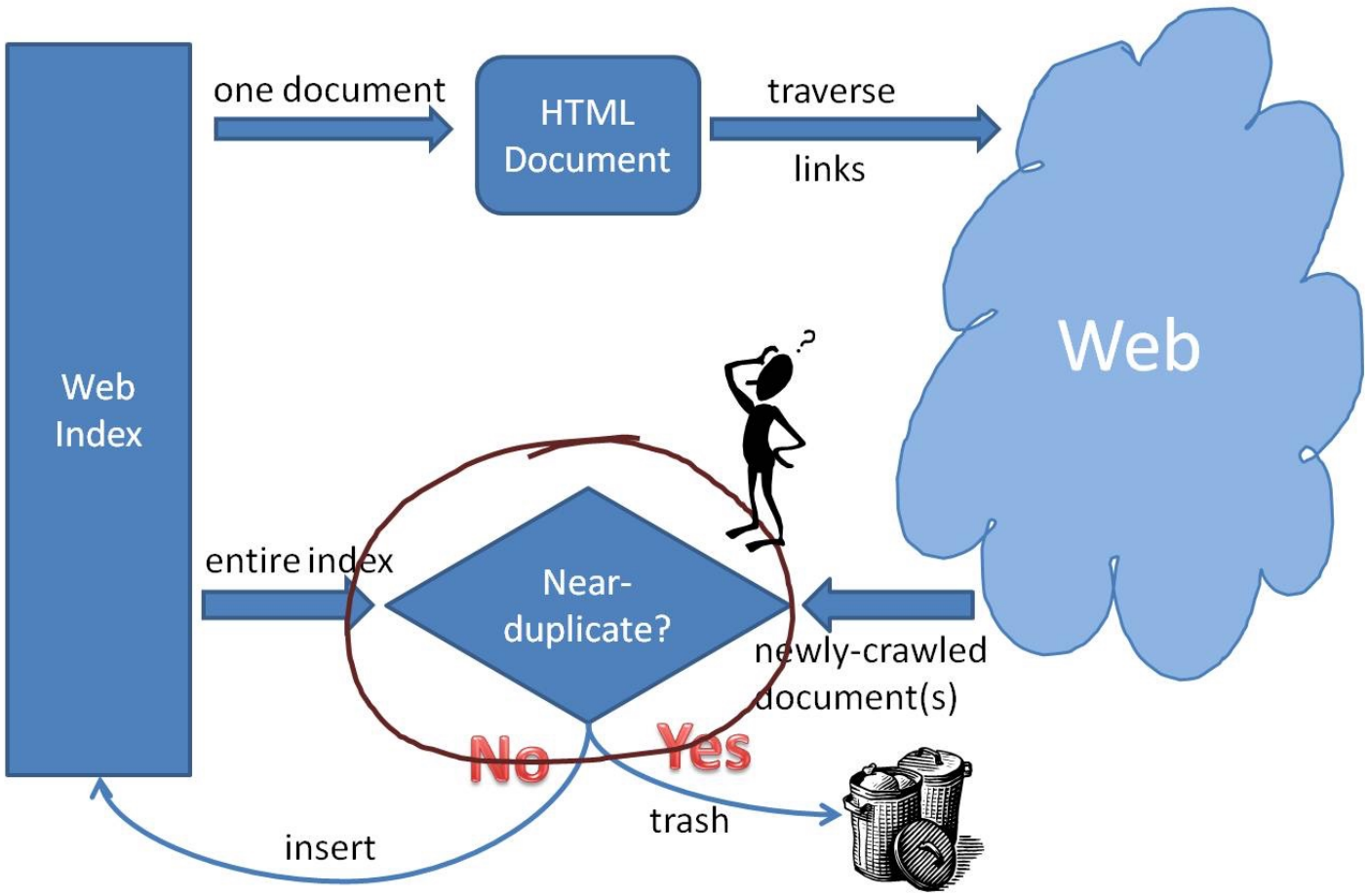
数据结构与算法（17）：simhash

一、引入

随着信息爆炸时代的来临，互联网上充斥着着大量的近重复信息，有效地识别它们是一个很有意义的课题。例如，对于搜索引擎的爬虫系统来说，收录重复的网页是毫无意义的，只会造成存储和计算资源的浪费；同时，展示重复的信息对于用户来说也并不是最好的体验。造成网页近重复的可能原因主要包括：

- 镜像网站
- 内容复制
- 嵌入广告
- 计数改变
- 少量修改

一个简化的爬虫系统架构如下图所示：



事实上，传统比较两个文本相似性的方法，大多是将文本分词之后，转化为特征向量距离的度量，比如常见的欧氏距离、海明距离或者余弦角度等等。两两比较固然能很好地适应，但这种方法的一个最大的缺点就是，无法将其扩展到海量数据。例如，试想像Google那种收录了数以几十亿互联网信息的大型搜索引擎，每天都会通过爬虫的方式为自己的索引库新增的数百万网页，如果待收录每一条数据都去和网页库里面的每条记录算一下余弦角度，其计算量是相当恐怖的。

我们考虑采用为每一个web文档通过hash的方式生成一个指纹（fingerprint）。传统的加密式hash，比如md5，其设计的目的是为了让整个分布尽可能地均匀，输入内容哪怕只有轻微变化，hash就会发生很大地变化。我们理想当中的哈希函数，需要对几乎相同的输入内容，产生相同或者相近的hashcode，换句话说，hashcode的相似程度要能直接反映输入内容的相似程度。很明显，前面所说的md5等传统hash无法满足我们的需求。

二、simhash的原理

simhash是locality sensitive hash（局部敏感哈希）的一种，最早由Moses Charikar在《similarity estimation techniques from rounding algorithms》一文中提出。Google就是基于此算法实现网页文件查重的。simhash算法的主要思想是降维，将高维的特征向量映射成一个f-bit

的指纹(fingerprint)，通过比较两篇文章的f-bit指纹的Hamming Distance来确定文章是否重复或者高度近似。我们假设有以下三段文本：

- the cat sat on the mat
- the cat sat on a mat
- we all scream for ice cream

使用传统hash可能会产生如下的结果：

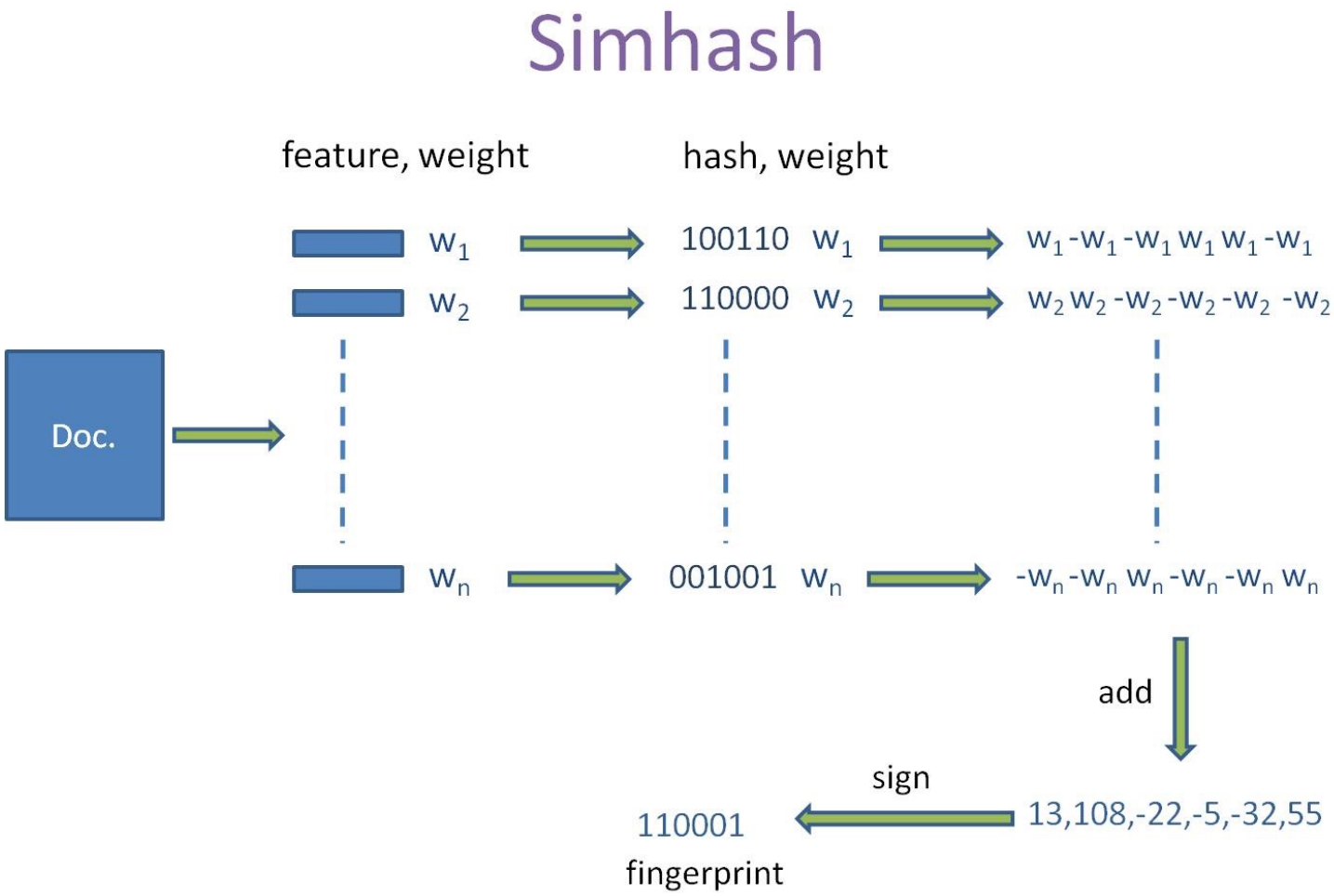
```
irb(main):006:0> p1 = 'the cat sat on the mat'
irb(main):005:0> p2 = 'the cat sat on a mat'
irb(main):007:0> p3 = 'we all scream for ice cream'
irb(main):007:0> p1.hash
=> 415542861
irb(main):007:0> p2.hash
=> 668720516
irb(main):007:0> p3.hash
=> 767429688
```

使用simhash会应该产生类似如下的结果:

```
irb(main):003:0> p1.simhash
=> 851459198
00110010110000000011110001111110
irb(main):004:0> p2.simhash
=> 847263864
00110010100000000011100001111000
irb(main):002:0> p3.simhash
=> 984968088
00111010101101010110101110011000
```

海明距离的定义，为两个二进制串中不同位的数量。上述三个文本的simhash结果，其两两之间的海明距离为(p1,p2)=4，(p1,p3)=16以及(p2,p3)=12。事实上，这正好符合文本之间的相似度，p1和p2间的相似度要远大于与p3的。

如何实现这种hash算法呢？图解如下：



算法过程大概如下：

1. 将Doc进行关键词抽取(其中包括分词和计算权重)，抽取出n个(关键词，权重)对， 即图中的 `(feature, weight)` 们。 记为 `feature_weight_pairs = [fw1, fw2 … fwn]`，其中 `fwn = (feature_n,weight_n )`。
2. `hash_weight_pairs = [ (hash(feature), weight) for feature, weight in feature_weight_pairs ]` 生成图中的 `(hash,weight)` 们, 此时假设hash生成的位数bits_count = 6（如图）；
3. 然后对 `hash_weight_pairs` 进行位的纵向累加，如果该位是1，则+weight,如果是0，则-weight，最后生成bits_count个数字，如图所示是[13, 108, -22, -5, -32, 55], 这里产生的值和hash函数所用的算法相关。
4. `[13,108,-22,-5,-32,55] -> 110001` 这个就很简单啦，正1负0。

三、海明距离

当我们算出所有doc的simhash值之后，需要计算doc A和doc B之间是否相似的条件是：A和B的海明距离是否小于等于n，这个n值根据经验一般取值为3,

那海明距离怎么计算呢？二进制串A 和 二进制串B 的海明距离 就是 A xor B 后二进制中1的个

数。

```
举例如下：
A = 100111;
B = 101010;
hamming_distance(A, B) = count_1(A xor B) = count_1(001101) = 3;
```

simhash本质上是局部敏感性的hash，和md5之类的不一样。 正因为它的局部敏感性，所以我们可以使用海明距离来衡量simhash值的相似度。