

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

# 字符串

zsy

2022 年 2 月 6 日

- 用  $\Sigma$  表示字符集,  $\sigma = |\Sigma|$  表示字符集大小, 一般默认是常数。
- 用  $\Sigma^n$  表示字符集  $\Sigma$  下所有长度为  $n$  的串构成的集合。
- 用  $\epsilon$  表示空串。注意区分  $\{\epsilon\}$  和  $\emptyset$ 。
- 若无特殊说明, 认为字符串下标从 1 开始<sup>1</sup>。
- 用  $S + T$  或者  $ST$  表示字符串  $S$  与字符串  $T$  的拼接。
- 用  $|S|$  表示字符串  $S$  的长度。
- 用  $S_i$  表示字符串  $S$  的第  $i$  个字符。
- 用  $S_{[i,j]}$  表示字符串  $S$  第  $i$  个字符到第  $j$  个字符组成的子串。
- 用  $\text{lcp}(i, j)$  表示  $S_{[i, |S|]}$  与  $S_{[j, |S|]}$  的最长公共前缀 (的长度)。
- 用  $\text{lcs}(i, j)$  表示  $S_{[1, i]}$  与  $S_{[1, j]}$  的最长公共后缀 (的长度)。

<sup>1</sup> 这是因为我不喜欢出现  $-1$  这样的记号。理解万岁。

- 最小循环表示法
- Manacher
- ex-KMP
- Hash
- KMP
- Trie
- Aho-Corasick Automaton
- Suffix Array
- Suffix Automaton
- Palindrome Automaton
- 习题课

部分内容超出了 NOI 大纲，所以就当听着图一乐吧。

## 最小循环表示法

给一个字符串  $S$ ，求一个  $i \in [1, |S|]$  使  $S_{[i, |S|]} + S_{[1, i-1]}$  的字典序最小，要求线性时间复杂度。

将  $S$  倍长，维护两个指针  $i, j$  表示最小循环表示的起始位置，暴力求出  $\text{lcp}(i, j)$ ，设其为  $k$ ，则比较  $S_{i+k}$  和  $S_{j+k}$ ，不失一般性地假设  $S_{i+k} < S_{j+k}$ ，则  $[j, j+k]$  范围内的所有下标均不可能成为最小循环表示的起始位置<sup>2</sup>，直接令  $j \leftarrow j+k+1$  后继续比较，直至  $[1, n]$  内仅剩下一个起始位置。

由于指针  $i, j$  始终是递增的，因而该算法的时间复杂度为  $O(|S|)$ 。

<sup>2</sup> 想一想，为什么？

## 最小循环表示法

给一个字符串  $S$ , 求一个  $i \in [1, |S|]$  使  $S_{[i, |S|]} + S_{[1, i-1]}$  的字典序最小, 要求线性时间复杂度。

将  $S$  倍长, 维护两个指针  $i, j$  表示最小循环表示的起始位置, 暴力求出  $\text{lcp}(i, j)$ , 设其为  $k$ , 则比较  $S_{i+k}$  和  $S_{j+k}$ , 不失一般性地假设  $S_{i+k} < S_{j+k}$ , 则  $[j, j+k]$  范围内的所有下标均不可能成为最小循环表示的起始位置<sup>2</sup>, 直接令  $j \leftarrow j+k+1$  后继续比较, 直至  $[1, n]$  内仅剩下一个起始位置。

由于指针  $i, j$  始终是递增的, 因而该算法的时间复杂度为  $O(|S|)$ 。

<sup>2</sup> 想一想, 为什么?

以下的  $s$  已经被倍长,  $n$  表示原串长。

```
int i = 1, j = 2;
while (i <= n && j <= n) {
    int k = 0;
    while (k <= n && s[i + k] == s[j + k])
        ++k;
    if (s[i + k] < s[j + k])
        j += k + 1;
    else
        i += k + 1;
    j += (i == j);
}
```

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

## description

求有多少长度为  $k$  的，最小循环表示唯一（指分割位置唯一）且就是自身的字符串。

## solution

考虑若最小循环表示不唯一，则该串必然为整周期串。因此“最小循环表示唯一”的串必然是无整周期的。

用  $f(k)$  表示长度为  $k$  且最小循环表示唯一的串的数量，易得

$$f(k) = \sigma^k - \sum_{d|k} f(d) \quad \text{或者等价的} \quad f(k) = \sum_{d|k} \mu\left(\frac{k}{d}\right) \sigma^d$$

加上第二条限制，答案为  $f(k)/k$ 。

## description

求有多少长度为  $k$  的，最小循环表示唯一（指分割位置唯一）且就是自身的字符串。

## solution

考虑若最小循环表示不唯一，则该串必然为整周期串。因此“最小循环表示唯一”的串必然是无整周期的。

用  $f(k)$  表示长度为  $k$  且最小循环表示唯一的串的数量，易得

$$f(k) = \sigma^k - \sum_{d|k} f(d) \quad \text{或者等价的} \quad f(k) = \sum_{d|k} \mu\left(\frac{k}{d}\right) \sigma^d$$

加上第二条限制，答案为  $f(k)/k$ 。



## [ZJOI2017] 字符串

## description

维护字符串  $S \in \mathbb{Z}^n$ , 支持  $m$  次如下两种操作:

- 给出  $l, r, d$ , 对于  $i \in [l, r]$ ,  $S_i \leftarrow S_i + d$ ;
- 给出  $l, r$ , 询问串  $S_{[l, r]}$  的最小后缀。

## constraint

$$n \leq 2 \times 10^5, m \leq 3 \times 10^4, |S_i| \leq 10^8.$$



## [ZJOI2017] 字符串

## solution

$S$  的最小后缀是  $S_0 \not\equiv Sc$  的最小后缀是  $S_0c$ 。

我们称满足“存在  $T$  使  $ST$  的最小后缀是  $S_0T$ ”的后缀  $S_0$  构成集合  $\text{candidates}(S)$ 。有结论表示<sup>3</sup> $|\text{candidates}(S)| \leq \log_2 |S|$ 。

回到原问题。对于第二种操作，可以在线段树的每个节点  $[L, R]$  维护  $\text{candidates}(S_{[L,R]})$ ，一次询问  $[l, r]$  时便可通过线段树定位出大小不超过  $\log_2^2 n$  的  $\text{candidates}$  集合。

考虑在带修改下如何实现比较两个后缀大小。采用二分 Hash，此时需要动态维护 Hash 数组：观察到询问次数 ( $m \log^3 n$ ) 远多于修改 ( $m$ )，采用分块实现  $O(\sqrt{n})$  修改与  $O(1)$  查询。

算法的总时间复杂度是  $O(n\sqrt{n} + n \log^2 n + m\sqrt{n} + m \log^3 n)$ 。

<sup>3</sup>事实上， $\text{candidates}(S)$  中长度相邻的两个后缀的长度至少差一倍。◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

给定字符串  $S$ ，Manacher 算法可以对  $S$  的每个位置  $i$  求出以其为中心的最长奇回文串长度。

通过间隔地插入某一种不属于  $\Sigma$  的字符，可以实现求原串中的所有极长回文串。

记  $len_i$  表示以位置  $i$  为中心的最长奇回文串半径，考虑从左往右依次求  $len_i$ 。记录前缀中  $i + len_i$  的最大值  $mx$  及其对应的位置  $pos$ 。当需要计算  $len_j$  时，根据回文的对称性可以利用到之前已得到的信息，即

$$len_j \geq \min(len_{2pos-j}, mx - j)$$

这样每次暴力扩展都必然导致  $mx$  增大，从而时间复杂度为  $O(|S|)$ 。

给定字符串  $S$ ，Manacher 算法可以对  $S$  的每个位置  $i$  求出以其为中心的最长奇回文串长度。

通过间隔地插入某一种不属于  $\Sigma$  的字符，可以实现求原串中的所有极长回文串。

记  $len_i$  表示以位置  $i$  为中心的最长奇回文串半径，考虑从左往右依次求  $len_i$ 。记录前缀中  $i + len_i$  的最大值  $mx$  及其对应的位置  $pos$ 。当需要计算  $len_j$  时，根据回文的对称性可以利用到之前已得到的信息，即

$$len_j \geq \min(len_{2pos-j}, mx - j)$$

这样每次暴力扩展都必然导致  $mx$  增大，从而时间复杂度为  $O(|S|)$ 。

给定字符串  $S$ ，Manacher 算法可以对  $S$  的每个位置  $i$  求出以其为中心的最长奇回文串长度。

通过间隔地插入某一种不属于  $\Sigma$  的字符，可以实现求原串中的所有极长回文串。

记  $len_i$  表示以位置  $i$  为中心的最长奇回文串半径，考虑从左往右依次求  $len_i$ 。记录前缀中  $i + len_i$  的最大值  $mx$  及其对应的位置  $pos$ 。当需要计算  $len_j$  时，根据回文的对称性可以利用到之前已得到的信息，即

$$len_j \geq \min(len_{2pos-j}, mx - j)$$

这样每次暴力扩展都必然导致  $mx$  增大，从而时间复杂度为  $O(|S|)$ 。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

```
int mx = 0, pos = 0;
for (int i = 1; i <= n; ++i) {
    len[i] = mx > i ? min(len[2 * pos - i], mx - i) : 1;
    while (s[i - len[i]] == s[i + len[i]])
        ++len[i];
    if (i + len[i] > mx)
        mx = i + len[i], pos = i;
}
```

## description

对于字符串  $S$ ，定义运算  $f(S)$  为将  $S$  的前  $|S| - 1$  个字符倒序接在  $S$  后面形成的长为  $2|S| - 1$  的新字符串。

现给出字符串  $T$ ，询问有哪些串长度不超过  $|T|$  的串  $S$  经过若干次  $f$  运算后得到的串包含  $T$  作为前缀。

## constraint

$$1 \leq |T| \leq 5 \times 10^6.$$

## solution

合法的串  $S$  一定是  $T$  的前缀。

先用 Manacher 求出每个位置的回文半径，从后往前处理，一个位置合法当且仅当其回文半径右侧达到  $|T|$ ，或者左侧达到 1 且右侧位置合法。



## description

对于字符串  $S$ ，定义运算  $f(S)$  为将  $S$  的前  $|S| - 1$  个字符倒序接在  $S$  后面形成的长为  $2|S| - 1$  的新字符串。

现给出字符串  $T$ ，询问有哪些串长度不超过  $|T|$  的串  $S$  经过若干次  $f$  运算后得到的串包含  $T$  作为前缀。

## constraint

$$1 \leq |T| \leq 5 \times 10^6.$$

## solution

合法的串  $S$  一定是  $T$  的前缀。

先用 Manacher 求出每个位置的回文半径，从后往前处理，一个位置合法当且仅当其回文半径右侧达到  $|T|$ ，或者左侧达到 1 且右侧位置合法。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

# 最长双回文串

## description

求串  $S$  最长的可写成  $X + Y$  的子串长度，其中  $X, Y$  都是回文串。

## constraint

$$|S| \leq 10^6.$$

## solution

利用 Manacher 的结果，可以求出以每个位置为开头/结尾的最长回文串长度。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

# 最长双回文串

## description

求串  $S$  最长的可写成  $X + Y$  的子串长度，其中  $X, Y$  都是回文串。

## constraint

$$|S| \leq 10^6.$$

## solution

利用 Manacher 的结果，可以求出以每个位置为开头/结尾的最长回文串长度。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

ex-KMP 又称 Z-algorithm, 可以对串  $S$  求出所有  $\text{lcp}(1, i), i = 2, \dots, |S|$ , 即原串与所有后缀的最长公共前缀。

为什么先讲 ex-KMP 后讲 KMP? 是因为 ex-KMP 跟 KMP 没什么太大关系, 反倒是和 Manacher 很相似。

类似 Manacher, 记  $\text{len}_i$  表示  $\text{lcp}(1, i)$ , 同时记录  $i + \text{len}_i$  的最大值  $mx$  及其对应位置  $pos$ , 可以发现  $\text{len}_j \geq \min(\text{len}_{j-pos+1}, mx - j)$ , 其余部分与 Manacher 几乎完全相同。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

```
int mx = 1, pos = 1;
for (int i = 2; i <= n; ++i) {
    lcp[i] = mx > i ? min(lcp[i - pos + 1], mx - i) : 0;
    while (s[1 + lcp[i]] == s[i + lcp[i]])
        ++lcp[i];
    if (i + lcp[i] > mx)
        pos = i, mx = i + lcp[i];
}
```

## [JSOI2019] 节日庆典

## description

给出字符串  $S$ ，对  $S$  的每一个前缀求最小循环表示。

## constraint

$$1 \leq |S| \leq 3 \times 10^6.$$

## solution

对于  $p \in [1, |S|]$ ， $S_{[p, |S|]} + S_{[1, p-1]}$  是最小循环表示的必要条件是  $S_{[p, |S|]} \in \text{candidates}(S)$ 。

而我们知道  $|\text{candidates}(S)| \leq \log_2 |S|$ ，因此只需要按照长度维护  $\text{candidates}(S_{[1, i]})$ 。

问题还是在于如何实现比较。注意到  $\text{candidates}(S)$  中任意两个串都有后缀关系，可以简单地把比较两个循环表示的字典序规约到求  $\text{lcp}(1, *)$ 。Z-algorithm 即可。

## [JSOI2019] 节日庆典

## description

给出字符串  $S$ ，对  $S$  的每一个前缀求最小循环表示。

## constraint

$$1 \leq |S| \leq 3 \times 10^6.$$

## solution

对于  $p \in [1, |S|]$ ， $S_{[p, |S|]} + S_{[1, p-1]}$  是最小循环表示的必要条件是  $S_{[p, |S|]} \in \text{candidates}(S)$ 。

而我们知道  $|\text{candidates}(S)| \leq \log_2 |S|$ ，因此只需要按照长度维护  $\text{candidates}(S_{[1, i]})$ 。

问题还是在于如何实现比较。注意到  $\text{candidates}(S)$  中任意两个串都有后缀关系，可以简单地把比较两个循环表示的字典序规约到求  $\text{lcp}(1, *)$ 。Z-algorithm 即可。

把一个字符串映射到一个数。通过构造 Hash 函数以尽量减少冲突，理想情况是使映射趋于完全随机。

常见的构造函数形如：

$$\text{hash}(S) = \sum_{i=1}^{|S|} S_i \times \text{Base}^{|S|-i} \mod p$$

$p$  可以取一个大质数，也可以取  $2^{32}$  或  $2^{64}$ ，不过值得注意的是确定存在并且已经构造出了能使  $p = 2^{64}$  产生 Hash 冲突的字符串。



## [CTSC2014] 企鹅 QQ

## description

给  $n$  个长度相同且互不相同的串，询问有多少对串仅在一个位置上不同。

## constraint

$$1 \leq n \leq 30000, 1 \leq |S_i| \leq 200.$$

## solution

直接枚举哪一位不同，Hash 前后缀即可。

## [CTSC2014] 企鹅 QQ

## description

给  $n$  个长度相同且互不相同的串，询问有多少对串仅在一个位置上不同。

## constraint

$$1 \leq n \leq 30000, 1 \leq |S_i| \leq 200.$$

## solution

直接枚举哪一位不同，Hash 前后缀即可。

## [CQOI2014] 通配符匹配

## description

给出一个带通配符的字符串  $S$ ，通配符分两种，一种可以匹配恰好一个字符，另一种可以匹配任意个（包括 0 个）字符。再给出  $n$  个串  $T_i$ ，询问  $T_i$  能否与  $S$  匹配。

## constraint

$1 \leq n \leq 100, 1 \leq |S|, |T_i| \leq 10^5, 0 \leq \text{通配符个数} \leq 10$ .

## solution

设  $f_{i,j}$  表示前  $i$  个通配符匹配  $T$  的前  $j$  个字符是否可行，用 Hash 判断字符串相等即可。

转移“匹配任意个字符”的通配符时需要做一次前缀和。

## [CQOI2014] 通配符匹配

## description

给出一个带通配符的字符串  $S$ ，通配符分两种，一种可以匹配恰好一个字符，另一种可以匹配任意个（包括 0 个）字符。再给出  $n$  个串  $T_i$ ，询问  $T_i$  能否与  $S$  匹配。

## constraint

$1 \leq n \leq 100, 1 \leq |S|, |T_i| \leq 10^5, 0 \leq \text{通配符个数} \leq 10$ .

## solution

设  $f_{i,j}$  表示前  $i$  个通配符匹配  $T$  的前  $j$  个字符是否可行，用 Hash 判断字符串相等即可。

转移“匹配任意个字符”的通配符时需要做一次前缀和。

KMP 算法解决的是字符串匹配中单串匹配单串的问题。具体的，这个问题可以被描述为：给出一个长度为  $n$  的文本串  $S$  和一个长度为  $m$  的模式串  $T$ ，问  $T$  在  $S$  中出现了多少次，分别在哪些位置出现了。

朴素做法复杂度  $O(nm)$ ：枚举  $S$  的开头位置， $O(m)$  遍历比较两串。

在一次比较失败后，我们放弃了已比较部分的全部信息，直接选取  $S$  的下一个位置继续匹配。这在一定程度上造成了浪费。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

举个例子，假设文本串  $S = abbaabbbabaa$ ，模式串  $T = abbaaba$ ，从  $S$  的第一位开始匹配：

*abbaabbbabaa*  
*abbaaba*

匹配了 6 位，在第 7 位上失配了。按照朴素做法，我们会直接放弃“已匹配 6 位”的信息，转而从  $S$  的第二位开始匹配。这其中浪费掉了很多已知的信息，造成了复杂度过高，由此，KMP 算法应运而生。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

举个例子，假设文本串  $S = abbaabbbabaa$ ，模式串  $T = abbaaba$ ，从  $S$  的第一位开始匹配：

*abbaabbbabaa*

*abbaaba*

匹配了 6 位，在第 7 位上失配了。按照朴素做法，我们会直接放弃“已匹配 6 位”的信息，转而从  $S$  的第二位开始匹配。这其中浪费掉了很多已知的信息，造成了复杂度过高，由此，KMP 算法应运而生。

举个例子，假设文本串  $S = abbaabbbabaa$ ，模式串  $T = abbaaba$ ，从  $S$  的第一位开始匹配：

*abbaabbbabaa*

*abbaaba*

匹配了 6 位，在第 7 位上失配了。按照朴素做法，我们会直接放弃“已匹配 6 位”的信息，转而从  $S$  的第二位开始匹配。这其中浪费掉了很多已知的信息，造成了复杂度过高，由此，KMP 算法应运而生。



KMP 的本质是对每一个  $i$  求出最大的  $j < i$  满足  $S_{i-j+1\dots i} = pre_j$  (换言之即最长的相同前后缀), 记之为  $next_i$ 。

举个例子, 对于字符串 *aababaabaccc* 来说, 它的前缀 *aababaaba* 的相同前后缀有 *aaba*、*a* 以及空串, 所以它对应的 *next* 就等于 4。

# $nxt$ 数组的构造

在解决前面说过的字符串匹配问题时，需要先对  $T$  串构造  $nxt$  数组。 $nxt$  数组的构造方式是增量构造，即在已经得到  $nxt_1 \dots nxt_{i-1}$  的情况下，求  $nxt_i$ 。

如果  $nxt_{i-1}$  后一个字符与第  $i$  个字符相同，即  $T_{nxt_{i-1}+1} = T_i$ ，那么  $nxt_i = nxt_{i-1} + 1$ 。

否则我们需要找到对于  $i-1$  而言第二长的相同前后缀，这个实际上是  $nxt_{nxt_{i-1}}$ ，同理第三长的是  $nxt_{nxt_{nxt_{i-1}}}$ ，以此类推。

我们按照长度顺序依次判断  $i-1$  的某个相同前后缀的下一位是否与  $T_i$  相同，注意可能并不存在这样的前后缀，那么  $nxt_i$  将会等于 0。代码差不多长这样：

```
for (int i = 2, j = 0; i <= m; ++i) {
    while (j && T[j+1] != T[i])
        j = nxt[j];
    if (T[j+1] == T[i])
        ++j;
    nxt[i] = j;
}
```

# *nxt* 数组的构造

在解决前面说过的字符串匹配问题时，需要先对  $T$  串构造  $nxt$  数组。 $nxt$  数组的构造方式是增量构造，即在已经得到  $nxt_1 \dots nxt_{i-1}$  的情况下，求  $nxt_i$ 。

如果  $nxt_{i-1}$  后一个字符与第  $i$  个字符相同，即  $T_{nxt_{i-1}+1} = T_i$ ，那么  $nxt_i = nxt_{i-1} + 1$ 。

否则我们需要找到对于  $i-1$  而言第二长的相同前后缀，这个实际上是  $nxt_{nxt_{i-1}}$ ，同理第三长的是  $nxt_{nxt_{nxt_{i-1}}}$ ，以此类推。

我们按照长度顺序依次判断  $i-1$  的某个相同前后缀的下一位是否与  $T_i$  相同，注意可能并不存在这样的前后缀，那么  $nxt_i$  将会等于 0。代码差不多长这样：

```
for (int i = 2, j = 0; i <= m; ++i) {
    while (j && T[j+1] != T[i])
        j = nxt[j];
    if (T[j+1] == T[i])
        ++j;
    nxt[i] = j;
}
```

# $nxt$ 数组的构造

在解决前面说过的字符串匹配问题时，需要先对  $T$  串构造  $nxt$  数组。 $nxt$  数组的构造方式是增量构造，即在已经得到  $nxt_1 \dots nxt_{i-1}$  的情况下，求  $nxt_i$ 。

如果  $nxt_{i-1}$  后一个字符与第  $i$  个字符相同，即  $T_{nxt_{i-1}+1} = T_i$ ，那么  $nxt_i = nxt_{i-1} + 1$ 。

否则我们需要找到对于  $i-1$  而言第二长的相同前后缀，这个实际上是  $nxt_{nxt_{i-1}}$ ，同理第三长的是  $nxt_{nxt_{nxt_{i-1}}}$ ，以此类推。

我们按照长度顺序依次判断  $i-1$  的某个相同前后缀的下一位是否与  $T_i$  相同，注意可能并不存在这样的前后缀，那么  $nxt_i$  将会等于 0。代码差不多长这样：

```
for (int i = 2, j = 0; i <= m; ++i) {
    while (j && T[j+1] != T[i])
        j = nxt[j];
    if (T[j+1] == T[i])
        ++j;
    nxt[i] = j;
}
```

在解决前面说过的字符串匹配问题时，需要先对  $T$  串构造  $nxt$  数组。 $nxt$  数组的构造方式是增量构造，即在已经得到  $nxt_1 \dots nxt_{i-1}$  的情况下，求  $nxt_i$ 。

如果  $nxt_{i-1}$  后一个字符与第  $i$  个字符相同，即  $T_{nxt_{i-1}+1} = T_i$ ，那么  $nxt_i = nxt_{i-1} + 1$ 。

否则我们需要找到对于  $i-1$  而言第二长的相同前后缀，这个实际上是  $nxt_{nxt_{i-1}}$ ，同理第三长的是  $nxt_{nxt_{nxt_{i-1}}}$ ，以此类推。

我们按照长度顺序依次判断  $i-1$  的某个相同前后缀的下一位是否与  $T_i$  相同，注意可能并不存在这样的前后缀，那么  $nxt_i$  将会等于 0。代码差不多长这样：

```
for (int i = 2, j = 0; i <= m; ++i) {
    while (j && T[j+1] != T[i])
        j = nxt[j];
    if (T[j+1] == T[i])
        ++j;
    nxt[i] = j;
}
```

在解决前面说过的字符串匹配问题时，需要先对  $T$  串构造  $nxt$  数组。 $nxt$  数组的构造方式是增量构造，即在已经得到  $nxt_1 \dots nxt_{i-1}$  的情况下，求  $nxt_i$ 。

如果  $nxt_{i-1}$  后一个字符与第  $i$  个字符相同，即  $T_{nxt_{i-1}+1} = T_i$ ，那么  $nxt_i = nxt_{i-1} + 1$ 。

否则我们需要找到对于  $i-1$  而言第二长的相同前后缀，这个实际上是  $nxt_{nxt_{i-1}}$ ，同理第三长的是  $nxt_{nxt_{nxt_{i-1}}}$ ，以此类推。

我们按照长度顺序依次判断  $i-1$  的某个相同前后缀的下一位是否与  $T_i$  相同，注意可能并不存在这样的前后缀，那么  $nxt_i$  将会等于 0。代码差不多长这样：

```
for (int i = 2, j = 0; i <= m; ++i) {
    while (j && T[j+1] != T[i])
        j = nxt[j];
    if (T[j+1] == T[i])
        ++j;
    nxt[i] = j;
}
```

# 解决原问题

在使用 *nxt* 数组解决匹配问题前，先考虑一下我们可以怎样优化匹配的复杂度？还是用前面的例子：

*abbaabbbabaa*

*abbaaba*

对于已经匹配的 6 位，我们考虑：*S* 的前 6 个起始位置中，哪些还可能模式串的匹配？可以断言只有第 5 位还有可能，这是因为串 *abbaab* 的相同前后缀只有 *ab* 和空串。

代码和构造 *nxt* 数组差不多：

```
for (int i = 1, j = 0; i <= n; ++i) {
    while (j && T[j+1] != S[i])
        j = nxt[j];
    if (T[j+1] == S[i])
        ++j;
    if (j == m) {
        //match success
    }
}
```

## 解决原问题

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

在使用 *next* 数组解决匹配问题前，先考虑一下我们可以怎样优化匹配的复杂度？还是用前面的例子：

*abbaabbbabaa*

*abbaaba*

对于已经匹配的 6 位，我们考虑：*S* 的前 6 个起始位置中，哪些还可能模式串的匹配？可以断言只有第 5 位还有可能，这是因为串 *abbaab* 的相同前后缀只有 *ab* 和空串。

代码和构造 *next* 数组差不多：

```
for (int i = 1, j = 0; i <= n; ++i) {
    while (j && T[j+1] != S[i])
        j = next[j];
    if (T[j+1] == S[i])
        ++j;
    if (j == m) {
        //match success
    }
}
```



## 解决原问题

在使用 *nxt* 数组解决匹配问题前，先考虑一下我们可以怎样优化匹配的复杂度？还是用前面的例子：

*abbaabbbabaa*

*abbaaba*

对于已经匹配的 6 位，我们考虑：*S* 的前 6 个起始位置中，哪些还可能模式串的匹配？可以断言只有第 5 位还有可能，这是因为串 *abbaab* 的相同前后缀只有 *ab* 和空串。

代码和构造 *nxt* 数组差不多：

```
for (int i = 1, j = 0; i <= n; ++i) {
    while (j && T[j+1] != S[i])
        j = nxt[j];
    if (T[j+1] == S[i])
        ++j;
    if (j == m) {
        //match success
    }
}
```

# KMP 的时间复杂度

前面说了朴素做法的时间复杂度是  $O(nm)$ ，那么 KMP 算法又如何呢？

```
for (int i = 2, j = 0; i <= m; ++i) {
    while (j && T[j+1] != T[i])
        j = nxt[j];
    if (T[j+1] == T[i])
        ++j;
    nxt[i] = j;
}
```

考虑构建  $nxt$  数组的部分，注意观察  $j$  指针的变化，可以发现至多只有  $m$  次  $++j$  操作，而每次  $j = nxt[j]$  都一定会使  $j$  变小，因此  $for$  循环中的  $while$  语句在整个算法流程中只会被执行不超过  $m$  次。

因此构造  $nxt$  数组的时间复杂度为  $O(m)$ ，类似地可以分析出匹配时的时间复杂度为  $O(n)$ ，因此算法的总时间复杂度为  $O(n + m)$ 。

# KMP 的时间复杂度

前面说了朴素做法的时间复杂度是  $O(nm)$ ，那么 KMP 算法又如何呢？

```
for (int i = 2, j = 0; i <= m; ++i) {
    while (j && T[j+1] != T[i])
        j = nxt[j];
    if (T[j+1] == T[i])
        ++j;
    nxt[i] = j;
}
```

考虑构建  $nxt$  数组的部分，注意观察  $j$  指针的变化，可以发现至多只有  $m$  次  $++j$  操作，而每次  $j = nxt[j]$  都一定会使  $j$  变小，因此 for 循环中的 while 语句在整个算法流程中只会被执行不超过  $m$  次。

因此构造  $nxt$  数组的时间复杂度为  $O(m)$ ，类似地可以分析出匹配时的时间复杂度为  $O(n)$ ，因此算法的总时间复杂度为  $O(n + m)$ 。

# KMP 的时间复杂度

前面说了朴素做法的时间复杂度是  $O(nm)$ ，那么 KMP 算法又如何呢？

```
for (int i = 2, j = 0; i <= m; ++i) {
    while (j && T[j+1] != T[i])
        j = nxt[j];
    if (T[j+1] == T[i])
        ++j;
    nxt[i] = j;
}
```

考虑构建  $nxt$  数组的部分，注意观察  $j$  指针的变化，可以发现至多只有  $m$  次  $++j$  操作，而每次  $j = nxt[j]$  都一定会使  $j$  变小，因此  $for$  循环中的  $while$  语句在整个算法流程中只会被执行不超过  $m$  次。

因此构造  $nxt$  数组的时间复杂度为  $O(m)$ ，类似地可以分析出匹配时的时间复杂度为  $O(n)$ ，因此算法的总时间复杂度为  $O(n + m)$ 。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

在 KMP 求  $nxt$  的基础上, 额外求出  $trans_{i,j}$  表示从  $i$  位置开始往后匹配一个  $j$  字符后会转移到什么状态, 可以实现真正严格  $O(1)$  而非均摊  $O(1)$  的转移。但需要注意预处理  $trans$  的复杂度是  $O(m\sigma)$  的。

## description

给出  $n$  阶排列  $p$  和  $m$  阶排列  $\{h_i\}_{i=1}^m$ , 定义  $p$  在  $h$  的  $k$  位置出现 ( $1 \leq k \leq m - n + 1$ ), 如果  $\forall i, j \in [1, m], p_i < p_j \Leftrightarrow h_{k+i-1} < h_{k+j-1}$ 。求  $p$  在  $h$  中的所有出现位置。

## constraint

$$1 \leq m \leq n \leq 10^6.$$

## solution

寻求一种用于描述“相对大小”的记号。事实上只需要建立  $n - 1$  对正确的偏序关系就可以确定  $p$  的一次出现了。考虑对  $p$  的每一位求“这一位前面第一个大于和小于这个位置上的数的出现位置 (维护下标差)”, 那么在  $h$  中的出现就只需要按照这些偏序关系进行匹配。先使用链表对  $p$  求出上述内容, 然后 KMP 即可。

<sup>4</sup> 此题被选入北京大学《数据结构与算法 A (实验班)》(2021 秋) 期末考试。

[CEOI2011] Matching<sup>4</sup>

## description

给出  $n$  阶排列  $p$  和  $m$  阶排列  $\{h_i\}_{i=1}^m$ , 定义  $p$  在  $h$  的  $k$  位置出现 ( $1 \leq k \leq m - n + 1$ ), 如果  $\forall i, j \in [1, m], p_i < p_j \Leftrightarrow h_{k+i-1} < h_{k+j-1}$ 。求  $p$  在  $h$  中的所有出现位置。

## constraint

$$1 \leq m \leq n \leq 10^6.$$

## solution

寻求一种用于描述“相对大小”的记号。事实上只需要建立  $n - 1$  对正确的偏序关系就可以确定  $p$  的一次出现了。

考虑对  $p$  的每一位求“这一位前面第一个大于和小于这个位置上的数的出现位置 (维护下标差)”, 那么在  $h$  中的出现就只需要按照这些偏序关系进行匹配。

先使用链表对  $p$  求出上述内容, 然后 KMP 即可。

<sup>4</sup> 此题被选入北京大学《数据结构与算法 A (实验班)》(2021 秋) 期末考试。

## description

给出  $n$  阶排列  $p$  和  $m$  阶排列  $\{h_i\}_{i=1}^m$ , 定义  $p$  在  $h$  的  $k$  位置出现 ( $1 \leq k \leq m - n + 1$ ), 如果  $\forall i, j \in [1, m], p_i < p_j \Leftrightarrow h_{k+i-1} < h_{k+j-1}$ 。求  $p$  在  $h$  中的所有出现位置。

## constraint

$$1 \leq m \leq n \leq 10^6.$$

## solution

寻求一种用于描述“相对大小”的记号。事实上只需要建立  $n - 1$  对正确的偏序关系就可以确定  $p$  的一次出现了。

考虑对  $p$  的每一位求“这一位前面第一个大于和小于这个位置上的数的出现位置 (维护下标差)”, 那么在  $h$  中的出现就只需要按照这些偏序关系进行匹配。

先使用链表对  $p$  求出上述内容, 然后 KMP 即可。

<sup>4</sup> 此题被选入北京大学《数据结构与算法 A (实验班)》(2021 秋) 期末考试。



[CEOI2011] Matching<sup>4</sup>

## description

给出  $n$  阶排列  $p$  和  $m$  阶排列  $\{h_i\}_{i=1}^m$ , 定义  $p$  在  $h$  的  $k$  位置出现 ( $1 \leq k \leq m - n + 1$ ), 如果  $\forall i, j \in [1, m], p_i < p_j \Leftrightarrow h_{k+i-1} < h_{k+j-1}$ 。求  $p$  在  $h$  中的所有出现位置。

## constraint

$$1 \leq m \leq n \leq 10^6.$$

## solution

寻求一种用于描述“相对大小”的记号。事实上只需要建立  $n - 1$  对正确的偏序关系就可以确定  $p$  的一次出现了。

考虑对  $p$  的每一位求“这一位前面第一个大于和小于这个位置上的数的出现位置 (维护下标差)”, 那么在  $h$  中的出现就只需要按照这些偏序关系进行匹配。

先使用链表对  $p$  求出上述内容, 然后 KMP 即可。

<sup>4</sup> 此题被选入北京大学《数据结构与算法 A (实验班)》(2021 秋) 期末考试。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

## border 与周期理论

称字符串  $S$  有长度为  $p$  ( $1 \leq p < |S|$ ) 的 border, 如果

$$S_{[1,p]} = S_{[|S|-p+1,|S|]}.$$

称  $r$  ( $1 \leq r < |S|$ ) 是字符串  $S$  的周期, 如果  $S_i = S_{i+r}$  对所有  $i \in [1, |S| - r]$  均成立。

## 定理

$r$  是  $S$  的周期当且仅当  $S$  有长度为  $|S| - r$  的 border。

## 引理

$S$  所有不超过  $\frac{|S|}{2}$  的周期都是其最短周期的倍数<sup>5</sup>。或者等价的,  $S$  所有长度不小于  $\frac{|S|}{2}$  的 border 长度构成等差数列。

## 定理

$S$  的所有 border 长度 (周期) 构成  $O(\log n)$  个值域不交的等差数列。

<sup>5</sup>Weak Periodicity Lemma: 若  $p, q$  是  $S$  的周期, 且  $p + q \leq |S|$ , 则  $\text{lcm}(p, q)$  是  $S$  的周期。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

## border 与周期理论

称字符串  $S$  有长度为  $p$  ( $1 \leq p < |S|$ ) 的 border, 如果

$$S_{[1,p]} = S_{[|S|-p+1,|S|]}.$$

称  $r$  ( $1 \leq r < |S|$ ) 是字符串  $S$  的周期, 如果  $S_i = S_{i+r}$  对所有  $i \in [1, |S| - r]$  均成立。

## 定理


$r$  是  $S$  的周期当且仅当  $S$  有长度为  $|S| - r$  的 border。

## 引理

$S$  所有不超过  $\frac{|S|}{2}$  的周期都是其最短周期的倍数<sup>5</sup>。或者等价的,  $S$  所有长度不小于  $\frac{|S|}{2}$  的 border 长度构成等差数列。

## 定理

$S$  的所有 border 长度 (周期) 构成  $O(\log n)$  个值域不交的等差数列。

<sup>5</sup>Weak Periodicity Lemma: 若  $p, q$  是  $S$  的周期, 且  $p + q \leq |S|$ , 则  $\gcd(p, q)$  是  $S$  的周期。 

字符串

zsy

# 无限猴子问题: [CTSC2006] 歌唱王国

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

## description

均匀随机地往初始为  $\epsilon$  的字符串末尾添加  $\Sigma$  中的字符，直到给定模式串  $S$  作为子串出现时停止。求停止时字符串的期望长度。

## constraint

$$|S| \leq 10^6.$$

## solution

记  $f(i)$  表示停止时长度为  $i$  的概率， $g(i)$  表示长度为  $i$  时尚未停止的概率，即  $g(i) = \sum_{j>i} f(j)$ 。

$$g(i) \left(\frac{1}{\sigma}\right)^n = \sum_{j=1}^n a_j \left(\frac{1}{\sigma}\right)^{n-j} f_{i+j}$$

其中  $a_j \in \{0, 1\}$  表示  $S$  是否有长度为  $j$  的 border。  
容易得到  $\mathbb{E}[\text{length}] = \sum_{i \geq 0} i f(i) = \sum_{i \geq 1} g(i) = \sum_{j=1}^n a_j \sigma^j$ 。

## 无限猴子问题: [CTSC2006] 歌唱王国

## description

均匀随机地往初始为  $\epsilon$  的字符串末尾添加  $\Sigma$  中的字符, 直到给定模式串  $S$  作为子串出现时停止。求停止时字符串的期望长度。

## constraint

$$|S| \leq 10^6.$$

## solution

记  $f(i)$  表示停止时长度为  $i$  的概率,  $g(i)$  表示长度为  $i$  时尚未停止的概率, 即  $g(i) = \sum_{j>i} f(j)$ 。

$$g(i) \left(\frac{1}{\sigma}\right)^n = \sum_{j=1}^n a_j \left(\frac{1}{\sigma}\right)^{n-j} f_{i+j}$$

其中  $a_j \in \{0, 1\}$  表示  $S$  是否有长度为  $j$  的 border。  
容易得到  $\mathbb{E}[\text{length}] = \sum_{i \geq 0} i f(i) = \sum_{i \geq 1} g(i) = \sum_{j=1}^n a_j \sigma^j$ 。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

# [JSOI2016] 无界单词

## description

没有非平凡 border 的小写字母字符串被称为无界单词。

给定  $n$ , 求  $\{a, b\}^n$  中 (1) 有多少无界单词; (2) 字典序第  $k$  小的无界单词是什么。

## constraint

$n \leq 64$ . 1000 组数据。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

## [JSOI2016] 无界单词

## solution

一个字符串的最小 border 长度 (如果有的话) 不超过串长的一半。  
考虑第一问。设  $f(n)$  表示长度为  $n$  的无界单词数量, 有转移

$$f(n) = 2^n - \sum_{m=1}^{n/2} 2^{n-2m} f(m)$$

考虑第二问。求“第  $k$  小”问题的经典套路是从高位到低位确定, 并把问题规约到计数。这里要实现“固定前缀的无界单词计数”。  
假设固定的前缀是  $S$ ,  $|S| = l$ 。效仿前面的计数方法, 可以得到

$$f(n) = \begin{cases} [S_{[1,n]} \text{ is borderless}], n \leq l \\ 2^{n-l} - \sum_{m=1}^{n/2} f(m) \begin{cases} 2^{n-2m}, m \geq l \\ [S_{[1,m+l-n]} = S_{[n-m+1,l]}], m < l, m+l > n \\ 2^{n-m-l}, m < l, m+l \leq n \end{cases} \end{cases}$$

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

## [JSOI2016] 无界单词

## solution

一个字符串的最小 border 长度 (如果有的话) 不超过串长的一半。  
考虑第一问。设  $f(n)$  表示长度为  $n$  的无界单词数量, 有转移

$$f(n) = 2^n - \sum_{m=1}^{n/2} 2^{n-2m} f(m)$$

考虑第二问。求“第  $k$  小”问题的经典套路是从高位到低位确定, 并把问题规约到计数。这里要实现“固定前缀的无界单词计数”。  
假设固定的前缀是  $S$ ,  $|S| = l$ 。效仿前面的计数方法, 可以得到

$$f(n) = \begin{cases} [S_{[1,n]} \text{ is borderless}], n \leq l \\ 2^{n-l} - \sum_{m=1}^{n/2} f(m) \begin{cases} 2^{n-2m}, m \geq l \\ [S_{[1,m+l-n]} = S_{[n-m+1,l]}], m < l, m+l > n \\ 2^{n-m-l}, m < l, m+l \leq n \end{cases} \end{cases}$$



## solution

一个字符串的最小 border 长度 (如果有的话) 不超过串长的一半。  
考虑第一问。设  $f(n)$  表示长度为  $n$  的无界单词数量, 有转移

$$f(n) = 2^n - \sum_{m=1}^{n/2} 2^{n-2m} f(m)$$

考虑第二问。求“第  $k$  小”问题的经典套路是从高位到低位确定, 并把问题规约到计数。这里要实现“固定前缀的无界单词计数”。  
假设固定的前缀是  $S$ ,  $|S| = l$ 。效仿前面的计数方法, 可以得到

$$f(n) = \begin{cases} [S_{[1,n]} \text{ is borderless}], n \leq l \\ 2^{n-l} - \sum_{m=1}^{n/2} f(m) \begin{cases} 2^{n-2m}, m \geq l \\ [S_{[1,m+l-n]} = S_{[n-m+1,l]}], m < l, m+l > n \\ 2^{n-m-l}, m < l, m+l \leq n \end{cases} \end{cases}$$

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

# Gym100958I. Substring Pairs

## description

求有多少对字符串  $(S, T)$  满足

- $|S| = n$ ,
- $|T| = m$ ,
- $T$  是  $S$  的子串。

由于答案可能很大，你只需要输出对  $10^9 + 7$  取模的结果。

## constraint

$1 \leq m \leq 50, m \leq n \leq 200$ .

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

## Gym100958I. Substring Pairs

## solution

先考虑对给定的  $T$  怎么求满足条件的  $S$  的数量。用  $f(n)$  表示长度为  $n$  且不包含  $T$  作为子串的字符串数量,  $g(n)$  表示长度为  $n$  且只在结尾包含  $T$  的字符串数量, 有

$$\sigma f(n-1) = f(n) + g(n)$$

$$f(n-m) = \sum_{T \text{ has a border of length } i} g(n-m+i)$$

因此可以  $O(n^2)$  求解。发现答案只和  $T$  的 border 集合有关, 可以爆搜出所有可能的 border 集合<sup>6</sup>, 容斥即可求出每一种 border 集合对应着多少个  $T$ 。

<sup>6</sup>可以参考【知乎】长度为  $n$  的字符串的可行 border 集合种类数是多少      

## Gym100958I. Substring Pairs

## solution

先考虑对给定的  $T$  怎么求满足条件的  $S$  的数量。用  $f(n)$  表示长度为  $n$  且不包含  $T$  作为子串的字符串数量,  $g(n)$  表示长度为  $n$  且只在结尾包含  $T$  的字符串数量, 有

$$\sigma f(n-1) = f(n) + g(n)$$

$$f(n-m) = \sum_{T \text{ has a border of length } i} g(n-m+i)$$

因此可以  $O(n^2)$  求解。发现答案只和  $T$  的 border 集合有关, 可以爆搜出所有可能的 border 集合<sup>6</sup>, 容斥即可求出每一种 border 集合对应着多少个  $T$ 。

<sup>6</sup>可以参考【知乎】长度为  $n$  的字符串的可行 border 集合种类数是多少?         

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

# [HNOI2008] GT 考试

## description

给定数字串  $S$ ，求有多少长度为  $n$  的数字串  $T$  不包含  $S$  作为子串。

## constraint

$$1 \leq |S| \leq 100, 1 \leq n \leq 10^9.$$

## solution

对  $S$  建出 KMP 自动机，把匹配到串  $S$  的每个位置视作一个状态，转移是在串的末尾加一个字符，相当于是在 KMP 自动机上走一条出边，矩阵快速幂转移即可。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

# [HNOI2008] GT 考试

## description

给定数字串  $S$ ，求有多少长度为  $n$  的数字串  $T$  不包含  $S$  作为子串。

## constraint

$$1 \leq |S| \leq 100, 1 \leq n \leq 10^9.$$

## solution

对  $S$  建出 KMP 自动机，把匹配到串  $S$  的每个位置视作一个状态，转移是在串的末尾加一个字符，相当于是在 KMP 自动机上走一条出边，矩阵快速幂转移即可。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

## [HNOI2008] GT 考试 · 改

## description

给定数字串  $S$ ，求有多少长度为  $n$  的数字串  $T$  不包含  $S$  作为子串。

## constraint

$$1 \leq |S| \leq n \leq 10^5.$$

## solution

传统做法是动态规划， $g(n)$  表示长度为  $n$  且只在结尾包含  $S$  的字符串数量，

$$g(n) = \sigma^{n-|S|} - \sum_{m=n-|S|+1}^{n-1} a_{|S|+m-n} g(m) - \sum_{m=|S|}^{n-|S|} \sigma^{n-m-|S|} g(m)$$

利用多项式技术优化至  $O(n \log n)$ 。但考虑利用“border 形成  $O(\log n)$  段等差数列”的性质，只需要按照  $O(\log n)$  个等差维护  $O(\log n)$  个前缀和数组，就能轻松做到同样的复杂度。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

## [HNOI2008] GT 考试 · 改

## description

给定数字串  $S$ ，求有多少长度为  $n$  的数字串  $T$  不包含  $S$  作为子串。

## constraint

$$1 \leq |S| \leq n \leq 10^5.$$

## solution

传统做法是动态规划， $g(n)$  表示长度为  $n$  且只在结尾包含  $S$  的字符串数量，

$$g(n) = \sigma^{n-|S|} - \sum_{m=n-|S|+1}^{n-1} a_{|S|+m-n} g(m) - \sum_{m=|S|}^{n-|S|} \sigma^{n-m-|S|} g(m)$$

利用多项式技术优化至  $O(n \log n)$ 。但考虑利用 “border 形成  $O(\log n)$  段等差数列” 的性质，只需要按照  $O(\log n)$  个等差维护  $O(\log n)$  个前缀和数组，就能轻松做到同样的复杂度。



## [HNOI2008] GT 考试 · 改

## description

给定数字串  $S$ ，求有多少长度为  $n$  的数字串  $T$  不包含  $S$  作为子串。

## constraint

$$1 \leq |S| \leq n \leq 10^5.$$

## solution

传统做法是动态规划， $g(n)$  表示长度为  $n$  且只在结尾包含  $S$  的字符串数量，

$$g(n) = \sigma^{n-|S|} - \sum_{m=n-|S|+1}^{n-1} a_{|S|+m-n} g(m) - \sum_{m=|S|}^{n-|S|} \sigma^{n-m-|S|} g(m)$$

利用多项式技术优化至  $O(n \log n)$ 。但考虑利用 “border 形成  $O(\log n)$  段等差数列” 的性质，只需要按照  $O(\log n)$  个等差维护  $O(\log n)$  个前缀和数组，就能轻松做到同样的复杂度。

又称字典树，结构是每条边上有一条字符，每个节点所代表的字符串就是从根节点出发到该节点路径上所有字符顺序连接形成的串。

两个串的最长公共前缀长度对应两点在 Trie 树上 LCA 的深度。

01 Trie 是指  $\Sigma = \{0, 1\}$  的 Trie，可以用来解决“给定  $A, x$ ，求  $\max_{y \in A} x \oplus y$ ”的问题。

又称字典树，结构是每条边上有一条字符，每个节点所代表的字符串就是从根节点出发到该节点路径上所有字符顺序连接形成的串。

两个串的最长公共前缀长度对应两点在 Trie 树上 LCA 的深度。

01 Trie 是指  $\Sigma = \{0, 1\}$  的 Trie，可以用来解决“给定  $A, x$ ，求  $\max_{y \in A} x \oplus y$ ”的问题。

又称字典树，结构是每条边上有一条字符，每个节点所代表的字符串就是从根节点出发到该节点路径上所有字符顺序连接形成的串。

两个串的最长公共前缀长度对应两点在 Trie 树上 LCA 的深度。

01 Trie 是指  $\Sigma = \{0, 1\}$  的 Trie，可以用来解决“给定  $A, x$ ，求  $\max_{y \in A} x \oplus y$ ”的问题。

## [FJOI2015] 火星商店问题

## description

有  $n$  个初始均为空的集合和  $m$  次操作，每次操作为向某个集合内加入一个数  $x$ ，或者查询最近的  $d$  次向编号在  $[l, r]$  内的集合加入的元素中，与  $x$  异或和的最大值。

## constraint

$$1 \leq n, m \leq 10^5, 0 \leq x < 2^{30}.$$

## solution

以  $n$  为下标建一棵线段树，每个节点上开一棵 01 Trie，为了处理加入时间的限制，可以对 Trie 的每个节点记录这个节点的最新更新时间，查询时根据这个更新时间来判断即可。

## [FJOI2015] 火星商店问题

## description

有  $n$  个初始均为空的集合和  $m$  次操作，每次操作为向某个集合内加入一个数  $x$ ，或者查询最近的  $d$  次向编号在  $[l, r]$  内的集合加入的元素中，与  $x$  异或和的最大值。

## constraint

$$1 \leq n, m \leq 10^5, 0 \leq x < 2^{30}.$$

## solution

以  $n$  为下标建一棵线段树，每个节点上开一棵 01 Trie，为了处理加入时间的限制，可以对 Trie 的每个节点记录这个节点的最新更新时间，查询时根据这个更新时间来判断即可。

## Aho-Corasick automaton

本质是 Trie 上实现 KMP 自动机。

按照深度由浅到深做。对每个节点求出  $fail_i$  表示这个点对应的串在 Trie 树中存在的最长严格后缀，构造方式也与 KMP 类似，通过不断跳  $fail$  以找到一条与当前字符相同的出边。

```
for (int i = 0; i < 26; ++i)
    if (tr[0][i])
        Q.push(tr[0][i]);
while (!Q.empty()){
    int u = Q.front();
    Q.pop();
    for (int i = 0; i < 26; ++i)
        if (tr[u][i]) {
            fail[tr[u][i]] = tr[fail[u]][i];
            Q.push(tr[u][i]);
        }
    else
        tr[u][i] = tr[fail[u]][i];
}
```

## Aho-Corasick automaton

本质是 Trie 上实现 KMP 自动机。

按照深度由浅到深做。对每个节点求出  $fail_i$  表示这个点对应的串在 Trie 树中存在的最长严格后缀，构造方式也与 KMP 类似，通过不断跳 fail 以找到一条与当前字符相同的出边。

```
for (int i = 0; i < 26; ++i)
    if (tr[0][i])
        Q.push(tr[0][i]);
while (!Q.empty()){
    int u = Q.front();
    Q.pop();
    for (int i = 0; i < 26; ++i)
        if (tr[u][i]) {
            fail[tr[u][i]] = tr[fail[u]][i];
            Q.push(tr[u][i]);
        }
    else
        tr[u][i] = tr[fail[u]][i];
}
```



## Aho-Corasick automaton

本质是 Trie 上实现 KMP 自动机。

按照深度由浅到深做。对每个节点求出  $fail_i$  表示这个点对应的串在 Trie 树中存在的最长严格后缀，构造方式也与 KMP 类似，通过不断跳 fail 以找到一条与当前字符相同的出边。

```
for (int i = 0; i < 26; ++i)
    if (tr[0][i])
        Q.push(tr[0][i]);
while (!Q.empty()){
    int u = Q.front();
    Q.pop();
    for (int i = 0; i < 26; ++i)
        if (tr[u][i]) {
            fail[tr[u][i]] = tr[fail[u]][i];
            Q.push(tr[u][i]);
        }
    else
        tr[u][i] = tr[fail[u]][i];
}
```

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

# Aho-Corasick automaton

一般的写法是直接建出自动机，在构建时只需要从其  $\text{fail}_i$  转移而来即可。

同时，根据  $i \rightarrow \text{fail}_i$  可以建出一个树形结构，这棵树被称为 fail 树，它具有有一些优美的性质，具体来说是在 fail 树上存在祖孙关系的两点对应的字符串具有后缀关系。

# Aho-Corasick automaton

一般的写法是直接建出自动机，在构建时只需要从其  $\text{fail}_i$  转移而来即可。

同时，根据  $i \rightarrow \text{fail}_i$  可以建出一个树形结构，这棵树被称为 fail 树，它具有一些优美的性质，具体来说是在 fail 树上存在祖孙关系的两点对应的字符串具有后缀关系。

# [NOI2014] 动物园

## description

对  $S$  的每个前缀  $S_{[1,i]}$ , 求它有多少个长度  $\leq \lfloor \frac{i}{2} \rfloor$  的 border。

## constraint

$$|S| \leq 5 \times 10^6.$$

## solution

维护 AC 自动机上的两个状态  $p_i, q_i$ , 其中  $p_i$  表示  $S_{[1,i]}$ ,  $q_i$  表示其最长的  $\leq \lfloor \frac{i}{2} \rfloor$  的 border。可以发现  $i$  的答案就是  $q_i$  在 fail 树上的深度。

$p_i \rightarrow p_{i+1}$  的转移显然是  $O(1)$  的, 容易发现  $q_i \rightarrow q_{i+1}$  的转移也是均摊  $O(1)$  的。

## description

对  $S$  的每个前缀  $S_{[1,i]}$ , 求它有多少个长度  $\leq \lfloor \frac{i}{2} \rfloor$  的 border。

## constraint

$$|S| \leq 5 \times 10^6.$$

## solution

维护 AC 自动机上的两个状态  $p_i, q_i$ , 其中  $p_i$  表示  $S_{[1,i]}$ ,  $q_i$  表示其最长的  $\leq \lfloor \frac{i}{2} \rfloor$  的 border。可以发现  $i$  的答案就是  $q_i$  在 fail 树上的深度。

$p_i \rightarrow p_{i+1}$  的转移显然是  $O(1)$  的, 容易发现  $q_i \rightarrow q_{i+1}$  的转移也是均摊  $O(1)$  的。

# [NOI2011] 阿狸的打字机

## description

给你一棵  $n$  个节点的 Trie 树， $q$  次询问 Trie 树上  $x$  节点代表的字符串在  $y$  节点代表的字符串中出现了多少次。

## constraint

$$1 \leq n, q \leq 10^6.$$

## solution

问题等价于问  $y$  节点代表的字符串有多少个前缀包含  $x$  作为后缀，也就是问 Trie 树上根节点到  $y$  路径上的所有点中有多少点在 fail 树上  $x$  的子树里，树状数组维护即可。

# [NOI2011] 阿狸的打字机

## description

给你一棵  $n$  个节点的 Trie 树， $q$  次询问 Trie 树上  $x$  节点代表的字符串在  $y$  节点代表的字符串中出现了多少次。

## constraint

$$1 \leq n, q \leq 10^6.$$

## solution

问题等价于问  $y$  节点代表的字符串有多少个前缀包含  $x$  作为后缀，也就是问 Trie 树上根节点到  $y$  路径上的所有点中有多少点在 fail 树上  $x$  的子树里，树状数组维护即可。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

## description

从前有一棵树，树上的每条边上都有一个小写字母。

有  $q$  次这样的询问：每次指出树上的两点  $x, y$  并给出一个由小写字母构成的字符串  $str$ ，询问串  $str$  在  $S_{x,y}$  中出现了多少次，其中  $S_{x,y}$  表示树上从点  $x$  到点  $y$  的唯一路径上每条边上的小写字母顺次连接形成的字符串。

注意  $S_{x,y} \neq S_{y,x}$ 。

## constraint

$$1 \leq n, q \leq 10^5, 1 \leq \sum |str| \leq 3 \times 10^5.$$

<sup>7</sup> 去年考过应该。



字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

## solution

假设询问串为  $T$ 。

先解决路径在 LCA 处“拐弯”的情况。提取“拐弯”前后总长不超过  $2|T| - 2$  的字符串，暴力做一遍 KMP 即可解决这一部分的贡献。

考虑不“拐弯”的情况。“ $T$  在  $S$  中出现了多少次”这个问题等同于“ $T$  是  $S$  多少个前缀的后缀”，等同于问一条直链上有多少点在 fail 树上  $T$  对应节点的子树中。

处理链询问的惯用手段是离线 DFS 并差分，这里需要额外维护个树状数组以实现 fail 树上的子树查询。

时间复杂度  $O(n \log n + \sum |T|)$ 。

# Suffix Array

求一个长为  $n$  的串  $S$  的后缀排序，得到一个  $[1, n] \rightarrow [1, n]$  的置换 SA。这个过程一般使用倍增 + 基数排序，复杂度为  $O(n \log n)$ 。

通常会额外记录  $\text{Rank} = \text{SA}^{-1}$ 。

定义  $\text{Height}(i) = \text{lcp}(\text{SA}(i-1), \text{SA}(i))$ ，于是有  
 $\text{lcp}(\text{SA}(l), \text{SA}(r)) = \min_{l < i \leq r} \text{Height}(i)$ ，可以通过 RMQ 算法实现  
 $O(n \log n) - O(1)$  求任意两后缀 lcp。

Height 可以  $O(n)$  求出，得益于如下引理：

## 引理

$$\text{Height}(\text{Rank}(i)) \geq \text{Height}(\text{Rank}(i-1)) - 1$$

$S$  的本质不同子串数量为  $\frac{n(n+1)}{2} - \sum_{i=2}^n \text{Height}(i)$ 。

求一个长为  $n$  的串  $S$  的后缀排序，得到一个  $[1, n] \rightarrow [1, n]$  的置换 SA。这个过程一般使用倍增 + 基数排序，复杂度为  $O(n \log n)$ 。

通常会额外记录  $\text{Rank} = \text{SA}^{-1}$ 。

定义  $\text{Height}(i) = \text{lcp}(\text{SA}(i-1), \text{SA}(i))$ ，于是有  
 $\text{lcp}(\text{SA}(l), \text{SA}(r)) = \min_{l < i \leq r} \text{Height}(i)$ ，可以通过 RMQ 算法实现  
 $O(n \log n) - O(1)$  求任意两后缀 lcp。

Height 可以  $O(n)$  求出，得益于如下引理：

## 引理

$$\text{Height}(\text{Rank}(i)) \geq \text{Height}(\text{Rank}(i-1)) - 1$$

$S$  的本质不同子串数量为  $\frac{n(n+1)}{2} - \sum_{i=2}^n \text{Height}(i)$ 。

求一个长为  $n$  的串  $S$  的后缀排序，得到一个  $[1, n] \rightarrow [1, n]$  的置换 SA。这个过程一般使用倍增 + 基数排序，复杂度为  $O(n \log n)$ 。

通常会额外记录  $\text{Rank} = \text{SA}^{-1}$ 。

定义  $\text{Height}(i) = \text{lcp}(\text{SA}(i-1), \text{SA}(i))$ ，于是有  
 $\text{lcp}(\text{SA}(l), \text{SA}(r)) = \min_{l < i \leq r} \text{Height}(i)$ ，可以通过 RMQ 算法实现  
 $O(n \log n) - O(1)$  求任意两后缀 lcp。

Height 可以  $O(n)$  求出，得益于如下引理：

## 引理

$$\text{Height}(\text{Rank}(i)) \geq \text{Height}(\text{Rank}(i-1)) - 1$$

$S$  的本质不同子串数量为  $\frac{n(n+1)}{2} - \sum_{i=2}^n \text{Height}(i)$ 。

## [NOI2015] 品酒大会

## description

给出长度为  $n$  的串  $S$  和  $w: [1, n] \rightarrow \mathbb{Z}$ , 对每个  $r = 0, 1, \dots, n-1$ , 求

$$\sum_{i,j \in [1,n], \text{lcp}(i,j)=r} 1, \max_{i,j \in [1,n], \text{lcp}(i,j)=r} w(i) \times w(j)$$

## constraint

$$n \leq 3 \times 10^5, |w(i)| \leq 10^9.$$

## solution

按 Height 的大小顺序合并即可。

## [NOI2015] 品酒大会

## description

给出长度为  $n$  的串  $S$  和  $w: [1, n] \rightarrow \mathbb{Z}$ , 对每个  $r = 0, 1, \dots, n-1$ , 求

$$\sum_{i,j \in [1,n], \text{lcp}(i,j)=r} 1, \max_{i,j \in [1,n], \text{lcp}(i,j)=r} w(i) \times w(j)$$

## constraint

$$n \leq 3 \times 10^5, |w(i)| \leq 10^9.$$

## solution

按 Height 的大小顺序合并即可。

# 平方串

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

定义平方串为可以写成  $s + s$  的字符串。

存在一种  $O(n \ln n)$  定位长为  $n$  的串  $S$  中所有平方串的算法：

- 枚举平方串的长度  $2l$ ，在  $S$  中每  $l$  位设立一个标记。任意长为  $2l$  的平方串会覆盖恰好两个相邻的标记。
- 枚举一对相邻标记所在下标  $i, j$ ，求出  $i, j$  的 lcp 和 lcs。若两者长度合起来达到了  $l$ ，则说明找到了一列下标连续的长为  $2l$  的平方串。

掌握这项技术，就可以获得 [NOI2016] 优秀的拆分 的最后 5 分啦！

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

## 51nod Double Number

## description

给定  $S \in \{0, 1, \dots, 9\}^*$ , 求  $S$  中所有没有前导零的平方串的一半对应的数的和。结果对 998244353 取模。

## constraint

$$|S| \leq 10^5.$$

## solution

通过通用的平方串算法, 我们可以得到  $O(n \ln n)$  个三元组  $(l, r, 2p)$ , 表示左端点为  $[l, r]$ , 长度为  $2p$  的所有串是平方串。用  $f(i)$  表示  $S$  长度为  $i$  的前缀表示的数。如果不要求没有前导零, 则  $(l, r, 2p)$  对答案的贡献就是

$$\sum_{k=l}^r (f(k+p-1) - 10^p f(k-1))$$

事实上只要把满足  $S_{i+1} = 0$  的  $f(i)$  改成 0 就能处理前导零了。





## 51nod Double Number

## description

给定  $S \in \{0, 1, \dots, 9\}^*$ , 求  $S$  中所有没有前导零的平方串的一半对应的数的和。结果对 998244353 取模。

## constraint

$$|S| \leq 10^5.$$

## solution

通过通用的平方串算法, 我们可以得到  $O(n \ln n)$  个三元组  $(l, r, 2p)$ , 表示左端点为  $[l, r]$ , 长度为  $2p$  的所有串是平方串。用  $f(i)$  表示  $S$  长度为  $i$  的前缀表示的数。如果不要求没有前导零, 则  $(l, r, 2p)$  对答案的贡献就是

$$\sum_{k=l}^r (f(k+p-1) - 10^p f(k-1))$$

事实上只要把满足  $S_{i+1} = 0$  的  $f(i)$  改成 0 就能处理前导零了。

## SAM: Definition

串  $S$  的后缀自动机 (Suffix Automaton, SAM) 是一个能接受  $S$  的所有后缀的 (确定性) 有限自动机。具体地:

- SAM 是一张 DAG, 图的结点被称为状态, 边被称为转移。
- 存在一个状态  $t_0$  称为初始状态, 其余状态均可通过  $t_0$  到达。
- 每个转移都被一个字符  $c \in \Sigma$  标识。
- 存在一些终止状态。一个串  $s$  是  $S$  的后缀, 当且仅当存在一条从  $t_0$  出发到达某个终止状态的路径, 把路径上的所有字符顺序连接起来后恰好得到  $s$ 。

自然地定义“路径”与“字符串”的对应, 我们会说 SAM 的一个状态对应一个字符串集合, 且这个字符串集合中所有字符串的 endpos 集合<sup>8</sup>相同。特别地, 我们认为  $t_0$  对应  $\{\epsilon\}$ 。

<sup>8</sup> 将在下一页定义。

## SAM: Notation

对于  $S$  的一个子串  $s$ , 用  $\text{endpos}(s)$  表示  $s$  在  $S$  中的所有结束位置。把  $\text{endpos}$  集合相同视作一种等价关系, 我们陈述三条引理:

- ① 若  $S$  的两个子串  $u, w$  满足  $|u| \leq |w|$ , 则  $\text{endpos}(u) = \text{endpos}(w) \Leftrightarrow u$  只以  $w$  后缀的形式出现在  $S$  中;
- ② 要么  $\text{endpos}(u) \cap \text{endpos}(w) = \emptyset$ , 要么  $\text{endpos}(w) \subseteq \text{endpos}(u)$ ;
- ③ 同一等价类中的所有子串按照长度递增排序, 前一个串是后一个串的后缀, 且长度相差恰好 1。

对于 SAM 中的一个状态  $t$ ,  $\text{endpos}(t)$  表示  $t$  对应的字符串集合中某个串  $s$  的  $\text{endpos}(s)$ 。这是良定的。

我们认为  $\text{endpos}(t_0) = \{0, 1, \dots, |S|\}$ 。

承认前述引理后，我们约定一些记号。对于状态  $t$ ,

- $\text{longest}(t), \text{shortest}(t)$  分别表示  $t$  对应的字符串集合中最长串与最短串；
- $\text{len}(t) = |\text{longest}(t)|$ 。

一个状态  $t$  的后缀链接<sup>9</sup> $\text{link}(t)$  是另一个状态  $t'$ ，满足  $\text{endpos}(t) \subsetneq \text{endpos}(t')$  且  $|\text{endpos}(t')|$  是最小的<sup>10</sup>。

可以验证比  $\text{shortest}(t)$  长度少 1 的后缀所在的  $\text{endpos}$  等价类就是  $\text{link}(t)$ ，即

$$\text{len}(\text{link}(t)) = |\text{shortest}(t)| - 1$$

<sup>9</sup> 也有人将其记作  $\text{fail}$  或  $\text{fa}$ 。

<sup>10</sup> 由第二条引理，这是良定的。

# SAM: Construction

接下来介绍后缀自动机的构造方法。

这个过程是在线的或者说增量的，即始终维护  $S$  的一个前缀的 SAM，每次根据这个 SAM 构造  $S$  下一个前缀的 SAM。

在构造过程中，考虑额外维护  $\text{len}$  和  $\text{link}$  两个值。用  $\text{trans}(t, c)$  表示状态  $t$  通过标识有字符  $c$  的转移出边到达的新状态。

$S$  的第一个前缀是  $\epsilon$ ，对应的 SAM 只有一个状态  $t_0$ ，约定  $\text{len}(t_0) = 0, \text{link}(t_0) = \text{undefined}$ 。

假设已知  $S$  的 SAM，需要构造  $S + c$  的 SAM，其中  $c \in \Sigma$ 。

- 令  $last$  表示  $S$  对应的状态。新建一个状态  $cur$ , 赋值  $\text{len}(cur) \leftarrow \text{len}(last) + 1$ ,  $\text{link}(cur)$  待确定。
- 找到状态  $p$  满足  $\text{endpos}(p) \subseteq \text{endpos}(last)$ , 且  $p$  有字符  $c$  的转移出边, 且  $|\text{endpos}(p)|$  是最小的。  
对所有  $p'$  满足  $\text{endpos}(last) \subseteq \text{endpos}(p') \subsetneq \text{endpos}(p)$ , 赋值  $\text{trans}(p', c) \leftarrow cur$ 。
  - 若  $p$  不存在, 直接令  $\text{link}(cur) \leftarrow t_0$ , 构造结束。
  - 否则记  $\text{trans}(p, c) = q$ 。我们断言  $\text{len}(q) \geq \text{len}(p) + 1$ , 分类讨论:
    - 若  $\text{len}(q) = \text{len}(p) + 1$ , 则令  $\text{link}(cur) \leftarrow q$ , 构造结束。
    - 若  $\text{len}(q) > \text{len}(p) + 1$ , 则需要分裂状态  $q$ 。  
具体的, 新建状态  $clone$ , 赋值  $\text{link}(clone) \leftarrow \text{link}(q)$ ,  $\text{trans}(clone, *) \leftarrow \text{trans}(q, *)$ ,  $\text{len}(clone) = \text{len}(p) + 1$ , 以及  $\text{link}(q) \leftarrow clone, \text{link}(cur) \leftarrow clone$ 。  
还需要部分原本指向  $q$  的转移出边改为指向  $clone$ , 来自于  $p'$  满足  $\text{endpos}(p) \subseteq \text{endpos}(p')$ 。
- 更新  $last \leftarrow cur$ 。

```

void extend(char c){
    int cur = ++tot, p = last;
    len[cur] = len[last] + 1;
    while (p && !trans[p][c]) tr[p][c] = cur, p = link[p];
    if (!p) link[cur] = 1; // stand for t_0
    else{
        int q = trans[p][c];
        if (len[q] == len[p] + 1) link[cur] = q;
        else{
            int clone = ++tot;
            memcpy(trans[clone], trans[q], sigma * 4);
            link[clone] = link[q]; len[clone] = len[p] + 1;
            link[q] = link[cur] = clone;
            while (p && trans[p][c] == q)
                trans[p][c] = clone, p = link[p];
        }
    }
    last = cur;
}

```

# SAM: Why It Works

考虑从  $S$  到  $S + c$ , 我们其实只希望能够出现一个新状态来识别  $S + c$  这个新后缀。进一步地, 我们希望知道与  $S + c$  这个串  $\text{endpos}$  等价的串 (对应状态为  $cur$ ) 还有哪些。

一种情况是  $c$  这个字符在  $S$  中没出现过, 即 “即使是字符串  $c$  也与  $S + c \text{ endpos}$  等价”, 此时  $\text{link}(cur) \leftarrow t_0$  表明  $|\text{shortest}(cur)| = 1$ 。

否则, 存在串  $x$  满足  $x + c$  在  $S$  中出现过。我们希望找到最长<sup>11</sup>的满足  $\text{endpos}(S + c) \subsetneq \text{endpos}(x + c)$  的  $x + c$ , 并让  $|\text{shortest}(cur)| = |x + c| + 1$ 。

这个最长的  $x$  就是  $\text{longest}(p)$ , 所以  $x + c \in q$  对应的字符串集合。当  $\text{len}(q) = \text{len}(p) + 1$  时,  $x + c = \text{longest}(q)$ , 所以可以直接  $\text{link}(cur) \leftarrow q$ 。当  $\text{len}(q) > \text{len}(p) + 1$  时, 需要分裂出状态  $clone$  使  $x + c = \text{longest}(clone)$ , 从而有  $\text{link}(cur) \leftarrow clone$ 。

<sup>11</sup> 这也是良定的。



# SAM: Complexity Analysis

由构造可以看出，一个长度为  $n (n \geq 2)$  的字符串  $S$  的 SAM 有不超过  $2n - 1$  个状态，因为增量构造的前两步一定不会产生 *clone*。串 `abbb...bbb` 达到了这个上界。

此外转移数是线性的，构造 SAM 的时间复杂度也是线性的。后者甚至与字符集大小无关，某种程度上来说也是出乎意料的。

一个需要注意的问题在于实现中 `trans` 要怎么存。一般认为  $\sigma$  是一个常数，可以开  $n \cdot \sigma$  大小的数组实现存储。如果  $\sigma$  很大，可以使用平衡树 (`std::map<int, int>`)。

关于这部分任何细节以及证明可参考 [OI-Wiki 后缀自动机](#)。

## SAM: Applications

给定字符串  $S$ , 建立  $S$  的后缀自动机, 可以

- 对于给定的  $T$ , 判断  $T$  是不是  $S$  的子串。
- 求  $S$  不同子串数量、总长度、字典序第  $k$  小子串。
- 对于给定的  $T$ , 求  $T$  在  $S$  中的出现次数、所有出现位置。
- 求两个串、多个串的最长公共子串。
- etc.

建出 SAM 后, 可以通过线段树合并维护出每个状态的  $\text{endpos}$ 。

随便找了一些不一定做过的题，请根据实际情况自行选择练习。

- 【模板】后缀自动机 (SAM)
- [SDOI2016] 生成魔咒
- [TJOI2015] 弦论
- [AHOI2013] 差异
- Rikka with String
- [NOI2018] 你的名字
- 「雅礼集训 2017 Day1」字符串
- 「雅礼集训 2017 Day7」事情的相似度
- [集训队作业 2018] 后缀树节点数

# PAM: Introduction

PAM 比 Manacher 好写。

— gold\_genius

类似 SAM，我们也希望构造一种识别串  $S$  所有回文子串的自动机。  
如下引理使这种设想成为可能。

## 引理

$S$  的本质不同回文子串只有  $O(|S|)$  个。

## 证明.

向后添加一个字符时，只有最长回文后缀可能是新的。



# PAM: Introduction

PAM 比 Manacher 好写。

— gold\_genius

类似 SAM，我们也希望构造一种识别串  $S$  所有回文子串的自动机。  
如下引理使这种设想成为可能。

## 引理

$S$  的本质不同回文子串只有  $O(|S|)$  个。

## 证明.

向后添加一个字符时，只有最长回文后缀可能是新的。



## PAM: Definition

串  $S$  的回文自动机 (Palindrome Automaton, PAM) 是包含如下内容的 (确定性) 有限状态自动机:

- 每个状态  $t$  代表一个回文串  $\text{str}(t)$ 。有两个特殊状态  $t_0, t_{-1}$ , 约定  $\text{str}(t_0) = \epsilon, \text{str}(t_{-1}) = \text{"the string of length -1"}$ 。
- 存在  $t$  到  $t'$  标有字符  $c$  的转移当且仅当  $\text{str}(t') = c\text{str}(t)c$ 。  
约定  $c\text{str}(t_{-1})c = c$ 。
- 状态  $t$  的后缀链接  $\text{link}(t)$  是另一个状态  $t' \neq t_0$  满足  $\text{str}(t')$  是  $\text{str}(t)$  的后缀, 且是最长的。约定  $\text{link}(t_0) = t_{-1}, \text{link}(t_{-1}) = \text{undefined}$ 。

通过构造过程可知所有的约定都是自然的。

仍考虑增量构造。当向后添加字符时，我们只关心包含这个字符的最长回文子串，也即最长回文后缀。

$S + c$  的最长回文后缀必然形如  $cS'c$ ，其中  $S'$  要么是一个回文子串，要么是  $\epsilon$  或者 "the string of length -1"。不论如何，一定存在一个 PAM 中已有状态  $p$  使  $\text{str}(p) = S'$ 。找到这个  $p$ ，检查  $p$  是否有  $c$  的转移出边，若没有则新建状态  $cur$  并建立转移，表示发现了新的回文串。

至于如何找到  $p$ ，考虑  $S'$  是  $S$  的回文后缀，我们记录  $last$  为  $S$  的最长回文后缀对应的状态，那么通过  $last$  的 link 链 (不断求 link 得到的状态序列) 便可以找到  $p$ 。

建立  $cur$  后需要构造  $\text{link}(cur)$ 。 $\text{str}(\text{link}(cur))$  是  $\text{str}(cur) = cS'c$  的回文后缀，故必然形如  $cS''c$ ，其中  $S''$  是  $S'$  的回文后缀。同理，满足  $\text{str}(p') = S''$  的  $p'$  也可以通过  $p$  的 link 链找到。

```

int getf(int t) {
    while (s[n - len[t] - 1] != s[n])
        t = link[t];
    return t;
}

void extend(char c) {
    s[++n] = c;
    int p = getf(last);
    if (!trans[p][c]) {
        int cur = ++tot;
        len[cur] = len[p] + 2;
        link[cur] = trans[getf(link[p])][c];
        trans[p][c] = cur;
    }
    last = trans[p][c];
}

```



# PAM: Property

类似 border 的 “ $O(\log n)$  段等差数列”，回文串也有类似的性质。

## 引理

$S$  的所有回文后缀长度构成  $O(\log n)$  段不交的等差数列。

一些问题需要利用到这个性质，比如 [Codeforces 932G Palindrome Partition](#)，[区间本质不同回文子串计数](#) 等。

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

# Gym103447A. So Many Lucky Strings

## description

有一列小写字母字符串  $s_1, s_2, \dots, s_n$ , 求有多少个子序列  $\{i_1, i_2, \dots, i_k\} \subseteq [1, n]$ , 满足  $s_{i_1} + s_{i_2} + \dots + s_{i_k}$  是回文串。

## constraint

$n \leq 100, \sum |s_i| \leq 10^5, |\Sigma| = 26.$

## Gym103447L. Karshilov's Matching Problem

## description

有一些字符串  $t_1, t_2, \dots, t_n$ , 每个串都有价值  $w_i$ 。

你需要维护一个串  $S$ , 支持  $m$  次如下两种操作:

- 给出  $l, c$ , 把  $S$  的后  $l$  个字符全部改成  $c$ ;
- 给出  $l$ , 查询  $S_{[1,l]}$  的总价值, 被定义为  $\{t_i\}$  中每个串的出现次数乘上价值的和。

## constraint

$$\sum |t_i| \leq 10^5, m \leq 3 \times 10^5, |S| \leq 3 \times 10^5, |\Sigma| = 10.$$

字符串

zsy

outline

最小表示法

Manacher

ex-KMP

Hash

KMP

border 与周期理论

Trie

AC automaton

SA

平方串

SAM

PAM

习题课

讲完了

# 谢谢大家！

## 祝大家学业有成！