专注于IT职业提升，为工程师提供优质完善的成长体系。

缩短工程师的学习时间，

增强工程师的学习效果，提升工程师的资薪待遇。

为工程师的技术提升穿针引线，为工程师的职业成长搭桥铺路。

办学宗旨：**一切只为渴望更优秀的你**。

讲师介绍

King老师 | 零声学院联合创始人 曾供职于微软亚洲研究院

曾供职于微软亚洲研究院，创维集团深圳研究院。后任知名创业公司系统架构师，负责数据传输协议设计和集群架构设计。曾获得第二届全国物联网大赛特等奖。微软“Image Cup”软件设计二等奖，第一代国内Paas云平台开发者。著有多个软件专利，在全球化，高可用的物联网云平台架构与智能硬件设计方面有丰富的产品实战经验。

Darren老师 | 零声学院联合创始人 曾供职于深圳联发科

曾供职于国内知名半导体公司（珠海扬智/深圳联发科），曾在某互联网公司担任音视频通话项目经理。主要从事音视频驱动、多媒体中间件、流媒体服务器的开发，开发过即时通讯+音视频通话的大型项目，在音视频、C/C++/GO Linux服务器领域有丰富的实战经验。

内容和服务为王

零声学院

www.0voice.com

一切只为渴望更优秀的你！

讲师介绍

零声学院 Darren老师 中文名：廖庆富

高级资深工程师

QQ：326873713

- 毕业于广东工业大学，硕士学历
- 曾供职于国内知名半导体公司（珠海扬智/深圳联发科），曾在某互联网公司担任音视频通话项目经理。
- 主要从事音视频驱动、多媒体中间件、流媒体服务器的开发，开发过即时通讯+音视频通话的大型项目，在音视频、C/C++/GO Linux服务器领域有丰富的实战经验。

课题：布隆过滤器

- 上亿字符串查重
- 为什么不用HashMap
- 布隆过滤器原理
- 布隆过滤器设计与实现

什么是布隆过滤器

布隆过滤器是一种数据结构，比较巧妙的概率型数据结构（probabilistic data structure），特点是高效地插入和查询，可以用来告诉你 “**某样东西一定不存在或者可能存在**”。

相比于传统的 List、Set、Map 等数据结构，它更高效、占用空间更少，但是缺点是其返回的结果是概率性（存在误差）的，而不是确切的

注意：不同的数据结构有不同的适用场景和优缺点，你需要仔细权衡自己的需求之后妥善适用它们，布隆过滤器就是践行这句话的代表。

这些复杂问题怎么办？

使用word文档时，判断某个单词是否拼写正确？

网络爬虫程序，不去爬相同的url页面？

垃圾邮件过滤算法如何设计？

缓存崩溃后造成的缓存击穿？

FBI，一个嫌疑人的名字是否已经在嫌疑名单上？

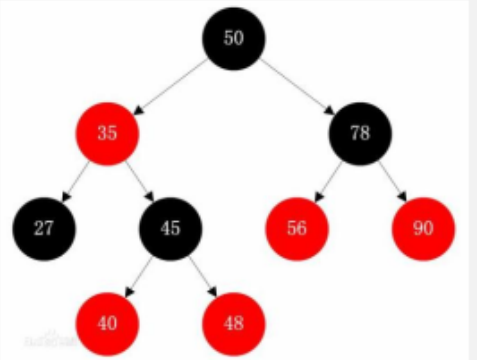
用在允许误差的场景

判定一个字符串是否存在

还有别的应用场景吗

这样行不行？

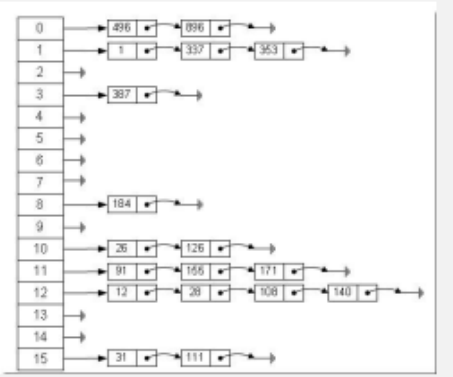
map



$O(log_2 n)$

100W个记录 = 20

hashMap



$O(1)$ + hash时间

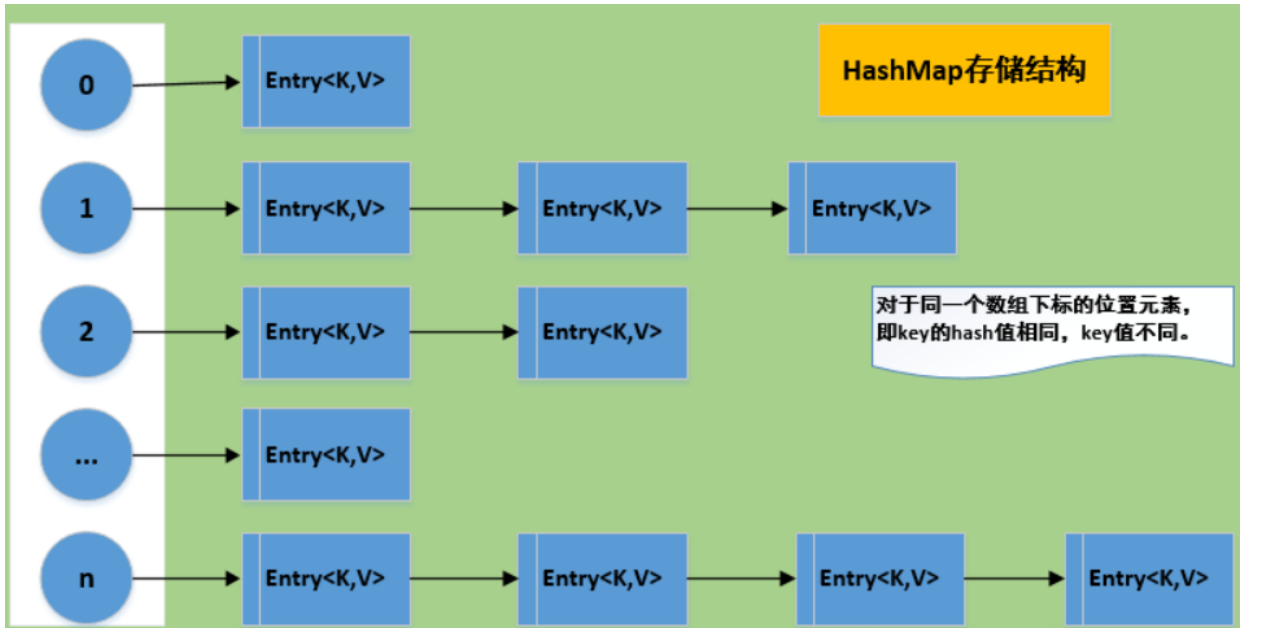
100W个记录 = 1

10亿条记录时怎么办？

为什么不用HashMap

为什么不用HashMap?

值映射到 HashMap 的 Key，然后可以在 O(1) 的时间复杂度内返回结果，效率也非常高。



HashMap的问题

hashmap的问题：

- 存储容量占比高，考虑到负载因子的存在，通常空间是不能被用满；
- 存储了key，类似URL则非常占空间

测试结果：

```
unordered_map<string, bool> unordermp;
main_key = "https://blog.csdn.net/muyuyuzhong/article/details/";
sub_key = to_string(i);
key = main_key + sub_key;
```

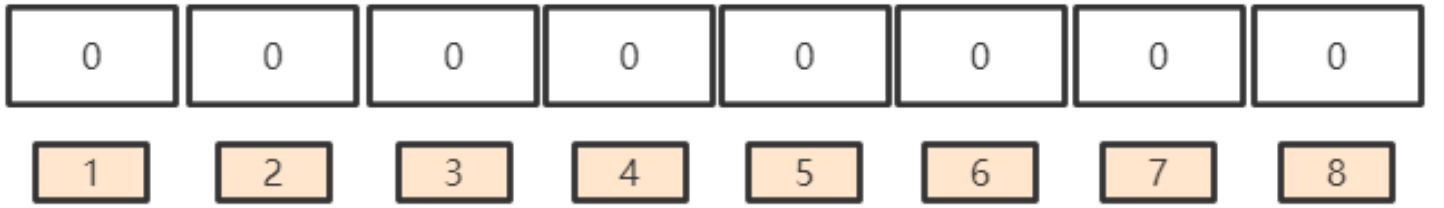
容量	key size(byte)	占用内存(MB)	插入耗时(ms)
10 0000	53	16.7	111
100 0000	53	158.0	1165
1000 0000	54	1599.5	13318
1 0000 0000		?	

1亿条URL?

1000万， 误差万分之一 布隆过滤器 22.852562MB

布隆过滤器数据结构

布隆过滤器是一个 bit 向量或者说 bit 数组（长度到底到底多长），如下所示：



这里是耗内存，增加key的反而不会耗内存（因为它不存储KEY）

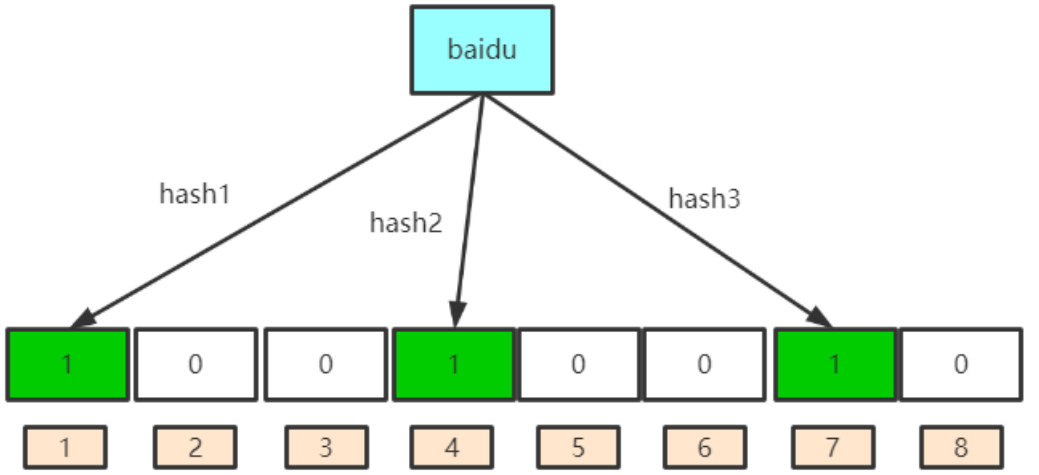
布隆过滤器的原理是，当一个元素被加入集合时，通过K个Hash函数将这个元素映射成一个位数组中的K个点，把它们置为1。
检索时，我们只要看看这些点是不是都是1就（大约）知道集合中有没有它了：

- 如果这些点有任何一个0，则被检元素一定不在；
- 如果都是1，则被检元素很可能（我们期望存在的概率是多少可以设置）存在。

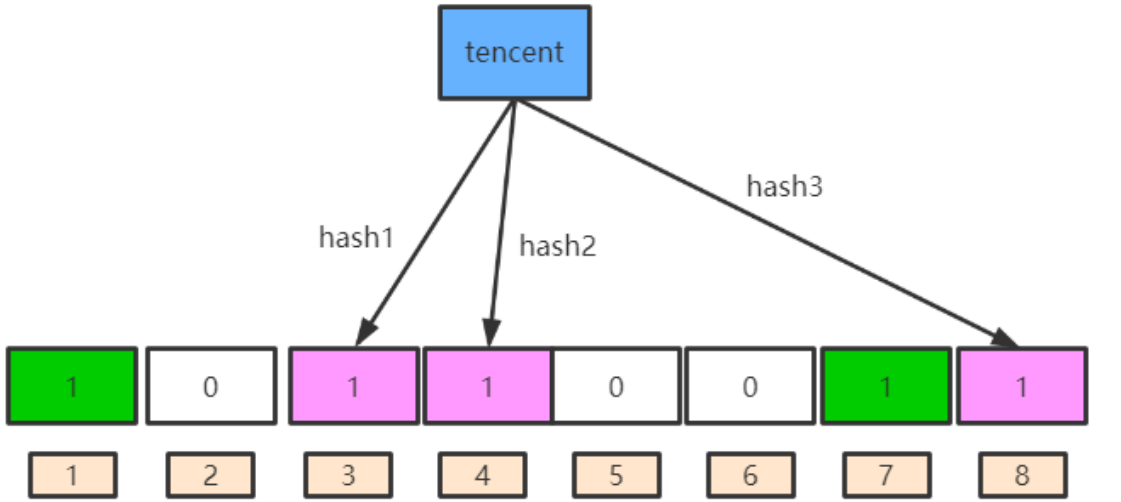
这就是布隆过滤器的基本思想。

布隆过滤器原理

(1) 例如针对值 “baidu” 和三个不同的哈希函数分别生成了哈希值 1、4、7，则上图转变为：



(2) 再存一个值 “tencent”，如果哈希函数返回 3、4、8 的话，图继续变为：



4 这个 bit 位由于两个哈希函数都返回了这个 bit 位，因此它被覆盖了。现在我们如果想查询 “0voice” 这个值是否存在，哈希函数返回了 1、5、8三个值，结果我们发现 5 这个 bit 位上的值为 0，**说明没有任何一个值映射到这个 bit 位上**，因此我们可以很确定地说 “0voice” 这个值不存在。而当我们需要查询 “baidu” 这个值是否存在的话，那么哈希函数必然会返回 1、4、7，然后我们检查发现这三个 bit 位上的值均为 1，那么我们可以说 “baidu” **存在了么？答案是不可以，只能是 “baidu” 这个值可能存在。**

这是为什么呢？答案更简单，因为随着增加的值越来越多，被置为 1 的 bit 位也会越来越多，这样某个值 “taobao” 即使没有被存储过，但是万一哈希函数返回的三个 bit 位都被其他值置位了 1，那么程序还是会判断 “taobao” 这个值存在。 **-> 误判率（假阳）**

为什么不直接使用hash table?

哈希表的存储效率一般只有 50%（为了避免高碰撞，一般哈希存到一半时都翻倍或采取其他策略），所以很费内存;

Hash面临的问题就是冲突。假设 Hash 函数是良好的，如果我们的位阵列长度为 m 个点，那么如果我们想将冲突率降低到例如 1%, 这个散列表就只能容纳 m/100 个元素。

解决方法较简单，使用k>1的布隆过滤器，即k个函数将每个元素改为对应于k个bits，因为误判度会降低很多，并且如果参数k和m选取得好，一半的m可被置为为1

误判概率的证明和计算1

假设布隆过滤器中的hash函数满足simple uniform hashing(简单一致散列)假设：每个元素都等概率地hash到**m**个slot中的任何一个，与其它元素被hash到哪个slot无关。若**m**为bit数，则对某一特定bit位在一个元素由某特定hash函数插入时**没有被置位为1的概率**为：

$1-\frac{1}{m}$ m 就是就我们刚才讲的向量表的长度

则**k**个hash函数中**没有一个对其置位**的概率为：

$\left(1-\frac{1}{m}\right)^k$ (1-1/100) = 0.99, 0.9801, 0.9801

如果插入了**n**个元素，但都**没有将其置位**的概率为：

$\left(1-\frac{1}{m}\right)^{kn}$

现在考虑查询阶段，若对应某个待查询元素的k bits全部置位为1，则可判定其在集合中。因此将某元素误判的概率**p**为：

$\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^k$ 如果继续推导？

误判概率的证明和计算2–(续)

现在考虑查询阶段，若对应某个待query元素的k bits全部置位为1，则可判定其在集合中。
因此将某元素误判的概率**p**为：

$$\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k}$$

此时n p m k 的定义都出来了。

由于 $(1+x)^{\frac{1}{x}} \sim e$,当 $x \to 0$ 时 ,并且 $-\frac{1}{m}$ 当m很大时趋近于0，所以

$$\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k}=\left(1-\left(1-\frac{1}{m}\right)^{-m\cdot\frac{-kn}{m}}\right)^{k}\sim\left(1-e^{-\frac{nk}{m}}\right)^{k}$$

从上式中可以看出，当m增大或n减小时，都会使得误判率减小

误判概率的证明和计算3–(续)

现在计算对于给定的m和n，k为何值时可以使得误判率最低。设误判率为k的函数为：

$$f(k) = \left(1 - e^{-\frac{nk}{m}}\right)^k$$

设 $b = e^{\frac{n}{m}}$，则简化为

$f(k) = (1 - b^{-k})^k$，两边取对数

$lnf(k) = k \cdot ln(1 - b^{-k})$，两边对k求导

$$\frac{1}{f(k)} \cdot f'(k) = ln(1 - b^{-k}) + k \cdot \frac{1}{1 - b^{-k}} \cdot (-b^{-k}) \cdot lnb \cdot (-1)$$

$$= ln(1 - b^{-k}) + k \cdot \frac{b^{-k} \cdot lnb}{1 - b^{-k}}$$

下面求最值，即是误差趋近于0

$$ln(1 - b^{-k}) + k \cdot \frac{b^{-k} \cdot lnb}{1 - b^{-k}} = 0$$

$$\Rightarrow (1 - b^{-k}) \cdot ln(1 - b^{-k}) = -k \cdot b^{-k} \cdot lnb$$

$$\Rightarrow (1 - b^{-k}) \cdot ln(1 - b^{-k}) = b^{-k} \cdot lnb^{-k}$$

$$\Rightarrow 1 - b^{-k} = b^{-k}$$

$$\Rightarrow b^{-k} = \frac{1}{2}$$

$$\Rightarrow e^{-\frac{kn}{m}} = \frac{1}{2}$$

$$\Rightarrow \frac{kn}{m} = ln2$$

$$\Rightarrow k = ln2 \cdot \frac{m}{n} = 0.7 \cdot \frac{m}{n}$$

因此，即当 $k = 0.7 \cdot \frac{m}{n}$ 时误判率最低，此时误判率为：

$$P(error) = \left(1 - \frac{1}{2}\right)^k = 2^{-k} = 2^{-ln2 \cdot \frac{m}{n}} \approx 0.6185^{\frac{m}{n}}$$

$$\left(1 - e^{-\frac{nk}{m}}\right)^k$$

可以看出若要使得误判率≤1/2，则：

$$k \geq 1 \Rightarrow \frac{m}{n} \geq \frac{1}{ln2}$$

这说明了若想保持某固定误判率不变，布隆过滤器的bit数**m**与被增加的元素数**n**应该是线性同步增加的。

更准确的值：ln2 = 0.69314718055995

设计和应用布隆过滤器的方法

应用时首先要先由用户决定要增加的最多**元素个数n**和希望的**误差率P**。这也是一个设计完整的布隆过滤器需要用户输入的仅有的两个参数（加入hash种子则为3个），之后的所有参数将由系统计算，并由此建立布隆过滤器。

(1) 首先根据传入的**n和p**计算需要的内存大小**m** bits:

$$P = 2^{-ln2\cdot\frac{m}{n}} \Longrightarrow \ln p = \ln 2 \cdot (-\ln 2)\cdot\frac{m}{n} \Longrightarrow m = -\frac{n\cdot lnp}{(ln2)^2}$$

先计算**m**，bit向量的长度

(2) 再由m，n得到hash function的个数：

$$k = ln2\cdot\frac{m}{n} = 0.7\cdot\frac{m}{n}$$

更准确的值：ln2 = 0.69314718055995

再计算**k**，哈希函数的 个数

至此系统所需的参数已经备齐，接下来增加**n**个元素至布隆过滤器中，再进行查询。

布隆过滤器空间利用率问题

根据公式，当k最优时：

$$P(error) = 2^{-k} \Rightarrow \log_2 P = -k \Rightarrow \mathrm{k} = \log_2 \frac{1}{P} \Rightarrow \ln 2 \frac{\mathrm{m}}{\mathrm{n}} = \log_2 \frac{1}{P}$$

$$\Rightarrow \frac{\mathrm{m}}{\mathrm{n}} = ln2 \cdot \log_2 \frac{1}{P} = 1.44 \cdot \log_2 \frac{1}{P}$$

因此可验证当P=1%时，存储每个元素需要9.6 bits：

$$\frac{\mathrm{m}}{\mathrm{n}} = 1.44 \cdot \log_2 \frac{1}{0.01} = 9.6\ bits$$

而每当想将误判率降低为原来的1/10，则存储每个元素需要增加4.8 bits：

$$\frac{\mathrm{m}}{\mathrm{n}} = 1.44 \cdot (\log_2 10a - \log_2 a) = 1.44 \cdot \log_2 10 = 4.8\ bits$$

布隆过滤器误判率对照表

如果方便知道需要使用多少位才能降低错误概率，可以从下表所示的存储项目和位数比率估计布隆过滤器的误判率。

表1 布隆过滤器误判率表

比率(items:bits) （n:m）	误判率(False-positive)
1：1	0.63212055882856
1：2	0.39957640089373
1：4	0.14689159766038
1：8	0.02157714146322
1：16	0.00046557303372
1：32	0.00000021167340
1：64	0.00000000000004

为每个URL分配两个字节就可以达到千分之几的冲突。比较保守的实现是，为每个URL分配4个字节，项目和位数比是1：32，误判率是0.00000021167340。对于5000万数量级的URL，布隆过滤器只占用200MB的空间

Hash函数选择

- 常见的应用比较广的hash函数有MD5， SHA1， SHA256，一般用于信息安全方面，比如签名认证和加密等。
  比如我们传输文件时习惯用对原文件内容计算它的MD5值，生成128 bit的整数，通常我们说的32位MD5值，是转换为HEX格式后的32个字符。
- MurmurHash是2008年发明的，相比较MD5， MurMurhash不太安全（当然MD5也被破译了， sha也可以被破译），但是性能是MD5的几十倍； MurmurHash有很多个版本， MurmurHash3修复了MurmurHash2的一些缺陷同时速度还要快一些，因此很多开源项目有用，**比如nginx、 redis、 memcashed、 Hadoop等，比如用于计算一致性hash**等。
- **MurmurHash**被比较好的测试过了，测试方法见 https://github.com/aappleby/smhasher, MurMurhash的实现也可以参考smhasher，或者参考https://sites.google.com/site/murmurhash。
  我们演示的布隆过滤器中的hash函数选择MurmurHash2算法。

在线验证公式

测试网址：https://hur.st/bloomfilter/

n

Number of items in the filter (optionally with SI units: k, M, G, T, P, E, Z, Y)

4000

p

Probability of false positives, fraction between 0 and 1 or a number indicating 1-in-p

1.0E-7

m

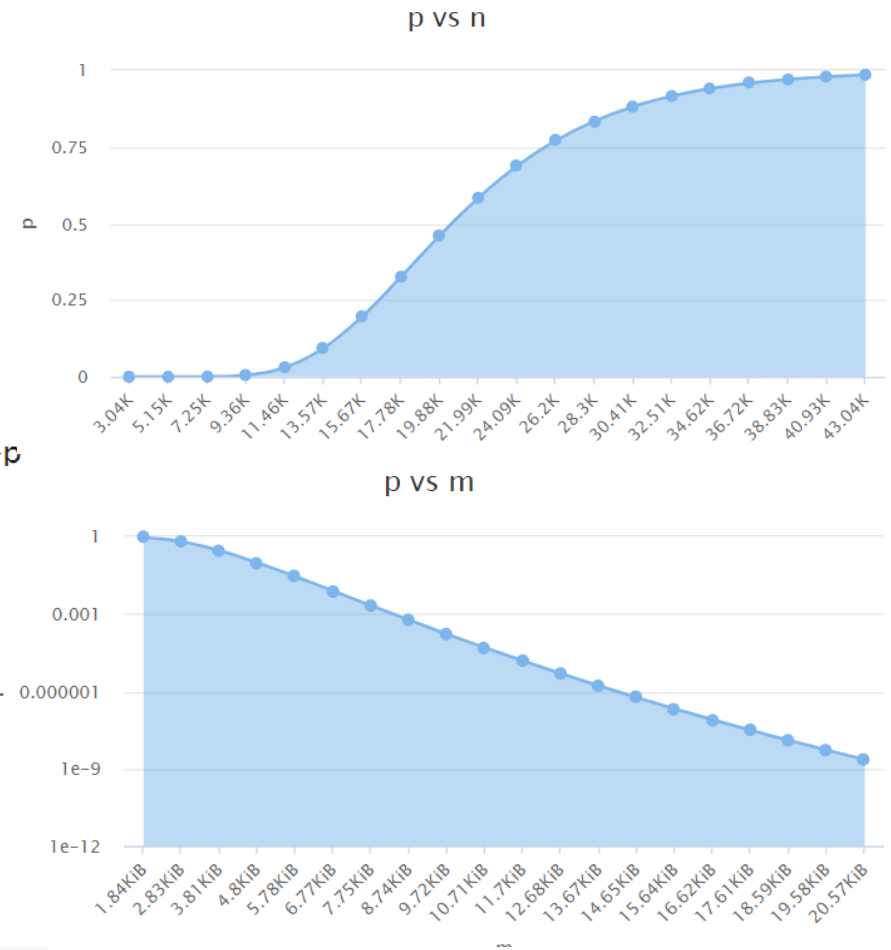
Number of bits in the filter (or a size with KB, KiB, MB, Mb, GiB, etc)

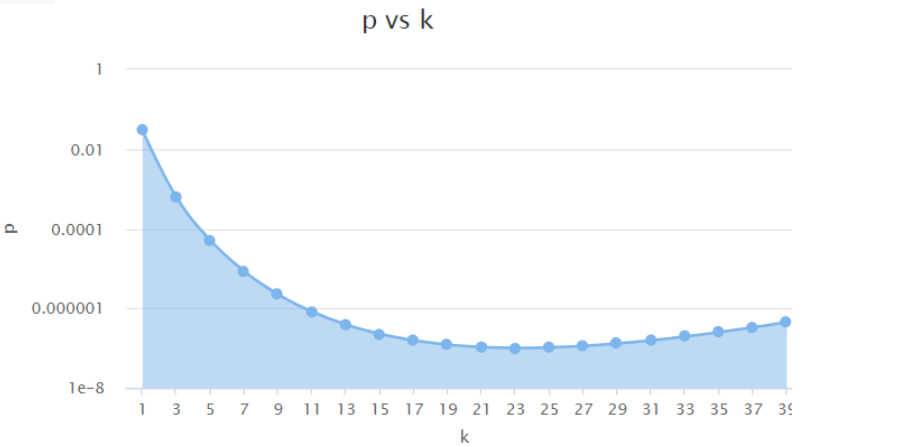
k

Number of hash functions

提交

n = 4,000
p = **0.0000001 (1 in 9,994,297)**
m = **134,191 (16.38KiB)**
k = **23**





可以删除元素不

在布隆过滤器算法中，不能因为有碰撞的可能，那即添加一个元素后，如果设置了k个bit为1，且某个bit位碰撞后，我们删除该元素时恰恰设置该bit位为0，则碰撞的元素无法判断，那么即不能在布隆过滤器中删除元素 。

补充1-c语言math.h库

C语言 math.h库数学函数

其中log相当于数学中的ln(即loge)。
而log10相当于数学中的lg。
loge和log10可以直接表示了。

补充2-Hash算法：双重散列

双重散列是线性开型寻址散列（开放寻址法）中的冲突解决技术。双重散列使用在发生冲突时将第二个散列函数应用于键的想法。

此算法使用：

(hash1(key) + i * hash2(key)) % TABLE_SIZE

来进行双哈希处理。hash1() 和 hash2() 是哈希函数，而 TABLE_SIZE 是哈希表的大小。当发生碰撞时，我们通过重复增加步长i 来寻找键。

具体原理可以参考：《散列函数之双重散列算法解决冲突问题》，
https://www.cnblogs.com/organic/p/6283476.html

具体见演示代码：**bloom_hash函数,** 在布隆过滤器里的目的是构造k个hash函数，如果不那么考虑性能可以使用k个MurmurHash喂以不同的seed实现k个hash函数。

零声学院

www.0voice.com

一切只为渴望更优秀的你！

为您的职业
添砖加瓦
升职加薪

努力方向

系统提升

项目实战

全职指导

下　　节　　课　　再　　见

联系Darren老师

课程顾问微信