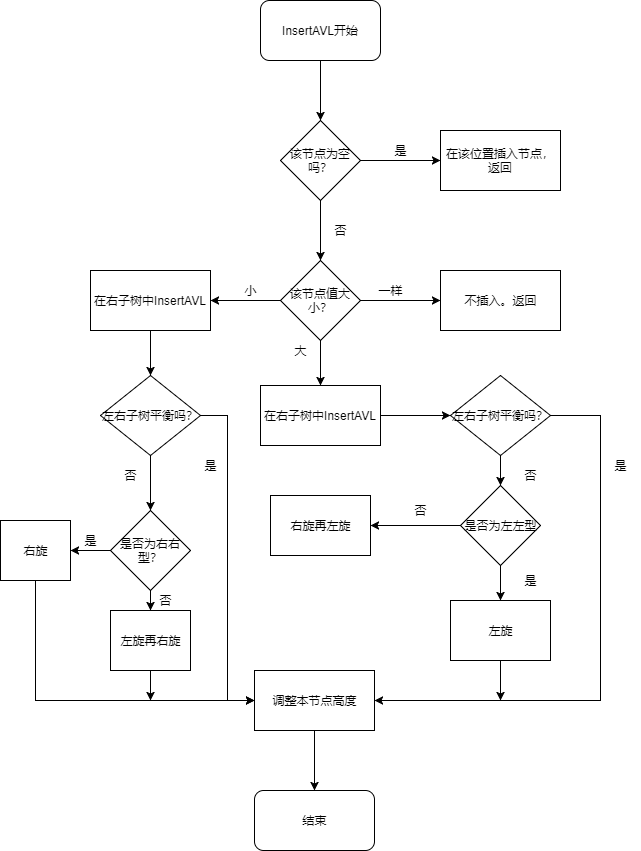


图3-1 添加操作设计流程图

删除操作（DeleteAVL）:

算法思想：

采用递归的思想，从根节点开始逐层寻找要删除的节点，找到位置后删除节点，同时寻找其左右子树新的位置，放到新的位置后调整新位置的树高，利用左旋右旋操作维持树的平衡，在其左右子树中递归调整树高

时间复杂度： O（logn）

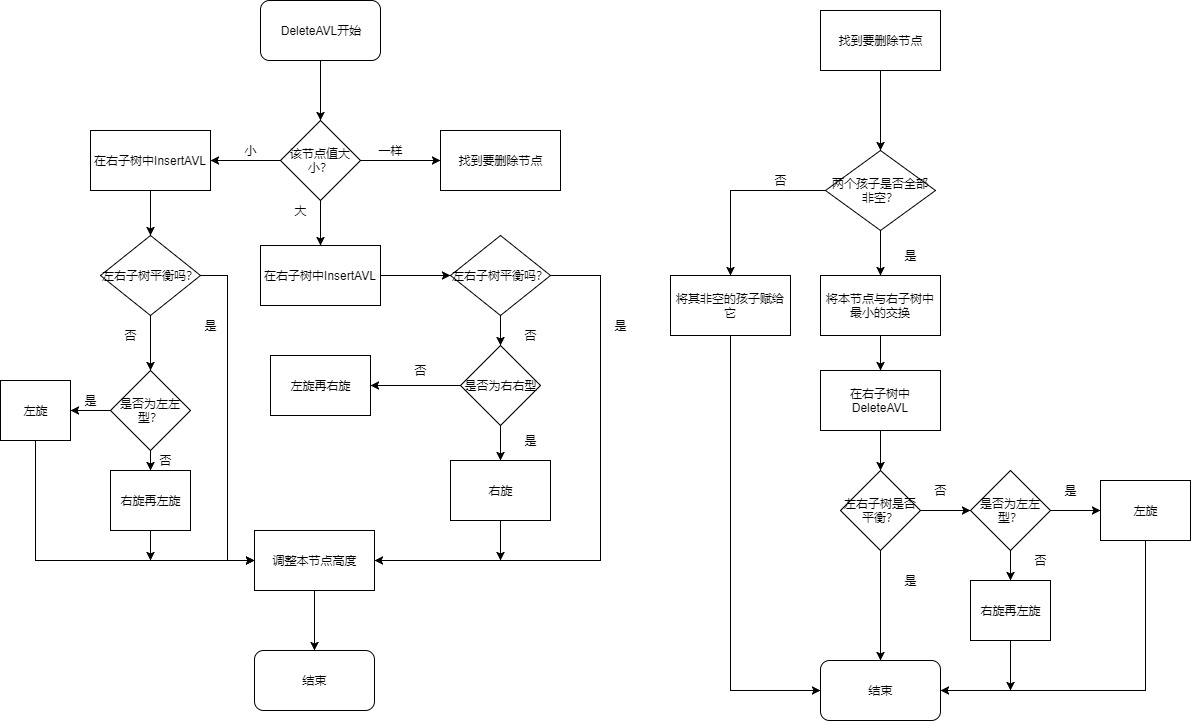


图3-2 删除操作设计流程图

查询操作（SearchAVL）:

算法思想：

采用递归的思想，从根节点开始逐层寻找要查询的节点，比当前节点小则进入左子树，大则进入右子树，相同即找到了。

时间复杂度： O（logn）

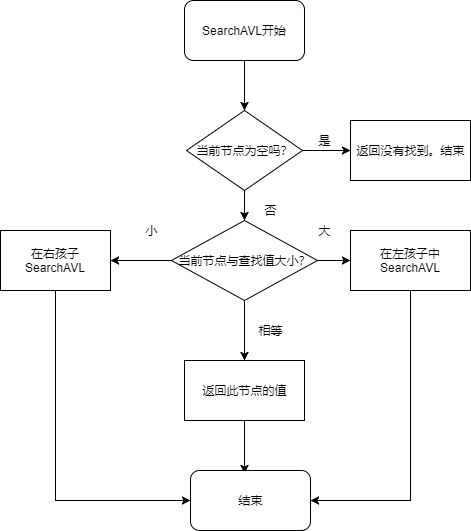


图3-3查询操作设计流程图

## 3.2集合抽象数据结构设计

**3.2.1 抽象数据结构相关函数定义与参数**

void set\_init(set\* group);// 初始化集合

void set\_destroy(set \* group); //销毁集合

int set\_insert(set \* group,info data); //插入元素

int set\_remove(set\* group,info data); //删除元素

set set\_intersection(set groupA ,set groupB); //集合的交

set set\_union(set groupA ,set groupB); //集合的并

set set\_diffrence(set groupA ,set groupB); //集合的差

int set\_size(set group); //集合大小

int set\_member(set group ,info data); //判断一个元素是否为集合成员

int set\_subset(set groupA ,set groupB); //判断一个集合是否为另一个集合的子集

int set\_equal(set groupA ,set groupB); //判断两个集合是否相等

void treemerge(BitTree a,BitTree b,set \* p);//求两棵树的交并插入新的集合中

void treeadd(BitTree a,set \* p);//将一棵树插入新的集合中

void set\_minus(set groupA ,set groupB,set\*p);//在A中除去B中的元素 并放到p中

int tree\_subset(BitTree A,set p);//判断树A中元素是否都在p中

void set\_Traverse(set group);//遍历输出集合

**3.2.2 主要功能流程图**

求集合的交（set\_intersecton）：

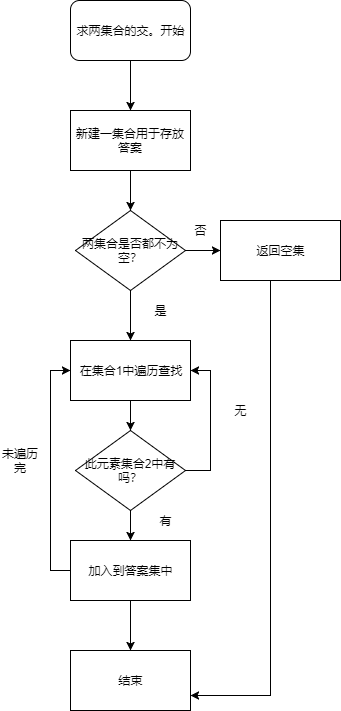


图3-4 交运算流程图

求集合的并（set\_union）：

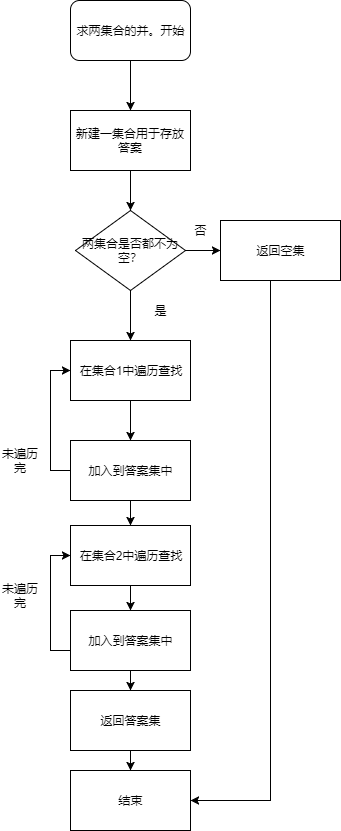


图3-5 并运算流程图

求集合的差（set\_difference）:

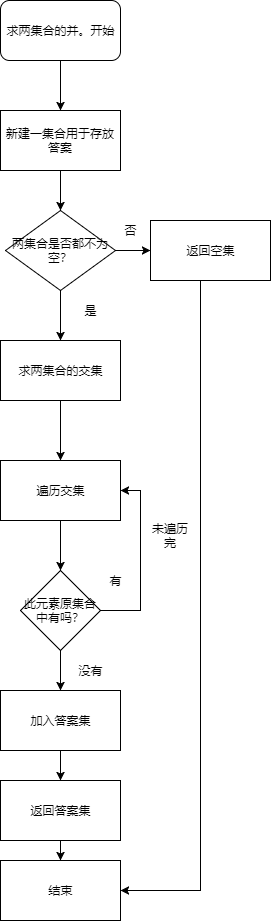


图3-6 集合差运算流程图

## 3.3 应用层结构设计

**3.3.1 抽象数据结构相关函数定义与参数**

void init\_person(person \* ren);//初始化一个人

void destroy\_person(person \*ren);//删除一个人

int add\_friends(person \*a,person \*b);//a,b互相添加为好友

int remove\_friends(person \*a,person \*b);//a,b互相解除好友

int search\_friends(person \*a,person \*b);//查询a，b是否是好友

int add\_fans(person \*a,person \*b);//a成为b的粉丝

int remove\_fans(person \*a,person \*b);//a解除b的粉丝

int search\_fans(person \*a,person \*b);//查询a是不是b的粉丝

int add\_idols(person \*a,person \*b);//a成为b的关注

int remove\_idols(person \*a ,person \*b);//a解除是b的关注

int search\_idols(person \*a,person \*b);//查询a是不是b的关注

set common\_idols(person \*a,person \*b);//求a和b的共同关注

set common\_fans(person \*a,person \*b);//求a和b的共同粉丝

set two\_degree\_friends(person \*a);//求a的二度好友

void printPerson(person a);//打印输出一个人的信息

**3.3.2 主要功能流程图**

求a和b的共同关注（common\_idols）:

图片包含 物体

已生成极高可信度的说明

图3-7 共同关注流程图

求a的二度好友(two\_degree\_friends):

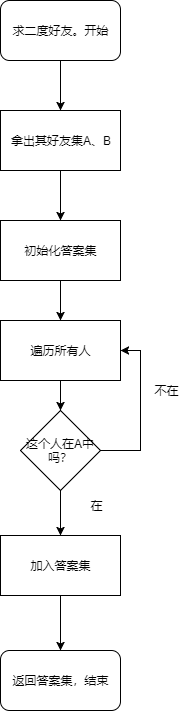


图3-8 二度好友流程图

a,b互相添加为好友(add\_friends):

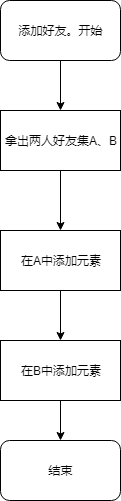


图3-9 添加好友流程图

# 4 系统实现

## 4.1开发环境与相关支持

开发环境：Windows10

编译环境：GCC4.9.2 32-bit

运行环境：Windows10

相关库支持：C标准库、C++标准库

## 4.2 各数据结构C语言描述

**基本数据项：**

typedef struct {

char name[30];

int age;

int gender; //0代表男，1代表女

}info;

**底层AVL节点定义：**

typedef struct \_BitNode{

int height;

info data;

\_BitNode\* lson,\*rson;

}BitNode ,\*BitTree;

**集合抽象结构定义：**

typedef struct{

BitTree avlTree;

int number;

}Set,\*set;

**应用层集合结构定义：**

typedef struct{

info Myinfo;

set Myfriends;

set Myfans;

set Myidols;

//string hobby[5];

}Person,\*person;

## 4.3主要函数说明与复杂度分析

/\*函数功能：在树中查找节点x

参数：树根root,节点名称：x

返回值：空

复杂度：O(logn)

\*/

info SearchAVL(BitTree root,char\* x)

/\*函数功能：给树添加一个元素

参数： 树根，元素

返回值： 1代表添加成功，0代表失败（已存在）

复杂度：O(logn)

\*/

int InsertAVL(BitTree \*root,info x)

/\*函数功能：前序序遍历二叉树

参数：树根

返回值： 无

复杂度：O(n)

\*/

void TraverseAVL(BitTree root)

/\*函数功能：删除二叉树的一个节点

参数：树根root，节点x

返回值： 未找到返回-1，删除成功返回0

复杂度：O(logn)

\*/

int DeleteAVL(BitTree \*root,char\* x)

/\*函数功能： 插入元素

参数： 集合指针 ，元素数据

返回值：-1代表已经存在，1代表插入成功 0代表非法操作

复杂度：O(logn)

\*/

int set\_insert(set \* group,info data)

/\*函数功能： 求两个集合的交集

参数： 集合A，B

返回值：交集 非法操作返回空

复杂度：O(n)

\*/

set set\_intersection(set groupA ,set groupB)

/\*函数功能： 求两集合的差

参数： 集合A,B

返回值：差集 非法操作返回空

复杂度：O(n)

\*/

set set\_diffrence(set groupA ,set groupB)

/\*函数功能： 判断集合A是否是B的子集

参数： 集合A,集合B

返回值：是则为1，不是为0，非法操作为-1;

复杂度：O(n)

\*/

int set\_subset(set groupA ,set groupB)

# 5 系统测试与结果

**数据集：共18个人，73条关注、28对好友，共101条数据，如下图：**

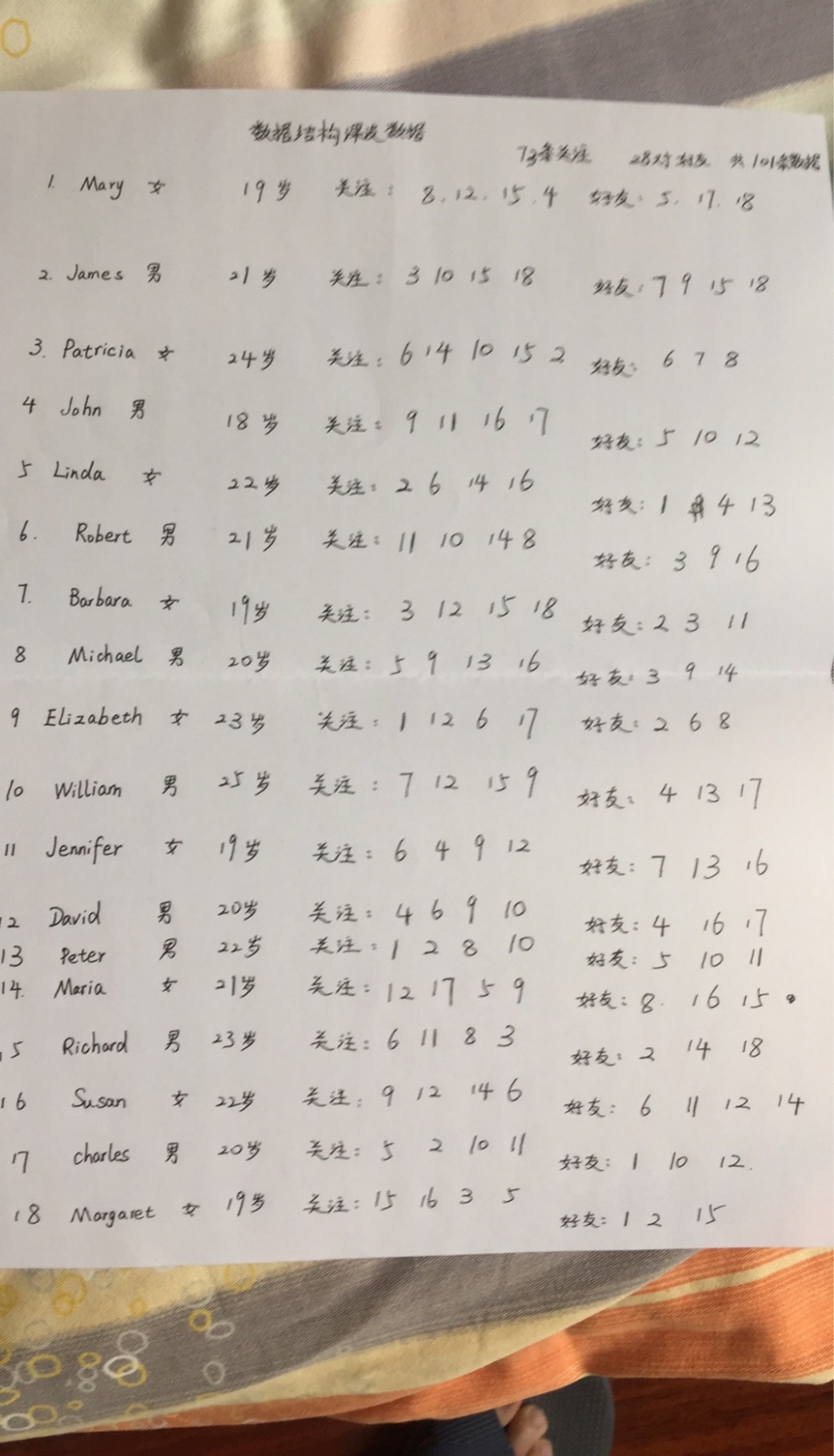
****

图5-1 测试数据总览

**程序主界面：**



图5-2 文本菜单界面图

**功能演示：**

测试功能:用户的添加

测试方法：添加名为Lucy的用户

结果：添加成功。

是否与预期相符：是

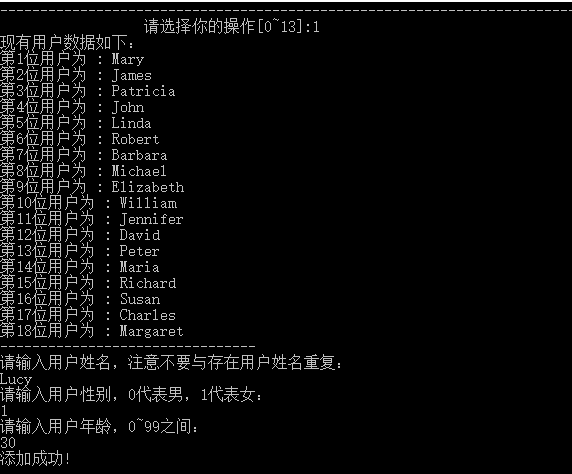


图5-3 插入用户操作结果图

测试功能:用户的删除

测试方法：删除第19号用户

结果：删除成功。

是否与预期相符：是

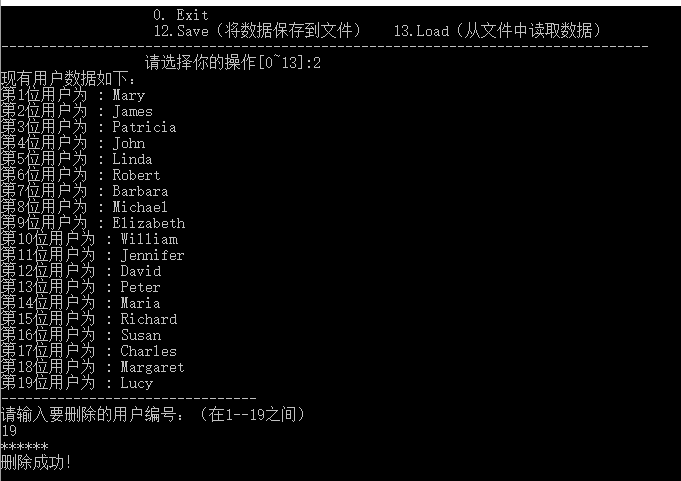


图5-4 删除用户操作结果图

测试功能:用户之间建立好友关系

测试方法：第3号用户与第4号用户建立好友关系

结果：建立成功。

是否与预期相符：是

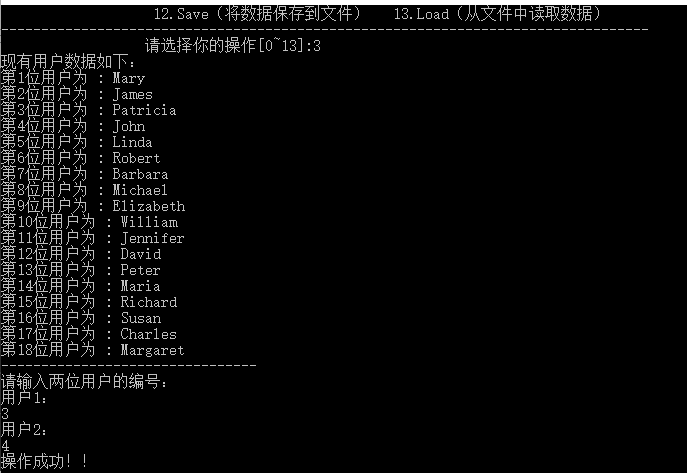


图5-5 建立好友关系结果图

测试功能:求两位用户的共同好友

测试方法：求第6号用户与第9号用户的好友关系

结果：共同好友有3人，Jennifer,David,Susan。

是否与预期相符：是

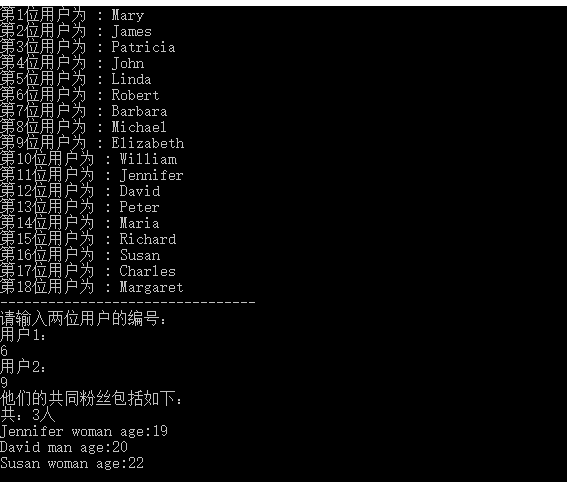


图5-6 查询共同粉丝结果图

测试功能:求两位用户的共同粉丝

测试方法：求第2号用户与第7用户的好友关系

结果：共同关注3人,Patricia，Margaret，Richard。

是否与预期相符：是

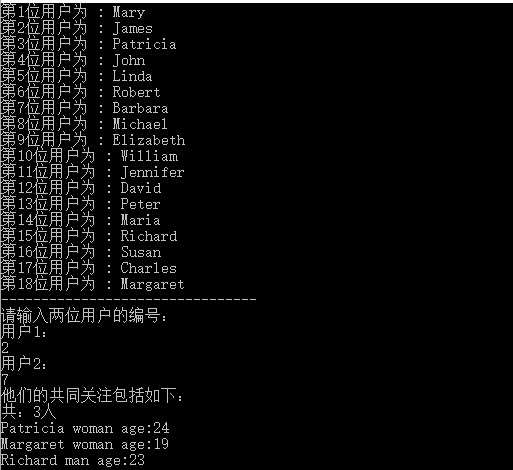


图5-7 查询共同关注结果图

测试功能:求一位用户的二度好友

测试方法：求第12号用户的二度好友

结果：共同好友有6人，如图。

是否与预期相符：是

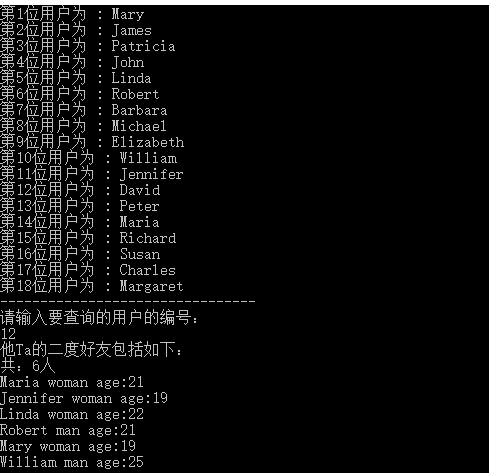


图5-8 查询二度好友结果图

# 6 使用说明

本程序自带一部分数据，运行后进行读取数据操作即可将数据读入程序，完成后可以按照程序内各个操作的说明来进行相关操作，完成后进行写入文件操作，即可将所有现存数据保存以供下次运行时使用，请注意使用时仔细阅读相关提醒！现在版本最多可以添加20名用户，请不要超出限额！

# 7 特色与不足

**特色**：界面简洁清晰、数据量大、处理速度快、对于集合的各类操作支持度非常高，可拓展型强，稳定性高。

**不足之处**：界面不够美观、用户交互性不强、一些功能为硬编码实现、功能不够丰富。

# 8 感悟与体会

在该接触到本次《数据结构》的课程设计时，心里是没有底的，一开始就为了确定题目思考了很久，最后发现第三个题目的逻辑思路比较清晰，实现也有一定思路就选择了第三个题目，然后开始上网查阅相关AVL的资料，查资料的过程中我发现AVL还有很多不同的变种，我在仔细研究了他们的区别后选择了现在用的这个，同时扩增了我的知识面。

整体的实现过程比较顺利，因为我采用的是单元测试的方法，在每完成一个模块的编写后就进行测试，这样可以有效防止最后依赖关系复杂时的调试困难，整体的编码历时一周时间完成，文本菜单界面参考了数据结构实验中使用的界面，总体来说简洁明了，为了不本末倒置，我没有采用Qt图形界面，而利用命令行界面来突出各种功能的实现。

经过本次数据结构课程设计，我进一步掌握了基础数据结构的设计与实现，同时提高了利用所学知识解决实际问题的能力。最后感谢在我完成过程中提供帮助的老师和同学们，以及网络上各类资料的提供者，谢谢！

# 

# 9主要参考文献

[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]cnblog博客：https://blog.csdn.net/silence2015/article/details/50966748

# 附录

#include<iostream>

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

#include<string.h>

#include<algorithm>

using namespace std;

typedef struct {

char name[30];

int age;

int gender; //0代表男，1代表女

}info;

typedef struct \_BitNode{

int height;

info data;

\_BitNode\* lson,\*rson;

}BitNode ,\*BitTree;

void InitAVL(BitTree \*root);

void DestroyAVL(BitTree \*root);

info SearchAVL(BitTree root,char\* x);

int InsertAVL(BitTree \*root,info x);

int DeleteAVL(BitTree \*root,char\* x);

void TraverseAVL(BitTree root);

int getHeight(BitTree root);

void DestroyNode(BitTree \*root);

int Max(int a,int b);

int namebigger(char \*a,char \*b);

int namesmaller(char \*a,char \*b);

void zig(BitTree \*x);

void zig(BitTree \*x);

void zigzag(BitTree \*x);

void zagzig(BitTree \*x);

/\*函数功能：判断两个人是否相同

参数：待比较的两个值

返回值：0代表不相同，1代表相同

\*/

int IsSamePeople(info a,info b){

if(strcmp(a.name,b.name)!=0) return 0;

if(a.age!=b.age) return 0;

if(a.gender!=b.gender) return 0; //当三个数据均相同时才认为两人相同

return 1;

}

/\*函数功能：返回一棵树的树高

参数：树根 root

返回值：如果root为空返回-1，只有一个节点返回0，否则为其子树高度+1

\*/

int getHeight(BitTree root){

if(root!=NULL) return root->height;

return -1;

}

/\*函数功能：求二者的最大值

参数：待比较的两个数

返回值：大的数

\*/

int Max(int a,int b){

if(a>b) return a;

return b;

}

/\*函数功能：判定a串比b串大

参数： a,b字符串

返回值：a比b大则返回1，否则返回0

\*/

int namebigger(char \*a,char \*b){

if(strcmp(a,b)>0) return 1;

else return 0;

}

/\*函数功能：判定a串比b串小

参数： a,b字符串

返回值：a比b小则返回1，否则返回0

\*/

int namesmaller(char \*a,char \*b){

if(strcmp(a,b)<0) return 1;

else return 0;

}

void printNode(info data){

if(data.age==-1){

printf("无此人！！！\n");

return ;

}

printf("%s ",data.name);

if(data.gender==1) printf("woman ");

else printf("man ");

printf("age:%d\n",data.age);

}

/\*函数功能：对单个节点右旋

参数：节点

返回值：空

\*/

void zig(BitTree \*x){

BitTree y=(\*x)->lson;

(\*x)->lson=y->rson;

y->rson=(\*x);

(\*x)->height=Max(getHeight((\*x)->lson),getHeight((\*x)->rson))+1;

y->height=Max(getHeight((\*x)),getHeight(y->lson))+1;

(\*x)=y;

}

/\*函数功能：对单个节点左旋

参数：节点

返回值：空

\*/

void zag(BitTree \*x){

BitTree y=(\*x)->rson;

(\*x)->rson=y->lson;

y->lson=(\*x);

(\*x)->height=Max(getHeight((\*x)->lson),getHeight((\*x)->rson))+1;

y->height=Max(getHeight((\*x)),getHeight(y->rson))+1;

(\*x)=y;

}

/\* 函数功能：对单个节点下的子树完成右旋再左旋

参数：节点

返回值：空

\*/

void zigzag(BitTree \*x){

zig(&((\*x)->rson));

zag(x);

}

/\* 函数功能：对单个节点下的子树完成左旋再右旋

参数：节点

返回值：新的根节点

\*/

void zagzig(BitTree \*x){

zag(&((\*x)->lson));

zig(x);

}

/\*函数功能：在树中查找节点x

参数：树根root,节点名称：x

返回值：空

\*/

info SearchAVL(BitTree root,char\* x){

if(root==NULL){

info\* temp= (info\*)malloc(sizeof(info));

temp->age=-1;

return \*temp;

}

if(namebigger(root->data.name,x)){

return SearchAVL(root->lson,x);

}

else if(namesmaller(root->data.name,x)){

return SearchAVL(root->rson,x);

}

else return root->data;

}

/\*函数功能：给树添加一个元素

参数： 树根，元素

返回值： 1代表添加成功，0代表失败（已存在）

\*/

int InsertAVL(BitTree \*root,info x){

int temp=0;

if((\*root)==NULL){//如果节点为空,就在此节点处加入x信息

(\*root)=(BitTree )malloc(sizeof(BitNode));

(\*root)->data=x;

(\*root)->height=0;

(\*root)->lson=NULL;(\*root)->rson=NULL;

return 1;

}

if(namebigger((\*root)->data.name,x.name)){//如果x小于节点的值,就继续在节点的左子树中插入x

if(InsertAVL(&((\*root)->lson),x)) temp=1;

if(2==getHeight((\*root)->lson)-getHeight((\*root)->rson)){

if(namebigger((\*root)->lson->data.name,x.name))

zig(root);

else

zagzig(root);

}

(\*root)->height=Max(getHeight((\*root)->lson),getHeight((\*root)->rson))+1;

if(temp) return 1;

else return 0;

}

else if(namesmaller((\*root)->data.name,x.name)){//如果x大于节点的值,就继续在节点的右子树中插入x

if(InsertAVL(&((\*root)->rson),x)) temp=1;

//如果高度之差为2的话就失去了平衡,需要旋转

if(2==getHeight((\*root)->rson)-getHeight((\*root)->lson)) {

if(namesmaller((\*root)->rson->data.name,x.name))

zag(root);

else

zigzag(root);

}

(\*root)->height=Max(getHeight((\*root)->lson),getHeight((\*root)->rson))+1;

if(temp) return 1;

else return 0;

}

else

return 0;

}

/\*函数功能：前序序遍历二叉树

参数：树根

返回值： 无

\*/

void TraverseAVL(BitTree root){

if(root==NULL) return;

printNode(root->data);//输出根节点

TraverseAVL(root->lson); //遍历左子树

TraverseAVL(root->rson);//再遍历右子树

}

/\*函数功能：初始化一个二叉树

参数：指向二叉树树根的指针地址

返回值：无

\*/

void InitAVL(BitTree \* root){

(\*root)=NULL;

//(\*root)->height=-1;

//(\*root)->lson=NULL;(\*root)->rson=NULL;

}

/\*函数功能：删除二叉树的一个节点

参数：树根root，节点x

返回值： 未找到返回-1，删除成功返回0

\*/

int DeleteAVL(BitTree \*root,char\* x){

if((\*root)==NULL) return -1;//未找到此节点

if(namebigger((\*root)->data.name,x)){//在左子树内删除节点x

DeleteAVL(&((\*root)->lson),x);

if(2==getHeight((\*root)->rson)-getHeight((\*root)->lson)){

if((\*root)->rson->lson!=NULL&&getHeight((\*root)->rson->lson)>getHeight((\*root)->rson->rson))

zigzag(root);

else zag(root);

}

}

else if(namesmaller((\*root)->data.name,x)){//在右子树内删除节点x

DeleteAVL(&((\*root)->rson),x);

if(2==getHeight((\*root)->lson)-getHeight((\*root)->rson)){

if((\*root)->lson->rson!=NULL&&getHeight((\*root)->lson->rson)>getHeight((\*root)->lson->lson))

zagzig(root);

else zig(root);

}

}

else{//找到节点x

if((\*root)->lson!=NULL&&(\*root)->rson!=NULL){

BitTree temp=(\*root)->rson;

while(temp->lson!=NULL) temp=temp->lson;

(\*root)->data=temp->data;

DeleteAVL(&((\*root)->rson),temp->data.name);

if(2==getHeight((\*root)->lson)-getHeight((\*root)->rson)){

if((\*root)->lson->rson!=NULL&&getHeight((\*root)->lson->rson)>getHeight((\*root)->lson->lson))

zagzig(root);

else

zig(root);

}

}

else{

BitTree temp=\*root;

if((\*root)->lson==NULL)//有右儿子或没有儿子

(\*root)=(\*root)->rson;

else if((\*root)->rson=NULL)//有左儿子

(\*root)=(\*root)->lson;

free(temp);

}

}

if((\*root)==NULL) return 1;

(\*root)->height=Max(getHeight((\*root)->lson),getHeight((\*root)->rson))+1;

return 1;

}

/\*函数功能：对二叉树节点递归删除

参数：根节点

返回值：空

\*/

void DestroyNode(BitTree \*root){

if(\*root==NULL) return ;

if((\*root)->lson!=NULL) DestroyNode(&(\*root)->lson);

if((\*root)->rson!=NULL) DestroyNode(&(\*root)->rson);

free(\*root);

}

/\*函数功能：销毁二叉树

参数：指向二叉树树根的指针地址

返回值：空

\*/

void DestroyAVL(BitTree \*root){

DestroyNode(root);

\*root=NULL;

}

typedef struct{

BitTree avlTree;

int number;

}Set,\*set;

set setList[10];

void set\_init(set\* group);// 初始化集合

void set\_destroy(set \* group); //销毁集合

int set\_insert(set \* group,info data); //插入元素

int set\_remove(set\* group,info data); //删除元素

set set\_intersection(set groupA ,set groupB); //集合的交

set set\_union(set groupA ,set groupB); //集合的并

set set\_diffrence(set groupA ,set groupB); //集合的差

int set\_size(set group); //集合大小

int set\_member(set group ,info data); //判断一个元素是否为集合成员

int set\_subset(set groupA ,set groupB); //判断一个集合是否为另一个集合的子集

int set\_equal(set groupA ,set groupB); //判断两个集合是否相等

void treemerge(BitTree a,BitTree b,set \* p);//求两棵树的交并插入新的集合中

void treeadd(BitTree a,set \* p);//将一棵树插入新的集合中

void set\_minus(set groupA ,set groupB,set\*p);//在A中除去B中的元素 并放到p中

int tree\_subset(BitTree A,set p);//判断树A中元素是否都在p中

void set\_Traverse(set group);//遍历输出集合

/\*函数功能： 初始化集合

参数： 集合指针

返回值：空

\*/

void set\_init(set\* group){

(\*group)= (set) malloc(sizeof(Set));

(\*group)->avlTree=NULL;

InitAVL(&((\*group)->avlTree));

(\*group)->number=0;

}

/\*函数功能： 销毁集合

参数： 集合指针

返回值： 空

\*/

void set\_destroy(set \* group){

DestroyAVL(&((\*group)->avlTree));

free(\*group);

\*group=NULL;

}

/\*函数功能： 插入元素

参数： 集合指针 ，元素数据

返回值：-1代表已经存在，1代表插入成功 0代表非法操作

\*/

int set\_insert(set \* group,info data){

if((\*group)==NULL) return 0;

if(InsertAVL(&((\*group)->avlTree),data)==1) {

(\*group)->number++;

return 1;

}

else

return -1;

}

/\*函数功能： 删除元素

参数： 集合指针 数据元素

返回值：-1代表元素不存在，1代表删除成功 0代表非法操作

\*/

int set\_remove(set\* group,info data){

if((\*group)==NULL) return 0;

if(DeleteAVL(&((\*group)->avlTree),data.name)) {

(\*group)->number--;

return 1;

}

else return -1;

}

/\*函数功能：求两棵树的交集

参数： 树a，b 集合p

返回值：空

\*/

void treemerge(BitTree a,BitTree b,set \* p){

if(a==NULL) return ;

info temp;

temp=SearchAVL(b,a->data.name);

if((temp.age!=-1)) set\_insert(p,temp);

treemerge(a->lson,b,p);

treemerge(a->rson,b,p);

}

/\*函数功能： 将一棵树中元素加入到集合中

参数： 树a，集合p

返回值：空

\*/

void treeadd(BitTree a,set \* p){

if(a==NULL) return ;

set\_insert(p,a->data);

treeadd(a->lson,p);

treeadd(a->rson,p);

}

/\*函数功能： 求两个集合的交集

参数： 集合A，B

返回值：交集 非法操作返回空

\*/

set set\_intersection(set groupA ,set groupB){

set temp; set\_init(&temp);

if(groupA==NULL||groupB==NULL){

set\_destroy(&temp);

return temp;

}

BitTree treeA =groupA->avlTree;

BitTree treeB = groupB->avlTree;

treemerge(treeA,treeB,&temp);

return temp;

}

/\*函数功能： 求两集合的并

参数： 集合A，B

返回值：并集 非法操作返回空

\*/

set set\_union(set groupA ,set groupB){

set temp; set\_init(&temp);

if(groupA==NULL||groupB==NULL){

set\_destroy(&temp);

return temp;

}

BitTree treeA =groupA->avlTree;

BitTree treeB = groupB->avlTree;

treeadd(treeA,&temp);

treeadd(treeB,&temp);

return temp;

}

/\*函数功能： 将两棵树中不一样的元素放到新集合中

参数： 树A,B 集合P

返回值：空

\*/

void set\_minus(BitTree A ,BitTree B,set\* p){

if(A==NULL) return;

info temp;temp=SearchAVL(B,A->data.name);

if(temp.age==-1) set\_insert(p,A->data);

set\_minus(A->lson,B,p);

set\_minus(A->rson,B,p);

}

/\*函数功能： 求两集合的差

参数： 集合A,B

返回值：差集 非法操作返回空

\*/

set set\_diffrence(set groupA ,set groupB){

set crossgroup=set\_intersection(groupA,groupB);

set\_Traverse(crossgroup);

set temp; set\_init(&temp);

if(groupA==NULL||crossgroup==NULL){

set\_destroy(&temp);

return temp;

}

BitTree treeA =groupA->avlTree;

BitTree treeB = crossgroup->avlTree;

set\_minus(treeA,treeB,&temp);

return temp;

}

/\*函数功能： 返回集合的大小

参数： 集合

返回值：集合大小 ,非法操作为-1;

\*/

int set\_size(set group){

if(group==NULL) return -1;

return group->number;

}

/\*函数功能： 判断元素是否为集合成员

参数： 集合group 元素data

返回值：是为1，否为0，非法操作为-1

\*/

int set\_member(set group ,info data){

if (group==NULL) return -1;

if(SearchAVL(group->avlTree,data.name).age!=-1) return 1;

else return 0;

}

/\*函数功能： 判断集合A是否是B的子集

参数： 集合A,集合B

返回值：是则为1，不是为0;

\*/

int tree\_subset(BitTree A,set p){

if(A==NULL) return 1;

if(SearchAVL(p->avlTree,A->data.name).age==-1) return 0;

else{

return tree\_subset(A->lson,p)&tree\_subset(A->rson,p);

}

}

/\*函数功能： 判断集合A是否是B的子集

参数： 集合A,集合B

返回值：是则为1，不是为0，非法操作为-1;

\*/

int set\_subset(set groupA ,set groupB){

if(groupA==NULL||groupB==NULL) return -1;

if(tree\_subset(groupA->avlTree,groupB)) return 1;

else return 0;

}

/\*函数功能： 判断两个集合是否相等

参数： 集合A，B

返回值：相等为1，否则为0，非法操作为-1

\*/

int set\_equal(set groupA ,set groupB){

if(groupA==NULL||groupB==NULL) return -1;

if(set\_subset(groupA,groupB)==1&&groupA->number==groupB->number) return 1;

else return 0;

}

/\*函数功能： 遍历输出一个集合

参数： 集合group

返回值：空

\*/

void set\_Traverse(set group){

printf("共：%d人\n",group->number);

TraverseAVL(group->avlTree);

//printf("\*\*\*\*\*\*完毕\*\*\*\*\*\*\*\n");

}

typedef struct{ //定义应用层结构：人

info Myinfo;

set Myfriends;

set Myfans;

set Myidols;

}Person,\*person;

void init\_person(person \* ren);//初始化一个人

void destroy\_person(person \*ren);//删除一个人

int add\_friends(person \*a,person \*b);//a,b互相添加为好友

int remove\_friends(person \*a,person \*b);//a,b互相解除好友

int search\_friends(person \*a,person \*b);//查询a，b是否是好友

int add\_fans(person \*a,person \*b);//a成为b的粉丝

int remove\_fans(person \*a,person \*b);//a解除b的粉丝

int search\_fans(person \*a,person \*b);//查询a是不是b的粉丝

int add\_idols(person \*a,person \*b);//a成为b的关注

int remove\_idols(person \*a ,person \*b);//a解除是b的关注

int search\_idols(person \*a,person \*b);//查询a是不是b的关注

set common\_idols(person \*a,person \*b);//求a和b的共同关注

set common\_fans(person \*a,person \*b);//求a和b的共同粉丝

set two\_degree\_friends(person \*a);//求a的二度好友

void printPerson(person a);//打印输出一个人的信息

/\*函数功能： 初始化一个人

参数： 结构指针 a

返回值： 空

\*/

void init\_person(person \*a){

if((\*a)!=NULL) destroy\_person(a);

(\*a)= (person)malloc(sizeof(Person));

set\_init(&((\*a)->Myfriends));

set\_init(&((\*a)->Myfans));

set\_init(&((\*a)->Myidols));

}

/\*函数功能： 销毁一个人

参数： 结构指针 a

返回值： 空

\*/

void destroy\_person(person \*ren){

if((\*ren)==NULL) return ;

set\_destroy(&((\*ren)->Myfriends));

set\_destroy(&((\*ren)->Myfans));

set\_destroy(&((\*ren)->Myidols));

free(\*ren);

}

/\*函数功能： 添加两人互为朋友关系

参数： 两个人a，b

返回值： 添加成功为1，非法为-1

\*/

int add\_friends(person \*a,person \*b){

if((\*a)==NULL||(\*b)==NULL) return -1;

if(set\_insert(&((\*a)->Myfriends),(\*b)->Myinfo)==1&&set\_insert(&((\*b)->Myfriends),(\*a)->Myinfo)==1) return 1;

else return 0;

}

/\*函数功能： 解除两人的好友关系

参数： 两个人a,b

返回值： 解除成功为1，解除失败为0，非法操作为-1

\*/

int remove\_friends(person \*a,person \*b){

if((\*a)==NULL||(\*b)==NULL) return -1;

if(search\_friends(a,b)!=1) return 0;

if(set\_remove(&((\*a)->Myfriends),(\*b)->Myinfo)==1&&set\_remove(&((\*b)->Myfriends),(\*a)->Myinfo)==1) return 1;

else return 0;

}

/\*函数功能： 查找两人是否为好友关系

参数： 两个人a,b

返回值： 是则为1，否为0，非法为-1

\*/

int search\_friends(person \*a,person \*b){

if((\*a)==NULL||(\*b)==NULL) return -1;

if(set\_member((\*a)->Myfriends,(\*b)->Myinfo)==1) return 1;

else return 0;

}

/\*函数功能： 添加a为b的粉丝

参数： 两个人a,b

返回值： 添加成功为1，失败为0，非法为-1

\*/

int add\_fans(person \*a,person \*b){

if((\*a)==NULL||(\*b)==NULL) return -1;

if(set\_insert(&((\*b)->Myfans),(\*a)->Myinfo)==1&&set\_insert(&((\*a)->Myidols),(\*b)->Myinfo)==1) return 1;

else return 0;

}

/\*函数功能： 解除a是b的粉丝

参数： 两个人a,b

返回值： 解除成功为1，失败为0，非法为-1

\*/

int remove\_fans(person \*a,person \*b){

if((\*a)==NULL||(\*b)==NULL) return -1;

if(search\_fans(a,b)!=1) return 0;

if(set\_remove(&((\*b)->Myfans),(\*a)->Myinfo)==1&&set\_remove(&((\*a)->Myidols),(\*b)->Myinfo)==1) return 1;

else return 0;

}

/\*函数功能： 查询a是不是b的粉丝

参数： 两个人a,b

返回值： 是则返回，否则返回0，非法为-1

\*/

int search\_fans(person \*a,person \*b){

if((\*a)==NULL||(\*b)==NULL) return -1;

if(set\_member((\*b)->Myfans,(\*a)->Myinfo)==1) return 1;

else return 0;

}

/\*函数功能： 添加a为b的关注

参数： 两个人a,b

返回值： 添加成功为1，失败为0，非法为-1

\*/

int add\_idols(person \*a,person \*b){

if((\*a)==NULL||(\*b)==NULL) return -1;

if(set\_insert(&((\*b)->Myidols),(\*a)->Myinfo)==1&&set\_insert(&((\*a)->Myfans),(\*b)->Myinfo)==1) return 1;

else return 0;

}

/\*函数功能： 解除a是b的关注

参数： 两个人a,b

返回值： 解除成功为1，失败为0，非法为-1

\*/

int remove\_idols(person \*a ,person \*b){

if((\*a)==NULL||(\*b)==NULL) return -1;

if(search\_idols(a,b)!=1) return 0;

if(set\_remove(&((\*b)->Myidols),(\*a)->Myinfo)==1&&set\_remove(&((\*a)->Myfans),(\*b)->Myinfo)==1) return 1;

else return 0;

}

/\*函数功能： 查询a是不是b的关注

参数： 两个人a,b

返回值： 是则返回，否则返回0，非法为-1

\*/

int search\_idols(person \*a,person \*b){

if((\*a)==NULL||(\*b)==NULL) return -1;

if(set\_member((\*b)->Myidols,(\*a)->Myinfo)==1) return 1;

else return 0;

}

/\*函数功能： 求两个人的共同关注

参数： 两个人a,b

返回值： 共同关注的集合

\*/

set common\_idols(person \*a,person \*b){

return set\_intersection((\*a)->Myidols,(\*b)->Myidols);

}

/\*函数功能： 求两个人的共同粉丝

参数： 两个人a,b

返回值： 共同粉丝的集合

\*/

set common\_fans(person \*a,person \*b){

return set\_intersection((\*a)->Myfans,(\*b)->Myfans);

}

person personList[20];

/\*函数功能： 求一个人的二度好友

参数： 一个人a

返回值： a的所有二度好友构成的集合

\*/

set two\_degree\_friends(person \*a){

set temp; set\_init(&temp);

set one\_degree\_friendss=(\*a)->Myfriends;

for(int i=0;i<20;i++){

if(personList[i]==NULL)

continue;

if(search\_friends(&personList[i],a)==1) {

temp=set\_union(temp,personList[i]->Myfriends);

}

}

set\_remove(&temp,(\*a)->Myinfo);

return temp;

}

/\*函数功能： 输出一个人

参数： 一个人a

返回值： void

\*/

void printPerson(person a){

if(a==NULL) return ;

printf("名字是：%s 性别是：",a->Myinfo.name);

if(a->Myinfo.gender==0) printf("男 ");

else printf("女 ");

printf("年龄是：%d\n",a->Myinfo.age);

if(a->Myfriends->number>0){

printf("Ta的好友包括：\n");

set\_Traverse(a->Myfriends);

}

else printf("Ta没有好友\n");

if(a->Myfans->number>0){

printf("Ta的粉丝包括：\n");

set\_Traverse(a->Myfans);

}

else printf("Ta没有粉丝\n");

if(a->Myidols->number>0){

printf("Ta的关注包括：\n");

set\_Traverse(a->Myidols);

}

else printf("Ta没有关注\n");

printf("\*\*\*\*\*\*完毕\*\*\*\*\*\*\*\n");

}

int op=1;

/\*函数功能： 打印现有成员信息

参数： void

返回值： 现有成员个数

\*/

int print\_now\_person(){

int ans=0;

for(int i=0;i<20;i++){

if(personList[i]==NULL) continue;

ans=i+1;

printf("第%d位用户为 : %s\n",ans,personList[i]->Myinfo.name);

}

if(ans==0){

printf("当前还无用户！！\n");

}

return ans;

}

/\*函数功能： 添加一位用户

参数： void

返回值： void

\*/

void Insert\_A\_Person(){

printf("现有用户数据如下：\n");

int now=print\_now\_person();

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

info temp;

printf("请输入用户姓名，注意不要与存在用户姓名重复：\n");

scanf("%s",temp.name);

printf("请输入用户性别，0代表男，1代表女：\n");

scanf("%d",&temp.gender);

printf("请输入用户年龄，0~99之间：\n");

scanf("%d",&temp.age);

for(int i=0;i<20;i++){

if(personList[i]==NULL) continue;

if(strcmp(personList[i]->Myinfo.name,temp.name)==0){

printf("输入已存在，请重新操作\n");

return ;

}

}

if(temp.gender!=0&&temp.gender!=1){

printf("性别输入错误，请重新操作\n");

return ;

}

if(temp.age<0||temp.age>99){

printf("年龄输入错误，请重新操作\n");

return ;

}

init\_person(&personList[now]);

personList[now]->Myinfo=temp;

printf("添加成功!\n");

}

/\*函数功能： 删除一位用户

参数： void

返回值： void

\*/

void Delete\_A\_Person(){

printf("现有用户数据如下：\n");

int now=print\_now\_person();

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

int del;

printf("请输入要删除的用户编号：（在1--%d之间）\n",now);

scanf("%d",&del);

if(del<1||del>now){

printf("编号输入有误，请重新操作\n");

return ;

}

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

if(i+1==del) continue;

if(search\_fans(&personList[del-1],&personList[i])==1) remove\_fans(&personList[del-1],&personList[i]);

if(search\_friends(&personList[del-1],&personList[i])==1) remove\_friends(&personList[del-1],&personList[i]);

if(search\_idols(&personList[del-1],&personList[i])==1) remove\_idols(&personList[del-1],&personList[i]);

}printf("\*\*\*\*\*\*\n");

destroy\_person(&personList[del-1]);

for(int i=del;i<now;i++)

personList[i-1]=personList[i];

personList[now-1]=NULL;

printf("删除成功！\n");

}

/\*函数功能： 增添两个人的好友关系

参数： void

返回值： void

\*/

void Make\_Friends(){

printf("现有用户数据如下：\n");

int now=print\_now\_person();

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

int per1,per2;

printf("请输入两位用户的编号：\n用户1：\n");

scanf("%d",&per1);

printf("用户2：\n");

scanf("%d",&per2);

if(per1<1||per1>now||per2<1||per2>now){

printf("编号输入有误，请重新操作\n");

return ;

}

if(search\_friends(&personList[per1-1],&personList[per2-1])==1){

printf("这两位已经是好友了！！\n");

return ;

}

else{

if(add\_friends(&personList[per1-1],&personList[per2-1])==1) printf("操作成功！！\n");

else printf("操作失败！！\n");

return ;

}

}

/\*函数功能： 删除两个人的好友关系

参数： void

返回值： void

\*/

void Remove\_Friends(){

printf("现有用户数据如下：\n");

int now=print\_now\_person();

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

int per1,per2;

printf("请输入两位用户的编号：\n用户1：\n");

scanf("%d",&per1);

printf("用户2：\n");

scanf("%d",&per2);

if(per1<1||per1>now||per2<1||per2>now){

printf("编号输入有误，请重新操作\n");

return ;

}

if(search\_friends(&personList[per1-1],&personList[per2-1])!=1){

printf("这两位还不是好友！！\n");

return ;

}

else{

if(remove\_friends(&personList[per1-1],&personList[per2-1])==1) printf("操作成功！！\n");

else printf("操作失败！！\n");

return ;

}

}

/\*函数功能： 查询两个人是不是好友

参数： void

返回值： void

\*/

void Search\_Friends(){

printf("现有用户数据如下：\n");

int now=print\_now\_person();

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

int per1,per2;

printf("请输入两位用户的编号：\n用户1：\n");

scanf("%d",&per1);

printf("用户2：\n");

scanf("%d",&per2);

if(per1<1||per1>now||per2<1||per2>now){

printf("编号输入有误，请重新操作\n");

return ;

}

if(search\_friends(&personList[per1-1],&personList[per2-1])!=1){

printf("这两位还不是好友！！\n");

return ;

}

else{

printf("这两位确实是好友！！\n");

return ;

}

}

/\*函数功能： 添加一个人为另一个的粉丝

参数： void

返回值： void

\*/

void Add\_A\_Fans(){

printf("现有用户数据如下：\n");

int now=print\_now\_person();

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

int per1,per2;

printf("请输入两位用户的编号：\n用户1：（粉丝）\n");

scanf("%d",&per1);

printf("用户2：（偶像）\n");

scanf("%d",&per2);

if(per1<1||per1>now||per2<1||per2>now){

printf("编号输入有误，请重新操作\n");

return ;

}

if(search\_fans(&personList[per1-1],&personList[per2-1])==1){

printf("Ta已经是Ta的粉丝了！！\n");

return ;

}

else{

if(add\_fans(&personList[per1-1],&personList[per2-1])==1) printf("操作成功！！\n");

else printf("操作失败！！\n");

return ;

}

}

/\*函数功能： 解除一个人为另一个的粉丝

参数： void

返回值： void

\*/

void Remove\_Fans(){

printf("现有用户数据如下：\n");

int now=print\_now\_person();

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

int per1,per2;

printf("请输入两位用户的编号：\n用户1：（粉丝）\n");

scanf("%d",&per1);

printf("用户2：（偶像）\n");

scanf("%d",&per2);

if(per1<1||per1>now||per2<1||per2>now){

printf("编号输入有误，请重新操作\n");

return ;

}

if(search\_fans(&personList[per1-1],&personList[per2-1])!=1){

printf("Ta还不是Ta的粉丝！！\n");

return ;

}

else{

if(remove\_fans(&personList[per1-1],&personList[per2-1])==1) printf("操作成功！！\n");

else printf("操作失败！！\n");

return ;

}

}

/\*函数功能： 查询一个人是不是另一个的粉丝

参数： void

返回值： void

\*/

void Search\_Fans(){

printf("现有用户数据如下：\n");

int now=print\_now\_person();

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

int per1,per2;

printf("请输入两位用户的编号：\n用户1：（粉丝）\n");

scanf("%d",&per1);

printf("用户2：（偶像）\n");

scanf("%d",&per2);

if(per1<1||per1>now||per2<1||per2>now){

printf("编号输入有误，请重新操作\n");

return ;

}

if(search\_fans(&personList[per1-1],&personList[per2-1])!=1){

printf("Ta还不是Ta的粉丝！！\n");

return ;

}

else{

printf("Ta确实是Ta的粉丝！！\n");

return ;

}

}

/\*函数功能： 查询两个人的共同粉丝

参数： void

返回值： void

\*/

void Commom\_Fans(){

printf("现有用户数据如下：\n");

int now=print\_now\_person();

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

int per1,per2;

printf("请输入两位用户的编号：\n用户1：\n");

scanf("%d",&per1);

printf("用户2：\n");

scanf("%d",&per2);

if(per1<1||per1>now||per2<1||per2>now){

printf("编号输入有误，请重新操作\n");

return ;

}

printf("他们的共同粉丝包括如下：\n");

set\_Traverse(common\_fans(&personList[per1-1],&personList[per2-1]));

return ;

}

/\*函数功能： 查询两个人的共同关注

参数： void

返回值： void

\*/

void Commom\_Idols(){

printf("现有用户数据如下：\n");

int now=print\_now\_person();

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

int per1,per2;

printf("请输入两位用户的编号：\n用户1：\n");

scanf("%d",&per1);

printf("用户2：\n");

scanf("%d",&per2);

if(per1<1||per1>now||per2<1||per2>now){

printf("编号输入有误，请重新操作\n");

return ;

}

printf("他们的共同关注包括如下：\n");

set\_Traverse(common\_idols(&personList[per1-1],&personList[per2-1]));

return ;

}

/\*函数功能： 查询两一个人的二度好友

参数： void

返回值： void

\*/

void TwoDFriends(){

printf("现有用户数据如下：\n");

int now=print\_now\_person();

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

int per1;

printf("请输入要查询的用户的编号：\n");

scanf("%d",&per1);

if(per1<1||per1>now){

printf("编号输入有误，请重新操作\n");

return ;

}

printf("他Ta的二度好友包括如下：\n");

set\_Traverse(two\_degree\_friends(&personList[per1-1]));

return ;

}

/\*函数功能： 将数据写入到文件中

参数： void

返回值： void

\*/

void SaveToFile(){

FILE\* fp ;

info temp;

temp.age=-1;

if ((fp=fopen("data.txt","w"))==NULL){

printf("File open error\n ");

return ;

}

int ans=0;

for(int i=0;i<20;i++){

if(personList[i]==NULL) continue;

ans=i+1;

fwrite(&(personList[i]->Myinfo),sizeof(info),1,fp);

}

int zero=0;

fwrite((&temp),sizeof(info),1,fp);

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

for(int j=1;j<=ans;j++){

if(search\_friends(&personList[i-1],&personList[j-1])==1) {

fwrite(&i,sizeof(int),1,fp);

fwrite(&j,sizeof(int),1,fp);

}

}

fwrite(&zero,sizeof(int),1,fp);

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

for(int j=1;j<=ans;j++){

if(search\_fans(&personList[i-1],&personList[j-1])==1) {

fwrite(&i,sizeof(int),1,fp);

fwrite(&j,sizeof(int),1,fp);

}

}

fwrite(&zero,sizeof(int),1,fp);

fclose(fp);

printf("已成功保存！！！\n");

}

/\*函数功能： 从文件中读取数据

参数： void

返回值： void

\*/

void LoadFromFile(){

FILE\* fp ;

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

printf("File open error\n ");

return ;

}

int ans=0;

init\_person(&personList[ans]);

while(fread(&(personList[ans]->Myinfo),sizeof(info),1,fp)){

if(personList[ans]->Myinfo.age==-1) {

destroy\_person(&personList[ans]);

personList[ans]=NULL;

break;

}

ans++;

init\_person(&personList[ans]);

}

int a,b;

while(fread(&a,sizeof(int),1,fp)){

if(a==0) break;

fread(&b,sizeof(int),1,fp);

add\_friends(&personList[a-1],&personList[b-1]);

}

while(fread(&a,sizeof(int),1,fp)){

if(a==0) break;

fread(&b,sizeof(int),1,fp);

add\_fans(&personList[a-1],&personList[b-1]);

}

fclose(fp);

printf("读取成功！！！\n");

}

int main(){

while(op){

system("cls"); printf("\n\n");

printf(" 微博关注、粉丝、好友集管理系统 \n");

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

printf(" 1. InsertPerson（新增一名用户） \n");

printf(" 2. DeletePerson（删除一名用户） \n");

printf(" 3. MakeFriend （两名用户建立好友关系） \n");

printf(" 4. ReliveFriend（两名用户解除好友关系） \n");

printf(" 5. SearchFriend（查询两名用户是否为好友关系） \n");

printf(" 6. Add\_A\_Fans （一名用户成为另一名用户的粉丝） \n");

printf(" 7. Relive\_AFans（一名用户取消成为另一名的粉丝） \n");

printf(" 8. SearchFans （查询一名用户是否是另一名的粉丝） \n");

printf(" 9. Common\_Fans （查询两个用户的共同粉丝） \n");

printf(" 10. Commom\_idols（查询两个用户的共同关注） \n");

printf(" 11. Friend's\_Friends（查询一位用户的二度好友） \n");

printf(" 0. Exit\n");

printf(" 12.Save（将数据保存到文件） 13.Load（从文件中读取数据）\n");

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

printf(" 请选择你的操作[0~13]:");

scanf("%d",&op);

switch(op){

case 1:

Insert\_A\_Person();

getchar();getchar();

break;

case 2:

Delete\_A\_Person();

getchar();getchar();

break;

case 3:

Make\_Friends();

getchar();getchar();

break;

case 4:

Remove\_Friends();

getchar();getchar();

break;

case 5:

Search\_Friends();

getchar();getchar();

break;

case 6:

Add\_A\_Fans();

getchar();getchar();

break;

case 7:

Remove\_Fans();

getchar();getchar();

break;

case 8:

Search\_Fans();

getchar();getchar();

break;

case 9:

Commom\_Fans();

getchar();getchar();

break;

case 10:

Commom\_Idols();

getchar();getchar();

break;

case 11:

TwoDFriends();

getchar();getchar();

break;

case 12:

SaveToFile();

getchar();getchar();

break;

case 13:

LoadFromFile();

getchar();getchar();

break;

case 0:

printf("感谢使用本系统！！\n");

break;

}//end of switch

}

return 0;

}