转自<http://hi.baidu.com/zfy0701/blog/item/f2278a0928991dca3bc763a0.html>

字符串算法通常是非常优雅的，尤其是后缀数组啊，前、后缀自动机啊，Trie，KMP，BM，BOM，Shift - Or，RK ......

我准备写写POJ上一些字符串问题的文章，第一篇是我认为在竞赛中字符串问题用处最大的一个工具：后缀数组。

**先帖点后缀数组的背景资料：**

**然后解决如下三个POJ上的问题：**

**PKU 2774 - Long Long Message  
PKU 1743 - Musical Theme  
PKU 3415 - Common Substrings**

<http://www.lirmm.fr/~rivals/ALGOSEQ/DOC/Manber-Myers-table-des-suffixes.pdf>  
这是后缀数组的原始论文，作者Manber是个超级大牛，许智磊那篇文章基本是翻译，E文不差的还是看看原文吧。

注意到这篇论文中提出的构造后缀数组的算法是O(nlogn)的，称倍增算法，现在也有线性算法，但多半情况实际性能似乎不如倍增算法。倒是现在有个O(nloglogn)的算法实际效率最高：论文见Linear time construction of suffix arrays，这里面提到了两种算法，其中理论线性的那个不如非线性的。

这篇论文作者是D. K. Kim, J. S. Sim, H. Park, and K. Park，网上没得，要去图书馆下，另外要区别其与DC3的那个线性算法，两者题目很像，作者不同，DC3空间消耗较大，过多基数排序导致常数非常大，据说有1000，远远大于log1000，

DC3的论文(Linear Work Suffix Array Construction)的地址：  
<http://www.cs.helsinki.fi/u/tpkarkka/publications/jacm05-revised.pdf>

下面这篇文章对现在的后缀数组算法有个很好的overview，见其intro部分  
<http://www.brics.dk/~cstorm/courses/StrAlg_e07/papers/SchurmannStoye2005_SuffixArray.pdf>

当然，就竞赛而言我觉得以上这些都是废话。

对于后缀数组，最重要的应该就是掌握倍增算法，更要熟练掌握height数组的性质和应用，要理解height数组，最好仔细看看对height数组性质的几个证明。

有些题目需要用RMQ问题的办法进行预处理以便O(1)时间求出任意两后缀的(LCP)Longest common prefix（必须这样才能做出来的题目我现在还没见到，且代码量会巨大）。但大多数情况一个height数组就够了！

后缀数组的用途大概有：匹配，求两串的最长公共子串，两串的公共子串的数量，单串的重复子串数列，单串的最长可重叠/不可重叠子串…

好：基本的东西介绍完了，现在来看题：

**PKU 2774 - Long Long Message**

这个题要求两个串的公共最长子串（不是子序列，即子串是连续的，子序列不连续——可用动态规划解决）。这道题不需要知道任意两个后缀的LCP值，只要连接两个字符串，求后缀数组与height数组，然后只要贪心地选择height数组中最大的就行（唯一约束就是对于一个height[i]，sa[i-1], sa[i]必须分属与两个不同串的位置，如只是对于单串，就没有这个限制）。

算法的正确性就是因为后缀数组是按字母顺序排序的，我们假设两个串的最长公共子串在后缀数组中不相临，那么必然有其他的子串插在这两串之间，那么插在其中的串显然拥有更大的LCP，与假设矛盾了。所以相邻是必然的，而height数组刚好记录的是后缀数组中相邻子串的LCP，所以很容易的可以求解了。

这个题也可以用后缀自动机，Oracle Factor自动机求解(非常高效)，见<http://hi.baidu.com/zfy0701/blog/item/d9fedbd14581113d9b5027ab.html>

**PKU 1743 - Musical Theme**

对于这道，要先把串转换一下，根据题目的性质，应该把输入得到的串前后相减得到方便求解的新的串——设其为s，再求该串s中最长的不重叠重复子串。由于不能重叠，导致height数组的最大值不一定是解，因为相邻两串可能会重叠。

此题用后缀数组也有两种解法，1：二分枚举答案，2：用栈线性扫描，主要说做法1，因为它更以理解些

现在我们假设-最长,不重叠,重复-子串的长度为len，那么在一个height数组中有一些hgt[i]会小于len，在这个i左右的两个子串，他们LCP是不可能大于或等于len的，这样，就可以吧height数组看做很多LCP >= len的段，我们在每一段中进行扫描，记录这一段中最大和最小的子串串索引（sa[x]），如果两者之和小于len，说明重叠了，否则就找到了一个可行解。

易证：如果该串存在len1的不重叠子串，且len1 > len2，则该串也存在长度len2的不重叠子串，解有连续性，所以我们在上面这个过程外我们可以二分枚举解。

对了，二分查找之前之前可先调用查找len = 4是否成立，不成立就直接return了。

这道题的关键点就是：判断解的时候对height数组进行分段处理，这个分段的思想与其说重要，还不说就是height数组的基本性质。类似的分段法在下题中再次用到。

更详细的过程见解题报告到这去下，上面说的应该比我清楚。

<http://www.oibh.org/bbs/viewthread.php?tid=21743>

它属于楼天成男人8题中的一道。

核心代码：

bool SearchAns(int \*sa, int \*hgt, int n, int len) {  
     int mi, ma, k, i = 20000, j = 0;  
     for (k = 1; k < n; k++) {  
         if (hgt[k] < len) {i = 20000; j = 0;}    //复原统计  
         else {  
              mi = min(sa[k], sa[k-1]);  
              ma = max(sa[k], sa[k-1]);  
              j = max(j, ma);  
              i = min(i, mi);  
              if (j - i >= len)   return true;  
         }  
     }  
     return false;  
}

线性扫描的做法：

借助栈扫描，使用栈思路上基本上同3415，但解题的核心思想也和上面的方法所说的一样，所以不详述，根据hgt数组的大小出入栈，扫描时记录每一段的sa[i]最大和最小值，在出栈的时候注意将值更新的上一段。

两种方法我都用了,这题的数据不是很强,后者之比前者快15ms, disscus中还有人说用线性扫描比二分慢的.

**PKU 3415 - Common Substrings**

求两个字符串不小于k的公共子串的个数，我下面简称k前缀子串（随便编的）。

这道题比较难，是一道非常经典的组合模式匹配问题，官方解题报告我看了一头雾水

后来在google上搜啊搜啊，终于搜到一篇论文，上面略微提到了单个串的不小于k的公共子串的个数的求法，  
<http://www-hto.usc.edu/people/tingchen/MyPublication/CPM-TrieSearch-1997.pdf>  
见其3.1的开头部分

然后又得到ImLazy兄的提示，艰难的写出了对于单个串的不小于k的公共子串的个数的程序。我把两个串拼接起来，求k前缀子串数量a，再分别求两个串k前缀子串的数量b, c，然后用a – b – c。其中求完拼接串的后缀数组，两个串的后缀数组可以由品拼接串的后缀数组直接拆分得到，不需要重新O(N \* log N)求。

先帖下求单个串的k前缀子串数量核心算法：

假设有m个子串在heigt数组中连续且值等于b，那么他们所产生的k前缀数量为 C(m, 2) \* (b – k + 1)（括号内的东西下称贡献量）

用一个栈扫描height(简称hgt)数组

**规则1、**如果hgt[i]大于栈顶元素，入栈，i++，循环continue  
**规则2、**如果等于栈顶元素，不入栈，i++，循环continue  
**规则3、**如果小于栈顶元素，计算i与栈顶元素之间的长度，按照入栈规则，这一段的hgt值都是一样的，可以容易地按上一段所说的规则进行统计。这里还要分两种情况  
**1、**hgt[i]小于栈顶下面的第一个元素，我是先把栈顶元素的hgt值“消”成和其下面的一个元素值一样，等于将他们合并，统计这部分的差值（下称相对贡献量），；  
**2、**hgt[i]大于栈顶下面的第一个元素，把栈顶元素的hgt值“消”和hgt[i]一样，其他同1

不继续扫描i，循环continue  
**规则4、**hgt[i]小于k且栈中有元素，执行统计，且执行出栈操作，不继续扫描i，循环continue  
**规则5、**hgt[i]小于k且栈中无元素，i++，循环continue

核心代码

int top = STACK\_BASE;  
stack[0] = 0;  
hgt[0] = hgt[n + 1] = k - 1;//尾部控制

while (i <= n + 1) {  
     hg1 = hgt[stack[top]];  
     if (hgt[i] < k && top == STACK\_BASE) i++;//如果还未进入统计的状态       
     else if (hgt[i] == hg1) i++;  
     else if (hgt[i] > hg1) stack[++top] = i++;  
     else {  
         interval = i - stack[top] + 1;  
         if (hgt[i] >= k && hgt[i] > hgt[stack[top-1]]) { //这种情况小结算,不出栈  
              fac = hg1 - hgt[i];  
              hgt[stack[top]] = hgt[i];  
         } else {//满足消栈情况  
              fac = hg1 - hgt[stack[top-1]];  
              top--;  
         }  
         counter += (long long)interval \* (interval - 1) / 2 \* fac;   //统计  
     }  
}

这样的做法813MS过了

一个星期后我想到了一次扫描的办法求两串的k前缀子串数量的方法，我的扫描规则还和上面一样，代码也和上面几乎完全一样。只不过主要统计的时候，直接用此段中 a 串的数量 \* b 串的数量 \* 相对贡献量就行了。

改进之后提高60MS，但是对比一下：

1 2766802 crazyb0y 45796K **406MS** G++ 1977B  
14 3437536(31) zfy0701 4404K **766MS** G++ 2964B

差距比较大，一问原来他是用后缀自动机做的。郁闷了，我对后缀自动机的认识仅停留在基于子串的简单串匹配算法上。