**后缀数组（Suffix Array）——理论和思想**

　　这一篇和下一篇博文我准备写一下我在参加ACM/ICPC期间曾经研究过的[后缀数组](http://jgshining.com/blog/tag/%E5%90%8E%E7%BC%80%E6%95%B0%E7%BB%84/" \o "查看 后缀数组 的全部文章" \t "_blank)。关于[后缀数组](http://jgshining.com/blog/tag/%E5%90%8E%E7%BC%80%E6%95%B0%E7%BB%84/" \o "查看 后缀数组 的全部文章" \t "_blank)，网上有很多英文资料，但是很多现在的研究结果都是受1991年Udi Manber & Gene Myers的《*Suffix arrays: a new method for on-line string searches*》中所提的方法的启发，采用倍增的思想。当然，现在有国外学者提到的三分+分治的线性构造方法除外。

　　写这两篇文章主要是为了把一种思想写下来，同时练练已经生疏了很久的算法。我曾经在老的博客上写过一个比较丑陋的后缀数组构造算法，在后一篇文章中我将结合近期我看到的资料对它进行优化，使其变得比较美观 :-)  
  
　　我们定义一个字符串A，其表示：

A = a0a1a2…an-1

　　其中ai(0 ≤ i < n)都是字符集E中的字符，这个字符集是全序的，也就是说其中任意两个字符都是可以比较大小的。例如，用ASCII码编码的英文字符集就是这样的一个E的特例。

　　|A|表示字符串的长度，其值为n；Ai(0 ≤ i < n)表示一个后缀，它其实表示一个字符串：

Ai = aiai+1ai+2…an-1

　　我们让运算符“≤”表示两个串按照字典序比较，然后定义运算符“≤h”表示两个串的前h个字符按照字典序比较（=h、<h等同理），那么就有：

**结论1 若Aj =h Ak且Aj+h ≤h Ak+h，则Aj ≤2h Ak （j+h, k+h < n，“≤”换成“=、<、>”等等依然成立）**

　　这个是1989年Udi Manber & Gene Myers发明 *n*log*n* 后缀数组生成算法的主要理论依据。然而他们的天才之处不是在于看到了这个结论而是将这个结论与“复用”的思想结合在了一起。  
  
　　什么是后缀数组呢？

　　后缀数组（[Suffix Array](http://jgshining.com/blog/tag/suffixarray/" \o "查看 Suffix Array 的全部文章" \t "_blank)）是一个存放索引的数组，如果把这个数组命名为SA，那么有：

A[SA[x]] ≤ A[SA[y]]，其中0 ≤ x ≤ y < n

　　要产生这样一个数组，我们可以用最普通的sort/qsort结合strcmp，这个看起来是一个不错的想法，但是考虑到strcmp其实不是常数时间复杂度而是线性时间复杂度的，所以这个算法就显得不是那么高效了。Udi Manber & Gene Myers的聪明之处就是将结论1中的h取成了“1、2、4、8、16…”这样的指数数列，只要每趟比较保证字符串A的所有后缀按≤h有序，那么相应地，每个后缀扩展成2h长度之后，只要比较其后h个字符就行了，而这后h个字符其实就是其他某个后缀的前h个字符，其实已经比较过了，直接结合结论1可以得出结果，仅仅花了常数时间。这样，经过 log*n* 趟比较，后缀数组就生成了。例如：

A = abba  
A0 = abba  
A1 = bba  
A2 = ba  
A3 = a

　　h = 1 时，A0 A1 A2 A3 的≤h有序序列为：A0 =h A3 <h A2 =h A1  
　　h = 2 时，要决定A1与A2的≤2h比较结果，因为 A2 =h A1，根据结论1，只要看a与ba的≤h比较结果就行了，而我们欣喜地发现，这个结果其实就是A2与A3的≤h比较结果，在h = 1的时候早就得出了结论——A3 <h A2！

　　所以，我们只要经过 log*n* 趟比较，每趟比较花费O(*n*)的时间进行2个关键字的桶排序，那么就可以得到一个后缀数组了！

　　PS：现在有更好的线性的结果了，但是算法相对复杂，我也没有怎么看过 :-)