第 8 章 扩展

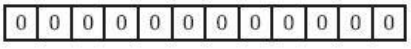
8.1 布隆过滤器

在日常生活中，包括在设计计算机软件时，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 googlechinablog.com/2006/08/blog-post.html，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。

布隆过滤器只需要哈希表 1/8 到 1/4 的大小就能解决同样的问题。

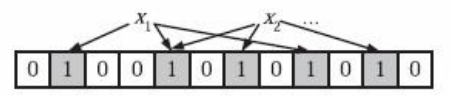
Bloom Filter 是一种空间效率很高的随机数据结构，它利用位数组很简洁地表示一个集合，并能判断一个元素是否属于这个集合。Bloom Filter 的这种高效是有一定代价的：在判断一个元素是否属于某个集合时，有可能会把不属于这个集合的元素误认为属于这个集合（false positive）。因此，Bloom Filter 不适合那些“零错误”的应用场合。而在能容忍低错误率的应用场合下，Bloom Filter 通过极少的错误换取了存储空间的极大节省。

下面我们具体来看 Bloom Filter 是如何用位数组表示集合的。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。

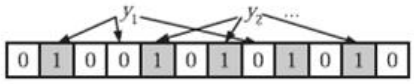


为了表达 S={x1, x2,…,xn}这样一个 n 个元素的集合，Bloom Filter 使用 k 个相互独立的

哈希函数（Hash Function），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 个哈希函数映射的位置 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，且有两个哈希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 次哈希函数，如果所有 $h_i(y)$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 y_1 就不是集合中的元素。y_2 或者属于这个集合，或者刚好是一个 false positive。



- 为了 add 一个元素，用 k 个 hash function 将它 hash 得到 bloom filter 中 k 个 bit 位，将这 k 个 bit 位置 1。
- 为了 query 一个元素，即判断它是否在集合中，用 k 个 hash function 将它 hash 得到 k 个 bit 位。若这 k bits 全为 1，则此元素在集合中；若其中任一位不为 1，则此元素比不在集合中（因为如果在，则在 add 时已经把对应的 k 个 bits 位置为 1）。
- 不允许 remove 元素，因为那样的话会把相应的 k 个 bits 位置为 0，而其中很有可能有其他元素对应的位。因此 remove 会引入 false negative，这是绝对不被允许的。

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。

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。

布隆过滤器具体算法高级内容，如错误率估计，最优哈希函数个数计算，位数组大小计算，请参见 http://blog.csdn.net/jiaomeng/article/details/1495500。