華中科技大學

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：U201614890

姓　　名：徐光磊

指导教师：李剑军

报告日期：2018 年 3 月 5 日

计算机科学与技术学院

任 务 书

□ 设计内容

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

□ 设计要求

（1）交互式操作界面(并非一定指图形式界面)；

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

（3）基于 AVL 表示及调用其 6 种基本运算实现集合 ADT 的基本运算：初始化 set_init，销毁 set_destroy，插入 set_insert，删除 set_remove，交 set_intersection，并 set_union，差 set_diffrence，成员个数 set_size，判断元素是否为集合成员的查找 set_member，判断是否为子集 set_subset，判断集合是否相等 set_equal；

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

（5）主要数据对象的数据文件组织与存储。

□ 参考文献

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

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

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

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

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

目录

1. 引言

1.1 课题背景与意义

本次课程设计内容是基于 AVL 自平衡二叉树来实现集合的运算，并以此为基础实现一个简单的社交网络数据管理系统。如集合这样的离散量的运算在计算机学科中十分常见，如何通过合理的数据结构设计，来实现高性能的大数据量吞吐处理成为了一个非常关键的问题。

1.2 国内外研究现状

目前大部分数据库系统及文件系统都采用 B-Tree 或其变种 B+Tree 作为索引结构，来提供大数据量下针对 HDD 的优化数据结构。拿 MySQL 来进行举例，其中原生提供了并集运算，但对于交、差等运算则没有较为高性能的支持。对于研发实力强劲的公司来说，会对某个应用场景再进行针对优化。如 Twitter 对于社交网络场景下开发的 Flockdb，基于图来实现高速 CURD 操作和超大规模邻接矩阵查询等功能。

1.3 课程设计的主要研究工作

1. 查阅了 Github 等网络社区上开源的简易社交网络系统的源代码和文档。
2. 查阅了《Algorithem 4th Edition》，对 AVL 等自平衡树的实现方式和性质特征有了进一步掌握。

2. 系统需求分析与总体设计

2.1 系统需求分析

使用 AVL 自平衡二叉树实现集合抽象数据的多种基本运算，并以此为基础搭建简单的社交网络数据管理系统，实现好友集、粉丝集、关注人集，共同关注、共同喜好、二度好友等查询功能，同时保证一定的性能。

2.2 系统总体设计

系统设计在总体的维度上来看，能对社交网络中的任意单个用户进行增删改查操作，对任意用户间的关注关系进行增删改查操作，能进行二度好友、共同喜欢、共同关注等高级查询功能（如图3-1所展示）。

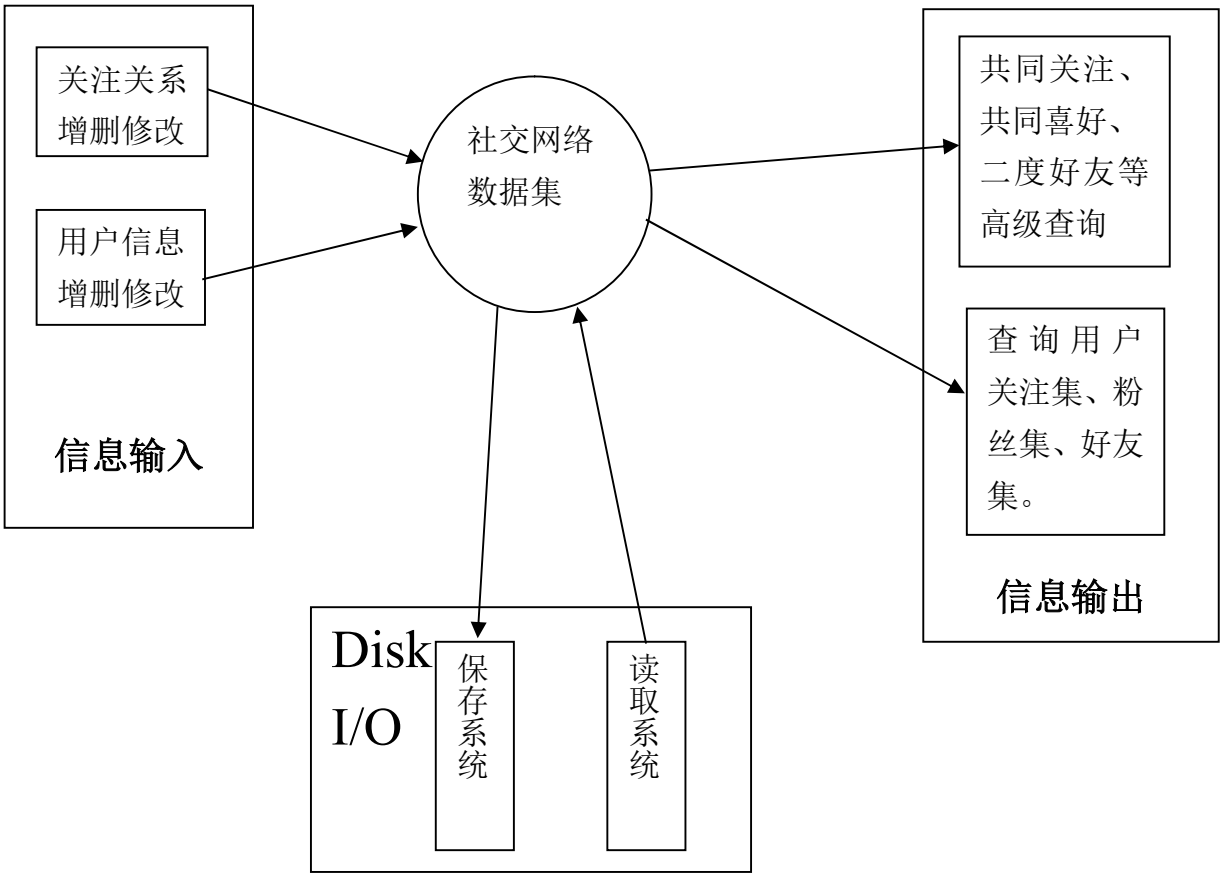

图 2-1 系统总体设计（箭头代表信息流向）

3. 系统详细设计

3.1 有关数据结构的定义

3.1.1 用户数据结构体

用户数据 (Info)	数据类型
name	char [30]
hobby	char [10]
age	int
sex	int

表 3-1

3.1.2 树节点结构体

用户树节点(UNode) 数据	数据类型
info	Info *
height	int
following	FNode *
followed	FNode *
left	UNode *
right	UNode *

表 3-2

关注树节点(FNode *) 数据	数据类型
info	Info *
height	int
left	FNode *
right	FNode *

表 3-3

3.1.3 数据关联方式

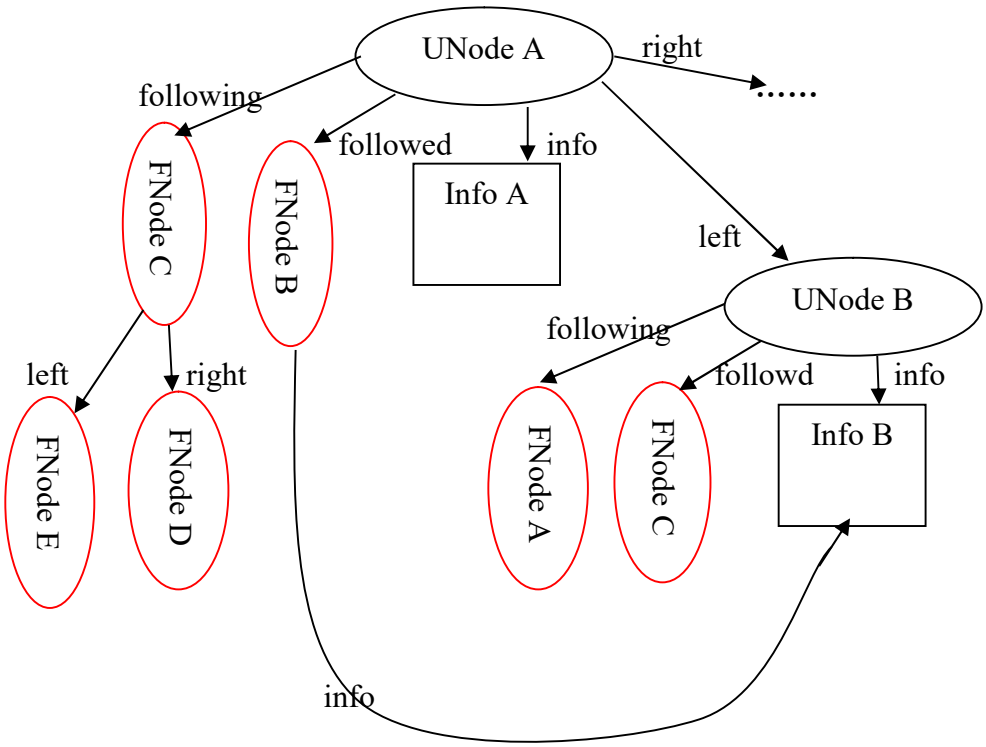

图 3-1 数据的关联示意图

通过设计图3-1的数据关联图，起到节省Info结构体数据冗余储存的作用，也避免了过渡设计，与此同时节省了一定的时间开销。每棵平衡二叉树中的的节点键值为其姓名（故本系统不允许有同名用户）。

3.2 主要算法设计

3.2.1 集合基本操作算法

（1）从集合中获取指定姓名的节点：getFNodeFromName_F(char * name, FNode * node)

初始条件：name 非空字符串，node 为某一平衡二叉树的根节点

操作结果：返回一个以 name 为名的树中节点，若没有找到则返回 NULL

算法思想：通过递归调用，前序遍历整棵树。若为空节点则返回 NULL，若当前遍历到的节点的 name 与参数 name 相同，则返回该节点，否则向其左子树和右子树递归调用当前函数。

复杂度：时间复杂度 T(n) = O(logn)

空间复杂度 S(n) = O(logn)

（2）销毁树：destroyAVL_F(FNode * node)

初始条件：传入的参数 node 为欲销毁的二叉树的根节点

操作结果：销毁整棵平衡二叉树

算法思想：通过递归调用自身，通过前序遍历，从二叉树的左子树最底端开始到右子树最底端到根节点，一一清空内存占用。

复杂度：时间复杂度 T(n) = O(n)

空间复杂度 S(n) = O(logn)

（3）为节点赋值高度：updateHeight_F(FNode * node) (FNode * node)

初始条件：传入的参数 node 为所想要知道高度的节点

操作结果：该节点的高度值将会更新

算法思想：直接获得左右孩子节点的高度值，哪个高就取其值再加一。

复杂度：时间复杂度 T(n) = O(1)

空间复杂度 S(n) = O(1)

（4）对失衡树进行一次左旋：LL_Rotation_F(FNode * oldHead)

初始条件：传入的参数 oldHead 为待左旋的子树的根节点

操作结果：进行一次左旋，更新内部节点的高度值，返回新的子树根节点。

算法思想：oldHead 的左孩子成为根节点，而 oldHead 成为现根节点的右孩子，原左孩子的右孩子挪为 oldHead 的左孩子，从而实现不打破排序性的前提下，实现了平衡。见图 3-2

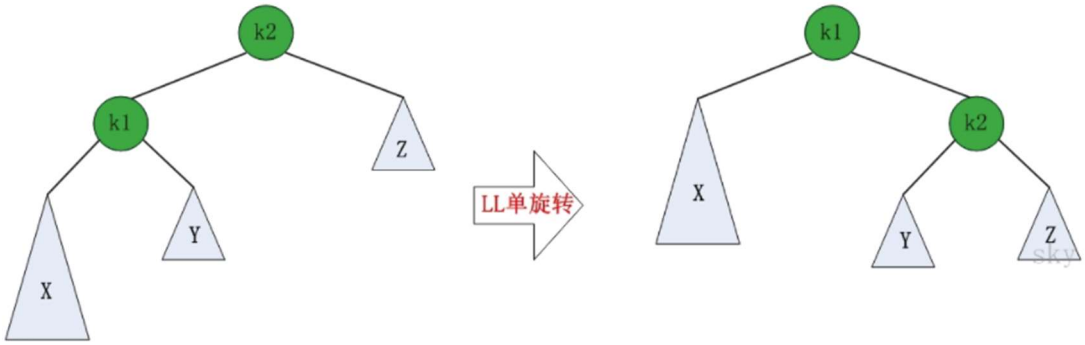

图 3-2

复杂度：时间复杂度 T(n) = O(1)

空间复杂度 S(n) = O(1)

（5）对失衡树进行一次右旋：RR_Rotation_F(FNode * oldHead)

初始条件：传入的参数 oldHead 为待右旋的子树的根节点

操作结果：进行一次右旋，更新内部节点的高度值，返回新的子树根节点。

算法思想：oldHead 的右孩子成为根节点，而 oldHead 成为现根节点的左孩子，原右孩子的左孩子挪为 oldHead 的右孩子。即为左旋函数的对称。见图 3-3。

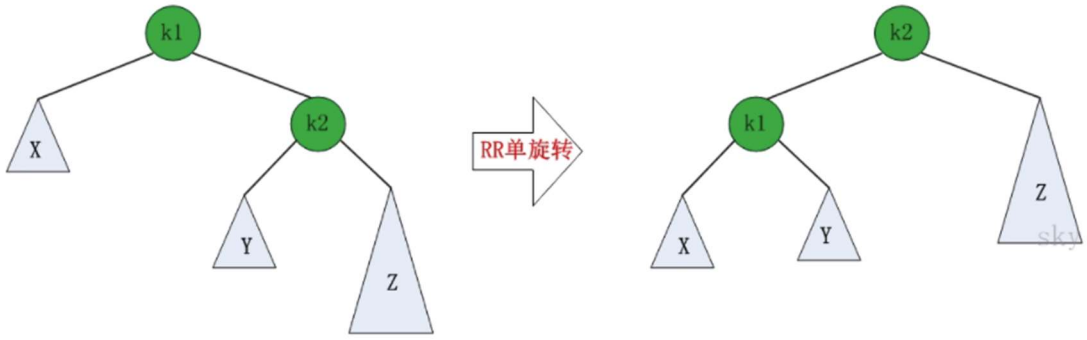

图 3-3

复杂度：时间复杂度 T(n) = O(1)

空间复杂度 S(n) = O(1)

（6）对失衡树进行一次左旋再右旋： RL_Rotation_F(FNode * oldHead)

初始条件：传入的参数 oldHead 为待左右旋的子树的根节点

操作结果：进行一次左旋，更新内部节点的高度值，再进行一次右旋再更新高度，最后返回新的子树根节点。

算法思想：调用一次左旋函数，再调用一次右旋函数。见图 3-4。

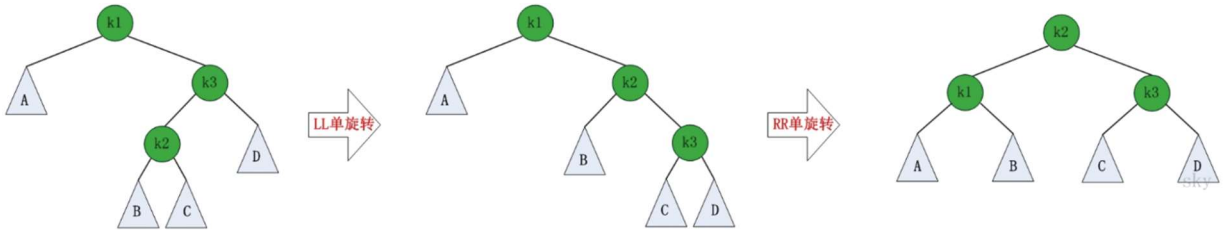

图 3-4

复杂度：时间复杂度 T(n) = O(1)

空间复杂度 S(n) = O(1)

（7）对失衡树进行一次右旋再左旋：LR_Rotation_F(FNode * oldHead)

初始条件：传入的参数 oldHead 为待右左旋的子树的根节点

操作结果：进行一次右旋，更新内部节点的高度值，再进行一次左旋再更新高度，最后返回新的子树根节点。

算法思想：调用一次右旋函数，再调用一次左旋函数。见图 3-5。

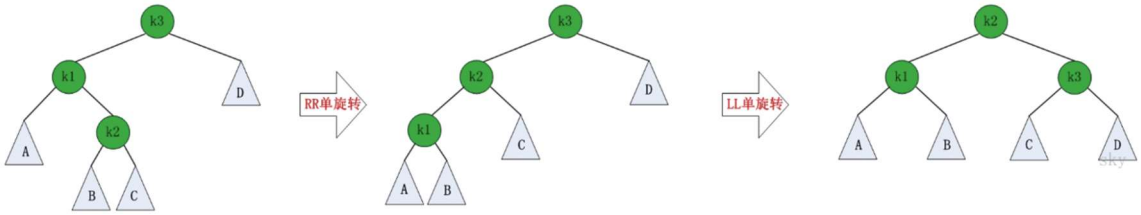

图 3-5

复杂度：时间复杂度 T(n) = O(1)

空间复杂度 S(n) = O(1)

（8）向树中插入节点：insertAVL_F(Info * newInfo, FNode * node)

初始条件：传入的参数 node 为待插入平衡二叉树的根节点，newInfo 为欲插入的新节点

操作结果：插入该信息，更新高度，并返回新的二叉树根节点，失败则返回 NULL。

算法思想：拿传入参数 newInfo 的 name 与当前 node->name 进行对比，若相等则插入失败，若小于则向左子树递归调用，反之右子树。若 node 为 NULL，则说明抵达了树的底部，此时创建指向 newInfo 的节点。创建完后开始对被修改子树中的所有节点更新高度，与此同时，若某一级子树插入后出现了失衡（左右子树高度相差 2）现象，则进行对应的旋转策略。

复杂度：时间复杂度 T(n) = O(logn)

空间复杂度 S(n) = O(logn)

（9）删除树中指定节点：deleteAVL_F(char * key, FNode * node)

初始条件：传入的参数 node 为待操作树的根节点，key 为欲删除的节点的 name。

操作结果：返回完成删除后的二叉树的根节点。

算法思想：先通过递归调用函数、逐次比较 node->name 和 key，通过结果来判断向左或右孩子来递归调用，直到找到欲删除的节点，若该节点的左右孩子中有一个为 NULL，则将该孩子和待删除节点调换位置，并删除待删除节点，若左右孩子都不为 NULL，则比较待删除节点的左右子树的高度，选择较高的子树，获得该子树最贴近待删除节点键值的节点（即左子树找最大，右子树找最小），将其与待删除节点交换顺序，最后删除待删除节点。以此在删除的同时保持树的平衡性。

复杂度：时间复杂度 T(n) = O(logn)

空间复杂度 S(n) = O(logn)

（10）输出平衡树中所有节点的 name：traverseAVL_F(FNode * node)

初始条件：传入的参数 node 为该平衡二叉树的根节点

操作结果：输出所有节点的 name 属性

算法思想：通过递归，若当前 node 为空则退出函数，否则输出当前 node 的 name 属性，然后向左右孩子节点分别递归调用自身函数。

复杂度：时间复杂度 T(n) = O(n)

空间复杂度 S(n) = O(logn)

（11）获得任意树的深度：getDepth_F(FNode * fRoot)

初始条件：传入的参数 fRoot 为该树的根节点

操作结果：返回树的深度，也就是根节点的高度。

算法思想：外部声明 全局变量 depthMax，通过递归逐步深入给定二叉树，若当前深度大于 depthMax，则更新 depthMax。最后在递归栈的最底部返回根节点对应的高度，也就是树的深度。

复杂度：时间复杂度 T(n) = O(n)

空间复杂度 S(n) = O(logn)

3.2.2 集合高级查询算法

（1）两棵树求并集：getSetUnion(FNode * aFNode, FNode * bFNode)

初始条件：传入的两个参数为两棵树各自的根节点

操作结果：返回并集平衡二叉树的根节点

算法思想：声明一棵空的平衡二叉树，递归遍历两棵平衡二叉树的每一个节点，插入到返回树中，最后返回新树的根节点。

复杂度：时间复杂度 T(n) = O(nlogn+eloge)

空间复杂度 S(n) = O(logn+loge)

（2）两棵树求交集：getSetIntersec(FNode * aFNode, FNode * bFNode)

初始条件：传入的两个参数为两棵树各自的根节点

操作结果：返回交集平衡二叉树的根节点

算法思想：递归 aFNode 所领的平衡二叉树 A 的每一个节点，查询 bFNode 所领的平衡二叉树 B 内是否有同名的节点，若有则插入到返回树中，最后返回新树的根节点。

复杂度：时间复杂度 T(n) = O(nloge)

空间复杂度 S(n) = O(logn)

（3）两棵树求差集：getSetDifference(FNode * aFNode, FNode * bFNode)

初始条件：传入的两个参数为两棵树各自的根节点

操作结果：输出 A 树减去 B 树后的新树根节点

算法思想：递归 aFNode 所领的平衡二叉树 A 的每一个节点，查询 bFNode 所领的平衡二叉树 B 内是否有同名的节点，若无则插入到返回树中，最后返回新树的根节点。

复杂度：时间复杂度 T(n) = O(nloge)

空间复杂度 S(n) = O(logn)

（4）查询两树是否有子集关系：isAsub2B(FNode * aFNode, FNode * bFNode)

初始条件：传入的两个参数为两棵树各自的根节点

操作结果：输出判断的结果，若 A 是 B 的子集则返回 1，若不是返回 0。

算法思想：利用前文中的求差集函数，getSetDifference(aFNode, bFNode)，若返回的为 NULL，则说明 A-B=∅，即 A 为 B 的子集。

复杂度：时间复杂度 T(n) = O(nloge)

空间复杂度 S(n) = O(logn)

（5）在原树上做集合删除：setRemove(FNode * aFNode, FNode * bFNode)

初始条件：传入的两个参数为两棵树各自的根节点

操作结果：输出被删除后的 A 树的新的根节点。

算法思想：利用前文中的求差集函数，getSetDifference(aFNode, bFNode)，销毁原来的 A 树，返回新树的节点，外界进行更新。

复杂度：时间复杂度 T(n) = O(nloge)

空间复杂度 S(n) = O(logn)

（6）集合间的插入操作：insertA2B(FNode * aFNode, FNode * sampleRoot)

初始条件：传入的两个参数为两棵树各自的根节点

操作结果：将 A 树插入至 B 树中，返回 B 树的新根节点。

算法思想：遍历 A 树，将每个结点都插入 sampleRoot 所领的平衡二叉树中。

复杂度：时间复杂度 T(n) = O(nloge)

空间复杂度 S(n) = O(logn+loge)

（7）查询共同喜好集合：getSameHobbyUsers(UNode * sampleNode, UNode * root)

初始条件：传入的参数 sampleNode 为标准样例的节点，root 为待搜索的平衡二叉树。

操作结果：返回存有共同好友集合的平衡二叉树的根节点。

算法思想：声明一棵空树，遍历 root 所领的平衡二叉树，每次都检查当前节点的喜好是否和 sampleNode 的喜好相同，若相同则存入待返回的树中。遍历结束后返回这课共同喜好好友数。

复杂度：时间复杂度 T(n) = O(nloge)

空间复杂度 S(n) = O(logn)

（8）查询二度好友：getSecFriend(UNode * uNode)

初始条件：传入的参数 uNode 为待查询的节点

操作结果：返回存有所有二度好友集合的平衡二叉树的根节点。

算法思想：声明一棵新树，先通过前文中的求并集函数求出好友集合，然后遍历好友集合，针对每个好友节点，都找出其各自的好友集合，最后遍历添加入待返回的树中。最后在这棵树中删除 uNode 自身，返回出去。

复杂度：时间复杂度 T(n) = O(n^2)

空间复杂度 S(n) = O(n^2)

3.2.3 文件读取保存

（1）切割一维字符数组为二维：setDoubleCharArray(char** out, char* in)

初始条件：参数 out 为开辟空间好的二维字符数组，in 为预定格式化的单行字符串

操作结果：切分 in 字符串，并将结果存入 out 中。

算法思想：预先声明三个索引，allIndex, outInde, inIndex，分别为字符串中的索引，out 两个维度上的索引。遍历 in 字符串，拷贝到 out 的起始区域，若读取到的字符为’,’（即切分符），则结束当前子字符串的写入，令 outIndex 自加跳转到下一个子字符串的读写过程中。

复杂度：时间复杂度 T(n) = O(n)

空间复杂度 S(n) = O(1)

（2）释放二维字符数组空间：freeDoubleCharArray(int num, char ** info)

初始条件：num 为二维字符数组持有的字符指针数量，info 为待释放空间的二维字符数组。

操作结果：完整释放该二维数组占用的空间。

算法思想：遍历 info 内的所有字符指针，对每个指针调用系统函数 free()。

复杂度：时间复杂度 T(n) = O(n)

空间复杂度 S(n) = O(1)

（3）读取用户信息：getUsers(FILE * file)

初始条件：参数 file 为外部指定的文本存档。

操作结果：从存档中读取所有用户的个人信息，并创建为平衡二叉树，返回该二叉树的根节点。

算法思想：先声明一个空树。在主循环中循环读取每一行的文本数据，直到文件尾为止，对每一行文本数据利用前文的切割函数进行切分，再将每个子字符串存入一个新的用户节点中，然后释放子字符串数据，再将这个新用户节点插入这个新的平衡二叉树中（在插入过程中能够自动维护结构体成员高度）。最后返回这棵树的根节点。

复杂度：时间复杂度 T(n) = O(nlogn)

空间复杂度 S(n) = O(nlogn)

（4）读取用户的关注关系：getRelation(UNode * root, FILE * file)

初始条件：参数 file 为外部指定的文本存档，root 为用户集合树的根节点。

操作结果：为用户集合树中的用户节点恢复存档中的关系数据。

算法思想：在主循环中循环读取每一行的文本数据，直到文件尾为止，对每一行文本数据利用前文的切割函数进行切分。两个子字符串分别为关注者和被关注者。从用户集合中搜索获得对应的用户节点，向其关注树和粉丝树插入对应的节点，释放子字符串数据。

复杂度：时间复杂度 T(n) = O(nlogn)

空间复杂度 S(n) = O(nlogn)

（5）保存用户信息至文本存档中：saveUser(UNode * node, FILE * file)

初始条件：参数 file 为外部指定的文本存档，node 为用户集合树的根节点。

操作结果：向 file 中写入文本的存档数据。

算法思想：递归遍历系统中的用户二叉树，持有任意一个用户节点的时候，按照预定格式化写入以’,’为分隔符的单行数据以此声明该用户的所有数据。

复杂度：时间复杂度 T(n) = O(n)

空间复杂度 S(n) = O(logn)

（6）保存用户关注信息：saveRelation_1(UNode * node, FILE * file)/ saveRelation_2(FNode * node, FILE * file, char * following)

初始条件：参数 file 为外部指定的文本存档，UNode *node 为用户树的根节点， FNode * node 为关注树的根节点，following 为关注者的 name。

操作结果：将整个系统中的所有的用户关注关系存入存档中。

算法思想：整个保存的过程由于涉及到了两层树的迭代，故使用了两个函数分别进行两层各自的递归使用。第一个函数递归遍历用户树中的每个节点，对其的关注树调用第二个函数。在第二个函数中，递归遍历关注树中的每个节点，访问到其成员 name，从而向 file 中写入单行格式化文本数据。

复杂度：时间复杂度 T(n) = O(n*e)

空间复杂度 S(n) = O((logn)*(loge))

（6）保存用户关注信息：saveRelation_1(UNode * node, FILE * file)/ saveRelation_2(FNode * node, FILE * file, char * following)

初始条件：参数 file 为外部指定的文本存档，UNode *node 为用户树的根节点， FNode * node 为关注树的根节点，following 为关注者的 name。

操作结果：将整个系统中的所有的用户关注关系存入存档中。

算法思想：整个保存的过程由于涉及到了两层树的迭代，故使用了两个函数分别进行两层各自的递归使用。第一个函数递归遍历用户树中的每个节点，对其的关注树调用第二个函数。在第二个函数中，递归遍历关注树中的每个节点，访问到其成员 name，从而向 file 中写入单行格式化文本数据。

复杂度：时间复杂度 T(n) = O(n*e)

空间复杂度 S(n) = O((logn)*(loge))

4. 系统实现与测试

4.1 系统实现

本系统意在实现一个基于 AVL 自平衡二叉树为底层数据结构的社交网络数据管理系统，支持较好的性能表现，避免了较高的内存、硬盘冗余占用。系统界面如图 4-1。

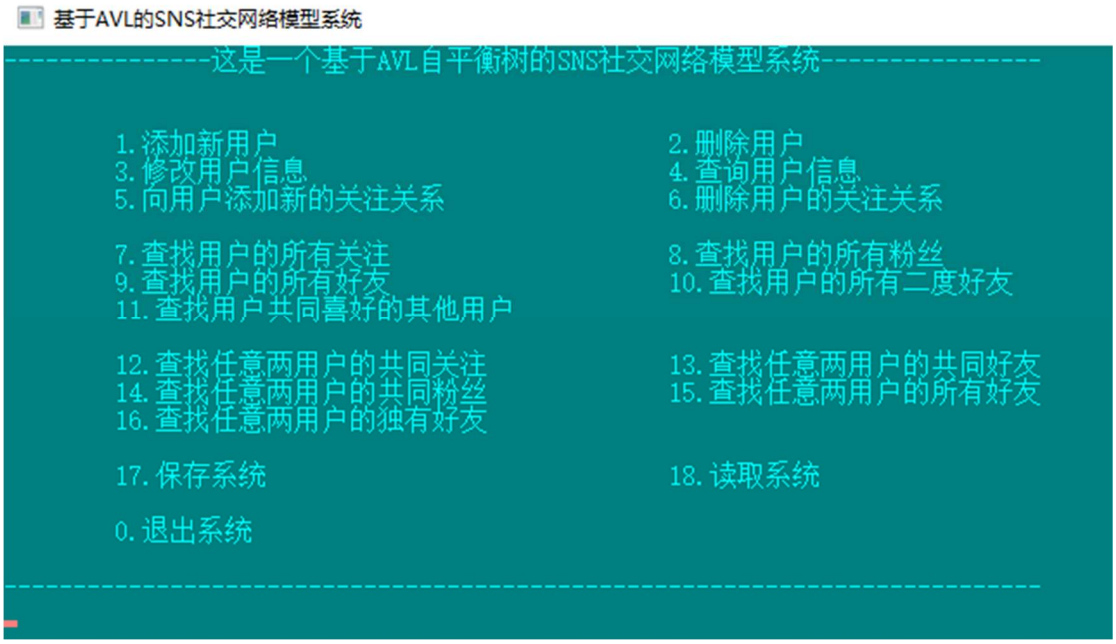

图 4-1

系统实现的软硬件环境：系统：Windows 10 64-bits ver.1709 ，编译器：GNU GCC Compiler。

C语言中的数据结构定义：

```
typedef struct info {
    char * name;//length 30, and this would be the key to find user
    char * hobby;//length 10
    int age;
    int sex;//1->man 2->woman
            //and othe infomations
} Info;
```

```
typedef struct fNode {
   Info * info;
   int height;
   struct fNode * left;
   struct fNode * right;
} FNode;

typedef struct userNode {
   Info * info;
   int height;
   FNode * following;
   FNode * followed;
   struct userNode * left;
   struct userNode * right;
} UNode;
```

系统所展示的社交网络数据管理系统的功能与底层函数的链接示意图：

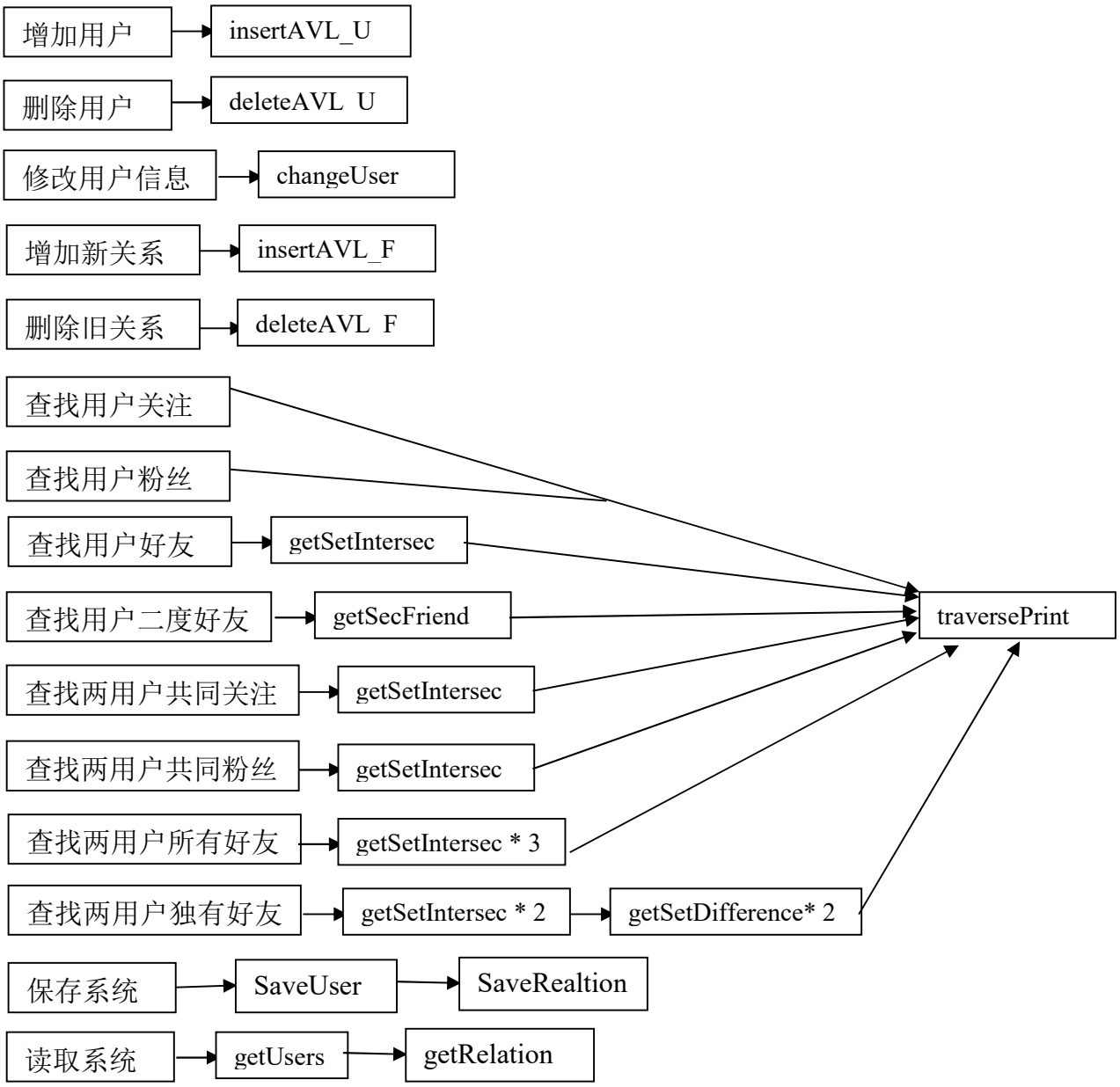

图 4-2

4.2 系统测试

4.2.1 常用的软件测试方法：

1）黑盒测试：只关注输入数据后，输出的数据是否符合预期。

2）白盒测试：关注内部实现逻辑。

3）单元测试：对软件中某个特定的最小可测试部分进行单独的可行性测

试。

4）系统测试：将整个系统视为整体，包括对性能、功能等方面进行详尽的测试。

4.2.2 系统模块预计功能

1）用户信息的增删改查

2）用户关系的增删

3）社交网络中常见的高级查询功能（二度好友等）

4）系统保存与读取

4.2.3 测试数据

测试数据由Python脚本随机生成的ANSI编码的文本存档文件（脚本的源代码见附1），存有十万条用户信息和五十万条关注关系。

4.2.4 预期测试结果

图 4-3

根据随机生成与手工修改的存档文件（用户关注关系如图 4-3）我们设置了如下的测试大纲。

测试功能	输入数据	预期结果
添加用户	测试者，篮球，18 岁，男	成功添加用户
修改用户信息	测试者，修改年龄为 20 岁	成功修改用户信息
查询用户信息	测试者	成功输出其信息
删除用户	测试者	成功删除
查看用户的关注	庞米小	梁邵冯，赵玉，熊姚光，臧玲米
查看用户的粉丝	庞米小	纪玲卫 ，董堪陈 ，狄庞冯 ，纪光舒 ，庞戴周 ，堪大吴 ，祁宋毛
添加新用户关系	庞米小，纪玲卫	成功添加关系
删除用户关系	庞米小，赵玉	成功删除关系
查看用户的好友	庞米小	纪玲卫
查找用户的所有二度好友	庞米小	成庞谈
查找共同喜好的用户	庞米小	输出具有共同喜欢的用户集合
查找任意两用户的共同关注	董堪陈，纪玲卫	庞米小
查找任意两用户的共同好友	A, B	E
查找任意两用户的共同粉丝	庞米小，成庞谈	纪玲卫
查找任意两用户的所有好友	庞米小，成庞谈	纪玲卫
查找任意两用户的独有好友	A, B	A->D, B->C

保存系统	无	成功保存系统
读取系统		成功读取系统

4.2.5 测试运行结果

1）添加用户

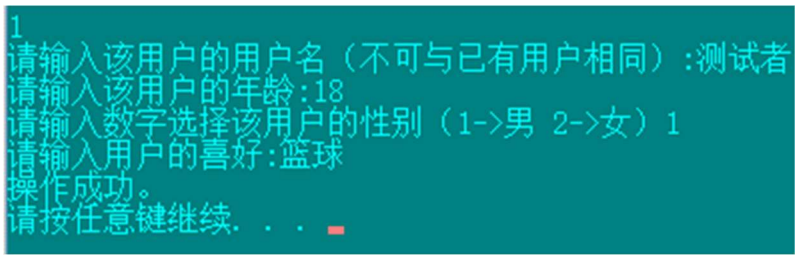

图4-4

如图 4-4 所示，系统提醒成功添加了用户，操作成功。

2）修改用户信息

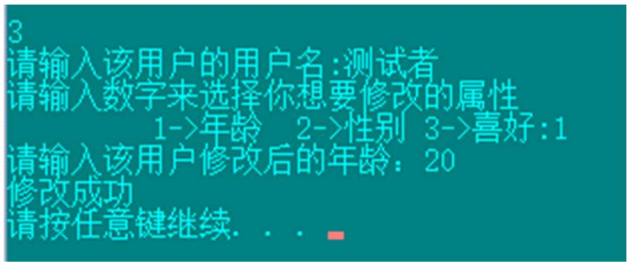

图4-5

如图4-5所示，系统提醒成功修改了用户的个人信息，操作成功。

3）查询用户信息

图 4-6

如图 4-6 所示，系统成功输出了用户的所有个人信息，操作成功。

4）删除用户

图 4-7

如图 4-7 所示，系统成功删除了用户，操作成功。

5）查看用户的关注

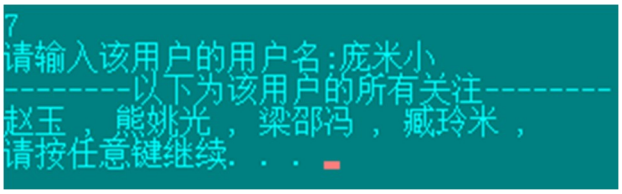

图 4-8

如图 4-8 所示，系统成功输出了用户的所有关注，操作成功。

6）查看用户的粉丝

图 4-9

如图 4-9 所示，系统成功输出了用户的所有粉丝，操作成功。

7）添加新用户关系

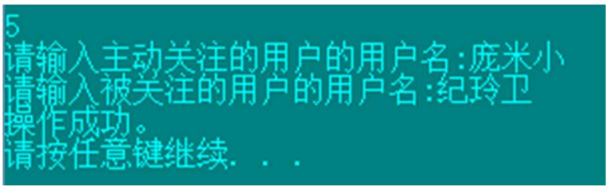

图 4-10

如图 4-10 所示，系统成功添加了新的用户关系，操作成功。

8）查看用户的好友

图 4-11

如图 4-11 所示，系统成功输出了用户的所有好友，操作成功。

9）删除用户的关系

图 4-12

如图 4-12 所示，系统成功删除了指定的用户关系。

10）查询用户的二度好友

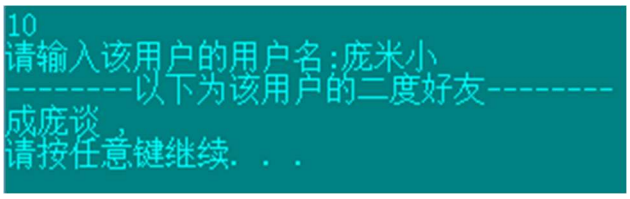

图 4-13

如图 4-13 所示，系统成功输出了用户的所有二度好友，操作成功。

11）查找共同喜好的用户

图 4-14

如图 4-14 所示，系统成功输出了所有与指定用户有共同喜好的用户，操作成功。

12）查找任意两用户的共同关注

图 4-15

如图 4-15 所示，系统成功输出了指定两用户的共同关注，操作成功。

13）查找任意两用户的共同粉丝

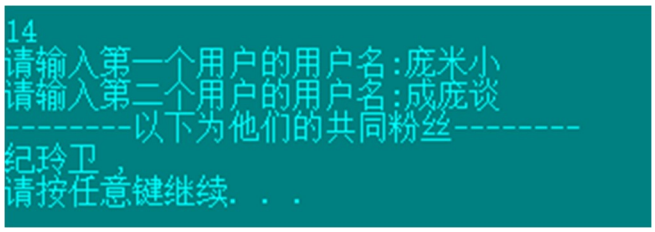

图 4-16

如图 4-16 所示，系统成功输出了指定两用户的共同粉丝，操作成功。

14）查找任意两用户的共同好友

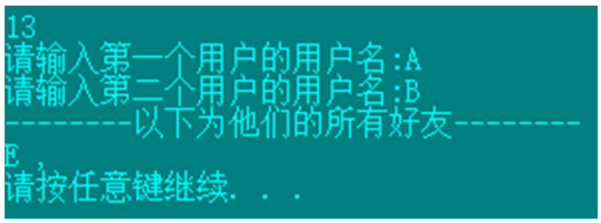

图 4-17

如图 4-17 所示，系统成功输出了指定两用户的共同好友，操作成功。

15）查找任意两用户的所有好友

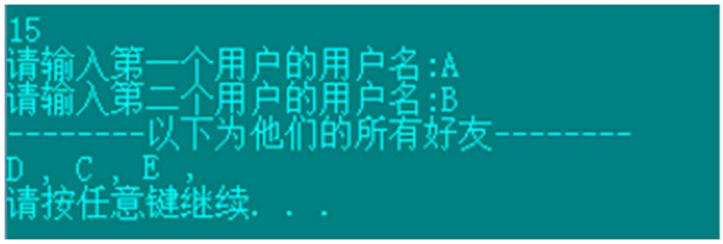

图 4-18

如图 4-18 所示，系统成功输出了指定两用户的所有好友，操作成功。

16）查找任意两用户的独有好友

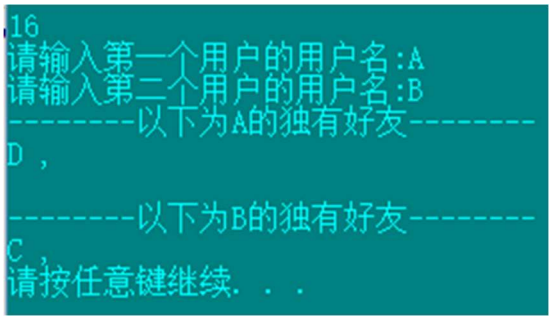

图 4-19

如图 4-19 所示，系统成功输出了指定两用户的独有好友，操作成功。

17）保存系统

图 4-20

如图 4-20 所示，系统成功保存了，操作成功。

18）读取系统

图 4-21

如图 4-21 所示，成功读取了存档到系统中，操作成功。

5. 总结与展望

5.1 全文总结

（1）自己能够通过这样的课程设计进一步认识到基础数据结构在底层起到的优化提速作用，并加深对数据结构的认识。

（2）通过脚本的高扩展能力来生成十万级数量的存档来对系统进行性能测试的过程中，又让我深刻感触到了不同语言的不同适用性。

（3）在搭建底层数据架构的时候，更是让我感觉到底层设计对上层调用开销产生的巨大影响。

5.1 工作展望

（1）如果有更多的时间的话，我会考虑把存档的格式进一步进行优化，实现带树结构方式来保存，可以大大缩短读取内部存档（有序）所消耗的时间复杂度至线性。不过这样一来导入外部存档(无序)的时候就需要另一个读入函数了。

（2）类似社交网络这样的关系密集的集合，更适合使用图来保存，如果有机会，我觉得利用图来完成同样的社交网络模型设计会更加高效。

6. 体会

本次实验中，我从底层开始实现了一种相对数据结构实验中更为复杂的数据结构——AVL 自平衡二叉树。通过失衡后的左右旋来维护二叉树的平衡性，以此来保证对二叉树的增删改查操作能达到对数级别的复杂度，大大提高了系统在面对较大数据量时的快速反应能力。

手动写这样较为复杂的数据结构并不是第一次，但是在面对比较复杂的递归函数的编写的时候仍然会遇到卡壳的问题，这时候，利用纸笔来搭建一个简易的纸上图形 debug“系统”成为了一个快速理解递归函数运行过程的一个好助手。

本次课程设计一共有三个题目，初意选择第三个是因为第三个题目已经规定了基础的数据结构，不需要大量的规划设计。我的课设作品中唯一感到有设计感的架构便是图 3-1 所展示的数据的关联示意图，通过这样的底层设计，能够尽可能地在保证拥有 AVL 自平衡树带来的时间节省，同时不会有多余的内存开销。

FNode 和 UNode 两个结构体看上去有一些重复，受限于 C 语言没有泛型等高级语言的特性，带来了大量的额外维护代码。但是这样的设计也有它的好处，比如能够减少一定的内存开销，提高代码的可读性（两种二叉树的功能不同）。

使用二叉树来保存社交网络这个类型的数据其实并不是很合适，因为社交网络中有大量的求好友间连续关系的需求，若使用二叉树来实现则会带来大量重复搜索的问题，并且需要多持有一些用户信息的指针，又带来了额外的开销。

AVL 自平衡二叉树是一种经典的自平衡二叉树，相对红黑树等其他类型，它具有更高的平衡性和更简单的代码实现。通过这次课程设计还是极大地锻炼了自己的动手能力。

7. 参考文献

[1] Robert Sedgewick & Kevin Wayne 《Algorthms Fourth Edition》

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

附录

生成存档脚本：

generate_random_user_infomation.py：

```
import random as r

txtName = "10w.u"

f = open(txtName, "w")

check = []

names_a = ['赵', '钱', '孙', '李', '周', '吴', '郑', '王', '冯', '陈',
'褚', '卫', '蒋', '沈', '韩', '杨', '朱', '秦', '尤', '许',
           '姚', '邵', '堪', '汪', '祁', '毛', '禹', '狄', '米', '贝',
'明', '臧', '计', '伏', '成', '戴', '谈', '宋', '茅', '庞',
           '熊', '纪', '舒', '屈', '项', '祝', '董', '梁', '张', '金']

names_b = ['玉', '明', '龙', '芳', '军', '玲', '赵', '钱', '孙', '李',
'周', '吴', '郑', '王', '冯', '陈', '褚', '卫', '蒋', '沈', '韩', '杨
',
           '朱', '秦', '尤', '许',
           '姚', '邵', '堪', '汪', '祁', '毛', '禹', '狄', '米', '贝',
'明', '臧', '计', '伏', '成', '戴', '谈', '宋', '茅', '庞',
           '熊', '纪', '舒', '屈', '项', '祝', '董', '梁', '张', '金',
'阿', '大', '小', '光', '晓']
names_c = ['玉', '', '明', '龙', '芳', '军', '玲', '赵', '钱', '孙', '
李', '周', '吴', '', '郑', '', '王', '冯', '', '陈', '褚', '卫',
           '蒋', '', '沈', '韩', '杨', '朱', '秦', '尤', '许',
           '姚', '邵', '堪', '汪', '祁', '毛', '禹', '', '狄', '米', '
贝', '明', '', '臧', '计', '伏', '成', '戴', '谈', '宋', '茅', '庞',
           '熊', '纪', '舒', '屈', '项', '', '祝', '董', '梁', '', '
张', '', '金', '阿', '', '大', '小', '光', '晓']
```

```
hobby = ['旅行', '摄影', '音乐', '电子游戏', '篮球', '足球', '乒乓球',
'手球', '钢琴', '吉他', '人声', '代码', '学习', '阅读', '吃', '户外', '
跑步', '旅游',
         '羽毛球', '电影', '跳舞', '跳高', '跳远',
         '茶道', '插花', '绘画', '书法', '电视剧', '相声', '小品', '
音乐剧', '购物', '时尚', '明星', '铅球', '三级跳', '游泳', '跳绳', '
跳水', '潜水', '驾驶',
         '极限运动']

i = 1
while i <= 100000:
    name = r.choice(names_a) + r.choice(names_b) + r.choice(names_c)

    if name not in check:
        info =  name+ "," + r.choice(hobby) + "," + str(r.randint(1,80))
+ ";" + str(r.randint(1, 2)) + "\n"
        check.append(name)
        f.write(info)
        i= i+1
        print(i)

f.close()
```

generate_random_follow_relationships.py：

```
# -*- coding: utf-8 -*-

import random as r
import csv
uText = "10w.u"
fText = "10w.r"

name = []

with open(uText, 'r', encoding="utf-8") as csvFile:
    read = csv.reader(csvFile)
```

```
    for item in read:
        # print(item[0])
        name.append(item[0])


# now name has all names

saveFile = open(fText, "w", encoding="utf-8")
for i in range(len(name)):
    followedArray = [name[i]]  # itself
    j = 1
    while j <= 5:
        followed = r.choice(name)
        if followed not in followedArray:
            rowText = name[i] + "," + r.choice(name) + "\n"
            saveFile.write(rowText)
            j = j + 1
            print(str(i*5+j))


saveFile.close()
```

系统源代码：

AVL-SNS.c：

```
#include <IO.h>
#include <windows.h>
#include <wincon.h>
#include <windef.h>
#include "shlobj.h"
#include <wingdi.h>
#define NULLCODE 90017

UNode * uRoot;

char *fgetsNoN(char *buf, int bufsize, FILE *stream);
```

```
void printInfo(int mode);

/*
Trick
1->系统堆空间满导致失败
2->操作成功
3->无此名的用户
4->没有找到相关信息
*/
void printInfo(int mode) {
    switch (mode) {
    case 1:
        printf("系统堆空间满导致失败。\n");
        break;
    case 2:
        printf("操作成功。\n");
        break;
    case 3:
        printf("无此名的用户。\n");
        break;
    case 4:
        printf("没有找到相关信息\n");
        break;
    }
}

void initConsole(){
    system("mode con:cols=100 lines=30");
    system("color 3B");
    SetConsoleTitle("基于 AVL 的 SNS 社交网络模型系统");
}

int printMenu() {
    int op = 90071;

    printf("---------------这是一个基于 AVL 自平衡树的 SNS 社交网络模型
```

```
系统----------------\n\n\n");

    printf("\t1.添加新用户\t\t\t\t2.删除用户\n");
    printf("\t3.修改用户信息\t\t\t\t4.查询用户信息\n");
    printf("\t5.向用户添加新的关注关系\t\t6.删除用户的关注关系\n\n");

    printf("\t7.查找用户的所有关注\t\t\t8.查找用户的所有粉丝\n");
    printf("\t9.查找用户的所有好友\t\t\t10.查找用户的所有二度好友
\n");
    printf("\t11.查找用户共同喜好的其他用户\n\n");

    printf("\t12.查找任意两用户的共同关注\t\t13.查找任意两用户的共同
好友\n");
    printf("\t14.查找任意两用户的共同粉丝\t\t15.查找任意两用户的所有
好友\n");
    printf("\t16.查找任意两用户的独有好友\n\n");

    printf("\t17.保存系统\t\t\t\t18.读取系统\n\n");

    printf("\t0.退出系统\n\n");

    printf("-------------------------------------------------------
------------------\n");
    scanf("%d%*c", &op);

    return op;
}

//add a new user
void addUser() {

    printf("请输入该用户的用户名（不可与已有用户相同）:");
    char * newUserName = (char *)malloc(sizeof(char) * 30);
    fgetsNoN(newUserName, 30, stdin);

    UNode * check = getUNodeFromName_U(newUserName, uRoot);
```

```
    if (check) {
        printf("此用户名已存在。\n");
    } else {
        printf("请输入该用户的年龄:");
        int age = 0; scanf("%d%*c", &age);
        int sex = 0;
        while (sex != 1 && sex != 2) {
            printf("请输入数字选择该用户的性别（1->男 2->女）");
            scanf("%d%*c", &sex);
        }

        printf("请输入用户的喜好:");
        char * newhobby = (char *)malloc(sizeof(char) * 10);
        fgetsNoN(newhobby, 10, stdin);

        Info * newInfo = (Info *)malloc(sizeof(Info));
        newInfo->age = age;
        newInfo->sex = sex;
        newInfo->name = newUserName;
        newInfo->hobby = newhobby;
        uRoot = insertAVL_U(newInfo, uRoot);

        if (uRoot) printInfo(2);
        else printf("failed!\n");
    }

}

//del a user from this system
void delUser() {

    printf("请输入该用户的用户名:");
    char * userName = (char *)malloc(sizeof(char) * 50);
    fgetsNoN(userName, 50, stdin);

    UNode * check = getUNodeFromName_U(userName, uRoot);
```

```
    if (!check) {
        printInfo(3);
    } else {
        uRoot = deleteAVL_U(userName, uRoot);
        printInfo(2);
    }
    free(userName);

}

//change user's infomation
void changeUser() {
    printf("请输入该用户的用户名:");
    char * userName = (char *)malloc(sizeof(char) * 50);
    fgetsNoN(userName, 50, stdin);

    UNode * getUNode = getUNodeFromName_U(userName, uRoot);
    if (!getUNode) {
        printInfo(3);
    } else {
        int select = 0;
        while (select != 1 && select != 2 && select != 3) {
            printf("请输入数字来选择你想要修改的属性\n\t 1->年龄  2->性
别 3->喜好:");
            scanf("%d%*c", &select);
        }
        if (select == 1) {
            printf("请输入该用户修改后的年龄：");
            int newAge = 0; scanf("%d%*c", &newAge);
            getUNode->info->age = newAge;
        } else if (select == 2) {
            printf("请输入数字选择进行修改用户的性别：(1->男  2->女)");
            int newSex = 0; scanf("%d%*c", &newSex);
            getUNode->info->sex = newSex;
        } else if (select == 3) {
            printf("请输入该用户新的喜好：");
```

```
            fgetsNoN(getUNode->info->hobby, 10, stdin);
        }
        printf("修改成功\n");
    }
    free(userName);
}

//show all infomation of a specified user
void showUser() {
    printf("请输入该用户的用户名:");
    char * userName = (char *)malloc(sizeof(char) * 50);
    fgetsNoN(userName, 50, stdin);

    UNode * getUNode = getUNodeFromName_U(userName, uRoot);
    if (!getUNode) {
        printInfo(3);
    } else {
        Info * info = getUNode->info;
        printf("用户名:%s ; 年龄:%d ; 爱好:%s ; 性别:", info->name,
info->age, info->hobby);
        if (info->sex == 1) {
            printf("男\n");
        } else {//sex == 2
            printf("女\n");
        }
    }
    free(userName);
}

//add following relationship between two users
void addRelation() {
    printf("请输入主动关注的用户的用户名:");
    char * followUserName = (char *)malloc(sizeof(char) * 50);
    fgetsNoN(followUserName, 50, stdin);

    UNode * followUNode = getUNodeFromName_U(followUserName, uRoot);
```

```
    if (!followUNode) {
        printInfo(3);
    } else {
        printf("请输入被关注的用户的用户名:");
        char * followedUserName = (char *)malloc(sizeof(char) * 50);
        fgetsNoN(followedUserName, 50, stdin);
        UNode * followedUNode = getUNodeFromName_U(followedUserName,
uRoot);
        if (!followedUNode) {
            printInfo(3);
        } else {
            //assume we have so much stake space
            followedUNode->followed = insertAVL_F(followUNode->info,
followedUNode->followed);
            if (followedUNode->followed) {
               followUNode->following = insertAVL_F(followedUNode->info,
followUNode->following);
                printInfo(2);
            } else {
                printf("这两个用户已经有了这样的关注关系\n");
            }
        }
        free(followedUserName);
    }
    free(followUserName);
}

//delete following relationship between two users
void delRelation() {
    printf("请输入主动关注的用户的用户名:");
    char * followUserName = (char *)malloc(sizeof(char) * 50);
    fgetsNoN(followUserName, 50, stdin);

    UNode * followUNode = getUNodeFromName_U(followUserName, uRoot);
    if (!followUNode) {
        printInfo(3);
```

```
    } else {
        printf("请输入被关注的用户的用户名:");
        char * followedUserName = (char *)malloc(sizeof(char) * 50);
        fgetsNoN(followedUserName, 50, stdin);
        UNode * followedUNode = getUNodeFromName_U(followedUserName,
uRoot);
        if (!followedUNode) {
            printInfo(3);
        } else {
            deleteAVL_F(followUNode->info->name,
followedUNode->followed);
            deleteAVL_F(followedUNode->info->name,
followUNode->following);
            printInfo(2);
        }
        free(followedUserName);
    }
    free(followUserName);
}

//print all user's name when traverse whole AVL tree
void traversePrint(FNode * node) {
    if (node == NULL) return;
    printf("%s , ", node->info->name);
    traversePrint(node->left);
    traversePrint(node->right);
}

//print all user's following
void showFollowing() {
    printf("请输入该用户的用户名:");
    char * userName = (char *)malloc(sizeof(char) * 30);
    fgetsNoN(userName, 30, stdin);

    UNode * getUNode = getUNodeFromName_U(userName, uRoot);
    if (!getUNode) {
```

```
        printInfo(3);
    } else {
        if(getUNode->following){
            printf("--------以下为该用户的所有关注--------\n");
            traversePrint(getUNode->following); printf("\n");
        }else{
            printInfo(4);
        }
    }
    free(userName);
}

//print all user's followed
void showFollowed() {
    printf("请输入该用户的用户名:");
    char * userName = (char *)malloc(sizeof(char) * 30);
    fgetsNoN(userName, 30, stdin);

    UNode * getUNode = getUNodeFromName_U(userName, uRoot);
    if (!getUNode) {
        printInfo(3);
    } else {

        if(getUNode->followed){
            printf("--------以下为该用户的所有粉丝--------\n");
            traversePrint(getUNode->followed); printf("\n");
        }else{
            printInfo(4);
        }

    }
    free(userName);
}

//print all user's fans ( fans are both following and followed)
void showFriends() {
```

```
    printf("请输入该用户的用户名:");
    char * userName = (char *)malloc(sizeof(char) * 30);
    fgetsNoN(userName, 30, stdin);

    UNode * getUNode = getUNodeFromName_U(userName, uRoot);
    if (!getUNode) {
        printInfo(3);
    } else {
        FNode * friendsRoot = getSetIntersec(getUNode->following,
getUNode->followed);//BOTH
        if (friendsRoot) {
            printf("--------以下为该用户的所有好友--------\n");
            traversePrint(friendsRoot); printf("\n");
            destroyAVL_F(friendsRoot);
        } else {
            printInfo(4);
        }

    }
    free(userName);
}

//print second friends
void showSecondFriend() {
    printf("请输入该用户的用户名:");
    char * userName = (char *)malloc(sizeof(char) * 30);
    fgetsNoN(userName, 30, stdin);

    UNode * getUNode = getUNodeFromName_U(userName, uRoot);
    if (!getUNode) {
        printInfo(3);
    } else {
        FNode * secFriRoot = getSecFriend(getUNode, uRoot);
        if (secFriRoot) {
            printf("--------以下为该用户的二度好友--------\n");
            traversePrint(secFriRoot); printf("\n");
```

```
            destroyAVL_F(secFriRoot);
        } else {
            printInfo(4);
        }
    }
    free(userName);
}

//show common following of two specified users
void showCommonFollowing() {
    printf("请输入第一个用户的用户名:");
    char * userNameA = (char *)malloc(sizeof(char) * 30);
    fgetsNoN(userNameA, 30, stdin);

    UNode * uNodeA = getUNodeFromName_U(userNameA, uRoot);
    if (!uNodeA) {
        printInfo(3);
    } else {
        printf("请输入第二个用户的用户名:");
        char * userNameB = (char *)malloc(sizeof(char) * 30);
        fgetsNoN(userNameB, 30, stdin);
        UNode * uNodeB = getUNodeFromName_U(userNameB, uRoot);
        if (uNodeB) {
            FNode * comFollowing = getSetIntersec(uNodeB->following,
uNodeA->following);
            if (comFollowing) {
                printf("--------以下为他们的共同关注--------\n");
                traversePrint(comFollowing); printf("\n");
                destroyAVL_F(comFollowing);
            } else {
                printInfo(4);
            }
        } else {
            printInfo(3);
        }
        free(userNameB);
```

```
    }
    free(userNameA);

}

//show common followed of two specified users
void showCommonFollowed() {
    printf("请输入第一个用户的用户名:");
    char * userNameA = (char *)malloc(sizeof(char) * 30);
    fgetsNoN(userNameA, 30, stdin);

    UNode * uNodeA = getUNodeFromName_U(userNameA, uRoot);
    if (!uNodeA) {
        printInfo(3);
    } else {
        printf("请输入第二个用户的用户名:");
        char * userNameB = (char *)malloc(sizeof(char) * 30);
        fgetsNoN(userNameB, 30, stdin);
        UNode * uNodeB = getUNodeFromName_U(userNameB, uRoot);
        if (uNodeB) {
            FNode * comFollowed = getSetIntersec(uNodeB->followed,
uNodeA->followed);
            if (comFollowed) {
                printf("--------以下为他们的共同粉丝--------\n");
                traversePrint(comFollowed); printf("\n");
                destroyAVL_F(comFollowed);
            } else {
                printInfo(4);
            }
        } else {
            printInfo(3);
        }
        free(userNameB);
    }
    free(userNameA);
```

```
}

//show common friends of two specified users
void showCommonFriends() {
    printf("请输入第一个用户的用户名:");
    char * userNameA = (char *)malloc(sizeof(char) * 30);
    fgetsNoN(userNameA, 30, stdin);

    UNode * uNodeA = getUNodeFromName_U(userNameA, uRoot);
    if (!uNodeA) {
        printInfo(3);
    } else {
        printf("请输入第二个用户的用户名:");
        char * userNameB = (char *)malloc(sizeof(char) * 30);
        fgetsNoN(userNameB, 30, stdin);
        UNode * uNodeB = getUNodeFromName_U(userNameB, uRoot);
        if (uNodeB) {
            FNode * friRootA = getSetIntersec(uNodeA->followed,
uNodeA->following);
            FNode * friRootB = getSetIntersec(uNodeB->followed,
uNodeB->following);
            FNode * comFriRoot = getSetUnion(friRootA, friRootB);
            destroyAVL_F(friRootA); destroyAVL_F(friRootB);//clear
            if (comFriRoot) {
                printf("--------以下为他们的所有好友--------\n");
                traversePrint(comFriRoot); printf("\n");
                destroyAVL_F(comFriRoot);
            } else {
                printInfo(4);
            }
        } else {
            printInfo(3);
        }
        free(userNameB);
    }
    free(userNameA);
```

```
}

//show friends-union of two specified users
void showUnionFriends() {
    printf("请输入第一个用户的用户名:");
    char * userNameA = (char *)malloc(sizeof(char) * 30);
    fgetsNoN(userNameA, 30, stdin);

    UNode * uNodeA = getUNodeFromName_U(userNameA, uRoot);
    if (!uNodeA) {
        printInfo(3);
    } else {
        printf("请输入第二个用户的用户名:");
        char * userNameB = (char *)malloc(sizeof(char) * 30);
        fgetsNoN(userNameB, 30, stdin);
        UNode * uNodeB = getUNodeFromName_U(userNameB, uRoot);
        if (uNodeB) {
            FNode * friRootA = getSetIntersec(uNodeA->followed,
uNodeA->following);
            FNode * friRootB = getSetIntersec(uNodeB->followed,
uNodeB->following);
            FNode * unionFriRoot = getSetUnion(friRootA, friRootB);
            destroyAVL_F(friRootA); destroyAVL_F(friRootB);//clear
            if (unionFriRoot) {
                printf("--------以下为他们的所有好友--------\n");
                traversePrint(unionFriRoot); printf("\n");
                destroyAVL_F(unionFriRoot);
            } else {
                printInfo(4);
            }
        } else {
            printInfo(3);
        }
        free(userNameB);
    }
    free(userNameA);
```

```
}

//show one's Own Friends
void showOwnFriends() {
    printf("请输入第一个用户的用户名:");
    char * userNameA = (char *)malloc(sizeof(char) * 30);
    fgetsNoN(userNameA, 30, stdin);

    UNode * uNodeA = getUNodeFromName_U(userNameA, uRoot);
    if (!uNodeA) {
        printInfo(3);
    } else {
        printf("请输入第二个用户的用户名:");
        char * userNameB = (char *)malloc(sizeof(char) * 30);
        fgetsNoN(userNameB, 30, stdin);
        UNode * uNodeB = getUNodeFromName_U(userNameB, uRoot);
        if (uNodeB) {
            FNode   *   friRootA   =   getSetIntersec(uNodeA->followed,
uNodeA->following);
            FNode   *   friRootB   =   getSetIntersec(uNodeB->followed,
uNodeB->following);
            FNode * ownFriRootA = getSetDifference(friRootA, friRootB);
            FNode * ownFriRootB = getSetDifference(friRootB, friRootA);
            destroyAVL_F(friRootA); destroyAVL_F(friRootB);//clear

                                                    //output
            printf("--------以下为%s 的独有好友--------\n", userNameA);
            if (ownFriRootA) {
                traversePrint(ownFriRootA); printf("\n");
                destroyAVL_F(ownFriRootA);
            } else {
                printInfo(4);
            }

            printf("\n-------- 以 下 为 %s  的 独 有 好 友 --------\n",
userNameB);
```

```
            if (ownFriRootB) {
                traversePrint(ownFriRootB); printf("\n");
                destroyAVL_F(ownFriRootB);
            } else {
                printInfo(4);
            }
        } else {
            printInfo(3);
        }
        free(userNameB);
    }
    free(userNameA);
}

/*---------------save system---------- */
//without sequence
void saveUser(UNode * node, FILE * file){
    if(!node) return;
    Info * info = node->info;
    fprintf(file, "%s,%s,%d,%d\n", info->name, info->hobby, info->age,
info->sex);

    saveUser(node->left, file);
    saveUser(node->right, file);
}

//for all users' relationships( here is following tree)
void saveRelation_2(FNode * node, FILE * file, char * following){
    if (!node) return;

    fprintf(file, "%s,%s\n",following, node->info->name);//latter one
is the followed

    saveRelation_2(node->left, file, following);
    saveRelation_2(node->right, file, following);
}
```

```
//this is for all relationships of one user
void saveRelation_1(UNode * node, FILE * file){
    if(!node) return;

    saveRelation_2(node->following, file, node->info->name);

    saveRelation_1(node->left, file);
    saveRelation_1(node->right, file);
}

void saveSystem() {
    printf("请输入存档名（有重名将覆盖原存档）:");
    char * fileName = (char *)malloc(sizeof(char) * 30);
    char * path = (char *)malloc(sizeof(char) * 35);
    fgetsNoN(fileName, 30, stdin);

    strcpy(path, fileName);
    strcat(path, ".u");
    FILE * fileU = fopen(path, "w+");
    if (!fileU) {
        printInfo(1);
    } else {
        strcpy(path, fileName);
        strcat(path, ".r");
        FILE * fileR = fopen(path , "w+");//to save relationship
        saveUser(uRoot, fileU);
        saveRelation_1(uRoot, fileR);

        //close file
        fclose(fileU);
        fclose(fileR);
        printInfo(2);
    }
}

/*-------------Load System-------------*/
```

```
void cleanRelation(UNode * node) {
    if (!node) return;
    destroyAVL_F(node->followed);
    destroyAVL_F(node->following);
}

//load whole system from disk
void loadSystem() {

    printf("请输入存档名:");
    char * fileName = (char *)malloc(sizeof(char) * 30);
    char * path = (char *)malloc(sizeof(char)*35);
    fgetsNoN(fileName, 30, stdin);
    strcpy(path, fileName);
    strcat(path, ".u");
    FILE * fileU = fopen(path, "r+");
    if (!fileU) {
        printf("无以此为名的存档\n");
    } else {
        //clean cache
        cleanRelation(uRoot);
        destroyAVL_U(uRoot);
        uRoot = NULL;

        uRoot = getUsers(fileU);
        strcpy(path, fileName);
        strcat(path, ".r");
        FILE * fileR = fopen(path, "r+");
        getRelation(uRoot, fileR);

        fclose(fileU);
        fclose(fileR);

        //TODO: update heights!!!!

        printInfo(2);
```

```
    }
}

//show those users having same hobby
void showSameHobby(){
    printf("请输入该用户的用户名:");
    char * userName = (char *)malloc(sizeof(char) * 30);
    fgetsNoN(userName, 30, stdin);

    UNode * getUNode = getUNodeFromName_U(userName, uRoot);
    if (!getUNode) {
        printInfo(3);
    } else {
        FNode * resultNode = getSameHobbyUsers(getUNode, uRoot);
        if(resultNode){
            printf("以下为所有相同喜好的用户名单:\n");
            traversePrint(resultNode);printf("\n");
            destroyAVL_F(resultNode);
        }else{
            printInfo(4);
        }
    }
    free(userName);
}

int main() {
    uRoot = NULL;//init
    int op = 1;
    initConsole();
    while (op) {
        system("cls");//clean content on screen
        op = printMenu();

        switch (op) {
        case 1:
            addUser();
```

```
    break;
case 2:
    delUser();
    break;
case 3:
    changeUser();
    break;
case 4:
    showUser();
    break;
case 5:
    addRelation();
    break;
case 6:
    delRelation();
    break;
case 7:
    showFollowing();
    break;
case 8:
    showFollowed();
    break;
case 9:
    showFriends();
    break;
case 10:
    showSecondFriend();
    break;
 case 11:
      showSameHobby();
      break;
case 12:
    showCommonFollowing();
    break;
case 13:
    showCommonFriends();
```

```
            break;
        case 14:
            showCommonFollowed();
            break;
        case 15:
            showUnionFriends();
            break;
        case 16:
            showOwnFriends();
            break;
        case 17:
            saveSystem();
            break;
        case 18:
            loadSystem();
            break;
        case 0:
            printf("欢迎下次使用!\n");
            break;
        }
        system("pause");
    }
    return 0;
}

char *fgetsNoN(char *buf, int bufsize, FILE *stream) {
    char * returnP = fgets(buf, bufsize, stream);
    int i = 0;
    while (buf[i] != '\n') {
        i++;
    }
    buf[i] = '\0';
    return returnP;
}
```

AvlCal.h:

```
#include <AvlUtils.h>
void traInsertAVL(FNode * node);

/*
reNode would lead a AVL-Tree of friends set. GLOBAL
MUST BE FREE after show
*/
FNode * reNode;
FNode * getSetIntersec(FNode * aFNode, FNode * bFNode);
/**
Second friend functions¡£
A new AVL-Tree would be created to save those second-friends and return
it for traversing outside.
*/
void getSecFriend_U2T(FNode * fNode, UNode * uRoot);

FNode * getSecFriend(UNode * uNode, UNode * uRoot) {
    reNode = NULL;
    //traverse this user-tree to call U21-func
    FNode     *     friRoot     =     getSetIntersec(uNode->followed,
uNode->following);//get first-friends
//    traInsertAVL(friRoot);
    getSecFriend_U2T(friRoot, uRoot);
    destroyAVL_F(friRoot);
    reNode = deleteAVL_F(uNode->info->name, reNode);//delete itself
    //now we would get the root node we want
    return reNode;
}
/**
recursion, unode into tree
call this func outside with a root node of friends
*/
void getSecFriend_U2T(FNode * fNode, UNode * uRoot) {
    if (fNode == NULL) {
        return;
    } else {
```

```
        UNode * uNode = getUNodeFromName_U(fNode->info->name, uRoot);
        FNode    *    friRoot    =    getSetIntersec(uNode->followed,
uNode->following);
         traInsertAVL(friRoot);
         destroyAVL_F(friRoot);
        getSecFriend_U2T(fNode->left, uRoot);
        getSecFriend_U2T(fNode->right, uRoot);
    }
}

/**
SET_UNION.
*/
FNode * getSetUnion(FNode * aFNode, FNode * bFNode) {
    reNode = NULL;
    traInsertAVL(aFNode);
    traInsertAVL(bFNode);
    //now we would get the root node we want
    return reNode;
}

/**
insert all nodes into AVL-tree
call this func outside with a root node
*/
void traInsertAVL(FNode * node) {
    if (node == NULL) return;
    else {
        reNode = insertAVL_F(node->info, reNode);//keep updating reNode
        traInsertAVL(node->left);
        traInsertAVL(node->right);
    }
}

FNode * reNodeIntersec = NULL;
/**
```

```
Traverse a FNode-AVL-Tree, check whether it exist in another F-A-L or not.
yes->insert into returning tree
Call this func outside with a root node.
FOR "getSetIntersec"
*/
void getIntersec_check_insert(FNode * node, FNode * sampleRoot) {
    if (node == NULL) {
        return;
    } else {
        FNode * resultNode = getFNodeFromName_F(node->info->name,
sampleRoot);
        if (resultNode) {//exist in another avl tree
            reNodeIntersec = insertAVL_F(resultNode->info,
reNodeIntersec);
        }
        getIntersec_check_insert(node->left, sampleRoot);
        getIntersec_check_insert(node->right, sampleRoot);
    }
}

/**
set_intersection
And the return-FNode should be destroyed outside.
*/

FNode * getSetIntersec(FNode * aFNode, FNode * bFNode) {
    reNodeIntersec = NULL;
    getIntersec_check_insert(aFNode, bFNode);
    return reNodeIntersec;
}



FNode * getSetDifference(FNode * aFNode, FNode * bFNode);
/**
set_equal.
1->equal, 0->not equal
```

```
*/
int setEqual(FNode * aFNode, FNode * bFNode) {
    int flag = 0;
    FNode * resultA = getSetDifference(aFNode, bFNode);
    FNode * resultB = getSetDifference(bFNode, aFNode);
    if (resultA == NULL && resultB == NULL) {
        flag = 1;
    }
    destroyAVL_F(resultA);//free this temporary tree
    destroyAVL_F(resultB);
    return flag;
}

void getSetDiff_recursion(FNode * minusNode, FNode * sampleRoot);
/*
Set_diffrence -> a-b,
Traverse each node of a=tree, find out if it's exist in b-tree or not.
*/
FNode * getSetDifference(FNode * aFNode, FNode * bFNode) {
    reNode = NULL;
    getSetDiff_recursion(aFNode, bFNode);
    return reNode;
}
/**
Traverse a FNode-AVL-Tree, check whether it exist in another F-A-L or not.
yes->insert into returning tree
Call this func outside with a root node to be para "node"
*/
void getSetDiff_recursion(FNode * minusNode, FNode * sampleRoot) {
    if (minusNode == NULL) {
        return;
    } else {
        FNode * resultNode = getFNodeFromName_F(minusNode->info->name,
sampleRoot);
        if (resultNode == NULL) {//not exist
            reNode = insertAVL_F(minusNode->info, reNode);
```

```
        }
        getSetDiff_recursion(minusNode->left, sampleRoot);
        getSetDiff_recursion(minusNode->right, sampleRoot);
    }
}

/**
set_subset
If a is subset of b, which means that a-b=0
@return 1->yes 0->no
*/
int isAsub2B(FNode * aFNode, FNode * bFNode) {
    FNode * resultNode = getSetDifference(aFNode, bFNode);
    int flag = 0;
    if (resultNode) {
        flag = 1;
    }
    destroyAVL_F(resultNode);
    return flag;
}

/**
Set-remove
Logistic : a=a-b
@return a new A-node for updating data
*/
FNode * setRemove(FNode * aFNode, FNode * bFNode) {
    FNode * resultNode = getSetDifference(aFNode, bFNode);
    destroyAVL_F(aFNode);
    return resultNode;
}

/*
set_insert.
Insert a-set into b-set.
WARNING: it would change b-set, if you wanna get a new tree to storage
```

```
those sum-data then use getComFriend
Traverse a FNode-AVL-Tree, and insert all nodes into sample tree
*/

FNode * insertA2B(FNode * aFNode, FNode * sampleRoot) {
    if (aFNode == NULL) {
        return NULL;
    } else {
        sampleRoot = insertAVL_F(aFNode->info, sampleRoot);
        sampleRoot = insertA2B(aFNode->left, sampleRoot);
        sampleRoot = insertA2B(aFNode->right, sampleRoot);
        return sampleRoot;
    }
}

/*
get a avl tree filled with users having same hobby with sampleNode
*/
void getSameHobbyUsers_re(UNode * node, char * sampleHobby) {
    if (!node) return;

    if (strcmp(node->info->hobby, sampleHobby) == 0) {
        reNode = insertAVL_F(node->info, reNode);
    }
    getSameHobbyUsers_re(node->left, sampleHobby);
    getSameHobbyUsers_re(node->right, sampleHobby);
}
FNode * getSameHobbyUsers(UNode * sampleNode, UNode * root) {
    if (sampleNode == NULL) {
        return NULL;
    } else {
        reNode = NULL;
        getSameHobbyUsers_re(root, sampleNode->info->hobby);
        return reNode;
    }
}
```

AvlUtils.h:

```
#include <Structs.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/**
Basic Function For AVL-Trees
*/

//-------------For Follow Node-------------

/**
This func is set for getting a specific Node.
@nullptr means we can't find such a node with this name.
Calling this func when incoming root node as parameters.
THIS IS "SearchAVL"
*/
FNode * getFNodeFromName_F(char * name, FNode * node) {
    if (node == NULL) return NULL;//which means that we can't find such
a node with this name
    else {
        int result = strcmp(name, node->info->name);
        if (result == 0) {
            return node;
        } else if (result < 0) {
            return getFNodeFromName_F(name, node->left);
        } else {//result > 0
            return getFNodeFromName_F(name, node->right);
        }
    }
}

/**
Destroy this tree and free all space it had taken except those user info
*/
```

```
void destroyAVL_F(FNode * node) {
    if (node == NULL) return;
    destroyAVL_F(node->left); node->left = NULL;
    destroyAVL_F(node->right); node->right = NULL;
    free(node);
}

/**
Get a AVL-Tree's size.
@retrun a integer.
*/
int getAvlSize_F(FNode * node) {
    if (node == NULL) return 0;
    return getAvlSize_F(node->left) + getAvlSize_F(node->right) + 1;
}

//This func is only called to reduce checking nullptr when you want to
get height of FNode.
int getHeight_F(FNode * node) {
    if (node) return node->height;
    else return 0;
}

//update parameter node's height.
void updateHeight_F(FNode * node) {
    int leftHeight = 0;
    int rightHeight = 0;
    if (node->left) leftHeight = node->left->height;
    if (node->right) leftHeight = node->right->height;
    node->height = (leftHeight > rightHeight ? leftHeight : rightHeight)
+ 1;
}

//compare keys between a and b( To reduce my typing time outside this func
int compare_F(FNode * a, FNode * b) {
    return strcmp(a->info->name, b->info->name);
```

```
}

//Do left-left rotation, and return a new root for outside to update
FNode * LL_Rotation_F(FNode * oldHead) {
    FNode * oldLeft = oldHead->left;
    oldHead->left = oldLeft->right;
    oldLeft->right = oldHead;

    //change height
    updateHeight_F(oldLeft);
    updateHeight_F(oldHead);

    return oldLeft;
}

//Do right-right rotation, and return a new root for outside to update
FNode * RR_Rotation_F(FNode * oldHead) {
    FNode * oldRight = oldHead->right;
    oldHead->right = oldRight->left;
    oldRight->left = oldHead;

    //change height
    updateHeight_F(oldRight);
    updateHeight_F(oldHead);

    return oldRight;
}

//Do left-right rotation, and return a new root for outside to update
FNode * LR_Rotation_F(FNode * oldHead) {
    oldHead->left = RR_Rotation_F(oldHead->left);
    return LL_Rotation_F(oldHead);
}

//Do right-left rotation, and return a new root for outside to update
FNode * RL_Rotation_F(FNode * oldHead) {
```

```
    oldHead->right = LL_Rotation_F(oldHead->right);
    return RR_Rotation_F(oldHead);
}

/**
@return: new root. Nullptr->something bad happen.
the parameter-newInfo is set for creat new space for new FNode
*/
FNode * insertAVL_F(Info * newInfo, FNode * node) {

    if (node == NULL) {//find a right position to insert this new node
                    //initalize new node
        FNode * newNode = (FNode *)malloc(sizeof(FNode));
        newNode->info = newInfo;
        newNode->left = NULL;
        newNode->right = NULL;//add to be a leave
        newNode->height = 0;
        node = newNode;
    } else {
        int result = strcmp(newInfo->name, node->info->name);

        if (result == 0) {//there is a same name already
            return NULL;
        } else {
            if (result < 0) {
                node->left = insertAVL_F(newInfo, node->left);

                //check balance( the part below would be run for many times
                if (getHeight_F(node->left) - getHeight_F(node->right) ==
2) {
                    if (strcmp(newInfo->name, node->left->info->name) < 0)
{
                        node = LL_Rotation_F(node);//make right rotation
                    } else {
                        node  =  LR_Rotation_F(node);//make  left-riight
rotation
```

```
                }
            }

        } else {//result > 0
            node->right = insertAVL_F(newInfo, node->right);

            //check balance( the part below would be run for many times
            if (getHeight_F(node->right) - getHeight_F(node->left) ==
2) {
                if (strcmp(newInfo->name, node->right->info->name) >
0) {
                    node = RR_Rotation_F(node);//make right rotation
                } else {
                    node  =  RL_Rotation_F(node);//make  left-riight
rotation
                }
            }
        }
    }

    //updata height
    updateHeight_F(node);//would update for enough times
    return node;
}

//get the max FNode in AVL-Tree
FNode * getMaxFNode_F(FNode * node) {
    if (!node) return NULL;
    while (node->right) node = node->right;
    return node;
}

//get the min FNode in AVL-Tree
FNode * getMinFNode_F(FNode * node) {
    if (!node) return NULL;
```

```
    while (node->left) node = node->left;
    return node;
}

/**
Delete a FNode in this AVL-tree with parameter key(user's name). WOUNDN'T
FREE INFO SPACE
@parameter key: the key as a user's name to find the correct FNode
node: sub-tree's root node
@return: a new root node for outside updating
*/
FNode * deleteAVL_F(char * key, FNode * node) {
    if (node == NULL || key == NULL) {
        return NULL;
    }

    int result = strcmp(key, node->info->name);
    if (result < 0) {//go left
        node->left = deleteAVL_F(key, node->left);
        //check balance
        if (getHeight_F(node->right) - getHeight_F(node->left) == 2) {
            if            (getHeight_F(node->right->left)            >=
getHeight_F(node->right->right)) {
                node = RL_Rotation_F(node);
            } else {
                node = RR_Rotation_F(node);
            }
        }
    } else if (result > 0) {//go right
        node->right = deleteAVL_F(key, node->right);
        //check balance
        if (getHeight_F(node->left) - getHeight_F(node->right) == 2) {
            if            (getHeight_F(node->left->right)            >=
getHeight_F(node->left->left)) {
                node = LR_Rotation_F(node);
            } else {
```

```
                node = LL_Rotation_F(node);
            }
        }
    } else {//result == 0 ( found the node
        if (node->left != NULL && node->right != NULL) {
            if (getHeight_F(node->left) > getHeight_F(node->right)) {
                FNode * maxFNode = getMaxFNode_F(node->left);//get the max
node in left sub-tree
                node->info = maxFNode->info;
                node->left      =      deleteAVL_F(maxFNode->info->name,
node->left);
            } else {
                FNode * minFNode = getMinFNode_F(node->right);//get the
min node in right sub-tree
                node->info = minFNode->info;
                node->right     =     deleteAVL_F(minFNode->info->name,
node->right);
            }
        } else {
            FNode * delNode = node;
            if (node->left) node = node->left;
            else node = node->right;
              free(delNode);
        }
    }

    return node;
}

//based on pre-order traverse
void traverseAVL_F(FNode * node) {
    if (!node) return;
    printf("%s;", node->info->name);
    traverseAVL_F(node->left);
    traverseAVL_F(node->right);
}
```

```
//------------------For User Node-------------

/**
This func is set for getting a specific Node.
@nullptr means we can't find such a node with this name.
Calling this func when incoming root node as parameters.
THIS IS "SearchAVL"
*/
UNode * getUNodeFromName_U(char * name, UNode * node) {
    if (node == NULL) return NULL;//which means that we can't find such
a node with this name
    else {
        char * nodeName = node->info->name;
        int result = strcmp(name, nodeName);
        if (result == 0) {
            return node;
        } else if (result < 0) {
            return getUNodeFromName_U(name, node->left);
        } else {//result > 0
            return getUNodeFromName_U(name, node->right);
        }
    }
}

/**
Destroy this tree and free all space it had taken except those user info
*/
void destroyAVL_U(UNode * node) {
    if (node == NULL) return;
    destroyAVL_U(node->left); node->left = NULL;
    destroyAVL_U(node->right); node->right = NULL;
    if (node->info) {
        free(node->info->name);
```

```
        free(node->info);
    }
    free(node);
}

/**
Get a AVL-Tree's size.
@retrun a integer.
*/
int getAvlSize_U(UNode * node) {
    if (node == NULL) return 0;
    return getAvlSize_U(node->left) + getAvlSize_U(node->right) + 1;
}

//This func is only called to reduce checking nullptr when you want to
get height of UNode.
int getHeight_U(UNode * node) {
    if (node) return node->height;
    else return 0;
}

//update parameter node's height.
void updateHeight_U(UNode * node) {
    int leftHeight = 0;
    int rightHeight = 0;
    if (node->left) leftHeight = node->left->height;
    if (node->right) leftHeight = node->right->height;
    node->height = (leftHeight > rightHeight ? leftHeight : rightHeight)
+ 1;
}

//compare keys between a and b( To reduce my typing time outside this func
int compare_U(UNode * a, UNode * b) {
    return strcmp(a->info->name, b->info->name);
}
```

```
//Do left-left rotation, and return a new root for outside to update
UNode * LL_Rotation_U(UNode * oldHead) {
    UNode * oldLeft = oldHead->left;
    oldHead->left = oldLeft->right;
    oldLeft->right = oldHead;

    //change height
    updateHeight_U(oldLeft);
    updateHeight_U(oldHead);

    return oldLeft;
}

//Do right-right rotation, and return a new root for outside to update
UNode * RR_Rotation_U(UNode * oldHead) {
    UNode * oldRight = oldHead->right;
    oldHead->right = oldRight->left;
    oldRight->left = oldHead;

    //change height
    updateHeight_U(oldRight);
    updateHeight_U(oldHead);

    return oldRight;
}

//Do left-right rotation, and return a new root for outside to update
UNode * LR_Rotation_U(UNode * oldHead) {
    oldHead->left = RR_Rotation_U(oldHead->left);
    return LL_Rotation_U(oldHead);
}

//Do right-left rotation, and return a new root for outside to update
UNode * RL_Rotation_U(UNode * oldHead) {
    oldHead->right = LL_Rotation_U(oldHead->right);
    return RR_Rotation_U(oldHead);
```

```
}

/**
Make sure unode is not a nullptr.
@return: new root. Nullptr->something bad happen, but it's hard to capture
outside,
but it would show up at cutting this avl-tree
*/
void printInfo(int mode);
UNode * insertAVL_U(Info * newInfo, UNode * node) {

    if (node == NULL) {//find a right position to insert this new node
        UNode * newNode = (UNode *)malloc(sizeof(UNode));
        if (newNode == NULL) {
            printInfo(1);
            exit(1);
        }
        //initalize new node
        newNode->left = NULL;
        newNode->right = NULL;
        newNode->info = newInfo;
        newNode->followed = NULL;
        newNode->following = NULL;
        newNode->height = 0;
        node = newNode;

    } else {
        int result = strcmp(newInfo->name, node->info->name);

        if (result == 0) {//there is a same name already
            return NULL;
        } else {
            if (result < 0) {
                node->left = insertAVL_U(newInfo, node->left);

                //check balance( the part below would be run for many times
```

```
            if (getHeight_U(node->left) - getHeight_U(node->right) ==
2) {
               if (strcmp(newInfo->name, node->left->info->name) < 0)
{
                   node = LL_Rotation_U(node);//make right rotation
                } else {
                   node  =  LR_Rotation_U(node);//make  left-riight
rotation
                }
            }

        } else {//result > 0
            node->right = insertAVL_U(newInfo, node->right);

            //check balance( the part below would be run for many times
            if (getHeight_U(node->right) - getHeight_U(node->left) ==
2) {
                if (strcmp(newInfo->name, node->right->info->name) >
0) {
                   node = RR_Rotation_U(node);//make right rotation
                } else {
                   node  =  RL_Rotation_U(node);//make  left-riight
rotation
                }
            }
        }
    }



    //updata height
    updateHeight_U(node);
    return node;
}
```

```
//get the max UNode in AVL-Tree
UNode * getMaxUNode_U(UNode * node) {
    if (!node) return NULL;
    while (node->right) node = node->right;
    return node;
}

//get the min UNode in AVL-Tree
UNode * getMinUNode_U(UNode * node) {
    if (!node) return NULL;
    while (node->left) node = node->left;
    return node;
}

/**
Delete a UNode in this AVL-tree with parameter key( user's name)
@parameter key: the key as a user's name to find the correct UNode
node: sub-tree's root node
@return: a new root node for outside updating
*/
UNode * deleteAVL_U(char * key, UNode * node) {
    if (node == NULL || key == NULL) {
        return NULL;
    }

    int result = strcmp(key, node->info->name);
    if (result < 0) {//go left
        node->left = deleteAVL_U(key, node->left);
        //check balance
        if (getHeight_U(node->right) - getHeight_U(node->left) == 2) {
            if (getHeight_U(node->right->left) >=
getHeight_U(node->right->right)) {
                node = RL_Rotation_U(node);
            } else {
                node = RR_Rotation_U(node);
            }
```

```
        }
    } else if (result > 0) {//go right
        node->right = deleteAVL_U(key, node->right);
        //check balance
        if (getHeight_U(node->left) - getHeight_U(node->right) == 2) {
            if (getHeight_U(node->left->right) >= getHeight_U(node->left->left)) {
                node = LR_Rotation_U(node);
            } else {
                node = LL_Rotation_U(node);
            }
        }
    } else {//result == 0 ( found the node
        if (node->left != NULL && node->right != NULL) {
            if (getHeight_U(node->left) > getHeight_U(node->right)) {
                UNode * maxUNode = getMaxUNode_U(node->left);//get the max node in left sub-tree
                node->info = maxUNode->info;
                node->left = deleteAVL_U(maxUNode->info->name, node->left);
            } else {
                UNode * minUNode = getMinUNode_U(node->right);//get the min node in right sub-tree
                node->info = minUNode->info;
                node->right = deleteAVL_U(minUNode->info->name, node->right);
            }
        } else {
            UNode * delNode = node;
            if (node->left) node = node->left;
            else node = node->right;

            free(delNode);
        }
    }
```

```
    return node;
}

//based on pre-order traverse
void traverseAVL_U(UNode * node) {
    if (!node) return;
    printf("%s;", node->info->name);
    traverseAVL_U(node->left);
    traverseAVL_U(node->right);
}

//get depth
int depthMax = 0;
void getDepth_U_re(UNode * node, int i) {
    if (!node) return;
    if (i > depthMax) depthMax = i;
    getDepth_U_re(node->left, i + 1);
    getDepth_U_re(node->right, i + 1);
}
int getDepth_U(UNode * uRoot) {
    depthMax = 0;
    getDepth_U_re(uRoot, 1);
    return depthMax;
}
void getDepth_F_re(FNode * node, int i) {
    if (!node) return;
    if (i > depthMax) depthMax = i;
    getDepth_F_re(node->left, i + 1);
    getDepth_F_re(node->right, i + 1);
}
int getDepth_F(FNode * fRoot) {
    depthMax = 0;
    getDepth_F_re(fRoot, 1);
    return depthMax;
}
```

IO.h:

```
#include <AvlCal.h>
#define NULLCODE 90017
#define INIT_USER_LENGTH 100000
#define INIT_RELATION_LENGTH 1000

// "," would be the split mark
char** setDoubleCharArray(char** out, char* in) {
    if (out == NULL || in[0] == '\0') {
        return NULL;
    }
    register int allIndex = 0;
    register int outIndex = 0;
    register int inIndex = 0;
    char read = 0;

    while ((read = *(in + allIndex)), read != '\n') {
        if (read == ',') {
            out[outIndex][inIndex] = '\0';//end this string
            outIndex++;//reset
            inIndex = 0;
        } else {
            out[outIndex][inIndex] = read;
            inIndex++;
        }
        allIndex++;
    }
    out[outIndex][inIndex] = '\0';//end the final string
    return out;
}

//free stake memory( for the double char array)
void freeDoubleCharArray(int num, char ** info) {
    register int i;
    for (i = 0; i < num; i++) {
        free(info[i]);
```

```
    }
    free(info);

}

char ** readUser(char * in) {
    char** strArray = (char**)malloc(sizeof(char *) * 4);
    strArray[0] = (char*)malloc(sizeof(char) * 30);
    strArray[1] = (char*)malloc(sizeof(char) * 10);
    strArray[2] = (char*)malloc(sizeof(char) * 10);
    strArray[3] = (char*)malloc(sizeof(char) * 10);
    if (setDoubleCharArray(strArray, in) != NULL) {
        return strArray;
    } else {
        freeDoubleCharArray(4, strArray);
        return NULL;
    }
}

char ** readRelation(char * in) {
    char ** strArray = (char**)malloc(sizeof(char *) * 2);
    //former is "follow" split with ";"
    strArray[0] = (char*)malloc(sizeof(char) * 30);
    strArray[1] = (char*)malloc(sizeof(char) * 30);
    if (setDoubleCharArray(strArray, in) != NULL) {
        return strArray;
    } else {
        freeDoubleCharArray(2, strArray);
        return NULL;
    }
}

//get users' info from disk and make them become avl tree
UNode * getUsers(FILE * file) {
    UNode * newRoot = NULL;
```

```
    UNode * unode = (UNode*)malloc(sizeof(UNode));
    unode->followed = NULL; unode->following = NULL; unode->left = NULL;
unode->right = NULL;
    Info * info = (Info *)malloc(sizeof(Info));
    info->name = (char *)malloc(sizeof(char) * 30);
    info->hobby = (char *)malloc(sizeof(char) * 10);
    unode->info = info;

    char StrLine[200];//one line one user
    fgets(StrLine, 200, file);
    char ** property = readUser(StrLine);
    if (!property) {
        free(info->name);
        free(info->hobby);
        free(info);
        return NULL;
    }
    strcpy(info->name, property[0]);
    strcpy(info->hobby, property[1]);
    info->age = atoi(property[2]);
    info->sex = atoi(property[3]);
    freeDoubleCharArray(4, property);

    newRoot = unode;

    while (1) {
        info = (Info *)malloc(sizeof(Info));

        info->name = (char *)malloc(sizeof(char) * 30);
        info->hobby = (char *)malloc(sizeof(char) * 10);

        char StrLine[200];//one line one user
        fgets(StrLine, 200, file);

        if (feof(file)) {
            free(info->name);
```

```
            free(info->hobby);
            free(info);
            break;
        }

        char ** property = readUser(StrLine);
        if (!property) return NULL;
        strcpy(info->name, property[0]);
        strcpy(info->hobby, property[1]);
        info->age = atoi(property[2]);
        info->sex = atoi(property[3]);
        freeDoubleCharArray(4, property);

        newRoot = insertAVL_U(info, newRoot);

    }

    return newRoot;
}

//addRelation
void getRelation(UNode * root, FILE * file) {
    char StrLine[200];//one line one user
    fgets(StrLine, 200, file);
    char ** property = readRelation(StrLine);
    if (!property) return;
    UNode * followingNode = getUNodeFromName_U(property[0], root);
    UNode * followedNode = getUNodeFromName_U(property[1], root);
    if(followedNode == NULL || followingNode == NULL ||
followedNode->info == NULL || followingNode->info == NULL){
        followingNode->following = insertAVL_F(followedNode->info,
followingNode->following);
        followedNode->followed = insertAVL_F(followingNode->info,
followedNode->followed);
    }
    freeDoubleCharArray(2, property);
```

```
    while (1) {
        char StrLine[200];//one line one user
        fgets(StrLine, 200, file);

        if (feof(file)) {
            break;
        }

        char ** property = readRelation(StrLine);
        if (!property) return;
        UNode * followingNode = getUNodeFromName_U(property[0], root);
        UNode * followedNode = getUNodeFromName_U(property[1], root);
        if(followedNode == NULL || followedNode == NULL) continue;
         if(followedNode->info == NULL || followingNode->info == NULL)
continue;
        followingNode->following   =   insertAVL_F(followedNode->info,
followingNode->following);
        followedNode->followed    =    insertAVL_F(followingNode->info,
followedNode->followed);
        freeDoubleCharArray(2, property);

    }
}

//generate heights in avl trees including Users and all follow-trees
void generateHeight_F(FNode * node) {
    if (!node) return;
    node->height = getDepth_F(node);

    generateHeight_F(node->left);
    generateHeight_F(node->right);
}
void generateHeight_U(UNode * node) {
    if (!node) {
        return;
```

```
    }
    node->height = getDepth_U(node);

    generateHeight_F(node->followed);
    generateHeight_F(node->following);

    generateHeight_U(node->left);
    generateHeight_U(node->right);
}
//Overload
void genHeightAfterLoad(UNode * uRoot) {
    generateHeight_U(uRoot);
}
```

Structs.h:

```
/**
Basic Data Structre
*/

/**
Info is a struct including all infomations of one user. One user would have only one
real-object.
*/
typedef struct info {
    char * name;//length 30, and this would be the key to find user
    char * hobby;//length 10
    int age;
    int sex;//1->man 2->woman
            //and othe infomations
} Info;

/**
FNode here means those node in AVL-Trees of users' follow and followed.
*/
typedef struct fNode {
    Info * info;
    int height;
    struct fNode * left;
```

```
    struct fNode * right;
} FNode;

/**
UserNode would be used to set up a AVL-Tree to manage all users without any relationships among
*/
typedef struct userNode {
    Info * info;
    int height;
    FNode * following;
    FNode * followed;
    struct userNode * left;
    struct userNode * right;
} UNode;
```