[**后缀数组**](http://www.cnblogs.com/zgmf_x20a/archive/2008/11/19/1336958.html)

后缀数组就是将字符串所有后缀排序后的数组，设字符串为S，令后缀Suffix(i)表示S[i..len(S)]。用两个数组记录所有后缀的排序结果：

* Rank[i]记录Suffix(i)排序后的序号，即Suffix[i]在所有后缀中是第Rank[i]小的后缀
* SA[i]记录第i位后缀的首字母位置，即Suffix[SA[i]]在所有后缀中是第i小的后缀

然后就是怎么快速求所有后缀的顺序了，其中的关键是如何减少两个后缀比较的复杂度  
方法是倍增法，定义一个字符串的k-前缀为该字符串的前k个字符组成的串，关于在k-后缀上的定义Suffix(k,i)、SA[k,i]和Rank[k,i]类似于前，则有

* 若Rank[k,i]＝Rank[k,j]且Rank[k,i+k]＝Rank[k,j+k]，则Suffix[2k,i]＝Suffix[2k,j]
* 若Rank[k,i]＝Rank[k,j]且Rank[k,i+k]<Rank[k,j+k]，则Suffix[2k,i]<Suffix[2k,j]
* 若Rank[k,i]<Rank[k,j]，则Suffix[2k,i]<Suffix[2k,j]

这样就能在常数时间内比较Suffix(2^k, i)之间的大小，从而对Suffix(2^k,i)时行排序，最后当2^k>n时，Suffix(2^k, i)之间的大小即为所有后缀之间的大小  
  
于是求出了所有后缀的排序，有什么用呢？主要是用于求它们之间的最长公共前缀（Longest Common Prefix，LCP）  
  
令LCP(i,j)为第i小的后缀和第j小的后缀（也就是Suffix(SA[i])和Suffix(SA[j])）的最长公共前缀的长度，则有如下两个性质：

1. 对任意i<=k<=j，有LCP(i,j) = min(LCP(i,k),LCP(k,j))
2. LCP(i,j)=min(i<k<=j)(LCP(k-1,k))

第一个性质是显然的，它的意义在于可以用来证明第二个性质。第二个性质的意义在于提供了一个将LCP问题转换为RMQ问题的方法：  
令height[i]＝LCP(i-1,i)，即height[i]代表第i小的后缀与第i-1小的后缀的LCP，则求LCP(i,j)就等于求height[i+1]~height[j]之间的RMQ，套用RMQ算法就可以了，复杂度是预处理O(nlogn)，查询O(1)  
  
然后height的求法要用到另一个数组：令h[i]=height[Rank[i]]，即h[i]表示Suffix(i)的height值（同时height[i]就表示Suffix(SA[i])的height值），则有height[i]=h[SA[i]]  
然后h[i]有个性质：

* h[i] >= h[i-1]-1

用这个性质我们在计算h[i]的时候进行后缀比较时只需从第h[i-1]位起比较，从而总的比较的复杂度是O(n)，也就是说h数组在O(n)的时间内解决了。求出了h数组，根据关系式height[i]=h[SA[i]]可以在O(n)时间内求出height数组，于是可以在O(n)时间内求出height数组，从而整个LCP问题就解决了^\_^  
  
然后后缀数组的应用就是利用它的LCP在需要字符串比较时降低复杂度。同时由于后缀数组的有序性可以很方便地使用二分  
  
于是总结一下要点：

* **利用倍增算法在O(nlogn)的时间内对后缀数组进行排序**
* **利用h数组的性质在O(n)的时间内求出储存排序后相邻后缀间的LCP数的组height**
* **利用LCP的性质将平凡LCP问题转化为height数组上的RMQ问题**