

# 字符串算法

hz

2022-01-28

# 符号介绍

字符集:  $\Sigma = \{'a', 'b', \dots, 'z'\}$

字符串:  $s = s_1 s_2 \dots s_n, |s| = n$

子串:  $\text{substring}(s) = \{s[l, r] \mid 1 \leq l \leq r \leq n\}, s[l, r] = s_l, s_{l+1}, \dots, s_r$

字典序:  $s < t \iff (s_1 = t_1 \vee s[2, |s|] < t[2, |t|])$

前缀:  $\text{prefix}(s) = \{s[1, i] \mid 1 \leq i \leq n\}$

最长公共前缀:  $\text{lcp}(s, t) = \max\{i \mid s[1, i] = t[1, i]\}$

后缀:  $\text{suffix}(s) = \{s[i, n] \mid 1 \leq i \leq n\}$

最长公共后缀:  $\text{lcs}(s, t) = \max\{i \mid s[|s| - i + 1, |s|] = t[|t| - i + 1, |t|]\}$

# 字符串哈希

哈希能将一个字符串映射到一个整数上，一般用于判断字符串是否相等

通常使用的哈希方法是：

$$(\text{Hash}(s) = s_1 c + s_2 c^2 + s_3 c^3 + \cdots + s_n c^n) \bmod p$$

注意：使用自然溢出取模的哈希是可以被攻击的

哈希最常用的例子是二分 + 哈希  $O(\log n)$  求 lcp

# border

定义字符串的 border 集合为前缀等于后缀的位置，next 为 border 中的最大值，形式化地：

$$\text{border}(s) = \{i \mid s[1, i] = s[n - i + 1, n], 1 \leq i < n\}$$

$$\text{next}(s) = \begin{cases} \max\{\text{border}(s)\} & \text{border}(s) \neq \emptyset \\ -1 & \text{border}(s) = \emptyset \end{cases}, \text{next}_i = \text{next}(s[1, i])$$

例如： $\text{border}(\text{"abaabaab"}) = \{2, 5\}$

border 集有很多优秀的性质，这里只讨论其中比较常见的性质：  
(比较复杂的例如：弱/强周期定理、等差数列划分等，详见金策《字符串算法选讲》)

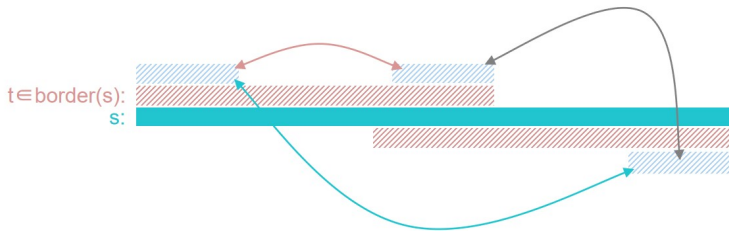
# border

包含关系:

$$\forall i < j, j \in \text{border}(s) :$$

$$i \in \text{border}(s) \iff s[1, i] = s[n - i + 1, n]$$

$$\iff s[1, i] = s[j - i + 1, j] \iff i \in \text{border}(s[1, j])$$



因此,

$$\text{border}(s) = \text{border}(s[1, \text{next}(s)]) \cup \text{next}(s)$$

$$\text{border}(s) = \{\text{next}_n, \text{next}_{\text{next}_n}, \dots\}$$

例如:

$$\text{border}(\text{"a"}) = \{\}$$

$$\text{border}(\text{"ab"}) = \{\}$$

$$\text{border}(\text{"aba"}) = \{1\}$$

$$\text{border}(\text{"abaa"}) = \{1\}$$

$$\text{border}(\text{"abaab"}) = \{2\}$$

$$\text{border}(\text{"abaaba"}) = \{1, 3\}$$

$$\text{border}(\text{"abaabaa"}) = \{1, 4\}$$

$$\text{border}(\text{"abaabaab"}) = \{2, 5\}$$

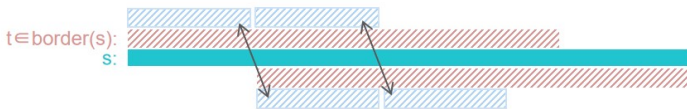
# border

定义字符串的周期为某个循环出现的子串：

$$\text{period}(s) = \{t \mid \forall 1 \leq i \leq n - t, s_i = s_{i+t}\}$$

border 与周期有对应关系：

$$\begin{aligned} t \in \text{period}(s) &\iff \forall 1 \leq i \leq n - t, s_i = s_{i+t} \\ &\iff s[1, n - t] = s[t + 1, n] \iff n - t \in \text{border}(s) \end{aligned}$$



## border

相邻关系:

考虑  $\text{border}(s[1, j-1])$  和  $\text{border}(s[1, j])$  之间的关系:

$$\begin{aligned}i \in \text{border}(s[1, j]) &\iff s[1, i] = s[j-i+1, j] \\&\implies s[1, i-1] = s[j-i+1, j-1] \\&\iff i-1 \in \text{border}(s[1, j-1])\end{aligned}$$

因此,  $\text{next}_j \in (\{-1\} \cup \{i \mid i-1 \in \text{border}(s[1, j-1])\})$

从  $\text{next}_{j-1}$  转移到  $\text{next}_j$  只需要在  $j-1$  的 border 集中查找比较, 找到最大的合法值即可

又因为  $s[1, i-1] = s[j-i+1, j-1]$ , 比较时只需判断  $[s_i == s_j]$

每次比较过后, 有两种情况:

- ▶ 1.  $\text{next}_j + 1, j + 1$
- ▶ 2.  $\text{next}_j$  减少

能在均摊  $O(n)$  复杂度下递推求出  $\text{next}_1 \dots \text{next}_n$

# KMP 算法

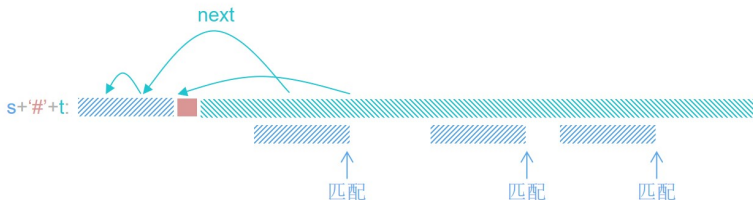
KMP 算法能  $O(|s|)$  预处理,  $O(|t|)$  找到  $s$  在  $t$  中出现的所有位置

利用 border 性质的简单证明:

考虑字符串  $s + \text{'\#'} + t$  的 next 值, 其中  $\text{'\#'}$  是一个字符集  $\Sigma$  以外的字符:

由 border 定义,  $\text{next}_{|s|+1+j} = \max\{i \mid s[1, i] = t[j - i + 1, j]\}$ ,

$s$  在  $t[j - |s| + 1, j]$  中出现  $\iff \text{next}_{|s|+1+j} = |s|$





# KMP 算法

尝试从匹配状态的角度证明 KMP 算法：

假设当前已经扫描到了  $j$

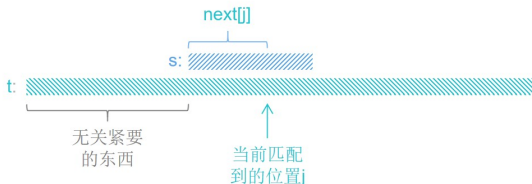
(已经考虑完了所有  $j' < j$ ,  $[s == t[j' - |s| + 1, j']]$  的匹配)

我们发现字符串  $t$  最前面的字符对于当前及之后的匹配是没有影响的

形式化地：从小到大考虑每一个位置  $i$ ，如果  $\nexists x \geq 0, t[i, j+x] = s$ ，那么  $t_i$  的值对当前及之后的匹配没有影响

由于  $j$  之后的字符还未扫描，我们将条件放宽到： $t[i, j] \in \text{prefix}(s)$

此时，我们只需要储存  $\max\{i \mid t[j-i+1, j] = s[1, i]\}$ ，这个值能唯一描述当前所需的所有信息，可以发现这就是  $\text{next}_j$ ，递推与动态规划思想相同



# AC 自动机

trie 树：用树储存字符串集合  $s_1, s_2, \dots, s_n$ ，每个前缀  $p \in \text{prefix}(s_x)$  唯一对应一个节点  $\text{trans}(p)$

AC 自动机：在 trie 树上的字符串匹配算法：

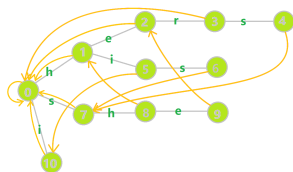
$O(\sum |s_x|)$  预处理， $O(|t|)$  找到  $s_x$  在  $t$  中出现的所有位置

与匹配状态的角度 KMP 证明同理，匹配时只需要储存当前最大匹配前缀的信息  $\text{trans}(t[j-i+1, j])$

其中  $i = \max\{i \mid t[j-i+1, j] \in \text{prefix}(s_x)\}$

AC 自动机中同样存在与 next 相似的 fail 指针，指向 trie 树中的节点， $\text{fail}_{\text{trans}(p)} = \text{trans}(q)$  说明  $q$  为出现在 trie 树中最长的  $p$  的后缀（还可以通过储存  $\text{fail}_{p,c}$  得到非均摊复杂度的查询算法）

唯一与 next 不同的是，AC 自动机中的 fail 递推时，长度短的前缀的 fail 必须比长度长的前缀的 fail 先求出，因此要使用队列

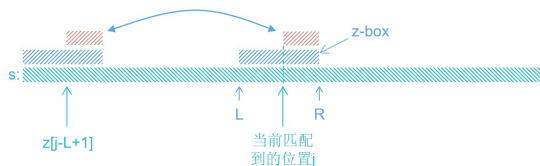


## 扩展 KMP (Z 函数)

扩展 KMP 算法能  $O(n)$  求出,  $z_1, z_2 \dots z_n$  其中  $z_i = \text{lcp}(s, s[i, n])$

假设现在正在计算  $z_j$ , 定义当前的 z-box  $[L, R] = [i, i + z[i] - 1]$ , 其中  $i + z[i] - 1$  最大, (根据 z 函数的定义, 显然有  $s[1, z[i]] = s[L, R]$ )

可以发现在整个算法中  $R$  的值递增



因此,  $s[j, R] = s[j-L+1, R-L+1]$ , 对  $z[j-L+1]$  分两类讨论:

- ▶ 1.  $z[j-L+1] < R-j+1$ ,  $z[j] = z[j-L+1]$
- ▶ 2.  $z[j-L+1] \geq R-j+1$ ,  $z[j] \geq R-j+1$ , 手动比较 lcp, 每次比较会让接下来的  $R+1$ , 复杂度  $O(n)$

# Manacher 算法

回文串：正向读与反向读相同的字符串

$$s \in \text{palindrome} \iff \forall 1 \leq i \leq n, s_i = s_{n-i+1}$$

定义字符串的反转操作： $s_i^R = s_{n-i+1}$ ，有  $s \in \text{palindrome} \iff s = s^R$

回文串的两边删除一个字符还是回文串：

$$n \geq 2, s \in \text{palindrome} \implies s[2, n-1] \in \text{palindrome}$$

长度为奇数的回文串  $s$  有回文中心  $s_{\frac{n+1}{2}}$ ，而长度为偶数的回文串没有

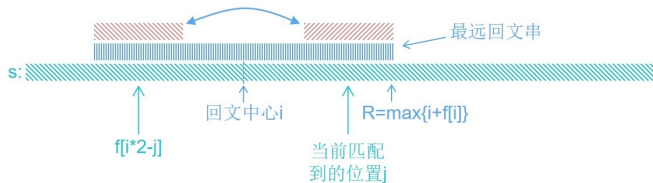
因此，对字符串作变换  $s_1 s_2 \dots s_n \implies s_1 \# s_2 \# \dots \# s_n$ ，使得原来的每一个回文串都有回文中心

Manacher 算法：  $O(n)$  对变换后的字符串求出

$$f_i = \max\{l \mid s[i-l, i+l] \in \text{palindrome}\}$$

# Manacher 算法

采用与扩展 kmp 中 z-box 相似的思路, 假设现在正在计算  $f_j$ , 当前找到的最远 ( $i + f_i$  最大) 的回文子串为  $s[i - f_i, i + f_i]$



因此,  $s[2j - R, R]^R = s[2i - R, 2i - (2j - R)]$ , 同样对  $f_{i*2-j}$  分两类讨论:

- ▶ 1.  $f_{i*2-j} < R - j$ ,  $f_j = f_{i*2-j}$
- ▶ 2.  $f_{i*2-j} \geq R - j$ ,  $f_j \geq R - j$ , 手动比较是否回文, 每次比较会让接下来的  $R + 1$ , 复杂度  $O(n)$

这也说明了字符串  $s$  中本质不同的回文子串数  $\leq n$

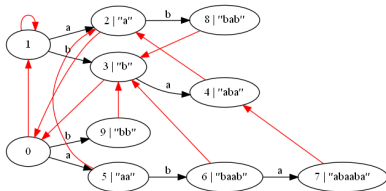
# 回文自动机 (PAM)

注意到回文串还有另一个性质：

$s \in \text{palindrome}, t \in \text{suffix}(s) : t \in \text{palindrome} \iff t \in \text{prefix}(s)$

定义回文自动机：

- ▶ 每个本质不同的回文子串对应一个节点，此外还有两个空节点 0, -1
- ▶  $\text{nxt}(s, c) = c + s + c$ , 特殊地,  $\text{nxt}(-1, c) = c$ ,  $\text{nxt}(0, c) = cc$
- ▶  $\text{fail}(s) = s[1, \text{next}(s)]$ , 由刚才的结论，回文串的所有回文后缀就是其所有 border



设以  $j$  为右端点的最长回文子串为  $p$ ，那么以  $j$  为右端点的所有回文子串就是  $p, \text{fail}_p, \text{fail}_{\text{fail}_p}, \dots$

# 回文自动机 (PAM)

构造:

设当前扫描到  $j$ , 只需要新增以  $j$  为右端点的所有回文串, 而除了以  $j$  为右端点的最长回文子串外, 其他回文子串一定已经出现过了

$s[i, j] \in \text{palindrome} \implies s[i+1, j-1] \in \text{palindrome}$ , 在  $j-1$  对应节点上跳 fail 查找即可

manacher 将回文子串统计到它们的回文中心, PAM 将回文子串统计到它们的右端点

而回文后缀与 border 之间的联系使得 PAM 能利用 border 的高级性质处理一些问题

# 后缀数组 (SA)

$O(n \log n)$  对  $s$  的所有后缀进行排序, 算法原理是做  $\log n$  次双关键字基数排序

有  $O(n)$  算法但比较复杂

将所有字符串按排序后顺序并列,

定义  $SA[i]$  为排名为  $i$  的后缀,  $rk[i]$  为后缀  $s[i, n]$  的排名

$h[i] = \text{lcp}(s[SA[i], n], s[SA[i-1], n])$

S = babba

a	suffix(5), rank=1
abba	suffix(2), rank=2, H=1
ba	suffix(4), rank=3, H=0
babba	suffix(1), rank=4, H=2
bba	suffix(3), rank=5, H=1

rk = [4,2,5,3,1]  
SA = [5,2,4,1,3]  
H = [1,0,2,1]



# 后缀数组 (SA)

可以  $O(n)$  求  $h_1, h_2, \dots, h_n$

常用性质:

$$\text{lcp}(SA[i], SA[j]) = \min_{k=i+1}^j h[k]$$

同一字符串出现位置为 SA 中的一段区间

S = babba

a	suffix(5), rank=1
abba	suffix(2), rank=2, H=1
ba	suffix(4), rank=3, H=0
babba	suffix(1), rank=4, H=2
bba	suffix(3), rank=5, H=1

rk = [4,2,5,3,1]

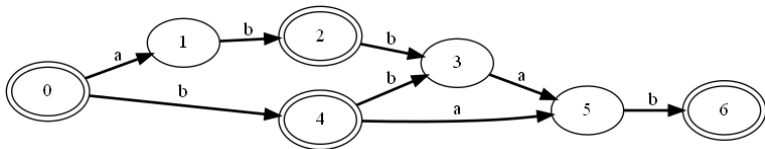
SA = [5,2,4,1,3]

H = [1,0,2,1]

# 后缀自动机 (SAM)

确定有限自动机 (DFA) 可用一个五元组  $\langle Q, \Sigma, \delta, q_0, F \rangle$  表示:

- ▶  $Q$ : 状态集合
- ▶  $\Sigma$ : 符号的有限集合, 自动机接受的“字母表”
- ▶  $\delta$ : 转移函数, 在某一状态下, 接受字母, 转移到唯一新状态 ( $\delta: (Q, \Sigma) \rightarrow Q$ ) (下文记作  $\text{nxt}(q, x)$ )
- ▶  $q_0$ : 开始状态 ( $q_0 \in Q$ )
- ▶  $F$ : 终止状态集合 ( $F \subset Q$ , 对于可识别字符串, 自动机一定停止于  $F$  中的某个状态)



上图是一个恰能识别“abbab”, “bbab”, “bab”, “ab”, “b”, “” 的 DFA

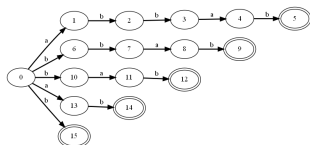
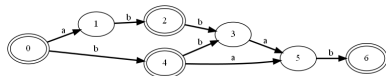
# 后缀自动机 (SAM)

$\text{nxt}(q, x)$  表示状态  $q$  接受字符  $x$  后转移到的节点

$\text{trans}(q, s)$  表示状态  $q$  接受字符串  $s$  后转移到的节点

定义状态  $q$  可以识别的字符串为  $L(q) = \{s \mid \text{trans}(q, s) \in F\}$ , 自动机能识别的集合就是  $L(q_0)$

有一些不同形状的 DFA 能识别相同的字符串集合, 但能识别指定字符串集合的最简 DFA 唯一



在最简 DFA 中,  $L(q)$  不为空且两两不同, 可证明最简 DFA 唯一  
形式化地,

$$\forall L_0, (\exists q, L(q) = L_0) \iff (\exists t, L_0 = \{s \mid s \in L(q_0), t \in \text{prefix}(s)\})$$

定义字符串  $s$  的后缀自动机 (SAM) 为  $L(q_0)$  为  $s$  所有后缀的最简 DFA, 可以证明 SAM 的节点数  $\leq 2n$

# 后缀自动机 (SAM)

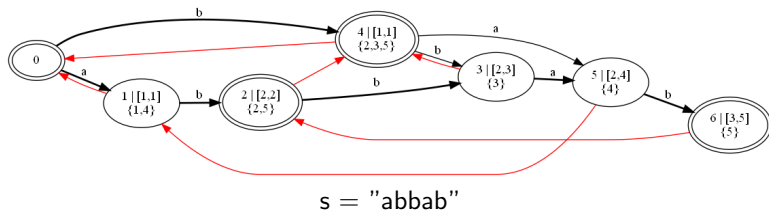
首先, 根据最简 DFA 性质可以得到:

$$\text{trans}(q_0, s) \in F \iff s \in \text{suffix}(S)$$

$$\text{trans}(q_0, s) \neq \text{null} \iff s \in \text{substring}(S)$$

$\text{right}(q)$  是  $q$  能识别的后缀的左端点集合 (假设  $q$  能识别  $s[r_1 + 1, n], s[r_2 + 1, n] \dots s[r_m + 1, n]$ , 则  $\text{right}(q) = \{r_1, r_2, \dots, r_m\}$ )

若  $\text{trans}(q_0, s[l, r]) = q$ , 因为  $\text{trans}(q_0, s[l, n]) \in F$ , 所以一定有  $\text{trans}(q, s[r + 1, n]) \in F, r \in \text{right}(q)$



## 后缀自动机 (SAM)

上文已得出  $\text{trans}(q_0, s[l, r]) = q \implies r \in \text{right}(q)$ , 现在考虑对  $l$  的要求:

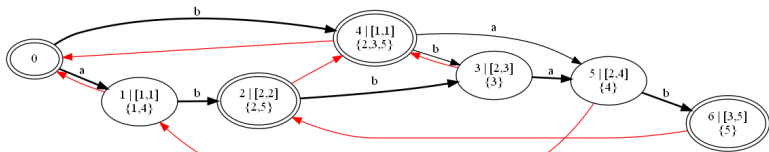
注意到  $\text{right}(\text{trans}(q_0, s[l_1, r])) \subseteq \text{right}(\text{trans}(q_0, s[l_2, r]))$ , 因此满足  $\text{trans}(q_0, s[l, r]) = q$  的  $l$  一定在某个区间内

综上,  $\text{trans}(q_0, s[l, r]) = q \iff r \in \text{right}(q) \wedge r - l \in [\min(q), \max(q)]$

这说明了  $\text{trans}(q_0, s) = q$  的本质不同的字符串有  $\max(q) - \min(q) + 1$  个, 每个出现了  $|\text{right}(q)|$  次

(一个没什么用但有助于理解的结论:

$$\sum_{q \neq q_0} [\max(q) - \min(q) + 1] \times |\text{right}(q)| = \frac{n(n-1)}{2}$$



$s = \text{"abbab"}$

# 后缀自动机 (SAM)

发现任意两个状态的  $\text{right}$  集合不交或包含，这形成了一颗树，称为  $\text{parent}$  树， $q$  在  $\text{parent}$  树上的父亲记作  $\text{pre}(q)$

$\text{parent}$  树还有这些性质：

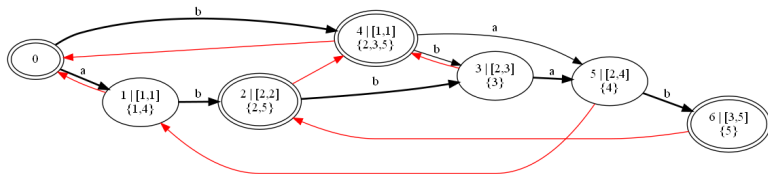
$$\text{right}(q) \subset \text{right}(\text{pre}(q))$$

$$\min(q) = \max(\text{pre}(q)) + 1$$

$\text{trans}(q_0, s[l, r])$  的值在  $\text{parent}$  树的一条链上，随  $l$  增大，深度连续、递减（例如我们可以通过倍增  $O(\log)$  计算  $\text{trans}(q_0, s[l, r])$ ）

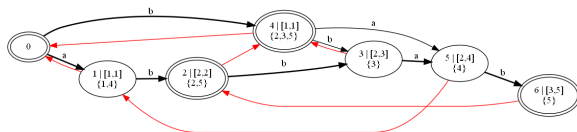
$\text{pre}$  指针具有类似  $\text{fail}$  的作用

$\text{trans}(q_0, s[l, r])$  最深的节点是  $\text{trans}(q_0, s[1, r])$  这很容易计算



$s = \text{"abbab"}$

# SAM 建后缀树



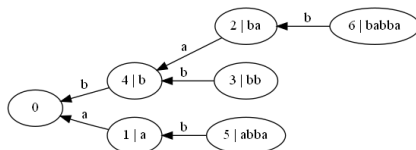
$s = \text{"abbab"}$

$\text{trans}(q_0, s[l_1, r]) = q, \text{trans}(q_0, s[l_2, r]) = \text{pre}(q)$

$s[l_1, r]$  与  $s[l_2, r]$  有相同后缀, 在 parent 树上是父子关系

因此, 对反串  $s^R$  建 SAM, 得到的 parent 树就是  $s$  的后缀树

(例如可用于求后缀之间的 lcp)



$\text{SAM}(\text{"abbab"}), s = \text{"babba"}$

注意到儿子之间的顺序只由一个位置的字符决定, 且两两不同, 可以通过这个对儿子排序