Breaking the $\mathcal{O}(n)$-Barrier in the Construction of Compressed Suffix Arrays

Dominik Kempa*
Johns Hopkins University
kempa@cs.jhu.edu

Tomasz Kociumaka†
University of California, Berkeley
kociumaka@berkeley.edu

Abstract

The suffix array, describing the lexicographic order of suffixes of a given text, is the central data structure in string algorithms, with dozens of applications in data compression, bioinformatics, and information retrieval. The suffix array of a length-n text uses $\Theta(n \log n)$ bits, which is prohibitive in many applications. To address this, Grossi and Vitter [STOC 2000] and, independently, Ferragina and Manzini [FOCS 2000] introduced space-efficient versions of the suffix array, known as the *compressed suffix array* (CSA) and the *FM-index*. For a length-n text over an alphabet of size σ, these data structures use only $\mathcal{O}(n \log \sigma)$ bits. Immediately after their discovery, they almost completely replaced plain suffix arrays in practical applications, and a race started to develop efficient construction procedures. Yet, after more than 20 years, even for $\sigma = 2$, the fastest algorithm remains stuck at $\mathcal{O}(n)$ time [Hon et al., FOCS 2003], which is slower by a $\Theta(\log n)$ factor than the lower bound of $\Omega(n/\log n)$ (following simply from the necessity to read the entire input).

We break this long-standing barrier with a new data structure that takes $\mathcal{O}(n \log \sigma)$ bits, answers suffix array queries in $\mathcal{O}(\log^{\epsilon} n)$ time, and can be constructed in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time using $\mathcal{O}(n \log \sigma)$ bits of space. Our solution matches the size and the query time of the CSA and the FM-index but, unlike these two, admits a sublinear-time construction for small alphabets. (For example, if $\sigma = 2$, then it can be built in $\mathcal{O}(n/\sqrt{\log n})$ time.) Our result is based on several new insights into the recently developed notion of *string synchronizing sets* [STOC 2019]. In particular, compared to their previous applications, we eliminate orthogonal range queries, replacing them with new queries that we dub *prefix rank* and *prefix selection queries*. As a further demonstration of our techniques, we present a new pattern-matching index that *simultaneously* minimizes the construction time and the query time among all known compact indexes (i.e., those using $\mathcal{O}(n \log \sigma)$ bits).

1 Introduction

For a text T of length n, the suffix array $\mathrm{SA}[1\,..\,n]$ stores the permutation of $\{1, \ldots, n\}$ such that $\mathrm{SA}[i]$ is the starting position of the ith lexicographically smallest suffix of T. The simplest application of SA is as a *text index*: given any pattern $P[1\,..\,m]$, the suffix array lets us find all occurrences of P in T using only $\mathcal{O}(m \log n)$ operations [47]. We simply perform a binary search in SA, resulting in a range $[b\,..\,e)$ of suffixes of T having P as a prefix. Then, $\mathrm{SA}[b\,..\,e)$ contains the starting positions of all occurrences of P in T. The simplicity, space-efficiency, and elegance of SA (often augmented with the *LCP array* [47, 38] storing the longest common prefixes of adjacent suffixes) has led to its widespread use in applications that benefit from having the lexicographic order of suffixes. As evidenced from the classical textbook of Gusfield [34], as well as more recent

*Supported by NIH HG011392, NSF DBI-2029552, 1652303, 1934846, and an Alfred P. Sloan Fellowship.
†Partly supported by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship.

textbooks on string processing [45, 52, 59], this turns out to include a vast set of problem, ranging from finding repetitions (MaximalRepeats, LongestRepeatedFactor, TandemRepeats) and special subwords (MinimalAbsentWord, ShortestUniqueSubstring), to sequence comparisons (LongestCommonSubstring, MatchingStatistics, MaximalUniqueMatches) and data compression (LZ77Factorization, CdawgCompression).[1] There are even textbooks dedicated entirely to suffix arrays and the closely related Burrows–Wheeler transform (BWT) [1].

With the increasing size of datasets that need processing, even suffix arrays, however, have become expensive to use, particularly in applications where the input text is over a small alphabet $[0\,..\,\sigma)$. Such text requires $n\lceil\log\sigma\rceil$ bits, whereas the suffix array always uses $n\lceil\log n\rceil$ bits of space, regardless of σ. Depending on the application, the gap $\frac{\log n}{\log\sigma}$ can be quite large, e.g., in computational biology, where we usually have $\sigma = 4$, the gap is typically between 16 and 32. This shortcoming was addressed by Grossi and Vitter [32, 33] and, independently, Ferragina and Manzini [22, 23] at the turn of the millennium. They introduced space-efficient versions of the suffix array, known as the *compressed suffix array (CSA)* and the *FM-index*. For a length-n text over an alphabet of size σ, these data structures use $\mathcal{O}(n\log\sigma)$ bits, and they can answer SA queries (asking for $\mathrm{SA}[i]$ given $i \in [1\,..\,n]$) in $\mathcal{O}(\log^{\epsilon} n)$ time, where $\epsilon > 0$ is an arbitrary predefined constant. With such data structure, we can execute any algorithm that uses the suffix array, but consuming less space and only incurring a factor of $\mathcal{O}(\log^{\epsilon} n)$ penalty in the runtime.[2] Shortly after their discovery, Sadakane [62] developed a space-efficient representation of the LCP array and, eventually extended the compressed suffix array into a *compressed suffix tree (CST)* [63]. This powerful structure can be plugged into an even larger set of algorithms [29].

Nowadays, CSAs and CSTs are widely used in practice. Modern string algorithms textbooks focus on the use and applications of CSAs and related data structures [45], or even entirely on the emerging notion of *compressed data structures* [52]. The FM-index occupies the central role in some of the most commonly used bioinformatics tools, like `Bowtie` [42], `BWA` [43], and `Soap2` [44], and mature and highly engineered implementations of CSAs and CSTs are available through the `sdsl` library[3] of Gog et al. [30, 29]. Despite these developments in functionality and practical adoption of CSAs, fast construction is the aspect of these structures that remained beyond the reach of contemporary techniques. The original paper of Grossi and Vitter [32], describes a method, that given a length-n text over alphabet $\Sigma = [0\,..\,\sigma)$, constructs the CSA in $\mathcal{O}(n\log\sigma)$ time and using $\mathcal{O}(n\log n)$ bits of working space. In 2003, a celebrated result of Hon et al. [36] lowered the time complexity to $\mathcal{O}(n\log\log\sigma)$ and the space to the optimal $\mathcal{O}(n\log\sigma)$ bits. For the most challenging case of $\sigma = 2$, however, this algorithm still runs in $\Theta(n)$ time, which is slower by a $\Theta(\log n)$ factor than the lower bound of $\Omega(n/\log n)$, following simply from the necessity to read the entire input. Recently, Belazzougui [4] and, independently, Munro et al. [48], improved the time complexity of the CSA/CST construction to $\mathcal{O}(n)$ (while using the optimal space of $\mathcal{O}(n\log\sigma)$ bits), making it independent of the alphabet size σ. Despite these advances, 18 years after the result of Hon et al. [36], the bound of $\Omega(n)$ still stands on the construction of CSAs for the hardest case $\sigma = 2$. Given the versatile applications and the wide adoption of compressed suffix array, we study the following problem:

Problem 1.1. *Given a text over alphabet $\Sigma = [0\,..\,\sigma)$ represented using $\mathcal{O}(n\log\sigma)$ bits, can we construct a compressed suffix array of T in $o(n)$ time and using $\mathcal{O}(n\log\sigma)$ bits of space?*

Our Contribution We answer the question posed in Problem 1.1 affirmatively by describing a new data structure that takes $\mathcal{O}(n\log\sigma)$ bits, answers the SA queries in $\mathcal{O}(\log^{\epsilon} n)$ time, and

[1] We omit the formal definitions of these classical problems, and refer the reader to the cited textbooks.
[2] This is often acceptable: a slower algorithm remains usable, but insufficient memory can thwart it entirely.
[3] The latest version is available at `https://github.com/simongog/sdsl-lite`.

can be constructed in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time using $\mathcal{O}(n \log \sigma)$ bits of space. Thus, our solution matches the size and the query time of the CSA and the FM-index but, unlike them, admits a sublinear-time construction for small σ. For example, it can be built in $\mathcal{O}(n/\sqrt{\log n}) = o(n)$ time if $\sigma = 2$, which constitutes the first improvement since the 2003 result of Hon et al. [36].

Our data structure differs significantly from the CSA of Grossi and Vitter [32] and the FM-index [22], which are based on the so-called Ψ function [32] and the Burrows–Wheeler transform [13], respectively. Instead, we utilize the recently developed notion of *string synchronizing sets (SSS)* [39]. The work defining SSS laid out its basic properties sufficient to construct BWT, but it cannot be easily turned into a CSA, because it heavily relies on *orthogonal range counting* queries [15], which are *provably not capable* of supporting SA queries fast enough: Pătraşcu [60] showed a lower bound $\Omega(\frac{\log n}{\log \log n})$ on the query time of any structure using near-linear space.

By a novel analysis of string synchronizing sets, we side-step these obstacles and demonstrate that *general* orthogonal range counting queries [16, 15] are in fact not needed at all. We show that each of their uses can either be: (1) eliminated completely (see Proposition 3.9), or (2) replaced with new queries that we dub *prefix rank* and *prefix selection queries* (see Section 2.1), or (3) improved, utilizing the fact that the instances arising in our construction have properties that permit a fast custom solution (see Proposition 3.11 applied in the proof of Proposition 3.12).

After presenting the CSA, we turn our attention to the closely related problem of text indexing [55]. Using our new techniques, we show (see Theorem 5.21) how, given a length-n text T stored using $\mathcal{O}(n \log \sigma)$ bits, to construct in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time an index of size $\mathcal{O}(n \log \sigma)$ bits that, given the packed representation (i.e., using $\mathcal{O}(m \log \sigma)$ bits) of any pattern $P[1\,..\,m]$, counts the occurrences of P in T in $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$ time (where $\epsilon \in (0,1)$ is an arbitrary predefined constant). The best previous solutions using the compact space (i.e., $\mathcal{O}(n \log \sigma)$ bits) achieve $\mathcal{O}(n \log \sigma / \sqrt{\log n})$-time[4] construction and $\mathcal{O}(m/\log_\sigma n + \log n \cdot \log_\sigma n)$-time queries [49], or $\mathcal{O}(n)$-time construction and $\mathcal{O}(m/\log_\sigma n + \log^\epsilon_\sigma n)$-time queries [50]. Thus, for the most difficult case of $\sigma = 2^{\mathcal{O}(\sqrt{\log n})}$, our construction subsumes both these indexes in both aspects.[5]

Related Work In recent years, there has also been progress in the query time of $\mathcal{O}(n \log n)$-bit pattern matching indexes. The original $\mathcal{O}(m \log n)$-time pattern search via SA came already with an $\mathcal{O}(m+\log n)$-time variant [47]. This was further reduced to $\mathcal{O}(m+\log \sigma)$ by Cole et al. [19], and to $\mathcal{O}(m+\log \log \sigma)$ by Fischer and Gawrychowski [24]. Finally, Navarro and Nekrich [56] achieved $\mathcal{O}(m)$ time. If the pattern is given using $\mathcal{O}(m \log \sigma)$ bits, Bille et al. [11] achieve $\mathcal{O}(m/\log_\sigma n + \log m + \log \log \sigma)$ time, which Navarro and Nekrich [56] improved to $\mathcal{O}(m/\log_\sigma n + 1)$.

Surprisingly, the size of some CSAs, CSTs, and compact indexes can be reduced *below* $n\lceil \log \sigma \rceil$ bits for statistically compressible texts. For example, already the original FM-index [22] takes only $\mathcal{O}(nH_k(T)) + o(n \log \sigma)$ bits, where $H_k(T)$ denotes the *empirical kth-order entropy* of text [20]. Currently, the smallest indexes reach $nH_k(T) + o(n(H_k(T)+1))$ bits [3, 8]. Navarro and Mäkinen [55], and Belazzougui and Navarro [7] survey the achievable trade-offs for such *fully compressed* indexes, Chan et al. [14], and Mäkinen and Navarro [46] describe *dynamic* compressed pattern-matching indexes maintaining a collection of texts supporting insertions and deletions.

Compressed indexes based on LZ77 [64] and run-length BWT [13] rapidly gain popularity. The early indexes [6, 10, 12, 25, 26] support only pattern search and random-access operations. Subsequent works generalized them to other dictionary compressors [17, 40, 57] and added dynamism [28, 58]. Support for SA queries is a recent addition of Gagie et al. [27]. Navarro surveys these indexes [54] and the intricate network of the underlying compressibility measures [53]. Interestingly, some of these pattern matching indexes can be constructed in compressed time.

[4] Although CSA lets us implement pattern counting queries, an index implementing pattern counting queries does *not* let us implement SA queries; thus, although built in $o(n)$ time, [49] cannot be used to answer SA queries.

[5] Note that $\log^\epsilon_\sigma n = \Theta(\log^{\epsilon'} n)$ with $\epsilon' = \frac{\epsilon}{2}$ holds if $\log \sigma = \mathcal{O}(\sqrt{\log n})$.

For example, the index of [28] can be constructed in $\mathcal{O}(z \log^3 n)$ from the LZ77 representation of T (with z phrases), and then it locates pattern occurrences in $\mathcal{O}(m + \mathsf{occ} \log n)$ time. On the other hand, the only index supporting SA queries [27] is only constructible in $\Omega(n)$ time, but it can be built in compressed space $\mathcal{O}(r \log(n/r))$ given the run-length BWT of T (with r runs).

2 Preliminaries

A *string* is a finite sequence of characters from a given *alphabet*. The length of a string S is denoted $|S|$. For $i \in [1\,..\,|S|]$,[6] the ith character of S is denoted $S[i]$. A *substring* of S is a string of the form $S[i\,..\,j) = S[i]S[i+1]\cdots S[j-1]$ for some $1 \le i \le j \le |S|+1$. *Prefixes* and *suffixes* are of the form $S[1\,..\,j)$ and $S[i\,..\,|S|]$, respectively. We use $\overline{S}$ to denote the *reverse* of S. We denote the *concatenation* of two strings U and V by UV or $U \cdot V$. Furthermore, $S^k = \bigodot_{i=1}^{k} S$ is the concatenation of k copies of S; note that $S^0 = \varepsilon$ is the *empty string*. An integer $p \in [1\,..\,|S|]$ is a *period* of S if $S[1\,..\,|S|-p] = S[1+p\,..\,|S|]$; we denote the shortest period as $\mathrm{per}(S)$.

Throughout the paper, we consider a string (called the *text*) T of length $n \ge 1$ over an integer alphabet $\Sigma = [0\,..\,\sigma)$, where $\sigma = n^{\mathcal{O}(1)}$. We use $\preceq$ to denote the order on Σ, extended to the *lexicographic* order on Σ^* (the set of strings over Σ) so that $U, V \in \Sigma^*$ satisfy $U \preceq V$ if and only if either U is a prefix of V, or $U[1\,..\,i) = V[1\,..\,i)$ and $U[i] \prec V[i]$ holds for some $i \in [1\,..\,\min(|U|,|V|)]$.

The *suffix array* $\mathrm{SA}[1\,..\,n]$ of T is a permutation of $[1\,..\,n]$ such that $T[\mathrm{SA}[1]\,..\,n] \prec T[\mathrm{SA}[2]\,..\,n] \prec \cdots \prec T[\mathrm{SA}[n]\,..\,n]$, i.e., $\mathrm{SA}[i]$ is the starting position of the lexicographically ith suffix of T; see Fig. 1. The *inverse suffix array* $\mathrm{ISA}[1\,..\,n]$ is the inverse permutation of SA, i.e., $\mathrm{ISA}[j] = i$ holds if and only if $\mathrm{SA}[i] = j$. Intuitively, $\mathrm{ISA}[j]$ tells the *rank* of a suffix $T[j\,..\,n]$ among suffixes of T. By $\mathrm{lcp}(U,V)$ we denote the length of the longest common prefix of U and V. For $j_1, j_2 \in [1\,..\,n]$, we let $\mathrm{LCE}(j_1,j_2) = \mathrm{lcp}(T[j_1\,..],T[j_2\,..])$.

i	$\mathrm{SA}[i]$	$T[\mathrm{SA}[i]\,..\,n]$
1	19	a
2	14	aababa
3	5	aababababaababa
4	17	aba
5	12	abaababa
6	3	abaababababaababa
7	15	ababa
8	10	ababaababa
9	8	abababaababa
10	6	abababababaababa
11	18	ba
12	13	baababa
13	4	baababababaababa
14	16	baba
15	11	babaababa
16	2	babaababababaababa
17	9	bababaababa
18	7	babababaababa
19	1	bbabaababababaababa

Figure 1: A list of of all sorted suffixes of $T =$ bbabaababababaababa along with the suffix array.

We use the word RAM model of computation [35] with w-bit *machine words*, where $w \ge \log n$. In this model, strings are typically represented as arrays, with each character occupying a single memory cell. A single character, however, only needs $\lceil \log \sigma \rceil$ bits, which might be much less than w. We can therefore store (the *packed representation* of) a text $T \in [0\,..\,\sigma)^n$ using $\mathcal{O}\big(\lceil \frac{n \log \sigma}{w} \rceil\big)$ memory cells.

2.1 (Prefix) Rank and Selection Queries

Let us recall the (ordinary) rank and selection queries on a string $S \in \Sigma^n$.

Rank query $\mathsf{rank}_{S,a}(j)$: Given $a \in \Sigma$ and $j \in [1\,..\,n]$, compute $|\{i \in [1\,..\,j] : S[i] = a\}|$.

Selection query $\mathsf{select}_{S,a}(r)$: Given $a \in \Sigma$ and $r \in [1\,..\,\mathsf{rank}_{S,a}(n)]$, find the rth smallest element of $\{i \in [1\,..\,n] : S[i] = a\}$.

Theorem 2.1 (Rank and selection queries in bitvectors [37, 18, 51, 2])**.** *For every string $S \in \{0,1\}^*$, there exists a data structure of $\mathcal{O}(|S|)$ bits answering rank and selection queries in $\mathcal{O}(1)$ time. Moreover, given the packed representations of m strings of total length n, the data structures for all these strings can be constructed in $\mathcal{O}(m + n/\log n)$ time.*

[6] For $i, j \in \mathbb{Z}$, denote $[i\,..\,j] = \{k \in \mathbb{Z} : i \le k \le j\}$, $[i\,..\,j) = \{k \in \mathbb{Z} : i \le k < j\}$, and $(i\,..\,j] = \{k \in \mathbb{Z} : i < k \le j\}$.

Next, we provide a generalization of rank and selection queries specific to sequences of strings (strings whose characters are strings themselves). Let $W \in (\Sigma^*)^m$ be a sequence of m strings.

Prefix rank query $\mathsf{rank}_{W,X}(j)$**:** Given $X \in \Sigma^*$ and $j \in [1\,..\,m]$, compute $|\{i \in [1\,..\,j] : X \text{ is a prefix of } W[i]\}|$.

Prefix selection query $\mathsf{select}_{W,X}(r)$**:** Given $X \in \Sigma^*$ and $r \in [1\,..\,\mathsf{rank}_{W,X}(m)]$, find the rth smallest element of $\{i \in [1\,..\,m] : X \text{ is a prefix of } W[i]\}$.

The following result, proved in Appendix A by building on the results of Belazzougui and Puglisi [9], provides an efficient implementation of prefix rank and selection queries. Note that we require W to consists of same-length strings over an integer alphabet.

Theorem 2.2. *For all integers $m, b, \sigma \in \mathbb{Z}_{\geq 1}$ satisfying $m \geq \sigma^b \geq 2$, every constant $\epsilon \in (0, 1)$, and every string $W \in ([0\,..\,\sigma)^b)^{\leq m}$, there exists a data structure of size $\mathcal{O}(m)$ answering prefix rank and selection queries in $\mathcal{O}(\log^\epsilon m)$ time. Moreover, it can be constructed in $\mathcal{O}(m\sqrt{\log m})$ time using $\mathcal{O}(m)$ working space given the packed representation of W and the parameter ϵ.*

2.2 String Synchronizing Sets

Definition 2.3 (τ-synchronizing set [39])**.** Let $T \in \Sigma^n$ be a string and let $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$ be a parameter. A set $\mathsf{S} \subseteq [1\,..\,n - 2\tau + 1]$ is called a *τ-synchronizing set* of T if it satisfies the following *consistency* and *density* conditions:

1. If $T[i\,..\,i+2\tau) = T[j\,..\,j+2\tau)$, then $i \in \mathsf{S}$ holds if and only if $j \in \mathsf{S}$ (for $i, j \in [1\,..\,n-2\tau+1]$),
2. $\mathsf{S} \cap [i\,..\,i+\tau) = \emptyset$ if and only if $\mathrm{per}(T[i\,..\,i+3\tau-2]) \leq \frac{1}{3}\tau$ (for $i \in [1\,..\,n-3\tau+2]$).

In most applications, we want to minimize $|\mathsf{S}|$. Note, however, that the density condition imposes a lower bound $|\mathsf{S}| = \Omega(\frac{n}{\tau})$ for strings of length $n \geq 3\tau - 1$ that do not contain substrings of length $3\tau - 1$ which are periodic with period $\leq \frac{1}{3}\tau$. Thus, we cannot hope to achieve an upper bound improving in the worst case upon the following one.

Theorem 2.4 ([39, Theorem 8.10])**.** *For any string T of length n and parameter $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$, there exists a τ-synchronizing set S of size $|\mathsf{S}| = \mathcal{O}\left(\frac{n}{\tau}\right)$. Moreover, if $T \in [0\,..\,\sigma)^n$, where $\sigma = n^{\mathcal{O}(1)}$, such S can be (deterministically) constructed in $\mathcal{O}(n)$ time.*

Note, that when $\tau = \omega(1) \cap \mathcal{O}(\log_\sigma n)$ and $T \in [0\,..\,\sigma)^n$ is given in the packed representation, the first part of the above result opens the possibility of an algorithm running in $\mathcal{O}(\frac{n}{\tau}) = o(n)$ time. In [39], it was shown that this lower bound is achievable (the upper bound $\tau = \mathcal{O}(\log_\sigma n)$ follows from the fact that every algorithm needs to at least read the input, which takes $\mathcal{O}(n/\log_\sigma n)$ time; thus, for larger τ, the algorithm cannot run in $\mathcal{O}(\frac{n}{\tau})$ time).

Theorem 2.5 ([39, Theorem 8.11])**.** *For every constant $\mu < \frac{1}{5}$, given the packed representation of a text $T \in [0\,..\,\sigma)^n$ and a positive integer $\tau \leq \mu \log_\sigma n$, one can (deterministically) construct in $\mathcal{O}(\frac{n}{\tau})$ time a τ-synchronizing set of size $\mathcal{O}(\frac{n}{\tau})$.*

3 ISA Queries

In this section, we provide an index answering ISA queries only. However, the infrastructure developed for this index is defined so that it can be re-used in Sections 4 and 5, where we focus on SA queries and pattern matching queries, respectively. For now, our goal is to pre-process the packed representation of a text $T \in [0\,..\,\sigma)^n$ in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time and $\mathcal{O}(n/\log_\sigma n)$ space so that, given $j \in [1\,..\,n]$, one can compute $\mathrm{ISA}[j]$ in $\mathcal{O}(\log^\epsilon n)$ time. Note that the plain

representation of $\mathrm{ISA}[1\,..\,n]$ can be constructed in $\mathcal{O}(n)$ time and space, which satisfies the announced complexity guarantees if $\sigma = n^{\Omega(1)}$. Hence, we hereafter assume $\sigma < n^{1/6}$.

Let $\tau = \mu \log_\sigma n$, where μ is any positive constant smaller than $\frac{1}{6}$ (such μ exists by $\sigma < n^{1/6}$). We fix a τ-synchronizing set S of T obtained using Theorem 2.5 and use the following notation:

- $n' := |\mathsf{S}| = \mathcal{O}(n/\tau)$ is the size of S;
- $\mathrm{succ}_\mathsf{S}(i) := \min\{j \in \mathsf{S} \cup \{n - 2\tau + 2\} : j \geq i\}$ for any $i \in [1\,..\,n - 2\tau + 1]$;
- $\mathsf{R} := \{i \in [1\,..\,n - 3\tau + 2] : \mathrm{per}(T[i\,..\,i + 3\tau - 2]) \leq \frac{1}{3}\tau\}$;
- $\mathcal{D} := \{D_i : i \in [1\,..\,n - 3\tau + 2] \setminus \mathsf{R}\}$, where $D_i := T[i\,..\,\mathrm{succ}_\mathsf{S}(i) + 2\tau)$,
- $\mathcal{F} := \{T[i\,..\,i + 3\tau - 2] : i \in \mathsf{R}\}$.

The set $\mathcal{D} \cup \mathcal{F}$ plays a very important role in our indexes. Observe that any sufficiently long suffix of T has a prefix in either $\mathcal{D}$ or $\mathcal{F}$. On the other hand, [39, Lemma 6.1] shows that $\mathcal{D}$ is prefix-free, i.e., no string $X \in \mathcal{D}$ is a prefix of another $X' \in \mathcal{D}$. Moreover, by the density condition of S, no string in $\mathcal{D}$ is a prefix of any string $X \in \mathcal{F}$. Thus, the set $\mathcal{D} \cup \mathcal{F}$ is prefix-free and, consequently, all suffixes in the SA of T can be partitioned into blocks according to their prefix from $\mathcal{D} \cup \mathcal{F}$. Letting $X \in \mathcal{D} \cup \mathcal{F}$ for any $j \in [1\,..\,n]$ be the prefix of $T[j\,..\,n]$, we define $\mathrm{pos}(j) = \{j' \in [1\,..\,n] : \mathrm{LCE}_T(j, j') \geq |X| \text{ and } T[j'\,..\,n] \preceq T[j\,..\,n]\}$, and denote $\delta(j) := |\mathrm{pos}(j)|$.

Organization The section is organized as follows. Our description is split into four parts. First (Section 3.1), we describe the data structures, called collectively the index "core", that enable efficiently checking if $j \in \mathsf{R}$, and to compute the prefix $X \in \mathcal{D} \cup \mathcal{F}$ as well as the endpoints of the corresponding block $\mathrm{SA}(b\,..\,e]$. The structure and query algorithm to compute $\delta(j)$ (and thus $\mathrm{ISA}[j] = b + \delta(j)$) is different depending on whether $X \in \mathcal{D}$ (i.e., $j \in [1\,..\,n] \setminus \mathsf{R}$; we call such positions *nonperiodic*) or $X \in \mathcal{F}$ (i.e., $j \in \mathsf{R}$, and such positions are called *periodic*), and the structures used for the two cases are described in the next two parts (Sections 3.2 and 3.3). All ingredients are finally put together in Section 3.4.

3.1 The Index Core

We use the following definitions. Let $B[1\,..\,n]$ be a bitvector defined so that $B[i] = 1$ holds if and only if $i \in \mathsf{S}$. Define $\mathrm{ISA}_{3\tau-1}$ to be a mapping from $X \in [0\,..\,\sigma)^{\leq 3\tau-1} := \{\varepsilon\} \cup [0\,..\,\sigma) \cup \ldots \cup [0\,..\,\sigma)^{3\tau-1}$ to the pair of integers (b, e) such that $b = |\{i \in [1\,..\,n] : T[i\,..\,n] \prec X\}|$ and $e - b = |\{i \in [1\,..\,n] : X \text{ is a prefix of } T[i\,..\,n]\}|$. Note, that if $b \neq e$, these conditions are equivalent to $\mathrm{SA}(b\,..\,e]$ containing the starting positions of all suffixes of T that have X a prefix. When accessing $\mathrm{ISA}_{3\tau-1}$, the strings $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$ are injectively converted to small integers as follows. For any $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, we first append $6\tau - 2|X|$ zeros and $|X|$ cs (where $c = \sigma - 1$) to X, and then we interpret the resulting string as a base-σ representation of an integer in $[0\,..\,\sigma^{6\tau})$. We denote it $\mathrm{int}(X)$. It is easy to check that $X \neq X'$ implies $\mathrm{int}(X) \neq \mathrm{int}(X')$.

The index core, denoted $\mathrm{C}_{\mathrm{ISA}}(T, \mathsf{S})$, consists of three components. First, we store the packed representation of T using $\mathcal{O}(n/\log_\sigma n)$ space. Second, we store the bitvector B augmented to answer $\mathcal{O}(1)$-time rank and selection queries (Theorem 2.1). The bitvector takes $\mathcal{O}(n)$ bits, i.e., $\mathcal{O}(n/\log n)$ space. Finally, we store the lookup table $\mathrm{ISA}_{3\tau-1}$. By $\mathrm{int}(X) \in [0\,..\,\sigma^{6\tau})$ (where $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$), we can store the mapping $\mathrm{ISA}_{3\tau-1}$ in $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n^{6\mu}) = \mathcal{O}(n/\log n)$ space.

Lemma 3.1. *Given $\mathrm{C}_{\mathrm{ISA}}(T, \mathsf{S})$, for any $j \in [1\,..\,n]$ we can in $\mathcal{O}(1)$ time determine if $j \in \mathsf{R}$, compute the prefix $X \in \mathcal{D} \cup \mathcal{F}$ of $T[j\,..\,n]$, and integers b, e such that $\mathrm{SA}(b\,..\,e]$ contains the starting indexes of all suffixes of T prefixed with X.*

Proof. By the density condition, we have $j \in \mathsf{R}$ if and only if $\mathrm{succ}_\mathsf{S}(j) - j \geq \tau$. Thus, given the position $j \in [1\,..\,n]$, we first obtain $x = \mathsf{rank}_{B,1}(j - 1)$. The position $s = \mathsf{select}_{B,1}(x + 1)$ is then equal to $\mathrm{succ}_\mathsf{S}(j)$. If it holds $s - j \geq \tau$, then we have $j \in \mathsf{R}$ and $X = T[j\,..\,j + 3\tau - 1) \in$

$\mathcal{F}$. Otherwise, we have $j \in [1\,..\,n] \setminus \mathsf{R}$ and $X = T[j\,..\,s+2\tau) \in \mathcal{D}$. Finally, we obtain $\mathrm{ISA}_{3\tau-1}[\mathrm{int}(X)] = (b, e)$. It is easy to see that all operations, including the computation of $\mathrm{int}(X)$ from the packed representation of X, take $\mathcal{O}(1)$ time. □

Proposition 3.2. *Given the packed representation of $T \in [0\,..\,\sigma)^n$ and the array containing elements of S, we can construct $\mathrm{C}_{\mathrm{ISA}}(T, \mathsf{S})$ in $\mathcal{O}(n/\log_\sigma n)$ time.*

Proof. Given S, we easily initialize the bitvector B in $\mathcal{O}(n/\log_\sigma n)$ time; this time also suffices to augment B using Theorem 2.1.

To compute $\mathrm{ISA}_{3\tau-1}$, we first compute for every $X \in [0\,..\,\sigma)^\ell$ (where $\ell = 3\tau-1$), its frequency $f_X := |\{i \in [1\,..\,n] : X \text{ is a prefix of } T[i\,..\,n]\}|$. Using the simple generalization of the algorithm described in [39, Section 6.1.2], this takes $\mathcal{O}(n/\log_\sigma n)$ time (note that the algorithm requires $\ell\sigma^{2\ell+1} = O(n/\log_\sigma n)$, which is satisfied here, since $2\ell+1 < 6\mu\log_\sigma n$ and $\mu < \frac{1}{6}$). From the frequencies of $X \in [0\,..\,\sigma)^{3\tau-1}$ we then compute the values of f_X for all $X \in [0\,..\,\sigma)^{<3\tau-1}$ by observing that unless X is a nonempty suffix of T, it holds $f_X = \sum_{c\in[0\,..\,\sigma)} f_{Xc}$, i.e., the frequency of each string shorter than $3\tau-1$ is obtained in $\mathcal{O}(\sigma)$ time. If X is a nonempty suffix of T (which we can check in $\mathcal{O}(1)$ time), we additionally add one to the count. Since each string contributes exactly once to the frequency of another string, over all $X \in [0\,..\,\sigma)^{<3\tau-1}$, this takes $\mathcal{O}(\sigma^{3\tau-1}) = \mathcal{O}(n/\log_\sigma n)$ time. Once f_X is computed for all $X \in [0\,..\,\sigma)^{\le 3\tau-1}$, we compute $\mathrm{ISA}_{3\tau-1}$ as follows. Denote $\Sigma = [0\,..\,\sigma)$. Assume that $\mathrm{ISA}_{3\tau-1}[\mathrm{int}(X)] = (b, e)$ holds for some $X \in [0\,..\,\sigma)^{<3\tau-1}$. Then, for any $c \in \Sigma$, it holds $\mathrm{ISA}_{3\tau-1}[\mathrm{int}(Xc)] = (e - x - f_{Xc}, e - x)$, where $x = \sum_{c'\in\Sigma, c'>c} f_{Xc'}$, e.g., for $\sigma = 2$, $\mathrm{ISA}_{3\tau-1}[\mathrm{int}(X0)] = (e - f_{X1} - f_{X0}, e - f_{X1})$. We thus compute $\mathrm{ISA}_{3\tau-1}[\mathrm{int}(X)]$ by initializing $\mathrm{ISA}_{3\tau-1}[\mathrm{int}(\varepsilon)] = (0, n)$, and then enumerating all $X \in [0\,..\,\sigma)^{\le 3\tau-1}$ in the order of nondecreasing length (and, in case of ties, in the reverse lexicographical order). During the enumeration of strings of the form Xc, where $c \in \Sigma$, we maintain the sum $x = \sum_{c'\in\Sigma, c'>c} f_{Xc'}$. Then, using the above formula, the value of $\mathrm{ISA}_{3\tau-1}[\mathrm{int}(Xc)]$ can be obtained in $\mathcal{O}(1)$ time. Over all X, the computation of $\mathrm{ISA}_{3\tau-1}[\mathrm{int}(X)]$ thus takes $\mathcal{O}(\sigma^{3\tau-1}) = \mathcal{O}(n/\log_\sigma n)$ time. □

3.2 The Nonperiodic Positions

In this section, we describe a data structure to compute the value $\mathrm{ISA}[j]$ for any $j \in [1\,..\,n] \setminus \mathsf{R}$.

The main idea of the data structure is as follows. Let $X \in \mathcal{D}$ be the prefix of $T[j\,..\,n]$. By consistency of S, the offset of the first position from S in every suffix prefixed with X is $\delta_{\mathrm{text}} = |X| - 2\tau$. Thus, the order of these suffixes is the same as the order of corresponding suffixes starting at positions of S. Hence, it suffices to sort only those suffixes, and then computing $\delta(j)$ reduces to counting the number of positions in S that occur earlier in the sorted order than the position corresponding to j, and are followed by $X(\delta_{\mathrm{text}}\,..\,|X|]$ and preceded with $X[1\,..\,\delta_{\mathrm{text}}]$ in the text. We show how to implement that query using Theorem 2.2.

We use the following definitions. Let $(s_t)_{t\in[1\,..\,n']}$ be the sequence containing the elements of S in sorted order, i.e., if $i < j$ then $s_i < s_j$. Let $(s'_t)_{t\in[1\,..\,n']}$ be the sequence containing elements of S sorted according to the lexicographical order of the corresponding suffixes, i.e., if $i < j$ then $T[s'_i\,..\,n] \prec T[s'_j\,..\,n]$. Let $\mathrm{ISA}_{\mathsf{S}}[1\,..\,n']$ be an array storing a permutation of $[1\,..\,n']$ such that $\mathrm{ISA}_{\mathsf{S}}[j] = i$ holds if and only if $s_j = s'_i$. Finally, let $W[1\,..\,n']$ be a sequence of length-3τ strings such that $W[i] = \overline{X_i}$, where $X_i = T^\infty[s'_i - \tau\,..\,s'_i + 2\tau)$.

Lemma 3.3. *Let $j \in [1\,..\,n] \setminus \mathsf{R}$. Let $X = D_j \in \mathcal{D}$ be the prefix of $T[j\,..\,n]$ and y be an integer in $[1\,..\,n']$ such that $s'_y = j + |X| - 2\tau$. Then, $\delta(j) = \mathsf{rank}_{W,\overline{X}}(y)$.*

Proof. Denote $\delta_{\mathrm{text}} = |X| - 2\tau$ and $s = j + \delta_{\mathrm{text}}$. By definition of $\mathcal{D}$, we have $s \in \mathsf{S}$. By the consistency of S, there exists a bijection (given by the mapping $j' \mapsto j' + \delta_{\mathrm{text}}$) between positions $j' \in [1\,..\,n]$ such that $T[j'\,..\,n] \preceq T[j\,..\,n]$ and $D_{j'} = D_j$, and positions $s' \in \mathsf{S}$ such that

$T^\infty[s' - \delta_{\text{text}} \,..\, s' + 2\tau) = X$ and $T[s' \,..\, n] \preceq T[s \,..\, n]$. Thus, letting $y \in [1 \,..\, n']$ be such that $s'_y = s$, we obtain that $\delta(j) = |\{i \in [1 \,..\, y] : T^\infty[s'_i - \delta_{\text{text}} \,..\, s'_i + 2\tau) = X\}|$. Since we defined $W[i] = \overline{X_i}$, where $X_i = T^\infty[s'_i - \tau \,..\, s'_i + 2\tau)$, we conclude that $\delta(j) = \mathsf{rank}_{W,\overline{X}}(y)$. □

Proposition 3.4. *For every constant $\epsilon \in (0, 1)$, given $\mathrm{C}_{\mathrm{ISA}}(T, \mathsf{S})$, we can in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time and using $\mathcal{O}(n/\log_\sigma n)$ working space augment it into a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that, given $j \in [1 \,..\, n] \setminus \mathsf{R}$, returns $\mathrm{ISA}[j]$ in $\mathcal{O}(\log^\epsilon n)$ time.*

Proof. The data structure, in addition to $\mathrm{C}_{\mathrm{ISA}}(T, \mathsf{S})$, contains three components. First, we store the array $\mathrm{ISA}_{\mathsf{S}}[1 \,..\, n']$ in plain form, using $n' = \mathcal{O}(n/\log_\sigma n)$ words of space. Second, we store a lookup table of size $\mathcal{O}(n^\delta)$, for some $\delta < 1$, that given the packed representation of any string $X \in [0 \,..\, \sigma)^*$ of length $|X| = \mathcal{O}(\log_\sigma n)$, allows us to compute the packed representation of $\overline{X}$ in $\mathcal{O}(1)$ time. Third, and final, we store the data structure of Theorem 2.2 for the sequence W. Due to $n' = \mathcal{O}(n/\log_\sigma n)$ and $\sigma^{3\tau} = o(n/\log n)$, this requires $\mathcal{O}(n/\log_\sigma n)$ space.

Using $\mathrm{C}_{\mathrm{ISA}}(T, \mathsf{S})$ and the above two components, given $j \in [1 \,..\, n] \setminus \mathsf{R}$, we compute $\mathrm{ISA}[j]$ as follows. First, using Lemma 3.1, in $\mathcal{O}(1)$ time we compute $X = D_j$ and integers b, e such that $\mathrm{SA}(b \,..\, e]$ contains the starting positions of all suffixes of T starting with X. To compute y we first obtain $x = \mathsf{rank}_{B,1}(j-1)$ in $\mathcal{O}(1)$ time and and then let $y = \mathrm{ISA}_{\mathsf{S}}[x+1]$. By Lemma 3.3, it remains to determine $\delta(j) = \mathsf{rank}_{W,\overline{X}}(y)$. Using the data structure from Theorem 2.2 for the prefix rank query, computing $\mathrm{ISA}[j] = b + \delta(j)$ takes $\mathcal{O}(\log^\epsilon n)$ time.

The data structure is constructed as follows. We first construct the array $\mathrm{ISA}_{\mathsf{S}}$. We start by creating the sequence $(s_t)_{t \in [1 \,..\, n']}$ using select queries on B. This takes $\mathcal{O}(n/\log_\sigma n)$ time. Then, given $(s_t)_{t \in [1 \,..\, n']}$, and the packed representation of T, by [39, Theorem 4.3], we compute the sequence $(s'_t)_{t \in [1 \,..\, n']}$ in $\mathcal{O}(n/\log_\sigma n)$ time. Given $(s'_t)_{t \in [1 \,..\, n']}$, we then easily obtain the array $\mathrm{ISA}_{\mathsf{S}}$: simply scan the sequence $(s'_t)_{t \in [1 \,..\, n']}$ and for each $i \in [1 \,..\, n']$, let $j = \mathsf{rank}_{B,1}(s'_i)$ and note that then $s_j = s'_i$ and hence we can set $\mathrm{ISA}_{\mathsf{S}}[j] = i$. Next, we initialize the lookup table used to reverse short strings. In the RAM model, such array is easily initialized in $\mathcal{O}(n^\delta)$ time. The sequence $W[1 \,..\, n']$ is then easily obtained from $(s'_t)_{t \in [1 \,..\, n']}$ using the above lookup table in $\mathcal{O}(n/\log_\sigma n)$ time. We then process W using Theorem 2.2, which takes $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time and $\mathcal{O}(n/\log_\sigma n)$ space. □

3.3 The Periodic Positions

In this section, we describe a data structure to compute the value $\mathrm{ISA}[j]$ for any $j \in \mathsf{R}$.

Preliminaries We start by introducing the definitions that will allow us to express the properties utilized in our data structures. For $X \in \mathcal{F}$, we define the *Lyndon root* of X as $\text{L-root}(X) = \min\{X[t \,..\, t + p) : t \in [1 \,..\, p]\}$, where $p = \mathrm{per}(X)$. Then, for any $j \in \mathsf{R}$, we let $\text{L-root}(j) = \text{L-root}(T[j \,..\, j + 3\tau - 1))$. We define $\mathcal{H} = \{\text{L-root}(X) : X \in \mathcal{F}\}$, and for any $H \in \mathcal{H}$, we denote $\mathsf{R}_H = \{j \in \mathsf{R} : \text{L-root}(j) = H\}$. Next, we define the *L-decomposition*. For any $j \in \mathsf{R}$, let $e_j = \min\{j' \geq j : j' \notin \mathsf{R}\} + 3\tau - 2$. By [39, Fact 3.2], for $j \in \mathsf{R}$, $T[j \,..\, e_j)$ is the longest prefix of $T[j \,..\, n]$ having period $|\text{L-root}(j)|$. Thus, for every $j \in \mathsf{R}$, we can write $T[j \,..\, e_j) = H' H^k H''$, where $H = \text{L-root}(j)$, and H' (resp. H'') is a proper suffix (resp. prefix) of H. Note that L-decomposition is unique, since otherwise it would contradict the synchronization property of primitive strings [21, Lemma 1.11]. Hence, we denote $\text{L-head}(j) = |H'|$, $\text{L-exp}(j) = k$, and $\text{L-tail}(j) = |H''|$. For every $H \in \mathcal{H}$ and $s \in [0 \,..\, |H|)$, we denote $\mathsf{R}_{s,H} = \{j \in \mathsf{R}_H : \text{L-head}(j) = s\}$. For $j \in \mathsf{R}$, we let $\mathrm{type}(j) = +1$ if $e_j \leq n$ and $T[e_j] \succ T[e_j - p]$ (where $p = |\text{L-root}(j)|$), and $\mathrm{type}(j) = -1$ otherwise. We denote $\mathsf{R}^- = \{j \in \mathsf{R} : \mathrm{type}(j) = -1\}$ and $\mathsf{R}^+ = \mathsf{R} \setminus \mathsf{R}^-$. We also let $\mathsf{R}^-_H = \mathsf{R}^- \cap \mathsf{R}_H$ and $\mathsf{R}^-_{s,H} = \mathsf{R}^- \cap \mathsf{R}_{s,H}$, where $H \in \mathcal{H}$ and $s \in [0 \,..\, |H|)$.

The following lemma proves that the set of positions $\mathsf{R}_{s,H}$ occupies a contiguous block in the SA of T and describes the structure of such block.

Lemma 3.5. *If $j \in \mathsf{R}_{s,H}$ then for any $j' \in [1\,..\,n]$, $\mathrm{LCE}_T(j,j') \geq 3\tau-1$ if and only if $j' \in \mathsf{R}_{s,H}$. Moreover, if $j' \in \mathsf{R}_{s,H}$, then:*

1. *If $\mathrm{type}(j) = -1$ and $\mathrm{type}(j') = +1$, then $T[j\,..\,n] \prec T[j'\,..\,n]$.*
2. *If $\mathrm{type}(j) = \mathrm{type}(j') = -1$ and $e_j - j < e_{j'} - j'$, then $T[j\,..\,n] \prec T[j'\,..\,n]$.*
3. *If $\mathrm{type}(j) = \mathrm{type}(j') = +1$ and $e_j - j > e_{j'} - j'$, then $T[j\,..\,n] \prec T[j'\,..\,n]$.*

Proof. Let $j' \in [1\,..\,n]$ be such that $\mathrm{LCE}_T(j,j') \geq 3\tau-1$. Then, by definition, $\text{L-root}(j') = \text{L-root}(T[j'\,..\,j'+3\tau-1)) = \text{L-root}(T[j\,..\,j+3\tau-1)) = \text{L-root}(j)$. To show that $\text{L-head}(j') = s$, note that by $|H| \leq \tau$, the string $H'H^2$ (where H' is a length-s suffix of H) is a prefix of $T[j\,..\,j+3\tau-1) = T[j'\,..\,j'+3\tau-1)$. On the other hand, $\text{L-head}(j') = s'$ implies that $\widehat{H}'H^2$ (where $\widehat{H}'$ is a length-s' suffix of H) is a prefix of $T[j'\,..\,j'+3\tau-1)$. Thus, by the synchronization property of primitive strings applied to the two copies of H, we have $s' = s$, and consequently, $j' \in \mathsf{R}_{s,H}$.

For the converse implication, assume $j, j' \in \mathsf{R}_{s,H}$. This implies that both $T[j\,..\,e_j)$ and $T[j'\,..\,e_{j'})$ are prefixes of $H'H^\infty$ (where H' is as above). Thus, by $e_j - j$, $e_{j'} - j' \geq 3\tau-1$, we obtain $\mathrm{LCE}_T(j,j') \geq 3\tau-1$.

1. Let $S = H'H^\infty$, where H' is a length-s suffix of H. We will prove $T[j\,..\,n] \prec S \prec T[j'\,..\,n]$, which implies the claim. First, we note that $\mathrm{type}(j) = -1$ implies that either $e_j = n+1$, or $e_j \leq n$ and $T[e_j] \prec T[e_j - |H|]$. In the first case, $T[j\,..\,e_j) = T[j\,..\,n]$ is a proper prefix of S and hence $T[j\,..\,n] \prec S$. In the second case, letting $\ell = e_j - j$, we have $T[j\,..\,j+\ell) = S[1\,..\,\ell]$ and $T[j+\ell] \prec T[j+\ell-|H|] = S[1+\ell-|H|] = S[1+\ell]$. Consequently, $T[j\,..\,n] \prec S$. To show $S \prec T[j'\,..\,n]$ we observe that $\mathrm{type}(j') = +1$ implies $e_{j'} \leq n$. Thus, letting $\ell' = e_{j'} - j'$, we have $S[1\,..\,\ell'] = T[j'\,..\,j'+\ell')$ and $S[1+\ell'] = S[1+\ell'-|H|] = T[j'+\ell'-|H|] \prec T[j'+\ell']$. Hence, we obtain $S \prec T[j'\,..\,n]$.

2. Similarly as above, we consider two cases for e_j. If $e_j = n+1$, then by $e_j - j < e_{j'} - j'$, $T[j\,..\,e_j) = T[j\,..\,n]$ is a proper prefix of $T[j'\,..\,e_{j'})$ and hence $T[j\,..\,n] \prec T[j'\,..\,e_{j'}) \preceq T[j'\,..\,n]$. If $e_j \leq n$, then letting $\ell = e_j - j$, we have $T[j\,..\,j+\ell) = T[j'\,..\,j'+\ell)$ and by $e_j - j < e_{j'} - j'$, $T[j+\ell] \prec T[j+\ell-|H|] = T[j'+\ell-|H|] = T[j'+\ell]$. Consequently, $T[j\,..\,n] \prec T[j'\,..\,n]$.

3. By $\mathrm{type}(j') = +1$ we have $e_{j'} \leq n$. Thus, letting $\ell' = e_{j'} - j'$, by $e_j - j > e_{j'} - j'$, we have $T[j\,..\,j+\ell') = T[j'\,..\,j'+\ell')$ and $T[j+\ell'] = T[j+\ell'-|H|] = T[j'+\ell'-|H|] \prec T[j'+\ell']$. Consequently, $T[j\,..\,n] \prec T[j'\,..\,n]$. □

The key to the efficient computation of $\delta(j)$ is processing of the elements of R in blocks. The starting positions of these blocks are defined as $\mathsf{R}' := \{j \in \mathsf{R} : j-1 \notin \mathsf{R}\}$. We also let $\mathsf{R}'^{-} = \mathsf{R}^{-} \cap \mathsf{R}'$ and $\mathsf{R}'^{-}_H = \mathsf{R}' \cap \mathsf{R}^{-}_H$ for $H \in \mathcal{H}$. The next lemma justifies our strategy.

Lemma 3.6. *If $j \in \mathsf{R} \setminus \mathsf{R}'$, then $\text{L-root}(j-1) = \text{L-root}(j)$, $e_{j-1} = e_j$, and $\mathrm{type}(j-1) = \mathrm{type}(j)$.*

Proof. Denote $p = \mathrm{per}(T[j-1\,..\,j-1+3\tau-1))$ and $p' = \mathrm{per}(T[j\,..\,j+3\tau-1))$. By $j-1, j \in \mathsf{R}$ we have $p, p' \leq \frac{\tau}{3}$. Consider $S = T[j\,..\,j+\tau)$. The string S has both periods p and p'. If $p \neq p'$, then by the weak periodicity lemma, S has a period $p'' = \gcd(p,p') < p'$. Since $p'' \mid p'$ we obtain that $T[j\,..\,j+p')$ is not primitive, which contradicts $\mathrm{per}(T[j\,..\,j+3\tau-1)) = p'$. Thus, $p = p'$. Then, by $p \leq \frac{\tau}{3}$, we have $T[j-1\,..\,j-1+p) = T[j-1+p\,..\,j-1+2p)$. Consequently, $\{T[j-1+t\,..\,j-1+t+p) : t \in [0\,..\,p)\} = \{T[j+t\,..\,j+t+p) : t \in [0\,..\,p)\}$, and hence $\text{L-root}(j-1) = \text{L-root}(j)$.

By [39, Fact 3.2], $T[j-1\,..\,e_{j-1})$ (resp. $T[j\,..\,e_j)$) is the longest substring starting at position $j-1$ (resp. j) with period p (resp. p'). Equivalently, $e_{j-1} = j-1+p+\mathrm{LCE}(j-1, j-1+p)$ and $e_j = j+p'+\mathrm{LCE}(j, j+p')$. Thus, by $p = p'$ and $T[j-1] = T[j-1+p]$, we have $e_{j-1} = j-1+p+\mathrm{LCE}(j-1, j-1+p) = j+p+\mathrm{LCE}(j, j+p) = j+p'+\mathrm{LCE}(j, j+p') = e_j$.

The third claim follows from the definition of type and equalities $e_{j-1} = e_j$ and $p = p'$. □

Next, we develop a component computing the basic values characterizing periodic positions.

Proposition 3.7. *In $\mathcal{O}(n/\log_\sigma n)$ time, we can augment $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$ into a data structure that, given any $j \in \mathsf{R}$, returns* L-root(j), L-head(j), L-exp(j), L-tail(j), *and* type(j) *in $\mathcal{O}(1)$ time.*

Proof. We use the following definitions. Let L denote the mapping from $[0\,..\,\sigma)^{3\tau-1}$ to $\mathbb{N}^2$ such that for any $X \in [0\,..\,\sigma)^{3\tau-1}$ satisfying $\mathrm{per}(X) \le \frac{1}{3}\tau$, L maps X to a pair (s,p), where $p = \mathrm{per}(X)$ and $s \in [0\,..\,p)$ is such that $X[1{+}s\,..\,1{+}s{+}p) = \min\{X[t\,..\,t+p) : t \in [1\,..\,p]\}$.

We now define the data structure. In addition to $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$, the data structure contains the lookup table L. When accessing L, strings $X \in [0\,..\,\sigma)^{3\tau-1}$ are converted to integers $\mathrm{int}(X)$. Thus, the mapping L needs $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ space.

Using $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$ and L, we implement the queries as follows. Given $j \in \mathsf{R}$, we first compute $x \in [0\,..\,\sigma^{6\tau})$ such that $x = \mathrm{int}(T[j\,..\,j+3\tau-1))$. Given the packed encoding of text T, such x is obtained in $\mathcal{O}(1)$ time. We then look up $(s,p) = L[x]$, and in $\mathcal{O}(1)$ time obtain L-root$(j) = T[j{+}s\,..\,j{+}s{+}p)$ and L-head$(j) = s$. Next, we compute L-exp(j) and L-tail(j). For this we recall that by [39, Section 6.1.2], it holds $e_j = \mathrm{succ}_{\mathsf{S}}(j) + 2\tau - 1$. The value $\mathrm{succ}_{\mathsf{S}}(j)$ is computed using rank and select query on the bitvector B from $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$, as explained in Lemma 3.1. Thus, in $\mathcal{O}(1)$ time we obtain e_j, and consequently L-exp$(j) = \lfloor \frac{e_j - j - s}{p} \rfloor$ and L-tail$(j) = (e_j - j - s) \bmod p$. Finally, to test if type$(j) = +1$, we check whether $e_j \le n$, and if so, whether $T[e_j] \succ T[e_j - p]$.

To construct the data structure, we observe that, given $X \in [0\,..\,\sigma)^{3\tau-1}$, we can check in $\mathcal{O}(\tau^2)$ time if $\mathrm{per}(X) \le \frac{1}{3}\tau$, and if so, determine the value $L[\mathrm{int}(X)] = (s,p)$. To compute $\mathrm{per}(X)$, we try all $\ell \in [1\,..\,\lfloor \frac{\tau}{3} \rfloor]$ until we find that ℓ is a period of X, or that there is no such ℓ. Assuming $p := \mathrm{per}(X) \le \frac{1}{3}\tau$, finding $s \in [0\,..\,p)$ satisfying $X[1{+}s\,..\,1{+}s{+}p) = \min\{X[t\,..\,t{+}p) : t \in [1\,..\,p]\}$ also takes $\mathcal{O}(\tau^2)$ time. Initializing L takes $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$. Over all $X \in [0\,..\,\sigma)^{3\tau-1}$, we spend $\mathcal{O}(\sigma^{3\tau-1}\tau^2) = \mathcal{O}(n^{1/2}\log^2 n) = \mathcal{O}(n/\log_\sigma n)$ time. □

We focus on computing $\delta(j)$ for $j \in \mathsf{R}^-$. The elements of R^+ are processed symmetrically (the details are provided in the proof of Proposition 3.13). For any $H \in \mathcal{H}$, $s \in [0\,..\,|H|)$, and $j \in \mathsf{R}^-_{s,H}$, we define $\mathrm{pos}^a(j) = \{j' \in \mathsf{R}^-_{s,H} : \text{L-exp}(j') \le \text{L-exp}(j)\}$ and $\mathrm{pos}^s(j) = \{j' \in \mathsf{R}^-_{s,H} : \text{L-exp}(j') = \text{L-exp}(j) \text{ and } T[j'\,..\,n] \succ T[j\,..\,n]\}$. For any $j \in \mathsf{R}^-$, we denote $\delta^a(j) = |\mathrm{pos}^a(j)|$ and $\delta^s(j) = |\mathrm{pos}^s(j)|$.

Lemma 3.8. *For any $j \in \mathsf{R}^-$, it holds $\delta(j) = \delta^a(j) - \delta^s(j)$.*

Proof. We will prove that $\mathrm{pos}^a(j)$ is a disjoint union of $\mathrm{pos}(j)$ and $\mathrm{pos}^s(j)$. This implies $\delta(j) + \delta^s(j) = \delta^a(j)$, and consequently, the equality in the claim.

By Lemma 3.5, letting $j \in \mathsf{R}^-_{s,H}$, we have $\mathrm{pos}(j) = \{j' \in \mathsf{R}^-_{s,H} : T[j'\,..\,n] \preceq T[j\,..\,n]\}$, and moreover, if $j' \in \mathrm{pos}(j)$, then $e_{j'} - j' \le e_j - j$. In particular, L-exp$(j') = \lfloor \frac{e_{j'} - j' - s}{|H|} \rfloor \le \lfloor \frac{e_j - j - s}{|H|} \rfloor =$ L-exp(j). Hence, $\mathrm{pos}(j) \subseteq \mathrm{pos}^a(j)$. On the other hand, clearly $\mathrm{pos}^s(j) \subseteq \mathrm{pos}^a(j)$ and $\mathrm{pos}^s(j) \cap \mathrm{pos}(j) = \emptyset$. Thus, to obtain the claim, it suffices to show that $\mathrm{pos}^a(j) \setminus \mathrm{pos}^s(j) \subseteq \mathrm{pos}(j)$.

Let $j' \in \mathrm{pos}^a(j) \setminus \mathrm{pos}^s(j)$. Consider two cases. If L-exp$(j') =$ L-exp(j), then by definition of $\mathrm{pos}^s(j)$, it must hold $T[j'\,..\,n] \preceq T[j\,..\,n]$. Thus, we have $j' \in \mathrm{pos}(j)$. Let us therefore assume L-exp$(j') <$ L-exp(j). Then, $e_{j'} - j' = s + \text{L-exp}(j') \cdot |H| + \text{L-tail}(j') < s + \text{L-exp}(j') \cdot |H| + |H| \le s + \text{L-exp}(j) \cdot |H| \le s + \text{L-exp}(j) \cdot |H| + \text{L-tail}(j) = e_j - j$. By Item 2 of Lemma 3.5, this implies $T[j'\,..\,n] \prec T[j\,..\,n]$, and consequently, $j' \in \mathrm{pos}(j)$. □

Computing $\delta^a(j)$ We now describe a data structure that allows computing $\delta^a(j)$ for $j \in \mathsf{R}^-$.

Proposition 3.9. *In $\mathcal{O}(n/\log_\sigma n)$ time, we can augment the structure of Proposition 3.7 so that $\delta^a(j)$ can be computed in $\mathcal{O}(1)$ time given $j \in \mathsf{R}^-$.*

Proof. We use the following definitions. Let $E[1\,..\,n]$ be a bitvector such that for every $i \in [1\,..\,n]$, it holds $E[i] = 0$ if and only if $\mathrm{SA}[i] \in [1\,..\,n] \setminus \mathsf{R}^-$, or $i < n$ and the positions $j = \mathrm{SA}[i]$ and $j' = \mathrm{SA}[i+1]$ satisfy $j, j' \in \mathsf{R}^-_{s,H}$ and $\text{L-exp}(j) = \text{L-exp}(j')$ for some $H \in \mathcal{H}$ and $s \in [0\,..\,|H|)$. For any $H \in \mathcal{H}$ and $s \in [0\,..\,|H|)$, let $E^-_{s,H}$ denote the block of E corresponding to suffixes starting in $\mathsf{R}^-_{s,H}$, i.e., $E^-_{s,H} = E(b\,..\,e]$, where $(b\,..\,e] \subseteq [1\,..\,n]$ is such that $\mathsf{R}^-_{s,H} = \{\mathrm{SA}[i] : i \in (b\,..\,e]\}$ (such $(b\,..\,e]$ exists by Item 1 of Lemma 3.5). We define $Q : [0\,..\,\sigma)^{3\tau-1} \to [1\,..\,n]$ as the mapping such that for every $X \in \mathcal{F}$, if we let $H = \text{L-root}(X)$ and $s \in [0\,..\,|H|)$ be such that $X[1{+}s\,..\,1{+}s{+}|H|) = H$, then assuming $\mathsf{R}^-_{s,H} \neq \emptyset$, Q maps X to $\min\{\text{L-exp}(j) : j \in \mathsf{R}^-_{s,H}\}$. Finally, let $\mathrm{unary}(x) := \mathtt{0}^x\mathtt{1}$ denote the unary encoding of an integer $x \geq 0$, and let $\mathrm{unary}^+(x)$ be $\mathrm{unary}(x)$ with the first symbol removed (in particular, $\mathrm{unary}^+(0)$ is the empty string). If $(a_i)_{i\in[1\,..\,k]}$ is a sequence of non-negative integers, we define $\mathrm{unary}((a_i)_{i\in[1\,..\,k]}) := \bigodot_{i=1}^{k} \mathrm{unary}(a_i)$, where $\bigodot$ denotes concatenation. Analogously, $\mathrm{unary}^+((a_i)_{i\in[1\,..\,k]}) := \bigodot_{i=1}^{k} \mathrm{unary}^+(a_i)$. The definitions of unary and unary^+ are naturally extended to infinite sequences $(a_i)_{i\in[1\,..\,\infty)}$.

The data structure, in addition to the data structure from Proposition 3.7, contains two components. First, we store the bitvector E augmented using Theorem 2.1, which takes $\mathcal{O}(n)$ bits, i.e., $\mathcal{O}(n/\log n)$ space. Second, we store the mapping Q. Assuming, similarly as in Section 3.1, that when accessing Q, strings $X \in [0\,..\,\sigma)^{3\tau-1}$ are converted to integers $\mathrm{int}(X)$, the mapping Q needs $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ space.

Using the above data structures, given $j \in \mathsf{R}^-$, we compute $\delta^a(j)$ as follows. First, using Lemma 3.1, in $\mathcal{O}(1)$ time we compute a prefix $X \in \mathcal{F}$ of $T[j\,..\,n]$ and integers b, e such that $\mathrm{SA}(b\,..\,e]$ contains the starting positions of all suffixes of T starting with X. Equivalently, by Lemma 3.5, $\mathrm{SA}(b\,..\,e]$ contains all positions from $\mathsf{R}_{s,H}$, where $H = \text{L-root}(j)$ and $s = \text{L-head}(j)$. Next, using Proposition 3.7, we compute in $\mathcal{O}(1)$ time the value $k = \text{L-exp}(j)$. Finally, we retrieve $k_{\min} = Q[\mathrm{int}(X)]$. Observe now that by Lemma 3.5, all positions in $\mathsf{R}^-_{s,H}$ occur in $\mathrm{SA}(b\,..\,e]$ before $\mathsf{R}^+_{s,H}$. Furthermore, by Item 2 in Lemma 3.5, the L-exponents in the $\mathrm{SA}(b\,..\,e]$ range corresponding to $\mathsf{R}^-_{s,H}$ are nondecreasing. It is easy to see that $[k_{\min}\,..\,k] \subseteq \{\text{L-exp}(j') : j' \in \mathsf{R}^-_{s,H}\}$ (for $k' \in (k_{\min}\,..\,k]$, we can take $j' = j + (k - k')|H|$). Thus, by the definition of E, we can finally return $\delta^a(j) = \mathsf{select}_{E,1}(\mathsf{rank}_{E,1}(b) + (k - k_{\min}) + 1) - b$ in $\mathcal{O}(1)$ time.

The data structure is constructed as follows. Let $\alpha < 1$ be a positive constant. We first show an algorithm that, given the data structure from Proposition 3.7 and the set of positions R'^-_H (where $H \in \mathcal{H}$) as input, computes all bitvectors $E^-_{0,H}, \ldots, E^-_{|H|-1,H}$ in $\mathcal{O}(|\mathsf{R}'^-_H| + |\mathsf{R}^-_H|/\log n + n^\alpha)$ time. For any $s \in [0\,..\,|H|)$ and $k \geq 0$, denote $e_{s,k,H} = |\{j' \in \mathsf{R}^-_{s,H} : \text{L-exp}(j') = k\}|$. We start by observing that by Item 2 in Lemma 3.5, $E^-_{s,H} = \mathrm{unary}^+((e_{s,k,H})_{k\in[0\,..\,\infty)})$. The values $e_{s,k,H}$ can be efficiently determined based on the following observation. First, note that if $j \in \mathsf{R}'^-_H$, then $[j\,..\,e_j - 3\tau + 2) \subseteq \mathsf{R}^-_H$, and $j - 1, e_j - 3\tau + 2 \notin \mathsf{R}$, i.e., the block of positions in R^-_H is maximal. By Lemma 3.6, for any $j' \in [j\,..\,e_j - 3\tau + 2)$, it holds $e_{j'} = e_j$. Thus, for any $j' \in [j\,..\,e_j - 3\tau + 2)$, we have $\text{L-exp}(j') = \lfloor \frac{e-j'}{|H|} \rfloor$ and $\text{L-head}(j') = (e - j') \bmod |H|$, where $e = e_j - \text{L-tail}(j)$. With this in mind, for any $j \in \mathsf{R}'^-_H$, we let $\mathcal{I}_j = (3\tau - 2 - t\,..\,s + k|H|]$, where $s = \text{L-head}(j)$, $k = \text{L-exp}(j)$, and $t = \text{L-tail}(j)$. By the above discussion, for any $s \in [0\,..\,|H|)$ and $k \geq 0$, we have $e_{s,k,H} = |\{j \in \mathsf{R}'^-_H : s + k|H| \in \mathcal{I}_j\}|$. The algorithm consists of three steps:

1. First, we compute the string $\mathrm{unary}((e_{0,k,H})_{k=0}^{k_{\max}})$, where $k_{\max} = \max\{\text{L-exp}(j') : j' \in \mathsf{R}^-_H\}$. We start by computing $k_{\max}$. For this we observe that $k_{\max} = \max\{\text{L-exp}(j') : j' \in \mathsf{R}'^-_H\}$. Thus, using Proposition 3.7, we can compute $k_{\max}$ in $\mathcal{O}(|\mathsf{R}'^-_H|)$ time. To compute $\mathrm{unary}((e_{0,k,H})_{k\in[0\,..\,k_{\max}]})$, we generate the sequence of "events" from R'^-_H, sort them, and then output $\mathrm{unary}((e_{0,k,H})_{k\in[0\,..\,k_{\max}]})$ left-to-right. More precisely, let $m = |\mathsf{R}'^-_H|$, and let $(p_i, v_i)_{i\in[0\,..\,2m]}$ be a sequence containing the multiset $\{(0,0), (k_{\max}+1, 0)\} \cup \{(\lceil \min \mathcal{I}_j/|H| \rceil, +1) : j \in \mathsf{R}'^-_H\} \cup \{(\lfloor \max \mathcal{I}_j/|H| \rfloor + 1, -1) : j \in \mathsf{R}'^-_H\}$ such that for any $i \in [1\,..\,2m]$, it holds $p_{i-1} \leq p_i$. To compute the sequence $(p_i, v_i)_{i\in[0\,..\,2m]}$, we observe that, given $j \in \mathsf{R}'^-_H$, we can compute $\mathcal{I}_j$ in $\mathcal{O}(1)$ time using Proposition 3.7. Thus, in $\mathcal{O}(m)$ time

we can generate all pairs in the above multiset. We then sort the pairs by the first element. Using $\lceil 1/\alpha \rceil$-round radix sort, this takes $\mathcal{O}(m + n^\alpha)$ time. Consequently, we can compute $(p_i, v_i)_{i \in [0\,..\,2m]}$ in $\mathcal{O}(|\mathsf{R}'^{-}_H| + n^\alpha)$ time. Given the sequence $(p_i, v_i)_{i \in [0\,..\,2m]}$, we compute $\mathrm{unary}((e_{0,k,H})_{k \in [0\,..\,k_{\max}]})$ as follows. First, we initialize the output bitvector to the empty string and set $v = 0$. We then iterate through $i = 1, \ldots, 2m$. For every i, we first append $p_i - p_{i-1}$ copies of the string $\mathrm{unary}(v)$ to the output string. We then add v_i to v. To efficiently append multiple copies of $\mathrm{unary}(v)$ to the output, we first precompute (in $\mathcal{O}(\log^2 n) = \mathcal{O}(n^\alpha)$ time) the prefix of length $\log n$ of the string $\mathrm{unary}(x)^\infty$ for every $x \in [0\,..\,\log n)$. This way, we can append $\mathrm{unary}(v)^\ell$ to the output in $\mathcal{O}(1 + (v+1)\ell/\log n)$ time. Consequently, the construction of $\mathrm{unary}((e_{0,k,H})_{k \in [0\,..\,k_{\max}]})$ takes $\mathcal{O}(|\mathsf{R}'^{-}_H| + |\mathrm{unary}((e_{0,k,H})_{k \in [0\,..\,k_{\max}]})|/\log n + n^\alpha) = \mathcal{O}(|\mathsf{R}'^{-}_H| + |E^-_{0,H}|/\log n + n^\alpha) = \mathcal{O}(|\mathsf{R}'^{-}_H| + |\mathsf{R}^-_H|/\log n + n^\alpha)$ time. To show the first upper bound, observe that $k_{\max} \le |E^-_{0,H}| + \mathcal{O}(\tau/|H|)$. Thus, $|\mathrm{unary}((e_{0,k,H})_{k \in [0\,..\,k_{\max}]})| = |\mathrm{unary}^+((e_{0,k,H})_{k \in [0\,..\,\infty)})| + k_{\max} + 1 = |E^-_{0,H}| + k_{\max} + 1 \le 2|E^-_{0,H}| + \mathcal{O}(\log n)$. The second upper bound follows by observing that $|E^-_{0,H}| + \cdots + |E^-_{|H|-1,H}| = |\mathsf{R}^-_H|$.

2. The second step of the algorithm is to compute the strings $\mathrm{unary}((e_{s,k,H})_{k \in [0\,..\,k_{\max}]})$ for $s \in [1\,..\,|H|)$. For any $s \in [1\,..\,|H|)$, let $(q_i^{(s)}, p_i^{(s)}, v_i^{(s)})_{i \in [0\,..\,m_s]}$ denote the sequence containing all the elements (q, p, v) of the multiset $\{(q, 0, 0) : q \in [1\,..\,|H|)\} \cup \{(q, k_{\max}+1, 0) : q \in [1\,..\,|H|)\} \cup \{(\min \mathcal{I}_j \bmod |H|, \lfloor \min \mathcal{I}_j/|H| \rfloor, +1) : j \in \mathsf{R}'^{-}_H\} \cup \{((\max \mathcal{I}_j + 1) \bmod |H|, \lfloor (\max \mathcal{I}_j + 1)/|H| \rfloor, -1) : j \in \mathsf{R}'^{-}_H\}$ that satisfy $q = s$, and for any $i \in [1\,..\,m_s]$, it holds $p_{i-1}^{(s)} \le p_i^{(s)}$ (note that the elements of this multiset satisfying $q = 0$ are not included in any sequence). To compute the sequences $(q_i^{(s)}, p_i^{(s)}, v_i^{(s)})_{i \in [0\,..\,m_s]}$ for all $s \in [1\,..\,|H|)$, we first enumerate all triples in the above multiset. Using Proposition 3.7, this takes $\mathcal{O}(m)$ time. We then sort the triples lexicographically. Using $\lceil 1/\alpha \rceil$-round radix sort, this takes $\mathcal{O}(m + n^\alpha)$ time. This yields all sequences concatenated together. It is easy to discard unused elements, and to detect boundaries between lists with a single scan. Consequently, we can construct all sequences in $\mathcal{O}(|\mathsf{R}'^{-}_H| + n^\alpha)$ time. Given the above sequences, we can compute the strings $\mathrm{unary}((e_{s,k,H})_{k \in [0\,..\,k_{\max}]})$ for $s \in [1\,..\,|H|)$ as follows. The algorithm computes the strings in the order of increasing s. More precisely, given the string $U := \mathrm{unary}((e_{s-1,k,H})_{k \in [0\,..\,k_{\max}]})$ and the sequence $(q_i^{(s)}, p_i^{(s)}, v_i^{(s)})_{i \in [0\,..\,m_s]}$ (where $s \in [1\,..\,|H|)$), we compute the string $\mathrm{unary}((e_{s,k,H})_{k \in [0\,..\,k_{\max}]})$ in $\mathcal{O}(m_s + |U|/\log n)$ time as follows. First, we initialize the output bitvector to the empty string, and set $v = 0$ and $y = 0$. We then iterate through $i = 1, \ldots, m_s$. For every i, we first check if $p_i^{(s)} > p_{i-1}^{(s)}$. If yes, we perform the following three steps. First, find the position y' of the $p_i^{(s)}$th 1-bit in U. Second, append the substring $U(y\,..\,y']$ to the output, except we first prepend it with v zeros (if $v \ge 0$) or discard its first $-v$ bits (if $v < 0$). Finally, we set $y = y'$ and $v = 0$. Then (regardless of whether $p_i^{(s)} > p_{i-1}^{(s)}$), we add $v_i^{(s)}$ to v. To efficiently compute y' we observe that the arguments of the consecutive select queries are increasing. We can thus precompute in $\mathcal{O}(n^\alpha)$ time a lookup table such that the computation of y' takes $\mathcal{O}(1 + (y' - y)/\log n)$ time (these lookup tables can be shared among algorithms for different s). Note that for any $s \in [0\,..\,|H|)$, we have $k_{\max} \le |E^-_{s,H}| + \mathcal{O}(\tau/|H|)$. Thus, $|U| \le 2|E^-_{s-1,H}| + \mathcal{O}(\log n)$, and hence the algorithm runs in $\mathcal{O}(m_s + |E^-_{s-1,H}|/\log n)$ time. Consequently, by $m_0 + \cdots + m_{|H|-1} \le 2|\mathsf{R}'^{-}_H| + 2|H|$ and $|E^-_{0,H}| + \cdots + |E^-_{|H|-1,H}| = |\mathsf{R}^-_H|$, over all $s \in [1\,..\,|H|)$, we spend $\mathcal{O}(|\mathsf{R}'^{-}_H| + |\mathsf{R}^-_H|/\log n + n^\alpha)$ time.
3. The third and final step of the algorithm is to convert the string $\mathrm{unary}((e_{s,k,H})_{k \in [0\,..\,k_{\max}]})$ into $\mathrm{unary}^+((e_{s,k,H})_{k \in [0\,..\,k_{\max}]}) = E^-_{s,H}$ for every $s \in [0\,..\,|H|)$. Let us fix some $s \in [0\,..\,|H|)$. Observe that to implement the conversion, it suffices to remove the first bit, as well as every bit following a 1-bit in $\mathrm{unary}((e_{s,k,H})_{k \in [0\,..\,k_{\max}]})$. In the RAM model, such local operation is easy implemented in $\mathcal{O}(1 + |\mathrm{unary}((e_{s,k,H})_{k \in [0\,..\,k_{\max}]})|/\log n)$ time after a $\mathcal{O}(n^\alpha)$-time preprocessing (we do the preprocessing once for all $s \in [0\,..\,|H|)$). As observed

above, $|\mathrm{unary}((e_{s,k,H})_{k\in[0..k_{\max}]})| \le 2|E^-_{s,H}| + \mathcal{O}(\log n)$. Thus, the total time to perform the conversion for all s is $\mathcal{O}(|R^-_H|/\log n + n^\alpha)$.

Using the above algorithm, we construct E as follows. We start by computing the set $\{(\mathrm{int}(\text{L-root}(j)), j)\}_{j\in\mathsf{R}^-}$. By the consistency condition (see also [39, Section 6.1.2]), $i \in \mathsf{R}'$ implies that either $i-1 \in \mathsf{S}$ or $i = 1$. Thus, using the select queries on the bitvector B (stored as part of $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$), Lemma 3.1, and Proposition 3.7, we can enumerate the above set of pairs in $\mathcal{O}(1+|\mathsf{S}|) = \mathcal{O}(n/\log_\sigma n)$ time. Using $\lceil 1/\alpha\rceil$-round radix sort we then sort in $\mathcal{O}(|\mathsf{R}'^-| + n^\alpha) = \mathcal{O}(n/\log_\sigma n + n^\alpha)$ time the set of pairs by the first coordinate. This yields the representation of sets R'^-_H for all $H \in \mathcal{H}$. For each $H \in \mathcal{H}$, we then use the above algorithm to compute bitvectors $E^-_{0,H}, \ldots, E^-_{|H|-1,H}$ in $\mathcal{O}(|\mathsf{R}'^-_H| + |\mathsf{R}^-_H|/\log n + n^\alpha)$ time. By $\mathcal{H} \subseteq [0..\sigma)^{\le\tau}$, over all H, this takes $\mathcal{O}(|\mathsf{R}'^-| + |R^-|/\log n + n^{\alpha+\mu})$ time (recall that $\tau = \mu\log n$). Choosing $\alpha < 1-\mu$ results in $\mathcal{O}(|\mathsf{S}| + n/\log n) = \mathcal{O}(n/\log_\sigma n)$ total time. When bitvectors $E^-_{s,H}$ are computed for all $H \in \mathcal{H}$ and $s \in [0..|H|)$, we initialize E to the string $\mathsf{0}^n$ in $\mathcal{O}(n/\log n)$ time, and then "paste" all the non-empty bitvectors $E^-_{s,H}$ into their correct positions. Given $H \in \mathcal{H}$ and $s \in [0..|H|)$, we first compute in $\mathcal{O}(\log n)$ time the corresponding string $X \in \mathcal{F}$, and then compute the position to paste $E^-_{s,H}$ using Lemma 3.1. Over all $H \in \mathcal{H}$ and $s \in [0..|H|)$, this takes $\mathcal{O}(n^\mu \log^2 n + n/\log n) = \mathcal{O}(n/\log_\sigma n)$ time. Thus, altogether, constructing E and augmenting it using Theorem 2.1 takes $\mathcal{O}(n/\log_\sigma n)$ time. It remains to initialize the lookup table Q. For this, we observe that if i is the position of the leftmost 0-bit in $\mathrm{unary}((e_{s,k,H})_{k\in[0..k_{\max}]})$, then $\min\{\text{L-exp}(j) : j \in \mathsf{R}^-_{s,H}\} = i-1$. Given the packed representation of $\mathrm{unary}((e_{s,k,H})_{k\in[0..k_{\max}]})$, the position i can be easily found in $\mathcal{O}(1+|\mathrm{unary}((e_{s,k,H})_{k\in[0..k_{\max}]})|/\log n)$ time. Thus, accounting for the computation of $X \in \mathcal{F}$ corresponding to the choice of $H \in \mathcal{H}$ and $s \in [0..|H|)$, we can initialize Q in $\mathcal{O}(n/\log n + n^\mu \log^2 n) = \mathcal{O}(n/\log_\sigma n)$ time. □

Computing $\delta^s(j)$ We now describe the data structure to compute $\delta^s(j)$ for any position $j \in \mathsf{R}^-$.

We use the following definition. For any position $j \in \mathsf{R}$, we define $e^{\mathrm{full}}_j = e_j - \text{L-tail}(j)$.

Lemma 3.10. *Assume $i, j \in \mathsf{R}^-_H$ and let $\ell = e_i - i - 3\tau + 2$. Then $|\mathrm{pos}^s(j) \cap [i..i+\ell)| \le 1$. Moreover, $|\mathrm{pos}^s(j) \cap [i..i+\ell)| = 1$ if and only if $T[e^{\mathrm{full}}_i..n] \succ T[e^{\mathrm{full}}_j..n]$ and $e^{\mathrm{full}}_i - i \ge e^{\mathrm{full}}_j - j$.*

Proof. By Lemma 3.6, we have $[i..i+\ell) \subseteq \mathsf{R}^-_H$ with $e_{i+\delta} = e_i$ for every $\delta \in [0..\ell)$. Moreover, by the uniqueness of L-decomposition, $\text{L-tail}(i+\delta) = \text{L-tail}(i)$. Together, these imply that $e^{\mathrm{full}}_{i+\delta} = e^{\mathrm{full}}_i$, and consequently $e^{\mathrm{full}}_{i+\delta} - (i+\delta) = e^{\mathrm{full}}_i - i - \delta$. It remains to observe that, letting $j \in \mathsf{R}^-_{s,H}$, for $j' \in \mathrm{pos}^s(j)$ it holds $e^{\mathrm{full}}_{j'} - j' = s + \text{L-exp}(j')\cdot|H| = s + \text{L-exp}(j)\cdot|H| = e^{\mathrm{full}}_j - j$. Thus, $i+\delta \in \mathrm{pos}^s(j)$ implies $e^{\mathrm{full}}_{i+\delta} - (i+\delta) = e^{\mathrm{full}}_i - (i+\delta) = e^{\mathrm{full}}_j - j$, or equivalently, $\delta = (e^{\mathrm{full}}_i - i) - (e^{\mathrm{full}}_j - j)$, and therefore, $|\mathrm{pos}^s(j) \cap [i..i+\ell)| \le 1$.

For the second part, assume first that $i+\delta \in \mathrm{pos}^s(j)$ holds for some $\delta \in [0..\ell)$. Then, as noted above, we have $e^{\mathrm{full}}_j - j = e^{\mathrm{full}}_i - (i+\delta) \le e^{\mathrm{full}}_i - i$. Moreover, letting $j \in \mathsf{R}^-_{s,H}$, by definition of $\mathrm{pos}^s(j)$, we have $i+\delta \in \mathsf{R}^-_{s,H}$, $\text{L-exp}(j) = \text{L-exp}(i+\delta)$, and $T[i+\delta..n] \succ T[j..n]$. Therefore, we obtain that $T[i+\delta..e^{\mathrm{full}}_{i+\delta}) = T[i+\delta..e^{\mathrm{full}}_i) = T[j..e^{\mathrm{full}}_j) = H'H^k$ (where $k = \text{L-exp}(j)$ and H' is the length-s prefix of H), and consequently, $T[e^{\mathrm{full}}_i..n] \succ T[e^{\mathrm{full}}_j..n]$. To show the converse implication, assume $T[e^{\mathrm{full}}_i..n] \succ T[e^{\mathrm{full}}_j..n]$ and $e^{\mathrm{full}}_i - i \ge e^{\mathrm{full}}_j - j$. Let $\delta = (e^{\mathrm{full}}_i - i) - (e^{\mathrm{full}}_j - j)$. We will prove that $\delta \in [0..\ell)$ and $i+\delta \in \mathrm{pos}^s(j)$. Clearly $\delta \ge 0$. To show $\delta < \ell$, we first prove $e_i - e^{\mathrm{full}}_i \ge e_j - e^{\mathrm{full}}_j$. Suppose that $q = e_i - e^{\mathrm{full}}_i < e_j - e^{\mathrm{full}}_j$. By $i \in \mathsf{R}^-_H$, we then either have $e^{\mathrm{full}}_i + q = n+1$, or $e^{\mathrm{full}}_i + q \le n$ and $T[e^{\mathrm{full}}_i + q] \prec T[e^{\mathrm{full}}_i + q - |H|] = T[e^{\mathrm{full}}_j + q - |H|] = T[e^{\mathrm{full}}_j + q]$, both of which contradict $T[e^{\mathrm{full}}_i..n] \succ T[e^{\mathrm{full}}_j..n]$. Thus, $e_i - e^{\mathrm{full}}_i \ge e_j - e^{\mathrm{full}}_j$. This implies, $e_i - (i+\delta) = (e^{\mathrm{full}}_i - (i+\delta)) + (e_i - e^{\mathrm{full}}_i) = (e^{\mathrm{full}}_j - j) + (e_i - e^{\mathrm{full}}_i) \ge (e^{\mathrm{full}}_j - j) + (e_j - e^{\mathrm{full}}_j) = e_j - j \ge 3\tau - 1$, or equivalently $\delta \le e_i - i - 3\tau + 1 < \ell$. To show $i+\delta \in \mathrm{pos}^s(j)$, it remains to observe that $e^{\mathrm{full}}_{i+\delta} - (i+\delta) = e^{\mathrm{full}}_i - (i+\delta) = e^{\mathrm{full}}_j - j$ and $i+\delta, j \in \mathsf{R}^-_H$ imply $T[i+\delta..e^{\mathrm{full}}_i) = T[j..e^{\mathrm{full}}_j)$. This in particular gives, letting $j \in \mathsf{R}_{s,H}$, that $i+\delta \in \mathsf{R}_{s,H}$ and $\text{L-exp}(i+\delta) = \text{L-exp}(j)$. Moreover,

combining it with $T[e_i^{\text{full}} \,..\, n] \succ T[e_j^{\text{full}} \,..\, n]$ yields $T[i+\delta \,..\, n] \succ T[j \,..\, n]$. Finally, by Lemma 3.6, $\text{type}(i+\delta) = \text{type}(i) = -1$. Therefore, $i + \delta \in \text{pos}^s(j)$. □

Before presenting how to compute $\delta^s(j)$, we first need to prove the following auxiliary result. Let $A[1 \,..\, m]$ be an array of nonnegative integers. We define the following queries on A:

Range counting query $\mathsf{rcount}_A(v, j)$**:** Given an integer $v \geq 0$ and a position $j \in [1 \,..\, m]$, return $|\{i \in [1 \,..\, j] : A[i] \geq v\}|$.

Range selection query $\mathsf{rselect}_A(v, r)$**:** Given integers $v \geq 0$ and $r \in [1 \,..\, \mathsf{rcount}_A(v, m)]$, return the position of the rth smallest element of $\{i \in [1 \,..\, m] : A[i] \geq v\}$.

The currently fastest general-purpose data structure for range counting/selection queries is described in [15, Theorem 2.3, Theorem 3.3]. The instances in our construction, however, satisfy an additional property, namely, that the sum $\sum_{i=1}^{m} A[i]$ is bounded. This lets us obtain a solution with faster query and smaller construction time.

Proposition 3.11. *An array $A[1..m']$ of $m' \in [2..m]$ nonnegative integers satisfying $\sum_{i=1}^{m'} A[i] = \mathcal{O}(m \log m)$ can be preprocessed in $\mathcal{O}(m)$ time so that that range counting and selection queries can be answered in $\mathcal{O}(\log \log m)$ time and $\mathcal{O}(1)$ time, respectively.*

Proof. Denote $h = \lfloor \log m \rfloor$. For any $k \geq 0$, by $P_k[1 \,..\, m_k]$, where $m_k = \mathsf{rcount}_A(kh, m)$, we denote the array defined by $P_k[i] = \mathsf{rselect}_A(kh, i)$. Let $v \geq 0$. We define a bitvector $M_v[1 \,..\, m_k]$, where $k = \lfloor \frac{v}{h} \rfloor$ as follows. For any $i \in [1 \,..\, m_k]$, $M_v[i] = 1$ holds if and only if $A[P_k[i]] \geq v$. For any $k \geq 0$, we define the concatenation $M'_k = M_{kh} M_{kh+1} \cdots M_{(k+1)h-1}$. Let $k_{\max} = \max\{k \geq 0 : m_k > 0\}$. Since all elements of A are nonnegative, and $\sum_{i=1}^{m'} A[i] \in \mathcal{O}(m \log m)$, we obtain $\max_{i \in [1..m']} A[i] \in \mathcal{O}(m \log m)$, and consequently, $k_{\max} = \lfloor \frac{1}{h} \max_{i \in [1..m']} A[i] \rfloor \in \mathcal{O}(m)$.

The data structure consists of two components. First, for $k \in [0 \,..\, k_{\max}]$, we store a plain representation of the sequence $P_k[1 \,..\, m_k]$ using $\mathcal{O}(m_k)$ space. Each array is augmented with a static predecessor data structure. We use [24, Proposition 7], and hence achieve linear space and $\mathcal{O}(\log \log m)$ query time. Each $i \in [1 \,..\, m']$ occurs in $\lceil \frac{A[i]+1}{h} \rceil$ arrays. Thus, $\sum_{k \geq 0} m_k = \sum_{i=1}^{m'} \lceil \frac{A[i]+1}{h} \rceil \leq 2m' + \sum_{i=1}^{m'} \lfloor \frac{A[i]}{h} \rfloor \leq 2m' + \frac{1}{h} \sum_{i=1}^{m'} A[i] \in \mathcal{O}(m)$ and hence we can store the arrays P_k (including the associated predecessor data structures) using $\mathcal{O}((k_{\max}+1) + \sum_{k \geq 0} m_k) \subseteq \mathcal{O}(m)$ space, so that we can access each array in $\mathcal{O}(1)$ time. Second, for every $k \in [0 \,..\, k_{\max}]$, we store the plain representation of bitvector M'_k, augmented using Theorem 2.1. By $|M'_k| = h \cdot m_k$, the total length of bitvectors M'_k is $\sum_{k \geq 0} |M'_k| = h \sum_{k \geq 0} m_k \in \mathcal{O}(m \log m)$. All bitvectors M'_k can thus be stored in $\mathcal{O}((k_{\max}+1) + \frac{1}{\log m} \sum_{k \geq 0} |M'_k|) \subseteq \mathcal{O}(m)$ words of space, so that we can access each in $\mathcal{O}(1)$ time. For a bitvector of length t, the augmentation of Theorem 2.1 adds only $\mathcal{O}(\log m + t)$ bits of space, and hence does not increase the space usage.

Using the above two components, we answer range counting/selection queries on A as follows. To compute $\mathsf{rcount}_A(v, j)$, we observe that if $j' = \max\{i \in [1 \,..\, m_k] : P_k[i] \leq j\}$, where $k = \lfloor \frac{v}{h} \rfloor$, then $\mathsf{rcount}_A(v, j) = \mathsf{rank}_{M_v,1}(j')$. Computing j' using the predecessor data structure takes $\mathcal{O}(\log \log m)$ time, and then $\mathsf{rank}_{M_v,1}(j')$ is computed using the rank support data structure of the bitvector M'_k as $\mathsf{rank}_{M'_k,1}(j' + (v - kh)m_k) - \mathsf{rank}_{M'_k,1}((v - kh)m_k)$ in $\mathcal{O}(1)$ time. To compute $\mathsf{rselect}_A(v, s)$, we observe that letting again $k = \lfloor \frac{v}{h} \rfloor$, it holds $\mathsf{rselect}_A(v, s) = P_k[\mathsf{select}_{M_v,1}(s)]$. The value $\mathsf{select}_{M_v,1}(s)$ is computed using the select support data structure of the bitvector M'_k as $\mathsf{select}_{M'_k,1}(\mathsf{rank}_{M'_k,1}((v - kh)m_k) + s) - (v - kh)m_k$ in $\mathcal{O}(1)$ time.

The data structure is constructed as follows. We start by initializing $P_0[i] = i$ for $i \in [1 \,..\, m']$. For $k \in [1 \,..\, k_{\max}]$, the array P_k is computed by iterating over P_{k-1} and including only elements $P_{k-1}[i]$ satisfying $A[P_{k-1}[i]] \geq kh$. By $\sum_{k \geq 0} m_k \in \mathcal{O}(m)$, this takes $\mathcal{O}(m)$ time in total. We then augment all arrays P_k with the predecessor data structures. Since the arrays are sorted, using [24, Proposition 7], the construction altogether again takes $\mathcal{O}(m)$ time. We then construct

bitvectors M'_k in the order of increasing $k \in [0\,..\,k_{\max}]$. To build M'_k we first scan P_k and check if there exists $i \in [1\,..\,m_k]$ such that $A[P_k[i]] < (k+1)h$.

1. If there is no such i, we set $M'_k := 1^{hm_k}$ in $\mathcal{O}(1 + \frac{1}{\log m} hm_k) = \mathcal{O}(m_k)$ time.
2. Otherwise, we scan again $P_k[1\,..\,m_k]$ and prepare h lists $L_0, L_1, \ldots, L_{h-1}$ such that L_y contains all $i \in [1\,..\,m_k]$ satisfying $A[P_k[i]] = kh + y$. Construction of all lists takes $\mathcal{O}(m_k + h)$ time. The bitvector M'_k is then obtained as the concatenation of bitvectors $M_{kh}, M_{kh+1}, \ldots, M_{(k+1)h-1}$ computed in this order. We first initialize $M_{kh} := 1^{m_k}$ in $\mathcal{O}(1 + \frac{m_k}{\log m})$ time. The bitvector M_{kh+y} for $y > 0$ is obtained by first copying the bitvector M_{kh+y-1} in $\mathcal{O}(1 + \frac{m_k}{\log m})$ time, and then setting $M_{kh+y}[i] = 0$ for every position i stored in L_{y-1}. The total length of all lists L_y is bounded by m_k. Thus, the construction of M'_k takes $\mathcal{O}(h + m_k + \frac{1}{\log m} hm_k) \subseteq \mathcal{O}(h + m_k)$ time.

To bound the total time spent constructing bitvectors M'_k, we consider two cases:

- $k \leq \frac{m}{h}$: The total time spent in the construction of bitvectors M'_k for such k is bounded by the sum $\sum_{k=0}^{\lfloor m/h \rfloor} \mathcal{O}(h + m_k) \subseteq \mathcal{O}(m + \sum_{k \geq 0} m_k) \subseteq \mathcal{O}(m)$.
- $k > \frac{m}{h}$: Let $k' = \lfloor \frac{m}{h} \rfloor + 1$. Note that for any t, it holds $m_{t+1} \leq m_t$. Moreover, whenever Case 2 above happens for some t, it holds $m_{t+1} < m_t$. Thus, Case 2 above can happen for $k > \frac{m}{h}$ only $m_{k'}$ times. Since for every $i \in [1\,..\,m_{k'}]$ we have $A[P_{k'}[i]] \geq m$, by $\sum_{i \in [1\,..\,m']} A[i] \in \mathcal{O}(m \log m)$ it holds $m_{k'} \in \mathcal{O}(\log m)$. The total time spend computing M'_k for $k > \frac{m}{h}$ is thus bounded by $\mathcal{O}(m_{k'}(h + m_{k'}) + \sum_{k \geq k'} m_k) \subseteq \mathcal{O}(\log^2 m + \sum_{k \geq 0} m_k) \subseteq \mathcal{O}(m)$.

The total length of bitvectors M'_k for $k \in [0\,..\,k_{\max}]$, is $\sum_{k \in [0\,..\,k_{\max}]} hm_k \in \mathcal{O}(hm)$. Thus, augmenting them all using Theorem 2.1 takes $\mathcal{O}((k_{\max} + 1) + \frac{1}{\log m} hm) \subseteq \mathcal{O}(m)$ time. □

Proposition 3.12. *In $\mathcal{O}(n/\log_\sigma n)$ time, we can augment the structure from Proposition 3.7 so that $\delta^s(j)$ can be computed in $\mathcal{O}(\log \log n)$ time given $j \in \mathsf{R}^-$.*

Proof. The main idea of the data structure is as follows. We group all $i \in \mathsf{R}'^-$ by L-root(i) and then sort by $T[e^{\text{full}}_i\,..\,n]$ all positions within each group. Then, by Lemma 3.10, to compute $\delta^s(j)$ given $j \in \mathsf{R}^-_H$, it suffices to count all $i \in \mathsf{R}'^-_H$ that satisfy $T[e^{\text{full}}_i\,..\,n] \succ T[e^{\text{full}}_j\,..\,n]$ and $e^{\text{full}}_i - i \geq e^{\text{full}}_j - j$. The condition $T[e^{\text{full}}_i\,..\,n] \succ T[e^{\text{full}}_j\,..\,n]$ is ensured by considering only $i \in \mathsf{R}'^-_H$ that are later in the sorted order than $j' \in \mathsf{R}'^-_H$ satisfying $j \in [j'\,..\,e_{j'} - 3\tau + 2)$, since for such j' we have $T[e^{\text{full}}_{j'}\,..\,n] = T[e^{\text{full}}_j\,..\,n]$. Counting the elements that simultaneously satisfy also the second condition can then be reduced to a range counting query.

We use the following definitions. Let $B_{\mathsf{R}'}[1\,..\,n]$ be a bitvector defined such that $B_{\mathsf{R}'}[i] = 1$ holds if and only if $i \in \mathsf{R}'$. Let $(r_i)_{i \in [1\,..\,|\mathsf{R}'^-|]}$ be a sequence containing all elements of R'^- in sorted order, i.e, for any $i, i' \in [1\,..\,|\mathsf{R}'^-|]$, $i < i'$ implies $r_i < r_{i'}$. Let $(r'_i)_{i \in [1\,..\,|\mathsf{R}'^-|]}$ also be a sequence containing all elements $k \in \mathsf{R}'^-$, but sorted first according to L-root(k) and in case of ties, by $T[e^{\text{full}}_k\,..\,n]$. Formally, for any $i, i' \in [1\,..\,|\mathsf{R}'^-|]$, $i < i'$ implies that L-root$(r'_i) \prec$ L-root$(r'_{i'})$, or L-root$(r'_i) =$ L-root$(r'_{i'})$ and $T[e^{\text{full}}_k\,..\,n] \prec T[e^{\text{full}}_{k'}\,..\,n]$, where $k = r'_i$ and $k' = r'_{i'}$. Based on (r'_i) we define the sequence of integers $(\ell_i)_{i \in [1\,..\,|\mathsf{R}'^-|]}$ as $\ell_i = e^{\text{full}}_k - k$, where $k = r'_i$. We define the array $A[1\,..\,|\mathsf{R}'^-|]$ by $A[i] = \ell_i$. In addition, let $M[1\,..\,|\mathsf{R}'^-|]$ be a mapping of (r_i) to (r'_i), i.e., M contains a permutation of $[1\,..\,|\mathsf{R}'^-|]$ such that $M[i] = i'$ holds if and only if $r_i = r'_{i'}$. Finally, let C be a lookup table, mapping $H \in \mathcal{H}$ to the value $\sum_{H' \in \mathcal{H}: H' \preceq H} |\mathsf{R}'^-_{H'}|$.

The data structure, in addition to the data structure from Proposition 3.7, contains four components. First, we store the bitvector $B_{\mathsf{R}'}$ augmented using Theorem 2.1 to answer $\mathcal{O}(1)$-time rank queries. The augmented bitvector takes $\mathcal{O}(n/\log n)$ words of space. Second, we store the array M in plain form using $\mathcal{O}(1 + |\mathsf{R}'^-|) = \mathcal{O}(n/\log_\sigma n)$ space. Third, we store the lookup table C. Assuming, similarly as in Section 3.1, that when accessing C, strings H are converted to integers as int(H), C can be stored using $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ space. Finally, we store the

array $A[1\,..\,|\mathsf{R}'^{-}|]$ augmented with a range counting data structure from Proposition 3.11. The augmented array needs $\mathcal{O}(1+|\mathsf{R}'^{-}|)=\mathcal{O}(n/\log_\sigma n)$ space. To see that the sum of elements of A is $\mathcal{O}(n)$, we note that if $j\in\mathsf{R}'$, then by [39, Fact 3.2], it holds $\mathrm{per}(T[j\,..\,e_j))\le\frac{1}{3}\tau$, $e_j-j\ge 3\tau-1$, and the substring $T[j\,..\,e_j)$ cannot be extended in T in either left of right without changing its shortest period. Thus, by [41, Fact 2.2.4], if $j,j'\in\mathsf{R}'$ and $j\neq j'$, then the fragments $T[j\,..\,e_j)$ and $T[j'\,..\,e_{j'})$ do not overlap by more than $\frac{2}{3}\tau$ symbols. Thus, each position of T belongs to at most two intervals in the collection $\{[j\,..\,e_j):j\in\mathsf{R}'\}$, and consequently, $\sum_{i=1}^{|\mathsf{R}'^{-}|}\ell_i\le 2n$.

Using the data structure from Proposition 3.7 and the above four components, we answer $\delta^s(j)$ queries as follows. Given $j\in\mathsf{R}^-$, we first compute $H=\text{L-root}(j)$, $s=\text{L-head}(j)$, and $k=\text{L-exp}(j)$. By Proposition 3.7, this takes $\mathcal{O}(1)$ time. This lets us deduce $e_j^{\mathrm{full}}=j+s+k|H|$. Then, we compute $i\in[1\,..\,|\mathsf{R}'^{-}|]$ satisfying $j\in[r_i\,..\,e_{r_i}-3\tau+2)$, i.e., j is in the maximal block of positions from R^- starting at position r_i. Using $B_{\mathsf{R}'}$ we obtain $i=\mathsf{rank}_{B_{\mathsf{R}'},1}(j)$ in $\mathcal{O}(1)$ time. Observe now that, letting $j'=r_i$, by $e_{j'}^{\mathrm{full}}=e_j^{\mathrm{full}}$, we have $T[e_{j'}^{\mathrm{full}}\,..\,n]=T[e_j^{\mathrm{full}}\,..\,n]$. Therefore, letting $x=M[i]$ and $x'=C[\mathrm{int}(H)]$, by Lemma 3.10 we have $\delta^s(j)=|\{i'\in(x\,..\,x']:\ell_{i'}\ge e_j^{\mathrm{full}}-j\}|=\mathsf{rcount}_A(e_j^{\mathrm{full}}-j,x')-\mathsf{rcount}_A(e_j^{\mathrm{full}}-j,x)$, which we compute in $\mathcal{O}(\log\log n)$ time using the data structure from Proposition 3.11.

The data structure is constructed as follows. We start by computing $B_{\mathsf{R}'}$. As seen in the proof of Proposition 3.9, we can enumerate R', and thereby compute $B_{\mathsf{R}'}$, in $\mathcal{O}(|\mathsf{S}|+n/\log n)=\mathcal{O}(n/\log_\sigma n)$ time. Augmenting $B_{\mathsf{R}'}$ with Theorem 2.1 takes $\mathcal{O}(n/\log n)$ time. Since for any $j\in\mathsf{R}$, we can in $\mathcal{O}(1)$ compute $\text{L-root}(j)$, e_j, $s=\text{L-head}(j)$, and $k=\text{L-exp}(j)$, in $\mathcal{O}(1+|\mathsf{S}|)=\mathcal{O}(n/\log_\sigma n)$ time we can also enumerate all $j\in\mathsf{R}'^{-}$. The key challenge is computing the sequence (r'_i). Assume that we have computed the array $\mathrm{ISA}_{\mathsf{S}}[1\,..\,n']$, as defined in Section 3.2. As explained in Proposition 3.4, we can construct it from $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$ in $\mathcal{O}(n/\log_\sigma n)$ time. Next, for each $j\in\mathsf{R}'^{-}$, letting $H=\text{L-root}(j)$ and $j_{\mathsf{S}}=\mathsf{rank}_{B,1}(e_j-2\tau+1)$ (recall that $e_j-2\tau+1\in\mathsf{S}$), we form a tuple $(\mathrm{int}(H),e_j-e_j^{\mathrm{full}},\mathrm{ISA}_{\mathsf{S}}[j_{\mathsf{S}}],j)$. As observed in Proposition 3.9, $X\prec X'$ holds if and only if $\mathrm{int}(X)<\mathrm{int}(X')$. Let $j,j'\in\mathsf{R}'^{-}_H$. Note that since both $T[e_j^{\mathrm{full}}\,..\,e_j)$ and $T[e_{j'}^{\mathrm{full}}\,..\,e_{j'})$ are prefixes of H, by definition of R^-, $e_j-e_j^{\mathrm{full}}<e_{j'}-e_{j'}^{\mathrm{full}}$ implies $T[e_j^{\mathrm{full}}\,..\,n]\prec T[e_{j'}^{\mathrm{full}}\,..\,n]$. If $e_j-e_j^{\mathrm{full}}=e_{j'}-e_{j'}^{\mathrm{full}}$, then $T[e_j-2\tau+1\,..\,e_j^{\mathrm{full}})=T[e_{j'}-2\tau+1\,..\,e_{j'}^{\mathrm{full}})$, and consequently, $T[e_j^{\mathrm{full}}\,..\,n]\prec T[e_{j'}^{\mathrm{full}}\,..\,n]$ holds if and only if $\mathrm{ISA}_{\mathsf{S}}[j_{\mathsf{S}}]<\mathrm{ISA}_{\mathsf{S}}[j'_{\mathsf{S}}]$. We have thus shown that sorting the tuples lexicographically yields a sequence (r'_i) on the fourth coordinate. Given $j\in\mathsf{R}'^{-}$, we can compute the corresponding tuple in $\mathcal{O}(1)$ time. Thus, since all its elements are integers in the range $[1\,..\,n]$, using LSD radix-sort, we can compute (r'_i) in $\mathcal{O}(n/\log_\sigma n)$ time. With a single scan of (r'_i) (and the help of rank queries on $B_{\mathsf{R}'}$) we can then compute tables C and M in $\mathcal{O}(n/\log_\sigma n)$ time. Finally, from (r'_i) we construct in $\mathcal{O}(n/\log_\sigma n)$ time the sequence $(\ell_i)_{i\in[1\,..\,|\mathsf{R}'^{-}|]}$, and then build the array $A[1\,..\,|\mathsf{R}'^{-}|]$ and augment it with a range counting data structure. Using Proposition 3.11, by $|\mathsf{R}'^{-}|=\mathcal{O}(n/\log_\sigma n)$ and $\sum_{i=1}^{|\mathsf{R}'^{-}|}A[i]=\mathcal{O}(n)$, this takes $\mathcal{O}(n/\log_\sigma n)$ time. □

Summary By combining all above results, we obtain the following data structure computing $\mathrm{ISA}[j]$ for periodic positions.

Proposition 3.13. *In $\mathcal{O}(n/\log_\sigma n)$ time, we can augment $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$ so that $\mathrm{ISA}[j]$ can be computed in $\mathcal{O}(\log\log n)$ time given $j\in\mathsf{R}$.*

Proof. The data structure is a composition of four data structures. First, we store the two data structures presented in Propositions 3.9 and 3.12. These structures are designed to compute values $\delta^a(j)$ and $\delta^s(j)$ for $j\in\mathsf{R}^-$. To handle $j\in\mathsf{R}^+$ we store their symmetric versions adapted according to Lemma 3.5 (more precisely, if $j\in\mathsf{R}^+_{s,H}$, the data structures compute $\delta^a(j)=|\mathrm{pos}^a(j)|$ and $\delta^s(j)=|\mathrm{pos}^s(j)|$, where $\mathrm{pos}^a(j)=\{j'\in\mathsf{R}^+_{s,H}:\text{L-exp}(j')\le\text{L-exp}(j)\}$ and $\mathrm{pos}^s(j)=\{j'\in\mathsf{R}^+_{s,H}:\text{L-exp}(j)=\text{L-exp}(j)\text{ and }T[j'\,..\,n]\prec T[j\,..\,n]\}$). All four data structures

take $\mathcal{O}(n/\log_\sigma n)$ space. All structures are built on top of the structure from Proposition 3.7, but it suffices to only keep its single copy (in particular, we keep a single copy of $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$).

Using the above data structures, given $j \in \mathsf{R}$, we compute $\mathrm{ISA}[j]$ as follows. First, using Lemma 3.1, in $\mathcal{O}(1)$ time we compute integers b, e such that $\mathrm{SA}(b\,..\,e]$ contains the starting positions of all suffixes of T starting with the prefix of $T[j\,..\,n]$ from the set $\mathcal{F}$. Then, using Proposition 3.7 we determine $\mathrm{type}(j)$. Depending on whether $j \in \mathsf{R}^-$ or $j \in \mathsf{R}^+$ we use either a combination of Propositions 3.9 and 3.12 or their symmetric counterparts, to compute $\delta^a(j)$ and $\delta^s(j)$ in $\mathcal{O}(1)$ and $\mathcal{O}(\log\log n)$ time, respectively. If $j \in \mathsf{R}^-$, then by Lemma 3.8 we have $\delta(j) = \delta^a(j) - \delta^s(j)$. Otherwise, by the counterpart of Lemma 3.8, $\delta(j) = (e-b) - (\delta^a(j) - \delta^s(j))$. Finally, we return $\mathrm{ISA}[j] = b + \delta(j)$ as the answer. In total, the query takes $\mathcal{O}(\log\log n)$ time.

The data structure is constructed as follows. First, given $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$ we construct the data structure from Proposition 3.7. This takes $\mathcal{O}(n/\log_\sigma n)$ time. We then augment it into the two data structures to compute $\delta^a(j)$ and $\delta^s(j)$ for $j \in \mathsf{R}^-$. Using Propositions 3.9 and 3.12, this takes $\mathcal{O}(n/\log_\sigma n)$. Finally, we analogously construct the two data structures to compute $\delta^a(j)$ and $\delta^s(j)$ for $j \in \mathsf{R}^+$. □

3.4 The Final Data Structure

Theorem 3.14. *Given a constant $\epsilon \in (0,1)$ and the packed representation of a text $T \in [0\,..\,\sigma)^n$ with $2 \le \sigma < n^{1/6}$, we can construct in $\mathcal{O}(n\log\sigma/\sqrt{\log n})$ time and $\mathcal{O}(n/\log_\sigma n)$ working space a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that, given any $j \in [1\,..\,n]$, returns $\mathrm{ISA}[j]$ in $\mathcal{O}(\log^\epsilon n)$ time.*

Proof. The data structure is a composition of the data structures from Proposition 3.4 and Proposition 3.13. Both data structures take $\mathcal{O}(n/\log_\sigma n)$ space. Each of the two structures is built on top of the index core $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$, but it suffices to store a single copy of $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$.

Given $j \in [1\,..\,n]$, we use Lemma 3.1 to check in $\mathcal{O}(1)$ time if $j \in \mathsf{R}$. Depending on whether $j \in \mathsf{R}$ or not, we use Proposition 3.4 or Proposition 3.13 to compute $\mathrm{ISA}[j]$ in $\mathcal{O}(\log^\epsilon n)$ or $\mathcal{O}(\log\log n)$ time (respectively).

The data structure is constructed as follows. First, using Theorem 2.5, from a packed representation of T, we construct a τ-synchronizing set S of size $\mathcal{O}(n/\tau)$ in $\mathcal{O}(n/\tau) = \mathcal{O}(n/\log_\sigma n)$ time. The set S is returned as an array taking $\mathcal{O}(n/\log_\sigma n)$ space. Using this array and the packed representation of T, we construct $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$ in $\mathcal{O}(n/\log_\sigma n)$ time using Proposition 3.2. Finally, using Propositions 3.4 and 3.13, we augment $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$ in $\mathcal{O}(n\log\sigma/\sqrt{\log n})$ and $\mathcal{O}(n/\log_\sigma n)$ time (respectively) and using $\mathcal{O}(n/\log_\sigma n)$ working space into structures to compute $\mathrm{ISA}[j]$. □

4 SA Queries

In this section, we describe the index answering the SA queries. More precisely, we show how, given the packed representation of $T \in [0\,..\,\sigma)^n$ with $2 \le \sigma < n^{1/6}$, in $\mathcal{O}(n\log\sigma/\sqrt{\log n})$ time and using $\mathcal{O}(n/\log_\sigma n)$ working space construct a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that, given $i \in [1\,..\,n]$, returns $\mathrm{SA}[i]$ in $\mathcal{O}(\log^\epsilon n)$ time. Note, that we can assume $\sigma < n^{1/6}$, since for larger σ, the construction in [5, Theorem 9] already gives a construction in $\mathcal{O}(n) = \mathcal{O}(n\log\sigma/\sqrt{\log n})$ time.

As in Section 3, we let $\tau = \mu\log n$, where μ is any positive constant smaller than $\frac{1}{6}$, and we let S be any τ-synchronizing set satisfying $n' := |\mathsf{S}| = \mathcal{O}(n/\tau)$. We again exploit the fact that the set $\mathcal{D} \cup \mathcal{F}$ is prefix-free, and all suffixes in the SA of T can be partitioned into blocks, according to their prefix from $\mathcal{D} \cup \mathcal{F}$, except rather than computing the value $\delta(j)$ (see Section 3) given the position $j \in [1\,..\,n]$ in text, and then returning $\mathrm{ISA}[j] = b + \delta(j)$ (where $\mathrm{SA}(b\,..\,e]$ contains the starting positions of all suffixes of T prefixed with the string $X \in \mathcal{D} \cup \mathcal{F}$ that is a prefix of

$T[j\,..\,n]$), we instead, given index $i \in [1\,..\,n]$, first determine the range $\mathrm{SA}(b\,..\,e]$ corresponding to some $X \in \mathcal{D} \cup \mathcal{F}$ and containing index i, and then find $j \in [1\,..\,n]$ such that $\delta(j) = i - b$.

Organization The section is organized as follows. Our description is split into four parts. First (Section 4.1), we describe the data structures, called collectively the index "core", that enable efficiently checking if $\mathrm{SA}[i] \in \mathsf{R}$, and to compute the prefix $X \in \mathcal{D} \cup \mathcal{F}$ as well as the endpoints of the corresponding block $\mathrm{SA}(b\,..\,e]$. The structure and query algorithm to compute j such that $\delta(j) = i - b$ (and thus $\mathrm{SA}[i] = j$) is, similarly as in Section 3, different depending on whether $X \in \mathcal{D}$ (i.e., $\mathrm{SA}[i] \in [1\,..\,n] \setminus \mathsf{R}$) or $X \in \mathcal{F}$ (i.e., $\mathrm{SA}[i] \in \mathsf{R}$), and the structures used for the two cases are described in the next two parts (Sections 4.2 and 4.3). All ingredients are finally put together in Section 4.4.

4.1 The Index Core

We use the following definitions. Let $B'[0\,..\,n]$ be a bitvector defined so that $B'[0] = 1$ and for any $i \in [1\,..\,n]$, $B'[i] = 1$ holds if and only if $i = n$, or $i < n$ and $X_{\mathrm{SA}[i]} \neq X_{\mathrm{SA}[i+1]}$, where for any $j \in [1\,..\,n]$, X_j denotes the prefix of $T[j\,..\,n]$ satisfying $X_j \in \mathcal{D} \cup \mathcal{F}$. Next, define $\mathrm{SA}_{\mathcal{D}\cup\mathcal{F}}[1\,..\,t]$ (where $t = |\mathcal{D}\cup\mathcal{F}|$) to be an array storing the lexicographically sorted strings from $\mathcal{D}\cup\mathcal{F}$. Finally, let $F[1\,..\,t]$ denote a bitvector such that $F[i] = 1$ if and only if $\mathrm{SA}_{\mathcal{D}\cup\mathcal{F}}[i] \in \mathcal{F}$.

The index core, denoted $\mathrm{C}_{\mathrm{SA}}(T, \mathsf{S})$, consists of five components. First, we store the packed representation of T using $\mathcal{O}(n/\log_\sigma n)$ space. Second, we store the bitvectors B (as defined in Section 3.1) and B' augmented using Theorem 2.1 for rank and selection queries. The two bitvectors take $\mathcal{O}(n/\log n)$ space. Third, we store the array $\mathrm{SA}_{\mathcal{D}\cup\mathcal{F}}$. Every string $X \in \mathcal{D} \cup \mathcal{F}$ is encoded as $\mathrm{int}(X)$ (see Section 3.1) using $6\tau \log \sigma = \mathcal{O}(\log n)$ bits. This implicitly encodes the length of the string and ensures that all strings are encoded using the same number of bits. By $\mathcal{D}\cup\mathcal{F} \subseteq [0\,..\,\sigma)^{\leq 3\tau}$, we have $t = \mathcal{O}(n^{1/2})$, and hence the array $\mathrm{SA}_{\mathcal{D}\cup\mathcal{F}}$ needs $\mathcal{O}(n^{1/2}) = \mathcal{O}(n/\log n)$ space. Fourth and final, we store the bitvector F in plain form using $\lceil t/\log n \rceil$ words of space.

Lemma 4.1. *Given $\mathrm{C}_{\mathrm{SA}}(T, \mathsf{S})$, for any $i \in [1\,..\,n]$ we can in $\mathcal{O}(1)$ time determine if $\mathrm{SA}[i] \in \mathsf{R}$, compute the prefix $X \in \mathcal{D} \cup \mathcal{F}$ of $T[\mathrm{SA}[i]\,..\,n]$, and integers b, e such that $\mathrm{SA}(b\,..\,e]$ contains the starting indexes of all suffixes of T prefixed with X.*

Proof. Given the position $i \in [1\,..\,n]$, we first compute $y = \mathsf{rank}_{B',1}(i-1)$. Then, using $\mathrm{SA}_{\mathcal{D}\cup\mathcal{F}}[y]$, we obtain the string X that is a prefix of $T[\mathrm{SA}[i]\,..\,n]$ and $X \in \mathcal{D} \cup \mathcal{F}$. We then compute $b = \mathsf{select}_{B',1}(y)$ and $e = \mathsf{select}_{B',1}(y+1)$. Finally, to determine if $\mathrm{SA}[i] \in \mathsf{R}$, we check if $F[y] = 1$. All operations, including the decoding of X from $\mathrm{int}(X)$, take $\mathcal{O}(1)$ time. □

Proposition 4.2. *Given the packed representation of $T \in [0\,..\,\sigma)^n$ and the array containing elements of S, we can construct $\mathrm{C}_{\mathrm{SA}}(T, \mathsf{S})$ in $\mathcal{O}(n/\log_\sigma n)$ time.*

Proof. Given S, we easily initialize the bitvector B in $\mathcal{O}(|\mathsf{S}| + n/\log n) = \mathcal{O}(n/\log_\sigma n)$ time. Augmenting B using Theorem 2.1 takes $\mathcal{O}(n/\log n)$ time.

To initialize the remaining three components of $\mathrm{C}_{\mathrm{SA}}(T, \mathsf{S})$, we first compute the frequency $f_X = |\{i \in [1\,..\,n] : X \text{ is a prefix of } T[i\,..\,n]\}|$ for every $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$. Using the algorithm presented in Proposition 3.2, this takes $\mathcal{O}(n/\log_\sigma n)$ time. Next, we compute a lookup table that for every $X \in [0\,..\,\sigma)^{2\tau}$ tells whether $T[j\,..\,j+2\tau) = X$ implies $j \in \mathsf{S}$ (by consistency of S this does not depend on j). Given the array containing the positions in S and the packed representation of T, this takes $\mathcal{O}(\sigma^{2\tau} + |\mathsf{S}|) = \mathcal{O}(n/\log_\sigma n)$ time. To compute $\mathrm{SA}_{\mathcal{D}\cup\mathcal{F}}$, B', and F, we first initialize the bitvector B' in $\mathcal{O}(n/\log n)$ time to $\mathtt{10}^n$. We then simulate a preorder traversal of the trie of $[0\,..\,\sigma)^{3\tau-1}$. For each visited node with label X satisfying $|X| \geq 2\tau$, we check if the length-2τ suffix of X is in the lookup table.

- If the suffix is in the lookup table, we check if $f_X > 0$. If so we report that $X \in \mathcal{D}$. We then (regardless of whether $f_X > 0$) skip the traversal of the current subtree.
- Otherwise (i.e., if the length-2τ suffix is not in the lookup table), we descend into the subtree rooted in the current node.

Whenever during the traversal we reach a node at depth $3\tau - 1$, we check if its frequency is positive. If so, we report that $X \in \mathcal{F}$. During the above algorithm, we maintain the number k, and the total frequency f of the strings $X \in \mathcal{D} \cup \mathcal{F}$ reported so far. For every reported string X, we increment k, set $\mathrm{SA}_{\mathcal{D}\cup\mathcal{F}}[k] = \mathrm{int}(X)$, increment f by f_X, set $B'[f] = 1$, and finally, set $F[k]$ to 1 if $X \in \mathcal{F}$. Since the labels of nodes visited during the preorder traversal are lexicographically sorted, the strings in $\mathcal{D} \cup \mathcal{F}$ are reported in correct order. The traversal takes $\mathcal{O}(\sigma^{3\tau}) = \mathcal{O}(n^{3\mu}) = \mathcal{O}(n/\log n)$ time. Finally, we use Theorem 2.1 to augment bitvector B' in $\mathcal{O}(n/\log n)$ time. □

4.2 The Nonperiodic Positions

In this section, we describe a data structure to compute the value $\mathrm{SA}[i]$ for any $i \in [1\,..\,n]$ such that $\mathrm{SA}[i] \in [1\,..\,n] \setminus \mathsf{R}$.

The main idea of the data structure is as follows. Let $X \in \mathcal{D}$ be the prefix of $T[\mathrm{SA}[i]\,..\,n]$. As observed in Section 3.2, the offset of the first position from S in every suffix prefixed with X is $\delta_{\text{text}} = |X| - 2\tau$ and the order of these suffixes is the same as the order of corresponding suffixes starting at positions of S. Thus, if we sort only the suffixes starting at positions in S, then computing j such that $\delta(j) = i - b$ (where b is defined as in Lemma 4.1) reduces to selecting the $(i-b)$th position in S in the above order that is followed by $X(\delta_{\text{text}}\,..\,|X|]$ and preceded with $X[1\,..\,\delta_{\text{text}}]$ in the text. Similarly as in Section 3.1, we implement that query using Theorem 2.2.

We use the following definitions. Let $\mathrm{SA}_{\mathsf{S}}[1\,..\,n']$ be an array storing a permutation of $[1\,..\,n']$ such that $\mathrm{SA}_{\mathsf{S}}[i] = j$ holds if and only if $s_j = s'_i$, where $(s_t)_{t\in[1\,..\,n']}$ and $(s'_t)_{t\in[1\,..\,n']}$ are the sequences defined as in Section 3.2.

Proposition 4.3. *Given a constant $\epsilon \in (0,1)$ and $\mathrm{C}_{\mathrm{SA}}(T,\mathsf{S})$, we can in $\mathcal{O}(n\log\sigma/\sqrt{\log n})$ time and using $\mathcal{O}(n/\log_\sigma n)$ working space augment it into a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that, given $i \in [1\,..\,n]$ such that $\mathrm{SA}[i] \in [1\,..\,n] \setminus \mathsf{R}$, returns $\mathrm{SA}[i]$ in $\mathcal{O}(\log^\epsilon n)$ time.*

Proof. The data structure, in addition to $\mathrm{C}_{\mathrm{SA}}(T,\mathsf{S})$, contains three components. First, we store the array $\mathrm{SA}_{\mathsf{S}}[1\,..\,n']$ in plain form, using $n' = \mathcal{O}(n/\log_\sigma n)$ words of space. Second, we store a lookup table of size $\mathcal{O}(n^\delta)$, for some $\delta < 1$, that given the packed representation of any string $X \in [0\,..\,\sigma)^*$ of length $|X| = \mathcal{O}(\log_\sigma n)$, allows us to compute the packed representation of $\overline{X}$ in $\mathcal{O}(1)$ time (see also Proposition 3.4). Third and final, we store the sequence W (defined as in Section 3.2) augmented with a component of Theorem 2.2. Due to $n' = \mathcal{O}(n/\log_\sigma n)$ and $\sigma^{3\tau} = o(n/\log n)$, this requires $\mathcal{O}(n/\log_\sigma n)$ space.

Using $\mathrm{C}_{\mathrm{SA}}(T,\mathsf{S})$ and the above two components, given $i \in [1\,..\,n]$ such that $\mathrm{SA}[i] \in [1\,..\,n] \setminus \mathsf{R}$, we compute $\mathrm{SA}[i]$ as follows. First, using Lemma 4.1, in $\mathcal{O}(1)$ time we compute $X = D_{\mathrm{SA}[i]}$ and integers b, e such that $\mathrm{SA}(b\,..\,e]$ contains the starting positions of all suffixes of T starting with X. We then compute the position $y \in [1\,..\,n']$ of the $(i-b)$th string in $W[1\,..\,n']$ having $\overline{X}$ as a prefix, i.e., $y = \mathsf{select}_{W,\overline{X}}(i-b)$. For this, we use a prefix selection query of Theorem 2.2. Finally, we compute $j' = \mathsf{select}_{B,1}(\mathrm{SA}_{\mathcal{D}\cup\mathcal{F}}[y])$ and return $\mathrm{SA}[i] = j' - \delta_{\text{text}}$, where $\delta_{\text{text}} = |X| - 2\tau$. Altogether, computing $\mathrm{SA}[i]$ thus takes $\mathcal{O}(\log^\epsilon n)$ time.

The data structure is constructed as follows. We first construct the array SA_{S}. We start by computing the sequence $(s'_t)_{t\in[1\,..\,n']}$. In Proposition 3.4 we have shown how to achieve this in $\mathcal{O}(n/\log_\sigma n)$, given the packed representation of T, and bitvector B augmented with $\mathcal{O}(1)$-time select queries (which are part of $\mathrm{C}_{\mathrm{SA}}(T,\mathsf{S})$). We then easily obtain the array SA_{S}: simply scan

the sequence $(s'_t)_{t\in[1..n']}$ and for each $i \in [1..n']$, let $j = \mathsf{rank}_{B,1}(s'_i)$ and note that then $s_j = s'_i$ and hence we can set $\mathrm{SA}_{\mathsf{S}}[i] = j$. Next, in $\mathcal{O}(n^\delta)$ time we initialize the lookup table used to reverse short strings (see also Proposition 3.4). Finally, we construct the sequence W and the component of Theorem 2.2. In the proof of Proposition 3.4, we have shown how to achieve this in $\mathcal{O}(n \log \sigma/\sqrt{\log n})$ time and $\mathcal{O}(n/\log_\sigma n)$ working space, given the sequence $(s'_t)_{t\in[1..n']}$ and the packed representation of T. □

4.3 The Periodic Positions

In this section, we describe a data structure to compute the value $\mathrm{SA}[i]$ for any $i \in [1..n]$ such that $\mathrm{SA}[i] \in \mathsf{R}$.

Preliminaries We start the description of the data structure, by first showing the component to compute the basic values characterizing periodic positions.

Proposition 4.4. *In $\mathcal{O}(n/\log_\sigma n)$ time, we can augment $\mathrm{C}_{\mathrm{SA}}(T, \mathsf{S})$ into a data structure that, given any $i \in [1..n]$ such that $\mathrm{SA}[i] \in \mathsf{R}$, returns $\text{L-root}(\mathrm{SA}[i])$ and $\text{L-head}(\mathrm{SA}[i])$ in $\mathcal{O}(1)$ time.*

Proof. In addition to $\mathrm{C}_{\mathrm{SA}}(T, \mathsf{S})$, the data structure contains the lookup table L (as defined in Proposition 3.7) in plain form, using $\mathcal{O}(\sigma^{3\tau-1}) = \mathcal{O}(n/\log_\sigma n)$ space.

Using $\mathrm{C}_{\mathrm{SA}}(T, \mathsf{S})$ and table L, we implement the queries as follows. Given $i \in [1..n]$ such that $\mathrm{SA}[i] \in \mathsf{R}$, we first compute the prefix X of $T[\mathrm{SA}[i]..n]$ of length $3\tau - 1$ and $x \in [0..\sigma^{6\tau})$ such that $x = \mathrm{int}(X)$. Using Lemma 4.1, such X and x can be obtained in $\mathcal{O}(1)$ time. We then look up $(s, p) = L[x]$, and in $\mathcal{O}(1)$ time obtain $\text{L-root}(\mathrm{SA}[i]) = X[1+s..1+s+p)$ and $\text{L-head}(\mathrm{SA}[i]) = s$.

The construction algorithm for the table L is the same as presented in Proposition 3.7. □

The main idea in the algorithm to query $\mathrm{SA}[i]$ for $i \in [1..n]$ satisfying $\mathrm{SA}[i] \in \mathsf{R}$ is as follows. Let $X \in \mathcal{F}$ be the prefix of $T[\mathrm{SA}[i]..n]$. Suppose that we know the range $\mathrm{SA}(b..e]$ containing the starting positions of all suffixes of T prefixed with X. We then observe that, given b and i, we have $i - b = \delta^a(\mathrm{SA}[i]) - \delta^s(\mathrm{SA}[i])$. The outline of the query is to first compute $\text{L-exp}(\mathrm{SA}[i])$ and $\delta^a(\mathrm{SA}[i])$. This gives us $\delta^s(\mathrm{SA}[i]) = b + \delta^a(\mathrm{SA}[i]) - i$, which, along with $\text{L-exp}(\mathrm{SA}[i])$, is then used to compute $\mathrm{SA}[i]$.

Computing $\text{L-exp}(\mathrm{SA}[i])$ *and* $\delta^s(\mathrm{SA}[i])$ We now describe the first step during the computation of $\mathrm{SA}[i]$ for $i \in [1..n]$ satisfying $\mathrm{SA}[i] \in \mathsf{R}$.

Proposition 4.5. *In $\mathcal{O}(n/\log_\sigma n)$ time, we can augment the structure of Proposition 4.4 so that, given any $i \in [1..n]$ such that $\mathrm{SA}[i] \in \mathsf{R}$, we can check if $\mathrm{type}(\mathrm{SA}[i]) = -1$, and if so, return $\text{L-exp}(\mathrm{SA}[i])$ and $\delta^s(\mathrm{SA}[i])$, all in $\mathcal{O}(1)$ time.*

Proof. The data structure, in addition to the data structure from Proposition 4.4, contains two components. First, we store the bitvector E (as defined in Proposition 3.9), augmented using Theorem 2.1 for rank and selection queries. It takes $\mathcal{O}(n/\log n)$ space. Second, we store the mapping Q, as defined in Proposition 3.9. It needs $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ space.

Using the above data structures, given $i \in [1..n]$ such that $\mathrm{SA}[i] \in \mathsf{R}$, we implement the queries as follows. To check if $\mathrm{type}(\mathrm{SA}[i]) = -1$, we first compute integers b, e such that $\mathrm{SA}(b..e]$ contains the starting positions of all suffixes of T starting the prefix $X \in \mathcal{F}$ of $T[\mathrm{SA}[i]..n]$. Using Lemma 4.1, this takes $\mathcal{O}(1)$ time. By Lemma 3.5 we then have $\mathrm{type}(\mathrm{SA}[i]) = -1$ if and only if $E[i..e]$ contains a bit with value 1. This can be checked in $\mathcal{O}(1)$ time by checking if $\mathsf{rank}_{E,1}(e) > \mathsf{rank}_{E,1}(i-1)$. Let us assume $\mathrm{type}(\mathrm{SA}[i]) = -1$. To compute $\text{L-exp}(\mathrm{SA}[i])$, we first compute X (as defined above). Using Proposition 4.4, this takes $\mathcal{O}(1)$ time. We then in $\mathcal{O}(1)$

retrieve $k_{\min} = Q[\text{int}(X)]$, and then compute $\text{L-exp}(\text{SA}[i]) = k_{\min} + (\mathsf{rank}_{E,1}(i-1) - \mathsf{rank}_{E,1}(b))$. Then, $\delta^a(\text{SA}[i])$ can be computed in $\mathcal{O}(1)$ time as $\delta^a(\text{SA}[i]) = \mathsf{select}_{E,1}(\mathsf{rank}_{E,1}(i-1)+1) - b$. Finally, we then obtain $\delta^s(\text{SA}[i]) = b + \delta^a(\text{SA}[i]) - i$.

The data structure is constructed as follows. We first compute the sequence $(s_t)_{t\in[1..n']}$ containing the elements of S in sorted order (see Section 3.2). It can be obtained in $\mathcal{O}(1+|\mathsf{S}|) = \mathcal{O}(n/\log_\sigma n)$ time using select queries on the bitvector B. We then combine Propositions 3.2, 3.7 and 3.9 to construct the data structure from Proposition 3.9 in $\mathcal{O}(n/\log_\sigma n)$ time. This gives us bitvector E and the lookup table Q. We then discard all remaining components of the data structure from Proposition 3.9. □

Computing SA[i] We now describe the data structure to complete the computation of SA[i] for any $i \in [1..n]$ such that $\text{SA}[i] \in \mathsf{R}^-$.

Proposition 4.6. *In $\mathcal{O}(n/\log_\sigma n)$ time, we can augment the structure of Proposition 4.4 so that, given any $i \in [1..n]$ such that $\text{SA}[i] \in \mathsf{R}^-$, along with $\text{L-exp}(\text{SA}[i])$ and $\delta^s(\text{SA}[i])$, we can compute $\text{SA}[i]$ in $\mathcal{O}(\log\log n)$ time.*

Proof. The main idea of the data structure is as follows. Similarly as in Proposition 3.12, we group all $j \in \mathsf{R}'^-$ by $\text{L-root}(j)$, and then sort by $T[e^{\text{full}}_j..n]$ all positions within each group. Then, by Lemma 3.10, to compute $\text{SA}[i]$, given $\text{L-head}(\text{SA}[i])$, $\text{L-root}(\text{SA}[i])$ (which can both computed in $\mathcal{O}(1)$ time using Proposition 4.4) along with $\text{L-exp}(\text{SA}[i])$ and $\delta^s(\text{SA}[i])$, it suffices to first select the element $j \in \mathsf{R}'^-_H$ (where $H = \text{L-root}(\text{SA}[i])$) for which it holds $e^{\text{full}}_j - j \geq \text{L-head}(\text{SA}[i]) + \text{L-exp}(\text{SA}[i])|\text{L-root}(\text{SA}[i])|$, and which is simultaneously the $(\delta^s(\text{SA}[i])+1)$th largest element of R'^-_H according to the string $T[e^{\text{full}}_j..n]$. The position $j' \in [j..e_j - 3\tau + 2)$ satisfying $\text{L-exp}(j') = \text{L-exp}(\text{SA}[i])$ and $\text{L-head}(j') = \text{L-head}(\text{SA}[i])$ must then satisfy $\text{SA}[i] = j'$. To compute position j we use a data structure for range counting/selection. The position j' is then easily obtained.

We use the following definitions. By $M^{-1}[1..|\mathsf{R}'^-|]$ we denote a mapping from (r'_t) to (r_t) (where $(r'_t)_{t\in[1..|\mathsf{R}'^-|]}$ and $(r_t)_{t\in[1..|\mathsf{R}'^-|]}$ are sequences defined as in Proposition 3.12), i.e., M^{-1} contains a permutation of $[1..|\mathsf{R}'^-|]$ such that $M^{-1}[i'] = i$ holds if and only if $r_i = r'_{i'}$.

The data structure, in addition to the data structure from Proposition 4.4, contains four components. First, we store the bitvector $B_{\mathsf{R}'}$ (as defined in Proposition 3.12) augmented using Theorem 2.1; it takes $\mathcal{O}(n/\log n)$ space. Second, we store the array M^{-1} in plain form using $\mathcal{O}(1+|\mathsf{R}'^-|) = \mathcal{O}(n/\log_\sigma n)$ space. Third, we store the lookup table C, as defined in Proposition 3.12. It can be stored using $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ space. Finally, we store the data structure answering range counting/selection queries on the array $A[1..|\mathsf{R}'^-|]$ (as defined in Proposition 3.12). Using Proposition 3.11, the structure needs $\mathcal{O}(1+|\mathsf{R}'^-|) = \mathcal{O}(n/\log_\sigma n)$ space.

Using the data structure from Proposition 4.4 and the above four components, given $i \in [1..n]$ such that $\text{SA}[i] \in \mathsf{R}^-$, along with $\text{L-exp}(\text{SA}[i])$ and $\delta^s(\text{SA}[i])$, we compute $\text{SA}[i]$ as follows. First, we compute $H = \text{L-root}(\text{SA}[i])$ and $\text{L-head}(\text{SA}[i])$ in $\mathcal{O}(1)$ time using Proposition 4.4. This lets us deduce that $e^{\text{full}}_{\text{SA}[i]} - \text{SA}[i] = \ell$, where $\ell = \text{L-head}(\text{SA}[i]) + \text{L-exp}(\text{SA}[i])|H|$. Let $x = C[\text{int}(X)]$, where $X \in \mathcal{F}$ is the prefix of $T[\text{SA}[i]..n]$ (we can compute X in $\mathcal{O}(1)$ time using Proposition 4.2). Next, we compute $\delta = \mathsf{rcount}_A(\ell, x)$. Using the structure from Proposition 3.11, this takes $\mathcal{O}(\log\log n)$ time. Let $q = \delta - \delta^s(\text{SA}[i])$. We then compute the position $p \in [1..|\mathsf{R}'^-|]$ of the qth leftmost element in A that it greater or equal than ℓ. Using Proposition 3.11, we compute $p = \mathsf{rselect}_A(\ell, q)$ in $\mathcal{O}(1)$ time. Next, we compute in $\mathcal{O}(1)$ time $p' = M^{-1}[p]$ and $j = \mathsf{select}_{B_{R'},1}(p')$. By Lemma 3.10, we now have that $j \in \mathsf{R}'^-_H$, and $\text{SA}[i]$ is one of the positions in the block $[j..e_j - 3\tau + 2) \subseteq \mathsf{R}^-_H$. In any such block there is at most one element with a given value of L-exp and L-head. Thus, to compute $\text{SA}[i]$, we first in $\mathcal{O}(1)$ time compute

$e_j = \mathsf{select}_{B,1}(\mathsf{rank}_{B,1}(j)+1) + 2\tau - 1$. We then in $\mathcal{O}(1)$ time compute $s = \text{L-head}(j)$ using the lookup table L (which is a part of the data structure from Proposition 4.4). This lets us determine $e_j^{\text{full}} = e_j - ((e_j - j - s) \bmod |H|)$. Consequently, we have $\text{SA}[i] = e_j^{\text{full}} - \text{L-head}(\text{SA}[i]) - \text{L-exp}(\text{SA}[i])|H|$. In total, the query takes $\mathcal{O}(\log\log n)$ time.

The data structure is constructed as follows. We first compute the sequence $(s_t)_{t\in[1..n']}$ containing the elements of S in sorted order (see Section 3.2). It can be obtained in $\mathcal{O}(1+|\mathsf{S}|) = \mathcal{O}(n/\log_\sigma n)$ time using select queries on the bitvector B. We then combine Propositions 3.2, 3.7 and 3.12 to construct the data structure from Proposition 3.12 in $\mathcal{O}(n/\log_\sigma n)$ time. This gives us bitvector $B_{\mathsf{R}'}$, the arrays C and M, and the sequence $(\ell_t)_{t\in[1..|\mathsf{R}'^-|]}$. We discard all remaining components of the data structure from Proposition 3.12. Given M, we can compute M^{-1} in $\mathcal{O}(|\mathsf{R}'^-|) = \mathcal{O}(n/\log_\sigma n)$ time, since these two arrays are inverses of each other. We use the sequence (ℓ_t) to construct the array $A[1..|\mathsf{R}'^-|]$. Finally, we augment A with the data structure for range counting/selection queries. Using Proposition 3.11, by $|\mathsf{R}'^-| = \mathcal{O}(n/\log_\sigma n)$ and $\sum_{i=1}^{|\mathsf{R}'^-|} A[i] = \mathcal{O}(n)$, this takes $\mathcal{O}(n/\log_\sigma n)$ time. □

Summary By combining all above results, we obtain the following data structure for compute $\text{SA}[i]$ for periodic positions.

Proposition 4.7. *In $\mathcal{O}(n/\log_\sigma n)$ time, we can augment $\text{C}_{\text{SA}}(T,\mathsf{S})$ so that, given $i \in [1..n]$ such that $\text{SA}[i] \in \mathsf{R}$, we can compute $\text{SA}[i]$ in $\mathcal{O}(\log\log n)$ time.*

Proof. The data structure is a composition of four data structures. First, we store the two data structures presented in Propositions 4.5 and 4.6. These structures are designed to compute $\text{SA}[i]$ for $i \in [1..n]$ such that $\text{SA}[i] \in \mathsf{R}^-$. To handle the case $\text{SA}[i] \in \mathsf{R}^+$ we store their symmetric versions adapted according to Lemma 3.5 (see also Proposition 3.13). All four data structures take $\mathcal{O}(n/\log_\sigma n)$ space. All structures are built on top of the structure from Proposition 4.4, but it suffices to only keep its single copy (in particular, we keep a single copy of $\text{C}_{\text{SA}}(T,\mathsf{S})$).

Using the above data structures, given $i \in [1..n]$, we compute $\text{SA}[i]$ as follows. First, using Proposition 4.5, in $\mathcal{O}(1)$ time we compute $\text{type}(\text{SA}[i])$. Depending on whether $\text{SA}[i] \in \mathsf{R}^-$ or $\text{SA}[i] \in \mathsf{R}^+$, we use either a combination of Propositions 4.5 and 4.6 or their symmetric counterparts, to first compute $\text{L-exp}(\text{SA}[i])$ and $\delta^s(\text{SA}[i])$ in $\mathcal{O}(1)$ time, and then $\text{SA}[i]$ in $\mathcal{O}(\log\log n)$ time.

The data structure is constructed as follows. First, given $\text{C}_{\text{SA}}(T,\mathsf{S})$ we construct the data structure from Proposition 4.4. This takes $\mathcal{O}(n/\log_\sigma n)$ time. We then augment it into the two data structures to compute $\text{SA}[i]$ for $i \in [1..n]$ satisfying $\text{SA}[i] \in \mathsf{R}^-$. Using Propositions 4.5 and 4.6, this takes $\mathcal{O}(n/\log_\sigma n)$ time. Finally, we analogously construct the two data structures to compute $\text{SA}[i]$ for $i \in [1..n]$ satisfying $\text{SA}[i] \in \mathsf{R}^+$. □

4.4 The Final Data Structure

Theorem 4.8. *Given a constant $\epsilon \in (0,1)$ and the packed representation of a text $T \in [0..\sigma)^n$ with $2 \le \sigma < n^{1/6}$, we can construct in $\mathcal{O}(n\log\sigma/\sqrt{\log n})$ time and $\mathcal{O}(n/\log_\sigma n)$ working space a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that, given any $i \in [1..n]$, returns the value of $\text{SA}[i]$ in $\mathcal{O}(\log^\epsilon n)$ time.*

Proof. The data structure is a composition of the data structures from Proposition 4.3 and Proposition 4.7. Both data structures take $\mathcal{O}(n/\log_\sigma n)$ space. Each of the two structures is built on top of the index core $\text{C}_{\text{SA}}(T,\mathsf{S})$, but it suffices to store a single copy of $\text{C}_{\text{SA}}(T,\mathsf{S})$.

Given $i \in [1..n]$, we use Lemma 4.1 to check in $\mathcal{O}(1)$ time if $\text{SA}[i] \in \mathsf{R}$. Depending on whether $\text{SA}[i] \in \mathsf{R}$ or not, we use Proposition 4.3 or Proposition 4.7 to compute $\text{SA}[i]$ in $\mathcal{O}(\log^\epsilon n)$ or $\mathcal{O}(\log\log n)$ time (respectively).

The data structure is constructed as follows. First, using Theorem 2.5, from a packed representation of T, we construct a τ-synchronizing set S of size $\mathcal{O}(n/\tau)$ in $\mathcal{O}(n/\tau) = \mathcal{O}(n/\log_\sigma n)$ time. The set S is returned as an array taking $\mathcal{O}(n/\log_\sigma n)$ space. Using this array and the packed representation of T, we then construct $\mathrm{C}_{\mathrm{SA}}(T, \mathsf{S})$ in $\mathcal{O}(n/\log_\sigma n)$ time using Proposition 4.2. Finally, using Propositions 4.3 and 4.7, we augment $\mathrm{C}_{\mathrm{SA}}(T, \mathsf{S})$ in $\mathcal{O}(n \log \sigma/\sqrt{\log n})$ and $\mathcal{O}(n/\log_\sigma n)$ time (respectively) and using $\mathcal{O}(n/\log_\sigma n)$ working space into structures to compute $\mathrm{SA}[i]$. □

5 Pattern Matching Queries

For strings P and T, we define $\mathrm{Occ}(P,T) = \{j \in [1\,..\,|T|-|P|+1] : T[j\,..\,j+|P|) = P\}$, $\mathrm{RangeBeg}(P,T) = |\{i \in [1\,..\,|T|] : T[i\,..\,|T|] \prec P\}|$, and $\mathrm{RangeEnd}(P,T) = \mathrm{RangeBeg}(P,T) + |\mathrm{Occ}(P,T)|$. If SA is the suffix array of T, it then holds $(\mathrm{RangeBeg}(P,T)\,..\,\mathrm{RangeEnd}(P,T)] = \{i \in [1\,..\,|T|] : P \text{ is a prefix of } T[\mathrm{SA}[i]\,..\,|T|]\}$. In this section we show how, given a constant $\epsilon \in (0,1)$ and the packed representation of a text $T \in [0\,..\,\sigma)^n$ with $2 \le \sigma < n^{1/6}$, to construct in $\mathcal{O}(n \log \sigma/\sqrt{\log n})$ time and $\mathcal{O}(n/\log_\sigma n)$ working space a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that, given the packed representation of a pattern $P \in [0\,..\,\sigma)^m$, returns $\mathrm{RangeBeg}(P,T)$ and $\mathrm{RangeEnd}(P,T)$ in $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$ time.

As in Section 3, we let $\tau = \mu \log_\sigma n$, where μ is any positive constant smaller than $\frac{1}{6}$ and let S be any τ-synchronizing set of T satisfying $n' = |\mathsf{S}| = \mathcal{O}(n/\log_\sigma n)$. Recall that the set of strings $\mathcal{D} \cup \mathcal{F} \subseteq [0\,..\,\sigma)^{\le 3\tau-1}$ is prefix-free. Thus, for any string P, there exists at most one string $X \in \mathcal{D} \cup \mathcal{F}$ that is a prefix of P.

Definition 5.1. A string $P \in [0\,..\,\sigma)^m$ satisfying $m \ge 3\tau - 1$ is said to have *nonperiodic prefix* (resp. *periodic prefix*) if there exists $X \in \mathcal{D}$ (resp. $X \in \mathcal{F}$) that is a prefix of P.

Organization The section is organized as follows. Our description is split into four parts. First (Section 5.1), we describe the data structures, called collectively the index "core", that enable efficiently handling short patterns ($m < 3\tau-1$), or (if $m \ge 3\tau-1$) check whether there exists a prefix $X \in \mathcal{D} \cup \mathcal{F}$ of P, and if so, return such X as well as the information whether $X \in \mathcal{D}$ or $X \in \mathcal{F}$. The data structure and query algorithm to compute $\mathrm{RangeBeg}(P,T)$ and $\mathrm{RangeEnd}(P,T)$ is different depending on whether $X \in \mathcal{D}$ or $X \in \mathcal{F}$, and the structures used for the two cases are described in the next two parts (Sections 5.2 and 5.3). All ingredients are finally put together in Section 5.4.

5.1 The Index Core

We use the following definitions. Let $P_{\mathcal{D}\cup\mathcal{F}}$ be a mapping from $[0\,..\,\sigma)^{3\tau-1}$ to $[0\,..\,\sigma)^{\le 3\tau-1} \times \{0,1\}$ such that for any $Y \in [0\,..\,\sigma)^{3\tau-1}$, if there exists $X \in \mathcal{D} \cup \mathcal{F}$ that is a prefix of Y, then $P_{\mathcal{D}\cup\mathcal{F}}$ maps Y to (X,d) and $d = 1$ holds if and only if $X \in \mathcal{D}$. If there is no such X, then $P_{\mathcal{D}\cup\mathcal{F}}$ maps Y to $(\varepsilon, 0)$.

The index core, denoted $\mathrm{C}_{\mathrm{count}}(T, \mathsf{S})$, consists of four components. First, we store the packed representation of T using $\mathcal{O}(n/\log_\sigma n)$ space. Second, we store the bitvector B (as defined in Section 3.1) augmented using Theorem 2.1 for rank and selection queries. The bitvector takes $\mathcal{O}(n/\log n)$ space. Third, we store the lookup table $\mathrm{ISA}_{3\tau-1}$ (as defined in Section 3.1). When accessing $\mathrm{ISA}_{3\tau-1}$, the strings $Y \in [0\,..\,\sigma)^{\le 3\tau-1}$ are converted to $\mathrm{int}(Y)$. By $\mathrm{int}(Y) \in [0\,..\,\sigma^{6\tau})$, we can store the table using $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ space. Fourth and final, we store the lookup table $P_{\mathcal{D}\cup\mathcal{F}}$. Similarly as for $\mathrm{ISA}_{3\tau-1}$, we can store it using $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ space.

Lemma 5.2. *Given* $\mathrm{C}_{\mathrm{count}}(T,\mathsf{S})$ *and a packed representation of* $P \in [0\,..\,\sigma)^m$*, we can in* $\mathcal{O}(1)$ *time compute the pair* $(\mathrm{RangeBeg}(P,T),\mathrm{RangeEnd}(P,T))$ *if* $m < 3\tau-1$*, or* $m \geq 3\tau-1$ *and no element of* $\mathcal{D}\cup\mathcal{F}$ *is a prefix of* P*. Otherwise, the algorithm returns the prefix* $X \in \mathcal{D}\cup\mathcal{F}$ *of* P*, along with information whether* $X \in \mathcal{D}$*.*

Proof. If $m < 3\tau-1$, then in $\mathcal{O}(1)$ time we compute the value $p = \mathrm{int}(P)$ and return the pair $(\mathrm{RangeBeg}(P,T),\mathrm{RangeEnd}(P,T)) = \mathrm{ISA}_{3\tau-1}[p]$ as output. If $m \geq 3\tau-1$, we first compute in $\mathcal{O}(1)$ time the value $p' = \mathrm{int}(P[1\,..\,3\tau-1])$, and then obtain $(X,d) = P_{\mathcal{D}\cup\mathcal{F}}[p']$. If $X \neq \varepsilon$ then we return X and d indicates whether $D \in \mathcal{D}$. Otherwise (i.e., if $X = \varepsilon$) we return $(\mathrm{RangeBeg}(P,T),\mathrm{RangeEnd}(P,T)) = \mathrm{ISA}_{3\tau-1}[p']$. □

Proposition 5.3. *Given the packed representation of* $T \in [0\,..\,\sigma)^n$ *and the array containing elements of* S*, we can construct* $\mathrm{C}_{\mathrm{count}}(T,\mathsf{S})$ *in* $\mathcal{O}(n/\log_\sigma n)$ *time.*

Proof. Given S, we easily initialize the bitvector B in $\mathcal{O}(|\mathsf{S}| + n/\log n) = \mathcal{O}(n/\log_\sigma n)$ time. Augmenting B using Theorem 2.1 takes $\mathcal{O}(n/\log n)$ time.

The lookup table $\mathrm{ISA}_{3\tau-1}$ is initialized in $\mathcal{O}(n/\log_\sigma n)$ time as described in Proposition 3.2. To compute $P_{\mathcal{D}\cup\mathcal{F}}$, we observe that in the proof of Proposition 4.2, we have shown how in $\mathcal{O}(n/\log_\sigma n)$ time to enumerate all $X \in \mathcal{D}\cup\mathcal{F}$, along with the information whether $X \in \mathcal{D}$ for every X. We first initialize the array $P_{\mathcal{D}\cup\mathcal{F}}$ to $(\varepsilon,0)$ for every $Y \in [0\,..\,\sigma)^{3\tau-1}$. We then perform the enumeration of $\mathcal{D}\cup\mathcal{F}$ as in Proposition 4.2. Whenever $X \in \mathcal{D}$, we generate all $X' \in X\cdot[0\,..\,\sigma)^{3\tau-1-|X|}$ and for each X' we set $P_{\mathcal{D}\cup\mathcal{F}}[\mathrm{int}(X')] = (\mathrm{int}(X),1)$. If $X \in \mathcal{F}$, we set $P_{\mathcal{D}\cup\mathcal{F}}[\mathrm{int}(X)] = (\mathrm{int}(X),0)$. Since the set $\mathcal{D}\cup\mathcal{F}$ is prefix-free, we never set the same string twice, and hence the whole procedure takes $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ time. □

Before we present how to compute the pair $(\mathrm{RangeBeg}(P,T),\mathrm{RangeEnd}(P,T))$, we first prove the following auxiliary result.

Proposition 5.4. *Given a packed representation of* $T \in [0\,..\,\sigma)^n$*, and an array* $A[1\,..\,q]$ *of* $q = \mathcal{O}(n/\log_\sigma n)$ *positions in* T *such that for any* $1 \leq i < j \leq q$*, it holds* $T[A[i]\,..\,n] \prec T[A[j]\,..\,n]$*, we can in* $\mathcal{O}(n(\log\log n)^2/\log_\sigma n)$ *time and* $\mathcal{O}(n/\log_\sigma n)$ *working space construct a data structure of size* $\mathcal{O}(n/\log_\sigma n)$ *that, given the packed representation of any* $P \in [0\,..\,\sigma)^m$*, returns in* $\mathcal{O}(m/\log_\sigma n + \log\log n)$ *time a pair of integers* $(b_{\mathrm{pre}},e_{\mathrm{pre}})$ *satisfying* $b_{\mathrm{pre}} = |\{i \in [1\,..\,q] : T[A[i]\,..\,n] \prec P\}|$ *and* $(b_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}] = \{i \in [1\,..\,q] : P$ *is a prefix of* $T[A[i]\,..\,n]\}$*.*

Proof. The basic idea is to construct the compacted trie of strings in $\{T[A[i]\,..\,n]\}_{i\in[1\,..\,q]}$ converted into strings over the alphabet of metasymbols (of $\Theta(\log_\sigma n)$ original symbols each). Each node v is augmented with two values $\mathrm{lrank}(v)$ and $\mathrm{rrank}(v)$ containing respectively the exclusive rank of the leftmost and the rank of the rightmost leaf in the subtree rooted in v. During query, we find the longest prefix of P present in the trie. We then use the predecessor data structure to compute the rank of P and the range of children corresponding to strings prefixed with P. Finally, using lrank and rrank values we deduce the output pair $(b_{\mathrm{pre}},e_{\mathrm{pre}})$. The main challenge lies in ensuring that using a trie of metasymbols is equivalent to using the trie of strings over the original alphabet.

We use the following definitions. First, we introduce the mapping of strings over $[0\,..\,\sigma)$ into strings over metasymbols. Let $\kappa = 3\tau-1$. For any string $S \in [0\,..\,\sigma)^*$ of length $\ell \geq 0$, we define $\mathrm{mstr}(S)$ as a string of length $\ell' = \lceil\frac{\ell+1}{\kappa}\rceil > 0$ over alphabet $[0\,..\,\sigma^{6\tau})$ such that for any $i \in [1\,..\,\ell']$ it holds $\mathrm{mstr}(S)[i] = \mathrm{int}(S((i-1)\cdot\kappa\,..\,\min(\ell,i\cdot\kappa))]$, where $\mathrm{int}(\cdot)$ is defined as in Section 3.1. Note that if ℓ is a multiple of κ, then the last symbol of $\mathrm{mstr}(S)$ is $\mathrm{int}(\varepsilon) = 0$. We then define the complementary mapping mstr'. For any $X \in [0\,..\,\sigma)^{\leq 3\tau}$, we first let $\mathrm{int}'(X)$ denote an integer constructed by appending $6\tau-2|X|$ cs (where $c = \sigma-1$) and $|X|$ zeros to X, and then interpreting

the resulting string as a base-σ representation of a number in $[0\,..\,\sigma^{6\tau})$. For $S \in [0\,..\,\sigma)^*$ of length $\ell \geq 0$, we define $\mathrm{mstr}'(S)$ as a string of length $\ell' = \lceil \frac{\ell+1}{\kappa} \rceil > 0$ over alphabet $[0\,..\,\sigma^{6\tau})$ such that $\mathrm{mstr}'(S)[1\,..\,\ell') = \mathrm{mstr}(S)[1\,..\,\ell')$ and $\mathrm{mstr}'(S)[\ell'] = \mathrm{int}'(S((\ell'-1)\cdot\kappa\,..\,\min(\ell, \ell'\cdot\kappa))]$. Observe, that

- For any set of strings $\mathcal{S} \subseteq [0\,..\,\sigma)^*$, the set $\{\mathrm{mstr}(X) : X \in \mathcal{S}\}$ is prefix-free.
- For any strings $X, Y \in [0\,..\,\sigma)^*$, $X \prec Y$ holds if and only if $\mathrm{mstr}(X) \prec \mathrm{mstr}(Y)$.
- A string $P \in [0\,..\,\sigma)^*$ is a prefix of $X \in [0\,..\,\sigma)^*$ if and only if $\mathrm{mstr}(P) \preceq \mathrm{mstr}(X) \prec \mathrm{mstr}'(P)$.

Let $\mathcal{C}$ denote the compacted trie (i.e., a trie, in which unary paths have been compacted into single edges labeled by the substrings of the input strings) of the set $\{\mathrm{mstr}(T[A[i]\,..\,n])\}_{i\in[1\,..\,q]}$. For any node v of $\mathcal{C}$, we define $\mathrm{lrank}(v) = |\{i \in [1\,..\,q] : \mathrm{mstr}(T[A[i]\,..\,n]) \prec S_v^{\min}\}|$ and $\mathrm{rrank}(v) = |\{i \in [1\,..\,q] : \mathrm{mstr}(T[A[i]\,..\,n]) \preceq S_v^{\max}\}|$, where $S_v^{\min}$ (resp. $S_v^{\max}$) denotes the string (over alphabet $[0\,..\,\sigma^{6\tau})$) corresponding to the leftmost (resp. rightmost) leaf in the subtree rooted at v.

The data structure consists of two components. First, we store the packed representation of T using $\mathcal{O}(n/\log_\sigma n)$ space. Second, we store the trie $\mathcal{C}$ (with the label of each edge encoded as a pointer to the substring of T). Each node v of $\mathcal{C}$ stores the precomputed values $\mathrm{lrank}(v)$ and $\mathrm{rrank}(v)$. In addition, each node v stores a deterministic dictionary that, given a symbol $c \in [0\,..\,\sigma^{6\tau})$, returns the pointer to the child v' of v for which the label of the edge from v to v' starts with c, or tells that no such child exists. We use the dictionary from [61, Theorem 3] and hence achieve linear space and $\mathcal{O}(1)$ query time. Each node v also stores a predecessor data structure that, given any $c \in [0\,..\,\sigma^{6\tau})$, computes the number of children v' of v for which the label of the edge from v to v' starts with a symbol c' satisfying $c' < c$. Using the data structure from [24, Proposition 7], we achieve linear space and $\mathcal{O}(\log\log n)$ query time.

Using T and $\mathcal{C}$, given the packed representation of $P \in [0\,..\,\sigma)^m$, we compute the output pair $(b_{\mathrm{pre}}, e_{\mathrm{pre}})$ as follows. First, note that, given the packed representation of P and T, we can in $\mathcal{O}(1)$ time access any symbol of $\mathrm{mstr}(P)$, $\mathrm{mstr}'(P)$, and $\mathrm{mstr}(T[i\,..\,n])$ for any $i \in [1\,..\,n]$. Denote $\widehat{m} = |\mathrm{mstr}(P)|$. We start by computing the length ℓ of the longest prefix of $\mathrm{mstr}(P)[1\,..\,\widehat{m})$ that is represented in $\mathcal{C}$. This takes $\mathcal{O}(|\mathrm{mstr}(P)|) = \mathcal{O}(1 + m/\log_\sigma n)$ time. We then consider two cases:

- If $\mathrm{mstr}(P)[1\,..\,\ell]$ ends in the middle of an edge in $\mathcal{C}$, then let v be the deeper endpoint of that edge, and c be the next unmatched symbol on the edge. If $\mathrm{mstr}'(P)[\ell+1] \leq c$, then we return $(b_{\mathrm{pre}}, e_{\mathrm{pre}}) = (\mathrm{lrank}(v), \mathrm{lrank}(v))$. If $\mathrm{mstr}(P)[\ell+1] \leq c < \mathrm{mstr}'(P)[\ell+1]$, then we return $(b_{\mathrm{pre}}, e_{\mathrm{pre}}) = (\mathrm{lrank}(v), \mathrm{rrank}(v))$. Finally, if $c < \mathrm{mstr}(P)[\ell+1]$, then we return $(b_{\mathrm{pre}}, e_{\mathrm{pre}}) = (\mathrm{rrank}(v), \mathrm{rrank}(v))$.
- If $\mathrm{mstr}(P)[1\,..\,\ell]$ ends in the explicit node v of $\mathcal{C}$, let $v_1, \ldots, v_k$ denote the sequence of children of v ordered by the first symbol of the edge from v, and let c_i denote the first symbol of edge connecting v and v_i. To streamline the formulae, we define $\mathrm{rrank}(v_0) = \mathrm{lrank}(v_1)$ and $\mathrm{lrank}(v_{k+1}) = \mathrm{rrank}(v_k)$. Using the predecessor data structure of v, we compute in $\mathcal{O}(\log\log n)$ time the values $x = |\{i \in [1\,..\,k] : c_i < \mathrm{mstr}(P)[\ell+1]\}|$ and $y = |\{i \in [1\,..\,k] : c_i < \mathrm{mstr}'(P)[\ell+1]\}|$. We then return $(b_{\mathrm{pre}}, e_{\mathrm{pre}}) = (\mathrm{lrank}(v_{x+1}), \mathrm{rrank}(v_y))$.

Altogether, the query algorithms runs in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time.

The data structure is constructed as follows. We start by building a data structure that supports LCE queries for suffixes of T. Using [39, Theorem 5.4], the construction takes $\mathcal{O}(n/\log_\sigma n)$ time, and the resulting data structure answers queries in $\mathcal{O}(1)$ time. Denote the length of the longest common prefix between suffixes $T[i\,..\,n]$ and $T[j\,..\,n]$ as $\mathrm{LCE}(i,j)$. Observe that for any $i, j \in [1\,..\,n]$ such that $i \neq j$, the longest common prefix of $\mathrm{mstr}(T[i\,..\,n])$ and $\mathrm{mstr}(T[j\,..\,n])$ has length $\lfloor \frac{\mathrm{LCE}(i,j)}{\kappa} \rfloor$, which can be computed in $\mathcal{O}(1)$ time. We construct $\mathcal{C}$ by inserting elements of

$\{\mathrm{mstr}(T[A[i]\,..\,n])\}_{i\in[1\,..\,q]}$ in the order given by A. We maintain a stack containing the internal nodes on the rightmost path, with the deepest node on top. When inserting each string, we first determine the depth at which that string branches from the rightmost path using LCE queries on T. We then update the rightmost path of the trie. Adding each string first removes some elements from the stack, and then adds at most one new element. Since the total number of elements pushed on stack is $\mathcal{O}(q)$, the construction of $\mathcal{C}$ takes $\mathcal{O}(q)$ time. With the single traversal of $\mathcal{C}$, we then precompute in $\mathcal{O}(q)$ time the values $\mathrm{lrank}(v)$ and $\mathrm{rrank}(v)$ for every node v. Finally, we augment every node with the dictionary and the predecessor data structure. Using [61, Theorem 3], the linear-space dictionary for every node v with $k = \mathcal{O}(\sigma^{6\tau})$ children can be deterministically constructed in $\mathcal{O}(k(\log\log k)^2)$ time. Over all nodes, this takes $\mathcal{O}(q(\log\log q)^2)$ time. Note, that by $\mu < \frac{1}{6}$, it holds $\mathcal{O}(k(\log\log k)^2) = \mathcal{O}(n/\log_\sigma n)$ for every node v. Thus, the working space never exceeds $\mathcal{O}(n/\log_\sigma n)$. To construct the predecessor data structures, note that the keys in every node are sorted. Thus, using [24, Proposition 7], over all nodes of $\mathcal{C}$, the construction takes $\mathcal{O}(q)$ time. After the initialization of the LCE data structure in $\mathcal{O}(n/\log_\sigma n)$ time, the construction of $\mathcal{C}$ takes $\mathcal{O}(q(\log\log q)^2)$ time. By $q = \mathcal{O}(n/\log_\sigma n)$, this is $\mathcal{O}(n(\log\log n)^2/\log_\sigma n)$. □

5.2 The Nonperiodic Prefix

The main idea of the data structure is as follows. Let $P = XY \in [0\,..\,\sigma)^m$, where $X \in \mathcal{D}$. By consistency of S, the offset of the first position from S in every suffix prefixed with P is $\delta = |X| - 2\tau$. Thus, the order of these suffixes is the same as the order of corresponding suffixes starting at positions of S. Hence, it suffices to sort only those suffixes, and then computing $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ can be achieved by first finding the range $(e_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}]$ of suffixes starting in S that are prefixed with $X(\delta\,..\,|X|]\cdot Y = P(\delta\,..\,m]$, and then counting the suffixes in the range that are preceded with $X[1\,..\,\delta]$ in the text. We formalize this as follows, using the sequence $W[1\,..\,n']$ defined in Section 3.2.

Lemma 5.5. *Let $P = XY \in [0\,..\,\sigma)^m$, where $X \in \mathcal{D}$. Denote $\delta = |X| - 2\tau$. Assume that $\mathrm{ISA}_{3\tau-1}[\mathrm{int}(X)] = (b,\cdot)$, and let $(b_{\mathrm{pre}}, e_{\mathrm{pre}})$ be a pair of integers such that $b_{\mathrm{pre}} = |\{i \in [1\,..\,n'] : T[s'_i\,..\,n] \prec P(\delta\,..\,m]\}|$ and $(b_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}] = \{i \in [1\,..\,n'] : P(\delta\,..\,m]$ is a prefix of $T[s'_i\,..\,n]\}$. Then, it holds $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T)) = (b + \mathsf{rank}_{W,\overline{X}}(b_{\mathrm{pre}}), b + \mathsf{rank}_{W,\overline{X}}(e_{\mathrm{pre}}))$.*

Proof. By the consistency of S, there exists a bijection (given by the mapping $j \mapsto j+\delta$) between $\mathrm{Occ}(P,T)$, and positions $s \in \mathsf{S}$ satisfying $T^\infty[s-\delta\,..\,s-\delta+m) = P$, or equivalently, $s \in \mathsf{S}$ satisfying: (1) $T^\infty[s\,..\,s-\delta+m) = P(\delta\,..\,m]$, and (2) $T^\infty[s-\delta\,..\,s+2\tau) = X$. By definition of the sequence $(s'_t)_{t\in[1\,..\,n']}$ (see Section 3.2) and the pair $(b_{\mathrm{pre}}, e_{\mathrm{pre}})$, all positions $s \in \mathsf{S}$ satisfying the first condition are in $\{s'_{b_{\mathrm{pre}}+1}, \ldots, s'_{e_{\mathrm{pre}}}\}$. Since $W[j] = \overline{X_j}$, where $X_j = T^\infty[s'_j - \tau\,..\,s'_j + 2\tau)$, for any $j \in (b_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}]$, the position s'_j additionally satisfies the second condition if and only if $\overline{X}$ is a prefix of $W[j]$. Thus, $|\mathrm{Occ}(P,T)| = \mathsf{rank}_{W,\overline{X}}(e_{\mathrm{pre}}) - \mathsf{rank}_{W,\overline{X}}(b_{\mathrm{pre}})$ follows from the definition of prefix rank queries. Adding all suffixes of T having $X' \in \mathcal{D}\cup\mathcal{F}$ satisfying $X' \prec X$ as a prefix furthermore yields $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T)) = (b + \mathsf{rank}_{W,\overline{X}}(b_{\mathrm{pre}}), b + \mathsf{rank}_{W,\overline{X}}(e_{\mathrm{pre}}))$, where $\mathrm{ISA}_{3\tau-1}[\mathrm{int}(X)] = (b,\cdot)$. Note, that since by Proposition 5.4 the range $(b_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}]$ is well defined even if $e_{\mathrm{pre}} - b_{\mathrm{pre}} = 0$, the range $(\mathrm{RangeBeg}(P,T)\,..\,\mathrm{RangeEnd}(P,T)]$ is computed correctly even if $|\mathrm{Occ}(P,T)| = 0$. □

Proposition 5.6. *Given a constant $\epsilon \in (0,1)$ and $\mathrm{C}_{\mathrm{count}}(T,\mathsf{S})$, we can in $\mathcal{O}(n\log\sigma/\sqrt{\log n})$ time and in $\mathcal{O}(n/\log_\sigma n)$ working space augment it into a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that, given the packed representation of $P \in [0\,..\,\sigma)^m$ having a nonperiodic prefix, returns the pair $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ in $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$ time.*

Proof. We use the following definitions. Let D denote the data structure from Proposition 5.4 for the array $A[1\,..\,n']$ defined by $A[i] = s'_i$ (where $(s'_t)_{t\in[1\,..\,n']}$ is a sequence as defined in Section 3.2).

The data structure, in addition to $\mathrm{C}_{\mathrm{count}}(T, \mathsf{S})$, consists of three components. First, we store a lookup table of size $\mathcal{O}(n^\delta)$, for some $\delta < 1$, that given the packed representation of any string $X \in [0\,..\,\sigma)^*$ of length $|X| = \mathcal{O}(\log_\sigma n)$, allows us to compute the packed representation of $\overline{X}$ in $\mathcal{O}(1)$ time (see also Proposition 3.4). Second, we store the sequence W augmented with the component of Theorem 2.2, which needs $\mathcal{O}(n/\log_\sigma n)$ space. Second, we store the data structure D. By $n' = \mathcal{O}(n/\log_\sigma n)$ and Proposition 5.4, D needs $\mathcal{O}(n/\log_\sigma n)$ space.

Using $\mathrm{C}_{\mathrm{count}}(T, \mathsf{S})$ and the above two components, given a packed $P \in [0\,..\,\sigma)^m$, we compute $(\mathrm{RangeBeg}(P, T), \mathrm{RangeEnd}(P, T))$ as follows. First, using Lemma 5.2, in $\mathcal{O}(1)$ time we compute the prefix $X \in \mathcal{D}$ of P. Let $\delta = |X| - 2\tau$. Using Proposition 5.4, we then compute in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time the pair of integers $(b_{\mathrm{pre}}, e_{\mathrm{pre}})$ for the pattern $P' := P(\delta\,..\,m]$, as defined in Lemma 5.5. Next, letting $\mathrm{ISA}_{3\tau-1}[\mathrm{int}(X)] = (b, \cdot)$, we compute $(\mathrm{RangeBeg}(P, T), \mathrm{RangeEnd}(P, T)) = (b + \mathsf{rank}_{W,\overline{X}}(b_{\mathrm{pre}}), b + \mathsf{rank}_{W,\overline{X}}(e_{\mathrm{pre}}))$, with the two prefix rank queries implemented using Theorem 2.2, in $\mathcal{O}(\log^\epsilon n)$ time each. Altogether, the query time is $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$.

The data structure is constructed as follows. First, in $\mathcal{O}(n^\delta)$ time we initialize the lookup table used to reverse short strings (see also Proposition 3.4). Next, we compute the sequence $(s_t)_{t\in[1\,..\,n']}$ containing the elements of S in sorted order (see Section 3.2). It can be obtained in $\mathcal{O}(1 + |\mathsf{S}|) = \mathcal{O}(n/\log_\sigma n)$ time using select queries on the bitvector B (which is part of $\mathrm{C}_{\mathrm{count}}(T, \mathsf{S})$). We then combine Propositions 3.2 and 3.4 (recall that the packed representation of T is a component of $\mathrm{C}_{\mathrm{count}}(T, \mathsf{S})$) to construct the data structure from Proposition 3.4 in $\mathcal{O}(n\log\sigma/\sqrt{\log n})$ time and using $\mathcal{O}(n/\log_\sigma n)$ working space. This gives us the sequence W preprocessed using Theorem 2.2. During the construction, we also compute (and keep) the sequence $(s'_t)_{t\in[1\,..\,n']}$. We discard all remaining components of the data structure from Proposition 3.4. We then initialize $A[i] = s'_i$ for $i \in [1\,..\,n']$ and use Proposition 5.4 to construct D in $\mathcal{O}(n(\log\log n)^2/\log_\sigma n)$ time and using $\mathcal{O}(n/\log_\sigma n)$ working space. We then discard the sequence A. The overall runtime is $\mathcal{O}(n\log\sigma/\sqrt{\log n})$. The working space never exceed $\mathcal{O}(n/\log_\sigma n)$ words. □

5.3 The Periodic Prefix

Preliminaries Consider a string $P \in [0\,..\,\sigma)^m$ having a periodic prefix, i.e., satisfying $m \geq 3\tau-1$ and $P[1\,..\,3\tau-1] \in \mathcal{F}$ (see Definition 5.1). Let $H = \text{L-root}(P[1\,..\,3\tau-1])$. We define $e(P) = \max\{i \in [1\,..\,m] : P[1\,..\,i] \text{ has period } |H|\} + 1$. Thus, we can write $P[1\,..\,e(P)) = H'H^kH''$, where H' (resp. H'') is a proper suffix (resp. prefix) of H. By $e(P) \geq 3\tau$ and $|H| \leq \tau$, such decomposition is unique (see also Section 3.3). The integers $\text{L-head}(P) = |H'|$, $\text{L-exp}(P) = k$, and $\text{L-tail}(P) = |H''|$ are called the *L-head*, the *L-exponent*, and the *L-tail* of P, respectively. We denote $e^{\mathrm{full}}(P) = e(P) - \text{L-tail}(P)$. We define $\mathrm{type}(P) = +1$ if $e(P) \leq m$ and $P[e(P)] \succ P[e(P) - p]$ (where $p = |\text{L-root}(P[1\,..\,3\tau-1])|$), and $\mathrm{type}(P) = -1$ otherwise.

Lemma 5.7. *Let $P \in [0\,..\,\sigma)^m$ be a string with a periodic prefix. Denote $s = \text{L-head}(P)$ and $H = \text{L-root}(P)$. For any $j \in [1\,..\,n]$, $\mathrm{lcp}(T[j\,..\,n], P) \geq 3\tau - 1$ holds if and only if $j \in \mathsf{R}_{s,H}$.*

Proof. Let $j \in [1\,..\,n]$ be such that $\mathrm{lcp}(T[j\,..\,n], P) \geq 3\tau - 1$. Then, by definition, $\text{L-root}(j) = \text{L-root}(T[j\,..\,j + 3\tau - 1)) = \text{L-root}(P[1\,..\,3\tau - 1]) = \text{L-root}(P)$. To show that $\text{L-head}(j) = s$, note that by $|H| \leq \tau$, the string $H'H^2$ (where H' is a length-s suffix of H) is a prefix of $P[1\,..\,3\tau - 1] = T[j\,..\,j + 3\tau - 1)$. On the other hand, $\text{L-head}(j) = s'$ implies that $\widehat{H}'H^2$ (where $\widehat{H}'$ is a length-s' suffix of H) is a prefix of $T[j\,..\,j + 3\tau - 1)$. Thus, by the synchronization property of primitive strings applied to the two copies of H, we have $s' = s$, and hence, $j \in \mathsf{R}_{s,H}$.

For the converse implication, assume $j \in \mathsf{R}_{s,H}$. This implies that both $P[1\,..\,e(P))$ and $T[j\,..\,e_j)$ are prefixes of $H'H^\infty$ (where H' is as above). Thus, by $e_j - j$, $e(P) - 1 \geq 3\tau - 1$, we obtain $\mathrm{lcp}(T[j\,..\,n], P) \geq 3\tau - 1$. □

Lemma 5.8. *Let $P \in [0\,..\,\sigma)^m$ be a string with a periodic prefix. Let $j \in \mathsf{R}_{s,H}$, where $s =$ L-head(P) and $H =$ L-root(P). Then:*

1. *If* $\mathrm{type}(j) \neq \mathrm{type}(P)$, *then* $T[j\,..\,n] \neq P$, *and* $P \prec T[j\,..\,n]$ *iff* $\mathrm{type}(P) < \mathrm{type}(j)$.
2. *If* $\mathrm{type}(j) = \mathrm{type}(P) = -1$ *and* $e_j - j \neq e(P) - 1$, *then* $T[j\,..\,n] \prec P$ *iff* $e_j - j < e(P) - 1$.
3. *If* $\mathrm{type}(j) = \mathrm{type}(P) = +1$ *and* $e_j - j \neq e(P) - 1$, *then* $T[j\,..\,n] \succ P$ *iff* $e_j - j < e(P) - 1$.

Proof. 1. Let $S = H'H^\infty$, where H' is a length-s suffix of H. In the proof of Lemma 3.5, it is shown that $\mathrm{type}(j) = -1$ implies $T[j\,..\,n] \prec S$, and $\mathrm{type}(j) = +1$ implies $S \prec T[j\,..\,n]$. We prove an analogous fact for P. We first note that $\mathrm{type}(P) = -1$ implies that either $e(P) = m + 1$, or $e(P) \leq m$ and $P[e(P)] \prec P[e(P) - |H|]$. In the first case, $P[1\,..\,e(P)) = P$ is a proper prefix of S and hence $P \prec S$. In the second case, letting $\ell = e(P) - 1$, we have $P[1\,..\,\ell] = S[1\,..\,\ell]$ and $P[1+\ell] \prec P[1+\ell-|H|] = S[1+\ell-|H|] = S[1+\ell]$. Consequently, $P \prec S$. If $\mathrm{type}(P) = +1$ holds, then $e(P) \leq m$. Thus, letting $\ell' = e(P) - 1$, we have $S[1\,..\,\ell'] = P[1\,..\,\ell']$ and $S[1+\ell'] = S[1+\ell'-|H|] = P[1+\ell'-|H|] \prec P[1+\ell']$. Hence, we obtain $S \prec P$. Combining the above, we either have $\mathrm{type}(j) = -1$ and $\mathrm{type}(P) = +1$ (in which case $T[j\,..\,n] \prec S \prec P$), or $\mathrm{type}(j) = +1$ and $\mathrm{type}(P) = -1$ (in which case $P \prec S \prec T[j\,..\,n]$), establishing the claim.

2. Assume $e_j - j < e(P) - 1$. If $e_j = n + 1$, then by $e_j - j < e(P) - 1$, $T[j\,..\,e_j) = T[j\,..\,n]$ is a proper prefix of $P[1\,..\,e(P))$, and hence $T[j\,..\,n] \prec P[1\,..\,e(P)) \preceq P$. If $e_j \leq n$, then letting $\ell = e_j - j$, we have $T[j\,..\,j+\ell) = P[1\,..\,\ell]$ and by $e_j - j < e(P) - 1$, $T[j+\ell] \prec T[j+\ell-|H|] = P[1+\ell-|H|] = P[1+\ell]$. Consequently, $T[j\,..\,n] \prec P$. Assume now $e(P) - 1 < e_j - j$. If $e(P) = m+1$, then $P[1\,..\,e(P)) = P$ is proper prefix of $T[j\,..\,e_j)$, and hence $P \prec T[j\,..\,e_j) \preceq T[j\,..\,n]$. If $e(P) \leq m$, then letting $\ell = e(P) - 1$, we have $P[1\,..\,\ell] = T[j\,..\,j+\ell)$ and by $e(P)-1 < e_j-j$, $P[1+\ell] \prec P[1+\ell-|H|] = T[j+\ell-|H|] = T[j+\ell]$. Thus, $P \prec T[j\,..\,n]$.

3. Assume $e_j - j < e(P) - 1$. By $\mathrm{type}(j) = +1$, we have $e_j \leq n$. Thus, letting $\ell = e_j - j$, by $e_j - j < e(P) - 1$, we have $P[1\,..\,\ell] = T[j\,..\,j+\ell)$ and $P[1+\ell] = P[1+\ell-|H|] = T[j+\ell-|H|] \prec T[j+\ell]$. Consequently, $P \prec T[j\,..\,n]$. Assume now $e_j - j > e(P) - 1$. By $\mathrm{type}(P) = +1$, we have $e(P) \leq m$. Thus, letting $\ell' = e(P) - 1$, by $e_j - j > e(P) - 1$, we have $T[j\,..\,j+\ell') = P[1\,..\,\ell']$ and $T[j+\ell'] = T[j+\ell'-|H|] = P[1+\ell'-|H|] \prec P[1+\ell']$. Consequently, $T[j\,..\,n] \prec P$. □

Proposition 5.9. *In $\mathcal{O}(n/\log_\sigma n)$ time, we can augment* $\mathrm{C}_{\mathrm{count}}(T, \mathsf{S})$ *into a data structure that, given the packed representation of $P \in [0\,..\,\sigma)^m$ with a periodic prefix, returns* L-root$(P[1\,..\,3\tau{-}1])$, L-head(P), L-exp(P), L-tail(P), *and* $\mathrm{type}(P)$ *in $\mathcal{O}(1 + m/\log_\sigma n)$ time.*

Proof. In addition to $\mathrm{C}_{\mathrm{count}}(T, \mathsf{S})$, the data structure contains the lookup table L (as defined in Proposition 3.7) in plain form, using $\mathcal{O}(\sigma^{3\tau-1}) = \mathcal{O}(n/\log_\sigma n)$ space.

Using $\mathrm{C}_{\mathrm{count}}(T, \mathsf{S})$ and L, we implement the queries as follows. We first compute $x \in [0\,..\,\sigma^{6\tau})$ such that $x = \mathrm{int}(P[1\,..\,3\tau{-}1])$. Given the packed encoding of P, such x is obtained in $\mathcal{O}(1)$ time. We then look up $(s, p) = L[x]$, and in $\mathcal{O}(1)$ time obtain L-root$(P[1\,..\,3\tau{-}1]) = P[1{+}s\,..\,1{+}s{+}p)$ and L-head$(P) = s$. Next, we compute L-exp(P) and L-tail(P). For this, we first determine the length ℓ of the longest common prefix of P and $P(p\,..\,m]$. Using the packed representation of P, we can do this in $\mathcal{O}(1 + m/\log_\sigma n)$ time (see, e.g., [39, Proposition 2.3]). Consequently, we obtain $e(P) = 1 + p + \ell$, L-exp$(P) = \lfloor \frac{e(P)-1-s}{p} \rfloor$, and L-tail$(P) = (e(P) - 1 - s) \bmod p$. Finally, to test if $\mathrm{type}(P) = +1$, we check whether $e(P) \leq m$, and if so, whether $P[e(P)] \succ P[e(P) - p]$.

The data structure is constructed as follows. We first compute the sequence $(s_t)_{t\in[1\,..\,n']}$ containing the elements of S in sorted order (see Section 3.2). It can be obtained in $\mathcal{O}(1 + |\mathsf{S}|) = \mathcal{O}(n/\log_\sigma n)$ time using select queries on the bitvector B (which is part of $\mathrm{C}_{\mathrm{count}}(T, \mathsf{S})$). We

then combine Propositions 3.2 and 3.7 (recall that the packed representation of T is a component of $\mathrm{C}_{\mathrm{count}}(T, \mathsf{S})$) to construct the data structure from Proposition 3.7 in $\mathcal{O}(n/\log_\sigma n)$ time. This gives us the table L. We then discard all remaining components of the data structure from Proposition 3.7. □

5.3.1 Computing $|\mathrm{Occ}(P,T)|$

Let $P \in [0\,..\,\sigma)^m$ be a string with a periodic prefix. Denote $s = \text{L-head}(P)$ and $H = \text{L-root}(P[1\,..\,3\tau{-}1])$. We define $\mathrm{Occ}^a(P,T) = \{j \in \mathsf{R}_{s,H} \cap \mathrm{Occ}(P,T) : \text{L-exp}(j) > \text{L-exp}(P)\}$ and $\mathrm{Occ}^s(P,T) = \{j \in \mathsf{R}_{s,H} \cap \mathrm{Occ}(P,T) : \text{L-exp}(j) = \text{L-exp}(P)\}$.

Lemma 5.10. *For any $P \in [0\,..\,\sigma)^m$ having a periodic prefix, the set $\mathrm{Occ}(P,T)$ is a disjoint union of $\mathrm{Occ}^a(P,T)$ and $\mathrm{Occ}^s(P,T)$.*

Proof. By definition, $\mathrm{Occ}^a(P,T) \cap \mathrm{Occ}^s(P,T) = \emptyset$ and $\mathrm{Occ}^a(P,T) \cup \mathrm{Occ}^s(P,T) \subseteq \mathrm{Occ}(P,T)$. Thus, it suffices to show $\mathrm{Occ}(P,T) \subseteq \mathrm{Occ}^a(P,T) \cup \mathrm{Occ}^s(P,T)$. Assume $j \in \mathrm{Occ}(P,T)$. This implies $T[j\,..\,j{+}3\tau{-}1) = P[1\,..\,3\tau{-}1]$ and hence $\text{L-root}(j) = \text{L-root}(P[1\,..\,3\tau{-}1])$. Let $s = \text{L-head}(P)$ and $H = \text{L-root}(P[1\,..\,3\tau{-}1])$. By definition of $\text{L-head}(P)$ and $|H| \le \tau$, the string $H'H^2$ (where H' is a length-s suffix of H) is a prefix of $P[1\,..\,3\tau{-}1]$. Consequently, $H'H^2$ is a prefix of $T[j\,..\,n]$. Thus, by the synchronization property of primitive strings, we have $\text{L-head}(j) = s$ (see Lemma 3.5 for a similar argument), and consequently, $j \in \mathsf{R}_{s,H}$. Let $k = \text{L-exp}(P)$. By definition of $\text{L-exp}(P)$, the string $H'H^k$ is a prefix of P. Thus, $H'H^k$ is also a prefix of $T[j\,..\,n]$. Since $\text{L-root}(j) = H$ and $\text{L-head}(j) = |H'|$, we thus have that $\text{L-exp}(j) \ge k$. Therefore, $j \in \mathrm{Occ}^a(P,T) \cup \mathrm{Occ}^s(P,T)$. □

By the above lemma, if $P \in [0\,..\,\sigma)^m$ has a periodic prefix, then $\mathrm{Occ}(P,T) \subseteq \mathsf{R}$. We focus on computing sizes of sets $\mathrm{Occ}^{a-}(P,T) := \mathrm{Occ}^a(P,T) \cap \mathsf{R}^-$ and $\mathrm{Occ}^{s-}(P,T) := \mathrm{Occ}^s(P,T) \cap \mathsf{R}^-$. The sizes of the sets $\mathrm{Occ}^{a+}(P,T) := \mathrm{Occ}^a(P,T) \cap \mathsf{R}^+$ and $\mathrm{Occ}^{s+}(P,T) := \mathrm{Occ}^s(P,T) \cap \mathsf{R}^+$ are computed analogously.

Computing $|\mathrm{Occ}^{a-}(P,T)|$ We now describe a data structure, that computes $|\mathrm{Occ}^{a-}(P,T)|$ for any pattern $P \in [0\,..\,\sigma)^m$ with a periodic prefix.

Lemma 5.11. *Let $P \in [0\,..\,\sigma)^m$ be a string with a periodic prefix. Denote $s = \text{L-head}(P)$ and $H = \text{L-root}(P[1\,..\,3\tau{-}1])$. If $e(P) \le m$, then it holds $\mathrm{Occ}^{a-}(P,T) = \emptyset$. Otherwise, it holds $\mathrm{Occ}^{a-}(P,T) = \{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) > \text{L-exp}(P)\}$.*

Proof. Let $e(P) \le m$. Denote $k = \text{L-exp}(P)$. Suppose $\mathrm{Occ}^{a-}(P,T) \neq \emptyset$, and let $j \in \mathrm{Occ}^{a-}(P,T)$. By definition, $s + k|H| \le e(P) - 1 < s + (k+1)|H|$ and $P[e(P)] \neq P[e(P) - |H|]$. On the other hand, by $j \in \mathsf{R}_{s,H}$ and $\text{L-exp}(j) > k$, the string $H'H^{k+1}$ (where H' is a length-s suffix of H) is a prefix of $T[j\,..\,n]$. Thus, we have $T[j{+}e(P){-}1] = T[j{+}e(P){-}1{-}|H|] = P[e(P){-}|H|] \neq P[e(P)]$. This implies $j \notin \mathrm{Occ}(P,T)$, contradicting $j \in \mathrm{Occ}^{a-}(P,T)$. Thus, $\mathrm{Occ}^{a-}(P,T) = \emptyset$.

Let $e(P) > m$. The inclusion $\mathrm{Occ}^{a-}(P,T) \subseteq \{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) > \text{L-exp}(P)\}$ follows by definition. To show the opposite inclusion, let $j \in \mathsf{R}^-_{s,H}$ be such that $\text{L-exp}(j) > \text{L-exp}(P)$. Denote $k = \text{L-exp}(P)$. Then, $P = H'H^kH''$, where $|H'| = s$, and H' (resp. H'') is a suffix (resp. prefix) of H. Thus, P is a prefix of $H'H^{k+1}$. The latter string, on the other hand, is by $\text{L-exp}(j) \ge k+1$ and $j \in \mathsf{R}_{s,H}$, a prefix of $T[j\,..\,n]$. Thus, $j \in \mathrm{Occ}(P,T)$. By $j \in \mathsf{R}^-_{s,H}$ and $\text{L-exp}(j) > \text{L-exp}(P)$, we therefore also have $j \in \mathrm{Occ}^{a-}(P,T)$. □

Proposition 5.12. *In $\mathcal{O}(n/\log_\sigma n)$ time, we can augment the data structure from Proposition 5.9 so that, given the packed representation of $P \in [0\,..\,\sigma)^m$ having a periodic prefix, we can compute $|\mathrm{Occ}^{a-}(P,T)|$ in $\mathcal{O}(1 + m/\log_\sigma n)$ time.*

Proof. The data structure, in addition to the structure from Proposition 5.9, contains two components: the bitvector E, and the mapping Q, as defined in Proposition 3.9. The bitvector E is augmented using Theorem 2.1 for rank and selection queries. As shown in Proposition 3.9, both structures need $\mathcal{O}(n/\log_\sigma n)$ space.

Using the structure from Proposition 5.9 and the above three components, we answer the queries as follows. First, using the data structure from Proposition 5.9, we compute $s = \text{L-head}(P)$, $H = \text{L-root}(P[1\,..\,3\tau{-}1])$, $k = \text{L-exp}(P)$, and $t = \text{L-tail}(P)$ in $\mathcal{O}(1{+}m/\log_\sigma n)$ time. This lets us determine $e(P) = 1{+}s{+}k|H|{+}t$. If $e(P) \le m$, then by Lemma 5.11, we return $|\mathrm{Occ}^{a-}(P,T)| = 0$. Otherwise, using the array $\mathrm{ISA}_{3\tau-1}$ (stored as a part of data structure from Proposition 5.9), we compute in $\mathcal{O}(1)$ time a pair of integers b, e such that $\mathrm{SA}(b\,..\,e]$ contains the starting positions of all suffixes of T prefixed with $P[1\,..\,3\tau{-}1]$. Equivalently, by Lemma 3.5 (see also the implementation of queries in Proposition 3.9), $\mathrm{SA}(b\,..\,e]$ contains all positions from $\mathsf{R}_{s,H}$. Our goal now is to determine the subrange of $\mathrm{SA}(b\,..\,e]$ containing all positions in $\{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) > \text{L-exp}(P)\}$ (these positions form a subrange by Lemma 3.5). For that, we first compute $d = \mathsf{rank}_{E,1}(e) - \mathsf{rank}_{E,1}(b)$ in $\mathcal{O}(1)$ time. If $d = 0$, then $\mathsf{R}^-_{s,H} = \emptyset$, and hence we return $|\mathrm{Occ}^{a-}(P,T)| = 0$. Otherwise, we retrieve $k_{\min} = Q[\text{int}(H)]$ in $\mathcal{O}(1)$ time. Then, letting $k_{\max} = k_{\min} + d - 1$, we have $k_{\min} \le k_{\max}$ and $[k_{\min}\,..\,k_{\max}] = \{\text{L-exp}(j) : j \in \mathsf{R}^-_{s,H}\}$ (see the proof of Proposition 3.9). If $k \ge k_{\max}$, by Lemma 5.11, we return $|\mathrm{Occ}^{a-}(P,T)| = 0$. Otherwise, we have two cases. Let $p = \mathsf{rank}_{E,1}(b)$. If $k < k_{\min}$, then we return $|\mathrm{Occ}^{a-}(P,T)| = |\mathsf{R}^-_{s,H}| = \mathsf{select}_{E,1}(p+d) - b$. Otherwise (i.e., $k \ge k_{\min}$), we return $|\mathrm{Occ}^{a-}(P,T)| = \mathsf{select}_{E,1}(p{+}d) - \mathsf{select}_{E,1}(p{+}k{-}k_{\min}{+}1)$. In total, the query takes $\mathcal{O}(1 + m/\log_\sigma n)$ time.

The data structure is constructed as follows. We first compute the sequence $(s_t)_{t\in[1\,..\,n']}$ containing the elements of S in sorted order (see Section 3.2). It can be obtained in $\mathcal{O}(1 + |\mathsf{S}|) = \mathcal{O}(n/\log_\sigma n)$ time using select queries on the bitvector B (which is part of $\mathrm{C}_{\text{count}}(T,\mathsf{S})$). We then in $\mathcal{O}(n/\log_\sigma n)$ time construct $\mathrm{C}_{\text{ISA}}(T,\mathsf{S})$ using Proposition 3.2 (recall that the packed representation of T is a component of $\mathrm{C}_{\text{count}}(T,\mathsf{S})$). We then use Propositions 3.7 and 3.9 to augment in $\mathcal{O}(n/\log_\sigma n)$ time the structure $\mathrm{C}_{\text{ISA}}(T,\mathsf{S})$ into a data structure from Proposition 3.9. This gives us bitvector E (augmented for rank and selection queries) and table Q. We then discard all remaining components of the data structures from Proposition 3.2 and Proposition 3.9. □

Computing $|\mathrm{Occ}^{s-}(P,T)|$ We now describe a data structure, that computes $|\mathrm{Occ}^{s-}(P,T)|$ for any $P \in [0\,..\,\sigma)^m$ with a periodic prefix.

Lemma 5.13. *Let $P{\in}[0\,..\,\sigma)^m$ be a string with a periodic prefix. Denote $H{=}\text{L-root}(P[1\,..\,3\tau{-}1])$. Assume $i \in \mathsf{R}^-_H$ and let $\ell = e_i - i - 3\tau + 2$. Then, $|\mathrm{Occ}^{s-}(P,T) \cap [i\,..\,i+\ell)| \le 1$. Moreover, $|\mathrm{Occ}^{s-}(P,T) \cap [i\,..\,i+\ell)| = 1$ holds if and only if $P[e^{\text{full}}(P)\,..\,m]$ is a prefix of $T[e^{\text{full}}_i\,..\,n]$ and $e^{\text{full}}_i - i \ge e^{\text{full}}(P) - 1$.*

Proof. As observed in the proof of Lemma 3.10, $[i\,..\,i+\ell) \subseteq \mathsf{R}^-_H$, and for any $\delta \in [0\,..\,\ell)$, it holds $e_{i+\delta} = e_i$, $\text{L-tail}(i+\delta) = \text{L-tail}(i)$, and consequently, $e^{\text{full}}_{i+\delta} = e^{\text{full}}_i$ and $e^{\text{full}}_{i+\delta} - (i+\delta) = e^{\text{full}}_i - i - \delta$. Moreover, by definition of $\mathrm{Occ}^{s-}(P,T)$, letting $\text{L-head}(P) = s$, for any $j \in \mathrm{Occ}^{s-}(P,T)$ it holds $e^{\text{full}}_j - j = s + \text{L-exp}(j)\cdot|H| = s + \text{L-exp}(P)\cdot|H| = e^{\text{full}}(P) - 1$. Thus, $i + \delta \in \mathrm{Occ}^{s-}(P,T)$ implies $e^{\text{full}}_{i+\delta} - (i+\delta) = e^{\text{full}}_i - (i+\delta) = e^{\text{full}}(P) - 1$, or equivalently, $\delta = (e^{\text{full}}_i - i) - (e^{\text{full}}(P) - 1)$, and therefore, $|\mathrm{Occ}^{s-}(P,T) \cap [i\,..\,i+\ell)| \le 1$.

For the second part, assume first that $i{+}\delta \in \mathrm{Occ}^{s-}(P,T)$ holds for some $\delta \in [0\,..\,\ell)$. Then, as noted above, we have $e^{\text{full}}(P) - 1 = e^{\text{full}}_i - (i+\delta) \le e^{\text{full}}_i - i$. Moreover, letting $\text{L-head}(P) = s$, by definition of $\mathrm{Occ}^{s-}(P,T)$, we have $i{+}\delta \in \mathsf{R}^-_{s,H}$, $\text{L-exp}(P) = \text{L-exp}(i{+}\delta)$, and $T[i{+}\delta\,..\,i{+}\delta{+}m) = P$. Therefore, we obtain that $T[i{+}\delta\,..\,e^{\text{full}}_{i+\delta}) = T[i{+}\delta\,..\,e^{\text{full}}_i) = P[1\,..\,e^{\text{full}}(P)) = H'H^k$ (where $k = \text{L-exp}(P)$ and H' is the length-s prefix of H), and consequently, $P[e^{\text{full}}(P)\,..\,m]$ is a prefix of $T[e^{\text{full}}_i\,..\,n]$. To show the converse implication, assume that $P[e^{\text{full}}(P)\,..\,m]$ is a prefix of $T[e^{\text{full}}_i\,..\,n]$

and $e_i^{\text{full}} - i \geq e^{\text{full}}(P) - 1$. Let $\delta = (e_i^{\text{full}} - i) - (e^{\text{full}}(P) - 1)$. We will prove that $\delta \in [0\,..\,\ell)$ and $i + \delta \in \text{Occ}^{s-}(P, T)$. Clearly $\delta \geq 0$. To show $\delta < \ell$, we first prove $e_i - e_i^{\text{full}} \geq e(P) - e^{\text{full}}(P)$. Suppose that $q = e_i - e_i^{\text{full}} < e(P) - e^{\text{full}}(P)$. By $i \in \mathsf{R}_H^-$, we then either have $e_i^{\text{full}} + q = n + 1$, or $e_i^{\text{full}} + q \leq n$ and $T[e_i^{\text{full}} + q] \neq T[e_i^{\text{full}} + q - |H|] = P[e^{\text{full}}(P) + q - |H|] = P[e^{\text{full}}(P) + q]$, both of which contradict that $P[e^{\text{full}}(P)\,..\,m]$ is a prefix of $T[e_i^{\text{full}}\,..\,n]$. Thus, $e_i - e_i^{\text{full}} \geq e(P) - e^{\text{full}}(P)$. This implies, $e_i - (i + \delta) = (e_i^{\text{full}} - (i + \delta)) + (e_i - e_i^{\text{full}}) = (e^{\text{full}}(P) - 1) + (e_i - e_i^{\text{full}}) \geq (e^{\text{full}}(P) - 1) + (e(P) - e^{\text{full}}(P)) = e(P) - 1 \geq 3\tau - 1$, or equivalently $\delta \leq e_i - i - 3\tau + 1 < \ell$. To show $i + \delta \in \text{Occ}^{s-}(P, T)$, it remains to observe that $e_{i+\delta}^{\text{full}} - (i + \delta) = e_i^{\text{full}} - (i + \delta) = e^{\text{full}}(P) - 1$ and $\text{L-root}(i + \delta) = \text{L-root}(P[1\,..\,3\tau{-}1]) = H$ imply $T[i + \delta\,..\,e_i^{\text{full}}) = P[1\,..\,e^{\text{full}}(P))$. This in particular gives, letting $\text{L-head}(P) = s$, that $i + \delta \in \mathsf{R}_{s,H}$ and $\text{L-exp}(i + \delta) = \text{L-exp}(P)$. Moreover, combining it with $P[e^{\text{full}}(P)\,..\,m]$ being a prefix of $T[e_i^{\text{full}}\,..\,n]$ yields $T[i + \delta\,..\,i + \delta + m) = P$. Finally, by Lemma 3.6, $\text{type}(i + \delta) = \text{type}(i) = -1$. Therefore, $i + \delta \in \text{Occ}^{s-}(P, T)$. □

Proposition 5.14. *Given the data structure from Proposition 5.9, in $\mathcal{O}(n(\log\log n)^2/\log_\sigma n)$ time and $\mathcal{O}(n/\log_\sigma n)$ working space we can augment it into a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that, given the packed representation of $P \in [0\,..\,\sigma)^m$ having a periodic prefix, returns $|\text{Occ}^{s-}(P, T)|$ in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time.*

Proof. Let $q = |\mathsf{R}'^{-}|$ and let $(r'_i)_{i\in[1\,..\,q]}$ be the sequence containing all positions $j \in \mathsf{R}'^{-}$ sorted first by $\text{L-root}(j)$, and in case of ties, by $T[e_j^{\text{full}}\,..\,n]$ (see also the proof of Proposition 3.12). Recall that $\mathcal{H} = \{\text{L-root}(X) : X \in \mathcal{F}\}$. For any string $H \in \mathcal{H}$, let $\text{pow}(H) = H^\infty[1\,..\,|H|\lceil\frac{\tau}{|H|}\rceil]$. This function satisfies the following properties:

- The set $\{\text{pow}(H) : H \in \mathcal{H}\}$ is prefix-free.
- For any $X, Y \in \mathcal{H}$, $X \prec Y$ implies $\text{pow}(X) \prec \text{pow}(Y)$.

For a proof, consider $X, Y \in \mathcal{H}$ such that $X \prec Y$. By [41, Fact 9.1.6], it holds $X \preceq \text{pow}(X) \prec X^\infty \prec Y \preceq \text{pow}(Y)$. Since $|Y| < \tau \leq |\text{pow}(X)|$, the set $\{\text{pow}(X), \text{pow}(Y)\}$ is prefix-free.

Next, we define an array $A[1\,..\,q]$ so that, for any $i \in [1\,..\,q]$, $A[i] = e_j^{\text{full}} - |\text{pow}(H_i)|$, where $j = r'_i$ and $H_i = \text{L-root}(r'_i)$. Observe that $T[A[i]\,..\,n] = \text{pow}(H_i) \cdot T[e_j^{\text{full}}\,..\,n]$. Together with the properties of the pow function and with the definition of $(r'_i)_{i\in[1\,..\,q]}$, this implies that the positions in A are sorted according to the lexicographic order of the corresponding suffixes of T, i.e., $i < i'$ implies $T[A[i]\,..\,n] \prec T[A[i']\,..\,n]$. Let D denote the data structure of Proposition 5.4 for the array $A[1\,..\,q]$ (it is well defined due to $q = \mathcal{O}(n/\log_\sigma n)$). Finally, let $(\ell_i)_{i\in[1\,..\,q]}$ be a sequence of integers defined by $\ell_i = e_j^{\text{full}} - j$, where $j = r'_i$ (see also the proof of Proposition 3.12). We define the array $A'[1\,..\,q]$ by $A'[i] = \ell_i$.

The data structure, in addition to the structure from Proposition 5.9, contains two components. First, we store the data structure D, which needs $\mathcal{O}(n/\log_\sigma n)$ space by Proposition 5.4. Second, we store a data structure supporting range counting/selection queries on A'. Using Proposition 3.11, the structure needs $\mathcal{O}(1 + q) = \mathcal{O}(n/\log_\sigma n)$ space.

We answer the queries as follows. First, using the data structure from Proposition 5.9, we compute $s = \text{L-head}(P)$, $H = \text{L-root}(P[1\,..\,3\tau{-}1])$, and $k = \text{L-exp}(P)$ in $\mathcal{O}(1 + m/\log_\sigma n)$ time. This lets us determine $e^{\text{full}}(P) = 1{+}s{+}k|H|$ and $P' := P[e^{\text{full}}(P) - |\text{pow}(H)|\,..\,m]$. Then, using Proposition 5.4, we compute in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time a range $(b_{\text{pre}}\,..\,e_{\text{pre}}] = \{i \in [1\,..\,q] : P' \text{ is a prefix of } T[A[i]\,..\,n]\}$. Observe that the set $\{r'_i : i \in (b_{\text{pre}}\,..\,e_{\text{pre}}]\}$ consists of all positions $j \in \mathsf{R}'^{-}_H$ such that $P[e^{\text{full}}(P)\,..\,m]$ is a prefix of $T[e_j^{\text{full}}\,..\,n]$. Thus, by Lemma 5.13, we have $|\text{Occ}^{s-}(P, T)| = |\{i \in (b_{\text{pre}}\,..\,e_{\text{pre}}] : \ell_i \geq e^{\text{full}}(P) - 1\}|$, which we compute in $\mathcal{O}(\log\log n)$ time using the range counting structure as $\mathsf{rcount}_{A'}(e^{\text{full}}(P) - 1, e_{\text{pre}}) - \mathsf{rcount}_{A'}(e^{\text{full}}(P) - 1, b_{\text{pre}})$.

The data structure is constructed as follows. First, as in Proposition 5.12, we construct in $\mathcal{O}(n/\log_\sigma n)$ time the sequence $(s_t)_{t\in[1\,..\,n']}$ containing the elements of S in sorted order (see Section 3.2), and then use it to compute $\text{C}_{\text{ISA}}(T, \mathsf{S})$ via Proposition 3.2. We then in $\mathcal{O}(n/\log_\sigma n)$

time augment it into a data structure from Proposition 3.7. Using that data structure, we construct in $\mathcal{O}(n/\log_\sigma n)$ time the sequences $(r'_i)_{i\in[1..q]}$ and $(\ell_i)_{i\in[1..q]}$, as explained in the proof of Proposition 3.12. Using Proposition 3.7, we can now compute $A[i]$ for any $i \in [1..q]$ in $\mathcal{O}(1)$ time. We then discard $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$ and the structure from Proposition 3.7. Next, in $\mathcal{O}(n(\log\log n)^2/\log_\sigma n)$ time and $\mathcal{O}(n/\log_\sigma n)$ working space, we construct the data structure D using Proposition 5.4. Finally, we construct the array A' and augment it with the range counting data structure. Using Proposition 3.11, by $q = \mathcal{O}(n/\log_\sigma n)$ and $\sum_{i=1}^{q} A'[i] = \mathcal{O}(n)$, this takes $\mathcal{O}(n/\log_\sigma n)$ time. □

Summary By combining all above results, we obtain the following data structure to compute $|\mathrm{Occ}(P,T)|$ for patterns P having a periodic prefix.

Proposition 5.15. *Given the structure $\mathrm{C}_{\mathrm{count}}(T,\mathsf{S})$, we can in $\mathcal{O}(n(\log\log n)^2/\log_\sigma n)$ time and in $\mathcal{O}(n/\log_\sigma n)$ working space augment it into a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that, given the packed representation of $P \in [0..\sigma)^m$ having a periodic prefix, returns $|\mathrm{Occ}(P,T)|$ in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time.*

Proof. The data structure is a composition of four data structures. First, we store the two data structures presented in Propositions 5.12 and 5.14. These structures are designed to compute $|\mathrm{Occ}^{a-}(P,T)|$ and $|\mathrm{Occ}^{s-}(P,T)|$. To compute $|\mathrm{Occ}^{a+}(P,T)|$ and $|\mathrm{Occ}^{s+}(P,T)|$, we store their symmetric versions. All four data structures take $\mathcal{O}(n/\log_\sigma n)$ space. All structures are built on top of the structure from Proposition 5.9, but it suffices to only keep its single copy (in particular, we keep a single copy of $\mathrm{C}_{\mathrm{count}}(T,\mathsf{S})$).

Using the above data structures, given a packed representation of $P \in [0..\sigma)^m$ such that $m \geq 3\tau - 1$ and $P[1..3\tau{-}1] \in \mathcal{F}$, we compute $|\mathrm{Occ}(P,T)| = |\mathrm{Occ}^{a-}(P,T)| + |\mathrm{Occ}^{s-}(P,T)| + |\mathrm{Occ}^{a+}(P,T)| + |\mathrm{Occ}^{s+}(P,T)|$ using the data structures from Propositions 5.12 and 5.14 and their symmetric counterparts. The total time is $\mathcal{O}(m/\log_\sigma n + \log\log n)$.

The data structure is constructed as follows. First, given $\mathrm{C}_{\mathrm{count}}(T,\mathsf{S})$ we construct the data structure from Proposition 5.9. This takes $\mathcal{O}(n/\log_\sigma n)$ time. We then augment it into the two data structures to compute $|\mathrm{Occ}^{a-}(P,T)|$ and $|\mathrm{Occ}^{s-}(P,T)|$. Using Propositions 5.12 and 5.14, this takes $\mathcal{O}(n/\log_\sigma n)$ and $\mathcal{O}(n(\log\log n)^2/\log_\sigma n)$ time (respectively) and $\mathcal{O}(n/\log_\sigma n)$ working space. Finally, we analogously construct the two data structures to compute $|\mathrm{Occ}^{a-}(P,T)|$ and $|\mathrm{Occ}^{s+}(P,T)|$. The total time is $\mathcal{O}(n(\log\log n)^2/\log_\sigma n)$, and the working space never exceeds the optimal $\mathcal{O}(n/\log_\sigma n)$ words. □

5.3.2 Computing $\mathrm{RangeBeg}(P,T)$ and $\mathrm{RangeEnd}(P,T)$

Let $P \in [0..\sigma)^m$ be a string with a periodic prefix. We now show how to extend the above data structure computing $|\mathrm{Occ}(P,T)|$ to instead return $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$.

Let $\mathrm{pos}(P,T) = \{j \in [1..n] : \mathrm{lcp}(T[j..n],P) \geq 3\tau - 1 \text{ and } T[j..n] \prec P\}$. Denote $\delta(P,T) = |\mathrm{pos}(P,T)|$. Let $\mathrm{SA}(b..e]$ be the range containing the starting positions of all suffixes of T having $X = P[1..3\tau{-}1]$ as a prefix. Since $\mathcal{D} \cup \mathcal{F}$ is prefix-free, we have $b \leq \mathrm{RangeBeg}(P,T) \leq \mathrm{RangeEnd}(P,T) \leq e$, and it is easy to see that $\mathrm{RangeBeg}(P,T) = b + \delta(P,T)$. Thus, since the pair (b,e), is easy to compute, the value $|\mathrm{Occ}(P,T)|$ can be obtained using Proposition 5.15, and it holds $\mathrm{RangeEnd}(P,T) = \mathrm{RangeBeg}(P,T) + |\mathrm{Occ}(P,T)|$, the difficulty in obtaining the pair $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ lies in computing $\delta(P,T)$.

We focus on computing $\delta(P,T)$ for P satisfying $\mathrm{type}(P) = -1$ (the structure for P satisfying $\mathrm{type}(P) = +1$ is symmetric; see the proof of Proposition 5.20). Denote $s = \text{L-head}(P)$ and $H = \text{L-root}(P[1..3\tau{-}1])$. We define $\mathrm{pos}^a(P,T) = \{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) \leq \text{L-exp}(P)\}$ and $\mathrm{pos}^s(P,T) = \{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) = \text{L-exp}(P) \text{ and } T[j..n] \succeq P\}$. We denote $\delta^a(P,T) = |\mathrm{pos}^a(P,T)|$ and $\delta^s(P,T) = |\mathrm{pos}^s(P,T)|$.

Lemma 5.16. *For any pattern $P \in [0\,..\,\sigma)^m$ that has a periodic prefix and satisfies* $\mathrm{type}(P) = -1$, *it holds* $\delta(P,T) = \delta^a(P,T) - \delta^s(P,T)$.

Proof. We will prove that $\mathrm{pos}^a(P,T)$ is a disjoint union of $\mathrm{pos}(P,T)$ and $\mathrm{pos}^s(P,T)$. This implies $\delta(P,T) + \delta^s(P,T) = \delta^a(P,T)$, and consequently, the equality in the claim.

By Lemmas 5.7 and 5.8, letting $j \in \mathsf{R}^-_{s,H}$, we have $\mathrm{pos}(P,T) = \{j \in \mathsf{R}^-_{s,H} : T[j\,..\,n] \prec P\}$, and moreover, if $j \in \mathrm{pos}(P,T)$, then $e_j - j \leq e(P) - 1$. In particular, $\text{L-exp}(j) = \lfloor \frac{e_j - j - s}{|H|} \rfloor \leq \lfloor \frac{e(P)-1-s}{|H|} \rfloor = \text{L-exp}(P)$. Hence, $\mathrm{pos}(P,T) \subseteq \mathrm{pos}^a(P,T)$. On the other hand, clearly $\mathrm{pos}^s(P,T) \subseteq \mathrm{pos}^a(P,T)$ and $\mathrm{pos}^s(P,T) \cap \mathrm{pos}(P,T) = \emptyset$. Thus, to obtain the claim, it suffices to show that $\mathrm{pos}^a(P,T) \setminus \mathrm{pos}^s(P,T) \subseteq \mathrm{pos}(P,T)$.

Let $j \in \mathrm{pos}^a(P,T) \setminus \mathrm{pos}^s(P,T)$. Consider two cases. If $\text{L-exp}(j) = \text{L-exp}(P)$, then by definition of $\mathrm{pos}^s(P,T)$, it must hold $T[j\,..\,n] \prec P$. Thus, we have $j \in \mathrm{pos}(P,T)$. Let us therefore assume $\text{L-exp}(j) < \text{L-exp}(P)$. Then, $e_j - j = s + \text{L-exp}(j) \cdot |H| + \text{L-tail}(j) < s + \text{L-exp}(j) \cdot |H| + |H| \leq s + \text{L-exp}(P) \cdot |H| \leq s + \text{L-exp}(P) \cdot |H| + \text{L-tail}(P) = e(P) - 1$. By Item 2 of Lemma 5.8, this implies $T[j\,..\,n] \prec P$, and consequently, $j \in \mathrm{pos}(P,T)$. □

Computing $\delta^a(P,T)$ We now describe a data structure that given any $P \in [0\,..\,\sigma)^m$ that has a periodic prefix and satisfies $\mathrm{type}(P) = -1$, allows computing $\delta^a(P,T)$.

Proposition 5.17. *Given the data structure from Proposition 5.12 and the packed representation of $P \in [0\,..\,\sigma)^m$ having a periodic prefix and satisfying* $\mathrm{type}(P) = -1$, *we can in $\mathcal{O}(1 + m/\log_\sigma n)$ time compute $\delta^a(P,T)$.*

Proof. First, using the data structure from Proposition 5.9 (which is stored as a part of structure from Proposition 5.12), we compute $H = \text{L-root}(P)$ and $k = \text{L-exp}(P)$ in $\mathcal{O}(1 + m/\log_\sigma n)$ time. Then, using the array $\mathrm{ISA}_{3\tau-1}$ (stored as a part of data structure from Proposition 5.9), we compute in $\mathcal{O}(1)$ time a pair of integers b, e such that $\mathrm{SA}(b\,..\,e]$ contains the starting positions of all suffixes of T prefixed with $P[1\,..\,3\tau{-}1]$. Equivalently, by Lemma 5.7, $\mathrm{SA}(b\,..\,e]$ contains all positions from $\mathsf{R}_{s,H}$, where $s = \text{L-head}(P)$. Our goal now is to determine the subrange of $\mathrm{SA}(b\,..\,e]$ containing all positions in $\{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) \leq \text{L-exp}(P)\}$ (these positions form a subrange by Lemma 3.5). For that, we first compute $d = \mathsf{rank}_{E,1}(e) - \mathsf{rank}_{E,1}(b)$ in $\mathcal{O}(1)$ time. If $d = 0$, then $\mathsf{R}^-_{s,H} = \emptyset$, and hence we return $\delta^a(P,T) = 0$. Otherwise, we retrieve $k_{\min} = Q[\mathrm{int}(H)]$ in $\mathcal{O}(1)$ time. Then, letting $k_{\max} = k_{\min} + d - 1$, we have $k_{\min} \leq k_{\max}$ and $[k_{\min}\,..\,k_{\max}] = \{\text{L-exp}(j) : j \in \mathsf{R}^-_{s,H}\}$ (see the proof of Proposition 3.9). If $k < k_{\min}$, we return $\delta^a(P,T) = 0$. Otherwise, we have two cases. Let $p = \mathsf{rank}_{E,1}(b)$. If $k \geq k_{\max}$, then we return $\delta^a(P,T) = |\mathsf{R}^-_{s,H}| = \mathsf{select}_{E,1}(p + d) - b$. Otherwise (i.e., $k < k_{\max}$), we return $\delta^a(P,T) = \mathsf{select}_{E,1}(p + k - k_{\min} + 1) - b$. In total, the query takes $\mathcal{O}(1 + m/\log_\sigma n)$ time. □

Computing $\delta^s(P,T)$ We now describe a data structure that given any $P \in [0\,..\,\sigma)^m$ that has a periodic prefix and satisfies $\mathrm{type}(P) = -1$, allows computing $\delta^s(P,T)$.

Lemma 5.18. *Let $P \in [0\,..\,\sigma)^m$ be a pattern that has a periodic prefix and satisfies* $\mathrm{type}(P) = -1$. *Denote $H = \text{L-root}(P[1\,..\,3\tau{-}1])$. Assume $i \in \mathsf{R}^-_H$ and let $\ell = e_i - i - 3\tau + 2$. Then, we have $|\mathrm{pos}^s(P,T) \cap [i\,..\,i+\ell)| \leq 1$. Moreover, $|\mathrm{pos}^s(P,T) \cap [i\,..\,i+\ell)| = 1$ holds if and only if $T[e^{\mathrm{full}}_i\,..\,n] \succeq P[e^{\mathrm{full}}(P)\,..\,m]$ and $e^{\mathrm{full}}_i - i \geq e^{\mathrm{full}}(P) - 1$.*

Proof. In the proof of Lemma 5.13, it is shown that $[i\,..\,i+\ell) \subseteq \mathsf{R}^-_H$, and for any $\delta \in [0\,..\,\ell)$, it holds $e^{\mathrm{full}}_{i+\delta} - (i+\delta) = e^{\mathrm{full}}_i - i - \delta$. By definition of $\mathrm{pos}^s(P,T)$, letting $s = \text{L-head}(P)$, for any $j \in \mathrm{pos}^s(P,T)$ it holds $e^{\mathrm{full}}_j - j = s + \text{L-exp}(j) \cdot |H| = s + \text{L-exp}(P) \cdot |H| = e^{\mathrm{full}}(P) - 1$. Thus, $i + \delta \in \mathrm{pos}^s(P,T)$ implies $e^{\mathrm{full}}_{i+\delta} - (i+\delta) = e^{\mathrm{full}}_i - (i+\delta) = e^{\mathrm{full}}(P) - 1$, or equivalently, $\delta = (e^{\mathrm{full}}_i - i) - (e^{\mathrm{full}}(P) - 1)$, and therefore, $|\mathrm{pos}^s(P,T) \cap [i\,..\,i+\ell)| \leq 1$.

For the second part, assume first that $i+\delta \in \mathrm{pos}^s(P,T)$ holds for some $\delta \in [0\,..\,\ell)$. Then, as noted above, we have $e^{\mathrm{full}}(P) - 1 = e_i^{\mathrm{full}} - (i+\delta) \le e_i^{\mathrm{full}} - i$. Moreover, letting $\text{L-head}(P) = s$, by definition of $\mathrm{pos}^s(P,T)$, we have $i+\delta \in \mathsf{R}^-_{s,H}$, $\text{L-exp}(P) = \text{L-exp}(i+\delta)$, and $T[i+\delta\,..\,n] \succeq P$. Therefore, we obtain that $T[i+\delta\,..\,e^{\mathrm{full}}_{i+\delta}) = T[i+\delta\,..\,e^{\mathrm{full}}_i) = P[1\,..\,e^{\mathrm{full}}(P)) = H'H^k$ (where $k = \text{L-exp}(P)$ and H' is the length-s prefix of H), and consequently, $T[e_i^{\mathrm{full}}\,..\,n] \succeq P[e^{\mathrm{full}}(P)\,..\,m]$. To show the converse implication, assume that $T[e_i^{\mathrm{full}}\,..\,n] \succeq P[e^{\mathrm{full}}(P)\,..\,m]$ and $e_i^{\mathrm{full}} - i \ge e^{\mathrm{full}}(P) - 1$. Let $\delta = (e_i^{\mathrm{full}} - i) - (e^{\mathrm{full}}(P) - 1)$. We will prove that $\delta \in [0\,..\,\ell)$ and $i+\delta \in \mathrm{pos}^s(P,T)$. Clearly $\delta \ge 0$. To show $\delta < \ell$, we first prove $e_i - e_i^{\mathrm{full}} \ge e(P) - e^{\mathrm{full}}(P)$. Suppose that $q = e_i - e_i^{\mathrm{full}} < e(P) - e^{\mathrm{full}}(P)$. By $i \in \mathsf{R}^-_H$, we then either have $e_i^{\mathrm{full}} + q = n+1$, or $e_i^{\mathrm{full}} + q \le n$ and $T[e_i^{\mathrm{full}} + q] \prec T[e_i^{\mathrm{full}} + q - |H|] = P[e^{\mathrm{full}}(P) + q - |H|] = P[e^{\mathrm{full}}(P) + q]$, both of which contradict $T[e_i^{\mathrm{full}}\,..\,n] \succeq P[e^{\mathrm{full}}(P)\,..\,m]$. Thus, $e_i - e_i^{\mathrm{full}} \ge e(P) - e^{\mathrm{full}}(P)$. This implies, $e_i - (i+\delta) = (e_i^{\mathrm{full}} - (i+\delta)) + (e_i - e_i^{\mathrm{full}}) = (e^{\mathrm{full}}(P) - 1) + (e_i - e_i^{\mathrm{full}}) \ge (e^{\mathrm{full}}(P) - 1) + (e(P) - e^{\mathrm{full}}(P)) = e(P) - 1 \ge 3\tau - 1$, or equivalently $\delta \le e_i - i - 3\tau + 1 < \ell$. To show $i+\delta \in \mathrm{pos}^s(P,T)$, it remains to observe that $e^{\mathrm{full}}_{i+\delta} - (i+\delta) = e_i^{\mathrm{full}} - (i+\delta) = e^{\mathrm{full}}(P) - 1$ and $\text{L-root}(i+\delta) = \text{L-root}(P[1\,..\,3\tau{-}1]) = H$ imply $T[i+\delta\,..\,e_i^{\mathrm{full}}) = P[1\,..\,e^{\mathrm{full}}(P))$. This in particular gives, letting $\text{L-head}(P) = s$, that $i+\delta \in \mathsf{R}_{s,H}$ and $\text{L-exp}(i+\delta) = \text{L-exp}(P)$. Moreover, combining it with $T[e_i^{\mathrm{full}}\,..\,n] \succeq P[e^{\mathrm{full}}(P)\,..\,m]$ yields $T[i+\delta\,..\,n] \succeq P$. Finally, by Lemma 3.6, $\mathrm{type}(i+\delta) = \mathrm{type}(i) = -1$. Therefore, $i+\delta \in \mathrm{pos}^s(P,T)$. □

Proposition 5.19. *Given the data structure from Proposition 5.9, in $\mathcal{O}(n(\log\log n)^2/\log_\sigma n)$ time and $\mathcal{O}(n/\log_\sigma n)$ working space we can augment it into a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that, given the packed representation of $P \in [0\,..\,\sigma)^m$ having a periodic prefix and satisfying $\mathrm{type}(P) = -1$, returns $\delta^s(P,T)$ in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time.*

Proof. The data structure, in addition to the structure from Proposition 5.9, contains three components. First, we store the data structure D (as defined in Proposition 5.14), which needs $\mathcal{O}(n/\log_\sigma n)$ space by Proposition 5.4. Second, we store a data structure supporting range counting/selection queries on A' (as defined in Proposition 5.14). Using Proposition 3.11, the structure needs $\mathcal{O}(1+q) = \mathcal{O}(n/\log_\sigma n)$ space. Third, and final, we store a lookup table C, as defined in Proposition 3.12. As shown in Proposition 3.12, C needs $\mathcal{O}(n/\log_\sigma n)$ space.

We answer the queries as follows. First, using the data structure from Proposition 5.9, we compute $s = \text{L-head}(P)$, $H = \text{L-root}(P[1\,..\,3\tau{-}1])$, and $k = \text{L-exp}(P)$ in $\mathcal{O}(1 + m/\log_\sigma n)$ time. This lets us determine $e^{\mathrm{full}}(P) = 1{+}s{+}k|H|$ and $P' := P[e^{\mathrm{full}}(P) - |\mathrm{pow}(H)|\,..\,m]$. Then, using Proposition 5.4, we compute in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time a value $x = |\{i \in [1\,..\,q] : T[A[i]\,..\,n] \prec P'\}|$ (where A and q are defined as in Proposition 5.14). Then, letting $x' = C[\mathrm{int}(H)]$, by definition of A and properties of function pow (see the proof of Proposition 5.14), the set $\{r'_i : i \in (x\,..\,x']\}$ (where r'_i is defined as in the proof of Proposition 5.14) consists of all positions $j \in \mathsf{R}'^-_H$ satisfying $T[e_j^{\mathrm{full}}\,..\,n] \succeq P[e^{\mathrm{full}}(P)\,..\,m]$. Thus, by Lemma 5.18, it holds $\delta^s(P,T) = |\mathrm{pos}^s(P,T)| = |\{i \in (x\,..\,x'] : \ell_i \ge e^{\mathrm{full}}(P) - 1\}|$ (where ℓ_i is defined as in Proposition 5.14), which we compute in $\mathcal{O}(\log\log n)$ time using the range counting structure as $\mathsf{rcount}_{A'}(e^{\mathrm{full}}(P) - 1, x') - \mathsf{rcount}_{A'}(e^{\mathrm{full}}(P) - 1, x)$.

The data structure is constructed as follows. First, as in Proposition 5.12, we construct in $\mathcal{O}(n/\log_\sigma n)$ time the sequence $(s_t)_{t\in[1\,..\,n']}$ containing the elements of S in sorted order (see Section 3.2), and then use it to compute $\mathrm{C}_{\mathrm{ISA}}(T,\mathsf{S})$ via Proposition 3.2. We then in $\mathcal{O}(n/\log_\sigma n)$ time augment it into a data structure from Proposition 3.7. Using that data structure, we construct in $\mathcal{O}(n/\log_\sigma n)$ time the lookup table C, as explained in the proof of Proposition 3.12. The data structure D and the structure supporting range counting/selection queries on A' are constructed in $\mathcal{O}(n(\log\log n)^2/\log_\sigma n)$ time and $\mathcal{O}(n/\log_\sigma n)$ working space as explained in the proof of Proposition 5.14. □

Summary By combining the above results, we obtain the following data structure to compute $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ for patterns with periodic prefixes.

Proposition 5.20. *Given the structure* $\mathrm{C}_{\mathrm{count}}(T,\mathsf{S})$*, we can in* $\mathcal{O}(n(\log\log n)^2/\log_\sigma n)$ *time and in* $\mathcal{O}(n/\log_\sigma n)$ *working space augment it into a data structure of size* $\mathcal{O}(n/\log_\sigma n)$ *that, given the packed representation of* $P \in [0\,..\,\sigma)^m$ *having a periodic prefix, returns the pair* $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ *in* $\mathcal{O}(m/\log_\sigma n + \log\log n)$ *time.*

Proof. The data structure is a composition of five data structures. First, we store the two data structures presented in Propositions 5.17 and 5.19. These structures are designed to compute values $\delta^a(P,T)$ and $\delta^s(P,T)$ for P satisfying $\mathrm{type}(P) = -1$. To handle $\mathrm{type}(P) = +1$, we store their symmetric versions adapted according to Lemma 5.8 (more precisely, if $\mathrm{type}(P) = +1$, the data structures compute $\delta^a(P,T) = |\mathrm{pos}^a(P,T)|$ and $\delta^s(P,T) = |\mathrm{pos}^s(P,T)|$, where $\mathrm{pos}^a(P,T) = \{j \in \mathsf{R}^+_{s,H} : \text{L-exp}(j) \le \text{L-exp}(P)\}$ and $\mathrm{pos}^s(P,T) = \{j \in \mathsf{R}^+_{s,H} : \text{L-exp}(j) = \text{L-exp}(j) \text{ and } T[j\,..\,n] \prec P\}$). All four data structures take $\mathcal{O}(n/\log_\sigma n)$ space. All structures are built on top of the structure from Proposition 5.9, but it suffices to only keep its single copy (in particular, we keep a single copy of $\mathrm{C}_{\mathrm{count}}(T,\mathsf{S})$). Finally, we store a data structure to compute $|\mathrm{Occ}(P,T)|$ from Proposition 5.15. It also needs $\mathcal{O}(n/\log_\sigma n)$ space.

Using the above data structures, given the packed representation of $P \in [0\,..\,\sigma)^m$ having a periodic prefix, we compute $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ as follows. First, using Proposition 5.15 in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time we compute $|\mathrm{Occ}(P,T)|$. Next, using Lemma 5.2, in $\mathcal{O}(1)$ time we compute integers b, e such that $\mathrm{SA}(b\,..\,e]$ contains the starting positions of all suffixes of T starting with $P[1\,..\,3\tau-1]$. Then, in $\mathcal{O}(1 + m/\log_\sigma n)$ time using Proposition 5.9 we determine $\mathrm{type}(P)$. Depending on whether $\mathrm{type}(P) = -1$ or $\mathrm{type}(P) = +1$, we use either a combination of Propositions 5.17 and 5.19 or their symmetric counterparts, to compute $\delta^a(P,T)$ and $\delta^s(P,T)$ in $\mathcal{O}(1 + m/\log_\sigma n)$ and $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time, respectively. If $\mathrm{type}(P) = -1$, then by Lemma 5.16 we have $\delta(P,T) = \delta^a(P,T) - \delta^s(P,T)$. Otherwise, by the counterpart of Lemma 5.16, $\delta(P,T) = (e-b) - (\delta^a(P,T) - \delta^s(P,T))$. Finally, we return $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T)) = (b + \delta(P,T), b + \delta(P,T) + |\mathrm{Occ}(P,T)|)$ as the answer. In total, the query takes $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time.

The data structure is constructed as follows. First, given $\mathrm{C}_{\mathrm{count}}(T,\mathsf{S})$, we construct the data structure from Proposition 5.9. This takes $\mathcal{O}(n/\log_\sigma n)$ time. We then augment it into the two data structures to compute $\delta^a(P,T)$ and $\delta^s(P,T)$ for P satisfying $\mathrm{type}(P) = -1$. Using Propositions 5.17 and 5.19, this takes $\mathcal{O}(n/\log_\sigma n)$ and $\mathcal{O}(n(\log\log n)^2/\log_\sigma n)$ time (respectively), and $\mathcal{O}(n/\log_\sigma n)$ working space. We analogously construct the two data structures to compute $\delta^a(P,T)$ and $\delta^s(P,T)$ for P satisfying $\mathrm{type}(P) = +1$. Finally, using Proposition 5.15, we construct the structure to compute $|\mathrm{Occ}(P,T)|$ in $\mathcal{O}(n(\log\log n)^2/\log_\sigma n)$ time and $\mathcal{O}(n/\log_\sigma n)$ working space. □

5.4 The Final Data Structure

Theorem 5.21. *Given a constant* $\epsilon \in (0,1)$ *and the packed representation of a text* $T \in [0\,..\,\sigma)^n$ *with* $2 \le \sigma < n^{1/6}$*, we can construct in* $\mathcal{O}(n\log\sigma/\sqrt{\log n})$ *time and* $\mathcal{O}(n/\log_\sigma n)$ *working space a data structure of size* $\mathcal{O}(n/\log_\sigma n)$ *that, given the packed representation of* $P \in [0\,..\,\sigma)^m$*, returns the pair* $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ *in* $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$ *time.*

Proof. The data structure is a composition of the data structures from Proposition 5.6 and Proposition 5.20. Both data structures take $\mathcal{O}(n/\log_\sigma n)$ space. Each of the two structures is built on top of the index core $\mathrm{C}_{\mathrm{count}}(T,\mathsf{S})$, but it suffices to store a single copy of $\mathrm{C}_{\mathrm{count}}(T,\mathsf{S})$.

Given the packed encoding of $P \in [0\,..\,\sigma)^m$, we compute $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ as follows. If $m < 3\tau - 1$, or $m \ge 3\tau - 1$ and no element of $\mathcal{D} \cup \mathcal{F}$ is a prefix of P, we

compute $(\mathrm{RangeBeg}(P,T),\mathrm{RangeEnd}(P,T))$ in $\mathcal{O}(1)$ time from $\mathrm{C}_{\mathrm{count}}(T,\mathsf{S})$ using Lemma 5.2. If $m \geq 3\tau-1$ and P is prefixed with some $X \in \mathcal{D}$, then we use Proposition 5.6 to compute $(\mathrm{RangeBeg}(P,T),\mathrm{RangeEnd}(P,T))$ in $\mathcal{O}(m/\log_\sigma n+\log^\epsilon n)$ time. Otherwise (i.e., $P[1\,..\,3\tau-1] \in \mathcal{F}$) we use Proposition 5.20 to return the answer in $\mathcal{O}(m/\log_\sigma n+\log\log n)$ time.

The data structure is constructed as follows. First, using Theorem 2.5, from a packed representation of T, we construct a τ-synchronizing set S of size $\mathcal{O}(n/\tau)$ in $\mathcal{O}(n/\tau)=\mathcal{O}(n/\log_\sigma n)$ time. The set S is returned as an array taking $\mathcal{O}(n/\log_\sigma n)$ space. Using this array and the packed representation of T, we then construct $\mathrm{C}_{\mathrm{count}}(T,\mathsf{S})$ in $\mathcal{O}(n/\log_\sigma n)$ time using Proposition 5.3. Finally, using Propositions 5.6 and 5.20, we augment $\mathrm{C}_{\mathrm{count}}(T,\mathsf{S})$ in $\mathcal{O}(n\log\sigma/\sqrt{\log n})$ and $\mathcal{O}(n(\log\log n)^2/\log_\sigma n)$ time (respectively) and using $\mathcal{O}(n/\log_\sigma n)$ working space into structures to compute $(\mathrm{RangeBeg}(P,T),\mathrm{RangeEnd}(P,T))$. □

A Proof of Theorem 2.2

We start with an implementation of rank and selection queries for larger alphabets.

Lemma A.1 (Belazzougui and Puglisi [9])**.** *For all integers $N \geq n \geq \sigma \geq 2$ and every string $S \in [0\,..\,\sigma)^{\leq n}$ there exists a data structure of $\mathcal{O}(n\log\sigma)$ bits that answers rank queries in $\mathcal{O}(\log\log N)$ time and selection queries in $\mathcal{O}(1)$ time. Moreover, given a table precomputed in $\mathcal{O}(N)$ time (shareable across all instances with common parameter N) and the packed representation of S, the data structure can be constructed in $\mathcal{O}(\sigma+n\log\sigma/\sqrt{\log N})$ time using $\mathcal{O}(n\log\sigma)$ bits of space.*

Proof. If $\log^2\sigma \geq \log N$, we use the data structure of [9, Lemma C.2], which occupies $\mathcal{O}(n\log\sigma)$ bits, answers rank queries in $\mathcal{O}(\log\log n)$ time and selection queries in $\mathcal{O}(1)$ time, and can be constructed in $\mathcal{O}(n)$ time using $\mathcal{O}(n\log\sigma)$ bits of space.[7] Otherwise, we use the data structure of [9, Lemma C.3], which occupies $\mathcal{O}(n\log\sigma)$ bits, answers rank queries in $\mathcal{O}(\log\log N)$ time and selection queries in $\mathcal{O}(1)$ time, and can be constructed in $\mathcal{O}(\sigma+n\log^2\sigma/\log N)$ time using $\mathcal{O}(n\log\sigma)$ bits of space. □

The following proposition, instantiated with $h=\lceil\log^{\epsilon/2} m\rceil$, immediately yields Theorem 2.2.

Proposition A.2. *For all integers $h,m,b,\sigma \in \mathbb{Z}_{\geq 1}$ satisfying $h \geq 2$ and $m \geq \sigma^b \geq 2$, and every string $W \in ([0\,..\,\sigma)^b)^{\leq m}$, there exists a data structure of size $\mathcal{O}(m\log_h(hb))$ that answers prefix rank queries in $\mathcal{O}(h\log\log m\log_h(hb))$ time and prefix selection queries in $\mathcal{O}(h\log_h(hb))$ time. Moreover, it can be constructed in $\mathcal{O}(m\sqrt{\log m}\log_h(hb))$ time using $\mathcal{O}(m\log_h(hb))$ space given the packed representation of W and the parameter h.*

Proof. The data structure consists in the wavelet tree of W and, when $h \leq b$, an instance constructed recursively for an auxiliary string $\tilde{W}$.

Wavelet tree Let $\Sigma=[0\,..\,\sigma)$ so that the alphabet of W is Σ^b. The wavelet tree of W [31] is the trie of Σ^b with each internal node v_X (representing a string $X \in \Sigma^{\leq b-1}$) associated to a string $B_X[1\,..\,\mathsf{rank}_{W,X}(|W|)] \in \Sigma^*$ such that $B_X[r]=W[\mathsf{select}_{W,X}(r)][|X|+1]$ for $r \in [1\,..\,\mathsf{rank}_{W,X}(|W|)]$. The strings B_X are augmented with the component of Lemma A.1 for $N=m$.

[7] The statement of [9, Lemma C.2] does not bound the space consumption of the construction algorithm. Nevertheless, it is straightforward to implement the underlying construction procedure in $\mathcal{O}(n\log\sigma)$ bits of working space. The original algorithm scans the input sequence S from left to right and, for each $a \in \Sigma$, builds an array $P_a[1\,..\,n_a]$ such that $n_a=\mathsf{rank}_{S,a}(|S|)$ and $P_a[r]=\mathsf{select}_{S,a}(r)$ for $r \in [1\,..\,n_a]$. The array $P_a[1\,..\,n_a]$ is then converted to the Elias–Fano representation: an array $A_a[1\,..\,n_a]$ with $A_a[r]=P_a[r] \bmod \sigma$ for $r \in [1\,..\,n_a]$ and a bit vector $V_a=\mathrm{unary}((\lfloor P_a[r]/\sigma\rfloor-\lfloor P_a[r-1]/\sigma\rfloor)_{r\in[1\,..\,n_a]})$, where we assume $P_a[0]=0$ to streamline the formula. To achieve $\mathcal{O}(n\log\sigma)$ bits of working space, instead of storing P_a explicitly, we convert P_a to the Elias–Fano representation on the fly as subsequent positions are appended to P_a.

Recursive instance We shall define $\tilde{W}$ as a string of length $|W|$ over the alphabet $\tilde{\Sigma}^{\tilde{b}}$, where $\tilde{b} := \lfloor b/h \rfloor$ and $\tilde{\Sigma} := [0\,..\,\sigma^h)$. We shall define identify $\tilde{\Sigma}$ with Σ^h, treating each string in Σ^h as the h-digit base-σ representation of an integer in $\tilde{\Sigma}$. For every string $X \in \Sigma^*$, define $\tilde{X} \in \tilde{\Sigma}^*$ so that $|\tilde{X}| = \lfloor |X|/h \rfloor$ and $\tilde{X}[i] = X(h(i-1)\,..\,hi]$ for $i \in [1\,..\,|\tilde{X}|]$. Moreover, we set $\tilde{W}[1\,..\,|W|]$ so that $\tilde{W}[j] = W[j]$ for $j \in [1\,..\,|W|]$. Note that the recursive application of Proposition A.2 to $\tilde{W}$ is possible because $2 \le \tilde{\sigma}^{\tilde{b}} \le \sigma^b \le m$ and $\tilde{b} \ge 1$ hold when $h \le b$.

Data structure size It is easy to see that, for a fixed length $d \in [0\,..\,b)$, the strings B_X for $X \in \Sigma^d$ are of total length m. Across all $X \in \Sigma^{\le b-1}$, this sums up to mb, so the raw strings B_X occupy $\mathcal{O}(mb \log \sigma) = \mathcal{O}(m \log m)$ bits. The augmentation of B_X using Lemma A.1 adds $\mathcal{O}((\sigma + |B_X|) \log \sigma)$ extra bits, which sums up to $\mathcal{O}((\sigma^b + mb) \log \sigma) = \mathcal{O}(m \log m)$ bits, i.e., $\mathcal{O}(m)$ machine words. The recursion depth is $\mathcal{O}(\log_h(hb))$, so the overall size is $\mathcal{O}(m \log_h(hb))$.

Answering queries To handle any query concerning $X \in \Sigma^{\le b}$, we compute auxiliary strings $\tilde{X}$ and $X' := X[1\,..\,|X| - (|X| \bmod h)]$ (obtained by expanding the letters in $\tilde{X}$ into length-h strings).

Answering a prefix rank query $\mathsf{rank}_{W,X}(j)$, we traverse the path from $v_{X'}$ to v_X, maintaining a value r such that $r = \mathsf{rank}_{W,Y}(j)$ holds while the algorithm visits v_Y. We initialize $r := j = \mathsf{rank}_{W,\varepsilon}(j)$ if $X' = \varepsilon$ and $r := \mathsf{rank}_{\tilde{W},\tilde{X}}(j)$ (computed recursively) otherwise; this is valid due to $\mathsf{rank}_{\tilde{W},\tilde{X}}(j) = \mathsf{rank}_{W,X'}(j)$. Upon entering a node v_{Ya} from its parent v_Y, we set $r := \mathsf{rank}_{B_Y,a}(r)$ since $\mathsf{rank}_{W,Ya}(j) = \mathsf{rank}_{B_Y,a}(\mathsf{rank}_{W,Y}(j))$; see [31]. When reaching v_X, we return $r = \mathsf{rank}_{W,X}(j)$. The running time is $\mathcal{O}(h \log\log m)$ per recursive level, for a total of $\mathcal{O}(h \log\log m \cdot \log_h(hb))$.

Answering a prefix selection query $\mathsf{select}_{W,X}(r)$, we traverse the path from v_X to $v_{X'}$, maintaining a value q such that $\mathsf{select}_{W,Y}(q) = \mathsf{select}_{W,X}(r)$ holds while the algorithm visits v_Y. We initialize $q := r$ and, upon entering a node v_Y from its child v_{Ya}, we set $q := \mathsf{select}_{B_Y,a}(q)$ since $\mathsf{select}_{W,Ya}(r) = \mathsf{select}_{W,Y}(\mathsf{select}_{B_Y,a}(r))$; see [31]. When reaching $v_{X'}$, we return $q = \mathsf{select}_{W,\varepsilon}(q)$ if $X' = \varepsilon$ and $\mathsf{select}_{\tilde{W},\tilde{X}}(q)$ otherwise; this is valid due to $\mathsf{select}_{\tilde{W},\tilde{X}}(q) = \mathsf{select}_{W,X'}(q)$. The running time is $\mathcal{O}(h)$ per recursive level, for a total of $\mathcal{O}(h \cdot \log_h(hb))$.

Construction algorithm If $\log \sigma \ge \sqrt{\log m}$, we use the original wavelet tree construction algorithm [31], which takes $\mathcal{O}(mb) = \mathcal{O}(m\sqrt{\log m})$ time and $\mathcal{O}(m)$ space. Otherwise, we apply the bit-parallel algorithm of [51, 2], which has been adapted to large alphabets in [39, Lemma 6.4]. This procedure takes $\mathcal{O}(mb \log \sigma / \sqrt{\log m} + mb \log^2 \sigma / \log m) = \mathcal{O}(m\sqrt{\log m})$ time and $\mathcal{O}(m)$ space. Building the data structure of Lemma A.1 for B_X takes $\mathcal{O}(\sigma + |B_X| \log \sigma / \sqrt{\log m})$ time and $\mathcal{O}(\sigma + |B_X| \log \sigma / \log m)$ space, which sums up to $\mathcal{O}(m\sqrt{\log m})$ time and $\mathcal{O}(m)$ space across $X \in \Sigma^{\le b-1}$. Precomputing the table shared by all instances of Lemma A.1 takes $\mathcal{O}(m)$ time and space. Considering all levels of recursion, we get a multiplicative overhead of $\mathcal{O}(\log_h(hb))$. □

References

[1] Donald Adjeroh, Tim Bell, and Amar Mukherjee. *The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching*. Springer, Boston, MA, USA, 2008. doi:10.1007/978-0-387-78909-5.

[2] Maxim Babenko, Paweł Gawrychowski, Tomasz Kociumaka, and Tatiana Starikovskaya. Wavelet trees meet suffix trees. In *SODA*, pages 572–591, 2015. doi:10.1137/1.9781611973730.39.

[3] Jérémy Barbay, Francisco Claude, Travis Gagie, Gonzalo Navarro, and Yakov Nekrich. Efficient fully-compressed sequence representations. *Algorithmica*, 69(1):232–268, 2014. doi:10.1007/s00453-012-9726-3.

[4] Djamal Belazzougui. Linear time construction of compressed text indices in compact space. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 148–193. ACM, 2014. doi:10.1145/2591796.2591885.

[5] Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. Linear-time string indexing and analysis in small space. *ACM Trans. Algorithms*, 16(2):17:1–17:54, 2020. doi:10.1145/3381417.

[6] Djamal Belazzougui, Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Queries on LZ-bounded encodings. In *DCC*, pages 83–92, 2015. doi:10.1109/DCC.2015.69.

[7] Djamal Belazzougui and Gonzalo Navarro. Alphabet-independent compressed text indexing. *ACM Trans. Algorithms*, 10(4):23:1–23:19, 2014. doi:10.1145/2635816.

[8] Djamal Belazzougui and Gonzalo Navarro. Optimal lower and upper bounds for representing sequences. *ACM Trans. Algorithms*, 11(4):31:1–31:21, 2015. doi:10.1145/2629339.

[9] Djamal Belazzougui and Simon J. Puglisi. Range predecessor and Lempel-Ziv parsing. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 2053–2071. SIAM, 2016. doi:10.1137/1.9781611974331.ch143.

[10] Philip Bille, Mikko Berggren Ettienne, Inge Li Gørtz, and Hjalte Wedel Vildhøj. Time-space trade-offs for Lempel-Ziv compressed indexing. *Theor. Comput. Sci.*, 713:66–77, 2018. doi:10.1016/j.tcs.2017.12.021.

[11] Philip Bille, Inge Li Gørtz, and Frederik Rye Skjoldjensen. Deterministic indexing for packed strings. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, volume 78 of *LIPIcs*, pages 6:1–6:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.6.

[12] Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. *SIAM J. Comput.*, 44(3):513–539, 2015. doi:10.1137/130936889.

[13] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.

[14] Ho-Leung Chan, Wing-Kai Hon, Tak Wah Lam, and Kunihiko Sadakane. Compressed indexes for dynamic text collections. *ACM Trans. Algorithms*, 3(2):21, 2007. doi:10.1145/1240233.1240244.

[15] Timothy M. Chan and Mihai Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *SODA*, pages 161–173, 2010. doi:10.1137/1.9781611973075.15.

[16] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988. doi:10.1137/0217026.

[17] Anders Roy Christiansen, Mikko Berggren Ettienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Optimal-time dictionary-compressed indexes. *ACM Trans. Algorithms*, 17(1):8:1–8:39, 2021. doi:10.1145/3426473.

[18] David R. Clark. *Compact Pat Trees*. PhD thesis, University of Waterloo, 1998.

[19] Richard Cole, Tsvi Kopelowitz, and Moshe Lewenstein. Suffix trays and suffix trists: Structures for faster text indexing. *Algorithmica*, 72(2):450–466, 2015. doi:10.1007/s00453-013-9860-6.

[20] T. M. Cover and J. A. Thomas. *Elements of information theory 2nd edition*. Wiley, 2006.

[21] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, Cambridge, UK, 2007. doi:10.1017/cbo9780511546853.

[22] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November*

2000, Redondo Beach, California, USA, pages 390–398. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892127.

[23] Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.

[24] Johannes Fischer and Paweł Gawrychowski. Alphabet-dependent string searching with Wexponential search trees. In *CPM*, pages 160–171, 2015. doi:10.1007/978-3-319-19929-0_14.

[25] Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. Lz77-based self-indexing with faster pattern matching. In Alberto Pardo and Alfredo Viola, editors, *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, March 31 - April 4, 2014. Proceedings*, volume 8392 of *Lecture Notes in Computer Science*, pages 731–742. Springer, 2014. doi:10.1007/978-3-642-54423-1_63.

[26] Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. A faster grammar-based self-index. In *LATA*, pages 240–251, 2012. doi:10.1007/978-3-642-28332-1_21.

[27] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM*, 67(1):1–54, apr 2020. doi:10.1145/3375890.

[28] Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łącki, and Piotr Sankowski. Optimal dynamic strings, 2015. arXiv:1511.02612.

[29] Simon Gog. *Compressed suffix trees: design, construction, and applications*. PhD thesis, University of Ulm, 2011. URL: http://vts.uni-ulm.de/docs/2011/7786/vts_7786_11228.pdf.

[30] Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337, 2014. doi:10.1007/978-3-319-07959-2_28.

[31] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 841–850, 2003.

[32] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 397–406. ACM, 2000. doi:10.1145/335305.335351.

[33] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.*, 35(2):378–407, 2005. doi:10.1137/S0097539702402354.

[34] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.

[35] Torben Hagerup. Sorting and searching on the word RAM. In Michel Morvan, Christoph Meinel, and Daniel Krob, editors, *15th Annual Symposium on Theoretical Aspects of Computer Science, STACS 1998*, volume 1373 of *LNCS*, pages 366–398. Springer, 1998. doi:10.1007/BFb0028575.

[36] Wing-Kai Hon, Kunihiko Sadakane, and Wing-Kin Sung. Breaking a time-and-space barrier in constructing full-text indices. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 251–260. IEEE Computer Society, 2003. doi:10.1109/SFCS.2003.1238199.

[37] Guy Jacobson. Space-efficient static trees and graphs. In *FOCS*, pages 549–554, 1989. doi:10.1109/SFCS.1989.63533.

[38] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *CPM*, pages 181–192, 2001. doi:10.1007/3-540-48194-X_17.

[39] Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: Sublinear-time BWT construction and optimal LCE data structure. In *STOC*, pages 756–767, 2019. doi:10.1145/3313276.3316368.

[40] Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: String attractors. In *STOC*, pages 827–840, 2018. doi:10.1145/3188745.3188814.

[41] Tomasz Kociumaka. *Efficient Data Structures for Internal Queries in Texts*. PhD thesis, University of Warsaw, 2018.

[42] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.*, 10(3):R25, 2009.

[43] Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinform.*, 25(14):1754–1760, 2009. doi:10.1093/bioinformatics/btp324.

[44] Ruiqiang Li, Chang Yu, Yingrui Li, Tak Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, and Jun Wang. SOAP2: an improved ultrafast tool for short read alignment. *Bioinform.*, 25(15):1966–1967, 2009. doi:10.1093/bioinformatics/btp336.

[45] Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-scale algorithm design: Biological sequence analysis in the era of high-throughput sequencing*. Cambridge University Press, Cambridge, UK, 2015. doi:10.1017/cbo9781139940023.

[46] Veli Mäkinen and Gonzalo Navarro. Dynamic entropy-compressed sequences and full-text indexes. *ACM Trans. Algorithms*, 4(3):32:1–32:38, 2008. doi:10.1145/1367064.1367072.

[47] Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993. doi:10.1137/0222058.

[48] J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Space-efficient construction of compressed indexes in deterministic linear time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 408–424. SIAM, 2017. doi:10.1137/1.9781611974782.26.

[49] J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Fast compressed self-indexes with deterministic linear-time construction. *Algorithmica*, 82(2):316–337, 2020. doi:10.1007/s00453-019-00637-x.

[50] J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Text indexing and searching in sublinear time. In *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, June 17-19, 2020, Copenhagen, Denmark*, volume 161 of *LIPIcs*, pages 24:1–24:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CPM.2020.24.

[51] J. Ian Munro, Yakov Nekrich, and Jeffrey Scott Vitter. Fast construction of wavelet trees. *Theor. Comput. Sci.*, 638:91–97, 2016. doi:10.1016/j.tcs.2015.11.011.

[52] Gonzalo Navarro. *Compact data structures: A practical approach*. Cambridge University Press, Cambridge, UK, 2016. doi:10.1017/cbo9781316588284.

[53] Gonzalo Navarro. Indexing highly repetitive string collections, part I: Repetitiveness measures. *ACM Computing Surveys*, 54(2), 2021. doi:10.1145/3434399.

[54] Gonzalo Navarro. Indexing highly repetitive string collections, part II: Compressed indexes. *ACM Computing Surveys*, 54(2), 2021. doi:10.1145/3432999.

[55] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1):2, 2007. doi:10.1145/1216370.1216372.

[56] Gonzalo Navarro and Yakov Nekrich. Time-optimal top-k document retrieval. *SIAM J. Comput.*, 46(1):80–113, 2017. doi:10.1137/140998949.

[57] Gonzalo Navarro and Nicola Prezza. Universal compressed text indexing. *Theor. Comput. Sci.*, 762:41–50, 2019. doi:10.1016/j.tcs.2018.09.007.

[58] Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Dynamic index and LZ factorization in compressed space. *Discret. Appl. Math.*, 274:116–129, 2020. doi:10.1016/j.dam.2019.01.014.

[59] Enno Ohlebusch. *Bioinformatics algorithms: Sequence analysis, genome rearrangements, and phylogenetic reconstruction*. Oldenbusch Verlag, Ulm, Germany, 2013.

[60] Mihai Patrascu. Lower bounds for 2-dimensional range counting. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 40–46. ACM, 2007. doi:10.1145/1250790.1250797.

[61] Milan Rŭzić. Constructing efficient dictionaries in close to sorting time. In *ICALP*, pages 84–95, 2008. doi:10.1007/978-3-540-70575-8_8.

[62] Kunihiko Sadakane. Succinct representations of lcp information and improvements in the compressed suffix arrays. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 225–232. ACM/SIAM, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545410.

[63] Kunihiko Sadakane. Compressed suffix trees with full functionality. *Theory Comput. Syst.*, 41(4):589–607, 2007. doi:10.1007/s00224-006-1198-x.

[64] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *Trans. Inf. Theory*, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.