arXiv:2301.01571v1 [cs.DM] 4 Jan 2023

使用子词或因子查询重构词

Gwena¨el Richomme*，Matthieu Rosenfeld†

2023年1月5日

摘要

我们研究词重构问题。在改进 P. Fleischmann、M. Lejeune、F. Manea、D. Nowotka 和 M. Rigo 先前的一项结果后，我们证明，对于字母表基数为 k、长度为 n 的任意未知词w，可以根据 $O(k^2\sqrt{n}\log_2(n))$ 个词中 w 作为子~~词（或散~~布因子）出现的次数重构该词。我们也略微改进了 S. S. Skiena 和 G. Sundaram 得出的两个先前上界：一个针对考虑子词是否存在（而非其出现次数）的信息，另一个针对考虑因子是否存在的信息。

*保罗-瓦莱里蒙彼利埃第三大学、蒙彼利埃大学、法国国家科学研究中心、法国蒙彼利埃
†蒙彼利埃大学，法国国家科学研究中心，法国蒙彼利埃

1 引言

一个自然的组合学问题是：需要关于某个对象的多少部分信息，才能重构该对象（示例见下文及参考文献）。例如，在[2, 3], 中，P. Fleischmann、M. Lejeune、F. Manea、D. Nowotka 和 M. Rigo 研究了这样一个问题：如何根据某些词中 w 作为子词出现次数的信息，重构词 w 。回顾一下，如果词 u 和词 w 可分别分解为 $u = u_1 \cdots u_\ell$和 $w = v_0 u_1 v_1 \cdots u_\ell v_\ell$，其中$u_1$、.....、 u_ℓ、 v_0、.....、 v_ℓ是某些词，那么词 u 就是词 w 的子词（或词 w的散布子词）。这样的双重分解标示了 u作为w的子词的一次出现。 u 作为 w 的子词的出现次数有时记作二项式系数 $\binom{w}{u}$ ，因为当词u 和 w 都由同一个字母组成时，该次数与传统系数 $\binom{|w|}{|u|}$ 相同（这里，按照词组合学中的惯例， $|w|$ 表示 w的长度），例如参见 [8, 第 6]章。Fleischmann 等人研究的问题被表述为一个猜词游戏，玩家必须猜出一个未知词。玩家不断以特定形式提问，直到获得足够的信息，能够唯一确定该词。更具体地说，每一轮中，玩家根据先前得到的回答选择一个词 u ，并询问 $\binom{w}{u}$的值。玩家的目标是尽量减少提问次数。Fleischmann 等人证明，当 w定义在二元字母表$\{a, b\}$ 上时，存在一种策略，能够确保最多只需 $\min(|w|_a, |w|_b) + 1 \le \lfloor \frac{|w|}{2} \rfloor + 1$ 个问题（对于字母 α， $|w|_\alpha = \binom{w}{\alpha}$ 表示 α 在 w中的出现次数）。对于字母表为 $\{1, \ldots, k\}$ 上的任意词w，他们证明所需问题数的上界为$\sum_{i \in \{1,\ldots,k\}} |w|_i (k + 1 - i)$。我们的主要结果（定理 2.1 和推论 2.6）证明，问题数至多为 $\binom{k}{2}\left(7\left\lceil\sqrt{|w|\log_2(|w|)}\right\rceil + 4\right)$。对于任意固定的 k，随着词的长度趋于无穷，我们的上界在渐近意义上显著更优。特别地，对于二元词，他们的上界为 $\frac{|w|}{2} + 1$，而我们的上界为 $7\left\lceil\sqrt{|w|\log_2(|w|)}\right\rceil + 4$.。我们还将这一策略（定理 2.2）加以调整，给出一种算法；对于长度为 n 的均匀随机二元词，其期望运行时间为 $\mathcal{O}(\log_2 n)$。

回顾一下，前面讨论的游戏与另一个问题有关；这个问题似乎最早由 L. O. Kalashnik [5]提出：最小的 ℓ 是什么，使得我们能够根据所有长度为 ℓ 的词 u 对应的数值 $\binom{w}{u}$ 重构w？据我们所知，I. Krasikov 和 Y. Roditty 在 1997 [6] 中利用该问题与丢番图分析中的普鲁赫-塔里-埃斯科特问题之间的联系，得到了这个问题目前最好的上界 $\lfloor \frac{16}{7}\sqrt{|w|} \rfloor + 5$。此外，目前已知的最好下界 $3^{(\sqrt{2/3} - o(1))\log_3^{1/2}(|w|)}$ 由 [1] 给出。我们的结果并未改进这个上界，因为在二元情形下，至少有一个查询涉及长度至少为 $\min(|w|_0, |w|_1)$ 的词 u ；对于许多词而言，这个长度约为 $|w|/2$ w。

在先前问题的一个变体中，形式为“ $\binom{w}{u}$的值是多少？”的查询被替换为形式为“ $\binom{w}{u} \ge 1$是吗？”或等价地“ u 是 w的子词吗？”的查询。更准确地说，该问题旨在确定最小的值 ℓ ，使得长度为 ℓ的子词集合能够唯一确定一个词 w。这一问题出现在多个领域中。在 [8, 第 6], 章中，证明了字母表 $\mathcal{A}$ 上的任意长度为 n 的词 w ，都可以由其长度至多为 $\lceil |w|_a + |w|_b + 1/2 \rceil$ 、形式为 $a^* b^*$ 的子词集合唯一确定，其中 a 和 b 是 $\mathcal{A}$中的不同字母。 [7]中也研究了这个问题。

在 DNA 杂交测序的背景下，S. S. Skiena 和 G. Sundaram 研究了如何最小化形如“ u w 的子词吗？”的查询数量。他们证明，可以通过 $O(n\log_2(k) + k\log_2(n))$ 次此类查询，重构字母表基数为 k 、长度为 n 的词 w 。更准确地说， [10] 中的定理15指出，只需 $1.59n\log_2(k) + 2k\log_2(n) + 5k$ 次查询即可重构 w。S. S. Skiena 和 G. Sundaram 还运用基本的信息论方法，给出了查询数量的下界 $n\log_2 k$ 。在第3节中，我们略微改进了 S. S. Skiena 和 G. Sundaram 的策略，并给出了一个新的上界，从而缩小了与下界之间的差距。更准确地说，

我们指出，至多需要 $n\log_2(k) + k(2 + \lfloor \log_2(n+1) \rfloor)$ 次查询即可重构 w，从而将上下界之间的差距从 $0.59n\log_2(k) + O(k\log_2(n))$ 缩小到 $O(k\log_2(n))$。

在第4节中，我们考虑因子而非子词（如果存在词 p 和 s，使得 $w = pus$，那么词 u 就是词 w 的因子），以及相应的问题：为了重构一个未知词 w，最少需要进行多少次“ u 是 w 的因子吗？”形式的查询。在 [9, 10], 中，S. S. Skiena 和 G. Sundaram 证明，对于字母表 $\mathcal{A}$ 的基数为 k、长度 n 已知的未知词 w，可以用数量为 $(k-1)n + 2\log_2(n) + O(k)$ 的查询重构 w。实际上，他们的证明给出的上界为 $(k-1)n + \log_2(n) + O(k)$；在二元情形下，该上界为 $n + \log_2(n) + O(1)$。这一更精确的上界此前已在 [10] 中针对二元情形提及。一个简单的双重计数论证（长度为 n 的词有 k^n 个，且每个问题有两种可能的答案）给出了下界 $n \log_2 k$。我们改进了他们的策略，将上界降至 $(k-1)(n+2) + \left\lceil \log_2 \binom{n}{2} \right\rceil + 3$。在二元情形下，这将下界与上界之间的差距从 $\log_2(n) + O(1)$ 缩小至 $\left\lceil \log_2 \binom{n}{2} \right\rceil + 5$。

S.S. Skiena 和 G. Subraman 等人 [10] 也研究过“词 u 作为词 w 的因子出现多少次”这一形式的查询。据我们所知，他们给出的所需查询次数下界 $nk/4 - o(n)$ 是目前已知的最佳结果。根据词 u 在 w 中的出现次数，可以推断词 u 是否作为因子出现在词 w 中。这一观察使他们能够为这个第四个问题得出与前一个问题相同的上界。同样，我们的界也适用。因此，我们也略微改进了这种情况下的上界，但与下界和上界之间的差距相比，这一改进微不足道。

基本定义和记号已在前文回顾过（遵循 [8]）。注意，$\#S$ 表示集合 S 的基数。此外，给定字母表 $\mathcal{A}$ 上的词 w，我们将简记 n 表示 w 的长度 $|w|$，并用 k 表示 $\mathcal{A}$ 的基数 $\#\mathcal{A}$。

2 子词计数查询

本节将重点讨论以下形式的查询：“w 包含多少次作为子词出现的 u？”或等价地：“ $\binom{w}{u}$的值是多少？”我们将此类查询称为 #-子串查询。关于这类查询，我们的主要结果如下。当然，与下一节中的其他查询一样，我们假设无需知道 w 即可回答此类查询。

定理 2.1. 重构一个长度为 n、取自 $\{0,1\}$ 的词，至多需要 $7\lceil \sqrt{n} \log\ n \rceil + 4$ 次 #-子串查询，无论是否已知 n。

一个包含 m 个 1 的词 w 总可以写成 $w = 0^{s_0}10^{s_1}1 \ldots 10^{s_m}$，其中 s_i 为非负整数。由于 $m = \binom{w}{1}$，只需一次查询即可找到 m。我们的目标是找出所有 s_i 的值。我们的策略依赖于这样一个事实：如果我们知道哪些 s_i 是“大”的，并且知道它们的值，那么只需一次查询，就可以确定另外多个 s_i（这在引理 2.4 中有所证明）。另一方面，由于不可能有太多“大”的 s_i，我们便有一种高效的策略来找出所有这些 s_i（见引理 2.5）。结合这两个事实，并优化“大”的含义，即可得到所需结果。

实际上，对于均匀随机词，我们预计不会有任何 s_i 大于 $\mathcal{O}(\log n)$，这将带来一种更高效的平均情形算法。

定理 2.2。存在一种确定性策略，对于任意整数 n，能够通过平均 $\mathcal{O}(\log_2(n))$次查询，重构从长度为 n的所有二元词中均匀随机选取的任意词 w。

下面的引理可用于证明引理 2.4。

引理 2.3. 设 r、 ℓ、 s_1、...、 s_r 为非负整数，使得 $1 \le r \le \ell+1$，且对所有 $j \in \{1,\dots,r\}$，$s_j < \frac{\ell+1}{r}$。 s_1，...， s_r 的值由 $\binom{0^{s_r}10^{s_{r-1}}1\cdots 0^{s_2}10^{s_1}1^\ell}{01^\ell}$、 r 和 ℓ 的值唯一确定。

证明。首先，我们表示 01^ℓ 作为子词在 $0^{s_r}10^{s_{r-1}}1\cdots 0^{s_1}1^\ell$ 中出现的次数。分别考虑出现中 0 的各种可能位置，可得

$$\binom{0^{s_r}10^{s_{r-1}}1\cdots 0^{s_2}10^{s_1}1^\ell}{01^\ell} = \sum_{j=1}^{r} s_j \binom{\ell+j-1}{\ell} = \sum_{j=1}^{r} s_j \binom{\ell+j-1}{j-1}. \tag{1}$$

设 $\beta = \max_j s_j$。我们先证明，对所有 $t \in \{1,\dots,r\}$，

$$\sum_{j=1}^{t} s_j \binom{\ell+j-1}{j-1} \le \beta \binom{\ell+t}{t-1}. \tag{2}$$

我们对 t 使用归纳法。容易验证 $t=1$ 的情形。现在，若 (2) 对 t 成立，则

$$\begin{aligned}\sum_{j=1}^{t+1} s_j \binom{\ell+j-1}{j-1} &= \sum_{j=1}^{t} s_j \binom{\ell+j-1}{j-1} + s_{t+1}\binom{\ell+t}{t} \le \beta\binom{\ell+t}{t-1} + s_{t+1}\binom{\ell+t}{t} \\ &\le \beta\left(\binom{\ell+t}{t-1} + \binom{\ell+t}{t}\right) = \beta\binom{\ell+t+1}{t}\end{aligned}$$

由此完成了 (2) 的归纳证明。

此外，对所有 $t \in \{1,\dots,r\}$, $\beta\binom{\ell+t}{t-1} < \frac{\ell+1}{r}\binom{\ell+t}{t-1} \le \frac{\ell+1}{t}\binom{\ell+t}{t-1} = \binom{\ell+t}{t}$。结合 (2)，可推出对所有 $t \in \{1,\dots,r\}$，

$$0 \le \sum_{j=1}^{t} s_j \binom{\ell+j-1}{j-1} < \binom{\ell+t}{t}. \tag{3}$$

注意，对于所有 $t \in \{1,\dots,r-1\}$，

$$s_{t+1} = \frac{\sum_{j=1}^{t+1} s_j\binom{\ell+j-1}{j-1} - \sum_{j=1}^{t} s_j\binom{\ell+j-1}{j-1}}{\binom{\ell+t}{t}}.$$

但 s_{t+1} 是一个整数，并且根据方程(3)，左边分数的右侧部分属于 $[0,1[$，由此我们可以推出

$$s_{t+1} = \left\lfloor \frac{\sum_{j=1}^{t+1} s_j\binom{\ell+j-1}{j-1}}{\binom{\ell+t}{t}} \right\rfloor. \tag{4}$$

根据方程 (1) 和 (4)，我们可以从 r、 l 和 $\sum_{j=1}^{r} s_j\binom{\ell+j-1}{j-1}$ 推出 s_r 的值，而这些值本身又是从 $\binom{0^{s_r}10^{s_{r-1}}1\cdots 0^{s_2}10^{s_1}1^\ell}{01^\ell}$ 推出的。根据 s_r 的值，我们现在可以推出 $\sum_{j=1}^{r-1} s_j\binom{\ell+j-1}{j-1}$，进而根据 (4) 推出 s_{r-1}。因此，通过从 $r-1$ 到 1 的“逆向归纳”，我们可以推出所有 s_j 的值。

□

引理 2.3 使我们能够在某些强假设下，仅通过一次查询确定多个连续 0-块的长度，但我们可以将这些假设放宽如下。思路是，如果我们有一些较大的 s_i 和一个前缀，那么只需知道这些 s_i 和前缀的值，就足以在应用前一个引理之前消除它们的贡献。

引理 2.4. 设 p和 v是词， r和 $s_1,\dots,s_r$是非负整数，使得 $1 \le r \le |v|_1 + 2$，并设 $w = p0^{s_r}10^{s_{r-1}}\dots 10^{s_1}1v$。假设已知 p, $|v|_1$和 r，且对所有j，要么已知s_j，要么 $s_j < \frac{|v|_1+2}{r}$，那么 $\binom{w}{01^{1+|v|_1}}$ 的值唯一确定所有未知的s_j在 $j \in \{1,\dots,r\}$时的值。

证明. 对所有 $j \in \{1,\dots,r\}$，设 s'_j 满足：若 $s_j < \frac{|v|_1+2}{r}$，则 $s'_j = s_j$ 且 $s'_j = 0$；否则。于是，对所有 j，$s_j - s'_j$ 都是已知的（若已知 s_j，则其值为 s_j；否则为 0），且对所有 j，$s'_j < \frac{|v|_1+2}{r}$。现在，考虑 $01^{1+|v|_1}$的出现中 0 可能出现的位置，可得

$$\begin{aligned}
\binom{w}{01^{1+|v|_1}} &= \binom{p1^{r+|v|_1}}{01^{1+|v|_1}} + \binom{0^{s_r}10^{s_{r-1}}\dots10^{s_1}1^{1+|v|_1}}{01^{1+|v|_1}} \\
&= \binom{p1^{r+|v|_1}}{01^{1+|v|_1}} + \sum_{j=1}^{r} s_j \binom{j+|v|_1}{1+|v|_1} \\
&= \binom{p1^{r+|v|_1}}{01^{1+|v|_1}} + \sum_{j=1}^{r} (s_j - s'_j)\binom{j+|v|_1}{1+|v|_1} + \sum_{j=1}^{r} s'_j \binom{j+|v|_1}{1+|v|_1} \\
&= \binom{p1^{r+|v|_1}}{01^{1+|v|_1}} + \sum_{j=1}^{r} (s_j - s'_j)\binom{j+|v|_1}{1+|v|_1} + \binom{0^{s'_r}10^{s'_{r-1}}\dots10^{s'_1}1^{1+|v|_1}}{01^{1+|v|_1}}.
\end{aligned}$$

这意味着，

$$\binom{0^{s'_r}10^{s'_{r-1}}\dots10^{s'_1}1^{1+|v|_1}}{01^{1+|v|_1}} = \binom{w}{01^{1+|v|_1}} - \binom{p1^{r+|v|_1}}{01^{1+|v|_1}} - \sum_{j=1}^{r}(s_j - s'_j)\binom{j+|v|_1}{1+|v|_1}.$$

根据假设，$\binom{w}{01^{1+|v|_1}}$, p, r, $|v|_1$并且对于所有 j，$(s_j - s'_j)$ 都是已知的。因此，量 $\binom{0^{s'_r}10^{s'_{r-1}}\dots10^{s'_1}1^{1+|v|_1}}{01^{1+|v|_1}}$ 是唯一确定的。对于所有 j, $s'_j < \frac{|v|_1+2}{r}$，由引理 2.3 可推知所有 s'_j 的值都是唯一确定的，从而完成证明。 □

对于任意由 $\{0,1\}$ 上的词 w 分解得到的 $w = 0^{s_0}10^{s_1}1\cdots0^{s_{t-1}}10^{s_t}$，我们称 i 为 0 块 0^{s_i} 的索引。若要使用前一个引理重构一个词，我们首先需要确定所有长度超过某个预设长度的 0 块的索引。

引理 2.5。设 $w \in \{0,1\}^*$ 和 m 是一个整数。设 I 为 w中长度至少为 m的 0-块的索引集合。假设我们已知 $|w|$ 和 $|w|_0$（因此也已知 $|w|_1 = |w| - |w|_0$），那么确定 I 所需的 #-子串查询次数至多为

$$\frac{2|w|_0\lceil\log_2(|w|_1+1)\rceil}{m}.$$

证明。我们使用算法 1 来确定 I，调用时使用 $\ell = 0$和 $u = |w|_1$。注意，$|w|_1 = |w| - |w|_0$ 是已知的。

算法 1：打印 w中出现的、长度至少为 m的 0-块的索引 $i \in \{\ell,\dots,u\}$ 的算法

```
过程 RecBlocks(w, m, ℓ, u)
    如果 (w choose 1^ℓ 0^m 1^{|w|_1 − u}) ≥ 1，则
        如果 u = ℓ 那么
            打印 ℓ
        else
            RecBlocks(w, m, ℓ, ⌊(ℓ+u)/2⌋)
            RecBlocks(w, m, ⌊(ℓ+u)/2⌋ + 1, u)
```

主“if”条件会验证索引属于$\{\ell,\dots,u\}$ 的 0-块的长度总和至少为 m。如果不满足这一条件，我们就知道这些块中没有任何一个的长度能达到 m，因此无需对其中任何一个递归调用该函数。由此，验证算法的正确性相当直接。

现在来界定查询总数。为此，我们考虑如下定义的对 Recblocks 的递归调用树：树的根是初始调用，参数为 $\ell = 0$ 和 $u = |w|_1$；如果调用 a 是在调用 b中进行的，那么调用 a 就是调用 b 的子节点。一次调用的深度是它到根节点的距离。一次调用的权重是量 $u + 1 - \ell$。对于任意权重为 x的调用，其子节点的权重为 $\lceil x/2 \rceil$ 或 $\lfloor x/2 \rfloor$（两个子节点的权重之和为 x）。设 f是满足 $f\colon x \to \lceil \frac{x}{2} \rceil$的函数。根节点的权重为 $|w|_1 + 1$，且 f是非递减函数，因此任何深度为 d的调用，其权重至多为 $f^d(|w|_1 + 1)$。对于任意整数x，$f(x) \le \frac{x+1}{2}$，特别地，对于所有 $d \ge 1$，$f^d(|w|_1 + 1) \le f^{d-1}(\frac{|w|_1+1}{2}) + 1$。对$d$使用归纳法可得，$f^d(|w|_1 + 1) < \frac{|w|_1+1}{2^d} + 1$。任何深度为 $\lceil \log_2(|w|_1 + 1) \rceil$ 的调用，其权重至多为 1（权重是小于 2 的整数），并且是树的叶节点。因此，任何调用的深度至多为$\lceil \log_2(|w|_1 + 1) \rceil$。

此外，通过对深度进行归纳法，很容易验证：对于同一深度上的任意两个不同调用 c and c'，相应的区间 $[\ell, u]$ and $[\ell', u']$ 不相交。我们称，取值为 ℓ 和 u 的调用拥有所有索引在 ℓ 和 u 之间的块中 0 的出现次数。由前面的观察可知，同一深度上两个调用所拥有的 0 的出现次数集合互不相交。由于当调用拥有至少 m 个 0 的出现次数时，第一个“if”条件为真，因此我们可以推断，在任意给定深度上，这样的调用至多有 $\frac{|w|_0}{m}$ 个。由于每个这样的调用都有两个子节点，我们可以推断，任意深度上的调用数至多为 $2\frac{|w|_0}{m}$。因此，调用总数至多为 $2^{\frac{2|w|_0}{m}\lceil \log_2 |w|_1+1 \rceil}$()。由于每次调用都会进行一次查询，证明得证。□

现在我们可以证明主要结果了。首先使用引理 2.5 中的算法找出所有长度为 $\lceil \sqrt{n} \log n \rceil$ 的块，然后使用引理 2.4 确定所有其他块。

定理 2.1 的证明。

阶段 1。设 w 为未知词。通过两次查询即可获得 $|w|_0 = \binom{w}{0}$ 和 $|w|_1 = \binom{w}{1}$。于是 $n = |w| = |w|_0 + |w|_1$ 已知。不失一般性，假设 $\binom{w}{0} \ge n/2 \ge \binom{w}{1}$（否则只需在下文中交换 0 和 1 的角色）。

阶段 2。令 $m = \lceil \sqrt{n} \log n \rceil$。我们使用引理 2.5 中的算法定位所有长度至少为 m的 0-块。这样的块至多有 $\frac{n}{m}$ 个，我们可以对每个块进行一次查询，以确定其长度：事实上，如果该块的索引为 i，且满足$i \in \{0, \ldots, |w|_1\}$，那么它的长度为 $\binom{w}{1^i 0 1^{|w|_1 - i}}$。因此，定位所有长度至少为 m 的 0-块并确定其长度，至多需要 $\frac{2|w|_0\lceil \log(\frac{|w|_1+1}{m}) \rceil}{m} + \frac{n}{m}$)次查询。查询次数少于 $3^{n \log_m n} \le 3\sqrt{n} \log n$。

阶段 3。现在我们需要确定长度不超过 m 的 0-块的长度。我们首先确定出现在最后 $\left\lceil \frac{|w|_1}{2} \right\rceil$ 次 1 之前的 0-块。其次，我们确定出现在最先 $\left\lceil \frac{|w|_1}{2} \right\rceil$ 次 1 之后的 0-块。此后，所有 0-块的长度都已知，我们也知道 w。我们只描述如何确定前半部分的块，因为可以通过对称方式重建后半部分的块。

在 1 的最后 $\left\lceil \frac{|w|_1}{2} \right\rceil$ 次出现之前有 $\left\lceil \frac{|w|_1}{2} \right\rceil + 1$ 0 个块。我们从左到右确定其中未知的块，最多需要 m 步，每一步至多考虑 $r = \left\lfloor \frac{|w|_1}{2m} \right\rfloor$ 个块。由于 $mr \ge \frac{|w|_1}{2} - m$，之后我们可能会漏掉多达 $m + 1$ 个块，而这些块可以通过至多 $m + 1$ 次额外查询逐个恢复。在某一步 $w = p0^{s_r}10^{s_{r-1}} \cdots 10^{s_1}1v$，已知前缀为 w（起初 p 是空单词），且 $|v|_1 \ge \left\lceil \frac{|w|_1}{2} \right\rceil$。对于每个 $i \in \{1, \ldots, r\}$，若 s_i 未知，则 $s_i < m = \frac{|w|_1/2}{|w|_1/2m}() < \frac{|v|_1+2}{r}$。根据引理 2.4，只需一次查询即可获知这些 r 个块。因此，我们至多通过 $2m + 1 = 1 + 2\lceil \sqrt{n} \log n \rceil$ 次查询，确定 1 的最后 $\left\lceil \frac{|w|_1}{2} \right\rceil$次出现之前的 0 块（类似地，也能获知

1 的最后 $\left\lceil \frac{|w|_1}{2} \right\rceil$ 次出现之后的 0 块）。

总之，我们的策略使用 $2 + 3\lceil \sqrt{n \log n} \rceil + 2(1 + 2\lceil \sqrt{n \log n} \rceil) = 7\lceil \sqrt{n \log n} \rceil + 4$。 □

对于任意字母表 $\mathcal{A}$ 和 $\mathcal{B} \subseteq \mathcal{A}$ 以及任意在 $\mathcal{A}$上的词 u，词 u 在 $\mathcal{B}$ 上的投影是从 u中删除所有不属于 $\mathcal{B}$的字母后得到的词。我们将其记为 $\pi_{\mathcal{B}}(u)$。例如，$\pi_{\{0,1\}}(0120201) = 01001$。对于基数为 k 的字母表，如果我们知道所有二元子字母表上的投影，就能唯一确定整个词 [8, 引理 6.2.19]。因此，定理 2.1 有如下推论。

推论 2.6。重构一个长度为 n、取自基数为 k 的字母表的词，最多需要 $\binom{k}{2}(7\lceil \sqrt{n \log n} \rceil + 4)$ 个 #- 子串查询。

在定理 2.1 和推论 2.6 中，我们没有尝试优化乘法常数，因为我们认为 $\sqrt{n \log n}$ 界并非“在乘法常数意义下紧确”。正如定理 2.2 所示，定理 1 和推论 6 中所需的查询次数可能为 $O(\log n)$。

正如我们将在引理 2.7 中看到的，存在长度大于$\lceil 2\log_2(n) \rceil$ 的 0 块的概率很小。

引理 2.7. 设 w 是从所有长度为 n 的二元词中均匀随机地选取的一个词。w 包含因子 $0^{\lceil 2\log_2(n) \rceil}$ 的概率至多为 $1/n$。

证明. 令 $m = \lceil 2\log_2(n) \rceil$。令 $w_1, \ldots, w_n \in \{0,1\}$ 满足 $w = w_1 \cdots w_n$。对于所有 $i \in \{1, \ldots, n-m+1\}$，令 E_i 表示事件 $w_i w_{i+1} \ldots w_{i+m-1} = 0^m$。那么对于所有 i，$\mathbb{P}(E_i) = 2^{-m} \leq 1/n^2$。由并集界，

$$\mathbb{P}(0^m \text{ is a factor of } w) = \mathbb{P}(\cup_{i=1}^{n-m+1} E_i) \leq \sum_{i=1}^{n-m+1} \mathbb{P}(E_i) \leq \frac{1}{n}$$

如所需。 □

定理 2.2 的证明。首先，我们用 w 次查询确定 0 和 1 的数量。令 $m = \lceil 2\log_2(n) \rceil$。我们先假设 w 中没有因子 0^m。现在可以像定理 2.1 的证明中阶段 3 那样应用引理 2.4，但令 $m = \lceil 2\log_2(n) \rceil$。现在我们得到一个候选词 w'，并且可以再提出一个问题 $\binom{w}{w'}$，以验证 $w = w'$ 是否成立（如果最初的假设不成立，情况可能并非如此）。这一切需要 $\mathcal{O}(\log_2(n))$ 次查询。

如果没有得到正确的词，我们就知道假设不成立，并使用定理 2.1，通过额外的 $\mathcal{O}(\sqrt{n}\log_2(n))$ 次查询找到 w。根据引理 2.7，这种情况发生的概率至多为 $1/n$，因此该过程的预期查询次数至多为 $\mathcal{O}(\log_2(n)) + \mathcal{O}(\sqrt{n}\log_2(n)/n) = \mathcal{O}(\log_2(n))$。 □

3 存在子词查询

本节中，我们重点讨论形式为“u 是 w 的子词吗？”或等价地“是$\binom{w}{u} \geq 1$吗？”的查询。我们称这类查询为 $\exists$ 子词查询。S. S. Skiena 和 G. Sundaram [9, 10] 使用 $1.59n\log_2(k) + 2k\log_2(n) + 5k$ 次查询，解决了通过 $\exists$ 子词查询重构长度未知、n 长的词 w，其中该词取自基数为 k 的字母表 $\mathcal{A}$。我们改进了该界的主要系数，将 1.59 降至 1；这是最优的（任何此类算法在最坏情况下至少需要 $n\log_2(k)$ 次查询 [9, 10]）。

定理 3.1。重构长度未知、由基数为 k 的字母表 $\mathcal{A}$ 上的字符构成的未知词 w，所需的 $\exists$ 子词查询数量至多为

$$n\lceil \log_2(k) \rceil + k\left(2 + \lfloor \log_2(n+1) \rfloor\right).$$

实际上，我们的方法与 [9, 10] 中使用的方法相似。我们的做法本质上是在字母表上应用二分法；但在根据较小字母表上的投影重构词时，我们改进了查询次数的上界。此外，对于较小的字母表，我们使用线性分解而非二元分解，以减少推断某些字母出现次数所需的查询数量。

为证明定理 3.1，我们使用下面两个引理。第一个引理讨论字母表仅含一个字母时的重构问题。第二个引理给出了根据在不相交字母表上的投影重构一个词所需查询次数的上界。

引理 3.2. 给定一个长度为 n、字母表为 $\mathcal{A}$ 的未知非空单词 w，以及一个字母 $\alpha \in \mathcal{A}$，可以通过以下方式确定 $|w|_\alpha$

- 至多使用 $2\lfloor 1 + \log_2(|w|_\alpha + 1) \rfloor$ ∃次子词查询（若 n 未知），并且
- 若 n 已知，则至多使用 $\lceil \log_2(n+1) \rceil$ ∃次子词查询。

这个引理的证明是一个简单的二分搜索。详细内容见附录 A。在下一个引理中，我们将说明如何根据一个词 w 在两个互不相交的互补字母表上的投影来重构该词。请注意，引理 [10, 14], 的结果与此几乎相同，只是查询次数为 $2.18|\pi_{\mathcal{B}}(w)| + |\pi_{\mathcal{C}}(w)| + 5$，而不是 $|\pi_{\mathcal{B}}(w)| + |\pi_{\mathcal{C}}(w)| + 1$。主要区别在于，组合这两个词时，我们不使用二分搜索，而是从左到右直接贪心地进行。这个引理几乎完全解释了我们相较于定理 [10, 2] 所取得的改进。

引理 3.3。设 w 是字母表 $\mathcal{A}$ 上长度为 n 的未知词。设 $\mathcal{B}$ 和 $\mathcal{C}$ 是两个不相交字母表，且 $\mathcal{A} = \mathcal{B} \cup \mathcal{C}$，则

1. 若已知两个投影 $\pi_{\mathcal{B}}(w)$ 和 $\pi_{\mathcal{C}}(w)$，则至多使用 $n-1$ ∃次子词查询即可重构词 w；2. 若已知词 $\pi_{\mathcal{B}}(w)$ 且 $\#\mathcal{C} = 1$，则至多使用 $n+1$ ∃次子词查询即可重构词 w。

可以注意到，在引理 3.3 的第 1 项中，无需进行任何查询便可确定 w 的长度，因为它等于 $|\pi_{\mathcal{B}}(w)| + |\pi_{\mathcal{C}}(w)|$。第 2 项则并非如此。在这两种情况下，证明中都没有直接用到长度。

对于任意词 $x = x_1 \cdots x_\ell \in \{0,1\}^\ell$ 和整数 i、$j \in \{1, \ldots, \ell\}$，当 $i \le j$ 时，令 $x[i \ldots j] = x_i x_{i+1} \cdots x_j$。推广来说，若 $i > j$（且 $i = |x| + 1$ 或 $j = 0$ 也可能成立），则 $x[i \ldots j]$ 是空单词。

引理 3.3 的证明。首先假设已知 $u = \pi_{\mathcal{B}}(w)$ 和 $v = \pi_{\mathcal{C}}(w)$。w 的第一个字母要么是 u_1，要么是 v_1。更确切地说，$u_1 v$ 是 w 的子词，当且仅当 u_1 是 w的第一个字母；否则，v_1 是 w 的第一个字母。因此，通过一次查询，我们就能确定 w的第一个字母，以及两个投影 $\pi_{\mathcal{B}}(w[2 \ldots n])$ 和 $\pi_{\mathcal{C}}(w[2 \ldots n])$。我们可以重复这一过程，每次新的查询后，都能得到 w 的下一个字母，以及 w 剩余部分在 $\mathcal{B}$和 $\mathcal{C}$上的两个投影。

因此，算法 2 可以根据 u和 v重构 w。在此算法中，i 和 j 分别存储 $\pi_{\mathcal{B}}(w[1 \ldots i+j])$ 和 $\pi_{\mathcal{C}}(w[1 \ldots i+j])$ 的连续长度：在每次 while 循环开始时，我们都知道 $p = w[1 \ldots i+j]$。

根据初步说明可以直接看出，在算法 $p = w$结束时，所提出的 ∃-子词查询数量至多为 $n-1$。

从现在起，假设我们只知道词 $\pi_{\mathcal{B}}(w)$，以及对于某个字母 a，$\mathcal{C} = \{a\}$ 这一事实。我们采用与前一种情况类似的策略，也就是说，尝试以贪心的方式在 $\pi_{\mathcal{B}}(w)$ 的字母之间插入字母 a 的出现次数。一旦确定了 $\pi_{\mathcal{B}}(w)$ 中所有字母的位置，就必须确定 w 末尾剩余的 a 的出现次数。这就引出了

算法 2 返回未知词的算法 w over $\mathcal{B}\cup\mathcal{C}$ with $\mathcal{B}\cap\mathcal{C}=\emptyset$ from $u=\pi_{\mathcal{B}}(w)$ 和 $v=\pi_{\mathcal{C}}(w)$

$p\leftarrow\varepsilon$；$i\leftarrow 0$；$j\leftarrow 0$
当 $i<|u|$ and $j<|v|$ do
  如果 $pu_{i+1}v[j+1..|v|]$ 是 w 的子词 then
    $p\leftarrow pu_{i+1}$；$i\leftarrow i+1$
  else
    $p\leftarrow pv_{j+1}$；$j\leftarrow j+1$
$p\leftarrow pu[i+1..|u|]v[j+1..|v|]$
返回 p

变体算法 3，其所提出的 $\exists$-子词查询数量恰好为 $n+1$：针对 $\pi_{\mathcal{B}}(w)$ 和 $\pi_{\mathcal{C}}(w)$ 的每个字母各有一次查询，此外还需要一次额外查询，以确定 $\pi_{\mathcal{C}}(w)$ 中是否已没有字母。

□

算法 3 返回未知词 w，该词由 $u=\pi_{\mathcal{B}}$ 的 $a\notin\mathcal{B}$ 个子词组成，范围为 $\mathcal{B}\cup\{a\}$ (w)

$p\leftarrow\varepsilon$；$i\leftarrow 0$
当 $i<|u|$ 时执行
  如果 $pau[i+1..|u|]$ 是 w 的子词，则
    $p\leftarrow pa$
  else
    $p\leftarrow pu_{i+1}$；$i\leftarrow i+1$
当 pa 是 w 的子词时
  $p\leftarrow pa$
返回 p

下一个结果的证明说明了如何使用$\exists$-子词查询来解决重构问题。w 的长度可能未知。

命题 3.4. 设 w 是一个取自基数为 k的字母表的未知词。对于任意满足 $\#\mathcal{B}\geq 2$的 $\mathcal{B}\subseteq\mathcal{A}$，重构词 $\pi_{\mathcal{B}}(w)$所需的 $\exists$-子词查询次数至多为

$$\lceil\log_2(\#\mathcal{B})\rceil|\pi_{\mathcal{B}}(w)|+\#\mathcal{B}\left(2+\max_{\alpha\in\mathcal{B}}\lfloor\log_2(|w|_\alpha+1)\rfloor\right).$$

证明. 我们对 $\mathcal{B}$ 的基数使用归纳法，两个基本情形分别为 $\#\mathcal{B}=2$和 $\#\mathcal{B}=3$。

若 $\mathcal{B}=\{x,y\}\subseteq\mathcal{A}$ with $x\neq y$，我们可以应用引理 3.2，在至多 $2\lfloor 1+\log_2(|w|_x+1)\rfloor$ 次查询内确定 $\pi_{\{x\}}(w)=x^{|w|_x}$。引理 3.3 的情形 2 表明，随后我们可以再用至多 $|\pi_{\{x,y\}}(w)|+1$ 次额外查询确定$\pi_{\{x,y\}}(w)$。查询总数至多为

$$|\pi_{\{x,y\}}(w)|+1+2\lfloor 1+\log_2(|w|_x+1)\rfloor\leq\lceil\log_2(\#\mathcal{B})\rceil|\pi_{\mathcal{B}}(w)|+\#\mathcal{B}\left(2+\max_{\alpha\in\mathcal{B}}\lfloor\log_2(|w|_\alpha+1)\rfloor\right)$$

正如所需。

如果 $\mathcal{B}=\{x,y,z\}$ 对于某些互不相同的字母 x、y、$z\in\mathcal{A}$，我们就采用上一段的策略来确定 $\pi_{\{x,y\}}(w)$，并再次使用引理 3.3 的情形 2，以至多 $|\pi_{\{x,y,z\}}(w)|+1$ 次额外查询得到 $\pi_{\{x,y,z\}}(w)$。因此，查询总数至多为

$$|\pi_{\{x,y\}}(w)|+|\pi_{\mathcal{B}}(w)|+2+2\lfloor 1+\log_2(|w|_x+1)\rfloor\leq\lceil\log_2(\#\mathcal{B})\rceil|\pi_{\mathcal{B}}(w)|+\#\mathcal{B}\left(2+\max_{\alpha\in\mathcal{B}}\lfloor\log_2(|w|_\alpha+1)\rfloor\right)$$

如所求。

现在我们需要处理归纳法。假设 $\#\mathcal{B} \geq 4$。设 $\mathcal{C}$、$\mathcal{C}' \subseteq \mathcal{B}$ 是两个不相交的字母表，使得 $\mathcal{B} = \mathcal{C} \cup \mathcal{C}'$、$\#\mathcal{C} = \lfloor \frac{\#\mathcal{B}}{2} \rfloor$ 和 $\#\mathcal{C}' = \lceil \frac{\#\mathcal{B}}{2} \rceil$。最后两个条件意味着

$$\lceil \log_2 \#\mathcal{C} \rceil \leq \lceil \log_2 \#\mathcal{C}' \rceil = \lceil \log_2 \#\mathcal{B} \rceil - 1 .$$

归纳　假设下，确定 $\pi_{\mathcal{C}}(w)$ 和 $\pi_{\mathcal{C}'}(w)$ 所需的查询次数至多为

$$\begin{aligned}
&\lceil \log_2(\#\mathcal{C}) \rceil |\pi_{\mathcal{C}}(w)| + \#\mathcal{C} \left(2 + \max_{\alpha \in \mathcal{C}} \lfloor \log_2(|w|_\alpha + 1) \rfloor \right) \\
+ &\lceil \log_2(\#\mathcal{C}') \rceil |\pi_{\mathcal{C}'}(w)| + \#\mathcal{C}' \left(2 + \max_{\alpha \in \mathcal{C}'} \lfloor \log_2(|w|_\alpha + 1) \rfloor \right) \\
&\leq (\lceil \log_2(\#\mathcal{B}) \rceil - 1)(|\pi_{\mathcal{C}}(w)| + |\pi_{\mathcal{C}'}(w)|) + (\#\mathcal{C}' + \#\mathcal{C}) \left(2 + \max_{\alpha \in \mathcal{C}' \cup \mathcal{C}} \lfloor \log_2(|w|_\alpha + 1) \rfloor \right) \\
&\leq (\lceil \log_2(\#\mathcal{B}) \rceil - 1)(|\pi_{\mathcal{B}}(w)|) + \#\mathcal{B} \left(2 + \max_{\alpha \in \mathcal{B}} \lfloor \log_2(|w|_\alpha + 1) \rfloor \right) .
\end{aligned}$$

根据引理 3.3 的情形 1，我们只需额外进行 $|\pi_{\mathcal{B}}(w)|$ 次查询，就能确定 $\pi_{\mathcal{B}}(w)$。总共我们至多使用了 $\lceil \log_2(\#\mathcal{B}) \rceil (|\pi_{\mathcal{B}}(w)|) + \#\mathcal{B} \left(2 + \max_{\alpha \in \mathcal{B}} \lfloor \log_2(|w|_\alpha + 1) \rfloor \right)$ 次查询，正如所需。 □

定理 3.1 的证明。取 $\mathcal{B} = \mathcal{A}$，并使用 $\max_{\alpha \in \mathcal{B}} \lfloor \log_2(|w|_\alpha + 1) \rfloor \leq \lfloor \log_2(|w| + 1) \rfloor$，定理 3.1 立即可由命题 3.4 得出。 □

4 存在因子查询

本节中，我们着重讨论形式为“ u是 w的因子吗？”的查询。我们的目标是证明定理 16。至于我们在此改进的 [10] 中的结果，本节假设待判定的词的长度已知。

若存在一个字母 a，使得 ua 也是词 w 的因子，则称因子 u 在词 w 中可右延伸。词 ua 是 u 的右扩展。词 w 的不可右延伸因子 u 是 w 的后缀，但反过来并不成立。例如，词 $u = a$ 是词 $w = aa$ 的后缀，但它是可右延伸的。实际上，可以直接验证：因子 u 在 w 中不可右延伸，当且仅当 u 是 w 的后缀，且它作为 w 的因子只出现一次。可左扩展性和左扩展的概念定义类似。

使用因子查询重构未知词 w 的总体策略包括以下三个步骤。首先，我们找到一个由固定字母 α 组成的长块（引理 4.4 的证明）。其次，我们确定一个不可右延伸因子 w，它以这个长块 α 为前缀。引理 4.2 和引理 4.3 的证明中提出了两种不同的方法。根据先前找到的长块 α 的长度，其中一种方法会比另一种更高效。最后，我们根据先前找到的不可右延伸因子确定 w（引理 4.1）。首先来解释最后这一步。

引理 4.1。设 w 是字母表基数为k、长度 n已知的未知词。如果我们知道一个 w 的不可右延伸因子 s，那么最多只需进行 $(k-1)(n-|s|)$ ∃-因子查询，就能重构 w。

证明。假设 $|s| < n$。那么 s 是 w 的一个真后缀。固定一个字母 α。对于每个不同于 α 的字母 β，我们都可以询问“ βs 是 w 的一个因子吗？”。如果某个 β 的答案为是，那么我们就知道 βs 是 w 的一个不可右延伸因子；如果所有 β 的答案都为否，那么我们就知道 αs 是 w 的一个不可右延伸因子。然后重复相同的过程，直到得到一个长度为 n 的词（这个词必然是 w）。对于每个需要确定的字母，我们至多需要进行 $k-1$ 次查询，即需要进行 $(k-1)(n-|s|)$ 次查询。 □

现在我们说明如何高效地找到 w 的一个不可右延伸因子。为此，固定一个字母 α，并假设我们知道最大的 t，使得 α^t 在 w 中作为因子出现。我们将介绍两种不同的策略，并在定理 4.5 的证明中针对 t 的不同取值使用它们。第一种策略适用于 t 不太大的情况。它将在以下结果的证明中描述。

引理 4.2. 设 w 是一个长度已知为 n、取自基数为 k 的字母表 $\mathcal{A}$ 的未知词。设 $\alpha \in \mathcal{A}$。若已知满足 α^t 是 w 的因子的最大整数 t，则至多通过 $(k-1)(|s|+2)$ 个 $\exists$-因子查询，即可确定 w 的一个不可右延伸因子 s。

证明。设变量 σ 用于存放所要查找的 w 的不可右延伸因子。我们将 σ 初始化为词 α^t。我们通过对每个字母 $\beta \neq \alpha$ 查询“$\sigma\beta$ 是 w 的因子吗？”，来寻找 σ 的连续右扩展。若某个 $\beta \neq \alpha$ 对应的回答为“是”，则我们知道 $\sigma\beta$ 是 w 的因子，并将 $\sigma\beta$ 设为 σ 的新值。

若对所有 $\beta \neq \alpha$ 的回答都是“否”，则 $\sigma\alpha$ 要么是 w 的因子，要么 σ 是不可右延伸的。若 σ 不以后缀 α^t 结尾，我们就将 $\sigma\alpha$ 设为 σ 的新值。此时 σ 可能不再是 w 的因子（因此 σ 不是 w 的不可右延伸因子）；尤其是，当 σ 的前一个值本身已经是所查找的 w 的不可右延伸因子时，就会如此。但如果之后在尝试添加字母 $\beta \neq \alpha$ 时，我们得到“是”的回答，就能推断此前的每一个假设都是正确的。若连续 $t+1$ 次得到“否”的回答，则我们已在 σ 末尾添加了 α 的 $t+1$ 次出现。这意味着我们的判断有误，因为根据 t 的定义，α^{t+1} 不是 w 的因子。此时，对于某个以不同于 α 的字母结尾的词 v，有 $\sigma = v\alpha^{t+1}$，并且存在 $r \leq t$，使得 $v\alpha^r$ 是 w 的后缀，且 $v\alpha^{r+1}$ 以及所有满足 $\beta \neq \alpha$ 的词 $v\alpha^r\beta$ 都不是 w 的因子：$v\alpha^r$ 就是所查找的不可右延伸因子 w。我们可以通过从 $r = 0$ 开始，反复查询“$v\alpha^{r+1}$ 是 w 的因子吗？”，直到得到否定回答，来确定 r。

现在，我们给出查询次数的上界。令 $v\alpha^{t+1}$ 表示经过 $t+1$ 次连续的否定查询后得到的 σ 的值，令 $r+1$ 表示为确定 σ 的最终值 s 而进行的额外查询次数。注意，v 是通过 $(k-1)(|v|-t)$ 次查询确定的。接着，我们通过 $(k-1)(t+1)$ 次查询得到 $v\alpha^{t+1}$，最后再通过 $r+1$ 次查询确定最终值。因此，查询总次数的上界为 $(k-1)((|v|-t)+(t+1)+(r+1))$。由于 $|s| = |v| + r$，查询次数的上界为 $(k-1)(|s|+2)$。 □

我们通过一个例子来说明引理 4.2 的证明中所用的策略。假设要重构的词为 $w = 00011100111011$，并且我们使用 $\alpha = 1$。我们有 $t = 3$，且初始时 $\sigma = 111$。前两个查询的答案为肯定，因此得到 $\sigma = 11100$。接下来的三个查询答案为否定，我们据此假设 $\sigma = 11100111$ 是预期结果的一个前缀。下一个查询确认了这一点，并确定 $v = 111001110$。对 $v0$、$v10$、$v110$ 和 $v1110$ 进行的接下来四次否定查询意味着，不可右延伸因子是 v、$v1$、$v11$ 或 $v111$。再进行三次额外查询后，我们得知 **11100111011** 是 w 的一个不可右延伸因子（因此也是一个后缀）。

如果 t 较大（基本上是指 $t \geq \lceil 4\sqrt{n} \rceil$；参见定理 4.5 的证明），那么在构造不可右延伸因子时，更好的策略是更频繁地核实我们的假设是否正确。这样便得到下一条结果证明中给出的另一种策略。

引理 4.3. 设 w 是一个未知词，其长度 n 已知，且由基数为 k 的字母表 $\mathcal{A}$ 构成。设 $\alpha \in \mathcal{A}$ 是在 w 中至少出现一次的字母。假设我们知道 n，以及满足 α^t 是 w 的因子的最大正整数 t。可以使用至多 $(k-1)(|s|-t) + k\lceil\sqrt{n}\rceil + 1$ $\exists$-因子查询来确定 w 的一个不可右延伸因子 s。

证明。该策略与前一种几乎完全相同。我们用词 α^t 初始化 σ，并尝试通过查询 $\sigma\beta$ 是否是 w 的某个 $\beta \neq \alpha$ 因子来扩展它，然后按照之前的方法继续。

如果我们连续 r 次得到“否”的答案，那么我们就在 s 的末尾添加了 r 次 α 的出现。在这种情况下，每当 α 连续出现 $\lceil\sqrt{n}\rceil$ 次时，我们就验证当前的 σ 值是否是 w 的因子。如果成立，我们就继续。否则，令 v 为满足 $\sigma = v\alpha^{\lceil\sqrt{n}\rceil}$ 的词； $v\alpha^{\lceil\sqrt{n}\rceil}$ 不是 w 的因子。我们需要找到最大的 r，使得 $v\alpha^r$ 是 w 的因子。这可以通过设置 $\sigma = v$ 并查询“ $\sigma\alpha^i$ 是 w 的因子吗？”来完成，其中 i 从 1 开始递增，直到我们得到“否”的答案。

现在来计算查询次数。在第一阶段，直到达到 $v\alpha^{\lceil\sqrt{n}\rceil}$， σ 的长度从 t 增加到 $|v\alpha^{\lceil\sqrt{n}\rceil}|$。每个新字母最多需要 $k-1$ 次查询，但每进行 $\lceil\sqrt{n}\rceil$ 次查询，就会进行一次验证查询。因此，第一阶段的查询次数至多为（请记住 $t \geq 1$）

$$(k-1)(|v\alpha^{\lceil\sqrt{n}\rceil}| - t) + \left\lfloor \frac{|v\alpha^{\lceil\sqrt{n}\rceil}| - t}{\lceil\sqrt{n}\rceil} \right\rfloor \leq (k-1)(|v\alpha^{\lceil\sqrt{n}\rceil}| - t) + 1 + \left\lfloor \frac{|w|-1}{\lceil\sqrt{n}\rceil} \right\rfloor$$

其上界为 $(k-1)(|v| - t) + k\lceil\sqrt{n}\rceil$。

在第二阶段，会进行一次验证查询，其他每次查询都会使 i 的值从 1 增加到 $r+1$。因此，第二阶段至多还有 $1 + r = 1 + |s| - |v| \leq 1 + (k-1)(|s| - |v|)$次查询。将第一阶段和第二阶段的查询次数相加，可得总共至多使用 $(k-1)(|s| - t) + k\lceil\sqrt{n}\rceil + 1$ 次查询。 □

在使用引理 4.2 或引理 4.3 之前，我们需要确定词 w 中某个字母的最大幂。可以通过二分搜索来完成，查询形式为“ a^t 是否为 w 的因子？”，其中 $1 \leq t \leq n$。对查询“ a^1 是否为 w 的因子？”的否定回答表明，字母 a 不出现在 w 中。下一个结果适用于任意字母表。其证明说明了如何进行二分搜索。

引理 4.4。设 w 是一个未知词。设 a 是一个字母， x、 y 是两个已知整数， t 是满足 a^t 是 w 的因子的最大整数。若已知 $x \leq t \leq y$，则至多需要 $\lceil\log_2(y+1-x)\rceil$ 次因子查询来确定 t 的值。

同样，这个引理的思路是使用二分搜索，证明的详细内容见附录 B。

依次应用引理 4.4、引理 4.2 或引理 4.3，最后应用引理 4.1，即可得到下一个结果。

定理 4.5. 一个长度已知为 n、取自基数为 $k \geq 2$ 的字母表的未知非空单词 w，最多可通过 $(k-1)(n+2) + \lceil\frac{\log_2 n}{2}\rceil + 3$ 次因子查询重构。

证明。我们首先查询“ $\alpha^{\lceil 4\sqrt{n}\rceil}$ 是 w 的因子吗？”。

如果得到肯定回答，我们使用引理 4.4（其中 $x = \lceil 4\sqrt{n}\rceil$ 和 $y = n$ $(n \geq 1)$），在至多 $\lceil \log_2 n\rceil$ 次查询内计算满足 α^t 是 w 的因子的最大 t。然后应用引理 4.3，在至多 $(k-1)(|s|-t) + k\lceil\sqrt{n}\rceil + 1$ 次查询内找到一个不可右延伸因子 s。由于 $t \geq \lceil 4\sqrt{n}\rceil \geq 4\lceil\sqrt{n}\rceil - 3$,

$$(k-1)(|s| - t) + k\lceil\sqrt{n}\rceil + 1 \leq (k-1)(|s|+3) - (3k-4)\lceil\sqrt{n}\rceil + 1\,.$$

最后，我们应用引理 4.1，通过 $(k-1)(n-|s|)$ 次查询找到 w。在这种情况下，包括初始查询在内，我们最多需要总计 $(k-1)(n+3) + \lceil\log_2 n\rceil - (3k-4)\lceil\sqrt{n}\rceil + 2 \leq (k-1)(n+2)$ 次查询（此不等式中我们使用 $k \geq 2$ 和 $n \geq 1$）。

如果我们得到否定回答，就使用引理 4.4（结合 $x = 0$ and $y = \lceil 4\sqrt{n}\rceil - 1$），在至多 $\lceil\log_2(4\sqrt{n})\rceil = \lceil\frac{\log_2 n}{2}\rceil + 2$ 次查询内计算最大的 t，使得 α^t 是 w 的因子。然后，我们应用引理 4.2，在 $(k-1)(|s|+2)$ 次查询内找到一个不可右延伸因子 s；最后，应用引理 4.1，在 $(k-1)(n-|s|)$ 次查询内找到 w。在这种情况下，计入初始查询在内，我们总共需要 $(k-1)(n+2)$ 次查询。 □

5 结论

我们研究了三个重构问题，并分别改进了所需查询次数的上界。对于使用 ∃-子词查询来重构一个长度为 n 、取自基数为 k 的字母表的词 w ，我们得到下界 $n\log_2(k)$；在第3节中，我们将上下界之间的差距缩小到 $O(k\log_2(n))$。一个尚未解决的问题是，这一差距能否进一步缩小到 $O(k)$ 次查询，甚至更低。

对于第2节所讨论的使用 #-子串查询进行重构的问题，据我们所知，目前尚无已知的下界。我们的上界远低于此前的结果，但仍可能与真实情况相差甚远。具体而言，我们证明了存在一种确定性算法，平均需要 $O(\log n)$ 次查询，即可重构长度为 n 的均匀随机二元词；但在最坏情况下，该算法需要 $\Theta(\sqrt{n}\log\ n)$ 次查询。或~~许可以~~找到一种确定性算法，使其在最坏情况下只需 $O(\log n)$ 次查询。我们未能找到一个简单的证明，来说明无法做到只依赖字母表大小、且运行时间为常数的算法。

对于第4节所讨论的使用 ∃-因子查询进行重构的问题，简单的计数论证给出了查询次数的下界 $n\log_2(k)$。S. S. Skiena 和 G. Sundaram 在 [10] 中给出了一个以 $kn/4 - o(n)$ 次查询表示的下界，该下界在字母表较大时更优。在二元情形下，我们成功缩小了上下界之间的差距，将其降至 $\left\lceil \log_2\left(\frac{n}{2}\right)\right\rceil + 5$。在一般情形下，尽管我们的结果缩小了上下界之间的差距，但差距仍然显著。正如引言中所述，S. S. Skiena 和 G. Sundaram 给出的下界 $kn/4 - o(n)$，对于如下形式的查询同样成立：“ w 中作为因子出现的 u 有多少次？”从某种意义上说，与关于因子是否存在的信息相比，考虑因子的出现次数并未为重构带来显著的额外信息。这与子词情形形成对比：出现次数所提供的信息远多于是否存在相关出现这一信息。

最后，我们提及二元情形下存在一种确定性算法：对于均匀随机的词 [4]，该算法平均需要 $n + \mathcal{O}(1)$ ∃-因子查询，且在加法常数范围内是最优的。该算法的主要思想与第4节所用的方法类似，但确定最长的0块 t 的速度更快。事实上，对于长度为 n 且均匀随机选取的二元词，$|t - \log_2(n)|$ 的平均值为 $\mathcal{O}(1)$。至于在最坏情况下使用 $n + \mathcal{O}(1)$ 次 ∃-因子查询的确定性算法是否存在，目前仍是一个开放问题。

致谢

作者感谢 Victor Poupet 提供有益的讨论，也感谢各位审稿人认真审阅并提出宝贵建议。

参考文献

[1] M. Dudik 和 L.J. Schulman。从子序列重构。组合理论杂志 A 辑，103：337–348，2003。

[2] P. Fleischmann、M. Lejeune、F. Manea、D. Nowotka 和 M. Rigo。从右有界块词重构词。载于 N. Jonoska 和 D. Savchuk（编），《语言理论发展——第24届国际会议，DLT 2020》，2020年5月11日至15日，美国佛罗里达州坦帕，会议论文集，计算机科学讲义第12086卷，第96–109页。施普林格出版社，2020年。

[3] P. Fleischmann、M. Lejeune、F. Manea、D. Nowotka 和 M. Rigo。从右有界块词重构词。《国际计算机科学基础杂志》，32(6):619–640，2021年。[4] Kazuo Iwama、Junichi Teruyama 和 Shuntaro Tsuyama。从子串重构字符串：最优随机化与平均情况算法。arXiv 电子预印本，2018年。[5] L.O. Kalashnik。从片段重构一个词。载于《数值数学与计算机技术》，预印本 IV，第56–57页。乌克兰苏维埃社会主义共和国科学院数学研究所，1973年。[6] I. Krasikov 和 Y. Roditty。关于序列的一个重构问题。《组合理论杂志 A 辑》，77:344–348，1997年。[7] V. I. Levenshtein。从子序列或超序列高效重构序列。载于《组合理论杂志 A 辑》，第93卷，第310–332页，2001年。[8] M. Lothaire。《词组合学》，数学及其应用百科全书第17卷。Addison-Wesley，1983年。后收入剑桥数学丛书，剑桥大学出版社，英国，1997年。[9] S. Skiena 和 G. Sundaram。从子串重构字符串（扩展摘要）。载于 F. Dehne、J.-R. Sack、N. Santoro 和 S. Whitesides（编），第三届算法与数据结构研讨会论文集（WADS ’93），加拿大蒙特利尔，1993年8月11日至13日，计算机科学讲义第709卷，第565–576页。施普林格出版社，柏林，1993年。[10] S.S. Skiena 和 G. Sundaram。从子串重构字符串。《计算生物学杂志》，2(2):333–353，1995年。

引理 3.2 的证明

引理 3.2 的证明。首先假设 n 未知。我们先找出 M 大于 $|w|_\alpha$的最小 2 的幂。可以通过询问 α^i 是否是 w 从 $i=1$开始的子词来完成；若答案为肯定，则将 i 加倍。答案为否时，便达到上界 $M=i$。

如果 $M=1$，那么 $|w|_\alpha=0$，且恰好询问了一次（并且 $1\le 2\lfloor 1+\log_2(|w|_\alpha+1)\rfloor$，正如所需）。否则，可在 $\lfloor$次查询中找到 $M=2^{\lfloor\log_2|w|_\alpha\rfloor+1}$，需要 $_2|w|_\alpha\rfloor+2\log$ 次查询。此时我们知道 $M/2\le|w|_\alpha<M$，并且可以通过二分搜索找到 $|w|_\alpha$ 的值。区间 $\{M/2,\dots,M-1\}$ 包含 $2^{\lfloor\log_2|w|_\alpha\rfloor}$ 个值，因此二分搜索需要 $\lfloor\log_2|w|_\alpha\rfloor$ ∃次子词查询。在整个过程中，$|w|_\alpha$ 可以通过 $2\lfloor 1+\log_2|w|_\alpha\rfloor\le 2\lfloor 1+\log_2(|w|_\alpha+1)\rfloor$ ∃次子词查询找到，正如所需。

已知 n 时，n 是 $|w|_\alpha$ 的上界，因此可以在区间 $[0,n]$内进行二分搜索。由此，最多使用 $\lceil\log_2(n+1)\rceil$ ∃次子词查询即可确定 $|w|_\alpha$。 □

B 引理 4.4 的证明

引理 4.4 的证明。我们对 $y+1-x$的值进行归纳。若 $x=y$，则我们知道 t 的值，无需再进行查询，这正如预期。若 $y>x$，则我们提出查询：“$a^{\lceil(x+y)/2\rceil}$ 是 w的因子吗？”

我们推得 $x'\le t\le y'$；其中，若答案为“是”，则 $x'=\lceil(x+y)/2\rceil$ 且 $y'=y$；若答案为“否”，则 $x'=x$ 且 $y'=\lceil(x+y)/2\rceil-1$。在这两种情况下，

$$y'-x'+1\le 1+\lfloor(y-x)/2\rfloor\,. \tag{5}$$

映射 $f\colon z\mapsto\lfloor\frac{z-1}{2}\rfloor+1$ 在非负实数上是非递减的，并且对所有整数n，$f(2^n)=2^{n-1}$，因此对所有 $z\le 2^n$，都有 $f(z)\le 2^{n-1}$。由于 (5) 可以改写为$y'-x'+1\le f(y+1-x)$，我们推得，对所有整数 n，若 $y+1-x\le 2^n$则 $y'+1-x'\le 2^{n-1}$。特别地，取 $n=\lceil\log_2(y+1-x)\rceil$，可得 $y'+1-x'\le 2^{\lceil\log_2(y+1-x)\rceil-1}$，因此

$$\lceil\log_2(y'+1-x')\rceil\le\lceil\log_2(y+1-x)\rceil-1\,.$$

根据归纳假设，我们至多还需要 $\lceil\log_2(y+1-x)\rceil-1$ 次查询来确定 t 的值。加上初始查询，总共至多需要 $\lceil\log_2(y+1-x)\rceil$次查询，正如所需。

□