布隆过滤器的原理和实现

Jump to bottom

Liufeng Cheng edited this page on 6 Jan 2017 · 3 revisions

什么情况下需要布隆过滤器？

先来看几个比较常见的例子

- 字处理软件中，需要检查一个英语单词是否拼写正确
- 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
- yahoo, gmail等邮箱垃圾邮件过滤功能

这几个例子有一个共同的特点： **如何判断一个元素是否存在一个集合中？**

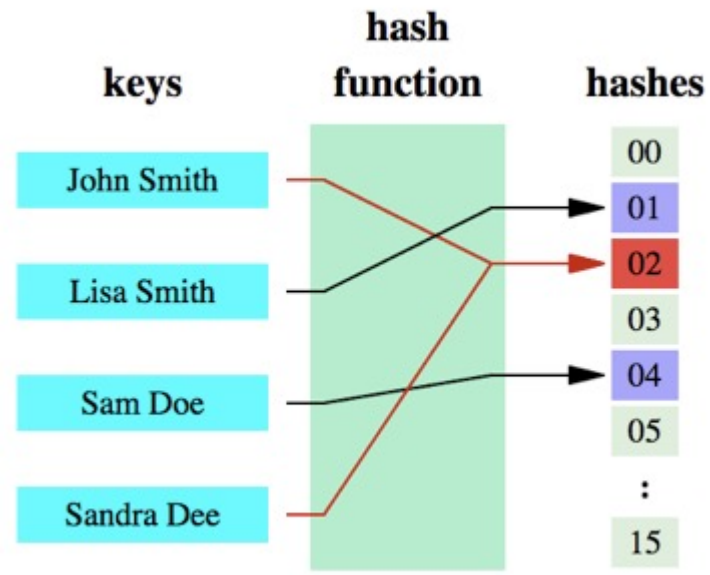
常规思路

- 数组
- 链表
- 树、平衡二叉树、Trie
- Map (红黑树)
- 哈希表

虽然上面描述的这几种数据结构配合常见的排序、二分搜索可以快速高效的处理绝大部分判断元素是否存在集合中的需求。但是当集合里面的元素数量足够大，如果有500万条记录甚至1亿条记录呢？这个时候常规的数据结构的问题就凸显出来了。数组、链表、树等数据结构会存储元素的内容，一旦数据量过大，消耗的内存也会呈现线性增长，最终达到瓶颈。有的同学可能会问，哈希表不是效率很高吗？查询效率可以达到O(1)。但是哈希表需要消耗的内存依然很高。使用哈希表存储一亿 个垃圾 email 地址的消耗？哈希表的做法：首先，哈希函数将一个email地址映射成8字节信息指纹；考虑到哈希表存储效率通常小于50%（哈希冲突）；因此消耗的内存：8 * 2 * 1亿字节 = 1.6G 内存，普通计算机是无法提供如此大的内存。这个时候，布隆过滤器（Bloom Filter）就应运而生。在继续介绍布隆过滤器的原理时，先讲解下关于哈希函数的预备知识。

哈希函数

哈希函数的概念是：将任意大小的数据转换成特定大小的数据的函数，转换后的数据称为哈希值或哈希编码。下面是一幅示意图：



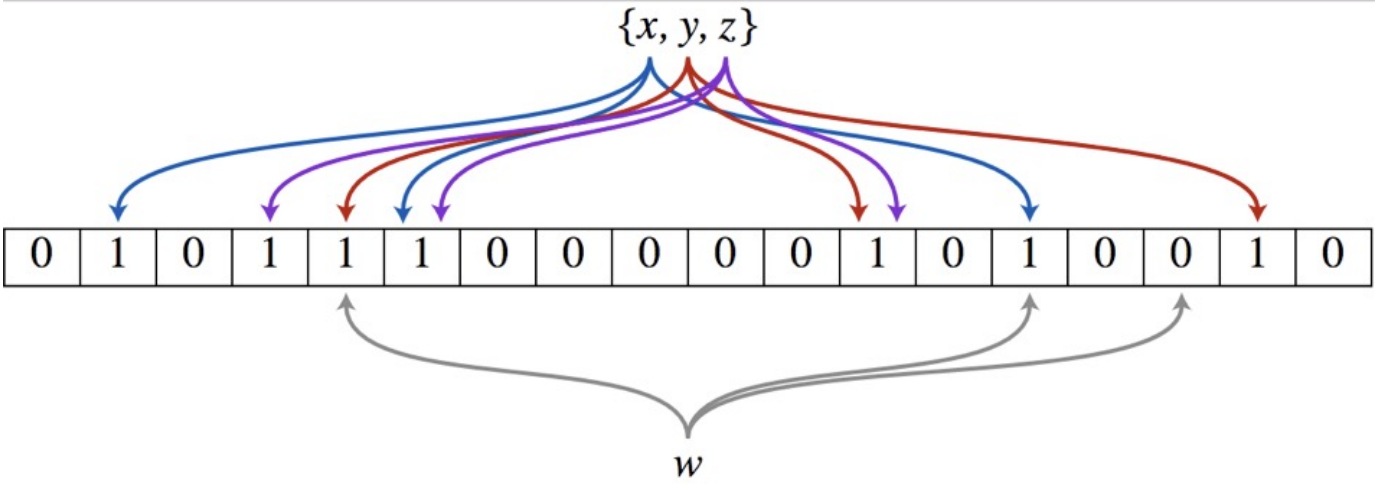
可以明显的看到，原始数据经过哈希函数的映射后称为了一个个的哈希编码，数据得到压缩。哈希函数是实现哈希表和布隆过滤器的基础。

布隆过滤器介绍

- 巴顿.布隆于一九七零年提出
- 一个很长的二进制向量 （ 位数组 ）
- 一系列随机函数 (哈希)
- 空间效率和查询效率高
- 有一定的误判率（ 哈希表是精确匹配 ）

布隆过滤器原理

布隆过滤器（ Bloom Filter ）的核心实现是一个超大的位数组和几个哈希函数。假设位数组的长度为m，哈希函数的个数为k



以上图为例，具体的操作流程：假设集合里面有3个元素{x, y, z}，哈希函数的个数为3。首先将位数组进行初始化，将里面每个位都设置位0。对于集合里面的每一个元素，将元素依次通过3个哈希函数进行映射，每次映射都会产生一个哈希值，这个值对应位数组上面的一个点，然后将位数组对应的位置标记为1。查询W元素是否存在集合中的时候，同样的方法将W通过哈希映射到位数组上的3个点。如果3个点的其中有一个点不为1，则可以判断该元素一定不存在集合中。反之，如果3个点都为1，则该元素可能存在集合中。注意：此处不能判断该元素是否一定存在集合中，可能存在一定的误判率。可以从图中可以看到：假设某个元素通过映射对应下标为4，5，6这3个点。虽然这3个点都为1，但是很明显这3个点是不同元素经过哈希得到的位置，因此这种情况说明元素虽然不在集合中，也可能对应的都是1，这是误判率存在的原因。

布隆过滤器添加元素

- 将要添加的元素给k个哈希函数
- 得到对应于位数组上的k个位置
- 将这k个位置设为1

布隆过滤器查询元素

- 将要查询的元素给k个哈希函数
- 得到对应于位数组上的k个位置
- 如果k个位置有一个为0，则肯定不在集合中
- 如果k个位置全部为1，则可能在集合中

布隆过滤器实现

下面给出python的实现，使用murmurhash算法

```
import mmh3
from bitarray import bitarray


# zhihu_crawler.bloom_filter

# Implement a simple bloom filter with murmurhash algorithm.
# Bloom filter is used to check wether an element exists in a collection, and it has a
good performance in big data situation.
# It may has positive rate depend on hash functions and elements count.



BIT_SIZE = 5000000

class BloomFilter:

    def __init__(self):
        # Initialize bloom filter, set size and all bits to 0
        bit_array = bitarray(BIT_SIZE)
```

```
        bit_array.setall(0)

        self.bit_array = bit_array

    def add(self, url):
        # Add a url, and set points in bitarray to 1 (Points count is equal to hash
funcs count.)
        # Here use 7 hash functions.
        point_list = self.get_postions(url)

        for b in point_list:
            self.bit_array[b] = 1

    def contains(self, url):
        # Check if a url is in a collection
        point_list = self.get_postions(url)

        result = True
        for b in point_list:
            result = result and self.bit_array[b]

        return result

    def get_postions(self, url):
        # Get points positions in bit vector.
        point1 = mmh3.hash(url, 41) % BIT_SIZE
        point2 = mmh3.hash(url, 42) % BIT_SIZE
        point3 = mmh3.hash(url, 43) % BIT_SIZE
        point4 = mmh3.hash(url, 44) % BIT_SIZE
        point5 = mmh3.hash(url, 45) % BIT_SIZE
        point6 = mmh3.hash(url, 46) % BIT_SIZE
        point7 = mmh3.hash(url, 47) % BIT_SIZE


        return [point1, point2, point3, point4, point5, point6, point7]
```

爬虫工作原理和抓取策略

Clone this wiki locally

https://github.com/cpselvis/zhihu-crawler.wiki.git