浅谈 Lyndon 分解

雅礼中学 胡昊

摘 要

字符串是信息学奥林匹克竞赛的重要考点，Lyndon 常用于与字典序最优化相关的问题。

1 基本定义

定义 $|S|$ 为字符串 S 的长度，S_i 为 S 中的第 i 个字符（$i \in [0, |S|)$）。

定义两个字符串 S, T 相等，则须满足 $|S| = |T|, \forall i \in [0, |S|), S_i = T_i$。

定义一个字符串 S 的一个子串 $S_{l...r}$ 为 S 中的第 l 个字符到第 r 个字符组成的字符串。

定义字符串 S 是另一字符串 T 的前缀，当且仅当 $T_{0...|S|-1} = S$，可以看出任意字符串 A 一定是 A 的前缀。

定义字符串 S 是另一字符串 T 的后缀，当且仅当 $T_{|T|-|S|...|T|-1} = S$，可以看出任意字符串 A 一定是 A 的后缀。

两个不相等的字符串 S, T 比较大小时，若 T 是 S 的前缀，则 $S > T$；若 S 是 T 的前缀，则 $S < T$；否则找到第一个位置 p，使得 $S_{0...p-1} = T_{0...p-1}, S_p \neq T_p$，则 S, T 的大小关系与 S_p, T_p 的大小关系相同。

对于两个字符串 A, B，定义 AB 为将 B 直接连接在 A 后得到的字符串，即：$|AB| = |A| + |B|, (AB)_i = \begin{cases} A_i & i < |A| \\ B_{i-|A|} & i \geq |A| \end{cases}$。

根据上一条定义，可以定义出 $S^k = S^{k-1}S, S^1 = S$。

2 后缀数组

后缀数组作为处理字符串问题的有力工具，是理解 Lyndon 分解的重要前置知识，本章节将大致地提一下，详细学习可以参考“参考文献 1”。

2.1 定义

将 S 的每一个后缀排序（它们肯定互不相同），可以得到后缀数组 $sa_{0\dots|S|-1}$，是一个 0 到 $|S|-1$ 的排列，满足 $\forall i \in [1,|S|), S_{sa_{i-1}\dots|S|-1} < S_{sa_i\dots|S|-1}$。

定义 rk 数组，满足 $rk_{sa_i} = i$。

2.2 后缀排序

后缀排序的常用方法为倍增排序。

根据字符串大小比较的定义，若 $S_{0\dots p} < T_{0\dots p}, p \le q$，则 $S_{0\dots q} < T_{0\dots q}$。

于是，可以先仅取所有后缀的前 2^0 个字符进行比较，然后根据仅取前 2^i 个字符排序后结果进行双关键字排序，就可以得到取前 2^{i+1} 个字符排序后的结果。

双关键字排序时使用基数排序**可以做到 $O(n\log n)$ 求后缀数组**，使用 SA-IS 算法或 DC3 算法可以做到 $O(n)$，因为这不是本文重点，故不在此介绍。

3 Lyndon 分解

3.1 定义

对于一个字符串 S，若 S 小于它的所有不为 S 的后缀，则称 S 为简单串（或 Lyndon 串）。

定义 S 的 Lyndon 分解为 $S = w_1w_2w_3\dots w_m$，其中所有 w 均为简单串，且 $w_1 \ge w_2 \ge w_3 \ge \dots \ge w_m$。

Lyndon 分解是唯一的，且对于每个字符串都必定有 Lyndon 分解，下面将证明 Lyndon 分解是唯一的，并通过一个较劣的算法（时间复杂度 $O(n\log n)$）来证明每一个字符串都有 Lyndon 分解。

3.2 简单串的性质

1. 字符串 S 为简单串，当且仅当 S 小于所有与 S 循环同构的串，即 $\forall i \in (0,|S|), S < (SS)_{i\dots i+|S|-1}$。

   S 为简单串时，则 S 小于所有与 S 循环同构的串的证明：

   若 $S = AB$，且 $S \ge BA$，则 $B > AB \ge BA$，不成立，则原命题成立。

   S 小于所有与 S 循环同构的串，则 S 为简单串时的证明：

   若 $S = AB$，且 $S \ge B$，B 取所有符合条件中最短的，则 $BA > AB \ge B$，则 B 是 A 的前缀或 A 是 B 的前缀。

若 B 是 A 的前缀，则令 $A=BC$，则 $S=BCB$，则 $BBC>BCB\geq B$，则 $BC>CB$，又 $CBB>BCB$，矛盾。

若 A 是 B 的前缀，则令 $B=AC$，则 $S=AAC$，则 $ACA>AAC\geq AC$，则 $CA>AC\geq C$，又 B 取所有符合条件中最短的，所以 $S=AAC<C$，所以 $AC\geq C>AAC$，所以 $C>AC$，矛盾。

故原命题得证。

2. 若 Sa 为简单串，其中 a 为一个字符，则 $\forall b\geq a, Sb$ 为简单串，证明：

   对于 S 的每个非 S 后缀 T，有 $Tb>Ta>Sa$，因为 $|Tb|<|Sa|$，所以 Tb,Sa 第一个不同的字符位置不为 $|Sa|-1$，所以将 a 换为 b 不等号保持不变，即 $Tb\geq Sb$。

3.3 唯一性证明

若 $S=w_1w_2w_3\cdots=w'_1w'_2w'_3\ldots$，不妨设 $w_1\neq w'_1$（找到第一个 $w_p\neq w'_p$ 的 p，令 $S'=w_pw_{p+1}\ldots$，可以转化为这种情况），并假设 $|w_1|<|w'_1|$（不然交换 w,w'）。

令 $w'_1=w_1A, w_2=AB$，则 $A>w_1A>w_1\geq AB$，可以推得 $A>AB$，这显然是错误的，所以可以推得：Lyndon 分解是唯一的。

3.4 $O(n\log n)$ 的分解方法

定义数组 a_i 为最小的 j，大于 i 且 $S_{j\ldots|S|-1}<S_{i\ldots|S|-1}$，若不存在这样的 j，可以认为 $a_i=|S|$。

那么，S 的 Lyndon 分解的第一项为 $S_{0\ldots a_0-1}$，且后面 $m-1$ 项就是 $S_{a_0\ldots|S|-1}$ 的 Lyndon 分解。

3.4.1 正确性证明

证明可以分为 $S_{0\ldots a_0-1}$ 是简单串，和 $S_{0\ldots a_0-1}\geq S_{a_0\ldots a_1-1}$ 两部分。

1. 令 $S=ABC$，其中 $S_{0\ldots a_0-1}=AB$，根据算法有 $C<ABC<BC$，假设 $AB\geq BA$，那么 B 一定是 AB 的前缀。

   设 $AB=BD$，则 $C<BDC<BC$，则 $DC<C$。

   因为算法得到 C 是比 S 小的最长的后缀，与 $DC<C$ 矛盾，所以原命题成立。

   所以 $S_{0\ldots a_0-1}$ 是简单串。

2. 令 $S=ABC$，其中 $S_{0\ldots a_0-1}=A, S_{a_0\ldots a_1-1}=B$，根据算法有 $C<BC<ABC$，假设 $A<B$，那么 A 一定是 B 的前缀。

设 $B = AD$，则 $C < ADC < AADC$，则 $DC < ADC$。

因为算法得到 C 是比 ADC 小的最长的后缀，与 $DC < ADC$ 矛盾，所以原命题成立。

所以 $S_{0 \ldots a_0 - 1} \geq S_{a_0 \ldots a_1 - 1}$。

所以，我们证明了这个算法可以求出一个字符串的 Lyndon 分解，同时**每一个字符串都必定有 Lyndon 分解**。

3.4.2 算法伪代码

Algorithm 1 Algorithm based on SA

Input: 字符串 S

Output: S 的 Lyndon 分解

$n \leftarrow |S|$
$sa \leftarrow \text{Sufsort}(S)$
$X \leftarrow \{n\}$
for $i \in [0, n)$ do
    $a_{sa_i} \leftarrow X.\text{upperbound}(sa_i)$
    $X.\text{insert}(sa_i)$
end
$i \leftarrow 0$
while $i < n$ do
    $\text{Print}(S_{i \ldots a_i - 1})$
    $i \leftarrow a_i - 1$
end

3.5 Duval 算法

定义字符串 S 为近似简单的，当且仅当 $S = www \ldots w\overline{w}$，其中 $\overline{w}$ 是 w 的前缀，且 w 是简单串，据此定义，简单串也是近似简单串。

在算法运行过程中，S 分为 3 部分 $s_1 s_2 s_3$，其中 s_1 是已经完成分解的部分，s_2 是近似简单串，s_3 是没有任何处理的部分。

我们可以用三个指针 i, j, k 来保存运行状态：i 为 s_2 的开始字符，j 为 s_3 的开始字符，k 则记录近似简单串 s_2 的情况：$k = j - |w|$。

Duval 算法即不断尝试往 s_2 的末尾加入 s_3 的首字符，根据 S_j 和 S_k 的大小关系，有：

- $S_j = S_k$

显然，将 S_{j+1} 加入 s_2 后，s_2 还是一个近似简单串。

可以令 $j \leftarrow j+1, k \leftarrow k+1$。

- $S_j > S_k$

  令 $T = s_2 S_j = www\ldots\overline{w}S_j = w^a\overline{w}S_j$，因为 $\overline{w}S_j > w$，将 T 循环位移（位移长度在 $(0, |T|)$ 间），则：

  若循环向前移 $b|w|$ 步，则 $T' = w^{a-b}\overline{w}S_j w^b > w^{a-b}ww^{b-1}\overline{w}S_j = T$。

  若循环向前移 u 步，u 不是 $|w|$ 的倍数，且 $u < |T|-|w|$，因为 w 是简单串，则 $(T')_{0,|w|-1} > w$，则 $T' > T$；当 $u > |T|-|w|$ 时，因为 S_2 大于 w 的对应项，所以 $T' > T$。

  综上，若循环位移长度在 $(0, |T|)$ 间，$T' > T$，所以，此时 s_2S_j 是一个简单串，根据定义，可以认为加入 S_j 后，s_2 依然是近似简单串。

  可以令 $j \leftarrow j+1, k \leftarrow i$。

- $S_j < S_k$

  令 $T = s_2S_j = w^a\overline{w}S_j$，因为 $\overline{w}S_j < w$，循环位移 $|w|$ 步后，显然有 $T' < T$，此时 T 不再是简单串。

  因为 $\overline{w}S_j < w$，考虑 $O(n\log n)$ 的算法的过程，比 $w^u\overline{w}S_j$ 小的最长后缀为 $w^{u-1}\overline{w}S_j$，所以前面的 a 个 w 可以加入 Lyndon 分解中。

  修改 i 的值，然后令 $j \rightarrow i+1, k \rightarrow i$（一个单独的字符一定是简单串，所以直接令 s_2 为 s_1 后的第一个字符）。

3.5.1 算法伪代码

Algorithm 2 Duval Algorithm

Input: 字符串 S

Output: S 的 Lyndon 分解

```
n ← |S|
i ← 0
while i < n do
    j ← i + 1
    k ← i
    while j < n do
        if S_j = S_k then
            j ← j + 1
            k ← k + 1
            Continue
        end
        if S_j > S_k then
            j ← j + 1
            k ← i
            Continue
        end
        Break
    end
    while i ≤ k do
        Print(S_{i...i+(j−k)−1})
        i ← i + (j − k)
    end
end
```

3.5.2 复杂度证明

在一次外层循环内，向 s_1 中添加 u 个字符，那么 j 的值一定小于 $i+2u$，内层循环次数不大于 $2u$。

对它求和，内层循环总的次数不大于 $2n$，外层循环次数不大于 n，所以 Duval 算法的时间复杂度为 $O(n)$。

4 例题

4.1 例题 1

4.1.1 题面

给定长为 n 的字符串 S，求出 S 的最小表示法。

4.1.2 题解

将 SS Lyndon 分解，找到分解后最后一个字符串，它的首字符为 S_p，且 $p \in [0, |S|)$，S 的最小表示法就是 $S_{p \ldots p+|S|-1}$。

考虑到 $SS = w_1 w_2 \ldots w_m$，其中开头为 S_p 的为 w_i。

因为若取 w_{i-1} 为开头，当 $w_{i-1} > w_i$ 时，显然取 w_i 为开头较优，当 $w_{i-1} = w_i$ 时，找到第一个 $j > i$，使得 $w_j \neq w_i$，显然 $w_j < w_i$，则 w_j 越靠前越优，故应选 w_i 作为开头。

时间复杂度 $O(n)$。

4.2 例题 2

4.2.1 题面

给定长度为 n 的字符串 S，将 S 分为最多 k 个串 $c_1 c_2 \ldots c_k$，求 $\max c_i$ 的最小值。

4.2.2 题解

考虑 S 的 Lyndon 分解，我们令 $S = w_1^m w_{m+1} \ldots$。

如果 $k > m$，可以划分为 m 个 w_1，和 1 个 $w_{m+1} \ldots$，此时答案是 w_1。

如果 $k|m$，可以划分为 $k-1$ 个 $w_1^{\frac{m}{k}}$，和 1 个 $w_1^{\frac{m}{k}} w_{m+1} \ldots$，此时答案为 $w_1^{\frac{m}{k}} w_{m+1} \ldots$。

否则，可以额外花费一次划分，将最后的一个 $w_1^{\lfloor \frac{m}{k} \rfloor}$，和它后面的 $w_{m+1} \ldots$ 划开，答案为 $w_1^{\lceil \frac{m}{k} \rceil}$。

时间复杂度 $O(n)$。

4.3 例题 3

4.3.1 题面

给定长度为 n 的字符串 S，将 S 分为最多 k 个串 $c_1 c_2 \ldots c_k$，求 $\max c_i$ 的最小值。

q 次询问，每次询问 S 的一个后缀，并重新给定 k。

4.3.2 题解

用 $O(n \log n)$ 的 Lyndon 分解算法可以求出每个后缀的 Lyndon 分解的 w_1，并记录每个位置开始时，Lyndon 分解出的 w_1 的重复次数。

知道这两个信息，就可以和上题用相同的方法做出来了。

时间复杂度 $O(n \log n)$。

4.4 例题 4

4.4.1 题面

给定一个字符串 S，求出 S 的每个前缀的最小后缀。

4.4.2 题解

将 S Lyndon 分解，令 $S = w_1 w_2 w_3 \dots w_m$，显然，前缀 $w_1 w_2 \dots w_k$ 的最小后缀为 w_k。

但是如果前缀是 $w_1 w_2 \dots \overline{w_k}$（$\overline{w}_k$ 指 w_k 的前缀），答案就不一定是 $\overline{w_k}$，例如 aab 的 Lyndon 分解为 aab，前缀 aa 的最小后缀为 a 而不是 aa。

Duval 算法的运行过程，我们考虑怎么在处理 s_2 时求出每个前缀的最小后缀。

- 当 $s_2 = w$ 时，最小后缀显然为 w。
- 当 $s_2 \neq w$ 时，s_2 为 w 重复若干次（最后一次不一定完整）得到。

  如果起始字符不在 $\overline{w}$ 内，那么它比 w 大，一定不优。

  在 $\overline{w}$ 内的情况，与去掉末尾 $|w|$ 个字符的情况相同，答案长度一样。

5 总结

Lyndon 分解可以作为处理字符串最优化的有力工具。

在上文中，简单介绍了求 Lyndon 分解的两种算法，并证明了它们的正确性。

其中 $O(n \log n)$ 的算法思路简单，且能够求出每一个后缀的 Lyndon 分解，在例题 23 中有所说明。

其中 $O(n)$ 的算法复杂度优，代码简单，能够求出每一个前缀的 Lyndon 分解，在例题 4 中有所说明。

参考文献

[1] 许智磊，《后缀数组》，IOI2004 国家集训队论文。

[2] cp-algorithms

感谢

[1] 雅礼中学的同学老师们为我提供的帮助。

[2] CCF 给予的本次撰写论文的机会。