Bloom Filter

0. 综述

- 主要结构是一个bitmap（set），不知道课件前一半讲那么多set干嘛
- 如果Bloom filter没有查到，就一定没有。（不存在漏报）
- 如果Bloom filter查到，不一定有。（误报）

1.增删改查

- 创建一个m位BitSet（C++自带，Python为bitarray）
- 将所有位初始化为0
- 选择k个不同的哈希函数。第i个哈希函数对字符串str哈希的结果记为h（i，str），且h（i，str）的范围是0到m-1 。

插入

将k个hash结果对应位&1

查询

查询k个hash结果对应位

- 只要出现0：不存在
- 全是1：存在（有误报）

删除

不能直接改成0，会影响到其他字符串。这个数据结构做不到。

布谷鸟**Cuckoo filter**：使用两个哈希函数对一个`key`进行哈希，得到桶中的两个位置，此时

- 如果两个位置都为为空则将`key`随机存入其中一个位置
- 如果只有一个位置为空则存入为空的位置
- 如果都不为空，则随机踢出一个元素，踢出的元素再重新计算哈希找到相应的位置

当然假如存在绝对的空间不足，那老是踢出也不是办法，所以一般会设置一个**踢出阈值**，如果在某次插入行为过程中连续踢出超过阈值，则进行扩容。

（相关论文Cuckoo Filter: Practically Better Than Bloom ，该路径下）

2. 优化

最小化误报率

没有漏判（不会将1改为0）

- 任一Bit而言，被置为1的概率 $P_1 = \frac{1}{m}$ ，依然是0的概率 $P_0 = 1 - \frac{1}{m}$
- 插入一个元素时，其 k 个Hash Function 都没有将该Bit置为1的概率 $P_0^1 = （1 - \frac{1}{m})^k$
- 插入全部n个元素后，该Bit依然为0的概率$P_0^n = （1 - \frac{1}{m})^{kn}$

  该Bit为1的概率$P_1^n = 1 - （1 - \frac{1}{m})^{kn}$

判定一个元素存在，要求k个哈希值对应的Bit的值均为1。

其误判率 P(true)：$P(true) = (P_1^n)^k = [1 - （1 - \frac{1}{m})^{kn}]^k$

> 根据基本极限 $\lim_{x \to \infty}(1 - \frac{1}{x})^{-x} = e$

可知：$P(true) = (1 - e^{\frac{-kn}{m}})^k = (1-p)^k = e^{k\ln(1-p)}$

> 当`BitArray`数组的大小m增大 或 插入Bloom Filter中的元素数目n 减小时，均可以使得误判率P(true)下降

最小化$P(true)$，$g = k\ln(1-p) = -\frac{m}{n}\ln p \ln(1-p)$

当$p = e^{\frac{-kn}{m}} = \frac{1}{2}$ **时，**$k = \ln 2 \frac{m}{n} = 0.7\frac{m}{n}$ **误报率最小**

最小误判率：$P(true) = (1 - e^{\frac{-kn}{m}})^k = 0.6185^{\frac{m}{n}}$

评价

- 使用了k个哈希函数，每个字符串跟k个bit对应。从而降低了冲突的概率。
- 快速查看是否存在
- 不能删除
- 有误报