# 实验内容与要求

# 实验四 布隆过滤器

一、实验目的

* 熟悉布隆滤波器算法原理和优缺点
* 能够设计布隆滤波器判断测试数据是否在容器中
* 能够计算不同哈希函数和位数组长度下的误报数
* 通过实验了解布隆滤波器如何节省内存空间

1. 实验内容

* 根据案例，了解布隆过滤器工作原理
* 完成给定的代码框架中的函数
* 记录不同哈希函数和位数组大小下的误报数，查看位数组所占内存大小

三、实验原理、方法和手段

1.原理

布隆滤波器是一种轻量级的散列表。散列表和布隆滤波器都支持高效的插入和查找。布隆滤波器比散列表更具空间效率，但这是以查找的“误报”为代价的。也就是说，滤波器可以肯定地说一个元素没有被插入（没有漏报的可能性），但是可能错误报告一个元素已经被插入，但实际上并没有插入（误报）。

2.方法

当需要一个允许快速查找和插入的数据结构时，关心数据结构使用多大空间，可以考虑使用布隆滤波器。例如：运营一个网站，我并不特别在意被拦截的URL是否偶尔能够访问我的网站（允许误报），但我在意不在拦截列表中的用户无法访问该网站（不允许漏报）。本次实验内容将以恶意URL为背景设计布隆过滤器。

1. 手段

在实验前，我们先通过一个案例，了解一下哈希函数、位数组大小对误报率的影响。

这里，我们先通过一个有趣的案例快速熟悉布隆过滤器是如何使用的。在案例中需要安装pyhash库，如实验一所述，已经讲解过如和python环境中安装需要的库，安装pyhash过程如下：

* 安装pyhash所依赖的Boost库

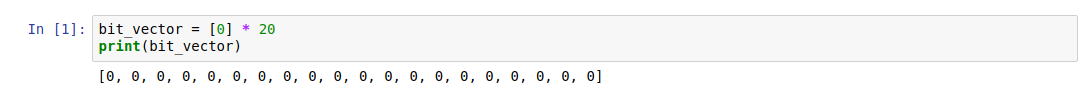
$ sudo apt-get install libboost-all-dev

* 使用PyPI安装pyhash

$ sudo pip install pyhash

下面开始进入案例演示：

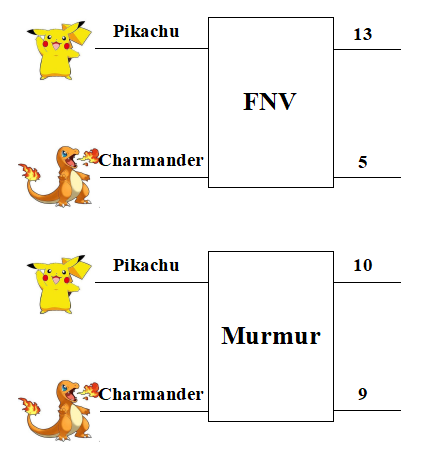
1.我们从一个元素为0或1的位向量开始。首先，我们将位向量初始化为全零。就这个简单的例子而言，我们将注意力限制在一个大小为20的位向量上。



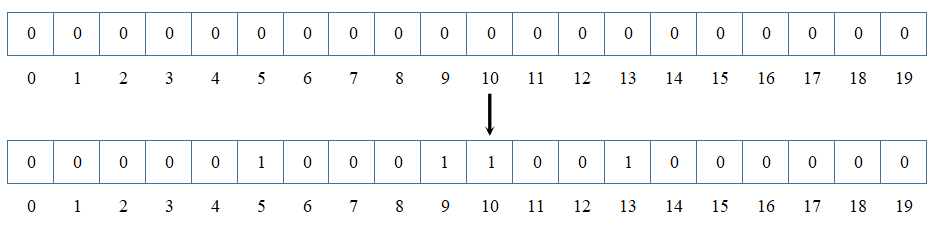
1. 接下来使用一对哈希函数将任意大小的数据映射到固定大小的数据的函数。布隆滤波器中使用的哈希函数的类型通常不是“加密变体”，例如，通常不会使用类似MD5加密的方法。而像Murmur和FNV这样的非加密散列函数是最常用的，主要是因为它们的运算速度超过了大多数加密哈希函数。Python中有一个很好的库叫做pyhash，它完全由非加密哈希函数组成。因此，我们导入pyhash。



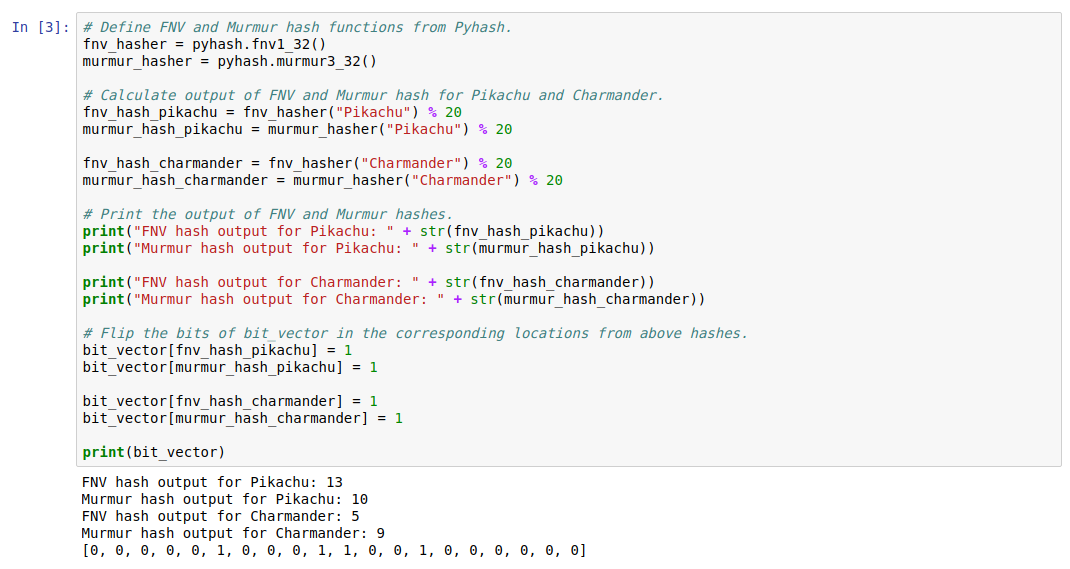
1. 这里结合位向量和非加密哈希函数来组合一个布隆滤波器的案例。在本案例中，我们将布隆滤波器作为神奇宝贝球，一个跟踪我们已经抓到的神奇宝贝的设备。每次我们抓到一只神奇宝贝，我们通过两个哈希函数（FNV哈希函数和Murmur哈希函数）运行神奇宝贝的名字来更新我们的神奇宝贝球。哈希函数的输出即在位向量中对应更新的位。使用FNV哈希函数对20取余（因为20是本案例位向量的大小），然后对字符串“Pikachu”和“Charmander”进行哈希，结果是13和5。同样，使用Murmur哈希函数对20取余，然后对相同的字符串进行哈希，结果分别为10和9。（Pikachu：皮卡丘；Charmander：小火龙-神奇宝贝的名字）。因此，将位向量中第5、9、10、13更新。

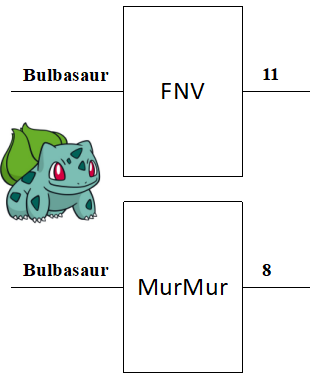


我们使用上述哈希算法的输出的值来更新位向量中对应索引处的值。例如：

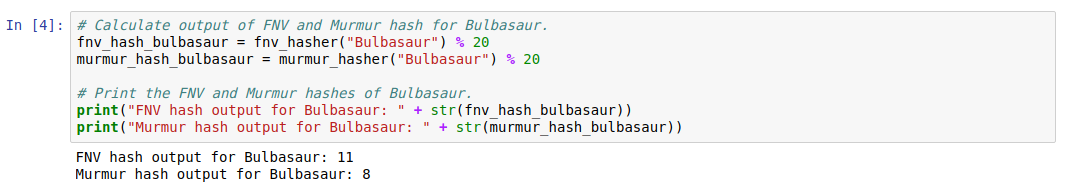


python实现如下：

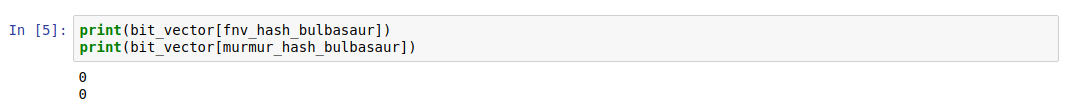
4.这时，前方出现一只妙蛙种子(Bulbasaur)，让我们来查看一下布隆滤波器-神奇宝贝球，看看是否已经捕获了妙蛙种子。对字符串“Bulbasaur”进行哈希（对20取余）的输出结果分别对应FNV和Murmur的11和8。



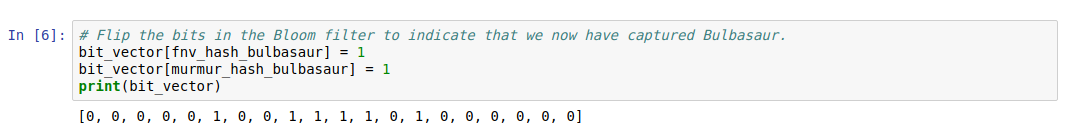
代码实现如下：



5.查看布隆滤波器的位向量对应11和8索引处是“开”还是“关”。



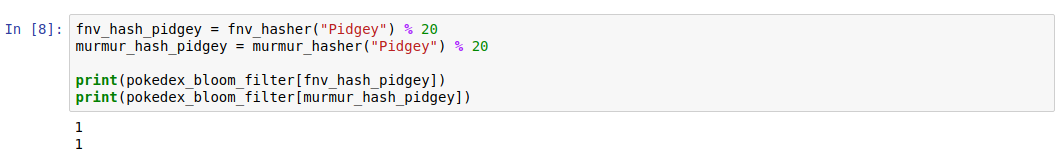
6.由于对Bulbasaur(妙蛙种子)进行哈希的两个函数结果都是0，所以妙蛙种子不在我们的神奇宝贝球中。让我们抓住妙蛙种子并更新滤波布隆器。



7.在我们的案例中，布隆滤波器由20个元素组成。 对于我们的神奇宝贝球，滤波器的长度很可能过小。为了扩展算法应用性，我们假设已经抓住了151个神奇宝贝中的150个。



8.这时前方出现了一只波波鸟(字符串”Pidget”)，在决定抓波波鸟之前，神奇宝贝训练家拿出他的神奇宝贝球看是否已经抓住了它。



1. 布隆滤波器中的元素表明Pidget已经在神奇宝贝中。如前所述，布隆滤波器只会给出误报，而不会产生漏报。也就是说，如果Pidgey真的在我们的布隆过滤器中，就不可能报告它不存在。但是如果波波鸟不在我们的布隆滤波器中，可能会报告在其中。这正是我们目前所面临的问题，因为本案例中Pidgey不在数组元素中。由于布隆滤波器的大小只有20，元素的总数是151，因此不可避免地会出现一些哈希冲突，导致类似我们现在遇到的情况。

误报率大约是：



其中k是所使用的哈希函数的数量，n是插入元素的数量，m是布隆滤波器的位向量的长度。

所以为了降低碰撞概率，我们需要调整参数k和m; 即我们使用的哈希函数的数量和位向量的大小。假设哈希函数完全随机地选择位向量中的索引，则布隆滤波器中的元素未被设置为1的概率可以通过下列公式进行计算：



我们就可以得到布隆滤波器中的第k个元素没有被设置为1的可能性：



如果我们在布隆滤波器中插入了n个神奇宝贝，那么向量中某个位仍然为0的概率为：



例如，在案例中，布隆滤波器的位向量大小m = 20，哈希函数k = 2，即FNV和Murmur。当我们抓到Pikachu和Charmander时，对应n = 2。因此，在捕获妙蛙种子(Bulbasaur)之前，位向量中的对应位置的元素仍然是0的概率为：



这个结果还好，但是当我们捕捉到更多的神奇宝贝时，这个数字会下降很多。由于总共有151个神奇宝贝，因此保持位向量中的所有元素仍然是0并不需要很长时间。例如，即使我们抓到了20个神奇宝贝，向量中的任何元素仍然是0的可能性只有0.1285：



我们需要为布隆滤波器分配更多空间并且关注使用哈希函数的数量。

如果我们有太多的哈希函数，那么我们的布隆滤波器将会运算非常慢，而且布隆过滤器中的元素会更快填满。另外，如果我们限制哈希函数数目，会增加误报率。

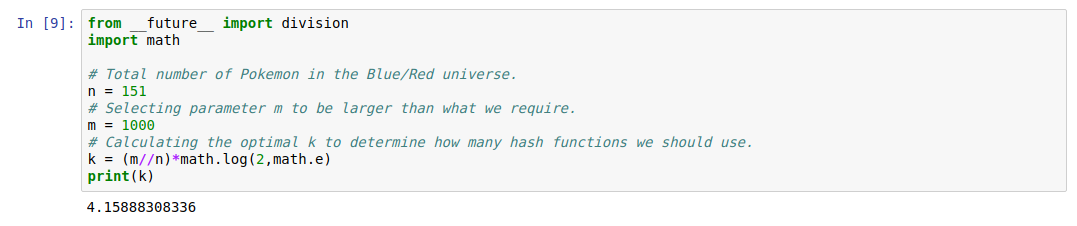
如果我们同时选择m和n，下面的公式可以给出我们k的最优值，即使用的哈希函数的数量：



使得布隆滤波器误报率最小化，函数如下：



所以n = 151时。对于m，我们设置m = 1000 作为布隆滤波器位向量中的元素数。使用这些参数，我们可以计算k。



使用这些选定的参数，我们遇到误报的概率是：



四、实验条件

运行Windows操作系统的计算机1台，VMware以及对应的ubuntu系统，并根据实验需要安装必要的python库。

1. 实验步骤

1.实验四代码说明：

参照Bloom\_Filter.py文件完成实验任务。

我们提供的代码框架已经实现了位图初始化、读取文件、写文件以及最终打印输出。

1. 数据集说明:

数据源：Alexa前100万个网址。由于需要翻墙下载，文件包里已经提供。我们取前1万条网址作为测试数据，即待检测的URL，保存为top2m.csv。我们取第8001条~第100万条网址作为恶意网址数据库，保存为top1m.csv(实际情况下，该文件将存储在脱机服务器中）。因此待检测的1万条URL中有2000条是恶意的。

1. 实验内容：

在实验步骤1.2的基础上，画出恶意URL误报数和位图大小变化的关系，并尝试不同数量的哈希函数，查看误报数的变化。~~查看bloom\_filter.txt文件大小的变化，~~结合hash表思考布隆过滤器的优缺点。