# 3 串串划分

**Author: FizzyDavid**

## 3.1 命题背景

这道题是一道原创题。作者之前在训练中遇到了循环串相关的问题，独立思考出了一种处理循环子串的时间复杂度优秀的算法。

子任务 2 的分析中的线性状态数 dp 是这道题的基础做法，这道题的本质是对这个做法的优化。

## 3.2 一些约定

- 一个字符串 $S$，它的长度是 $|S|$。
- 字符从 1 开始编号，$S[i]$ 表示第 $i$ 个字符。
- 两个字符串相等当且仅当他们的长度与编号相同的字符都相等。
- 一个 $l$ 到 $r$ 的子串用 $S[l...r]$ 表示，注意为了方便，当 $l \geq r$ 时 $S[l...r]$ 为空串。
- 若 $S = \overline{S_1S_2...S_k}$，且 $S_i = S_{i+1}(1 \leq i < k)$，那么 $S_1, S_2, ..., S_k$ 就是 $S$ 的循环节，它们的长度是 $S$ 的循环节长度，$k$ 称为循环次数。注意一个串可能有多个循环节长度和循环次数，并且至少有一个循环节长度和循环次数。

## 3.3 子任务 1

### 3.3.1 数据范围

$|S| \leq 300$

### 3.3.2 期望时间复杂度

$\Theta(n^3)$

### 3.3.3 算法分析

本题对于我们划分的每个子串的要求有两个，第一是相邻的不相同，第二是不循环。若我们预处理出哪些子串是不循环的，就可以用 $\Theta(n^3)$ 的简单 dp 来计算答案。dp 可以记录最后一次划分的子串是什么，状态数 $\Theta(n^2)$，每次转移 $\Theta(n)$。

考虑如何预处理出哪些子串是不循环的。我们可以枚举循环子串最靠左的一个循环节，再从小到大枚举次数，每次判断新增的循环节是否与枚举的循环节的一样，这样就可以获得所有的循环子串。对于一个左端点，枚举的次数是一个调和级数，该部分时间复杂度为 $\Theta(n^2 \log n)$。

注意以上做法涉及到判断两个子串是否相等，可以使用字符串哈希来做到 $\Theta(n)$ 预处理 $\Theta(1)$ 询问的复杂度。总复杂度为 $\Theta(n^3)$。

## 3.4 子任务 2

### 3.4.1 数据范围

$|S| \leq 2000$

### 3.4.2 期望时间复杂度

$\Theta(n^2 \log n)$

### 3.4.3 算法分析

子任务 1 中的做法的时间复杂度瓶颈在 dp，考虑如何优化。我们发现转移时相邻的相同的串最多只有一个，于是我们只需要计算任意划分的方案数减去相邻的相同的方案数。可以使用简单的前缀和优化做到转移 $\Theta(1)$，于是总时间复杂度可以降到 $\Theta(n^2 \log n)$。

由于这个方法的 dp 状态数高达 $\Theta(n^2)$，要想解决之后 $n \leq 200000$ 的数据规模，需要使用状态数更小的 dp。接下来的算法是后面几个子任务的基础。

我们考虑使用一个线性状态数的 dp，即 $dp(i)$ 表示合法的划分 $s[1...i]$ 的方案数。转移仍然使用枚举最后一个划分的子串是什么的方法，但是为了满足题目中 **相邻两个子串不相等**的条件，可以使用容斥来计算。

我们对题目中的条件 **相邻两个子串不相等**容斥后，就转化为限制某些相邻的子串相等。我们需要对于所有可能的限制，计算满足这些相邻的子串的相等的方案数，并且每个限制将会对方案数贡献 $-1$ 的系数。容斥后的条件将子串序列划分成了若干个 **段**，满足每一段中的相邻两个子串存在限制，且相邻两段的子串之间没有限制。我们 dp 转移就使用 **段**来转移，即枚举最后一个子串所在的段的长度。

一个例子：

假设最后划分的子串序列为：$S_1, S_2, S_3, S_4, S_5, S_6$。

一种容斥后的限制为：$S_1, S_2 = S_3 = S_4, S_5 = S_6$。其中每个等号贡献了 $-1$ 的系数。转移时将从初始状态转移到 $S_1$，再转移到 $\overline{S_1S_2S_3S_4}$，再转移到原串。

我们可以先枚举最后一个子串是什么，再枚举最后一个 **段**是什么。由于枚举的子串要求是不循环的，所以等同于枚举的子串恰好为这个段所表示的子串的最小循环节。下面是 dp 转移方程：

$$dp_i = \sum_{l=1}^{i} \sum_{k=1}^{\lfloor \frac{i}{l} \rfloor} (-1)^{k-1} dp_{i-lk} [s[i-lk+1...i] \text{的最小循环节长度为} l]$$

其中 $[P]$ 是一个表达式，当 $P$ 为真时值为 1，否则为 0。即枚举要满足最后一段的最小循环节长度恰好为 $l$。

记字符串 $s$ 的最小循环节长度为 $\text{mnper}(s)$，最大循环次数为 $\text{mxrep}(s)$。这里一个串若不循环，我们定义它的最小循环节长度为串的长度，最大循环次数为 1。注意这里有 $\text{mnper}(s)\text{mxrep}(s) = |s|$。我们不妨改变一下枚举顺序，直接枚举 $j = lk$ ：

$$dp_i = \sum_{j=1}^{i} \sum_{l=1}^{j} (-1)^{\frac{j}{l}-1} dp_{i-j} [s[i-j+1...i] \text{的最小循环节长度为} l]$$

由于最小循环节长度唯一，所以推出式子：

$$dp_i = \sum_{j=1}^{i} dp_{j-1} (-1)^{\text{mxrep}(s[j...i])-1}$$

我们仍然可以用子任务 1 中的 $\Theta(n^2 \log n)$ 算法预处理出每个子串的最大循环次数。之后我们可以直接用这个转移方程在 $\Theta(n^2)$ 的时间求得答案。

## 3.5 子任务 3

### 3.5.1 数据范围

$|S| \leq 8000$

### 3.5.2 期望时间复杂度

$\Theta(n^2)$

### 3.5.3 算法分析

考虑优化子任务 2 中预处理的复杂度。这里我们需要使用最小循环节长度的性质：

- 设一个长度为 $n$ 的字符串 $s$ 的最大 border 长度为 $p$，若 $n-p$ 整除 $n$，那么就有 $\text{mnper}(s) = n-p$，否则 $\text{mnper}(s) = n$。

如果有 $0 \leq p < |s|$，$s[1...p] = s[n-p+1...n]$，那么 $p$ 就是 $s$ 的一个 border 长度，$n-p$ 就是 $s$ 的一个 period。最大 border 长度就是满足条件的最大的 $p$。

KMP 算法可以对于每个前缀计算出它的最大 border 长度，也就是说可以对每个前缀计算出它的最小循环节长度和最大循环次数。那么我们在 dp 转移时，对前缀倒着跑一遍 KMP 就可以计算出该前缀的每个后缀的最大循环次数。总复杂度就降到了 $\Theta(n^2)$。

## 3.6 子任务 4

### 3.6.1 数据范围

$|S| \leq 200000$ 且 $S$ 的每个字符从指定的某个字符集中随机生成。

### 3.6.2 期望时间复杂度

$\Theta(np)$，其中 $p$ 为一个 60 左右的常数。

### 3.6.3 算法分析

首先特判掉字符全相等的情况。注意到一个随机串，在长度大的时候有极大的概率最大循环次数为 1。我们使用下面的转移方程:

$$dp_i = (\sum_{j=1}^{i} dp_{j-1}) - 2(\sum_{j=1}^{i} dp_{j-1}[mxper(s[j...i]) \bmod 2 == 0])$$

可以使用前缀和优化。由于 $[mxper(s[j...i]) \bmod 2 == 0]$ 在 $i - j + 1$ 大的时候几乎不可能为 1，所以我们设置参数 $p$，只枚举 $j \geq i - p$ 的情况。$p$ 可以设为 60 左右。

## 3.7 子任务 5

### 3.7.1 数据范围

$|S| \leq 200000$ 且保证对于 $S$ 的任意子串，要么它是不循环的，要么它的循环次数不会超过 10。

### 3.7.2 期望时间复杂度

$\Theta(n \log n)$ 或 $\Theta(n \log^2 n)$

### 3.7.3 算法分析

由子任务 4 中的做法启发，我们考虑找到那些循环次数大于等于 2 的子串。我们先枚举循环节长度 $L$。我们认为下标是 $L$ 的倍数的位置是一个关键点。由于一个循环次数大于等于 2 的子串必然覆盖至少两个相邻的关键点，所以我们对于 $kL$ 和 $(k+1)L$，计算它们向前的 lcp 与向后的 lcp。即我们找到最大的 $l1$ 与 $l2$，使得 $s[kL-l1+1...kL] = s[(k+1)L-l1+1...(k+1)L]$ 且 $s[kL...kL+l2-1] = s[(k+1)L...(k+1)L+l2-1]$。若 $l1+l2 \geq L+1$，那我们就找到了至少一个循环次数大于等于 2 的子串。

具体来说，设 $l = kL - l1 + 1$，$r = (k+1)L + l2 - 1$，那么对于满足下列条件的 $l'$ 和 $r'$，$s[l'...r']$ 都有一个长度为 $L$ 的循环节：

$$l \leq l' \leq r' \leq r,\ r' - l' + 1 \geq L,\ (r' - l' + 1) \bmod L = 0$$

我们可以用一个三元组 $(L, l, r)$ 来表示循环节长度以及我们找到的区间。由于该子任务限制所有子串的循环次数不超过 10，所以循环次数大于等于 2 的子串个数不会很多（其实下面我们可以证明在该限制下不会超过 $\Theta(10n \log n)$ ），我们考虑直接记录所有循环次数大于等于 2 的子串，dp 转移时直接用我们记录的子串转移就好了。

接下来分析时间复杂度。关键点的个数是 $\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + ... = \Theta(n \ln n)$。如果我们使用后缀数组与 ST 表 $\Theta(n \log n)$ 预处理 $\Theta(1)$ 询问 lcp，那么求相邻关键点的 lcp 这一部分就可以做到 $\Theta(n \log n)$。

## 3.8 子任务 6

### 3.8.1 数据范围

$|S| \leq 200000$。

### 3.8.2 期望时间复杂度

$\Theta(n \log n)$ 或 $\Theta(n \log^2 n)$

### 3.8.3 算法分析

我们继续改进子任务 5 中的做法。对于上述做法中的一个三元组 $(L, l, r)$，表示 $s[l...r]$ 中所有长度是 $L$ 的倍数的子串都存在一个长度为 $L$ 的循环节，但是我们的 dp 转移式要求我们算出它的最小循环节长度（最大循环次数），而不是任意一个循环节。我们尝试修正我们的做法：

- 若 $s[l...l+L-1]$ 不是不循环的，那么 $s[l...r]$ 中长度是 $L$ 的倍数的所有子串一定拥有一个更小的循环节。我们在遇到这种情况时就不处理 $(L, l, r)$ 这个三元组。

这样我们就可以保证 $L$ 是其中长度是 $L$ 的倍数的子串的最小循环节长度。$s[l...r]$ 中可能会包含很多循环次数大于等于 2 的子串，但是由于右端点固定时，左端点是一个公差为 $L$ 的等差数列，不难发现可以使用前缀和的技巧来优化转移，其中每个位置使用 s[l...r] 中的子串进行 dp 转移都可以做到 $\Theta(1)$。注意由于只有在长度大于等于 $2L$ 是循环次数才会大于等于 2，所以我们实际上需要转移的位置数量为 $r-l+1-2L$。

接下来我们来分析复杂度。关键点与 lcp 的部分的复杂度仍是 $\Theta(n \log n)$，我们只需要分析转移的复杂度，即 $r-l+1-2L$ 的和。我们发现 $r-l+1-2L$ 恰好为 $s[l...r]$ 中长度为 $2L$ 的子串的个数，即最大循环次数恰好为 2 且最小循环节长度为 $L$ 的串的个数。我们对所有的 $L$ 求和后，这个和就变成了最大循环次数恰好为 2 的子串个数。我们接下来证明这种子串个数最多只有 $\Theta(n \log n)$ 个：

对于所有左端点相同的子串，我们考虑那些满足条件的子串，并且列出他们的循环节长度 $l_i, l_2...l_k$，并且将其从小到大排好序。一个字符串若存在两个 period $p_1, p_2$，那么必然存在一个长度为 $\gcd(p_1, p_2)$ 的 period（$p_1, p_2 \leq |S|/2$）。而对于一个前缀 $s[1...len]$，如果存在一个长度为 $p$ 的 period，那么 $l_i (l_i \leq len)$ 中不会出现大于 $p$ 的倍数，不然就不满足条件了。

我们从后往前考虑 $l_i$ 及 $s[1...2l_i]$ 的 period，首先对于 $s[1...2l_k]$ 肯定存在一个长度为 $l_k$ 的 period，设我们知道的 $s[1...2l_i]$ 的 period 长度为 $p$，我们先将 $p$ 设为 $l_k$。

当我们考虑到 $i(i < k)$ 时，有下面几种情况：

- 情况 1：$2l_i < p$ ： 将 $p$ 设为 $l_i$。
- 情况 2：$l_i \neq p$ ： 将 $p$ 设为 $\gcd(p, l_i)$。此时由于 $l_i$ 不可能是 $p$ 的倍数，所以 $2\gcd(p, l_i) \leq p$。
- 情况 3：$l_i = p$。

注意到情况 1、2 中 $p$ 都减少了至少一半，而当 $p = 1$ 时则不可能有新的 $l_i$ 了，情况 1、2 只会出现 $\Theta(\log n)$ 次。那么不同的 $p$ 只有 $\Theta(\log n)$ 个，所以第三种情况只会出现 $\Theta(\log n)$ 次。而每个 $l_i(i < k)$ 都对应着某一种情况，所以 $k$ 是 $\Theta(\log n)$ 的。所以总共只有 $\Theta(n \log n)$ 个最大循环次数恰好为 2 的子串。

至此，我们就分析出算法的总复杂度为 $\Theta(n \log n)$。

顺带一提，其实在这个算法中处理的三元组 $(L, l, r)$ 都对应到不同的 run，根据 Runs Theorem，run 的个数是 $\Theta(n)$ 的，可以知道三元组的个数也是 $\Theta(n)$ 的。