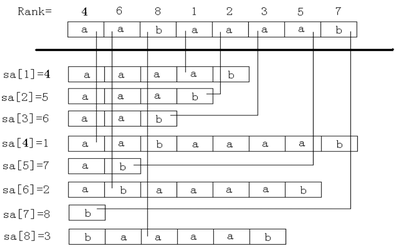
# 后缀数组

一、[字符串](http://www.nocow.cn/index.php?title=%E5%AD%97%E7%AC%A6%E4%B8%B2&action=edit&redlink=1)的大小比较： 关于字符串的大小比较，是指通常所说的 “ 字典顺序 ” 比较， 也就是对于两个字符串 u 、v ，令 i 从 1 开始顺次比较 u[i] 和 v[i] ，如果u[i]=v[i] 则令 i 加 1 ，否则若 u[i]<v[i] 则认为 u<v ，u[i]>v[i] 则认为 u>v，比较结束。如果 i>len(u) 或者 i>len(v) 仍比较不出结果，那么若 len(u)<len(v)则认为 u<v ， 若 len(u)=len(v) 则 认 为 u=v ，若 len(u)>len(v) 则 u>v 。

注：从字符串的大小比较的定义看，字符串s的所有后缀中任其中一对(u,v)不可能会相等，因为必要条件 len(u) ≠ len(v)不可能满足。所以任一字符串s中有len(s)个互不相同的后缀。我们可以将s的所有后缀排列，利用 后缀数组sa 与 名次数组rank 储存。

二、后缀数组sa：将s的n个后缀从小到大排序后将 排序后的后缀的**开头位置** 顺次放入sa中，则sa[i]储存的是排第i大的后缀的开头位置。简单的记忆就是“**排第几的是谁**”。  
三、名次数组rank：rank[i]保存的是suffix(i)｛后缀｝在所有后缀中从小到大排列的名次。则 若 sa[i]=j，则 rank[j]=i。简单的记忆就是“**你排第几**”。  
对于 后缀数组sa 与 名次数组rank ，有

rank[ sa[i] ]=i （这是很重要的一点，通过sa与rank的关系可以求出后缀数组）

[](http://www.nocow.cn/index.php/%E6%96%87%E4%BB%B6:20120630fc_sa&rank.png)

[http://www.nocow.cn/skins/common/images/magnify-clip.png](http://www.nocow.cn/index.php/%E6%96%87%E4%BB%B6:20120630fc_sa&rank.png)

字符串aabaaaab的sa数组与rank数组

由此可看出，后缀数组sa 与名次数组rank的关系为互逆关系。

[[编辑](http://www.nocow.cn/index.php?title=%E5%90%8E%E7%BC%80%E6%95%B0%E7%BB%84&action=edit&section=2)]倍增算法

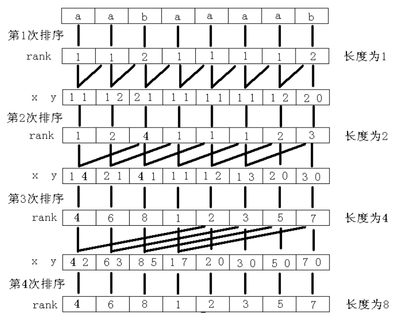
一、主要思路：[倍增](http://www.nocow.cn/index.php?title=%E5%80%8D%E5%A2%9E&action=edit&redlink=1)，s[*i*..*i* + 2*k* − 1]的排名通过s[*i*..*i* + 2*k*− 1 − 1]和s[*i* + 2*k*− 1..*i* + 2*k* − 1]的排名得到。

二、简要过程：已知每个长度为2*k*− 1的字符串的排名，则可作为每个长度为2*k*的字符串求排名的关键字xy，s[*i*..*i* + 2*k* − 1]第一关键字x为s[*i*..*i* + 2*k*− 1 − 1]的排名，第二关键字y为s[*i* + 2*k*− 1..*i* + 2*k* − 1]的排名。以字符串aabaaaab为例：

1. k=0,对每个字符开始的长度为20 = 1的子串进行排序，得到rank[1..8]={1,1,2,1,1,1,1,2}
2. k=1,对每个字符开始的长度为21 = 2的子串进行排序：由k=0的rank得关键字xy[1..8]={11,12,21,11,11,11,12,20}，得到rank[1..8]={1,2,4,1,1,1,2,3}
3. k=2,对每个字符开始的长度为22 = 4的子串进行排序：由k=1的rank得关键字xy[1..8]={14,21,41,11,12,13,20,30}，得到rank[1..8]={4,6,8,1,2,3,5,7}
4. k=3,对每个字符开始的长度为23 = 8的子串进行排序：由k=2的rank得关键字xy[1..8]={42,63,85,17,20,30,50,70}，得到rank[1..8]={4,6,8,1,2,3,5,7}

**注意：在排序过程中，rank[]可以有相同排名，但是sa[]排第几是没有相同的（就像Excel的排序，sa相当于编号，rank相当于排名）。这点可以从程序中体现。建议读者跟踪一下程序体会一下。**

**整个过程**如图：

[](http://www.nocow.cn/index.php/%E6%96%87%E4%BB%B6:20120630fc_sort.png)

[http://www.nocow.cn/skins/common/images/magnify-clip.png](http://www.nocow.cn/index.php/%E6%96%87%E4%BB%B6:20120630fc_sort.png)

倍增算法的计算过程

三、时间复杂度分析：每一趟的计数排序的时间复杂度是O(n)，排序的次数共log n次，总的时间复杂度为O(n log n)。

算法代码如下：

void sorting(int j)//基数排序

{

memset(sum,0,sizeof(sum));

for (int i=1; i<=s.size(); i++) sum[ rank[i+j] ]++;

for (int i=1; i<=maxlen; i++) sum[i]+=sum[i-1];

for (int i=s.size(); i>0; i--)

tsa[ sum[ rank[i+j] ]-- ]=i;

//对第二关键字计数排序，tsa代替sa为排名为i的后缀是tsa[i]

memset(sum,0,sizeof(sum));

for (int i=1; i<=s.size(); i++) sum[ rank[i] ]++;

for (int i=1; i<=maxlen; i++) sum[i]+=sum[i-1];

for (int i=s.size(); i>0; i--)

sa[ sum[ rank[ tsa[i] ] ]-- ]= tsa[i];

//对第一关键字计数排序,构造互逆关系

}

void get\_sa()

{

int p;

for (int i=0; i<s.size(); i++) trank[i+1]=s[i];

for (int i=1; i<=s.size(); i++) sum[ trank[i] ]++;

for (int i=1; i<=maxlen; i++) sum[i]+=sum[i-1];

for (int i=s.size(); i>0; i--)

sa[ sum[ trank[i] ]-- ]=i;

rank[ sa[1] ]=1;

for (int i=2,p=1; i<=s.size(); i++)

{

if (trank[ sa[i] ]!=trank[ sa[i-1] ]) p++;

rank[ sa[i] ]=p;

}//第一次的sa与rank构造完成

for (int j=1; j<=s.size(); j\*=2)

{

sorting(j);

trank[ sa[1] ]=1; p=1; //用trank代替rank

for (int i=2; i<=s.size(); i++)

{

if ((rank[ sa[i] ]!=rank[ sa[i-1] ]) || (rank[ sa[i]+j ]!=rank[ sa[i-1]+j ])) p++;

trank[ sa[i] ]=p;//空间要开大一点，至少2倍

}

for (int i=1; i<=s.size(); i++) rank[i]=trank[i];

}

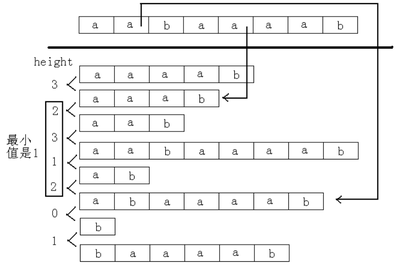
}

求出了rank和sa数组还不够，通常我们需要由rank与sa数组计算出一个辅助工具height数组——最长公共前缀(LCP)。

**height 数组**： 定义height[i]=suffix(sa[i-1]) 和 suffix(sa[i]) 的最长公共前缀，也就是排名相邻的两个后缀的最长公共前缀。

由height数组可得，对于j和k ，不妨设rank[j]<rank[k], 则有以下**性质**：suffix(j) 和 suffix(k) 的最长公共前缀为 height[rank[j]+1],height[rank[j]+2], height[rank[j]+3], … ,height[rank[k]] 中的最小值。

以"aabaaaab"为例，求后缀"abaaaab"和后缀"aaab"的最长公共前缀，如图，可见其最长公共前缀等于1。

[](http://www.nocow.cn/index.php/%E6%96%87%E4%BB%B6:20120630fc_height.png)

[http://www.nocow.cn/skins/common/images/magnify-clip.png](http://www.nocow.cn/index.php/%E6%96%87%E4%BB%B6:20120630fc_height.png)

计算后缀"abaaaab"和后缀"aaab"的最长公共前缀

所以说，计算最长公共前缀是一个典型的[RMQ](http://www.nocow.cn/index.php/RMQ)问题。

如果直接按照sa的顺序一个一个求解，每一次比较最坏的时间复杂度是O(len(s))，一共要比较len(s)次，所以时间复杂度是*O*(*len*(*s*)2)。这样求height数组是非常慢的，而且没有用到之前所说height数组的性质。

那么，如何**高效**地计算后缀间的最长公共前缀呢？

当然是使用之前所说的性质。定义h[i]为suffix(i)和前一名次后缀的最长公共前缀{sa[ rank[ i ]-1 ]}。由性质可得，

h[i] \ge h[i-1]-1。

简单的证明如下：设suffix(k)是排在suffix(i-1)前一位的后缀，则它们的最长公共前缀显然是h[i-1]。那么，suffix(k+1)显然将排在suffix(i)的前面。并且，suffix(k+1)&suffix(i) 相对于 suffix(k)&suffix(i-1)来说就是同时去掉了第一位，即少了一位的匹配数。所以suffix(i)和前一名次后缀的最长公共前缀至少是h[i-1]-1。

显然，我们可以按照h数组的顺序计算height。时间复杂度分析：求一次height后位数-1，一共有len(s)个后缀，所以只能退len(s)次，也就是说，求解的时间复杂度是O(len(s))。

算法代码如下：

void get\_height()

{

for (int i=1,j=0; i<=s.size(); i++)//用j代替上面的h数组

{

if (rank[i]==1) continue;

for (; s[i+j-1]==s[ sa[ rank[i]-1 ]+j-1 ]; ) j++;//注意越界之类的问题

height[ rank[i] ]=j;

if (j>0) j--;

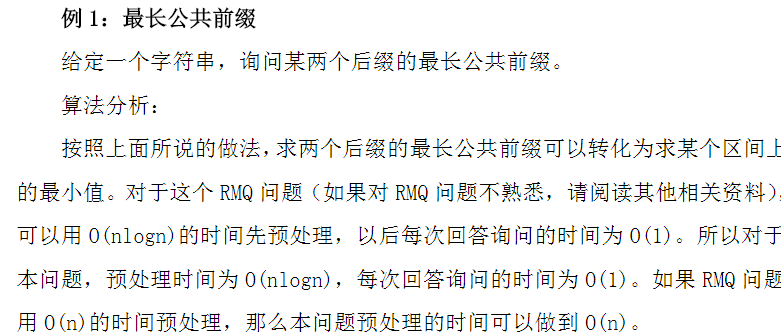
}

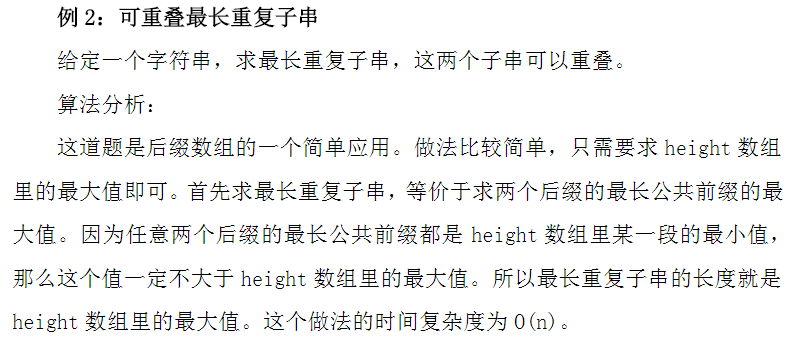
}

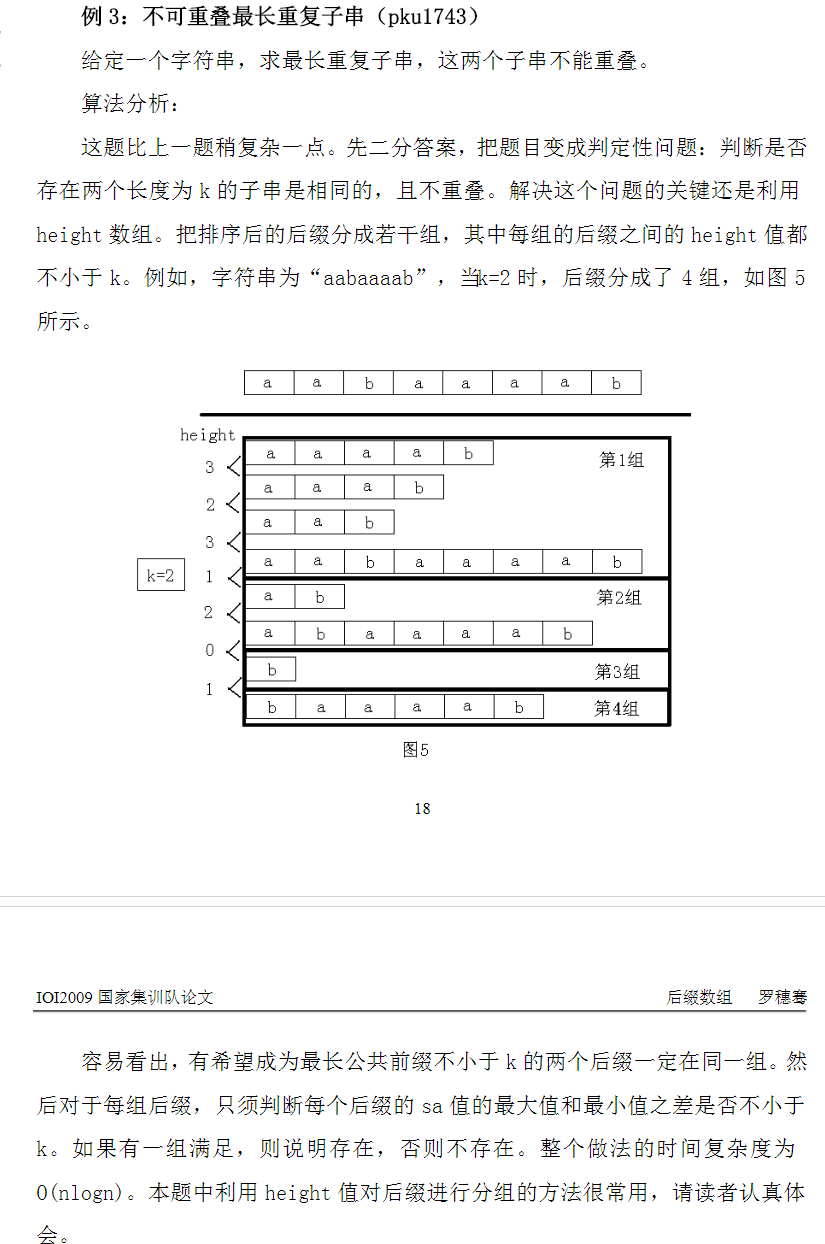
Codeforce gym 101194F

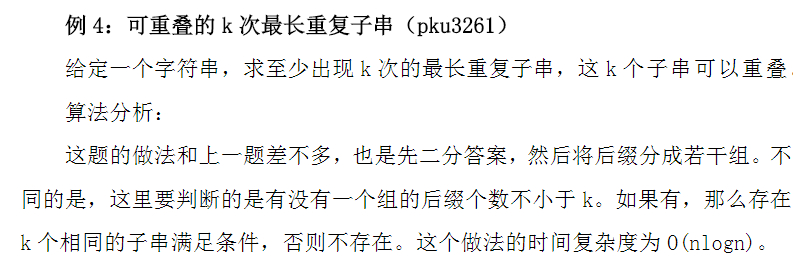
题意：给n个串，要求一个这样的子串，它满足长度最短的情形下字典序最小，使得它在第一个字符串中出现过且在其他n-1个串中没有出现过。保证n串总长度小于25w

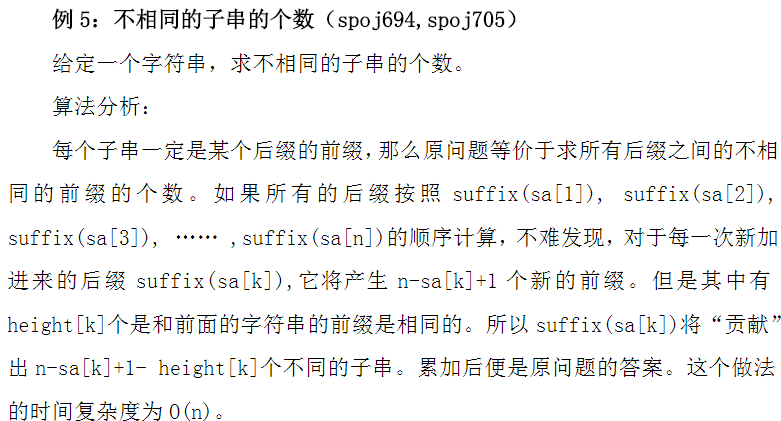
题解:把n个字符串用n-1个不相同的字符连起来,跑一遍后缀数组，求出height,然后对于每一个属于第一个串，检查height[i]和height[i+1]，要想以sa[i]开头的字符串不在其他字符串中出现，则它至少需要len=max(height[i],height[i+1])+1的长度，我们可以check一下当前位置加上len还在第一个字符串中么。所以我们扫一遍找出最短长度，然后再扫一遍碰到第一个长度==最短长度输出即可（后缀数组自动按字典序排好序了）

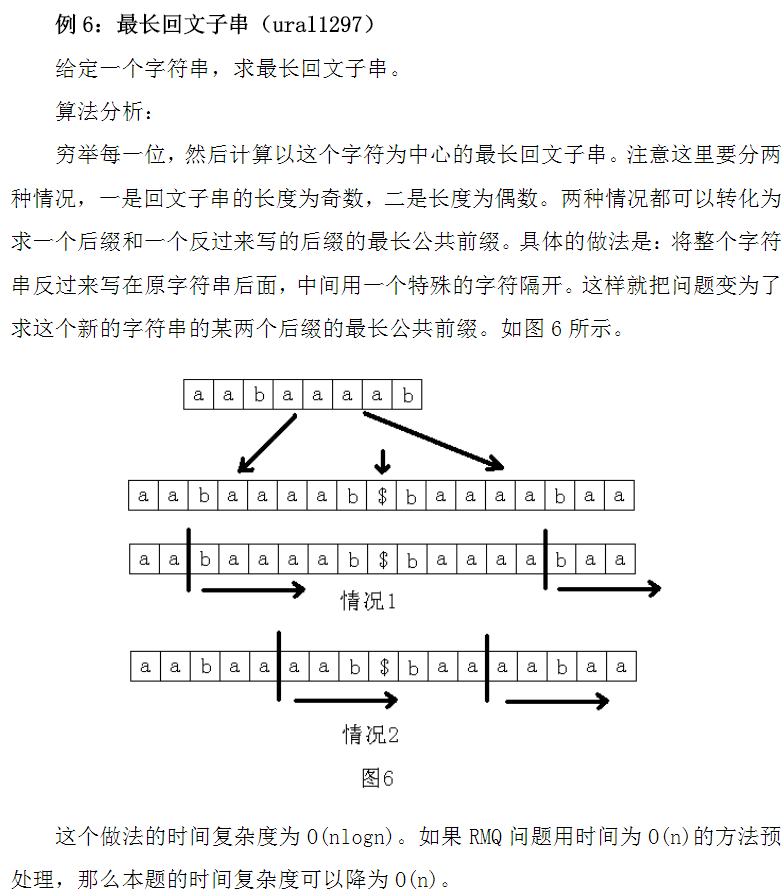




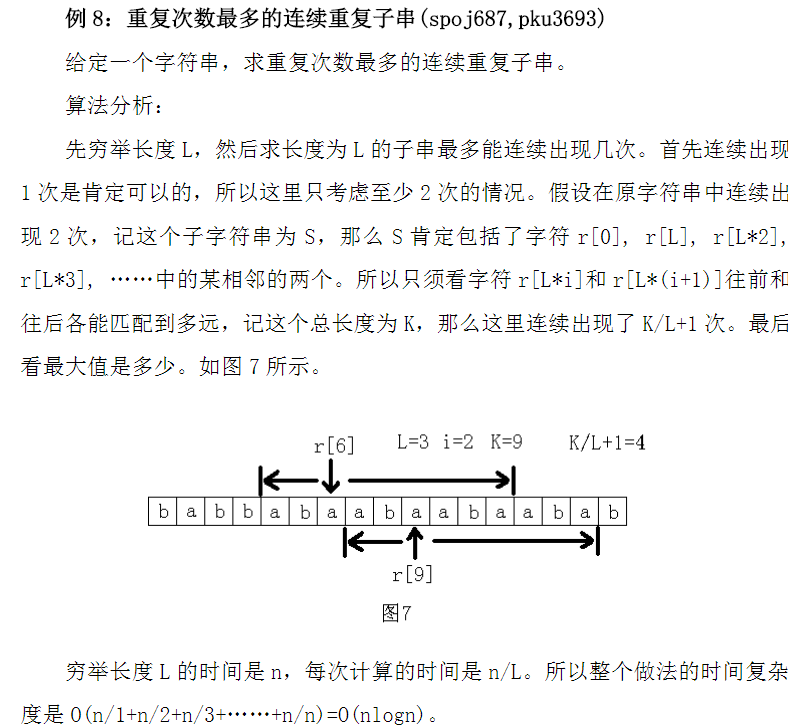


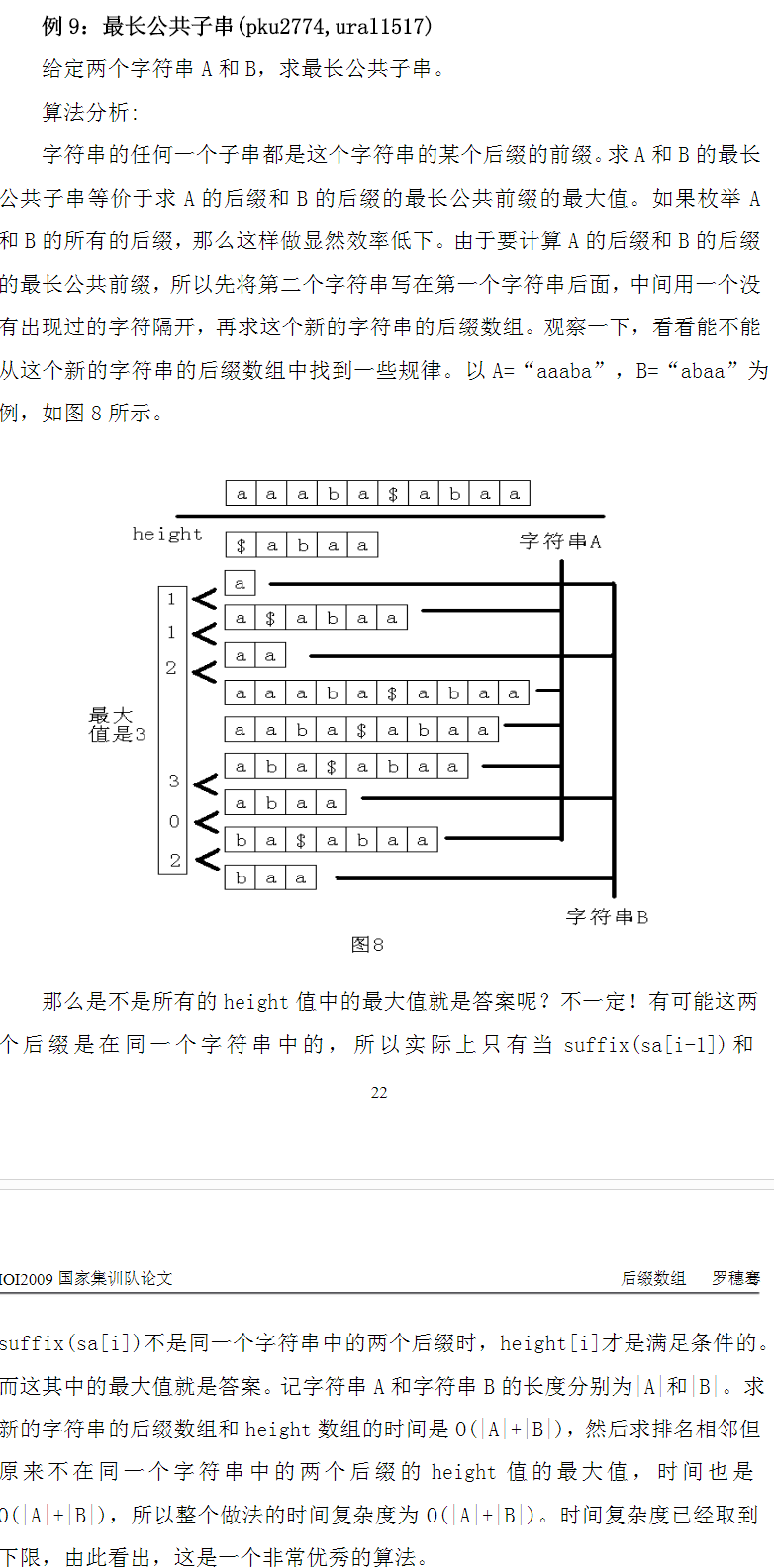


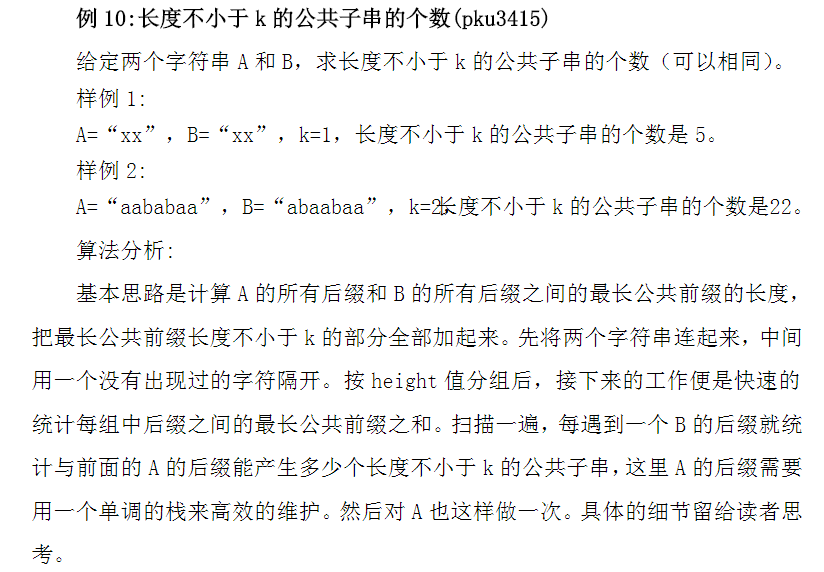


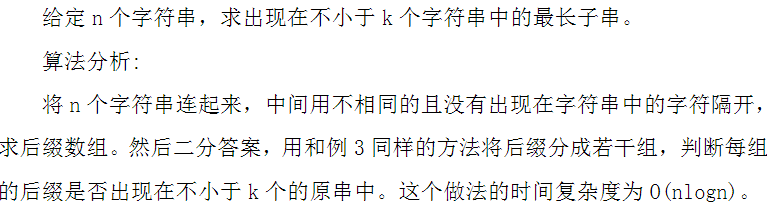


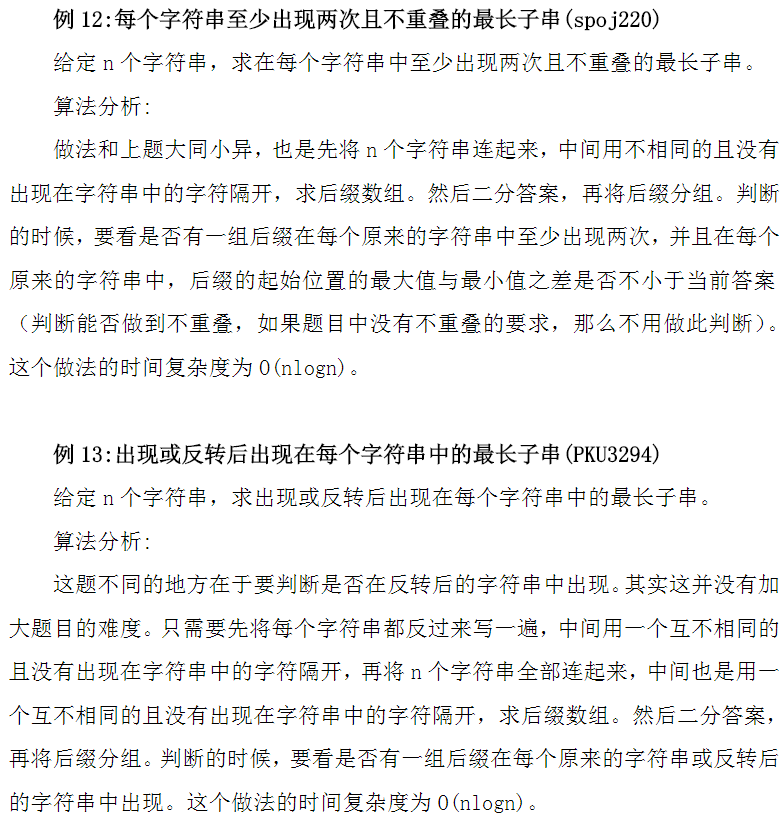












例14：求第K大子串（hdu5008）

题目：  
food所有的组成情况如下：  
d             
f  
fo  
foo  
food  
  
按照字母顺序排列如上所示，现要求第k大的子串，例如第8大子串为oo

o  
od  
oo  
ood

solve:

首先可以知道height本身就是按字典序来弄的,对于sa[i-1]和sa[i]而言只要减去

它们的公共部分即height[i],就是不同的子串.

可以借此处理出来[1,n]即到字典序第i大的后缀时总共有多少个不同的子串.然后

利用二分查找就能找到位置

还要注意就是当有多个的时候输出最小的位置,可以判断当前子串是否是公共前缀的

一部分.往后枚举找出最小的位置即可

例15：求出现次数正好为k的子串（hdu6194）

题目：如标题

方法：求出height后丢进线段树倒着一遍Ans=Ans+max(0,ans-height[i+1]);

意思是查当前i到i-k最多有多少位的前缀相同，并且要减去i+1个串与i串相同的部分这样就求出了出现次数大于K的串个数，再求一个K+1一减即可