布隆过滤器

我们经常要判断一个元素是否在一个集合中。比如在字处理软件中，需要检查一个英语单词是否拼写正确（也就是要判断它 是否在已知的字典中）；在 FBI，一个嫌疑人的名字是否已经在嫌疑名单上；在网络爬虫里，一个网址是否被访问过等等。最直接的方法就是将集合中全部的元素存在计算机中，遇到一个新 元素时，将它和集合中的元素直接比较即可。一般来讲，计算机中的集合是用哈希表（hash table）来存储的。它的好处是快速准确，缺点是费存储空间。当集合比较小时，这个问题不显著，但是当集合巨大时，哈希表存储效率低的问题就显现出来 了。比如说，一个象 Yahoo,Hotmail 和 Gmai 那样的公众电子邮件（email）提供商，总是需要过滤来自发送垃圾邮件的人（spamer）的垃圾邮件。一个办法就是记录下那些发垃圾邮件的 email 地址。由于那些发送者不停地在注册新的地址，全世界少说也有几十亿个发垃圾邮件的地址，将他们都存起来则需要大量的网络服务器。如果用哈希表，每存储一亿 个 email 地址， 就需要 1.6GB 的内存（用哈希表实现的具体办法是将每一个 email 地址对应成一个八字节的信息指纹 然后将这些信息指纹存入哈希表，由于哈希表的存储效率一般只有 50%，因此一个 email 地址需要占用十六个字节。一亿个地址大约要 1.6GB， 即十六亿字节的内存）。因此存贮几十亿个邮件地址可能需要上百 GB 的内存。除非是超级计算机，一般服务器是无法存储的。

今天，我们介绍一种称作布隆过滤器的数学工具，它只需要哈希表 1/8 到 1/4 的大小就能解决同样的问题。布隆过滤器可以插入元素，但不可以删除已有元素。其中的元素越多，false positive rate(误报率)越大，但是false negative (漏报)是不可能的。

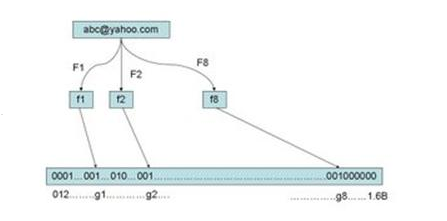
Bloom filter 可以看做是对 bit-map 的扩展, 它的原理是：

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阵列（Bit array）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，则被检索元素****一定不在****；
* 如果都是 1，则被检索元素****很可能****在。

它的优点是空间效率和查询时间都远远超过一般的算法，布隆过滤器存储空间和插入 / 查询时间都是常数O(k)。

应用1：假定我们存储一亿个电子邮件地址，我们先建立一个十六亿二进制（比特），即两亿字节的向量，然后将这十六亿个二进制全部设置为零。对于每一个电子邮件地址 X，我们用八个不同的随机数产生器（F1,F2, ...,F8） 产生八个信息指纹（f1, f2, ..., f8）。再用一个随机数产生器 G 把这八个信息指纹映射到 1 到十六亿中的八个自然数 g1, g2, ...,g8。现在我们把这八个位置的二进制全部设置为一。当我们对这一亿个 email 地址都进行这样的处理后。一个针对这些 email 地址的布隆过滤器就建成了。



布隆过滤器决不会漏掉任何一个在黑名单中的可 疑地址。但是，它有一条不足之处。也就是它有极小的可能将一个不在黑名单中的电子邮件地址判定为在黑名单中，因为有可能某个好的邮件地址正巧对应个八个都 被设置成一的二进制位。好在这种可能性很小。我们把它称为误识概率。在上面的例子中，误识概率在万分之一以下。

布隆过滤器的好处在于快速，省空间。但是有一定的误识别率。常见的补救办法是在建立一个小的白名单，存储那些可能别误判的邮件地址。

应用2：A,B 两个文件，各存放 50 亿条 URL，每条 URL 占用 64 字节，内存限制是 4G，让你找出 A,B 文件共同的 URL。如果是三个乃至 n 个文件呢？

分析 ：如果允许有一定的错误率，可以使用 Bloom filter，4G 内存大概可以表示 340 亿 bit。将其中一个文件中的 url 使用 Bloom filter 映射为这 340 亿 bit，然后挨个读取另外一个文件的 url，检查是否与 Bloom filter，如果是，那么该 url 应该是共同的 url（注意会有一定的错误率）。”

布隆过滤器的误判率的大小影响因素：

1. 最优的散列函数的个数
2. 位数组的大小