Counting distinct (non-)crossing substrings

Haruki Umezaki[1], Hiroki Shibata[2][0009−0006−6502−7476], Dominik Köppl[3][0000−0002−8721−4444], Yuto Nakashima[4][0000−0001−6269−9353], Shunsuke Inenaga[4][0000−0002−1833−010X], and Hideo Bannai[5][0000−0002−6856−5185]

[1] Department of Information Science and Technology, Kyushu University
umezaki.haruki.314@s.kyushu-u.ac.jp
[2] Joint Graduate School of Mathematics for Innovation, Kyushu University
shibata.hiroki.753@s.kyushu-u.ac.jp
[3] Department of Computer Science and Engineering, University of Yamanashi
dkppl@yamanashi.ac.jp
[4] Department of Informatics, Kyushu University
{nakashima.yuto.003, inenaga.shunsuke.380}@m.kyushu-u.ac.jp
[5] M&D Data Science Center, Institute of Integrated Research,
Institute of Science Tokyo
hdbn.dsc@tmd.ac.jp

Abstract. Let w be a string of length n. The problem of counting factors crossing a position - Problem 64 from the textbook "125 Problems in Text Algorithms" [Crochemore, Leqroc, and Rytter, 2021], asks to count the number $\mathcal{C}(w,k)$ (resp. $\mathcal{N}(w,k)$) of distinct substrings in w that have occurrences containing (resp. not containing) a position k in w. The solutions provided in their textbook compute $\mathcal{C}(w,k)$ and $\mathcal{N}(w,k)$ in $O(n)$ time *for a single position* k in w, and thus a direct application would require $O(n^2)$ time for *all positions* $k = 1,\ldots,n$ in w. Their solution is designed for constant-size alphabets. In this paper, we present new algorithms which compute $\mathcal{C}(w,k)$ in $O(n)$ total time for general ordered alphabets, and $\mathcal{N}(w,k)$ in $O(n)$ total time for linearly sortable alphabets, for all positions $k = 1,\ldots,n$ in w.

Keywords: string algorithms, distinct substrings, runs, LPF arrays

1 Introduction

Let w be a string of length n. The problem of counting factors crossing a position - Problem 64 from the textbook "125 Problems in Text Algorithms" [3], asks to count the number $\mathcal{C}(w,k)$ (resp. $\mathcal{N}(w,k)$) of distinct substrings in w that have occurrences containing (resp. not containing) a position k in w. According to the textbook [3], the notions of $\mathcal{C}(w,k)$ and $\mathcal{N}(w,k)$ are inspired by the notion of *string attractors* [8], which form a set $\mathcal{P} = \{p_1,\ldots,p_\gamma\}$ of γ positions such that any substring of w has an occurrence containing a position $p_i \in \mathcal{P}$. Besides this origin, how efficiently one can compute $\mathcal{C}(w,k)$ and $\mathcal{N}(w,k)$ for a given string w, is an intriguing stringology question.

The solutions provided in the textbook [3] compute $\mathcal{C}(w,k)$ and $\mathcal{N}(w,k)$ in $O(n)$ time *for a single position* k in w for constant-size alphabets. Thus, a direct application of their solutions to the *all-position variant* of the problems, which ask to compute $\mathcal{C}(w,k)$ and $\mathcal{N}(w,k)$ for *all positions* $k = 1,\ldots,n$ in w, requires $O(n^2)$ total time.

In this paper, we present new algorithms which compute for all positions $k = 1,\ldots,n$, $\mathcal{C}(w,k)$ in $O(n)$ total time and space for general ordered alphabets, and $\mathcal{N}(w,k)$ in $O(n)$ total time and space for linearly sortable alphabets. Our solution for computing $\mathcal{C}(w,k)$ for $k = 1,\ldots,n$ exploits the combinatorial property of the problem and utilizes the *runs* (a.k.a. *maximal repetitions*) [9] occurring in w, which is completely different from the original solution from the textbook [3].

2 Preliminaries

2.1 Strings

Let Σ be an ordered alphabet. An element of Σ^* is called a *string*. The length of a string $w \in \Sigma^*$ is denoted by $|w|$. The *empty string* ε is the string of length 0. Let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For string $w = xyz$, x, y, and z are called a *prefix*, *substring*, and *suffix* of w, respectively. Let $\mathsf{Substr}(w)$ and $\mathsf{Suffix}(w)$ denote the sets of substrings and suffixes of w, respectively. For a string w of length n, $w[i]$ denotes the ith character of w and $w[i..j] = w[i] \cdots w[j]$ denotes the substring of w that begins at position i and ends at position j for $1 \leq i \leq j \leq n$. For convenience, let $w[i..j] = \varepsilon$ for $i > j$.

For two non-empty strings s and w, let $\mathsf{occ}(s, w) = \{i \mid w[i..i + |s| - 1] = s\}$ denote the set of occurrences of s in w, where we identify an occurrence of s with its starting position. For each position $1 \leq k \leq |w|$ in w, let

$$\mathsf{cocc}_k(s, w) = \{i \in \mathsf{occ}(s, w) \mid i \leq k \leq i + |s| - 1\}$$
$$\mathsf{ncocc}_k(s, w) = \{i \in \mathsf{occ}(s, w) \mid i + |s| - 1 < k \text{ or } k < i\}$$

denote the sets of occurrences of string s that cross (resp. do not cross) the position k in w. Let

$$\begin{aligned} \mathsf{C}(w, k) &= \{s \in \Sigma^+ \mid \mathsf{cocc}_k(s, w) \neq \emptyset\} \\ \mathsf{N}(w, k) &= \{s \in \Sigma^+ \mid \mathsf{ncocc}_k(s, w) \neq \emptyset\} \\ &= \mathsf{Substr}(w[1..k-1]) \cup \mathsf{Substr}(w[k+1..|w|]) \end{aligned}$$

denote the sets of substrings s of string w that have crossing (resp. non-crossing) occurrence(s) for the position k in w.

Problem 1 (Counting distinct substrings with (non-)crossing occurrences). Given a string w of length n, compute $\mathcal{C}(w, k) = |\mathsf{C}(w, k)|$ and $\mathcal{N}(w, k) = |\mathsf{N}(w, k)|$ for all positions $k = 1, \ldots, n$ in w.

2.2 Repetitions and runs

For a string s, an integer p ($1 \leq p \leq |s|$) is a period of s if $s[i] = s[i+p]$ for all $1 \leq i \leq |s| - p$. The *exponent* of s is the rational $|s|/p$, where p is the smallest period of s. A string $s \in \Sigma^+$ is said to be *periodic* if the exponent of s is at least 2, or equivalently, s's smallest period is at most $|s|/2$. A maximal periodic substring $s = w[i..j]$ of w, i.e., the smallest period p of s does not extend to the left of position i nor to the right of position j, namely, $i = 1$ or $w[i-1] \neq w[i+p-1]$ and $j = |w|$ or $w[j+1] \neq w[j-p+1]$, is called a *maximal repetition*, or *run*, in w. We identify a run $w[i..j]$ with the smallest period p by a tuple $\langle i, j, p \rangle$. Let $\mathsf{Runs}(w) = \{\langle i, j, p \rangle \mid w[i..j] \text{ is a run in } w\}$ denote the set of runs in w.

Theorem 1 ([1]). *$|\mathsf{Runs}(w)| < n$ holds for any string w of length n.*

Theorem 2 ([5]). *$\mathsf{Runs}(w)$ can be computed in $O(n)$ time for any string $w[1..n]$ over an ordered alphabet.*

2.3 Suffix trees

The *suffix tree* [10] of a string w, denoted $\mathsf{STree}(w)$, is a path-compressed trie representing $\mathsf{Suffix}(w)$ such that (1) each internal node has at least two children, (2) each edge is labeled by a non-empty substring of w, and (3) the labels of out-going edges of the same node begin with distinct characters. Each leaf of $\mathsf{STree}(w)$ is associated with the occurrence of its corresponding suffix of w.

For a node v of $\mathsf{STree}(w)$, let $\mathsf{str}(v)$ denote the string label of the path from the root to v. Each node v stores its string depth $|\mathsf{str}(v)|$. The *locus* of a substring $s \in \mathsf{Substr}(w)$ in $\mathsf{STree}(w)$ is the position where s is spelled out from the root. The number of nodes in $\mathsf{STree}(w)$ is at most $2n-1$, where $n = |w|$. We can represent $\mathsf{STree}(w)$ in $O(n)$ space by representing each edge label s with a pair (i, j) of positions in w such that $w[i..j] = s$.

Suppose that string w terminates with an end-marker \$ that does not occur anywhere else in w. Then, since $|\mathsf{occ}(y, w)| = 1$ holds for every suffix y of w, $\mathsf{STree}(w)$ has exactly $|w|$ leaves.

Theorem 3 ([6]). *$\mathsf{STree}(w)$ can be built in $O(n)$ time for any string $w[1..n]$ over a linearly-sortable alphabet.*

3 Computing $\mathcal{C}(w, k)$ for all positions k in a string w

In this section, we show how to compute $\mathcal{C}(w, k)$ in $O(n)$ total time for all positions k in a given string w of length n over an ordered alphabet.

In our algorithm for computing $\mathcal{C}(w, k)$, we first compute the size of the multiset of substrings that cross position k in w, and then subtract the number $\mathcal{D}(w, k)$ of duplicates. Let $\mathsf{U}(w, k)$ be the multiset of substrings crossing k in a given string w. Since $|\mathsf{U}(w, k)|$ is equal to the number of intervals including k in w, $|\mathsf{U}(w, k)| = k(|w| - k + 1)$ holds: $[i, j]$ includes k iff $i \in [1, k]$ and $j \in [k, |w|]$.

Let us consider how to compute $\mathcal{D}(w, k)$. The following observation and lemma are a key.

Observation 1 *For any substring x and position k in string w, if $\mathsf{cocc}_k(x, w) \geq 2$, then x is a substring of a run of w with smallest period $p < |x|$.*

We use the following well-known result:

Lemma 1 (Weak periodicity lemma [7]). *If p and q are periods of a string w, then $\gcd(p, q)$ is also a period of w.*

Lemma 2. *For a run $r = \langle i, j, p\rangle$ of a string w, the distance d between any two consecutive occurrences of a substring x in r with $|x| \geq p$ must be p.*

Proof. Due to the periodicity of r, any two consecutive occurrences are at distance $d \leq p$. If $d < p$, it follows from the weak periodicity lemma that d and p are periods of a substring of length $d + |x| > d + p$, implying that $p' = \gcd(d, p) < p$ is a period of r, which contradicts the minimality of p. □

Let $\mathsf{Runs}(w, k) = \{\langle i, j, p\rangle \in \mathsf{Runs}(w) \mid i \leq k \leq j\}$ denote the set of runs in w that cross position k. For a run $\langle i, j, p\rangle \in \mathsf{Runs}(w, k)$, let

$$S(\langle i, j, p\rangle, k) = \{x \in \Sigma^+ \mid x = w[g..h], i \leq g \leq k \leq h \leq j, |x| = h - g + 1 > p\}$$

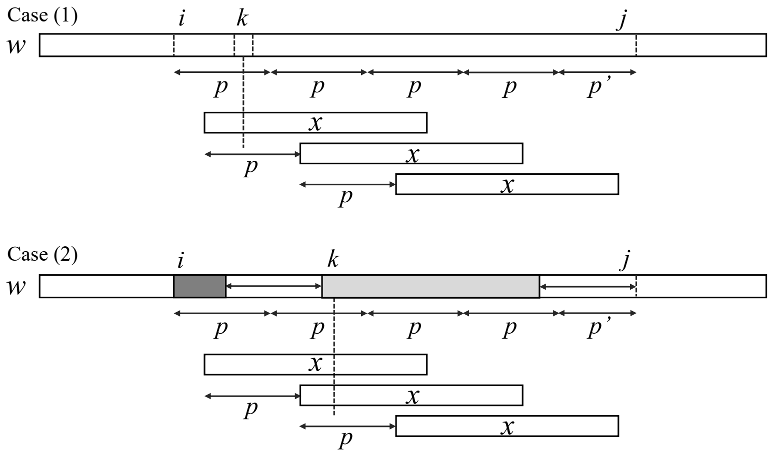

Fig. 1. Illustration for $\mathsf{dup}(\langle i,j,p\rangle,k)$ for Cases (1) and (2).

denote the set of substrings x of length at least $p+1$ that occur in the run $\langle i,j,p\rangle$ and cross k. Let $\mathsf{dup}(\langle i,j,p\rangle,k) = \sum_{x \in S(\langle i,j,p\rangle,k)} (|\mathsf{cocc}_k(x,w)|-1)$ be the number of duplicates contained in the run $\langle i,j,p\rangle$. From Observation 1 and Lemma 2, it follows that

$$\mathsf{dup}(\langle i,j,p\rangle,k) = \begin{cases} 0 & \text{if } i \le k \le i+p-1, \quad (1) \\ (k-i-p+1)(j-p+1-k) & \text{if } i+p \le k \le j-p, \quad (2) \\ 0 & \text{if } j-p+1 \le k \le j. \quad (3) \end{cases}$$

See Fig. 1. In Case (1), there is only one crossing occurrence for each substring x of length at least $p+1$ in the run $\langle i,j,p\rangle$, and thus $\mathsf{dup}(\langle i,j,p\rangle,k) = 0$. Case (3) is symmetric. In Case (2), for each substring x of length at least $p+1$ in the run $\langle i,j,p\rangle$, we count all occurrences crossing k except for the rightmost one. Notice that any substring x that starts in the dark gray region of length $k-i-p+1$ and ends in the light gray region of length $j-p+1-k$ crosses k and has exactly one occurrence that starts in the region of length p between the two gray regions and crosses k. Therefore, a total of $(k-i-p+1)(j-p+1-k)$ duplicate occurrences are counted in Case (2).

Lemma 3. $\mathcal{D}(w,k) = \sum_{\langle i,j,p\rangle \in \mathsf{Runs}(w,k)} \mathsf{dup}(\langle i,j,p\rangle,k)$.

Proof. It is clear that $\mathcal{D}(w,k) \le \sum_{\langle i,j,p\rangle \in \mathsf{Runs}(w,k)} \mathsf{dup}(\langle i,j,p\rangle,k)$. We prove the lemma by showing that the same duplicates are counted in different runs. Assume for a contradiction that the same substring x is counted in $\mathsf{dup}(\langle i,j,p\rangle,k)$ and in $\mathsf{dup}(\langle i',j',p'\rangle,k)$ by two distinct runs $\langle i,j,p\rangle$, $\langle i',j',p'\rangle \in \mathsf{Runs}(w,k)$. Without loss of generality suppose that $p \le p'$.

If x has only a single occurrence that crosses k within the run $\langle i,j,p\rangle$, then it is not counted in $\mathsf{dup}(\langle i,j,p\rangle,k)$ because of the definition $\mathsf{dup}(\langle i,j,p\rangle,k) = \sum_{x \in S(\langle i,j,p\rangle,k)} (|\mathsf{cocc}_k(x,w)|-1)$. The other case with $\langle i',j',p'\rangle$ is analogous.

For the case where x has at least two occurrences that cross k in each of the runs $\langle i,j,p\rangle$ and $\langle i',j',p'\rangle$, Lemma 2 implies that $p = p'$. However, since the runs overlap by at least p positions, the periodicity of one run extends into the other, contradicting their maximality. $\square$

After $O(n)$-time preprocessing for computing $\mathsf{Runs}(w)$ with Theorem 2, Observation 1 and Lemma 3 immediately lead us to an $O(n)$-time solution to compute $\mathcal{C}(w,k)$ for a *fixed* k.

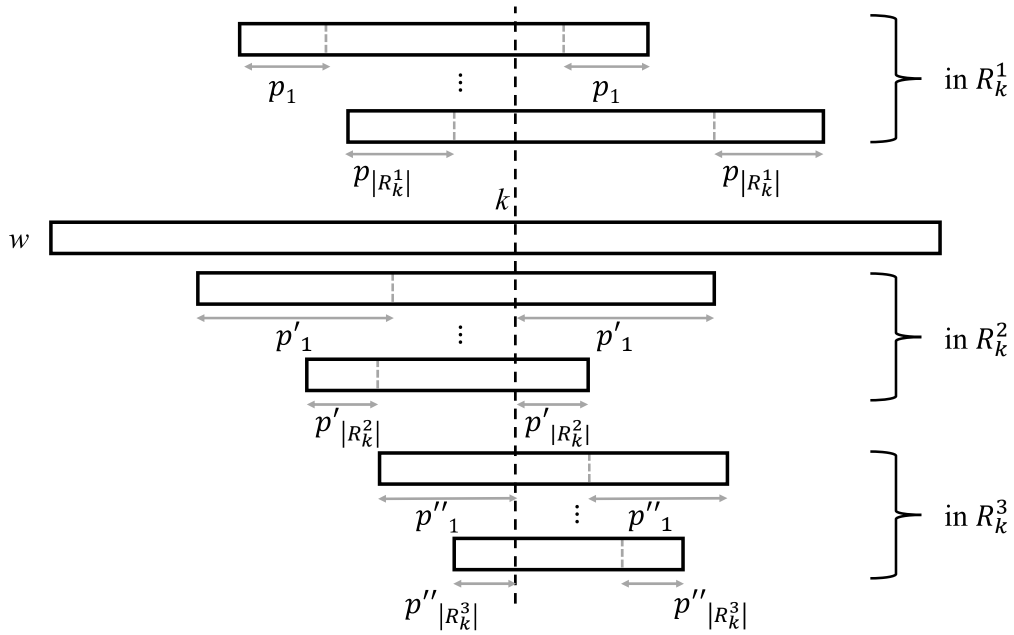

Fig. 2. Illustration for R_k^1, R_k^2, and R_k^3.

Our strategy to compute $\mathcal{C}(w,k)$ for all $k = 1,\ldots,n$ is first to compute $\mathcal{C}(w,1) = |\mathsf{U}(w,1)| - \mathcal{D}(w,1)$ for $k = 1$ in $O(n)$ time, and compute $\mathcal{C}(w,k) = |\mathsf{U}(w,k)| - \mathcal{D}(w,k)$ in amortized $O(1)$ time for increasing $k = 2,\ldots,n$. Since $|\mathsf{U}(w,k)|$ is computable in $O(1)$ time by a simple arithmetic for every k, in what follows we focus on how to compute $\mathcal{D}(w,k)$.

The next lemma exploits a useful structure of $\mathsf{dup}(\langle i,j,p\rangle,k)$ for the consecutive positions $k = i+p,\ldots,j-p$.

Lemma 4. *For each run $\langle i,j,p\rangle$, consider the sequence*

$$num_{\langle i,j,p\rangle} = \mathsf{dup}(\langle i,j,p\rangle, i+p),\ldots,\mathsf{dup}(\langle i,j,p\rangle, j-p)$$

of $j-i-2p+1$ integers. Then, $num_{\langle i,j,p\rangle}$ is an integer sequence whose difference sequence is an arithmetic progression that starts with $\mathsf{dup}(\langle i,j,p\rangle, i+p)$ and has common difference -2.

Proof. Let $b = \mathsf{dup}(\langle i,j,p\rangle, i+p) = j-i-2p+1$ from Case (2). Let $num_{\langle i,j,p\rangle}[a]$ denote the ath term in the sequence. Then $num_{\langle i,j,p\rangle}[a] = a(b+1-a)$, and hence $num_{\langle i,j,p\rangle}[a+1]-num_{\langle i,j,p\rangle}[a] = (a+1)(b-a)-a(b+1-a) = ab-a^2+b-a-ab-a+a^2 = b-2a$. This is the general term of the arithmetic progression that starts with b and has common difference -2. Therefore, $num_{\langle i,j,p\rangle}$ is an integer sequence whose difference sequence is an arithmetic progression that starts with $b = \mathsf{dup}(\langle i,j,p\rangle, i+p)$ and has common difference -2. □

For each position k, let $R_k = \{\langle i,j,p\rangle \in \mathsf{Runs}(w,k) \mid i+p \le k \le j-p\}$ be the set of runs $\langle i,j,p\rangle$ such that $\mathsf{dup}(\langle i,j,p\rangle,k) > 0$. We divide runs $\langle i,j,p\rangle \in R_k \cup R_{k+1}$ into the following three disjoint subsets (see also Fig. 2):

$$R_k^1 = R_k \cap R_{k+1},$$
$$R_k^2 = R_k \setminus R_{k+1},$$
$$R_k^3 = R_{k+1} \setminus R_k.$$

Recall that for each $\langle i,j,p\rangle \in R_k$, we have $\mathsf{dup}(\langle i,j,p\rangle,k) = (k-i-p+1)(j-p+1-k)$. By Lemma 4, $num_{\langle i,j,p\rangle}$ is an integer sequence whose difference sequence is an arithmetic progression that starts with $\mathsf{dup}(\langle i,j,p\rangle, i+p)$ and has a common difference -2.

By Lemma 3, $\mathcal{D}(w,k) = \sum_{\langle i,j,p\rangle \in \mathsf{Runs}(w,k)} \mathsf{dup}(\langle i,j,p\rangle,k)$ holds. For computing $\mathcal{D}(w,k)$ for increasing k, we maintain the following invariants:

$$\begin{aligned}
m_k &= |R_k|, \\
f_k &= \sum_{\langle i,j,p\rangle \in R_k} \mathsf{dup}(\langle i,j,p\rangle, i+p) = \sum_{\langle i,j,p\rangle \in R_k} num_{\langle i,j,p\rangle}[1], \\
d_k &= \sum_{\langle i,j,p\rangle \in R_k} (k-(i+p)), \\
e_k &= \sum_{\langle i,j,p\rangle \in R_k^2} \mathsf{dup}(\langle i,j,p\rangle, j-p) = \sum_{\langle i,j,p\rangle \in R_k^2} num_{\langle i,j,p\rangle}[1].
\end{aligned}$$

f_k is the sum of the first terms of $num_{\langle i,j,p\rangle}$, for $\langle i,j,p\rangle \in R_k$. d_k is the sum of the distances between k and $i+p$, for $\langle i,j,p\rangle \in R_k$. e_k is the sum of the last terms of $num_{\langle i,j,p\rangle}$, for $\langle i,j,p\rangle \in R_k^2$
.

By Lemma 4, for a run $\langle i,j,p\rangle$, $\mathsf{dup}(\langle i,j,p\rangle,k+1)$ can be maintained with the following recurrence:

$$\mathsf{dup}(\langle i,j,p\rangle,k+1) = \mathsf{dup}(\langle i,j,p\rangle,k) + num_{\langle i,j,p\rangle}[1] - 2(k-(i+p)).$$

This leads to the following recurrence for $\mathcal{D}(w,k+1)$:

$$\begin{aligned}
\mathcal{D}(w,k+1) &= \sum_{\langle i,j,p\rangle \in \mathsf{Runs}(w,k+1)} \mathsf{dup}(\langle i,j,p\rangle,k+1) \\
&= \sum_{\langle i,j,p\rangle \in R_k^1} \mathsf{dup}(\langle i,j,p\rangle,k+1) + \sum_{\langle i,j,p\rangle \in R_k^3} \mathsf{dup}(\langle i,j,p\rangle,k+1) \\
&= \sum_{\langle i,j,p\rangle \in R_k^1} (\mathsf{dup}(\langle i,j,p\rangle,k) + num_{\langle i,j,p\rangle}[1] - 2(k+1-(i+p))) \\
&\quad + \sum_{\langle i,j,p\rangle \in R_k^3} num_{\langle i,j,p\rangle}[1] \\
&= f_{k+1} + \sum_{\langle i,j,p\rangle \in R_k^1} (\mathsf{dup}(\langle i,j,p\rangle,k) - 2(k+1-(i+p))) \\
&= f_{k+1} + \sum_{\langle i,j,p\rangle \in R_k} \mathsf{dup}(\langle i,j,p\rangle,k) - \sum_{\langle i,j,p\rangle \in R_k^2} \mathsf{dup}(\langle i,j,p\rangle,k) \\
&\quad -2 \sum_{\langle i,j,p\rangle \in R_k^1} (k+1-(i+p)) \\
&= f_{k+1} + \mathcal{D}(w,k) - e_k - 2d_{k+1}.
\end{aligned}$$

Therefore, $\mathcal{D}(w,k+1)$ can be computed with this recurrence relation $\mathcal{D}(w,k+1) = \mathcal{D}(w,k) + f_{k+1} - e_k - 2d_{k+1}$. We show how to compute m_k, f_k, d_k, e_k. First, $m_1 = 0, f_1 = 0, d_1 = 0, e_1 = 0$ because $R_1 = \emptyset$ and $R_1^2 = \emptyset$. Then, $m_{k+1}, f_{k+1}, d_{k+1}$ can be computed from m_k, f_k, d_k as follows: m_{k+1} can be computed from m_k by adding the number of runs $\langle i,j,p\rangle$ such that $i+p = k+1$.

A pseudo-code of the proposed algorithm is shown in Algorithm 1. Below, we describe our algorithm.

Algorithm 1: Compute $\mathcal{C}(w,k)$ for all positions

Input: a string $w[1..n]$ over an ordered alphabet
Output: $\mathcal{C}(w,k)$ for all $k = 1, 2, ..., n$

Compute the sorted list L of the runs $\langle i,j,p\rangle \in \mathsf{Runs}(w)$ in increasing order of $i+p$;
Compute the sorted list R of the runs $\langle i,j,p\rangle \in \mathsf{Runs}(w)$ in increasing order of $j-p$;
for *each* $\langle l_1, r_1, p_1\rangle, \ldots, \langle l_{|L|}, r_{|L|}, p_{|L|}\rangle \in \mathsf{L}$ **do**
    $\mathsf{L}_l[q] \leftarrow l_q$, $\mathsf{L}_r[q] \leftarrow r_q$;
end
for *each* $\langle l_1, r_1, p_1\rangle, \ldots, \langle l_{|R|}, r_{|R|}, p_{|R|}\rangle \in \mathsf{R}$ **do**
    $\mathsf{R}_l[q] \leftarrow l_q$, $\mathsf{R}_r[q] \leftarrow r_q$;
end
$y \leftarrow 1, z \leftarrow 1, m \leftarrow 0, d \leftarrow 0, f \leftarrow 0, \mathcal{D}(w,0) \leftarrow 0$;
for *all* $k = 1, \ldots, n$ **do**
    $e \leftarrow 0$;
    $d \leftarrow d + m$;
    while $\mathsf{L}_l[y] = k$ **do**
        $f \leftarrow f + \mathsf{L}_r[y] - \mathsf{L}_l[y] + 1$;
        $m \leftarrow m + 1$;
        $y \leftarrow y + 1$;
    end
    while $\mathsf{R}_r[z] = k$ **do**
        $f \leftarrow f - (\mathsf{R}_r[z] - \mathsf{R}_l[z] + 1)$;
        $m \leftarrow m - 1$;
        $z \leftarrow z + 1$;
        $e \leftarrow e - (\mathsf{R}_r[z] - \mathsf{R}_l[z] + 1)$;
    end
    $d \leftarrow d - e$;
    $\mathcal{D}(w,k) \leftarrow \mathcal{D}(w,k-1) + f - 2d - e$;
    $\mathcal{C}(w,k) \leftarrow k(n-k+1) - \mathcal{D}(w,k)$;
end

For each run $r = \langle i,j,p\rangle$, we call the interval $[i+p, j-p]$ the run interval for r. To find runs by the starting and ending positions of their run intervals, we create two sorted lists L and R of pairs composed of positions and runs. The list L (resp. R) is sorted by the positions, which are the starting positions (resp. the ending positions) of the run intervals of the respective runs. L and R help us to access a run in amortized constant time, when we process the string positions $k = 1, \ldots, n$ in increasing order. The sorted lists L and R can be computed in linear time with an integer sorting algorithm.

f_{k+1} can be computed from f_k by adding $num_{i,j,p}[1]$ for runs $\langle i,j,p\rangle$ such that $i+p = k+1$ and subtracting $num_{i,j,p}[1]$ for runs $\langle i,j,p\rangle$ such that $j-p = k$.

We have $d_k = \sum_{\langle i,j,p\rangle \in R_k}(k - (i+p))$ and $d_{k+1} = \sum_{\langle i,j,p\rangle \in R_{k+1}}(k+1-(i+p))$, and therefore the sum increases by $|R_k| = m_k$ and decreases by $\sum_{\langle i,j,p\rangle \in R_k^2}(k+1-(i+p)) = \sum_{\langle i,j,p\rangle \in R_k^2}(j-p+1-(i+p)) = \sum_{\langle i,j,p\rangle \in R_k^2} num_{i,j,p}[1] = e_k$. This is why d_{k+1} can be computed by recurrence relation $d_{k+1} = d_k + m_k - e_k$.

Finally, e_k can be directly computed by summing the last term of $num_{\langle i,j,p\rangle}$ for runs $\langle i,j,p\rangle$ such that $j-p = k$.

4 Computing $\mathcal{N}(w, k)$ for all positions k in w

We show the following result.

Theorem 4. *Given a string $w[1..n]$ of length n over a linearly-sortable alphabet, we can sequentially output $\mathcal{N}(w, 1), \ldots, \mathcal{N}(w, n)$ such that the first value needs $O(n)$ time, but all subsequent values need constant-time delay.*

Let $A_x = \mathsf{Substr}(w[1..x])$ and $B_x = \mathsf{Substr}(w[x..n])$. Then, $\mathcal{N}(w, x) = |A_{x-1} \cup B_{x+1}|$. The idea is to compute, for increasing values of x, the two differences $|A_x \cup B_{x+1}| - |A_{x-1} \cup B_{x+1}|$ and $|A_x \cup B_{x+2}| - |A_x \cup B_{x+1}|$ so that $\mathcal{N}(w, x+1) = |A_x \cup B_{x+2}|$ can be computed from $\mathcal{N}(w, x)$ by adding these differences. If we can find the two differences in constant time for each x, then we can solve the addressed problem using the $O(n)$ textbook algorithm for $\mathcal{N}(w, 1)$.

We make use of the following two data structures that can be built in $O(n)$ time. The *longest previous non-overlapping factor table* (LPnF) of w is an integer array $\mathsf{LPnF}_w[1..n]$ whose i-th integer is the length of the longest prefix of $w[i..n]$ that has an occurrence in $w[1..i-1]$. The *longest next factor table* (LNF) of w is an integer array $\mathsf{LNF}_w[1..n]$ whose i-th integer is the length of the longest prefix of $w[i..n]$ that has an occurrence in $w[i+1..n]$.

Lemma 5 ([2], [4]). *We can build LPnF_w in $O(n)$ time.*

Lemma 6. *We can build LNF_w in $O(n)$ time.*

Proof. First, we build the suffix tree STree over w by Theorem 3. Next, we select sequentially the leaves of STree in ascending order with respect to their suffix numbers. For each such leaf λ with suffix number i, we move to its parent, write its string depth into $\mathsf{LNF}[i]$, delete λ, and continue the iteration. We keep the invariant that an internal node always has two children. In case that we deleted the penultimate leaf of a node, we merge this node with its remaining child. □

We first claim that having the arrays $\mathsf{LNF}_w, \mathsf{LPnF}_w$ for w at hand, $|A_x \cup B_{x+2}| - |A_x \cup B_{x+1}|$ can be computed in $O(1)$ time. Since $A_x \cup B_{x+2} \subseteq A_x \cup B_{x+1}$, we only need to count how many elements are removed, which must be prefixes of $w[x+1..n]$. The removed prefixes are the prefixes of $w[x+1..n]$ that do not occur in A_x and do not occur in B_{x+2}. From the definitions, $\alpha = \mathsf{LPnF}_w[i+1]$ is the length of the longest prefix of $w[x+1..n]$ that has an occurrence in $w[1..x]$ thus included in A_x, and $\beta = \mathsf{LNF}_w[i+1]$ is the length of the longest prefix of $w[x+1..n]$ that has an occurrence in $w[x+2..n]$ thus included in B_{x+2}. Therefore, the prefixes of $w[x+1..n]$ are removed if and only if they are longer than $\max(\alpha, \beta)$, and their number is $n - x - \max(\alpha, \beta)$.

The case for $|A_x \cup B_{x+1}| - |A_{x-1} \cup B_{x+1}|$ is symmetric and can be computed using the arrays LNF_{w^R} and LPnF_{w^R} for the reverse string w^R in a similar fashion.

Acknowledgments. This work was supported by JSPS KAKENHI Grant Numbers JP23K24808, JP23K18466 (SI), JP23H04378, JP25K21150 (DK), JP24K02899 (HB).

References

1. Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM J. Comput.*, 46(5):1501–1514, 2017.
2. Maxime Crochemore and Lucian Ilie. Computing longest previous factor in linear time and applications. *Inf. Process. Lett.*, 106(2):75–80, 2008.
3. Maxime Crochemore, Thierry Lecroq, and Wojciech Rytter. *125 Problems in Text Algorithms.* Cambridge University Press, Cambridge, 2021.

4. Maxime Crochemore and German Tischler. Computing longest previous non-overlapping factors. *Inf. Process. Lett.*, 111(6):291–295, 2011.
5. Jonas Ellert and Johannes Fischer. Linear time runs over general ordered alphabets. In *ICALP 2021*, volume 198 of *LIPIcs*, pages 63:1–63:16, 2021.
6. Martin Farach-Colton, Paolo Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *J. ACM*, 47(6):987–1011, 2000.
7. Nathan J Fine and Herbert S Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965.
8. Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In *STOC 2018*, pages 827–840. ACM, 2018.
9. Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *FOCS 1999*, pages 596–604, 1999.
10. Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory*, pages 1–11, 1973.