字符串进阶

TQX

2024 年 8 月 14 日

Manacher

manacher 算法能在线性时间内计算一个串的所有回文子串，用各个对称中心上回文串的最大长度表示。它的思路类似于扩展 KMP。

Manacher

manacher 算法能在线性时间内计算一个串的所有回文子串，用各个对称中心上回文串的最大长度表示。它的思路类似于扩展 KMP。

洛谷 P3805 【模板】manacher

给定字符串 s，求 s 最长回文子串的长度。$|s| \leq 1.1 \times 10^7$。

Manacher

manacher 算法能在线性时间内计算一个串的所有回文子串，用各个对称中心上回文串的最大长度表示。它的思路类似于扩展 KMP。

洛谷 P3805 【模板】manacher

给定字符串 s，求 s 最长回文子串的长度。$|s| \leq 1.1 \times 10^7$。

对于这样的问题，可以想到枚举每一个位置作为回文串的中心，然后二分能延伸的最长长度，直接哈希检验，但这样的复杂度是 $\mathcal{O}(n \log n)$，无法通过此题，我们需要一个更快的算法：这就是 $manacher$ 算法了。

Manacher

依然考虑通过回文串的中心来寻找回文串，但是观察以下两种字符串：$aabaa$ 与 $aaaa$，二者都是回文的，但它们的对称中心不同，前者的中心是 b，后者的中心在两个 a 之间，这会导致记录回文中心十分困难，因此我们可以在每两个之间插入一个像 $\#$ 一样的字符，这样所有的字符串的中心就都是字符了，同时为了区分边界，我们在字符串最前添加一个 $。

```
scanf("%s",s+1);len=strlen(s+1);
t[++n]='$';
for(int j=1;j<=len;++j){
    t[++n]='#';
    t[++n]=s[j];
}
```

Manacher

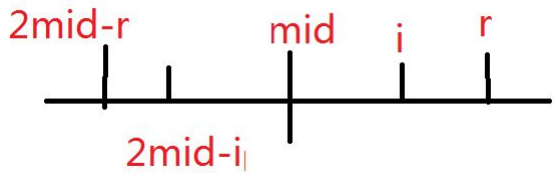
记 $p[i]$ 为新字符串中以 i 为对称中心的最长回文串的半径，则除去 $\#$ 后 $\max p[i]-1$ 就是原字符串中最长的回文串长度。

Manacher

记 $p[i]$ 为新字符串中以 i 为对称中心的最长回文串的半径，则除去 $\#$ 后 $\max p[i]-1$ 就是原字符串中最长的回文串长度。

类似扩展 KMP，定义 mid 与 r 表示目前找到的所有回文串的右边界中最远的一个是 r，这个回文串的中心是 mid。

考虑计算 $p[i]$，且 i 在 mid 与 r 之间，假设 $j=2mid-i$ 为 i 关于 mid 的对称点。

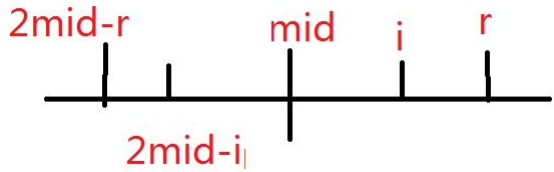


Manacher

记 $p[i]$ 为新字符串中以 i 为对称中心的最长回文串的半径，则除去 $\#$ 后 $\max p[i]-1$ 就是原字符串中最长的回文串长度。

类似扩展 KMP，定义 mid 与 r 表示目前找到的所有回文串的右边界中最远的一个是 r，这个回文串的中心是 mid。

考虑计算 $p[i]$，且 i 在 mid 与 r 之间，假设 $j=2mid-i$ 为 i 关于 mid 的对称点。



由对称性，在 $i+p[j]-1\le r$ 时：
$[j-p[j]+1,j+p[j]-1]=[i-p[j]+1,i+p[j]-1]$ 是回文串。因此 $p[i]\ge p[j]$。当 $i+p[j]-1>r$ 时，超出的部分无法从 j 那边继承过来只能去掉，因此总结得到 $p[i]\ge\min(p[j],r-i+1)$。将它赋值为 $p[i]$ 的初始值再暴力扩展。

Manacher

过程与扩展 KMP 比较类似，复杂度证明也类似：因为我们始终保证 r 是最右边的回文串端点，所以暴力拓展时，要么 j 的回文串左端点越界，要么 i 越界，此时 i 的回文串右端一定超出 r，于是每次暴力拓展一定会更新一次 r，于是暴力拓展就相当于将 r 移动到最右侧，只会进行 $\mathcal{O}(n)$ 次，于是总复杂度是 $\mathcal{O}(n)$ 的。

洛谷 P1659

Description

给定字符串 s 与整数 k，求 s 的前 k 大奇回文子串长度的积。$|s| \leq 10^6, k \leq 10^{12}$。

洛谷 P1659

Description

给定字符串 s 与整数 k，求 s 的前 k 大奇回文子串长度的积。$|s| \leq 10^6, k \leq 10^{12}$。

通过 manacher 求出以 i 为中心的最长回文串半径为 $p[i]$ 时，意味着以 i 为中心半径为 $1 \sim p[i]$ 的串都是回文串。

因此将 p 排个序，差分一下，就能对任意 i 算出半径为 i 的回文子串有多少个。

洛谷 P4555 最长双回文串

Description

给定字符串 s，求 s 的最长双回文子串 t。最长双回文串是指可以划分为两部分 $T = XY$ 使得 $|X|, |Y| \geq 1$ 且 X, Y 都是回文串。$|S| \leq 10^5$。

洛谷 P4555 最长双回文串

Description

给定字符串 s，求 s 的最长双回文子串 t。最长双回文串是指可以划分为两部分 $T = XY$ 使得 $|X|, |Y| \geq 1$ 且 X, Y 都是回文串。$|S| \leq 10^5$。

将 s 的相邻字符之间增加 $\#$，对每一位求出 $l[i], r[i]$ 分别表示以 i 开头/结尾的最长回文串长度。那么答案只需要枚举 $\#$ 做这个中间点。

洛谷 P4555 最长双回文串

Description

给定字符串 s，求 s 的最长双回文子串 t。最长双回文串是指可以划分为两部分 $T = XY$ 使得 $|X|, |Y| \geq 1$ 且 X, Y 都是回文串。$|S| \leq 10^5$。

将 s 的相邻字符之间增加 $\#$，对每一位求出 $l[i], r[i]$ 分别表示以 i 开头/结尾的最长回文串长度。那么答案只需要枚举 $\#$ 做这个中间点。

l, r 首先利用求出的回文串 $[i-p[i]+1, i-p[i]+1]$ 来更新：$l[i-p[i]+1] \leftarrow p[i]-1, r[i+p[i]-1] \leftarrow p[i]+1$。

$[i-p[i]+1, i-p[i]+1]$ 是以 i 为中心的最长字符串，还需要考虑以 i 为中心的其他回文子串，这可以通过将最大回文子串向内推，即让 l 向右推，r 向左推来实现：$l[i] \leftarrow l[i-2]-2, r[i] \leftarrow r[i+2]-2$。复杂度 $\mathcal{O}(|s|)$。

洛谷 P9646 Paper-cutting

Description

一个 $n \times m$ 的网格，格子上填写了 $0/1$，可以沿行/列的格线对折，要求格线必须是原矩阵的对称轴，对折后保留较大的部分，问最终图中的 0 连通块最少是多少。$n \times m \leq 10^6$。

洛谷 P9646 Paper-cutting

Description

一个 $n \times m$ 的网格，格子上填写了 $0/1$，可以沿行/列的格线对折，要求格线必须是原矩阵的对称轴，对折后保留较大的部分，问最终图中的 0 连通块最少是多少。$n \times m \leq 10^6$。

对折要求对称，也就相当于对折部分每一行/列都是偶回文串。

对折一定不劣于不对折，考虑尽量地对折所有回文区块。对折顺序是无所谓的，不影响结果。

因此只需要 manacher 找到每行每列以每个格线为中心的最长回文子串，四个方向贪心折。最后 BFS。

回文自动机

回文自动机 PAM 是一个能够识别一个字符串所有的回文子串的自动机，是一个字符串所有回文子串的信息的高度压缩得到的结果。因此，PAM 能够方便地解决一系列关于回文串的问题。

回文自动机

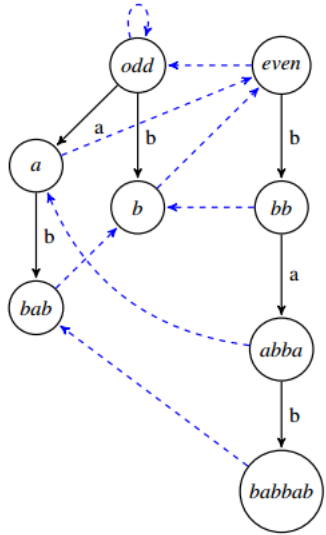
回文自动机 PAM 是一个能够识别一个字符串所有的回文子串的自动机，是一个字符串所有回文子串的信息的高度压缩得到的结果。因此，PAM 能够方便地解决一系列关于回文串的问题。

回文自动机维护了原串上的所有本质不同的回文串。

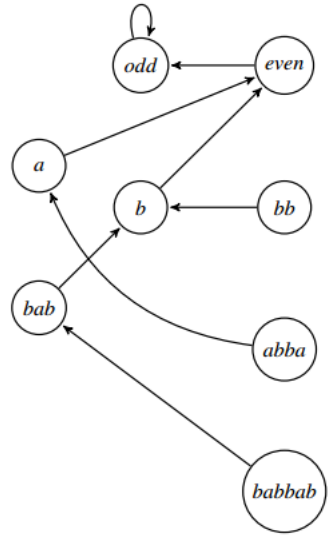
回文自动机的结构可以看成是两棵树，一棵的根是奇根 odd，代表着一个长度为 -1 的实际上不存在的回文串，存储所有长度为奇数的回文串，一棵的根是偶根 $even$，代表着长度为 0 的回文串，存储所有长度为偶数的回文串。

自动机的每一个转移对应树上的一条边，同时带有一个字符 c，如果这条边连接 $u \to v$，那么 s_v 就是 s_u 在前后分别加上字符 c 组成的，s_u 是 u 节点维护的回文串。显然，这样做可以表示出所有的回文子串。
此外，同样对每个节点 x 连接一条失配边 $fail[x]$ 指向它对应回文串的最长回文后缀。那么通过一路条 $fail$，可以找到该回文串的所有回文后缀。$fail$ 边同样组成一棵近似树的结构。

回文自动机



(a) babbab的回文树



(b) babbab的fail树

复杂度分析

引理

对于一个字符串 s，它的本质不同的回文子串至多有 $|s|$ 个。

复杂度分析

引理

对于一个字符串 s，它的本质不同的回文子串至多有 $|s|$ 个。

当 $|s| = 1$ 时定理显然成立，当 $|s| > 1$ 时，设 $s = tc$，假设结论对 t 成立，那么增加 c 后设新增的回文串的左端点从小到大排序后依次为 $l_1, l_2, \dots l_k$，于是有 $s[l_1 \dots |s|]$ 是回文串。

因此 $s[l_i \dots |s|] = s[l_1, |s| - l_1 + l_i]$，当 $i \neq 1$ 时 $|s| - l_1 + l_i \leq |t|$，也就是说 $s[l_i \dots |s|]$ 这个回文串已经在 t 中出现过了。因此，每增加一个字符，本质不同的回文子串数量最多增加 1。

复杂度分析

引理

对于一个字符串 s，它的本质不同的回文子串至多有 $|s|$ 个。

当 $|s| = 1$ 时定理显然成立，当 $|s| > 1$ 时，设 $s = tc$，假设结论对 t 成立，那么增加 c 后设新增的回文串的左端点从小到大排序后依次为 $l_1, l_2, \dots l_k$，于是有 $s[l_1 \dots |s|]$ 是回文串。

因此 $s[l_i \dots |s|] = s[l_1, |s| - l_1 + l_i]$，当 $i \neq 1$ 时 $|s| - l_1 + l_i \leq |t|$，也就是说 $s[l_i \dots |s|]$ 这个回文串已经在 t 中出现过了。因此，每增加一个字符，本质不同的回文子串数量最多增加 1。

PAM 中的节点个数 $=$ 本质不同的回文子串的个数 $+2$（去掉 odd 与 $even$），于是 PAM 的状态数是线性的，每个状态由唯一的转移边转移而来（两棵树），因此转移数也是线性的。

PAM 的构造

对于每一个节点，需要维护它对应回文串的长度 len 以及 $fail$，并用类似 $Trie$ 树的方法维护它的儿子，用 0 表示 $even$，1 表示 odd，初始化 $len[0] = 0, len[1] = -1$

PAM 的构造

对于每一个节点，需要维护它对应回文串的长度 len 以及 $fail$，并用类似 $Trie$ 树的方法维护它的儿子，用 0 表示 $even$，1 表示 odd，初始化 $len[0] = 0, len[1] = -1$

采用增量法构造 PAM，因为上面的定理，我们每次插入最多只会增加一个节点，即新字符串的最长回文后缀。记 $last$ 为上次插入后字符串的最长回文后缀对应的点，初始为 0，设插入的是第 i 个字符 u，那么插入后新字符串的最长回文后缀，应当是由原来的一个点 x 通过 u 转移而来，并且必须满足 $s[i - len[x] - 1] = u$，这样就能满足它是回文串了。

PAM 的构造

对于每一个节点，需要维护它对应回文串的长度 len 以及 $fail$，并用类似 $Trie$ 树的方法维护它的儿子，用 0 表示 $even$，1 表示 odd，初始化 $len[0] = 0, len[1] = -1$

采用增量法构造 PAM，因为上面的定理，我们每次插入最多只会增加一个节点，即新字符串的最长回文后缀。记 $last$ 为上次插入后字符串的最长回文后缀对应的点，初始为 0，设插入的是第 i 个字符 u，那么插入后新字符串的最长回文后缀，应当是由原来的一个点 x 通过 u 转移而来，并且必须满足 $s[i - len[x] - 1] = u$，这样就能满足它是回文串了。

找到 x 的方法则是从 $last$ 开始一路跳 $fail$ 遍历所有回文后缀直到满足条件为止。设新点为 tot，那么 $len[tot] = len[x] + 2$。

新点的 $fail$ 怎么求呢？如果 x 是奇根，则最长回文后缀长为 1，直接将 $fail$ 指向 0，否则同样从 $fail[x]$ 开始一路跳 $fail$，直到找到一个满足 $s[i - len[x] - 1] = u$ 的 x，x 的 u 儿子就是新点的 $fail$。

复杂度分析

唯一看起来复杂度不对的地方在于跳 $fail$。但每跳一次 fail 都会导致最长回文后缀长度减少至少 1，增加字符最多只会增加 1，因此复杂度 $\mathcal{O}(n)$。

复杂度分析

唯一看起来复杂度不对的地方在于跳 $fail$。但每跳一次 fail 都会导致最长回文后缀长度减少至少 1，增加字符最多只会增加 1，因此复杂度 $\mathcal{O}(n)$。

洛谷 P5496 【模板】回文自动机

给定字符串 s，求 s 的本质不同回文子串个数。$|s| \leq 5 \times 10^5$。

复杂度分析

唯一看起来复杂度不对的地方在于跳 $fail$。但每跳一次 fail 都会导致最长回文后缀长度减少至少 1，增加字符最多只会增加 1，因此复杂度 $\mathcal{O}(n)$。

洛谷 P5496 【模板】回文自动机

给定字符串 s，求 s 的本质不同回文子串个数。$|s| \leq 5 \times 10^5$。

建立 PAM，答案为 PAM 节点数 -2（奇根，偶根）。

[APIO2014] 回文串

Description

给定字符串 s，定义 s 的一个子串的存在值为该子串在 s 中的出现次数乘以子串长度。求 s 所有回文子串的最大存在值。$|s| \leq 3 \times 10^5$。

[APIO2014] 回文串

Description

给定字符串 s，定义 s 的一个子串的存在值为该子串在 s 中的出现次数乘以子串长度。求 s 所有回文子串的最大存在值。$|s| \leq 3 \times 10^5$。

即求每个回文子串在原串中的出现次数。

[APIO2014] 回文串

Description

给定字符串 s，定义 s 的一个子串的存在值为该子串在 s 中的出现次数乘以子串长度。求 s 所有回文子串的最大存在值。$|s| \leq 3 \times 10^5$。

即求每个回文子串在原串中的出现次数。

在增量构造过程中，每新增一个字符时它的所有回文后缀出现次数都会 $+1$。于是给它的最大回文后缀打一个 tag，然后按拓扑序从大到小枚举，将当前点的 tag 添加到 $fail$ 上即可。

在 PAM 中，节点本就是按 fail 的拓扑序加入的（新增节点的 fail 一定已经被加入），因此直接按编号从大到小扫一遍即可。

[SHOI2011] 双倍回文

Description

定义一个字符串 s 是双倍回文串当且仅当 s 长度为 4 的倍数，且 s, s 的前半部分, s 的后半部分都是回文串。给定字符串求最长双倍回文子串 $n \leq 5 \times 10^5$。

[SHOI2011] 双倍回文

Description

定义一个字符串 s 是双倍回文串当且仅当 s 长度为 4 的倍数，且 s, s 的前半部分, s 的后半部分都是回文串。给定字符串求最长双倍回文子串 $n \leq 5 \times 10^5$。

在 PAM 上定义 $trans$ 指针，每个节点的 $trans$ 指针指向不超过该回文串长度一半的最长回文后缀，则本题只需要判断每个回文串的 $trans$ 的长度是否恰好是它的一半。

根据定义可以类似 $fail$ 确定 $trans$ 的维护方式：新增一个节点时，若节点长度 ≤ 2 则令 $trans = fail$。否则，设它的父亲为 u，先跳到 $trans[u]$，再从 $trans[u]$ 一路跳 $fail$ 直到跳到一个长度不超过当前回文串一半的回文后缀。

loj 141. 回文子串

Description

给定字符串 s，支持在末尾增加一个字符串，在开头增加一个字符串，查询本质不同非空回文子串数量。$|s| \leq 10^5, q \leq 10^5$，最终 $|s| \leq 4 \times 10^5$。

loj 141. 回文子串

Description

给定字符串 s，支持在末尾增加一个字符串，在开头增加一个字符串，查询本质不同非空回文子串数量。$|s| \leq 10^5, q \leq 10^5$，最终 $|s| \leq 4 \times 10^5$。

在末尾增加沿用 PAM 的增量构造没有问题，考虑如何在开头增加。

初始将字符串的左右端点设为 $4e5+1$ 和 $4e5$，对前后插入时分别维护一个 $llast$ 与 $rlast$ 表示最长回文前缀和最长回文后缀。

对每个节点维护最长回文前缀与最长回文后缀，后者就是 $fail$，而因为是回文串，所以最长回文前缀就等于最长回文后缀，因此只需要一个 fail，可以一同修改，这样两边的增加就几乎独立了，只有当插入字符后整个字符串成为回文串时，要将 $llast$ 与 $rlast$ 合并。复杂度还是线性。

CF932G Palindrome Partition

Description

给定字符串 s，将 s 划分为偶数个字符串 $t_1 t_2 \cdots t_k$，使得 $t_i = t_{k-i+1}$，求方案数。$|s| \leq 10^6$。

CF932G Palindrome Partition

Description

给定字符串 s，将 s 划分为偶数个字符串 $t_1 t_2 \cdots t_k$，使得 $t_i = t_{k-i+1}$，求方案数。$|s| \leq 10^6$。

由于对称字符串长度对位相等，可以将 s 划分为相等长度的两个字符串，分别划分。

对称下标的字符串几乎是对位的：$[l, r]$ 与 $[n-r+1, n-l+1]$，但是并不是回文子串的翻转对位而是按顺序对位。这启示我们将翻转后半部分字符串，重组 s 变为 $s_1 s_n s_2 s_{n-2} \ldots s_{n/2} s_{n/2+1}$。则原本对位的 $[l, r]$ 与 $[n-r+1, n-l+1]$ 变成了回文串 $[2l+1, 2r+2]$。

于是原问题变为将重组后的字符串划分为若干个偶回文串的方案数。

CF932G Palindrome Partition

可以写一个暴力 DP：f_i 表示划分了 $s[1,i]$，最后一个回文串以 i 结尾。转移枚举 $s[1,i]$ 的所有偶回文后缀：$f_i = \sum_j f_{j-1}\mathsf{valid}(j,i)$。

遍历这些偶回文后缀，可以在 PAM 上跳 fail，但是复杂度不正确。

CF932G Palindrome Partition

可以写一个暴力 DP：f_i 表示划分了 $s[1,i]$，最后一个回文串以 i 结尾。转移枚举 $s[1,i]$ 的所有偶回文后缀：$f_i = \sum_j f_{j-1}\text{valid}(j,i)$。

遍历这些偶回文后缀，可以在 PAM 上跳 fail，但是复杂度不正确。

引理

一个回文串的所有回文后缀长度构成 $\mathcal{O}(\log n)$ 个等差数列。

CF932G Palindrome Partition

可以写一个暴力 DP：f_i 表示划分了 $s[1,i]$，最后一个回文串以 i 结尾。转移枚举 $s[1,i]$ 的所有偶回文后缀：$f_i = \sum_j f_{j-1}\text{valid}(j,i)$。

遍历这些偶回文后缀，可以在 PAM 上跳 fail，但是复杂度不正确。

引理

一个回文串的所有回文后缀长度构成 $\mathcal{O}(\log n)$ 个等差数列。

证明：回文串的回文后缀一定是 border，border 是 $\mathcal{O}(\log n)$ 个等差数列。(划掉)

Weak Periodicity Lemma

Lemma

p 和 q 是字符串 s 的周期，$p+q \leq |s|$，则 $\gcd(p,q)$ 也是 s 的周期。

Weak Periodicity Lemma

Lemma

p 和 q 是字符串 s 的周期，$p+q \le |s|$，则 $\gcd(p,q)$ 也是 s 的周期。

不妨设 $p \le q$，由 $p+q \le |s|$，$\forall i \in [1,|s|]$ 有 $i > p$ 或 $i+q \le |s|$。二者均可推出 $s[i]=s[i+d]$：$s[i]=s[i-p]=s[i-p+q], s[i]=s[i+q]=s[i+q-p]$。

辗转相除则得证。

Weak Periodicity Lemma

Lemma

p 和 q 是字符串 s 的周期，$p+q \le |s|$，则 $\gcd(p,q)$ 也是 s 的周期。

不妨设 $p \le q$，由 $p+q \le |s|$，$\forall i \in [1,|s|]$ 有 $i > p$ 或 $i+q \le |s|$。二者均可推出 $s[i]=s[i+d]$：$s[i]=s[i-p]=s[i-p+q], s[i]=s[i+q]=s[i+q-p]$。

辗转相除则得证。

Periodicity Lemma

若 $p+q-\gcd(p,q) \le |s|$，则 $\gcd(p,q)$ 也是 s 的周期。

border 的性质

引理

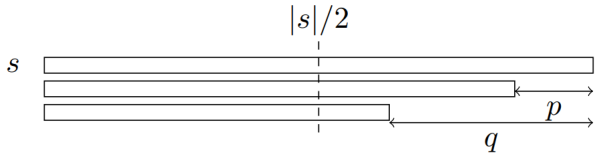
字符串 s 的所有不小于 $|s|/2$ 的 border 长度组成一个等差数列。

border 的性质

引理

字符串 s 的所有不小于 $|s|/2$ 的 border 长度组成一个等差数列。

设 s 的最大 border 长度为 $n-p(p \le |s|/2)$, 另外某个 border 的长度为 $n-q(q \le |s|/2)$, 则 $gcd(p,q)$ 是 s 的周期, 即 $n-gcd(p,q)$ 是 s 的 border 长度, 于是 $gcd(p,q) \ge p \Rightarrow p|q$。$p$ 是周期则所有 p 的倍数都是周期，因此长度大于字符串一半的 border 长度形如 $[n-p, n-2p, n-3p, \dots]$。

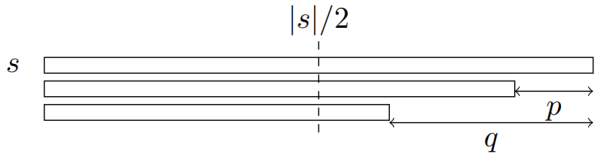


border 的性质

引理

字符串 s 的所有不小于 $|s|/2$ 的 border 长度组成一个等差数列。

设 s 的最大 border 长度为 $n-p(p \le |s|/2)$, 另外某个 border 的长度为 $n-q(q \le |s|/2)$, 则 $gcd(p,q)$ 是 s 的周期, 即 $n-gcd(p,q)$ 是 s 的 border 长度, 于是 $gcd(p,q) \ge p \Rightarrow p|q$。$p$ 是周期则所有 p 的倍数都是周期，因此长度大于字符串一半的 border 长度形如 $[n-p, n-2p, n-3p, \dots]$。



对于长度 $\le \frac{|s|}{2}$ 的 border，它们都是其中最长一个的 border，可以对这个最长 border 递归下去。最多递归 $\log$ 层，于是原引理得证。

原问题

现在我们要利用这个 $\log$ 个等差数列的性质来优化转移。

对每个节点 u，维护 $dif[u] = len[u] - len[fail[u]]$，并维护 $jmp[u]$ 表示 u 到祖先链上最近的满足 $dif \neq dif[u]$ 的节点，这两个数组在增量构造时容易维护。
在维护 $g[u]$ 为 $u \sim jum[u]$ 之间所有节点的 f 之和。由于等差数列的性质，
$g[u] = f[len[i] - len[jmp[j]]] + f[i - len[jmp[j]] - dif[j]] + \cdots =$
$f[len[i] - len[jmp[j]]] + g[fail[i]]$ 可以 $\mathcal{O}(1)$ 转移。

那么 dp 转移时，只需要暴力跳 jmp，将 g 累加到 f 上即可，复杂度 $\mathcal{O}(n \log n)>$

后缀数组

洛谷 P3809 【模板】后缀排序

读入一个长度为 n 的由大小写英文字母或数字组成的字符串，请把这个字符串的所有非空后缀按字典序（用 ASCII 数值比较）从小到大排序，然后按顺序输出后缀的第一个字符在原串中的位置。位置编号为 1 到 n。用后缀 i 表示第 i 位开始的后缀。

$n \leq 10^6$

后缀数组

后缀数组 $(SA, Suffix\ Array)$，是将字符串的所有后缀排序得到的数组，主要包括两个数组：

$sa[i]$：将所有后缀按字典序排序后第 i 小的后缀的开头位置。

$rk[i]$：表示从第 i 个字符开始的后缀（我们将它称为后缀 i）的字典序排名

它们满足 $sa[rk[i]] = rk[sa[i]] = i$

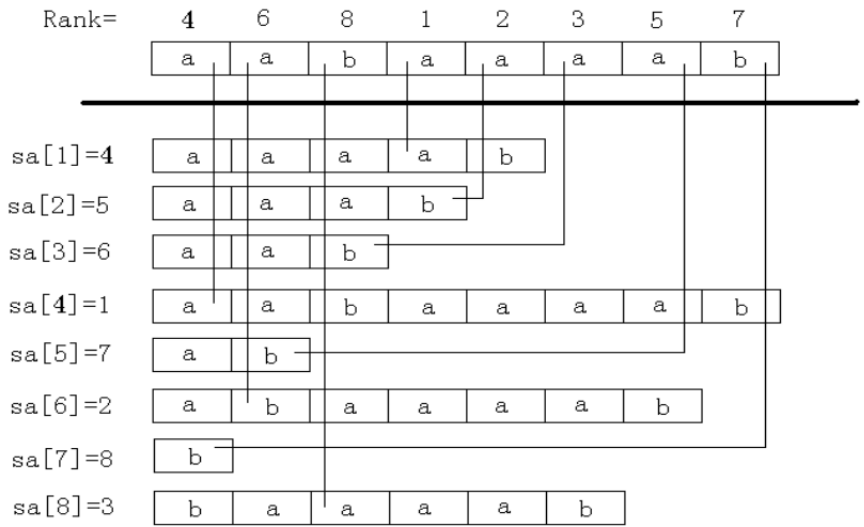
后缀数组

后缀数组 $(SA, Suffix\ Array)$，是将字符串的所有后缀排序得到的数组，主要包括两个数组：

$sa[i]$：将所有后缀按字典序排序后第 i 小的后缀的开头位置。

$rk[i]$：表示从第 i 个字符开始的后缀（我们将它称为后缀 i）的字典序排名

它们满足 $sa[rk[i]] = rk[sa[i]] = i$



倍增求后缀数组

后缀数组能够帮助我们快速解决许多关于子串、后缀的字符串题目，这里先介绍后缀数组的 $\mathcal{O}(n \log n)$ 倍增求法。

倍增求后缀数组

后缀数组能够帮助我们快速解决许多关于子串、后缀的字符串题目，这里先介绍后缀数组的 $\mathcal{O}(n \log n)$ 倍增求法。

正常比较字符串的字典序需要先比较第一位，再比较第二位 $\ldots$。倍增法，顾名思义就是依次比较所有后缀的前 $k = 1, 2, 4, \ldots 2^c$ 位。

倍增求后缀数组

后缀数组能够帮助我们快速解决许多关于子串、后缀的字符串题目，这里先介绍后缀数组的 $\mathcal{O}(n \log n)$ 倍增求法。

正常比较字符串的字典序需要先比较第一位，再比较第二位 $\ldots$。倍增法，顾名思义就是依次比较所有后缀的前 $k = 1, 2, 4, \ldots 2^c$ 位。

首先按照每个后缀的第一个字符对后缀进行排序，这相当于将这个字符串的每个字符进行排序。

倍增求后缀数组

接下来考虑已经对 $k = n$ 排好了序，如何对 $k = 2n$ 排序：

倍增求后缀数组

接下来考虑已经对 $k=n$ 排好了序，如何对 $k=2n$ 排序：
将每个子串拆分为前 k 位与第 $k+1\sim 2k$ 位，如果能分别求出这两部分的排序，那么对 $k=2n$ 排序相当于对这样一个二元组进行双关键字排序。
前 k 位的排序已知，对于第 $k+1\sim 2k$ 位，注意到后缀 i 的第 $k+1\sim 2k$ 位，就是后缀 $i+k$ 的前 k 位，因此这部分的排序可以通过前 k 位的排序整体移 k 位 $\mathcal{O}(n)$ 得到。

倍增求后缀数组

接下来考虑已经对 $k=n$ 排好了序，如何对 $k=2n$ 排序：
将每个子串拆分为前 k 位与第 $k+1\sim 2k$ 位，如果能分别求出这两部分的排序，那么对 $k=2n$ 排序相当于对这样一个二元组进行双关键字排序。
前 k 位的排序已知，对于第 $k+1\sim 2k$ 位，注意到后缀 i 的第 $k+1\sim 2k$ 位，就是后缀 $i+k$ 的前 k 位，因此这部分的排序可以通过前 k 位的排序整体移 k 位 $\mathcal{O}(n)$ 得到。

最后的双关键字排序由于值域只有 n，可以使用基数排序做到 $\mathcal{O}(n)$。
这样就可以 $\mathcal{O}(n)$ 得到 $k=2n$ 的排序，总复杂度 $\mathcal{O}(n\log n)$。

代码理解

具体如何使用基数排序可能初学者很难理解，我们直接结合代码讲解。

代码理解

具体如何使用基数排序可能初学者很难理解，我们直接结合代码讲解。

记 $s[i]$ 表示原字符串第 i 位，$rk[i]$ 表示所有后缀按照前 k 位排序得到的结果（前 k 位相同的后缀 rk 相同），一开始排序的是第一个字母，那么 $rk[i]$ 就是 $s[i]$。

代码理解

具体如何使用基数排序可能初学者很难理解，我们直接结合代码讲解。

记 $s[i]$ 表示原字符串第 i 位，$rk[i]$ 表示所有后缀按照前 k 位排序得到的结果（前 k 位相同的后缀 rk 相同），一开始排序的是第一个字母，那么 $rk[i]$ 就是 $s[i]$。

可以根据 rk （因为有相同值，叫做权值或许更好），求出此时的 sa：

```
for(int i=1;i<=n;++i)
    rk[i]=s[i],++c[rk[i]];//刚开始第一关键字就是该后缀的第一个字母
for(int i=2;i<=S;++i)
    c[i]+=c[i-1];
for(int i=n;i>=1;--i) sa[c[rk[i]]--]=i;
```

这里 S 是目前排名集合的大小。我们用桶排序的思想，用 $c[x]$ 表示 $rk=x$ 的后缀个数，做前缀和之后 $c[x]$ 表示权值 $\leq x$ 的字符串个数。
据此可以将所有后缀填到 sa 上：i 应该 $c_{rk_i}-cnt_{rk_i}+1\sim c_{rk_i}$ 中的某个位置上了。在出现相同权值时，我们现在不关心它们的内部排名，就直接让位置靠后的字符串排名较大，最后一个直接排名为 $c[rk[i]]$，然后将 $c--$ 作为下一个权值相同的字符串的位置以保证实际排名互不相同。

代码理解

接着开始倍增，枚举 k 表示目前我们已经知道前 k 位的排序，想要推出前 $2k$ 位的排序。

代码理解

接着开始倍增，枚举 k 表示目前我们已经知道前 k 位的排序，想要推出前 $2k$ 位的排序。

```
int num=0;
for(int i=n-k+1;i<=n;++i) y[++num]=i;
for(int i=1;i<=n;++i)
    if(sa[i]>k) y[++num]=sa[i]-k;//y[i]：第二关键字排名为i位的后缀的起始位置
```

这一段代码是求出第二关键字，即第 $k+1-2k$ 位的字典序排序，我们用 $y[i]$ 表示第二关键字排名第 i 位的后缀的其起始位置，对于后缀 $n-k+1-n$，它们没有 $k+1-2k$ 位的东西，因此直接排在最前面，紧接着枚举第一关键字的排名，被先枚举到的 $sa[i]$ 意味着后缀 $sa[i]-k$ 的第二关键字排名靠前，因此我们按序加入 y 中。

代码理解

```
for(int i=1;i<=S;++i) c[i]=0;
for(int i=1;i<=n;++i) c[rk[i]]++;
for(int i=2;i<=S;++i) c[i]+=c[i-1];
for(int i=n;i>=1;--i)
    sa[c[rk[y[i]]]--]=y[i],y[i]=rk[i];
//桶排序优先保证了第一关键字的排名，因为是从后往前考虑y所以说相对靠后的是第二关键字排行靠后的
```

接下来开始基数排序，我们还是先按 rk 放在桶中，但这次对于 rk 相同的后缀我们不能随意排序了，要第二关键字靠后的排在后面，于是我们从后往前枚举 $y[i]$，这样先枚举到的第二关键字一定更大，于是给它较大的排名。$y[i] = rk[i]$ 则是我们接下来要重新计算 rk，但会用到之前的 rk，于是我们直接用 y 保存下来。

代码理解

```
rk[sa[1]]=num=1;
for(int i=2;i<=n;++i){
    if(y[sa[i]]==y[sa[i-1]]&&y[sa[i]+k]==y[sa[i-1]+k])
    else rk[sa[i]]=++num;
}
if(num==n) return ;
S=num;
```

接下来我们利用 sa 重新计算 rk，这时如果两个后缀的两个关键字都相同，那么直接使用 num，否则 $num++$，当 $num=n$ 时所有后缀的排名互不相同，于是我们就完成了排序。

其他后缀数组算法

DC3

可以参见集训队论文【后缀数组——处理字符串的有力工具罗穗骞】。
复杂度 $\mathcal{O}(n)$，常数巨大，几乎没人写。

其他后缀数组算法

DC3

可以参见集训队论文【后缀数组——处理字符串的有力工具罗穗骞】。
复杂度 $\mathcal{O}(n)$，常数巨大，几乎没人写。

SA-IS

可以参见【诱导排序与 SA-IS 算法】
(https://riteme.site/blog/2016-6-19/sais.html)。
复杂度 $\mathcal{O}(n)$，常数比 DC3 小，可以使用，但 oi 中应该不会有毒瘤出题人卡后缀排序复杂度。

最长公共前缀 LCP

定义 $LCP(i,j)$ 表示后缀 $sa[i]$ 与后缀 $sa[j]$ 的最长公共前缀。

最长公共前缀 LCP

定义 $LCP(i,j)$ 表示后缀 $sa[i]$ 与后缀 $sa[j]$ 的最长公共前缀。

首先可以只用考虑 $i<j$ 的情况：

- $LCP(i,j)=LCP(j,i)$
- $LCP(i,i)=n-sa[i]+1$

最长公共前缀 LCP

定义 $LCP(i,j)$ 表示后缀 $sa[i]$ 与后缀 $sa[j]$ 的最长公共前缀。

首先可以只用考虑 $i<j$ 的情况：

- $LCP(i,j)=LCP(j,i)$
- $LCP(i,i)=n-sa[i]+1$

LCP Lemma

$LCP(i,j)=\min(LCP(i,k),LCP(k,j))(1\le i\le k\le j\le n)$

最长公共前缀 LCP

定义 $LCP(i,j)$ 表示后缀 $sa[i]$ 与后缀 $sa[j]$ 的最长公共前缀。

首先可以只用考虑 $i<j$ 的情况：

- $LCP(i,j)=LCP(j,i)$
- $LCP(i,i)=n-sa[i]+1$

LCP Lemma

$LCP(i,j)=\min(LCP(i,k),LCP(k,j))(1\le i\le k\le j\le n)$
证明：令 $t=min(LCP(i,k),LCP(k,j))$，那么 $LCP(i,k)\ge t, LCP(k,j)\ge t$，后缀 $sa[i]$ 前 t 位 $=sa[k]$ 前 t 位 $=sa[j]$ 的前 t 位，故 $LCP(i,j)\ge t$。

若 $LCP(i,j)=q>t$，那么 i,j 的前 q 个字符相等，因为 $t=min(LCP(i,k),LCP(k,j))$，所以要么 $sa[i][t+1]$(表示后缀 $sa[i]$ 的第 $t+1$ 位)$<sa[k][t+1]$，要么 $sa[k][t+1]<sa[j][t+1]$，并且 $sa[i][t+1]\le sa[k][t+1]\le sa[j][t+1]$，所以 $sa[i][t+1]\ne sa[j][t+1]$，与假设矛盾，所以 $LCP(i,j)=t$

最长公共前缀 LCP

LCP Theorem

$LCP(i,j) = min(LCP(k-1,k))\ , k \in (i,j]$

最长公共前缀 LCP

LCP Theorem

$LCP(i,j) = min(LCP(k-1,k))\ , k \in (i,j]$
证明：由 $LCP\ Lemma$：$LCP(i,j) = min(LCP(i,i+1), LCP(i+1,j)$, 然后继续拆下去即可证明。

最长公共前缀 LCP

LCP Theorem

$LCP(i,j) = min(LCP(k-1,k))\ , k \in (i,j]$
证明：由 $LCP\ Lemma$：$LCP(i,j) = min(LCP(i,i+1), LCP(i+1,j)$, 然后继续拆下去即可证明。

令 $height[i] = LCP(i,i-1), height[1] = 0$，那么求出 $height\ LCP$ 就是个区间 min 了。

最长公共前缀 LCP

令 $h[i] = height[rk[i]]$，于是 $height[i] = h[sa[i]]$，对 $h[i]$，有定理：

$$h[i] \geq h[i-1] - 1$$

证明

首先我们假设 $sa[rk[i]-1] = j, sa[rk[i-1]-1] = k$，于是 $h[i] = LCP(j,i), h[i-1] = LCP(k,i-1)$，于是我们只需证明 $LCP(j,i) \geq LCP(k,i-1) - 1$。

如果后缀 k 与后缀 $i-1$ 首字母不同，显然成立。

如果后缀 k 与后缀 $i-1$ 首字母相同，那么分别去掉首字母后得到后缀 $k+1$ 与后缀 i，必有 $rk[k+1]$ 也 $< rk[i]$，于是 $LCP(k+1,i) = h[i-1] - 1$，对于字符串 i，所有排名比它靠前的字符串中，与它相似度最高也就是 LCP 最大的一定是紧挨着它的字符串，即 j，但我们已知 $k+1$ 排在 i 前面并且 $LCP(k+1,i) = h[i-1] - 1$，那么必然有 $LCP(j,i) \geq LCP(k+1,i) = h[i-1] - 1$，即 $h[i] \geq h[i-1] + 1$。

最长公共前缀 LCP

令 $h[i] = height[rk[i]]$，于是 $height[i] = h[sa[i]]$，对 $h[i]$，有定理：

$$h[i] \geq h[i-1] - 1$$

证明

首先我们假设 $sa[rk[i]-1] = j, sa[rk[i-1]-1] = k$，于是 $h[i] = LCP(j,i), h[i-1] = LCP(k,i-1)$，于是我们只需证明 $LCP(j,i) \geq LCP(k,i-1) - 1$。

如果后缀 k 与后缀 $i-1$ 首字母不同，显然成立。

如果后缀 k 与后缀 $i-1$ 首字母相同，那么分别去掉首字母后得到后缀 $k+1$ 与后缀 i，必有 $rk[k+1]$ 也 $< rk[i]$，于是 $LCP(k+1,i) = h[i-1] - 1$，对于字符串 i，所有排名比它靠前的字符串中，与它相似度最高也就是 LCP 最大的一定是紧挨着它的字符串，即 j，但我们已知 $k+1$ 排在 i 前面并且 $LCP(k+1,i) = h[i-1] - 1$，那么必然有 $LCP(j,i) \geq LCP(k+1,i) = h[i-1] - 1$，即 $h[i] \geq h[i-1] + 1$。

一条定理，我们就可以直接按 $rk[i]$ 从小到大枚举 i 然后从 $height[rk[i-1]] - 1$ 作为起始点枚举 $height[rk[i]]$ 达到 $\mathcal{O}(n)$ 求出所有 $height$ 了。

后缀数组练习题

无特别说明，字符集都是小写字母。

后缀数组练习题

spoj LCS

求两个串的最长公共子串长度，每个串的长度不超过 250000。

后缀数组练习题

spoj LCS

求两个串的最长公共子串长度，每个串的长度不超过 250000。

将两个串接在一起跑后缀排序，变成找两个 LCP 最大的不同颜色后缀，对每个后缀找前后最近的颜色不同后缀即可。

后缀数组练习题

POJ 1743

给定一个字符串 s，求在 s 中不重叠地出现两次的串的最大长度。串长不超过 250000。

后缀数组练习题

POJ 1743

给定一个字符串 s，求在 s 中不重叠地出现两次的串的最大长度。串长不超过 250000。

二分答案 k，转化为找两个 $LCP \geq k$ 且距离 $\geq k$ 的后缀。后缀排序后取出 $height \geq k$ 的区间分别考虑岂可。

后缀数组练习题

[NOI2015] 品酒大会

给定一个字符串 s，每个后缀 $s[i \dots n]$ 有权值 a_i，对于每个 $r = 1, 2, \dots, n$，统计出有多少种方法可以选出两个 $lcp \geq r$ 的后缀，并回答两个后缀权值乘积的最大值。

$|s| \leq 3 \times 10^5, |a_i| \leq 10^9$。

后缀数组练习题

[NOI2015] 品酒大会

给定一个字符串 s，每个后缀 $s[i \dots n]$ 有权值 a_i，对于每个 $r = 1, 2, \dots, n$，统计出有多少种方法可以选出两个 $lcp \geq r$ 的后缀，并回答两个后缀权值乘积的最大值。
$|s| \leq 3 \times 10^5, |a_i| \leq 10^9$。

从大到小枚举 r 并维护 $height \geq r$ 的区间，从 r 到 $r-1$ 相当于合并一些区间。并查集维护最大次大。

后缀数组练习题

[NOI2016] 优秀的拆分

如果一个字符串可以被拆分为 AABB 的形式，其中 A 和 B 是任意非空字符串，则我们称该字符串的这种拆分是优秀的。
现在有 T 组询问，每组询问给出一个长度为 n 的字符串 S，求出在它所有子串的所有拆分方式中，优秀拆分的总个数。
其中出现在不同位置的相同子串，我们认为是不同的子串，它们的优秀拆分均会被记入答案。
$1 \leq T \leq 10, n \leq 30000$

后缀数组练习题

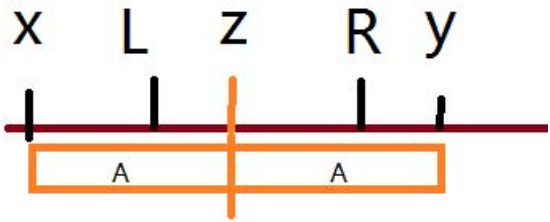
[NOI2016] 优秀的拆分

如果一个字符串可以被拆分为 AABB 的形式，其中 A 和 B 是任意非空字符串，则我们称该字符串的这种拆分是优秀的。
现在有 T 组询问，每组询问给出一个长度为 n 的字符串 S，求出在它所有子串的所有拆分方式中，优秀拆分的总个数。
其中出现在不同位置的相同子串，我们认为是不同的子串，它们的优秀拆分均会被记入答案。
$1 \leq T \leq 10, n \leq 30000$

先枚举 AA 的末尾位置 i，那么 $ans = \sum_i f_i g_{i+1}$。
其中 f_i 表示以第 i 位作为结尾的形如 AA 的串的数量，g_i 表示以第 i 位作为开头的形如 AA 的串的数量。

后缀数组练习题

枚举 A 的长度 len，然后将原串中第 $len, 2len, 3len, \ldots$ 位作为关键点。那么**所有的 AA 串都必定恰好经过两个关键点**，不妨设其为 l, r，显然 $r = l + len$。



如图所示，$x \sim L$ 段与 $z \sim R$ 段相同，$L+1-Z$ 段与 $R+1-y$ 段相同。其中前者是 $pre[L]$ 与 $pre[R]$ 的公共后缀，后者是 $suf[L+1]$ 与 $suf[R+1]$ 的公共前缀。因此，分别求出 lcs 为 $pre[L]$ 与 $pre[R]$ 的最长公共后缀，lcp 为 $suf[L+1]$ 与 $suf[R+1]$ 的最长公共前缀。

那么只要 $lcs + lcp > len$，我们就能找到前后各一段合法的开始位置与结束位置，可以调和级数枚举所有这样的 $[L, R]$，差分统计即可。

后缀数组练习题

模板题

给定字符串 s，求 s 的本质不同子串个数。
$|s| \leq 10^5$

后缀数组练习题

模板题

给定字符串 s，求 s 的本质不同子串个数。
$|s| \leq 10^5$

将 s 后缀排序后，用子串总数量减去相邻两个串的 lcp。

后缀数组练习题

floj4105 人类补完计划

给定字符串 s，求 s 的本质不同子串个数。
本质不同定义为：
两个子串本质不同当且仅当以下两种条件满足其一：

- 字符串长度不同
- 对于长度相同的两个字符串，不存在一个字符集的映射，使得第二个串的字符映射后等于第一个串。

$|s| \leq 10^5$

后缀数组练习题

对于一个字符串 s，定义 $a_i = i - lst_{s_i}$ 其中 lst_c 为 c 在 s 中第一次出现的位置；如果 s_i 是第一次出现则令 $a_i = 0$。可以发现两个字符串本质相同当且仅当其对应的 a 序列本质相同。

后缀数组练习题

对于一个字符串 s，定义 $a_i = i - lst_{s_i}$ 其中 lst_c 为 c 在 s 中第一次出现的位置；如果 s_i 是第一次出现则令 $a_i = 0$。可以发现两个字符串本质相同当且仅当其对应的 a 序列本质相同。

考虑先求出整个字符串的 a 序列 $a_{1\sim n}$，对于该字符串的一个子串 $s[l \sim r]$，它的 a 序列与 $a[l \sim r]$ 相比的变化就是将每个字符的第一次出现位置对应的 a 变为 0，也就是说变化的位置是 $\mathcal{O}(\sum)$ 的（字符集大小）。

后缀数组练习题

对于一个字符串 s，定义 $a_i = i - lst_{s_i}$ 其中 lst_c 为 c 在 s 中第一次出现的位置；如果 s_i 是第一次出现则令 $a_i = 0$。可以发现两个字符串本质相同当且仅当其对应的 a 序列本质相同。

考虑先求出整个字符串的 a 序列 $a_{1\sim n}$，对于该字符串的一个子串 $s[l \sim r]$，它的 a 序列与 $a[l \sim r]$ 相比的变化就是将每个字符的第一次出现位置对应的 a 变为 0，也就是说变化的位置是 $\mathcal{O}(\sum)$ 的（字符集大小）。

注意到所有子串的 a 序列就是所有后缀的 a 序列的前缀，因此可以使用类似 SA 求本质不同子序列数量的方法，将所有后缀的 a 序列排序后，用总长度减去相连两个串的 lcp 即可得到答案。排序可以使用 lcp 实现，现在考虑怎么求解 lcp：

后缀数组练习题

对于一个字符串 s，定义 $a_i = i - lst_{s_i}$ 其中 lst_c 为 c 在 s 中第一次出现的位置；如果 s_i 是第一次出现则令 $a_i = 0$。可以发现两个字符串本质相同当且仅当其对应的 a 序列本质相同。

考虑先求出整个字符串的 a 序列 $a_{1\sim n}$，对于该字符串的一个子串 $s[l \sim r]$，它的 a 序列与 $a[l \sim r]$ 相比的变化就是将每个字符的第一次出现位置对应的 a 变为 0，也就是说变化的位置是 $\mathcal{O}(\sum)$ 的（字符集大小）。

注意到所有子串的 a 序列就是所有后缀的 a 序列的前缀，因此可以使用类似 SA 求本质不同子序列数量的方法，将所有后缀的 a 序列排序后，用总长度减去相连两个串的 lcp 即可得到答案。排序可以使用 lcp 实现，现在考虑怎么求解 lcp：

求解后缀 l 与后缀 r 的 lcp 时，考虑从小到大枚举后缀 l，后缀 r 相比 $a[l \sim n], a[r \sim n]$ 改变的位置，相邻两个改变位置之间的部分可以对原串的 a 序列后缀排序预处理后快速求出 $\mathcal{O}(1)lcp$，这样就可以做到单次询问复杂度 $\mathcal{O}(\sum)$，总复杂度 $\mathcal{O}(n\sum \log n)$。

后缀树

我们知道 AC 自动机可以处理模式串集合固定，文本串任意的字符串匹配问题。

后缀树

我们知道 AC 自动机可以处理模式串集合固定，文本串任意的字符串匹配问题。如何求解文本串固定，模式串未知的字符串匹配问题？

后缀树

我们知道 AC 自动机可以处理模式串集合固定，文本串任意的字符串匹配问题。
如何求解文本串固定，模式串未知的字符串匹配问题？
考虑对文本串预处理，用某种数据结构维护文本串的所有子串。

后缀树

我们知道 AC 自动机可以处理模式串集合固定，文本串任意的字符串匹配问题。
如何求解文本串固定，模式串未知的字符串匹配问题？
考虑对文本串预处理，用某种数据结构维护文本串的所有子串。

注意到子串是后缀的前缀，因此可以将所有后缀插入 $trie$ 树中，则模式串的匹配位置（开头匹配的位置）就是对应节点子树内所有后缀。

后缀树

我们知道 AC 自动机可以处理模式串集合固定，文本串任意的字符串匹配问题。
如何求解文本串固定，模式串未知的字符串匹配问题？
考虑对文本串预处理，用某种数据结构维护文本串的所有子串。

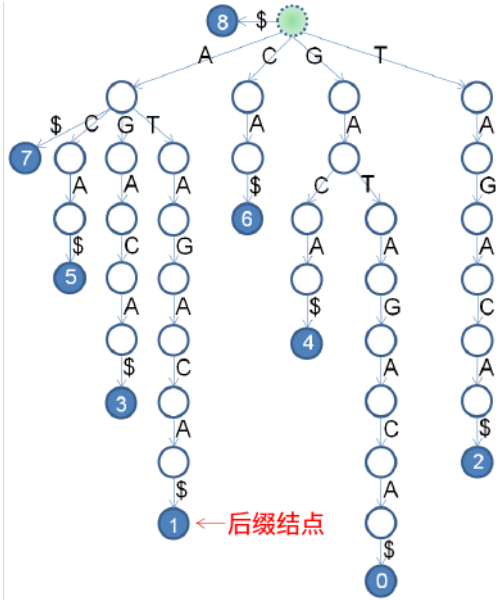
注意到子串是后缀的前缀，因此可以将所有后缀插入 $trie$ 树中，则模式串的匹配位置（开头匹配的位置）就是对应节点子树内所有后缀。

但是 $\mathcal{O}(n^2)$ 个节点太多了。考虑将只有 1 个儿子的节点于儿子删掉，即建立后缀节点的虚树，就可以将节点降至 $\mathcal{O}(n)$，这就是后缀树。

后缀树

以串 GATAGACA 为例。首先在串的末尾加一个特殊字符 $，它可以确保所有后缀都是 trie 上的叶子结点，便于处理，实际实现时不一定需要。

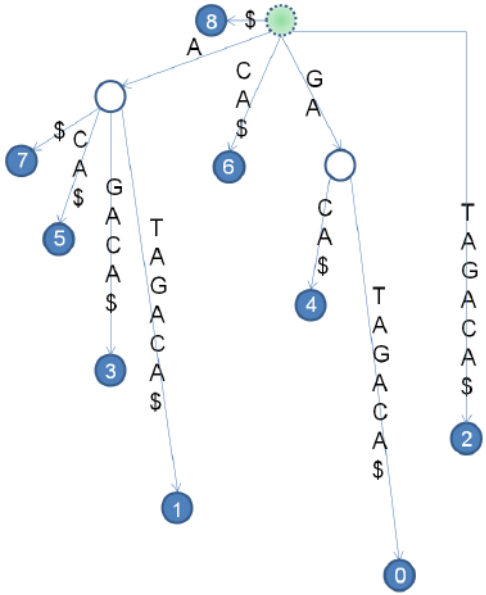
后缀 *trie*:



后缀树

以串 GATAGACA 为例。首先在串的末尾加一个特殊字符 $，它可以确保所有后缀都是 trie 上的叶子结点，便于处理，实际实现时不一定需要。

后缀树：



后缀树

可以发现后缀树依然满足一个模式串在文本串中出现的位置就是这个串的子树中所有的叶子。

后缀树

可以发现后缀树依然满足一个模式串在文本串中出现的位置就是这个串的子树中所有的叶子。

同时，按照后缀树的 dfs 序，我们可以直接得到所有后缀排序的结果，即后缀数组。
$height$ 数组就是 dfs 序上两个相邻叶子的 LCA 深度。
使用后缀自动机，可以 $\mathcal{O}(n)$ 构建后缀树。

后缀自动机

后缀自动机（suffix automaton,SAM）是一个能识别字符串 s 的所有后缀的最小 DFA，不理解什么是 DFA 也没关系，总之就是它满足以下性质：

后缀自动机

后缀自动机（suffix automaton,SAM）是一个能识别字符串 s 的所有后缀的最小 DFA，不理解什么是 DFA 也没关系，总之就是它满足以下性质：

- SAM 是一个 DAG，节点被称为状态，边被称为转移，每个转移上是一个字母。

后缀自动机

后缀自动机（suffix automaton,SAM）是一个能识别字符串 s 的所有后缀的最小 DFA，不理解什么是 DFA 也没关系，总之就是它满足以下性质：

- SAM 是一个 DAG，节点被称为状态，边被称为转移，每个转移上是一个字母。
- 存在一个初始状态，从初始状态出发能到达任意节点，并且将从初始状态出发的任意路径上的所有转移的字母写下来就是 s 的一个子串，s 的每一个子串均能被这样的路径表示出来。

后缀自动机

后缀自动机（suffix automaton,SAM）是一个能识别字符串 s 的所有后缀的最小 DFA，不理解什么是 DFA 也没关系，总之就是它满足以下性质：

- SAM 是一个 DAG，节点被称为状态，边被称为转移，每个转移上是一个字母。
- 存在一个初始状态，从初始状态出发能到达任意节点，并且将从初始状态出发的任意路径上的所有转移的字母写下来就是 s 的一个子串，s 的每一个子串均能被这样的路径表示出来。
- 存在若干个终止状态，从初始状态到终止状态的任意路径都是 s 的一个后缀，s 的每一个后缀都可以由这样的方式表示出来。

后缀自动机

后缀自动机（suffix automaton,SAM）是一个能识别字符串 s 的所有后缀的最小 DFA，不理解什么是 DFA 也没关系，总之就是它满足以下性质：

- SAM 是一个 DAG，节点被称为状态，边被称为转移，每个转移上是一个字母。
- 存在一个初始状态，从初始状态出发能到达任意节点，并且将从初始状态出发的任意路径上的所有转移的字母写下来就是 s 的一个子串，s 的每一个子串均能被这样的路径表示出来。
- 存在若干个终止状态，从初始状态到终止状态的任意路径都是 s 的一个后缀，s 的每一个后缀都可以由这样的方式表示出来。
- SAM 的状态数、转移数都是线性的。

后缀自动机

定义结束位置 $endpos$：对于字符串 s 的任意子串 t，$endpos(t)$ 表示 t 在 s 中的所有结束位置组成的集合，例如对于
$s =$ “$dabda$”, $endpos($“da”$) = \{2, 5\}$, $endpos($“b”$) = \{3\}$。

后缀自动机

定义结束位置 $endpos$：对于字符串 s 的任意子串 t，$endpos(t)$ 表示 t 在 s 中的所有结束位置组成的集合，例如对于 $s =$ “$dabda$”, $endpos($“da”$) = \{2, 5\}$, $endpos($“b”$) = \{3\}$。

可以发现之前的后缀树实际就是将 $endpos$ 相同的子串合并到了一个点上，后缀自动机其实也是如此。
因为 $endpos$ 相同的子串可能有很多，但它们在后面加上相同字母时，得到的新串的 $endpos$ 一定依然相同。所以每个节点通过同一个字母只能转移到一个节点上，可以对 $endpos$ 等价类建立 SAM，当然每个节点可能有多个入点。

后缀自动机

$endpos$ 满足许多优美的性质：

引理 1

对于字符串的任意非空子串 u 与 v，如果 $endpos(u) = endpos(v)$，且 $|u| \leq |v|$，那么 u 是 v 的后缀。

后缀自动机

$endpos$ 满足许多优美的性质：

引理 1

对于字符串的任意非空子串 u 与 v，如果 $endpos(u) = endpos(v)$，且 $|u| \leq |v|$，那么 u 是 v 的后缀。

引理 2

对于字符串的任意非空子串 u 与 v，如果 $|u| \leq |v|$，那么 $endpos(v) \in endpos(u)$ 或者 $endpos(u) \cap endpos(v) = \varnothing$。

后缀自动机

$endpos$ 满足许多优美的性质：

引理 1

对于字符串的任意非空子串 u 与 v，如果 $endpos(u) = endpos(v)$，且 $|u| \leq |v|$，那么 u 是 v 的后缀。

引理 2

对于字符串的任意非空子串 u 与 v，如果 $|u| \leq |v|$，那么 $endpos(v) \in endpos(u)$ 或者 $endpos(u) \cap endpos(v) = \varnothing$。

证明：如果 u 是 v 的后缀，那么 v 出现时 u 一定出现，于是 $endpos(v) \in endpos(u)$，否则 u 与 v 一定不会同时出现。

后缀自动机

因此，我们可以按照不同的 $endpos$ 将原串的所有子串分为若干个 $endpos$ 等价类。

引理 3

对于任意一个 $endpos$ 等价类，其中的所有子串将它们按长度从大到小排序，那么每一个子串都是前一个子串的后缀，长度 $=$ 前者 -1，等价类的长度值域恰好覆盖连续的一段。

后缀自动机

因此，我们可以按照不同的 $endpos$ 将原串的所有子串分为若干个 $endpos$ 等价类。

引理 3

对于任意一个 $endpos$ 等价类，其中的所有子串将它们按长度从大到小排序，那么每一个子串都是前一个子串的后缀，长度 $=$ 前者 -1，等价类的长度值域恰好覆盖连续的一段。

证明：如果等价类仅包含一个字符串那么引理显然成立，否则设 u 为该等价类最短的字符串，v 为最长的字符串，那么 v 的所有长度 $\geq |u|$ 的后缀，根据引理 2，它们也一定属于这一等价类中。

后缀自动机

因此，我们可以按照不同的 $endpos$ 将原串的所有子串分为若干个 $endpos$ 等价类。

引理 3

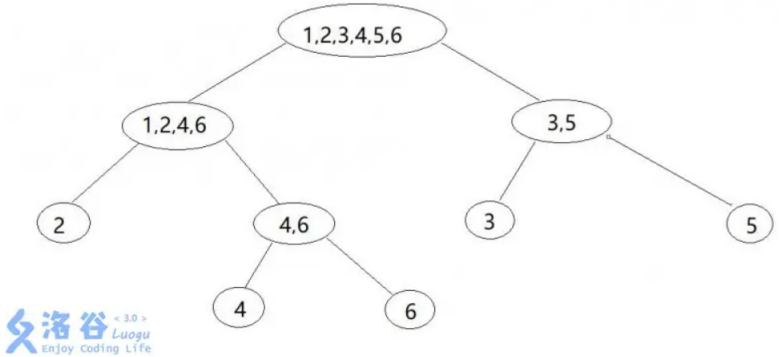
对于任意一个 $endpos$ 等价类，其中的所有子串将它们按长度从大到小排序，那么每一个子串都是前一个子串的后缀，长度 $=$ 前者 -1，等价类的长度值域恰好覆盖连续的一段。

证明：如果等价类仅包含一个字符串那么引理显然成立，否则设 u 为该等价类最短的字符串，v 为最长的字符串，那么 v 的所有长度 $\geq |u|$ 的后缀，根据引理 2，它们也一定属于这一等价类中。

对于任意一个等价类，我们设 v 为该等价类最长的子串，在 v 前加另一个字符，如果依然得到原串的子串，那么它一定属于另一个等价类，这个等价类的 $endpos$ 一定 $\in endpos(v)$，且在 v 前加不同的字符会得到不同的等价类，它们的 $endpos$ 一定不相交，于是我们相当于将 $endpos(v)$ 拆分为了若干个新的集合并保留原来的集合。我们将从 A 分割得到 B 看作是父子关系，借此我们就能建出一棵树。

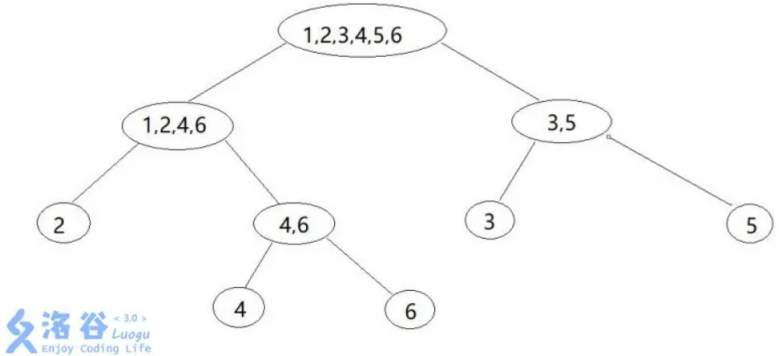
后缀自动机

下图是对 $aababa$ 建出的树，根节点是空串，我们认为它的 $endpos$ 为全集：



后缀自动机

下图是对 $aababa$ 建出的树，根节点是空串，我们认为它的 $endpos$ 为全集：

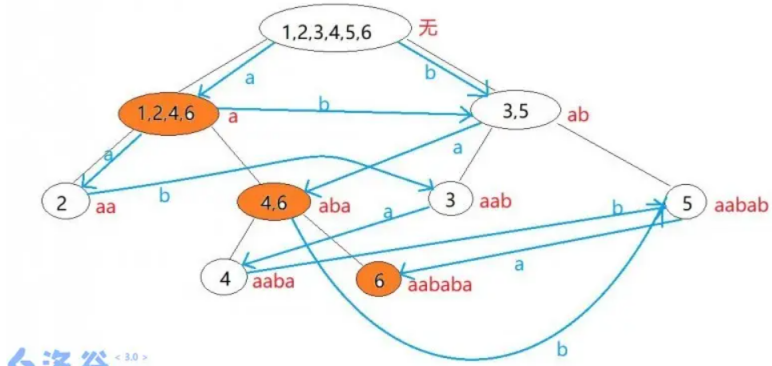


根据分割关系可以发现树的节点是不超过 $2\mathcal{O}(n)$ 的。这棵树称为 parent tree，实际上，这棵树与后缀树的区别就是该树通过在前面加字符完成地，所以 parent tree 其实就是反串的后缀树。

后缀自动机

我们认为任意等价类 x 中的最长字符串的长度为 $len(x)$，最短字符串长度为 $minlen(x)$，它在 $parent\ tree$ 上的父亲为 $link(x)$，那么显然有 $minlen(x) = len(link(x)) + 1$，因为 x 正是由 $fa(x)$ 增加一个字符得来的。

但 SAM 的转移并不是 $parent\ tree$ 上的边，因为这些边是在字符串前面加一个字符，而 SAM 中的转移边应该是在最后加一个字符，我们希望能在 $parent\ tree$ 的节点之间连边，使起点出发到任意点的路径都对应属于该点的一个字符串。具体建出来的结果是这样的：



后缀自动机

考虑增量法构造 SAM，即依次将 s 的每一个字母加入 SAM 中并使 SAM 维护新出现的子串（即以当前字母为结尾的子串）。

后缀自动机

考虑增量法构造 SAM，即依次将 s 的每一个字母加入 SAM 中并使 SAM 维护新出现的子串（即以当前字母为结尾的子串）。

以下原串表示 $s[1,i-1]$，新串表示 $s[1,i]$，记录 c 为要加入的这个字母，它是第 i 个字母，$last$ 为旧串对应的节点编号，用 $ch[i][c]$ 表示 i 的各条转移连向的点，设 1 为初始节点。

首先，新建一个节点 cur 表示新产生的最大的串 $s[1,i]$，显然它的 $endpos=\{i\}$ 是全新的，应该是一个新的节点，有 $len[cur]=len[last]+1$。

```
int cur=++tot;len[cur]=len[last]+1;
```

后缀自动机

接下来我们考虑哪些子串的 $endpos$ 需要被修改——新串的后缀，它们都是原串的一个后缀通过转移 c 得来的。
我们通过遍历原串的后缀来找到它们，遍历后缀的方法则是在 parent 树上不断往父亲跳直到跳到根。

后缀自动机

接下来我们考虑哪些子串的 $endpos$ 需要被修改——新串的后缀，它们都是原串的一个后缀通过转移 c 得来的。
我们通过遍历原串的后缀来找到它们，遍历后缀的方法则是在 parent 树上不断往父亲跳直到跳到根。

注意到我们是按后缀的长度从大到小遍历的，一开始的几个后缀可能没有 c 这个转移，也就是说它们对应的新串后缀从未在旧串中出现过，因此它们的 $endpos = \{i\}$，应该属于 cur：

```
while(p&&!ch[p][c]){//p=0表明我们已经遍历完所有后缀了
    ch[p][c]=cur;
    p=link[p];
}
if(!p) link[cur]=1;
```

后缀自动机

如果 $p = 0$，那么所有新串后缀都已并入 cur，这样的节点在 $parent\ tree$ 上只能从初始节点转移而来了，因此将 $link[cur]$ 设为 1。

后缀自动机

如果 $p = 0$，那么所有新串后缀都已并入 cur，这样的节点在 $parent\ tree$ 上只能从初始节点转移而来了，因此将 $link[cur]$ 设为 1。

否则，假设现在来到了节点 p，$ch[p][c] = q$ 是已经出现过的点，则在 q 中 $len \leq len[p] + 1$ 的串都是新串的后缀，$endpos$ 应该增加一个 i 进去，其余则不是。所以需要对 q 进行分割。

后缀自动机

如果 $p=0$，那么所有新串后缀都已并入 cur，这样的节点在 $parent\ tree$ 上只能从初始节点转移而来了，因此将 $link[cur]$ 设为 1。

否则，假设现在来到了节点 p，$ch[p][c]=q$ 是已经出现过的点，则在 q 中 $len \le len[p]+1$ 的串都是新串的后缀，$endpos$ 应该增加一个 i 进去，其余则不是。所以需要对 q 进行分割。

如果 $len[q]=len[p]+1$，则 q 中所有串均为新串的后缀，$endpos$ 都要增加一个 i，于是不用修改它，因为 cur 就是 q 前加一个字母得到的，将 $link[cur]$ 指向 q 即可。

```
int q=ch[p][c];
if(len[q]==len[p]+1) link[cur]=q;
```

后缀自动机

于是我们新建一个 np 表示 q 中 $len \le len[p]+1$ 的部分，将它们从 q 中分割出来。q 中只保留剩余部分，这些字符串 $endpos$ 不变。

那么 np 中的最长字符串是新串的后缀，它的所有后缀都是新串的后缀。
$len[np]=len[p]+1$，np 的 ch 数组应当与 q 相同。
考虑 $link[np]$，之前的 $link[q]$ 表示什么？表示 $link[q]$ 通过加一个字符得到了 q 中最短的一个子串，这个子串一定属于 np，于是 $link[np]=link[q]$，$link[q]$ 自然就应该更换为 np，$link[cur]$ 也应该指向 np。

后缀自动机

拆分完后，我们不仅要考虑 q 与 np 的转移，还要考虑原先连向 np 与 q 的转移，一些原先在字符串最后增加一个字符能得到 q 的点，我们要将它们的转移边改为连向 np：它们一定是 p 的祖先。

于是我们继续跳 $link[p]$，接下来的点也应该在 $endpos$ 中增加 i，而如果它们本来连向的是 q，直接改为连向 np，当 $ch[p][c] \neq q$ 时，意味着现在 p 连向的是 q 的祖先了，节点中所有串 $endpos$ 都是增加 i，因此不需要进行修改，不用再往上跳了。

```
else{
    int np=++tot;
    len[np]=len[p]+1;
    link[np]=link[q];
    ch[np]=ch[q];//实际上ch[np]作为一个数组它不能这么用，我们这里只是表示这个意思
    while(p&&ch[p][c]==q){
        ch[p][c]=np;
        p=link[p];
    }
    link[cur]=link[q]=np;
}
```

后缀自动机

第一种情况从 p 再往上跳也是这样的结果，因此也不用继续往上跳了。
最后将 $last$ 更新为 cur，我们就完成了整个构造的过程。依次加入每个字符，我们就完成了 SAM 的构造：

后缀自动机

第一种情况从 p 再往上跳也是这样的结果，因此也不用继续往上跳了。
最后将 $last$ 更新为 cur，我们就完成了整个构造的过程。依次加入每个字符，我们就完成了 SAM 的构造：

关于复杂度：首先状态数已经证明是 $\mathcal{O}(n)$ 的，转移数显然是 $\mathcal{O}(\sum n)$ 的，也可以证明是 $\mathcal{O}(n)$ 的，

证明

找出 SAM 的一个生成树，将其他边舍去，然后从每一个后缀对应的终止状态出发沿 SAM 上起始状态到它的路径 (这个是唯一的) 往起始状态跑，如果遇见了一条非树边，就把这条边加入 SAM 中，走过这条边，设到达了 u，之后改为改沿生成树上 u 到起点的路径跑回起点。
这样的一条路径得到的字符串不一定是这个后缀，但一定也是原串的一个后缀，并且没有被跑过 (不然不会现在才连上)。
此时，我们将这个后缀划掉不跑了，继续回到 u 跑之前的后缀，重复这样的流程。
整个流程中，我们每增加的一条非树边都唯一对应一个后缀，因此非树边的数量是 $\mathcal{O}(n)$ 的，而树边数量等于状态数也是 $\mathcal{O}(n)$ 的，因此总转移数是 $\mathcal{O}(n)$ 的。

后缀自动机

考虑构造的复杂度：构造中看起来不是线性的应该就是以下几部分：

- 遍历旧串后缀，增加到 cur 的转移
- 增加新节点后复制之前 q 的转移到 np 上去 ($ch[np] = ch[q]$)
- 修改之前到 q 的转移到 np 上

对于前两个，它们每操作一次，就意味着新增一条转移，因此总复杂度是 $\mathcal{O}(n)$ 的，(字符集较大时用 $unordered\_map$ 维护 ch，会带有一些常数，较小时（大部分情况下）直接把 ch 开为数组暴力转移)

对于第三个，我们修改的转移就是连向新节点的所有转移，这些边不可能再被操作三遍历到了（已经满足了 $len[np] = len[p] + 1$），因此每条边最多只会被修改一次，操作三的总复杂度依然是线性的。

后缀自动机练习题

利用后缀自动机自动机的性质：可以通过沿文本串在自动机上走一遍找到该文本串的节点，判断文本串是否是原串的子串。
后缀自动机上维护了原串的所有子串，节点 p 包含长度为 $[len[link[p]]+1, len[p]]$ 的子串。

后缀自动机练习题

利用后缀自动机自动机的性质：可以通过沿文本串在自动机上走一遍找到该文本串的节点，判断文本串是否是原串的子串。
后缀自动机上维护了原串的所有子串，节点 p 包含长度为 $[len[link[p]]+1, len[p]]$ 的子串。

后缀自动机更重要的是 parent tree 的性质。通过在 $parent$ 树上跳父亲可以访问该串所有后缀。

后缀自动机练习题

利用后缀自动机自动机的性质：可以通过沿文本串在自动机上走一遍找到该文本串的节点，判断文本串是否是原串的子串。
后缀自动机上维护了原串的所有子串，节点 p 包含长度为 $[len[link[p]]+1, len[p]]$ 的子串。

后缀自动机更重要的是 parent tree 的性质。通过在 $parent$ 树上跳父亲可以访问该串所有后缀。

直接开数组维护 $endpos$ 是不行的，空间会炸。
但注意到某个节点的 $endpos$ 就是其 parent tree 树上子树中的所有原串前缀。所以可以用线段树合并来求出每个节点的 $endpos$。

后缀自动机练习题

利用后缀自动机自动机的性质：可以通过沿文本串在自动机上走一遍找到该文本串的节点，判断文本串是否是原串的子串。
后缀自动机上维护了原串的所有子串，节点 p 包含长度为 $[len[link[p]]+1, len[p]]$ 的子串。

后缀自动机更重要的是 parent tree 的性质。通过在 $parent$ 树上跳父亲可以访问该串所有后缀。

直接开数组维护 $endpos$ 是不行的，空间会炸。
但注意到某个节点的 $endpos$ 就是其 parent tree 树上子树中的所有原串前缀。所以可以用线段树合并来求出每个节点的 $endpos$。

抓住这些性质应该能解决后缀自动机大部分基础题了。

后缀自动机练习题

洛谷 P3804 【模板】后缀自动机

给定字符串 s，求 s 的所有出现次数不为 1 的子串乘上该子串长度的最大值。$|s| \leq 10^6$。

后缀自动机练习题

洛谷 P3804 【模板】后缀自动机

给定字符串 s，求 s 的所有出现次数不为 1 的子串乘上该子串长度的最大值。$|s| \leq 10^6$。

建出 SAM，只需要求 $endpos$ 集合的大小的话只需要 dfs 一遍 parent tree 进行合并就行了。

后缀自动机练习题

洛谷 P3804 【模板】后缀自动机

给定字符串 s，求 s 的所有出现次数不为 1 的子串乘上该子串长度的最大值。$|s| \leq 10^6$。

建出 SAM，只需要求 $endpos$ 集合的大小的话只需要 dfs 一遍 parent tree 进行合并就行了。

实际上也可以不显示的建出树，注意到父亲的 len 一定比儿子小，因此直接将所有节点按长度排序就可以得到原树的一个拓扑序，可以按拓扑序从大到小依次遍历将该节点的 $endpos$ 集合合并到父亲上。
这里的拓扑排序可以用和后缀数组一样的桶排实现，复杂度 $\mathcal{O}(n)$。

后缀自动机练习题

[TJOI2015] 弦论

给定字符串 s，求 s 的第 k 小子串。
$T = 0$: 不同位置的相同子串算作一个。
$T = 1$: 不同位置的相同子串算作多个。
$|s| \leq 10^6$。

后缀自动机练习题

[TJOI2015] 弦论

给定字符串 s，求 s 的第 k 小子串。
$T = 0$: 不同位置的相同子串算作一个。
$T = 1$: 不同位置的相同子串算作多个。
$|s| \leq 10^6$。

预处理在 SAM 上从某个节点往后走，能走到多少个子串（也就是多少个子串经过该节点）即可。

后缀自动机练习题

loj6401. yww 与字符串

给定字符串 s 及常数 k，s 中有一些位置是坏的。
询问有多少个字符串 t 满足 t 作为 s 的子串出现过，且 t 的某一次出现中覆盖的所有的位置中坏位置不超过 k 个。
$|s| \leq 10^6$。

后缀自动机练习题

loj6401. yww 与字符串

给定字符串 s 及常数 k，s 中有一些位置是坏的。
询问有多少个字符串 t 满足 t 作为 s 的子串出现过，且 t 的某一次出现中覆盖的所有的位置中坏位置不超过 k 个。
$|s| \leq 10^6$。

双指针对每个前缀预处理其最长的好后缀长度 pre。
对于 SAM 上某个节点，符合条件的子串就是 $[minlen, len]$ 与 $[1, pre]$ 的交集。
当然每个子串的 $endpos$ 需要考虑它的整棵子树，在 parent 树上 dp，父节点取子节点 pre 的最大值。

后缀自动机练习题

[NOI2018] 你的名字

给定字符串 s，多次给定字符串 t，及区间 $[l,r]$，询问 t 有多少个本质不同的子串。没有在 $s[l,r]$ 中出现。
$|s|, \sum |t| \leq 5 \times 10^5$

后缀自动机练习题

[NOI2018] 你的名字

给定字符串 s，多次给定字符串 t，及区间 $[l,r]$，询问 t 有多少个本质不同的子串。没有在 $s[l,r]$ 中出现。
$|s|, \sum |t| \leq 5 \times 10^5$

首先转化为求出现的子串数量。
如果能建出 $s[l,r]$ 的后缀自动机，可以通过将 t 在后缀自动机上跑一遍找出 t 的每个前缀中最长的在 $s[l,r]$ 中出现的后缀 pre：沿 t 的字母走转移边并维护当前串的长度，有转移边直接转移，长度 $+1$。没有转移边就跳父亲，长度变为父亲的 len 再看父亲有没有该转移边。
最后对 T 建后缀自动机，类似上一题将 pre 打在 parent 树上，dp 一遍即可。

后缀自动机练习题

对于原题，考虑直接在 s 的自动机上实现上面的匹配：

后缀自动机练习题

对于原题，考虑直接在 s 的自动机上实现上面的匹配：

判断转移边是否在 $s[l,r]$ 的自动机上存在：线段树合并预处理每个点的 $endpos$，判断转移到点是否有在 $[l,r]$ 内的 $endpos$。

后缀自动机练习题

对于原题，考虑直接在 s 的自动机上实现上面的匹配：

判断转移边是否在 $s[l,r]$ 的自动机上存在：线段树合并预处理每个点的 $endpos$，判断转移到点是否有在 $[l,r]$ 内的 $endpos$。

读取某节点在 $s[l,r]$ 自动机上的 len：$s[l,r]$ 的 len 不同的唯一可能是该串只覆盖了 $s[l,r]$ 的一个前缀，取 $\min(len, maxendpos(l,r) - l + 1)$ 即可。

后缀自动机练习题

对于原题，考虑直接在 s 的自动机上实现上面的匹配：

判断转移边是否在 $s[l,r]$ 的自动机上存在：线段树合并预处理每个点的 $endpos$，判断转移到点是否有在 $[l,r]$ 内的 $endpos$。

读取某节点在 $s[l,r]$ 自动机上的 len：$s[l,r]$ 的 len 不同的唯一可能是该串只覆盖了 $s[l,r]$ 的一个前缀，取 $\min(len, maxendpos(l,r) - l + 1)$ 即可。

跳 $parent$ 树没有影响。

后缀自动机练习题

洛谷 P6816 [PA2009] Quasi-template

给定字符串 s，求有多少个串 t 能可超出头尾地覆盖 s 且 t 是 s 的子串。以及求合法串中最短的。

$|s| \leq 2 \times 10^5$

后缀自动机练习题

洛谷 P6816 [PA2009] Quasi-template

给定字符串 s，求有多少个串 t 能可超出头尾地覆盖 s 且 t 是 s 的子串。以及求合法串中最短的。
$|s| \leq 2 \times 10^5$

t 匹配 s 需要满足以下条件：

- t 在 s 的相邻两次出现的最大距离不能超过 $|s|$
- 设 t 第一次出现的记为位置为 l，$\exists x \in [l-|t|, |s|], s[1,x]$ 是 t 的一个后缀。
- 设 t 最后一次出现的结尾位置为 r，$\exists x \in [1, r+1], s[x,|s|]$ 是 t 的一个前缀。

后缀自动机练习题

洛谷 P6816 [PA2009] Quasi-template

给定字符串 s，求有多少个串 t 能可超出头尾地覆盖 s 且 t 是 s 的子串。以及求合法串中最短的。
$|s| \leq 2 \times 10^5$

t 匹配 s 需要满足以下条件：

- t 在 s 的相邻两次出现的最大距离不能超过 $|s|$
- 设 t 第一次出现的记为位置为 l，$\exists x \in [l-|t|, |s|], s[1,x]$ 是 t 的一个后缀。
- 设 t 最后一次出现的结尾位置为 r，$\exists x \in [1, r+1], s[x,|s|]$ 是 t 的一个前缀。

对每个等价类依次考虑其中有多少个子串符合条件，第一条中的距离，l，r，都可以在 SAM 上线段树合并维护。

后缀自动机练习题

对于第二条：同一个等价类的点后缀是固定的，只需要找到最长的后缀使得匹配 s 的一段前缀即可，注意到匹配等价于该串是 $s[1,l]$ 的 $border$，因此 kmp 预处理出 $s[1,l]$ 的最长 $border$ 看长度是否超过 $l-|t|$ 即可。

后缀自动机练习题

对于第二条：同一个等价类的点后缀是固定的，只需要找到最长的后缀使得匹配 s 的一段前缀即可，注意到匹配等价于该串是 $s[1,l]$ 的 $border$，因此 kmp 预处理出 $s[1,l]$ 的最长 $border$ 看长度是否超过 $l-|t|$ 即可。

对于第三条：对于每个子串 t，考虑其起始位置 i，则 $s[i,|s|]$ 是 t 的一个前缀且 $i \le r$，于是 $s[i,r-k]$ 为 s 的一个后缀，k 可任取。即 $s[i,r-k]$ 是后缀的 i 的一个 border。记 $border_i$ 为后缀 i 的最长 border，则只要 $r \ge \max(border_i, n-border_i)$ 即可。注意到如果 $border_i$ 大于后缀 i 长度的一半，则 $[i,r]$ 不可能属于当前等价类，所以只需保留 $r \ge n-border_i$。

预处理出每个子串的最长匹配长度，最终关于子串长度有上下界，转化为求区间内 $\le x$ 的数个数，可以离线树状数组。

广义后缀自动机

处理有多个文本串的问题：将后缀自动机建立在 Trie 树上。

广义后缀自动机

处理有多个文本串的问题：将后缀自动机建立在 Trie 树上。

常见的伪广义后缀自动机:
1. 通过用特殊符号将多个串直接连接后，再建立 SAM。
2. 对每个串，重复在同一个 SAM 上进行建立，每次建立前，将 last 指针置零。
正确性是对的，但时间复杂度较为危险。或者说两种字符串都不满足 SAM 节点数最小的定义。

离线建立广义后缀自动机

先对所有文本串建 Trie 树。然后在 Trie 树上 dfs/bfs 遍历建立 SAM，逐个加入 Trie 上的点，$insert$ 时 $last$ 为该点的父亲。
注意 dfs 的复杂度是 Trie 上所有叶子的深度之和，而 bfs 只是 Trie 的大小。如果题目直接给定 Trie，dfs 可能被卡掉。

在线建立广义后缀自动机

使用刚才提到的算法对每个串，重复在同一个 SAM 上进行建立，每次建立前，将 last 指针置零。但是加入特判。

在线建立广义后缀自动机

使用刚才提到的算法对每个串，重复在同一个 SAM 上进行建立，每次建立前，将 last 指针置零。但是加入特判。

设新加入的字母为 c，如果 $ch[last][c]$ 已经存在（在普通 SAM 中不会出现这种情况，因为新加入的一定是最长的串），就不进行新建节点的操作，直接进入分割 $ch[last][c]$ 的步骤。

广义后缀自动机

P6139 【模板】广义后缀自动机（广义 SAM）

给定 n 个由小写字母组成的字符串 $s_1, s_2 \dots s_n$，求本质不同的子串个数。(不包含空串)

进一步，设 Q 为能接受 $s_1, s_2, \dots, s_n$ 的所有后缀的最小 DFA，请你输出 Q 的点数。(如果你无法理解这句话，可以认为就是输出 $s_1, s_2, \dots, s_n$ 建成的“广义后缀自动机”的点数)。

$\sum |s_i| \leq 10^6$。

广义后缀自动机

P6139 【模板】广义后缀自动机（广义 SAM）

给定 n 个由小写字母组成的字符串 $s_1, s_2 \dots s_n$，求本质不同的子串个数。(不包含空串)
进一步，设 Q 为能接受 $s_1, s_2, \dots, s_n$ 的所有后缀的最小 DFA，请你输出 Q 的点数。(如果你无法理解这句话，可以认为就是输出 $s_1, s_2, \dots, s_n$ 建成的“广义后缀自动机”的点数)。
$\sum |s_i| \leq 10^6$。

模板。这道题判定了广义 SAM 的最小性，可以用来检测你的广义 SAM。

广义后缀自动机

[ZJOI2015] 诸神眷顾的幻想乡

给出一颗叶子结点不超过 20 个的无根树，每个节点上都有一个不超过 10 的数字，求树上本质不同的路径个数（两条路径相同定义为：其路径上所有节点上的数字依次相连组成的字符串相同）。
节点数 10^5

广义后缀自动机

[ZJOI2015] 诸神眷顾的幻想乡

给出一颗叶子结点不超过 20 个的无根树，每个节点上都有一个不超过 10 的数字，求树上本质不同的路径个数（两条路径相同定义为：其路径上所有节点上的数字依次相连组成的字符串相同）。
节点数 10^5

树上每条路径都会在以某个叶子节点为根时变成祖孙链。
对以每个叶子为根的前缀串建立广义 SAM。

广义后缀自动机

CF666E

给出主串 S 以及 m 个字符串 $T[1 \dots m]$ 。有若干次询问，每次查询 S 的子串 $S[pl \sim pr]$ 在 $T[l \sim r]$ 中的哪个串 T_i 里的出现次数最多，输出 i 以及出现次数，有多解则取最靠前的那一个。

广义后缀自动机

CF666E

给出主串 S 以及 m 个字符串 $T[1 \dots m]$ 。有若干次询问，每次查询 S 的子串 $S[pl \sim pr]$ 在 $T[l \sim r]$ 中的哪个串 T_i 里的出现次数最多，输出 i 以及出现次数，有多解则取最靠前的那一个。

将所有字符串插入广义 SAM 中，每个节点开一棵动态开点线段树维护它在 $T[1 \sim m]$ 中出现的次数。线段树合并即可。
查询从前缀 $S[1, pr]$（预处理）出发倍增找到包含 $S[pl, pr]$ 所在等价类节点进行查询。

完结撒花

字符串的世界博大精深，SA 与 SAM 的应用都远不止此，这里只是一点浅薄的介绍，大家可以下来继续了解。
谢谢大家！