Longest Unbordered Factors on Run-Length Encoded Strings

Shoma Sekizaki[1] and Takuya Mieno[1]

University of Electro-Communications, Chofu, Japan
s2431091@edu.cc.uec.ac.jp, tmieno@uec.ac.jp

Abstract. A border of a string is a non-empty proper prefix of the string that is also a suffix. A string is unbordered if it has no border. The longest unbordered factor is a fundamental notion in stringology, closely related to string periodicity. This paper addresses the longest unbordered factor problem: given a string of length n, the goal is to compute its longest factor that is unbordered. While recent work has achieved subquadratic and near-linear time algorithms for this problem, the best known worst-case time complexity remains $O(n \log n)$ [Kociumaka et al., ISAAC 2018]. In this paper, we investigate the problem in the context of compressed string processing, particularly focusing on run-length encoded (RLE) strings. We first present a simple yet crucial structural observation relating unbordered factors and RLE-compressed strings. Building on this, we propose an algorithm that solves the problem in $O(m^{1.5} \log^2 m)$ time and $O(m \log^2 m)$ space, where m is the size of the RLE-compressed input string. To achieve this, our approach simulates a key idea from the $O(n^{1.5})$-time algorithm by [Gawrychowski et al., SPIRE 2015], adapting it to the RLE setting through new combinatorial insights. When the RLE size m is sufficiently small compared to n, our algorithm may show linear-time behavior in n, potentially leading to improved performance over existing methods in such cases.

Keywords: string algorithms · unbordered factors · run-length encoding

1 Introduction

A non-empty string b is called a *border* of another string T if b is both a prefix and a suffix of T. A string is said to be *bordered* if it has a border, and *unbordered* otherwise. Unbordered factors are known to have a deep connection with the smallest period of the string. The length of a string uv is called a *period* of a string T if $T = (uv)^k u$ for some strings u, v and an integer $k \geq 1$. The concept of string periodicity is fundamental and has applications in various areas of string processing, including pattern matching, text compression, and sequence assembly in bioinformatics [16,5,20].

In 1979, Ehrenfeucht and Silberger [9] posed the problem of determining the conditions under which $\tau(T) = \pi(T)$ holds for a string T of length n, where $\tau(T)$ denotes the length of the longest unbordered factor of T and $\pi(T)$ denotes the smallest period of T. They further conjectured that $\tau(T) \leq n/2$ implies $\tau(T) = \pi(T)$. However, this conjecture was disproved by Assous and Pouzet [2], who provided a counterexample. Subsequently, some progress was made toward the conjecture [6,21,7,12,14,13]. Finally, in 2012, Holub and Nowotka [15] solved this longstanding open problem, showing that $\tau(T) = \pi(T)$ holds if $\tau(T) \leq 3n/7$, and that this bound is tight due to the counterexample of [2].

This result led to increased research activity on algorithms for computing the longest unbordered factor [8,19,11,17]. As a special case, when a string of length n is periodic (i.e., its smallest period is at most $n/2$), its longest unbordered factor can be computed in $O(n)$ time [8]. Unfortunately, since many strings are non-periodic on average [19], this linear-time approach has limited applicability. For the general case, a straightforward $O(n^2)$-time algorithm can be designed by constructing *border arrays* [16], which can be computed in linear time, for all suffixes of a string T of length n. The resulting n border arrays indicate whether each factor of T is bordered or unbordered. The first non-trivial algorithm for computing the longest unbordered factor was proposed by Loptev et al. [19], who presented an algorithm with average-case running time $O(n^2/\sigma^4)$, where σ is the alphabet size. Furthermore, Cording et al. [4] proved that the expected length of the longest unbordered factor of a random string is $n - O(\sigma^{-1})$, and used this result to propose an average-case $O(n)$-time algorithm. In terms of worst-case time complexity, all of the above are quadratic-time algorithms. The first worst-case subquadratic-time algorithm was given by Gawrychowski et al. [11], whose algorithm runs in $O(n\sqrt{n})$ time. We will later review the basic strategy of their algorithm, which we simulate in our approach. The state-of-the-art algorithm for this problem is an $O(n \log n)$-time algorithm proposed by Kociumaka et al. [17,18], which exploits combinatorial properties of unbordered factors and sophisticated data structures, including the *prefix-suffix query* (PSQ) data structure. As of 2018 [17], their algorithm was reported to run in $O(n \log n \log^2 \log n)$ time in the worst case due to the cost of constructing the PSQ data structures. Later improvements [18] sped up the construction of the PSQ data structure to linear time, bringing the overall algorithm down to $O(n \log n)$ time. Whether the longest unbordered factor can be computed in $O(n)$ time remains open.

In this paper, instead of directly aiming to an $O(n)$-time algorithm, we propose an efficient solution in the context of *compressed string processing*. Especially, this work focuses on the *run-length encoding* (RLE) of a string. We first give a simple but important relationship between unbordered strings and RLE strings (Lemma 2). Using this relationship, we propose an RLE-based algorithm for computing all longest unbordered factors that runs in $O(m\sqrt{m} \log^2 m)$ time and uses $O(m \log^2 m)$ space, where m is the size of the RLE-compressed string. When m is sufficiently small (e.g., $m < n^{2/3-\varepsilon}$ for a small constant $\varepsilon > 0$), our approach achieves $O(n)$ time, thus improving the worst-case complexity over existing methods for such cases. On the one hand, the high-level idea of our approach is inspired by the algorithm of Gawrychowski et al. [11], which achieves subquadratic time via a non-trivial combination of fundamental string data structures and combinatorial techniques on strings. On the other hand, our algorithm differs in details and require techniques specially tailored to unbordered factors in RLE strings, particularly in Sections 4.2 and 4.3.

Several proofs are omitted due to space limitations. All the omitted proofs are provided in Appendix A.

2 Preliminaries

Let Σ be an alphabet. An element in Σ is called a character. An element in $\Sigma^\star$ is called a string. The empty string is the string of length 0, which is denoted by ε. A string in which all characters are identical is called a unary string. The concatenation of strings S and T is written as $S \cdot T$, or simply ST when there is no confusion. Let T be a non-empty string. If $T = X \cdot Y \cdot Z$ holds for some strings X, Y, and Z, then X, Y, and Z are called a prefix, a factor, and a suffix of T, respectively. Further, they are called a proper prefix, a proper factor, and a proper suffix of T if $X \neq T$, $Y \neq T$, and $Z \neq T$, respectively. A non-empty string B is called a border of T if B is both a proper prefix and a proper suffix of T. We call the occurrence of B as a prefix (resp. suffix) of T the prefix-occurrence (resp. suffix-occurrence) of border B. We call the longest border of T *the* border of T. A string T is said to be bordered If T has a border, and is said to be unbordered otherwise. We denote by $|T|$ the length of T. For an integer i with $1 \leq i \leq |T|$, we denote by $T[i]$ the ith character of T. For integers i, j with $1 \leq i \leq j \leq |T|$, we denote by $T[i..j]$ the factor of T that starts at position i and ends at position j. For strings S, T, we denote by $lcp(S, T)$ the length of the longest common prefix (in short, lcp) of S and T. An integer p with $1 \leq p \leq |T|$ is called a period of T if $T[i] = T[i + p]$ holds for all i with $1 \leq i \leq |T| - p$. We call the smallest period of T *the* period of T. The *border array* Bord_T of a string T is an array of length $|T|$, where $\mathsf{Bord}_T[i]$ stores the length of the border of $T[1..i]$ for each $1 \leq i \leq |T|$ [16]. The *border-group array* BG_T of a string T is an array of length $|T|$ such that, for each $1 \leq i \leq |T|$, $\mathsf{BG}_T[i]$ stores the length of the shortest border of $T[1..i]$ whose smallest period equals that of $T[1..i]$ if such a border exists, and $\mathsf{BG}_T[i] = i$ otherwise [22]. It is known that the border array and the border-group array of a string T can be computed by $O(|T|)$ character comparisons [16,22].

The *range maximum query* (*RMQ*) over an integer array A of length N is, given a query range $[i, j] \subseteq [1, N]$, to output a position p such that $A[p]$ is a maximum value among sub-array $A[i..j]$. The *range minimum query* (*RmQ*) is defined analogously. The following result is known.

Lemma 1 (e.g., [3]). *There is a data structure of size $O(N)$ that can answer any RMQ (and RmQ) over an integer array of length N in $O(1)$ time. The data structure can be constructed in $O(N)$ time.*

In what follows, we fix an arbitrarily *non-unary* string T of length n for our purpose. This is because the longest unbordered factor of a unary string $\mathtt{a} \cdots \mathtt{a}$ is simply $\mathtt{a}$.

3 Tools for RLE strings

This section provides some tools for RLE strings that are commonly used in Section 4.

3.1 Run-Length Encoding; RLE

The Run-Length Encoding (RLE) of a string T, denoted by $\mathrm{rle}(T)$, is a compressed representation of T that encodes every maximal character run $T[i..i+e-1]$ in T by c^e if (1) $T[j]=c$ for all j with $i \le j \le i+e-1$, (2) $i=1$ or $T[i-1] \neq c$, and (3) $i+e-1=n$ or $T[i+e] \neq c$. We simply call a maximal character run in T a run in T. Also, we call the number e of characters c in a run c^e the exponent of the run. The RLE size of string T, denoted by $r(T)$, is the number of runs in T. For each i with $1 \le i \le r(T)$, we denote by R_i the ith run of $\mathrm{rle}(T)$. Also, we denote by beg_i (resp., end_i) the beginning (resp., the ending) position of R_i, and by exp_i the exponent of R_i. A factor of T is said to be *RLE-bounded* if the factor starts at beg_i and ends at end_j for some i, j with $i \le j$. In what follows, we use m to denote the RLE size of the given string T.

Example 1. The RLE of string $T = \mathtt{aaabbcccccabbbb}$ is $\mathtt{a}^3\mathtt{b}^2\mathtt{c}^5\mathtt{a}^1\mathtt{b}^4$. The exponents of the first two runs $\mathtt{a}^3$ and $\mathtt{b}^2$ are three and two, respectively. The factor $T[4..11] = \mathtt{bbccccca}$ of T is RLE-bounded since it starts at $\mathsf{beg}_2 = 4$ and ends at $\mathsf{end}_4 = 11$. The RLE size of T is 5.

The following lemma establishes a significant connection between unbordered strings and RLE strings.

Lemma 2. *Let u be a string of length at least two, and let $a = u[1]$ and $b = u[|u|]$. If u is unbordered, then both au and ub are also unbordered.*

Proof. For the sake of contradiction, assume that au is bordered. Let k be the length of the border of au. If $k = 1$, then the border of au is a, which implies $a = b$, contradicting the assumption that u is unbordered. If $k > 1$, let x be the border of au. Then, $x[2..k]$ is a border of u, a contradiction. Therefore, au must be unbordered. The proof for ub is symmetric. $\square$

From Lemma 2, any longest unbordered factor of a non-unary string T must be RLE-bounded. Furthermore, the number of occurrences of longest unbordered factors is at most $m-1$ since any factor starting at the beginning position of the rightmost run is bordered. Also, the upper bound $m-1$ is tight: For string $(\mathtt{a}^e\mathtt{b}^e)^{\frac{m}{2}}$ where e is a positive integer, all the occurrences of $\mathtt{a}^e\mathtt{b}^e$ and $\mathtt{b}^e\mathtt{a}^e$ are the longest unbordered factors. Similarly, the next observation holds:

Observation 1 *Let w be a non-empty string. If $w \cdot w[|w|]$ has a border of RLE size p, then w also has a border of RLE size p.*

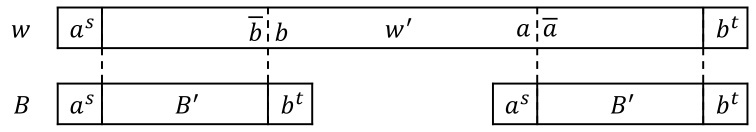

Fig. 1. Illustration for border $a^sB'b^t$ of w. String B' is an RLE-bounded border of w'.

3.2 RLE shortest border array

We define the *RLE shortest border array* rSBord of T as follows: For each i with $1 \leq i \leq m$, $\mathsf{rSBord}[i]$ stores the RLE size of the *shortest* border of the prefix $T[1..\mathsf{end}_i]$ of T. For example, when $T =$ aaabbbbaaaaaccaaaabbbaa, $\mathsf{rSBord} = [1, 0, 1, 0, 1, 2, 1]$.

To design an efficient algorithm for computing rSBord, we give an observation. Let w be a non-unary string, and let a^s and b^t be the first and the last run of w, respectively. Further let w' be the factor of w such that $w = a^sw'b^t$. If w has a border B with $r(B) \geq 3$, then $B = a^sB'b^t$ holds for non-empty string B' that is a border of w'. Also, the occurrences of B' in w' as a suffix and as a prefix are RLE-bounded (see Fig. 1). Namely, each border B of w with $r(B) \geq 3$ can be obtained from some RLE-bounded border B' of w' by prepending a^s and appending b^t to B'.

From the above observation, we can compute rSBord as follows: for each $i \leq m$, check whether each RLE-bounded border of a string $R_2 \cdots R_{i-1}$ can be extended to the left by R_1 and to the right by R_i. A naïve implementation of this algorithm runs in $O(m^2)$ time because all RLE-bounded borders of all prefixes of $T' = R_2 \cdots R_{m-1}$ can be computed by considering $\mathrm{rle}(T')$ as a string of length $m-2$ over the alphabet $\Sigma \times \mathbb{N}$ and constructing the border array of $\mathrm{rle}(T')$. To speed up, we make use of the following well-known fact about the periodicity of borders:

Lemma 3 ([16]). *The set of borders of a string w can be partitioned into $O(\log |w|)$ groups according to their smallest periods.*

Within such a group of borders, the characters that follow prefix-occurrences of all borders except the longest one must be identical due to periodicity. Thus, at most two distinct characters can follow prefix-occurrences of borders within the group. The same holds for the number of distinct characters preceding suffix-occurrences of borders within a group. Finally, by utilizing the border array and the border-group array of $\mathrm{rle}(T')$, we can compute rSBord in $O(m \log m)$ time:

Lemma 4. *Given $\mathrm{rle}(T)$, the RLE shortest border array rSBord of T can be computed in $O(m \log m)$ time.*

Proof. Let $T' = R_2 \cdots R_{m-1}$. We first construct the border array $\mathsf{Bord}_{\mathrm{rle}(T')}$ and the border-group array $\mathsf{BG}_{\mathrm{rle}(T')}$ of $\mathrm{rle}(T')$, considering $\mathrm{rle}(T')$ as a string of length $m-2$ over $\Sigma \times \mathbb{N}$. For each j with $1 \leq j \leq m-2$, we scan the borders of $\mathrm{rle}(T')[1..j] = \mathrm{rle}(T)[2..j+1]$ in decreasing order of their lengths, skipping some of them and processing the rest as follows: We check whether the current border B of $\mathrm{rle}(T')$ can be extended to the left by R_1 and to the right by R_{j+2}, by examining the runs immediately after the prefix-occurrence and before the suffix-occurrence of B in $\mathrm{rle}(T')[1..j]$. If the group to which B belongs has at least two borders, we perform the same procedure for the shortest border in the group. We then find the next group using the border-group array $\mathsf{BG}_{\mathrm{rle}(T')}$, update B to the longest border in that group, and repeat the above procedure. The time required to process each group is $O(1)$, and there are $O(\log m)$ groups for each prefix of $\mathrm{rle}(T')$ by Lemma 3. Therefore, the total running time is $O(m \log m)$. □

3.3 Some functions for RLE strings

Given $\mathrm{rle}(T)$, we construct array ExpSum of size m such that $\mathsf{ExpSum}[j] = \sum_{k=1}^{j} \mathsf{exp}_k$ for each $1 \leq j \leq m$. Then, given a text position i with $1 \leq i \leq n$, we can compute the run to which the ith character $T[i]$ belongs in $O(\log m)$ time by performing binary search on ExpSum. Tamakoshi et al. [23] proposed an $O(m)$-space data structure based on RLE, called a *truncated RLE suffix array* (tRLESA). They showed that tRLESA enhanced with some additional information of size $O(m)$ supports several standard string queries, such as pattern matching. By applying tRLESA and related data structures in conjunction with RMQ and/or RmQ (Lemma 1), several additional queries can be efficiently supported, as detailed below.

For positive integers i, p, j, q satisfying $i \leq m$, $p \leq \mathsf{exp}_i$, $j \leq m$, and $q \leq \mathsf{exp}_j$, let $\mathsf{rlcp}(i, p, j, q)$ denote the length of the longest common prefix of $T[\mathsf{beg}_i + p - 1..n]$ and $T[\mathsf{beg}_j + q - 1..n]$.

Lemma 5. *After $O(m \log m)$-time and $O(m)$-space preprocessing for $\mathrm{rle}(T)$, the value $\mathsf{rlcp}(i, p, j, q)$ can be computed in $O(1)$ time for given integers i, p, j, and q.*

For positive integers x, y, h, ℓ with $h < x \leq y \leq m$, we define

$$\mathsf{ridx}_{x,y}(h, \ell) = \begin{cases} -1 & \text{if } lcp(T[\mathsf{end}_h..n], T[\mathsf{end}_z..\mathsf{end}_y]) > \ell \text{ for all } x \leq z \leq y, \\ \arg\max_{z: x \leq z \leq y} \{lcp(T[\mathsf{end}_h..n], T[\mathsf{end}_z..\mathsf{end}_y]) \leq \ell\} & \text{otherwise.} \end{cases}$$

In words, $\mathsf{ridx}_{x,y}(h, \ell)$ is the index $x \leq z \leq y$ of a run such that $T[\mathsf{end}_z..\mathsf{end}_y]$ has the longest lcp with $T[\mathsf{end}_h..n]$ where the lcp length is at most ℓ, if such z exists.

Lemma 6. *After $O(m \log m)$-time and $O(m)$-space preprocessing for $\mathrm{rle}(T)$ and integers x, y with $1 \leq x \leq y \leq m$, the value $\mathsf{ridx}_{x,y}(h, \ell)$ can be computed in $O(\log m)$ time for given integers h and ℓ with $h < x$.*

4 Algorithm for computing longest unbordered factors

In this section, we prove our main theorem:

Algorithm 1 Algorithm for computing the longest unbordered factor

```
Input: String $T$ of RLE size $m$.
Output: The set of longest unbordered factors of $T$.
 1: $LUB \leftarrow$ LONGEST-SHORT-UB($T$)          ▷ $LUB$: set of longest unbordered factors.
 2: $\ell^\star = \max\{|x| : x \in LUB\}$          ▷ $\ell^\star$: length of the longest unbordered factor.
 3: Preprocess for RM-LONG-BORDERED
 4: for $k \leftarrow 5$ to $\lceil\sqrt{m}\,\rceil$ do
 5:     Preprocess for CANDIDATE$_k$
 6:     $C \leftarrow \{\varepsilon\}$
 7:     for $i \leftarrow 1$ to $(k-4)\lfloor\sqrt{m}\rfloor$ do
 8:         $C \leftarrow C \cup \{$CANDIDATE$_k(i)\}$
 9:     end for
10:     $U \leftarrow$ RM-LONG-BORDERED$(k, C)$          ▷ All strings in $U$ are unbordered.
11:     $\ell \leftarrow \max\{|u| : u \in U\}$
12:     if $\ell < \ell^\star$ then
13:         continue          ▷ Do nothing and continue to the next stage.
14:     else if $\ell > \ell^\star$ then
15:         $LUB \leftarrow \emptyset$          ▷ Clear the current tentative solutions.
16:         $\ell^\star \leftarrow \ell$
17:     end if
18:     $LUB \leftarrow LUB \cup \{u \in U : |u| = \ell^\star\}$
19: end for
20: return $LUB$
```

Theorem 1. *Given an RLE encoded string* $\mathrm{rle}(T)$ *of RLE size* m*, we can compute the set of longest unbordered factors of* T *in* $O(m\sqrt{m}\log^2 m)$ *time using* $O(m\log^2 m)$ *space.*

The high-level strategy of our algorithm, presented in Algorithm 1, is essentially the same as that of Gawrychowski et al. [11]. We divide the input string T into $\lceil\sqrt{m}\,\rceil$ blocks $J_1, J_2, \ldots, J_{\lceil\sqrt{m}\,\rceil}$, where each block J_k is RLE-bounded and has RLE size $\lfloor\sqrt{m}\rfloor$ for every $1 \le k < \lceil\sqrt{m}\,\rceil$. Throughout this section, we refer to a border of RLE size at most $\sqrt{m}$ as a *short* border, and otherwise as a *long* border. Let ρ_T denote the RLE size of a longest unbordered factor of T. Also, we refer to a factor starting at $\mathtt{beg}_i$ and ending within the kth block J_k as an (i,k)-*factor*. Note that the longest unbordered factor of T must be an (i,k)-factor for some i and k since it is RLE-bounded (see Lemma 2).

Let us look at Algorithm 1. The set LUB represents the current tentative solution and the variable $\ell^\star$ represents the length of an element in LUB. First of all, we invoke the subroutine LONGEST-SHORT-UB, which outputs the longest unbordered factors of RLE size at most $4\sqrt{m}$, and tentatively update LUB and $\ell^\star$ (lines 1–2). The main part of our algorithm consists of $O(\sqrt{m})$ stages, corresponding to the outer **for** loop (lines 4–19). In each stage, say the kth stage, we first compute a set C of *candidates* for longest unbordered factors that end within J_k by calling the subroutine CANDIDATE$_k$ $O(m)$ times (lines 7–9). Here, as we will show later in Lemma 7, the subroutine CANDIDATE$_k(i)$ returns one of the following three strings: (1) the longest unbordered (i,k)-factor, if such a factor exists; (2) the empty string, if all (i, k)-factors have short borders; or (3) an (i, k)-factor that has no short border but has a long border, otherwise. Then, the set C is guaranteed to contain a longest unbordered factor of T if (i) $\rho_T > 4\sqrt{m}$ and (ii) there is a longest unbordered factor ending within J_k. We then eliminate from C all factors that have a long border by calling the subroutine RM-LONG-BORDERED(k, C), which checks whether each string in C has a long border and removes it if so (line 10). If any candidates remain, we select the longest ones and update $\ell^\star$ and LUB if necessary (lines 11–18). After the outer **for** loop, we have the final answer LUB, thus output it.

The correctness of Algorithm 1 is straightforward from the properties of the three subroutines LONGEST-SHORT-UB, CANDIDATE$_k$, and RM-LONG-BORDERED. In what follows, we describe how to implement the subroutines efficiently.

4.1 Implementation of LONGEST-SHORT-UB

To implement LONGEST-SHORT-UB, we simply apply Lemma 4 for all RLE-bounded factors of RLE size $4\sqrt{m}$. By doing so, we can compute the longest unbordered factors of RLE size at most $4\sqrt{m}$ in a total of $O(m\sqrt{m}\log m)$ time.

4.2 Implementation of CANDIDATE$_k$

Throughout this subsection, we fix an integer $5 \le k \le \lceil \sqrt{m} \rceil$ arbitrarily. The definition of function CANDIDATE$_k(i)$ is as follows: CANDIDATE$_k(i)$ returns the longest (i, k)-factor that has no short border, if it exist; or the empty string, otherwise. This definition leads to the following properties:

Lemma 7. *(1) If there is an unbordered (i, k)-factor, then* CANDIDATE$_k(i)$ *returns the longest unbordered (i, k)-factor. (2) If all (i, k)-factors have short borders, then* CANDIDATE$_k(i)$ *returns the empty string. (3) Otherwise,* CANDIDATE$_k(i)$ *returns an (i, k)-factor that has no short border but has a long border.*

Let $J_k.\mathsf{first}$ and $J_k.\mathsf{last}$ be the indices of runs such that $T[\mathsf{beg}_{J_k.\mathsf{first}}..\mathsf{end}_{J_k.\mathsf{last}}] = J_k$. Let $D_k = J_{k-1}J_k$, $x = J_{k-1}.\mathsf{first}$, $y = J_k.\mathsf{first}$, and $z = J_k.\mathsf{last}$. Namely, $T[\mathsf{beg}_x..\mathsf{end}_z] = D_k$ and $T[\mathsf{beg}_y..\mathsf{end}_z] = J_k$ hold. Let P_i be the longest prefix of $T[\mathsf{end}_i..\mathsf{end}_z]$ that occurs in D_k. If $P_i = \varepsilon$, then CANDIDATE$_k(i)$ returns $T[\mathsf{beg}_i..\mathsf{end}_z]$ since it has no short border. Otherwise, let $p = r(P_i)$. If $p = 1$, CANDIDATE$_k(i)$ can be easily computed by comparing the characters of the ith run and the zth run. Thus, we assume $p > 1$ in the following. Let $P_i = aub^{e_1}$ where $a = P_i[1]$, b^{e_1} is the last run of P_i, and $u \in \Sigma^\star$ is the rest. Further let Γ be the set of exponents of b following some occurrences of au in D_k. If $\min \Gamma > e_1$, the next character of aub^{e_1} in D_k is always b. Then, any factor starting at end_i and ending at $\mathsf{end}_{j'}$ for some $y \le j' \le z$ can not have a short border of RLE size exactly p because if such a border exists, the border forms aub^e with $e \le e_1$, contradicts that $\min \Gamma > e_1$. If $\min \Gamma \le e_1$, we define $e_2 = \max\{\gamma \in \Gamma \mid \gamma \le e_1\}$. Let end_t be the starting position of an occurrence of aub^{e_2} in D_k. We further define $F(t, j) = T[\mathsf{end}_t..\mathsf{end}_z]\$T[\mathsf{beg}_x..\mathsf{end}_{y+j-1}]$ for j with $1 \le j \le \sqrt{m}$, where \$ is a special character with $\$ \notin \Sigma$. The next lemma holds:

Lemma 8. *Assume* $\min \Gamma \leq e_1$ *holds. For each* $1 \leq j \leq \sqrt{m}$*,* $T[\mathsf{end}_i..\mathsf{end}_{y+j-1}]$ *has a short border of RLE size* p *if and only if* $F(t,j)$ *has a short border of RLE size* p*.*

Proof. Let $j' = y + j - 1$. ($\Rightarrow$) If $T[\mathsf{end}_i..\mathsf{end}_{j'}]$ has a short border of RLE size $p \leq \sqrt{m}$, the border forms $aub^{\mathsf{exp}_{j'}}$ and it holds that $\mathsf{exp}_{j'} \leq e_1$. Also, $\mathsf{exp}_{j'} \leq e_2$ holds since $e_2 < \mathsf{exp}_{j'} \leq e_1$ contradicts the definition of e_2. Therefore, $F(t,j)$ has border $aub^{\mathsf{exp}_{j'}}$ since $F(t,j)$ starts with aub^{e_2}. ($\Leftarrow$) If $F(t,j)$ has a short border of RLE size $p \leq \sqrt{m}$, the border forms $aub^{\mathsf{exp}_{j'}}$ and it holds that $\mathsf{exp}_{j'} \leq e_2$. Also, $\mathsf{exp}_{j'} \leq e_2 \leq e_1$ holds by the definition of e_2. Therefore, $T[\mathsf{end}_i..\mathsf{end}_{j'}]$ has border $aub^{\mathsf{exp}_{j'}}$ since $T[\mathsf{end}_i..\mathsf{end}_{j'}]$ starts with aub^{e_1}. □

Let $t^\star = t$ if $\min \Gamma \leq e_1$; otherwise, let $t^\star$ be the starting position of an occurrence of aub in D_k. Regardless of the value of $\min \Gamma$, for each $1 \leq j \leq \sqrt{m}$, the set of short borders of $T[\mathsf{end}_i..\mathsf{end}_{y+j-1}]$ of RLE size q is equivalent to that of $F(t^\star, j)$ for any $q < p$, since such borders are prefixes of their common prefix au. Also, the RLE size of any short border of $T[\mathsf{end}_i..\mathsf{end}_{y+j-1}]$ is upper bounded by p from the definition of P_i. To summarize, the set of short borders of $T[\mathsf{end}_i..\mathsf{end}_{y+j-1}]$ is equivalent to the set of short borders of $F(t^\star, j)$ of RLE size at most $p - 1$ if $\min \Gamma > e_1$; otherwise, it is equivalent to the set of short borders of $F(t^\star, j)$ of RLE size at most p by Lemma 8.

Next, for $1 \leq j \leq \sqrt{m}$, let us consider the short borders of $T[\mathsf{beg}_i..\mathsf{end}_{j'}]$ where $j' = y+j-1$. A short border of $T[\mathsf{beg}_i..\mathsf{end}_{j'}]$ can be obtained by extending a short border of $T[\mathsf{end}_i..\mathsf{end}_{j'}]$ to the left by $\mathsf{exp}_i - 1$ characters. Consider all the short borders $B_1, B_2, \ldots, B_g$ of $T[\mathsf{end}_i..\mathsf{end}_{j'}]$, which satisfy that $r(B_s) \leq \min\{p, \sqrt{m}\}$ for all s. Note that all such borders are also borders of $F(t^\star, j)$ as discussed above. Let e_s be the exponent of the first run of the suffix-occurrence of B_s for each $1 \leq s \leq g$. Let $\mathcal{E}^p_{j'}$ be the set of such e_s for all $B_1, B_2, \ldots, B_g$. If $e_s < \mathsf{exp}_i$ for all $e_s \in \mathcal{E}^p_{j'}$, i.e., $\max \mathcal{E}^p_{j'} < \mathsf{exp}_i$, then $T[\mathsf{beg}_i..\mathsf{end}_{j'}]$ cannot have a short border. Conversely, if $\max \mathcal{E}^p_{j'} \geq \mathsf{exp}_i$, then $T[\mathsf{beg}_i..\mathsf{end}_{j'}]$ has a short border.

Based on the observations above, we design an algorithm for computing $\text{CANDIDATE}_k(i)$.

Preprocessing. We construct a data structure for $\mathsf{ridx}_{x,z}(\cdot,\cdot)$ by using Lemma 6. Next, let us *conceptually* consider a $\sqrt{m} \times \sqrt{m}$ table M_τ defined as follows: $M_\tau[r][j] = \infty$ for all r if the first and the last characters of $F(\tau, j)$ are the same; otherwise, $M_\tau[r][j]$ stores the maximum exponent among the first runs of the suffix-occurrences of those borders of $F(\tau, j)$ whose RLE size is at most r; if there is no such a border, then $M_\tau[r][j] = 0$ (see also the left part of Fig. 2). Note that we do not explicitly construct such $\sqrt{m} \times \sqrt{m}$ tables. The details of their implementation will be described later.

Query. Given a position i, we first compute $\alpha = \mathsf{ridx}_{x,z}(i, \infty)$, which satisfies that the lcp of $T[\mathsf{end}_i..n]$ and $T[\mathsf{end}_\alpha..\mathsf{end}_z]$ equals P_i. Also, we compute $e_1 = |P_i| - |au|$ and $p = r(P_i)$. If $\mathsf{exp}_{\alpha+p-1} = e_1$, then we set $t = \alpha$

10	12	12	∞	12	7	7	∞	7	7	7
9	12	12	∞	9	7	7	∞	7	7	7
8	12	12	∞	9	7	5	∞	7	7	7
7	12	9	∞	9	5	5	∞	7	7	7
6	9	9	∞	9	5	5	∞	7	7	7
5	9	9	∞	9	5	5	∞	7	7	7
4	9	9	∞	5	5	5	∞	7	7	7
3	9	9	∞	5	5	0	∞	7	7	0
2	9	0	∞	5	0	0	∞	7	0	0
1	0	0	∞	0	0	0	∞	0	0	0
	1	2	3	4	5	6	7	8	9	10

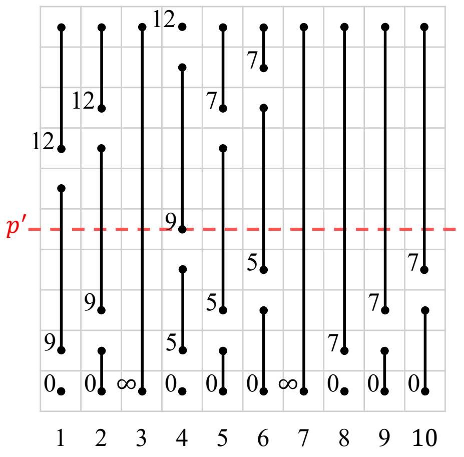

Fig. 2. Left: An example of table M_τ. Each column is non-decreasing from bottom to top. The ∞ symbols in the 3rd and 7th columns indicate that the first and last characters of $F(\tau, 3)$ and $F(\tau, 7)$ are the same, respectively. Right: Line segments corresponding to the runs in all columns of M_τ. If $p' = 5$ and $e = 7$, the largest $j^\star$ such that $M_\tau[p'][j^\star] < e$ is 6.

since $e_2 = e_1$. Otherwise, we compute $\beta = \mathsf{ridx}_{x,z}(\alpha, |P_i|)$. Next, we compute $L = lcp(T[\mathsf{end}_i..n], T[\mathsf{end}_\beta..\mathsf{end}_z])$ by calling $\mathsf{rlcp}(i, \mathsf{exp}_i, \beta, \mathsf{exp}_\beta)$. If $|L| \leq |au|$, then $\min \Gamma > e_1$ holds and thus we set $t = \alpha$. Otherwise, we set $t = \beta$ since $e_2 = |L| - |au| \leq e_1$. Furthermore, we set $p' = \min\{p - 1, \sqrt{m}\}$ if $\min \Gamma > e_1$; otherwise, set $p' = \min\{p, \sqrt{m}\}$. Next, we find the largest $j^\star$ such that $M_t[p'][j^\star] < \mathsf{exp}_i$. If there is no such $j^\star$, $\textsc{Candidate}_k(i)$ returns ε. Otherwise, it returns $T[\mathsf{beg}_i..\mathsf{end}_{y+j^\star-1}]$ since $T[\mathsf{beg}_i..\mathsf{end}_\iota]$ has a short border for all ι with $y + j^\star - 1 < \iota \leq z$, and hence, by Observation 1, $T[\mathsf{beg}_i..q]$ has a short border for all text positions q with $\mathsf{beg}_{y+j^\star} \leq q \leq \mathsf{end}_z$.

Implementing table M_τ. The remaining task is to efficiently implement M_τ so that $j^\star$ can be found quickly. For each column of a table M_τ, say jth column, the values are non-decreasing by definition. Also, there are only $O(\log m)$ distinct values due to periodicity of borders of $T[\mathsf{end}_\tau + 1..\mathsf{end}_z]\$T[\mathsf{beg}_x..\mathsf{end}_{y+j-2}]$. Namely, there are $O(\log m)$ runs of integers in the column. We define a (vertical) line segment that corresponds to each run, and assign the integer representing a run to each segment as its weight (see Fig. 2). Then we have $O(\sqrt{m} \log m)$ weighted line segments for M_τ in total. By using such weighted line segments, we can compute, for given p' and e, the largest $j^\star$ such that $M_\tau[p'][j^\star] < e$ as follows: find the rightmost line segment that intersects the horizontal line $r = p'$ and has weight less than e, then $j^\star$ is the j-coordinate of the line segment (again, see Fig. 2).

The set S_τ of such line segments can be computed in $O(\sqrt{m} \log m \log\log m)$ time by adapting the idea of the construction algorithm for the RLE shortest border array (Lemma 4) as follows: For each prefix of $F(\tau, z)$, we enumerate all $O(\log m)$ possible exponents of the first runs of the suffix-occurrences of borders, and for each exponent E, compute the minimum RLE size of a border whose first run has exponent E. By sorting these values by their RLE size in ascending order and then scanning their exponents, we can obtain the desired segments.

For each $x \leq \tau \leq z$, we construct a data structure of the *weighted lowest stabbing query (WLSQ)* on S_τ, where S_τ is rotated 90 degrees to the right, as defined below:

Definition 1 (Weighted Lowest Stabbing Query; WLSQ). *A set S of weighted horizontal segments over $N \times N$ grid are given for preprocessing. The query is, given integers v, w_1, and w_2, to report the lowest segment s such that s is stabbed by vertical line $x = v$ and the weight of s is between w_1 and w_2.*

Given i, we can obtain $j^\star$ by answering WLSQ on S_t for $v = p', w_1 = 0$, and $w_2 = \mathsf{exp}_i - 1$.

Very recently, Akram and Mieno proposed an algorithm for a problem called the *2D top-k stabbing query with weight constraint* (Definition 8 in [1]), which subsumes WLSQ as a special case. Although not stated explicitly, their data structure can be constructed in $O(|S| \log^2 |S|)$ time. We propose a simpler data structure specialized for WLSQ and show that it can be constructed slightly faster, in $O(|S| \log |S|)$ time:

Lemma 9. *A set S of segments is given as the input of WLSQ. We can build a data structure of size $O(|S| \log |S|)$ in $O(|S| \log |S|)$ time that can answer any WLSQ in $O(\log |S|)$ time.*

Thus, we can implement $\textsc{candidate}_k$ so that $\textsc{candidate}_k(i)$ can be computed in $O(\log m)$ time for each i after $O(\sum_{x \leq \tau \leq z} |S_\tau| \log |S_\tau|) = O(m \log^2 m)$ time and space preprocessing.

4.3 Implementation of RM-LONG-BORDERED

We define a new notion called the *RLE pseudo period* as follows: for a string w of RLE size r, the RLE pseudo period $pp(w)$ of w is the value $r - b$ where b is the RLE size of the border of w. Note that the RLE size p of the prefix (or suffix) of w whose length equals the period of w is not always equal to $pp(w)$, but it holds that $p - 1 \leq pp(w) \leq p$.

Example 2. Consider string $w = \texttt{abaababa}$ of RLE size $r = 7$. For this string, $pp(w) = 4$ holds since the border of w is $\texttt{aba}$ of RLE size $b = 3$. The period of w is 5. On the one hand, the RLE size p_1 of the length-5 prefix $\texttt{abaab}$ of w is 4, and then $pp(w) = p_1$ holds. On the other hand, the RLE size p_2 of the length-5 suffix $\texttt{ababa}$ of w is 5, and then $pp(w) = p_2 - 1$ holds.

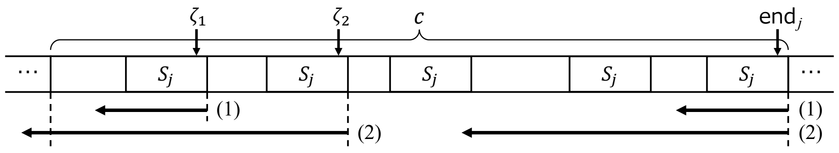

Fig. 3. String $c \in C_j$ is the candidate for an unbordered factor that ends at end_j we focus on. (1) The lcs of $T[1..\mathsf{end}_j]$ and the prefix $T[1..\zeta_1]$ of T that ends at the leftmost occurrence of S_j in the figure does not reach the starting position of c. We cannot yet determine whether c is bordered. (2) The lcs of $T[1..\mathsf{end}_j]$ and the prefix $T[1..\zeta_2]$ of T that ends at the second leftmost occurrence of S_j in this figure reaches the starting position of c, which reveals that c is bordered.

For each $j > \sqrt{m}$, let S_j be the longer string within (1) the shortest suffix of $\$T[1..\mathsf{end}_j]$ whose RLE pseudo period is greater than $\sqrt{m}/2 - 1$, and (2) $T[\mathsf{beg}_{j-\sqrt{m}+1}..\mathsf{end}_j]$. Note that S_j is well-defined since the RLE pseudo period of $\$T[1..\mathsf{end}_j]$ must be greater than $\sqrt{m}/2 - 1$ for $j > \sqrt{m}$.

Lemma 10. *If the shortest border of $T[\mathsf{beg}_i..\mathsf{end}_j]$ is a long border, then S_j is a suffix of each border of $T[\mathsf{beg}_i..\mathsf{end}_j]$.*

Proof. Let B be the shortest border of $T[\mathsf{beg}_i..\mathsf{end}_j]$. Since B is unbordered, the pseudo period of B equals the RLE size of B, which is greater than $\sqrt{m}$. Thus, $|S_j| \leq |B|$ holds, and thus S_j is a suffix of B. Therefore, S_j is a suffix of each border of $T[\mathsf{beg}_i..\mathsf{end}_j]$. $\square$

Lemma 11. *The number of occurrences of S_j in T is in $O(\sqrt{m})$.*

Lemma 12. *Given $j > \sqrt{m}$, string S_j and its all occurrences can be computed in $O(\sqrt{m}\log m)$ time after $O(m \log m)$ time preprocessing.*

Preprocessing We construct the data structure of Lemma 12.

Query algorithm Again, let us consider the kth stage and let $x = J_{k-1}.\mathsf{first}$, $y = J_k.\mathsf{first}$, and $z = J_k.\mathsf{last}$. $D_k = T[\mathsf{beg}_x..\mathsf{end}_z]$ and $J_k = T[\mathsf{beg}_y..\mathsf{end}_z]$. For each j with $y \leq j \leq z$, we compute S_j and its occurrences, and sort them by using Lemma 12. Let L_j be the sorted list of occurrences of S_j. Given a set C of candidates for the longest unbordered factors, we first radix sort the elements of C using their starting points as the primary key and their ending points as the secondary key. Let C' be the sorted list of the elements of C. In the following, we scan elements in C' in order throughout the algorithm. We denote by C_j the list of elements in C' whose end positions are end_j. Then, C_j is a consecutive sub-list of C' from the condition of the radix sorting. For every j with $y \leq j \leq z$, we execute the following, scanning C_j and L_j from left to right: Let variables c and *occ* store elements in C_j and L_j we focus on, respectively. By using the data structure of Lemma 5 for the reversal of T, compute the longest common suffix (lcs) of $T[1..\mathsf{end}_j]$ and the $T[1..\zeta]$ where ζ is the ending position of *occ* (see Fig. 3).

– If the lcs does not reach the starting position of c, then update *occ* to the next element in L_j. If such an element in L_j does not exists, then the current c is the longest string in C_j that has no long border by Lemma 10.
– If the lcs reaches the starting position of c, then c is bordered and hence update c to the next element in C_j that starts after ζ. If such an element in C_j does not exists, there is no unbordered factor in C_j. So we update c to the next element in C' and increment j accordingly.

The query algorithm can be performed in $O(|C|\log|C| + \sum_{y\le j\le z}|L_j|) = O(m\log m)$ time with $O(m)$ working space for each stage k.

Putting all pieces together, we complete the proof of Theorem 1.

Acknowledgments. This work was supported by JSPS KAKENHI Grant Number JP24K20734 (TM).

References

1. Akram, W., Mieno, T.: Sorted consecutive occurrence queries in substrings. In: CPM 2025 (to appear)
2. Assous, R., Pouzet, M.: Une caracterisation des mots periodiques. Discret. Math. **25**(1), 1–5 (1979). https://doi.org/10.1016/0012-365X(79)90146-8
3. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1776, pp. 88–94. Springer (2000). https://doi.org/10.1007/10719839_9
4. Cording, P.H., Gagie, T., Knudsen, M.B.T., Kociumaka, T.: Maximal unbordered factors of random strings. Theor. Comput. Sci. **852**, 78–83 (2021). https://doi.org/10.1016/J.TCS.2020.11.019
5. Crochemore, M., Mignosi, F., Restivo, A., Salemi, S.: Text compression using antidictionaries. In: Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1644, pp. 261–270. Springer (1999). https://doi.org/10.1007/3-540-48523-6_23
6. Duval, J.: Relationship between the period of a finite word and the length of its unbordered segments. Discret. Math. **40**(1), 31–44 (1982). https://doi.org/10.1016/0012-365X(82)90186-8
7. Duval, J., Harju, T., Nowotka, D.: Unbordered factors and lyndon words. Discret. Math. **308**(11), 2261–2264 (2008). https://doi.org/10.1016/J.DISC.2006.09.054
8. Duval, J., Lecroq, T., Lefebvre, A.: Linear computation of unbordered conjugate on unordered alphabet. Theor. Comput. Sci. **522**, 77–84 (2014). https://doi.org/10.1016/J.TCS.2013.12.008
9. Ehrenfeucht, A., Silberger, D.M.: Periodicity and unbordered segments of words. Discret. Math. **26**(2), 101–109 (1979). https://doi.org/10.1016/0012-365X(79)90116-X
10. Fine, N.J., Wilf, H.S.: Uniqueness theorems for periodic functions. Proceedings of the American Mathematical Society **16**(1), 109–114 (1965)

11. Gawrychowski, P., Kucherov, G., Sach, B., Starikovskaya, T.: Computing the longest unbordered substring. In: String Processing and Information Retrieval - 22nd International Symposium, SPIRE 2015, London, UK, September 1-4, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9309, pp. 246–257. Springer (2015). https://doi.org/10.1007/978-3-319-23826-5_24
12. Harju, T., Nowotka, D.: Minimal Duval extensions. Int. J. Found. Comput. Sci. **15**(2), 349–354 (2004). https://doi.org/10.1142/S0129054104002467
13. Harju, T., Nowotka, D.: Periodicity and unbordered words: A proof of the extended duval conjecture. J. ACM **54**(4), 20 (2007). https://doi.org/10.1145/1255443.1255448
14. Holub, S.: A proof of the extended Duval's conjecture. Theor. Comput. Sci. **339**(1), 61–67 (2005). https://doi.org/10.1016/J.TCS.2005.01.008
15. Holub, S., Nowotka, D.: The ehrenfeucht-silberger problem. J. Comb. Theory A **119**(3), 668–682 (2012). https://doi.org/10.1016/J.JCTA.2011.11.004
16. Knuth, D.E., Jr., J.H.M., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. **6**(2), 323–350 (1977). https://doi.org/10.1137/0206024
17. Kociumaka, T., Kundu, R., Mohamed, M., Pissis, S.P.: Longest unbordered factor in quasilinear time. In: 29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan. LIPIcs, vol. 123, pp. 70:1–70:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018). https://doi.org/10.4230/LIPICS.ISAAC.2018.70
18. Kociumaka, T., Radoszewski, J., Rytter, W., Walen, T.: Internal pattern matching queries in a text and applications. SIAM J. Comput. **53**(5), 1524–1577 (2024). https://doi.org/10.1137/23M1567618
19. Loptev, A., Kucherov, G., Starikovskaya, T.: On maximal unbordered factors. In: Combinatorial Pattern Matching - 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 - July 1, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9133, pp. 343–354. Springer (2015). https://doi.org/10.1007/978-3-319-19929-0_29
20. Margaritis, D., Skiena, S.: Reconstructing strings from substrings in rounds. In: 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995. pp. 613–620. IEEE Computer Society (1995). https://doi.org/10.1109/SFCS.1995.492591
21. Mignosi, F., Zamboni, L.Q.: A note on a conjecture of Duval and Sturmian words. RAIRO Theor. Informatics Appl. **36**(1), 1–3 (2002). https://doi.org/10.1051/ITA:2002001
22. Mitani, K., Mieno, T., Seto, K., Horiyama, T.: Shortest cover after edit. In: 35th Annual Symposium on Combinatorial Pattern Matching, CPM 2024, June 25-27, 2024, Fukuoka, Japan. LIPIcs, vol. 296, pp. 24:1–24:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024). https://doi.org/10.4230/LIPICS.CPM.2024.24
23. Tamakoshi, Y., Goto, K., Inenaga, S., Bannai, H., Takeda, M.: An opportunistic text indexing structure based on run length encoding. In: Algorithms and Complexity - 9th International Conference, CIAC 2015, Paris, France, May 20-22, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9079, pp. 390–402. Springer (2015). https://doi.org/10.1007/978-3-319-18173-8_29

A Omitted proofs

Proof (of Lemma 5). Suppose that an RLE-compressed string $\mathrm{rle}(T)$ of RLE size m is given. In the preprocessing phase, we construct length-m arrays tRLESA,

$\mathsf{tRLEISA}$, and $\mathsf{tRLELCP}$ of string T, defined as follows: The *truncated RLE suffix array* tRLESA of T is the integer array such that $\mathsf{tRLESA}[r] = k$ iff the rth lexicographically smallest suffix among $\mathcal{S} = \{T[\mathsf{end}_x..n] \mid 1 \leq x \leq m\}$ is $T[\mathsf{end}_k..n]$. For convenience, we denote by $T\langle r\rangle = T[\mathsf{end}_{\mathsf{tRLESA}[r]}..n]$ the rth lexicographically smallest suffix among $\mathcal{S}$. The *truncated RLE inverse suffix array* $\mathsf{tRLEISA}$ of T is the integer array such that $\mathsf{tRLEISA}[k] = r$ iff $\mathsf{tRLESA}[r] = k$. The *truncated RLE longest common prefix array* $\mathsf{tRLELCP}$ of T is the integer array such that $\mathsf{tRLELCP}[1] = -1$ and $\mathsf{tRLELCP}[r] = \mathit{lcp}(T\langle r\rangle, T\langle r-1\rangle)$ for $2 \leq r \leq m$. We further construct an RmQ (range *minimum* query) data structure on $\mathsf{tRLELCP}$. Given query integers i, p, j, and q, we first compare $\mathsf{exp}_i - p + 1$ and $\mathsf{exp}_j - q + 1$. If they are not equal, the lcp value is the minimum of those values. Otherwise, let $r_1 = \min\{\mathsf{tRLEISA}[i], \mathsf{tRLEISA}[j]\}$ and $r_2 = \max\{\mathsf{tRLEISA}[i], \mathsf{tRLEISA}[j]\}$. Then, the lcp value to return is $\mathsf{exp}_i - p + \min\{\mathsf{tRLELCP}[r] \mid r_1 + 1 \leq r \leq r_2\}$, which can be computed in $O(1)$ time by answering RmQ on $\mathsf{tRLELCP}$. □

Proof (of Lemma 6). Given $\mathrm{rle}(T)$ of RLE size m and integers x, y, we consider string $T' = T\$T[\mathsf{beg}_x..\mathsf{end}_y]$ whose RLE size is at most $2m + 1$. Then, if a string occurs within $T[\mathsf{beg}_x..\mathsf{end}_y]$, then it also occurs at position $p > n + 1$ in T'. As in the proof of Lemma 5, we construct three arrays of T' and RmQ data structure on $\mathsf{tRLELCP}$. In addition, we construct an RMQ (range *maximum* query) data structure on tRLESA. Given query integers h and ℓ, we first obtain $r_h = \mathsf{tRLEISA}[h]$. Then, we compute the maximal range $[r_1, r_2] \subset [1, m]$ such that $\mathit{lcp}(T\langle r_1\rangle, T\langle r_h\rangle) \geq \ell + 1$ and $\mathit{lcp}(T\langle r_h\rangle, T\langle r_2\rangle) \geq \ell + 1$ hold. The range $[r_1, r_2]$ can be computed in $O(\log m)$ time by combining binary search and RmQ on $\mathsf{tRLELCP}$. Next, we compute the maximum value $r_3 \in [1, r_1 - 1]$ such that $\mathit{lcp}(T\langle r_3\rangle, T\langle r_h\rangle) \leq \ell$ and $\mathsf{tRLESA}[r_3] \geq m + 2$. The value r_3 can be computed in $O(\log m)$ time by combining binary search and RMQ on tRLESA in range $[1, r_1 - 1]$. Similarly, we compute the minimum value $r_4 \in [r_2 + 1, m]$ such that $\mathit{lcp}(T\langle r_h\rangle, T\langle r_4\rangle) \leq \ell$ and $\mathsf{tRLESA}[r_4] \geq m + 2$ in $O(\log m)$ time. Finally, we return $\mathsf{tRLESA}[r^\star]$ where $r^\star = \arg\max_{r \in \{r_3, r_4\}}\{\mathit{lcp}(T\langle r\rangle, T\langle r_h\rangle)\}$. □

Proof (of Lemma 7). (1) Consider the case where there is an unbordered (i, k)-factor, and let U be the longest unbordered (i, k)-factor. For the sake of contradiction, assume that U is shorter than $\textsc{candidate}_k(i)$. Then, $\textsc{candidate}_k(i)$ has a long border since it has no short border. However, the suffix-occurrence of the long border starts before J_k since $r(J_k) = \sqrt{m}$, which contradicts that U is unbordered. Thus, $\textsc{candidate}_k(i)$ is the longest unbordered (i, k)-factor in this case. (2) If all (i, k)-factors have short borders, then $\textsc{candidate}_k(i)$ returns ε by definition. (3) Otherwise, i.e., all (i, k)-factors have some borders. By definition, $\textsc{candidate}_k(i)$ has no short border, thus it must have a long border. □

Proof (of Lemma 9). In the preprocessing phase, we first sort the segments by their weights. Then, we construct the segment tree of S with N leaves by inserting segments in the sorted order. Each segment stored in a node is represented as a tuple (w, y, id) where w is the weight of the segment, y is the y-coordinate of the segment, and id is the unique id of the segment in S. The resulting segment tree satisfies the condition that segments stored in each node are sorted by their weights. Subsequently, for each node of the segment tree, we build an RmQ data structure for the list of y-coordinates stored in the node. Lastly, we apply the fractional cascading to the series of tuples over the segment tree, using weight w as the key.

Given a query (v, w_1, w_2), we first find the path π from the root to the leaf that corresponds to v in the segment tree. We then traverse π starting from the root. At each node $u \in \pi$, we compute the range corresponding to $[w_1, w_2]$ in the list of tuples in u by using the fractional cascading structure, and compute the smallest y-coordinate within the range by using the RmQ data structure. At the end, we output the id that corresponds to the smallest y-coordinate within the minima.

The size of the data structure is $O(|S| \log |S|)$, which is dominated by the segment tree. The construction time is $O(|S| \log |S|)$: sorting the segments takes $O(|S| \log |S|)$ time, building the segment tree requires $O(|S| \log |S|)$ time, and constructing the RmQ data structures and applying fractional cascading take time linear in the total size of the lists, i.e., $O(|S| \log |S|)$. Given a query, we first find the root-to-leaf path π of length $O(\log |S|)$. Detecting all the ranges corresponding to $[w_1, w_2]$ over the lists in π can be done in $O(\log |S|)$ time thanks to fractional cascading. We then perform an $O(1)$-time RmQ $|\pi|$ times, which takes $O(\log |S|)$ time. Thus, the query time is $O(\log |S|)$ in total. □

Proof (of Lemma 11). Since the pseudo period pp of S_j is greater than $\sqrt{m}/2-1$, the RLE size of the overlap of any two occurrences of S_j is at most $r(S_j) - pp < r(S_j) - \sqrt{m}/2 + 1$. Thus, the lemma holds. □

Proof (of Lemma 12). We first compute the RLE pseudo period π of $T[\mathsf{beg}_{j-\sqrt{m}+1}..\mathsf{end}_j]$ in $O(\sqrt{m}\log m)$ time. If $\pi > \sqrt{m}/2 - 1$, then $S_j = T[\mathsf{beg}_{j-\sqrt{m}+1}..\mathsf{end}_j]$. Otherwise, we compute the longest suffix $T[s..\mathsf{end}_j]$ of $T[1..\mathsf{end}_j]$ whose RLE pseudo period is π by computing the longest common *suffix* of $T[1..\mathsf{end}_{j-\sqrt{m}}$ and $T[1..\iota]$ where ι is the ending position of the prefix-occurrence of the border of $T[\mathsf{beg}_{j-\sqrt{m}+1}..\mathsf{end}_j]$. Such suffix can be computed by using the lcp data structure of Lemma 5 for the reversal of T. If $s = 1$, then we have $S_j = \$T[1..\mathsf{end}_j]$. Otherwise, we show the following claim:

Claim. The RLE pseudo period ψ of $T[s-1..\mathsf{end}_j]$ is greater than $\sqrt{m}/2 - 1$.

Proof (of Claim). Now we consider the RLE pseudo period ψ of $T[s-1..\mathsf{end}_j]$ which satisfies $\psi > \pi$. For the sake of contradiction, we assume that $\psi \leq \sqrt{m}/2 - 1$ holds. Then, it holds that $\pi + 1 + \psi + 1 < \sqrt{m} \leq r(T[s..\mathsf{end}_j])$ since $\psi > \pi$. Let p and q be the periods of $T[s..\mathsf{end}_j]$ and $T[s-1..\mathsf{end}_j]$, respectively. Further, let π' and ψ' be the RLE sizes of $T[s..s+\mathsf{p}-1]$ and $T[s..s+\mathsf{q}-1]$, respectively. Then, $\pi' \leq \pi + 1$ and $\psi' \leq \psi' + 1$ hold by the definition of RLE pseudo borders. Thus, we obtain $\pi' + \psi' \leq \pi + 1 + \psi + 1 < r(T[s..\mathsf{end}_j])$. Therefore, $\mathsf{p} + \mathsf{q} < |T[s..\mathsf{end}_j]|$ holds. From the periodicity lemma [10], $\gcd(\mathsf{p}, \mathsf{q})$ is a period of $T[s..\mathsf{end}_j]$ where $\gcd(\mathsf{p}, \mathsf{q})$ is the greatest common diviser of p and q. Since p is the smallest period of $T[s..\mathsf{end}_j]$, q is a multiple of p. Then, $T[s-1..\mathsf{end}_j]$ also has period p. However, the RLE pseudo period of $T[s-1..\mathsf{end}_j]$ is π, which contradicts the definition of s. □

Namely, $S_j = T[s-1..\mathsf{end}_j]$ holds if $s \geq 2$. Finally, we compute each occurrence of S_j in $O(occ + \log m)$ time by using the index structure based on tRLESA [23], where occ is the number of occurrences of S_j in T. The total running time is $O(\sqrt{m} \log m)$ since $occ = O(\sqrt{m})$ by Lemma 11. □