**4.4.1去重原理**

“去重”是日常工作中会经常用到的一项技能，在爬虫领域更是常用，并且规模一般都比较大。去重需要考虑两个点：去重的数据量、去重速度。为了保持较快的去重速度，一般选择在内存中进行去重。

简单来说就是通过Hash函数存储网络爬虫的遍历轨迹，并规定某一Web页被遍历过，则在哈希表中的相应槽位填充1，否则填充0。也就是说，此类方式用1或。的值表示Web页是否被爬虫抓取过。在具体实现过程中，哈希函数起到至关重要的作用，目前一般使用MD5 ( )函数，将网页文件的地址即URL字符串转换为128位散列值。

MD5就是将任意长度的消息转换成 128位固定长度的消息摘要的函数，显然，MD5 ( )函数产生的值很大，为2个不同的数，需要的内存空间巨大。因此，在实际处理

中还要将MD5()函数的值进行模运算映射到哈希表中。其公式可设为:

MD5 (URL) MOD N

其中，URL为抓取的地址，N为存储哈希表的位长。通过该式的转换，可使输入的URL地址被映射到大小为N的哈希表的某个位上，以便确定其地址是否被抓取

过。

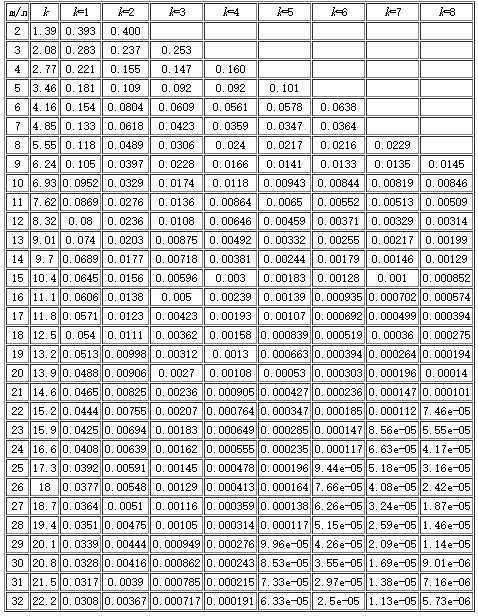
为了解决重复搜集网页的问题，可以定义两个数据库:“未爬行的U1u库”和“已爬行的URL库”。“未爬行的URL库”存储待访问队列的URL“已爬行的U1u库”存储已遍历过的URL。对于已访问过的、未访问过的URL利用MD5(URL)函数分别作MD5摘要，以获取其惟一标识，并建立两个集合。新解析出的URL，首先根据已经访问过URL集合判断是否已抓取过.如没有被抓取.则放入“未爬行的URL数据库”中.否则放入“已爬行的URI库”

**4.4.2 Redis去重**

* 数据量不大时，可以直接放在内存里面进行去重，例如python可以使用set()进行去重。
* 当去重数据需要持久化时可以使用redis的set数据结构。
* 当数据量再大一点时，可以用不同的加密算法先将长字符串压缩成 16/32/40 个字符，再使用上面两种方法去重；
* 当数据量达到亿（甚至十亿、百亿）数量级时，内存有限，必须用“位”来去重，才能够满足需求。Bloomfilter就是将去重对象映射到几个内存“位”，通过几个位的 0/1值来判断一个对象是否已经存在。
* 然而Bloomfilter运行在一台机器的内存上，不方便持久化（机器down掉就什么都没啦），也不方便分布式爬虫的统一去重。如果可以在Redis上申请内存进行Bloomfilter，以上两个问题就都能解决了。

本文即是用Python，基于Redis实现Bloomfilter去重。

1. Bloomfilter算法如何使用位去重，这个百度上有很多解释。简单点说就是有几个seeds，现在申请一段内存空间，一个seed可以和字符串哈希映射到这段内存上的一个位，几个位都为1即表示该字符串已经存在。插入的时候也是，将映射出的几个位都置为1。



1. 需要提醒一下的是Bloomfilter算法会有漏失概率，即不存在的字符串有一定概率被误判为已经存在。这个概率的大小与seeds的数量、申请的内存大小、去重对象的数量有关。下面有一张表，m表示内存大小（多少个位），n表示去重对象的数量，k表示seed的个数。例如我代码中申请了256M，即1<<31（m=2^31，约21.5亿），seed设置了7个。看k=7那一列，当漏失率为8.56e-05时，m/n值为23。所以n = 21.5/23 = 0.93(亿），表示漏失概率为8.56e-05时，256M内存可满足0.93亿条字符串的去重。同理当漏失率为0.000112时，256M内存可满足0.98亿条字符串的去重。
2. 基于Redis的Bloomfilter去重，其实就是利用了Redis的String数据结构，但Redis一个String最大只能512M，所以如果去重的数据量大，需要申请多个去重块（代码中blockNum即表示去重块的数量）。
3. 代码中使用了MD5加密压缩，将字符串压缩到了32个字符（也可用hashlib.sha1()压缩成40个字符）。它有两个作用，一是Bloomfilter对一个很长的字符串哈希映射的时候会出错，经常误判为已存在，压缩后就不再有这个问题；二是压缩后的字符为 0~f 共16中可能，我截取了前两个字符，再根据blockNum将字符串指定到不同的去重块进行去重。