字符串算法选讲

金策

清华大学交叉信息研究院

February 3, 2017

字符串: $s[1..n]$, $|s| = n$。

字符集: $s[i] \in \Sigma$。算法竞赛中常见的 Σ 是 26 个小写英文字母。

子串 $s[i..j] = s[i]s[i+1]\cdots s[j]$。

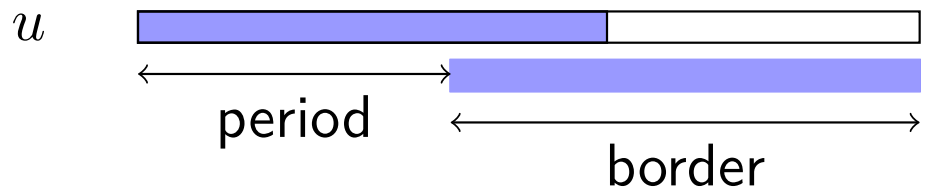
前缀 $\mathrm{pre}(s, x) = s[1..x]$, 后缀 $\mathrm{suf}(s, x) = s[n-x+1..n]$。

周期和 border

若 $0 < p \leq |s|$, $s[i] = s[i+p], \forall i \in \{1, \cdots, |s| - p\}$, 就称 p 是 s 的周期 (period)。

若 $0 \leq r < |s|$, $\mathrm{pre}(s, r) = \mathrm{suf}(s, r)$, 就称 $\mathrm{pre}(s, r)$ 是 s 的 border。

$\mathrm{pre}(s, r)$ 是 s 的 border $\Leftrightarrow$ $|s| - r$ 是 s 的周期。



比如 abaaaba 就有周期 $4, 6, 7$, 对应的 border 是 aba,a, 和 ϵ。

KMP 算法

可以在 $O(n)$ 时间求出数组 $\mathrm{fail}[1..n]$, 其中 $\mathrm{fail}[i]$ 表示前缀 $s[1..i]$ 的最大 border 长度。

s 的所有 border 长度?

$\{\mathrm{fail}[n], \mathrm{fail}[\mathrm{fail}[n]], \cdots\}$

后缀数组和 LCP 查询

在 $O(n \log n)$ 时间空间预处理后（或较复杂的 $O(n)$ 时间空间预处理），可以 $O(1)$ 回答: 两个子串的最长公共前缀 (LCP)、最长公共后缀 (LCS)。

对于拥有周期 p 的串 s, $\mathrm{LCP}(s[1..n], s[1+p..n]) = n - p$。

输入 i, $O(1)$ 回答最大的 l 使得 $s[i..i+l-1]$ 拥有周期 p。

Weak Periodicity 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。

证明: 令 $d = q - p$ $(p < q)$, 则由 $i - p > 0$ 或 $i + q \leq |s|$ 均可推出 $s[i] = s[i+d]$。

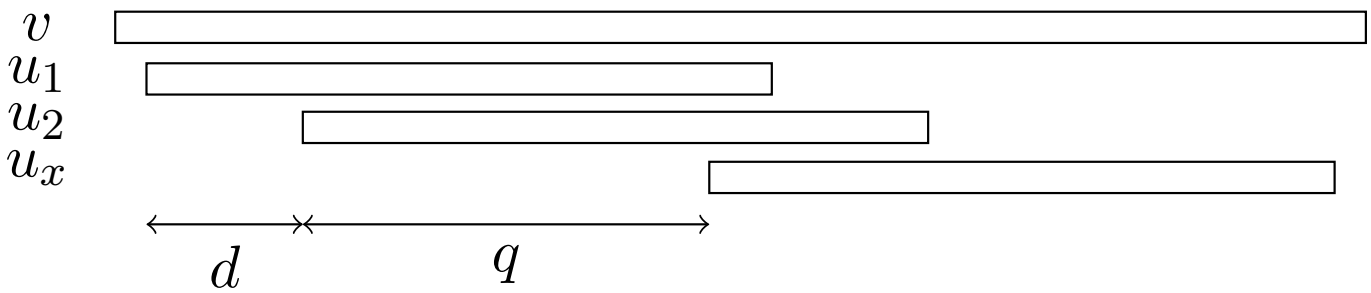
Periodicity Lemma: p 和 q 是 s 的周期, $p + q - \gcd(p,q) \leq |s|$, 则 $\gcd(p,q)$ 也是 s 的周期。

abaaba

字符串匹配

引理: 字符串 u, v 满足 $2|u| \geq |v|$, 则 u 在 v 中的所有匹配位置组成一个等差数列。

证明: 只需讨论至少匹配 3 次的情况。考虑 u 在 v 中的（最左边）第一次和第二次匹配, 间距为 d。另外某次匹配与第二次匹配的间距为 q。



可知 d, q 都是 u 的周期, 从而 $r = \gcd(d, q)$ 也是 u 的周期。设 u 的最小周期为 $p \leq r$。

由 $p \leq r \leq q \leq |u_1 \cap u_2|$ 知 p 也是 $u_1 \cup u_2$ 的周期。若 $p < d$, 则 u_1 右移 p 的距离也产生一次匹配, 矛盾。

于是 $d \leq p \leq r = \gcd(d, q)$ 即 $p = d, d \mid q$。

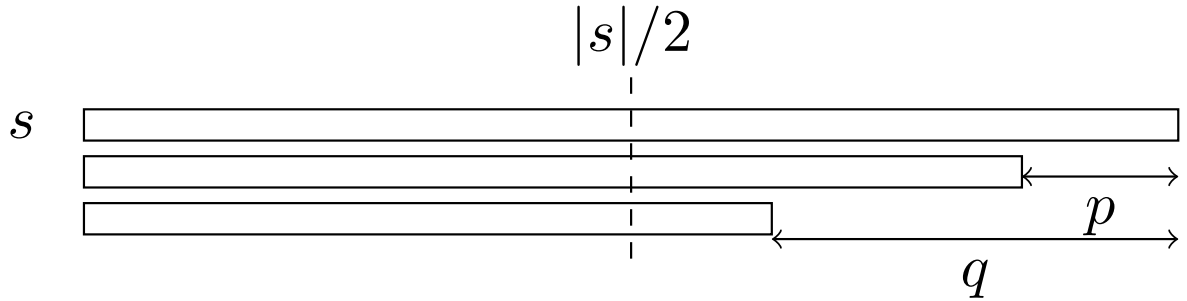
引理: 字符串 u, v 满足 $2|u| \geq |v|$, 则 u 在 v 中的所有匹配位置组成一个等差数列。若这个等差数列至少含有 3 项, 则其公差 d 等于 u 的最小周期 $\mathrm{per}(u)$。此时易知 $\mathrm{per}(u) \leq |u|/2$。

注: 仅含 2 项时不一定有 $\mathrm{per}(u) = d$, 反例为 $u =$aabaa, $v =$aabaaabaa, $\mathrm{per}(u) = 3, d = 4$。

Border 的结构

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。

证明: 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 $\gcd(p,q) \ge p \Rightarrow p \mid q$。



长度小的 border?

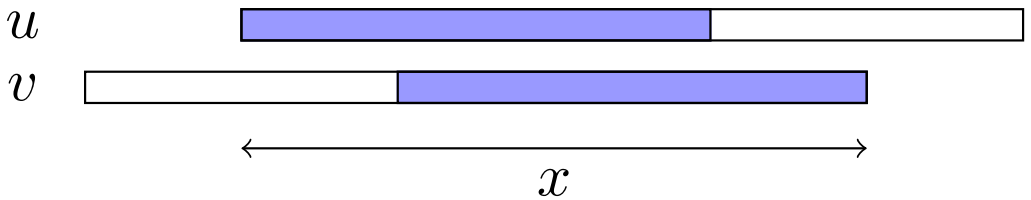
将 $s[1..n]$ 的所有 border 按长度 x 分类:
$x \in [1,2), [2,4), [4,8), \cdots, [2^{k-1}, 2^k), [2^k, n)$。
两种情况:

- $x \in [2^k, n),\ (2^k \geq n/2)$, 已经讨论过
- $x \in [2^{i-1}, 2^i)$

若 $|u| = |v|$, 记 $\mathrm{PS}(u,v) = \{k : \mathrm{pre}(u,k) = \mathrm{suf}(v,k)\}$
记 $\mathrm{LargePS}(u,v) = \{k \in \mathrm{PS}(u,v) : k \geq |u|/2\}$
则 $[2^{i-1}, 2^i)$ 内的 border 长度集合即为
$\mathrm{LargePS}\left(\mathrm{pre}(s,2^i), \mathrm{suf}(s,2^i)\right)$

引理: $\mathrm{LargePS}(u, v)$ 组成一个等差数列。

证明: 设其中最大的元素为 x, 则剩下的元素均为 $\mathrm{pre}(u, x)$ 的 border。



推论: 字符串 s 的所有 border 长度排序后可分成 $O(\log |s|)$ 段, 每段是一个等差数列。

子串周期查询

给定字符串 $s[1..n]$, 每次询问子串 $t = s[l..r]$ 的所有周期（等价于询问所有 border）, 用 $O(\log n)$ 个等差数列表示。

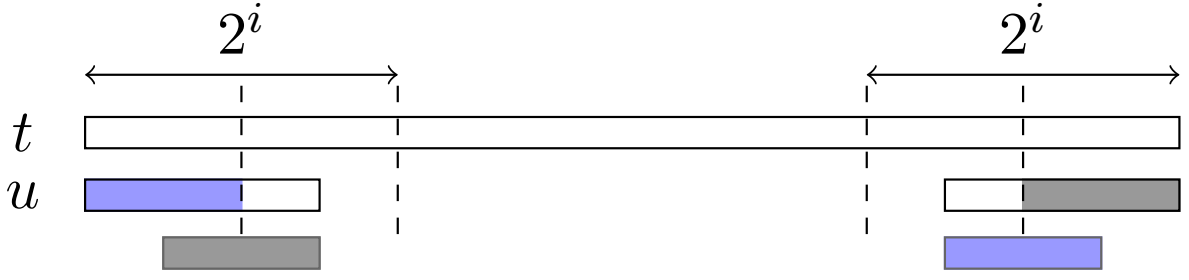
按 border 长度 x 分类,

- $x \in [2^{i-1}, 2^i)$
- $x \in [2^k, m), m = r - l + 1$

Case 1

$x \in [2^{i-1}, 2^i)$, 即计算 $\text{LargePS}\left(\text{pre}(t, 2^i), \text{suf}(t, 2^i)\right)$

若 u 是一个 Large Prefix-Suffix, 则 $\text{pre}(t, 2^{i-1})$ 是 u 的前缀, 且 $\text{suf}(t, 2^{i-1})$ 是 u 的后缀。



求出 $\text{pre}(t, 2^{i-1})$ 在 $\text{suf}(t, 2^i)$ 中匹配的所有位置, 以及 $\text{suf}(t, 2^{i-1})$ 在 $\text{pre}(t, 2^i)$ 中匹配的所有位置。移位后取交集即可。

Internal Pattern Matching (IPM) Queries (Easy)

由于 $|v| \geq |w|/2$, 子串 v 在 w 中匹配的所有位置组成一个等差数列

如何求出这些位置？

求出 v 在 w 中的左边第一次匹配, 第二次匹配, 和最右边一次匹配即可。

也就是要实现一个 $\mathrm{succ}(v, i)$, 能求出 v 在原串 s 中起点不小于 i 的的第一次匹配。（以及反过来的 $\mathrm{pred}(v, i)$ 操作）

注意到, v 的长度是 2 的幂。

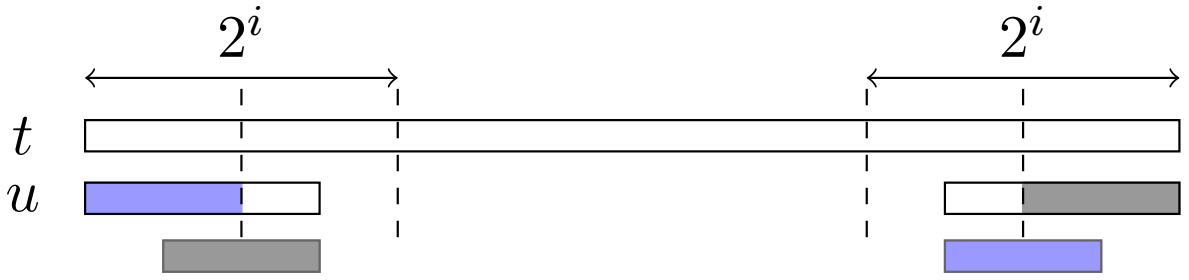
Dictionary of Basic Factors

因此只需把原串 s 中所有长度为 2 的幂的子串拿出来, 相同的子串按起始位置排好序即可。

用类似于倍增求后缀数组的方法。

每次求 $\mathrm{succ}(v, i)$ 时, 在 v 所在的下标列表中二分。

Case 1



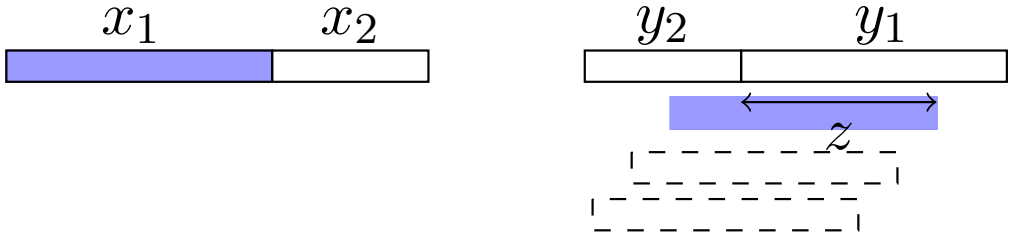
求出 $\mathrm{pre}(t, 2^{i-1})$ 在 $\mathrm{suf}(t, 2^i)$ 中匹配的所有位置，以及 $\mathrm{suf}(t, 2^{i-1})$ 在 $\mathrm{pre}(t, 2^i)$ 中匹配的所有位置。移位后取交集即可。

求两个等差数列的交集？

引理：四个长度相等的字符串 x_1, x_2, y_2, y_1，满足 x_1 在 y_2y_1 中至少匹配 3 次，y_1 在 x_1x_2 中至少匹配 3 次，则 x_1 和 y_1 的最小周期相等。

因此，若两个等差数列数列长度都不少于 3，则它们的公差相等，容易 $O(1)$ 时间求交集。

引理: 四个字符串满足 $|x_1| = |y_1| \geq |x_2| = |y_2|$, 且 x_1 在 y_2y_1 中至少匹配 3 次, y_1 在 x_1x_2 中至少匹配 3 次, 则 x_1 和 y_1 的最小周期相等。



证明: 否则不妨设 $\mathrm{per}(x_1) > \mathrm{per}(y_1)$, 考虑 x_1 在 y_2y_1 中的最右边一次匹配, 设它与 y_1 的重叠部分为 z, 则 $|z| \geq 2\,\mathrm{per}(x_1) > \mathrm{per}(x_1) + \mathrm{per}(y_1)$, 则 z 拥有周期 $d = \gcd(\mathrm{per}(x_1), \mathrm{per}(y_1)) \mid \mathrm{per}(x_1)$, 于是 d 也是 x_1 的周期。但 $d < \mathrm{per}(x_1)$, 矛盾。

Case 2

$L = 2^k \geq m/2$, 求出 $t[1..m]$ 的所有不小于 L 的 border。
即要求 $\mathrm{PS}_{\geq L}(t, t)$。与 Case 1 做法相同。

子串周期查询

于是我们做到了空间 $O(n \log n)$, 预处理时间 $O(n \log n)$, 每次询问 $O(\log^2 n)$ 的在线算法。

小优化?

注意到在 $\mathrm{succ}(v, i)$ 询问中, 我们只关心起点位于 $[i, i + |v|]$ 的匹配。

将 $s[1..n]$ 按 $|v|$ 的间隔分段, 每段中的匹配位置是一个等差数列。

（需要用 hash 表存储每个非空段中的信息）

从而将 IPM 查询的时间降为 $O(1)$。

于是每次子串周期查询时间是 $O(\log n)$, 预处理时间期望 $O(n \log n)$。

Internal Pattern Matching(IPM) Queries

每次给出 s 的子串 u, v, 满足 $2|u| \geq |v|$, 询问 u 在 v 中的所有匹配位置。

对于 u 是 2 的幂的情形, 已经得到了预处理时间期望 $O(n \log n)$, 空间 $O(n \log n)$, 每次询问 $O(1)$ 的做法。

若 u 不是 2 的幂, 令 $L = 2^k \in (|u|/2, |u|)$, 求出 $\mathrm{pre}(u, L)$ 和 $\mathrm{suf}(u, L)$ 的匹配位置, 移位后求交集。时间是 $O(\log n)$。

仍然可以使用 hash 表进行预处理, 实现 $O(1)$ 回答 succ 询问。此时需要优化对等差数列求交集的过程:

取一个 u 的长度为 L 的子串, 使得它与 $\mathrm{pre}(u, L)$ 和 $\mathrm{suf}(u, L)$ 均有不小于 $L/2$ 长度的重叠。若这三个子串都在某个 $2L$ 长度的串中匹配了 ≥ 5 次, 则它们的最短周期都不超过 $L/4$。由重叠长度即可得知它们的最短周期都相等。

循环移位

$s[1..n]$ 有 n 个循环移位, 记为 $\mathrm{cyc}(s,i) = s[i+1..n]s[1..i], (i = 0, 1, \cdots, n-1)$。

若 $s[1..n]$ 的最小周期 p 满足 $p \mid n$, 则只有 p 种不同的循环移位, 每种出现 n/p 次。

若存在 i 使得 $\mathrm{cyc}(s,i) = t$, 则称 s, t 是循环同构串。

s, t 循环同构 $\Leftrightarrow$ 存在串 a, b 使得 $s = ab, t = ba$

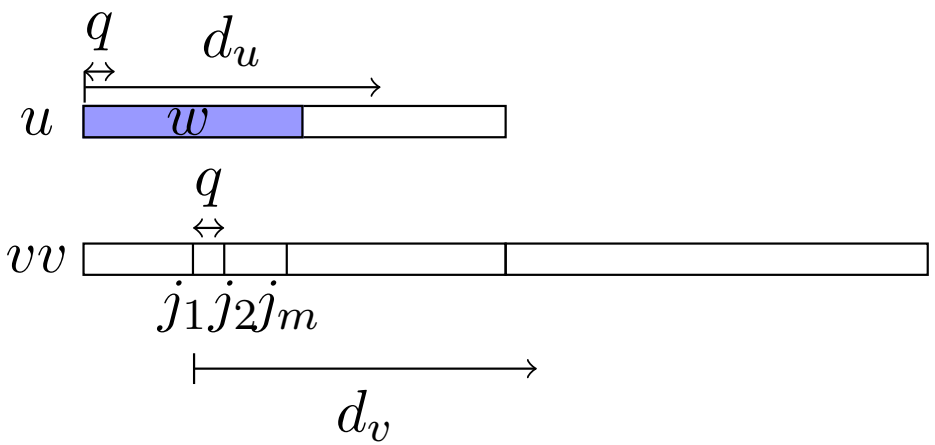
$\Leftrightarrow$ s 是 tt 的子串

子串循环同构判定

每次给出 s 的两个长度相等的子串 u, v, 回答 u 和 v 是否是循环同构串, 是则输出所有可行的移位长度（表示成等差数列）。

设 $u = ab, v = ba$, 不妨令 $|a| \geq |u|/2$。

记 $w = \mathrm{pre}(a, \lceil |u|/2 \rceil)$, IPM 查询 w 在 v 中出现的所有位置 $j_1, j_2, \cdots, j_m$, 是一个公差为 q 的等差数列。



当 $m \geq 3$ 时, $q = \mathrm{per}(w)$。设 u 最长的拥有周期 q 的前缀长度为 d_u, 设 $v[j_1..|v|]v$ 最长的拥有周期 q 的前缀长度为 d_v。

j_i 是合法起始点的必要条件是 $\min(|u|, d_v - (i-1)q) = d_u$。

于是可以 $O(1)$ 回答询问。

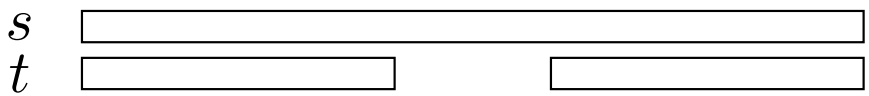
回文串

记 $s^R = s[n]s[n-1]\cdots s[1]$。

回文串 s: 满足 $s = s^R$。

Manacher 算法: $O(n)$ 时间计算以每个位置为中心的最长回文半径。

引理: s 是回文串, 则 s 的后缀 t 是回文串当且仅当 t 是 s 的 border。



推论: 一个字符串的所有回文后缀的长度可以表示成 $O(\log n)$ 个等差数列。

最小回文串拆分

将字符串 s 分解成 $s = s_1 s_2 \cdots s_k$ 使得 s_i 都是回文串, 并使 k 最小。

直接 DP: $f[i] = \min_j\{f[j-1]+1 : s[j..i]\text{是回文串}\}$

时间是 $O(n^2)$

维护集合 $P_i = \{j : s[j..i]\text{是回文串}\}$, 表示为 $O(\log n)$ 段等差数列。

当 $i \leftarrow i+1$ 时, 需要检查每个 j 是否能扩展为回文串 $s[j-1..i+1]$。

对于同一个等差数列中的 j 可以同时判断。

如何对一个等差数列中的 j 同时更新 DP 数组？

如果 P_i 包含 (极长) 等差数列 $j, j+d, \cdots, j+kd$, 则 P_{i-d} 包含（极长）等差数列 $j, j+d, \cdots, j+(k-1)d$。

在计算 $f[i-d]$ 时, 已经算出了 $f_d[i-d] = \min\{f[j-1], f[j+d-1], \cdots, f[j+(k-1)d-1]\}+1$ 的值, 用它来更新 $f_d[i]$ 即可。

时间 $O(n \log n)$。空间可以优化到 $O(n)$。

(非平凡) 回文串拆分

判断 s 能否拆成若干个长度不小于 2 的回文串。

设 $f[i]$ 是以 i 开头的最短（≥ 2）回文串长度。可以 $O(n)$ 计算。

从左往右贪心？

设实际的段长为 d。若 $f[i] \neq d$, 则

- $d/2 \leq f[i] \leq d$, 可以找到更小的前缀回文串
- $f[i] < d/2$, 可将 d 切成 3 段

注意 $d = 2f[i] + 1$ 或 $d = 2f[i] - 1$ 时可能会出现长为 1 的回文串。

以 i 开头的回文串长度只需在 $\{f[i], 2f[i] + 1, 2f[i] - 1\}$ 中选择。DP 即可, 时间 $O(n)$。

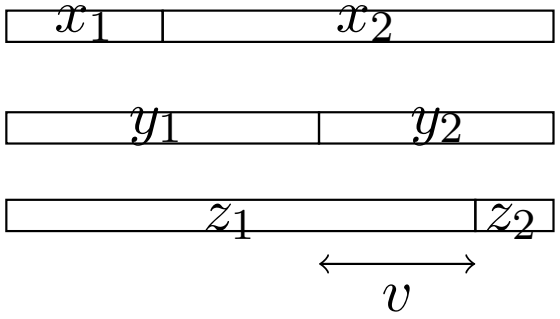
双回文串

如果 $s = ab$, a, b 都是回文串, 则称 s 是一个双回文串 (Palindromic pair)。

引理: 如果 s 是一个双回文串, 则存在一种拆分方法 $s = ab$, 使得 a 是 s 的最长回文前缀, 或者 b 是 s 的最长回文后缀。

引理: 如果 $s = x_1x_2 = y_1y_2 = z_1z_2$, $|x_1| < |y_1| < |z_1|$, x_2, y_1, y_2, z_1 是回文串, 则 x_1 和 z_2 也是回文串。

引理: 如果 $s = x_1x_2 = y_1y_2 = z_1z_2$, $|x_1| < |y_1| < |z_1|$, x_2, y_1, y_2, z_1 是回文串, 则 x_1 和 z_2 也是回文串。



证明: 设 $z_1 = y_1v$。

v 是 x_2 的前缀, 于是 x_1v 是 z_1 的前缀。

$|v|$ 是 z_1 的周期, 因此也是 x_1v 的周期。可知 x_1 是 v^∞ 的后缀。

v^R 是 z_1 的前缀, 因此 x_1 是 $(v^R)^\infty$ 的前缀。

所以 x_1 是回文串。

判断 s 能否拆成不超过 4 个回文串。

只要判断每个前缀、后缀是否是双回文串。

算出以每个点为开头（或结尾）的最长回文串长度。

$O(n)$ 时间。

这个方法只能解决 $k \leq 4$ 的情形。有其他算法可以对更大的 k 进行判断。

字典序

许多字符串问题会涉及到字符的大小顺序, $\mathtt{a} < \mathtt{b} < \cdots < \mathtt{z}$

有时为了方便会添加最小的和最大的字符。这里记作 0 和 ∞

字典序: $s < t$ 当且仅当 $s \neq t$ 且 s 是 t 的前缀, 或者存在 i 使得 $s[1..i-1] = t[1..i-1]$ 且 $s[i] < t[i]$。

用 $s \ll t$ 表示 $s < t$ 且 s 不是 t 的前缀。

后缀数组与字典序

预处理后缀数组之后, $O(1)$ 回答

- 比较两个子串大小
- $s[i..n], s[i+1..n], s[i+2..n], \cdots, s[j..n]$ 中最小者

最小、最大后缀和最小、最大循环移位

记 $t = ss$, 若 $t[i+1..2n]$ 是 t 的最大后缀, 则 $\mathrm{cyc}(s,i) = s[i+1..n]s[1..i] = t[i+1..i+n]$ 是 s 的最大循环移位。

最小循环移位 $\leftrightarrow$ 最大循环移位

最小、最大后缀可以由后缀数组直接得到

子串最小后缀查询

给定字符串 $s[1..n]$, 每次询问子串 $s[l..r]$ 的最小后缀。

子串最小后缀查询

给定字符串 $s[1..n]$, 每次询问子串 $s[l..r]$ 的最小后缀, 记为 $\mathrm{minsuf}(l, r)$。

引理: 设 $s[p..n]$ 是 $s[i..n], (l \le i \le r)$ 中最小者, 则 $\mathrm{minsuf}(l, r)$ 等于 $s[p..r]$ 的最短非空 border。

例: $s = \mathtt{cabacabaa}$, $s[5..8] = s[1..4] = \mathtt{caba}$

证明:

这给出了一个每次询问 $O(\log n)$ 的做法。

子串最小后缀查询

设 $s[p..n]$ 是 $s[i..n], (l \le i \le r)$ 中最小者

设 $m = 2^k$, 且 $m < r - l + 1 \le 2m$。

则 $\mathrm{minsuf}(l, r) = \min\{s[p..r], \mathrm{minsuf}(r - m + 1, r)\}$

证明: 因为非 border-free 的串的最短非空 border 的长度不超过串长的一半。

这给出了预处理时间空间 $O(n \log n)$, 询问 $O(1)$ 的做法。

子串最大后缀查询

最大后缀 $\leftrightarrow$ 最小后缀

$\mathrm{minsuf}(\mathtt{abaa}) = \mathtt{a}, \mathrm{maxsuf}(\mathtt{zyzz}) = \mathtt{zz}$

$\mathrm{minsuf}(\mathtt{abaa}\infty) = \mathtt{aa}\infty$

给定字符串 $s[1..n]$, 每次询问子串 $s[l..r]$ 的最大后缀。

子串最大后缀查询

给定字符串 $s[1..n]$, 每次询问子串 $s[l..r]$ 的最大后缀。

取 $m = 2^k$ 使得 $m < r - l + 1 \leq 2m$。

求一个 p, 使得 $\mathrm{maxsuf}(l, r) = \max\{s[p..r], \mathrm{maxsuf}(r - m + 1, r)\}$ 成立。

设 $\mathrm{maxsuf}(l, r) = s[x..r]$。只需考虑 $l \leq x < r - m + 1$ 的情形, 即需要求出 x。

（询问 $s[1..n]$ 的子串 $s[l,r]$, $m < r-l+1 \le 2m$。$\mathrm{maxsuf}(l,r) = s[x..r]$, 要对 $l \le x < r-m+1$ 情形求出 x）

引理: 设 $s[p_1..n]$ 是 $s[i..n], (l \le i < r-m+1)$ 中最大的。则 $s[p_1..r]$ 是 $s[x..r]$ 的前缀。

于是只需考虑 $l < p_1$ 的情形（否则 $x = p_1$）。

设 $s[p_2..n]$ 是 $s[i..n], (l \le i < p_1)$ 中最大的。

如果 $s[p_1..r] > s[p_2..r]$ 则 $x = p_1$。否则 $s[p_1..r]$ 是 $s[p_2..r]$ 的前缀, 且 $s[p_2..r]$ 是 $s[x..r]$ 的前缀。

$s[p_1..r]$ 是 $s[p_2..r]$ 的前缀, 且 $s[p_2..r]$ 是 $s[x..r]$ 的前缀。

$q = p_1 - p_2$ 是 $s[p_2..r]$ 的周期, 因此也是 $s[x..r]$ 的周期。

由 $s[p_1..n] > s[p_2..n]$ 知 $s[r+1..n] > s[r+1-q..n]$

由 p_2 的定义可知 $q \mid p_2 - x$

求 $s[1..p_2-1]$ 和 $s[1..p_1-1]$ 的最长公共后缀, 即可得到周期 q 往左最远延伸的位置。时间 $O(1)$。

因此同样得到了预处理时间空间 $O(n \log n)$, 询问 $O(1)$ 的做法。

Lyndon word

满足 s 的最小后缀等于 s 本身的串 s 称为 Lyndon 串。
等价于: s 是它自己的所有循环移位中唯一最小的一个。
引理: u, v 是 Lyndon 串, 且 $u < v$, 则 uv 是 Lyndon 串。
证明: 先证 $v > uv$。

Chen-Fox-Lyndon Theorem

任意字符串 s 可以分解为 $s = s_1 s_2 \cdots s_k$, 其中 s_i 是 Lyndon 串, $s_i \geq s_{i+1}$。且这种分解方法是唯一的。

存在性: 初始时每段一个字符。不断地将相邻两段 $s_i < s_{i+1}$ 合并。

唯一性: 若有两种方案, 取第一次不同的位置, 设 $|s_i| > |s'_i|$, 令 $s_i = s'_i s'_{i+1} \cdots s'_k \mathrm{pre}(s'_{k+1}, l)$, 则
$s_i < \mathrm{pre}(s'_{k+1}, l) \leq s'_{k+1} \leq s'_i < s_i$。

Duval 算法

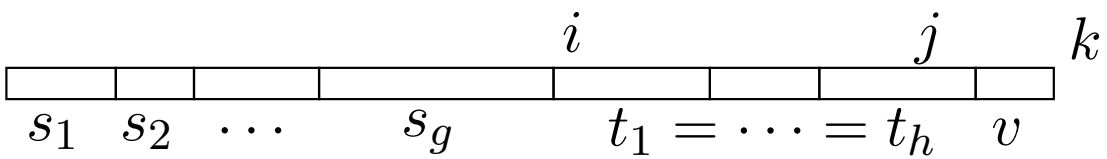
$O(n)$ 时间, $O(1)$ 额外空间, 求出 $s[1..n]$ 的 Lyndon Decomposition。（规定 $s[n+1]=0$）

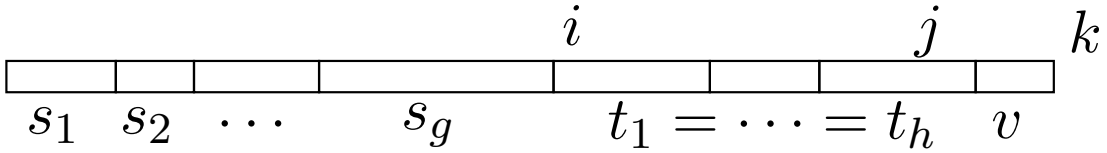
引理: 若字符串 v 和字符 c 满足 vc 是某个 Lyndon 串的前缀, 则对于字符 $d>c$ 有 vd 是 Lyndon 串。

三个循环变量 i, j, k, 维持一个循环不变式:

- $s[1..i-1] = s_1 s_2 \cdots s_g$ 是已经固定下来的分解, 满足 s_l 是 Lyndon 串, 且 $s_l \geq s_{l+1}$。
- $s[i..k-1] = t_1 t_2 \cdots t_h v (h \geq 1)$ 是没有固定的分解, 满足 t_1 是 Lyndon 串, $t_1 = t_2 = \cdots = t_h$, v 是 t_h 的（可为空的）真前缀, 且有 $s_g \gg s[i..k-1]$。

当前读入的字符是 $s[k]$。令 $j = k - |t_1|$。





- 当 $s[k] = s[j]$ 时, 周期 $k - j$ 继续保持。
- 当 $s[k] > s[j]$ 时, 合并得到 $t_1 \leftarrow t_1 t_2 \cdots t_h v s[k]$ 是 Lyndon 串。
- 当 $s[k] < s[j]$ 时, $t_1, t_2, \cdots, t_h$ 的分解被固定下来。算法从 v 的开头处重新开始。

复杂度分析: i 只往右移。每次 k 的左移距离不超过 i 的右移距离。

```
1:  input(s[1..n])
2:  i ← 1
3:  while i ≤ n do
4:      j ← i
5:      k ← i + 1
6:      while k ≤ n and s[j] ≤ s[k] do
7:          if s[j] < s[k] then
8:              j ← i
9:          else
10:             j ← j + 1
11:         end if
12:         k ← k + 1
13:     end while
14:     while i ≤ j do
15:         output(s[i..i + k − j − 1])
16:         i ← i + k − j
17:     end while
18: end while
```

用 Duval 算法求出 s 的最小、最大后缀, 最小循环移位。
另外, 可以 $O(n)$ 对所有 $s[1..i]$ 求出其最小、最大后缀。

参考文献

Crochemore, Maxime, Wojciech Rytter, and Maxime Crochemore. Text algorithms. Vol. 698. New York: Oxford University Press, 1994.

Kociumaka, Tomasz, et al. "Efficient data structures for the factor periodicity problem." International Symposium on String Processing and Information Retrieval. Springer Berlin Heidelberg, 2012.

Kociumaka, Tomasz, et al. "Internal pattern matching queries in a text and applications." Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2015.

Crochemore, Maxime, et al. "New simple efficient algorithms computing powers and runs in strings." Discrete Applied Mathematics 163 (2014): 258-267.

Fici, Gabriele, et al. "A subquadratic algorithm for minimum palindromic factorization." Journal of Discrete Algorithms 28 (2014): 41-48.

Galil, Zvi, and Joel Seiferas. "A Linear-Time On-Line Recognition Algorithm for 'Palstar'." Journal of the ACM (JACM) 25.1 (1978): 102-111.

Kosolobov, Dmitry, Mikhail Rubinchik, and Arseny M. Shur. "Pal k is linear recognizable online." International Conference on Current Trends in Theory and Practice of Informatics. Springer Berlin Heidelberg, 2015.

Babenko, Maxim, et al. "Computing minimal and maximal suffixes of a substring revisited." Symposium on Combinatorial Pattern Matching. Springer International Publishing, 2014.

Kociumaka, Tomasz. "Minimal Suffix and Rotation of a Substring in Optimal Time." arXiv preprint arXiv:1601.08051 (2016).

Chen, Kuo Tsai, Ralph H. Fox, and Roger C. Lyndon. "Free differential calculus, IV. The quotient groups of the lower central series." Annals of Mathematics (1958): 81-95.

Duval, Jean Pierre. "Factorizing words over an ordered alphabet." Journal of Algorithms 4.4 (1983): 363-381.

Apostolico, Alberto, and Maxime Crochemore. "Optimal canonization of all substrings of a string." Information and Computation 95.1 (1991): 76-95.

感谢叉姐对我的指导。
感谢各位老师同学的支持。
祝大家考出好成绩！