# 北京邮电大学软件学院

# 2019-2020学年第1学期实验报告

**课程名称： 算法与数据结构**

**实验名称： 实验六**

**实验完成人：**

**姓名：**\_\_王衔飞\_\_\_**学号：**\_ \_\_**成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_\_贾红娓 \_李璐路\_\_\_\_\_\_\_\_

**日 期： 2019 年 12 月 9 日**

1. **实验目的**

本次实验旨在集中对几个专门的问题作较为深入的探讨和理解，不强调对某些特定的编程技术的训练。

1. **实验内容**

**必做内容**

1. **二叉排序树**

**[问题描述]**

　　从键盘读入一组数据，建立二叉排序树并对其进行查找、遍历、格式化打印等有关操作。

**[基本要求]**

　　建立二叉排序树并对其进行查找，包括成功和不成功两种情况，并给出查找长度。

**[测试数据]**

由学生依据软件工程的测试技术自己确定。注意测试边界数据。

1. **哈希表设计**

**[问题描述]**

　　针对某个集体中人名设计一个哈希表，使得平均查找长度不超过R，并完成相应的建表和查表程序。

**[基本要求]**

　　假设人名为中国人姓名的汉语拼音形式。待填入哈希表的人名共有30个，取平均查找长度的上限为2。哈希函数用除留余数法构造，用线性探测再散列法或链地址法处理冲突。

**[测试数据]**

　　取你周围较熟悉的30个人名。

**选做内容**

* + - * 1. 实现二叉排序树的插入、删除操作。
        2. 从教科书上介绍的集中哈希函数构造方法中选出适用者并设计几个不同的哈希函数，比较他们的地址冲突率（可以用更大的名字集合作实验）。
        3. 研究必做实验2）的30个人名的特点，努力找一个哈希函数，使得对于不同的拼音名一定不发生地址冲突。
        4. 在哈希函数确定的前提下尝试各种不同处理冲突的方法，考察平均查找长度的变化和造好的哈希表中关键字的聚集性。

1. **实验环境**

OS：macOS10.15 IDE：CLion2019.3 编译器：GCC9 语言标准：C++20

1. **实验结果**

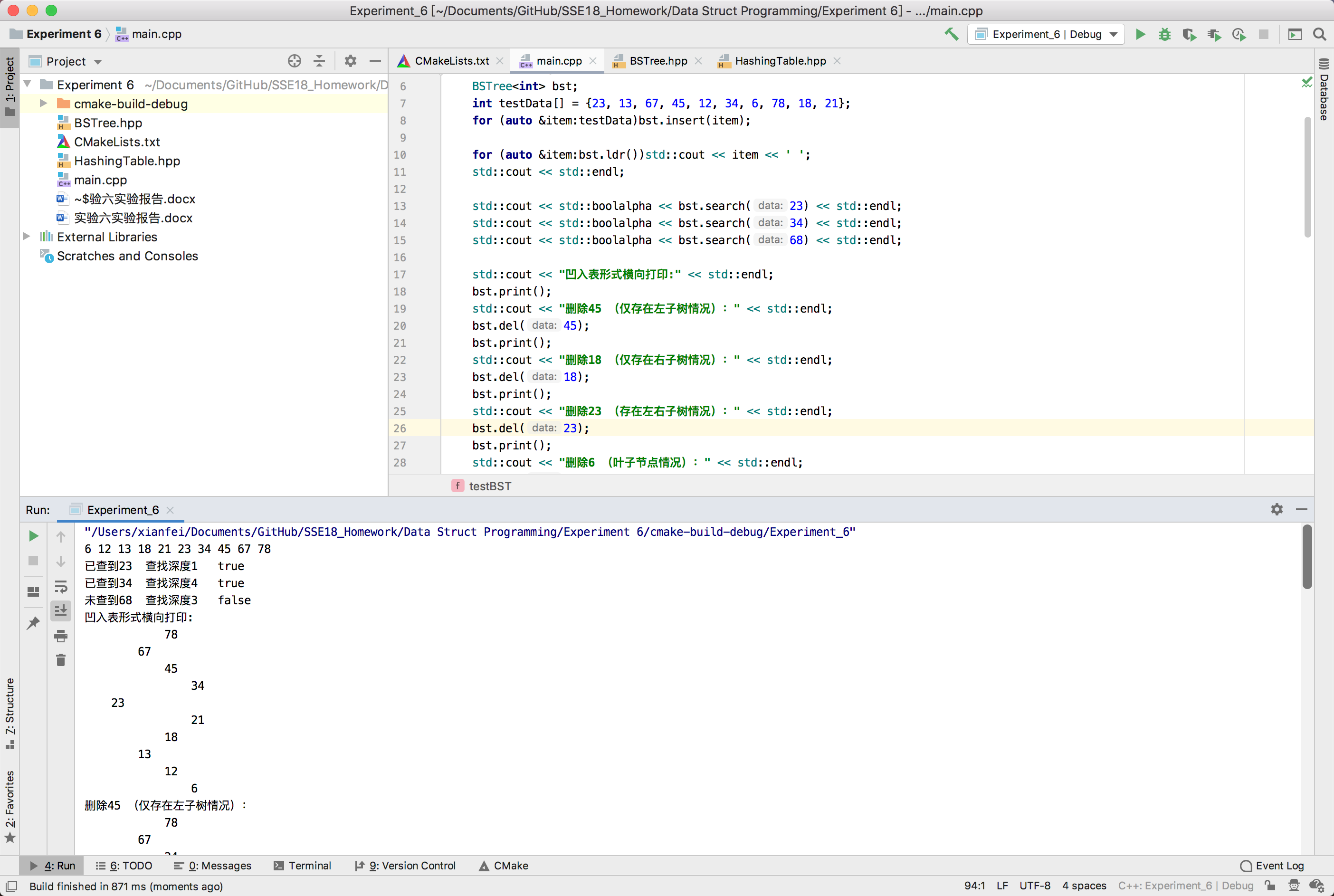
完成了实验要求所规定的二叉查找树的查找、创建、遍历、插入、删除、打印（同实验四中凹入表形式横向打印），并分别测试了各种情况下删除。对于哈希表，实现了创建、插入、查找功能，并测试了30组数据在不同空间下的平均查找成功率。测试数据及截图见附件。

1. **附录**

**二叉搜索树：**

测试数据：一组数字，23, 13, 67, 45, 12, 34, 6, 78, 18, 21

测试方法：首先ldr遍历这颗树，从小到大输出数据，然后测试搜索23 34 68这几个数据，并输出深度。接着测试凹入表形式横向打印，最后测试分别删除仅存在左/右子树节点、存在左右子树节点、叶子节点几种情况，测试代码及运行截图：



完整输出数据：

6 12 13 18 21 23 34 45 67 78

已查到23 查找深度1 true

已查到34 查找深度4 true

未查到68 查找深度3 false

凹入表形式横向打印:

78

67

45

34

23

21

18

13

12

6

删除45 （仅存在左子树情况）：

78

67

34

23

21

18

13

12

6

删除18 （仅存在右子树情况）：

78

67

34

23

21

13

12

6

删除23 （存在左右子树情况）：

78

67

34

21

13

12

6

删除6 （叶子节点情况）：

78

67

34

21

13

12

特殊情况测试：

删除仅剩的一个节点：



异常测试：删除找不到的节点



心得：二叉搜索树在ldr遍历得到的是从小到大序列，rdl遍历得到的结果是从大到小的序列。存储方式与树的实验相同，才有左右子树的指针+key值进行存储。

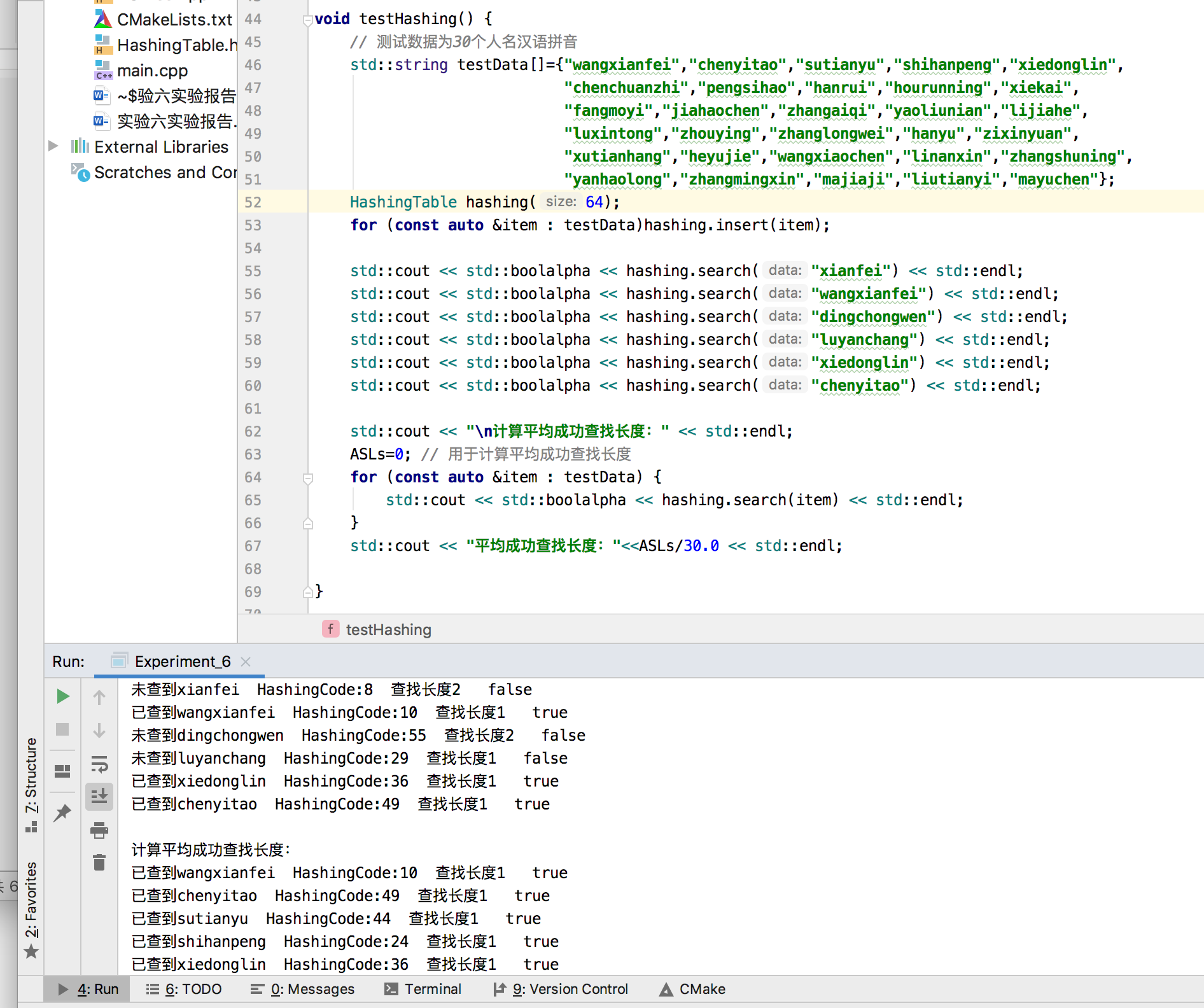
**哈希表**

算法：求哈希值采用把整个字符串中每一个字母的ascii码加起来取模小于表大小最大的质数，采用链地址法处理冲突。

测试数据：30个人名的汉语拼音，"wangxianfei","chenyitao","sutianyu","shihanpeng","xiedonglin", "chenchuanzhi","pengsihao","hanrui","hourunning","xiekai", "fangmoyi","jiahaochen","zhangaiqi","yaoliunian","lijiahe", "luxintong","zhouying","zhanglongwei","hanyu","zixinyuan", "xutianhang","heyujie","wangxiaochen","linanxin","zhangshuning", "yanhaolong","zhangmingxin","majiaji","liutianyi","mayuchen"

测试方法：查找了六个数据，三个在表中，三个不在表中，均能正确找到并打印搜索深度。接着查找表中所有数据，计算平均成功查找长度，该组测试数据的平均成功查找长度为1.1。

代码及截图如下：



完整输出数据：

未查到xianfei HashingCode:8 查找长度2 false

已查到wangxianfei HashingCode:10 查找长度1 true

未查到dingchongwen HashingCode:55 查找长度2 false

未查到luyanchang HashingCode:29 查找长度1 false

已查到xiedonglin HashingCode:36 查找长度1 true

已查到chenyitao HashingCode:49 查找长度1 true

计算平均成功查找长度：

已查到wangxianfei HashingCode:10 查找长度1 true

已查到chenyitao HashingCode:49 查找长度1 true

已查到sutianyu HashingCode:44 查找长度1 true

已查到shihanpeng HashingCode:24 查找长度1 true

已查到xiedonglin HashingCode:36 查找长度1 true

已查到chenchuanzhi HashingCode:52 查找长度1 true

已查到pengsihao HashingCode:43 查找长度1 true

已查到hanrui HashingCode:37 查找长度1 true

已查到hourunning HashingCode:3 查找长度1 true

已查到xiekai HashingCode:25 查找长度1 true

已查到fangmoyi HashingCode:4 查找长度1 true

已查到jiahaochen HashingCode:58 查找长度1 true

已查到zhangaiqi HashingCode:41 查找长度1 true

已查到yaoliunian HashingCode:44 查找长度2 true

已查到lijiahe HashingCode:55 查找长度1 true

已查到luxintong HashingCode:24 查找长度2 true

已查到zhouying HashingCode:39 查找长度1 true

已查到zhanglongwei HashingCode:12 查找长度1 true

已查到hanyu HashingCode:0 查找长度1 true

已查到zixinyuan HashingCode:31 查找长度1 true

已查到xutianhang HashingCode:42 查找长度1 true

已查到heyujie HashingCode:23 查找长度1 true

已查到wangxiaochen HashingCode:56 查找长度1 true

已查到linanxin HashingCode:11 查找长度1 true

已查到zhangshuning HashingCode:19 查找长度1 true

已查到yanhaolong HashingCode:35 查找长度1 true

已查到zhangmingxin HashingCode:17 查找长度1 true

已查到majiaji HashingCode:54 查找长度1 true

已查到liutianyi HashingCode:8 查找长度1 true

已查到mayuchen HashingCode:4 查找长度2 true

平均成功查找长度：1.1

Process finished with exit code 0

心得：为了降低平均成功查找长度，应该使哈希值尽可能分散的分布在整个哈希表中，而适当的降低装填因子可以提高哈希表的查找效率，用空间换时间。