SA

calabash_boy

2022 年 4 月 28 日

后缀数组

定义

后缀 $S[i]$：$S[i] = S[i, |S|]$。

字典序：对于两个字符串 S 和 T，从左到右找到第一个不同的字母，谁的字母小，谁的字典序就小。特殊的，如果 S 为 T 的前缀，认为 $S < T$。即空字符最小。

后缀数组

定义

后缀 $S[i]$：$S[i] = S[i, |S|]$。
字典序：对于两个字符串 S 和 T，从左到右找到第一个不同的字母，谁的字母小，谁的字典序就小。特殊的，如果 S 为 T 的前缀，认为 $S < T$。即空字符最小。
后缀排序：将所有后缀 $S[i]$ 看作独立的串，放在一起按照字典序进行升序排序。
后缀排名 $rk[i]$：$rk[i]$ 表示后缀 $S[i]$ 在后缀排序中的排名，即他是第几小的后缀。
后缀数组 $sa[i]$：$sa[i]$ 表示排名第 i 小的后缀。
$rk[sa[i]] = i$

求后缀数组 1

Naive Sort 法

用二分 Hash 写一个比较字典序的 cmp 函数，然后直接调 sort。
复杂度 $O(n \log^2 n)$

求后缀数组 1

Naive Sort 法

用二分 Hash 写一个比较字典序的 cmp 函数，然后直接调 sort。
复杂度 $O(n \log^2 n)$
Hash 检测次数高达 $n \log^2 n$，非常容易冲突。且复杂度过高。

求后缀数组 2

前缀倍增法

思路：将比较字典序的二分求 LCP 转化为倍增求 LCP。

求后缀数组 2

前缀倍增法

思路：将比较字典序的二分求 LCP 转化为倍增求 LCP。
首先等效的认为在字符串的末尾增添无限个空字符 $\emptyset$。

定义 $S(i,k)=S[i,i+2^k-1]$，即以 i 位置开头，长度为 2^k 的子串。
后缀 $S[i]$ 与 $S[j]$ 的字典序关系等价于 $S(i,\infty)$ 与 $S(j,\infty)$ 的字典序关系。
事实上，只需要将 $S(i,\lceil log_2 n\rceil)$，$i=1,2,\cdots,n$ 排序即可。

求后缀数组 2

前缀倍增法

于是便可以倍增的进行排序，假设当前已经得到了 $S(i,k)$ 的排序结果，即 $rk[S(i,k)]$ 与 $sa[S(i,k)]$，思考如何利用它们排序 $S(i,k+1)$。

求后缀数组 2

前缀倍增法

于是便可以倍增的进行排序，假设当前已经得到了 $S(i,k)$ 的排序结果，即 $rk[S(i,k)]$ 与 $sa[S(i,k)]$，思考如何利用它们排序 $S(i,k+1)$。

由于 $S(i,k+1)$ 是由 $S(i,k)$ 和 $S(i+2^k,k)$ 前后拼接而成。
因此比较 $S(i,k+1)$ 与 $S(j,k+1)$ 字典序可以转化为先比较 $S(i,k)$ 与 $S(j,k)$，再比较 $S(i+2^k,k)$ 与 $S(j+2^k,k)$。

因此可以将 $S(i,k+1)$ 看作一个两位数，高位是 $rk[S(i,k)]$，低位是 $rk[S(i+2^k,k)]$。

两位数的排序

一位数的排序

有一组一位数，将他们排序只需要一个桶数组辅助就可以完成。
复杂度 $O(n)$。

两位数的排序

一位数的排序

有一组一位数，将他们排序只需要一个桶数组辅助就可以完成。
复杂度 $O(n)$。

两位数的排序

有一组两位数，可以用 $\{A_i, B_i\}$ 表示，将他们排序时，需要先按照高位排序，高位相同时，按照低位排序。此过程为基数排序。
复杂度 $O(n)$。

基数排序

基数排序

基数排序的过程可以简单理解：

1. 首先统计 $cntA[x]$，表示高位 $A = x$ 的数字有多少个。于是可以确定第 i 个数字的最终排名一定在范围 $(\sum_{x=1}^{x<A_i} cntA[x], \sum_{x=1}^{x\le A_i} cntA[x]]$ 内，记为 $(SumA[A_i - 1], SumA[A_i]]$。即完成了数字的分块，以及完成了块与块之间的排序。

2. 接下来需要确定每个块 $(A = x)$ 内数字的顺序，问题变成排序一位数：按照低位 B 从大到小，依次地领取排名 $SumA[x], SumA[x] - 1, \cdots, SumA[x-1] + 1$。因此这一步需要事先对把所有数字按照低位排序。

求后缀数组 2

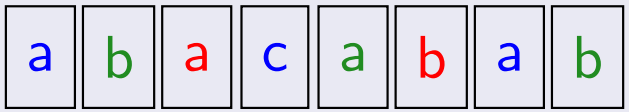
前缀倍增法

复杂度分析：
算法需要运行 $\log n$ 轮。每一轮使用基数排序，复杂度为 $O(n)$。
整体复杂度为 $O(n \log n)$。

求后缀数组 3

DC3 法

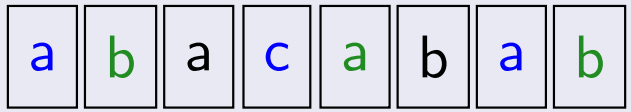
将所有后缀按照开头下标 Mod 3 分类：后缀 1，后缀 2，后缀 0。
首先将所有后缀 1，后缀 2排序，然后加入后缀 0归并，得到完整的排序。



求后缀数组 3

DC3 法

1. 排序后缀 1，后缀 2

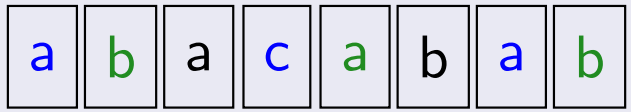


注意到所有后缀 1都是 **S[1]** 的后缀，且开头位置下标差都是 3 的倍数。
同理，所有后缀 2都是 **S[2]** 的后缀。

求后缀数组 3

DC3 法

1. 排序后缀 1，后缀 2



注意到所有后缀 1都是 **S[1]** 的后缀，且开头位置下标差都是 3 的倍数。
同理，所有后缀 2都是 **S[2]** 的后缀。
因此我们可以用如下方法处理，进行递归：
首先，如果 $S[1]$ 或者 $S[2]$ 的长度不是 3 的倍数，则在其后分别补 0 到 3 的倍数长度。如果是 3 的倍数，依然需要补 3 个 0。

求后缀数组 3

DC3 法

1. 排序后缀 1，后缀 2

a	b	a	c	a	b	a	b

注意到所有后缀 1都是 **S[1]** 的后缀，且开头位置下标差都是 3 的倍数。
同理，所有后缀 2都是 **S[2]** 的后缀。
因此我们可以用如下方法处理，进行递归：
首先，如果 $S[1]$ 或者 $S[2]$ 的长度不是 3 的倍数，则在其后分别补 0 到 3 的倍数长度。如果是 3 的倍数，依然需要补 3 个 0。
然后将 $S[1]$ 和 $S[2]$ 前后拼接。

a	b	a	**c**	a	b	**a**	b	∅	**b**	a	c	**a**	b	a	**b**	∅	∅

求后缀数组 3

DC3 法

1. 排序后缀 1，后缀 2

a	b	a	c	a	b	a	b

注意到所有后缀 1都是 **S[1]** 的后缀，且开头位置下标差都是 3 的倍数。
同理，所有后缀 2都是 **S[2]** 的后缀。
因此我们可以用如下方法处理，进行递归:
首先，如果 $S[1]$ 或者 $S[2]$ 的长度不是 3 的倍数，则在其后分别补 0 到 3 的倍数长度。如果是 3 的倍数，依然需要补 3 个 0。
然后将 $S[1]$ 和 $S[2]$ 前后拼接。

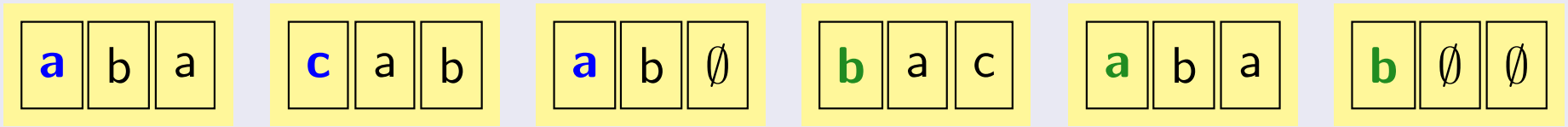
a	b	a	**c**	a	b	**a**	b	$\emptyset$	**b**	a	c	**a**	b	a	**b**	$\emptyset$	$\emptyset$

问题转化为排序 T 的所有 $Mod3 = 1$ 位置后缀。

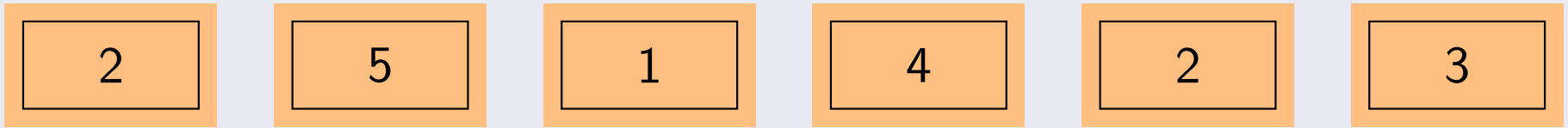
求后缀数组 3

DC3 法

将每 3 个位置打包在一起，打包之后每个块可以看作一个三位数。



使用基数排序将这些三位数排序。



因此排序所有 T 的所有 $Mod3 = 1$ 位置后缀转变为对 $T' = 251423$ 进行后缀排序。

求后缀数组 3

DC3 法

将每 3 个位置打包在一起，打包之后每个块可以看作一个三位数。

a b a	c a b	a b ∅	b a c	a b a	b ∅ ∅

使用基数排序将这些三位数排序。

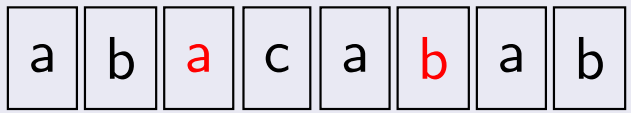
2	5	1	4	2	3

因此排序所有 T 的所有 $Mod3 = 1$ 位置后缀转变为对 $T' = 251423$ 进行后缀排序。
进行递归。

求后缀数组 3

DC3 法

2. 对所有后缀 0排序。



每个后缀 0可以表示为 $S[3k, n] = S[3k] + S[3k+1, n]$。即变成一个字母和一个后缀 1拼接而成，可以看作一个两位数，可以使用基数排序。

求后缀数组 3

DC3 法

3. 将排序过的后缀 0与排序过的后缀12进行归并。

求后缀数组 3

DC3 法

3. 将排序过的后缀 0与排序过的后缀12进行归并。

3.1. 如何比较后缀 0与后缀 1:

$S[3k, n] = S[3k] + S[3k+1, n]$

$S[3p+1, n] = S[3p+1] + S[3p+2, n]$

后缀 1与后缀 2的相对顺序已知，因此可以看作两位数 $O(1)$ 完成比较。

求后缀数组 3

DC3 法

3. 将排序过的后缀 0与排序过的后缀12进行归并。

3.1. 如何比较后缀 0与后缀 1:

$S[3k, n] = S[3k] + S[3k+1, n]$

$S[3p+1, n] = S[3p+1] + S[3p+2, n]$

后缀 1与后缀 2的相对顺序已知，因此可以看作两位数 $O(1)$ 完成比较。

3.2. 如何比较后缀 0与后缀 2:

$S[3k, n] = S[3k] + S[3k+1] + S[3k+2, n]$

$S[3p+2, n] = S[3p+2] + S[3p+3] + S[3p+4, n]$

同理，可以看作三位数 $O(1)$ 完成比较。

求后缀数组 3

DC3 法

复杂度 $T(n) = T(\frac{2n}{3}) + O(n)$。
于是复杂度为

$$O(n + \frac{2}{3}n + (\frac{2}{3})^2 n + \cdots)) = O(\frac{n}{1 - \frac{2}{3}}) = O(3n)$$

由于基数排序天然自带大常数，整个算法的常数会达到接近 20。实际表现只比倍增优秀一点点。

求后缀数组 4

SA-IS

略。小常数 $O(n)$。

后缀数组性质

LCP

设有一组排序过的字符串 $A = [A_1, A_2, \cdots, A_n]$。
如何快速的求任意 A_i 与 A_j 的 LCP ?

后缀数组性质

LCP

设有一组排序过的字符串 $A = [A_1, A_2, \cdots, A_n]$。
如何快速的求任意 A_i 与 A_j 的 LCP ?

“区间可加性”：对于任意的 $k \in [i, j]$，$LCP(A_i, A_j) = LCP(LCP(A_i, A_k), LCP(A_k, A_j)) = min(LCP(A_i, A_k), LCP(A_k, A_j))$。

后缀数组性质

LCP

设有一组排序过的字符串 $A = [A_1, A_2, \cdots, A_n]$。
如何快速的求任意 A_i 与 A_j 的 LCP ?

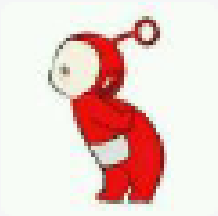
“区间可加性”：对于任意的 $k \in [i, j]$，$LCP(A_i, A_j) = LCP(LCP(A_i, A_k), LCP(A_k, A_j)) = min(LCP(A_i, A_k), LCP(A_k, A_j))$。
进而，
$LCP(A_i, A_j) = min(LCP(A_i, A_{i+1}), LCP(A_{i+1}, A_{i+2}), \cdots, LCP(A_{j-1}, A_j))$。
证明：

后缀数组性质

LCP

设有一组排序过的字符串 $A = [A_1, A_2, \cdots, A_n]$。
如何快速的求任意 A_i 与 A_j 的 LCP ?

“区间可加性”：对于任意的 $k \in [i, j]$，$LCP(A_i, A_j) = LCP(LCP(A_i, A_k), LCP(A_k, A_j)) = min(LCP(A_i, A_k), LCP(A_k, A_j))$。
进而，
$LCP(A_i, A_j) = min(LCP(A_i, A_{i+1}), LCP(A_{i+1}, A_{i+2}), \cdots, LCP(A_{j-1}, A_j))$。
证明：

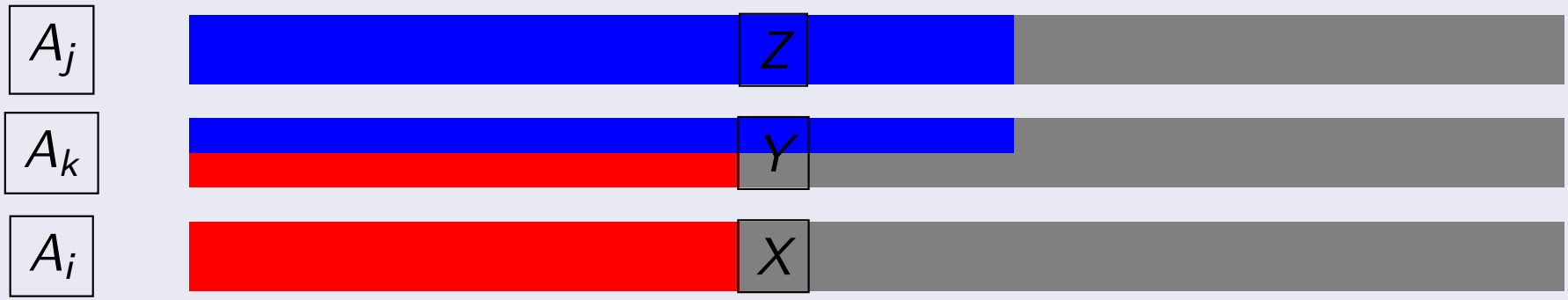


emo

我直接写《请读者自证》

证明

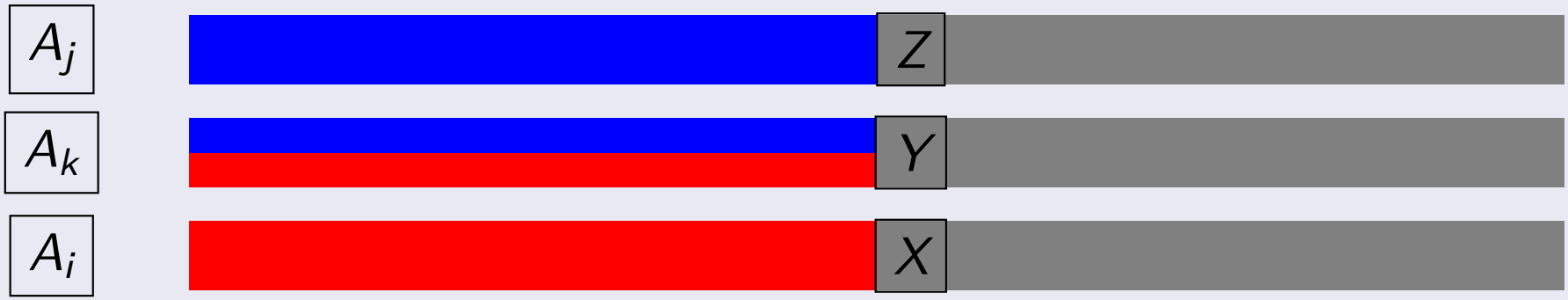
1. $LCP(A_i, A_k) \neq LCP(A_k, A_j)$



由于 $X \neq Y$，$Y = Z$，因此 $X \neq Z$。即
$LCP(A_i, A_j) = LCP(A_i, A_k) = min(LCP(A_i, A_k), LCP(A_k, A_j))$。

证明

2. $LCP(A_i, A_k) = LCP(A_k, A_j)$



已知 $X \neq Y$ 且 $Y \neq Z$。由于字典序 $A_i < A_k < A_j$，因此 $X < Y < Z$，故 $X \neq Z$，因此结论依然成立。

Height 数组

Height 数组

$Height[i]$ 为后缀 i 与排名在他前面一个的后缀的 LCP，即：
$Height[i] = LCP(S[i, n], S[sa[rk[i] - 1], n])$。
有了 $Height[i]$ 数组之后，利用上一节的性质，任意两个后缀的 LCP 就变为区间最小值查询。

Height 数组

Height 数组

$Height[i]$ 为后缀 i 与排名在他前面一个的后缀的 LCP，即：
$Height[i] = LCP(S[i, n], S[sa[rk[i] - 1], n])$。
有了 $Height[i]$ 数组之后，利用上一节的性质，任意两个后缀的 LCP 就变为区间最小值查询。
如何求 $Height$ 呢？

求 Height 数组

Height 数组

结论：$Height[i] \geq Height[i-1] - 1$。

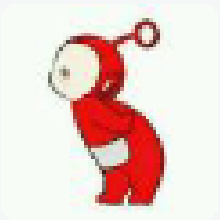
证明：

求 Height 数组

Height 数组

结论：$Height[i] \geq Height[i-1] - 1$。

证明：



emo

我直接写《请读者自证》

证明 $Height[i] \geq Height[i-1]-1$

1. 若 $Height[i-1] \leq 1$

$Height[i-1] = LCP(S[i-1,n], S[K1 = sa[rk[i-1]-1], n])$。
$Height[i] = LCP(S[i,n], S[K2 = sa[rk[i]-1], n])$。
1. 显然，$Height[i-1] = 1/0$，则结论 $Height[i] \geq 0$ 成立。

证明 $Height[i] \geq Height[i-1] - 1$

2. 若 $Height[i-1] > 1$

$S[K1, n]$

$S[i-1, n]$

证明 $Height[i] \geq Height[i-1]-1$

2. 若 $Height[i-1] > 1$

$S[K1, n]$

$S[i-1, n]$

则去掉首字母后，红色部分依然相等，且字典序关系不变。

$S[K1+1, n]$

$S[i, n]$

即：$S[K1+1, n] < S[i, n]$，且 $LCP(S[K1+1, n], S[i, n]) = Height[i-1]-1$。
由于 $S[K1+1, n] \leq S[K2, n] < S[i, n]$，因此应用“区间可加性”：

$$
\begin{aligned}
Height[i-1]-1 &= LCP(S[K1+1, n], S[i, n]) \\
&= min(LCP(S[K1+1, n], S[K2, n]), LCP(S[K2, n], S[i, n])) \\
&= min(LCP(S[K1+1, n], S[K2, n]), Height[i])
\end{aligned}
$$

因此结论成立。

求 Height 数组

求 Height

首先让 $Height[i]$ 继承 $Height[i-1]-1$，然后向后暴力延申。
利用势能分析，易得：势能增长量 $=$ 势能减少量 $=O(n)$。
实际使用时常令 $H[i]=Height[sa[i]]$（$H[rk[i]]=Height[i]$），$H[i]$ 表示排名为 i 与 $i-1$ 的串的 LCP。

模板题

某谷 3809

给出一个字符串，按字典序大小输出后缀位置。

Height 应用：求本质不同子串

SPOJ SUBST1

给出一个字符串，长度不超过 $50,000$，求本质不同子串数量。

牛客竞赛
AC.NOWCODER.COM

Height 应用：求本质不同子串

SPOJ SUBST1

给出一个字符串，长度不超过 $50,000$，求本质不同子串数量。

题解

答案 $= \sum n - sa[i] + 1 - H[i]$。

Height 应用：求本质不同子串

SPOJ SUBST1

给出一个字符串，长度不超过 $50,000$，求本质不同子串数量。

题解

答案 $= \sum n - sa[i] + 1 - H[i]$。
按字典序从小到大枚举所有后缀，统计有多少个新出现的前缀即可。
对于排名第 i 的后缀 $S[sa[i], n]$，共有 $n - sa[i] + 1$ 个前缀，其中有 $H[i]$ 个前缀同时出现在前一个排名的后缀 $S[sa[i-1], n]$ 中，因此减掉即可。

Height 应用：求本质不同子串

SPOJ SUBST1

给出一个字符串，长度不超过 $50,000$，求本质不同子串数量。

题解

答案 $= \sum n - sa[i] + 1 - H[i]$。
按字典序从小到大枚举所有后缀，统计有多少个新出现的前缀即可。
对于排名第 i 的后缀 $S[sa[i], n]$，共有 $n - sa[i] + 1$ 个前缀，其中有 $H[i]$ 个前缀同时出现在前一个排名的后缀 $S[sa[i-1], n]$ 中，因此减掉即可。
上述证明是不完整的，还需要证明所有在 $S[sa[i], n]$ 中出现，但没有在 $S[sa[i-1], n]$ 中出现的前缀，他们在所有更小排名的后缀串也都没有出现。

Height 应用：求本质不同子串

SPOJ SUBST1

给出一个字符串，长度不超过 $50,000$，求本质不同子串数量。

题解

答案 $= \sum n - sa[i] + 1 - H[i]$。
按字典序从小到大枚举所有后缀，统计有多少个新出现的前缀即可。
对于排名第 i 的后缀 $S[sa[i], n]$，共有 $n - sa[i] + 1$ 个前缀，其中有 $H[i]$ 个前缀同时出现在前一个排名的后缀 $S[sa[i-1], n]$ 中，因此减掉即可。
上述证明是不完整的，还需要证明所有在 $S[sa[i], n]$ 中出现，但没有在 $S[sa[i-1], n]$ 中出现的前缀，他们在所有更小排名的后缀串也都没有出现。

例题 1

牛客 17141

给出一个字符串，只由 abc 三种字母构成，求有多少置换意义下本质不同子串。
如果两个串在 {a,b,c} 的某种置换作用下相等，则认为是本质相同串。

例题 1

牛客 17141

给出一个字符串，只由 abc 三种字母构成，求有多少置换意义下本质不同子串。
如果两个串在 {a,b,c} 的某种置换作用下相等，则认为是本质相同串。

题解

由于字符集很小，可以枚举所有的 6 种置换，于是每种本质不同子串都会出现 6 种不同的版本。

例题 1

牛客 17141

给出一个字符串，只由 abc 三种字母构成，求有多少置换意义下本质不同子串。
如果两个串在 {a,b,c} 的某种置换作用下相等，则认为是本质相同串。

题解

由于字符集很小，可以枚举所有的 6 种置换，于是每种本质不同子串都会出现 6 种不同的版本。
然而有一个例外是：如果是全 a(b/c) 串，则在 6 种置换的作用下，只会出现 3 个不同版本。
因此本题需要统计本质不同子串，以及本质不同的全 a(b/c) 串。

例题 1

牛客 17141

给出一个字符串，只由 abc 三种字母构成，求有多少置换意义下本质不同子串。
如果两个串在 {a,b,c} 的某种置换作用下相等，则认为是本质相同串。

题解

由于字符集很小，可以枚举所有的 6 种置换，于是每种本质不同子串都会出现 6 种不同的版本。
然而有一个例外是：如果是全 a(b/c) 串，则在 6 种置换的作用下，只会出现 3 个不同版本。
因此本题需要统计本质不同子串，以及本质不同的全 a(b/c) 串。
可以将 6 个置换串，用互不相同的连接符 (例如：'z'+1 到'z'+5) 首尾拼接，求出 cntA 为本质不同全 a(b/c) 串个数，cntB 为至少包含两种不同字母的本质不同串（且不包含连接符）。
答案 $= \frac{cntA}{3} + \frac{cntB}{6}$。

例题 2

POJ3415

给出两个字符串 S 和 T，求有多少个长度大于 K 的公共子串（区间）。

牛客竞赛
AC.NOWCODER.COM

例题 2

POJ3415

给出两个字符串 S 和 T，求有多少个长度大于 K 的公共子串（区间）。

题解

用特殊字符连接两个串，进行后缀排序，得到 H 数组。
于是问题转化为对于所有区间 $[l, r]$，求 $max(0, min(H[l..r]) - K)$ 的和。
从左到右扫描维护单调栈，或者分治，或者离线数据结构都可。

例题 3

题意

给出两个串 S 和 T，求最长公共子串长度。

例题 3

题意

给出两个串 S 和 T，求最长公共子串长度。

题解

用特殊字符连接两个串，进行后缀排序，得到 H 数组。
求每一个 T 的后缀与所有的 S 后缀的最大 LCP，取最大值即为答案。
枚举每个属于 T 的后缀，向左向右寻找第一个属于 S 的后缀 S_l 和 S_r，
求所有 $max(LCP(T, S_l), LCP(T, S_r))$ 的最大值即可。

例题 4

题意

给出两个字符串 S 和 T，求有多少**本质不同公共子串**。

例题 4

题意

给出两个字符串 S 和 T，求有多少**本质不同公共子串**。

题解

用特殊字符连接两个串，进行后缀排序，得到 H 数组。
从左到右扫描属于 T 串的后缀，用上题的放法求每个 T 串的后缀与所有 S 串后缀的最大 LCP，记为 $MXLen$。
由于需要统计本质不同，需要找到前一个排名的属于 T 串的后缀，求出他们的 LCP 用于去重，记为 $MNLen$。
则答案 $= \sum MXLen - MNLen$

例题 5

WF 2019 First of Her Name

给出一棵反向的 Trie 树，每个点表示一个人的名字：从该点走到根边上的字符首尾拼接形成。
有若干次询问，每次询问给出一个字符串，问他是多少个人名字的前缀。

例题 5

WF 2019 First of Her Name

给出一棵反向的 Trie 树，每个点表示一个人的名字：从该点走到根边上的字符首尾拼接形成。
有若干次询问，每次询问给出一个字符串，问他是多少个人名字的前缀。

题解

之前讲过离线 AC 自动机的做法，今天讲在线树上 SA 的做法。

例题 5

WF 2019 First of Her Name

给出一棵反向的 Trie 树，每个点表示一个人的名字：从该点走到根边上的字符首尾拼接形成。
有若干次询问，每次询问给出一个字符串，问他是多少个人名字的前缀。

题解

之前讲过离线 AC 自动机的做法，今天讲在线树上 SA 的做法。
普通的 SA 是排序一个字符串的所有后缀，树上 SA 则是排序所有 Trie 树节点表示的字符串。

例题 5

WF 2019 First of Her Name

给出一棵反向的 Trie 树，每个点表示一个人的名字：从该点走到根边上的字符首尾拼接形成。
有若干次询问，每次询问给出一个字符串，问他是多少个人名字的前缀。

题解

之前讲过离线 AC 自动机的做法，今天讲在线树上 SA 的做法。
普通的 SA 是排序一个字符串的所有后缀，树上 SA 则是排序所有 Trie 树节点表示的字符串。
排序原理完全一样：使用 ST 表进行倍增 + 基数排序。

例题 5

WF 2019 First of Her Name

给出一棵反向的 Trie 树，每个点表示一个人的名字：从该点走到根边上的字符首尾拼接形成。
有若干次询问，每次询问给出一个字符串，问他是多少个人名字的前缀。

题解

之前讲过离线 AC 自动机的做法，今天讲在线树上 SA 的做法。
普通的 SA 是排序一个字符串的所有后缀，树上 SA 则是排序所有 Trie 树节点表示的字符串。
排序原理完全一样：使用 ST 表进行倍增 + 基数排序。
回答询问时，进行暴力二分求 lower_bound 和 upper_bound 即可。
复杂度 $O(\Sigma \log n)$。