

**2019** 级

《大数据数据存储与管理》课程

**课 程 报 告**

**姓 名 杨超淇**

**学 号 U201914974**

**班 号 计算机1903班**

**日 期 2022.04.18**

**目 录**

[一、报告选题和目的 1](#_Toc101210943)

[二、选题分析 1](#_Toc101210944)

[2.1 选题背景 1](#_Toc101210945)

[2.2 原理分析 1](#_Toc101210946)

[三、实验环境 3](#_Toc101210947)

[四、实验设计 3](#_Toc101210948)

[4.1 hash函数设计 3](#_Toc101210949)

[4.2 bloom filter设计 3](#_Toc101210950)

[五、性能测试 5](#_Toc101210951)

[5.1 数据集与内存占用 5](#_Toc101210952)

[5.2 错误率分析 5](#_Toc101210953)

[5.3 查询延迟对比 6](#_Toc101210954)

[六、课程总结 7](#_Toc101210955)

[参考文献 8](#_Toc101210956)

# 一、报告选题和目的

1. 课程报告选题为基于Bloom Filter设计的大数据存储查询系统。
2. 了解bloom filter的应用背景，掌握其基本原理。
3. 尝试基于bloom filter的存储结构开发大数据存储系统。
4. 通过实验分析bloom filter的优势以及缺陷。

# 二、选题分析

## 2.1 选题背景

Bloom Filter是1970年由布隆提出的。它实际上是一个很长的二进制向量和一系列随机映射函数，可以用于检索一个元素是否在一个集合中。

Bloom Filter是一种空间效率很高的随机数据结构，它利用位数组很简洁地表示一个集合，并能判断一个元素是否属于这个集合。Bloom Filter的这种高效是有一定代价的：在判断一个元素是否属于某个集合时，有可能会把不属于这个集合的元素误认为属于这个集合（false positive）。因此，Bloom Filter不适合那些 “零错误”的应用场合。而在能容忍低错误率的应用场合下，Bloom Filter通过极少的错误换取了存储空间的极大节省。

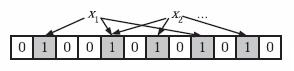
如需要判断一个元素是不是在一个集合中，我们通常做法是把所有元素保存下来，然后通过比较知道它是不是在集合内，链表、树都是基于这种思路，当集合内元素个数的变大，我们需要的空间和时间都线性变大，检索速度也越来越慢。 Bloom filter 采用的是hash函数的方法，将一个元素映射到一个m长度的阵列上的一个点，当这个点是1时，那么这个元素在集合内，反之则不在集合内。这个方法的缺点就是当检测的元素很多的时候可能有冲突，解决方法就是使用k个hash函数对应k个点，如果所有点都是1的话，那么元素在集合内，如果有0的话，元素则不在集合内。

## 2.2 原理分析

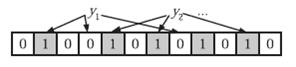
Bloom Filter使用一串位数组存储数据，并只支持对数据进行查询，无法读取。初始状态时，Bloom Filter是一个包含m位的位数组，每一位都置为0。



为了表达S={x1, x2,…,xn}这样一个n个元素的集合，Bloom Filter使用k个相互独立的hash函数，它们分别将集合中的每个元素映射到{1,…,m}的范围中。对任意一个元素x，第i个hash函数映射的位置hi(x)就会被置为1（1≤i≤k）。注意，如果一个位置多次被置为1，那么只有第一次会起作用，后面几次将没有任何效果。在下图中，k=3，且有两个hash函数选中同一个位置（从左边数第五位）。



在判断y是否属于这个集合时，我们对y应用k次hash函数，如果所有hi(y)的位置都是1（1≤i≤k），那么我们就认为y是集合中的元素，否则就认为y不是集合中的元素。下图中y1就不是集合中的元素。y2或者属于这个集合，或者刚好是一个false positive。



**错误率估计：**

前面已经提到，Bloom Filter在判断一个元素是否属于它表示的集合时会有一定的错误率，下面就来估计错误率的大小。在估计之前为了简化模型，我们假设kn<m且各个hash函数是完全随机的。当集合S={x1, x2,…,xn}的所有元素都被k个hash函数映射到m位的位数组中时，这个位数组中某一位还是0的概率是：

其中1/m表示任意一个hash函数选中这一位的概率，(1-1/m)表示hash一次没有选中这一位的概率。要把S完全映射到位数组中，需要做kn次hash。某一位还是0意味着kn次hash都没有选中它，因此这个概率就是（1-1/m）的kn次方。为了简化运算，可以使用计算e时常用的近似：

令ρ为位数组中0的比例，则ρ的数学期望E(ρ)= p。在ρ已知的情况下，错误率（即不属于集合的数据的hash选中位均为1的概率）为：

(1-ρ)为位数组中1的比例，(1-ρ)k就表示k次hash都刚好选中1的区域，即错误率。p只是ρ的数学期望，在实际中ρ的值有可能偏离它的数学期望值。位数组中0的比例非常集中地分布在它的数学期望值的附近。因此，第一步的近似得以成立。将p代入上式中，得：

上式即可用于估算Bloom Filter的错误率。

# 三、实验环境

操作系统：Windows 10 家庭中文版

处理器：Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz

内存：16GB

开发平台：IntelliJ IDEA 2020.3.2 x64

开发工具：java jdk-15.0.2

# 四、实验设计

本次实验目的为开发基于bloom filter的数据存储、查询系统，实验程序使用JAVA语言实现。实验程序分为3个类：

bloomfilter类实现对字符串数据的存储和查询功能；

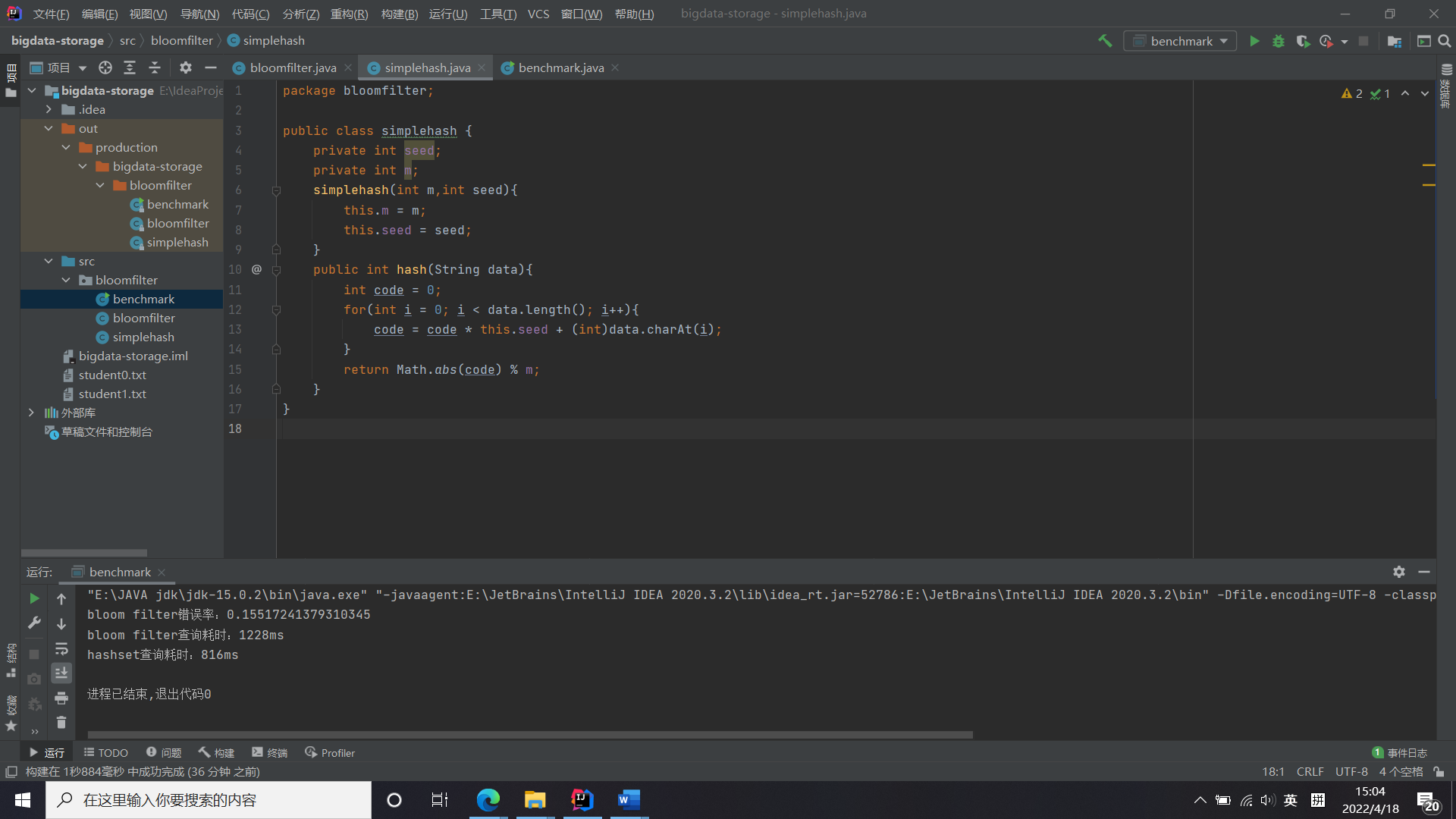
simplehash类用于根据随机种子生成不同的hash函数对象；

benchmark类用于对bloomfilter与JAVA自带的hashset进行性能测试和对比。

## 4.1 hash函数设计

由于用于实现bloom filter的hash函数数量k不确定，且每个hash函数生成的hash码要尽可能不同，因此使用simplehash类统一生成hash函数对象。同时在构造simplehash对象时，传入一个随机数（0~100）seed用于区分不同的hash函数。

simplehash类中统一实现的hash函数如下图所示。其中m为构造对象时，传入的位数组长度参数。hash函数对传入的字符串数据逐一获取字符的Unicode编码，并与随机种子seed进行循环运算，得到hash编码。由于hash编码的值可能为负数或超出位数组的范围，最后需要求绝对值并取余，将hash码约束在0~m的范围内。



## 4.2 bloom filter设计

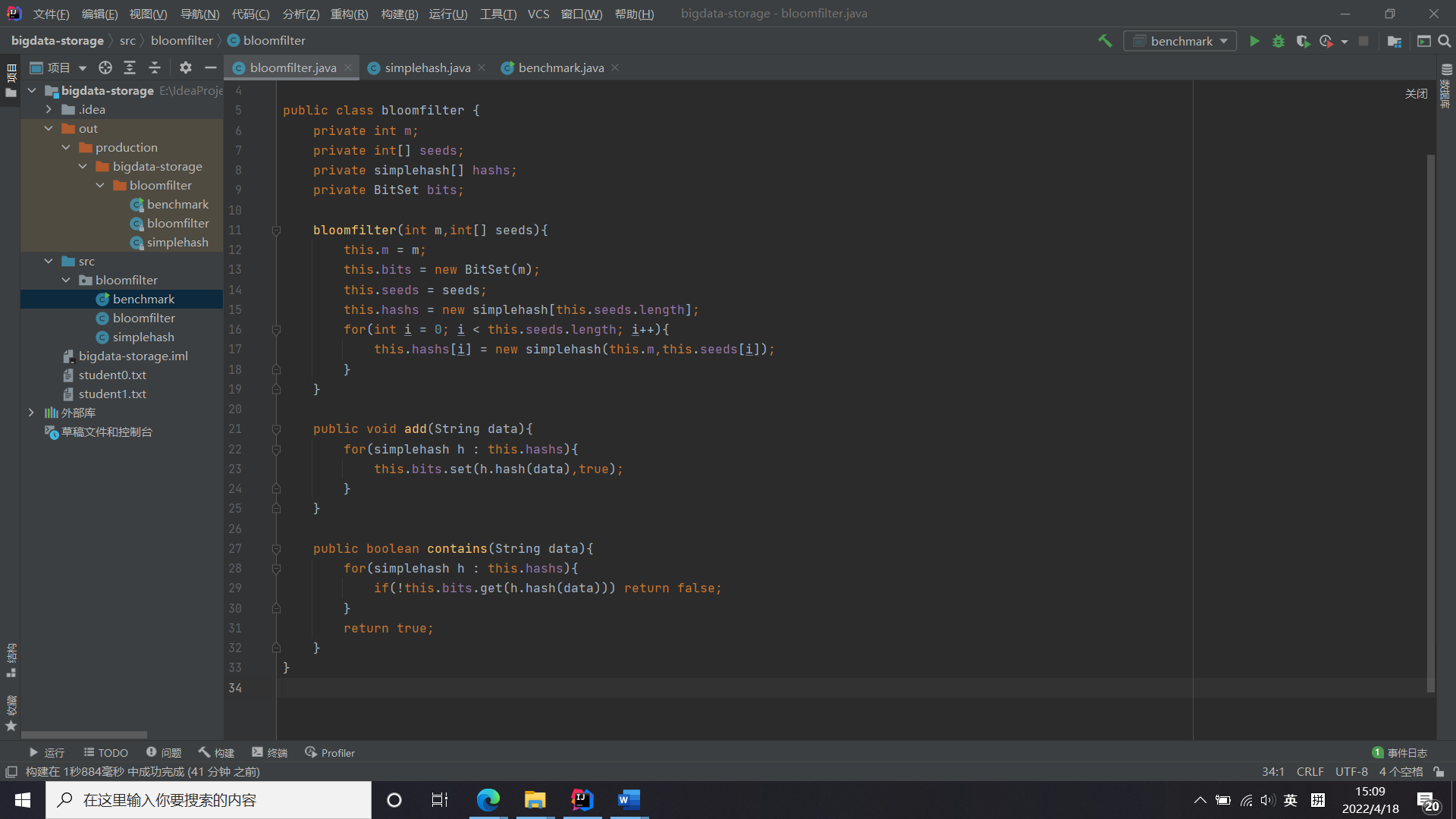
bloom filter类为基于Bloom Filter的技术设计的存储、查询系统，其具体实现的数据结构、构造函数、数据存储方法、数据查询方法如下图所示。

数据结构：由一个长度为m的位数组（通过实例化BitSet类得到）和k个hash函数（通过实例化simplehash类得到）组成。为了实现bloom filter的通用性，位数组的长度m和hash函数数量k不能够为常量，而是在构造bloom filter对象时作为参数传入。由于每个hash函数的种子不同，且随机种子应该由调用bloom filter类的使用者决定，所以参数k实际上是由随机数组成的数组seeds。

构造函数：在构造bloom filter对象时，根据传入的m和seeds，修改对应的数据成员。然后，根据位数组长度m构建BitSet对象，使用成员bits引用；根据随机数组seeds和hash码范围m构建k个不同的simplehash对象组成数组，使用成员hashs引用。

add方法：该方法实现bloom filter的存储功能。获取需要存储的字符串对象后，在k次循环中，使用hashs数组内不同的simplehash对象的hash方法解析字符串，得到hash码作为索引值，使用Bitset对象的set方法将数据成员bits中的对应位置为true。

contains方法：该方法实现bloom filter的查询功能。获取需要查询的字符串对象后，在k次循环中，使用hashs数组内不同的simplehash对象的hash方法解析字符串，得到hash码作为索引值，使用Bitset对象的get方法获取数据成员bits中对应位置的值，若该值不为true，则查询失败，返回false。当循环结束时，则查询成功，返回true。



# 五、性能测试

## 5.1 数据集与内存占用

实现simplehash类以及bloom filter类的设计后，数据存储查询系统基本上设计完成。在benchmark类中设置程序入口main，对该数据存储系统进行性能测试，并将其与java自带的hashset类进行性能对比。

测试使用数据集为计算机学院2019级新生名单（共329个学生名字），以及用于测试错误率的校交班学生名单（共60个学生名字），两个数据集交集为空集（除去重名），分别保存在student0.txt和student1.txt。

benchmark测试程序中使用BufferReader对象br0、br1读取数据集文件，获得字符串对象。构造bloom filter对象时，设置k为8，通过Random.nextInt(100)获取100以内的随机数作为simplehash的随机种子。

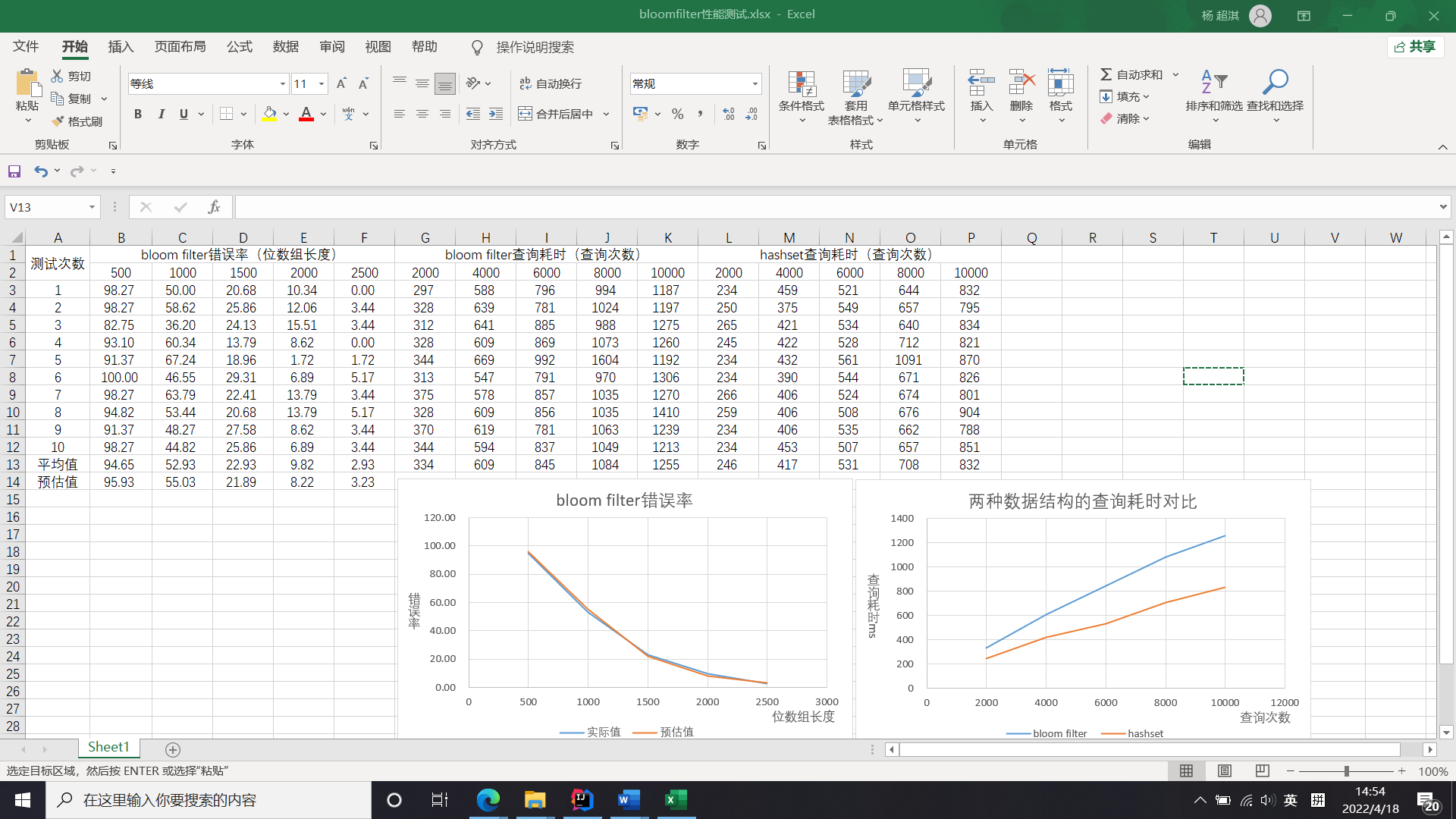
内存占用：使用对应的add方法将br0读取的数据存入bloom filter和hashset。bloom filter在k=8、n=329的情况下，要求错误率达到0.01，需要的位数组长度在3200左右，即占用400字节的内存空间。hashset类则存储完整的字符串数据，java字符集使用unicode编码，则存储927个字符至少需要占用1854字节的内存。

## 5.2 错误率分析

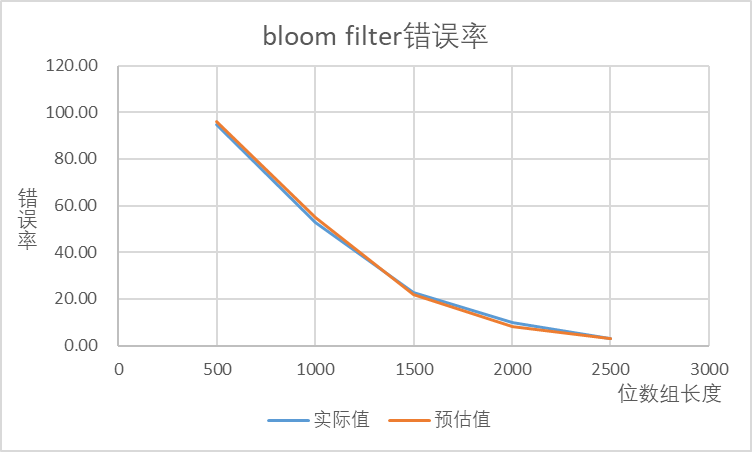
使用BufferReader对象br1读取student2数据集，该数据集与已存储的student1没有交集（可能有重名），可以用来测试bloom filter的错误率。

hashset的错误率为0，可以用来去除重名。在读取数据的循环中，通过contains方法判断出数据不在hashset内时，total++，再判断数据在bloom filter中时，wrong++得到bloom filter错误率为wrong/total。

为了数据的准确性，以及分析bloom filter的错误率与位数组长度的关系，将m在500~2500之间变化，并测试10次取平均值。



根据上述表格画出bloom filter错误率随位数组长度变化的折线图。由折线图可以看出，性能测试的错误率变化基本与预测值吻合。



## 5.3 查询延迟对比

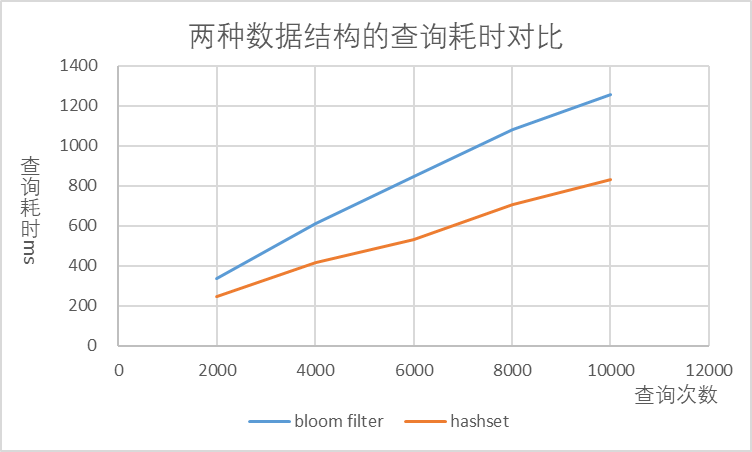
使用student0数据集进行数据查询，由于student0数据集已经存储在bloom filter中，因此查询耗时为最大值，即需要经过所有hash函数的检验。同时也对hashset进行数据查询，比较二者的查询性能。

计算查询耗时为在查询开始与结束时，使用System.currentTimeMillis()方法获取系统当前时间，记为starttime和endtime，通过两者相减得到数据查询循环消耗的时间（毫秒级）。

由于单次查询的时间极小，且为了求出耗时变化趋势，查询次数在2000~10000之间变化，每次查询整个数据集，重复测试10次求平均值。



下图为根据上述数据画出的bloom filter和hashset的查询耗时随查询次数的变化，以及两者查询耗时的对比。可以看出，由于bloom filter没有进行优化，且需要经过多个hash函数计算才能得到查询结果，其查询耗时性能不如Java自带的hashset存储结构。



# 六、实验总结

通过性能测试与对比，分析可得Bloom Filter与通常的hash表相比，主要优点在于其所占用的内存空间非常小，代价是由于数据并非真的被存储在Bloom Filter内，存在一定的错误率，也无法读取数据。在查询性能方面，随着数据的增多，通常的存储结构会越来越复杂，导致查询耗时增加，而Bloom Filter本身结构简单，在大数据情况下查询效率更高。

总的来说，Bloom Filter有着广阔的应用前景，例如网页URL的去重，垃圾邮件的判别，集合重复元素的判别，查询加速，数据库防止查询击穿等，借助Bloom Filter可以大幅度节省空间资源、提高效率。

# 参考文献

• F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, “Beyond Bloom Filters: From Approximate Membership Checks to Approximate State Machines,” Proc. ACM SIGCOMM, 2006.

• Y. Zhu and H. Jiang, “False Rate Analysis of Bloom Filter Replicas in Distributed Systems,” Proc. Int’l Conf. Parallel Processing (ICPP ’06), pp. 255-262, 2006.

• S. Dharmapurikar, P. Krishnamurthy, and D.E. Taylor, “Longest Prefix Matching Using Bloom Filters,” Proc. ACM SIGCOMM, pp. 201-212, 2003.

• L. Fan, P. Cao, J. Almeida, and A. Broder, “Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol,” IEEE/ACM Trans. Networking, vol. 8, no. 3, pp. 281-293, June 2000.

• B. Xiao and Y. Hua, “Using Parallel Bloom Filters for Multi- Attribute Representation on Network Services,” IEEE Trans. Parallel and Distributed Systems, vol. 21, no. 1, pp. 20-32, Jan. 2010.

• Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, “Scalable and Adaptive Metadata Management in Ultra Large-scale File Systems,” Proc. 28th Int’l Conf. Distributed Computing Systems (ICDCS ’08), pp. 403-410, 2008.

• D. Guo, J. Wu, H. Chen, and X. Luo, “Theory and Network Application of Dynamic Bloom Filters,” Proc. IEEE INFOCOM, 2006.