arXiv:2302.00724v1 [cs.DS] 1 Feb 2023

字符串中的保序平方

Pawe-l Gawrychowski[1],Samah Ghazawi*[2]，以及 Gad M. Landau*[2,3]

[1]波兰弗罗茨瓦夫大学计算机科学研究所,[2]以色列海法大学计算机科学系[3]美国纽约州布鲁克林，纽约大学纽约大学坦顿工程学院

摘要

字符串中的保序平方是指形如 uv 的片段，其中 $u \neq v$ 且 u 与 v序同构。我们证明，字母表大小为 σ 、长度为 n 的字符串 w 包含$\mathcal{O}(\sigma n)$ 个作为单词彼此不同的保序平方。这改进了 Kociumaka、Radoszewski、Rytter 和 Wale´n 在 [理论计算机科学 2016]中给出的 $\mathcal{O}(\sigma^2 n)$ 上界。此外，对于任意 σ和 n ，我们构造出一个含有 $\Omega(\sigma n)$ 个作为单词彼此不同的保序平方的字符串，从而证明我们的上界在渐近意义下是紧的。最后，我们设计了一种运行时间为 $\mathcal{O}(\sigma n)$ 的算法，可输出给定字符串中所有作为单词彼此不同的保序平方。根据我们的下界，该算法在最坏情况下是最优的。

1 引言

字符串中重复的一种自然定义是平方，即形如uu的片段，其中 u 是一个字符串。对字符串中重复的研究至少可以追溯到 Thue 在 1906 [28], 的工作；他构造了一个定义在三元字母表上的无限无平方词。此后，人们提出并研究了多种重复的定义，核心问题是分析长度为 n 的字符串中可以包含多少个这样的重复。当然，字符串 a^n 的任何偶数长度片段都是一个平方，因此我们希望统计不同平方的数量。利用 Crochemore 和 Rytter 的一个组合结果 [5], ，Fraenkel 和 Simpson [10]证明，长度为 n 的字符串至多包含 $2n$ 个不同平方（另见 Ilie 给出的一个更简洁的证明 [17]）。他们还给出了一个无限族的字符串，其长度为 n ，且包含 $n - o(n)$ 个不同平方。多年来，人们一直猜想真正的上界实际上是 n。有趣的是，若能证明该猜想对二元字母表成立，就能推出它对任意字母表也成立 [24]。最近，在上界 [7, 18, 23, 27], 历经一系列改进后，Brlek 和 Li [1], 最终解决了这一猜想，证明上界为 $n - \sigma + 1$，其中 σ 是字母表的大小。

对于许多应用而言，采用不同的相等定义似乎更为合适，这也使我们得以定义不同类型的平方。以下是三个有趣的例子：**(1)** 阿贝尔平方 [6, 8, 9, 16, 19–21, 26]（也称乱序平方），它们在自然语言处理应用中以及其他分类结果很大程度上取决于特征集合分布（而非特征序列分布）的领域中都很有用。**(2)** 参数化平方 [20]，它们用于查找相同代码片段的应用。**(3)** 保序

*部分经费由以色列科学基金会项目 1475/18 及美以双边科学基金会（BSF）项目编号 2018141 资助。

平方 $[4, 13, 20]$，它们对于股票价格分析和音乐旋律匹配等应用十分重要。

Kociumaka 等人 [20] 研究了三类平方的组合性质。给定一个长度为 n 、字母表大小为 σ 的字符串，作者首先将不同的阿贝尔平方的数量上界确定为 $\Theta(n^2)$。其次，将不同的参数化平方的数量上界确定为 $\mathcal{O}(2(\sigma!)^2 n)$，并将非等价参数化方块（定义见文中）的数量上界确定为 $\mathcal{O}(2\sigma! n)$。第三，作者给出了不同的保序平方数量的 $\mathcal{O}(\sigma^2 n)$ 上界。

从算法视角来看，研究者提出了多种算法，用于计算长度为 n 的字符串中的阿贝尔平方和保序平方。Cummings 和 Smyth [6] 提出了一种时间复杂度为 $\Theta(n^2)$ 的算法，用于计算所有由两个或更多阿贝尔等价子串拼接而成的子串。Kociumaka 等人 [21] 提出了一种算法，使用线性空间，以 $\mathcal{O}(n^2/\log^2 n)$ 的时间复杂度计算所有阿贝尔平方中的最长者、最短者及其数量。Gourdel 等人 [13] 证明，所有不可移位保序方块（定义见文中）均可在 $\mathcal{O}(n \log n)$ 时间内计算出来。此外，Crochemore 等人 [4] 提出了不完整的保序后缀树（详见文中），记为 T，它支持在与模式长度成正比的时间内执行保序模式匹配查询。T 的构建期望时间为 $\mathcal{O}(n \log\log n)$，最坏情况时间为 $\mathcal{O}(n \log^2 \log n / \log\log\log n)$。此外，作者证明，利用 T，可以在 $\mathcal{O}(n \log n + occ)$ 时间内计算出所有保序平方的出现位置，其中 occ 为保序平方出现位置的总数。请注意，保序平方的所有出现位置数量可能会大得不合理。特别是，每个普通平方都被视为保序平方，因此，一个长度为 n 的字符串包含 $\Theta(n^2)$ 个保序平方出现位置。因此，更自然的做法是只生成作为单词彼此不同的保序平方。

我们的结果。本文重点研究保序平方。在有序字母表上，等长字符串 u和 v 称为序同构，记作 $u \approx v$，当且仅当它们在对应位置上的字符之间的大小关系在 u 和 v中相同。例如，假设 $a < b < c < d < z$，则 $u = acb$ 和 $v = azd$序同构。本文所说的保序平方是形如 uv的字符串，其中 $u \approx v$，并且 $u \neq v$。

本文的主要结果是：长度为n 、字母表为 σ 的字符串中，保序平方的数量为 $\mathcal{O}(\sigma n)$。这改进了 Kociumaka 等人 [20]给出的界 $\mathcal{O}(\sigma^2 n)$。我们强调，在保序平方的定义中，要求 $u \neq v$；而 Kociumaka 等人 [20] 统计的是形如 uv的片段，其中 $u \approx v$，且这些片段作为单词彼此不同。我们认为，在本文的语境下，我们的定义更为自然。同时，根据 Brlek 和 Li [1] 的结果，长度为 n 的字符串中，形如 uu 且作为单词彼此不同的片段少于 n 个；因此，我们的结果也意味着，满足 $u \approx v$ 且作为单词彼此不同的片段 uv 的数量为 $\mathcal{O}(\sigma n)$。我们通过构造来补充这一上界：对于每个 σ，都存在一个无限族的字符串，其长度为 n，字母表大小为 σ，并包含 $\Theta(\sigma n)$个这样的片段。我们先在第 3 节介绍下界，然后在第 4 节给出上界。

定理 1.1。长度为 n 、取自大小为 σ 的字母表的字符串中，保序平方的数量为 $\mathcal{O}(\sigma n)$；即使只考虑作为单词彼此不同的保序平方，这一界限在渐近意义下仍是紧的。

接下来，我们设计一种算法，用于在 $\mathcal{O}(\sigma n)$ 时间内报告给定字符串中所有的保序平方。该字符串长度为 n，其字母表为 σ。根据我们的下界，该算法在最坏情况下是渐近最优的。我们再次强调，在保序平方的定义中，我们要求

$u \neq v$。然而，Gusfield 和 Stoye 的算法 [14], 可以在 $\mathcal{O}(n)$ 时间内报告所有形如 uu 且作为单词彼此不同的片段。因此，对于 $\sigma = o(\log n)$，这回答了 Crochemore 等人 [4], 提出的一个开放问题：是否存在一种 $o(n \log n)$ 时间算法来寻找最长的保序平方。相关内容见第 5 节。

定理 1.2。 在字母表大小为 σ 、长度为 n 的字符串中，所有保序平方均可在 $\mathcal{O}(\sigma n)$ 时间内找到。

我们技术的高层次描述。 对于下界，首先考虑递增字符串 $w = 123 \ldots n$，其中 $\sigma = n$。显然，任何偶数长度的片段都是一个保序平方，因此在长度为 n的字符串中，保序平方的数量达到最大值，即 $\Omega(n^2) = \Omega(\sigma n)$。为了减小字母表 σ的大小，我们将 w 替换为一个非递减字符串 $w = 11 \ldots 122 \ldots 2 \ldots \sigma\sigma \ldots \sigma$，其中每个字符重复相同的次数。我们展示了 w 中有 $\Omega(\sigma n)$ 个作为单词彼此不同的保序平方。更多细节见第 3 节。

对于上界，我们借鉴了 Kociumaka 等人 [20], 的见解，其高层策略是分别考虑 w 的每个后缀。对于每个后缀，他们考虑由字母表中每个字符的首次出现组成的最左出现位置集合。因此，每个后缀中最多有 σ 个最左出现位置。对于一个固定后缀，他们将其所有前缀都视为可能的保序平方 uv。接下来，他们证明，由于 $u \neq v$， uv 由一对最左出现位置确定，其中一个出现位置属于 u，另一个则在相同的相对位置属于 v 。例如，设 $acbadbxz$ 为该后缀，那么最左出现位置 2 和 5 这一对定义了保序平方 $acbadb$。因此，他们将作为所考虑后缀前缀的保序平方数量上界定为$\binom{\sigma}{2}$, 所以总计为 $\binom{\sigma}{2} n$ 。

我们还分别给出了输入字符串 w 的某个后缀的前缀中保序平方数量的上界。然而，我们的目标是证明这类前缀只有 $\mathcal{O}(\sigma)$ 个，因此总共只有 $\mathcal{O}(\sigma n)$ 个。为此，我们首先将保序平方划分为若干组。令O_k 由所有满足 $2^k \leq |uv| < 2^{k+1}$ 的保序平方 uv 组成。类似地，我们将最左出现位置划分为若干组。令 L_k 由所有满足 $2^k \leq i < 2^{k+1}$的最左出现位置 i 组成。现在，我们的策略是证明，如果 $|O_k|$ 大于某个固定常数，那么$|O_k| = \mathcal{O}(|L_{k-2}|)$。论证结构如下。我们首先注意到，两个保序平方 uv 和 $u'v'$ 意味着 $|u| - \Delta$是 u 的一种所谓的保序边界，其中 $\Delta = |u'| - |u|$。我们记作 $u = b_1 b_2 \ldots b_f b_{f+1}$，其中 $|b_1| = |b_2| = \ldots |b_f| = \Delta$ 和 $|b_{f+1}| < \Delta$；通过仔细选择来自 O_k 的 uv 和 $u'v'$ ，可以得出 b_2 包含一个最左出现位置，且 f 与 $|O_k|$成正比。接着，我们论证， b_2 包含一个最左出现位置，实际上意味着每个 b_j 都包含一个最左出现位置，因此 $|O_k| = \mathcal{O}(|L_{k-2}|)$。对所有 k 求和，并分别考虑所有满足 $|O_k|$ 小于该固定常数的 k ，我们便能得出 $\sum_k |O_k| = \sum_k \mathcal{O}(|L_k|) < \mathcal{O}(\sigma)$。更多细节请参见第 4 节。

为了获得一种高效报告所有保序平方的算法，我们采用 Crochemore 等人 [4]定义的保序后缀树。借助这一结构，只需检查两个叶节点的最近公共祖先的字符串深度是否至少为 ℓ，即可判断 $w[i..i + 2\ell - 1]$ 是否为保序平方。首先，我们需要说明如何在 $\mathcal{O}(\sigma n)$ 时间内构建保序树。其次，我们将上述推理扩展为高效生成仅有的 $\mathcal{O}(\sigma n)$ 个片段，然后以常数时间复杂度逐一检验它们是否为保序平方。尽管用于界定保序平方数量的基本论证与此本质相同，但为了实现高效的算法，需要采用不同的执行方式。更多详情请参见第 5 节。

2 预备知识

设 $\Sigma = \{1, \ldots, \sigma\}$ 是大小为 σ 的固定有限字母表。设 $|s|$ 表示字符串 s 的长度。对于字符串 s, $s[i]$ 表示 s 在位置 i 的字符，而 $s[i..j]$ 表示 s 中从位置 i 开始、到位置 j 结束的片段。当且仅当 $|u| = |v|$ 且对每个 i, j，都有 $u[i] \leq u[j]$ 当且仅当 $v[i] \leq v[j]$ 时，我们称字符串 u 和 v 序同构，记作 $u \approx v$。两个字符串 u 和 v 的拼接记作 uv。形如 uv 的字符串在满足 $u \neq v$ 且 $u \approx v$ 时称为保序平方，或 op-平方。我们称 u 为其左臂，称 v 为其右臂。需要强调的是，普通平方，即形如 xx 的字符串，并不是 op-平方。当且仅当 $uv \neq u'v'$ 时，两个 op-平方 uv 和 $u'v'$作为单词彼此不同。

字典树是一种有根树，每条边都标记一个字符，且从同一节点出发的边具有不同的标记。字典树的节点 u 表示从根节点到 u的路径上各边标记依次组成的字符串。压缩字典树由字典树经过如下操作得到：将由恰有一个子节点的节点组成的极大路径替换为一条边，该边标记为路径上各边标记的拼接。字符串 w 的后缀树 ST 是一种压缩字典树，其叶节点对应于 w\$ 的各个后缀。节点 u 在 ST 中的字符串深度，是它所对应字符串的长度。 ST 的显式节点就是 ST的节点。 ST 的隐式节点是对应于 ST的未压缩字典树中的节点，换句话说，也就是 ST的一条边上的位置。

接下来，我们需要一些专用于序同构的定义。沿用 Kubica 等人 [22],的说法，当 $s[1..b] \approx s[n-b+1..n]$时，我们称 $s[1..n]$ 的字符串 b 具有 op-边界。沿用 Gourdel 等人 [13]（以及 Matsuoka 等人 [25]）的说法，当 $s = b_1 b_2 \ldots b_f b_{f+1}$且满足 $|b_1| = |b_2| = \ldots = |b_f| = p$和 $|b_{f+1}| < p$（因此 $f = \lfloor n/p \rfloor$）、 $b_1 \approx b_2 \approx \ldots \approx b_f$以及 $b_1[1..|b_{f+1}|] \approx b_2[1..|b_{f+1}|] \approx \ldots \approx b_f[1..|b_{f+1}|] \approx b_{f+1}$时，我们称 $s[1..n]$的 p具有初始 op-周期。 b_1, b_2, ..., b_f称为由 p 在 s中定义的块，而 b_{f+1} （可以为空）称为不完整块。在经典情形下，p 是 $s[1..n]$ 的周期，当且仅当 $n-b$ 是 $s[1..n]$的边界；而在保序情形下，我们只有一个方向的蕴含关系。

命题 2.1. 如果 b 是 $s[1..n]$ 的 *op*-边界，那么 $n-b$ 是 $s[1..n]$的初始 *op*-周期。

证明。设 $p = n-b$和 $f = \lfloor n/p \rfloor$。我们将 $s[1..n]$表示为 $s = b_1 b_2 \ldots b_f b_{f+1}$，其中 $|b_1| = |b_2| = \ldots = |b_f| = p$和 $|b_{f+1}| < p$。由于 b是 $s[1..n]$的 op-边界，因此有 $s[1..b] \approx s[n-b+1..n]$，所以 $s[1..n-p] \approx s[p+1..n]$。我们注意到 $s[1..n-p] = b_1 b_2 \ldots b_{f-1} b_f[1..|b_{f+1}|]$和 $s[p+1..n] = b_2 b_3 \ldots b_f b_{f+1}$。于是， $b_1 b_2 \ldots b_{f-1} b_f[1..|b_{f+1}|] \approx b_2 b_3 \ldots b_f b_{f+1}$推出 $b_i \approx b_{i+1}$，对每个 $i = 1, 2, \ldots\ldots, f-1$均成立；而 $b_i[1..|b_{f+1}|] \approx b_{i+1}[1..|b_{f+1}|]$对每个 $i = 1, 2, \ldots\ldots, f$均成立。因此，$b_1 \approx b_2 \approx b_3 \approx \ldots \approx b_{f-1} \approx b_f$且 $b_1[1..|b_{f+1}|] \approx b_2[1..|b_{f+1}|] \approx \ldots \approx b_f[1..|b_{f+1}|] \approx b_{f+1}$，所以 $p = n-b$ 确实是 $s[1..n]$的初始 op-周期。 □

根据命题 2.1， b是 $s[1..n]$ 的 op-边界意味着 $s[1..n] = b_1 b_2 \ldots b_f b_{f+1}$，其中 $b_1 b_2 \ldots b_f[1..|b_{f+1}|] \approx b_2 b_3 \ldots b_f b_{f+1}$、 $|b_1| = |b_2| = \ldots = |b_f| = n-b$和 $|b_{f+1}| < p$（因此$f = \lfloor n/(n-b) \rfloor$， $b_1 \approx b_2 \approx \ldots \approx b_f$和 $b_1[1..|b_{f+1}|] \approx b_2[1..|b_{f+1}|] \approx \ldots \approx b_f[1..|b_{f+1}|] \approx b_{f+1}$。我们称这些块由 b定义。

3 下界

回顾一下， $\Sigma = \{1, \ldots, \sigma\}$。我们定义字符串 $w = 11\ldots122\ldots2\ldots\sigma\sigma\ldots\sigma$，即由 σ 个块拼接而成的字符串，每个块都由同一字符重复 k 次组成。注意，$|w| = \sigma k$。对于 $i = 1, 2, \ldots, \lfloor \sigma/2 \rfloor$，我们考虑长度为 $2ik$、起始位置为 $j = 1, 2, \ldots, |s| - 2ik + 1$ 的 w的所有片段。对于 $j = 1$模 k，该片段由 $2i$ 个块拼接而成，

每个块都由同一字符重复 k 次组成。对于 $j \neq 1$ 模 k，该片段以同一字符的 $r \in [1, k-1]$ 次重复开头，随后是 $2i-1$ 个块，每个块都由同一字符重复k 次组成，最后是同一字符的 $\ell = k - r$ 次重复。见图1。



图1：红色方框对应一个长度为 10、包含 3 种不同字符的 op-平方。蓝色方框对应一个长度为 20、包含 5 种不同字符的 op-平方。

每个这样的片段都是一个 op-平方。对于 $j = 1 \bmod k$，左臂和右臂都由i 块组成，每块包含字符 c，$c+1$、.....、$c+i-1$ 的 k 次重复。对于 $j \neq 1 \bmod k$，左臂和右臂都先由字符 c 的 r 次重复组成，接着由 k块组成，每块包含字符 $c+1$，$c+2$、.....、$c+i-2$ 的 $i-1$ 次重复，最后再由字符 $c+i-1$ 的 ℓ 次重复组成。因此，左臂和右臂总是序同构的。此外，对于 i 的每种取值和每个起始位置，我们都会得到一个不同的单词，因为两个长度相同的此类片段要么以不同字符开头，要么相同字符的第一个块长度不同。

现在，我们分析 w 中此类 op-平方的数量。考虑每个 $1 \le i \le \lfloor \sigma/2 \rfloor$ 以及起始位置 1，2，....., $|s| - 2ik + 1$，可得 w 中的 op-平方数量至少为：

$$\sum_{i=1}^{\lfloor \sigma/2 \rfloor} (|s| - 2ik + 1) = \sum_{i=1}^{\lfloor \sigma/2 \rfloor} (\sigma k - 2ik + 1) = \lfloor \sigma/2 \rfloor \cdot (\sigma k - k(\lfloor \sigma/2 \rfloor + 1) + 1)$$
$$\ge \lfloor \sigma/2 \rfloor \cdot (k(\lceil \sigma/2 \rceil - 1) + 1).$$

对于 $\sigma \ge 3$，对于任意 k，此数量至少为 $\sigma^2 k/12 = \sigma n/12$ 。对于 $\sigma = 1$，2，我们还假设 $k \ge 2$，并统计形式为 1^{2i} 的 op-平方，共有 $\lfloor k/2 \rfloor \ge n/6 \ge \sigma n/12$ 个。因此，在这两种情况下，对于每个 $k \ge 2$，我们都能得到一个长度为 $n = k\sigma$ 、由 Σ 构成的字符串，其中包含 $\sigma n/12$ 个作为单词彼此不同的 op-平方。

定理3.1。对于任意字母表 $\Sigma = \{1, 2, \ldots, \sigma\}$，都存在一个无限族字符串，其长度为 $n = k\sigma$，由 Σ 构成，并包含 $\Omega(\sigma n)$ 个作为单词彼此不同的 *op*-平方。

4 上界

本节的目标是给定一个长度为n、定义在字母表 $\Sigma = \{1, \ldots, \sigma\}$上的字符串 w，为其中 op-平方的数量给出上界。回忆一下，当 $u \neq v$且 $u \approx v$时，uv是一个 op-平方。我们将证明这个数量为 $\mathcal{O}(\sigma n)$。正如引言中所述，根据 Brlek 和 Li 的结果 [1], , 正规平方的数量，也就是形如 uu且作为单词彼此不同的片段数量，小于 n。因此，我们的结果实际上也给出了形如 uv且满足 $u \approx v$、作为单词彼此不同的片段数量的上界，即 $\mathcal{O}(\sigma n)$。

我们分别考虑 w 的每个后缀。对于每个后缀 $w[i..n]$，我们将 op-平方且为 $w[i..n]$ 前缀的数量上界限定为 $\mathcal{O}(\sigma)$。因此，为避免本节余下部分的记号过于繁琐，我们假设有一个长度为 m、定义在字母表 $\Sigma = \{1, \ldots, \sigma\}$ 上的字符串 s，并希望将作为 s 前缀的 op-平方 uv 的数量上界限定为 $\mathcal{O}(\sigma)$。见图2。

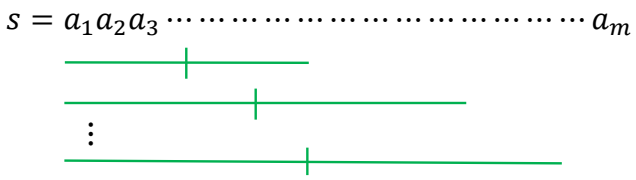

图2： s 的绿色前缀是 op-平方。

Kociumaka 等人 [20] 观察到，每个作为 s 前缀的 op-平方 uv 都可以按如下方式得到（回想一下，在我们的定义中 $u \neq v$）。当 $s[j] \neq s[i]$ 对每个 $i < j$ 都成立时，我们称 i 为最左出现位置。于是，存在 i 和 j，使得 i 和 j 都是最左出现位置，i 属于 u，j 属于 v，并且还满足 $|u| = j - i$。更形式化地说：

命题 4.1 ([20, 引理 4.2 和推论 4.3])。 我们可以按如下方式构造一个单射函数 g，将作为 s 前缀的 *op*-平方映射到字母表的二元素子集。我们选择属于 v 的最小 i，使得 $s[i] = a$ 不出现在 u中，并令 $s[i - |u|] = b$ 为其在 u中的对应项，然后设 $g(uv) = \{a, b\}$。 i和 $i - |u|$ 都是最左出现位置。

我们将所有作为 s 前缀的 op-平方划分为若干组。令 O_k 表示作为 s 前缀、长度至少为 2^k 且至多为 $2^{k+1} - 1$的 op-平方所构成的组：

定义 4.2. $O_k = \{uv \,|\, u \neq v$和 $u \approx v$和 $2^k \leq |uv| < 2^{k+1}\}$ 对于 $0 \leq k \leq$对数 m.

换言之，我们将 s 划分为长度呈指数递增的连续区间，使得第 k 个区间长度为 2^{k-1}，在 s 中从位置 2^k 开始，到位置 $2^{k+1} - 1$ 结束（其中$0 \leq k \leq \log m$，且当 $m < 2^{k+1} - 1$时，最后一个区间可能不完整）。于是， O_k由以第 k 个区间结束的 op-平方组成。见图 3。

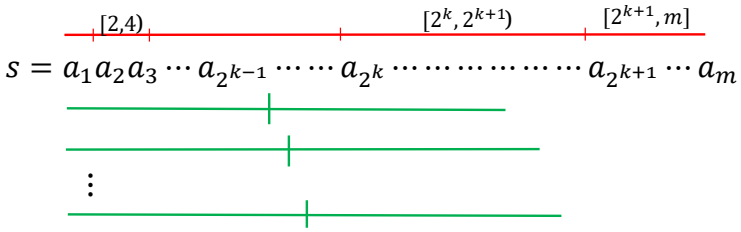

图 3： s 的绿色前缀是以第 k 个区间结束的 op-平方。红线表示这些区间。

作为 s 前缀的 op-平方 uv 的数量为 $\sum_{k=0}^{\log m} |O_k|$。为了给总和提供上界，我们将分别为每组的大小提供上界。首先需要一些命题。

命题 4.3。 对于任意满足 $|u| < |u'|$ 和 $\Delta = |u'| - |u|$的 $\{uv, u'v'\} \in O_k$， $|u| - \Delta$ 是 u 和 v 的 *op*-边界。

证明。 因为 $u \approx v$，只需证明 Δ 是 u的 op-边界。根据 uv 和 $u'v'$ 都是 op-平方这一假设，我们有：

$$u[1..|u| - \Delta] = u'[1..|u| - \Delta] \approx v'[1..|u| - \Delta] = v[\Delta + 1..|u|] \approx u[\Delta + 1..|u|].$$

参见图4。 □

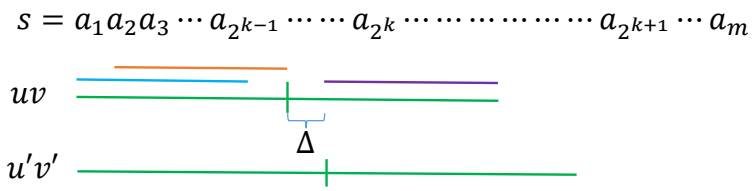

图4：绿色线条对应于 uv 和 $u'v'$。蓝色线条对应于 $u[1..|u|-\Delta]$。橙色线条对应于 $u[\Delta+1..|u|]$。紫色线条对应于 $v[\Delta+1..|u|]$。

在本节余下部分中，我们将经常考虑满足 $|u|<|u'|$ 且 $\Delta=|u'|-|u|$ 的 $\{uv,u'v'\}\in O_k$。那么，根据命题 4.3，Δ 是 u 的 op-边界，因此根据命题 2.1，u 可以表示为 $f=\lfloor|u|/\Delta\rfloor$ 个块 b_1，b_2，……，b_f 以及一个不完整块 b_{f+1} 的拼接，其中 $|b_1|=|b_2|=\ldots=|b_f|=\Delta$ 且 $|b_{f+1}|<\Delta$，满足 $b_1b_2\ldots b_f[1..|b_{f+1}|]\approx b_2b_3\ldots b_fb_{f+1}$、$b_1\approx b_2\approx\ldots\approx b_f$ 和 $b_1[1..|b_{f+1}|]\approx b_2[1..|b_{f+1}|]\approx\ldots\approx b_f[1..|b_{f+1}|]\approx b_{f+1}$。参见图5。为简洁起见，在本文余下部分，我们将用“$\{uv,u'v'\}\in O_k$ 定义块 b_1，b_2，……，b_f，b_{f+1}”来描述这种情况。

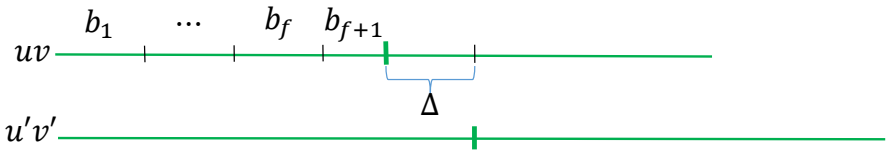

图5：u 中由 $\{uv,u'v'\}$ 定义的块。

命题 4.4。若 $|O_k|\geq 3$，则存在 $\{uv,u'v',u''v''\}\in O_k$，使得 $0<|u'|-|u|,|u''|-|u'|<2^k/(|O_k|-2)$。

证明。O_k 中每个 op-平方的长度都属于 $[2^k,2^{k+1})$，因此其左臂的长度落在 $[2^{k-1},2^k)$ 内。令 $O_k=u_1v_1,u_2,v_2,\ldots,u_\ell v_\ell$，其中 $|u_1|<|u_2|<\ldots<|u_\ell|$。那么，对于某个 $i\in\{1,2,\ldots,\lfloor(\ell-1)/2\rfloor\}$，必有 $|u_{2i+1}|<|u_{2i-1}|+2^{k-1}/\lfloor(\ell-1)/2\rfloor$（否则就会有 $u_\ell\geq u_1+2^{k-1}$）。所求的 op-平方为 $u_{2i-1}v_{2i-1},u_{2i}v_{2i},u_{2i+1}v_{2i+1}$，因为：

$$\begin{aligned}|u_{2i}|-|u_{2i-1}|,|u_{2i+1}|-|u_{2i}|&<|u_{2i+1}|-|u_{2i-1}|<2^{k-1}/\lfloor(\ell-1)/2\rfloor\\&\leq 2^{k-1}/(\ell/2-1)=2^k/(|O_k|-2).\end{aligned}$$

证明至此完成。□

有了所有命题之后，我们现在可以开始讨论技术引理了。我们的目标是用最左出现位置的数量给出 $\sum_k|O_k|$ 的上界。为此，我们需要证明：如果某个 O_k 很大，那么在某个范围内就有许多最左出现位置。具体做法是将以下推理应用于通过应用命题 4.4 选出的三个 op-平方。

引理 4.5. 若 $|O_k|\geq 3$，则对于任意满足 $|u|<|u'|<|u''|$ 的 $\{uv,u'v',u''v''\}\in O_k$，其中 $\{uv,u'v'\}$ 定义了 $b_1,\ldots,b_f,b_{f+1}$，且 $\{u'v',u''v''\}$ 定义了 $b'_1,\ldots,b'_f,b'_{f+1}$，在 b_j 中存在一个最左出现位置，使得 $j\neq 1$；或者在 $b'_{j'}$ 中存在一个最左出现位置，使得 $j'\neq 1$。

证明。令 $\Delta = |u'| - |u|$ 为每个块 b_j 的长度，$\Delta' = |u''| - |u'|$ 为每个块 b'_j 的长度。根据命题 4.1，我们知道必定存在一个最左出现位置 i，它位于 u' 内；与之对应的最左出现位置 $i + |u'|$ 位于 v' 内。若 $s[i]$ 属于一个块 b'_j，且 $j \neq 1$，则结论成立。因此，我们假设 i 属于 b'_1。我们断言，最左出现位置 $i + |u'|$ 位于 v 内。为验证这一点，我们计算：

$$i + |u'| \leq \Delta' + |u'| = |u''| - |u'| + |u'| = |u''| < 2^k \leq |uv|.$$

我们已经证明，$i + |u'|$ 是最左出现位置，并且位于 v内。因此，对于每个 $i' \in [1, i + |u'|)$，都有 $s[i'] \neq s[i + |u'|]$。由于 $u \approx v$，这又意味着，对于每个$i' \in [|u| + 1, i + |u'|)$，都有 $s[i' - |u|] \neq s[i + |u'| - |u|]$。因此，$i + |u'| - |u|$ 也是最左出现位置。我们声称，$s[i + |u'| - |u|]$不可能属于 b_1。为验证这一点，我们计算：

$$i + |u'| - |u| \geq 1 + |u'| - |u| = 1 + \Delta.$$

因此，我们找到了一个最左出现位置 $i + |u'| - |u|$，它位于 u 内，并属于一个满足 $j \neq 1$的块b_j。参见图 6。 □

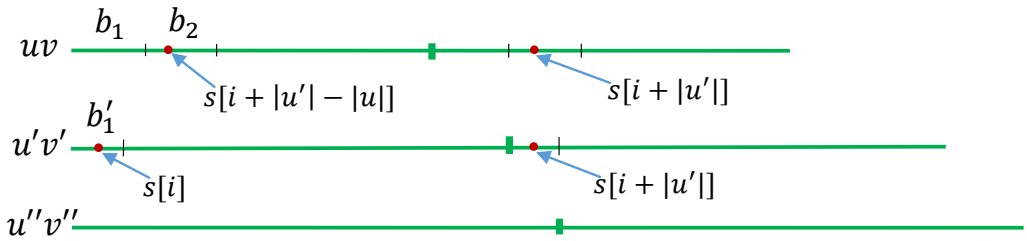

图 6：红点对应于引理 4.5 的证明中所考虑的最左出现位置。

接下来，我们证明：如果 $\{uv, u'v'\} \in O_k$定义了块 b_1、b_2、……、b_f，b_{f+1}，且某个 $j \neq 1$ 对应的块 b_j 中存在最左出现位置，那么实际上每个块 b_j中都存在最左出现位置。证明分两步进行。

引理 4.6. 设 b 是 $u = s[1..|u|]$ 的一个 *op*-边界，它定义了块 b_1、b_2、……、b_f、b_{f+1}，并假设对于某个 $j \in [1, f + 1]$，块 b_j 中存在一个最左出现位置。那么每个块 b_1、b_2、……、b_j 中都存在最左出现位置。

证明。设 $\Delta = |b_1| = |b_2| = \ldots = |b_f|$ 和 $|b_{f+1}| < \Delta$。根据归纳法，只需证明：如果某个 $j \geq 2$ 对应的 b_j 中存在最左出现位置，那么 b_{j-1}中也存在最左出现位置。设 i 是属于 b_j的一个最左出现位置。那么对于每个 $i' \in [1, i$，都有 $u[i'] \neq u[i]$ ）。由于$b_1 b_2 \ldots b_{f-1} b_f[1..|b_{f+1}|] \approx b_2 b_3 \ldots b_f b_{f+1}$，这意味着对于每个 $i' \in [\Delta + 1, i$，都有 $u[i' - \Delta] \neq u[i - \Delta]$）。但这样一来，$i - \Delta$ 也是一个最左出现位置，并且它属于 b_{j-1}，符合要求。见图 7。 □

引理 4.7. 设 b 是 $u = s[1..|u|]$ 的一个 *op*-边界，它定义了块 b_1、b_2、……、b_f、b_{f+1}，并假设块 b_2 中存在最左字符。那么，每个块 b_1、b_2、……、b_f 中都存在最左字符。

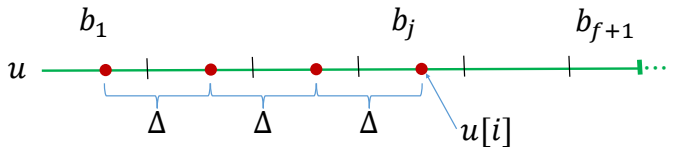

图 7：每个红点都对应于每个块 b_1、 b_2、……、 b_j中相对位置相同的一个最左字符。

证明。 设 $\Delta = |b_1| = |b_2| = \ldots |b_f|$ 和 $|b_{f+1}| < \Delta$。根据假设，块 b_2 中存在最左字符$s[i]$，也就是说， $i \in [\Delta + 1, 2\Delta]$。我们的目标是证明每个块 b_1， b_2、……、 b_f 中都存在最左字符。

由于 $b_1 b_2 \ldots b_f[1..|b_{f+1}|] \approx b_2 b_3 \ldots b_f b_{f+1}$，每个位置 $x \in [1..\Delta]$ 恰好满足以下情形之一：

1. 对所有整数 $p \in [0, f)$， $s[x + p \cdot \Delta]$ 都相同；2. 对所有整数 $p \in [0, f-1)$， $s[x + p \cdot \Delta] < s[x + (p+1) \cdot \Delta]$；3. 对所有整数 $p \in [0, f-1)$， $s[x + p \cdot \Delta] > s[x + (p+1) \cdot \Delta]$。

注意，由于 $s[i]$ 与 $s[1]$、 $s[2]$、……、 $s[i-1]$不同， $i_0 = i - \Delta$ 满足 (2) 或 (3)，因此特别有 $s[i - \Delta] \neq s[i]$。将字母表的顺序反转后，只需证明在 i_0 满足 (2) 的假设下引理成立。此外，我们按如下方式在 b_1 中选取若干位置 i_1、 i_2、……、 i_ℓ。令 C 为出现在 b_1 中的字符集合。选择 $i_1 \in [1, \Delta]$，使得 $s[i_1]$ 是 C 中 $s[i_0]$ 的严格后继；随后选择 $i_2 \in [1, \Delta]$，使得 $s[i_2]$ 是C 中 $s[i_1]$ 的严格后继，依此类推。如果下一个 $i_j \in [1, \Delta]$ 有多个选项，则取最小的那个。当以下两种情形之一成立时停止：

(a) $s[i_{\ell+1}]$ 未定义，即 $s[i_\ell]$ 是 b_1中的最大字符。

(b) $i_{\ell+1}$ 满足（1）或（3）。

注意，根据定义， i_0、 i_1、……、 i_ℓ都满足（2）。此外， $s[i_0]$是最左字符，因为$s[i]$ 是最左字符，所以对于每个 $i' \in [1, i]$， $s[i'] \neq s[i]$；同样，对于每个 $i' \in [\Delta + 1, i]$， $b_1 \approx b_2$，因此 $s[i' - \Delta] \neq s[i - \Delta]$。接下来， $s[i_1]$、……、 $s[i_\ell]$都是最左字符，因为我们始终选择满足 $s[i_j]$等于某个特定字符的最小 i_j，其中 $j = 1$、2、……、 ℓ。

我们总结一下目前的情况。对于每个整数 $p \in [0, f)$， $s[i_\ell + p \cdot \Delta]$ 都属于块b_{p+1}，我们想证明它是一个最左字符。我们知道 $s[i_\ell]$ 是一个最左字符，因此根据 $b_1 \approx b_{p+1}$ 可得， $s[i_\ell + p \cdot \Delta]$ 在 b_{p+1}中不会更早出现。我们需要证明它在 $b_1, b_2, \ldots, b_p$中也不会更早出现。我们分别考虑两种可能的情况 (a) 和 (b)。

(a) $s[i_{\ell+1}]$ 未定义，即 $s[i_\ell]$ 是 b_1中的最大字符。我们知道 i_ℓ 满足 (2)，因此 $s[i_\ell] < s[i_\ell + \Delta] < \ldots < s[i_\ell + (p-1) \cdot \Delta] < s[i_\ell + p \cdot \Delta]$。对于所有整数 $q \in [0, p)$，根据 $b_1 \approx b_{q+1}$ 可得， $s[i_\ell + q \cdot \Delta]$ 是 b_{q+1}中的最大字符。因此，事实上，对于每个整数 $q \in [0, p)$， $s[i_\ell + p \cdot \Delta]$ 都大于整个块 b_{q+1}中的所有字符，这使得 $i_\ell + p \cdot \Delta$ 成为最左出现位置。

(b) $i_{\ell+1}$ 已定义，且满足 (1) 或 (3)，因此 $s[i_{\ell+1}] \geq s[i_{\ell+1} + \Delta] \geq \ldots \geq s[i_{\ell+1} + (p-1) \cdot \Delta] \geq s[i_{\ell+1} + p \cdot \Delta]$。见图 8。我们知道 i_ℓ满足 (2)，所以 $s[i_\ell] < s[i_\ell + \Delta] < \ldots <$

$s[i_\ell + (p-1)\cdot\Delta] < s[i_\ell + p\cdot\Delta]$。回顾一下，$s[i_{\ell+1}]$ 是 b_1中 $s[i_\ell]$ 的严格后继。因此，对于每个 $i' \in [1,\Delta]$，我们都有 $s[i']$ 不属于区间 $(s[i_\ell], s[i_{\ell+1}])$。由于我们有 $b_1 \approx b_q$，对于每个整数 $q \in [0,p)$，这意味着 $s[i'+q\cdot\Delta]$ 不属于区间 $(s[i_\ell+q\cdot\Delta], s[i_{\ell+1}+q\cdot\Delta])$。如前所述，$s[i_\ell+q\cdot\Delta] < s[i_\ell+p\cdot\Delta]$ 和$s[i_{\ell+1}+q\cdot\Delta] \geq s[i_{\ell+1}+p\cdot\Delta]$。由此可得，对于每个 $i' \in [1,\Delta]$，都有 $s[i'+q\cdot\Delta]$不属于区间 $[s[i_\ell+p\cdot\Delta], s[i_{\ell+1}+p\cdot\Delta])$（该区间非空，因为这两个位置属于同一个块 b_q，并且根据 $b_q \approx b_1$，$s[i_{\ell+1}+q\cdot\Delta]$ 是 b_q中 $s[i_\ell+q\cdot\Delta]$ 的严格后继）。特别地，$s[i'+q\cdot\Delta] \neq s[i_\ell+p\cdot\Delta]$，因此 $s[i_\ell+p\cdot\Delta]$ 不会出现在 b_{q+1}中，使其成为最左字符。

我们已经证明，对于每个整数 $p \in [0,f)$，$i_\ell + p\cdot\Delta$ 都是最左出现位置。 □

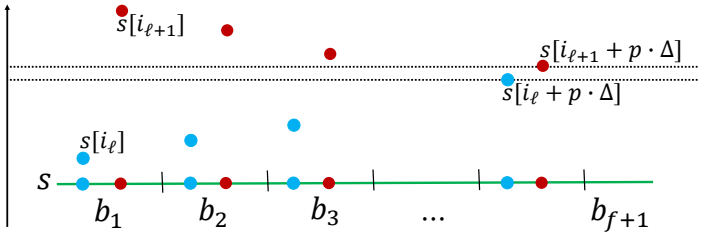

图 8：红点对应于 $s[i_{\ell+1}], \ldots, s[i_{\ell+1}+p\cdot\Delta]$。蓝点对应于$s[i_\ell], \ldots, s[i_\ell+p\cdot\Delta]$。黑色箭头表示字符的轴。

结合上述引理，我们得到以下结论。

引理 4.8。如果 $|O_k| \geq 3$，那么对于任意 $\{uv, u'v', u''v''\} \in O_k$，其中 $|u| < |u'| < |u''|$ 满足 $\{uv, u'v'\}$ 定义了 $b_1, \ldots, b_f$，b_{f+1}且 $\{u'v', u''v''\}$ 定义了 $b'_1, \ldots, b'_f$，b'_{f+1}，则每个块 $b_1, b_2, \ldots, b_f$中都存在一个最左出现位置，或者每个块 $b'_1, b'_2, \ldots, b'_{f'}$中都存在一个最左出现位置。

证明。回忆一下，根据引理 4.3，$|u'|-|u|$ 是 $u = s[1..|u|]$ 的一个 op-边界，而 $|u''|-|u'|$ 是 $u' = s[1..|u'|]$ 的一个 op-边界。根据引理 4.6，在 b_2 或 b'_2中存在一个最左出现位置。于是，根据引理 4.7，对由 op-边界 $|u'|-|u|$ 定义的块 b_1，b_2，……，b_f，b_{f+1}，或由 op-边界 $|u''|-|u'|$定义的块 b'_1，b'_2，……，$b'_{f'}$，$b'_{f'+1}$应用引理 4.7，可知每个块 b_1，b_2，……，b_f中，或每个块 b'_1，b'_2，……，$b'_{f'}$中，都存在一个最左出现位置。 □

现在，我们可以用最左字符的数量来给 $\sum_k |O_k|$ 一个上界。我们将证明，如果某个 O_k 很大，那么在某个范围内就会有许多最左字符。为此，我们定义最左出现位置组。令 L_k 由满足以下条件的最左出现位置 i 组成：$i \in [2^k, 2^{k+1})$ ：

定义 4.9。$L_k = \{i \mid i \in [2^k, 2^{k+1}) \wedge \forall_{j\in[1,i)} s[i] \neq s[j]\}$ 对于 $0 \leq k \leq$对数 m .

注意，这些组两两不相交，即对于任意 k 和 k'，都有 $L_k \cap L_{k'} = \emptyset$。因此 $\sum_{k=0}^{\log m} |L_k| \leq \sigma$。有了这个定义，我们就可以证明主要的技术引理了。

引理 4.10。作为 s 前缀的 *op*-平方数量为 $\mathcal{O}(\sigma)$。

证明。为证明该引理，我们希望将 $|O_k|$ 与 $|L_k|$联系起来，然后对 k的所有可能取值求和。$k = 0, 1$ 将单独处理；对于更大的 k，我们将分别针对 $|O_k| \geq 11$ 和 $|O_k| \leq 10$采用不同的论证。

我们首先考虑满足 $|O_k| \geq 11$的 $k \geq 2$。特别地， $|O_k| \geq 3$，因此根据命题 4.4，存在满足 $0 < |u'| - |u|$、 $|u''| - |u'| < 2^k/(|O_k| - 2)$ 的 $\{uv, u'v', u''v''\} \in O_k$。根据引理 4.8，对于任意满足 $|u| < |u'| < |u''|$ 的 $\{uv, u'v', u''v''\} \in O_k$， 若 $\{uv, u'v'\}$ 定义了 $b_1, \ldots, b_f, b_{f+1}$ 中的 u， 且 $\{u'v', u''v''\}$ 定义了 $b'_1, \ldots, b'_{f'}, b'_{f'+1}$ 中的 u'， 那么要么在每个块 $b_1, b_2, \ldots, b_f$ 中都有一个最左出现位置，要么在每个块 $b'_1, b'_2, \ldots, b'_{f'}$ 中都有一个最左出现位置。无论哪种情况，我们都找到了满足 $0 < \Delta < 2^k/(|O_k| - 2)$ 的 $\{uv, u'v'\} \in O_k$， 其中 $\Delta = |u'| - |u|$， 使得 $\{uv, u'v'\}$ 定义了 $b_1, \ldots, b_f$, b_{f+1} 满足 $f = \lfloor |u|/\Delta \rfloor$， 且每个块 $b_1, b_2, \ldots, b_f$ 中都有一个最左出现位置。我们希望得到 L_{k-2} 中最左出现位置数量的下界。为此，只需给出完全包含在范围 $[2^{k-2}, 2^{k-1})$ 内的块 b_i 数量的下界。回顾 $|u| \in [2^{k-1}, 2^k)$， 且 $u = b_1 b_2 \ldots b_f b_{f+1}$。因此， $u[2^{k-2}..2^{k-1} - 1]$ 由某个 b_j 的一个后缀（可能为空）、接着的 $b_{j+1}, b_{j+2}, \ldots, b_{j+\ell}$， 以及 $b_{j+\ell+1}$ 的一个前缀构成（其中 $b_{j+\ell+1}$ 可能是块 b_{f+1}， 即不完整块，不应计入下界）。因此，完全包含在范围 $[2^{k-2}, 2^{k-1})$ 内的块 b_i 的数量至少为 $\lfloor 2^{k-2}/\Delta \rfloor - 1$。参见图 9。结合 Δ 的上界，我们得到以下不等式：

$$|L_{k-2}| \geq \left\lfloor \frac{2^{k-2}}{\Delta} \right\rfloor - 1 \geq \frac{2^{k-2}}{\Delta} - 2 > \frac{|O_k| - 2}{4} - 2 = \frac{|O_k| - 10}{4}.$$

利用假设 $|O_k| \geq 11$， 我们得出 $|L_{k-2}| > |O_k|/44$。因此：

$$\sum_{k \geq 2 : |O_k| \geq 11} |O_k| < \sum_{k \geq 2} 44 \cdot |L_{k-2}| \leq 44 \sum_k |L_k| \leq 44 \cdot \sigma.$$

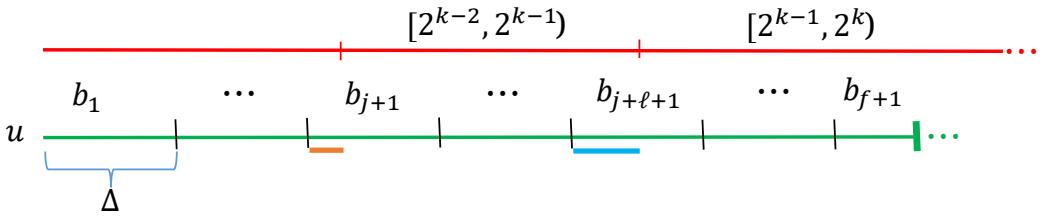

图 9：橙线对应于 b_j 的后缀。蓝线对应于 $b_{j+\ell+1}$ 的前缀。红线展示了这些范围。

接下来，我们考虑满足 $|O_k| \leq 10$ 的 $k \geq 2$。显然，我们有平凡的上界 $\sum_{k: |O_k| \leq 10} |O_k| \leq \sum_{k=0}^{\log m} 10 = \mathcal{O}(\log\ m)$。与前一种情况一样，我们希望利用最左出现位置来改进上界。回想一下，根据命题 4.1，每个 op-平方 $uv \in O_k$ 都由一对最左出现位置 i 和 j 定义，其中 i 属于 u， 而 j 属于 v。由于 $|uv| \in [2^k, 2^{k+1})$， 我们得出 j 落在范围 $[2^{k-1} + 1, 2^{k+1})$ 内，因此必属于 $L_{k-1} \cup L_k$。于是， O_k 只有在 L_{k-1} 或 L_k 非空时才可能非空。因此：

$$\begin{aligned} \sum_{k \geq 2 : |O_k| \leq 10} |O_k| \leq \sum_{k \geq 2 : |O_k| > 0} 10 \leq \sum_{k \geq 2 : |L_{k-1}| > 0} 10 + \sum_{k \geq 2 : |L_k| > 0} 10 \\ \leq 10 \sum_{k \geq 2} |L_{k-1}| \ + 10 \sum_{k \geq 2} |L_k| \ \leq 20\sigma. \end{aligned}$$

为了求 $\sum_k |O_k|$ 的上界，我们将总和分成三部分。对于 $k = 0$， 1，我们有 $|O_0| \leq 1$ 和 $|O_1| \leq 2$。然后，对于 $k \geq 2$， 我们分别考虑所有满足 $|O_k| \geq 11$ 和 $|O_k| \leq 10$ 的 k， 并代入

上述上界。总之，我们得到：

$$\sum_k |O_k| \le 1 + 2 + 44 \cdot \sigma + 20 \cdot \sigma = \mathcal{O}(\sigma).$$

因此，作为 s 前缀的 op-平方数量为 $\mathcal{O}(\sigma)$。 □

我们以主要定理结束本节。

定理 4.11。 长度为 n 、取自大小为 σ 的字母表的字符串 s 中，*op*-平方的数量为$\mathcal{O}$（σn）。

证明。 我们分别考虑 s 的每个后缀。对于每个后缀 $s[i..n]$，应用引理 4.10，可得作为 $s[i..n]$ 前缀的 op-平方数量的上界为 $\mathcal{O}(\sigma)$。因此，对所有 i 求和可得， s 中 op-平方的数量为 $\mathcal{O}(\sigma n)$。 □

5 算法

本节中，我们描述一种算法，该算法可以在 $\mathcal{O}(\sigma n)$ 时间内报告取自大小为 σ 的字母表的字符串 $w[1..n]$ 中所有 op-平方的出现位置。

该算法的总体思路是生成 $\mathcal{O}(\sigma n)$ 个 op-平方候选项，然后在常数时间复杂度内逐一检验。为此，我们首先介绍一种能够在常数时间复杂度内检查 $w[i..i+\ell-1] \approx w[i+\ell..i+2\ell-1]$ 的机制。根据 [3] 中的说明，可以通过对 w 的保序后缀树进行最低公共祖先查询来实现。不过，我们还需要说明如何在 $\mathcal{O}(\sigma n)$ 时间内构造这一结构。

根据 [3], 对字符串 $w[1..n]$ 的定义，我们将 code(w) 定义为 ($\phi(w,1)$ $\phi(w,2)\ldots,\phi(w,n)$)，其中 $\phi(w,i)$ = (prev$_<(w,i)$, prev$_=(w,i)$)，且 prev$_<(w,i) = |\{k < i\colon w[k] < w[i]\}|$，prev$_=(w,i) = |\{k < i\colon w[k] = w[i]\}|$。我们注意到，code($w$) = code($w'$) 当且仅当 $w \approx w'$。于是，$w[1..n]$ 的保序后缀树就是由所有形如 code($w[i..n]$)# 的字符串构成的压缩字典树，其中 $i = 1$、 2 、 . . . 、 n。显然， $w[i..i+\ell-1] \approx w[i+\ell..i+2\ell-1]$ 当且仅当对应于 code($w[i..n]$)\$ 和 code($w[i+\ell..n]$)\$ 的叶节点的最低公共祖先的字符串深度至少为 ℓ。因此，假设我们已经构造了 $w[1..n]$ 的保序后缀树，那么在进行 $\mathcal{O}(n)$ 的最低公共祖先查询预处理后，就能在常数时间复杂度内完成此类测试 [15]。剩下的问题是说明如何构造保序后缀树。需要强调的是，尽管 [3] 确实提供了一种高效的 $\mathcal{O}(n\log n/\log\log n)$ 时间构造算法（事实上，完整版 [4] 进一步将时间复杂度改进为 $\mathcal{O}(n\sqrt{\log n})$），但这一复杂度与我们的目标不符。

引理 5.1。 给定一个由大小为 σ 的字母表构成的字符串 $w[1..n]$，我们可以在 $\mathcal{O}(\sigma n)$ 时间和空间内构造其保序后缀树。

证明。 如 [3], 所述， $w[1..n]$ 的保序后缀树可以利用 Cole 和 Hariharan [2]提出的通用框架构造，该框架用于为字符串准后缀集合 $w_1, w_2, \ldots, w_n$构造后缀树。假设可以在常数时间复杂度内访问任意 w_i的第 j 个字符，他们的算法运行时间为 $\mathcal{O}(n)$，且失败概率几乎呈反指数级。用于访问 w_i 第 j 个字符的机制称为字符预言机。在这个具体应用中， $w_i = \phi(w[i..n])$#。我们将先说明如何为这类字符串实现常数时间复杂度的字符预言机，然后解释为什么在我们的场景中不需要随机化。

我们需要实现一个新的字符预言机，它能在预处理耗时和空间为 $\mathcal{O}(\sigma n)$ 后，以常数时间复杂度返回任意 i、j对应的 $\phi(w[i..n], j)$。这要求能够在常数时间复杂度内计算 $\text{prev}_<(w[i..n], j)$ 和 $\text{prev}_=(w[i..n], j)$。为此，我们定义一个二维数组 $\text{cnt}[i, x] = |\{k : k < i, w[k] < x\}|$，其中 $i = 0$ 取 $1, \ldots, n$，$x = 0$ 取 $1, \ldots, \sigma$。该数组中的所有元素都可以在总耗时和空间为 $\mathcal{O}(\sigma n)$ 的情况下计算出来。然后，我们可以按如下方式计算任意 $\text{prev}_<(w[i..n], j)$ 和 $\text{prev}_=(w[i..n], j)$：

$$\begin{aligned}\text{prev}_<(w[i..n], j) =& \text{cnt}[i + j - 2, w[i + j - 1]] - \text{cnt}[i - 1, w[i + j - 1]]\\ \text{prev}_=(w[i..n], j) =& (\text{cnt}[i + j - 2, w[i + j - 1]] - \text{cnt}[i + j - 2, w[i + j - 1] - 1])\\ & - (\text{cnt}[i - 1, w[i + j - 1]] - \text{cnt}[i - 1, w[i + j - 1] - 1]).\end{aligned}$$

为消除随机化，我们注意到，Cole 和 Hariharan 算法中随机化的唯一来源，是需要为当前树的每个显式节点维护一个以出边上的下一个字符为索引的字典。如果我们能证明这样的边最多有 $\mathcal{O}(\sigma)$ 条，那么就可以用简单列表来实现该字典，使构造时间增加到 $\mathcal{O}(\sigma n)$，仍在我们声称的界限之内。

考虑当前树的一个非叶节点 v。它对应于某个 $\text{code}(w[i..n])$# 的真前缀；根据 code(.) 的定义，这等于某个 $w[i..j]$ 的 $\text{code}(w[i..j])$。令 $c_1 < c_2 < \ldots < c_k$ 为 $w[i..j]$ 中的不同字符，并用 occ_x 表示 c_x 在 $w[i..j]$ 中的出现次数。现在考虑一条从 v 出发的边，并令 $\text{code}(w[i'..j' + 1])$ 对应于其后第一个节点（隐式或显式）。我们知道 $\text{code}(w[i'..j']) = \text{code}(w[i..j])$，因此 $w[i'..j']$ 中的不同字符为 $c'_1 < c'_2 < \ldots < c'_k$，其中 occ_x 是 c'_x 在 $w[i'..j']$ 中的出现次数。接下来，我们分析 ($\text{prev}_<(w[i'..j'+1], j'-i'+2)$, $\text{prev}_=(w[i'..j'+1], j'-i'+2)$) 的可能取值，也就是所考虑边上的第一个字符。第一个数始终等于某个 $\sum_{y=1}^{x-1} c_y$，其中 $x \in [1, k + 1]$。因此，第二个数要么为 0，要么为 occ_x。所以，总共只有 $2k \le 2\sigma$ 种可能的首字符，这将任意 v 的度限制为 $\mathcal{O}(\sigma)$。 □

算法的主要部分是高效地生成 op-平方的候选项 $\mathcal{O}(\sigma n)$。然后，按照上文所述，在假设已根据引理 5.1 完成预处理的前提下，可以在常数时间复杂度内测试每个候选项。与证明 op-平方数量上界的 $\mathcal{O}(\sigma n)$类似，我们将逐一考察输入字符串 $w[1..n]$ 的后缀。设 $s = w[i..n]$ 为当前考察的后缀，$x_1 < x_2 < \ldots < x_t$ 为 s中的最左出现位置。对于每个后缀花费 $\mathcal{O}(\sigma)$时间后，我们可以假设已知位置 x_1、x_2、.....、x_t，因为从$w[i..n]$ 移动到 $w[i - 1..n]$ 后，我们只需插入新的最左出现位置 $i - 1$，并可能删除之前的最左出现位置i'，使得 $w[i - 1] = w[i']$（除非此前未见过 $w[i - 1]$），这可以在 $\mathcal{O}(t) = \mathcal{O}(\sigma)$时间内完成。根据命题 4.1，$s$ 中任何一个属于 op-平方的前缀，都可以通过选择位置 x_i 和x_j处的两个最左字符来得到，其中 $i < j$，并将可能的平方长度设为 $2(x_j - x_i)$。这样便得到 $\mathcal{O}(\sigma^2)$个可能成为 op-平方的前缀候选项。然而，我们的目标是只生成 $\mathcal{O}(\sigma)$个这样的候选项。回顾一下，所有最左出现位置都被划分为若干组 L_0、L_1、.....。我们首先证明，只需考虑x_j 中所在组的最小或最大元素。

引理 5.2. *考虑一个 op-平方 $s[1..2\ell]$。那么存在 $i < j$，使得 $x_j \in [\ell + 1, 2\ell]$、$x_j - x_i = \ell$ 和 x_j 是其所在组中的最小或最大元素。*

证明。 根据命题 4.1，我们知道 $s[\ell + 1..2\ell]$ 中存在一个最左出现位置。选择最大的 k，使得 $2^k \le \ell$（因此 $2^{k+1} > \ell$）。考虑分别对应于组 L_k 和 L_{k+1} 的两个范围 $[2^k, 2^{k+1})$ 和 $[2^{k+1}, 2^{k+2})$。由于 $2^k \le \ell$ 和 $2^{k+1} > \ell$，我们有$2^k < \ell + 1$、$2^{k+1} \in [\ell + 1, 2\ell]$ 和 $2\ell < 2^{k+2}$。因此，$s[\ell + 1..2\ell]$ 可以表示为

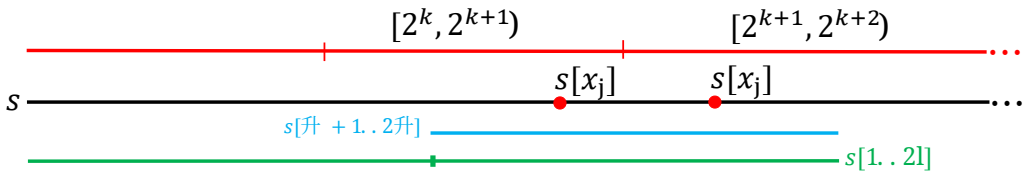

图10：op-平方 $s[1..2\ell]$ 以绿色标出。红点是$s[x_j]$的两种可能性。 $s[\ell+1..2\ell]$ 以蓝色标出。红线表示这些范围。

$s[2^k..2^{k+1})$ 的一个后缀与 $s[2^{k+1}..2^{k+2})$ 的一个前缀的拼接。落在 $s[\ell+1..2\ell]$ 内的最左出现位置属于该后缀或前缀。见图10。如果它落在后缀内， L_k 的最大元素属于 $[\ell+1, 2\ell]$。如果它落在前缀内， L_{k+1} 的最小元素属于 $[\ell+1, 2\ell]$。令 $x_j \in [\ell+1, 2\ell]$ 为相应的最左出现位置。为了完成证明，我们需要证明存在 $i < j$ 使得 $x_j - x_i = \ell$。 $s[x_j]$ 与所有 $s[1], s[2], \ldots, s[x_j - 1]$ 均不同；根据 $s[1..\ell] \approx s[\ell+1..2\ell]$，我们得到 $s[x_j - \ell]$ 与所有 $s[1], s[2], \ldots, s[x_j - \ell - 1]$ 均不同。因此， $x_j - \ell$ 是一个最左出现位置，故对于某个 $i < j$，有 $x_j - \ell = x_i$，如所需。

□

为了生成候选项，我们遍历所有满足以下条件的 j： x_j 是其组 L_k中的最小或最大元素。考虑满足 $x_j - x_i = \ell$和 $x_j \in [\ell+1, 2\ell]$ 的 $i < j$，其中有一个 op-平方 $s[1..2\ell]$。那么，由于 $x_j \in [2^k, 2^{k+1})$， $\ell \geq 2^{k-1}$。为避免图示过于杂乱，令 $s' = s[1..2^{k-1}]$。由于假设 $s[1..2\ell]$ 是一个 op-平方，因此 $s[\ell..\ell+2^{k-1}-1] \approx s'$。这启发出以下生成候选项的自然策略：我们遍历所有满足 $x_j \in [y, y+2^{k-1}-1]$ 和$s' \approx s[y..y+2^{k-1}-1]$的片段 $s[y..y+2^{k-1}-1]$，并输出 $s[1..2(y-1)]$ 作为可能的 op-平方（如前文所述，每个此类候选项随后都会在常数时间复杂度内进行测试）。见图11。我们首先证明，此类片段的数量可以由 $\mathcal{O}(1+|L_{k-2}|)$ 给出上界，然后说明如何在相同的时间复杂度内生成它们。

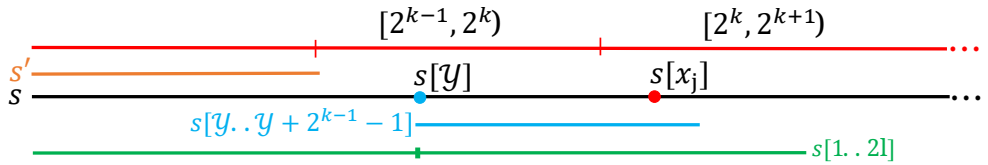

图11：op-平方 $s[1..2\ell]$、片段 $s[y..y+2^{k-1}-1]$ 和 s 分别以绿色、蓝色和黑色着色。 s' 以橙色着色。红线表示这些范围。

引理 5.3。 满足 $x_j \in [y, y+2^{k-1}-1]$ 和$s' \approx s[y..y+2^{k-1}-1]$ 的片段 $s[y..y+2^{k-1}-1]$ 的数量上界为 $\mathcal{O}(1+|L_{k-2}|)$。

证明。 考虑所有这类片段 $s[y_1..y_1+2^{k-1}-1]$， $s[y_2..y_2+2^{k-1}-1]$， ……， $s[y_t..y_t+2^{k-1}-1]$。由于 $x_j \in [y_i, y_i+2^{k-1}-1]$ 对于每个 $i=1$， 2， ……， t，要么 $t=1$，要么根据鸽巢原理，存在 i 使得 $y_{i+1} - y_i < 2^{k-1}/(t-1)$。若 $t=1$，则结论成立。否则，令$\ell' = |s[y_{i+1}..y_i+2^{k-1}-1]|$。根据假设， $s' \approx s[y_i..y_i+2^{k-1}-1]$ 且 $s' \approx s[y_{i+1}..y_{i+1}+2^{k-1}-1]$，因此由 $\approx$ 的传递性，也有 $s[y_i..y_i+2^{k-1}-1] \approx s[y_{i+1}..y_{i+1}+2^{k-1}-1]$。由此可得$s'[1..\ell'] \approx s'[y_{i+1}-y_i+1..2^{k-1}]$，换言之， ℓ'是 s'的 op-边界。令 b_1， b_2， ……， b_f， b_{f+1}为 ℓ' 在 $s' = w[i..i+|s'|-1]$中定义的块，其中每个块的长度为 $\Delta = 2^{k-1} - \ell'$。

见图12。回顾一下，x_j 是 $w[i..n]$中的一个最左出现位置，并且根据 y_i 和y_{i+1} 的定义，我们有 $x_j \in [y_{i+1}, y_i + 2^{k-1} - 1]$。于是，由 $s'[1..\ell'] \approx s'[y_i..y_i + 2^{k-1} - 1]$ 可得，$x_j - y_i + 1 \in [y_{i+1} - y_i + 1, 2^{k-1}]$ 也是 $w[i..n]$中的一个最左出现位置。因此，对于某个 $j \geq 2$，我们在 b_j 中有一个最左出现位置。这使我们可以应用引理 4.6，再应用引理 4.7，从而得出每个块 b_1，b_2，……，b_f中都有一个最左出现位置。我们计算这些最左出现位置落在范围 $[2^{k-2}, 2^{k-1})$内的数量下界：

$$\begin{aligned}\left\lfloor \frac{2^{k-2}}{\Delta} \right\rfloor - 1 &> \frac{2^{k-2}}{2^{k-1} - \ell'} - 2 \\ &= \frac{2^{k-2}}{2^{k-1} - (y_i + 2^{k-1} - y_{i+1})} - 2 = \frac{2^{k-2}}{y_{i+1} - y_i} - 2 \\ &> \frac{2^{k-2}}{2^{k-1}/(t-1)} - 2 = (t-5)/2.\end{aligned}$$

对于 $t < 6$，由于片段数为 $\mathcal{O}(1)$，证明完毕。否则，我们得到 $|L_{k-2}| \geq (t-5)/2 \geq t/12$，因此 $t = \mathcal{O}(1 + |L_{k-2}|)$ 始终成立，正如所声称的那样。 □

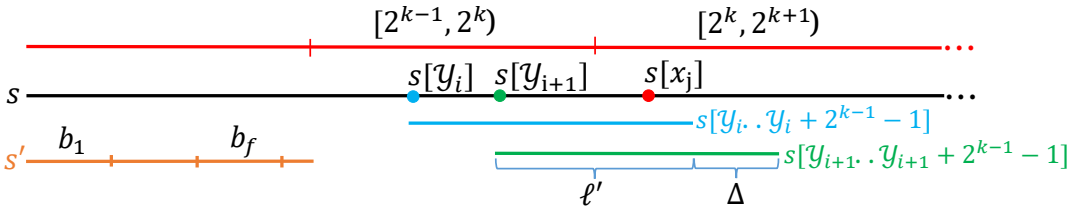

图12： s、 s'、$s[y_i..y_i + 2^{k-1} - 1]$和 $s[y_{i+1}..y_{i+1} + 2^{k-1} - 1]$ 分别标为黑色、橙色、蓝色和绿色。红线表示这些范围。

因此，对于每个满足 L_k 非空的 k，我们会生成 $\mathcal{O}(1+|L_{k-2}|)$ 个候选项。按照上述策略生成的候选项总数为 $\sum_{k:\ L_k \neq \emptyset} \mathcal{O}(1 + |L_{k-2}|) = \mathcal{O}(\sigma + \sum_k |L_{k-2}|) = \mathcal{O}(\sigma)$，如所承诺的那样。接下来需要说明如何用与其数量成正比的时间，访问所有满足 $x_j \in [y, y + 2^{k-1} - 1]$ 和 $s' \approx s[y..y + 2^{k-1} - 1]$ 的片段 $s[y..y + 2^{k-1} - 1]$。

我们将解决一个更一般的问题，并说明如何确保：在考虑 $s = w[i..n]$时，对于 s 中的每个最左出现位置 x_j，我们都能访问满足 $x_j \in [y, y + 2^{k-1} - 1]$ 和 $s[1..2^{k-1} - 1] \approx s[y..y + 2^{k-1} - 1]$的所有片段 $s[y..y+2^{k-1}-1]$的列表，其中 $2^k \leq x_j < 2^{k+1}$。我们称此列表为 i 和 x_j 的结果 。

回顾一下 $s = w[i..n]$，我们按此顺序依次考虑 $i = n$，$n-1$，……，1。考虑到 $s = w[i..n]$时，i 成为一个最左出现位置，并一直保持最左出现位置，直到遇到满足 $w[i] = w[\,\mathrm{prev}_i]$ 的 $s = w[\,\mathrm{prev}_i..n]$（也可能直到扫描结束，它都一直是最左出现位置）。按照前文所述，通过维护一个最左出现位置列表，我们可以在 $\mathcal{O}(\sigma n)$ 时间内为 i 中的每个 i 计算 prev_i。当 $i' \in [\,\mathrm{prev}_i,\ i]$ 且 $2^k \leq i - i' + 1 < 2^{k+1}$ 时，我们称位置 i 在位置 i' 处是 k-活跃的。我们观察到，随着考虑的 w 后缀越来越长，i 会先是 0-活跃，然后是 1-活跃，以此类推，直到变为 k_i-活跃，之后便不再活跃。此外，满足 i 在 i' 处是 k-活跃的索引 i' 构成一个连续范围 $[\mathrm{begin}_{i,\ k},\ \mathrm{end}_{i,\ k}]$（每个此类范围的长度都是 2^k，但 $k = k_i$ 的范围可能更短）。如下所示，这些范围的总长度很小。

命题 5.4。 $\sum_{i,\ k:\ k \leq k_i} 2^k = \mathcal{O}(\sigma n)$

证明。 对于 $k \geq 1$，我们可以用 $2 \cdot |[\mathsf{begin}_{i,\ k-1},\ \mathsf{end}_{i,\ k-1}]|$ 给出 2^k 的上界。于是求和变为：

$$\sum_{i,k:k\leq k_i} 2^k = n + 2 \sum_{i,k:1\leq k\leq k_i} |[\mathsf{begin}_{i,k-1}, \mathsf{end}_{i,k-1}]| \leq n + 2 \sum_{i,k:k\leq k_i} |[\mathsf{begin}_{i,k}, \mathsf{end}_{i,k}]|.$$

我们注意到，每个 $i' \in [\mathsf{begin}_{i,k}, \mathsf{end}_{i,k}]$ 都对应于 $i - i' + 1$ 在$w[i'..n]$中的一个最左出现位置。由于任何 $w[i'..n]$中最多有 σ 个最左出现位置，因此我们可以将总和的上界设为 $\mathcal{O}(\sigma n)$。

□

这样，我们就可以按如下方式在物理存储中保存结果。对于每个 x，我们都有一个以 $k \leq k_i$为索引的数组，记作 $R[x]$。该数组的每个条目都是一个以 $i \in [\mathsf{begin}_{i,k}, \mathsf{end}_{i,k}]$为索引的数组，记作 $R[x][k]$。最后，该数组的每个条目记作 $R[x][k][i]$，是一个指向 y列表的指针，其中的 y满足 $x \in [y, y + 2^{k-1} - 1]$ 且 $w[i..i + 2^{k-1} - 1] \approx w[y..y + 2^{k-1} - 1]$（注意，这是指向列表的指针，而非其单独的物理副本）。借助这些数组，我们可以通过获取指针 $R[x_j][k][i]$，在常数时间复杂度内访问每个 x_j、$k \leq k_{x_j}$ 和 i 对应的结果（首先验证$i \in [\mathsf{begin}_{i,k}, \mathsf{end}_{i,k}]$是否成立）。根据命题 5.4，所有数组 $R[x]$ 的总长度仅为 $\mathcal{O}(\sigma n)$。此外，根据以下论证，我们需要准备的所有出现位置列表的总长度也是 $\mathcal{O}(\sigma n)$（假设我们将每个$R[x][i]$ 存储为指向相应列表的指针，而非存储其物理副本）。考虑 i和 $k \leq k_i$。那么，我们需要一个位置列表 y，其中的位置满足 $i \in [y, y + 2^{k-1} - 1]$，且 $w[y..y + 2^{k-1} - 1]$ 与特定字符串 s'序同构。因此，我们可以将所有满足 $i \in [y, y + 2^{k-1} - 1]$ 的 y 划分为若干组，每组对应一个序同构片段$w[y..y + 2^{k-1} - 1]$，然后存储指向相应列表的指针（如果不存在 y，则该指针可以为空）。根据命题 5.4，所有 i 和 $k \leq k_i$对应的位置 y总数为 $\mathcal{O}(\sigma n)$，这给出了所有列表总长度的上界。

接下来需要说明如何高效地计算这些结果。这需要将所有片段 $w[y..y + 2^{k-1} - 1]$ 使得 $i \in [y, y + 2^{k-1} - 1]$ 且 $k \leq k_i$ 划分为序同构的组，并为每个 $i, k \leq k_i, i' \in [\mathsf{begin}_{i,k}, \mathsf{end}_{i,k}]$ 找到一个指向片段列表$w[y..y + 2^{k-1} - 1]$ 的指针，其中包含与 $w[i'..i' + 2^{k-1} - 1]$序同构的片段，且满足 $i \in [y, y + 2^{k-1} - 1]$。这两个步骤都可以借助保序后缀树实现：只需对其进行预处理，预处理的时间和空间复杂度均为 $\mathcal{O}(n)$，即可在常数时间复杂度内计算任意代码$(w[x..x + 2^\ell - 1])$的（确定性）指纹。这里，指纹指的是一个由 $\mathcal{O}(\log n)$ 位组成的整数，记作 $\mathsf{fingerprint}_\ell(x)$，满足 $\mathsf{fingerprint}_\ell(x) = \mathsf{fingerprint}_\ell(x')$ 当且仅当 $\mathsf{code}(w[x..x + 2^\ell - 1]) = \mathsf{code}(w[x'..x' + 2^\ell - 1])$（或等价地 $w[x..x + 2^\ell - 1] \approx w[x'..x' + 2^\ell - 1]$）。我们首先介绍这种机制，然后更详细地说明如何应用它。

引理 5.5。 具有 n 个叶节点的压缩字典树可以在 $\mathcal{O}(n)$ 时间内完成预处理，使得对于任意叶节点u 和整数 k，我们都能在常数时间复杂度内查询到 $\mathcal{O}(\log n)$位的指纹，该指纹对应于 u 在字符串深度 2^k处的祖先节点。

证明。 这是通过应用用于解决 [11, 引理 14]中提到的子串指纹问题的方法得出的。按照完整版 [12, 引理 12], 中的描述，具有 n 个叶节点的压缩字典树可以在 $\mathcal{O}(n)$时间内完成预处理，从而我们能够在常数时间复杂度内，找到给定叶节点在字符串深度 2^k 处对应的祖先节点（隐式或显式）。如果要查找的节点是隐式的（在压缩字典树中并不存在实际节点），我们就取回包含该节点的边。接下来，如果该节点是显式的，我们就返回其标识符；如果该节点是隐式的，我们就返回包含它的边的标识符。因此，所需的标识符范围为 $[2n]$。 □

我们在保序后缀树上应用引理 5.5。这样，我们就能在常数时间复杂度内计算出满足所需性质的任意指纹$\ell(x)$。现在考虑任意 i 和 $k \leq k_i$。我们首先对所有满足 $i \in [y, y + 2^{k-1} - 1]$的 y 计算指纹$_{k-1}(y)$。这需要 $\mathcal{O}(2^k)$

时间。接下来，我们对每个 $i' \in [\mathsf{begin}_{i,k}, \mathsf{end}_{i,k}]$计算指纹$_{k-1}(i')$，同样需要 $\mathcal{O}(2^k)$ 时间，因为$|[\mathsf{begin}_{i,k}, \mathsf{end}_{i,k}]| \leq 2^{k-1}$。我们对所有指纹进行排序，并将它们划分为对应于序同构片段的组。我们同样需要在 $\mathcal{O}(2^k)$ 时间内实现这一步。为此，我们注意到需要对由 $\mathcal{O}(\log n)$ 个比特组成的 $\mathcal{O}(2^k)$ 个整数进行排序，而这可以通过基数排序在 $\mathcal{O}(2^k + n)$ 时间内完成。为了避免为每个 i 和 $k \leq k_i$支付 $\mathcal{O}(n)$ 的代价，我们注意到这是一个离线问题，并且可以将对应于不同 i 和 $k \leq k_i$ 的所有集合一起排序。具体而言，我们对形如 $(i, k_i, \mathsf{fingerprint}_\ell(y)\ y)$ 和 $(i, k_i, \mathsf{fingerprint}_\ell(i')\ i')$ 的元组进行排序。根据引理 5.4，所有元组的总数为 $\mathcal{O}(\sigma n)$；由于每个元组都可视为由 $\mathcal{O}(\log n)$ 个比特组成的整数，因此可以在 $\mathcal{O}(\sigma n + n) = \mathcal{O}(\sigma n)$ 时间内完成排序。然后，我们从输出中提取每个 i 和 $k \leq k_i$ 的结果。对于每个 i 和 $k \leq k_i$，我们考察每个指纹相等的组。对于每个组，我们首先创建一个列表，其中包含该组中与指纹$_{k-1}(y)$对应的所有位置 y。然后，对于属于该组的每个指纹$_{k-1}(i')$，我们存储一个指向该列表的指针。总体而言，这需要 $\mathcal{O}(\sigma n)$ 时间，并使我们能够以相同的时间复杂度计算出所有结果。

6 开放问题

我们研究的一个有趣后续方向是，首先对并非序同构的保序平方数量给出上界，然后设计一种能够报告所有此类平方的算法。

参考文献

[1] S. Brlek and S. Li. On the number of 平方 in a finite word. *arXiv* 2022 *v*,.[2] R. Cole 和 R. Hariharan。采用缺失后缀链接加快后缀树构造。载于 *STOC*，第 407–415 页。ACM，2000 年。[3] M. Crochemore、C. S. Iliopoulos、T. Kociumaka、M. Kubica、A. Langiu、S. P. Pissis、J. Ra- doszewski、W. Rytter 和 T. Wale´n。保序不完整后缀树和保序索引。载于 *SPIRE*，第 8214 卷《计算机科学讲义》，第 84–95 页。Springer，2013 年。[4] M. Crochemore、C. S. Iliopoulos、T. Kociumaka、M. Kubica、A. Langiu、S. P. Pissis、J. Ra- doszewski、W. Rytter 和 T. Wale´n。保序索引。《理论计算机科学》，638：122–135，2016 年。[5] M. Crochemore 和 W. Rytter。平方、立方串与时间空间高效的字符串搜索。《*Algorithmica*》，13：405–425，1995 年。[6] L.J. Cummings 和 W.F. Smyth。字符串中的弱重复。《组合数学杂志》*Combinatorial Comput*，24：33–48，1997 年。[7] A. Deza、F. Franek 和 A. Thierry。一个字符串可以包含多少个双平方串？《离散应用数学》，180：52–69，2015 年。[8] P. Erd¨os。一些尚未解决的问题。*Magy. Tud. Akad. Mat. Kut. Int´ez. K¨ozl.*，6：221–254，1961 年。[9] A. A. Evdokimov。由有限个符号生成的强非对称序列。《苏联科学院通报》，179(6)：1268–1271，1968 年。

[10] A. S. Fraenkel 和 J. Simpson。一个字符串可以包含多少个平方？组合理论（A辑），82(1)：112–120，1998 年。[11] P. Gawrychowski。Lempel-Ziv 压缩字符串中的模式匹配：快速、简单且确定性。载于 ESA，《计算机科学讲义》第 6942 卷，第 421–432 页。Springer，2011 年。[12] P. Gawrychowski。Lempel-Ziv 压缩字符串中的模式匹配：快速、简单且确定性。CoRR，abs/1104.4203，2011 年。[13] G. Gourdel、T. Kociumaka、J. Radoszewski、W. Rytter、A. Shur 和 T. Wale´n。保序模型中的字符串周期。信息与计算，270：104463，2020 年。[14] D. Gusfield 和 J. Stoye。在线性时间内查找并表示字符串中所有串联重复的算法。《计算机与系统科学》，69(4)：525–546，2004 年。[15] D. Harel 和 R. E. Tarjan。查找最近公共祖先的快速算法。SIAM计算杂志，13(2)：338–355，1984 年。[16] M. Huova、J. Karhum¨aki 和 A. Saarela。介于单词与阿贝尔词之间的问题：k-阿贝尔可避免性。《理论计算机科学》，454：172–177，2012 年。[17] L. Ilie。一个简单的证明：长度为 n 的单词至多有 2n 个不同平方。组合理论杂志（A辑），112(1)：163–164，2005 年。[18] L. Ilie。关于单词中平方数量的短文。《理论计算机科学》，380(3)：373–376，2007 年。[19] V. Ker¨anen。阿贝尔平方在 4 个字母上是可避免的。自动机、语言与编程，第 41–52 页，1992 年。[20] T. Kociumaka、J. Radoszewski、W. Rytter 和 T. Wale´n。单词中不同且不等价的非标准平方的最大数量。《理论计算机科学》，648(C)：84–95，2016 年。[21] T. Kociumaka、J. Radoszewski 和 B. Wi´sniewski。阿贝尔字符串学问题的亚二次时间算法。计算机与信息科学的数学方面，第 320–334 页，2016 年。[22] M. Kubica、T. Kulczy´nski、J. Radoszewski、W. Rytter 和 T. Wale´n。用于连续置换模式匹配的线性时间算法。信息处理快报，113(12)：430–433，2013 年。[23] N. H. Lam。关于字符串中平方的数量。AdvOL报告 2，2013 年。[24] F. Manea 和 S. Seki。平方密度递增映射。载于第 10 届 WORDS，9304：160–169，2015 年。[25] Y. Matsuoka、T. Aoki、S. Inenaga、H. Bannai 和 M. Takeda。子串一致等价关系下的广义模式匹配与周期性。《理论计算机科学》，656：225–233，2016 年。[26] P. A. B. Pleasants。非重复序列。剑桥哲学学会数学会报，68(2)：267–274，1970 年。

[27] A. Thierry。一项证明：长度为 n 的单词包含少于 $1.5n$ 个不同平方。$arXiv$，2020。.[28] A. Thue。《关于无限字符序列》。特隆赫姆皇家科学与文学学会文献，数学-自然科学组（克里斯蒂安尼亚），7:1–22，1906。