

华中科技大学

课程设计报告

题目： 基于 AVL 树表示的
集合 ADT 实现与应用

课程名称： 数据结构课程设计
专业班级： 物联网 1601
学 号： U201614897
姓 名： 潘越
指导教师： 李剑军
报告日期： 2018 年 3 月 1 日

计算机科学与技术学院

项目 github 地址： <https://github.com/zxc479773533/DS-curriculum-design>

目录

任 务 书.....	I
1 引言.....	1
1.1 课题背景与意义.....	1
1.2 国内外研究现状.....	1
1.3 课程设计的主要研究工作.....	1
1.3.1 任务提炼.....	1
1.3.2 任务分析.....	2
2 系统需求分析与总体设计.....	3
2.1 系统需求分析.....	3
2.2 系统总体设计.....	3
2.3 项目架构.....	4
3 算法库详细设计.....	5
3.1 有关数据结构的定义.....	5
3.1.1 AVL 树的数据结构定义.....	5
3.1.2 哈希表的数据结构定义.....	5
3.1.3 集合 Set 的数据结构定义.....	6
3.1.4 多种数据结构之间的联系.....	7
3.2 主要算法设计.....	8
3.2.1 AVL 树核心操作算法.....	8
①平衡操作（旋转）.....	8
②寻找最大/最小值.....	12
③查找.....	13
④高度计算.....	14
⑤插入.....	15
⑥删除.....	17
⑦avltree 库所有 API 展示.....	20
3.2.2 Set 运算实现算法.....	21
①两集合的交.....	21
②两集合的并.....	21
③两集合的差.....	22

④判断子集	22
⑤判断两个集合是否相等	23
⑥set 库所有 API 展示	23
3.2.3 哈希表实现算法.....	24
①哈希函数	24
②冲突解决方法	24
③hashtable 库所有 API 展示	27
4 系统实现与测试.....	28
4.1 系统实现.....	28
4.1.1 开发环境.....	28
4.1.2 用户界面设计.....	28
①Start_Shell 函数	28
②eval 函数	29
③parseline 函数	30
④getopt 函数介绍	31
4.1.3 应用层设计.....	31
①内建命令的设计	31
②应用层命令的设计	32
③应用层函数原型	32
4.1.4 文件存储设计.....	33
4.1.5 编译处理.....	34
4.2 系统测试.....	34
4.2.1 Google Test 软件单元测试方法介绍.....	34
4.2.2 软件测试大纲.....	35
①运行时间测试	35
②较少次随机数据测试	36
③大量随机数据测试	36
④单一数据测试	37
⑤顺序数据测试	38
⑥重复数据测试	38
⑦复杂多次随机数据测试	39
4.2.3 测试结果与分析.....	40

4.3 系统运行展示.....	41
5 开源说明.....	49
5.1 声明.....	49
5.2 开源协议.....	49
6 总结与展望.....	50
6.1 课程设计总结.....	50
6.1.1 工作总结.....	50
6.1.2 系统的优点.....	50
6.1.3 系统的不足.....	50
6.2 工作展望.....	51
7 体会与感想.....	52
附录：程序源代码.....	53
参考文献.....	108

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

1 引言

1.1 课题背景与意义

在计算机科学中，AVL 树是最先发明的自平衡二叉查找树。在 AVL 树中任何节点的两个子树的高度最大差别为 1，所以它也被称为高度平衡树。查找、插入和删除在平均和最坏情况下的时间复杂度都是 $O(\log n)$ 。增加和删除可能需要通过一次或多次树旋转来重新平衡这个树。AVL 树得名于它的发明者 G. M. Adelson-Velsky 和 E. M. Landis，他们在 1962 年的论文《An algorithm for the organization of information》中发表了它。

节点的平衡因子是它的左子树的高度减去它的右子树的高度（有时相反）。带有平衡因子 1、0 或 -1 的节点被认为是平衡的。带有平衡因子 -2 或 2 的节点被认为是不平衡的，并需要重新平衡这个树。平衡因子可以直接存储在每个节点中，或从可能存储在节点中的子树高度计算出来。

平衡二叉树在实际中多用于在内存中组织数据，它在动态查找表中的查找效率非常高，在地理信息处理、医学、模型处理以及快速成型技术中都有广泛的应用。

1.2 国内外研究现状

当前国内外对于平衡二叉树的研究基本上是对它的扩展和改进，如伸展树，B-树，B+树还有红黑树等等。相关的操作处理和算法改进也基本都有了一定的成果，特别是简化操作和快速构建等方面有诸多论文。不同的数据结构在插入删除，查找，存储等等各个方面都各有各的优点和不足。

1.3 课程设计的主要研究工作

1.3.1 任务提炼

首先实现平衡二叉树的创建，然后通过平衡二叉树表示集合，并能够实现集合的插入删除交并差等各种运算，最后利用集合实现应用层的操作并设计一个用户友好的界面。同事还要设计好数据存储的格式，搜集一定规模的数据集用于测试。

1.3.2 任务分析

平衡二叉树又称 AVL 树。它或者是一棵空树,或者是具有下列性质的二叉树:它的左子树和右子树都是平衡二叉树,且二叉树上的所有结点的平衡因子绝对值不超过 1。

在平衡二叉树上插入或删除结点后,可能使树失去平衡,因此,需要对失去平衡的树进行平衡化调整。分别针对不同失衡结构采用 LL 旋转, LR 旋转, RL 旋转, RR 旋转四种旋转方法。而其他的操作都是建立在如何保证实现平衡,所以这也是整个课设的难点及重点。

此外对于用户本身来说,还应该设计一个数据结构来存储用户 ID-用户昵称这样的键值对,这里我选择了哈希表作为存储结构,其优秀的查找效率为之后的频繁获取用户昵称节省了很大的开销。

集合 Set 的实现只需要调用上一层实现的 AVL 树的接口即可,最后外部套一层用户界面,调用 Set 的接口就完整的实现了课程设计的要求。

2 系统需求分析与总体设计

2.1 系统需求分析

系统应该能够实现管理所有注册的用户，管理用户之间的关系，对好友、关注、粉丝实现增删改查。并且在此基础上实现高一层的查询功能，如共同好友、二度好友等等的查询，模拟微博后台的各种功能。

2.2 系统总体设计

系统设计分为三部分，数据结构库，应用层调用，用户界面。

数据结构库中实现了一个哈希表，用于存储用户相关的属性，如姓名，爱好等。实现了AVL树及其基本操作，并通过调用AVL树的基本的接口实现了集合set。

应用层通过调用set的接口实现各种对用户好友关注粉丝等的存储，管理，查询等等操作。

用户界面仿照linux系统的shell编写，执行help命令可以查看用户手册，通过执行各种相应的命令行，来实现应用层的各种功能操作。

系统和数据的设计模式如下图：

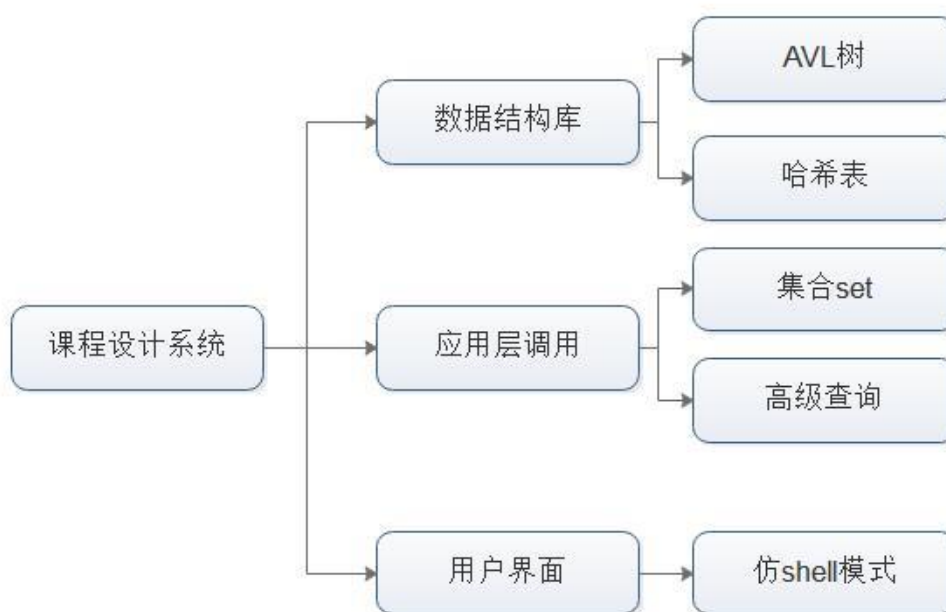


图 1 课程设计系统设计模式

2.3 项目架构

整个项目的文档目录树如下图：

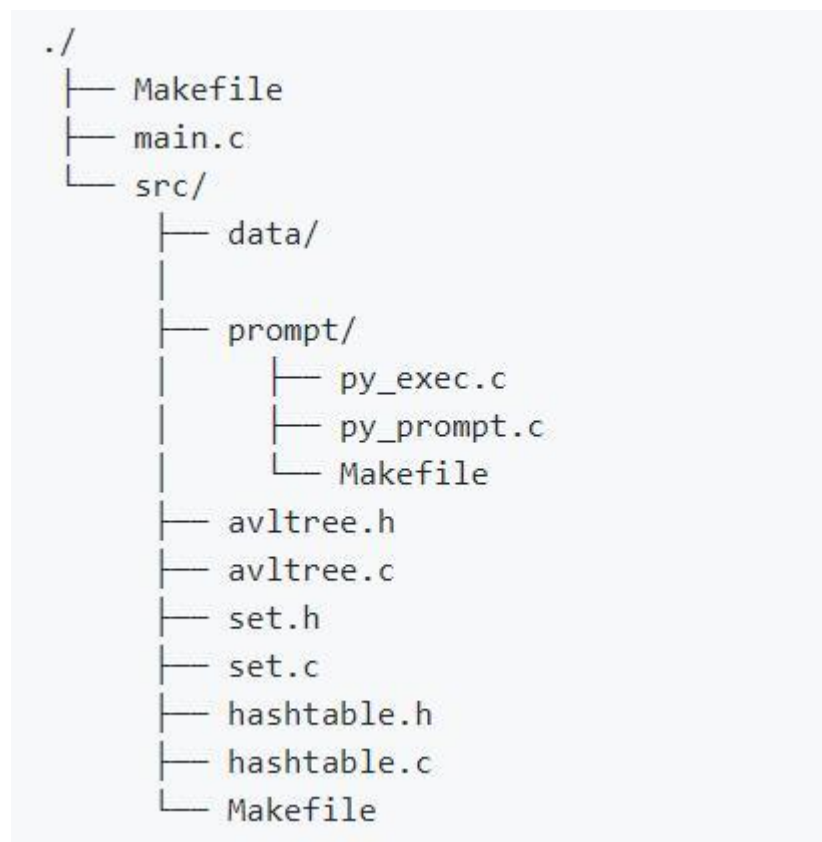


图 2 课程设计系统设计模式

各文件说明：

main.c：主函数所在的文件，调用 `Start_Shell` 函数启动程序

src：系统源码文件夹

data：存放数据的文件夹

prompt：用户界面文件夹

py_exec.c：构建应用层，调用接口

py_prompt.c：构建用户界面

avltree.h：AVL 树头文件，函数原型

avltree.c：AVL 树库函数具体实现

set.h：Set 头文件，函数原型，调用 AVL 树接口

set.c：Set 库函数具体实现

hashtable.h：哈希表头文件，函数原型

hashtable.c：哈希表库函数具体实现

Makefile：自动化编译文件

3 算法库详细设计

3.1 有关数据结构的定义

3.1.1 AVL 树的数据结构定义

1. AVL 树结构体

```
/* The tree struct */
typedef struct AVL_tree {
    int kind;          /* tree kind */
    int id;            /* tree id */
    int num;           /* node num */
    AVL_node *root;    /* root node */
} AVL_tree;
```

2. AVL 树结点

```
/* The AVL node struct */
typedef struct AVL_node {
    int key;           /* node key */
    int height;        /* node height */
    struct AVL_node *left_child;
    struct AVL_node *right_child;
} AVL_node;
```

说明：AVL树采用二叉链表的形式实现，为了简化存储，所有结点中只设置一个int类型的值作为用户的ID，此后所有涉及用户其他信息的操作则借助哈希表来实现。

此外相比普通二叉树多设计的height属性，是专为平衡而设计的，这样可以方便的查看左右子树是否平衡

AVL树结构体里的kind属性则是表明集合的属性（应用层中的好友集、关注集、粉丝集），通过宏定义的0，1，2来代替。

3.1.2 哈希表的数据结构定义

1. 哈希表结构体

```
/* The hashtable struct */
```

```
typedef struct HashTable {
    int num;
    int hashlen;
    HashNode **Hash;
} HashTable;
```

2. 哈希表结点

```
/* The hash node struct */
typedef struct HashNode {
    int count;
    int key;
    char value[18];
    struct HashNode *next;
} HashNode;
```

说明：哈希表内存存储着int-char[]的键值对，可以表示用户ID-用户昵称和用户ID-用户个人爱好两种情况。算法上采用简单的取模寻址法，采用链式结构处理冲突，在查找时有着非常高的效率。

count成员代表结点个数，在统计词频这种应用中会发挥很多作用，但是在本系统中由于限制了用户的唯一性，没有用到这个属性。

哈希表结构体中有的只是一个指向哈希表数组的指针，便于自定义哈希表的长度，增加了普适性。本系统中考虑到验收检查时候的数据量，采用了长度为128的哈希表

3.1.3 集合 Set 的数据结构定义

```
typedef struct Set {
    AVL_tree *Elem;
} Set;
```

说明：集合Set单纯地在AVL树外面套了一层外壳，封装一层的原因是本着部在最高层调用底层数据结构的原则，所有的应用层调用全部使用Set结构体和相关操作，使代码清洗明了。

附：数据结构汇总表

表 1 数据结构汇总表

结构体	成员	含义	示例
AVL_tree	int kind	AVL 树的种类	FRIENDS (0)
	int id	用户 ID	1001
	int num	树的结点个数	15
	AVL_node *root	指向根结点的指针	--
AVL_node	int key	用户 ID	1001
	int height	该结点的高度	3
	AVL_node *left_child	左儿子	--
	AVL_node *right_child	右儿子	--
Hashtable	int num	哈希表结点个数	58
	int hashlen	哈希表长度	128
	HashNode **Hash	哈希结点数组指针	--
HashNode	int count	该结点重复次数	1
	int key	用户 ID	1001
	char value[18]	用户昵称/喜好	Tom/Basketball
	HashNode *next	指向下一个结点	--
Set	AVL_tree *Elem	指向 AVL 树	--

3.1.4 多种数据结构之间的联系

多种数据结构直接的关系由下图清晰可示:

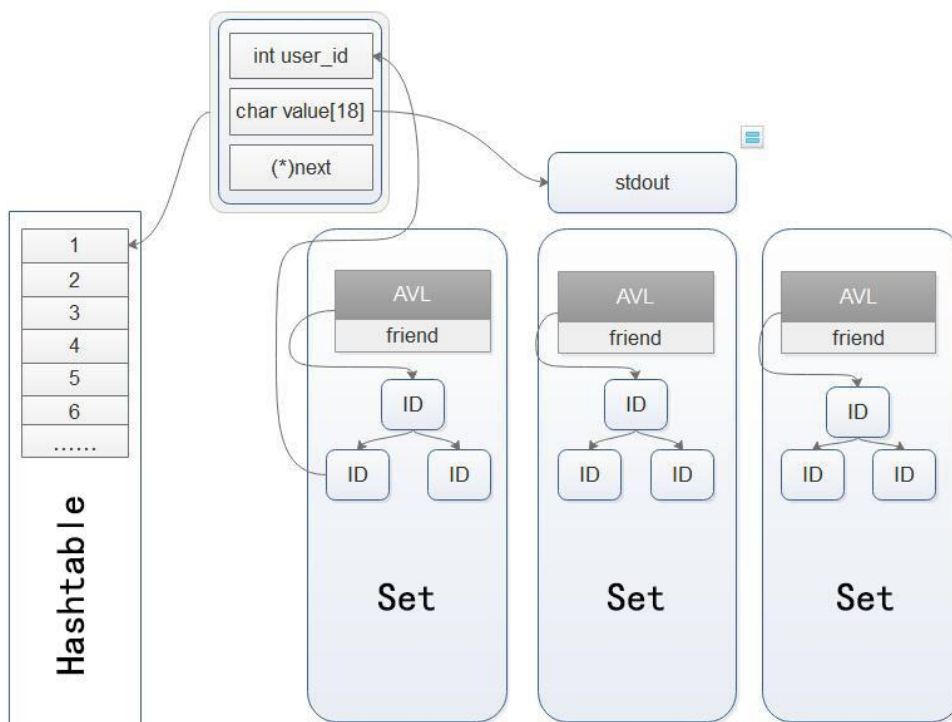


图 3 多种数据结构之间的关系

说明：AVL中存储的只有用户ID，所有的操作基于整形的ID来执行，凡是涉及到用户昵称的场景，则打开哈希表，通过ID找到用户的昵称并输出。

3.2 主要算法设计

3.2.1 AVL 树核心操作算法

①平衡操作（旋转）

对于AVL树的任意一个结点，假设其发生了失衡，则有以下四种情况：

- （1）在A的左儿子的左子树上进行了一次插入
- （2）在A的左儿子的右子树上进行了一次插入
- （3）在A的右儿子的左子树上进行了一次插入
- （4）在A的右儿子的右子树上进行了一次插入

这里显然（1）和（4）是对称的，（2）和（3）是对称的，我们只需要讨论其中两种情况的算法即可。

我们称（1）、（4）为单旋转，因为只需要旋转一次；称（2）、（3）为双旋转，需要旋转两次。

两种单旋转的示意图如下：



图4 LL单旋转平衡

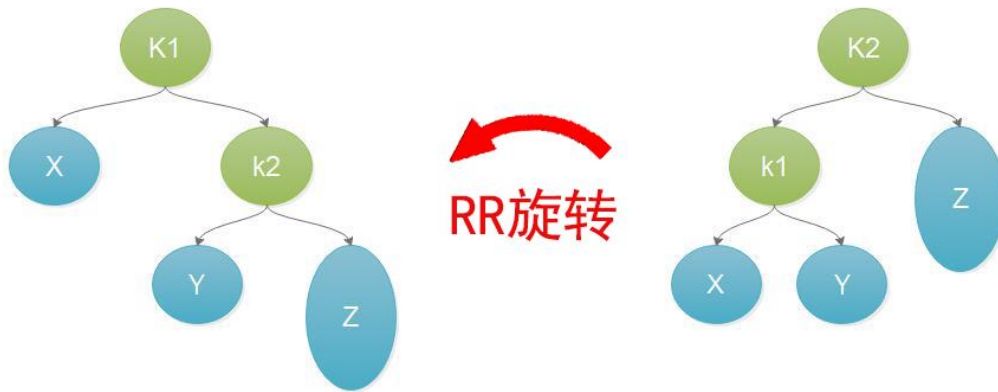


图5 RR单旋转平衡

两种操作看上去就像是进行了一次顺时针选择和逆时针旋转一样，这也就是其“旋转”操作得名的原因。实现起来非常简单，只需要简单的指针改变即可，代码如下：

```

/*
 * leftleft_rotate -> do a left-left rotate
 *
 *      k1          k2
 *      / \        / \
 *      k2  ##      ## k1
 *      / \        / \
 *      ##  @@      @@  ##
 *
 * return <- k2, the root after rotated
 */
AVL_node* leftleft_rotate(AVL_node *k1) {
    AVL_node *k2 = k1->left_child;
    k1->left_child = k2->right_child;
    k2->right_child = k1;
    k1->height = 1 + Max(avl_height(k1->left_child),
        avl_height(k1->right_child));
    k2->height = 1 + Max(avl_height(k2->left_child),
        avl_height(k2->right_child));
    return k2;
}
    
```

```

/*
 * rightright_rotate -> do a right-right rotate
 *
 *      k1                k2
 *     / \              / \
 *    ## k2  ----->  k1 ##
 *     / \              / \
 *    @@  ##           ##  @@
 *
 * return <- k2, the root after rotated
 */
AVL_node* rightright_rotate(AVL_node *k1) {
    AVL_node *k2 = k1->right_child;
    k1->right_child = k2->left_child;
    k2->left_child = k1;
    k1->height = 1 + Max(avl_height(k1->left_child),
        avl_height(k1->right_child));
    k2->height = 1 + Max(avl_height(k2->left_child),
        avl_height(k2->right_child));
    return k2;
}

```

两种双旋转的示意图如下，这实际上是两种单旋转的组合操作：

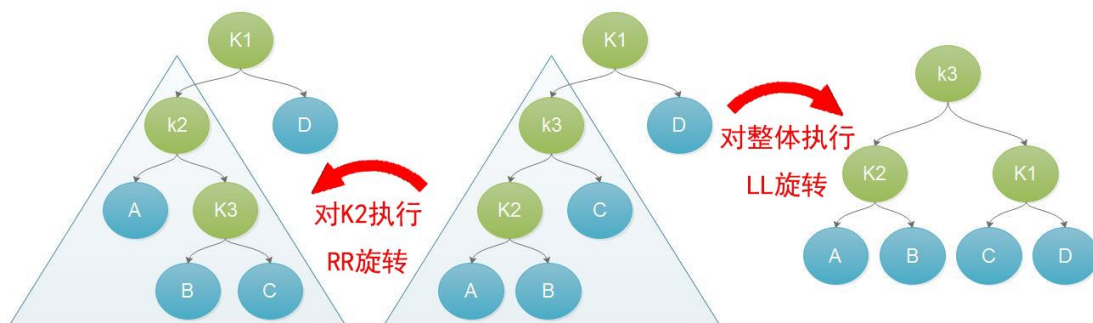


图6 LR双旋转平衡

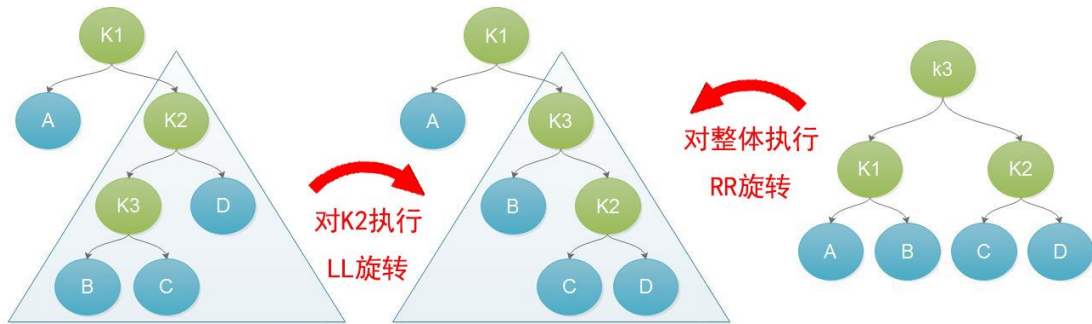


图7 RL双旋转平衡

这两种双旋转分别都是由两种单旋转组合而成，因此实现起来也十分简单，代码如下：

```

/*
 * leftright_rotate -> do a left-right rotate
 *
 *      k1                      k1                      k3
 *      / \                    / \                    / \
 *      k2  ##                 k3  ##                 k2   k1
 *      / \                   / \                   / \   / \
 *      ## k3                k2 %%                 ## @@  %% ##
 *      / \                   / \
 *      @@  %%                ##  @@
 *
 * return <- k3, the root after rotated
 */
AVL_node* leftright_rotate(AVL_node *k1) {
    k1->left_child = rightright_rotate(k1->left_child);
    return leftleft_rotate(k1);
}
    
```

```

/*
 * rightleft_rotate -> do a right-left rotate
 *
 *      k1                      k1                      k3
 *      / \                    / \                    / \
 *      ## k2                 ## k3
    
```

```

*      /  \  ----->      /  \  ----->      k1    k2
*      k3  ##                @@  k2          /  \    /  \
*      /  \                  /  \          ##  @@  %%  ##
*      @@  %%                %%  ##
*
* return <- k3, the root after rotated
*/
AVL_node* rightright_rotate(AVL_node *k1) {
    k1->right_child = leftleft_rotate(k1->right_child);
    return rightright_rotate(k1);
}

```

通过这四种旋转的图示我们可以看到，在经过旋转之后，原本失去平衡的二叉树得到了平衡，因此，我们只需要在插入或是删除操作执行后，对二叉树执行选择操作使其平衡即可。所有的旋转操作算法时间复杂度均为 $O(1)$ 数量级。

②寻找最大/最小值

由于二叉查找树的性质，对任意一个结点，其左子树上的所有结点的key比该结点的key小，其右子树上的所有结点的key比该结点大，于是我们我们可以轻松的写出这个操作——一直向左或者向右查找即可。这一操作是插入和删除的基础操作，算法的图示和实现的代码如下：

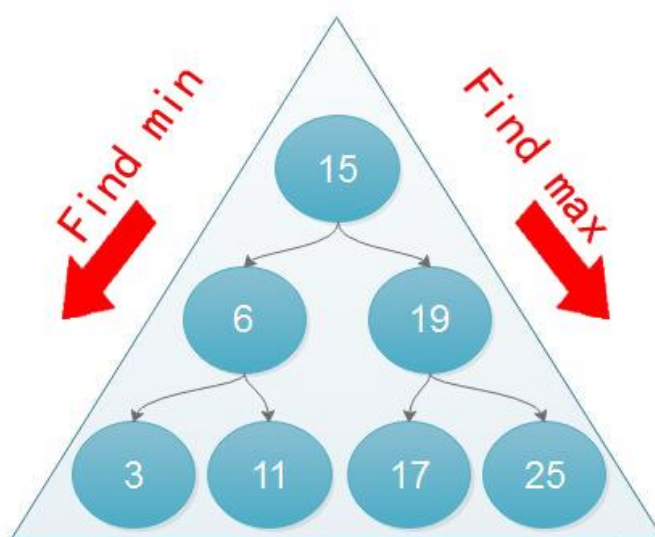


图8 寻找最大最小值

```

/*
 * find_min -> find the min node in an avl tree
 *
 * return <- the ptr to the node
 */
AVL_node* find_min(AVL_node *tree) {
    /* Find it */
    if (tree->left_child == NULL)
        return tree;
    /* Find in left child tree */
    return find_min(tree->left_child);
}

/*
 * find_max -> find the max node in an avl tree
 *
 * return <- the ptr to the node
 */
AVL_node* find_max(AVL_node *tree) {
    /* Find it */
    if (tree->right_child == NULL)
        return tree;
    /* Find in right child tree */
    return find_max(tree->right_child);
}

```

③查找

查找高效是AVL树最优秀的特性，正因为AVL树高度的平衡特性，其在最差和平均的情况下查找的时间复杂度都是 $O(\log n)$ 级别的，其中 n 为AVL树的高度。

这是因为，AVL树由其平衡特性，高度非常接近 $\text{floor}(\log(n) + 1)$ ，假设任意结点被查找的概率均等，即可计算出其平均时间复杂度为 $O(\log n)$ 级别。

这里我们实现查找的方式和普通的二叉查找树相同，故不在赘述，其实现的代码如下：

```

/*
 * search_avl -> find a node in an avl tree
 *
 * return <- the ptr to the node
 */
AVL_node* search_avl(AVL_node *tree, int key) {
    /* Find it */
    if (tree == NULL)
        return NULL;
    /* Find in child tree */
    if (tree->key == key)
        return tree;
    else if (tree->key > key)
        return search_avl(tree->left_child, key);
    else
        return search_avl(tree->right_child, key);
}

```

④高度计算

高度的计算对于AVL树来说也是非常重要的核心操作，其虽然简单，但却维持这整个树的平衡因子，这里我已经把他给设置为了AVL树结点的一个属性，在每次插入和删除后都及时递归进行所有结点的高度计算，这样也方便了插入和删除后确认AVL树是否失衡。

高度的计算使用“该结点高度” = 1 + “左子树高度” + “右子树高度”的公式来计算，代码实现如下：

```

/*
 * avl_height -> get height of an avl tree
 *
 * return <- the height
 */
int avl_height(AVL_node *tree) {
    if (tree == NULL)
        return 0;
    else
        return 1 + Max(avl_height(tree->left_child),
            avl_height(tree->right_child));
}

```

}

⑤插入

插入操作在有了旋转之后就变得十分简单了，我们只需要执行一次类似查找操作一样的递归深入，插入结点，然后针对失衡的情况执行对应的旋转操作即可。

失衡情况还是如上提到的四种：

- (1) 在A的左儿子的左子树上进行了一次插入
- (2) 在A的左儿子的右子树上进行了一次插入
- (3) 在A的右儿子的左子树上进行了一次插入
- (4) 在A的右儿子的右子树上进行了一次插入

其中，对于失去平衡的判断我们可以用如下语句，利用高度差2进行：

```
avl_height(tree->left_child) - avl_height(tree->right_child) == 2
```

在插入操作的最后，不要忘记进行插入后的高度计算，这个计算操作会随着函数的递归返回一层层更新所有结点的高度属性。

函数的时间复杂度和查找相同，仍然为 $O(\log n)$ 。

具体实现的代码如下，我们重点关注判断四种情况，并执行四种旋转：

```
/*
 * insert_avl -> insert a node in avl tree
 *
 * using recursive function, settle height and balance
 * every return
 *
 * return <- the root node after balanced
 */
AVL_node* insert_avl(AVL_node *tree, int key, int *safe_tag) {
    /* Create new node */
    if (tree == NULL) {
        tree = (AVL_node*)malloc(sizeof(AVL_node));
        tree->key = key;
        tree->height = 0;
        tree->left_child = NULL;
        tree->right_child = NULL;
    }
    /* Insert in left child tree */
    else if (key < tree->key) {
```

```

    tree->left_child    =    insert_avl(tree->left_child,    key,
safe_tag);
    /* Balance it */
    if                (avl_height(tree->left_child)                -
avl_height(tree->right_child) == 2) {
        /* Insert left-left */
        if (key < tree->left_child->key)
            tree = leftleft_rotate(tree);
        /* Insert left right */
        else
            tree = leftright_rotate(tree);
    }
}
/* Insert in right child tree */
else if (key > tree->key) {
    tree->right_child    =    insert_avl(tree->right_child,    key,
safe_tag);
    /* Balance it */
    if                (avl_height(tree->right_child)                -
avl_height(tree->left_child) == 2) {
        /* Insert right-left */
        if (key < tree->right_child->key)
            tree = rightleft_rotate(tree);
        /* Insert right-right */
        else
            tree = rightright_rotate(tree);
    }
}

/* If the key exists, let the safe tag be zero */
else {
    *safe_tag = 0;
}

/* Count height */
tree->height    =    1    +    Max(avl_height(tree->left_child),
avl_height(tree->right_child));

return tree;

```


}

⑥删除

删除操作分为三种情况，针对三种情况有着不同的处理方法。

首先先讲平衡的方法，将删除看作是插入的逆过程，在执行一次删除后，通过判断高度差，可以将四种删除的情况转换为插入的四种情况，具体的转换方式如下图：

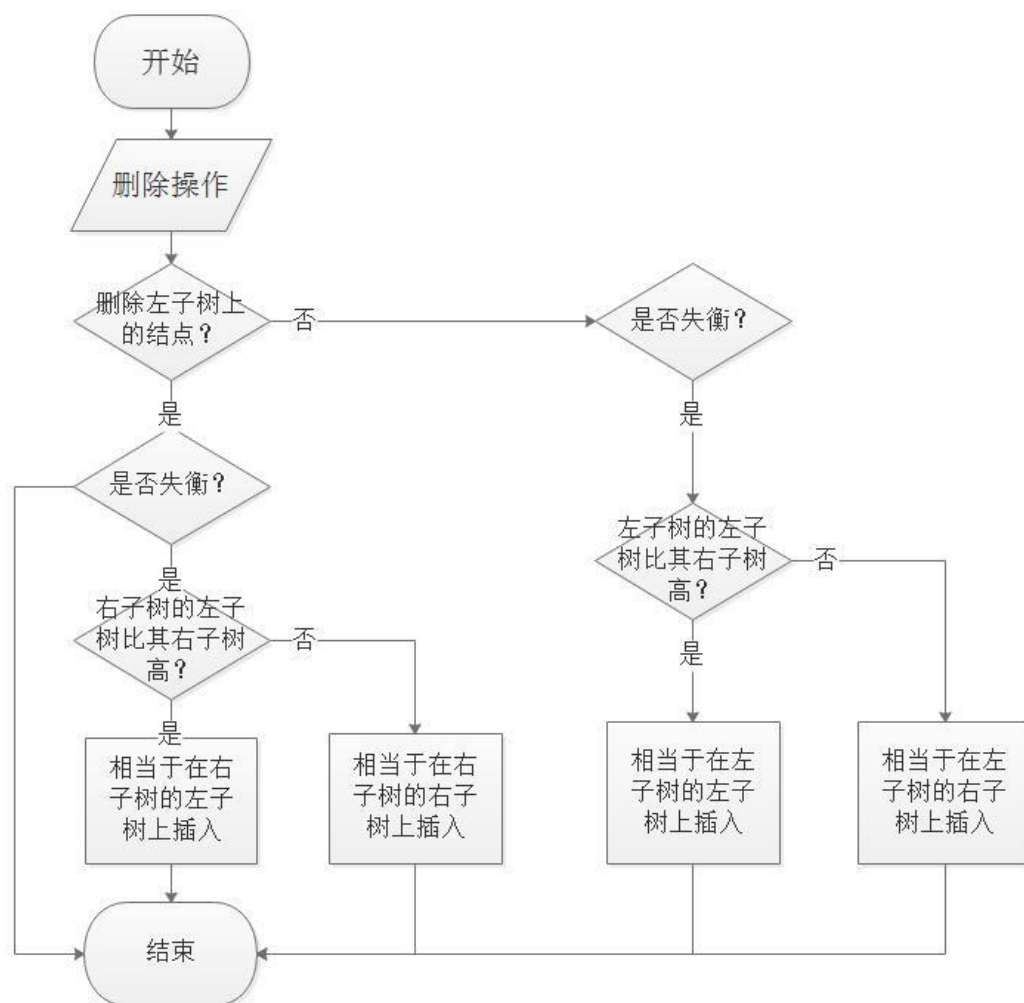


图9 删除和插入的情形转换

有了这种转换之后，我们就可以在删除之后利用旋转操作来平衡二叉树了。

具体的删除情况如下：

- (1) 删除的是叶子结点（度为0）
- (2) 删除的是度为1的结点
- (3) 删除的是度为2的结点

首先第一种情况下，我们直接删除就好，接下来进行平衡调整。

对于第二种情况，我们只需要将其唯一的子结点代替它即可，接下来进行平衡调整。

稍微麻烦一点的是第三种情况，此时我们需要将其左子树上最小的结点或是右子树上最大的结点拿来代替它，紧接着删去左子树上最小的结点或是右子树上最大的结点即可，这里我选择的是删去左子树上最小的结点。同时代替操作实际上没有这么复杂，我们选用赋值key值过来就可以了。做完这一步后同样进行平衡调整。

综合起来删除操作的时间复杂度仍然为 $O(\log n)$ ，主要还是查找过程。

整个删除过程实现的代码如下：

```
/*
 * delete_avl -> delete a node in avl tree
 *
 * Three conditions
 * 1. The node is leaf, just delete it
 * 2. The node has two node, replaced it by the max/min node in child
tree
 * 3. The node has ome node, replaced it by its child
 *
 * return <- the root node after balanced
 */
AVL_node* delete_avl(AVL_node *tree, AVL_node *node) {
    /* Nothing to delete */
    if (tree == NULL || node == NULL)
        return NULL;
    /* Delete in left child tree */
    if (node->key < tree->key) {
        tree->left_child = delete_avl(tree->left_child, node);
        /* Balanced it */
        if (avl_height(tree->right_child) -
avl_height(tree->left_child) == 2) {
            /* The same as insert right-left */
            if (avl_height(tree->right_child->left_child) >
avl_height(tree->right_child->right_child))
                tree = rightleft_rotate(tree);
            /* The same as insert right-right */
        }
        else
    }
```

```

        tree = rightright_rotate(tree);
    }
}
/* Delete in right child tree */
else if (node->key > tree->key) {
    tree->right_child = delete_avl(tree->right_child, node);
    /* Balanced it */
    if (avl_height(tree->left_child) -
avl_height(tree->right_child) == 2) {
        /* The same as left-left insert */
        if (avl_height(tree->left_child->left_child) >
avl_height(tree->left_child->right_child))
            tree = leftleft_rotate(tree);
        /* The same as left-right insert */
        else
            tree = leftright_rotate(tree);
    }
}
/* Found the node and delete it */
else {
    /* If the node is leaf */
    if (!(tree->left_child || tree->right_child)) {
        AVL_node* tmp = tree;
        tree = NULL;
        free(tmp);
    }
    /* If the node has two children */
    else if (tree->left_child && tree->right_child) {
        /* Left child tree higher */
        if (avl_height(tree->left_child) >
avl_height(tree->right_child)) {
            AVL_node *max_node = find_max(tree->left_child);
            tree->key = max_node->key;
            tree->left_child = delete_avl(tree->left_child,
max_node);
        }
        /* Right child tree higher */
        else {
            AVL_node *min_node = find_min(tree->right_child);

```

```

        tree->key = min_node->key;
        tree->right_child = delete_avl(tree->right_child,
min_node);
    }
}
/* If the node has one child */
else {
    AVL_node *tmp = tree;
    tree = (tree->left_child != NULL) ? tree->left_child :
tree->right_child;
    free(tmp);
}
}

return tree;
}

```

总的来说 AVL 树核心操作的算法就如上所示,更多的辅助函数操作等等详见附录的代码中,在此不在赘述。

⑦avltree 库所有 API 展示

根据任务书文档和具体的功能需求,设计九个接口如下:

```

/* APIs */
extern AVL_tree* Init_AVL(int kind, int id); //初始化 AVL 树
extern void Destroy_AVL(AVL_tree *AVL); //销毁 AVL 树
extern void Clear_AVL(AVL_tree *AVL); //清空 AVL 树
extern int Search_AVL(AVL_tree *AVL, int key); //在 AVL 树中查找
extern void Insert_AVL(AVL_tree *AVL, int key); //在 AVL 树中插入
extern void Delete_AVL(AVL_tree *AVL, int key); //在 AVL 树中删除
extern void Traverse_AVL(AVL_tree *AVL, int option, void
(*visit)(int)); //遍历 AVL 树
extern int Save_AVL(AVL_tree *AVL, const char *path); //保存 AVL
树
extern int Load_AVL(AVL_tree *AVL, const char *path); //删除 AVL
树

```

3.2.2 Set 运算实现算法

Set 的实现根据任务书文档要求，完全调用 avltree 库接口实现，大部分为直接封装一层的结果，这里只追加说明 Set 层面上独有的运算操作：

①两集合的交

这里我将交集设置为返回值，通过遍历集合A，判断其每个元素是否在集合B中，若其也在集合B中，就将其插入交集。该操作的时间复杂度为 $O(m*n)$ ，m和n分别为两个集合的元素个数。

实现的代码如下：

```
/*
 * Set_Intersection -> A ∩ B
 *
 * return <- A ∩ B
 */
Set* Set_Intersection(Set *Set_A, Set *Set_B) {
    Set *new_set = Set_Init(Set_A->Elem->kind, Set_A->Elem->id);
    Set_traverse(Set_A->Elem->root, Set_B->Elem, new_set->Elem,
visit_intersection);
    return new_set;
}
```

②两集合的并

求并集这里我和求交集实现的一样，区别仅仅是将每个元素都插入新的集合中。时间复杂度为 $O(m*n)$ ，m和n分别为两个集合的元素个数。这里实际上有更简单的利用归并方法来求并集，时间复杂度仅为 $O(m+n)$ ，但是这里由于AVL树的遍历没有线性表那么方便，于是就采用了这种保守的方法，同时由于其代码和交的操作十分相似，编写起来也非常方便。

实现的代码如下：

```
/*
 * Set_Union -> A ∪ B
 *
 * return <- A ∪ B
 */
```

```
Set* Set_Union(Set *Set_A, Set* Set_B) {
    Set *new_set = Set_Init(Set_A->Elem->kind, Set_A->Elem->id);
    Set_traverse(Set_A->Elem->root, Set_B->Elem, new_set->Elem,
visit_union);
    Set_traverse(Set_B->Elem->root, Set_A->Elem, new_set->Elem,
visit_union);
    return new_set;
}
```

③两集合的差

差集操作和交集操作也十分类似，通过遍历集合A，判断其每个元素是否在集合B中，若其不在集合B中，就将其插入差集。该操作的时间复杂度仍为 $O(m*n)$ ，m和n分别为两个集合的元素个数。

实现的代码如下：

```
/*
 * Set_Difference -> A - B
 *
 * return <- A - B
 */
Set* Set_Difference(Set* Set_A, Set* Set_B) {
    Set *new_set = Set_Init(Set_A->Elem->kind, Set_A->Elem->id);
    Set_traverse(Set_A->Elem->root, Set_B->Elem, new_set->Elem,
visit_difference);
    return new_set;
}
```

④判断子集

这里我使用 $A - B = \emptyset$ 的方法来判断集合A是否是集合B的子集。

实现的代码如下：

```
/*
 * Set_Subset -> Judge whether set A is subset of B
 *
 * Here I use  $A - B = \emptyset$  to judge
 *
 * return <- The status
```

```

*/
int Set_Subset(Set *Set_A, Set* Set_B) {
    Set *new_set = Set_Difference(Set_A, Set_B);
    if (new_set->Elem->num == 0)
        return OK;
    else
        return ERROR;
}

```

⑤判断两个集合是否相等

这里我使用 $A - B = \emptyset$ 且 $B - A = \emptyset$ 的方法来判断两个集合A、B是否相等。

实现的代码如下：

```

/*
 * Set_Equal -> Judge whether set A is equal to set B
 *
 * Here I use  $A - B = \emptyset$  &&  $B - A = \emptyset$  to judge
 *
 * return <- The status
 */
int Set_Equal(Set *Set_A, Set* Set_B) {
    if (Set_Subset(Set_A, Set_B) && Set_Subset(Set_B, Set_A))
        return OK;
    else
        return ERROR;
}

```

⑥set 库所有 API 展示

根据任务书文档和具体的功能需求，设计十二个接口如下：

```

/* Set operations */
Set* Set_Init(int kind, int id); //初始化集合
void Set_Destroy(Set *Set); //销毁集合
void Set_Clear(Set *Set); //清空集合
void Set_Insert(Set *Set, int elem); //向集合中插入元素
void Set_Delete(Set *Set, int elem); //从集合中删除元素
Set* Set_Intersection(Set *Set_A, Set *Set_B); //求集合 A 和集合 B

```

的交集

```
Set* Set_Union(Set *Set_A, Set* Set_B); //求集合 A 和集合 B 的并集
Set* Set_Difference(Set* Set_A, Set* Set_B); //求集合 A 和集合 B 的
差集
int Set_Size(Set *Set); //求集合的元素个数
int Set_Member(Set *Set, int elem); //判断元素是否在集合内
int Set_Subset(Set *Set_A, Set* Set_B); //判断集合 A 是否是集合 B
的子集
int Set_Equal(Set *Set_A, Set* Set_B); //判断集合 A 和集合 B 是否相
等
```

3.2.3 哈希表实现算法

①哈希函数

哈希函数使用简单的取模法，通过让用户IDmod哈希表长度来决定存储位置，哈希表长度这个参数在实现哈希表的过程中设计为了可选的，给哈希表的使用提供了更多的空间。

哈希函数的实现代码如下：

```
/*
 * HashFunc -> The hash function
 *
 * return <- the height
 */
int HashFunc(int key, int mod) {
    return key % mod;
}
```

②冲突解决方法

冲突解决方法使用链式存储方法，将哈希函数得到相同结果的项生成为一个链表，挂在哈希数组的每一个结点之后，这样查找、插入和删除的效率都是非常高的。

插入实现的代码如下，这里设计了插入成功，失败和已有该结点三种状态作为返回值。

```
/*
 * insert_Hash -> Insert a node in hash func
```



```

*
* return <- the status
*      OK: insert ok
*      ERROR: insert error
*      EXISTS: the node has existed, used for no-repeated
*/
int insert_Hash(HashNode **Hash, int key, char *value, int hashlen)
{
    int status = EXISTS;
    int index = HashFunc(key, hashlen);
    HashNode *node = Hash[index]->next;
    while (node != NULL) {
        if (node->key == key) {
            /* Here I set no-repeated hash table */
            if (node->count == 0) {
                node->count++;
                status = OK;
            }
            break;
        }
        node = node->next;
    }
    if (node == NULL) {
        HashNode *new_node = (HashNode*)malloc(sizeof(HashNode));
        /* Malloc ERROR */
        if (new_node == NULL)
            return ERROR;
        new_node->key = key;
        strcpy(new_node->value, value);
        new_node->count = 1;
        new_node->next = Hash[index]->next;
        Hash[index]->next = new_node;
        status = OK;
    }
    return status;
}

```

删除实现的代码如下，同样设计了删除成功和失败两种状态：

```

/*
 * delete_Hash -> Delete a node in hash func
 *
 * return <- the status
 *         OK: Delete ok
 *         ERROR: Delete error
 */
int delete_Hash(HashNode **Hash, int key, int hashlen) {
    int status = ERROR;
    int index = HashFunc(key, hashlen);
    HashNode *node = Hash[index]->next;
    while (node != NULL) {
        if (node->key == key) {
            if (node->count > 0) {
                node->count--;
                strcpy(node->value, "Disable");
                status = OK;
            }
            break;
        }
        node = node->next;
    }
    return status;
}

```

查找实现的代码:

```

/*
 * search_Hash -> Search a node in an Hash table
 *
 * return <- The node
 */
HashNode* search_Hash(HashNode **Hash, int key, int hashlen) {
    int index = HashFunc(key, hashlen);
    HashNode *node = Hash[index]->next;
    while (node != NULL) {
        if (node->key == key && node->count > 0)
            return node;
        node = node->next;
    }
}

```

```

    }
    return NULL;
}

```

③hashtable 库所有 API 展示

根据任务书文档和具体的功能需求，设计八个接口如下：

```

/* APIs */
extern HashTable* Init_Hash(int hashlen); //初始化哈希表
extern int Insert_Hash(HashTable *MyHash, int key, char *value);
//在哈希表中插入结点
extern int Delete_Hash(HashTable *MyHash, int key); //从哈希表中
删除结点
extern char* Search_Hash(HashTable *MyHash, int key); //在哈希表
中查找结点
extern int Change_Hash(HashTable *MyHash, int key, char *value);
//改变哈希表中某结点的 value 值
extern void Traverse_Hash(HashTable *MyHash ,void (*visit)(struct
HashNode*)); //遍历哈希表
extern int Save_Hash(HashTable *MyHash, const char *path); //保
存哈希表
extern int Load_Hash(HashTable *MyHash, const char *path); //读
取哈希表

```

4 系统实现与测试

4.1 系统实现

4.1.1 开发环境

系统环境：GNU/Linux Arch-4.15.5

编译器：GCC Version 7.3.0

编译工具：CMake Version 3.9

编辑器：Visual-Studio-Code

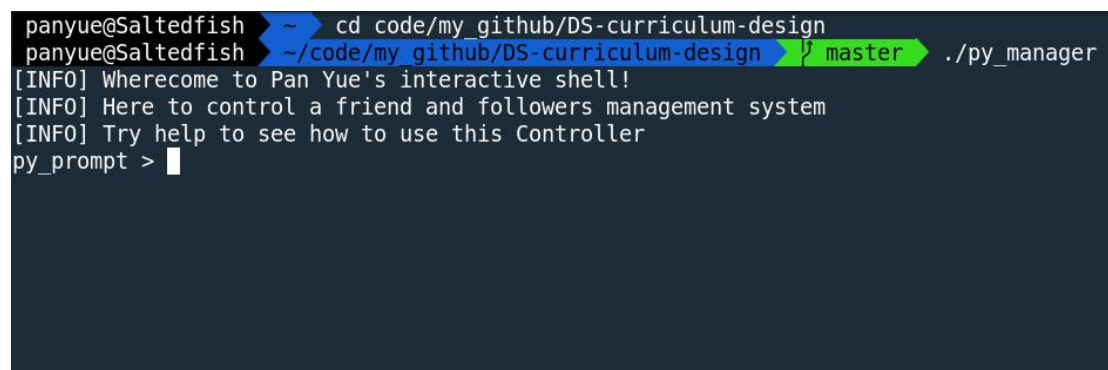
由于系统在 Linux 环境下开发并且有用到部分 UNIX 标准 API, 故项目在 Win 环境下不可编译。

4.1.2 用户界面设计

用户界面创新设计为 Linux Shell 的样式, 通过设计好 Shell 内建命令, 如用户指南 help 等, 和应用层操作命令, 如 add, delete 等来执行所有的操作。

Shell 的命令行解析实现的非常严格, 应按照标准的命令 + 参数的形式执行。这里设计了完备的错误处理, 针对各种情况会给出相应的错误提示。

用户界面外观如下图:



```
panyue@Saltedfish ~$ cd code/my_github/DS-curriculum-design
panyue@Saltedfish ~/code/my_github/DS-curriculum-design$ ./py_manager
[INFO] Wherecome to Pan Yue's interactive shell!
[INFO] Here to control a friend and followers management system
[INFO] Try help to see how to use this Controller
py_prompt > 
```

图10 用户界面外观

以下给出函数的说明:

①Start_Shell 函数

所在文件: py_prompt.c

函数原型: `int Start_Shell(int argc, char **argv);`

Start_Shell 函数为 shell 程序的主函数，功能为首先检测命令行参数，然后执行一个循环，包括获得输入的命令行和评估处理命令行两部分。

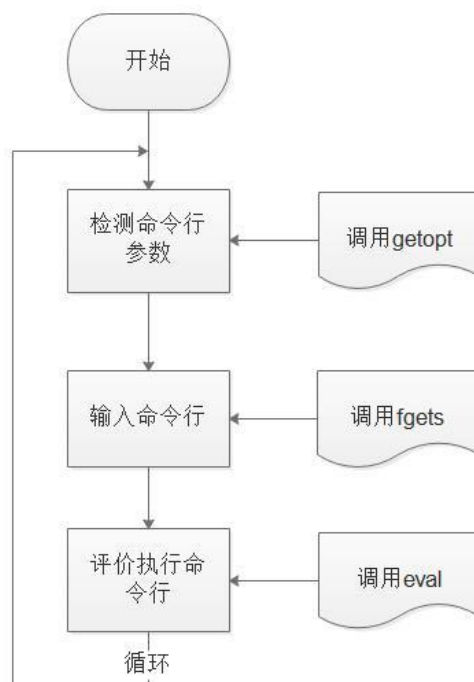


图11 Start_Shell函数流程

②eval 函数

所在文件: py_prompt.c

函数原型: `void eval(char *cmdline);`

eval 函数是对命令行的评估，功能为将命令行解析为命令和参数两部分，然后判断其是否为内建命令，并根据相应结果执行内建命令和应用层操作命令。

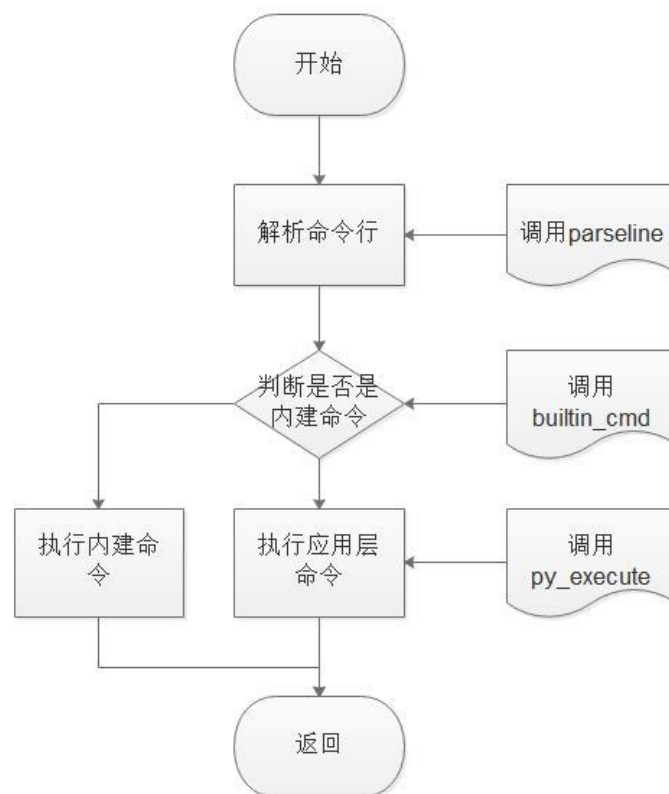


图12 eval函数流程

③parseline 函数

所在文件: py_prompt.c

函数原型: `int parseline(const char *cmdline, char **argv);`

`parseline` 函数是对命令行的解析, 将原本是一行的命令行解析为功能命令 + 参数两部分。

在算法上 `parseline` 函数实现了对空格和单引号的匹配, 这意味着参数可以直接跟在命令后也可以打上单引号输入。

④builtin_cmd 函数

所在文件: py_exec.c

函数原型: `int builtin_cmd(char **argv);`

`builtin_cmd` 函数传入解析好的命令行参数, 并判断是否是 shell 内建命令, 如果是就直接执行, 否则最后返回 0 表示不是内建命令, 则要调用 `py_execute` 函数执行应用层命令。

⑤py_execute 函数

所在文件: py_exec.c

函数原型: `int py_execute(char *func , int argc, char **argv);`

py_execute 函数传入要执行的功能和命令, 通过 if-else 语句选择要执行的函数, 为主要的功能层控制函数。

⑥getopt 函数介绍

函数原型: `int getopt(int argc, char * const argv[], const char * optstring);`

包含头文件: `#include<unistd.h>`

getopt 函数为 UNIX 环境下的 API, 作用为获取执行程序时的命令行参数, 将需要获得的参数写在 optstring 里, 循环执行即可。

以下是 getopt 函数在本系统中的调用。

```
/* Parse the command line */
while ((ch = getopt(argc, argv, "hp")) != EOF) {
    switch(ch) {
        case 'h':
            print_usage();
            exit(0);
        case 'p':
            emit_prompt = 0;
            break;
        default:
            print_usage();
            exit(1);
    }
}
```

4.1.3 应用层设计

在用户界面设计为 Linux shell 的情况下, 应用层完全设计成了 shell 内的各种命令, 通过各种命令, 来执行对应的应用层操作。

①内建命令的设计

内建命令设计了三个, 分别为:

exit: 退出程序

show: 显示当前数据库内的所有用户

help: 显示用户指南

②应用层命令的设计

应用层命令设计了，分别为：

add: 增加用户/爱好/好友/关注/粉丝

delete: 删除用户/好友/关注/粉丝

find: 查找用户/好友/关注/粉丝

display: 显示某用户全部的好友/关注/粉丝

common: 显示两个用户共同的好友/关注/粉丝

recommend: 向用户推荐好友/关注/粉丝（实质为二度查询）

rename: 用户改名

所有的操作采用命令 + 参数的形式，其中所有的用法在用户手册中有写明。

操作统一采用如下结构完成：

1. 打开存储文件，读取数据结构
2. 执行操作
3. 保存数据结构至文件，关闭文件。

这样的设计极大的简化的应用层的构建，不需要考虑内存中的数据。

③应用层函数原型

这里将每个打开文件修改保存等操作封装一层，设计了如下的一些函数，应用层只需要在解析清楚命令行，明确要做什么之后调用相应的函数即可。

```
/* Function prototypes */
```

```
/* Add functions */
```

```
void Add_user(int user_id, char *user_name);
```

```
void Add_hobby(int user_id, char *hobby);
```

```
void Add_friend(int user_id, int friend_id, int tag);
```

```
void Add_follower(int user_id, int follower_id, int tag);
```

```
void Add_following(int user_id, int following_id, int tag);
```



```

/* Delete functions */
void Delete_user(int user_id);
void Delete_friend(int user_id, int friend_id, int tag);
void Delete_follower(int user_id, int follower_id, int tag);
void Delete_following(int user_id, int following_id, int tag);

/* Find functions */
void Find_user(int user_id);
void Find_friend(int user_id, int friend_id);
void Find_follower(int user_id, int follower_id);
void Find_following(int user_id, int following_id);

/* Display functions */
void Display_friend(int user_id);
void Display_follower(int user_id);
void Display_following(int user_id);

/* Common display fuctions */
void Common_friends(int user_a, int user_b);
void Common_followers(int user_a, int user_b);
void Common_followings(int user_a, int user_b);

/* Recommend functions */
void Recommend_friends(int user_id);
void Recommend_followers(int user_id);
void Recommend_followings(int user_id);

/* Others */
void Change_name(int user_id, char *new_name);

```

在设计这一部分函数时，需要注意函数之间的关系，比如对 A 添加好友 B 的同时必须对 B 添加好友 A，对 A 添加关注 C 的同时必须对 C 添加粉丝 A。

4.1.4 文件存储设计

首先设计了两个哈希表来存放用户 ID-昵称，用户 ID-爱好的键值对，以每行一个键值对的形式存放在 users.txt 和 hobby.txt 两个文件里。

好友/关注/粉丝集合由其形式上的一致性（都由 AVL 树构建而成），统一被设计成了格式一致的文件，通过一个函数 `generate_filename` 函数来生成文件名保存。根据宏定义中的 0、1、2 分别代表好友、关注、粉丝生成“用户 ID-集

合属性.txt”的文件名。这样每一个用户都会有三个文件分别保存和他相关的三个集合。文件内第一行为 AVL 树相关属性，其后每行存储一个用户 ID。

4.1.5 编译处理

软件架构有三层，将每层编译为一个链接库，为上层所依赖。

用到的 CMakeLists 如下：

```
./CMakeLists.txt
```

```
cmake_minimum_required(VERSION 3.9)
project(DS_Curriculum_Design)
add_subdirectory(src)
aux_source_directory(. dir_main)
add_executable(py_manager ${dir_main})
target_link_libraries(py_manager lib)
```

```
./src/CMakeLists.txt
```

```
add_subdirectory(prompt)
aux_source_directory(. dir_srcs)
add_library(lib ${dir_srcs})
target_link_libraries(lib prompt)
```

```
./src/prompt/CMakeLists.txt
```

```
aux_source_directory(. dir_prompt)
add_library(prompt ${dir_prompt})
```

4.2 系统测试

4.2.1 Google Test 软件单元测试方法介绍

在计算机编程中，单元测试又称为模块测试，是针对程序模块（软件设计的最小单位）来进行正确性检验的测试工作。程序单元是应用的最小可测试部件。在过程化编程中，一个单元就是单个程序、函数、过程等；对于面向对象编程，

最小单元就是方法，包括基类（超类）、抽象类、或者派生类（子类）中的方法。

在本系统中，最需要测试的，也就是复杂一些的单元就是AVL树了，保证AVL树的正确性是保证set以及后续应用层正确性的根本。

googletest是一个开源的C++单元测试的框架，它是跨平台的，可应用在windows、linux、Mac等OS平台上。可以通过超大量的数据进行反复的执行来检验我们的AVL树是否正确。

4.2.2 软件测试大纲

根据如上所说，需要被测试的核心模块就是AVL树部分，其余的哈希表实现较为简单，可以通过简单的插入删除检测，应用层部分将在4.3节展示的时候给出相应的图示。

接下来我们拿C++ STL中的标准set来和我们创建的AVL树集合做对比，在执行相同的操作后结果是否一致。

以下是七个测试部分：

①运行时间测试

目标：作为数据库，任何插入删除查找的操作应该是在一个较快的时间内完成的，否则低下的效率会极大地影响软件的使用。

测试方案：如果接下来六个测试单个的运行时间超过了30秒，则被判断为低效，提示错误。

测试函数：时间测试函数在setTest类内部

```
class setTest : public testing::Test {
protected:
    virtual void SetUp() {
        start_time = time(NULL);
    }

    virtual void TearDown() {
        time_t end_time = time(NULL);
        std::cout << "This test takes " << end_time-start_time <<
"times." << std::endl;
        ASSERT_TRUE(end_time - start_time <= 30) << "The test took too
long.";
    }
}
```

```
time_t start_time;
};
```

②较少次随机数据测试

目标：我们的AVL树应保证完成基本的插入删除功能

测试方案：向标准set和我们的AVL树中插入随机的100个数据，比较两个集合内部的元素个数，并且以逐个删除的形式判断两个集合是否相等。

测试函数：

```
TEST_F(setTest, random_small) {

    std::default_random_engine e(810);
    std::uniform_int_distribution<int> dis(1, 100);

    std::set<int> std_set;
    AVL_tree *custom_set;

    custom_set = Init_AVL(0, 1);

    for (unsigned int i = 1; i < 100; i++) {
        int num = dis(e);
        std_set.insert(num);
        Insert_AVL(custom_set, num);
    }

    ASSERT_EQ(std_set.size(), custom_set->num);

    for (auto s:std_set) {
        ASSERT_EQ(Search_AVL(custom_set, s), s);
        Delete_AVL(custom_set, s);
    }
    ASSERT_EQ(custom_set->num, 0);
}
```

③大量随机数据测试

目标：我们的AVL树应该能适应极大量的数据插入，并且保证集合元素的唯一性。

测试方案：在上一次测试的基础上，向标准set和我们的AVL树中插入随机的500000个数据，进行同样的比较。

测试函数：

```
TEST_F(setTest, random_large) {
    std::default_random_engine e(810);
    std::uniform_int_distribution<int> dis(1,100);

    std::set<int> std_set;
    AVL_tree *custom_set;

    custom_set = Init_AVL(0, 1);

    for (unsigned int i = 1; i < 500000; i++) {
        int num = dis(e);
        std_set.insert(num);
        Insert_AVL(custom_set, num);
    }

    ASSERT_EQ(std_set.size(), custom_set->num);

    for (auto s:std_set) {
        ASSERT_EQ(Search_AVL(custom_set, s), s);
        Delete_AVL(custom_set, s);
    }
    ASSERT_EQ(custom_set->num, 0);
}
```

④单一数据测试

目标：我们的AVL树应该可以在一直反复插入同一数据时保证稳定。

测试方案：向标准set和我们的AVL树中反复插入5000次1，比较两个集合的大小和数据情况。

测试函数：

```
TEST_F(setTest, only_data) {
    std::set<int> std_set;
    AVL_tree *custom_set;
```

```

custom_set = Init_AVL(0, 1);

for (unsigned int i = 1; i < 5000; i++) {
    std_set.insert(1);
    Insert_AVL(custom_set, 1);
}
ASSERT_EQ(std_set.size(), custom_set->num);
ASSERT_EQ(Search_AVL(custom_set, 1), 1);
}

```

⑤顺序数据测试

目标：我们的AVL树应该可以适应连续插入顺序值的情况（此时会大量发生各种旋转操作），并且这个插入数值应是大量的，在保证的时间内完成的。此后再进行连续删除的操作，同样应该在保证正确性的同时保证在时间内完成。

测试方案：向标准set和我们的AVL树中按顺序插入1-5000，比较两个集合内部的元素个数，并且以逐个删除的形式判断两个集合是否相等。

测试函数：

```

TEST_F(setTest, ordered_data) {
    std::set<int> std_set;
    AVL_tree *custom_set;

    custom_set = Init_AVL(0, 1);

    for (unsigned int i = 1; i < 5000; i++) {
        std_set.insert(i);
        Insert_AVL(custom_set, i);
    }
    ASSERT_EQ(std_set.size(), custom_set->num);
    for (auto s:std_set) {
        Delete_AVL(custom_set, s);
    }
    ASSERT_EQ(custom_set->num, 0);
}

```

⑥重复数据测试

目标：我们的AVL树应该能适应大量的顺序的带重复的数据。这一测试是结

合了④和⑤的加强测试。

测试方案：向标准set和我们的AVL树中插顺序插入1-5000，但除去1和5000之外的每个数都要重复两次，比较两个集合内部的元素个数，并且以逐个删除的形式判断两个集合是否相等。

测试函数：

```
TEST_F(setTest, repeated_data) {
    std::set<int> std_set;
    AVL_tree *custom_set;

    custom_set = Init_AVL(0, 1);

    for (unsigned int i = 0; i < 5000; i++) {
        std_set.insert(i);
        std_set.insert(i+1);
        Insert_AVL(custom_set, i);
        Insert_AVL(custom_set, i+1);
    }
    ASSERT_EQ(std_set.size(), custom_set->num);
    for (auto s:std_set) {
        Delete_AVL(custom_set, s);
    }
    ASSERT_EQ(custom_set->num, 0);
}
```

⑦复杂多次随机数据测试

目标：我们的AVL树应该能适应高强度的反复插入删除测试。

测试方案：这次应该是完全的测试了，反复执行10次以下的测试

（1）向标准set和我们的AVL树中插入10000次随机数

（2）其中五次采用一个个删除元素的形式，五次采用清空函数的形式删除所有的数据

（3）在以上的插入和删除操作之后及时比较两个集合的差异

（4）在删除所有的数据后，两个集合的元素个数应该都是0

测试函数：

```
TEST_F(setTest, random) {
```

```

std::random_device rand_dev;

std::default_random_engine e(810);
std::uniform_int_distribution<int> dis(1,100);

std::set<int> std_set;
AVL_tree *custom_set;

custom_set = Init_AVL(0, 1);

for (unsigned int times = 0; times < 10; times++) {
    for (unsigned int i = 0; i < 10000; i++) {
        double num = dis(e);
        std_set.insert(num);
        Insert_AVL(custom_set, num);
    }

    ASSERT_EQ(std_set.size(), custom_set->num);

    if (times % 2) {
        for (auto s:std_set) {
            Delete_AVL(custom_set, s);
            ASSERT_EQ(Search_AVL(custom_set, s), 0);
        }
        std_set.clear();
    }
    else {
        Clear_AVL(custom_set);
        std_set.clear();
    }

    ASSERT_EQ(std_set.size(), 0);
    ASSERT_EQ(custom_set->num, 0);
}
}

```

4.2.3 测试结果与分析

gtest测试运行的结果如下图：

```

x panyue@Saltedfish ~/code/my_github/DS-curriculum-design/test/build > master ./set_test
Running main() from gtest_main.cc
[=====] Running 6 tests from 1 test case.
[-----] Global test environment set-up.
[-----] 6 tests from setTest
[ RUN      ] setTest.random_small
This test takes 0times.
[ OK      ] setTest.random_small (1 ms)
[ RUN      ] setTest.random_large
This test takes 10times.
[ OK      ] setTest.random_large (9757 ms)
[ RUN      ] setTest.only_data
This test takes 0times.
[ OK      ] setTest.only_data (0 ms)
[ RUN      ] setTest.ordered_data
This test takes 16times.
[ OK      ] setTest.ordered_data (16689 ms)
[ RUN      ] setTest.repeated_data
This test takes 29times.
[ OK      ] setTest.repeated_data (29249 ms)
[ RUN      ] setTest.random
This test takes 2times.
[ OK      ] setTest.random (1842 ms)
[-----] 6 tests from setTest (57538 ms total)

[-----] Global test environment tear-down
[=====] 6 tests from 1 test case ran. (57538 ms total)
[ PASSED  ] 6 tests.
    
```

图13 gtest测试结果

我们可以看到，系统顺利的通过了所有的测试，在大量的数据测试下表明我们的AVL树是完全正确的，达到了作为数据库结构的功能需求。

同时我们可以看到，第四个和第五个测试花费的时间比较长。而C++ STL中标准的set，其执行这些测试可以在数秒内完成。

这两个测试是基于顺序插入数据进行的，这也反映了AVL树在插入删除的性能不足，虽然在查找上有着优势，但是在顺序插入这种极端情况下，旋转的次数也大量增长，这也是我们在数据库的设计时需要着重重视的一项。

在后续的章节里，会提到关于AVL树性能方面的问题和改进。

4.2.4 应用层的测试

应用层中出现算法错误的问题不大，基本上多是由于手误造成，这一部分无需单独做测试，在后续大量的添加删除数据中便可检测出问题。

4.3 系统运行展示

这一部分展示一些软件运行时的示例。将着重展示应用层的各种命令操作和操作之后的数据显示。系统数据库内已录入一百个用户和复杂的好友关系网络。

首先是输入help命令后的用户手册：

```
[INFO] Try help to see how to use this Controller
py_prompt > help
User Manage System version 1.1
Author: Pan Yue
Basic function:
  help: Display User's Manual
  show: Show all users in database
  rename: Change a user's name
  rename [user ID] [new name]

  add: Add data
  add [option] [arg1] [arg2]
  option=user, add a user, [arg1]=user ID, [arg2]=user name
  option=hobby, add a user's hobby, [arg1]=user ID, [arg2]=hobby
  option=friend, add a friend, [arg1]=user ID, [arg2]=friend ID
  option=follower, add a follower, [arg1]=user ID, [arg2]=follower ID
  option=following, add a following, [arg1]=user ID, [arg2]=following ID

  delete: Delete data
  delete [option] [arg1] [arg2]
  option=user, delete a user, [arg1]=user ID
  option=friend, delete a friend, [arg1]=user ID, [arg2]=friend ID
  option=follower, delete a follower, [arg1]=user ID, [arg2]=follower ID
  option=following, delete a following, [arg1]=user ID, [arg2]=following ID

  find: Find data
  find [option] [arg1] [arg2]
  option=user, find a user, [arg1]=user ID
  option=friend, find a friend, [arg1]=user ID, [arg2]=friend ID
  option=follower, find a follower, [arg1]=user ID, [arg2]=follower ID
  option=following, find a following, [arg1]=user ID, [arg2]=following ID

  display: Display data
  display [option] [user ID]
  option=friend/follower/following, then display this user's all [option]

  recommend: Recommend for a user
  recommend [option] [user ID]
  option=friend/follower/following, then recommend [option] for user

  common: Display common data of 2 users
  common [option] [user A] [user B]
  option=friend/follower/following, then display 2 user's common [option]
py_prompt > 
```

图14 用户手册

输入show之后列出当前数据库内用户列表：

```
py_prompt > show
  User ID      Name
  128          twentyeight
  129          twentynine
  130           thirty
  131          thirtyone
  132          thirtytwo
  133          thirtythree
  134          thirtyfour
  135          thirtyfive
  136          thirtysix
  137          thirtyseven
  138          thirtyeight
  139          thirtynine
  140           forty
  141          fortyone
  142          fortytwo
  143          fortythree
  144          fortyfour
  145          fortyfive
  146          fourtysix
  147          fortyseven
  148          fortyeight
  149          fourtynine
  150           fifty
  151          fiftyone
  152          fiftytwo
```

图15 列出用户

插入数据演示，包括创建用户，加入好友、关注、粉丝并展示用户个人信息。
这里新创建一个用户进行展示。

```
py_prompt > add user 201 Tom
[INFO] Operation finished.

py_prompt > add friend 201 101
[INFO] Operation finished.

py_prompt > add follower 201 102
[INFO] Operation finished.

py_prompt > add following 201 103
[INFO] Operation finished.

py_prompt > add hobby 201 Coding
[INFO] Operation finished.

py_prompt > find user 201
User ID: 201
User Name: Tom
Friend: 1
Follower: 1
Following: 1
Hobby: Coding
[INFO] Operation finished.

py_prompt > █
```

图16 插入数据演示

删除数据演示，包括删除好友、关注、粉丝及展示用户个人信息

```
py_prompt > delete friend 201 101
[INFO] Operation finished.

py_prompt > delete follower 201 102
[INFO] Operation finished.

py_prompt > delete following 201 103
[INFO] Operation finished.

py_prompt > find user 201
User ID: 201
User Name: Tom
Friend: 0
Follower: 0
Following: 0
Hobby: Coding
[INFO] Operation finished.

py_prompt > █
```

图17 删除数据演示

查找演示，包括查找用户、好友、关注、粉丝，这里对数据库中已有的用户进行查找。

```
py_prompt > find user 101
User ID: 101
User Name: pan
Friend: 15
Follower: 6
Following: 5
Hobby: basketball
[INFO] Operation finished.

py_prompt > find friend 101 105
[INFO] Found it! Your friend five.
[INFO] Operation finished.

py_prompt > find friend 101 111
[INFO] Not Found!
[INFO] Operation finished.

py_prompt > find follower 101 116
[INFO] Found it! Your follower sixteen.
[INFO] Operation finished.

py_prompt > find following 101 115
[INFO] Not Found!
[INFO] Operation finished.

py_prompt > find following 101 114
[INFO] Found it! You has followed fourteen.
[INFO] Operation finished.

py_prompt > █
```

图18 查找数据演示

显示数据演示，这里查看101的所有好友，粉丝，关注。

```

py_prompt > display friend 101
User ID: 104, Name: four
User ID: 105, Name: five
User ID: 116, Name: sixteen
User ID: 123, Name: twentythree
User ID: 125, Name: twentyfive
User ID: 142, Name: fortytwo
User ID: 154, Name: fiftyfour
User ID: 158, Name: fiftyeight
User ID: 159, Name: fiftynine
User ID: 163, Name: sixtythree
User ID: 164, Name: sixtyfour
User ID: 173, Name: seventythree
User ID: 174, Name: seventyfour
User ID: 179, Name: seventynine
User ID: 198, Name: ninetyeight
[INFO] Operation finished.

py_prompt > display follower 101
User ID: 104, Name: four
User ID: 116, Name: sixteen
User ID: 117, Name: seventeen
User ID: 119, Name: nineteen
User ID: 135, Name: thirtyfive
User ID: 146, Name: fourtysix
[INFO] Operation finished.

py_prompt > display following 101
User ID: 113, Name: thirteen
User ID: 114, Name: fourteen
User ID: 119, Name: nineteen
User ID: 156, Name: fiftysix
User ID: 169, Name: sixtynine
[INFO] Operation finished.

py_prompt > █

```

图19 显示数据演示

共同集合演示，这里展示数据库内已有的几个人的共同好友、关注、粉丝集合。

```

py_prompt > common friend 101 102
User ID: 123, Name: twentythree
User ID: 154, Name: fiftyfour
User ID: 158, Name: fiftyeight
User ID: 198, Name: ninetyeight
[INFO] Operation finished.

py_prompt > common friend 101 103
User ID: 105, Name: five
User ID: 116, Name: sixteen
User ID: 154, Name: fiftyfour
User ID: 159, Name: fifty-nine
User ID: 164, Name: sixtyfour
User ID: 179, Name: seventynine
[INFO] Operation finished.

py_prompt > common follower 101 102
User ID: 116, Name: sixteen
User ID: 135, Name: thirtyfive
[INFO] Operation finished.

py_prompt > common following 101 102
User ID: 156, Name: fiftysix
[INFO] Operation finished.

py_prompt > 

```

图20 共同集合演示

更改用户名演示，这里对用户101更改用户名。

```

py_prompt > find user 101
User ID: 101
User Name: pan
Friend: 15
Follower: 6
Following: 5
Hobby: basketball
[INFO] Operation finished.

py_prompt > rename 101 wang
[INFO] Operation finished.

py_prompt > find user 101
User ID: 101
User Name: wang
Friend: 15
Follower: 6
Following: 5
Hobby: basketball
[INFO] Operation finished.

py_prompt > 

```

图20 更改用户名演示

推荐好友演示，这里使用二度好友来表示推荐的好友集合。

```
py_prompt > recommand friend 101
[INFO] pysh: command not found.
py_prompt > recommend friend 101
User ID: 102, Name: two
User ID: 103, Name: three
User ID: 106, Name: six
User ID: 107, Name: seven
User ID: 108, Name: eight
User ID: 110, Name: ten
User ID: 111, Name: eleven
User ID: 118, Name: eighteen
User ID: 121, Name: twentyone
User ID: 130, Name: thirty
User ID: 131, Name: thirtyone
User ID: 132, Name: thirtytwo
User ID: 136, Name: thirtysix
User ID: 137, Name: thirtyseven
User ID: 138, Name: thirtyeight
User ID: 140, Name: fourty
User ID: 143, Name: fortythree
User ID: 144, Name: fortyfour
User ID: 149, Name: fourtynine
User ID: 152, Name: fiftytwo
User ID: 153, Name: fiftythree
User ID: 161, Name: sixtyone
User ID: 170, Name: seventy
User ID: 178, Name: seventyeight
User ID: 180, Name: eighty
User ID: 181, Name: eightyone
User ID: 186, Name: eightysix
User ID: 188, Name: eightyeight
[INFO] Operation finished.
```

图21 推荐好友演示

结束程序命令演示，输入exit结束程序。


```
panyue@Saltedfish ~/code/my github/DS-curriculum-design  master  ./py_manager
[INFO] Wherecome to Pan Yue's interactive shell!
[INFO] Here to control a friend and followers management system
[INFO] Try help to see how to use this Controller
py_prompt > exit
[INFO] User-exit. Terminated!
```

图22 结束程序演示

系统的各功能均只截取了一个用户操作界面演示,更多的细节可以打开系统尽情体验。

5 开源说明

5.1 声明

本项目已于 2018.3.3 在 github 上开源公布, 对其他同学仅作为参考作用, 以学习 AVL 树写法和软件应用层构建方法, 不允许原样抄袭作为课程设计提交。潘越 U201614897 为原开发者。

项目地址: <https://github.com/zxc479773533/DS-curriculum-design>

5.2 开源协议

本项目遵守 GNU GENERAL PUBLIC LICENSE V3.0 协议, 若有大量引用代码或者套用代码进行修改, 请遵守协议规范。

协议地址:

<https://github.com/zxc479773533/DS-curriculum-design/blob/master/LICENSE>

6 总结与展望

6.1 课程设计总结

6.1.1 工作总结

整个课程设计从 2.18 开始，共花费六天时间完成第一个版本，大概每天晚上写个三到四小时。此后基本是 debug，做单元测试，不断完善文档的过程。

项目工作过程概览图如下：

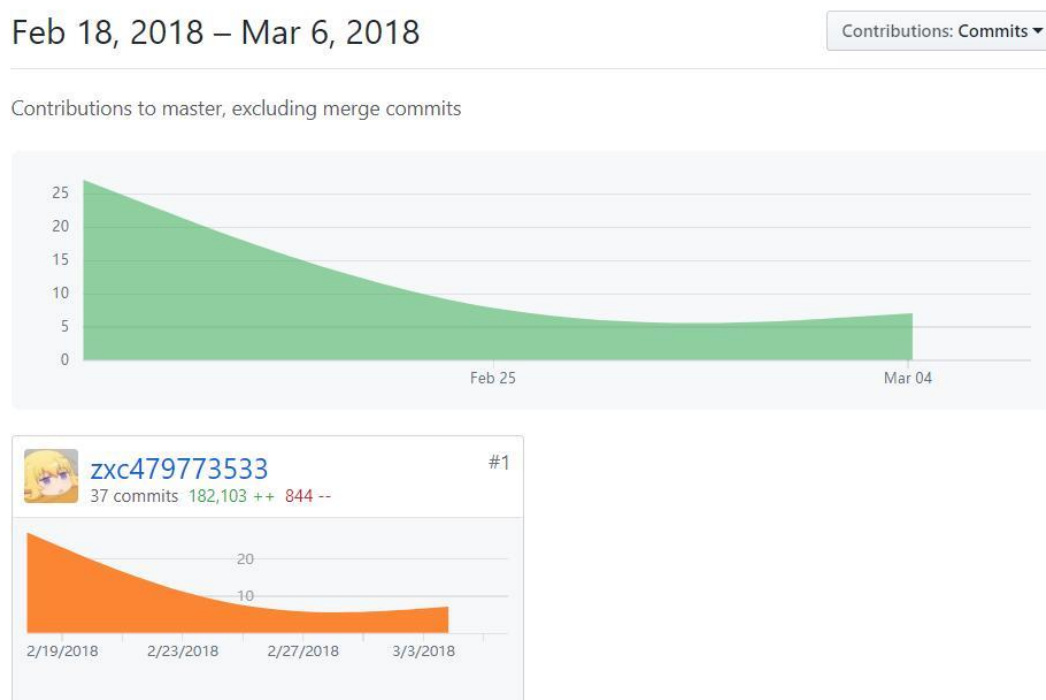


图 23 项目工作过程概览

6.1.2 系统的优点

系统的完备性达到了课程设计的全部要求，正确性通过了测试要求的水平，并且对于插入删除查找等等的处理也实现得当，并且做了完备的错误检查，对于任何输出可以保证系统不会崩溃。

界面也有一定的新颖度，在 Linux 下模仿了 shell 设计命令，操作也比较简洁。

6.1.3 系统的不足

系统比较明显的问题是操作速度的问题，在数据量很小的时候，并不会发现这个问题，但是在输入上千用户，上百好友之后，问题会很明显，插入和删除需要一定时间的相应。这个实际上是因为插入删除的时间变长了。

在前面章节的测试中，我们就可以发现，AVL 树在好几个测试中都出现了进行十秒甚至二十多秒，但是当使用 C++ STL 中的 set 来测试时，全部的测试可以在数秒内完成，为什么我们的 AVL 树这么慢呢？

查阅资料发现，C++ STL 中的 set 是由红黑树实现的，红黑树是一种对 AVL 树的改进，红黑树并不是完全平衡的，但是却通过各种旋转和变色操作保证了任意的插入和删除操作都可以在三次旋转之内完成，这样虽然牺牲了少量的查找效率，但是大大提高了插入和删除的效率，这也是为什么大量的数据库现在使用红黑树尤其是左倾红黑树为数据结构内核的原因。

6.2 工作展望

在今后的学习中，我将进一步学习算法方面的东西。虽然有在课外看过一些高级数据结构的书，但是还有很多问题类型没有接触过，比如动态规划等等问题，都还需要进一步的学习和理解。总之，前方的道路还长，还需要继续努力。

7 体会与感想

这次课程设计总得来说，对我还算是简单的，毕竟在课外做过很多功课，看过一些书。选择在 Linux 环境下编程，也是出于锻炼自己的目的去的。

在写代码的过程中，采用了一层层封装的模式，每写完一层，测试好接口再去完成下一层，这样的编程方式极大的提高了完成课设的效率，也让我认识到这样构建软件的好处。

这次数据结构课设是我自己第一次亲自写 Google test 单元测试来用大量数据测试自己的接口。从这次经历中，也确实是认识到了单元测试的好处与重要性。这种测试可以方便的检查出仅仅通过几次插入删除找不到的 bug，给程序的正确性提供了保证。

将用户界面设计为 shell 也是因为前不久做完了《深入理解计算机系统》的一个实验，shell-lab，体验了一个完整的 shell 的构建方式，然后这次课设就套用了这个框架，自己改动后实现了一个非常简单的 shell。当然这里和实际的 shell 还是差很多的，包括信号量的处理和前后台执行的任务等等都没有实现。

一个遗憾就是在 Linux 下构建图形界面不是特别方便，本来想做一个图形界面的，但是由于工期的原因后来就放弃了。在以后的学习中，有机会还是要掌握在 Linux 下构建图形界面的方法。

附录：程序源代码

I. 算法库文件

I.1 avltree.h – AVL 树头文件

```
/*
 * AVL Tree library
 * Developed by Pan Yue
 * zxc479773533@gmail.com
 */
#ifndef PY_AVL_LIB
#define PY_AVL_LIB

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

/* Status */
#define ERROR 0
#define OK 1

/* The kind of AVL tree */
#define FRIENDS 0
#define FOLLOWERS 1
#define FOLLOWING 2

/* The kind of traverse */
#define PREORDER 0
#define INORDER 1
#define POSTORDER 2

/* Assisr micro */
#define Max(a, b) ((a > b) ? a : b)

/* The AVL node struct */
typedef struct AVL_node {
    int key;          /* node key */
    int height;       /* node height */
    struct AVL_node *left_child;
    struct AVL_node *right_child;
} AVL_node;
```

```

/* The tree struct */
typedef struct AVL_tree {
    int kind;          /* tree kind */
    int id;            /* tree id */
    int num;           /* node num */
    AVL_node *root;    /* root node */
} AVL_tree;

/* APIs */
extern AVL_tree* Init_AVL(int kind, int id);
extern void Destroy_AVL(AVL_tree *AVL);
extern void Clear_AVL(AVL_tree *AVL);
extern int Search_AVL(AVL_tree *AVL, int key);
extern void Insert_AVL(AVL_tree *AVL, int key);
extern void Delete_AVL(AVL_tree *AVL, int key);
extern void Traverse_AVL(AVL_tree *AVL, int option, void
(*visit)(int));
extern int Save_AVL(AVL_tree *AVL, const char *path);
extern int Load_AVL(AVL_tree *AVL, const char *path);

/* Functions */
int avl_height(AVL_node *tree);
AVL_node* search_avl(AVL_node *tree, int key);
AVL_node* find_min(AVL_node *tree);
AVL_node* find_max(AVL_node *tree);
AVL_node* leftleft_rotate(AVL_node *tree);
AVL_node* rightright_rotate(AVL_node *tree);
AVL_node* leftright_rotate(AVL_node *tree);
AVL_node* rightright_rotate(AVL_node *tree);
AVL_node* insert_avl(AVL_node *tree, int key, int *safe_tag);
AVL_node* delete_avl(AVL_node *tree, AVL_node *node);
AVL_node* clear_avl(AVL_node *tree);
void traverse_avl(AVL_node *tree, int option, void (*visit)(int));
void save_avl(AVL_node *tree, FILE *fp);
AVL_node* load_avl(AVL_node *tree, FILE *fp);

#endif // !PY_AVL_LIB

```

I.2 avltree.c – AVL 树具体实现

```

/*
 * AVL Tree implementation
 * Developed by Pan Yue
 * zxc479773533@gmail.com
 */
#include "avltree.h"

/*
 * avl_height -> get height of an avl tree
 *
 * return <- the height
 */
int avl_height(AVL_node *tree) {
    if (tree == NULL)
        return 0;
    else
        return 1 + Max(avl_height(tree->left_child),
            avl_height(tree->right_child));
}

/*
 * search_avl -> find a node in an avl tree
 *
 * return <- the ptr to the node
 */
AVL_node* search_avl(AVL_node *tree, int key) {
    /* Find it */
    if (tree == NULL)
        return NULL;
    /* Find in child tree */
    if (tree->key == key)
        return tree;
    else if (tree->key > key)
        return search_avl(tree->left_child, key);
    else
        return search_avl(tree->right_child, key);
}

```

```

/*
 * find_min -> find the min node in an avl tree
 *
 * return <- the ptr to the node
 */
AVL_node* find_min(AVL_node *tree) {
    /* Find it */
    if (tree->left_child == NULL)
        return tree;
    /* Find in left child tree */
    return find_min(tree->left_child);
}

/*
 * find_max -> find the max node in an avl tree
 *
 * return <- the ptr to the node
 */
AVL_node* find_max(AVL_node *tree) {
    /* Find it */
    if (tree->right_child == NULL)
        return tree;
    /* Find in right child tree */
    return find_max(tree->right_child);
}

/*
 * leftleft_rotate -> do a left-left rotate
 *
 *      k1                k2
 *      / \              / \
 *      k2 ##  -----> ## k1
 *      / \              / \
 *      ## @@            @@ ##
 *
 * return <- k2, the root after rotated
 */
AVL_node* leftleft_rotate(AVL_node *k1) {
    AVL_node *k2 = k1->left_child;

```



```

    k1->left_child = k2->right_child;
    k2->right_child = k1;
    k1->height = 1 + Max(avl_height(k1->left_child),
    avl_height(k1->right_child));
    k2->height = 1 + Max(avl_height(k2->left_child),
    avl_height(k2->right_child));
    return k2;
}

```

```

/*
 * rightright_rotate -> do a right-right rotate
 *

```

```

 *      k1          k2
 *      / \        / \
 *      ## k2 -----> k1 ##
 *      / \        / \
 *      @@ ##      ## @@
 *

```

```

 * return <- k2, the root after rotated
 */

```

```

AVL_node* rightright_rotate(AVL_node *k1) {
    AVL_node *k2 = k1->right_child;
    k1->right_child = k2->left_child;
    k2->left_child = k1;
    k1->height = 1 + Max(avl_height(k1->left_child),
    avl_height(k1->right_child));
    k2->height = 1 + Max(avl_height(k2->left_child),
    avl_height(k2->right_child));
    return k2;
}

```

```

/*
 * leftright_rotate -> do a left-right rotate
 *

```

```

 *      k1          k1          k3
 *      / \        / \          / \
 *      k2 ##      k3 ##          k2  k1
 *      / \        / \          / \  / \
 *      ## k3      k2 %%          / \  / \

```

```

*      /  \              /  \              ## @@  %% ##
*      @@  %%              ##  @@
*
* return <- k3, the root after rotated
*/
AVL_node* leftright_rotate(AVL_node *k1) {
    k1->left_child = rightright_rotate(k1->left_child);
    return leftleft_rotate(k1);
}

/*
* rightright_rotate -> do a right-left rotate
*
*      k1              k1              k3
*      /  \          /  \          /  \
*      ## k2          ## k3          /  \
*      /  \ -----> /  \ -----> k1  k2
*      k3  ##          @@ k2          /  \ /  \
*      /  \          /  \          ## @@  %% ##
*      @@  %%          %%  ##
*
* return <- k3, the root after rotated
*/
AVL_node* rightright_rotate(AVL_node *k1) {
    k1->right_child = leftleft_rotate(k1->right_child);
    return rightright_rotate(k1);
}

/*
* insert_avl -> insert a node in avl tree
*
* using recursive function, settle height and balance
* every return
*
* return <- the root node after balanced
*/
AVL_node* insert_avl(AVL_node *tree, int key, int *safe_tag) {
    /* Create new node */
    if (tree == NULL) {

```

```

    tree = (AVL_node*)malloc(sizeof(AVL_node));
    tree->key = key;
    tree->height = 0;
    tree->left_child = NULL;
    tree->right_child = NULL;
}
/* Insert in left child tree */
else if (key < tree->key) {
    tree->left_child = insert_avl(tree->left_child, key,
safe_tag);
    /* Balance it */
    if (avl_height(tree->left_child) -
avl_height(tree->right_child) == 2) {
        /* Insert left-left */
        if (key < tree->left_child->key)
            tree = leftleft_rotate(tree);
        /* Insert left right */
        else
            tree = leftright_rotate(tree);
    }
}
/* Insert in right child tree */
else if (key > tree->key) {
    tree->right_child = insert_avl(tree->right_child, key,
safe_tag);
    /* Balance it */
    if (avl_height(tree->right_child) -
avl_height(tree->left_child) == 2) {
        /* Insert right-left */
        if (key < tree->right_child->key)
            tree = rightleft_rotate(tree);
        /* Insert right-right */
        else
            tree = rightright_rotate(tree);
    }
}

/* If the key exists, let the safe tag be zero */
else {

```

```

    *safe_tag = 0;
}

/* Count height */
tree->height = 1 + Max(avl_height(tree->left_child),
avl_height(tree->right_child));

return tree;
}

/*
 * delete_avl -> delete a node in avl tree
 *
 * Three conditions
 * 1. The node is leaf, just delete it
 * 2. The node has two node, replaced it by the max/min node in child
tree
 * 3. The node has ome node, replaced it by its child
 *
 * return <- the root node after balanced
 */
AVL_node* delete_avl(AVL_node *tree, AVL_node *node) {
    /* Nothing to delete */
    if (tree == NULL || node == NULL)
        return NULL;
    /* Delete in left child tree */
    if (node->key < tree->key) {
        tree->left_child = delete_avl(tree->left_child, node);
        /* Balanced it */
        if (avl_height(tree->right_child) -
avl_height(tree->left_child) == 2) {
            /* The same as insert right-left */
            if (avl_height(tree->right_child->left_child) >
avl_height(tree->right_child->right_child))
                tree = rightleft_rotate(tree);
            /* The same as insert right-right */
        }
        else
            tree = rightright_rotate(tree);
    }
}

```

```

    }
    /* Delete in right child tree */
    else if (node->key > tree->key) {
        tree->right_child = delete_avl(tree->right_child, node);
        /* Balanced it */
        if (avl_height(tree->left_child) -
avl_height(tree->right_child) == 2) {
            /* The same as left-left insert */
            if (avl_height(tree->left_child->left_child) >
avl_height(tree->left_child->right_child))
                tree = leftleft_rotate(tree);
            /* The same as left-right insert */
            else
                tree = leftright_rotate(tree);
        }
    }
    /* Found the node and delete it */
    else {
        /* If the node is leaf */
        if (!(tree->left_child || tree->right_child)) {
            AVL_node* tmp = tree;
            tree = NULL;
            free(tmp);
        }
        /* If the node has two children */
        else if (tree->left_child && tree->right_child) {
            /* Left child tree higher */
            if (avl_height(tree->left_child) >
avl_height(tree->right_child)) {
                AVL_node *max_node = find_max(tree->left_child);
                tree->key = max_node->key;
                tree->left_child = delete_avl(tree->left_child,
max_node);
            }
            /* Right child tree higher */
            else {
                AVL_node *min_node = find_min(tree->right_child);
                tree->key = min_node->key;
                tree->right_child = delete_avl(tree->right_child,

```

```

min_node);
    }
}
/* If the node has one child */
else {
    AVL_node *tmp = tree;
    tree = (tree->left_child != NULL) ? tree->left_child :
tree->right_child;
    free(tmp);
}
}

return tree;
}

/*
 * clear_avl -> clear all nodes in AVL tree
 *
 * return <- None
 */
AVL_node* clear_avl(AVL_node *tree) {
    if (tree == NULL)
        return NULL;
    /* Free children */
    if (tree->left_child != NULL)
        tree->left_child = clear_avl(tree->left_child);
    if (tree->right_child != NULL)
        tree->right_child = clear_avl(tree->right_child);
    /* Free itself */
    AVL_node *tmp = tree;
    tree = NULL;
    free(tmp);
    return tree;
}

/*
 * traverse_avl -> traverse the AVL tree
 *
 * return <- None

```

```

*/
void traverse_avl(AVL_node *tree, int option, void (*visit)(int))
{
    if (tree == NULL)
        return;
    if (option == PREORDER) {
        (*visit)(tree->key);
        traverse_avl(tree->left_child, option, visit);
        traverse_avl(tree->right_child, option, visit);
    }
    else if (option == INORDER) {
        traverse_avl(tree->left_child, option, visit);
        (*visit)(tree->key);
        traverse_avl(tree->right_child, option, visit);
    }
    else if (option == POSTORDER) {
        traverse_avl(tree->left_child, option, visit);
        traverse_avl(tree->right_child, option, visit);
        (*visit)(tree->key);
    }
}

/*
 * save_avl -> save the AVL tree to a file
 *
 * return <- None
 */
void save_avl(AVL_node *tree, FILE *fp) {
    if (tree == NULL)
        return;
    save_avl(tree->left_child, fp);
    fprintf(fp, "%d\n", tree->key);
    save_avl(tree->right_child, fp);
}

/*
 * save_avl -> save the AVL tree to a file
 *
 * return <- The root

```

```

*/
AVL_node* load_avl(AVL_node *tree, FILE *fp) {
    int key;
    /* This safe tag is of no use */
    int safe_tag = 1;
    while (fscanf(fp, "%d\n", &key) == 1) {
        tree = insert_avl(tree, key, &safe_tag);
    }
    return tree;
}

/*
 * APIs implementations
 */

/*
 * API name: Init_AVL
 * Usage: Initial a new AVL tree
 * Arguments:
 *   -> int kind: The kind of an AVL tree
 *   -> int id: The id of an AVL tree
 * Return:
 *   -> AVL_tree* new_avl: The new AVL tree created
 */
AVL_tree* Init_AVL(int kind, int id) {
    AVL_tree *new_avl = (AVL_tree*)malloc(sizeof(AVL_tree));
    /* Malloc error */
    if (new_avl == NULL)
        return NULL;
    /* Initial set */
    new_avl->kind = kind;
    new_avl->id = id;
    new_avl->root = NULL;
    new_avl->num = 0;
    return new_avl;
}

/*
 * API name: Destroy_AVL

```



```

* Usage: Clear and destroy an AVL tree
* Arguments:
*   -> AVL_tree *AVL: The AVL tree to be deleted
* Return: None
*/
void Destroy_AVL(AVL_tree *AVL) {
    AVL->root = clear_avl(AVL->root);
    free(AVL);
}

/*
* API name: Clear_AVL
* Usage: Clear an AVL tree
* Arguments:
*   -> AVL_tree *AVL: The AVL tree to be cleared
* Return: None
*/
void Clear_AVL(AVL_tree *AVL) {
    AVL->root = clear_avl(AVL->root);
    AVL->num = 0;
}

/*
* API name: Search_AVL
* Usage: Find a node in an AVL tree
* Arguments:
*   -> AVL_tree *AVL: The AVL tree to be searched
*   -> int key: The key of the node to be found
* Return:
*   -> 0 - Nothing found or int node->key - the result
*/
int Search_AVL(AVL_tree *AVL, int key) {
    AVL_node *node = search_avl(AVL->root, key);
    /* If not found, return 0 */
    if (node == NULL)
        return 0;
    else
        return node->key;
}

```

```

/*
 * API name: Insert_AVL
 * Usage: Insert a node in an AVL tree
 * Arguments:
 *   -> AVL_tree *AVL: The AVL tree to be inserted
 *   -> int key: The key of the node to be inserted
 * Return: None
 */
void Insert_AVL(AVL_tree *AVL, int key) {
    /* safe tag = 0 means insert error, default 1 */
    int safe_tag = 1;
    AVL->root = insert_avl(AVL->root, key, &safe_tag);
    /* If insert ok, add num */
    if (safe_tag == 1)
        AVL->num++;
}

/*
 * API name: Delete_AVL
 * Usage: Delete a node in an AVL tree
 * Arguments:
 *   -> AVL_tree *AVL: The AVL tree include the node to be deleted
 *   -> int key: The key of the node to be inserted
 * Return: None
 */
void Delete_AVL(AVL_tree *AVL, int key) {
    AVL_node *node = search_avl(AVL->root, key);
    if (node != NULL) {
        AVL->root = delete_avl(AVL->root, node);
        AVL->num--;
    }
}

/*
 * API name: Traverse_AVL
 * Usage: Traverse the AVL tree
 * Arguments:
 *   -> AVL_tree *AVL: The AVL tree to be traverse

```

```

*   -> int option: pre/in/postorder traverse
*   -> void (*visit)(int): The visit function
*   Return: None
*/
void Traverse_AVL(AVL_tree *AVL, int option, void (*visit)(int))
{
    traverse_avl(AVL->root, option, visit);
}

/*
*   API name: Save_AVL
*   Usage: Save AVL tree to a file
*   Arguments:
*   -> AVL_tree *AVL: The AVL tree to be saved
*   -> const char *path: The data file path
*   Return: status
*/
int Save_AVL(AVL_tree *AVL, const char *path) {
    FILE *fp = fopen(path, "w");
    if (fp == NULL)
        return ERROR;
    fprintf(fp, "%d %d %d\n", AVL->id, AVL->kind, AVL->num);
    save_avl(AVL->root, fp);
    fclose(fp);
    return OK;
}

/*
*   API name: Load_AVL
*   Usage: Load AVL tree from a file
*   Arguments:
*   -> AVL_tree *AVL: The AVL tree to receive data
*   -> const char *path: The data file path
*   Return: status
*/
int Load_AVL(AVL_tree *AVL, const char *path) {
    FILE *fp = fopen(path, "r");
    if (fp == NULL)
        return ERROR;

```

```

    int id, kind, num;
    fscanf(fp, "%d %d %d\n", &id, &kind, &num);
    AVL->id = id;
    AVL->kind = kind;
    AVL->num = num;
    AVL->root = load_avl(AVL->root, fp);
    fclose(fp);
    return OK;
}

```

I.3 hashtable.h – 哈希表头文件

```

/*
 * Hash Table library
 * Developed by Pan Yue
 * zxc479773533@gmail.com
 */
#ifndef PY_HASH_LIB
#define PY_HASH_LIB

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

/* Status define */
#define ERROR 0
#define OK 1
#define EXISTS 2

/* The hash node struct */
typedef struct HashNode {
    int count;
    int key;
    char value[18];
    struct HashNode *next;
} HashNode;

/* The hashtable struct */
typedef struct HashTable {
    int num;
    int hashlen;

```

```

    HashNode **Hash;
} HashTable;

/* APIs */
extern HashTable* Init_Hash(int hashlen);
extern int Insert_Hash(HashTable *MyHash, int key, char *value);
extern int Delete_Hash(HashTable *MyHash, int key);
extern char* Search_Hash(HashTable *MyHash, int key);
extern int Change_Hash(HashTable *MyHash, int key, char *value);
extern void Traverse_Hash(HashTable *MyHash, void (*visit)(struct HashNode*));
extern int Save_Hash(HashTable *MyHash, const char *path);
extern int Load_Hash(HashTable *MyHash, const char *path);

/* Functions */
int HashFunc(int key, int mod);
int insert_Hash(HashNode **Hash, int key, char *value, int hashlen);
int delete_Hash(HashNode **Hash, int key, int hashlen);
HashNode* search_Hash(HashNode **Hash, int key, int hashlen);
int change_Hash(HashNode **Hash, int key, char *value, int hashlen);
void traverse_Hash(HashNode **Hash, int hashlen, void (*visit)(struct HashNode*));
void save_Hash(HashNode **Hash, int hashlen, FILE *fp);
void load_Hash(HashNode **Hash, int hashlen, FILE *fp);

#endif // !PY_HASH_LIB

```

I.4 hashtable.c – 哈希表具体实现

```

/*
 * Hash Table implemention
 * Developed by Pan Yue
 * zxc479773533@gmail.com
 */
#include "hashtable.h"

/*
 * HashFunc -> The hash function
 *

```

```

* return <- the height
*/
int HashFunc(int key, int mod) {
    return key % mod;
}

/*
* insert_Hash -> Insert a node in hash func
*
* return <- the status
*      OK: insert ok
*      ERROR: insert error
*      EXISTS: the node has existed, used for no-repeated
*/
int insert_Hash(HashNode **Hash, int key, char *value, int hashlen)
{
    int status = EXISTS;
    int index = HashFunc(key, hashlen);
    HashNode *node = Hash[index]->next;
    while (node != NULL) {
        if (node->key == key) {
            /* Here I set no-repeated hash table */
            if (node->count == 0) {
                node->count++;
                status = OK;
            }
            break;
        }
        node = node->next;
    }
    if (node == NULL) {
        HashNode *new_node = (HashNode*)malloc(sizeof(HashNode));
        /* Malloc ERROR */
        if (new_node == NULL)
            return ERROR;
        new_node->key = key;
        strcpy(new_node->value, value);
        new_node->count = 1;
        new_node->next = Hash[index]->next;
        Hash[index]->next = new_node;
        status = OK;
    }
}

```

```

    return status;
}

/*
 * delete_Hash -> Delete a node in hash func
 *
 * return <- the status
 *         OK: Delete ok
 *         ERROR: Delete error
 */
int delete_Hash(HashNode **Hash, int key, int hashlen) {
    int status = ERROR;
    int index = HashFunc(key, hashlen);
    HashNode *node = Hash[index]->next;
    while (node != NULL) {
        if (node->key == key) {
            if (node->count > 0) {
                node->count--;
                strcpy(node->value, "Disable");
                status = OK;
            }
            break;
        }
        node = node->next;
    }
    return status;
}

/*
 * search_Hash -> Search a node in an Hash table
 *
 * return <- The node
 */
HashNode* search_Hash(HashNode **Hash, int key, int hashlen) {
    int index = HashFunc(key, hashlen);
    HashNode *node = Hash[index]->next;
    while (node != NULL) {
        if (node->key == key && node->count > 0)
            return node;
        node = node->next;
    }
    return NULL;
}

```

```

}

/*
 * change_Hash -> Change an Hash node's name
 *
 * return <- The state
 */
int change_Hash(HashNode **Hash, int key, char *value, int hashlen)
{
    HashNode *node = search_Hash(Hash, key, hashlen);
    if (node == NULL)
        return ERROR;
    else {
        strcpy(node->value, value);
        return OK;
    }
}

/*
 * traverse_Hash -> Traverse Hash table
 *
 * return <- None
 */
void traverse_Hash(HashNode **Hash, int hashlen, void
(*visit)(struct HashNode*)) {
    int index;
    for (index = 0; index < hashlen; index++) {
        HashNode *node = Hash[index]->next;
        while (node != NULL) {
            (*visit)(node);
            node = node->next;
        }
    }
}

/*
 * save_Hash -> Save Hash table to a file
 *
 * return <- None
 */
void save_Hash(HashNode **Hash, int hashlen, FILE *fp) {
    int index;

```



```

    for (index = 0; index < hashlen; index++) {
        HashNode *node = Hash[index]->next;
        while (node != NULL) {
            fprintf(fp, "%d %s\n", node->key, node->value);
            node = node->next;
        }
    }
}

/*
 * load_Hash -> Load Hash table from a file
 *
 * return <- None
 */
void load_Hash(HashNode **Hash, int hashlen, FILE *fp) {
    int index;
    int key;
    char value[12];
    while (fscanf(fp, "%d %s\n", &key, value) == 2) {
        insert_Hash(Hash, key, value, hashlen);
    }
}

/*
 * APIs implementations
 */

/*
 * API name: Init_Hash
 * Usage: Initial a new Hash table
 * Arguments:
 *   -> int hashlen: The length of an Hash table
 * Return:
 *   -> HashTable* MyHash: The new Hash table created
 */
HashTable* Init_Hash(int hashlen) {
    HashTable *MyHash = (HashTable*)malloc(sizeof(HashTable));
    MyHash->num = 0;
    MyHash->hashlen = hashlen;
    MyHash->Hash = (HashNode**)malloc(sizeof(HashNode) * hashlen);
    memset(MyHash->Hash, 0, sizeof(HashNode) * hashlen);
    for (int i = 0; i < hashlen; i++) {

```

```

    MyHash->Hash[i] = (HashNode*)malloc(sizeof(HashNode));
    MyHash->Hash[i]->key = 0;
    MyHash->Hash[i]->count = 0;
    MyHash->Hash[i]->next = NULL;
}
return MyHash;
}

/*
 * API name: Insert_Hash
 * Usage: Insert a node in hash table
 * Arguments:
 *   -> HashTable *MyHash: The Hash table to be inserted
 *   -> int key: The key of new node
 * Return:
 *   -> int status: The status of this function
 */
int Insert_Hash(HashTable *MyHash, int key, char *value) {
    int status = insert_Hash(MyHash->Hash, key, value,
MyHash->hashlen);
    if (status == OK)
        MyHash->num++;
    return status;
}

/*
 * API name: Delete_Hash
 * Usage: Delete a node in hash table
 * Arguments:
 *   -> HashTable *MyHash: The Hash table to be deleted
 *   -> int key: The key of node to be deleted
 * Return:
 *   -> int status: The status of this function
 */
int Delete_Hash(HashTable *MyHash, int key) {
    if (MyHash->num == 0)
        return ERROR;
    int status = delete_Hash(MyHash->Hash, key, MyHash->hashlen);
    /*if (status == OK)
        MyHash->num--;*/
    return status;
}

```

```

/*
 * API name: Search_Hash
 * Usage: Search a node in an Hash table
 * Arguments:
 *   -> HashTable *MyHash: The Hash table to be searched
 *   -> int key: The key of node to be searched
 * Return:
 *   -> int status: The value of the node
 */
char* Search_Hash(HashTable *MyHash, int key) {
    if (MyHash->num == 0)
        return NULL;
    HashNode *node = search_Hash(MyHash->Hash, key,
MyHash->hashlen);
    if (node != NULL)
        return node->value;
    else
        return NULL;
}

/*
 * API name: Save_Hash
 * Usage: Save Hast table to a file
 * Arguments:
 *   -> HashTable *MyHash: The Hash table to be saved
 *   -> const char *path: The data file path
 * Return: status
 */
int Save_Hash(HashTable *MyHash, const char *path) {
    FILE *fp = fopen(path, "w");
    if (fp == NULL)
        return ERROR;
    fprintf(fp, "%d %d\n", MyHash->num, MyHash->hashlen);
    save_Hash(MyHash->Hash, MyHash->hashlen, fp);
    fclose(fp);
    return OK;
}

/*
 * API name: Load_Hash
 * Usage: Load Hast table from a file

```

```

* Arguments:
*   -> HashTable *MyHash: The Hash table to receive data
*   -> const char *path: The data file path
* Return: status
*/
int Load_Hash(HashTable *MyHash, const char *path) {
    FILE *fp = fopen(path, "r");
    if (fp == NULL)
        return ERROR;
    int num, hashlen;
    fscanf(fp, "%d %d\n", &num, &hashlen);
    MyHash->num = num;
    MyHash->hashlen = hashlen;
    load_Hash(MyHash->Hash, MyHash->hashlen, fp);
    fclose(fp);
    return OK;
}

/*
* API name: Traverse_Hash
* Usage: Traverse Hash table
* Arguments:
*   -> HashTable *MyHash: The Hash table to be traversed
*   -> void (*visit)(struct HashNode*): The traverse function
* Return: None
*/
void Traverse_Hash(HashTable *MyHash ,void (*visit)(struct
HashNode*)) {
    traverse_Hash(MyHash->Hash, MyHash->hashlen, visit);
}

/*
* API name: Change_Hash
* Usage: Change an Hash node's name
* Arguments:
*   -> HashTable *MyHash: The Hash table to be changed
*   -> int key: The node's key
*   -> char *value: The new value
* Return: None
*/
int Change_Hash(HashTable *MyHash, int key, char *value) {
    return change_Hash(MyHash->Hash, key, value, MyHash->hashlen);
}

```

```
}
```

I.5 set.h – 集合头文件

```
/*
 * Set function prototype
 * Developed by Pan Yue
 * zxc479773533@gmail.com
 */
#ifndef PY_SET
#define PY_SET

/* Set - implemented by AVL tree */
#include "avltree.h"

typedef struct Set {
    AVL_tree *Elem;
} Set;

/* Set operations */
Set* Set_Init(int kind, int id);
void Set_Destroy(Set *Set);
void Set_Clear(Set *Set);
void Set_Insert(Set *Set, int elem);
void Set_Delete(Set *Set, int elem);
Set* Set_Intersection(Set *Set_A, Set *Set_B);
Set* Set_Union(Set *Set_A, Set* Set_B);
Set* Set_Difference(Set* Set_A, Set* Set_B);
int Set_Size(Set *Set);
int Set_Member(Set *Set, int elem);
int Set_Subset(Set *Set_A, Set* Set_B);
int Set_Equal(Set *Set_A, Set* Set_B);

#endif // !PY_SET
```

I.6 set.c – 集合具体实现

```
/*
```

```

* Set implementations
* Developed by Pan Yue
* zxc479773533@gmail.com
*/
#include "set.h"

/*
* Set_Init -> Create a new set
*
* return <- The new set created
*/
Set* Set_Init(int kind, int id) {
    Set* new_set = (Set*)malloc(sizeof(Set));
    new_set->Elem = Init_AVL(kind, id);
    return new_set;
}

/*
* Set_Destroy -> Destroy a set
*
* return <- None
*/
void Set_Destroy(Set *Set) {
    Destroy_AVL(Set->Elem);
    free(Set);
}

/*
* Set_Clear -> Clear a set
*
* return <- None
*/
void Set_Clear(Set *Set) {
    Clear_AVL(Set->Elem);
}

/*
* Set_Insert -> Insert an element in a set
*
* return <- None
*/
void Set_Insert(Set *Set, int elem) {

```

```

    Insert_AVL(Set->Elem, elem);
}

/*
 * Set_Delete -> Delete an element in a set
 *
 * return <- None
 */
void Set_Delete(Set *Set, int elem) {
    Delete_AVL(Set->Elem, elem);
}

/*
 * Set_traverse -> Assist function for intersection, union and
difference
 *
 * Here I use inorder traverse to keep the order
 *
 * The function visit do the operation
 *
 * return <- None
 */
void Set_traverse(AVL_node *tree, AVL_tree *Set_B, AVL_tree
*Set_result, void (*visit)(int, struct AVL_tree*, struct
AVL_tree*)) {
    if (tree == NULL)
        return;
    Set_traverse(tree->left_child, Set_B, Set_result, visit);
    visit(tree->key, Set_B, Set_result);
    Set_traverse(tree->right_child, Set_B, Set_result, visit);
}

/* Assist function fot intetsection */
void visit_intersection(int elem, AVL_tree *Set_B, AVL_tree
*Set_result) {
    int key = Search_AVL(Set_B, elem);
    if (key != 0)
        Insert_AVL(Set_result, key);
}

/* Assist function fot union */
void visit_union(int elem, AVL_tree *Set_B, AVL_tree *Set_result)

```

```

{
    Insert_AVL(Set_result, elem);
}

/* Assist function fot difference */
void visit_difference(int elem, AVL_tree *Set_B, AVL_tree
*Set_result) {
    int key = Search_AVL(Set_B, elem);
    if (key == 0)
        Insert_AVL(Set_result, elem);
}

/*
 * Set_Intersection -> A  $\cup$  B
 *
 * return <- A  $\cup$  B
 */
Set* Set_Intersection(Set *Set_A, Set *Set_B) {
    Set *new_set = Set_Init(Set_A->Elem->kind, Set_A->Elem->id);
    Set_traverse(Set_A->Elem->root, Set_B->Elem, new_set->Elem,
visit_intersection);
    return new_set;
}

/*
 * Set_Union -> A  $\cap$  B
 *
 * return <- A  $\cap$  B
 */
Set* Set_Union(Set *Set_A, Set* Set_B) {
    Set *new_set = Set_Init(Set_A->Elem->kind, Set_A->Elem->id);
    Set_traverse(Set_A->Elem->root, Set_B->Elem, new_set->Elem,
visit_union);
    Set_traverse(Set_B->Elem->root, Set_A->Elem, new_set->Elem,
visit_union);
    return new_set;
}

/*
 * Set_Difference -> A - B

```



```

*
* return <- A - B
*/
Set* Set_Difference(Set* Set_A, Set* Set_B) {
    Set *new_set = Set_Init(Set_A->Elem->kind, Set_A->Elem->id);
    Set_traverse(Set_A->Elem->root, Set_B->Elem, new_set->Elem,
visit_difference);
    return new_set;
}

/*
* Set_Size -> Get number of elements in a set
*
* return <- The set size
*/
int Set_Size(Set *Set) {
    if (Set == NULL || Set->Elem == NULL)
        return 0;
    else
        return Set->Elem->num;
}

/*
* Set_Member -> Judge whether an element is in a set
*
* return <- The status
*/
int Set_Member(Set *Set, int elem) {
    int key = Search_AVL(Set->Elem, elem);
    if (key == 0)
        return ERROR;
    else
        return OK;
}

/*
* Set_Subset -> Judge whether set A is subset of B
*
* Here I use  $A - B = \emptyset$  to judge
*
* return <- The status
*/

```

```

int Set_Subset(Set *Set_A, Set* Set_B) {
    Set *new_set = Set_Difference(Set_A, Set_B);
    if (new_set->Elem->num == 0)
        return OK;
    else
        return ERROR;
}

/*
 * Set_Equal -> Judge whether set A is equal to set B
 *
 * Here I use  $A - B = \emptyset$  &&  $B - A = \emptyset$  to judge
 *
 * return <- The status
 */
int Set_Equal(Set *Set_A, Set* Set_B) {
    if (Set_Subset(Set_A, Set_B) && Set_Subset(Set_B, Set_A))
        return OK;
    else
        return ERROR;
}

```

II.应用层文件

II.1 py_prompt.c – shell 框架

```

/*
 * PY prompt
 * Developed by Pan Yue
 * zxc479773533@gmail.com
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "py_exec.c"

/* Constants */
#define MAXLINE 1024
#define MAXARGS 128

```

```

/* Global variables */
char prompt[] = "py_prompt > ";
char theme[] = "Here to control a friend and followers management
system";

/* Founction prototypes */
void eval(char *cmdline);
int parseline(const char *cmdline, char **argv);

/* Print usage messages */
void print_usage(void) {
    printf("Usage: py_manager [options]\n");
    printf("Options:\n");
    printf("\t-h: print this massges\n");
    printf("\t-p: hide the prompt\n");
    printf("\tdefault: start shell\n");
}

int Start_Shell(int argc, char **argv) {
    char ch;
    /* The command line */
    char cmdline[MAXLINE];
    /* Decide whether print a prompt, default yes*/
    int emit_prompt = 1;

    /* Parse the command line */
    while ((ch = getopt(argc, argv, "hp")) != EOF) {
        switch(ch) {
            case 'h':
                print_usage();
                exit(0);
            case 'p':
                emit_prompt = 0;
                break;
            default:
                print_usage();
                exit(1);
        }
    }

    /* Print informations */
    printf("[INFO] Wherecome to Pan Yue's interactive shell!\n");
}

```

```

printf("[INFO] %s\n", theme);
printf("[INFO] Try help to see how to use this Controller\n");

/* Excute the shell's read/eval loop */
while (1) {

    /* Print prompt */
    if (emit_prompt) {
        printf("%s", prompt);
        fflush(stdout);
    }
    if ((fgets(cmdline, MAXLINE, stdin) == NULL) && ferror(stdin))
{
        printf("fgets error\n");
        exit(1);
    }
    if (feof(stdin)) {
        printf("[EXIT] Parser reached end-of-file. Terminated!\n");
        fflush(stdout);
        exit(0);
    }

    /* Evaluate the command line */
    eval(cmdline);
    fflush(stdout);
}

return 0;
}

/*
 * eval - Evaluate the command line that the user has just typed
in
 *
 * If the user has requested a built-in command, then excute it
immediately.
 */
void eval(char *cmdline) {
    char *argv[MAXARGS];
    char buf[MAXLINE];
    int argc;

```

```

/* Parse command */
strcpy(buf, cmdline);
argc = parseline(buf, argv);

if (argv[0] == NULL)
    return;

if (!builtin_cmd(argv)) {
    if (!py_execute(argv[0], argc, argv))
        printf("[INFO] pysh: command not found.\n");
    else
        printf("[INFO] Operation finished.\n\n");
}
}

/*
 * parseline - Parse the command line and build the argv array.
 */
int parseline(const char *cmdline, char **argv) {
    /* Holds local copy of command line */
    static char array[MAXLINE];
    char *buf = array;
    char *delim;
    int argc;

    strcpy(buf, cmdline);
    buf[strlen(buf) - 1] = ' ';
    /* Ignore leading spaces */
    while (*buf && (*buf == ' '))
        buf++;

    /* Build the argv list */
    argc = 0;
    if (*buf == '\\') {
        buf++;
        delim = strchr(buf, '\\');
    }
    else {
        delim = strchr(buf, ' ');
    }

    while(delim) {

```

```

    argv[argc++] = buf;
    *delim = '\0';
    buf = delim + 1;
    while (*buf && (*buf == ' '))
        buf++;

    if (*buf == '\\') {
        buf++;
        delim = strchr(buf, '\\');
    }
    else {
        delim = strchr(buf, ' ');
    }
}
argv[argc] = NULL;
return argc;
}

```

II.2 py_exec.c 应用层执行文件

```

/*
 * PY execute
 * Developed by Pan Yue
 * zxc479773533@gmail.com
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "../set.h"
#include "../hashtable.h"

/* Database HASH length */
#define HASHLEN 128

/* User database */
const char database_path[] = "../src/data/users.txt";
const char hobby__path[] = "../src/data/hobby.txt";

```

```
/* Function prototypes */

/* Add functions */
void Add_user(int user_id, char *user_name);
void Add_hobby(int user_id, char *hobby);
void Add_friend(int user_id, int friend_id, int tag);
void Add_follower(int user_id, int follower_id, int tag);
void Add_following(int user_id, int following_id, int tag);

/* Delete functions */
void Delete_user(int user_id);
void Delete_friend(int user_id, int friend_id, int tag);
void Delete_follower(int user_id, int follower_id, int tag);
void Delete_following(int user_id, int following_id, int tag);

/* Find functions */
void Find_user(int user_id);
void Find_friend(int user_id, int friend_id);
void Find_follower(int user_id, int follower_id);
void Find_following(int user_id, int following_id);

/* Display functions */
void Display_friend(int user_id);
void Display_follower(int user_id);
void Display_following(int user_id);

/* Common display fuctions */
void Common_friends(int user_a, int user_b);
void Common_followers(int user_a, int user_b);
void Common_followings(int user_a, int user_b);

/* Recommend functions */
void Recommend_friends(int user_id);
void Recommend_followers(int user_id);
void Recommend_followings(int user_id);

/* Others */
void Change_name(int user_id, char *new_name);
```

```

/*
 * print_help - Print help messages
 */
void print_help(void) {
    printf("User Manage System vesion 1.1\n");
    printf("Author: Pan Yue\n");
    printf("Basic function:\n");
    printf("    help: Display User's Manual\n");
    printf("    show: Show all users in database\n");
    printf("    rename: Change a user's name\n");
    printf("    rename [user ID] [new name]\n");
    printf("\n");
    printf("    add: Add data\n");
    printf("    add [option] [arg1] [arg2]\n");
    printf("        option=user, add a user, [arg1]=user ID,
[arg2]=user name\n");
    printf("        option=hobby, add a user's hobby, [arg1]=user ID,
[arg2]=hobby\n");
    printf("        option=friend, add a friend, [arg1]=user ID,
[arg2]=friend ID\n");
    printf("        option=follower, add a follower, [arg1]=user ID,
[arg2]=follower ID\n");
    printf("        option=following, add a following, [arg1]=user
ID, [arg2]=following ID\n");
    printf("\n");
    printf("    delete: Delete data\n");
    printf("    delete [option] [arg1] [arg2]\n");
    printf("        option=user, delete a user, [arg1]=user ID\n");
    printf("        option=friend, delete a friend, [arg1]=user ID,
[arg2]=friend ID\n");
    printf("        option=follower, delete a follower, [arg1]=user
ID, [arg2]=follower ID\n");
    printf("        option=following, delete a following,
[arg1]=user ID, [arg2]=following ID\n");
    printf("\n");
    printf("    find: Find data\n");
    printf("    find [option] [arg1] [arg2]\n");
    printf("        option=user, find a user, [arg1]=user ID\n");
    printf("        option=friend, find a friend, [arg1]=user ID,

```



```
[arg2]=friend ID\n");
    printf("        option=follower, find a follower, [arg1]=user ID,
[arg2]=follower ID\n");
    printf("        option=following, find a following, [arg1]=user
ID, [arg2]=following ID\n");
    printf("\n");
    printf("    display: Display data\n");
    printf("    display [option] [user ID]\n");
    printf("        option=friend/follower/following, then display
this user's all [option]\n");
    printf("\n");
    printf("    recommend: Recommend for a user\n");
    printf("    recommend [option] [user ID]\n");
    printf("        option=friend/follower/following, then
recommend [option] for user\n");
    printf("\n");
    printf("    common: Display common data of 2 users\n");
    printf("    common [option] [user A] [user B]\n");
    printf("        option=friend/follower/following, then display
2 user's common [option]\n");
}

/* Assist function to show users */
void show_users_visit(HashNode *node) {
    printf("%12d %18s\n", node->key, node->value);
}

void show_users(void) {
    HashTable *MyHash = Init_Hash(HASHLEN);
    Load_Hash(MyHash, database_path);
    printf("%12s %18s", "User ID", "Name\n");
    Traverse_Hash(MyHash, show_users_visit);
}

/*
 * builtin_cmd -> Judge buildin command
 *
 * return <- the result
 */
```

```

int builtin_cmd(char **argv) {
    /* Not a built-in command */
    if (!strcmp(argv[0], "exit")) {
        printf("[INFO] User-exit. Terminated!\n");
        exit(0);
    }
    if (!strcmp(argv[0], "help")) {
        print_help();
        return 1;
    }
    if (!strcmp(argv[0], "show")) {
        show_users();
        return 1;
    }

    return 0;
}

/*
 * generate_filename -> Generate the data file name
 *
 * return <- None
 */
void generate_filename(Set *Set, char *filename) {
    sprintf(filename,    "./src/data/%d-%d.txt",    Set->Elem->id,
Set->Elem->kind);
}

/*
 * py_execute -> The main execute function
 *
 * return <- None
 */
int py_execute(char *func , int argc, char **argv) {
    if (!strcmp(func, "add")) {
        if (argc == 4 && !strcmp(argv[1], "user")) {
            Add_user(atoi(argv[2]), argv[3]);
            return 1;
        }
    }
}

```

```

else if (argc == 4 && !strcmp(argv[1], "hobby")) {
    Add_hobby(atoi(argv[2]), argv[3]);
    return 1;
}
else if (argc == 4 && !strcmp(argv[1], "friend")) {
    Add_friend(atoi(argv[2]), atoi(argv[3]), 1);
    return 1;
}
else if (argc == 4 && !strcmp(argv[1], "follower")) {
    Add_follower(atoi(argv[2]), atoi(argv[3]), 1);
    return 1;
}
else if (argc == 4 && !strcmp(argv[1], "following")) {
    Add_following(atoi(argv[2]), atoi(argv[3]), 1);
    return 1;
}
}
else if (!strcmp(func, "delete")) {
    if (argc == 3 && !strcmp(argv[1], "user")) {
        Delete_user(atoi(argv[2]));
        return 1;
    }
    else if (argc == 4 && !strcmp(argv[1], "friend")) {
        Delete_friend(atoi(argv[2]), atoi(argv[3]), 1);
        return 1;
    }
    else if (argc == 4 && !strcmp(argv[1], "follower")) {
        Delete_follower(atoi(argv[2]), atoi(argv[3]), 1);
        return 1;
    }
    else if (argc == 4 && !strcmp(argv[1], "following")) {
        Delete_following(atoi(argv[2]), atoi(argv[3]), 1);
        return 1;
    }
}
else if (!strcmp(func, "find")) {
    if (argc == 3 && !strcmp(argv[1], "user")) {
        Find_user(atoi(argv[2]));
        return 1;
    }
}

```

```

    }
    else if (argc == 4 && !strcmp(argv[1], "friend")) {
        Find_friend(atoi(argv[2]), atoi(argv[3]));
        return 1;
    }
    else if (argc == 4 && !strcmp(argv[1], "follower")) {
        Find_follower(atoi(argv[2]), atoi(argv[3]));
        return 1;
    }
    else if (argc == 4 && !strcmp(argv[1], "following")) {
        Find_following(atoi(argv[2]), atoi(argv[3]));
        return 1;
    }
}
else if (!strcmp(func, "display")) {
    if (argc == 3 && !strcmp(argv[1], "friend")) {
        Display_friend(atoi(argv[2]));
        return 1;
    }
    else if (argc == 3 && !strcmp(argv[1], "follower")) {
        Display_follower(atoi(argv[2]));
        return 1;
    }
    else if (argc == 3 && !strcmp(argv[1], "following")) {
        Display_following(atoi(argv[2]));
        return 1;
    }
}
else if (!strcmp(func, "common")) {
    if (argc == 4 && !strcmp(argv[1], "friend")) {
        Common_friends(atoi(argv[2]), atoi(argv[3]));
        return 1;
    }
    else if (argc == 4 && !strcmp(argv[1], "follower")) {
        Common_followers(atoi(argv[2]), atoi(argv[3]));
        return 1;
    }
    else if (argc == 4 && !strcmp(argv[1], "following")) {
        Common_followings(atoi(argv[2]), atoi(argv[3]));
    }
}

```

```

        return 1;
    }
}
else if (argc == 3 && !strcmp(func, "recommend")) {
    if (argc == 3 && !strcmp(argv[1], "friend")) {
        Recommend_friends(atoi(argv[2]));
        return 1;
    }
    else if (argc == 3 && !strcmp(argv[1], "follower")) {
        Recommend_followers(atoi(argv[2]));
        return 1;
    }
    else if (argc == 3 && !strcmp(argv[1], "following")) {
        Recommend_followings(atoi(argv[2]));
        return 1;
    }
}
else if (argc == 3 && !strcmp(func, "rename")) {
    Change_name(atoi(argv[1]), argv[2]);
    return 1;
}
return 0;
}

/*
 * check_user - Check if user exists
 */
int check_user(int user_id) {
    HashTable *MyHash = Init_Hash(HASHLEN);
    Load_Hash(MyHash, database_path);
    char *user_name = Search_Hash(MyHash, user_id);
    if (user_name == NULL)
        return 0;
    else
        return 1;
}

/*
 * Add_user - Add a new user in database

```

```

*/
void Add_user(int user_id, char *user_name) {
    int status;
    HashTable *MyHash = Init_Hash(HASHLEN);
    Load_Hash(MyHash, database_path);
    status = Insert_Hash(MyHash, user_id, user_name);
    Save_Hash(MyHash, database_path);
    /* Print info */
    if (status == EXISTS)
        printf("[ERROR] The user has exists!\n");
    else if (status == ERROR)
        printf("[ERROR] Add user faild! Please check your imput.\n");
}

/*
 * Add_like - Add a user's hobby
 */
void Add_hobby(int user_id, char *hobby) {
    /* Check if user exists */
    if (!check_user(user_id)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    int status;
    HashTable *MyHash = Init_Hash(HASHLEN);
    Load_Hash(MyHash, hobby__path);
    /* Find the old hobby */
    char *old_hobby = Search_Hash(MyHash, user_id);
    if (old_hobby == NULL) {
        status = Insert_Hash(MyHash, user_id, hobby);
        Save_Hash(MyHash, hobby__path);
        /* Print info */
        if (status == EXISTS)
            printf("[ERROR] The user's hobby has existd!\n");
        else if (status == ERROR)
            printf("[ERROR] Add hobby faild! Please check your
imput.\n");
    }
    else {

```

```

        strcpy(old_hobby, hobby);
        Save_Hash(MyHash, hobby__path);
    }
}

/*
 * Add_friend - Add a new friend in a user's set
 */
void Add_friend(int user_id, int friend_id, int tag) {
    /* Check if user exists */
    if (!check_user(user_id) || !check_user(friend_id)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename[25];
    Set *new_set = Set_Init(FRIENDS, user_id);
    generate_filename(new_set, filename);
    Load_AVL(new_set->Elem, filename);
    Set_Insert(new_set, friend_id);
    Save_AVL(new_set->Elem, filename);
    if (tag == 1)
        Add_friend(friend_id, user_id, 0);
}

/*
 * Add_follower - Add a new follower in a user's set
 */
void Add_follower(int user_id, int follower_id, int tag) {
    /* Check if user exists */
    if (!check_user(user_id) || !check_user(follower_id)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename[25];
    Set *new_set = Set_Init(FOLLOWERS, user_id);
    generate_filename(new_set, filename);
    Load_AVL(new_set->Elem, filename);
    Set_Insert(new_set, follower_id);
    Save_AVL(new_set->Elem, filename);
}

```

```

    if (tag == 1)
        Add_following(follower_id, user_id, 0);
}

/*
 * Add_following - Add a new following in a user's set
 */
void Add_following(int user_id, int following_id, int tag) {
    /* Check if user exists */
    if (!check_user(user_id) || !check_user(following_id)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename[25];
    Set *new_set = Set_Init(FOLLOWING, user_id);
    generate_filename(new_set, filename);
    Load_AVL(new_set->Elem, filename);
    Set_Insert(new_set, following_id);
    Save_AVL(new_set->Elem, filename);
    if (tag == 1)
        Add_follower(following_id, user_id, 0);
}

/*
 * Delete_user - Delete a user in database
 */
void Delete_user(int user_id) {
    int status;
    HashTable *MyHash = Init_Hash(HASHLEN);
    Load_Hash(MyHash, database_path);
    status = Delete_Hash(MyHash, user_id);
    Save_Hash(MyHash, database_path);
    if (status == ERROR)
        printf("[ERROR] Detele faild! Please check your input.\n");
}

/*
 * Delete_friend - Delete a friend in a user's set
 */

```



```

void Delete_friend(int user_id, int friend_id, int tag) {
    /* Check if user exists */
    if (!check_user(user_id) || !check_user(friend_id)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename[25];
    Set *new_set = Set_Init(FRIENDS, user_id);
    generate_filename(new_set, filename);
    Load_AVL(new_set->Elem, filename);
    Set_Delete(new_set, friend_id);
    Save_AVL(new_set->Elem, filename);
    if (tag == 1)
        Delete_friend(friend_id, user_id, 0);
}

/*
 * Delete_follower - Delete a follower in a user's set
 */
void Delete_follower(int user_id, int follower_id, int tag) {
    /* Check if user exists */
    if (!check_user(user_id) || !check_user(follower_id)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename[25];
    Set *new_set = Set_Init(FOLLOWERS, user_id);
    generate_filename(new_set, filename);
    Load_AVL(new_set->Elem, filename);
    Set_Delete(new_set, follower_id);
    Save_AVL(new_set->Elem, filename);
    if (tag == 1)
        Delete_following(follower_id, user_id, 0);
}

/*
 * Delete_following - Delete a follower in a user's set
 */
void Delete_following(int user_id, int following_id, int tag) {

```

```

/* Check if user exists */
if (!check_user(user_id) || !check_user(following_id)) {
    printf("[ERROR] Failed! This user is not exists!\n");
    return;
}
char filename[25];
Set *new_set = Set_Init(FOLLOWING, user_id);
generate_filename(new_set, filename);
Load_AVL(new_set->Elem, filename);
Set_Delete(new_set, following_id);
Save_AVL(new_set->Elem, filename);
if (tag == 1)
    Delete_follower(following_id, user_id, 0);
}

/*
 * Find_user - Find a user in database
 */
void Find_user(int user_id) {
    char friend_file[20], follower_file[20], following_file[20];
    HashTable *MyHash = Init_Hash(HASHLEN);
    Load_Hash(MyHash, database_path);
    char *user_name = Search_Hash(MyHash, user_id);
    HashTable *HobbyHash = Init_Hash(HASHLEN);
    Load_Hash(HobbyHash, hobby__path);
    char *hobby = Search_Hash(HobbyHash, user_id);
    if (user_name == NULL) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    Set *friend_set = Set_Init(FRIENDS, user_id);
    Set *follower_set = Set_Init(FOLLOWERS, user_id);
    Set *following_set = Set_Init(FOLLOWING, user_id);
    generate_filename(friend_set, friend_file);
    generate_filename(follower_set, follower_file);
    generate_filename(following_set, following_file);
    Load_AVL(friend_set->Elem, friend_file);
    Load_AVL(follower_set->Elem, follower_file);
    Load_AVL(following_set->Elem, following_file);
}

```

```

printf("User ID: %d\n", user_id);
printf("User Name: %s\n", user_name);
printf("Friend: %d\n", friend_set->Elem->num);
printf("Follower: %d\n", follower_set->Elem->num);
printf("Following: %d\n", following_set->Elem->num);
if (hobby == NULL)
    printf("Hobby: None\n");
else
    printf("Hobby: %s\n", hobby);
}

/*
 * Find_friend - Find a friend in a user's set
 */
void Find_friend(int user_id, int friend_id) {
    /* Check if user exists */
    if (!check_user(user_id) || !check_user(friend_id)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename[25];
    Set *new_set = Set_Init(FRIENDS, user_id);
    generate_filename(new_set, filename);
    Load_AVL(new_set->Elem, filename);
    if (Set_Member(new_set, friend_id) == OK){
        HashTable *MyHash = Init_Hash(HASHLEN);
        Load_Hash(MyHash, database_path);
        printf("[INFO] Found it! Your friend %s.\n",
Search_Hash(MyHash, friend_id));
    }
    else {
        printf("[INFO] Not Found!\n");
    }
}

/*
 * Find_follower - Find a follower in a user's set
 */
void Find_follower(int user_id, int follower_id) {

```

```

/* Check if user exists */
if (!check_user(user_id) || !check_user(follower_id)) {
    printf("[ERROR] Failed! This user is not exists!\n");
    return;
}
char filename[25];
Set *new_set = Set_Init(FOLLOWERS, user_id);
generate_filename(new_set, filename);
Load_AVL(new_set->Elem, filename);
if (Set_Member(new_set, follower_id) == OK){
    HashTable *MyHash = Init_Hash(HASHLEN);
    Load_Hash(MyHash, database_path);
    printf("[INFO] Found it! Your follower %s.\n",
Search_Hash(MyHash, follower_id));
}
else {
    printf("[INFO] Not Found!\n");
}
}

/*
 * Find_following - Find a following in a user's set
 */
void Find_following(int user_id, int following_id) {
    /* Check if user exists */
    if (!check_user(user_id) || !check_user(following_id)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename[25];
    Set *new_set = Set_Init(FOLLOWING, user_id);
    generate_filename(new_set, filename);
    Load_AVL(new_set->Elem, filename);
    if (Set_Member(new_set, following_id) == OK){
        HashTable *MyHash = Init_Hash(HASHLEN);
        Load_Hash(MyHash, database_path);
        printf("[INFO] Found it! You has followed %s.\n",
Search_Hash(MyHash, following_id));
    }
}

```

```

    else {
        printf("[INFO] Not Found!\n");
    }
}

/* Assist function to show users */
void dispaly_visit(int user_id) {
    HashTable *MyHash = Init_Hash(HASHLEN);
    Load_Hash(MyHash, database_path);
    char *user_name = Search_Hash(MyHash, user_id);
    printf("User ID: %d, Name: %s\n", user_id, user_name);
}

/*
 * Display_friend - Display all of a user's friends
 */
void Display_friend(int user_id) {
    /* Check if user exists */
    if (!check_user(user_id)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename[25];
    Set *new_set = Set_Init(FRIENDS, user_id);
    generate_filename(new_set, filename);
    Load_AVL(new_set->Elem, filename);
    Traverse_AVL(new_set->Elem, INORDER, dispaly_visit);
}

/*
 * Display_follower - Display all of a user's followers
 */
void Display_follower(int user_id) {
    /* Check if user exists */
    if (!check_user(user_id)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename[25];

```

```

    Set *new_set = Set_Init(FOLLOWERS, user_id);
    generate_filename(new_set, filename);
    Load_AVL(new_set->Elem, filename);
    Traverse_AVL(new_set->Elem, INORDER, dispaly_visit);
}

/*
 * Display_following - Display all of a user's followings
 */
void Display_following(int user_id) {
    /* Check if user exists */
    if (!check_user(user_id)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename[25];
    Set *new_set = Set_Init(FOLLOWING, user_id);
    generate_filename(new_set, filename);
    Load_AVL(new_set->Elem, filename);
    Traverse_AVL(new_set->Elem, INORDER, dispaly_visit);
}

/*
 * Common_friends - Show 2 users'common friends
 */
void Common_friends(int user_a, int user_b) {
    /* Check if user exists */
    if (!check_user(user_a) || !check_user(user_b)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename_a[25], filename_b[25];
    Set *set_a = Set_Init(FRIENDS, user_a);
    Set *set_b = Set_Init(FRIENDS, user_b);
    generate_filename(set_a, filename_a);
    generate_filename(set_b, filename_b);
    Load_AVL(set_a->Elem, filename_a);
    Load_AVL(set_b->Elem, filename_b);
    Set *common_set = Set_Intersection(set_a, set_b);
}

```

```

    Traverse_AVL(common_set->Elem, INORDER, dispaly_visit);
}

/*
 * Common_followers - Show 2 users'common followers
 */
void Common_followers(int user_a, int user_b) {
    /* Check if user exists */
    if (!check_user(user_a) || !check_user(user_b)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename_a[25], filename_b[25];
    Set *set_a = Set_Init(FOLLOWERS, user_a);
    Set *set_b = Set_Init(FOLLOWERS, user_b);
    generate_filename(set_a, filename_a);
    generate_filename(set_b, filename_b);
    Load_AVL(set_a->Elem, filename_a);
    Load_AVL(set_b->Elem, filename_b);
    Set *common_set = Set_Intersection(set_a, set_b);
    Traverse_AVL(common_set->Elem, INORDER, dispaly_visit);
}

/*
 * Common_followings - Show 2 users'common followings
 */
void Common_followings(int user_a, int user_b) {
    /* Check if user exists */
    if (!check_user(user_a) || !check_user(user_b)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename_a[25], filename_b[25];
    Set *set_a = Set_Init(FOLLOWING, user_a);
    Set *set_b = Set_Init(FOLLOWING, user_b);
    generate_filename(set_a, filename_a);
    generate_filename(set_b, filename_b);
    Load_AVL(set_a->Elem, filename_a);
    Load_AVL(set_b->Elem, filename_b);
}

```

```

    Set *common_set = Set_Intersection(set_a, set_b);
    Traverse_AVL(common_set->Elem, INORDER, dispaly_visit);
}

/* Assist function fow recommending */
void traverse_and_add(AVL_node *node, Set *recommend_set) {
    if (node == NULL)
        return;
    traverse_and_add(node->left_child, recommend_set);
    Set_Insert(recommend_set, node->key);
    traverse_and_add(node->right_child, recommend_set);
}

/* Assist function fow recommending */
void traverse_and_recommend(AVL_node *node, Set *recommend_set)
{
    if (node == NULL)
        return;
    traverse_and_recommend(node->left_child, recommend_set);
    char filename[25];
    Set *friend_set = Set_Init(FRIENDS, node->key);
    generate_filename(friend_set, filename);
    Load_AVL(friend_set->Elem, filename);
    traverse_and_add(friend_set->Elem->root, recommend_set);
    traverse_and_recommend(node->right_child, recommend_set);
}

/*
 * Recommend_friends - Recommend friends for a user
 */
void Recommend_friends(int user_id) {
    /* Check if user exists */
    if (!check_user(user_id)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename[25];
    Set *friend_set = Set_Init(FRIENDS, user_id);
    Set *recommend_set = Set_Init(FRIENDS, user_id);

```



```

    generate_filename(friend_set, filename);
    Load_AVL(friend_set->Elem, filename);
    traverse_and_recommend(friend_set->Elem->root,
recommend_set);
    Set_Delete(recommend_set, user_id);
    recommend_set = Set_Difference(recommend_set, friend_set);
    Traverse_AVL(recommend_set->Elem, INORDER, dispaly_visit);
}

/*
 * Recommend_followers - Recommend followers for a user
 */
void Recommend_followers(int user_id) {
    /* Check if user exists */
    if (!check_user(user_id)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
    char filename[25];
    Set *follower_set = Set_Init(FOLLOWERS, user_id);
    Set *recommend_set = Set_Init(FOLLOWERS, user_id);
    generate_filename(follower_set, filename);
    Load_AVL(follower_set->Elem, filename);
    traverse_and_recommend(follower_set->Elem->root,
recommend_set);
    Set_Delete(recommend_set, user_id);
    recommend_set = Set_Difference(recommend_set, follower_set);
    Traverse_AVL(recommend_set->Elem, INORDER, dispaly_visit);
}

/*
 * Recommend_followings - Recommend followings for a user
 */
void Recommend_followings(int user_id) {
    /* Check if user exists */
    if (!check_user(user_id)) {
        printf("[ERROR] Failed! This user is not exists!\n");
        return;
    }
}

```

```

    char filename[25];
    Set *following_set = Set_Init(FOLLOWING, user_id);
    Set *recommend_set = Set_Init(FOLLOWING, user_id);
    generate_filename(following_set, filename);
    Load_AVL(following_set->Elem, filename);
    traverse_and_recommend(following_set->Elem->root,
recommend_set);
    Set_Delete(recommend_set, user_id);
    recommend_set = Set_Difference(recommend_set, following_set);
    Traverse_AVL(recommend_set->Elem, INORDER, dispaly_visit);
}

/*
 * Change_name - Change a user's name
 */
void Change_name(int user_id, char *new_name) {
    HashTable *MyHash = Init_Hash(HASHLEN);
    Load_Hash(MyHash, database_path);
    if(Change_Hash(MyHash, user_id, new_name) == ERROR)
        printf("[ERROR] Change failed! Please check your input.\n");
    Save_Hash(MyHash, database_path);
}

```

II.3 main.c – main 函数

```

/*
 * Start file
 * Developed by Pan Yue
 * zxc479773533@gmail.com
 */
#include "src/prompt/py_prompt.c"

int main(int argc, char **argv) {
    /* Let's start! */
    Start_Shell(argc, argv);
    return 0;
}

```

由 CMake 生成的 Makefile 在此就略去了，在根目录下执行 cmake 命令即可根据 CMakeLists 生成相应的 Makefile 文件。

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] Data Structures and Algorithm Analysis in C. Second Edition. [美] Mark Allen Weiss.
- [4] Algorithms. Fourth Edition. [美] Robert Sedgewick, Kevin Wayne
- [5] Computer Systems: A Programmer's Perspective. Third Edition. [美] Bryant, R. E