

Dynamic Suffix Array with Polylogarithmic Queries and Updates

Dominik Kempa
Stony Brook University
kempa@cs.stonybrook.edu

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

Abstract

The suffix array $\text{SA}[1..n]$ of a text T of length n is a permutation of $\{1, \dots, n\}$ describing the lexicographical ordering of suffixes of T , and it is considered to be among of the most important data structures in string algorithms, with dozens of applications in data compression, bioinformatics, and information retrieval. One of the biggest drawbacks of the suffix array is that it is very difficult to maintain under text updates: even a single character substitution can completely change the contents of the suffix array. Thus, the suffix array of a dynamic text is modelled using *suffix array queries*, which return the value $\text{SA}[i]$ given any $i \in [1..n]$.

Prior to this work, the fastest dynamic suffix array implementations were by Amir and Boneh. At ISAAC 2020, they showed how to answer suffix array queries in $\tilde{O}(k)$ time, where $k \in [1..n]$ is a trade-off parameter, with $\tilde{O}(\frac{n}{k})$ -time text updates. In a very recent preprint [arXiv, 2021], they also provided a solution with $\mathcal{O}(\log^5 n)$ -time queries and $\tilde{O}(n^{2/3})$ -time updates.

We propose the first data structure that supports both suffix array queries and text updates in $\mathcal{O}(\text{polylog } n)$ time (achieving $\mathcal{O}(\log^4 n)$ and $\mathcal{O}(\log^{3+o(1)} n)$ time, respectively). Our data structure is deterministic and the running times for all operations are worst-case. In addition to the standard single-character edits (character insertions, deletions, and substitutions), we support (also in $\mathcal{O}(\log^{3+o(1)} n)$ time) the “cut-paste” operation that moves any (arbitrarily long) substring of T to any place in T . To achieve our result, we develop a number of new techniques which are of independent interest. This includes a new flavor of dynamic locally consistent parsing, as well as a dynamic construction of string synchronizing sets with an extra *local sparsity* property; this significantly generalizes the sampling technique introduced at STOC 2019. We complement our structure by a hardness result: unless the Online Matrix-Vector Multiplication (OMv) Conjecture fails, no data structure with $\mathcal{O}(\text{polylog } n)$ -time suffix array queries can support the “copy-paste” operation in $\mathcal{O}(n^{1-\varepsilon})$ time for any $\varepsilon > 0$.

1 Introduction

For a text T of length n , the suffix array $\text{SA}[1..n]$ stores the permutation of $\{1, \dots, n\}$ such that $T[\text{SA}[i]..n]$ is the i th lexicographically smallest suffix of T . The original application of SA [MM93] was to solve the *text indexing* problem: construct a data structure such that, given a pattern $P[1..m]$ (typically with $m \ll n$), can quickly count (and optionally list) all occurrences of P in T . Since the sought set of positions occupies a contiguous block $\text{SA}[b..e]$ (for some $b, e \in [1..n+1]$) and since, given $j \in [1..n]$, we can in $\mathcal{O}(m)$ time check if the value $\text{SA}[j]$ is before, inside, or after this block, the indices b and e can be computed in $\mathcal{O}(m \log n)$ time with binary search. If $b < e$, we can then report all occurrences of P in T at the extra cost of $\mathcal{O}(e - b)$. Soon after the discovery of SA it was

*Supported by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship.

realized that a very large set of problems on strings is essentially solved (or at least becomes much easier) once we have a suffix array (often augmented with the *LCP array* [MM93, KLA⁺01]). The textbook of Gusfield [Gus97] lists many such problems including:

- finding repeats (e.g., MAXIMALREPEATS, LONGESTREPEATEDFACTOR, TANDEMREPEATS);
- computing special subwords (e.g., MINIMALABSENTWORD, SHORTESTUNIQUESUBSTRING);
- sequence comparisons (e.g., LONGESTCOMMONSUBSTRING, MAXIMALUNIQUEMATCHES); and
- data compression (e.g., LZ77FACTORIZATION, CDAWGCOMPRESSION).

This trend continues in more recent textbooks [Oh13, MBCT15, Nav16], with the latest suffix array representations (such as *FM-index* [FM05], *compressed suffix array* [GV05], and *r-index* [GNP20]) as central data structures. There are even textbooks such as [ABM08] solely dedicated to the applications of suffix arrays and the closely related Burrows–Wheeler transform (BWT) [BW94].

The power of suffix array comes with one caveat: It is very difficult to maintain it for a text undergoing updates. For example, for $T = \mathbf{b}^n$ (symbol \mathbf{b} repeated n times) we have $\text{SA}_T = [n, \dots, 2, 1]$, whereas for $T' = \mathbf{b}^{n-1}\mathbf{c}$ (obtained from T with a single substitution), it holds $\text{SA}_{T'} = [1, 2, \dots, n]$, i.e., the complete reversal. Even worse, if $n = 2m + 1$ and $T'' = \mathbf{b}^m\mathbf{a}\mathbf{b}^m$ (again, a single substitution), then $\text{SA}_{T''} = [m+1, n, m, n-1, m-1, \dots, m+2, 1]$, i.e., the near-perfect interleaving of two halves of SA_T . In general, even a single character substitution may permute SA in a very complex manner. Thus, if one wishes to maintain the suffix array of a dynamic text, its entries cannot be stored in plain form but must be obtained by querying a data structure. The quest for such *dynamic suffix array* is open since the birth of suffix array nearly three decades ago. We thus pose our problem:

Problem 1.1. *Can we support efficient SA queries for a dynamically changing text?*

Previous Work One of the first attempts to tackle the above problem is due to Salson, Lecroq, Léonard, and Mouchard [SLLM10]. Their dynamic suffix array achieves good practical performance on real-world texts (including English text and DNA sequences), but its update takes $\Theta(n)$ in the worst-case. A decade later, Amir and Boneh [AB20] proposed a structure that, for any parameter $k \in [1..n]$, supports SA and SA^{-1} queries in $\tilde{O}(k)$ time and character substitutions in $\tilde{O}(\frac{n}{k})$ time. This implies the first nontrivial trade-off for the dynamic suffix array, e.g., $\tilde{O}(\sqrt{n})$ -time operations. Very recently, Amir and Boneh [AB21] described a dynamic suffix array that supports updates (insertions and deletions) in the text in $\tilde{O}(n^{2/3})$ time and SA queries in $\mathcal{O}(\log^5 n)$ time. They also gave a structure that supports substitutions in $\tilde{O}(n^{1/2})$ time and SA^{-1} queries in $\mathcal{O}(\log^4 n)$ time.

A separate line of research focused on the related problem of *dynamic text indexing* introduced by Gu, Farach, and Beigel [GFB94]. This problem aims to design a data structure that permits updates to the text T and pattern searches (asking for all occurrences of a given pattern P in T). As noted in [AB20], the solution in [GFB94] achieves $\tilde{O}(\sqrt{n})$ -time updates to text and $\tilde{O}(m\sqrt{n} + \text{occ} \log n)$ pattern search query (where occ is the number of occurrences of P in T). Sahinalp and Vishkin [SV96] then proposed a solution based on the idea of *locally consistent parsing* that achieves $\mathcal{O}(\log^3 n)$ -time update and $\mathcal{O}(m + \text{occ} + \log n)$ pattern searching time. The update time was later improved by Alstrup, Brodal, and Rauhe [ABR00] to $\mathcal{O}(\log^2 n \log \log n \log^* n)$ at the expense of the slightly slower query $\mathcal{O}(m + \text{occ} + \log n \log \log n)$. This last result was achieved by building on techniques for the *dynamic string equality* problem proposed by Mehlhorn, Sundar, and Uhrig [MSU97]. This was improved in [GKK⁺15] to $\mathcal{O}(\log^2 n)$ -time update and $\mathcal{O}(m + \text{occ})$ -time search. A slightly different approach to dynamic text indexing, based on *dynamic position heaps* was proposed in [EMOW11]. It achieves $\mathcal{O}(m \log n + \text{occ})$ amortized search, but its update take $\Theta(n)$ time in the worst case. There is

also work on *dynamic compressed text indexing*. Chan, Hon, Lam, and Sadakane [CHLS07] proposed an index that uses $\mathcal{O}(\frac{1}{\varepsilon}(nH_0(T)+n))$ bits of space (where $H_0(T)$ is the zeroth order empirical entropy of T), searches in $\mathcal{O}(m \log^2 n (\log^\varepsilon n + \log \sigma) + \text{occ} \log^{1+\varepsilon} n)$ time (where σ is the alphabet size), and performs updates in $\mathcal{O}(\sqrt{n} \log^{2+\varepsilon} n)$ amortized time, where $0 < \varepsilon \leq 1$. Recently, Nishimoto, I, Inenaga, Bannai, and Takeda [NII⁺20] proposed a faster index for a text T with LZ77 factorization of size z . Assuming for simplicity $z \log n \log^* n = \mathcal{O}(n)$, their index occupies $\mathcal{O}(z \log n \log^* n)$ space, performs updates in amortized $\mathcal{O}((\log n \log^* n)^2 \log z)$ time (in addition to edits, they also support the “cut-paste” operation that moves a substring of T from one place to another), and searches in $\mathcal{O}\left(m \min\left\{\frac{\log \log n \log \log z}{\log \log \log n}, \sqrt{\frac{\log z}{\log \log z}}\right\} + \log z \log m \log^* n (\log n + \log m \log^* n) + \text{occ} \log n\right)$ time.

We point out that although the (compressed) dynamic text indexing problem [GFB94, NII⁺20] discussed above is related to dynamic suffix array, it is not the same. Assuming one accepts additional log factors, the dynamic suffix array problem is *strictly harder*: it solves dynamic indexing (by simply adding binary search on top), but no converse reduction is known; such a reduction would compute values of SA in $\mathcal{O}(\text{polylog } n)$ time using searches for short patterns. Thus, the many applications of SA listed above cannot be solved with these indexes. Unfortunately, due to lack of techniques, the dynamic suffix array problem has seen very little progress; as noted by Amir and Boneh [AB20], “(...) although a dynamic suffix array algorithm would be extremely useful to automatically adapt many static pattern matching algorithms to a dynamic setting, other techniques had to be sought”. They remark, however, that “Throughout all this time, an algorithm for maintaining the suffix tree or suffix array of a dynamically changing text had been sought”. To sum up, until now, the best dynamic suffix arrays have been those of [AB20], taking $\tilde{\mathcal{O}}(k)$ time to answer SA and SA^{-1} queries and $\tilde{\mathcal{O}}(\frac{n}{k})$ time for substitutions, and [AB21], taking $\tilde{\mathcal{O}}(n^{2/3})$ time for insertions/deletions and $\mathcal{O}(\log^5 n)$ time for SA queries, or $\tilde{\mathcal{O}}(n^{1/2})$ time for substitutions and $\mathcal{O}(\log^4 n)$ time for SA^{-1} queries. No solution with polylog n -time queries and updates (even amortized or expected) was known, not even for character substitutions only.

Our Results We propose the first dynamic suffix array with all operations (queries *and* updates) taking only $\mathcal{O}(\text{polylog } n)$ time. Our data structure is deterministic and the complexities of both queries and updates are worst-case. Thus, we leap directly to a solution satisfying the commonly desired properties on the query and update complexity for this almost thirty-years old open problem. In addition to single-character edits, our structure supports the powerful “cut-paste” operation, matching the functionality of state-of-the-art indexes [ABR00, NII⁺20]. More precisely, our structure supports the following operations (the bounds below are simplified overestimates; see Theorem 10.16):

- We support SA queries (given $i \in [1..n]$, return $\text{SA}[i]$) in $\mathcal{O}(\log^4 N)$ time.
- We support SA^{-1} queries (given $j \in [1..n]$, return $\text{SA}^{-1}[j]$) in $\mathcal{O}(\log^5 N)$ time.
- We support updates (insertion and deletion of a single symbol in T as well as the “cut-paste” operation, moving any block of T to any other place in T) in $\mathcal{O}(\log^3 N (\log \log N)^2)$ time.

Here, $N = n\sigma$ is the product of the current text length and the size of the alphabet $\Sigma = [0.. \sigma)$.

The above result may leave a sense of incompleteness regarding further updates such as “copy-paste”. We show that, despite its similarity with “cut-paste”, supporting this operation in the dynamic setting is most likely very costly. More precisely, we prove that, unless the Online Matrix-Vector Multiplication (OMv) Conjecture [HKNS15] fails, no data structure that supports SA or SA^{-1} queries in $\mathcal{O}(\text{polylog } n)$ time can support the “copy-paste” operation in $\mathcal{O}(n^{1-\varepsilon})$ time for any $\varepsilon > 0$.¹

¹In fact, we prove a more general result that the *product* of time needed to support SA/ SA^{-1} queries and copy-

Technical Contributions To achieve our result, we develop new techniques and significantly generalize several existing ones. Our first novel technique is the notion of *locally sparse* synchronizing sets. String synchronizing sets [KK19] have recently been introduced in the context of efficient construction of BWT and LCE queries for “packed strings” (where a single machine word stores multiple characters). Since then, they have found many other applications [ACI⁺19, CKPR21, KK20, KK21, AJ22]. Loosely speaking (a formal definition follows in Section 2), for any fixed $\tau \geq 1$, a set $S \subseteq [1..n]$ is a τ -synchronizing set of string $T \in \Sigma^n$ if S samples positions of T *consistently* (i.e., whether $i \in [1..n]$ is sampled depends only on the length- $\Theta(\tau)$ context of i in T) and does not sample positions inside highly periodic fragments (the so-called *density* condition). In all prior applications utilizing synchronizing sets, the goal is to ensure that S is *sparse on average*, i.e., that the size $|S|$ is minimized. In this paper, we prove that at the cost of increasing the size of $|S|$ by a mere factor $\mathcal{O}(\log^*(\tau\sigma))$, we can additionally ensure that S is also *locally sparse* (see Lemma 9.3). We then show that such S can be maintained dynamically using a new construction of S from the signature parsing (a technique introduced in [MSU97] and used, for example, in [ABR00, NII⁺20]). The crucial benefit of local sparsity is that any auxiliary structure that depends on the length- $\Theta(\tau)$ contexts of positions in S , including S itself, can be updated efficiently. Another result of independent interest is the first dynamic construction of string synchronizing sets. For this, we internally represent some substrings of T using the abstraction of *dynamic strings* [ABR00, SV94, MSU97, GKK⁺18]. The problem with all existing variants of dynamic strings, however, is that they rely on representing the strings using a hierarchical representation whose only goal is to ensure the string shrinks by a constant factor at each level. This, however, is not sufficient for our purpose: in order to satisfy the density condition for the resulting synchronizing set, we also need the expansions of all symbols at any given level to have some common upper bound on the expansion length. The notion of such “balanced” parsing is known [SV94, BGP20], but has only been implemented in static settings. We show the first variant of dynamic strings that maintains a “balanced” parsing at every level, and consequently lets us dynamically maintain a locally sparse string synchronizing set.

The mainstay among data structures for pattern matching or SA queries is the use of (typically 2D) orthogonal range searching [Cha88]. Example indexes include nearly all indexes based on LZ77 [ANS12, BGG⁺15, BEGV18, CEK⁺21, GGK⁺14, Kär99, KN13, NII⁺20], context-free grammars [BLR⁺15, CN11, CNP21, GGK⁺12, MNKS13, TTS14, TKN⁺20], and some recent BWT-based indexes [KK21, MNN20, CHS⁺15]. Nearly all these structures maintain a set of points corresponding to some set of suffixes of T (identified with their starting positions $P \subseteq [1..n]$) ordered lexicographically. The problem with adapting this to the dynamic setting is that once we modify T near the end, the order of (potentially all) elements in P changes. We overcome this obstacle as follows. Rather than on sampling of suffixes, we rely on $\log n$ levels of sampling of *substrings* (identified by the set S_k of the starting positions of their occurrences) of length roughly 2^k , where $k \in [0.. \lfloor \log n \rfloor]$ implemented using dynamic locally sparse synchronizing sets. With such structure, we can efficiently update the sets S_k and the associated geometric structures, but we lose the ability to compare suffixes among each other. In Section 6, we show that even with such partial sampling, we can still implement SA queries. In $\log n$ steps, our query algorithm successively narrows the range of SA to contain only suffixes prefixed with $T[\text{SA}[i].. \text{SA}[i] + \ell)$, while also maintaining the starting position of some arbitrary suffix in the range, where $\ell = 2^k$ is the current prefix length. In the technical

paste operation cannot be within $\mathcal{O}(n^{1-\varepsilon})$ for any $\varepsilon > 0$. Thus, the trade-off similar to the one achieved by Amir and Boneh [AB20] ($\tilde{O}(\frac{n}{k})$ -time query and $\tilde{O}(k)$ -time update) may still be possible for a dynamic suffix array with copy-pastes; we leave proving such upper bound as a possible direction for future work.

overview, we sketch the main ideas of this reduction, but we remark that the details of this approach are nontrivial and require multiple technical results (see, e.g., Section 6.3.2–6.3.8).

Finally, we remark that, even though (as noted earlier), dynamic suffix array is a strictly harder problem than text indexing (since one can be reduced to the other, but not the other way around), our result has implications even for the text indexing problem. The existing dynamic text indexes with fast queries and updates (such as [GKK⁺15, NII⁺20]) can only *list* all k occurrences of any pattern. One however, cannot obtain the number of occurrences (which is the standard operation supported by most of the static indexes [NM07]) in time that is always bounded by $\mathcal{O}(m \text{ polylog } n)$ (since k can be arbitrarily large).² Our dynamic suffix array, on the other hand, implements counting seamlessly: it suffices to perform the standard binary search [MM93] over SA for the pattern P , resulting in the endpoints of range $\text{SA}[b..e)$ of suffixes of T prefixed with P , and return $e - b$.

Related Work Chan, Hon, Lam, and Sadakane [CHLS07] introduced a problem of *indexing text collections*, which asks to store a dynamically changing collection \mathcal{C} of strings (under insertions and deletions of *entire strings*) so that pattern matching queries on \mathcal{C} can be answered efficiently. Although the resulting data structures are also called *dynamic full-text indexes* [MN08] or *dynamic suffix trees* [RNO08], we remark that they are solving a different problem than what, by analogy to dynamic suffix array, we would mean by “dynamic suffix tree”. Observe that we cannot simulate the insertion of a symbol in the middle of T using a collection of strings. Maintaining symbols of T as a collection of length-1 strings will not work because the algorithms in [CHLS07, MN08, RNO08] only report occurrences entirely inside one of the strings. Since the insertion of a string S of length m into \mathcal{C} takes $\Omega(m)$ time in [CHLS07, MN08, RNO08] (similarly for deletion), one also cannot efficiently use \mathcal{C} to represent the entire text T as a single element of \mathcal{C} .

Tangentially related to the problem of text indexing is the problem of *sequence representation* [NN14], where we store a string $S[1..n]$ under character insertions and deletions and support the access to S , $\text{rank}(i, c)$ (returning $|\{j \in [1..i] : S[j] = c\}|$) and $\text{select}(i, c)$ (returning the i th occurrence of c in S). A long line of research, including [MN08, GN09, HM10, NS14], culminated with the work of Navarro and Nekrich, who achieved $\mathcal{O}(\log n / \log \log n)$ time for all operations (amortized for updates), while using space close to $nH_0(S)$, where H_0 is the zeroth order empirical entropy.

Finally, there is a line of research focusing on storing a text T under updates so that we can support efficient *longest common extension* queries $\text{LCE}_T(i, j)$ that return the length of the longest common prefix of $T[i..|T|]$ and $T[j..|T|]$. The research was initiated with the seminal work in [MSU97, ABR00] (recently improved in [GKK⁺18]). More recently, a solution working in the compressed space (achieving similar runtimes as the index in [NII⁺20]) was proposed in [NII⁺16].

2 Preliminaries

A *string* is a finite sequence of characters from some set Σ called the *alphabet*. We denote the length of a string S as $|S|$. For any index $i \in [1..|S|]$,³ we denote the i th character of S as $S[i]$. A string of the form $S[i..j) = S[i]S[i+1] \dots S[j-1]$, where $i \leq i \leq j \leq |S| + 1$ is called a *fragment* of S . If S is known, we will encode $S[i..j)$ in $\mathcal{O}(1)$ space as a pair (i, j) . *Prefixes* and *suffixes* are of the

²Efficient counting for indexes relying on orthogonal range searching is a technically challenging problem that has been solved only recently for the static case [CEK⁺21]. It is possible that these ideas can be combined with [NII⁺20], but the result will nevertheless be significantly more complicated than counting using the suffix array.

³For $i, j \in \mathbb{Z}$, denote $[i..j) = \{k \in \mathbb{Z} : i \leq k < j\}$, $[i..j] = \{k \in \mathbb{Z} : i \leq k \leq j\}$, and $(i..j) = \{k \in \mathbb{Z} : i < k \leq j\}$.

form $S[1..j]$ and $S[i..|S|]$, respectively. By \bar{S} we denote the *reverse* of S , i.e., $\bar{S} = S[|S|] \dots S[2][1]$. The *concatenation* of two strings U and V is denoted UV or $U \cdot V$. Moreover, $S^k = \bigodot_{i=1}^k S$ is the concatenation of k copies of S ; note that $S^0 = \varepsilon$ is the *empty string*. An integer $p \in [1..|S|]$ is called a *period* of S if $S[1..|S| - p] = S[1 + p..|S|]$; we denote the shortest period of S as $\text{per}(S)$. 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 $i \in [1.. \min(|U|, |V|)]$.

Throughout, we consider a string (called the *text*) T of length $n \geq 1$ over an integer alphabet $\Sigma = [0..\sigma]$. We assume that $T[n] = \$$ (where $\$ = \min \Sigma$) and $\$$ does not occur in $T[1..n]$.

The *suffix array* $\text{SA}[1..n]$ of T is a permutation of $[1..n]$ such that $T[\text{SA}[1]..n] \prec T[\text{SA}[2]..n] \prec \dots \prec T[\text{SA}[n]..n]$, i.e., $\text{SA}[i]$ is the starting position of the lexicographically i th suffix of T ; see Fig. 1 for an example. The *inverse suffix array* $\text{ISA}[1..n]$ is the inverse permutation of SA , i.e., $\text{ISA}[j] = i$ holds if and only if $\text{SA}[i] = j$. By $\text{lcp}(U, V)$ (resp. $\text{lcs}(U, V)$) we denote the length of the longest common prefix (resp. suffix) of U and V . For $j_1, j_2 \in [1..n]$, we let $\text{LCE}_T(j_1, j_2) = \text{lcp}(T[j_1..], T[j_2..])$.

The *rotation* operation $\text{rot}(\cdot)$, given a string $S \in \Sigma^+$, moves the last character of S to the front so that $\text{rot}(S) = S[|S|] \cdot S[1..|S| - 1]$. The inverse operation $\text{rot}^{-1}(\cdot)$ moves the first character of S to the back so that $\text{rot}^{-1}(S) = S[2..|S|] \cdot S[1]$. For an integer $s \in \mathbb{Z}$, the operation $\text{rot}^s(\cdot)$ denotes the $|s|$ -time composition of $\text{rot}(\cdot)$ (if $s \geq 0$) or $\text{rot}^{-1}(\cdot)$ (if $s < 0$). Strings S, S' are *cyclically equivalent* if $S' = \text{rot}^s(S)$ for some $s \in \mathbb{Z}$.

We use the word RAM model of computation [Hag98] with w -bit *machine words*, where $w \geq \log n$.

Definition 2.1 (τ -synchronizing set [KK19]). Let $T \in \Sigma^n$ be a string and let $\tau \in [1.. \lfloor \frac{n}{2} \rfloor]$ be a parameter. A set $S \subseteq [1..n - 2\tau + 1]$ 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 $i \in S$ holds if and only if $j \in S$ (for $i, j \in [1..n - 2\tau + 1]$),
2. $S \cap [i..i + \tau] = \emptyset$ if and only if $i \in R(\tau, T)$ (for $i \in [1..n - 3\tau + 2]$), where

$$R(\tau, T) := \{i \in [1..|T| - 3\tau + 2] : \text{per}(T[i..i + 3\tau - 2]) \leq \frac{1}{3}\tau\}.$$

In most applications, we want to minimize $|S|$. The density condition imposes a lower bound $|S| = \Omega(\frac{n}{\tau})$ for strings of length $n \geq 3\tau - 1$ that do not contain highly periodic substrings of length $3\tau - 1$. Hence, in the worst case, we cannot hope to improve upon the following bound.

Theorem 2.2 ([KK19, Proposition 8.10]). *For every $T \in \Sigma^n$ and $\tau \in [1.. \lfloor \frac{n}{2} \rfloor]$, there exists a τ -synchronizing set S of size $|S| = \mathcal{O}(\frac{n}{\tau})$. Such S can be deterministically constructed in $\mathcal{O}(n)$ time.*

3 Technical Overview

We derive our dynamic suffix array gradually. We start (Sections 4 and 5), by introducing the auxiliary tools. In the first part of the paper (Section 6) we prove that if we have $\Theta(\log n)$ synchronizing sets

i	$\text{SA}[i]$	$T[\text{SA}[i]..n]$
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	ababababaababa
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 all sorted suffixes of $T = \text{bbabaababababaababa}$ along with the suffix array.

for values of τ spanning across the whole range $[1..n]$, and we can support some set of queries (stated as “assumptions”) concerning either positions in those synchronizing sets, gaps across successive elements, or their length- $\Theta(\tau)$ contexts in T , then we can support SA queries on T . In the second part of the paper (Sections 7 to 10) we then show how to satisfy these assumptions for text supporting update operations. This approach lets us separate the clean (combinatorial) details concerning SA queries from (more algorithmic and technical) details concerning the maintenance of the underlying synchronizing sets and the associated structures. This also lets us for now treat T as well as each synchronizing sets as static, since the reduction works for *any* collection of such sets, and thus if after the update these sets (and the associated structures) change, the query algorithm is unaffected.

We start with an overview of Section 6. Let $\ell \geq 1$. For any $j \in [1..n]$, we define

$$\begin{aligned} \text{Occ}_\ell(j) &= \{j' \in [1..n] : T^\infty[j'..j'+\ell] = T^\infty[j..j+\ell]\}, \\ \text{RangeBeg}_\ell(j) &= |\{j' \in [1..n] : T[j'..n] \prec T[j..n] \text{ and } \text{LCE}_T(j', j) < \ell\}|, \text{ and} \\ \text{RangeEnd}_\ell(j) &= \text{RangeBeg}_\ell(j) + |\text{Occ}_\ell(j)|. \end{aligned}$$

For the motivation of the above names, note that viewing $P := T^\infty[j..j+\ell]$ as a pattern, we have $\{\text{SA}[i] : i \in (\text{RangeBeg}_\ell(j) .. \text{RangeEnd}_\ell(j))\} = \{i \in [1..n] : T^\infty[i..i+|P|] = P\}$.

Moreover, for any $j \in [1..n]$, we define

$$\begin{aligned} \text{Pos}_\ell(j) &= \{j' \in [1..n] : T[j'..n] \prec T[j..n] \text{ and } \text{LCE}_T(j', j) \in [\ell..2\ell]\} \text{ and} \\ \text{Pos}'_\ell(j) &= \{j' \in [1..n] : T[j'..n] \succ T[j..n] \text{ and } \text{LCE}_T(j', j) \in [\ell..2\ell]\}. \end{aligned}$$

We denote the size of $\text{Pos}_\ell(j)$ and $\text{Pos}'_\ell(j)$ as $\delta_\ell(j) = |\text{Pos}_\ell(j)|$ and $\delta'_\ell(j) = |\text{Pos}'_\ell(j)|$. Note that by definition, $(\text{RangeBeg}_{2\ell}(j), \text{RangeEnd}_{2\ell}(j)) = (\text{RangeBeg}_\ell(j) + \delta_\ell(j), \text{RangeBeg}_\ell(j) + \delta_\ell(j) + |\text{Occ}_{2\ell}(j)|)$ and $(\text{RangeBeg}'_{2\ell}(j), \text{RangeEnd}'_{2\ell}(j)) = (\text{RangeEnd}_\ell(j) - \delta'_\ell(j) - |\text{Occ}_{2\ell}(j)|, \text{RangeEnd}_\ell(j) - \delta'_\ell(j))$.

The main idea of our algorithm is as follows. Suppose that we have obtained some $j \in \text{Occ}_{16}(\text{SA}[i])$ and $(\text{RangeBeg}_{16}(\text{SA}[i]), \text{RangeEnd}_{16}(\text{SA}[i]))$ (Assumption 6.1). Then, for $q = 4, \dots, \lceil \log n \rceil - 1$, denoting $\ell = 2^q$, we compute $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$ and some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$, by using as input some position $j \in \text{Occ}_\ell(\text{SA}[i])$ and the pair $(\text{RangeBeg}_\ell(\text{SA}[i]), \text{RangeEnd}_\ell(\text{SA}[i]))$, i.e., the output of the earlier step. This lets us compute $\text{SA}[i]$, since eventually we obtain some $j' \in \text{Occ}_{2^{\lceil \log n \rceil}}(\text{SA}[i])$, and for any $k \geq n$, $\text{Occ}_k(\text{SA}[i]) = \{\text{SA}[i]\}$, i.e., we must have $j' = \text{SA}[i]$.

Our goal is to show how to implement a single step of the above process. Let $\ell \in [16..n]$ and $i \in [1..n]$. Suppose that we are given $(\text{RangeBeg}_\ell(\text{SA}[i]), \text{RangeEnd}_\ell(\text{SA}[i]))$ and some $j \in \text{Occ}_\ell(\text{SA}[i])$ as input. We aim to show that under specific assumptions about the ability to perform some queries, we can compute some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ and the pair $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$.

Let $\tau := \lfloor \frac{\ell}{3} \rfloor$. Our algorithm works differently, depending on whether it holds $\text{SA}[i] \in \text{R}(\tau, T)$ or $\text{SA}[i] \in [1..n] \setminus \text{R}(\tau, T)$. Thus, we first need to efficiently implement this check. Observe that whether or not it holds $j \in \text{R}(\tau, T)$ depends only on $T[j..j+3\tau-1]$. Therefore, by $3\tau-1 \leq \ell$, if $j \in \text{Occ}_\ell(\text{SA}[i])$ then $\text{SA}[i] \in \text{R}(\tau, T)$ holds if and only if $j \in \text{R}(\tau, T)$. Consequently, given any $j \in \text{Occ}_\ell(\text{SA}[i])$, using Assumption 6.3 we can efficiently check if $\text{SA}[i] \in \text{R}(\tau, T)$ (such $\text{SA}[i]$ is called *periodic*) or $\text{SA}[i] \in [1..n] \setminus \text{R}(\tau, T)$ (i.e., the position $\text{SA}[i]$ is *nonperiodic*).

The Nonperiodic Positions Assume $\text{SA}[i] \in [1..n] \setminus \text{R}(\tau, T)$. We proceed in two steps. First, we show how to compute $|\text{Pos}_\ell(\text{SA}[i])|$ and $|\text{Occ}_{2\ell}(\text{SA}[i])|$ assuming we have some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$. By the earlier observation about $\delta(\text{SA}[i]) = |\text{Pos}_\ell(\text{SA}[i])|$ and the definition of $\text{Occ}_{2\ell}(\text{SA}[i])$, this gives us the pair $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$. We then explain how to find j' .

Let $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$. Observe that by $T^\infty[j'..j'+2\ell] = T[\text{SA}[i].. \text{SA}[i] + 2\ell]$, we have $|\text{Pos}_\ell(\text{SA}[i])| = |\text{Pos}_\ell(j')|$, $|\text{Occ}_{2\ell}(\text{SA}[i])| = |\text{Occ}_{2\ell}(j')|$, and $j' \notin \text{R}(\tau, T)$. We can thus focus on computing $|\text{Pos}_\ell(j')|$ and $|\text{Occ}_{2\ell}(j')|$. Let \mathbf{S} be any τ -synchronizing set of T . First, observe that by $j' \notin \text{R}(\tau, T)$, the position $s' = \text{succ}_5(j')$ satisfies $s' - j' < \tau$. Thus, by the consistency of \mathbf{S} and $3\tau \leq \ell$, all $j'' \in \text{Pos}_\ell(j')$ share a common offset $\delta_5 = s' - j'$ such that $j'' + \delta_5 = \min(\mathbf{S} \cap [j''..j'' + \tau])$ and hence the relative lexicographical order between $T[j''..n]$ and $T[j'..n]$ is the same as between $T[j'' + \delta_5..n]$ and $T[j' + \delta_5..n]$. Hence, to compute $|\text{Pos}_\ell(j')|$ it suffices to count $s'' \in \mathbf{S}$ that are:

1. Preceded in T by the string $T[j'..s']$, and
2. For which it holds $T[s''..n] \prec T[s'..n]$ and $\text{LCE}_T(s'', s') \in [\ell - \delta_5..2\ell - \delta_5]$.

Observe, that for any $q \geq 2\ell$, Condition 1 is equivalent to position s'' having a reversed left length- q context in the lexicographical range $[X..X']$, where $\bar{X} = T[j'..s']$ and $X' = Xc^\infty$ (where $c = \max \Sigma$), and Condition 2 is equivalent to position s'' having a right length- q context in $[Y..Y']$, where $Y = T^\infty[s'..j' + \ell]$ and $Y' = T^\infty[s'..j' + 2\ell]$ (Lemma 6.7). Consequently, the only queries needed to compute $|\text{Pos}_\ell(j')|$ are $\text{succ}_5(j')$ and generalized range queries on a set of points $\mathcal{P} = \{(T^\infty[s' - q..s'], T^\infty[s'..s' + q]) : s' \in \mathbf{S}\}$. Since $\tau = \lfloor \frac{\ell}{3} \rfloor$ and $\ell \geq 16$, it suffices to choose $q = 7\tau$ to satisfy $q \geq 2\ell$. Thus, under Assumption 6.5, we can efficiently compute $|\text{Pos}_\ell(j')| = |\text{Pos}_\ell(\text{SA}[i])|$.

The intuition for $|\text{Occ}_{2\ell}(j')|$ is similar, except we observe that Condition 2 is that s'' satisfies $T^\infty[s''..j'' + 2\ell] = T^\infty[s'..j' + 2\ell]$, which is equivalent to s'' having a right length- q context in $[Y'..Y'c^\infty]$. Thus, we can also count such s'' (and consequently, compute $|\text{Occ}_{2\ell}(\text{SA}[i])|$) using \mathcal{P} .

The above reductions are proved in Lemmas 6.10 and 6.11 and lead to the following result.

Proposition 3.1. *Under Assumption 6.5, assuming $\text{SA}[i] \in [1..n] \setminus \text{R}(\tau, T)$, given a position $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$, we can efficiently compute $|\text{Pos}_\ell(\text{SA}[i])|$ and $|\text{Occ}_{2\ell}(\text{SA}[i])|$.*

It remains to show how to find some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$. For this, observe that if we sort all $j'' \in \text{Occ}_\ell(\text{SA}[i])$ by their right length- 2ℓ context in T^∞ then for the k th position j'' in this order we have $T^\infty[j''..j'' + 2\ell] = T^\infty[\text{SA}[b+k].. \text{SA}[b+k] + 2\ell]$, since $\text{SA}(b..e)$ also contains all $j'' \in \text{Occ}_{2\ell}(\text{SA}[i])$ sorted by their length- 2ℓ right context, although potentially in a different order. Note, however, that $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ only requires $T^\infty[j'..j' + 2\ell] = T^\infty[\text{SA}[i].. \text{SA}[i] + 2\ell]$. Thus, the ability to find the k th element in the sequence of all $j'' \in \text{Occ}_\ell(\text{SA}[i])$ sorted by $T^\infty[j''..j'' + 2\ell]$ (with ties resolved arbitrarily) is all we need to compute j' . Note now that to show a common offset δ_5 above, we only used the fact that suffixes shared a prefix of length at least 3τ . Thus, by $3\tau \leq \ell$, here we also have that all $j'' \in \text{Occ}_\ell(\text{SA}[i])$ share a common offset $\delta_5 = \text{succ}_5(\text{SA}[i]) - \text{SA}[i]$ such that $j'' + \delta_5 = \min(\mathbf{S} \cap [j''..j'' + \tau])$. Consequently, to find j' we take some $q \geq 2\ell$ and:

1. First, letting $\delta_5 = \text{succ}_5(\text{SA}[i]) - \text{SA}[i]$, we compute the number m of positions $s' \in \mathbf{S}$ that have their reversed left length- q context in the range $[X..X']$ (where $\bar{X} = T[\text{SA}[i].. \text{SA}[i] + \delta_5]$ and $X' = Xc^\infty$) and their right length- q context in $[\varepsilon..Y]$ (where $Y = T^\infty[\text{SA}[i] + \delta_5.. \text{SA}[i] + \ell]$).
2. Then, for any $k \in [1..|\text{Occ}_\ell(\text{SA}[i])|]$, the $(m+k)$ th element in the sequence of all $s' \in \mathbf{S}$ sorted by the length- 2ℓ right context which simultaneously has its left length- q context in $[X..X']$, satisfies $T^\infty[s' - \delta_5..s' - \delta_5 + 2\ell] = T^\infty[\text{SA}[b+k].. \text{SA}[b+k] + 2\ell]$. In particular, the position s' for $k = i - b$ satisfies $T^\infty[s' - \delta_5..s' - \delta_5 + 2\ell] = T^\infty[\text{SA}[i].. \text{SA}[i] + 2\ell]$, i.e., $s' - \delta_5 \in \text{Occ}_{2\ell}(\text{SA}[i])$. Thus, finding j' reduces to a range selection query on \mathcal{P} .

The above reduction is proved in Lemma 6.14. One last detail is that we need X , Y , and δ_5 . We note, however, that they all depend only on $T^\infty[\text{SA}[i].. \text{SA}[i] + \ell]$, and thus can be computed

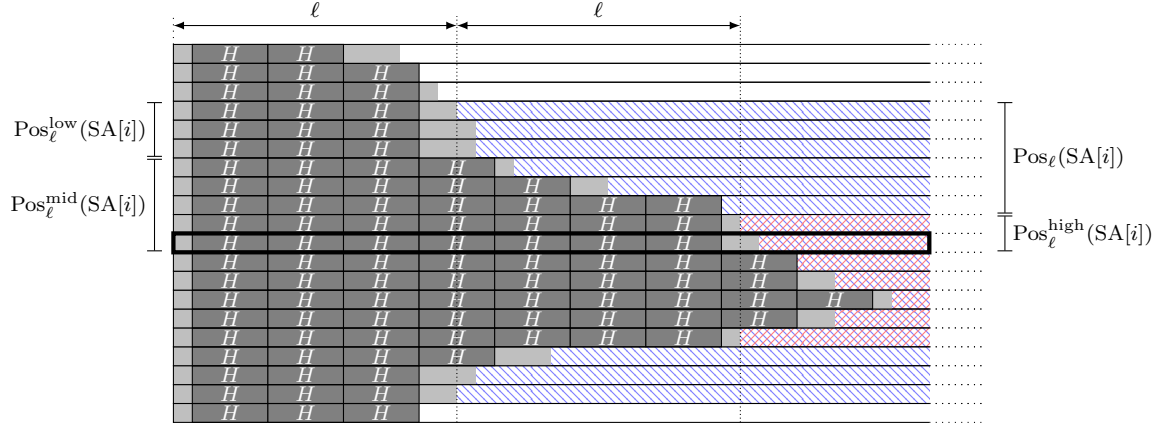


Figure 2: Example showing the sets $\text{Pos}_\ell(\text{SA}[i])$, $\text{Pos}_\ell^{\text{low}}(\text{SA}[i])$, $\text{Pos}_\ell^{\text{mid}}(\text{SA}[i])$, and $\text{Pos}_\ell^{\text{high}}(\text{SA}[i])$. Note, that $|\text{Pos}_\ell(\text{SA}[i])| = |\text{Pos}_\ell^{\text{low}}(\text{SA}[i])| + |\text{Pos}_\ell^{\text{mid}}(\text{SA}[i])| - |\text{Pos}_\ell^{\text{high}}(\text{SA}[i])|$ (see Lemma 6.23). The suffix $T[\text{SA}[i]..n]$ is highlighted in bold. The sets $\text{Occ}_\ell(\text{SA}[i])$ and $\text{Occ}_{2\ell}(\text{SA}[i])$ and marked with blue and red (respectively).

using $j \in \text{Occ}_\ell(\text{SA}[i])$ (which we have as input). We have thus proved the following result, which combined with Proposition 3.1 concludes the description of the SA query for nonperiodic $\text{SA}[i]$.

Proposition 3.2. *Under Assumption 6.5, assuming $\text{SA}[i] \in [1..n] \setminus R(\tau, T)$, given a position $j \in \text{Occ}_\ell(\text{SA}[i])$ and the pair $(\text{RangeBeg}_\ell(\text{SA}[i]), \text{RangeEnd}_\ell(\text{SA}[i]))$, we can efficiently find some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$.*

The Periodic Positions Assume $\text{SA}[i] \in R(\tau, T)$. The standard way to introduce structure among periodic positions (see, e.g., [KK19]) is as follows. Observe that if $j, j+1 \in R(\tau, T)$, then $\text{per}(T[j..j+3\tau-1]) = \text{per}(T[j+1..j+1+3\tau-1])$. This implies that any maximal block of positions in $R(\tau, T)$ defines a highly periodic fragment of T (called a “run”) with an associated period p . To compare runs, we “anchor” each run $T[j..j']$ by selecting some $H \in \Sigma^+$, called its *root*, so that $T[j..j']$ is a substring of H^∞ and no nontrivial rotation of H is selected for other runs. Every run $T[j..j']$ can then be uniquely written as $T[j..j'] = H'H^kH''$, where $k \geq 1$ and H' (resp. H'') is a proper suffix (resp. prefix) of H . The value k is called the *exponent* of j . We finally classify $\text{type}(j) = -1$ if $T[j'] \prec T[j'-p]$ and $\text{type}(j) = +1$ otherwise, where $p = |H|$.

The first major challenge in the periodic case is selecting roots efficiently. An easy solution in the static case (e.g. [KK19, KK21]) is to choose the lexicographically smallest rotation of H (known as the *Lyndon root*). This seems very difficult in the dynamic case, however. We instead show a construction that exploits the presence of symmetry-breaking component in the signature parsing (i.e., the deterministic coin tossing [CV86]) to develop a custom computation of roots (Section 8.10).

Denote $b = \text{RangeBeg}_\ell(\text{SA}[i])$ and $e = \text{RangeEnd}_\ell(\text{SA}[i])$. Observe that all $i' \in (b..e)$ with $\text{type}(\text{SA}[i']) < 0$ precede those with $\text{type}(\text{SA}[i']) > 0$. Moreover, $\text{exp}(\text{SA}[i'])$ among those with $\text{type}(\text{SA}[i']) < 0$ (resp. $\text{type}(\text{SA}[i']) > 0$) is increasing (resp. decreasing) as we increase i' ; see also Lemma 6.7. This structure of the periodic block has led to relatively straightforward processing of periodic positions in previous applications (e.g., the BWT construction in [KK19]), where the positions were first grouped by the type, and then by the exponent. In our case, however, we need to exclude the positions with very small and very large exponents. We thus employ the following modification to the above scheme: Rather than computing $|\text{Pos}_\ell(\text{SA}[i])|$ in one step, we prove (Lemma 6.23) that it can be expressed as a combination of three sets, $\text{Pos}_\ell^{\text{low}}(\text{SA}[i])$ (containing

exponents “truncated” at length ℓ), $\text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ (where truncation occurs for length 2ℓ), and $\text{Pos}_\ell^{\text{mid}}(\text{SA}[i])$ (all exponents in between); see Fig. 2 for an example. We then compute the following values: the type $\text{type}(\text{SA}[i])$, the size $|\text{Pos}_\ell^{\text{low}}(\text{SA}[i])|$, the exponent $\text{exp}(\text{SA}[i])$, the size $|\text{Pos}_\ell^{\text{mid}}(\text{SA}[i])|$, a position $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$, the size $|\text{Pos}_\ell^{\text{high}}(\text{SA}[i])|$, and the size $|\text{Occ}_{2\ell}(\text{SA}[i])|$. See the proof of Proposition 6.53 for the high-level algorithm and Sections 6.3.3 to 6.3.8 for the implementation of subsequent steps. Finally, we derive $|\text{Pos}_\ell(\text{SA}[i])|$, which equals $|\text{Pos}_\ell^{\text{low}}(\text{SA}[i])| + |\text{Pos}_\ell^{\text{mid}}(\text{SA}[i])| - |\text{Pos}_\ell^{\text{high}}(\text{SA}[i])|$, if $\text{type}(\text{SA}[i]) = -1$, and the values $\text{RangeBeg}_{2\ell}(\text{SA}[i]) = \text{RangeBeg}_\ell(\text{SA}[i]) + |\text{Pos}_\ell(\text{SA}[i])|$ as well as $\text{RangeEnd}_{2\ell}(\text{SA}[i]) = \text{RangeBeg}_{2\ell}(\text{SA}[i]) + |\text{Occ}_{2\ell}(\text{SA}[i])|$. The case of $\text{type}(\text{SA}[i]) = +1$ is symmetric: we then compute $|\text{Pos}'_\ell(\text{SA}[i])|$ instead of $|\text{Pos}_\ell(\text{SA}[i])|$.

Dynamic Text Implementation In the second part of the paper (Sections 7 to 10), we develop a data structure that maintains a dynamic text (subject to updates listed below) and provides efficient implementation of the auxiliary queries specified in the four assumptions of Section 6.

Definition 3.3. We say that a *dynamic text* over an integer alphabet Σ (with $\$ = \min \Sigma$) is a data structure that maintains a text $T \in \Sigma^+$ subject to the following updates:

initialize(σ): Given the alphabet size σ , initialize the data structure, setting $T := \$$;

insert(i, a): Given $i \in [1..|T|]$ and $a \in \Sigma \setminus \{\$\}$, set $T := T[1..i] \cdot a \cdot T[i..|T|]$.

delete(i): Given $i \in [1..|T|]$, set $T := T[1..i] \cdot T[i..|T|]$.

swap(i, j, k): Given $i, j, k \in [1..|T|]$ with $i \leq j \leq k$, set $T := T[1..i] \cdot T[j..k] \cdot T[i..j] \cdot T[k..|T|]$.

Observe that our interface does not directly support character substitutions; this is because **substitute(i, a)** can be implemented as **delete(i)** followed by **insert(i, a)**. Moreover, note that the interface enforces the requirement of Section 6 that $\{i \in [1..|T|] : T[i] = \$\} = \{|T|\}$.

Various components of our data structure, described in Section 10, maintain auxiliary information associated to individual positions of the text T (such as whether the position belongs to a synchronizing set). Individual updates typically alter the positions of many characters in T (for example, insertion moves the character at position j to position $j + 1$ for each $j \in [i..|T|]$), so addressing the characters by their position is volatile. To address this issue, we use a formalism of *labelled strings*, where each character is associated to a unique label that is fixed throughout character’s lifetime. This provides a clean realization of the concept of *pointers to characters*, introduced in [ABR00] along with $\mathcal{O}(\log n)$ -time conversion between labels (pointers) and positions.

The main challenge that arises in our implementation of a dynamic text is to maintain synchronizing sets. As for the succ_5 queries alone, we could generate synchronizing positions at query time. Nevertheless, Assumption 6.5 also entails supporting range queries over a set of points in one-to-one correspondence with the synchronizing positions, we cannot evade robust maintenance.

Dynamic Synchronizing Sets By the consistency condition in Definition 2.1, whether a position j belongs to a synchronizing set S is decided based on the right context $T[j..j + 2\tau)$. This means that the entire synchronizing set can be described using a family $\mathcal{F} \subseteq \Sigma^{2\tau}$ such that $j \in S$ if and only if $T[j..j + 2\tau) \in \mathcal{F}$. The construction algorithms in [KK19] select such \mathcal{F} adaptively (based on the contents of T) to guarantee that $|S|$ is small. In a dynamic scenario, we could either select \mathcal{F} non-adaptively and keep it fixed; or adaptively modify \mathcal{F} as the text T changes.

The second approach poses a very difficult task: the procedure maintaining \mathcal{F} does not know the future updates, yet it needs to be robust against any malicious sequence of updates that an adversary could devise. This is especially hard in the deterministic setting, where we cannot hide \mathcal{F} from the

adversary. Thus, we aim for non-adaptivity, which comes at the price of increasing the synchronizing set size by a factor of $\mathcal{O}(\log^*(\sigma\tau))$. On the positive side, a non-adaptive choice of \mathcal{F} means that \mathbf{S} only undergoes local changes; for example, a substitution of $T[i]$ may only affect $\mathbf{S} \cap (i - 2\tau \dots i]$. Moreover, since \mathcal{F} yields small synchronizing sets for all strings, this is in particular true for all substrings $T[i \dots j]$, whose synchronizing sets are in one-to-one correspondence with $\mathbf{S} \cap [i \dots j - 2\tau]$. This means that \mathbf{S} is *locally sparse* and that each update incurs $\mathcal{O}(\log^*(\sigma\tau))$ changes to \mathbf{S} .

The remaining challenge is thus to devise a non-adaptive synchronizing set construction. Although all existing constructions are adaptive, Birenzweige, Golan, and Porat [BGP20] provided a non-adaptive construction of a related notion of *partitioning sets*. While partitioning sets and synchronizing sets satisfy similar consistency conditions, the density condition of synchronizing sets is significantly stronger, which is crucial for a clean separation between periodic and nonperiodic positions. Thus, in Section 9, we strengthen the construction of [BGP20] so that it produces synchronizing sets. This boils down to ‘fixing’ the set in the vicinity of positions in $\mathbf{R}(\tau, T)$.

Dynamic Strings over Balanced Signature Parsing Unfortunately, the approach of [BGP20] only comes with a static implementation. Thus, we need to dive into their techniques and provide an efficient dynamic implementation. Their central tool is a locally consistent parsing algorithm that iteratively parses the text using deterministic coin tossing [CV86] to determine phrase boundaries. Similar techniques have been used many times (see e.g. [SV94, SV96, MSU97, ABR00, NII+20]), but the particular flavor employed in [BGP20] involves a mechanism that, up to date, has not been adapted to the dynamic setting. Namely, as subsequent levels of the parsing provide coarser and coarser partitions into phrases, the procedure grouping phrases into blocks (to be merged in the next level) takes into account the lengths of the phrases, enforcing very long phrases to form single-element blocks. This trick makes the phrase lengths much more balanced, which is crucial in controlling the context size that governs the local consistency of the synchronizing sets derived from the parsing.

In Section 7, we develop *balanced signature parsing*: a version of signature parsing (originating the early works on dynamic strings [MSU97, ABR00]) that involves the phrase balancing mechanism. Then, in Section 8, we provide a dynamic strings implementation based on the balanced signature parsing. The main difference compared to the previous work [MSU97, ABR00, GKK+18] stems from the fact that the size of the *context-sensitive* part of the parsing (that may change depending on the context surrounding a given substring) is bounded in terms of the number of individual letters rather than the number of phrases at the respective level of the parsing. Due to this, we need to provide new (slightly modified) implementations of the basic operations (such as updates and longest common prefix queries). However, we also benefit from this feature, obtaining faster running times for more advanced queries, such as the period queries (which we utilize in Section 9 to retrieve $\mathbf{R}(\tau, T)$) compared to solutions using existing dynamic strings implementations [CKW20].

4 Generalized Range Counting and Selection Queries

In this section we introduce generalized range counting and selection queries. The generalization lies in the fact that the ‘‘coordinates’’ of points can come from any ordered set. In particular, coordinates of points in some of our structures will be elements of \mathbb{Z}_+ or Σ^* (i.e., the strings over alphabet Σ). Furthermore, the points in our data structures are labelled with distinct integer identifiers; we allow multiple points with the same coordinates, though.

The section is organized as follows. We start with the definition of range counting/selection

queries. In the following two sections (Sections 4.1 and 4.2), we present two data structures supporting efficient range counting/selection queries over instances that will be of interest to us.

Let \mathcal{X} and \mathcal{Y} be some linearly ordered sets (we denote the order on both sets using \prec or \preceq). Let $\mathcal{P} \subseteq \mathcal{X} \times \mathcal{Y} \times \mathbb{Z}$ be a finite set of points with distinct integer labels. We define the notation for *range counting* and *range selection* queries as follows.

Range counting query $\text{r-count}_{\mathcal{P}}(X_l, X_u, Y_u)$: Given $X_l, X_r \in \mathcal{X}$ and $Y_u \in \mathcal{Y}$, return $|\{(X, Y, \ell) \in \mathcal{P} : X_l \preceq X \prec X_r \text{ and } Y \prec Y_u\}|$. We also let $\text{r-count}_{\mathcal{P}}^{\text{inc}}(X_l, X_r, Y_u) = |\{(X, Y, \ell) \in \mathcal{P} : X_l \preceq X \prec X_r \text{ and } Y \preceq Y_u\}|$ and $\text{r-count}_{\mathcal{P}}(X_l, X_r) = |\{(X, Y, \ell) \in \mathcal{P} : X_l \preceq X \prec X_r\}|$.

Range selection query $\text{r-select}_{\mathcal{P}}(X_l, X_u, r)$: Given $X_l, X_u \in \mathcal{X}$ and $r \in [1 \dots \text{r-count}_{\mathcal{P}}(X_l, X_u)]$, return any $\ell \in \text{r-select}_{\mathcal{P}}(X_l, X_u, r) := \{\ell : (X, Y, \ell) \in \mathcal{P}, X_l \preceq X \prec X_u, \text{ and } Y = Y_u\}$, where $Y_u \in \mathcal{Y}$ is such that $r \in (\text{r-count}_{\mathcal{P}}(X_l, X_r, Y_u) \dots \text{r-count}_{\mathcal{P}}^{\text{inc}}(X_l, X_r, Y_u))$.

Theorem 4.1. *Suppose that the elements of \mathcal{X} and \mathcal{Y} can be compared in $\mathcal{O}(t)$ time. Then, there is a deterministic data structure that maintains a set $\mathcal{P} \subseteq \mathcal{X} \times \mathcal{Y} \times [0 \dots 2^w)$ of size n , with insertions in $\mathcal{O}((t + \log n) \log n)$ time and deletions in $\mathcal{O}(\log^2 n)$ time so that range queries are answered in $\mathcal{O}((t + \log^2 n) \log n)$ time.*

Proof. The set \mathcal{P} is stored in a data structure of [LW82] (see also [WL85, Cha88]) for dynamic range counting queries; this component supports updates and queries in $\mathcal{O}(\log^2 n)$ assuming $\mathcal{O}(1)$ -time comparisons. As for range selection queries, we resort to binary search with range counting queries as an oracle (the universe searched consists of the second coordinates of all points in \mathcal{P}). Thus, range selection queries cost $\mathcal{O}(\log^3 n)$ time.

In order to substantiate the assumption on constant comparison time, we additionally maintain \mathcal{P} in two instances of the order-maintenance data structure [DS87, BFG⁺17], with points (X, Y, ℓ) ordered according to X in the first instance and according to Y in the second instance (ties are resolved arbitrarily). This allows for $\mathcal{O}(1)$ -time comparisons between points in \mathcal{P} . The overhead for deletions is $\mathcal{O}(1)$, but insertions to the order-maintenance structure require specifying the predecessor of the newly inserted element; we find the predecessor in $\mathcal{O}(t \log n)$ time using binary search. Similarly, at query time, we temporarily add the query coordinates $(X_l, X_r, \text{ and, if specified, } Y_u)$ to the appropriate order-maintenance structure, also at the cost of an extra $\mathcal{O}(t \log n)$ term in the query time. \square

4.1 A String-String Instance

Definition 4.2. Let $T \in \Sigma^n$. For any $q \in \mathbb{Z}_+$ and any set $\mathbf{P} \subseteq [1 \dots n]$, we define

$$\text{Points}_q(T, \mathbf{P}) = \{(\overline{T^\infty[p - q \dots p]}, T^\infty[p \dots p + q], p) : p \in \mathbf{P}\}.$$

In other words, $\text{Points}_q(T, \mathbf{P})$ represents the collection of string-pairs (X, Y) composed of reversed length- q left context and a length- q right context (in T^∞) of every $p \in \mathbf{P}$, and for any $(X, Y, \ell) \in \text{Points}_q(T, \mathbf{P})$, the label $\ell \in \mathbf{P}$ is the underlying position. Equivalently, $\text{Points}_q(T, \mathbf{P})$ can be interpreted as a set of labelled points (X, Y, ℓ) with coordinates X, Y and an integer label ℓ . The order among strings on the $\mathcal{X} = \Sigma^*$ and $\mathcal{Y} = \Sigma^*$ axis is the lexicographical order.

Next, we define the problem of supporting range counting/selection queries on $\text{Points}_q(T, \mathbf{P})$ for some special family of queries that can be succinctly represented as substrings of T^∞ or $\overline{T^\infty}$.

Problem 4.3 (String-String Range Queries). Let $T \in \Sigma^n$, $q \in [1..3n]$, and $P \subseteq [1..n]$ be a set satisfying $|P| = m$. Denote $\mathcal{P} = \text{Points}_q(T, P)$ and $c = \max \Sigma$. Provide efficient support for the following queries:

1. Given $i \in [1..n]$ and $q_l, q_r \in [0..2n]$, return $\text{r-count}_{\mathcal{P}}(X_l, X_u, Y_l)$ and $\text{r-count}_{\mathcal{P}}(X_l, X_u, Y_u)$, where $\overline{X}_l = T^\infty[i - q_l..i]$, $X_u = X_l c^\infty$, $Y_l = T^\infty[i..i + q_r]$, and $Y_u = Y_l c^\infty$, and
2. Given $i \in [1..n]$, $q_l \in [0..2n]$, and $r \in [1.. \text{r-count}_{\mathcal{P}}(X_l, X_u)]$ (where $\overline{X}_l = T^\infty[i - q_l..i]$ and $X_u = X_l c^\infty$), return some position $p \in \text{r-select}_{\mathcal{P}}(X_l, X_u, r)$.

4.2 An Int-String Instance

Definition 4.4. Let $T \in \Sigma^n$, $q \in \mathbb{Z}_+$, and suppose that $P \subseteq [1..n] \times \mathbb{Z}_{\geq 0}$ contains pairs with distinct first coordinates. We define

$$\text{Points}_q(T, P) = \{(d, T^\infty[p..p+q], p) : (p, d) \in P\}.$$

In other words, $\text{Points}_q(T, P)$ represents the collection of pairs (X, Y) composed of an integer $X = d$ and a length- q right context (in T^∞) of position p for every $(p, d) \in P$. The label of each point is set to p ; the assumption on P guarantees that points have distinct labels. Equivalently, $\text{Points}_q(T, P)$ can be interpreted as a set of labelled points from $\mathcal{X} \times \mathcal{Y} \times \mathbb{Z}$, where the coordinates of each point $(X, Y, \ell) \in \mathcal{P}$ are an integer X , and a string Y , whereas the label is ℓ . We use the standard order on $\mathcal{X} = \mathbb{Z}_+$ and the lexicographic order on $\mathcal{Y} = \Sigma^*$.

Next, we define the problem of efficiently supporting range counting and selection queries on $\text{Points}_q(T, P)$ for some restricted family of queries that can be succinctly represented as substrings of T^∞ .

Problem 4.5 (Int-String Range Queries). Let $T \in \Sigma^n$, $q \in [1..3n]$, and $P \subseteq [1..n] \times [0..n]$ be a set of size $|P| = m$. Denote $\mathcal{P} = \text{Points}_q(T, P)$ and $c = \max \Sigma$. Provide efficient support for the following queries:

1. Given $i \in [1..n]$, $x \in [0..n]$, and $q_r \in [0..2n]$, return $\text{r-count}_{\mathcal{P}}(x, n, Y_l)$, $\text{r-count}_{\mathcal{P}}(x, n, Y_u)$, and $\text{r-count}_{\mathcal{P}}(x, n)$, where $Y_l = T^\infty[i..i + q_r]$, $Y_u = Y_l c^\infty$, and
2. Given $x \in [0..n]$ and $r \in [1.. \text{r-count}_{\mathcal{P}}(x, n)]$, return some position $p \in \text{r-select}_{\mathcal{P}}(x, n, r)$.

5 Modular Constraint Queries

Let $\mathcal{I} \subseteq \mathbb{Z}_{\geq 0}^2 \times \mathbb{Z}$ be a finite set of integer intervals (represented by endpoints) with distinct integer labels (we allow multiple intervals with the same endpoints). We define the *modular constraint counting* and *modular constraint selection* queries as follows:

Modular constraint count query $\text{mod-count}_{\mathcal{I}}(h, r, q)$: Given $h \in \mathbb{Z}_+$, $r \in [0..h]$ and $q \in \mathbb{Z}_{\geq 0}$, return $\sum_{(b,e,\ell) \in \mathcal{I}} |\{j \in [b..e] : j \bmod h = r \text{ and } \lfloor \frac{j}{h} \rfloor \leq q\}|$. As a notational convenience, we define $\text{mod-count}_{\mathcal{I}}(h, r) = \sum_{(b,e,\ell) \in \mathcal{I}} |\{j \in [b..e] : j \bmod h = r\}|$.

Modular constraint selection query $\text{mod-select}_{\mathcal{I}}(h, r, c)$: Given $h \in \mathbb{Z}_+$, $r \in [0..h]$ and $c \in [1.. \text{mod-count}_{\mathcal{I}}(h, r)]$, return (the unique) positive integer q satisfying the condition $c \in (\text{mod-count}_{\mathcal{I}}(h, r, q - 1) .. \text{mod-count}_{\mathcal{I}}(h, r, q))$.

We extend the above notation to sets of labelled one-sided intervals (identified for simplicity with integers) as follows. For any finite set $Q \subseteq \mathbb{Z}_{\geq 0} \times \mathbb{Z}$ of coordinates labelled with distinct

integer labels (we allow equal coordinates), we define $\text{mod-count}_Q(h, r, q) := \text{mod-count}_{\mathcal{I}}(h, r, q)$, $\text{mod-count}_Q(h, r) := \text{mod-count}_{\mathcal{I}}(h, r)$, and $\text{mod-select}_Q(h, r, c) := \text{mod-select}_{\mathcal{I}}(h, r, c)$, where $\mathcal{I} = \{(0, e, \ell) : (e, \ell) \in Q\}$.

Proposition 5.1. *Let $h \in [0..2^w)$. There is a deterministic data structure that maintains a set of labelled coordinates $Q \subseteq [0..2^w) \times \mathbb{Z}$ of size $|Q| = n$ so that updates (insertions and deletions in Q) and modular constraint counting queries (returning $\text{mod-count}_Q(h, r, q)$, given any $r \in [0..h)$ and $q \in \mathbb{Z}_{\geq 0}$) take $\mathcal{O}(\log^2 n)$ time.*

Proof. Observe that for any $e \in \mathbb{Z}_{\geq 0}$, letting $x := e \bmod n$ and $y := \lfloor \frac{e}{h} \rfloor$, it holds

$$|\{j \in [0..e) : j \bmod h = r \text{ and } \lfloor \frac{j}{h} \rfloor \leq q\}| = \min(y, q) + \delta, \quad (5.1)$$

where $\delta = 1$ if and only if $r \leq x$ and $y \leq q$. The basic idea is to maintain two data structures, each responsible for computing the aggregate value (summed over all coordinates in Q) of one of the terms on the right-hand side of Eq. (5.1). The key property of this separation is that computing the sum of the first terms does not depend on the query argument r , and hence can be reduced to a prefix sum query. On the other hand, the second term for each integer in Q contributes either zero or one to the total sum, and hence can be reduced to an orthogonal range counting query.

Denote $Q = \{(e_1, \ell_1), \dots, (e_k, \ell_k)\}$, where $e_1 \leq \dots \leq e_k$. For any $i \in [1..k]$, let $x_i = e_i \bmod h$ and $y_i = \lfloor \frac{e_i}{h} \rfloor$. Denote $\mathcal{S} = ((y_1, \ell_1), \dots, (y_k, \ell_k))$ and $\mathcal{P} = \{(x_i, y_i, \ell_i) : i \in [1..k]\}$. Then, by Eq. (5.1), if we let $j \in [0..k]$ be the largest integer satisfying $\max\{y_1, \dots, y_j\} < q$, it holds

$$\begin{aligned} \text{mod-count}_Q(h, r, q) &= \sum_{i=1}^k \min(y_i, q) + |\{i \in [1..k] : r \leq x_i < h \text{ and } y_i \leq q\}| \\ &= \sum_{i=1}^j y_i + q(k - j) + \text{r-count}_{\mathcal{P}}(r, h, q+1). \end{aligned}$$

To compute the first two terms, we store elements (y_i, ℓ_i) of \mathcal{S} using a balanced binary search tree (e.g., an AVL tree [AVL62]) ordered by y_i . A node v_i corresponding to (y_i, ℓ_i) is augmented with the value $\sum_{j \in J(v_i)} y_j$, where $J(v_i)$ contains all $j \in [1..k]$ such v_j is in the subtree rooted in v_i . Given such representation of \mathcal{S} , we can compute j together with $\sum_{i=1}^j y_i$ in $\mathcal{O}(\log n)$ time. To compute the third term, we apply Theorem 4.1 for \mathcal{P} . The query takes $\mathcal{O}(\log^2 n)$ time.

To insert or delete a labelled coordinate $(v, \ell) \in [0..2^w) \times \mathbb{Z}$ from Q , we first compute $x = v \bmod h$ and $y = \lfloor \frac{v}{h} \rfloor$. Adding/deleting (y, ℓ) from the first component of our structure is straightforward and takes $\mathcal{O}(\log n)$ time. The extra values in each node can be computed using the information stored in the node and its children, and thus can be maintained during rotations. By Theorem 4.1, insertions and deletions on the second structure take $\mathcal{O}(\log^2 n)$ time. \square

Corollary 5.2. *Let $h, n \in [0..2^w)$. There is a deterministic data structure that maintains a set of labelled intervals $\mathcal{I} \subseteq [0..n)^2 \times \mathbb{Z}$ of size $|\mathcal{I}| \leq n$ so that updates (insertions and deletions in \mathcal{I}) take $\mathcal{O}(\log^2 n)$ time and modular constraint counting (resp. selection) queries are answered in $\mathcal{O}(\log^2 n)$ (resp. $\mathcal{O}(\log^3 n)$) time.*

Proof. We keep two instances of the structure from Proposition 5.1, one for the set of labelled coordinates Q^- containing left endpoints of intervals in \mathcal{I} and one for the set Q^+ of right endpoints.

Given any $r \in [0..h)$ and $q \in \mathbb{Z}_{\geq 0}$, we return $\text{mod-count}_{\mathcal{I}}(h, r, q) = \text{mod-count}_{Q^+}(h, r, q) - \text{mod-count}_{Q^-}(h, r, q)$ in $\mathcal{O}(\log^2 n)$ time. The select query on \mathcal{I} is implemented in $\mathcal{O}(\log^3 n)$ time by using binary search (in the range $[0..n)$) and modular constraint counting queries.

Updates simply insert/delete the left (resp. right) endpoint of the input interval into the structure for Q^- (resp. Q^+). By Proposition 5.1, each update to \mathcal{I} takes $\mathcal{O}(\log^2 n)$ time. \square

6 SA Query Algorithm

Let $T \in \Sigma^n$. In this section we show that under some small set of assumptions about the ability to perform queries on string synchronizing sets [KK19], we can perform SA queries (i.e., returning the value $\text{SA}[i]$, given any position $i \in [1..n]$) for T .

Let $\ell \geq 1$. For any $j \in [1..n]$, we define

$$\begin{aligned} \text{Occ}_{\ell}(j) &= \{j' \in [1..n] : T^{\infty}[j'..j'+\ell] = T^{\infty}[j..j+\ell]\}, \\ \text{RangeBeg}_{\ell}(j) &= |\{j' \in [1..n] : T[j'..n] \prec T[j..n] \text{ and } \text{LCE}_T(j', j) < \ell\}|, \text{ and} \\ \text{RangeEnd}_{\ell}(j) &= \text{RangeBeg}_{\ell}(j) + |\text{Occ}_{\ell}(j)|. \end{aligned}$$

For the motivation of the above names, note that viewing $P := T^{\infty}[j..j+\ell]$ as a pattern, we have $\{\text{SA}[i] : i \in (\text{RangeBeg}_{\ell}(j).. \text{RangeEnd}_{\ell}(j))\} = \{i \in [1..n] : T^{\infty}[i..i+|P|] = P\}$.

Moreover, for any $j \in [1..n]$, we define

$$\begin{aligned} \text{Pos}_{\ell}(j) &= \{j' \in [1..n] : T[j'..n] \prec T[j..n] \text{ and } \text{LCE}_T(j', j) \in [\ell..2\ell]\} \text{ and} \\ \text{Pos}'_{\ell}(j) &= \{j' \in [1..n] : T[j'..n] \succ T[j..n] \text{ and } \text{LCE}_T(j', j) \in [\ell..2\ell]\}. \end{aligned}$$

We denote the size of $\text{Pos}_{\ell}(j)$ and $\text{Pos}'_{\ell}(j)$ as $\delta_{\ell}(j) = |\text{Pos}_{\ell}(j)|$ and $\delta'_{\ell}(j) = |\text{Pos}'_{\ell}(j)|$. Then, it holds $(\text{RangeBeg}_{2\ell}(j), \text{RangeEnd}_{2\ell}(j)) = (\text{RangeBeg}_{\ell}(j) + \delta_{\ell}(j), \text{RangeBeg}_{\ell}(j) + \delta_{\ell}(j) + |\text{Occ}_{2\ell}(j)|)$, $(\text{RangeBeg}_{2\ell}(j), \text{RangeEnd}_{2\ell}(j)) = (\text{RangeEnd}_{\ell}(j) - \delta'_{\ell}(j) - |\text{Occ}_{2\ell}(j)|, \text{RangeEnd}_{\ell}(j) - \delta'_{\ell}(j))$.

Assumption 6.1. *For any $i \in [1..n]$, we can compute some $j \in \text{Occ}_{16}(\text{SA}[i])$ and the pair $(\text{RangeBeg}_{16}(\text{SA}[i]), \text{RangeEnd}_{16}(\text{SA}[i]))$ in $\mathcal{O}(t)$ time.*

The main idea of our algorithm to compute $\text{SA}[i]$ is as follows. Suppose that we have obtained some $j \in \text{Occ}_{16}(\text{SA}[i])$ and $(\text{RangeBeg}_{16}(\text{SA}[i]), \text{RangeEnd}_{16}(\text{SA}[i]))$ using Assumption 6.1. Then, for $q = 4, \dots, \lceil \log n \rceil - 1$, we compute $(\text{RangeBeg}_{2^{q+1}}(\text{SA}[i]), \text{RangeEnd}_{2^{q+1}}(\text{SA}[i]))$ and some $j' \in \text{Occ}_{2^{q+1}}(\text{SA}[i])$, by using as input some position $j \in \text{Occ}_{2^q}(\text{SA}[i])$ and the pair of integers $(\text{RangeBeg}_{2^q}(\text{SA}[i]), \text{RangeEnd}_{2^q}(\text{SA}[i]))$, i.e., the output of the earlier step. This algorithm lets us compute $\text{SA}[i]$, since eventually we obtain some $j' \in \text{Occ}_{2^{\lceil \log n \rceil}}(\text{SA}[i])$, and for any $k \geq n$, $\text{Occ}_k(\text{SA}[i]) = \{\text{SA}[i]\}$, i.e., the final computed position must satisfy $j' = \text{SA}[i]$. To implement the above strategy, we now need to present a query algorithm parameterized by any $\ell \geq 16$ that, given $i \in [1..n]$ along with some $j \in \text{Occ}_{\ell}(\text{SA}[i])$ and the pair $(\text{RangeBeg}_{\ell}(\text{SA}[i]), \text{RangeEnd}_{\ell}(\text{SA}[i]))$ as input, returns some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ and the pair $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$.

Let us now fix some $\ell \in [16..n)$. For any $\tau \in \mathbb{Z}_+$ we define

$$\text{R}(\tau, T) := \{i \in [1..|T| - 3\tau + 2] : \text{per}(T[i..i+3\tau-2]) \leq \frac{1}{3}\tau\}. \quad (6.1)$$

The rest of this section is organized into four subsections. In the first three subsections, we show that under three assumptions (Assumptions 6.3, 6.5, and 6.21) about the ability to perform some queries in a black-box manner, given any index $i \in [1..n]$ along with some position

$j \in \text{Occ}_\ell(\text{SA}[i])$ and the pair of integers $(\text{RangeBeg}_\ell(\text{SA}[i]), \text{RangeEnd}_\ell(\text{SA}[i]))$ as input, we can efficiently compute some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ and the pair $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$ as output. In the first of these three subsections (Section 6.1) we show the algorithm to efficiently determine if the query position $i \in [1..n]$ satisfies $\text{SA}[i] \in \mathbb{R}(\lfloor \frac{\ell}{3} \rfloor, T)$. In Section 6.2 (resp. Section 6.3) we then present the query algorithm to efficiently compute some position $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ and the pair $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$ from $j \in \text{Occ}_{2\ell}(\text{SA}[i])$ and $(\text{RangeBeg}_\ell(\text{SA}[i]), \text{RangeEnd}_\ell(\text{SA}[i]))$ for the case $\text{SA}[i] \in [1..n] \setminus \mathbb{R}(\lfloor \frac{\ell}{3} \rfloor, T)$ (resp. $\text{SA}[i] \in \mathbb{R}(\lfloor \frac{\ell}{3} \rfloor, T)$). All steps of the query algorithms are put together in Section 6.4.

Remark 6.2. Note that this section only presents a *reduction* of SA query to some other queries. To obtain the data structure for SA queries, we need to still prove that such queries can be implemented, i.e., that Assumptions 6.1, 6.3, 6.5, and 6.21 can be satisfied with some small query time t . We will prove this in later sections, and for now focus only on showing the reduction.

6.1 The Index Core

Assume $\ell \in [16..n]$. In this section, we show that if for any $i \in [1..n]$ we can efficiently check if $i \in \mathbb{R}(\lfloor \frac{\ell}{3} \rfloor, T)$ (Assumption 6.3), then given any $j \in \text{Occ}_\ell(\text{SA}[i])$ we can equally efficiently determine if $\text{SA}[i] \in \mathbb{R}(\lfloor \frac{\ell}{3} \rfloor, T)$ (i.e., if $\text{SA}[i]$ is a *periodic position*) or $\text{SA}[i] \in [1..n] \setminus \mathbb{R}(\lfloor \frac{\ell}{3} \rfloor, T)$ (such $\text{SA}[i]$ is called a *nonperiodic position*).

Assumption 6.3. For any $i \in [1..n]$, we can in $\mathcal{O}(t)$ time check if $i \in \mathbb{R}(\tau, T)$, where $\tau = \lfloor \frac{\ell}{3} \rfloor$.

Proposition 6.4. Let $i \in [1..n]$. Under Assumption 6.3, given any position $j \in \text{Occ}_\ell(\text{SA}[i])$, we can in $\mathcal{O}(t)$ time compute a bit indicating whether $\text{SA}[i] \in \mathbb{R}(\lfloor \frac{\ell}{3} \rfloor, T)$ holds.

Proof. Let $\tau = \lfloor \frac{\ell}{3} \rfloor$. Observe that for any $j \in [1..n - 3\tau + 2]$, $j \in \mathbb{R}$ holds if and only if $\text{per}(T[i..i+3\tau-1]) \leq \frac{1}{3}\tau$. In other words, $j \in \mathbb{R}(\tau, T)$ depends only on the substring $T[j..j+3\tau-1]$. Therefore, by $3\tau - 1 \leq \ell$ and $j \in \text{Occ}_\ell(\text{SA}[i])$, $\text{SA}[i] \in \mathbb{R}(\tau, T)$ holds if and only if $j \in \mathbb{R}(\tau, T)$. \square

6.2 The Nonperiodic Positions

Let $\tau = \lfloor \frac{\ell}{3} \rfloor$. In this section, we show that assuming that for some τ -synchronizing set \mathbb{S} we can efficiently perform $\text{succ}_\mathbb{S}$ queries (with $\text{succ}_\mathbb{S}(i) := \min\{j \in \mathbb{S} \cup \{|T| - 2\tau + 2 : j \geq i\}$ for any $i \in [1..n - 2\tau + 1]$), and support some string-string range queries (Assumption 6.5), given a position $i \in [1..n]$ satisfying $\text{SA}[i] \in [1..n] \setminus \mathbb{R}(\tau, T)$, along with some $j \in \text{Occ}_\ell(\text{SA}[i])$ and the pair $(\text{RangeBeg}_\ell(\text{SA}[i]), \text{RangeEnd}_\ell(\text{SA}[i]))$ as input, we can efficiently compute some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ and the pair $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$.

Assumption 6.5. For some τ -synchronizing set \mathbb{S} of T (where $\tau = \lfloor \frac{\ell}{3} \rfloor$) the queries $\text{succ}_\mathbb{S}(i)$ (where $i \in [1..n - 3\tau + 1] \setminus \mathbb{R}(\tau, T)$) and the string-string range queries (Problem 4.3) for text T , integer $q = 7\tau$, and the set of positions $\mathbb{P} = \mathbb{S}$ can be supported in $\mathcal{O}(t)$ time.

Remark 6.6. Note that by $\ell < n$ and $3\tau \leq \ell$, the value $q = 7\tau \leq 2\ell + \tau < 3n$ in the above assumption satisfies the requirement of Problem 4.3.

The section is organized into four parts. First, we present combinatorial results showing how to reduce LCE queries and prefix conditions to substring inequalities (Section 6.2.1). Next, we show how under Assumption 6.5 to combine the properties of synchronizing sets with these reductions to

compute the cardinalities of sets $\text{Pos}_\ell(j)$ and $\text{Occ}_{2\ell}(j)$ (Section 6.2.2). Then, in Section 6.2.3, we show how under the same assumptions to efficiently compute some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$. Finally, in Section 6.2.4, we put everything together.

6.2.1 Preliminaries

We start with a combinatorial result that shows the three fundamental reductions. In the first and second, we show an equivalence between LCE queries for suffixes of T and comparisons of substrings of T . These results will be used to derive the characterization of the set $\text{Pos}_\ell(j)$ and its components. In the third reduction we show an equivalence, where LCE queries are replaced with substring equalities. This will be used to characterize the set $\text{Occ}_{2\ell}(j)$.

Lemma 6.7. *Let $j \in [1..n]$ and $c = \max \Sigma$. Then:*

1. *If $0 \leq \ell_1 < \ell_2 \leq \ell_3$ then for any $j' \in [1..n]$, $T[j'..n] \prec T[j..n]$ and $\text{LCE}_T(j, j') \in [\ell_1.. \ell_2)$ holds if and only if $T^\infty[j..j + \ell_1] \preceq T^\infty[j'..j' + \ell_3] \prec T^\infty[j..j + \ell_2)$.*
2. *If $0 \leq \ell_1 \leq \ell_2$ then for any $j' \in [1..n]$, $T[j'..n] \succeq T[j..n]$ or $\text{LCE}_T(j, j') \geq \ell_1$ holds if and only if $T^\infty[j'..j' + \ell_2] \succeq T^\infty[j..j + \ell_1)$.*
3. *If $0 \leq \ell_1 \leq \ell_2$ then for any $j' \in [1..n]$, $T^\infty[j'..j' + \ell_1] = T^\infty[j..j + \ell_1)$ holds if and only if $T^\infty[j..j + \ell_1] \preceq T^\infty[j'..j' + \ell_2] \prec T^\infty[j..j + \ell_1)c^\infty$.*

Proof. 1. Assume $T[j'..n] \prec T[j..n]$ and $\text{LCE}_T(j, j') \in [\ell_1.. \ell_2)$. By $\ell_1 \leq \ell_3$, this implies $T^\infty[j..j + \ell_1] = T^\infty[j'..j' + \ell_1] \preceq T^\infty[j'..j' + \ell_3)$. To show the second condition, denote $\ell = \text{LCE}_T(j, j')$. From $T[j'..n] \prec T[j..n]$ we obtain $j \neq j'$. Thus, by the uniqueness of $T[n] = \$$ in T , we must have $T[j'..j' + \ell] = T[j..j + \ell)$ and $T[j' + \ell] \prec T[j + \ell]$, or equivalently, $T[j'..j' + \ell + 1] \prec T[j..j + \ell + 1)$. By $\ell + 1 \leq \ell_3$ and $\ell + 1 \leq \ell_2$, and since appending symbols at the end of the distinct equal-length substrings does not change their lexicographical order, we obtain $T^\infty[j'..j' + \ell_3] \prec T^\infty[j..j + \ell_2)$.

To show the opposite implication, assume $T^\infty[j..j + \ell_1] \preceq T^\infty[j'..j' + \ell_3] \prec T^\infty[j..j + \ell_2)$. First, note that $j \neq j'$, since otherwise, by $\ell_2 \leq \ell_3$, $T^\infty[j..j + \ell_2)$ would be a prefix of $T^\infty[j'..j' + \ell_3)$ and hence $T^\infty[j'..j' + \ell_3] \succeq T^\infty[j..j + \ell_2)$. Denote $\ell = \text{LCE}_T(j, j')$. By $j \neq j'$ and the uniqueness of $T[n] = \$$, we have $\max(j + \ell, j' + \ell) \leq n$. Suppose $\ell < \ell_1$ and consider two cases:

- If $T[j + \ell] \prec T[j + \ell]$ then $T[j'..j' + \ell + 1] \succ T[j..j + \ell + 1)$, which by $\ell + 1 \leq \ell_3$ and $\ell + 1 \leq \ell_2$ implies $T^\infty[j'..j' + \ell_3] \succ T^\infty[j..j + \ell_2)$, contradicting the assumption.
- On the other hand, if $T[j + \ell] \succ T[j' + \ell]$, then by $\ell + 1 \leq \ell_1$ and $\ell + 1 \leq \ell_3$ we obtain $T^\infty[j..j + \ell_1] \succ T^\infty[j'..j' + \ell_3)$.

Therefore, we must have $\ell \geq \ell_1$. Next, we note that $T^\infty[j'..j' + \ell_3] \prec T^\infty[j..j + \ell_2)$ implies $T^\infty[j'..j' + \ell_2] \prec T^\infty[j..j + \ell_2)$, which in turn gives $\ell < \ell_2$. Thus, it remains to show $T[j'..n] \prec T[j..n]$. For that it suffices to notice that knowing $\ell \in [\ell_1.. \ell_2)$, the assumption $T[j' + \ell] \succ T[j + \ell]$ implies $T^\infty[j'..j' + \ell + 1] \succ T^\infty[j..j + \ell + 1)$, which in turn implies $T^\infty[j'..j' + \ell_3] \succ T^\infty[j..j + \ell_2)$, contradicting the assumption.

2. Assume that it holds $T[j'..n] \succeq T[j..n]$ or $\text{LCE}_T(j, j') \geq \ell_1$. Let us first assume $T[j'..n] \succeq T[j..n]$. If $j' = j$, then by $\ell_1 \leq \ell_2$ we obtain $T^\infty[j'..j' + \ell_2] \succeq T^\infty[j'..j' + \ell_1] = T^\infty[j..j + \ell_1)$. Let us thus assume $j \neq j'$ and let $\ell = \text{LCE}_T(j, j')$. By $j \neq j'$ and the uniqueness of $T[n] = \$$, we have $\max(j + \ell, j' + \ell) \leq n$. Consider two cases:

- If $\ell \geq \ell_1$, then it holds $T^\infty[j'..j' + \ell_2] \succeq T^\infty[j'..j' + \ell_1] = T[j'..j' + \ell_1] = T[j..j + \ell_1) = T^\infty[j..j + \ell_1)$, i.e., we obtain the claim.

- Otherwise, (i.e., $\ell < \ell_1$), by $T[j'..n] \succeq T[j..n]$, we must have $T[j' + \ell] \succ T[j + \ell]$. Thus, $T[j'..j' + \ell + 1] \succ T[j..j + \ell + 1]$. Appending $\ell_1 - (\ell + 1)$ symbols after the mismatch does not change the other between suffixes. Thus, $T^\infty[j'..j' + \ell_2] \succeq T^\infty[j'..j' + \ell_1] \succ T^\infty[j..j + \ell_1]$.

Let us now assume $\text{LCE}_T(j, j') \geq \ell_1$. Then we obtain $T^\infty[j'..j' + \ell_2] \succeq T^\infty[j'..j' + \ell_1] = T[j'..j' + \ell_1] = T[j..j + \ell_1] = T^\infty[j..j + \ell_1]$.

To show the opposite implication, assume $T^\infty[j'..j' + \ell_2] \succeq T^\infty[j..j + \ell_1]$. If $j = j'$, then we immediately obtain the claim, since $T[j'..n] \succeq T[j..n]$. Let us thus assume $j \neq j'$ and let $\ell = \text{LCE}_T(j, j')$. By the uniqueness of $T[n] = \$$, it holds $\max(j + \ell, j' + \ell) \leq n$. If $\ell \geq \ell_1$, then we obtain the claim. Let us thus assume $\ell < \ell_1$. Then, $T[j' + \ell] \neq T[j + \ell]$. If $T[j' + \ell] \prec T[j + \ell]$, then $T[j'..j' + \ell + 1] \prec T[j..j + \ell + 1]$. Appending symbols after the mismatch does not change the other between suffixes. Thus, this implies $T^\infty[j'..j' + \ell_2] \prec T^\infty[j..j + \ell_1]$, contradicting the assumption. Thus, we must have $T[j' + \ell] \succ T[j + \ell]$. This implies $T[j'..n] \succ T[j..n]$.

3. Assume $T^\infty[j'..j' + \ell_1] = T^\infty[j..j + \ell_1]$. Then, by $\ell_1 \leq \ell_2$, we first obtain $T^\infty[j..j + \ell_1] = T^\infty[j'..j' + \ell_1] \preceq T^\infty[j'..j' + \ell_2]$. On the other hand, by $c = \max \Sigma$, $T^\infty[j'..j' + \ell_2] = T^\infty[j'..j' + \ell_1] \cdot T^\infty[j' + \ell_1..j' + \ell_2] \preceq T^\infty[j..j + \ell_1] \cdot c^{\ell_2 - \ell_1} \prec T^\infty[j..j + \ell_1] \cdot c^\infty$.

To show the opposite implication, assume $T^\infty[j..j + \ell_1] \preceq T^\infty[j'..j' + \ell_2] \prec T^\infty[j..j + \ell_1]c^\infty$. If $j' = j$, then we immediately obtain the claim. Let thus assume $j \neq j'$ and denote $\ell = \text{LCE}_T(j, j')$. By the uniqueness of $T[n] = \$$, we then have $\max(j + \ell, j' + \ell) \leq n$. Suppose $\ell < \ell_1$ and let us consider two cases:

- If $T[j + \ell] \prec T[j' + \ell]$, then we have $T[j..j + \ell + 1] \prec T[j'..j' + \ell + 1]$, which by $\ell + 1 \leq \ell_1$ implies $T^\infty[j..j + \ell_1] \prec T^\infty[j'..j' + \ell_1]$. Thus, in turn, by $\ell_1 \leq \ell_2$, implies $T^\infty[j..j + \ell_1]c^\infty \prec T^\infty[j'..j' + \ell_2]$, contradicting our assumption.
- On the other hand, if $T[j + \ell] \succ T[j' + \ell]$, then $T[j..j + \ell + 1] \succ T[j'..j' + \ell + 1]$, which by $\ell + 1 \leq \ell_1$ implies $T^\infty[j..j + \ell_1] \succ T^\infty[j'..j' + \ell_1]$. This in turn, by $\ell_1 \leq \ell_2$, implies $T^\infty[j..j + \ell_1] \succ T^\infty[j'..j' + \ell_2]$, contradicting the assumption.

We have thus proved $\ell \geq \ell_1$, i.e., $T^\infty[j..j + \ell_1] = T^\infty[j'..j' + \ell_1]$. □

Corollary 6.8. *For every $j \in [1..n]$ and $\ell \in \mathbb{Z}_+$, we have $\text{RangeBeg}_\ell(j) = |\{j' \in [1..n] : T^\infty[j'..j' + \ell] \prec T^\infty[j..j + \ell]\}|$ and $\text{RangeEnd}_\ell(j) = |\{j' \in [1..n] : T^\infty[j'..j' + \ell] \preceq T^\infty[j..j + \ell]\}| = \text{RangeBeg}_\ell(j) + |\text{Occ}_\ell(j)|$.*

6.2.2 Computing the Size of $\text{Pos}_\ell(j)$ and $\text{Occ}_{2\ell}(j)$

Let $j \in [1..n] \setminus \text{R}(\tau, T)$. In this section, we show how under Assumption 6.5 to efficiently compute the values $|\text{Pos}_\ell(j)|$ and $|\text{Occ}_{2\ell}(j)|$.

The section is organized as follows. First, we present two combinatorial results (Lemmas 6.10 and 6.11) characterizing the sets $\text{Pos}_\ell(j)$ and $\text{Occ}_{2\ell}(j)$ using the string synchronizing set \mathbf{S} . We then use these characterizations to prove a formula for the cardinality of these sets (Lemma 6.12). We conclude with Proposition 6.13 showing how under Assumption 6.5, to utilize this formula to quickly compute values $|\text{Pos}_\ell(j)|$ and $|\text{Occ}_{2\ell}(j)|$ given position j .

Remark 6.9. Before formally characterizing sets $\text{Pos}_\ell(j)$ and $\text{Occ}_{2\ell}(j)$, we first provide some intuition. By $j \notin \text{R}(\tau, T)$, the position $s = \text{succ}_\mathbf{S}(j)$ satisfies $s - j < \tau$. Thus, by the consistency of \mathbf{S} and $3\tau \leq \ell$, all $j' \in \text{Pos}_\ell(j)$ share a common offset $\delta_\mathbf{S} = s - j$ such that $j' + \delta_\mathbf{S} = \min(\mathbf{S} \cap [j'..j' + \tau])$ and hence the relative lexicographical order between $T[j'..n]$ and $T[j..n]$ is the same as between

$T[j' + \delta_S \dots n]$ and $T[j + \delta_S \dots n]$. Consequently, it suffices to only consider positions in \mathbf{S} . Then, computing $|\text{Pos}_\ell(j)|$ reduces to finding those $s' \in \mathbf{S}$ that are:

1. Preceded in T by the string $T[j \dots s]$, and
2. For which it holds $T[s' \dots n] \prec T[s \dots n]$ and $\text{LCE}_T(s', s) \in [\ell - \delta_S \dots 2\ell - \delta_S]$.

For any $q \geq 2\ell$, the above Condition 1 is equivalent to position s' having a reversed left length- q context in the lexicographical range $[X \dots X']$, where $\bar{X} = T[j \dots s]$ and $X' = Xc^\infty$ (where $c = \max \Sigma$), and Condition 2 is equivalent to position s' having a right length- q context in $[Y \dots Y']$, where $Y = T^\infty[s \dots j + \ell]$ and $Y' = T^\infty[s \dots j + 2\ell]$. Thus, counting such s' can be reduced to a two-dimensional weighted orthogonal range counting query on a set of points having length- q substrings of T^∞ or \bar{T}^∞ as coordinates (see Section 4). Crucially, since $\tau = \lfloor \frac{\ell}{3} \rfloor$ and $\ell \geq 16$, it suffices to choose $q = 7\tau$ to guarantee $q \geq 2\ell$, and we will later see that $q = \mathcal{O}(\tau)$ is the crucial property of this reduction that lets us generalize the index to the dynamic case.

The intuition for the computation of $|\text{Occ}_{2\ell}(j)|$ is similar, except we observe that Condition 2 is that s' satisfies $T^\infty[s' \dots j' + 2\ell] = T^\infty[s \dots j + 2\ell]$, which is equivalent to s' having a right length- q context in $[Y' \dots Y'c^\infty]$. Thus, we can count such s' (and consequently, compute $|\text{Occ}_{2\ell}(j)|$), using the same set of two-dimensional points as in the computation of $|\text{Pos}_\ell(j)|$.

Lemma 6.10. *Let $j \in [1 \dots n - 3\tau + 1] \setminus \mathbf{R}(\tau, T)$ and $s = \text{succ}_S(j)$. Let $X \in \Sigma^*$ and $Y, Y' \in \Sigma^+$ be such that $\bar{X} = T[j \dots s]$, $T^\infty[j \dots j + \ell] = \bar{X}Y$, and $T^\infty[j \dots j + 2\ell] = \bar{X}Y'$. Then, for any $j' \in [1 \dots n]$, letting $s' = j' + |X|$, it holds*

$$j' \in \text{Pos}_\ell(j) \text{ if and only if } s' \in \mathbf{S}, Y \preceq T^\infty[s' \dots s' + 7\tau] \prec Y', \text{ and } T^\infty[s' - |X| \dots s'] = \bar{X}.$$

Proof. Note that by the uniqueness of $T[n] = \$$, it holds $\text{per}(T[n - 3\tau + 2 \dots n]) = 3\tau - 1$ and hence $\mathbf{S} \cap [n - 3\tau + 2 \dots n - 2\tau + 2] \neq \emptyset$. Thus, by $j < n - 3\tau + 2$, $s = \text{succ}_S(j)$ is well-defined. Moreover, by $j \notin \mathbf{R}(\tau, T)$, it holds $\mathbf{S} \cap [j \dots j + \tau] \neq \emptyset$. Therefore, we have $|X| = s - j < \tau \leq \ell$, and hence the strings Y and Y' (of length $\ell - |X|$ and $2\ell - |X|$, respectively) are well-defined and nonempty.

Let $j' \in \text{Pos}_\ell(j)$, i.e., $T[j' \dots n] \prec T[j \dots n]$ and $\text{LCE}_T(j, j') \in [\ell \dots 2\ell]$. This implies $j \neq j'$ and $T[j' \dots j' + \ell] = T[j \dots j + \ell]$. Therefore, by $\ell - (s - j) \geq \ell - \tau \geq 2\tau$ and the consistency of the string synchronizing set \mathbf{S} (Definition 2.1) applied for positions $j_1 = j + t$ and $j_2 = j' + t$, where $t \in [0 \dots |X|]$, we obtain $\text{succ}_S(j') - j' = \text{succ}_S(j) - j$, or equivalently, $\text{succ}_S(j') = j' + (s - j) = s'$. Thus, it holds $s' \in \mathbf{S}$. Next, by $T[j' \dots j' + \ell] = T[j \dots j + \ell]$ and $|X| < \tau \leq \ell$, it holds $\text{LCE}_T(j, j') = |X| + \text{LCE}_T(s, s')$. Thus, $\text{LCE}_T(s, s') \in [\ell - |X| \dots 2\ell - |X|]$. By Item 1 in Lemma 6.7 (with parameters $\ell_1 = |Y| = \ell - |X|$, $\ell_2 = |Y'| = 2\ell - |X|$, and $\ell_3 = 7\tau$) and $2\ell \leq 7\tau$ (holding for $\ell \geq 16$), this implies $Y \preceq T^\infty[s' \dots s' + 7\tau] \prec Y'$. Finally, the equality $T^\infty[j' \dots j' + \ell] = T^\infty[j \dots j + \ell] = \bar{X}Y$ implies $T^\infty[s' - |X| \dots s'] = T^\infty[j' \dots s'] = \bar{X}$.

For the opposite implication, assume $s' \in \mathbf{S}$, $Y \preceq T^\infty[s' \dots s' + 7\tau] \prec Y'$, and $T^\infty[s' - |X| \dots s'] = \bar{X}$. By Item 1 in Lemma 6.7 (with the same parameters as above) and $2\ell \leq 7\tau$, this implies $T[s' \dots n] \prec T[s \dots n]$ and $\text{LCE}_T(s, s') \in [\ell - |X| \dots 2\ell - |X|]$. Since $T[j \dots s] = \bar{X}$ and by $s \in \mathbf{S}$ we have $s < n$, X does not contain the symbol $\$$, and hence $j' \geq 1$. Thus, $T^\infty[j' \dots s'] = X$ implies $T[j \dots s] = T[j' \dots s']$, and consequently, $T[j' \dots n] \prec T[j \dots n]$ and $\text{LCE}_T(j, j') = |X| + \text{LCE}_T(s, s') \in [\ell \dots 2\ell]$. Thus, $j' \in \text{Pos}_\ell(j)$. \square

Lemma 6.11. *Let $j \in [1 \dots n - 3\tau + 1] \setminus \mathbf{R}(\tau, T)$, $s = \text{succ}_S(j)$, and $d \in [\ell \dots 2\ell]$. Let $X \in \Sigma^*$ and $Y, Y' \in \Sigma^+$ be such that $\bar{X} = T[j \dots s]$, $T^\infty[j \dots j + \ell] = \bar{X}Y$, and $T^\infty[j \dots j + d] = \bar{X}Y'$. Then, for any $j' \in [1 \dots n]$, letting $s' = j' + |X|$ and $c = \max \Sigma$, it holds*

$$j' \in \text{Occ}_d(j) \text{ if and only if } s' \in \mathbf{S}, Y' \preceq T^\infty[s' \dots s' + 7\tau] \prec Y'c^\infty, \text{ and } T^\infty[s' - |X| \dots s'] = \bar{X}.$$

Proof. Similarly as in Lemma 6.10, we first observe that by $j < n - 3\tau + 2$, $\text{succ}_S(j)$ is well-defined. Moreover, by $j \notin R(\tau, T)$, it holds $S \cap [j..j + \tau] \neq \emptyset$. Therefore, $|X| = s - j < \tau \leq \ell$, and hence the strings Y and Y' (of length $\ell - |X|$ and $d - |X|$, respectively) are well-defined and nonempty.

Let $j' \in \text{Occ}_d(j)$, i.e., $T^\infty[j'..j' + d] = T^\infty[j..j + d]$. To show $s' \in S$, we consider two cases. If $j = j'$ then $s' = s \in S$ holds by definition. Otherwise, by $T^\infty[j..j + d] = T^\infty[j'..j' + d]$ and the uniqueness of $T[n] = \$$, we must have $T[j..j + \ell] = T[j'..j' + \ell]$. Thus, by $\ell - (s - j) \geq \ell - \tau \geq 2\tau$ and the consistency of S (applied as in the proof of Lemma 6.10) for $j_1 = j + t$ and $j_2 = j' + t$, where $t \in [0..|X|)$, we obtain $\text{succ}_S(j') = s'$. Thus, $s' \in S$. Next, by $T^\infty[s'..j' + d] = T^\infty[s..j + d]$ and $d \leq 2\ell \leq 7\tau$, we obtain from Item 3 in Lemma 6.7 (with parameters $\ell_1 = |Y'| = d - |X|$ and $\ell_2 = 7\tau$) that $Y' \preceq T^\infty[s'..s' + 7\tau] \prec Y'c^\infty$. Finally, $T^\infty[j'..j' + \ell] = T^\infty[j..j + \ell] = \overline{XY}$ implies $T^\infty[s' - |X|..s'] = T^\infty[j'..s'] = \overline{X}$, i.e., the third condition.

For the opposite implication, assume $s' \in S$, $Y' \preceq T^\infty[s'..s' + 7\tau] \prec Y'c^\infty$, and $T^\infty[s' - |X|..s'] = \overline{X}$. By Item 3 in Lemma 6.7 (with the same parameters as above) and $d \leq 2\ell \leq 7\tau$, this implies $T^\infty[s'..j' + d] = T^\infty[s..j + d]$. Combining this with the assumption $T^\infty[j'..s'] = \overline{X} = T^\infty[j..s]$, we obtain $T^\infty[j'..j' + d] = T^\infty[j..j + d]$, i.e., $j' \in \text{Occ}_d(j)$. \square

Lemma 6.12. *Let $j \in [1..n - 3\tau + 1] \setminus R(\tau, T)$ and $s = \text{succ}_S(j)$. Let $X \in \Sigma^*$ and $X', Y, Y' \in \Sigma^+$ be such that $\overline{X} = T[j..s]$, $X' = Xc^\infty$ (with $c = \max \Sigma$), $T^\infty[j..j + \ell] = \overline{XY}$, and $T^\infty[j..j + 2\ell] = \overline{XY'}$. Then, letting $q = 7\tau$ and $\mathcal{P} = \text{Points}_q(T, S)$, it holds:*

1. $|\text{Pos}_\ell(j)| = \text{r-count}_{\mathcal{P}}(X, X', Y') - \text{r-count}_{\mathcal{P}}(X, X', Y)$ and
2. $|\text{Occ}_{2\ell}(j)| = \text{r-count}_{\mathcal{P}}(X, X', Y'c^\infty) - \text{r-count}_{\mathcal{P}}(X, X', Y')$.

Proof. 1. By Lemma 6.10, we can write $\text{Pos}_\ell(j) = \{s' - |X| : s' \in S, Y \preceq T^\infty[s'..s' + 7\tau] \prec Y', \text{ and } T^\infty[s' - |X|..s'] = \overline{X}\}$. On the other hand, by Definition 4.2 and the definition of the rcount query, we have:

$$\begin{aligned} \text{r-count}_{\mathcal{P}}(X, X', Y) &= |\{s' \in S : T^\infty[s'..s' + 7\tau] \prec Y \text{ and } X \preceq \overline{T^\infty[s' - 7\tau..s']} \prec X'\}| \\ &= |\{s' \in S : T^\infty[s'..s' + 7\tau] \prec Y \text{ and } X \text{ is a prefix of } \overline{T^\infty[s' - 7\tau..s']}\}| \\ &= |\{s' \in S : T^\infty[s'..s' + 7\tau] \prec Y \text{ and } T^\infty[s' - |X|..s'] = \overline{X}\}|. \end{aligned}$$

Analogously, $\text{r-count}_{\mathcal{P}}(X, X', Y') = |\{s' \in S : T^\infty[s'..s' + 7\tau] \prec Y' \text{ and } T^\infty[s' - |X|..s'] = \overline{X}\}|$. Since any position $s' \in S$ that satisfies $T^\infty[s'..s' + 7\tau] \prec Y$ also satisfies $T^\infty[s'..s' + 7\tau] \prec Y'$, we obtain $\text{r-count}_{\mathcal{P}}(X, X', Y') - \text{r-count}_{\mathcal{P}}(X, X', Y) = |\{s' \in S : Y \preceq T^\infty[s'..s' + 7\tau] \prec Y' \text{ and } T^\infty[s' - |X|..s'] = \overline{X}\}|$. The cardinality of this set is clearly the same as the earlier set characterizing $\text{Pos}_\ell(j)$. Thus, $\text{r-count}_{\mathcal{P}}(X, X', Y') - \text{r-count}_{\mathcal{P}}(X, X', Y) = |\text{Pos}_\ell(j)|$.

2. By Lemma 6.11, we have $\text{Occ}_{2\ell}(j) = \{s' - |X| : s' \in S, Y' \preceq T^\infty[s'..s' + 7\tau] \prec Y'c^\infty, \text{ and } T^\infty[s' - |X|..s'] = \overline{X}\}$. On the other hand, by Definition 4.2 and the definition of rcount, we have $\text{r-count}_{\mathcal{P}}(X, X', Y') = |\{s' \in S : T^\infty[s'..s' + 7\tau] \prec Y' \text{ and } T^\infty[s' - |X|..s'] = \overline{X}\}|$ and $\text{r-count}_{\mathcal{P}}(X, X', Y'c^\infty) = |\{s' \in S : T^\infty[s'..s' + 7\tau] \prec Y'c^\infty \text{ and } T^\infty[s' - |X|..s'] = \overline{X}\}|$. Since any position $s' \in S$ that satisfies $T^\infty[s'..s' + 7\tau] \prec Y'$ also satisfies $T^\infty[s'..s' + 7\tau] \prec Y'c^\infty$, we thus obtain $\text{r-count}_{\mathcal{P}}(X, X', Y'c^\infty) - \text{r-count}_{\mathcal{P}}(X, X', Y') = |\{s' \in S : Y' \preceq T^\infty[s'..s' + 7\tau] \prec Y'c^\infty \text{ and } T^\infty[s' - |X|..s'] = \overline{X}\}|$. The cardinality of this set is clearly the same as the earlier set characterizing $\text{Occ}_{2\ell}(j)$. Thus, $\text{r-count}_{\mathcal{P}}(X, X', Y'c^\infty) - \text{r-count}_{\mathcal{P}}(X, X', Y') = |\text{Occ}_{2\ell}(j)|$. \square

Proposition 6.13. *Under Assumption 6.5, given a position $j \in [1..n] \setminus R(\tau, T)$, we can compute $|\text{Pos}_\ell(j)|$ and $|\text{Occ}_{2\ell}(j)|$ in $\mathcal{O}(t)$ time.*

Proof. We first check if $j > n - 3\tau + 1$. If yes, then by the uniqueness of $T[n] = \$$, it holds $|\text{Occ}_\ell(j)| = 1$. By $\text{Occ}_{2\ell}(j) \neq \emptyset$, $\text{Occ}_{2\ell}(j) \subseteq \text{Occ}_\ell(j)$, we can therefore return $|\text{Pos}_\ell(j)| = 0$ and $|\text{Occ}_{2\ell}(j)| = 1$. Let us thus assume $j \leq n - 3\tau + 1$ and recall from the proof of Lemma 6.10 that then $\text{succ}_\mathbb{S}(j)$ is well-defined. Using Assumption 6.5 we compute $s = \text{succ}_\mathbb{S}(j)$. Let $X \in \Sigma^*$ and $X', Y, Y' \in \Sigma^+$ be such that $\bar{X} = T[j \dots s]$, $X' = Xc^\infty$, $T^\infty[j \dots j + \ell] = \bar{X}Y$, and $T^\infty[j \dots j + 2\ell] = \bar{X}Y'$. Then:

1. By Lemma 6.12, we have $|\text{Pos}_\ell(j)| = \text{r-count}_{\mathcal{P}}(X, X', Y') - \text{r-count}_{\mathcal{P}}(X, X', Y)$ (where $\mathcal{P} = \text{Points}_{7\tau}(T, \mathbb{S})$) which under Assumption 6.5 we can efficiently compute using the query arguments $(i, q_l, q_r) = (s, s - j, 2\ell - (s - j))$ and then with arguments $(i, q_l, q_r) = (s, s - j, \ell - (s - j))$ (see Problem 4.3). By $j \notin \mathbb{R}(\tau, T)$ and the density property of \mathbb{S} , we have $s - j < \tau \leq \ell < n$. On the other hand, $q_r \leq 2\ell - (s - j) \leq 2\ell < 2n$. Thus, the arguments q_l and q_r of both queries satisfy the requirements in Problem 4.3.
2. By Lemma 6.12, we also have $|\text{Occ}_{2\ell}(j)| = \text{r-count}_{\mathcal{P}}(X, X', Y'^{c^\infty}) - \text{r-count}_{\mathcal{P}}(X, X', Y')$ (where \mathcal{P} is defined as above), which under Assumption 6.5 we can compute using the query arguments $(i, q_l, q_r) = (s, s - j, 2\ell - (s - j))$ (see Problem 4.3). As noted above, these arguments satisfy the requirements in Problem 4.3.

By Assumption 6.5, the query takes $\mathcal{O}(t)$ time in total. □

6.2.3 Computing a Position in $\text{Occ}_{2\ell}(\text{SA}[i])$

Assume that $i \in [1 \dots n]$ satisfies $\text{SA}[i] \in [1 \dots n] \setminus \mathbb{R}(\tau, T)$. In this section, we show how under Assumption 6.5, given the index i along with values $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$, and some position $j \in \text{Occ}_\ell(\text{SA}[i])$, to efficiently compute some position $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$.

The section is organized as follows. First, we present a combinatorial result (Lemma 6.14) that reduces the computation of $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ to a generalized range selection query (see Section 4). We then use this reduction to present the query algorithm for the computation of some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ in Proposition 6.15.

Lemma 6.14. *Assume $i \in [1 \dots n]$ is such that $\text{SA}[i] \in [1 \dots n - 3\tau + 1] \setminus \mathbb{R}(\tau, T)$. Denote $b = \text{RangeBeg}_\ell(\text{SA}[i])$, $d = |\text{Occ}_\ell(\text{SA}[i])|$, and $s = \text{succ}_\mathbb{S}(\text{SA}[i])$. Let $X \in \Sigma^*$ and $X', Y \in \Sigma^+$ be such that $\bar{X} = T[\text{SA}[i] \dots s]$, $X' = Xc^\infty$ (with $c = \max \Sigma$), and $T^\infty[\text{SA}[i] \dots \text{SA}[i] + \ell] = \bar{X}Y$. Let also $\mathcal{P} = \text{Points}_{7\tau}(T, \mathbb{S})$, $m = \text{r-count}_{\mathcal{P}}(X, X', Y)$, and $m' = \text{r-count}_{\mathcal{P}}(X, X')$. Then, $m + d \leq m'$. Moreover:*

1. *For $\delta \in [1 \dots d]$, any position $p \in \text{r-select}_{\mathcal{P}}(X, X', m + \delta)$ satisfies $T^\infty[p - |X| \dots p - |X| + 2\ell] = T^\infty[\text{SA}[b + \delta] \dots \text{SA}[b + \delta] + 2\ell]$.*
2. *For $\delta = i - b$, any position $p \in \text{r-select}_{\mathcal{P}}(X, X', m + \delta)$ satisfies $p - |X| \in \text{Occ}_{2\ell}(\text{SA}[i])$.*

Proof. Note that by the uniqueness of $T[n] = \$$, it holds $\text{per}(T[n - 3\tau + 2 \dots n]) = 3\tau - 1$ and hence $\mathbb{S} \cap [n - 3\tau + 2 \dots n - 2\tau + 2] \neq \emptyset$. Thus, by $\text{SA}[i] < n - 3\tau + 2$, $s = \text{succ}_\mathbb{S}(\text{SA}[i])$ is well-defined. Denote $q = |\mathbb{S}|$. Let $(a_j)_{j \in [1 \dots q]}$ be a sequence containing all positions $p \in \mathbb{S}$ ordered according to the string $T^\infty[p \dots p + 7\tau]$. In other words, for any $j, j' \in [1 \dots q]$, $j < j'$ implies $T^\infty[a_j \dots a_j + 7\tau] \preceq T^\infty[a_{j'} \dots a_{j'} + 7\tau]$. Note, that the sequence $(a_j)_{j \in [1 \dots q]}$ is not unique. Since $\{a_j : j \in [1 \dots q]\} = \mathbb{S}$, it holds $|\{a_j - |X| : j \in [1 \dots q]\}| = |\{j \in [1 \dots q] : T^\infty[a_j - |X| \dots a_j] = \bar{X}\}| = m'$, where the last equality follows by Item 3 of Lemma 6.7 and the definition of $\text{r-count}_{\mathcal{P}}(X, X')$ (see Section 4.1). Moreover, by the same argument (utilizing the definition of $\text{r-count}_{\mathcal{P}}(X, X', Y)$ instead of $\text{r-count}_{\mathcal{P}}(X, X')$), we have $|\{a_j - |X| : j \in [1 \dots q], T^\infty[a_j - |X| \dots a_j] = \bar{X}\}| = m$ and $T^\infty[a_j \dots a_j + 7\tau] \prec Y$.

$T^\infty[a_j - |X| \dots a_j] = \overline{X}$ and $T^\infty[a_j \dots a_j + 7\tau) \prec Y\} = m$. On the other hand, observe that by Lemma 6.11, for any $j \in [1 \dots n]$, it holds $j \in \text{Occ}_\ell(\text{SA}[i])$ if and only if $j + |X| \in \mathbf{S}$, $T^\infty[j \dots j + |X|] = \overline{X}$, and $Y \preceq T^\infty[j + |X| \dots j + |X| + 7\tau) \prec Yc^\infty$. In other words, $\text{Occ}_\ell(\text{SA}[i]) = \{a_j - |X| : j \in [1 \dots q], T^\infty[a_j - |X| \dots a_j] = \overline{X}, \text{ and } Y \preceq T^\infty[a_j \dots a_j + 7\tau) \prec Yc^\infty\}$. The latter set (whose cardinality is equal to d) is clearly a subset of $\{a_j - |X| : j \in [1 \dots q] \text{ and } T^\infty[a_j - |X| \dots a_j] = \overline{X}\}$ (whose cardinality, as shown above, is equal to m'). Thus, $d \leq m'$. On the other hand, the set $\{a_j - |X| : j \in [1 \dots q], T^\infty[a_j - |X| \dots a_j] = \overline{X}, \text{ and } T^\infty[a_j \dots a_j + 7\tau) \prec Y\}$ (whose cardinality, as shown above, is m) is also clearly a subset of $\{a_j - |X| : j \in [1 \dots q] \text{ and } T^\infty[a_j - |X| \dots a_j] = \overline{X}\}$ (whose cardinality is m'). Thus, $m \leq m'$. Since $j \in [1 \dots q]$ cannot simultaneously satisfy $T^\infty[a_j \dots a_j + 7\tau) \prec Y$ and $Y \preceq T^\infty[a_j \dots a_j + 7\tau)$, these subsets are disjoint. Hence, $m + d \leq m'$.

1. As shown above, $|\{j \in [1 \dots q] : T^\infty[a_j - |X| \dots a_j] = \overline{X}\}| = m'$. Let $(b_j)_{j \in [1 \dots m']}$ be a subsequence of $(a_j)_{j \in [1 \dots q]}$ containing all elements of $\{a_j : j \in [1 \dots q] \text{ and } T^\infty[a_j - |X| \dots a_j] = \overline{X}\}$ (in the same order as they appear in the sequence $(a_j)_{j \in [1 \dots q]}$). Our proof consists of three steps:

- (i) Let $j \in [1 \dots m']$. We start by showing that $b_j \in \text{r-select}_\mathcal{P}(X, X', j)$. Let Q, Q' be such that $\overline{Q} = T^\infty[b_j - 7\tau \dots b_j]$ and $Q' = T^\infty[b_j \dots b_j + 7\tau)$. Let

$$\begin{aligned} r_{\text{beg}} &= |\{a_t : t \in [1 \dots q], T^\infty[a_t - |X| \dots a_t] = \overline{X}, \text{ and } T^\infty[a_t \dots a_t + 7\tau) \prec Q'\}| \text{ and} \\ r_{\text{end}} &= |\{a_t : t \in [1 \dots q], T^\infty[a_t - |X| \dots a_t] = \overline{X}, \text{ and } T^\infty[a_t \dots a_t + 7\tau) \preceq Q'\}|. \end{aligned}$$

If $t \in [1 \dots q]$ satisfies $T^\infty[a_t - |X| \dots a_t] = \overline{X}$, then $a_t \in \{b_1, \dots, b_m\}$. Moreover, since for any $t, t' \in [1 \dots m]$, $t < t'$ implies $T^\infty[b_t \dots b_t + 7\tau) \preceq T^\infty[b_{t'} \dots b_{t'} + 7\tau)$, any $t \in [1 \dots q]$ that additionally satisfies $T^\infty[a_t \dots a_t + 7\tau) \prec T^\infty[b_j \dots b_j + 7\tau)$, also satisfies $a_t \in \{b_1, \dots, b_{j-1}\}$. Thus, $r_{\text{beg}} < j$. On the other hand, every $t \in [1 \dots j]$ satisfies $T^\infty[b_t - |X| \dots b_t] = \overline{X}$ and $T^\infty[b_t \dots b_t + 7\tau) \preceq T^\infty[b_j \dots b_j + 7\tau)$. Thus, $j \leq r_{\text{end}}$. Altogether, $j \in (r_{\text{beg}} \dots r_{\text{end}}]$. Recall now the definition $\mathcal{P} = \text{Points}_{7\tau}(T, \mathbf{S})$ (Definition 4.4) and note that by Item 3 of Lemma 6.7, it holds $r_{\text{beg}} = \text{r-count}_\mathcal{P}(X, X', Q')$ and $r_{\text{end}} = \text{r-count}_\mathcal{P}^{\text{inc}}(X, X', Q')$. We thus obtain that $j \in (\text{r-count}_\mathcal{P}(X, X', Q') \dots \text{r-count}_\mathcal{P}^{\text{inc}}(X, X', Q'))$. On the other hand, $(Q, Q', b_j) \in \mathcal{P}$ and $T^\infty[b_j - |X| \dots b_j] = \overline{X}$, so $b_j \in \text{r-select}_\mathcal{P}(X, X', j)$ holds as claimed.

- (ii) Let $j \in [1 \dots m']$. We will now show that, for any $p \in \text{r-select}_\mathcal{P}(X, X', j)$, it holds $T^\infty[p - |X| \dots p + 7\tau) = T^\infty[b_j - |X| \dots b_j + 7\tau)$. By Item (i) and the definition of $\text{r-select}_\mathcal{P}(X, X', j)$, the assumption $p \in \text{r-select}_\mathcal{P}(X, X', j)$ implies $T^\infty[p \dots p + 7\tau) = T^\infty[b_j \dots b_j + 7\tau)$. Moreover, letting Q be such that $\overline{Q} = T^\infty[p - |X| \dots p]$, it also implies $X \preceq Q \prec X'$. Thus, by Item 3 of Lemma 6.7, p is preceded by X in T . Since by definition of $(b_j)_{j \in [1 \dots m']}$, the position b_j is also preceded by X in T , we obtain $T^\infty[p - |X| \dots p + 7\tau) = T^\infty[b_j - |X| \dots b_j + 7\tau)$.

- (iii) We are now ready to prove the main claim. As observed above, $\text{Occ}_\ell(\text{SA}[i]) = \{a_j - |X| : j \in [1 \dots q], T^\infty[a_j - |X| \dots a_j] = \overline{X}, \text{ and } Y \preceq T^\infty[a_j \dots a_j + 7\tau) \prec Yc^\infty\}$. Note, that since the positions k in the sequence $(a_j)_{j \in [1 \dots q]}$ are sorted by $T^\infty[k \dots k + 7\tau)$, we can simplify the second condition. Denoting $j_{\text{skip}} = |\{j \in [1 \dots q] : T^\infty[a_j \dots a_j + 7\tau) \prec Y\}|$, we have

$$\text{Occ}_\ell(\text{SA}[i]) = \left\{ a_j - |X| : \begin{array}{l} j \in (j_{\text{skip}} \dots q], T^\infty[a_j - |X| \dots a_j] = \overline{X}, \\ \text{and } T^\infty[a_j \dots a_j + 7\tau) \prec Yc^\infty \end{array} \right\}.$$

Let us now estimate $|\{j \in [1 \dots j_{\text{skip}}] : T^\infty[a_j - |X| \dots a_j] = \overline{X}\}|$. Any j in this set satisfies $j \in [1 \dots q]$, $T^\infty[a_j - |X| \dots a_j] = \overline{X}$, and $T^\infty[a_j \dots a_j + 7\tau) \prec Y$. Earlier we observed that the number of such j is precisely m . Combining this fact with the above formula for $\text{Occ}_\ell(\text{SA}[i])$ and the definition of $(b_j)_{j \in [1 \dots m']}$, we have $\text{Occ}_\ell(\text{SA}[i]) = \{b_j - |X| : j \in (m \dots m + d)\}$.

On the other hand, $b + d = \text{RangeBeg}_\ell(\text{SA}[i]) + |\text{Occ}_\ell(\text{SA}[i])| = \text{RangeEnd}_\ell(\text{SA}[i])$. Thus, $\text{Occ}_\ell(\text{SA}[i]) = \{\text{SA}[j] : j \in (b..b+d)\}$. We now observe:

- Let $j_1, j_2 \in (m..m+d]$ and assume $j_1 < j_2$. Since the elements of (b_j) occur in the same order as in (a_j) , and positions p in (a_j) are sorted by $T[p..p+7\tau]$, it follows that $T^\infty[b_{j_1}..b_{j_1}+7\tau] \preceq T^\infty[b_{j_2}..b_{j_2}+7\tau]$. On the other hand, by definition of (b_j) , both positions b_{j_1} and b_{j_2} are preceded in T by the string \bar{X} . Thus, $T^\infty[b_{j_1} - |X|..b_{j_2} + 7\tau] \preceq T^\infty[b_{j_2} - |X|..b_{j_2} + 7\tau]$.
- On the other hand, by definition of lexicographical order, for any $j_1, j_2 \in [1..d]$, the assumption $j_1 < j_2$ implies $T^\infty[\text{SA}[b+j_1].. \text{SA}[b+j_1] + |X| + 7\tau] \preceq T^\infty[\text{SA}[b+j_2].. \text{SA}[b+j_2] + |X| + 7\tau]$.

We have thus shown that both sequences $\text{SA}[b+1], \dots, \text{SA}[b+d]$ and $b_{m+1} - |X|, \dots, b_{m+d} - |X|$ contain the same set of positions $\text{Occ}_\ell(\text{SA}[i])$ ordered according to the length- $(|X| + 7\tau)$ right context in T^∞ . Therefore, regardless of how ties are resolved in each sequence, for any $\delta \in [1..d]$, we have

$$T^\infty[\text{SA}[b+\delta].. \text{SA}[b+\delta] + |X| + 7\tau] = T^\infty[b_{m+\delta} - |X|.. b_{m+\delta} + 7\tau].$$

To finalize the proof of the claim, take any $p \in \text{r-select}_P(X, X', m+\delta)$. By Item (ii), for $j = m+\delta$, we have $T^\infty[p - |X|.. p + 7\tau] = T^\infty[b_{m+\delta} - |X|.. b_{m+\delta} + 7\tau] = T^\infty[\text{SA}[b+\delta].. \text{SA}[b+\delta] + |X| + 7\tau]$. In particular, by $0 \leq |X|$ and $2\ell \leq 7\tau$, we obtain $2\ell \leq |X| + 7\tau$ and $T^\infty[p - |X|.. p - |X| + 2\ell] = T^\infty[\text{SA}[b+\delta].. \text{SA}[b+\delta] + 2\ell]$, i.e., the claim.

2. Applying Item 1 for $\delta = i - b$, we conclude that any $p \in \text{r-select}_P(X, X', m + \delta)$, satisfies $T^\infty[p - |X|.. p - |X| + 2\ell] = T^\infty[\text{SA}[b+\delta].. \text{SA}[b+\delta] + 2\ell] = T^\infty[\text{SA}[i].. \text{SA}[i] + 2\ell]$, i.e., $p - |X| \in \text{Occ}_{2\ell}(\text{SA}[i])$. \square

Proposition 6.15. *Let $i \in [1..n]$ be such that $\text{SA}[i] \in [1..n] \setminus \text{R}(\tau, T)$. Under Assumption 6.5, given the values i , $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$, and some $j \in \text{Occ}_\ell(\text{SA}[i])$ as input, we can compute some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ in $\mathcal{O}(t)$ time.*

Proof. We first calculate $|\text{Occ}_\ell(\text{SA}[i])| = \text{RangeEnd}_\ell(\text{SA}[i]) - \text{RangeBeg}_\ell(\text{SA}[i])$ using the input arguments. If $|\text{Occ}_\ell(\text{SA}[i])| = 1$, then by $\text{Occ}_{2\ell}(\text{SA}[i]) \neq \emptyset$ and $\text{Occ}_{2\ell}(\text{SA}[i]) \subseteq \text{Occ}_\ell(\text{SA}[i])$ we must have $\text{RangeBeg}_{2\ell}(\text{SA}[i]) = \text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_{2\ell}(\text{SA}[i]) = \text{RangeEnd}_\ell(\text{SA}[i])$ and $\text{Occ}_{2\ell} = \{j\}$. Thus, we return $j' := j$. Let us thus assume $|\text{Occ}_\ell(\text{SA}[i])| > 1$, and observe that by the uniqueness of $T[n] = \$$, this implies $\text{SA}[i], j \in [1..n-3\tau+1]$. Moreover, by $3\tau-1 \leq \ell$, $j \in \text{Occ}_\ell(\text{SA}[i])$, and $\text{SA}[i] \notin \text{R}(\tau, T)$, we also have $j \notin \text{R}(\tau, T)$. Therefore, as noted in the proof of Lemma 6.10, $\text{succ}_5(j)$ is well-defined and using Assumption 6.5 we can compute $s = \text{succ}_5(j)$ in $\mathcal{O}(t)$ time. Let $X \in \Sigma^*$ and $X', Y \in \Sigma^+$ be such that $\bar{X} = T[j..s]$, $X' = Xc^\infty$, and $T^\infty[j..j+\ell] = \bar{X}Y$ (where $c = \max \Sigma$). Observe now that by $j \in \text{Occ}_\ell(\text{SA}[i])$, we have $T^\infty[j..j+\ell] = T^\infty[\text{SA}[i].. \text{SA}[i] + \ell]$. We also have $3\tau - 1 \leq \ell$. Thus, by the consistency condition of \mathbf{S} (Definition 2.1), for any $t \in [0.. \tau]$, $\text{SA}[i] + t \in \mathbf{S}$ holds if and only if $j + t \in \mathbf{S}$. Recall, however, that we assumed $\text{SA}[i] \notin \text{R}(\tau, T)$, i.e., $\mathbf{S} \cap [\text{SA}[i].. \text{SA}[i] + \tau] \neq \emptyset$. Therefore, denoting $s' = \text{succ}_5(\text{SA}[i])$, it holds $s' - \text{SA}[i] = s - j$. Consequently, it holds $T^\infty[\text{SA}[i].. s'] = T^\infty[j..s] = \bar{X}$ and $T^\infty[s'.. \text{SA}[i] + \ell] = T^\infty[s..j+\ell] = Y$. We can thus apply Lemma 6.14 without knowing the values of $\text{SA}[i]$ or s' (we only need to know $|X|$ and the starting position of some occurrence of $\bar{X}Y$ in T). First, using Item 1 of Problem 4.3 with the query arguments $(i, q_l, q_r) = (s, s - j, \ell - (s - j))$ we compute in $\mathcal{O}(t)$ time (which is possible under Assumption 6.5) the value $m := \text{r-count}_P(X, X', Y)$ (the arguments satisfy the

requirements of Problem 4.3 since $q_l = s - j < \tau \leq \ell < n$ and $q_r = \ell - (s - j) \leq \ell < n$). We then calculate $\delta = i - \text{RangeBeg}_\ell(\text{SA}[i])$ and using Item 2 of Problem 4.3 with the query arguments $(i, q_l, r) = (s, j - s, m + \delta)$ we compute in $\mathcal{O}(t)$ time some position $p \in \text{r-select}_P(X, X', m + \delta)$ (the arguments satisfy the requirements of Problem 4.3 by the argument as above). By Lemma 6.14, we then have $p - q_l \in \text{Occ}_{2\ell}(\text{SA}[i])$. In total, we spend $\mathcal{O}(t)$ time. \square

6.2.4 The Data Structure

By combining the above results, we obtain that under Assumption 6.5, given an index $i \in [1..n]$ satisfying $\text{SA}[i] \in [1..n] \setminus \text{R}(\tau, T)$, along with $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$ and some $j \in \text{Occ}_\ell(\text{SA}[i])$ as input, we can efficiently compute $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$ and some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$.

Proposition 6.16. *Let $i \in [1..n]$ be such that $\text{SA}[i] \in [1..n] \setminus \text{R}(\tau, T)$. Under Assumption 6.5, given the index i along with values $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$, and some position $j \in \text{Occ}_\ell(\text{SA}[i])$, we can compute $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$ and some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ in $\mathcal{O}(t)$ time.*

Proof. First, using Proposition 6.15, we compute some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$. This takes $\mathcal{O}(t)$ time and all the required values $(i, \text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]), \text{and some } j \in \text{Occ}_\ell(\text{SA}[i]))$ are given as input. We now observe that since for j' we have $T^\infty[j'..j' + 2\ell] = T^\infty[\text{SA}[i].. \text{SA}[i] + 2\ell]$, we have $j'' \in \text{Occ}_{2\ell}(\text{SA}[i])$ if and only if $j'' \in \text{Occ}_{2\ell}(j')$. Thus, $\text{Occ}_{2\ell}(\text{SA}[i]) = \text{Occ}_{2\ell}(j')$. On the other hand, by Item 1 of Lemma 6.7, we have $j'' \in \text{Pos}_\ell(\text{SA}[i])$ if and only if $T^\infty[\text{SA}[i].. \text{SA}[i] + \ell] \preceq T^\infty[j''..j'' + 2\ell] \prec T^\infty[\text{SA}[i].. \text{SA}[i] + 2\ell]$, i.e., whether $j'' \in \text{Pos}_\ell(\text{SA}[i])$ depends only on $T^\infty[\text{SA}[i].. \text{SA}[i] + 2\ell]$. Thus, $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ implies $\text{Pos}_\ell(\text{SA}[i]) = \text{Pos}_\ell(j')$. Therefore, in the second step of the query, using Proposition 6.13 we compute in $\mathcal{O}(t)$ time the values

$$\begin{aligned} \delta_\ell(\text{SA}[i]) &:= |\text{Pos}_\ell(j')| = |\text{Pos}_\ell(\text{SA}[i])| \text{ and} \\ m &:= |\text{Occ}_{2\ell}(j')| = |\text{Occ}_{2\ell}(\text{SA}[i])|. \end{aligned}$$

Letting $b = \text{RangeBeg}_\ell(\text{SA}[i])$, it then holds $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i])) = (b + \delta_\ell(\text{SA}[i]), b + \delta_\ell(\text{SA}[i]) + m)$. In total, the query takes $\mathcal{O}(t)$ time. \square

6.3 The Periodic Positions

In this section, we show that assuming we can compute basic characteristic of periodic positions and perform some int-string range and modular constraint queries (Assumption 6.21), given any position $i \in [1..n]$ satisfying $\text{SA}[i] \in \text{R}(\lfloor \frac{\ell}{3} \rfloor, T)$, along with some position $j \in \text{Occ}_\ell(\text{SA}[i])$ and the pair $(\text{RangeBeg}_\ell(\text{SA}[i]), \text{RangeEnd}_\ell(\text{SA}[i]))$ as input, we can efficiently compute some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ and the pair $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$.

The section is organized into nine parts. First (Section 6.3.1) we recall the standard definitions and notation for string synchronizing sets [KK19], that we will use to formalize the combinatorial properties used in our algorithms and then present a main assumption that we will use throughout this section. Next (Section 6.3.2), we formulate the main idea of the algorithm by describing three sets $\text{Pos}_\ell^{\text{low}}(\text{SA}[i])$, $\text{Pos}_\ell^{\text{mid}}(\text{SA}[i])$, and $\text{Pos}_\ell^{\text{high}}(\text{SA}[i])$, which play a central role in the SA query algorithm, and prove the combinatorial result showing how their sizes relate to the size of $\text{Pos}_\ell(\text{SA}[i])$. The following six sections (Section 6.3.3–Section 6.3.8) describe the individual steps of the query algorithm

in the order in which are used (with some minor exceptions). In Section 6.3.9 we put everything together to obtain the algorithm that computes the pair $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$ and some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ for $i \in [1..n]$ satisfying $\text{SA}[i] \in \mathbf{R}$.

6.3.1 Preliminaries

In this section, we define preliminary concepts used for answering SA queries for periodic positions.

The section is organized as follows. First, we recall basic definitions and notation for periodic positions in string synchronizing sets. We then recall two results characterizing blocks of periodic positions in lexicographical order (Lemma 6.18) and in text order (Lemma 6.19). We conclude with the central assumption that we will use all through the section.

Definition 6.17. A function $f : \Sigma^+ \rightarrow \Sigma^+$ is said to be *necklace-consistent* if it satisfies the following conditions for every $S, S' \in \Sigma^+$:

1. The strings $f(S)$ and S are cyclically equivalent.
2. If S and S' are cyclically equivalent, then $f(S) = f(S')$.

Let $S \in \Sigma^+$ and $\tau \in \mathbb{Z}_+$, and let f be some necklace-consistent function. For any $X \in \Sigma^+$, we define $\text{root}_f(X) := f(X[1..p])$, where $p = \text{per}(X)$.⁴ For any position $j \in \mathbf{R}(\tau, S)$, we then let $\text{root}_f(\tau, S, j) = \text{root}_f(S[j..j + 3\tau - 1])$. We denote $\text{Roots}_f(\tau, S) = \{\text{root}_f(\tau, S, j) : j \in \mathbf{R}(\tau, S)\}$. Next, we define the *run-decomposition*. For any $j \in \mathbf{R}(\tau, S)$, let $e(\tau, S, j) := \min\{j' \in [j..n] : j' \notin \mathbf{R}(\tau, S)\} + 3\tau - 2$. By [KK19, Fact 3.2], for $j \in \mathbf{R}(\tau, S)$, the substring $S[j..e(\tau, S, j))$ is the longest prefix of $S[j..n]$ that has a period $|\text{root}_f(\tau, S, j)|$. Moreover, by definition of root, letting $p = |\text{root}_f(\tau, S, j)|$, there exists $s \in [0..p)$ such that $S[j + s..j + s + p) = \text{root}_f(\tau, S, j)$. Thus, for every $j \in \mathbf{R}(\tau, S)$, we can write $S[j..e(\tau, S, j)) = H'H^kH''$, where $H = \text{root}_f(\tau, S, j)$, and H' (resp. H'') is a proper suffix (resp. prefix) of H . Note that there is always only one way to write $S[j..e(\tau, S, j))$ in this way, since the opposite would contradict the synchronization property of primitive strings [CHL07, Lemma 1.11]. In other words, for any fixed f , the run-decomposition is unique. We denote $\text{head}_f(\tau, S, j) = |H'|$, $\text{exp}_f(\tau, S, j) = k$, and $\text{tail}_f(\tau, S, j) = |H''|$. For $j \in \mathbf{R}(\tau, S, j)$, we let $\text{type}(\tau, S, j) = +1$ if $e(\tau, S, j) \leq |S|$ and $S[e(\tau, S, j)] \succ S[e(\tau, S, j) - p]$ (where $p = |\text{root}_f(\tau, S, j)|$), and $\text{type}(\tau, S, j) = -1$ otherwise.⁵ For any $j \in \mathbf{R}(\tau, S)$ and $t \geq 3\tau - 1$, we define $\text{exp}_f^{\text{cut}}(\tau, S, j, t) := \min(\text{exp}_f(\tau, S, j), \lfloor \frac{t-s}{|H|} \rfloor)$ and $e_f^{\text{cut}}(\tau, S, j, t) := j + s + \text{exp}_f^{\text{cut}}(\tau, S, j, t)|H|$, where $s = \text{head}_f(\tau, S, j)$ and $H = \text{root}_f(\tau, S, j)$. We denote $e_f^{\text{full}}(\tau, S, j) := e_f^{\text{cut}}(\tau, S, j, n)$, $e_f^{\text{low}}(\tau, S, j) := e_f^{\text{cut}}(\tau, S, j, \ell)$, and $e_f^{\text{high}}(\tau, S, j) := e_f^{\text{cut}}(\tau, S, j, 2\ell)$. Observe that it holds $e_f^{\text{full}}(\tau, S, j) = j + s + \text{exp}_f(\tau, S, j)|H|$.

We will repeatedly refer to the following subsets of $\mathbf{R}(\tau, S)$. First, we denote $\mathbf{R}^-(\tau, S) = \{j \in \mathbf{R}(\tau, S) : \text{type}(\tau, S, j) = -1\}$ and $\mathbf{R}^+(\tau, S) = \mathbf{R}(\tau, S) \setminus \mathbf{R}^-(\tau, S)$. For any $H \in \Sigma^+$ and $s \in \mathbb{Z}_{\geq 0}$, we then let $\mathbf{R}_{f,H}(\tau, S) = \{j \in \mathbf{R}(\tau, S) : \text{root}_f(\tau, S, j) = H\}$, $\mathbf{R}_{f,H}^-(\tau, S) = \mathbf{R}^-(\tau, S) \cap \mathbf{R}_{f,H}(\tau, S)$, $\mathbf{R}_{f,H}^+(\tau, S) = \mathbf{R}^+(\tau, S) \cap \mathbf{R}_{f,H}(\tau, S)$, $\mathbf{R}_{f,s,H}(\tau, S) = \{j \in \mathbf{R}_{f,H}(\tau, S) : \text{head}_f(\tau, S, j) = s\}$, $\mathbf{R}_{f,s,H}^-(\tau, S) = \mathbf{R}^-(\tau, S) \cap \mathbf{R}_{f,s,H}(\tau, S)$, and $\mathbf{R}_{f,s,H}^+(\tau, S) = \mathbf{R}^+(\tau, S) \cap \mathbf{R}_{f,s,H}(\tau, S)$.

The following two lemmas establish the basic properties of periodic positions. First, we show that for any necklace-consistent f , the set of positions $\mathbf{R}_{f,s,H}(\tau, T)$ occupies a contiguous block in the

⁴Note that this definition of root generalizes the original definition [KK19], in which $f(S)$ always returned the lexicographically minimal string that is cyclically equivalent with S . Such function is clearly necklace-consistent.

⁵Although the string $\text{root}_f(\tau, S, j)$ depends on the function f , the value $|\text{root}_f(\tau, S, j)| = \text{per}(S[j..j + 3\tau - 1])$ does not. Thus, it is correct to drop f in the notation for $\text{type}(\tau, S, j)$.

SA of T and describe the structure of such *lexicographical* block. The following lemma was proved in [KK21, Lemma 3.5], but for completeness, we provide its proof in Appendix A.

Lemma 6.18. *Let $S \in \Sigma^k$, $\tau \in \mathbb{Z}_+$, and let f be any necklace-consistent function. If $j \in R_{f,s,H}(\tau, S)$ then for any $j' \in [1..k]$, $\text{LCE}_S(j, j') \geq 3\tau - 1$ holds if and only if $j' \in R_{f,s,H}(\tau, S)$. Moreover, if $j' \in R_{f,s,H}(\tau, S)$, then:*

1. *If $\text{type}(\tau, S, j) = -1$ and $\text{type}(\tau, S, j') = +1$, then $S[j..] \prec S[j'..]$,*
2. *If $\text{type}(\tau, S, j) = \text{type}(\tau, S, j') = -1$ and $e(\tau, S, j) - j < e(\tau, S, j') - j'$, then $S[j..] \prec S[j'..]$,*
3. *If $\text{type}(\tau, S, j) = \text{type}(\tau, S, j') = +1$ and $e(\tau, S, j) - j > e(\tau, S, j') - j'$, then $S[j..] \prec S[j'..]$.*

The key to the efficient computation of $\delta_\ell(j)$ is processing of the elements of $R(\tau, S)$ in blocks (note that here by “blocks” we mean blocks of positions in *text order*, as opposed to blocks in *lexicographical order* (i.e., in the suffix array) considered in the previous lemma). The starting positions of these blocks are defined as

$$R'(\tau, S) := \{j \in R(\tau, S) : j - 1 \notin R(\tau, S)\}. \quad (6.2)$$

We also let $R'^-(\tau, S) = R^-(\tau, S) \cap R'(\tau, S)$, $R'^+(\tau, S) = R^+(\tau, S) \cap R'(\tau, S)$, $R'_{f,H}{}^-(\tau, S) = R'(\tau, S) \cap R_{f,H}^-(\tau, S)$, and $R'_{f,H}{}^+(\tau, S) = R'(\tau, S) \cap R_{f,H}^+(\tau, S)$. For any $H \in \Sigma^+$ we also denote:

$$E_{f,H}^- (\tau, S) := \{(e_f^{\text{full}}(\tau, S, j) - j, e_f^{\text{full}}(\tau, S, j)) : j \in R'_{f,H}{}^-(\tau, S)\}.$$

The set $E_{f,H}^+(\tau, S)$ is defined analogously, but with $R'_{f,H}{}^-(\tau, S)$ replaced by $R'_{f,H}{}^+(\tau, S)$. The next lemma justifies our strategy. As with the previous lemma, this is a standard result characterizing periodic positions of string synchronizing sets and was proved in [KK21, Lemma 3.6] for the original definition of root. For completeness, in Appendix A we provide its minimally modified proof adapting it to the more general version of root used in this paper.

Lemma 6.19. *Let $S \in \Sigma^+$, $\tau \in \mathbb{Z}_+$, and assume that f is a necklace-consistent function. For any position $j \in R(\tau, S) \setminus R'(\tau, S)$ it holds*

- $\text{root}_f(\tau, S, j-1) = \text{root}_f(\tau, S, j)$,
- $e(\tau, S, j-1) = e(\tau, S, j)$, and
- $\text{type}(\tau, S, j-1) = \text{type}(\tau, S, j)$.

Definition 6.20. For any $j \in R(\tau, S)$, denote $I_j(\tau, S) = (b + 1, e + 1, j)$, where $e = e_f^{\text{full}}(\tau, S, j) - j$, $t = e(\tau, s, j) - j - 3\tau + 2$, and $b = e - t$. Let $H \in \Sigma^+$. We define

$$\begin{aligned} \mathcal{I}_{f,H}^- (\tau, S) &:= \{I_j(\tau, S) : j \in R'_{f,H}{}^-(\tau, S)\}, \\ \mathcal{I}_{f,H}^+ (\tau, S) &:= \{I_j(\tau, S) : j \in R'_{f,H}{}^+(\tau, S)\}. \end{aligned}$$

We have now defined all the necessary notation to express the assumption that we will use in the rest of this section.

Assumption 6.21. *For $\tau = \lfloor \frac{\ell}{3} \rfloor$, some necklace-consistent function f , and any $H \in \Sigma^+$, the following queries on text T can be supported in $\mathcal{O}(t)$ time:*

1. *Given any $j \in R(\tau, T)$, return $|\text{root}_f(\tau, T, j)|$, $\text{head}_f(\tau, T, j)$, and $e(\tau, T, j)$.*
2. *Range queries (Problem 4.5) on $\text{Points}_{7\tau}(T, E_{f,H}^- (\tau, T))$ and $\text{Points}_{7\tau}(T, E_{f,H}^+ (\tau, T))$.*

3. Modular constraint queries (Section 5) on $\mathcal{I}_{f,H}^-(\tau, T)$ and $\mathcal{I}_{f,H}^+(\tau, T)$.

The string H for queries 2. and 3. is specified by some $i, p \in [1..n]$ such that $H = T[i..i+p]$.

The above notation was introduced in the general form, because it is needed in the section implementing Assumption 6.21 which supports these queries for *collections of strings*. For the rest of the section, we define $\text{root}(j)$, $\text{head}(j)$, $\text{exp}(j)$, $\text{exp}^{\text{cut}}(j, t)$, $\text{tail}(j)$, $e(j)$, $e^{\text{full}}(j)$, $e^{\text{low}}(j)$, $e^{\text{high}}(j)$, $e^{\text{cut}}(j, t)$, and $\text{type}(j)$ as a shorthand for, respectively, $\text{root}_f(\tau, S, j)$, $\text{head}_f(\tau, S, j)$, $\text{exp}_f(\tau, S, j)$, $\text{exp}_f^{\text{cut}}(\tau, S, j, t)$, $\text{tail}_f(\tau, S, j)$, $e(\tau, S, j)$, $e_f^{\text{full}}(\tau, S, j)$, $e_f^{\text{low}}(\tau, S, j)$, $e_f^{\text{high}}(\tau, S, j)$, $e_f^{\text{cut}}(\tau, S, j, t)$, and $\text{type}(\tau, S, j)$ with $S = T$, $\tau = \lfloor \frac{\ell}{3} \rfloor$, and f being some necklace-consistent function (recall that T is the main text defined globally all through the paper).

We also define \mathbf{R} , \mathbf{R}^- , \mathbf{R}^+ , \mathbf{R}_H , \mathbf{R}_H^- , \mathbf{R}_H^+ , $\mathbf{R}_{s,H}$, $\mathbf{R}_{s,H}^-$, $\mathbf{R}_{s,H}^+$, \mathbf{R}' , \mathbf{R}'^- , \mathbf{R}'^+ , $\mathbf{R}'_{s,H}$, $\mathbf{R}'_{s,H}^-$, $\mathbf{R}'_{s,H}^+$, \mathbf{E}_H^- , \mathbf{E}_H^+ , \mathcal{I}_H^- , \mathcal{I}_H^+ , and \mathbf{Roots} as a shorthand notation for, respectively, the corresponding sets $\mathbf{R}(\tau, S)$, $\mathbf{R}^-(\tau, S)$, $\mathbf{R}^+(\tau, S)$, $\mathbf{R}_{f,H}(\tau, S)$, $\mathbf{R}_{f,H}^-(\tau, S)$, $\mathbf{R}_{f,H}^+(\tau, S)$, $\mathbf{R}_{f,s,H}(\tau, S)$, $\mathbf{R}_{f,s,H}^-(\tau, S)$, $\mathbf{R}_{f,s,H}^+(\tau, S)$, $\mathbf{R}'(\tau, S)$, $\mathbf{R}'^-(\tau, S)$, $\mathbf{R}'^+(\tau, S)$, $\mathbf{R}'_{f,s,H}(\tau, S)$, $\mathbf{R}'_{f,s,H}^+(\tau, S)$, $\mathbf{E}_{f,H}^-(\tau, S)$, $\mathbf{E}_{f,H}^+(\tau, S)$, $\mathcal{I}_{f,H}^-(\tau, S)$, $\mathcal{I}_{f,H}^+(\tau, S)$, and finally $\mathbf{Roots}_f(\tau, S)$ with $S = T$, $\tau = \lfloor \frac{\ell}{3} \rfloor$, and f being some necklace-consistent function.

6.3.2 Decomposition of Pos

Let $j \in \mathbf{R}^-$. In this section we introduce the three sets $\text{Pos}_\ell^{\text{low}}(j)$, $\text{Pos}_\ell^{\text{mid}}(j)$, and $\text{Pos}_\ell^{\text{high}}(j)$ playing a central role in the SA query algorithm (positions $j \in \mathbf{R}^+$ are processed symmetrically; the details are provided in the proof of Proposition 6.53).

The section is organized as follows. First, we define $\text{Pos}_\ell^{\text{low}}(j)$, $\text{Pos}_\ell^{\text{mid}}(j)$, and $\text{Pos}_\ell^{\text{high}}(j)$. We then prove the central combinatorial result (Lemma 6.23) of the SA query, showing how the size $\delta_\ell(j) = |\text{Pos}_\ell(j)|$ relates to the sizes of the other three sets.

Definition 6.22. Assume $H \in \mathbf{Roots}$, $s \in [0..|H|)$, and $j \in \mathbf{R}_{s,H}^-$. Denote $k_1 := \text{exp}^{\text{cut}}(j, \ell) = \min(\text{exp}(j), \lfloor \frac{\ell-s}{|H|} \rfloor)$ and $k_2 := \text{exp}^{\text{cut}}(j, 2\ell) = \min(\text{exp}(j), \lfloor \frac{2\ell-s}{|H|} \rfloor)$. We define

$$\begin{aligned} \text{Pos}_\ell^{\text{low}}(j) &= \{j' \in \mathbf{R}_{s,H}^- : \text{exp}(j') = k_1 \text{ and } (T[j'..n] \succeq T[j..n] \text{ or } \text{LCE}_T(j, j') \geq \ell)\}, \\ \text{Pos}_\ell^{\text{mid}}(j) &= \{j' \in \mathbf{R}_{s,H}^- : \text{exp}(j') \in (k_1..k_2)\}, \text{ and} \\ \text{Pos}_\ell^{\text{high}}(j) &= \{j' \in \mathbf{R}_{s,H}^- : \text{exp}(j') = k_2 \text{ and } (T[j'..n] \succeq T[j..n] \text{ or } \text{LCE}_T(j, j') \geq 2\ell)\}. \end{aligned}$$

We denote $\delta_\ell^{\text{low}}(j) = |\text{Pos}_\ell^{\text{low}}(j)|$, $\delta_\ell^{\text{mid}}(j) = |\text{Pos}_\ell^{\text{mid}}(j)|$, and $\delta_\ell^{\text{high}}(j) = |\text{Pos}_\ell^{\text{high}}(j)|$.

Lemma 6.23. For any $j \in \mathbf{R}^-$, it holds $\delta_\ell(j) = \delta_\ell^{\text{low}}(j) + \delta_\ell^{\text{mid}}(j) - \delta_\ell^{\text{high}}(j)$.

Proof. By definition, it holds $\text{Pos}_\ell^{\text{low}}(j) \cap \text{Pos}_\ell^{\text{mid}}(j) = \emptyset$. On the other hand, by definition, we also have $\text{Pos}_\ell(j) \cap \text{Pos}_\ell^{\text{high}}(j) = \emptyset$. The main strategy of the proof is to show the equality $\text{Pos}_\ell^{\text{low}}(j) \cup \text{Pos}_\ell^{\text{mid}}(j) = \text{Pos}_\ell(j) \cup \text{Pos}_\ell^{\text{high}}(j)$. Since both elements of the equation are disjoint unions, this implies

$$\begin{aligned} \delta_\ell^{\text{low}}(j) + \delta_\ell^{\text{mid}}(j) &= |\text{Pos}_\ell^{\text{low}}(j)| + |\text{Pos}_\ell^{\text{mid}}(j)| = |\text{Pos}_\ell^{\text{low}}(j) \cup \text{Pos}_\ell^{\text{mid}}(j)| \\ &= |\text{Pos}_\ell(j) \cup \text{Pos}_\ell^{\text{high}}(j)| = |\text{Pos}_\ell(j)| + |\text{Pos}_\ell^{\text{high}}(j)| \\ &= \delta_\ell(j) + \delta_\ell^{\text{high}}(j), \end{aligned}$$

which yields the claim. It thus remains to show $\text{Pos}_\ell^{\text{low}}(j) \cup \text{Pos}_\ell^{\text{mid}}(j) = \text{Pos}_\ell(j) \cup \text{Pos}_\ell^{\text{high}}(j)$. Let $H \in \text{Roots}$ and $s \in [0 \dots |H|)$ be such that $j \in \mathcal{R}_{s,H}^-$. Before we move on to proving the main equality, we observe that since for any $j' \in \text{Pos}_\ell(j)$, it holds $\text{LCE}_T(j, j') \geq \ell \geq 3\tau - 1$, by Lemma 6.18, it holds $j' \in \mathcal{R}_{s,H}$ and $\text{type}(j') = -1$. Thus, we can equivalently write $\text{Pos}_\ell(j) = \{j' \in \mathcal{R}_{s,H}^- : T[j' \dots n] \prec T[j \dots n]\}$ and $\text{LCE}_T(j, j') \in [\ell \dots 2\ell]$.

We first show the inclusion $\text{Pos}_\ell^{\text{low}}(j) \cup \text{Pos}_\ell^{\text{mid}}(j) \subseteq \text{Pos}_\ell(j) \cup \text{Pos}_\ell^{\text{high}}(j)$. Let us first take a position $j' \in \text{Pos}_\ell^{\text{low}}(j)$. Observe that it holds $k_1 \leq k_2$. We consider two cases:

- If $k_1 < k_2$, or equivalently, $\min(\exp(j), \lfloor \frac{\ell-s}{|H|} \rfloor) < \min(\exp(j), \lfloor \frac{2\ell-s}{|H|} \rfloor)$ then $\lfloor \frac{\ell-s}{|H|} \rfloor < \exp(j)$ and hence $k_1 = \lfloor \frac{\ell-s}{|H|} \rfloor < \exp(j)$. Therefore, since for j' we have $\text{type}(j') = \text{type}(j) = -1$ and $e(j') - j' = s + \exp(j')|H| + \text{tail}(j') = s + k_1|H| + \text{tail}(j') < s + (k_1 + 1)|H| \leq s + \exp(j)|H| \leq s + \exp(j)|H| + \text{tail}(j) = e(j) - j$, we obtain from Item 2 of Lemma 6.18 that $T[j' \dots n] \prec T[j \dots n]$. Thus, by definition of $\text{Pos}_\ell^{\text{low}}(j)$ we must also have $\text{LCE}_T(j, j') \geq \ell$. On the other hand, by $j, j' \in \mathcal{R}_{s,H}$ and $e(j') - j' < e(j) - j$, we have $T[j \dots j+t] = T[j' \dots j'+t]$ (where $t = e(j') - j'$) and $T[j'+t] \neq T[j'+t - |H|] = T[j+t - |H|] = T[j+t]$. Thus, $\text{LCE}_T(j, j') = e(j') - j' = s + k_1|H| + \text{tail}(j') = s + \lfloor \frac{\ell-s}{|H|} \rfloor |H| + \text{tail}(j') \leq \ell + \text{tail}(j') < \ell + \tau < 2\ell$. We thus proved $T[j' \dots n] \prec T[j \dots n]$ and $\text{LCE}_T(j, j') \in [\ell \dots 2\ell)$, i.e., $j' \in \text{Pos}_\ell(j)$.
- If $k_1 = k_2$, then it suffices to consider two subcases. If $T[j' \dots n] \succeq T[j \dots n]$ or $\text{LCE}_T(j, j') \geq 2\ell$, then we immediately obtain $j' \in \text{Pos}_\ell^{\text{high}}(j)$. The other possibility is that $T[j' \dots n] \prec T[j \dots n]$ and $\text{LCE}_T(j, j') < 2\ell$. Combining this with $\text{LCE}_T(j, j') \geq \ell$ (from the definition of $\text{Pos}_\ell^{\text{low}}(j)$), we obtain $j' \in \text{Pos}_\ell(j)$.

We have thus proved that $\text{Pos}_\ell^{\text{low}}(j) \subseteq \text{Pos}_\ell(j) \cup \text{Pos}_\ell^{\text{high}}(j)$. Next, we show $\text{Pos}_\ell^{\text{mid}}(j) \subseteq \text{Pos}_\ell(j) \cup \text{Pos}_\ell^{\text{high}}(j)$. Consider $j' \in \text{Pos}_\ell^{\text{mid}}(j)$. If $k_1 = k_2$, the claim follows trivially, since $\text{Pos}_\ell^{\text{mid}}(j) = \emptyset$. Let us thus assume $k_1 < k_2$. As noted above, this implies $k_1 = \lfloor \frac{\ell-s}{|H|} \rfloor < \exp(j)$. Then, by $\exp(j') > k_1$, we obtain $e(j') - j' = s + \exp(j')|H| + \text{tail}(j') \geq s + (k_1 + 1)|H| = (s + |H|) + \lfloor \frac{\ell-s}{|H|} \rfloor |H| \geq (s + |H|) + (\ell - s - |H|) = \ell$. Similarly, by $\lfloor \frac{\ell-s}{|H|} \rfloor < \exp(j)$, we have $e(j) - j = s + \exp(j)|H| + \text{tail}(j) \geq s + (\lfloor \frac{\ell-s}{|H|} \rfloor + 1)|H| \geq \ell$. Thus, by definition of run-decomposition, we obtain $\text{LCE}_T(j, j') \geq \min(e(j) - j, e(j') - j') \geq \ell$. Recall now that we have $\exp(j') \in (k_1 \dots k_2]$. Consider now two cases:

- Let $\exp(j') = k_2$. If $T[j' \dots n] \succeq T[j \dots n]$ or $\text{LCE}_T(j, j') \geq 2\ell$, then we immediately obtain $j' \in \text{Pos}_\ell^{\text{high}}(j)$. The remaining case is $T[j' \dots n] \prec T[j \dots n]$ and $\text{LCE}_T(j, j') < 2\ell$. Combining with the above observation $\text{LCE}_T(j, j') \geq \ell$, we obtain $j' \in \text{Pos}_\ell(j)$.
- Let $\exp(j') \in (k_1 \dots k_2)$. We will show that it holds $j' \in \text{Pos}_\ell(j)$. By $\exp(j') < k_2 = \min(\exp(j), \lfloor \frac{2\ell-s}{|H|} \rfloor)$, it follows in particular that $\exp(j') < \lfloor \frac{2\ell-s}{|H|} \rfloor$. Thus, $e(j') - j' = s + \exp(j')|H| + \text{tail}(j') < s + (\exp(j') + 1)|H| \leq s + \lfloor \frac{2\ell-s}{|H|} \rfloor |H| \leq 2\ell$. On the other hand, by $\exp(j') < k_2$, it follows $\exp(j') < \exp(j)$. Thus, $e(j') - j' = s + \exp(j')|H| + \text{tail}(j') < s + (\exp(j') + 1)|H| \leq s + \exp(j)|H| \leq s + \exp(j)|H| + \text{tail}(j) = e(j) - j$. Therefore, for $t = e(j') - j'$, we have $T[j' \dots j'+t] = T[j \dots j+t]$ and $T[j'+t] \neq T[j'+t - |H|] = T[j+t - |H|] = T[j+t]$. Thus, $\text{LCE}_T(j, j') = e(j') - j' < 2\ell$. Combining with the above observation (holding for all $j' \in (k_1 \dots k_2]$) $\text{LCE}_T(j, j') \geq \ell$, we obtain $\text{LCE}_T(j, j') \in [\ell \dots 2\ell)$. Finally, by $e(j') - j' < e(j) - j$ and $\text{type}(j') = \text{type}(j) = -1$, it follows from Item 2 of Lemma 6.18, that $T[j' \dots n] \prec T[j \dots n]$. We have thus proved $j' \in \text{Pos}_\ell(j)$.

Thus, $\text{Pos}_\ell^{\text{mid}}(j) \subseteq \text{Pos}_\ell(j) \cup \text{Pos}_\ell^{\text{high}}(j)$. Combining with the above proof of $\text{Pos}_\ell^{\text{low}}(j) \subseteq \text{Pos}_\ell(j) \cup$

$\text{Pos}_\ell^{\text{high}}(j)$, this establishes the inclusion $\text{Pos}_\ell^{\text{low}}(j) \cup \text{Pos}_\ell^{\text{mid}}(j) \subseteq \text{Pos}_\ell(j) \cup \text{Pos}_\ell^{\text{high}}(j)$.

We now show the opposite inclusion, i.e., $\text{Pos}_\ell(j) \cup \text{Pos}_\ell^{\text{high}}(j) \subseteq \text{Pos}_\ell^{\text{low}}(j) \cup \text{Pos}_\ell^{\text{mid}}(j)$. Let us first assume $j' \in \text{Pos}_\ell(j)$. By $j \in \mathbb{R}_{s,H}^-$ and $T[j'..n] \prec T[j..n]$, we obtain from Item 2 of Lemma 6.18, that $e(j') - j' \leq e(j) - j$. Therefore, $\exp(j') = \lfloor \frac{e(j') - j' - s}{|H|} \rfloor \leq \lfloor \frac{e(j) - j - s}{|H|} \rfloor = \exp(j)$. On the other hand, by definition of the run-decomposition, we have $\text{LCE}_T(j, j') \geq \min(e(j') - j', e(j) - j)$. Thus, by $e(j') - j' < e(j) - j$ and $\text{LCE}_T(j, j') < 2\ell$, we obtain $e(j') - j' = \min(e(j') - j', e(j) - j) \leq \text{LCE}_T(j, j') < 2\ell$, and consequently, $\exp(j') = \lfloor \frac{e(j') - j' - s}{|H|} \rfloor \leq \lfloor \frac{2\ell - s}{|H|} \rfloor$. Combining the two upper bounds on $\exp(j')$, we thus obtain $\exp(j') \leq \min(\exp(j), \lfloor \frac{2\ell - s}{|H|} \rfloor) = k_2$. Next, we prove $\exp(j') \geq k_1$. For this, we consider two cases:

- Let us first assume $e(j) - j < \ell$. By definition of k_1 , we then have $k_1 = \min(\exp(j), \lfloor \frac{\ell - s}{|H|} \rfloor) = \min(\lfloor \frac{e(j) - j - s}{|H|} \rfloor, \lfloor \frac{\ell - s}{|H|} \rfloor) = \lfloor \frac{e(j) - j - s}{|H|} \rfloor = \exp(j)$. Observe now that by $\text{LCE}_T(j, j') \geq \ell$, we have $T[j..j + \ell] = T[j'..j' + \ell]$. Recall that by [KK19, Fact 3.2], $T[j..e(j)]$ is the longest prefix of $T[j..n]$ having period $|\text{root}(j)|$. Therefore, since $\text{root}(j') = \text{root}(j)$ and $T[j..e(j)]$ is a proper prefix of $T[j..j + \ell]$, applying this equivalent definition to $T[j'..n]$, we obtain $e(j') - j' = e(j) - j$. Thus, $\exp(j') = \lfloor \frac{e(j') - j' - s}{|H|} \rfloor = \lfloor \frac{e(j) - j - s}{|H|} \rfloor = \exp(j) = k_1$.
- Let us now assume $e(j) - j \geq \ell$. Then, by the above alternative definition of $T[j..e(j)]$, $T[j..j + \ell]$ has a period $|\text{root}(j)|$. Thus, by $\text{root}(j') = \text{root}(j)$ and $T[j..j + \ell] = T[j'..j' + \ell]$, the string $T[j'..j' + \ell]$ has period $|\text{root}(j')|$, and consequently, $e(j') - j' \geq \ell$. Therefore, $\exp(j') = \lfloor \frac{e(j') - j' - s}{|H|} \rfloor \geq \lfloor \frac{\ell - s}{|H|} \rfloor = \min(\lfloor \frac{\ell - s}{|H|} \rfloor, \lfloor \frac{e(j) - j - s}{|H|} \rfloor) = k_1$.

We have thus shown that in both cases, it holds $\exp(j') \geq k_1$. Combining with the earlier upper bound on $\exp(j')$, we therefore obtain $\exp(j') \in [k_1..k_2]$. To see that this immediately implies the inclusion $\text{Pos}_\ell(j) \subseteq \text{Pos}_\ell^{\text{low}}(j) \cup \text{Pos}_\ell^{\text{mid}}(j)$, it suffices to consider two cases. If $\exp(j') > k_1$, then $j' \in \text{Pos}_\ell^{\text{mid}}(j)$ holds by definition of $\text{Pos}_\ell^{\text{mid}}(j)$. On the other hand, if $\exp(j') = k_1$, then $j' \in \text{Pos}_\ell^{\text{low}}(j)$ follows from $\text{LCE}_T(j, j') \in [\ell..2\ell]$. It remains to show $\text{Pos}_\ell^{\text{high}}(j) \subseteq \text{Pos}_\ell^{\text{low}}(j) \cup \text{Pos}_\ell^{\text{mid}}(j)$. Let $j' \in \text{Pos}_\ell^{\text{high}}(j)$. We again consider two cases. If $k_1 < k_2$, then we immediately have $j' \in \text{Pos}_\ell^{\text{mid}}(j)$ since $\exp(j') \in (k_1..k_2]$. On the other hand, if $k_1 = k_2$, then either $T[j'..n] \succeq T[j..n]$ or $\text{LCE}_T(j, j') \geq 2\ell \geq \ell$. In either case, $j' \in \text{Pos}_\ell^{\text{low}}(j)$. This concludes the proof of the inclusion $\text{Pos}_\ell(j) \cup \text{Pos}_\ell^{\text{high}}(j) \subseteq \text{Pos}_\ell^{\text{low}}(j) \cup \text{Pos}_\ell^{\text{mid}}(j)$. \square

6.3.3 Computing the Size of Occ

Let $j \in \mathbb{R}$ and $d \geq \ell$. We define $\text{Occ}_d^-(j) := \text{Occ}_d(j) \cap \mathbb{R}^-$ and $\text{Occ}_d^+(j) := \text{Occ}_d(j) \cap \mathbb{R}^+$. In this section, we show how under Assumption 6.21, given any position $j \in \mathbb{R}$, to efficiently compute the cardinalities of the four sets Occ_ℓ^- , $\text{Occ}_\ell^+(j)$, $\text{Occ}_{2\ell}^-(j)$, and $\text{Occ}_{2\ell}^+(j)$. In the rest of this section, we focus on the computation of $|\text{Occ}_\ell^-(j)|$ and $|\text{Occ}_{2\ell}^-(j)|$. The cardinalities of $\text{Occ}_\ell^+(j)$ and $\text{Occ}_{2\ell}^+(j)$ are computed analogously (see the proof of Proposition 6.53).

The section is organized into four parts. First, we prove the combinatorial results (Lemma 6.24 and Corollary 6.25) characterizing the set $\text{Occ}_d^-(j)$, $d \geq \ell$, as a disjoint union of two sets $\text{Occ}_d^{\text{eq}-}(j)$ and $\text{Occ}_d^{\text{gt}-}(j)$ (in particular, each of the sets $\text{Occ}_\ell^-(j)$ and $\text{Occ}_{2\ell}^-(j)$ admits such decomposition). In the following two parts, we present a query algorithms (Proposition 6.28 and Proposition 6.31) to compute the cardinality of $\text{Occ}_d^{\text{eq}-}(j)$ for any $d \in [\ell..2\ell]$ (in particular, we can compute $|\text{Occ}_\ell^{\text{eq}-}(j)|$ and $|\text{Occ}_{2\ell}^{\text{eq}-}(j)|$) and the cardinality of $\text{Occ}_d^{\text{gt}-}(j)$ for any $d \geq \ell$ (in particular, we can compute $|\text{Occ}_\ell^{\text{gt}-}(j)|$ and $|\text{Occ}_{2\ell}^{\text{gt}-}(j)|$). In Proposition 6.32, we put them together to obtain the final query

algorithm.

The Main Idea Let $d \geq \ell$. Assume $H \in \text{Roots}$, $s \in [0..|H|)$, and $j \in \mathbb{R}_{s,H}$. Denote $k = \exp^{\text{cut}}(j, d) = \min(\exp(j), \lfloor \frac{d-s}{|H|} \rfloor)$. We define

$$\begin{aligned} \text{Occ}_d^{\text{eq}^-}(j) &= \{j' \in \mathbb{R}_{s,H} \cap \text{Occ}_d^-(j) : \exp(j') = k\}, \\ \text{Occ}_d^{\text{gt}^-}(j) &= \{j' \in \mathbb{R}_{s,H} \cap \text{Occ}_d^-(j) : \exp(j') > k\}. \end{aligned}$$

Lemma 6.24. *Assume $d \geq \ell$. If $j \in \mathbb{R}_{s,H}$ then for any $j' \in \text{Occ}_d(j)$, it holds:*

1. It holds $j' \in \mathbb{R}_{s,H}^-$,
2. $e^{\text{cut}}(j', d) - j' = e^{\text{cut}}(j, d) - j$, and
3. $T^\infty[e^{\text{cut}}(j', d) \dots j' + d] = T^\infty[e^{\text{cut}}(j, d) \dots j + d]$.

Proof. 1. Consider two cases. If $j' = j$, then immediately $j' \in \mathbb{R}$, $\text{head}(j') = s$ and $\text{root}(j') = H$. Otherwise, by definition of $\text{Occ}_d(j)$, it holds $T^\infty[j \dots j + d] = T^\infty[j' \dots j' + d]$. By $j \neq j'$ and the uniqueness of $T[n] = \$$, this is equivalent to $\text{LCE}_T(j, j') \geq d$. Since by definition of τ , it holds $3\tau \leq \ell \leq d$, we obtain $\text{LCE}_T(j, j') \geq 3\tau - 1$. Thus, by Lemma 6.18, $j' \in \mathbb{R}_{s,H}$.

2. We first prove that $\exp^{\text{cut}}(j', d) = \exp^{\text{cut}}(j, d)$. Recall that by [KK19, Fact 3.2], for $i \in \mathbb{R}$, the substring $T[i \dots e(i)]$ is the longest prefix of $T[i \dots n]$ that has a period $|\text{root}(i)|$. By $T^\infty[j \dots j + d] = T^\infty[j' \dots j' + d]$, this immediately implies $\min(e(j') - j', d) = \min(e(j) - j, d)$. Therefore, we obtain

$$\begin{aligned} \exp^{\text{cut}}(j', d) &= \min(\exp(j'), \lfloor \frac{d-s}{|H|} \rfloor) = \min(\lfloor \frac{e(j')-j'-s}{|H|} \rfloor, \lfloor \frac{d-s}{|H|} \rfloor) \\ &= \lfloor \frac{\min(e(j')-j', d)-s}{|H|} \rfloor = \lfloor \frac{\min(e(j)-j, d)-s}{|H|} \rfloor \\ &= \min(\lfloor \frac{e(j)-j-s}{|H|} \rfloor, \lfloor \frac{d-s}{|H|} \rfloor) = \min(\exp(j), \lfloor \frac{d-s}{|H|} \rfloor) \\ &= \exp^{\text{cut}}(j, d) \end{aligned}$$

Thus, $e^{\text{cut}}(j', d) - j' = \text{head}(j') + \exp^{\text{cut}}(j', d)|H| = \text{head}(j) + \exp^{\text{cut}}(j, d)|H| = e^{\text{cut}}(j, d) - j$.

3. By $j' \in \text{Occ}_d(j)$, $T^\infty[j' \dots j' + d] = T^\infty[j \dots j + d]$. By Item 2 this yields the claim. \square

Corollary 6.25. *For any $d \geq \ell$ and any $j \in \mathbb{R}$, the set $\text{Occ}_d^-(j)$ is a disjoint union of $\text{Occ}_d^{\text{eq}^-}(j)$ and $\text{Occ}_d^{\text{gt}^-}(j)$.*

Proof. By definition, it holds $\text{Occ}_d^{\text{eq}^-}(j) \cap \text{Occ}_d^{\text{gt}^-}(j) = \emptyset$ and $\text{Occ}_d^{\text{eq}^-}(j) \cup \text{Occ}_d^{\text{gt}^-}(j) \subseteq \text{Occ}_d^-(j)$. It remains to show $\text{Occ}_d^-(j) \subseteq \text{Occ}_d^{\text{eq}^-}(j) \cup \text{Occ}_d^{\text{gt}^-}(j)$. Let $j' \in \text{Occ}_d^-(j)$. By Lemma 6.24, this implies $j' \in \mathbb{R}_{s,H}$ and $\exp^{\text{cut}}(j', d) = \exp^{\text{cut}}(j, d)$. Thus, by $\exp(j') \geq \min(\exp(j), \lfloor \frac{d-s}{|H|} \rfloor) = \exp^{\text{cut}}(j, d)$, we obtain that $j' \in \text{Occ}_d^{\text{eq}^-}(j)$ or $j' \in \text{Occ}_d^{\text{gt}^-}(j)$. \square

Remark 6.26. Note that the differentiation between $j \in \mathbb{R}$ satisfying $\text{type}(j) = -1$ and $\text{type}(j) = +1$ used (without the loss of generality) in Section 6.3.2 is different from the symmetry used in this section. Here we partition the output set $\text{Occ}_{2\ell}^{\text{eq}}(j)$ (resp. $\text{Occ}_{2\ell}^{\text{gt}}(j)$) into two subsets $\text{Occ}_{2\ell}^{\text{eq}^-}(j)$ and $\text{Occ}_{2\ell}^{\text{eq}^+}(j)$ (resp. $\text{Occ}_{2\ell}^{\text{gt}^-}(j)$ and $\text{Occ}_{2\ell}^{\text{gt}^+}(j)$), but the computation is always performed regardless of $\text{type}(j)$, leading to two queries for each $j \in \mathbb{R}$. In Section 6.3.2, on the other hand, the computation is performed separately for $j \in \mathbb{R}^-$ and $j \in \mathbb{R}^+$, without the need to partition $\text{Pos}_\ell(j)$ within each case, leading to a single query but only on the appropriate structure depending on $\text{type}(j)$.

This subtle difference follows from the fact, then when computing $\delta_\ell(j) = |\text{Pos}_\ell(j)|$ by Lemma 6.18 we have $\text{Pos}_\ell(j) \subseteq \mathbb{R}^-$ for any $j \in \mathbb{R}^-$ (and $\text{Pos}_\ell(j) \subseteq \mathbb{R}^+$ for $j \in \mathbb{R}^+$). However, when computing $|\text{Occ}_{2\ell}(j)|$ for $j \in \mathbb{R}^-$, it is possible that $\text{Occ}_{2\ell}(j) \cap \mathbb{R}^- \neq \emptyset$ and $\text{Occ}_{2\ell}(j) \cap \mathbb{R}^+ \neq \emptyset$ hold. This is the reason for why the seemingly related computation of $|\text{Occ}_{2\ell}(j)|$ and $|\text{Pos}_\ell(j)|$ is (unlike for the nonperiodic positions; see Section 6.2.2) described separately.

Computing $|\text{Occ}_\ell^{\text{eq}^-}(j)|$ and $|\text{Occ}_{2\ell}^{\text{eq}^-}(j)|$ We now describe the query algorithm that, given any position $j \in \mathbb{R}$ and any $d \in [\ell..2\ell]$, returns the cardinality of the sets $\text{Occ}_d^{\text{eq}^-}(j)$ (in particular, it can return the cardinality of $\text{Occ}_\ell^{\text{eq}^-}(j)$ and $\text{Occ}_{2\ell}^{\text{eq}^-}(j)$). We start with a combinatorial result (Lemma 6.27) that shows how to count the elements of $\text{Occ}_d^{\text{eq}^-}(j)$ (where $d \in [\ell..2\ell]$) that belong to any right-maximal contiguous block of positions in \mathbb{R}^- . The query algorithm to compute the size of $\text{Occ}_d^{\text{eq}^-}(j)$ is presented next (Proposition 6.28).

Lemma 6.27. *Let $d \in [\ell..2\ell]$ and $j \in \mathbb{R}_H$. Assume $i \in \mathbb{R}_H^-$ and denote $t = e(i) - i - 3\tau + 2$. Then, $|\text{Occ}_d^{\text{eq}^-}(j) \cap [i..i+t]| \leq 1$. Moreover, $|\text{Occ}_d^{\text{eq}^-}(j) \cap [i..i+t]| = 1$ holds if and only if $e^{\text{full}}(i) - i \geq e^{\text{cut}}(j, d) - j$ and $T^\infty[e^{\text{cut}}(j, d)..j+d]$ is a prefix of $T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i) + 7\tau]$.*

Proof. Denote $k_d = \exp^{\text{cut}}(j, d)$. Recall that $e(i) = \min\{i' \in [i..n] : i' \notin \mathbb{R}\} + 3\tau - 2$. Thus, $t > 0$ and $[i..e(i) - 3\tau + 2] = [i..i+t] \subseteq \mathbb{R}$. For any $\delta \in [0..t]$, by Lemma 6.19, we have $\text{root}(i + \delta) = \text{root}(i)$ and $\text{type}(i + \delta) = \text{type}(i)$. Thus, $[i..i+t] \subseteq \mathbb{R}_H^-$. Moreover, by Lemma 6.19, $e(i + \delta) = e(i)$. Therefore, by the uniqueness of run-decomposition, it holds $\text{tail}(i + \delta) = \text{tail}(i)$. Together, these facts imply $e^{\text{full}}(i + \delta) = e^{\text{full}}(i)$, and consequently, that $e^{\text{full}}(i + \delta) - (i + \delta) = e^{\text{full}}(i) - i - \delta$. Let $s = \text{head}(j)$. Recall that for any $j' \in \text{Occ}_d^{\text{eq}^-}(j)$, we have $e^{\text{full}}(j') - j' = s + \exp(j')|H| = s + k_d(j)|H| = e^{\text{cut}}(j, d) - j$. Thus, $i + \delta \in \text{Occ}_d^{\text{eq}^-}(j)$ implies $e^{\text{full}}(i + \delta) - (i + \delta) = e^{\text{full}}(i) - i - \delta = e^{\text{cut}}(j, d) - j$, or equivalently, $\delta = (e^{\text{full}}(i) - i) - (e^{\text{cut}}(j, d) - j)$, and thus $|\text{Occ}_d^{\text{eq}^-}(j) \cap [i..i+t]| \leq 1$.

We now prove the equivalence. Let us first assume $|\text{Occ}_d^{\text{eq}^-}(j) \cap [i..i+t]| = 1$, i.e., that there exists $\delta \in [0..t]$ such that $i + \delta \in \text{Occ}_d^{\text{eq}^-}(j)$. Then, as observed above, $e^{\text{cut}}(j, d) - j = e^{\text{full}}(i) - i - \delta \leq e^{\text{full}}(i) - i$. This proves the first condition. To show the second one, let $s = \text{head}(j)$. By definition of $\text{Occ}_d^{\text{eq}^-}(j)$, it holds $i + \delta \in \mathbb{R}_{s,H}^-$, $\exp(i + \delta) = k$, and $T^\infty[i + \delta..i + \delta + d] = T^\infty[j..j + d]$. Therefore, by $s + k_d|H| \leq d$, we obtain $T^\infty[i + \delta..e^{\text{full}}(i + \delta)] = T^\infty[i + \delta..e^{\text{full}}(i)] = T[j..e^{\text{cut}}(j, d)] = H'H^k$, where H' is a length- s suffix of H . Thus, $T^\infty[e^{\text{full}}(i)..i + \delta + d] = T^\infty[e^{\text{cut}}(j, d)..j + d]$. Therefore, by $i + \delta + d \leq e^{\text{full}}(i) + 7\tau$, $T^\infty[e^{\text{cut}}(j, d)..j + d]$ is a prefix of $T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i) + 7\tau]$.

To prove the opposite implication, assume $e^{\text{full}}(i) - i \geq e^{\text{cut}}(j, d) - j$ and that $T^\infty[e^{\text{cut}}(j, d)..j + d]$ is a prefix of $T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i) + 7\tau]$. Let $\delta = (e^{\text{full}}(i) - i) - (e^{\text{cut}}(j, d) - j)$. We will show that $\delta \in [0..t]$ and $i + \delta \in \text{Occ}_d^{\text{eq}^-}(j)$. The inequality $\delta \geq 0$ follows from the definition of δ and our assumptions. To show $\delta < t$, we consider two cases:

- Let $e(j) - j < d$. We start by noting $k_d = \min(\exp(j), \lfloor \frac{d-s}{|H|} \rfloor) = \min(\lfloor \frac{e(j)-j-s}{|H|} \rfloor, \lfloor \frac{d-s}{|H|} \rfloor) = \lfloor \frac{e(j)-j-s}{|H|} \rfloor = \exp(j)$. Thus, $e^{\text{cut}}(j, d) = j + s + k_d|H| = j + s + \exp(j)|H| = e^{\text{full}}(j)$. We next show that it holds $e(i) - e^{\text{full}}(i) \geq e(j) - e^{\text{full}}(j)$. Let $q = e(i) - e^{\text{full}}(i)$ and suppose $q < e(j) - e^{\text{full}}(j)$. Then, by definition of run-decomposition, it holds $\text{LCE}_T(e^{\text{full}}(i), e^{\text{full}}(j)) \geq \min(e(i) - e^{\text{full}}(i), e(j) - e^{\text{full}}(j)) = q$. We claim that it holds $T[e^{\text{full}}(i) + q] \prec T[e^{\text{full}}(j) + q]$. To show this, we first note that by $T[n] = \$$ being unique in T , we have $e^{\text{full}}(i) + q = e(i) \leq n$. We then consider two subcases:

1. If $e^{\text{full}}(i) + q = n$, then by $e^{\text{full}}(j) + q < e(j) \leq n$, we have $T[e^{\text{full}}(i) + q] = \$ \prec T[e^{\text{full}}(j) + q]$.

2. On the other hand, if $e^{\text{full}}(i) + q < n$, then by $i \in \mathbb{R}^-$, we again have $T[e(i)] = T[e^{\text{full}}(i) + q] \prec T[e^{\text{full}}(i) + q - |H|] = T[e^{\text{full}}(j) + q - |H|] = T[e^{\text{full}}(j) + q]$.

We have thus obtained $T[e^{\text{full}}(i) \dots e^{\text{full}}(i) + q] \prec T[e^{\text{full}}(j) \dots e^{\text{full}}(j) + q]$. By $q < e(j) - e^{\text{full}}(j) < j + d - e^{\text{full}}(j) = j + d - e^{\text{cut}}(j, d) \leq d \leq 2\ell \leq 7\tau$, this contradicts $T^\infty[e^{\text{cut}}(j, d) \dots j + d]$ being a prefix of $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau]$. Thus, it holds $e(i) - e^{\text{full}}(i) \geq e(j) - e^{\text{full}}(j)$. Applying the definition of δ and using the above inequality, we obtain

$$\begin{aligned} e(i) - (i + \delta) &= (e^{\text{full}}(i) - (i + \delta)) + (e(i) - e^{\text{full}}(i)) = (e^{\text{cut}}(j, d) - j) + (e(i) - e^{\text{full}}(i)) \\ &= (e^{\text{full}}(j) - j) + (e(i) - e^{\text{full}}(i)) \geq (e^{\text{full}}(j) - j) + (e(j) - e^{\text{full}}(j)) \\ &= e(j) - j \geq 3\tau - 1. \end{aligned}$$

- Let $e(j) - j \geq d$. We first show that in this case it holds $e(i) - e^{\text{full}}(i) \geq j + d - e^{\text{cut}}(j, d)$. Let $q = e(i) - e^{\text{full}}(i)$ and suppose $q < j + d - e^{\text{cut}}(j, d)$. Then, by definition of run-decomposition, it holds $\text{LCE}_T(e^{\text{full}}(i), e^{\text{cut}}(j, d)) \geq \min(e(i) - e^{\text{full}}(i), e(j) - e^{\text{cut}}(j, d)) = q$. We claim that $T[e^{\text{full}}(i) + q] \prec T[e^{\text{cut}}(j, d) + q]$. To show this, we first note that by $T[n] = \$$, we have $e^{\text{full}}(i) + q = e(i) \leq n$. Thus:
 1. If $e^{\text{full}}(i) + q = n$, then by $e^{\text{cut}}(j, d) + q < d + j \leq e(j) \leq n$, we have $T[e^{\text{full}}(i) + q] = \$ \prec T[e^{\text{cut}}(j, d) + q]$.
 2. On the other hand, if $e^{\text{full}}(i) + q < n$, then by $i \in \mathbb{R}^-$, we again have $T[e(i)] = T[e^{\text{full}}(i) + q] \prec T[e^{\text{full}}(i) + q - |H|] = T[e^{\text{cut}}(j, d) + q - |H|] = T[e^{\text{cut}}(j, d) + q]$.

We thus obtained $T[e^{\text{full}}(i) \dots e^{\text{full}}(i) + q] \prec T[e^{\text{cut}}(j, d) \dots e^{\text{cut}}(j, d) + q]$. By $q < j + d - e^{\text{cut}}(j, d) \leq d \leq 7\tau$, this contradicts the string $T^\infty[e^{\text{cut}}(j, d) \dots j + d]$ being a prefix of $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau]$. Thus, $e(i) - e^{\text{full}}(i) \geq j + d - e^{\text{cut}}(j, d)$. Applying the definition of δ and using the above inequality, we obtain

$$\begin{aligned} e(i) - (i + \delta) &= (e^{\text{full}}(i) - (i + \delta)) + (e(i) - e^{\text{full}}(i)) = (e^{\text{cut}}(j, d) - j) + (e(i) - e^{\text{full}}(i)) \\ &\geq (e^{\text{cut}}(j, d) - j) + (j + d - e^{\text{cut}}(j, d)) = d \geq \ell \geq 3\tau - 1. \end{aligned}$$

Thus, in both cases, we have shown $e(i) - (i + \delta) \geq 3\tau - 1$, or equivalently, $\delta \leq e(i) - i - 3\tau + 1 < t$.

It remains to show $i + \delta \in \text{Occ}_d^{\text{eq}^-}(j)$. For this, we first note $e^{\text{full}}(i + \delta) - (i + \delta) = e^{\text{full}}(i) - (i + \delta) = e^{\text{cut}}(j, d) - j$. Combining this with $i + \delta, j \in \mathbb{R}_H$ gives $T[i + \delta \dots e^{\text{full}}(i + \delta)] = T[j \dots e^{\text{cut}}(j, d)]$. This implies $\text{head}(i + \delta) = \text{head}(j)$ and $\text{exp}(i + \delta) = \lfloor (e^{\text{cut}}(j, d) - j) / |H| \rfloor = k_d$. Moreover, by Lemma 6.19, $\text{type}(i + \delta) = \text{type}(i) = -1$. Thus, $i + \delta \in \mathbb{R}_{s,H}^-$. It remains to show $T^\infty[i + \delta \dots i + \delta + d] = T^\infty[j \dots j + d]$. For this, it suffices to combine $T^\infty[i + \delta \dots e^{\text{full}}(i)] = T^\infty[j \dots e^{\text{cut}}(j, d)]$ (shown above) and $T^\infty[e^{\text{full}}(i) \dots i + \delta + d] = T^\infty[e^{\text{cut}}(j, d) \dots j + d]$ (following from $T^\infty[e^{\text{cut}}(j, d) \dots j + d]$ being a prefix of $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau]$ and $i + \delta + d = e^{\text{full}}(i) - (e^{\text{cut}}(j, d) - j) + d \leq e^{\text{full}}(i) + d \leq e^{\text{full}}(i) + 7\tau$). \square

Proposition 6.28. *Under Assumption 6.21, given any position $j \in \mathbb{R}$ and an integer $d \in [\ell \dots 2\ell]$, we can compute $|\text{Occ}_d^{\text{eq}^-}(j)|$ in $\mathcal{O}(t)$ time.*

Proof. The main idea of the query algorithm is to group all positions $i \in \mathbb{R}'^-$ by $\text{root}(i)$ and then sort all positions within each group by $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau]$. Then, by Lemma 6.27, in order to compute $|\text{Occ}_d^{\text{eq}^-}(j)|$ given $j \in \mathbb{R}_H$, it suffices to count the all positions $i \in \mathbb{R}'_H^-$ that satisfy $e^{\text{full}}(i) - i \geq e^{\text{cut}}(j, d) - j$ and for which $T^\infty[e^{\text{cut}}(j, d) \dots j + d]$ is a prefix of $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau]$. The latter task is equivalent to the general range counting queries (Section 4) on a set of points

$\mathcal{P}_H \subseteq \mathcal{X} \times \mathcal{Y}$ (with $\mathcal{X} = \mathbb{Z}_{\geq 0}$ and $\mathcal{Y} = \Sigma^*$) containing the value $e^{\text{full}}(i) - i$ on the \mathcal{X} -coordinate and the string $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau]$ on the \mathcal{Y} -coordinate for every $i \in \mathbb{R}'_H$. Note that we need to provide efficient range counting queries on each collection \mathcal{P}_H separately for every $H \in \text{Roots}$. Since $|H|$ could be large, H cannot be given explicitly during the query. Thus, we specify H using its length and the position of some occurrence in T .

We use the following definitions. Let $q = 7\tau$ and $c = \max \Sigma$. Then, for any $H \in \text{Roots}$, we let $\mathcal{P}_H = \text{Points}_q(T, E_H^-)$ (see Section 6.3.1 for the definition of E_H^- and Definition 4.4 for the definition of \mathcal{P}_H). Note that by $\ell < n$ and $\tau = \lfloor \frac{\ell}{3} \rfloor$, the value $q = 7\tau \leq 2\ell + \tau < 3n$ satisfies the requirement in Problem 4.5.

Given any position $j \in \mathbb{R}$, we compute $|\text{Occ}_d^{\text{eq}^-}(j)|$ as follows. First, using Assumption 6.21 we compute values $s = \text{head}(j)$, $p = |\text{root}(j)|$, and $e(j)$ in $\mathcal{O}(t)$ time. Note, that then $\text{root}(j) = T[j + s \dots j + s + p]$, i.e., we have a starting position of an occurrence of $H := \text{root}(j)$ in T . We then calculate $k = \exp(j) = \lfloor \frac{e(j)-s}{p} \rfloor$ in $\mathcal{O}(1)$ time. Using those values, we further calculate $k' = \min(k, \lfloor \frac{d-s}{p} \rfloor)$ and $e^{\text{cut}}(j, d) = s + k'p$. By Lemma 6.27 (this is the place we use the assumption $d \in [\ell \dots 2\ell]$) and the definition of \mathcal{P}_H , we now have

$$|\text{Occ}_d^{\text{eq}^-}(j)| = \text{r-count}_{\mathcal{P}_H}(e^{\text{cut}}(j, d) - j, n, Yc^\infty) - \text{r-count}_{\mathcal{P}_H}(e^{\text{cut}}(j, d) - j, n, Y),$$

where $Y = T^\infty[e^{\text{cut}}(j, d) \dots j + d]$, which by Assumption 6.21 we can compute in $\mathcal{O}(t)$ time using the query defined by Item 1 of Problem 4.5 with the arguments $(i, x, q_r) = (e^{\text{cut}}(j, d), e^{\text{cut}}(j, d) - j, j + d - e^{\text{cut}}(j, d))$. To check that all arguments satisfy the requirements of Problem 4.5, recall that $\ell < n$. Thus, $q_r \leq d \leq 2\ell < 2n$. On the other hand, $e^{\text{cut}}(j, d) - j \leq e(j) - j < n$ hold by $T[n] = \$$. In total, the query takes $\mathcal{O}(t)$ time. \square

Computing $|\text{Occ}_\ell^{\text{gt}^-}(j)|$ and $|\text{Occ}_{2\ell}^{\text{gt}^-}(j)|$ We now describe a query algorithm, given any position $j \in \mathbb{R}$ and any $d \geq \ell$, returns the cardinality of the sets $\text{Occ}_d^{\text{gt}^-}(j)$ (in particular, it can return the cardinality of $\text{Occ}_\ell^{\text{gt}^-}(j)$ and $\text{Occ}_{2\ell}^{\text{gt}^-}(j)$). We start with a combinatorial result characterizing $\text{Occ}_d^{\text{gt}^-}(j)$ (where $d \geq \ell$) in terms of $e(j) - j$ (Lemma 6.29). We then present a combinatorial result (Lemma 6.30) that relates the number of positions j' in \mathbb{R}^- having the value $\exp(j')$ in a given interval to a modular contract queries. We conclude with the algorithm to compute $|\text{Occ}_d^{\text{gt}^-}(j)|$ (Proposition 6.31).

Lemma 6.29. *Assume $d \geq \ell$ and $j \in \mathbb{R}_{s,H}$. If $e(j) - j < d$, then it holds $\text{Occ}_d^{\text{gt}^-}(j) = \emptyset$. Otherwise, it holds $\text{Occ}_d^{\text{gt}^-}(j) = \{j' \in \mathbb{R}_{s,H}^- : \exp(j') > \lfloor \frac{d-s}{|H|} \rfloor\}$.*

Proof. Assume first $e(j) - j < d$. Then $k = \min(\exp(j), \lfloor \frac{d-s}{|H|} \rfloor) = \min(\lfloor \frac{e(j)-j-s}{|H|} \rfloor, \lfloor \frac{d-s}{|H|} \rfloor) = \lfloor \frac{e(j)-j-s}{|H|} \rfloor = \exp(j)$. Suppose that $\text{Occ}_d^{\text{gt}^-}(j) \neq \emptyset$ and let $j' \in \text{Occ}_d^{\text{gt}^-}(j)$. Then, we have $e(j) - j = s + \exp(j)|H| + \text{tail}(j) < s + (\exp(j) + 1)|H| = s + (k + 1)|H| \leq s + \exp(j')|H| \leq e(j') - j'$. This implies $j \neq j'$. Moreover, by definition of run-decomposition, if $j, j' \in \mathbb{R}_{s,H}$ and $e(j) - j \neq e(j') - j'$, then $\text{LCE}_T(j, j') = \min(e(j) - j, e(j') - j')$. Therefore, $\text{LCE}_T(j, j') = e(j) - j < d$. By $j \neq j'$ and the uniqueness of $T[n] = \$$, this implies $T^\infty[j' \dots j' + d] \neq T[j \dots j + d]$, and consequently, $j' \notin \text{Occ}_d(j)$, a contradiction. Thus, we must have $\text{Occ}_d^{\text{gt}^-}(j) = \emptyset$.

Assume now $e(j) - j \geq d$. Then, $k = \min(\exp(j), \lfloor \frac{d-s}{|H|} \rfloor) = \min(\lfloor \frac{e(j)-j-s}{|H|} \rfloor, \lfloor \frac{d-s}{|H|} \rfloor) = \lfloor \frac{d-s}{|H|} \rfloor$ and hence

$$\text{Occ}_d^{\text{gt}^-}(j) = \text{Occ}_d^{\text{gt}} \cap \mathbb{R}^-$$

$$\begin{aligned}
&= \{j' \in \mathbb{R}_{s,H}^- \cap \text{Occ}_d(j) : \exp(j') > k\} \\
&= \{j' \in \mathbb{R}_{s,H}^- \cap \text{Occ}_d(j) : \exp(j') > \lfloor \frac{d-s}{|H|} \rfloor\} \\
&\subseteq \{j' \in \mathbb{R}_{s,H}^- : \exp(j') > \lfloor \frac{d-s}{|H|} \rfloor\},
\end{aligned}$$

i.e., we obtain the first inclusion. To show the opposite inclusion let $j' \in \mathbb{R}_{s,H}^-$ be such that $\exp(j') > \lfloor \frac{d-s}{|H|} \rfloor$. By $e(j) - j \geq d$, we can write $T[j \dots j + d] = H'H^kH''$, where $|H'| = s$, and H' (resp. H'') is a proper prefix (resp. suffix) of H . Thus, $T[j \dots j + d]$ is a prefix of $H'H^{k+1}$. On the other hand, the string $H'H^{k+1}$ is, by $\exp(j') > k$ and $j' \in \mathbb{R}_{s,H}$, a prefix of $T[j' \dots n]$. Thus, $\text{LCE}_T(j', j) \geq d$, and consequently, $j' \in \text{Occ}_d(j)$. By $j' \in \mathbb{R}_{s,H}^-$ and $\exp(j') > \lfloor \frac{d-s}{|H|} \rfloor = k$ we thus have $j' \in \text{Occ}_d^{\text{gt}^-}(j)$. \square

Lemma 6.30. *Let $H \in \text{Roots}$ and $s \in [0 \dots |H|)$. For any $k \in \mathbb{Z}_+$, let us define $Q_k^- := \{j \in \mathbb{R}_{s,H}^- : \exp(j) \leq k\}$. Let also $I_i = (a_i, b_i, i)$ (Definition 6.20). Then:*

1. *For $i \in \mathbb{R}_H^-$, it holds $|Q_k^- \cap [i \dots e(i) - 3\tau + 2]| = |\{j \in [a_i \dots b_i) : j \bmod |H| = s \text{ and } \lfloor \frac{j}{|H|} \rfloor \leq k\}|$.*
2. *It holds $|Q_k^-| = \text{mod-count}_{\mathcal{I}_H^-}(|H|, s, k)$.*

Proof. 1. Denote $e = e^{\text{full}}(i) - i$, $t = e(i) - i - 3\tau + 1$, and $b = e - t$. As shown at the beginning of the proof of Lemma 6.27, for any $i \in \mathbb{R}_H^-$, we have $t > 0$, $[i \dots i + t] \subseteq \mathbb{R}_H^-$, and for any $\delta \in [0 \dots t)$ it holds $e(i + \delta) = e(i)$ and $e^{\text{full}}(i + \delta) = e^{\text{full}}(i)$. This implies $\text{head}(i + \delta) + \exp(i + \delta)|H| = e^{\text{full}}(i + \delta) - (i + \delta) = e^{\text{full}}(i) - (i + \delta) = (e^{\text{full}}(i) - i) - \delta = e - \delta$. Thus, $\text{head}(i + \delta) = (e - \delta) \bmod |H|$ and $\exp(i + \delta) = \lfloor \frac{e - \delta}{|H|} \rfloor$. Hence:

$$\begin{aligned}
|Q_k^- \cap [i \dots e(i) - 3\tau + 2]| &= |\{i + \delta : \delta \in [0 \dots t), \text{head}(i + \delta) = s, \text{ and } \exp(i + \delta) \leq k\}| \\
&= |\{i + \delta : \delta \in [0 \dots t), (e - \delta) \bmod |H| = s, \text{ and } \lfloor \frac{e - \delta}{|H|} \rfloor \leq k\}| \\
&= |\{i + e - j : j \in [b + 1 \dots e + 1), j \bmod |H| = s, \text{ and } \lfloor \frac{j}{|H|} \rfloor \leq k\}| \\
&= |\{j \in [b + 1 \dots e + 1) : j \bmod |H| = s \text{ and } \lfloor \frac{j}{|H|} \rfloor \leq k\}| \\
&= |\{j \in [a_i \dots b_i) : j \bmod |H| = s \text{ and } \lfloor \frac{j}{|H|} \rfloor \leq k\}|,
\end{aligned}$$

where the third equation utilizes that if $\delta \in [0 \dots t)$, then letting $j = e - \delta$, we have $i + \delta = i + e - j$ and $j \in (e - t \dots e] = [b + 1 \dots e + 1)$. We then (fourth equality) used the fact that since i and e are fixed, the size of the “shifted” set does not change. Note that, $b = (\text{head}(i) + \exp(i)|H|) - (e(i) - i - 3\tau + 2) = (e(i) - i - \text{tail}(i)) - (e(i) - i - 3\tau + 2) = 3\tau - 2 - \text{tail}(i) > 0$ follows by $\text{tail}(i) < |H| \leq \tau$. Thus, the interval $[a_i \dots b_i)$ contains only nonnegative integers.

2. By Item 1, for any $i \in \mathbb{R}_H^-$, we can reduce the computation of $|Q_k^- \cap [i \dots i + t)|$, where $t = e(i) - i - 3\tau + 2$, to a modular constraint counting query. Observe that by definition of $e(i)$, if additionally it holds $i \in \mathbb{R}'$, then the interval $[i \dots i + t)$ is a *maximal* interval of positions in \mathbb{R} , i.e., $i - 1, i + t \notin \mathbb{R}$. Thus, letting \mathcal{I}_H^- be the collection of weighted intervals (with weights corresponding to multiplicities) corresponding to all $i \in \mathbb{R}'_H^-$ (Definition 6.20), we obtain the claim by definition of the modular constraint counting query (see Section 5). \square

Proposition 6.31. *Under Assumption 6.21, given any $j \in \mathbb{R}$ and $d \geq \ell$, we can compute $|\text{Occ}_d^{\text{gt}^-}(j)|$ in $\mathcal{O}(t)$ time.*

Proof. The main idea of the algorithm is as follows. By Lemma 6.30, we can reduce the computation of $|\text{Occ}_d^{\text{gt}^-}(j) \cap [i..i+t)]|$, where $i \in \mathbb{R}^-$, $\text{root}(i) = \text{root}(j)$, and $t = e(i) - i - 3\tau + 2$, to an unweighted modular constraint counting query. Therefore, we can compute $|\text{Occ}_d^{\text{gt}^-}(j)|$ using the general (weighted) modular constraint counting queries (Section 5) on \mathcal{I}_H^- as defined in Lemma 6.30. Note that we need to provide efficient modular on each collection \mathcal{I}_H^- separately for every $H \in \text{Roots}$. Since $|H|$ could be large, H cannot be given explicitly during the query. Thus, we specify H using its length and the position of some occurrence in T .

Given any $j \in \mathbb{R}$, we compute $|\text{Occ}_d^{\text{gt}^-}(j)|$ as follows. First, using Assumption 6.21 we compute values $s = \text{head}(j)$, $p = |\text{root}(j)|$, and $e(j)$ in $\mathcal{O}(t)$ time. Note, that then $\text{root}(j) = T[j+s..j+s+p)$, i.e., we have a starting position of an occurrence of $H := \text{root}(j)$ in T . If $e(j) - j < d$, then by Lemma 6.29 we have $\text{Occ}_d^{\text{gt}^-}(j) = \emptyset$, and thus we return $|\text{Occ}_d^{\text{gt}^-}(j)| = 0$. Let us thus assume $e(j) - j \geq d$. We then calculate $\lfloor \frac{d-s}{p} \rfloor$ and by the combination of Lemma 6.29 and Item 2 of Lemma 6.30, we obtain

$$\begin{aligned} |\text{Occ}_d^{\text{gt}^-}(j)| &= |\{j' \in \mathbb{R}_{s,H}^- : \lfloor \frac{d-s}{|H|} \rfloor < \exp(j') \leq n\}| \\ &= \text{mod-count}_{\mathcal{I}_H^-}(p, s, n) - \text{mod-count}_{\mathcal{I}_H^-}(p, s, \lfloor \frac{d-s}{|H|} \rfloor) \end{aligned}$$

which by Assumption 6.21 we can compute in $\mathcal{O}(t)$ time. In total, the query takes $\mathcal{O}(t)$ time. \square

Summary By combining the above results, we obtain the following query algorithm to compute the values $|\text{Occ}_\ell^-(j)|$ and $|\text{Occ}_{2\ell}^-(j)|$, given any position $j \in \mathbb{R}$.

Proposition 6.32. *Under Assumption 6.21, given any position $j \in \mathbb{R}$, we can in $\mathcal{O}(t)$ time compute the values $|\text{Occ}_\ell^-(j)|$ and $|\text{Occ}_{2\ell}^-(j)|$.*

Proof. First, using Proposition 6.28, we compute $\delta_\ell^{\text{eq}} := |\text{Occ}_\ell^{\text{eq}^-}(j)|$ and $\delta_{2\ell}^{\text{eq}} := |\text{Occ}_{2\ell}^{\text{eq}^-}(j)|$ in $\mathcal{O}(t)$ time. Then, using Proposition 6.31, we compute $\delta_\ell^{\text{gt}} := |\text{Occ}_\ell^{\text{gt}^-}(j)|$ and $\delta_{2\ell}^{\text{gt}} := |\text{Occ}_{2\ell}^{\text{gt}^-}(j)|$ in $\mathcal{O}(t)$ time. By Corollary 6.25, we then have $|\text{Occ}_\ell^-(j)| = \delta_\ell^{\text{eq}} + \delta_\ell^{\text{gt}}$ and $|\text{Occ}_{2\ell}^-(j)| = \delta_{2\ell}^{\text{eq}} + \delta_{2\ell}^{\text{gt}}$. \square

6.3.4 Computing the Type

Assume that $i \in [1..n]$ satisfies $\text{SA}[i] \in \mathbb{R}$. In this section, we show how under Assumption 6.21, given i along with $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$ and some $j \in \text{Occ}_\ell(\text{SA}[i])$ (note, that we do not assume anything about $\text{type}(j)$; the query works correctly even if $\text{type}(j) \neq \text{type}(\text{SA}[i])$) to efficiently compute $\text{type}(\text{SA}[i])$, i.e., whether it holds $\text{SA}[i] \in \mathbb{R}^-$ or $\text{SA}[i] \in \mathbb{R}^+$.

The section is organized as follows. Given the input parameters as described above, our query algorithm computes $\text{type}(\text{SA}[i])$ by checking if it holds $\text{SA}[i] \in \text{Occ}_\ell^-(\text{SA}[i])$. To implement such check, we first present a combinatorial result proving that $\text{Occ}_\ell^-(\text{SA}[i])$ occupies a contiguous block of positions in SA and showing what are the endpoints of this block (Lemma 6.33). We then use this characterization to develop an efficient method of checking if $\text{SA}[i] \in \text{Occ}_\ell^-(\text{SA}[i])$ holds (Corollary 6.34). Finally, we develop a query algorithm that efficiently computes $\text{type}(\text{SA}[i])$ in Proposition 6.35.

Lemma 6.33. *Let $i \in [1..n]$ be such that $\text{SA}[i] \in \mathbb{R}$. Denote $b = \text{RangeBeg}_\ell(\text{SA}[i])$ and $e = b + |\text{Occ}_\ell^-(\text{SA}[i])|$. Then, it holds $\text{Occ}_\ell^-(\text{SA}[i]) = \{\text{SA}[i] : i \in (b..e)\}$.*

Proof. Let $s = \text{head}(\text{SA}[i])$ and $H = \text{root}(\text{SA}[i])$. By Item 1 of Lemma 6.24, it holds $\text{Occ}_\ell(\text{SA}[i]) \subseteq \text{R}_{s,H}$. On the other hand, by Lemma 6.18, all elements of $\text{R}_{s,H}$ occupy a contiguous block of positions in SA. Moreover, all elements of $\text{R}_{s,H}^-$ are earlier in the lexicographical order than the elements of $\text{R}_{s,H}^+$. Thus, since $\text{Occ}_\ell(\text{SA}[i])$ by definition also occupies a contiguous block in SA starting at index $b + 1$, the elements of $\text{Occ}_\ell^-(\text{SA}[i])$ (assuming the set is nonempty) must start at this position, and occupy the block of $|\text{Occ}_\ell(\text{SA}[i])|$ consecutive positions. \square

Corollary 6.34. *For any $i \in [1..n]$ such that $\text{SA}[i] \in \text{R}$, $\text{SA}[i] \in \text{Occ}_\ell^-(\text{SA}[i])$ holds if and only if $i - \text{RangeBeg}_\ell(\text{SA}[i]) \leq |\text{Occ}_\ell^-(\text{SA}[i])|$.*

Proof. Assume $\text{SA}[i] \in \text{Occ}_\ell^-(\text{SA}[i])$. By Lemma 6.40, $i \in (b..e]$, where $b = \text{RangeBeg}_\ell(\text{SA}[i])$ and $e = b + |\text{Occ}_\ell^-(\text{SA}[i])|$. In particular, $i \leq e = \text{RangeBeg}_\ell(\text{SA}[i]) + |\text{Occ}_\ell^-(\text{SA}[i])|$.

Assume now $i - \text{RangeBeg}_\ell(\text{SA}[i]) \leq |\text{Occ}_\ell^-(\text{SA}[i])|$. Let $b = \text{RangeBeg}_\ell(\text{SA}[i])$ and $e = b + |\text{Occ}_\ell^-(\text{SA}[i])|$. Then, $i \leq e$. On the other hand, by definition of $\text{Occ}_\ell(\text{SA}[i])$, we have $i \in (\text{RangeBeg}_\ell(\text{SA}[i]).. \text{RangeEnd}_\ell(\text{SA}[i]))$. In particular, $i > \text{RangeBeg}_\ell(\text{SA}[i]) = b$. Therefore, we obtain $i \in (b..e]$. By Lemma 6.33, this implies $\text{SA}[i] \in \text{Occ}_\ell^-(\text{SA}[i])$. \square

Proposition 6.35. *Let $i \in [1..n]$ be such that $\text{SA}[i] \in \text{R}$. Under Assumption 6.21, given i , $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$, and some position $j \in \text{Occ}_\ell(\text{SA}[i])$, we can compute $\text{type}(\text{SA}[i])$ in $\mathcal{O}(t)$ time.*

Proof. The main idea of the query is as follows. By Corollary 6.34, to compute $\text{type}(\text{SA}[i])$ it suffices to know i , $\text{RangeBeg}_\ell(\text{SA}[i])$ and $|\text{Occ}_\ell^-(\text{SA}[i])|$. The first two values are given as input. The third is computed using Proposition 6.32. Note, however, that the query in Proposition 6.32 can compute $\text{Occ}_\ell^-(j)$ only if given j . In our case we do not have $\text{SA}[i]$. We observe, however, that by definition, for any $j \in \text{Occ}_\ell(\text{SA}[i])$ (note that $\text{type}(j)$ can be either -1 or $+1$), it holds $|\text{Occ}_\ell^-(\text{SA}[i])| = |\text{Occ}_\ell^-(j)|$. Thus, we can use j instead of $\text{SA}[i]$.

Given the index i , along with values $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$, and some position $j \in \text{Occ}_\ell(\text{SA}[i])$, we compute $\text{type}(\text{SA}[i])$ as follows. First, using the query from Proposition 6.32, we compute the value $\delta := |\text{Occ}_\ell^-(j)|$ in $\mathcal{O}(t)$ time. By the above discussion, it holds $|\text{Occ}_\ell^-(\text{SA}[i])| = \delta$. By Corollary 6.34, we then have $\text{type}(\text{SA}[i]) = -1$ if and only if $i - \text{RangeBeg}_\ell(\text{SA}[i]) \leq \delta$, which we can evaluate in $\mathcal{O}(1)$ time. \square

6.3.5 Computing the Size of $\text{Pos}_\ell^{\text{low}}(j)$ and $\text{Pos}_\ell^{\text{high}}(j)$

In this section, we present an algorithm to compute the values $\delta_\ell^{\text{low}}(j)$ and $\delta_\ell^{\text{high}}(j)$ for $j \in \text{R}^-$.

The section is organized as follows. We start with the combinatorial result (Lemma 6.36) that shows how to count the elements of $\text{Pos}_\ell^{\text{low}}(j)$ and $\text{Pos}_\ell^{\text{high}}(j)$ that belong to any right-maximal contiguous block of elements of R^- . This result is a very general extension of Lemma 6.27. This generalization, however, is nontrivial and thus below we provide its complete proof (omitting and referring to the appropriate places in the proof of Lemma 6.27 only for parts that are identical). We then use this characterization to develop a query algorithm to compute $\delta_\ell^{\text{low}}(j)$ and $\delta_\ell^{\text{high}}(j)$ given any $j \in \text{R}^-$ (Proposition 6.37). We finally prove (Proposition 6.38) that the computation of $\delta_\ell^{\text{low}}(j)$ (resp. $\delta_\ell^{\text{high}}(j)$) does not actually need the value of j , but it is sufficient to only know that $\text{type}(j) = -1$ and some $j' \in \text{Occ}_\ell(j)$ (resp. $j' \in \text{Occ}_{2\ell}(j)$).

Lemma 6.36. *Assume $i, j \in \mathbb{R}_H^-$ and let $t = e(i) - i - 3\tau + 2$. Then, $|\text{Pos}_\ell^{\text{low}}(j) \cap [i..i+t]| \leq 1$ and $|\text{Pos}_\ell^{\text{high}}(j) \cap [i..i+t]| \leq 1$. Moreover, $|\text{Pos}_\ell^{\text{low}}(j) \cap [i..i+t]| = 1$ (resp. $|\text{Pos}_\ell^{\text{high}}(j) \cap [i..i+t]| = 1$) holds if and only if*

- $e^{\text{full}}(i) - i \geq e^{\text{low}}(j) - j$ (resp. $e^{\text{full}}(i) - i \geq e^{\text{high}}(j) - j$) and
- $T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i) + 7\tau] \succeq T^\infty[e^{\text{low}}(j)..j + \ell]$
(resp. $T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i) + 7\tau] \succeq T^\infty[e^{\text{high}}(j)..j + 2\ell]$).

Proof. As shown at the beginning of the proof of Lemma 6.27, for any $i \in \mathbb{R}_H^-$, letting $t = e(j) - i - 3\tau + 2$, we have $t > 0$, $[i..i+t] \subseteq \mathbb{R}_H^-$, and for any $\delta \in [0..t)$ it holds $e(i+\delta) = e(i)$ and $e^{\text{full}}(i+\delta) = e^{\text{full}}(i)$, which in turn implies $e^{\text{full}}(i+\delta) - (i+\delta) = e^{\text{full}}(i) - i - \delta$. Let $s = \text{head}(j)$. Recall that for any $j' \in \text{Pos}_\ell^{\text{low}}(j)$ (resp. $j' \in \text{Pos}_\ell^{\text{high}}(j)$), we have $e^{\text{full}}(j') - j' = s + \exp(j')|H| = s + k_1|H| = e^{\text{low}}(j) - j$ (resp. $e^{\text{full}}(j') - j' = s + \exp(j')|H| = s + k_2|H| = e^{\text{high}}(j) - j$). Thus, $i+\delta \in \text{Pos}_\ell^{\text{low}}(j)$ (resp. $i+\delta \in \text{Pos}_\ell^{\text{high}}(j)$) implies $e^{\text{full}}(i+\delta) - (i+\delta) = e^{\text{full}}(i) - i - \delta = e^{\text{low}}(j) - j$ (resp. $e^{\text{full}}(i+\delta) - (i+\delta) = e^{\text{full}}(i) - i - \delta = e^{\text{high}}(j) - j$), or equivalently, $\delta = (e^{\text{full}}(i) - i) - (e^{\text{low}}(j) - j)$ (resp. $\delta = (e^{\text{full}}(i) - i) - (e^{\text{high}}(j) - j)$), and thus $|\text{Pos}_\ell^{\text{low}}(j) \cap [i..i+t]| \leq 1$ (resp. $|\text{Pos}_\ell^{\text{high}}(j) \cap [i..i+t]| \leq 1$).

Next, we prove the equivalence. Let us first assume $|\text{Pos}_\ell^{\text{low}}(j) \cap [i..i+t]| = 1$ (resp. $|\text{Pos}_\ell^{\text{high}}(j) \cap [i..i+t]| = 1$), i.e., that $i+\delta \in \text{Pos}_\ell^{\text{low}}(j)$ (resp. $i+\delta \in \text{Pos}_\ell^{\text{high}}(j)$) holds for some $\delta \in [0..t)$. Then, as noted above, $e^{\text{low}}(j) - j = e^{\text{full}}(i) - i - \delta \leq e^{\text{full}}(i) - i$. (resp. $e^{\text{high}}(j) - j = e^{\text{full}}(i) - i - \delta \leq e^{\text{full}}(i) - i$) holds. This establishes the first condition. To show the second condition, let $s = \text{head}(j)$. By $k_1 \leq \exp(j)$ (resp. $k_2 \leq \exp(j)$), we have $T[j..e^{\text{low}}(j)] = H'H^{k_1}$ (resp. $T[j..e^{\text{high}}(j)] = H'H^{k_2}$), where H' is a length- s prefix of H . On the other hand, by definition, $i+\delta \in \text{Pos}_\ell^{\text{low}}(j)$ (resp. $i+\delta \in \text{Pos}_\ell^{\text{high}}(j)$) implies $\text{head}(i+\delta) = s$ and $\exp(i+\delta) = k_1$ (resp. $\exp(i+\delta) = k_2$). Thus, $T[i+\delta..e^{\text{full}}(i+\delta)] = T[i+\delta..e^{\text{full}}(i)] = H'H^{k_1} = T[j..e^{\text{low}}(j)]$ (resp. $T[i+\delta..e^{\text{full}}(i+\delta)] = T[i+\delta..e^{\text{full}}(i)] = H'H^{k_2} = T[j..e^{\text{high}}(j)]$). Therefore, the assumption $T[i+\delta..n] \succeq T[j..n]$ or $\text{LCE}_T(i+\delta, j) \geq \ell$ (resp. $\text{LCE}_T(i+\delta, j) \geq 2\ell$) following from $i+\delta \in \text{Pos}_\ell^{\text{low}}(j)$ (resp. $i+\delta \in \text{Pos}_\ell^{\text{high}}(j)$) is equivalent to $T[e^{\text{full}}(i)..n] \succeq T[e^{\text{low}}(j)..n]$ (resp. $T[e^{\text{full}}(i)..n] \succeq T[e^{\text{high}}(j)..n]$) or $\text{LCE}_T(i+\delta, j) = \text{LCE}_T(e^{\text{full}}(i), e^{\text{low}}(j)) \geq \ell - (e^{\text{low}}(j) - j)$ (resp. $\text{LCE}_T(i+\delta, h) = \text{LCE}_T(e^{\text{full}}(i), e^{\text{high}}(j)) \geq 2\ell - (e^{\text{high}}(j) - j)$). We thus consider two cases:

- If $\text{LCE}_T(e^{\text{full}}(i), e^{\text{low}}(j)) \geq \ell - (e^{\text{low}}(j) - j)$ (resp. $\text{LCE}_T(e^{\text{full}}(i), e^{\text{high}}(j)) \geq 2\ell - (e^{\text{high}}(j) - j)$) holds, then by the assumption $\ell \leq 7\tau$ (resp. $2\ell \leq 7\tau$), we obtain $T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i) + 7\tau] \succeq T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i) + \ell - (e^{\text{low}}(j) - j)] = T^\infty[e^{\text{low}}(j)..j + \ell]$ (resp. $T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i) + 7\tau] \succeq T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i) + 2\ell - (e^{\text{high}}(j) - j)] = T^\infty[e^{\text{high}}(j)..j + 2\ell]$).
- On the other hand, if $T[e^{\text{full}}(i)..n] \succeq T[e^{\text{low}}(j)..n]$ (resp. $T[e^{\text{full}}(i)..n] \succeq T[e^{\text{high}}(j)..n]$) holds, then $T[n]$ being smallest in T implies $T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i)+q] \succeq T^\infty[e^{\text{low}}(j)..e^{\text{low}}(j)+q]$ (resp. $T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i)+q] \succeq T^\infty[e^{\text{high}}(j)..e^{\text{high}}(j)+q]$) for any $q \geq 0$. In particular, for $q = 7\tau$ we have $T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i) + 7\tau] \succeq T^\infty[e^{\text{low}}(j)..e^{\text{low}}(j) + 7\tau] \succeq T^\infty[e^{\text{low}}(j)..j + \ell]$ (resp. $T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i) + 7\tau] \succeq T^\infty[e^{\text{high}}(j)..e^{\text{high}}(j) + 7\tau] \succeq T^\infty[e^{\text{high}}(j)..j + 2\ell]$), where the last inequality follows by $j + \ell \leq e^{\text{low}}(j) + 7\tau$ (resp. $j + 2\ell \leq e^{\text{high}}(j) + 7\tau$).

To prove the opposite implication, assume $e^{\text{full}}(i) - i \geq e^{\text{low}}(j) - j$ (resp. $e^{\text{full}}(i) - i \geq e^{\text{high}}(j) - j$) and $T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i) + 7\tau] \succeq T^\infty[e^{\text{low}}(j)..j + \ell]$ (resp. $T^\infty[e^{\text{full}}(i)..e^{\text{full}}(i) + 7\tau] \succeq T^\infty[e^{\text{high}}(j)..j + 2\ell]$). Let $\delta = (e^{\text{full}}(i) - i) - (e^{\text{low}}(j) - j)$ (resp. $\delta = (e^{\text{full}}(i) - i) - (e^{\text{high}}(j) - j)$). We will prove that $\delta \in [0..t)$ and $i+\delta \in \text{Pos}_\ell^{\text{low}}(j)$ (resp. $i+\delta \in \text{Pos}_\ell^{\text{high}}(j)$). The inequality $\delta \geq 0$ follows from the definition of δ and our assumptions. To show $\delta < t$, we consider two cases:

- Let $e(j) - j < \ell$ (resp. $e(j) - j < 2\ell$). We start by noting $k_1 = \min(\exp(j), \lfloor \frac{\ell-s}{|H|} \rfloor) =$

$\min(\lfloor \frac{e(j)-j-s}{|H|} \rfloor, \lfloor \frac{\ell-s}{|H|} \rfloor) = \lfloor \frac{e(j)-j-s}{|H|} \rfloor = \exp(j)$ (resp. $k_2 = \exp(j)$). Thus, $e^{\text{low}}(j) = j + s + k_1|H| = j + s + \exp(j)|H| = e^{\text{full}}(j)$ (resp. $e^{\text{high}}(j) = j + s + k_2|H| = j + s + \exp(j)|H| = e^{\text{full}}(j)$). Let $q = e(i) - e^{\text{full}}(i)$. Using the same argument as in the proof of Lemma 6.27 (when considering the two cases for the value $e(j) - j$), we obtain that assuming $q < e(j) - e^{\text{full}}(j)$ implies $T[e^{\text{full}}(i) \dots e^{\text{full}}(i) + q] \prec T[e^{\text{full}}(j) \dots e^{\text{full}}(j) + q]$. By $q < e(j) - e^{\text{full}}(j) < j + \ell - e^{\text{full}}(j) = j + \ell - e^{\text{low}}(j) \leq \ell \leq 7\tau$ (resp. $q < e(j) - e^{\text{full}}(j) < j + 2\ell - e^{\text{full}}(j) = j + 2\ell - e^{\text{high}}(j) \leq 2\ell \leq 7\tau$), this implies $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau] \prec T^\infty[e^{\text{low}}(j) \dots j + \ell]$ (resp. $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau] \prec T^\infty[e^{\text{high}}(j) \dots j + 2\ell]$), contradicting our assumption. Thus, it holds $e(i) - e^{\text{full}}(i) \geq e(j) - e^{\text{full}}(j)$ and hence using the same chain of inequalities as in the proof of Lemma 6.27 with $e^{\text{high}}(j)$ replaced by $e^{\text{low}}(j)$ (resp. without any change), we obtain $e(i) - (i + \delta) \geq 3\tau - 1$, or equivalently, $\delta \leq e(i) - i - 3\tau + 1 < t$.

- Let $e(j) - j \geq \ell$ (resp. $e(j) - j \geq 2\ell$). Denote $q = e(i) - e^{\text{full}}(i)$. Using the same argument as in the proof Lemma 6.27 (when considering the two cases for the value $e(j) - j$), we obtain that assuming $q < j + \ell - e^{\text{low}}(j)$ (resp. $q < j + 2\ell - e^{\text{high}}(j)$), it holds $T[e^{\text{full}}(i) \dots e^{\text{full}}(i) + q] \prec T[e^{\text{low}}(j) \dots e^{\text{low}}(j) + q]$ (resp. $T[e^{\text{full}}(i) \dots e^{\text{full}}(i) + q] \prec T[e^{\text{high}}(j) \dots e^{\text{high}}(j) + q]$). By $q < j + \ell - e^{\text{low}}(j) \leq \ell \leq 7\tau$ (resp. $q < j + 2\ell - e^{\text{high}}(j) \leq 2\ell \leq 7\tau$), this implies $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau] \prec T^\infty[e^{\text{low}}(j) \dots j + \ell]$ (resp. $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau] \prec T^\infty[e^{\text{high}}(j) \dots j + 2\ell]$), contradicting our assumption. Thus, $e(i) - e^{\text{full}}(i) \geq j + \ell - e^{\text{low}}(j)$ (resp. $e(i) - e^{\text{full}}(i) \geq j + 2\ell - e^{\text{high}}(j)$) and hence using the same chain of inequalities as in the proof of Lemma 6.27 with $e^{\text{high}}(j)$ replaced by $e^{\text{low}}(j)$ and 2ℓ replaced by ℓ (resp. without any change), we obtain $e(i) - (i + \delta) \geq 3\tau - 1$, or equivalently, $\delta \leq e(i) - i - 3\tau + 1 < t$.

It remains to show $i + \delta \in \text{Pos}_\ell^{\text{low}}(j)$ (resp. $i + \delta \in \text{Pos}_\ell^{\text{high}}(j)$). For this, we first observe that $e^{\text{full}}(i + \delta) - (i + \delta) = e^{\text{full}}(i) - (i + \delta) = e^{\text{low}}(j) - j$ (resp. $e^{\text{full}}(i + \delta) - (i + \delta) = e^{\text{full}}(i) - (i + \delta) = e^{\text{high}}(j) - j$). Combining this with $i + \delta, j \in \mathbb{R}_H$ gives $T[i + \delta \dots e^{\text{full}}(i + \delta)] = T[j \dots e^{\text{low}}(j)]$ (resp. $T[i + \delta \dots e^{\text{full}}(i + \delta)] = T[j \dots e^{\text{high}}(j)]$). This implies $\text{head}(i + \delta) = \text{head}(j)$ and $\exp(i + \delta) = \lfloor (e^{\text{low}}(j) - j)/|H| \rfloor = k_1$ (resp. $\exp(i + \delta) = \lfloor (e^{\text{high}}(j) - j)/|H| \rfloor = k_2$). Moreover, by Lemma 6.19, $\text{type}(i + \delta) = \text{type}(i) = -1$. Thus, we obtain $i + \delta \in \mathbb{R}_{s,H}^-$. It remains to show that it holds $T[i + \delta \dots n] \succeq T[j \dots n]$ or $\text{LCE}_T(i + \delta, j) \geq \ell$ (resp. $\text{LCE}_T(i + \delta, j) \geq 2\ell$). For this, we first note that by $e^{\text{full}}(i + \delta) = e^{\text{full}}(i)$ and $T[i + \delta \dots e^{\text{full}}(i + \delta)] = T[j \dots e^{\text{low}}(j)]$ (resp. $T[i + \delta \dots e^{\text{full}}(i + \delta)] = T[j \dots e^{\text{high}}(j)]$), we have $\text{LCE}_T(i + \delta, j) = e^{\text{low}}(j) - j + \text{LCE}_T(e^{\text{full}}(i), e^{\text{low}}(j))$ (resp. $\text{LCE}_T(i + \delta, j) = e^{\text{high}}(j) - j + \text{LCE}_T(e^{\text{full}}(i), e^{\text{high}}(j))$). Let $q' = \text{LCE}_T(e^{\text{full}}(i), e^{\text{low}}(j))$ (resp. $q' = \text{LCE}_T(e^{\text{full}}(i), e^{\text{high}}(j))$). We consider three cases:

1. First, if $q' \geq \ell - (e^{\text{low}}(j) - j)$ (resp. $q' \geq 2\ell - (e^{\text{high}}(j) - j)$), then we obtain $\text{LCE}_T(i + \delta, j) = e^{\text{low}}(j) - j + q' \geq \ell$ (resp. $\text{LCE}_T(i + \delta, j) = e^{\text{high}}(j) - j + q' \geq 2\ell$).
2. Second, if $q' < \ell - (e^{\text{low}}(j) - j)$ (resp. $q' < 2\ell - (e^{\text{high}}(j) - j)$) and $\max(e^{\text{full}}(i) + q', e^{\text{low}}(j) + q') = n + 1$ (resp. $\max(e^{\text{full}}(i) + q', e^{\text{high}}(j) + q') = n + 1$), then $T[n] = \$$ being unique in T gives $e^{\text{full}}(i) = e^{\text{low}}(j)$ (resp. $e^{\text{full}}(i) = e^{\text{high}}(j)$), and hence $T[i + \delta \dots n] = T[j \dots n]$.
3. Finally, if $q' < \ell - (e^{\text{low}}(j) - j)$ (resp. $q' < 2\ell - (e^{\text{high}}(j) - j)$) and $\max(e^{\text{full}}(i) + q', e^{\text{low}}(j) + q') \leq n$ (resp. $\max(e^{\text{full}}(i) + q', e^{\text{high}}(j) + q') \leq n$), then by $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau] \succeq T^\infty[e^{\text{low}}(j) \dots j + \ell]$ (resp. $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau] \succeq T^\infty[e^{\text{high}}(j) \dots j + 2\ell]$), it holds $T[e^{\text{full}}(i) + q'] \succ T[e^{\text{low}}(j) + q']$ (resp. $T[e^{\text{full}}(i) + q'] \succ T[e^{\text{high}}(j) + q']$) and hence $T[i + \delta \dots n] \succ T[j \dots n]$. \square

Proposition 6.37. *Under Assumption 6.21, given any position $j \in \mathbb{R}^-$, we can in $\mathcal{O}(t)$ time compute $\delta_\ell^{\text{low}}(j)$ and $\delta_\ell^{\text{high}}(j)$.*

Proof. By Lemma 6.36, in order to compute $\delta_\ell^{\text{low}}(j)$ (resp. $\delta_\ell^{\text{high}}(j)$) given $j \in \mathbb{R}_H^-$, it suffices to count the all positions $i \in \mathbb{R}_H^-$ that satisfy $e^{\text{full}}(i) - i \geq e^{\text{low}}(j) - j$ (resp. $e^{\text{full}}(i) - i \geq e^{\text{high}}(j) - j$) and $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau] \succeq T^\infty[e^{\text{low}}(j) \dots j + \ell]$ (resp. $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau] \succeq T^\infty[e^{\text{high}}(j) \dots j + 2\ell]$). The latter task is equivalent to the generalized orthogonal range counting queries (Section 4) on a set of points $\mathcal{P}_H \subseteq \mathcal{X} \times \mathcal{Y}$ (with $\mathcal{X} = \mathbb{Z}_{\geq 0}$ and $\mathcal{Y} = \Sigma^*$) containing the value $e^{\text{full}}(i) - i$ on the \mathcal{X} -coordinate and the string $T^\infty[e^{\text{full}}(i) \dots e^{\text{full}}(i) + 7\tau]$ on the \mathcal{Y} -coordinate for every $i \in \mathbb{R}_H^-$.

Given any position $j \in \mathbb{R}^-$, we compute $\delta_\ell^{\text{low}}(j)$ and $\delta_\ell^{\text{high}}(j)$ as follows. First, using Assumption 6.21 we compute values $s = \text{head}(j)$, $p = |\text{root}(j)|$, and $e(j)$ in $\mathcal{O}(t)$ time. Note, that then $\text{root}(j) = T[j + s \dots j + s + p]$, i.e., we have a starting position of an occurrence of $H := \text{root}(j)$ in T . We then calculate $k = \exp(j) = \lfloor \frac{e(j)-s}{p} \rfloor$ in $\mathcal{O}(1)$ time. Using those values, we further calculate $k_1 = \exp^{\text{cut}}(j, \ell) = \min(k, \lfloor \frac{\ell-s}{p} \rfloor)$, $k_2 = \exp^{\text{cut}}(j, 2\ell) = \min(k, \lfloor \frac{2\ell-s}{p} \rfloor)$, $e^{\text{low}}(j) = s + k_1 p$, and $e^{\text{high}}(j) = s + k_2 p$. By Lemma 6.36 and the definition of \mathcal{P}_H , we now have

$$\begin{aligned} \delta_\ell^{\text{low}}(j) &= \text{r-count}_{\mathcal{P}_H}(e^{\text{low}}(j) - j, n, c^\infty) - \text{r-count}_{\mathcal{P}_H}(e^{\text{low}}(j) - j, n, T^\infty[e^{\text{low}}(j) \dots j + \ell]) \text{ and} \\ \delta_\ell^{\text{high}}(j) &= \text{r-count}_{\mathcal{P}_H}(e^{\text{high}}(j) - j, n, c^\infty) - \text{r-count}_{\mathcal{P}_H}(e^{\text{high}}(j) - j, n, T^\infty[e^{\text{high}}(j) \dots j + 2\ell]), \end{aligned}$$

which by Assumption 6.21 we can compute in $\mathcal{O}(t)$ time using the query defined by Item 1 of Problem 4.5, first with the query arguments $(i, x, q_r) = (e^{\text{low}}(j), e^{\text{low}}(j) - j, j + \ell - e^{\text{low}}(j))$ and then with arguments $(i, x, q_r) = (e^{\text{high}}(j), e^{\text{high}}(j) - j, j + 2\ell - e^{\text{high}}(j))$. To check that all arguments satisfy the requirements of Problem 4.5, recall that $\ell < n$. Thus, $q_r \leq 2\ell < 2n$. On the other hand, $e^{\text{low}}(j) - j \leq e(j) - j < n$ and $e^{\text{high}}(j) - j \leq e(j) - j < n$ hold by $T[n] = \$$. \square

Proposition 6.38. *Let $j \in \mathbb{R}^-$. Under Assumption 6.21, given any position $j' \in \text{Occ}_\ell(j)$ (resp. $j' \in \text{Occ}_{2\ell}(j)$), we can in $\mathcal{O}(t)$ time compute $\delta_\ell^{\text{low}}(j)$ (resp. $\delta_\ell^{\text{high}}(j)$).*

Proof. The main idea of the query is as follows. The algorithm in the proof of Proposition 6.37 needs $s = \text{head}(j)$, $p = |\text{root}(j)|$, the value $e^{\text{low}}(j) - j$ (resp. $e^{\text{high}}(j) - j$), some position $x \in [1 \dots n]$ such that $T^\infty[e^{\text{low}}(j) \dots j + \ell] = T^\infty[x \dots x + j + \ell - e^{\text{low}}(j)]$ (resp. $T^\infty[e^{\text{high}}(j) \dots j + 2\ell] = T^\infty[x \dots x + j + 2\ell - e^{\text{high}}(j)]$), and some position $r \in [1 \dots n]$ such that $\text{root}(j) = T[r \dots r + p]$. By Item 1 of Lemma 6.24, we have $\text{head}(j') = \text{head}(j)$ and $\text{root}(j') = \text{root}(j)$. This implies that we can determine s , p , and position r using j' . On the other hand, to determine $e^{\text{low}}(j) - j$ (resp. $e^{\text{high}}(j) - j$) and the position x we utilize that by Items 2 and 3 of Lemma 6.24 it holds $e^{\text{low}}(j) - j = e^{\text{low}}(j') - j'$ (resp. $e^{\text{high}}(j) - j = e^{\text{high}}(j') - j'$) and position $x = e^{\text{low}}(j')$ (resp. $x = e^{\text{high}}(j')$) satisfies $T^\infty[x \dots x + j + \ell - e^{\text{low}}(j)] = T^\infty[e^{\text{low}}(j) \dots j + \ell]$ (resp. $T^\infty[x \dots x + j + 2\ell - e^{\text{high}}(j)] = T^\infty[e^{\text{high}}(j) \dots j + 2\ell]$).

Given any $j' \in \text{Occ}_\ell(j)$ (resp. $j' \in \text{Occ}_{2\ell}(j)$), we compute $\delta_\ell^{\text{low}}(j)$ (resp. $\delta_\ell^{\text{high}}(j)$) as follows. First, using Assumption 6.21 we compute values $s = \text{head}(j') = \text{head}(j)$, $p = |\text{root}(j')| = |\text{root}(j)|$, and $e(j')$ in $\mathcal{O}(t)$ time. Note, that then the position $r = j' + s$ satisfies $\text{root}(j) = T[r \dots r + p]$, i.e., we have a starting position of an occurrence of $H := \text{root}(j') = \text{root}(j)$ in T . We then calculate $k = \exp(j') = \lfloor \frac{e(j')-s}{p} \rfloor$ in $\mathcal{O}(1)$ time. Using those values, we further calculate $k_1 := \exp^{\text{cut}}(j', \ell) = \min(k, \lfloor \frac{\ell-s}{p} \rfloor)$ (resp. $k_2 := \exp^{\text{cut}}(j', 2\ell) = \min(k, \lfloor \frac{2\ell-s}{p} \rfloor)$) and $e^{\text{low}}(j') - j' = s + k_1 p$ (resp. $e^{\text{high}}(j') - j' = s + k_2 p$). Finally, as in the proof of Proposition 6.37, we obtain

$$\begin{aligned} \delta_\ell^{\text{low}}(j) &= \text{r-count}_{\mathcal{P}_H}(e^{\text{low}}(j) - j, n, c^\infty) - \text{r-count}_{\mathcal{P}_H}(e^{\text{low}}(j) - j, n, T^\infty[e^{\text{low}}(j) \dots j + \ell]) \\ &= \text{r-count}_{\mathcal{P}_H}(e^{\text{low}}(j') - j', n, c^\infty) - \text{r-count}_{\mathcal{P}_H}(e^{\text{low}}(j') - j', n, T^\infty[e^{\text{low}}(j') \dots j' + \ell]) \end{aligned}$$

(resp. $\delta_\ell^{\text{high}}(j) = \text{r-count}_{\mathcal{P}_H}(e^{\text{high}}(j) - j, n, c^\infty) - \text{r-count}_{\mathcal{P}_H}(e^{\text{high}}(j) - j, n, T^\infty[e^{\text{high}}(j) \dots j + 2\ell]) = \text{r-count}_{\mathcal{P}_H}(e^{\text{high}}(j') - j', n, c^\infty) - \text{r-count}_{\mathcal{P}_H}(e^{\text{high}}(j') - j', n, T^\infty[e^{\text{high}}(j') \dots j' + 2\ell])$) in $\mathcal{O}(t)$ time, by using the query defined by Item 1 of Problem 4.5 with the query arguments $(i, x, q_r) = (e^{\text{low}}(j'), e^{\text{low}}(j') - j', j' + \ell - e^{\text{low}}(j'))$ (resp. $(i, x, q_r) = (e^{\text{high}}(j'), e^{\text{high}}(j') - j', j' + 2\ell - e^{\text{high}}(j'))$). The arguments satisfy the requirements of Problem 4.5 by the same logic as in the proof of Proposition 6.37. In total, the query takes $\mathcal{O}(t)$ time. \square

Remark 6.39. Although it may seem that Proposition 6.38 can be simplified by proving that for any $j' \in \text{Occ}_\ell(j)$ (resp. $j' \in \text{Occ}_{2\ell}(j)$) it holds $\text{Pos}_\ell^{\text{low}}(j') = \text{Pos}_\ell^{\text{low}}(j)$ (resp. $\text{Pos}_\ell^{\text{high}}(j') = \text{Pos}_\ell^{\text{high}}(j)$) and it suffices to use Proposition 6.37 on position j' , this does not hold. The problem occurs when $e(j) - j \geq \ell$ (resp. $e(j) - j \geq 2\ell$). Then, it may hold $\text{type}(j') = +1$ and $\text{Pos}_\ell^{\text{low}}(j) \neq \text{Pos}_\ell^{\text{low}}(j')$ ($\text{Pos}_\ell^{\text{high}}(j) \neq \text{Pos}_\ell^{\text{high}}(j')$), where $\text{Pos}_\ell^{\text{low}}(j)$ (resp. $\text{Pos}_\ell^{\text{high}}(j)$) is generalized to j satisfying $\text{type}(j) = +1$ as shown in the proof of Proposition 6.53. We shall later see that in our application of Proposition 6.38 it is not possible to guarantee $\text{type}(j') = -1$ when $\text{type}(j) = -1$, and the current form of Proposition 6.38, in which we do not assume $\text{type}(j')$, is in fact necessary.

6.3.6 Computing the Exponent

Assume that $i \in [1..n]$ satisfies $\text{SA}[i] \in \mathbb{R}^-$ (positions $i \in [1..n]$ satisfying $\text{SA}[i] \in \mathbb{R}^+$ are processed symmetrically; see the proof of Proposition 6.53). In this section, we show that under Assumption 6.21, given the index i along with values $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$, $\delta_\ell^{\text{low}}(\text{SA}[i])$, and some position $j \in \text{Occ}_\ell(\text{SA}[i])$, we can efficiently compute $\text{exp}(\text{SA}[i])$.

The section is organized as follows. Given the input parameters as described above, our query algorithm first checks if it holds $\text{SA}[i] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$. To implement such check, we first present a combinatorial result proving that $\text{Pos}_\ell^{\text{low}}(\text{SA}[i])$ occupies a contiguous block of positions in SA and showing what are the endpoints of this block (Lemma 6.40). We then use this characterization to develop an efficient method of checking if $\text{SA}[i] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$ holds (Corollary 6.42). By definition of $\text{Pos}_\ell^{\text{low}}(\text{SA}[i])$, if $\text{SA}[i] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$, then $\text{exp}(\text{SA}[i]) = \text{exp}^{\text{cut}}(\text{SA}[i], \ell)$, which by Lemma 6.24 is equal to $\text{exp}^{\text{cut}}(j, \ell)$. Thus, the main difficulty is to compute $\text{exp}(\text{SA}[i])$ when $\text{SA}[i] \notin \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$. In Lemma 6.43 we show that in such case the computation of $\text{exp}(\text{SA}[i])$ can be reduced to modular constraint queries (see Section 5). We finally put everything together in Proposition 6.44 to obtain a general query algorithm for computing $\text{exp}(\text{SA}[i])$.

Lemma 6.40. *Let $i \in [1..n]$ be such that $\text{SA}[i] \in \mathbb{R}^-$. Denote $b = \text{RangeBeg}_\ell(\text{SA}[i])$ and $e = b + \delta_\ell^{\text{low}}(\text{SA}[i])$. Then, it holds $\text{Pos}_\ell^{\text{low}}(\text{SA}[i]) = \{\text{SA}[i] : i \in (b..e)\}$.*

Proof. The proof consists of two steps. First, we show that $\text{Pos}_\ell^{\text{low}}(\text{SA}[i]) \subseteq \{\text{SA}[i] : i \in (b..n)\}$. Then, we show that for $i' \in (b+1..n)$, $\text{SA}[i'] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$ implies $\text{SA}[i'-1] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$. This proves that $\text{Pos}_\ell^{\text{low}}(\text{SA}[i]) = \{\text{SA}[i] : i \in (b..e)\}$, where $e = b + |\text{Pos}_\ell^{\text{low}}(\text{SA}[i])| = b + \delta_\ell^{\text{low}}(\text{SA}[i])$, i.e., the claim.

By $\text{SA}[i] \in \text{Occ}_\ell(\text{SA}[i])$, the range $(\text{RangeBeg}_\ell(\text{SA}[i]) \dots \text{RangeEnd}_\ell(\text{SA}[i]))$ is nonempty. In particular, $\text{SA}[b+1] \in \text{Occ}_\ell(\text{SA}[i])$, i.e., $T^\infty[\text{SA}[b+1] \dots \text{SA}[b+1] + \ell] = T^\infty[\text{SA}[i] \dots \text{SA}[i] + \ell]$. Let $i' \in [1..b]$ and denote $j' = \text{SA}[i']$. The condition $i' \notin (b.. \text{RangeEnd}_\ell(\text{SA}[i]))$ implies $T^\infty[j' \dots j' + \ell] \neq T^\infty[\text{SA}[i] \dots \text{SA}[i] + \ell]$. On the other hand, by definition of lexicographical order, $i' < b+1$ implies $T^\infty[j' \dots j' + \ell] \preceq T^\infty[\text{SA}[b+1] \dots \text{SA}[b+1] + \ell] = T^\infty[\text{SA}[i] \dots \text{SA}[i] + \ell]$. Thus, we must have $T^\infty[j' \dots j' + \ell] \prec T^\infty[\text{SA}[i] \dots \text{SA}[i] + \ell]$. Let now $i'' \in [1..n]$ be such that for $j'' = \text{SA}[i'']$ it holds $j'' \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$. By definition of $\text{Pos}_\ell^{\text{low}}(\text{SA}[i])$, this implies $T[j'' \dots n] \succeq$

$T[\text{SA}[i]..n]$ or $\text{LCE}_T(\text{SA}[i], j'') \geq \ell$. By Item 2 in Lemma 6.7 this is equivalent to $T^\infty[j''..j''+\ell'] \succeq T^\infty[\text{SA}[i].. \text{SA}[i] + \ell]$ for any $\ell' \geq \ell$. In particular, $T^\infty[j''..j'' + \ell] \succeq T^\infty[\text{SA}[i].. \text{SA}[i] + \ell]$. We have thus proved that $T^\infty[j'..j' + \ell] \prec T^\infty[\text{SA}[i].. \text{SA}[i] + \ell] \preceq T^\infty[j''..j'' + \ell]$. In particular, $T[j'..j' + \ell] \prec T[j''..j'' + \ell]$, and hence $i' < i''$. Since i' was an arbitrary element of $[1..b]$, we thus obtain $\text{Pos}_\ell^{\text{low}}(\text{SA}[i]) \subseteq \{\text{SA}[i] : i \in (b..n)\}$.

Assume now that for some $i' \in (b+1..n)$ it holds $\text{SA}[i'] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$. We will show that this implies $\text{SA}[i' - 1] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$. Let $s = \text{head}(\text{SA}[i])$ and $H = \text{root}(\text{SA}[i])$.

- First, observe that by $b+1 \leq i' - 1$, the fact that $\text{SA}[b+1] \in \text{Occ}_\ell(\text{SA}[i])$ (see above), and the definition of the lexicographical order, we obtain $T^\infty[\text{SA}[i].. \text{SA}[i] + \ell] = T^\infty[\text{SA}[b+1].. \text{SA}[b+1] + \ell] \preceq T^\infty[\text{SA}[i' - 1].. \text{SA}[i' - 1] + \ell]$. By Item 2 of Lemma 6.7, this implies that $T[\text{SA}[i' - 1]..n] \succeq T[\text{SA}[i]..n]$ or $\text{LCE}_T(\text{SA}[i' - 1], \text{SA}[i]) \geq \ell$.
- Second, by $\text{SA}[b+1] \in \text{Occ}_\ell(\text{SA}[i])$ and Item 1 of Lemma 6.24, we have $\text{SA}[b+1] \in \mathbf{R}_{s,H}$. Furthermore, by $b+1 < i'$, $\text{type}(\text{SA}[i']) = -1$, and Lemma 6.18, it holds $\text{type}(\text{SA}[b+1]) = -1$. Thus, $\text{SA}[b+1] \in \mathbf{R}_{s,H}^-$. On the other hand, by definition of $\text{Pos}_\ell^{\text{low}}(\text{SA}[i])$, $\text{SA}[i'] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$ implies $\text{SA}[i] \in \mathbf{R}_{s,H}^-$. Thus, since by Lemma 6.18 the positions in $\mathbf{R}_{s,H}^-$ occupy a contiguous block in SA and $b+1 \leq i' - 1 < i'$, it holds $\text{SA}[i' - 1] \in \mathbf{R}_{s,H}^-$.
- Denote $k_1 = \exp^{\text{cut}}(\text{SA}[i], \ell)$. Applying Lemma 6.18, we obtain from $i' - 1 < i'$ and $\text{SA}[i'] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$ that $\exp(\text{SA}[i' - 1]) \leq \exp(\text{SA}[i']) = k_1$. On the other hand, by $\text{SA}[b+1] \in \text{Occ}_\ell(\text{SA}[i])$ and Items 1 and 2 of Lemma 6.24, it holds $\text{head}(\text{SA}[b+1]) = s$, $\text{root}(\text{SA}[b+1]) = H$, and $e^{\text{low}}(\text{SA}[b+1]) - \text{SA}[b+1] = e^{\text{low}}(\text{SA}[i]) - \text{SA}[i]$. Consequently,

$$\begin{aligned} \exp^{\text{cut}}(\text{SA}[b+1], \ell) &= \left\lfloor \frac{e^{\text{low}}(\text{SA}[b+1]) - \text{SA}[b+1] - s}{|\text{root}(\text{SA}[b+1])|} \right\rfloor \\ &= \left\lfloor \frac{e^{\text{low}}(\text{SA}[i]) - \text{SA}[i] - s}{|\text{root}(\text{SA}[i])|} \right\rfloor \\ &= \exp^{\text{cut}}(\text{SA}[i], \ell). \end{aligned}$$

Therefore, utilizing one last time Lemma 6.18 for $\text{SA}[b+1]$ and $\text{SA}[i' - 1]$ we obtain from $b+1 \leq i' - 1$ that $k_1 = \exp^{\text{cut}}(\text{SA}[b+1], \ell) \leq \exp(\text{SA}[b+1]) \leq \exp(\text{SA}[i' - 1])$. Combining with the earlier bound $\exp(\text{SA}[i' - 1]) \leq k_1$, this implies $\exp(\text{SA}[i' - 1]) = k_1$.

Combining the above three conditions yields (by definition) $\text{SA}[i' - 1] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$. \square

Remark 6.41. By the above characterization, the set $\text{Pos}_\ell^{\text{low}}(\text{SA}[i])$ occurs as a contiguous block of position in SA starting at index $\text{RangeBeg}_\ell(\text{SA}[i])$. Note, that the set $\text{Occ}_\ell(\text{SA}[i])$ has the same property. These two sets should not be confused, however, and they are not equal, nor one is always a subset of the other.

Corollary 6.42. *For any $i \in [1..n]$ such that $\text{SA}[i] \in \mathbf{R}^-$, $\text{SA}[i] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$ holds if and only if $i - \text{RangeBeg}_\ell(\text{SA}[i]) \leq \delta_\ell^{\text{low}}(\text{SA}[i])$.*

Proof. Assume $\text{SA}[i] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$. By Lemma 6.40, we then must have $i \in (b..e]$, where $b = \text{RangeBeg}_\ell(\text{SA}[i])$ and $e = b + \delta_\ell^{\text{low}}(\text{SA}[i])$. In particular, $i \leq e = \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i])$.

Assume now $i - \text{RangeBeg}_\ell(\text{SA}[i]) \leq \delta_\ell^{\text{low}}(\text{SA}[i])$. Let $b = \text{RangeBeg}_\ell(\text{SA}[i])$ and $e = b + \delta_\ell^{\text{low}}(\text{SA}[i])$. Then, $i \leq e$. On the other hand, by definition of the set $\text{Occ}_\ell(\text{SA}[i])$, we have $i \in (\text{RangeBeg}_\ell(\text{SA}[i]).. \text{RangeEnd}_\ell(\text{SA}[i]))$. In particular, $i > \text{RangeBeg}_\ell(\text{SA}[i]) = b$. Therefore, we obtain $i \in (b..e]$. By Lemma 6.40, this implies $\text{SA}[i] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$. \square

Lemma 6.43. *Assume that $i \in [1..n]$ is such that $\text{SA}[i] \in R_H^-$ (where $H \in \text{Roots}$) and $\text{SA}[i] \notin \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$. Denote $\mathcal{I} = \mathcal{I}_H^-$ (Definition 6.20), $s = \text{head}(\text{SA}[i])$, $p = |H|$, $k_1 = \text{exp}^{\text{cut}}(\text{SA}[i], \ell)$, $c = \text{mod-count}_{\mathcal{I}}(p, s, k_1)$, and $i' = \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i])$. Then, it holds $\text{exp}(\text{SA}[i]) = \text{mod-select}_{\mathcal{I}}(p, s, c + (i - i'))$.*

Proof. For any $k \geq k_1$, denote $P_k = \{j' \in R_{s,H}^- : \text{exp}(j') \in (k_1..k]\}$ and let $m_k = |P_k|$. Our proof consists of two parts.

1. We start by showing that for any $k \geq k_1$, it holds $P_k = \{\text{SA}[i''] : i'' \in (i'..i' + m_k)\}$. We will show this in two substeps. First, we will prove that for any $i'' \in [1..n]$ satisfying $\text{SA}[i''] \in P_k$, it holds $i'' \in (i'..n]$. Second, we show that for any $i'' \in (i' + 1..n]$, $\text{SA}[i''] \in P_k$ implies $\text{SA}[i'' - 1] \in P_k$. These two facts immediately imply the claim.

- Let $i'' \in [1..n]$ be such that $\text{SA}[i''] \in P_k$. Since $P_{k_1} = \emptyset$, this implies $k > k_1$. Note that $\text{SA}[i]$ satisfies all the conditions in the definition of $\text{Pos}_\ell^{\text{low}}(\text{SA}[i])$, except possibly $\text{exp}(\text{SA}[i]) = k_1$. Thus, $\text{SA}[i] \notin \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$ implies $\text{exp}(\text{SA}[i]) \neq \text{exp}^{\text{cut}}(\text{SA}[i], \ell)$. By $\text{exp}^{\text{cut}}(\text{SA}[i], \ell) = \min(\text{exp}(\text{SA}[i]), \lfloor \frac{\ell-s}{|H|} \rfloor) \leq \text{exp}(\text{SA}[i])$, we then must have $k_1 = \lfloor \frac{\ell-s}{|H|} \rfloor$ and $\text{exp}^{\text{cut}}(\text{SA}[i], \ell) < \text{exp}(\text{SA}[i])$. Then, the string $H'H^{k_1+1}$ (where H' is a length- s prefix of H) is a prefix of $T[\text{SA}[i]..n]$. But since $|H'H^{k_1+1}| \geq \ell$, we also obtain that $T^\infty[\text{SA}[i].. \text{SA}[i] + \ell)$ is a prefix of $H'H^{k_1+1}$. Similarly, $\text{SA}[i''] \in R_{s,H}^-$ and $\text{exp}(\text{SA}[i'']) > k_1$ imply that $T^\infty[\text{SA}[i''].. \text{SA}[i''] + \ell)$ is a prefix of $H'H^{k_1+1}$. Therefore, $\text{SA}[i''] \in \text{Occ}_\ell(\text{SA}[i])$ and, consequently, $i'' > \text{RangeBeg}_\ell(\text{SA}[i])$. Moreover, since by Lemma 6.40, $\text{Pos}_\ell^{\text{low}}(\text{SA}[i]) = \{\text{SA}[j] : j \in (\text{RangeBeg}_\ell(\text{SA}[i]).. \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i]))\}$ and $\text{SA}[i''] \notin \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$, we must have $i'' > \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i]) = i'$, or equivalently, $i'' \in (i'..n]$.
- Assume now that for some $i'' \in (i' + 1..n]$ it holds $\text{SA}[i''] \in P_k$. We will show that this implies $\text{SA}[i'' - 1] \in P_k$. As observed above, if $\text{SA}[i''] \in P_k$ for $k > k_1$, then $\text{SA}[i''] \in \text{Occ}_\ell(\text{SA}[i])$. Thus, $i'' - 1 < \text{RangeEnd}_\ell(\text{SA}[i])$. On the other hand, $\text{RangeBeg}_\ell(\text{SA}[i]) \leq i' < i'' - 1$. Consequently, $\text{RangeBeg}_\ell(\text{SA}[i]) < i'' - 1 \leq \text{RangeEnd}_\ell(\text{SA}[i])$, or equivalently, $\text{SA}[i'' - 1] \in \text{Occ}_\ell(\text{SA}[i])$. By Items 1 and 2 of Lemma 6.24, this implies $\text{SA}[i'' - 1] \in R_{s,H}^-$ and $k_1 = \text{exp}^{\text{cut}}(\text{SA}[i], \ell) = \text{exp}^{\text{cut}}(\text{SA}[i'' - 1], \ell) \leq \text{exp}(\text{SA}[i'' - 1])$. To obtain $\text{SA}[i'' - 1] \in P_k$ it thus remains to show $\text{exp}(\text{SA}[i'' - 1]) \neq k_1$. This follows by $\text{SA}[i'' - 1] \notin \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$ (which holds since $i' < i'' - 1$ and by Lemma 6.40 $\text{Pos}_\ell^{\text{low}}(\text{SA}[i]) \subseteq \{\text{SA}[j] : j \in [1..i']\}$) because $\text{SA}[i'' - 1]$ being in $\text{Occ}_\ell(\text{SA}[i])$ implies that it satisfies all other conditions in the definition of $\text{Pos}_\ell^{\text{low}}(\text{SA}[i])$.

2. We can now show $\text{exp}(\text{SA}[i]) = \text{mod-select}_{\mathcal{I}}(p, s, c + (i - i'))$. Denote $k = \text{exp}(\text{SA}[i])$. As noted above $\text{SA}[i] \notin \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$ implies $k_1 < k$. By definition of P_{k-1} and P_k , we have $\text{SA}[i] \in P_k \setminus P_{k-1}$. From Item 1, we have $P_{k-1} = \{\text{SA}[i''] : i'' \in (i'..i' + m_{k-1})\}$ and $P_k = \{\text{SA}[i''] : i'' \in (i'..i' + m_k)\}$. Therefore, $i \in (i' + m_{k-1}..i' + m_k]$. On the other hand, by Lemma 6.30, $m_{k-1} = \text{mod-count}_{\mathcal{I}}(p, s, k - 1) - c$ and $m_k = \text{mod-count}_{\mathcal{I}}(p, s, k) - c$. Therefore, $i \in (i' + \text{mod-count}_{\mathcal{I}}(p, s, k - 1) - c..i' + \text{mod-count}_{\mathcal{I}}(p, s, k) - c]$, or equivalently, $c + (i - i') \in (\text{mod-count}_{\mathcal{I}}(p, s, k - 1).. \text{mod-count}_{\mathcal{I}}(p, s, k))$. By definition of the select query, this implies $k = \text{mod-select}_{\mathcal{I}}(p, s, c + (i - i'))$. Since we defined $k = \text{exp}(\text{SA}[i])$, we obtain the main claim. \square

Proposition 6.44. *Let $i \in [1..n]$ be such that $\text{SA}[i] \in R^-$. Under Assumption 6.21, given the values i , $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$, $\delta_\ell^{\text{low}}(\text{SA}[i])$, and some $j \in \text{Occ}_\ell(\text{SA}[i])$, we can compute*

$\exp(\text{SA}[i])$ in $\mathcal{O}(t)$ time.

Proof. The main idea of the algorithm is as follows. The query algorithm first tests if $\text{SA}[i] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$ holds. By Corollary 6.42 such test can be performed quickly given the values i , $\text{RangeBeg}_\ell(\text{SA}[i])$, and $\delta_\ell^{\text{low}}(\text{SA}[i])$, which are all given as input. The rest of the query algorithm depends on this test. If $\text{SA}[i] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$, then by definition of $\text{Pos}_\ell^{\text{low}}(\text{SA}[i])$ and Lemma 6.24 (see below for details) we immediately obtain $\exp(\text{SA}[i]) = \exp^{\text{cut}}(\text{SA}[i], \ell) = \exp^{\text{cut}}(j, \ell)$. If $\text{SA}[i] \notin \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$, then $\exp(\text{SA}[i])$ is determined according to Lemma 6.43, and it is computed using the modular constraint queries on the set of intervals $\mathcal{I} = \mathcal{I}_H^-$ (where $H = \text{root}(\text{SA}[i])$), which are supported under Assumption 6.21.

Given any index $i \in [1..n]$ such that $\text{SA}[i] \in \mathbb{R}^-$, along with values $\text{RangeBeg}_\ell(\text{SA}[i])$, $\delta_\ell^{\text{low}}(\text{SA}[i])$, and some $j \in \text{Occ}_\ell(\text{SA}[i])$, we compute $\exp(\text{SA}[i])$ as follows. First, using Assumption 6.21 we compute values $s = \text{head}(j) = \text{head}(\text{SA}[i])$, $p = |\text{root}(j)| = |\text{root}(\text{SA}[i])|$, and $e(j)$ in $\mathcal{O}(t)$ time. Note, that then the position $r = j + s$ satisfies $\text{root}(\text{SA}[i]) = T[r..r+p]$, i.e., we have a starting position of an occurrence of $H := \text{root}(j) = \text{root}(\text{SA}[i])$ in T . Denote $\mathcal{I} := \mathcal{I}_H^-$ (see Definition 6.20). We then calculate $k = \exp(j) = \lfloor \frac{e(j)-s}{p} \rfloor$ in $\mathcal{O}(1)$ time. Using those values, we further calculate $k_1 := \min(k, \lfloor \frac{\ell-s}{p} \rfloor) = \exp^{\text{cut}}(j, \ell) = \exp^{\text{cut}}(\text{SA}[i], \ell)$ (the last equality follows by $j \in \text{Occ}_\ell(\text{SA}[i])$ and Items 1 and 2 of Lemma 6.24). Next, in $\mathcal{O}(1)$ time we determine if $\text{SA}[i] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$. By Corollary 6.42, $\text{SA}[i] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ holds if and only if $i \leq \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i])$. Consider two cases:

- Assume $i \leq \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i])$ (i.e., $\text{SA}[i] \in \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$). Then, by definition of $\text{Pos}_\ell^{\text{low}}(\text{SA}[i])$, we have $\exp(\text{SA}[i]) = k_1$.
- Assume now $i > \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i])$ (i.e., $\text{SA}[i] \notin \text{Pos}_\ell^{\text{low}}(\text{SA}[i])$). First, using a modular constraint counting query (Section 5) we compute the value $c = \text{mod-count}_{\mathcal{I}}(p, s, k_1)$. By Assumption 6.21, this takes $\mathcal{O}(t)$ time. Next, we compute $i' = \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i])$ in $\mathcal{O}(1)$ time. Finally, using a modular constraint selection query (Section 5) we compute $r = \text{mod-select}_{\mathcal{I}}(p, s, c + (i - i'))$. By Assumption 6.21, this takes $\mathcal{O}(t)$ time and by Lemma 6.43, we then have $\exp(\text{SA}[i]) = r$. \square

6.3.7 Computing the Size of $\text{Pos}_\ell^{\text{mid}}(j)$

In this section, we show that under Assumption 6.21, we can efficiently compute $\delta_\ell^{\text{mid}}(j) = |\text{Pos}_\ell^{\text{mid}}(j)|$ for any $j \in \mathbb{R}^-$ ($j \in \mathbb{R}^+$ can be processed symmetrically; see Proposition 6.53).

The section is organized as follows. We first develop an algorithm that takes the position j as input and returns $\delta_\ell^{\text{mid}}(j)$ (Proposition 6.45). We then prove (Proposition 6.46) that the computation of $\delta_\ell^{\text{mid}}(j)$ does not actually need the value of j , but it is sufficient to only know $\exp(j)$ and some $j' \in \text{Occ}_\ell(j)$.

Proposition 6.45. *Under Assumption 6.21, given any position $j \in \mathbb{R}^-$, we can in $\mathcal{O}(t)$ time compute $\delta_\ell^{\text{mid}}(j)$.*

Proof. The main idea of the query is as follows. By Lemma 6.30, we can reduce the computation of $|\text{Pos}_\ell^{\text{mid}}(j) \cap [i..i+t]|$, where $i \in \mathbb{R}^-$, $\text{root}(i) = \text{root}(j)$, and $t = e(i) - i - 3\tau + 2$, to two modular constraint counting queries. Observe that if additionally, $i \in \mathbb{R}'$, then this interval $[i..i+t]$ is a maximal interval of positions in \mathbb{R} , i.e., $i-1, i+t \notin \mathbb{R}$. Thus, letting \mathcal{I}_H^- be the collection of weighted intervals (with weights corresponding to multiplicities) constructed as in Definition 6.20

for all $i \in \mathbb{R}^-$ satisfying $\text{root}(i) = \text{root}(j)$, we can compute $\delta_\ell^{\text{mid}}(j)$ using the general (weighted) modular constraint counting queries (Section 5) on \mathcal{I}_H^- .

Given any $j \in \mathbb{R}^-$, we compute $\delta_\ell^{\text{mid}}(j)$ as follows. First, using Assumption 6.21 we compute values $s = \text{head}(j)$, $p = |\text{root}(j)|$, and $e(j)$ in $\mathcal{O}(t)$ time. Note, that then the position $r = j + s$ satisfies $\text{root}(j) = T[r \dots r + p]$, i.e., we have a starting position of an occurrence of $H := \text{root}(j)$ in T . We then calculate $k = \exp(j) = \lfloor \frac{e(j)-s}{p} \rfloor$ in $\mathcal{O}(1)$ time. Using those values, we further calculate $k_1 := \exp^{\text{cut}}(j, \ell) = \min(k, \lfloor \frac{\ell-s}{p} \rfloor)$ and $k_2 := \exp^{\text{cut}}(j, 2\ell) = \min(k, \lfloor \frac{2\ell-s}{p} \rfloor)$. By Lemma 6.30 we then obtain

$$\delta_\ell^{\text{mid}}(j) = \text{mod-count}_{\mathcal{I}_H^-}(p, s, k_2) - \text{mod-count}_{\mathcal{I}_H^-}(p, s, k_1)$$

which by Assumption 6.21 we can compute in $\mathcal{O}(t)$ time. In total, the query takes $\mathcal{O}(t)$ time. \square

Proposition 6.46. *Let $j \in \mathbb{R}^-$. Under Assumption 6.21, given some position $j' \in \text{Occ}_\ell(j)$, and the value $\exp(j)$, we can in $\mathcal{O}(t)$ time compute $\delta_\ell^{\text{mid}}(j)$.*

Proof. The main idea is as follows. The algorithm in the proof of Proposition 6.45 needs $s = \text{head}(j)$, $p = |\text{root}(j)|$, $k = \exp(j)$, and some $r \in [1..n]$ such that $\text{root}(j) = T[r \dots r + p]$. By Item 1 of Lemma 6.24, for $j' \in \text{Occ}_\ell(j)$, we have $\text{head}(j') = \text{head}(j)$ and $\text{root}(j') = \text{root}(j)$. This implies that we can determine s , p , and r from j' . The remaining information needed during the query ($\exp(j)$) is given as input.

Given some $j' \in \text{Occ}_\ell(j)$ and the value $k = \exp(j)$ as input, we compute $\delta_\ell^{\text{mid}}(j)$ as follows. First, using Assumption 6.21 we compute $s = \text{head}(j') = \text{head}(j)$ and $p = |\text{root}(j')| = |\text{root}(j)|$ in $\mathcal{O}(t)$ time. Note, that then the position $r = j' + s$ satisfies $\text{root}(j) = T[r \dots r + p]$, i.e., we have a starting position of an occurrence of $H := \text{root}(j) = \text{root}(j')$ in T . Using those values, we further calculate $k_1 := \exp^{\text{cut}}(j, \ell) = \min(k, \lfloor \frac{\ell-s}{p} \rfloor)$ and $k_2 := \exp^{\text{cut}}(j, 2\ell) = \min(k, \lfloor \frac{2\ell-s}{p} \rfloor)$. Finally, as in the proof of Proposition 6.45, we obtain $\delta_\ell^{\text{mid}}(j) = \text{mod-count}_{\mathcal{I}_H^-}(p, s, k_2) - \text{mod-count}_{\mathcal{I}_H^-}(p, s, k_1)$, which by Assumption 6.21 we can compute in $\mathcal{O}(t)$ time. \square

6.3.8 Computing a Position in $\text{Occ}_{2\ell}(\text{SA}[i])$

Assume that $i \in [1..n]$ satisfies $\text{SA}[i] \in \mathbb{R}^-$ ($i \in [1..n]$ satisfying $\text{SA}[i] \in \mathbb{R}^+$ are processed symmetrically; see the proof of Proposition 6.53). In this section, we show how under Assumption 6.21, given i along with $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$, $\delta_\ell^{\text{low}}(\text{SA}[i])$, $\delta_\ell^{\text{mid}}(\text{SA}[i])$, $\exp(\text{SA}[i])$, and some $j \in \text{Occ}_\ell(\text{SA}[i])$, to efficiently compute some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$.

The section is organized as follows. Given the input parameters as described above, our query algorithm first checks if it holds $\text{SA}[i] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$. To implement such check, we first present a combinatorial result proving that $\text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ occupies a contiguous block of positions in SA and showing what are the endpoints of this block (Lemma 6.47). We then use this characterization to develop an efficient method of checking if $\text{SA}[i] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ holds (Corollary 6.49). In the next two results (Lemmas 6.50 and 6.51), we show how in each of the two cases reduce the computation of some position $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ to a generalized range selection query (see Section 4). We then put everything together in Proposition 6.52.

Lemma 6.47. *Let $i \in [1..n]$ be such that $\text{SA}[i] \in \mathbb{R}^-$. Denote $b = \text{RangeBeg}_{2\ell}(\text{SA}[i])$ and $e = b + \delta_\ell^{\text{high}}(\text{SA}[i])$. Then, it holds $\text{Pos}_\ell^{\text{high}}(\text{SA}[i]) = \{\text{SA}[i] : i \in (b..e)\}$.*

Proof. The proof is analogous to the proof of Lemma 6.40. We will thus omit the parts that are identical as in the proof of Lemma 6.47. First, we show that $\text{Pos}_\ell^{\text{high}}(\text{SA}[i]) \subseteq \{\text{SA}[i] : i \in (b..n)\}$. Then, we show that for $i' \in (b+1..n)$, $\text{SA}[i'] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ implies $\text{SA}[i'-1] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$. This proves that $\text{Pos}_\ell^{\text{high}}(\text{SA}[i]) = \{\text{SA}[i] : i \in (b..e)\}$, where $e = b + |\text{Pos}_\ell^{\text{high}}(\text{SA}[i])| = b + \delta_\ell^{\text{high}}(\text{SA}[i])$, i.e., the claim.

By $\text{SA}[i] \in \text{Occ}_{2\ell}(\text{SA}[i])$, the range $(\text{RangeBeg}_{2\ell}(\text{SA}[i]).. \text{RangeEnd}_{2\ell}(\text{SA}[i]))$ is nonempty. In particular, $\text{SA}[b+1] \in \text{Occ}_{2\ell}(\text{SA}[i])$, i.e., $T^\infty[\text{SA}[b+1].. \text{SA}[b+1]+2\ell] = T^\infty[\text{SA}[i].. \text{SA}[i]+2\ell]$. Let $i' \in [1..b]$ and denote $j' = \text{SA}[i']$. The condition $i' \notin (b.. \text{RangeEnd}_{2\ell}(\text{SA}[i]))$ implies $T^\infty[j'..j'+2\ell] \neq T^\infty[\text{SA}[i].. \text{SA}[i]+2\ell]$. On the other hand, by definition of lexicographical order, $i' < b+1$ implies $T^\infty[j'..j'+2\ell] \preceq T^\infty[\text{SA}[b+1].. \text{SA}[b+1]+2\ell] = T^\infty[\text{SA}[i].. \text{SA}[i]+2\ell]$. Thus, we must have $T^\infty[j'..j'+2\ell] \prec T^\infty[\text{SA}[i].. \text{SA}[i]+2\ell]$. Let now $i'' \in [1..n]$ be such that for $j'' = \text{SA}[i'']$ it holds $j'' \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$. By definition of $\text{Pos}_\ell^{\text{high}}(\text{SA}[i])$, this implies $T[j''..n] \succeq T[\text{SA}[i]..n]$ or $\text{LCE}_T(\text{SA}[i], j'') \geq 2\ell$. By Item 2 in Lemma 6.7 this is equivalent to $T^\infty[j''..j''+\ell'] \succeq T^\infty[\text{SA}[i].. \text{SA}[i]+2\ell]$ for any $\ell' \geq 2\ell$. In particular, $T^\infty[j''..j''+2\ell] \succeq T^\infty[\text{SA}[i].. \text{SA}[i]+2\ell]$. We have thus proved that $T^\infty[j'..j'+2\ell] \prec T^\infty[\text{SA}[i].. \text{SA}[i]+2\ell] \preceq T^\infty[j''..j''+2\ell]$. In particular, $T[j'..j'+2\ell] \prec T[j''..j''+2\ell]$, and hence $i' < i''$. Since i' was an arbitrary element of $[1..b]$, we thus obtain $\text{Pos}_\ell^{\text{high}}(\text{SA}[i]) \subseteq \{\text{SA}[i] : i \in (b..n)\}$.

Assume now that for some $i' \in (b+1..n)$ it holds $\text{SA}[i'] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$. We will show that this implies $\text{SA}[i'-1] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$. Let $s = \text{head}(\text{SA}[i])$ and $H = \text{root}(\text{SA}[i])$.

- First, observe that by $b+1 \leq i'-1$, the fact that $\text{SA}[b+1] \in \text{Occ}_{2\ell}(\text{SA}[i])$ (see above), and the definition of the lexicographical order, we obtain $T^\infty[\text{SA}[i].. \text{SA}[i]+2\ell] = T^\infty[\text{SA}[b+1].. \text{SA}[b+1]+2\ell] \preceq T^\infty[\text{SA}[i'-1].. \text{SA}[i'-1]+2\ell]$. By Item 2 of Lemma 6.7, this implies that $T[\text{SA}[i'-1]..n] \succeq T[\text{SA}[i]..n]$ or $\text{LCE}_T(\text{SA}[i'-1], \text{SA}[i]) \geq 2\ell$.
- Second, by $\text{SA}[b+1] \in \text{Occ}_{2\ell}(\text{SA}[i])$ and Item 1 of Lemma 6.24, we have $\text{SA}[b+1] \in \mathbf{R}_{s,H}$. Thus, using the argument from the proof of Lemma 6.40, we have $\text{SA}[i'-1] \in \mathbf{R}_{s,H}^-$.
- Denote $k_2 = \exp^{\text{cut}}(\text{SA}[i], 2\ell)$. Applying Lemma 6.18, we obtain from $i'-1 < i'$ and $\text{SA}[i'] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ that $\exp(\text{SA}[i'-1]) \leq \exp(\text{SA}[i']) = k_2$. On the other hand, by $\text{SA}[b+1] \in \text{Occ}_{2\ell}(\text{SA}[i])$ and Items 1 and 2 of Lemma 6.24, it holds $\text{head}(\text{SA}[b+1]) = s$, $\text{root}(\text{SA}[b+1]) = H$, and $e^{\text{high}}(\text{SA}[b+1]) - \text{SA}[b+1] = e^{\text{high}}(\text{SA}[i]) - \text{SA}[i]$. Consequently, (see the proof of Lemma 6.40), $\exp^{\text{cut}}(\text{SA}[b+1], 2\ell) = \exp^{\text{cut}}(\text{SA}[i], 2\ell)$. Therefore, utilizing one last time Lemma 6.18 for $\text{SA}[b+1]$ and $\text{SA}[i'-1]$ we obtain from $b+1 \leq i'-1$ that $k_2 = \exp^{\text{cut}}(\text{SA}[b+1], 2\ell) \leq \exp(\text{SA}[b+1]) \leq \exp(\text{SA}[i'-1])$. Combining with the earlier bound $\exp(\text{SA}[i'-1]) \leq k_2$, this implies $\exp(\text{SA}[i'-1]) = k_2$.

Combining the above three conditions yields (by definition) $\text{SA}[i'-1] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$. \square

Remark 6.48. Similarly as for $\text{Pos}_\ell^{\text{low}}(\text{SA}[i])$ (see Lemma 6.40), by the above characterization, the set $\text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ occurs as a contiguous block of position in SA starting at index $\text{RangeBeg}_{2\ell}(\text{SA}[i])$. Note, that the set $\text{Occ}_{2\ell}(\text{SA}[i])$ has the same property. These two sets should not be confused, however, and they are not equal, nor one is always a subset of the other. We will, however, use $\text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ to infer about $\text{Occ}_{2\ell}(\text{SA}[i])$. More precisely, to compute a position $j \in \text{Occ}_{2\ell}(\text{SA}[i])$, we will distinguish two cases: $\text{SA}[i] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ and $\text{SA}[i] \notin \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$. In the first case, we will indeed locate an element of $\text{Occ}_{2\ell}(\text{SA}[i])$ from $\text{Pos}_\ell^{\text{high}}(\text{SA}[i])$. We first, however, need to develop an efficient test of whether $\text{SA}[i] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ holds.

Corollary 6.49. *For any $i \in [1..n]$ such that $\text{SA}[i] \in \mathbb{R}^-$, $\text{SA}[i] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ holds if and only if $i - \text{RangeBeg}_\ell(\text{SA}[i]) \leq \delta_\ell^{\text{low}}(\text{SA}[i]) + \delta_\ell^{\text{mid}}(\text{SA}[i])$.*

Proof. Assume $\text{SA}[i] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$. By Lemma 6.47, we then must have $i \in (b..e]$, where $b = \text{RangeBeg}_{2\ell}(\text{SA}[i])$ and $e = b + \delta_\ell^{\text{high}}(\text{SA}[i])$. In particular, $i \leq e = \text{RangeBeg}_{2\ell}(\text{SA}[i]) + \delta_\ell^{\text{high}}(\text{SA}[i]) = \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell(\text{SA}[i]) + \delta_\ell^{\text{high}}(\text{SA}[i]) = \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i]) + \delta_\ell^{\text{mid}}(\text{SA}[i])$, where the last equality follows by Lemma 6.23. This is equivalent to the claim.

Assume now $i - \text{RangeBeg}_\ell(\text{SA}[i]) \leq \delta_\ell^{\text{low}}(\text{SA}[i]) + \delta_\ell^{\text{mid}}(\text{SA}[i])$. Let $b = \text{RangeBeg}_{2\ell}(\text{SA}[i])$ and $e = b + \delta_\ell^{\text{high}}(\text{SA}[i])$. Then, by Lemma 6.23, we equivalently have $i \leq \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i]) + \delta_\ell^{\text{mid}}(\text{SA}[i]) = \text{RangeBeg}_{2\ell}(\text{SA}[i]) + \delta_\ell^{\text{high}}(\text{SA}[i]) = e$. On the other hand, by definition, we have $i \in (\text{RangeBeg}_{2\ell}(\text{SA}[i]).. \text{RangeEnd}_{2\ell}(\text{SA}[i]))$. In particular, $i > \text{RangeBeg}_{2\ell}(\text{SA}[i]) = b$. Therefore, we obtain $i \in (b..e]$. By Lemma 6.47, this implies $\text{SA}[i] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$. \square

Lemma 6.50. *Assume that $i \in [1..n]$ is such that $\text{SA}[i] \in \mathbb{R}_H^- \cap \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$, where $H \in \text{Roots}$. Let $\mathcal{P}_H = \{(e^{\text{full}}(r), e^{\text{full}}(r) - r) : r \in \mathbb{R}_H^-\}$, $\mathcal{P} = \text{Points}_{7\tau}(T, \mathcal{P}_H)$, $d = \delta_\ell^{\text{high}}(\text{SA}[i])$, $x = e^{\text{high}}(\text{SA}[i]) - \text{SA}[i]$, $m = \text{r-count}_{\mathcal{P}}(x, n)$, and $e = \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i]) + \delta_\ell^{\text{mid}}(\text{SA}[i])$. Then $d \leq m$. Moreover:*

1. For $\delta \in [0..d]$, any position $p \in \text{r-select}_{\mathcal{P}}(x, n, m - \delta)$ satisfies $T^\infty[p - x..p - x + 2\ell] = T^\infty[\text{SA}[e - \delta].. \text{SA}[e - \delta] + 2\ell]$.
2. For $\delta = e - i$, we have $\delta \in [0..d]$ and any position $p \in \text{r-select}_{\mathcal{P}}(x, n, m - \delta)$ satisfies $p - x \in \text{Occ}_{2\ell}(\text{SA}[i])$.

Proof. Note that \mathcal{P} is well-defined, since positions $e^{\text{full}}(r)$ are distinct among $r \in \mathbb{R}_H^-$. Denote $q = |\mathbb{R}_H^-|$. Let $(a_j)_{j \in [1..q]}$ be a sequence containing all positions $r \in \mathbb{R}_H^-$ ordered according to the string $T^\infty[e^{\text{full}}(r)..e^{\text{full}}(r) + 7\tau]$. In other words, for any $j, j' \in [1..q]$, $j < j'$ implies $T^\infty[e^{\text{full}}(r)..e^{\text{full}}(r) + 7\tau] \prec T^\infty[e^{\text{full}}(r')..e^{\text{full}}(r') + 7\tau]$, where $r = a_j$ and $r' = a_{j'}$. Note, that the sequence $(a_j)_{j \in [1..q]}$ is not unique. Since $\{(e^{\text{full}}(a_i), e^{\text{full}}(a_i) - a_i) : i \in [1..q]\} = \mathcal{P}_H$, it holds $|\{e^{\text{full}}(a_j) - x : j \in [1..q] \text{ and } e^{\text{full}}(a_j) - a_j \geq x\}| = |\{j \in [1..q] : e^{\text{full}}(a_j) - a_j \geq x\}| = |\{(p, v) \in \mathcal{P}_H : v \in [x..n]\}| = m$, where the last equality follows by the definition of $\text{r-count}_{\mathcal{P}}(x, n)$ (see Section 4.2), and by observing that for any $q \in \mathbb{R}$, it holds $e^{\text{full}}(q) - q < n$. On the other hand, observe that by Lemma 6.36, for any $j \in [1..q]$, the right-maximal run of positions in \mathbb{R} starting at position $r = a_j$ contains an element p' of $\text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ if and only if $e^{\text{full}}(r) - r \geq x$ and $T^\infty[e^{\text{full}}(r)..e^{\text{full}}(r) + 7\tau] \succeq T^\infty[e^{\text{high}}(\text{SA}[i]).. \text{SA}[i] + 2\ell]$. Moreover, if the latter condition is true, then since by definition on $\text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ any such element p' must satisfy $\text{exp}(p') = \text{exp}^{\text{cut}}(\text{SA}[i], 2\ell)$ (or equivalently, $e^{\text{full}}(p') - p' = s + \text{exp}^{\text{cut}}(\text{SA}[i], 2\ell)|_H = e^{\text{high}}(\text{SA}[i]) - \text{SA}[i] = x$, where $s = \text{head}(\text{SA}[i]) = \text{head}(p')$), we obtain $p' = e^{\text{full}}(p') - x = e^{\text{full}}(r) - x$. In other words, $\text{Pos}_\ell^{\text{high}}(\text{SA}[i]) = \{e^{\text{full}}(a_j) - x : j \in [1..q], e^{\text{full}}(a_j) - a_j \geq x, \text{ and } T^\infty[e^{\text{full}}(a_j)..e^{\text{full}}(a_j) + 7\tau] \succeq T^\infty[e^{\text{high}}(\text{SA}[i]).. \text{SA}[i] + 2\ell]\}$. The latter set (whose cardinality is equal to d) is clearly a subset of $\{e^{\text{full}}(a_j) - x : j \in [1..q] \text{ and } e^{\text{full}}(a_j) - a_j \geq x\}$ (whose size, as shown above, is m). Thus, $d \leq m$.

1. As shown above, $|\{j \in [1..q] : e^{\text{full}}(a_j) - a_j \geq x\}| = m$. Let $(b_j)_{j \in [1..m]}$ be a subsequence of $(a_j)_{j \in [1..q]}$ containing all the elements of the set $\{a_j : j \in [1..q] \text{ and } e^{\text{full}}(a_j) - a_j \geq x\}$ (in the same order as they appear in the sequence $(a_j)_{j \in [1..q]}$). Our proof consists of three steps:

- (i) Let $j \in [1..m]$. We first show $b_j \in \text{r-select}_{\mathcal{P}}(x, n, j)$. Denote $Y = T^\infty[e^{\text{full}}(b_j)..e^{\text{full}}(b_j) + 7\tau]$.

Let

$$r_{\text{beg}} = |\{a_t : t \in [1..q], e^{\text{full}}(a_t) - a_t \geq x, \text{ and } T^\infty[e^{\text{full}}(a_t) \dots e^{\text{full}}(a_t) + 7\tau] \prec Y\}| \text{ and}$$

$$r_{\text{end}} = |\{a_t : t \in [1..q], e^{\text{full}}(a_t) - a_t \geq x, \text{ and } T^\infty[e^{\text{full}}(a_t) \dots e^{\text{full}}(a_t) + 7\tau] \preceq Y\}|.$$

If an index $t \in [1..q]$ satisfies $e^{\text{full}}(a_t) - a_t \geq x$, then we also have $a_t \in \{b_1, \dots, b_m\}$. Moreover, since for any indexes $t, t' \in [1..m]$, $t < t'$ implies $T^\infty[e^{\text{full}}(b_t) \dots e^{\text{full}}(b_t) + 7\tau] \preceq T^\infty[e^{\text{full}}(b_{t'}) \dots e^{\text{full}}(b_{t'}) + 7\tau]$, if $t \in [1..q]$ additionally satisfies $T^\infty[e^{\text{full}}(a_t) \dots e^{\text{full}}(a_t) + 7\tau] \prec Y = T^\infty[e^{\text{full}}(b_j) \dots e^{\text{full}}(b_j) + 7\tau]$, then $a_t \in \{b_1, \dots, b_{j-1}\}$. Thus, $r_{\text{beg}} < j$. On the other hand, every $t \in [1..j]$ satisfies $e^{\text{full}}(b_t) - b_t \geq x$ and $T^\infty[e^{\text{full}}(b_t) \dots e^{\text{full}}(b_t) + 7\tau] \preceq T^\infty[e^{\text{full}}(b_j) \dots e^{\text{full}}(b_j) + 7\tau] = Y$. Thus, $j \leq r_{\text{end}}$. Altogether, $j \in (r_{\text{beg}} \dots r_{\text{end}}]$. By definition of $\mathcal{P} = \text{Points}_{7\tau}(T, \mathcal{P}_H)$ (Definition 4.4), it holds $r_{\text{beg}} = \text{r-count}_{\mathcal{P}}(x, n, Y)$ and $r_{\text{end}} = \text{r-count}_{\mathcal{P}}^{\text{inc}}(x, n, Y)$. Thus, $j \in (\text{r-count}_{\mathcal{P}}(x, n, Y) \dots \text{r-count}_{\mathcal{P}}^{\text{inc}}(x, n, Y)]$. On the other hand, by $(e^{\text{full}}(b_j), e^{\text{full}}(b_j) - b_j) \in \mathcal{P}_H$, we have $(e^{\text{full}}(b_j) - b_j, Y, f) \in \mathcal{P}$. Combining with $x \leq e^{\text{full}}(b_j) - b_j < n$ we obtain $b_j \in \text{r-select}_{\mathcal{P}}(x, n, j)$.

- (ii) Let $j \in [1..m]$. We will now show that for any $p \in \text{r-select}_{\mathcal{P}}(x, n, j)$, it holds $T^\infty[p - x \dots p + 7\tau] = T^\infty[e^{\text{full}}(b_j) - x \dots e^{\text{full}}(b_j) + 7\tau]$. By Item (i) and the definition of $\text{r-select}_{\mathcal{P}}(x, n, j)$, the assumption $p \in \text{r-select}_{\mathcal{P}}(x, n, j)$ implies $T^\infty[p \dots p + 7\tau] = T^\infty[b_j \dots b_j + 7\tau]$. Moreover, $x \leq e^{\text{full}}(p) - p < n$ and there exists $r \in \mathcal{R}_H^-$ such that $p = e^{\text{full}}(r)$. The position p is thus preceded in T^∞ by a string $H'H^z$, where H' is a prefix of H of length $s = \text{head}(\text{SA}[i])$ and $z = \text{exp}^{\text{cut}}(\text{SA}[i], 2\ell)$. Since by definition of $(b_j)_{j \in [1..m]}$, the position $e^{\text{full}}(b_j)$ is also preceded by $H'H^z$ in T and $|H'H^z| = x$, we obtain $T^\infty[p - x \dots p + 7\tau] = T^\infty[e^{\text{full}}(b_j) - x \dots e^{\text{full}}(b_j) + 7\tau]$.
- (iii) We are now ready to prove the main claim. As noted above, $\text{Pos}_\ell^{\text{high}}(\text{SA}[i]) = \{e^{\text{full}}(a_j) - x : j \in [1..q], e^{\text{full}}(a_j) - a_j \geq x, \text{ and } T^\infty[e^{\text{full}}(a_j) \dots e^{\text{full}}(a_j) + 7\tau] \succeq T^\infty[e^{\text{high}}(\text{SA}[i]) \dots \text{SA}[i] + 2\tau]\}$. Since the positions k in $(a_j)_{j \in [1..q]}$ are sorted by $T^\infty[e^{\text{full}}(k) \dots e^{\text{full}}(k) + 7\tau]$, we can eliminate the second condition. Denoting $j_{\text{skip}} = |\{j \in [1..q] : T^\infty[e^{\text{full}}(a_j) \dots e^{\text{full}}(a_j) + 7\tau] \prec T^\infty[e^{\text{high}}(\text{SA}[i]) \dots \text{SA}[i] + 2\tau]\}|$, we have

$$\text{Pos}_\ell^{\text{high}}(\text{SA}[i]) = \{e^{\text{full}}(a_j) - x : j \in (j_{\text{skip}} \dots q] \text{ and } e^{\text{full}}(a_j) - a_a \geq x\}.$$

Consequently, by definition of $(b_j)_{j \in [1..m]}$, we have $\text{Pos}_\ell^{\text{high}}(\text{SA}[i]) = \{e^{\text{full}}(b_j) - x : j \in (m - d \dots m)\}$. On the other hand, by Lemma 6.23, $e = \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i]) + \delta_\ell^{\text{mid}}(\text{SA}[i]) = \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell(\text{SA}[i]) + \delta_\ell^{\text{high}}(\text{SA}[i]) = \text{RangeBeg}_{2\ell}(\text{SA}[i]) + \delta_\ell^{\text{high}}(\text{SA}[i])$. Thus, by Lemma 6.47, we have $\text{Pos}_\ell^{\text{high}}(\text{SA}[i]) = \{\text{SA}[j] : j \in (e - d \dots e)\}$. We now observe:

- Let $j_1, j_2 \in (m - d \dots m]$ and assume $j_1 < j_2$. Since the elements of (b_j) occur in the same order as in (a_j) , and positions p in (a_j) are sorted by $T[e^{\text{full}}(p) \dots e^{\text{full}}(p) + 7\tau]$, it follows that $T^\infty[e^{\text{full}}(p_1) \dots e^{\text{full}}(p_1) + 7\tau] \preceq T^\infty[e^{\text{full}}(p_2) \dots e^{\text{full}}(p_2) + 7\tau]$, where $p_1 = b_{j_1}$ and $p_2 = b_{j_2}$. On the other hand, by $e^{\text{full}}(p_1) - x, e^{\text{full}}(p_2) - x \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$, both positions $e^{\text{full}}(p_1) - x$ and $e^{\text{full}}(p_2) - x$ are followed in T by the string $H'H^z$, where H' is a prefix of H of length $\text{head}(\text{SA}[i])$ and $z = \text{exp}^{\text{cut}}(\text{SA}[i], 2\ell)$. Recall, however, that $x = e^{\text{high}}(\text{SA}[i]) - \text{SA}[i] = \text{head}(\text{SA}[i]) + \text{exp}^{\text{cut}}(\text{SA}[i], 2\ell)$. Thus, we obtain $T^\infty[e^{\text{full}}(p_1) - x \dots e^{\text{full}}(p_1)] = T^\infty[e^{\text{full}}(p_2) - x \dots e^{\text{full}}(p_2)] = H'H^z$, and consequently, $T^\infty[e^{\text{full}}(p_1) - x \dots e^{\text{full}}(p_1) + 7\tau] \preceq T^\infty[e^{\text{full}}(p_2) - x \dots e^{\text{full}}(p_2) + 7\tau]$.
- On the other hand, by definition of lexicographical order, for any $j_1, j_2 \in [1..d]$, the assumption $j_1 < j_2$ implies $T^\infty[\text{SA}[e - d + j_1] \dots \text{SA}[e - d + j_1] + x + 7\tau] \preceq T^\infty[\text{SA}[e - d + j_2] \dots \text{SA}[e - d + j_2] + x + 7\tau]$.

We have shown that both sequences $\text{SA}[e-d+1], \dots, \text{SA}[e]$ and $e^{\text{full}}(b_{m-d+1})-x, \dots, e^{\text{full}}(b_m)-x$ contain the same set of positions $\text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ ordered according to the length- $(x+7\tau)$ right context in T^∞ . Therefore, regardless of how ties are resolved in each sequence, for any $j \in [1..d]$, we have $T^\infty[\text{SA}[e-d+j].. \text{SA}[e-d+j]+x+7\tau] = T^\infty[e^{\text{full}}(b_{m-d+j})-x..e^{\text{full}}(b_{m-d+j})+7\tau]$. Thus, by letting $\delta = d-j$, for any $\delta \in [0..d)$ we obtain

$$T^\infty[\text{SA}[e-\delta].. \text{SA}[e-\delta]+x+7\tau] = T^\infty[e^{\text{full}}(b_{m-\delta})-x..e^{\text{full}}(b_{m-\delta})+7\tau].$$

To finalize the proof of the main claim, let $\delta \in [0..d)$ and take any $p \in \text{r-select}_{\mathcal{P}}(x, n, m-\delta)$. By Item (ii) for $j = m-\delta$, we have $T^\infty[p-x..p+7\tau] = T^\infty[e^{\text{full}}(b_{m-\delta})-x..e^{\text{full}}(b_{m-\delta})+7\tau] = T^\infty[\text{SA}[e-\delta].. \text{SA}[e-\delta]+x+7\tau]$. In particular, by $0 \leq x$ and $2\ell \leq 7\tau$, we obtain $2\ell \leq x+7\tau$ and $T^\infty[p-x..p-x+2\ell] = T^\infty[\text{SA}[e-\delta].. \text{SA}[e-\delta]+2\ell]$, i.e., the claim.

2. By Lemma 6.23, it holds $e = \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell(\text{SA}[i]) + \delta_\ell^{\text{high}}(\text{SA}[i]) = \text{RangeBeg}_{2\ell}(\text{SA}[i]) + \delta_\ell^{\text{high}}(\text{SA}[i])$. Thus, by Lemma 6.47, we have $\text{Pos}_\ell^{\text{high}}(\text{SA}[i]) = \{\text{SA}[i] : i \in (e-d..e)\}$. Consequently, $\text{SA}[i] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ implies $i \in (e-d..e)$ and hence $\delta = e-i$ satisfies $\delta \in [0..d)$. By Item 1 with $\delta = e-i$, any $p \in \text{r-select}_{\mathcal{P}}(x, n, m-\delta)$ satisfies $T^\infty[p-x..p-x+2\ell] = T^\infty[\text{SA}[e-\delta].. \text{SA}[e-\delta]+2\ell] = T^\infty[\text{SA}[i].. \text{SA}[i]+2\ell]$, i.e., $p-x \in \text{Occ}_{2\ell}(\text{SA}[i])$. \square

Lemma 6.51. *Assume that $i \in [1..n]$ is such that $\text{SA}[i] \in \mathbb{R}_H^-$ (where $H \in \text{Roots}$) and $\text{SA}[i] \notin \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$. Let $\mathbf{P}_H = \{(e^{\text{full}}(p), e^{\text{full}}(p) - p) : p \in \mathbb{R}'_H\}$, $\mathcal{P} = \text{Point}_{7\tau}(T, \mathbf{P}_H)$, $s = \text{head}(\text{SA}[i])$, $z = \lfloor \frac{2\ell-s}{|H|} \rfloor$, and $x = s + (z+1)|H|$. Then:*

1. *It holds $\text{r-count}_{\mathcal{P}}(x, n) \geq 1$.*
2. *Any $p \in \text{r-select}_{\mathcal{P}}(x, n, 1)$ satisfies $p-x \in \text{Occ}_{2\ell}(\text{SA}[i])$.*

Proof. 1. By definition, if $\text{SA}[i] \notin \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$, then $\text{exp}(\text{SA}[i]) \neq \text{exp}^{\text{cut}}(\text{SA}[i], 2\ell)$, since all other conditions in the definition of $\text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ are satisfied for position $\text{SA}[i]$. However, since $\text{exp}^{\text{cut}}(\text{SA}[i], 2\ell) = \min(\text{exp}(\text{SA}[i]), \lfloor \frac{2\ell-s}{|H|} \rfloor)$, this implies $\lfloor \frac{2\ell-s}{|H|} \rfloor < \text{exp}(\text{SA}[i])$. Consequently, for $k = \text{SA}[i]$ it holds $e^{\text{full}}(k) - k = \text{head}(\text{SA}[i]) + \text{exp}(\text{SA}[i])|H| \geq s + (z+1)|H| = x$, and hence $(p, v) = (e^{\text{full}}(k), e^{\text{full}}(k) - k) \in \mathbf{P}_H$ satisfies $v \in [x..n)$. Thus, by definition of \mathcal{P} , $\text{r-count}_{\mathcal{P}}(x, n) \geq 1$.

2. By $\text{exp}(\text{SA}[i]) \geq z+1$, the string $H'H^{z+1}$ (where H' is a length- s prefix of H) is a prefix of $T[\text{SA}[i]..n]$. But since $|H'H^{z+1}| \geq 2\ell$, we also obtain that $T^\infty[\text{SA}[i].. \text{SA}[i]+2\ell]$ is a prefix of $H'H^{z+1}$. Thus, to find an element of $\text{Occ}_{2\ell}(\text{SA}[i])$, it suffices to find any position $p \in [1..n]$ satisfying $T^\infty[p-x..p] = H'H^{z+1}$ and then by the above argument, it holds $p-x \in \text{Occ}_{2\ell}(\text{SA}[i])$. Let now $p \in \text{r-select}_{\mathcal{P}}(x, n, 1)$. By definition of $\text{r-select}_{\mathcal{P}}(x, n, 1)$, this implies $x \leq v < n$. Since $(p, v) \in \mathbf{P}_H$ holds for some $v \in \mathbb{Z}_{\geq 1}$, there exists $r \in \mathbb{R}'_H$ such that $p = e^{\text{full}}(r)$ and $v = e^{\text{full}}(r) - r$, the position p is preceded in T^∞ by a string $H'H^{z+1}$ (recall that $|H'H^{z+1}| = x$). By the above argument, this implies $p-x \in \text{Occ}_{2\ell}(\text{SA}[i])$. \square

Proposition 6.52. *Let $i \in [1..n]$ be such that $\text{SA}[i] \in \mathbb{R}^-$. Under Assumption 6.21, given the values i , $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$, $\delta_\ell^{\text{low}}(\text{SA}[i])$, $\delta_\ell^{\text{mid}}(\text{SA}[i])$, $\text{exp}(\text{SA}[i])$, and some $j \in \text{Occ}_\ell(\text{SA}[i])$, we can compute some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ in $\mathcal{O}(t)$ time.*

Proof. The main idea is as follows. The query algorithm first tests if $\text{SA}[i] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ holds. By Corollary 6.49 such test can be performed quickly given the values i , $\text{RangeBeg}_\ell(\text{SA}[i])$, $\delta_\ell^{\text{low}}(\text{SA}[i])$, and $\delta_\ell^{\text{mid}}(\text{SA}[i])$, which are all given as input. The rest of the query algorithm depends on this test. If $\text{SA}[i] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$, then $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ is determined according to Lemma 6.50, and it

is computed using the generalized range selection query on the set of points $\mathcal{P} = \text{Points}_{7\tau}(T, P_H)$ (where $H = \text{root}(\text{SA}[i])$), which are supported using the algorithm from Proposition 6.28. On the other hand, if $\text{SA}[i] \notin \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$, then we use Lemma 6.51 to compute some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$. This again reduces to a generalized range selection query on \mathcal{P} , but with different input parameters.

Given any index $i \in [1..n]$ such that $\text{SA}[i] \in \mathbb{R}^-$, along with values $\text{RangeBeg}_\ell(\text{SA}[i])$, $k = \exp(\text{SA}[i])$, $\delta_\ell^{\text{low}}(\text{SA}[i])$, $\delta_\ell^{\text{mid}}(\text{SA}[i])$, and some $j \in \text{Occ}_\ell(\text{SA}[i])$, we compute $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ as follows. First, using Assumption 6.21 we compute values $s = \text{head}(j) = \text{head}(\text{SA}[i])$ and $p = |\text{root}(j)| = |\text{root}(\text{SA}[i])|$ in $\mathcal{O}(t)$ time. Note, that then the position $r = j + s$ satisfies $\text{root}(\text{SA}[i]) = T[r..r + p]$, i.e., we have a starting position of an occurrence of $H := \text{root}(j) = \text{root}(\text{SA}[i])$ in T . Denote $\mathcal{P} = \text{Points}_{7\tau}(T, E_H^-)$ (see Definition 4.4). In $\mathcal{O}(1)$ time we calculate $k_2 := \exp^{\text{cut}}(\text{SA}[i], 2\ell) = \min(k, \lfloor \frac{2\ell-s}{p} \rfloor)$. Next, in $\mathcal{O}(1)$ time we determine if $\text{SA}[i] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$. By Corollary 6.49, $\text{SA}[i] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$ holds if and only if $i \leq \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i]) + \delta_\ell^{\text{mid}}(\text{SA}[i])$. Consider two cases:

- Assume $i \leq \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i]) + \delta_\ell^{\text{mid}}(\text{SA}[i])$ (i.e., $\text{SA}[i] \in \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$). First, we compute $x = e^{\text{high}}(\text{SA}[i]) - \text{SA}[i] = \text{head}(\text{SA}[i]) + \exp^{\text{cut}}(\text{SA}[i], 2\ell)|\text{root}(\text{SA}[i])| = s + k_2p$, $e = \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i]) + \delta_\ell^{\text{mid}}(\text{SA}[i])$, and $\delta = e - i$ in $\mathcal{O}(1)$ time. Then, using the query defined by Item 1 of Problem 4.5 we compute the value $m = \text{r-count}_{\mathcal{P}}(x, n)$. By Assumption 6.21, this takes $\mathcal{O}(t)$ time. Next, using the query defined by Item 2 of Problem 4.5 we compute some $p \in \text{r-select}_{\mathcal{P}}(x, n, m - \delta)$. This again takes $\mathcal{O}(t)$ time. Finally, we calculate $j' = p - x$. By Item 2 from Lemma 6.50, it holds $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$.
- Assume now $i > \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell^{\text{low}}(\text{SA}[i]) + \delta_\ell^{\text{mid}}(\text{SA}[i])$ (i.e., $\text{SA}[i] \notin \text{Pos}_\ell^{\text{high}}(\text{SA}[i])$). First, calculate $z = \lfloor \frac{2\ell-s}{p} \rfloor$ and $x = s + (z + 1)p$. Next, using Item 2 of Problem 4.5 we compute some $p \in \text{r-select}_{\mathcal{P}}(x, n, 1)$. By Assumption 6.21, this takes $\mathcal{O}(t)$ time. Finally, we calculate $j' = p - x$. By Item 2 from Lemma 6.51, it holds $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$. \square

6.3.9 The Data Structure

By combining the above results, under Assumption 6.21, we obtain the following efficient algorithm to “refine” $(\text{RangeBeg}_\ell(\text{SA}[i]), \text{RangeEnd}_\ell(\text{SA}[i]))$ into $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$ and to simultaneously compute some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$.

Proposition 6.53. *Let $i \in [1..n]$ be such that $\text{SA}[i] \in \mathbb{R}$. Under Assumption 6.21, given the values i , $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$, and some $j \in \text{Occ}_\ell(\text{SA}[i])$, we can compute $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$ and some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ in $\mathcal{O}(t)$ time.*

Proof. The algorithm consists of seven steps:

1. First, using Proposition 6.35, we compute in $\mathcal{O}(t)$ time the value of $\text{type}(\text{SA}[i])$. The query algorithm needs the index i along with $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$, and some $j \in \text{Occ}_\ell(\text{SA}[i])$, which at this point we have as input. Let us assume $\text{type}(\text{SA}[i]) = -1$ (the case $\text{type}(\text{SA}[i]) = +1$ is explained at the end of the proof).
2. Next, using Proposition 6.38, we compute in $\mathcal{O}(t)$ time the value of $\delta_\ell^{\text{low}}(\text{SA}[i])$. The algorithm only needs some $j \in \text{Occ}_\ell(\text{SA}[i])$, which we are given as input.
3. In the third step, using Proposition 6.44, we compute in $\mathcal{O}(t)$ time the value of $\exp(\text{SA}[i])$. The algorithm needs the values of i , $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$, some $j \in \text{Occ}_\ell(\text{SA}[i])$, and $\delta_\ell^{\text{low}}(\text{SA}[i])$. The first four values are given as input, and the fifth value was computed in Step 2.

4. Next, using Proposition 6.46, we compute in $\mathcal{O}(t)$ time the value $\delta_\ell^{\text{mid}}(\text{SA}[i])$. The algorithm needs some position $j \in \text{Occ}_\ell(\text{SA}[i])$ and the value $\text{exp}(\text{SA}[i])$ as input. The first argument is given as input, and the second value we computed in Step 3.
5. Next, using Proposition 6.52, we compute in $\mathcal{O}(t)$ time some position $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$. The algorithm needs values i , $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$, some position $j \in \text{Occ}_\ell(\text{SA}[i])$, $\delta_\ell^{\text{low}}(\text{SA}[i])$, $\text{exp}(\text{SA}[i])$, and $\delta_\ell^{\text{mid}}(\text{SA}[i])$ as input. The first four arguments are given as input. The remaining three we computed in Step 2, Step 3, and Step 4.
6. Next, using Proposition 6.38, we compute in $\mathcal{O}(t)$ time the value of $\delta_\ell^{\text{high}}(\text{SA}[i])$. The algorithm only needs some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$, which we computed in Step 5.
7. Finally, using Proposition 6.32 and its symmetric version adapted according to Lemma 6.18 (see also below), we compute in $\mathcal{O}(t)$ time the values $m^- := |\text{Occ}_{2\ell}^-(j')|$ and $m^+ := |\text{Occ}_{2\ell}^+(j')|$, where $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$. Note that by definition of $\text{Occ}_{2\ell}(\text{SA}[i])$ and $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$, we have $|\text{Occ}_{2\ell}^-(\text{SA}[i])| = |\text{Occ}_{2\ell}^-(j')|$ and $|\text{Occ}_{2\ell}^+(\text{SA}[i])| = |\text{Occ}_{2\ell}^+(j')|$. Observe now that by Item 1 of Lemma 6.24, for any $p \in \text{Occ}_{2\ell}(\text{SA}[i])$, it holds $p \in \text{R}$. Thus, $\text{Occ}_{2\ell}(\text{SA}[i])$ is a disjoint union of $\text{Occ}_{2\ell}^-(\text{SA}[i])$ and $\text{Occ}_{2\ell}^+(\text{SA}[i])$, and hence denoting $m = |\text{Occ}_{2\ell}(\text{SA}[i])|$, we have $m = m^- + m^+$. To compute m^- and m^+ , the query algorithms in Proposition 6.32 and its symmetric counterpart only require some position $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ as input. We obtained such position in Step 5.

After executing the above steps, we already have $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ (computed in Step 5) Thus, it remains to calculate the pair $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$. For this, in $\mathcal{O}(1)$ time we first obtain $\delta_\ell(\text{SA}[i]) = \delta_\ell^{\text{low}}(\text{SA}[i]) + \delta_\ell^{\text{mid}}(\text{SA}[i]) - \delta_\ell^{\text{high}}(\text{SA}[i])$ by Lemma 6.23. We then obtain the output as $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i])) = (\text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell(\text{SA}[i]), \text{RangeBeg}_\ell(\text{SA}[i]) + \delta_\ell(\text{SA}[i]) + m)$. In total, the query takes $\mathcal{O}(t)$ time.

If in Step 1, we have $\text{type}(\text{SA}[i]) = +1$, then in Steps 2–6 instead of queries on $\text{Points}_{7\tau}(T, E_H^-)$ and \mathcal{I}_H^- in Assumption 6.21, we perform the queries on $\text{Points}_{7\tau}(T, E_H^+)$ and \mathcal{I}_H^+ . By Lemma 6.18, all remaining details are symmetrical. In particular, $\delta'_\ell(\text{SA}[i]) = \delta_\ell^{\text{low}}(\text{SA}[i]) + \delta_\ell^{\text{mid}}(\text{SA}[i]) - \delta_\ell^{\text{high}}(\text{SA}[i])$ and $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i])) = (\text{RangeEnd}_\ell(\text{SA}[i]) - \delta'_\ell(\text{SA}[i]) - m, \text{RangeEnd}_\ell(\text{SA}[i]) - \delta'_\ell(\text{SA}[i]))$. Definitions of sets $\text{Pos}_\ell^{\text{low}}(j)$, $\text{Pos}_\ell^{\text{mid}}(j)$, and $\text{Pos}_\ell^{\text{high}}(j)$ are adapted for $j \in \text{R}^+$ as follows:

$$\begin{aligned} \text{Pos}_\ell^{\text{low}}(j) &= \{j' \in \text{R}_{s,H}^+ : \text{exp}(j') = k_1 \text{ and } (T[j'..n] \preceq T[j..n] \text{ or } \text{LCE}_T(j, j') \geq \ell)\}, \\ \text{Pos}_\ell^{\text{mid}}(j) &= \{j' \in \text{R}_{s,H}^+ : \text{exp}(j') \in (k_1..k_2)\}, \text{ and} \\ \text{Pos}_\ell^{\text{high}}(j) &= \{j' \in \text{R}_{s,H}^+ : \text{exp}(j') = k_2 \text{ and } (T[j'..n] \preceq T[j..n] \text{ or } \text{LCE}_T(j, j') \geq 2\ell)\}. \end{aligned}$$

By the symmetric version of Lemma 6.23, for $j \in \text{R}^+$, $\delta'_\ell(j) = \delta_\ell^{\text{low}}(j) + \delta_\ell^{\text{mid}}(j) - \delta_\ell^{\text{high}}(j)$. \square

6.4 The Final Data Structure

We are ready to present the general SA query algorithm (returning $\text{SA}[i]$ given any $i \in [1..n]$).

The section is organized as follows. First, we show how to combine Section 6.1, Section 6.2, and Section 6.3 to obtain a query algorithm that for any $\ell \in [16..n]$, given any index i along with the values $\text{RangeBeg}_\ell(\text{SA}[i])$, $\text{RangeEnd}_\ell(\text{SA}[i])$, and some $j \in \text{Occ}_\ell(\text{SA}[i])$ as input, computes the pair $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$ and some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ (Proposition 6.54). We then combine this result with the query algorithm that given $i \in [1..n]$ computes the pair $(\text{RangeBeg}_{16}(\text{SA}[i]), \text{RangeEnd}_{16}(\text{SA}[i]))$ and some $j' \in \text{Occ}_{16}(\text{SA}[i])$ (Assumption 6.1) to obtain the final algorithm for SA queries (Proposition 6.55).

Proposition 6.54. For any $\ell \in [16..n]$, under Assumptions 6.3, 6.5, and 6.21, given an index $i \in [1..n]$ along with the pair $(\text{RangeBeg}_\ell(\text{SA}[i]), \text{RangeEnd}_\ell(\text{SA}[i]))$ and some position $j \in \text{Occ}_\ell(\text{SA}[i])$ as input, we can compute the pair $(\text{RangeBeg}_{2\ell}(\text{SA}[i]), \text{RangeEnd}_{2\ell}(\text{SA}[i]))$ and some $j' \in \text{Occ}_{2\ell}(\text{SA}[i])$ in $\mathcal{O}(t)$ time.

Proof. First, using Assumption 6.3 we check in $\mathcal{O}(t)$ if $\text{SA}[i] \in \mathbf{R}$. If $\text{SA}[i] \notin \mathbf{R}$, then we compute the output in $\mathcal{O}(t)$ time using Proposition 6.16. Otherwise, we compute the output in $\mathcal{O}(t)$ time using Proposition 6.53. \square

Proposition 6.55. Under Assumption 6.1 and Assumptions 6.3, 6.5, and 6.21 for $\ell = 2^q$, where $q \in [4.. \lceil \log n \rceil]$, given any $i \in [1..n]$, we can compute $\text{SA}[i]$ in $\mathcal{O}(t \log n)$ time.

Proof. Given any position $i \in [1..n]$, we compute $\text{SA}[i]$ as follows. First, using Assumption 6.1, we compute the pair $(\text{RangeBeg}_{16}(\text{SA}[i]), \text{RangeEnd}_{16}(\text{SA}[i]))$ and some $j \in \text{Occ}_{16}(\text{SA}[i])$ in $\mathcal{O}(t)$ time. Then, for $q = 4, \dots, \lceil \log n \rceil - 1$, we use Proposition 6.54 with $\ell = 2^q$ to compute in $\mathcal{O}(t)$ time the pair $(\text{RangeBeg}_{2^{q+1}}(\text{SA}[i]), \text{RangeEnd}_{2^{q+1}}(\text{SA}[i]))$ and some $j' \in \text{Occ}_{2^{q+1}}(\text{SA}[i])$, given position i along with the pair $(\text{RangeBeg}_{2^q}(\text{SA}[i]), \text{RangeEnd}_{2^q}(\text{SA}[i]))$ and some $j \in \text{Occ}_{2^q}(\text{SA}[i])$ as input. After executing all steps, we have $(\text{RangeBeg}_\ell(\text{SA}[i]), \text{RangeEnd}_\ell(\text{SA}[i]))$ and some $j' \in \text{Occ}_\ell(\text{SA}[i])$, where $\ell = 2^{\lceil \log n \rceil} \geq n$. Since for any $k \geq n$, we have $\text{Occ}_k(\text{SA}[i]) = \{\text{SA}[i]\}$, we finally return $\text{SA}[i] = j'$. In total, the query takes $\mathcal{O}(t \log n)$ time. \square

7 Balanced Signature Parsing

7.1 Symbols and Grammars

For an alphabet Σ , we define the set (algebraic data type) \mathcal{S} of *symbols* over Σ as the least fixed point of the following equation, where **letter** and **block** are two named constructors:

$$\mathcal{S} = \{\text{letter}(a) : a \in \Sigma\} \cup \{\text{block}(S) : S \in \mathcal{S}^+\},$$

We define a recursive *expansion* function $\text{str} : \mathcal{S} \rightarrow \Sigma^+$ with

$$\text{str}(X) = \begin{cases} a & \text{if } X = \text{letter}(a) \text{ for } a \in \Sigma, \\ \bigodot_{i=1}^{|S|} \text{str}(S[i]) & \text{if } X = \text{block}(S) \text{ for } S \in \mathcal{S}^+, \end{cases}$$

and we lift it to $\text{str} : \mathcal{S}^* \rightarrow \Sigma^*$ with $\text{str}(S) = \bigodot_{i=1}^{|S|} \text{str}(S[i])$. We further define a recursive *level* function $\text{level} : \mathcal{S} \rightarrow \mathbb{Z}_{\geq 0}$ with

$$\text{level}(X) = \begin{cases} 0 & \text{if } X = \text{letter}(a) \text{ for } a \in \Sigma, \\ \max_{i=1}^{|S|} \text{level}(S[i]) & \text{if } X = \text{block}(S) \text{ for } S \in \mathcal{S}^+. \end{cases}$$

We say that a set $\mathcal{G} \subseteq \mathcal{S}$ is a *grammar* if $\mathcal{G} \subseteq \{\text{letter}(a) : a \in \Sigma\} \cup \{\text{block}(S) : S \in \mathcal{G}^+\}$. Our algorithms maintain grammars using the following interface.

Lemma 7.1 (Grammar Representation). For every constant $C \in \mathbb{Z}_{\geq 0}$, there exists a data structure that maintains a grammar $\mathcal{G} \subseteq \mathcal{S}$ subject to the following operations, where each symbol $X \in \mathcal{G}$ is represented by a unique identifier $\text{id}(X) \in [0..g]$, where $g = |\mathcal{G}|$:

$\text{deg}(X)$: Given $X \in \mathcal{G}$, return 0 if $X = \text{letter}(a)$ and $|S|$ if $X = \text{block}(S)$.
 $\text{letter}^{-1}(X)$: Given $X = \text{letter}(a)$, return a .
 $\text{block}^{-1}(X, i)$: Given $X = \text{block}(S)$ and $i \in [1..|S|]$, return $S[i]$.
 $\text{level}(X)$: Given $X \in \mathcal{G}$, return $\text{level}(X)$.
 $\text{len}(X)$: Given $X \in \mathcal{G}$, return $|\text{str}(X)|$.
 $\text{pLen}(X, i)$: Given $X = \text{block}(S)$ and $i \in [0..|S|]$, return $|\text{str}(S[1..i])|$.
 $\text{insertLetter}(a)$: Given $a \in \Sigma$, set $\mathcal{G} := \mathcal{G} \cup \{\text{letter}(a)\}$ and return $\text{letter}(a)$.
 $\text{insertString}(S)$: Given $S \in \mathcal{G}^+$ with $|S| \leq C$, set $\mathcal{G} := \mathcal{G} \cup \{\text{block}(S)\}$ and return $\text{block}(S)$.
 $\text{insertPower}(Y, m)$: Given $Y \in \mathcal{G}$ and $m \in \mathbb{Z}_+$, set $\mathcal{G} := \mathcal{G} \cup \{\text{block}(Y^m)\}$ and return $\text{block}(Y^m)$.

In the $\Omega(\log(\sigma + \sum_{X \in \mathcal{G}} \text{len}(X)))$ -bit word RAM model, the data structure size is $\mathcal{O}(g)$, insertions cost $\mathcal{O}(\frac{\log^2 \log g}{\log \log \log g})$ time, and queries cost $\mathcal{O}(1)$ time.

Proof. Internally, the symbols are stored in array $A[0..g]$ with the following entries $A[\text{id}(X)]$ for $X \in \mathcal{G}$:

- if $X = \text{letter}(a)$, then $A[\text{id}(X)] = a$;
- if $X = \text{block}(Y^m)$ for $Y \in \mathcal{G}$ and $m \in \mathbb{Z}_+$, then $A[\text{id}(X)] = (\text{id}(Y), m)$;
- otherwise, $X = \text{block}(S)$ for $S \in \mathcal{G}^m$ with $m \in [1..C]$, and $A[\text{id}(X)] = \text{id}(S[1]) \cdots \text{id}(S[m])$.

Additionally, we maintain the inverse mapping A^{-1} using a dynamic deterministic dictionary [FG15], as well as the precomputed values $\text{level}(X)$ and $\text{len}(X)$ for all symbols $X \in \mathcal{G}$. The array A and the precomputed values allow for an easy implementation of all queries in $\mathcal{O}(1)$ time. As for insertions, we first build the internal representation of the inserted symbol X (note that $\text{insertString}(S)$ needs to check if $S = Y^m$ for some $Y \in \mathcal{S}$ and $m \in \mathbb{Z}_+$). Then, we perform a dictionary lookup for X in A^{-1} . If the lookup is successful, we retrieve $\text{id}(X)$ and return this identifier as the reference to X . Otherwise, we set $\text{id}(X) := g$, increment g , insert X to \mathcal{G} (by adding the corresponding entries to A and A^{-1}), and compute $\text{level}(X)$ as well as $\text{len}(X)$. The running time is dominated by the cost $\mathcal{O}(\frac{\log^2 \log g}{\log \log \log g})$ of a dictionary lookup and insertion. \square

7.2 Parse Trees

Every symbol $X \in \mathcal{S}$ can be interpreted as a rooted ordered *parse tree* $\mathcal{T}(X)$ with each node ν associated to a symbol $\text{symb}(\nu) \in \mathcal{S}$. The root of $\mathcal{T}(X)$ is a node ρ with $\text{symb}(\rho) = X$. If $X = \text{letter}(a)$ for $a \in \Sigma$, then ρ has no children. Otherwise, $X = \text{block}(S)$ for $S \in \mathcal{S}^m$ and $m \in \mathbb{Z}_+$; then, ρ has m children, and the subtree rooted at the i th child is (a copy of) $\mathcal{T}(S[i])$.

Each node ν of $\mathcal{T}(X)$ is associated with a fragment $\text{str}(\nu)$ of $\text{str}(X)$ matching $\text{str}(\text{symb}(\nu))$. For the root ν , we define $\text{str}(\rho) = \text{str}(X)[1..|\text{str}(X)|]$ to be the whole $\text{str}(X)$. Moreover, if ν_1, \dots, ν_m are the children of a node ν , then $\text{str}(\nu_i) = \text{str}(\nu)(r_{i-1}..r_i]$, where $r_i = \text{pLen}(\text{symb}(\nu), i)$; this way, $\text{str}(\nu)$ is the concatenation of fragments $\text{str}(\nu_i)$.

Lemma 7.2. *Based on the interface of \mathcal{G} provided in Lemma 7.1, for every $X \in \mathcal{G}$, one can implement references to nodes ν of $\mathcal{T}(X)$ so that the following operations cost $\mathcal{O}(1)$ time:*

$\text{root}(X)$: Given $X \in \mathcal{G}$, return the root of $\mathcal{T}(X)$.
 $\text{symb}(\nu)$: Given a node ν of $\mathcal{T}(X)$, return the symbol $\text{symb}(\nu)$.
 $\text{str}(\nu)$: Given a node ν of $\mathcal{T}(X)$, return the fragment $\text{str}(\nu)$.
 $\text{child}(\nu, i)$: Given a node ν of $\mathcal{T}(X)$ and $i \in [1..|\text{deg}(\text{symb}(\nu))|]$, return the i th child of ν .
 $\text{parent}(\nu)$: Given a node ν of $\mathcal{T}(X)$, return \perp if $\nu = \text{root}(X)$, and otherwise the parent of ν .

$\text{index}(\nu)$: Given a node ν of $\mathcal{T}(X)$, return \perp if $\nu = \text{root}(X)$, and otherwise the value $i \in \mathbb{Z}_+$ such that $\nu = \text{child}(\text{parent}(\nu), i)$.

Proof. Internally, each node ν is represented as a tuple $(\text{symb}(\nu), \text{str}(\nu), \text{index}(\nu), \text{parent}(\nu))$. Here, $\text{parent}(\nu)$ points to the parent of ν stored in the same representation. In other words, the internal of ν is a singly linked list describing to the path from ν to the root of its parse tree. This is a *functional* list, which means that many multiple references can share parts of the underlying lists. The non-trivial operations are $\text{root}(X)$, which returns $(X, \text{str}(X)[1..|\text{str}(X)|], \perp, \perp)$, and $\text{child}(\nu, i)$, which returns $(\text{block}^{-1}(\text{symb}(\nu), i), \text{str}(\nu)(\text{pLen}(\text{symb}(\nu), i - 1) .. \text{pLen}(\text{symb}(\nu), i)], i, \nu)$. \square

7.3 Deterministic Coin Tossing

For an integer $h \in \mathbb{Z}_{\geq 0}$, let us denote

$$\Sigma_h := \left[0 .. \underbrace{2^{2^{\cdot^{\cdot^2}}}}_h \right) \quad \text{and} \quad \bar{\Sigma}_h := \Sigma_h \cup \{\perp\}.$$

Theorem 7.3. For every integer $h \in \mathbb{Z}_+$, there is a function $g_h : \bar{\Sigma}_h^{h+11} \rightarrow \{0, 1\}$ such that the function $G_h : \bar{\Sigma}_h^{\mathbb{Z}} \rightarrow \{0, 1\}^{\mathbb{Z}}$, defined with $G_h(T)[i] = g_h(T(i - h - 7 .. i + 4))$ for $i \in \mathbb{Z}$, satisfies the following properties for all integers $n \in \mathbb{Z}_{\geq 0}$ and strings $T \in \bar{\Sigma}_h^{\mathbb{Z}}$ with $T[1..n] \in \Sigma_h^n$, $T[0] = T[n+1] = \perp$, and $T[j] \neq T[j+1]$ for all $j \in [1..n]$:

- (1) $G_h(T)[0] = G_h(T)[n] = 1$.
- (2) $G_h(T)[0..n]$ does not contain **11** and **000000** as proper substrings.
- (3) $G_h(T)[0..n]$ would not be affected by setting $T[j] := \perp$ for all $j \in \mathbb{Z} \setminus [1..n]$.

Moreover, given h and $T[1..n]$, the string $G_h(T)[0..n]$ can be computed in $\mathcal{O}(nh)$ time provided that each character of $T[1..n]$ is an $\mathcal{O}(w)$ -bit number.

Proof. The starting point of our construction is the function $f_h : \bar{\Sigma}_h^{h+11} \rightarrow \{0, 1\}$ behind [MSU97, Lemma 1], for which the derived $F_h : \bar{\Sigma}_h^{\mathbb{Z}} \rightarrow \{0, 1\}^{\mathbb{Z}}$ (defined analogously to G_h) satisfies the following property: For all integers $n \in \mathbb{Z}_{\geq 0}$ and strings $T \in \bar{\Sigma}_h^{\mathbb{Z}}$ with $T[1..n] \in \Sigma_h^n$, $T[j] = \perp$ for $j \in \mathbb{Z} \setminus [1..n]$, and $T[j] \neq T[j+1]$ for $j \in [1..n]$, the string $F_h(T)[1..n]$ does not contain **11** and **0000** as substrings.

For a string $X \in \bar{\Sigma}_h^{h+11}$, let $X(\ell..r)$ be the maximal fragment such that $\ell \leq h+7 \leq r$ and $X(\ell..r) \in \Sigma_h^*$. We set:

- (a) $g_h(X) = 1$ if $\ell = h+7$ or $r = h+7$;
- (b) $g_h(X) = 0$ if $\ell = h+6$ (and $r > h+7$) or $r = h+8$ (and $\ell < h+7$);
- (c) $g_h(X) = f_h(\perp^\ell X(\ell..r) \perp^{h+11-r})$ otherwise.

Let us fix a string $T \in \bar{\Sigma}_h^{\mathbb{Z}}$ with $T[1..n] \in \Sigma_h^n$, $T[0] = T[n+1] = \perp$, and $T[j] \neq T[j+1]$ for all $j \in [1..n]$. Define a string $\bar{T} \in \bar{\Sigma}_h^{\mathbb{Z}}$ so that $\bar{T}[j] = T[j]$ for $j \in [1..n]$ and $\bar{T}[j] = \perp$ otherwise.

Observe that $G_h(T)[0] = G_h(T)[n] = 1$ by (a). Moreover, if $n \geq 2$, then $G_h(T)[1] = G_h(T)[n-1] = 0$ by (b), and, if $n \geq 4$, then $G_h(T)[2..n-2] = F_h(\bar{T})[2..n-2]$ by (c). This immediately yields conditions (1) and (3).

As for condition (2), note that any occurrence **11** as a proper substring of $G_h(T)[0..n]$ would yield an occurrence of **11** as a substring $F_h(\bar{T})[2..n-2]$, and any occurrence of **000000** as a substring of $G_h(T)[0..n]$ would yield an occurrence of **0000** as a substring of $F_h(\bar{T})[2..n-2]$.

An efficient algorithm for computing $G_h(T)[0..n]$ follows from the corresponding algorithm for computing $F_h(\bar{T})[0..n]$. \square

Definition 7.4 (Restricted signature parsing). Consider a set $\mathcal{A} \subseteq \mathcal{S}$ of *active symbols*, an integer $h \in \mathbb{Z}_{>0}$, and an injective function $\text{sig} : \mathcal{A} \rightarrow \Sigma_h$. Given a string $T \in \mathcal{S}^*$, define a string $\text{sig}(T) \in \bar{\Sigma}_h^{\mathbb{Z}}$ so that $\text{sig}(T)[i] = \text{sig}(T[i])$ if $i \in [1..|T|]$ and $T[i] \in \mathcal{A}$, and $\text{sig}(T)[i] = \perp$ otherwise. The *restricted signature parsing* decomposes T into *blocks* that end at all positions $i \in [1..|T|]$ with $G_h(\text{sig}(T))[i] = 1$, where G_h is defined in Theorem 7.3.

Corollary 7.5. Consider the restricted signature parsing of a string $T \in \mathcal{S}^*$ such that no position $i \in [1..|T|]$ satisfies $T[i] = T[i+1] \in \mathcal{A}$. Then:

- (1) each block contains exactly one inactive symbol or up to six active symbols;
- (2) an active symbol forms a length-1 block if and only if it has no adjacent active symbols.

Proof. Let $T(\ell..r]$ be any maximal fragment of T consisting of active symbols. By Theorem 7.3, we have $G_h(\text{sig}(T))[\ell] = G_h(\text{sig}(T))[r] = 1$, so $T(\ell..r]$ consists of full blocks. Since $G_h(\text{sig}(T))(\ell..r]$ does not contain 000000, these blocks are of length at most six. Moreover, since $G_h(\text{sig}(T))[\ell..r]$ does not contain 11 as a proper substring, the only possibility for a length-1 block within $T(\ell..r]$ is when $r - \ell = 1$, i.e., when $T[r]$ is an active symbol with no adjacent active symbols. In the reasoning above, $G_h(\text{sig}(T))[\ell] = G_h(\text{sig}(T))[r] = 1$ also holds when $\ell = r$, so all inactive symbols form length-1 blocks. \square

Corollary 7.6. Consider the restricted signature parsing of strings $T, T' \in \mathcal{S}^*$. If a block ends at $T[i]$ but no block ends at $T'[i']$, then there exists:

- a string in $\mathcal{A}^{\leq 4}$ that is a prefix of exactly one of the suffixes $T(i..|T|]$ and $T'(i'..|T'|]$, or
- a string in $\mathcal{A}^{\leq h+7}$ that is a suffix of exactly one of the prefixes $T[1..i]$ and $T'[1..i']$.

Proof. Let $T(\ell..r]$ and $T'(\ell'..r']$ be maximal fragments with $i - h - 7 \leq \ell \leq i \leq r \leq i + 4$ and $i' - h - 7 \leq \ell' \leq i' \leq r' \leq i' + 4$, respectively, such that $T(\ell..r], T'(\ell'..r'] \in \mathcal{A}^*$. If $T(\ell..i] \neq T'(\ell'..i']$, this yields a desired suffix of exactly one of the prefixes $T[1..i]$ and $T'[1..i']$. Similarly, if $T(i..r] \neq T'(i'..r']$, this yields a desired prefix of exactly one of the suffixes $T(i..|T|]$ and $T'(i'..|T'|]$. By Theorem 7.3(3), we have $G_h(\text{sig}(T))[i] = g_h(\perp^{\ell-i+h+7} \text{sig}(T)(\ell..r] \perp^{i+4-r})$ and $G_h(\text{sig}(T'))[i'] = g_h(\perp^{\ell'-i'+h+7} \text{sig}(T')(\ell'..r'] \perp^{i'+4-r'})$. Due to the assumption that $T(\ell..i] = T'(\ell'..i']$ and $T(i..r] = T'(i'..r']$, this yields $G_h(\text{sig}(T))[i] = G_h(\text{sig}(T'))[i']$. Hence, blocks end at both $T[i]$ and $T'[i']$ or at neither $T[i]$ nor $T'[i']$. \square

7.4 Balanced Signature Parsing

Definition 7.7 (Restricted run parsing). Consider a set $\mathcal{A} \subseteq \mathcal{S}$ of *active symbols*. The *restricted run parsing* decomposes a string $T \in \mathcal{S}^*$ into *blocks* that end at all positions $i \in [1..|T|]$ except those with $T[i] = T[i+1] \in \mathcal{A}$.

For every $k \in \mathbb{Z}_{>0}$, we define the set of level- k *active symbols* $\mathcal{A}_k := \{X \in \mathcal{S} : \text{level}(X) = k \text{ and } \text{len}(X) \leq d_k\}$, where $d_k := 2^{\lfloor k/2 \rfloor}$. Moreover, we say that a function $\text{sig} : \mathcal{S} \rightarrow \mathbb{Z}_{\geq 0}$ is a *signature function* if $\text{sig}_k := \text{sig}|_{\mathcal{A}_k}$ injectively maps \mathcal{A}_k to Σ_{h_k} , where $h_k = \log^* |\mathcal{A}_k|$.

Fact 7.8. Every $k \in \mathbb{Z}_{>0}$ satisfies $h_k = \mathcal{O}(\log^*(\sigma k))$.

Proof. For $k, \ell \in \mathbb{Z}_{\geq 0}$, let $s_{k,\ell} = |\{X \in \mathcal{S} : \text{level}(X) \leq k \text{ and } \text{len}(X) \leq \ell\}|$. We shall inductively prove that $s_{k,\ell} \leq \sigma^\ell (k+1)^{\ell-1}$. The base case of $k=0$ is easy to check since $s_{0,0} = 0$ and $s_{0,\ell} = \sigma$ for $\ell \in \mathbb{Z}_+$. As for $k \in \mathbb{Z}_+$, we have

$$\begin{aligned} s_{k,\ell} &\leq s_{k-1,\ell} + \sum_{j=1}^{\ell-1} s_{k-1,j} \cdot s_{k,\ell-j} \leq \sigma^\ell k^{\ell-1} + \sum_{j=1}^{\ell-1} \sigma^j k^{j-1} \sigma^{\ell-j} (k+1)^{\ell-j-1} \\ &= \sigma^\ell (k+1)^{\ell-2} \left(\frac{k^{\ell-1}}{(k+1)^{\ell-2}} + \sum_{i=0}^{\ell-2} \frac{k^i}{(k+1)^i} \right) \leq \sigma^\ell (k+1)^{\ell-2} \left(\frac{k^{\ell-1}}{(k+1)^{\ell-2}} + \frac{1 - \frac{k^{\ell-1}}{(k+1)^{\ell-1}}}{1 - \frac{k}{k+1}} \right) = \sigma^\ell (k+1)^{\ell-1}. \end{aligned}$$

In particular, we conclude that $|\mathcal{A}_k| \leq \sigma^{2^{k/2}} (k+1)^{2^{k/2}-1} \leq 2^{2^{\sigma k}}$, so $h_k \leq 2 + \log^*(\sigma k)$. \square

Construction 7.9 (Balanced signature parsing). *For a signature function sig , the balanced signature parsing of a string $T \in \Sigma^+$ is a sequence $(T_k)_{k=0}^\infty$ of strings $T_k \in \mathcal{S}^+$ constructed as follows: The string T_0 is obtained from T by replacing each letter $T[j]$ with symbol $\text{letter}(T[j])$. For $k > 0$, we perform the restricted signature parsing of T_{k-1} with respect to \mathcal{A}_{k-1} , h_{k-1} , and sig_{k-1} (if k is even) or the restricted run parsing of T_{k-1} with respect to \mathcal{A}_{k-1} (if k is odd) and derive T_k by collapsing each block $T_{k-1}[j..j+m)$ into the corresponding symbol $\text{block}(T_{k-1}[j..j+m))$.*

Observe that $\text{str}(T_k) = T$ for $k \in \mathbb{Z}_{\geq 0}$. Hence, for every $j \in [1..|T_k|]$, we can associate $T_k[j]$ with a fragment $T(|\text{str}(T_k[1..j])|..|\text{str}(T_k[1..j])|)$ matching $\text{str}(T_k[j])$; these fragments are called *phrases* induced by T_k . We also define a set $B_k(T)$ of *phrase boundaries* induced by T_k :

$$B_k(T) = \{|\text{str}(T_k[1..j])| : j \in [0..|T_k|]\}.$$

The following result lets us use Corollary 7.5 when analyzing the restricted signature parsing of T_k for odd $k \in \mathbb{Z}_+$.

Fact 7.10. *For every $T \in \Sigma^+$ and odd $k \in \mathbb{Z}_+$, there is no $j \in [1..|T_k|)$ with $T_k[j] = T_k[j+1] \in \mathcal{A}_k$.*

Proof. For a proof by contradiction, suppose that $T_k[j] = T_k[j+1] \in \mathcal{A}_k$ holds for some $j \in [1..|T_k|)$. By definition of T_k , we have $T_k[j] = T_k[j+1] = \text{block}(X^\ell)$ for some $X \in \mathcal{S}$ and $\ell \in \mathbb{Z}_+$. In particular, $|\text{str}(X)| \leq |\text{str}(X^\ell)| \leq d_k = d_{k-1}$, so $X \in \mathcal{A}_{k-1}$. Let $T_{k-1}(i-\ell..i)$ and $T_{k-1}(i..i+\ell)$ be blocks of T_{k-1} collapsed to $T_k[j]$ and $T_k[j+1]$, respectively. Due to $T_{k-1}[i] = T_{k-1}[i+1] = X \in \mathcal{A}_{k-1}$, no block ends at position i in the restricted run parsing of T_{k-1} . This contradicts the existence of the block $T_{k-1}(i-\ell..i)$. \square

Next, we use Corollary 7.5 to derive upper and lower bounds on phrase lengths.

Fact 7.11. *For every $T \in \Sigma^+$, $k \in \mathbb{Z}_{\geq 0}$, and $j \in [1..|T_k|]$, the string $\text{str}(T_k[j])$ has length at most $3d_k$ or primitive root of length at most d_k .*

Proof. We proceed by induction on k . If $k=0$, then $|\text{str}(T_k[j])| = 1 < 3$. Thus, we may assume $k > 0$. Let $T_{k-1}[i..i+\ell)$ be the fragment of T_{k-1} obtained by expanding $T_k[j]$. If $\ell=1$, then the inductive assumption shows that $\text{str}(T_k[j])$ is of length at most $3d_{k-1} \leq 3d_k$ or its primitive root is of length at most $d_{k-1} \leq d_k$. If $\ell \geq 2$ and k is odd, then $T_{k-1}[i..i+\ell) = X^\ell$ for some $X \in \mathcal{A}_{k-1}$, and thus the primitive root of $\text{str}(T_k[j])$ is of length at most $\text{len}(X) \leq d_{k-1} = d_k$. If $\ell \geq 2$ and k is even, then, by Corollary 7.5, $T_{k-1}[i..i+\ell) \in \mathcal{A}_{k-1}^\ell$ and $\ell \leq 6$. Consequently, $|\text{str}(T_k[j])| \leq \ell \cdot d_{k-1} \leq 6d_{k-1} = 3d_k$. \square

Lemma 7.12. *For every $T \in \Sigma^+$, $k \in \mathbb{Z}_{\geq 0}$, and $j \in [1..|T_k|]$, we have $|\text{str}(T_k[j..j+1])| > \frac{1}{2}d_k$.*

Proof. We proceed by induction on k . For $k = 0$, the claim holds trivially: $|\text{str}(T_0[j..j+1])| = 2 > \frac{1}{2}$. Otherwise, let $T_{k-1}[i..i+\ell]$ be the fragment of T_{k-1} obtained by expanding both symbols of $T_k[j..j+1]$. If k is odd, then $|\text{str}(T_{k-1}[i..i+\ell])| \geq |\text{str}(T_{k-1}[i..i+1])| > \frac{1}{2}d_{k-1} = \frac{1}{2}d_k$. If k is even and $T_{k-1}[i] \in \mathcal{S} \setminus \mathcal{A}_{k-1}$ or $T_{k-1}[i+\ell-1] \in \mathcal{S} \setminus \mathcal{A}_{k-1}$, then $|\text{str}(T_k[j..j+1])| > d_{k-1} = \frac{1}{2}d_k$ by definition of \mathcal{A}_{k-1} . Otherwise, by Corollary 7.5, $\text{deg}(T_k[j]) \geq 2$ and $\text{deg}(T_k[j+1]) \geq 2$, and thus $|\text{str}(T_k[j..j+1])| \geq |\text{str}(T_{k-1}[i..i+3])| > 2 \cdot \frac{1}{2}d_{k-1} = \frac{1}{2}d_k$. \square

Let us define sequences $(\alpha_k)_{k=0}^\infty$ and $(\beta_k)_{k=0}^\infty$ with

$$\alpha_k = \begin{cases} 0 & \text{if } k = 0, \\ \alpha_{k-1} + d_{k-1} & \text{if } k \in \mathbb{Z}_+ \text{ is odd,} \\ \alpha_{k-1} + (h_{k-1} + 7)d_{k-1} & \text{if } k \in \mathbb{Z}_+ \text{ is even;} \end{cases} \quad \beta_k = \begin{cases} 0 & \text{if } k = 0, \\ \beta_{k-1} + d_{k-1} & \text{if } k \in \mathbb{Z}_+ \text{ is odd,} \\ \beta_{k-1} + 4d_{k-1} & \text{if } k \in \mathbb{Z}_+ \text{ is even.} \end{cases}$$

Fact 7.13. *For every $k \in \mathbb{Z}_+$, we have $d_k \leq \alpha_k < (2h_{k-1} + 16)d_{k-1}$ and $\beta_k < 10d_{k-1}$.*

Proof. The lower bound on α_k is shown by simple induction. The base case holds for $k = 1$ due to $\alpha_1 = d_0 = 1 = d_1$. For $k \geq 2$, the inductive assumption yields $\alpha_k \geq \alpha_{k-1} + d_{k-1} \geq 2d_{k-1} = 2^{\lfloor (k-1)/2 \rfloor + 1} \geq 2^{\lfloor k/2 \rfloor} = d_k$.

As for the upper bound on α_k , we use induction with a stronger upper bound of $\alpha_k < (h_{k-1} + 9)d_{k-1}$ for odd values k . The base case of $k = 1$ holds due to $\alpha_1 = 1 < h_0 + 9 = (h_0 + 9)d_0$. If k is odd, then $\alpha_k = \alpha_{k-1} + d_{k-1} < (2h_{k-2} + 16)d_{k-2} + d_{k-1} = (h_{k-2} + 9)d_{k-1} \leq (h_{k-1} + 9)d_{k-1}$ holds by the inductive assumption. If k is even, then $\alpha_k = \alpha_{k-1} + (h_{k-1} + 7)d_{k-1} < (h_{k-2} + 9)d_{k-2} + (h_{k-1} + 7)d_{k-1} \leq (2h_{k-1} + 16)d_{k-1}$ holds by the inductive assumption.

As for the upper bound on β_k , we use induction with a stronger upper bound of $\beta_k < 6d_{k-1}$ for odd values k . The base case of $k = 1$ holds due to $\beta_1 = 1 < 6 = 6d_0$. If k is odd, then $\beta_k = \beta_{k-1} + d_{k-1} < 10d_{k-2} + d_{k-1} = 6d_{k-1}$ holds by the inductive assumption. If k is even, then $\beta_k = \beta_{k-1} + 4d_{k-1} < 6d_{k-2} + 4d_{k-1} = 10d_{k-1}$ holds by the inductive assumption. \square

Lemma 7.14. *Consider strings $T, T' \in \Sigma^+$, an integer $k \in \mathbb{Z}_{\geq 0}$, and positions $i \in [0..|T|]$ and $i' \in [0..|T'|]$. Suppose that the following two conditions are satisfied for every string $U \in \Sigma^*$:*

- *If $cn \leq \beta_k$, then U is a prefix of $T(i..|T|)$ if and only if U is a prefix of $T'(i'..|T'|)$.*
- *If $cn \leq \alpha_k$, then U is a suffix of $T[1..i]$ if and only if U is a suffix of $T'[1..i']$.*

Then, $i \in B_k(T)$ if and only if $i' \in B_k(T')$.

Proof. We proceed by induction on k . The base case of $k = 0$ is trivially satisfied due to $B_0(T) = [0..|T|]$, $B_0(T') = [0..|T'|]$, and $\alpha_0 = \beta_0 = 0$.

For the inductive step, consider $k \in \mathbb{Z}_+$. We first claim that, for each $k' \in [0..k)$ and $\delta \in [\alpha_{k'} - \alpha_k.. \beta_k - \beta_{k'}]$, we have $i + \delta \in B_{k'}(T)$ if and only if $i' + \delta \in B_{k'}(T')$. By symmetry, we may assume for a proof by contradiction that $i + \delta \in B_{k'}(T)$ (and, in particular, $i + \delta \in [0..|T|]$) yet $i' + \delta \notin B_{k'}(T')$. We consider two cases.

$\delta \in [0.. \beta_k - \beta_{k'}]$. Due to $T(i..i+\delta) \in \Sigma^{\leq \beta_k}$, this string must occur as a prefix of $T'(i'..|T'|)$. Thus, $i' + \delta \leq |T'|$ and $T(i..i+\delta) = T'(i'..i'+\delta)$. If there is a string $U \in \Sigma^{\leq \beta_{k'}}$ that is a prefix of exactly one of the suffixes $T(i+\delta..|T|)$ and $T'(i'+\delta..|T'|)$, then $T(i..i+\delta) \cdot U \in \Sigma^{\leq \beta_k}$ is a prefix of exactly one of the suffixes $T(i..|T|)$ and $T'(i'..|T'|)$. Otherwise, the inductive

assumption yields a string $U \in \Sigma^{\leq \alpha_{k'}}$ that is a suffix of exactly one of the prefixes $T[1..i+\delta]$ and $T'[1..i'+\delta]$. In this case, $U(cn-\delta..cn) = T(i..i+\delta) = T'(i'..i'+\delta)$ and $U[1..cn-\delta] \in \Sigma^{\leq \alpha_k}$ is a suffix of exactly one of the prefixes $T[1..i]$ and $T'[1..i']$.

$\delta \in [\alpha_{k'} - \alpha_k .. 0]$. Due to $T(i+\delta..i) \in \Sigma^{\leq \alpha_k}$, this string must occur as a suffix of $T'[1..i']$. Thus, $i'+\delta \geq 0$ and $T(i+\delta..i) = T'(i'+\delta..i')$. If there is a string $U \in \Sigma^{\leq \alpha_{k'}}$ that is a suffix of exactly one of the prefixes $T[1..i+\delta]$ and $T'[1..i'+\delta]$, then $U \cdot T(i+\delta..i) \in \Sigma^{\leq \alpha_k}$ is a suffix of exactly one of the prefixes $T[1..i]$ and $T'[1..i']$. Otherwise, the inductive assumption yields a string $U \in \Sigma^{\leq \beta_{k'}}$ that is a prefix of exactly one of the suffixes $T(i+\delta..|T|)$ and $T'(i'+\delta..|T'|)$. In this case, $U[1..-\delta] = T(i+\delta..i) = T'(i'+\delta..i')$ and $U(-\delta..cn) \in \Sigma^{\leq \beta_k}$ is a prefix of exactly one of the suffixes $T(i..|T|)$ and $T'(i'..|T'|)$.

If $i \notin B_{k-1}(T)$ and $i' \notin B_{k-1}(T')$, then $i \notin B_k(T)$ and $i' \notin B_k(T')$, so the lemma holds trivially. Otherwise, the claim, instantiated with $k' = k-1$ and $\delta = 0$, implies that both $i \in B_{k-1}(T)$ and $i' \in B_{k-1}(T')$. Let us set j, j' so that $i = |\text{str}(T_{k-1}[1..j])|$ and $i' = |\text{str}(T'_{k-1}[1..j'])|$. By the assumption on i, i' , exactly one of the positions j, j' is an endpoint of a block of the parsing of T_{k-1} and T'_{k-1} .

If k is odd, this means that either $T_{k-1}[j] = T_{k-1}[j+1] \in \mathcal{A}_{k-1}$ or $T'_{k-1}[j'] = T'_{k-1}[j'+1] \in \mathcal{A}_{k-1}$ (but not both). Without loss of generality, suppose that $T_{k-1}[j] = T_{k-1}[j+1] = X$ for some $X \in \mathcal{A}_{k-1}$. By the claim, for each $k' \in [0..k)$, the fragments $T(i - |\text{str}(X)|..i + |\text{str}(X)|)$ and $T'(i' - |\text{str}(X)|..i' + |\text{str}(X)|)$ are parsed into level- k' phrases in the same way. In particular, this implies $T'_{k-1}[j'] = T_{k-1}[j] = T_{k-1}[j+1] = T'_{k-1}[j'+1] \in \mathcal{A}_{k-1}$, a contradiction.

If k is even, the aforementioned condition yields by Corollary 7.6 a string $S \in \mathcal{A}_{k-1}^{\leq 4}$ that is a prefix of exactly one of the suffixes $T_{k-1}(j..|T_{k-1}|)$ and $T'_{k-1}(j'..|T'_{k-1}|)$, or a string $S \in \mathcal{A}_{k-1}^{\leq h_{k-1}+7}$ that is a suffix of exactly one of the prefixes $T_{k-1}[1..j]$ and $T'_{k-1}[1..j']$. Without loss of generality, suppose that S is a prefix of $T_{k-1}(j..|T_{k-1}|)$ or a suffix of $T_{k-1}[1..j]$. If S is a prefix of $T_{k-1}(j..|T_{k-1}|)$ then, due to $|\text{str}(S)| \leq \beta_k - \beta_{k-1}$, the claim implies that, for each $k' \in [0..k)$, the fragments $T(i..i + |\text{str}(S)|)$ and $T'(i'..i' + |\text{str}(S)|)$ are parsed into level- k' phrases in the same way. In particular, S is also a prefix of $T'_{k-1}(j'..|T'_{k-1}|)$, a contradiction. Similarly, if S is a suffix of $T_{k-1}[1..j]$ then, due to $|\text{str}(S)| \leq \alpha_k - \alpha_{k-1}$, the claim implies that, for each $k' \in [0..k)$, the fragments $T(i - |\text{str}(S)|..i)$ and $T'(i' - |\text{str}(S)|..i')$ are parsed into level- k' phrases in the same way. In particular, S is also a suffix of $T'_{k-1}[1..j']$, a contradiction. \square

Corollary 7.15. *Consider strings $T, T' \in \Sigma^+$, an integer $k \in \mathbb{Z}_{\geq 0}$, and a string $S \in \mathcal{S}^*$.*

- *If S is a prefix of exactly one of the strings T_k, T'_k , then $\text{lcp}(T, T') < |\text{str}(S)| + \beta_k$.*
- *If S is a suffix of exactly one of the strings T_k, T'_k , then $\text{lcs}(T, T') < |\text{str}(S)| + \alpha_k$.*

Proof. We proceed by induction on k . The case of $k = 0$ is trivial. For a proof by contradiction, suppose that $\text{lcp}(T, T') \geq |\text{str}(S)| + \beta_k$ and S is prefix of T_k yet S is not a prefix of T'_k (the remaining cases are symmetric). Let S' be the prefix of T_{k-1} obtained by expanding symbols in S . By the inductive assumption, S' is also a prefix of T'_{k-1} . At the same time Lemma 7.14 yields $B_k(T) \cap [0.. \text{lcp}(T, T') - \beta_k] = B_k(T') \cap [0.. \text{lcp}(T, T') - \beta_k]$. Hence, the block boundaries within S' are placed in the same way in the parsing of T_{k-1} and T'_{k-1} . Consequently, S is also a prefix of T'_k , a contradiction. \square

8 Dynamic Strings

By Lemma 7.12, for every signature function sig and every string $T \in \Sigma^+$ $|T_k| = 1$ holds for sufficiently large $k \in \mathbb{Z}_{\geq 0}$ (whenever $k \geq 2\lceil \log(2|T|) \rceil$). Hence, we define $\text{symb}_{\text{sig}}(T)$ as the unique symbol in the string T_k for the smallest $k \in \mathbb{Z}_{\geq 0}$ with $|T_k| = 1$.

We maintain a growing set of strings $\mathcal{W} \subseteq \Sigma^+$ and represent each string $T \in \mathcal{W}$ as $\text{symb}_{\text{sig}}(T)$ for implicit signature function sig . Our data structure consists of a grammar $\mathcal{G} \subseteq \mathcal{S}$ (maintained using Lemma 7.1) and the values $\text{sig}(X)$ stored for all $X \in \mathcal{G}$. The key invariant is that $\text{symb}_{\text{sig}}(T) \in \mathcal{G}$ holds for each $T \in \mathcal{W}$; based on this, we represent each string $T \in \mathcal{W}$ using the identifier $\text{id}(\text{symb}_{\text{sig}}(T))$ of the underlying symbol. The following simple observation allows leaving $\text{sig}(X)$ unspecified for $X \in \mathcal{S} \setminus \mathcal{G}$.

Observation 8.1. *Consider a grammar $\mathcal{G} \subseteq \mathcal{S}$ and two signature functions sig, sig' such that $\text{sig}|_{\mathcal{G}} = \text{sig}'|_{\mathcal{G}}$. If $\text{symb}_{\text{sig}}(T) \in \mathcal{G}$ for some string $T \in \Sigma^+$, then $\text{symb}_{\text{sig}'}(T) = \text{symb}_{\text{sig}}(T)$.*

Efficiency of our data structure is supported by an additional invariant that, for each $k \in \mathbb{Z}_{\geq 0}$, the signature function sig injectively maps $\mathcal{A}_k \cap \mathcal{G}$ to $[0..|\mathcal{A}_k \cap \mathcal{G}|)$. Whenever a symbol $X \in \mathcal{A}_k \setminus \mathcal{G}$ is added to \mathcal{G} , it is assigned the smallest “free” signature $|\mathcal{A}_k \cap \mathcal{G}|$. This way, we make sure that the explicitly assigned signature values have $\mathcal{O}(\log |\mathcal{G}|)$ bits.

In the following, we describe various operations supported by this data structure. Internally, we measure the efficiency in terms of the grammar size $g := |\mathcal{G}|$ (after executing the operation), the alphabet size $\sigma := |\Sigma|$, and the lengths of the strings involved in the operations. We also assume that the machine word size w satisfies $w = \Omega(\log(\sigma g))$.

8.1 Access

The $\text{access}(T, i)$ operation retrieves $T[i]$ for a given string $T \in \mathcal{W}$ and position $i \in [1..|T|]$. For this, we traverse the parse tree $\mathcal{T}(\text{symb}_{\text{sig}}(T))$ maintaining a pointer to a node ν such that $\text{str}(\nu) = T[\ell..r]$ for $i \in (\ell..r]$. If ν is a leaf, we return $\text{letter}^{-1}(\text{symb}(\nu))$. If ν has $d \leq 6$ children, we compute $\text{pLen}(\text{symb}(\nu), j)$ for $j \in [0..d]$ and descend to the child ν_j such that $\text{pLen}(\text{symb}(\nu), j-1) < i - \ell \leq \text{pLen}(\text{symb}(\nu), j)$. Otherwise, we have $\text{pLen}(\text{symb}(\nu), j) = j \cdot \text{pLen}(\text{symb}(\nu), 1)$ for $j \in [0..d]$, so we descend to the $\lceil \frac{i-\ell}{\text{pLen}(\text{symb}(\nu), 1)} \rceil$ -th child. Overall, the running time is $\mathcal{O}(\text{level}(\text{symb}_{\text{sig}}(T))) = \mathcal{O}(\log |T|)$.

8.2 Longest Common Prefix

The operation $\text{lcp}(T, U)$ computes the (length of) the longest common prefix of any two strings $T, U \in \mathcal{W}$. Let $(T_k)_{k=0}^{\infty}$ and $(U_k)_{k=0}^{\infty}$ be the balanced signature parsing of T and S , respectively. Moreover, let $X = \text{symb}_{\text{sig}}(T)$ and $Y = \text{symb}_{\text{sig}}(U)$. Our algorithm traverses the parse trees $\mathcal{T}(X)$ and $\mathcal{T}(Y)$ maintaining two pointers ν_T and ν_U , as illustrated in Algorithm 1.

In this algorithm, we use a short-hand $\text{level}(\nu) := \text{level}(\text{symb}(\nu))$ and an extra $\text{right}(\nu)$ operation that, given a node ν corresponding to $T_k[i]$, returns the node corresponding to $T_k[i+1]$ or \perp if $i = |T_k|$ (and analogously for U_k). We maintain an invariant at Line 6 that $\text{level}(\nu_T) = \text{level}(\nu_U)$ and, denoting this common value by k , the pointers ν_T and ν_U correspond to symbols $T_k[i]$ and $U_k[i]$ such that $T_k[1..i] = U_k[1..i]$ and $\text{lcp} = |\text{str}(T_k[1..i])|$. It is easy to see that this invariant is indeed satisfied after executing Line 5 with $i = \text{lcp} = 0$ and $k = \min(\text{level}(\text{symb}_{\text{sig}}(T)), \text{level}(\text{symb}_{\text{sig}}(U)))$. The while loop of Line 7 maintains the invariant and increments i as far as possible. If i reaches

Algorithm 1: $\text{lcp}(T, U)$

```
1  $\nu_T := \text{root}(\text{symb}_{\text{sig}}(T));$ 
2  $\nu_U := \text{root}(\text{symb}_{\text{sig}}(U));$ 
3  $\text{lcp} := 0;$ 
4 while  $\text{level}(\nu_T) > \text{level}(\nu_U)$  do  $\nu_T := \text{child}(\nu_T, 1);$ 
5 while  $\text{level}(\nu_U) > \text{level}(\nu_T)$  do  $\nu_U := \text{child}(\nu_U, 1);$ 
6 while true do
7   while  $\nu_T \neq \perp$  and  $\nu_U \neq \perp$  and  $\text{symb}(\nu_T) = \text{symb}(\nu_U)$  do
8      $\text{lcp} := \text{lcp} + \text{len}(\text{symb}(\nu_T));$ 
9      $\nu_T := \text{right}(\nu_T);$ 
10     $\nu_U := \text{right}(\nu_U);$ 
11  if  $\nu_T = \perp$  or  $\nu_U = \perp$  or  $\text{level}(\nu_T) = 0$  then return  $\text{lcp};$ 
12   $\nu_T := \text{child}(\nu_T, 1);$ 
13   $\nu_U := \text{child}(\nu_U, 1);$ 
```

$|T_k| + 1$ or $|U_k| + 1$, then we conclude that T_k is a prefix of U_k (or vice versa), and thus the longest common prefix of T and S is of length $\text{lcp} = \min(|\text{str}(T_k)|, |\text{str}(U_k)|) = \min(|T|, cn)$; it is then reported correctly. Similarly, if $k = 0$ and $T_k[i] \neq U_k[i]$, then the longest common prefix of T and S is of length $i - 1 = \text{lcp}$. In the remaining case, by moving ν_T and ν_U to the leftmost children, we maintain the invariant and decrement the level k .

An efficient implementation of Algorithm 1 requires one optimization: if, at Line 12, we have $\text{symb}(\nu_T) = \text{block}(S_X)$ and $\text{symb}(\nu_U) = \text{block}(S_Y)$, then we immediately simulate the first $\ell := \text{lcp}(S_X, S_Y)$ subsequent steps of Line 6. (Since S_X and S_Y are symbol powers or strings of constant length, the value ℓ can be determined in $\mathcal{O}(1)$ time.) To carry out such simulation, we set $\text{lcp} := \text{lcp} + \text{pLen}(\nu_T, \ell)$, $\nu_T := \text{child}(\nu_T, \ell + 1)$, and $\nu_U := \text{child}(\nu_U, \ell + 1)$, where $\text{child}(\nu, \text{deg}(\nu) + 1) = \text{child}(\text{right}(\nu), 1)$.

As our traversal of $\mathcal{T}(X)$ and $\mathcal{T}(Y)$ always proceeds forward in the pre-order, the time complexity is proportional to the number of visited nodes (including the ancestors visited while traversing the paths from ν to $\text{right}(\nu)$). Corollary 7.15 guarantees that, for each k , the visited nodes at level k correspond to level- k phrases overlapping $T(\max(0, \text{lcp}(T, U) - \beta_{k+1}) \dots \text{lcp}(T, U))$. By Lemma 7.12 and Fact 7.13, the number of such phrases is $\mathcal{O}(\frac{\beta_{k+1}}{d_k}) = \mathcal{O}(1)$. Consequently, the overall running time is $\mathcal{O}(\text{level}(X) + \text{level}(Y)) = \mathcal{O}(\log |TU|)$.

8.3 Longest Common Suffix

A symmetric operation $\text{lcs}(T, U)$ computes the (length of) the longest common suffix of any two strings $T, U \in \mathcal{W}$. Its implementation is analogous to that of $\text{lcp}(T, U)$. However, in the analysis, the number of level- k nodes visited is $\mathcal{O}(\frac{\alpha_{k+1}}{d_k}) = \mathcal{O}(h_{k+1}) = \mathcal{O}(\log^*(k\sigma))$, which yields the total running time of $\mathcal{O}(\log |TU| \log^*(|TU|\sigma))$.

8.4 Insertion

The $\text{insert}(T)$ operation inserts to \mathcal{W} a given string $T \in \Sigma^+$. For this, we construct the subsequent levels of the balanced signature parsing $(T_k)_{k=0}^\infty$ until reaching $|T_k| = 1$ so that $\text{symb}_{\text{sig}}(T) = T_k[1]$.

The construction of T_k for $k = 0$ costs $\mathcal{O}(|T| \frac{\log^2 \log g}{\log \log \log g})$ time. The cost for odd values $k \in \mathbb{Z}_+$ is $\mathcal{O}(|T_{k-1}| \frac{\log^2 \log g}{\log \log \log g})$, whereas the cost for even values $k \in \mathbb{Z}_+$ is $\mathcal{O}(|T_{k-1}| (h_{k-1} + \frac{\log^2 \log g}{\log \log \log g})) = \mathcal{O}(|T_{k-1}| (\log^*(\sigma k) + \frac{\log^2 \log g}{\log \log \log g}))$ by Theorem 7.3 and Fact 7.8. Since Lemma 7.12 yields $|T_k| = \mathcal{O}(1 + \frac{|T|}{d_k}) = \mathcal{O}(1 + \frac{|T|}{2^{k/2}})$, the overall cost is $\mathcal{O}(|T| (\log^* \sigma + \frac{\log^2 \log g}{\log \log \log g}))$.

8.5 Concatenation

The $\text{concat}(L, R)$ operation, given two strings $L, R \in \mathcal{W}$, inserts their concatenation $T := L \cdot R$ to \mathcal{W} . Consider the balanced signature parsing $(L_k)_{k=0}^\infty$ of L , $(R_k)_{k=0}^\infty$ of R , and $(T_k)_{k=0}^\infty$ of T . For each $k \in \mathbb{Z}_{\geq 0}$, let P_k denote the longest prefix of T_k with $|\text{str}(P_k)| \leq \max(0, |L| - \beta_k)$, and let S_k denote the longest suffix of T_k with $|\text{str}(S_k)| \leq \max(0, |R| - \alpha_k)$. By Corollary 7.15, P_k is also the longest prefix of L_k with $|\text{str}(P_k)| \leq \max(0, |L| - \beta_k)$, and S_k is also the longest suffix of R_k with $|\text{str}(S_k)| \leq \max(0, |R| - \alpha_k)$. Moreover, define L'_k, R'_k, C_k so that $L_k = P_k L'_k$, $R_k = R'_k S_k$, and $T_k = P_k C_k S_k$. Furthermore, let P'_k be the suffix of P_k such that $\text{str}(P_k) = \text{str}(P_{k+1}) \text{str}(P'_k)$, and let S'_k be the prefix of S_k such that $\text{str}(S_k) = \text{str}(S'_k) \text{str}(S_{k+1})$.

Our implementation of $\text{concat}(L, R)$ constructs C_k for subsequent integers $k \in \mathbb{Z}_{\geq 0}$. For $k = 0$, the string C_0 is empty. For odd $k \in \mathbb{Z}_+$, we build $P'_{k-1} C_{k-1} S'_{k-1}$ and apply the restricted run-length parsing with respect to \mathcal{A}_{k-1} . Finally, we retrieve C_k by collapsing each block into the corresponding symbol, inserted to \mathcal{G} using insertPower . For even $k \in \mathbb{Z}_+$, we additionally retrieve the longest string $P''_{k-1} \in \mathcal{A}_{k-1}^{\leq h_{k-1}+7}$ such that $P''_{k-1} P'_{k-1}$ is a suffix of P_{k-1} , and the longest string $S''_{k-1} \in \mathcal{A}_{k-1}^{\leq 4}$ such that $S'_{k-1} S''_{k-1}$ is a prefix of S_{k-1} . Then, we perform the balanced signature parsing of $P''_{k-1} P'_{k-1} C_{k-1} S'_{k-1} S''_{k-1}$ with respect to \mathcal{A}_{k-1} , h_{k-1} , and sig_{k-1} . Finally, we cut $P'_{k-1} C_{k-1} S'_{k-1}$ (which is guaranteed to consist of full blocks by Corollary 7.6), and we obtain C_k by collapsing each block into the corresponding symbol, inserted to \mathcal{G} using insertString . We stop as soon as $|C_k| = 1$ and $|P_k| = |S_k| = 0$; the only symbol of C_k is then $Z := \text{symb}_{\text{sig}}(T)$.

The efficiency of this procedure follows from the fact that, by Lemma 7.12 and Fact 7.13, $|C_k| = \mathcal{O}(h_k)$, $|L'_k| = \mathcal{O}(1)$, and $|R'_k| = \mathcal{O}(h_k)$. For odd $k \in \mathbb{Z}_+$, we also have $|P''_{k-1} P'_{k-1}| = \mathcal{O}(h_k)$ and $|S'_{k-1} S''_{k-1}| = \mathcal{O}(h_k)$, whereas for even $k \in \mathbb{Z}_+$, the strings P'_{k-1} and S'_{k-1} have run-length parsing of size $\mathcal{O}(1)$ and $\mathcal{O}(h_k)$, respectively. In particular, all the required strings P'_{k-1} and P''_{k-1} can be generated in $\mathcal{O}(\log |T| \log^*(|T|\sigma))$ time by traversing the parse tree $\mathcal{T}(X)$, whereas all the required strings S'_{k-1} and S''_{k-1} can be generated in $\mathcal{O}(\log |T| \log^*(|T|\sigma))$ time by traversing the parse tree $\mathcal{T}(Y)$. The cost of run-length parsing at even levels $k \in \mathbb{Z}_+$ is $\mathcal{O}(h_k) = \mathcal{O}(\log^*(|T|\sigma))$, whereas the cost of signature parsing at odd levels $k \in \mathbb{Z}_+$ is $\mathcal{O}(h_{k-1} h_k) = \mathcal{O}((\log^* |T|\sigma)^2)$, for a total of $\mathcal{O}(\log |T| (\log^*(|T|\sigma))^2)$ across all levels. Collapsing each block into the corresponding symbol costs $\mathcal{O}(\frac{\log^2 \log g}{\log \log \log g})$ time, for a total of $\mathcal{O}(\log |T| \log^*(|T|\sigma) (\log^*(|T|\sigma) + \frac{\log^2 \log g}{\log \log \log g}))$ time.

8.6 Split

The $\text{split}(T, i)$ operation, given $T \in \mathcal{W}$ and $i \in [1 \dots |T|]$, inserts $L := T[1 \dots i]$ and $R := T[i \dots |T|]$ to \mathcal{W} . We utilize the same notation as in the implementation of the operation $\text{concat}(L, R)$.

To derive $\text{symb}(L, \text{sig})$, we build L'_k for subsequent integers $k \in \mathbb{Z}_{\geq 0}$. For $k = 0$, the string L'_0 is empty. For odd $k \in \mathbb{Z}_+$, we build $P'_{k-1} L'_{k-1}$ and apply the restricted run-length parsing with respect to \mathcal{A}_{k-1} . Finally, we retrieve L'_k by collapsing each block into the corresponding symbol, inserted to \mathcal{G} using insertPower . For even $k \in \mathbb{Z}_+$, we build $P''_{k-1} P'_{k-1} L'_{k-1}$ and apply the balanced signature

parsing with respect to \mathcal{A}_{k-1} , h_{k-1} , and sig_{k-1} . Finally, we cut $P'_{k-1}L'_{k-1}$ (which is guaranteed to consist of full blocks by Corollary 7.6), and we obtain L'_k by collapsing each block into the corresponding symbol, inserted to \mathcal{G} using `insertString`. We stop as soon as $|L'_k| = 1$ and $|P_k| = 0$; the only symbol of L'_k is then $\text{symb}(L, \text{sig})$.

Symmetrically, to derive $\text{symb}(R, \text{sig})$, we build R'_k for subsequent integers $k \in \mathbb{Z}_{\geq 0}$. For $k = 0$, the string R'_0 is empty. For odd $k \in \mathbb{Z}_+$, we build $R'_{k-1}S'_{k-1}$ and apply the restricted run-length parsing with respect to \mathcal{A}_{k-1} . Finally, we retrieve R'_k by collapsing each block into the corresponding symbol, inserted to \mathcal{G} using `insertPower`. For even $k \in \mathbb{Z}_+$, we build $R'_{k-1}S'_{k-1}S''_{k-1}$ and apply the balanced signature parsing with respect to \mathcal{A}_{k-1} , h_{k-1} , and sig_{k-1} . Finally, we cut $R'_{k-1}S'_{k-1}$ (which is guaranteed to consist of full blocks by Corollary 7.6), and we obtain R'_k by collapsing each block into the corresponding symbol, inserted to \mathcal{G} using `insertString`. We stop as soon as $|R'_k| = 1$ and $|S_k| = 0$; the only symbol of R'_k is then $\text{symb}(R, \text{sig})$. The complexity analysis is similar to that for `concat(L, R)`.

8.7 Lexicographic Order of Cyclic Fragments

For a string $T \in \Sigma^+$, let $\text{CF}(T) = \{T^\infty(i..j) : i, j \in \mathbb{Z} \text{ such that } i \leq j\}$; note that each fragment in $\text{CF}(T)$ can be represented using its endpoints i, j . We also define $\text{CF}^+(T) = \text{CF}(T) \cup \{F \cdot c^\infty : F \in \text{CF}(T)\}$, where $c = \max \Sigma$. Finally, $\text{CF}(\mathcal{W}) = \bigcup_{T \in \mathcal{W}} \text{CF}(T)$ and $\text{CF}^+(\mathcal{W}) = \bigcup_{T \in \mathcal{W}} \text{CF}^+(T)$.

Our next operation `compare(F1, F2)`, given $F_1, F_2 \in \text{CF}^+(\mathcal{W})$, decides whether $F_1 \prec F_2$. For each $i \in \{1, 2\}$, let $G_i = T_i^\infty(\ell_i..r_i) \in \text{CF}(T_i)$ be such that $F_i \in \{G_i, G_i \cdot c^\infty\}$ and $T_i \in \mathcal{W}$. Our initial goal is to determine $\text{lcp}(G_1, G_2)$. As the first step, we use the `split` and `concat` operations in order to insert $T_i^\infty(\ell_i.. \ell_i + |T_i|)$. Effectively, this lets us assume that $\ell_i = 0$. Moreover, by symmetry, we assume that $|T_1| \leq |T_2|$. Next, we compute $\min(r_1, r_2, \text{lcp}(T_1, T_2))$. If this value is less than $|T_1|$, then it is equal to the sought longest common prefix $\text{lcp}(T_1^\infty(0..r_1), T_2^\infty(0..r_2))$. Otherwise, we use the `split` and `concat` operations to insert $T_2^\infty(|T_1|..|T_1T_2|)$ to \mathcal{W} , and we compute $\min(r_1, r_2, |T_1| + \text{lcp}(T_2, T_2^\infty(|T_1|..|T_1T_2|)))$. If this value is less than $|T_1T_2|$, then it is equal to the sought longest common prefix $\text{lcp}(T_1^\infty(0..r_1), T_2^\infty(0..r_2))$. In the remaining case, we simply have $\text{lcp}(T_1^\infty(0..r_1), T_2^\infty(0..r_2)) = \min(r_1, r_2)$. If the reported value t is smaller than $\min(r_1, r_2)$, then we use the `access` operation to check $G_1[t+1] \prec G_2[t+1]$ and return the reported answer. If G_1 is a proper prefix of G_2 , we return YES if and only if $F_1 \in \text{CF}(T_1)$. If G_2 is a proper prefix of G_1 , we return YES if and only if $F_2 \notin \text{CF}(T_2)$. The running time is $\mathcal{O}(\log |T| \log^*(|T|\sigma)(\log^*(|T|\sigma) + \frac{\log^2 \log g}{\log \log \log g}))$.

A symmetric procedure lets us implement `compare(F1, F2)` if $F_1, F_2 \in \text{CF}^+(\overline{\mathcal{W}})$, where $\overline{\mathcal{W}} = \{\overline{T} : T \in \mathcal{W}\}$.

8.8 Internal Pattern Matching

The `ipm(P, T)` operation, given $P, T \in \mathcal{W}$ with $|P| \leq |T| \leq 2|P|$, computes $\text{Occ}(P, T) := \{o \in [0..|T| - |P|] : P = T(o..o + |P|)\}$. As shown in [Koc18, KRRW15], $\text{Occ}(P, T)$ forms an arithmetic progression.

Our implementation of `ipm(P, T)` first identifies the maximum level $k \in \mathbb{Z}_{\geq 0}$ with $|P| \geq \alpha_k + \beta_k + 3d_k$.⁶ We consider two cases depending on whether $B_k(P) \cap [\alpha_k..|P| - \beta_k] = \emptyset$.

If there exists $i \in B_k(P) \cap [\alpha_k..|P| - \beta_k]$ then, Lemma 7.14 shows that $i + o \in B_k(T)$ holds for any $o \in \text{Occ}(P, T)$. Thus, for each position $j \in B_k(T) \cap [i..|T|]$, we compute $o := j - i$, extract

⁶If there is no such level, i.e., if $|P| \leq 2$, we answer the query naively by decompressing P and T .

$T(o..|T|]$ via $\text{split}(T, o)$, and check whether P is a prefix of $T(o..|T|]$ via $\text{lcp}(P, T(o..|T|])$.

Thus, it remains to consider the case when $B_k(P) \cap [\alpha_k..|P| - \beta_k] = \emptyset$, i.e., when $P[\alpha_k..|P| - \beta_k + 1]$ is contained in a single level- k phrase. Suppose that this phrase is $P(\ell..r]$; due to $r - \ell > 3d_k$, Fact 7.11 shows that $P(\ell..r] = Q^c$ for some integer $c \in \mathbb{Z}_{\geq 2}$ and primitive string Q with $|Q| \leq d_k$. Moreover, if $k' \in \mathbb{Z}_{\geq 0}$ is the maximum level such that $P(\ell..r]$ consists of multiple level- k' phrases, then all these phrases match Q . By Lemma 7.14, for each $o \in \text{Occ}(P, T)$, the fragment $T(o + \alpha_k..o + |P| - \beta_k]$ is also contained in a single level- k phrase $T(\ell'..r']$. Moreover, another application of Lemma 7.14 (at level k') shows that the phrase $T(\ell'..r']$ is decomposed into $c' \in \mathbb{Z}_{\geq 2}$ level- k' phrases matching Q . Our algorithm computes the symbol of P_k corresponding to $P(\ell..r]$, the symbol of $P_{k'}$ corresponding to the primitive root Q , and all the candidate phrases $T(\ell'..r']$ with the same primitive root Q . We select an arbitrary phrase boundary $i \in B_{k'}(P) \cap [\alpha_k..|P| - \beta_k]$ and observe that, by Lemma 7.14, for each $o \in \text{Occ}(P, T)$, we have $i + o \in B_{k'}(T)$. Given that we assume that $T(o + \alpha_k..o + |P| - \beta_k]$ is contained in $T(\ell'..r']$, we further have $i + o \in (\ell'..r')$, i.e., $j := i + o$ is one of the boundaries in the decomposition of $T(\ell'..r'] = Q^{c'}$ into c' individual occurrences of Q . Our goal is to verify each candidate j by checking whether $P(i..|P|]$ is a prefix of $T(j..|T|]$ and whether $P[1..i]$ is a suffix of $T[1..j]$. A naive implementation performs one split , one lcp , and one lcs operation per candidate position j . The positions j form a so-called *periodic progression* (with period Q); thus, as shown in [Koc18, Lemma 7.1.4(c)], positions j maximizing $\text{lcp}(P(i..|P|], T(j..|T|])$ are contiguous elements of the periodic progression, and they can be retrieved using a constant number of lcp queries concerning suffixes of P and T . Thus, using $\mathcal{O}(1)$ calls to split and lcp , we can filter positions j for which $P(i..|P|]$ is a prefix of $T(j..|T|]$. As symmetric procedure using $\mathcal{O}(1)$ calls to split and lcs filters positions j for which $P(i..|P|]$ is a prefix of $T(j..|T|]$. The set of positions $o \in \text{Occ}(P, T)$ corresponding to $T(\ell'..r']$ is obtained by intersecting the two ranges and shifting it by i positions to the left. Finally, the occurrences corresponding to various candidate phrase $T(\ell'..r']$ are retrieved by combining all $o \in \text{Occ}(P, T)$ into a single arithmetic progression.

As for the running time, we observe that the number of candidate position j (in the first case) and the number of candidate phrases $T(\ell'..r']$ (in the second case) is bounded by $|T_k| = \mathcal{O}(1 + \frac{|T|}{d_k}) = \mathcal{O}(1 + \frac{|P|}{d_k}) = \mathcal{O}(h_k) = \mathcal{O}(\log^*(|T|\sigma))$ by Fact 7.13. The overall running time is therefore $\mathcal{O}(\log |T| (\log^*(|T|\sigma))^2 (\log^*(|T|\sigma) + \frac{\log^2 \log g}{\log \log \log g}))$.

8.9 2-Period Queries

A 2-period query $\text{period}(T)$, given $T \in \mathcal{W}$, asks to compute the shortest period $\text{per}(T)$ or to report that $\text{per}(T) > \frac{1}{2}|T|$. As shown in [Koc18, KRRW15], each of these queries can be reduced to a constant number of ipm and lcp queries on substrings of T . Consequently, $\text{period}(T)$ queries can also be answered in $\mathcal{O}(\log |T| (\log^*(|T|\sigma))^2 (\log^*(|T|\sigma) + \frac{\log^2 \log g}{\log \log \log g}))$ time.

8.10 Canonical Cyclic Shift

As shown in [KRRW15], cyclic equivalence of T and T' can be tested using a constant number of ipm and lcp queries on substrings of T ; the resulting algorithm also reports a shift s with $T' = \text{rot}^s(T)$. In this paper, however, we need a stronger operation that, given a string $T \in \mathcal{W}$, computes a *canonical cyclic shift* $\text{canShift}(T)$ such that $f_{\text{sig}}(T) := \text{rot}^{\text{canShift}(T)}(T)$ is a necklace-consistent function (as defined in Definition 6.17).

Before implementing an algorithm computing $\text{canShift}(T)$, let us provide a synthetic definition of the underlying function f in terms of the signature function sig .

Construction 8.2. For a string $T \in \Sigma^+$, let $k = \min\{t \in \mathbb{Z}_{\geq 0} : d_t \geq 4|T|\}$ and $U = T^c$, where $c = \lceil \frac{\alpha_k + \beta_k + 3d_k}{|T|} \rceil$. For a signature function sig , we set $f_{\text{sig}}(T) = \text{rot}^{\max(B_k(U) \cap [0.. \alpha_k])}(T)$.⁷

Lemma 8.3. For every signature function sig , the function $f_{\text{sig}} : \Sigma^+ \rightarrow \Sigma^+$ defined in Construction 8.2 is a necklace-consistent function.

Proof. Let us fix $T \in \Sigma^n$ with $n \in \mathbb{Z}_+$. By construction, the string $f_{\text{sig}}(T)$ is cyclically equivalent to T . However, before proving that $f_{\text{sig}}(T) = f_{\text{sig}}(\text{rot}^s(T))$ holds for every $s \in \mathbb{Z}$, let us analyze the properties of the level- k phrase $U(\ell..r)$ in the balanced signature parsing of U such that $\ell = \max(B_k(U) \cap [0.. \alpha_k])$. By definition, $\ell < \alpha_k \leq r$; our first claim is that $r > cn - \beta_k$. Thus, for a proof by contradiction, suppose that $r \in B_k(U) \cap [\alpha_k..cn - \beta_k]$. By Lemma 7.14, $B_k(U) \supseteq \{r' \in [\alpha_k..cn - \beta_k] : r' \equiv r \pmod{n}\}$. Due to $cn - \alpha_k - \beta_k \geq 3d_k > 2n$, this induces two subsequent phrases of total length at most $2n \leq \frac{1}{2}d_k$, contradicting Lemma 7.12. The contradiction completes the proof that $r > cn - \beta_k$. Consequently, $r - \ell > cn - \alpha_k - \beta_k \geq 3d_k$; by Fact 7.11, this means that $U(\ell..r)$ has primitive root of length at most d_k . At the same time, $U(\ell..r)$ has period n (inherited from U). Using the periodicity lemma, we conclude that $U(\ell.. \ell + n)$ is the primitive root of $U(\ell..r)$. In particular, $U(\ell..r)$ has been created at some level $j \in [1..k]$ by merging several phrases matching $U(\ell.. \ell + n)$. Thus, $B_{j-1}(U) \cap [\alpha_k..cn - \beta_k] = \{i \in [\alpha_k..cn - \beta_k] : i \equiv \ell \pmod{n}\}$ and $B_j(U) \cap [\alpha_k..cn - \beta_k] = \emptyset$.

We are now ready to consider the corresponding phrase $U'(\ell'..r')$ in the balanced signature parsing of $U' = \text{rot}^s(U) = (\text{rot}^s(T))^c$. By the above argument, $B_{j'-1}(U') \cap [\alpha_k..cn - \beta_k] = \{i \in [\alpha_k..cn - \beta_k] : i \equiv \ell' \pmod{n}\}$ and $B_{j'}(U') \cap [\alpha_k..cn - \beta_k] = \emptyset$ hold for some $j' \in [1..k]$. However, Lemma 7.14 also yields $B_{j-1}(U') \cap [\alpha_k..cn - \beta_k] = \{i \in [\alpha_k..cn - \beta_k] : i \equiv \ell - s \pmod{n}\}$ and $B_j(U') \cap [\alpha_k..cn - \beta_k] = \emptyset$. Consequently, $j' = j$ and $\ell' \equiv \ell - s \pmod{n}$. In particular, $\text{rot}^{\ell'}(\text{rot}^s(T)) = \text{rot}^{\ell'+s}(T) = \text{rot}^{\ell}(T)$ holds as claimed. \square

Our algorithm computing $\text{canShift}(T)$ simply computes k and c , builds $U = T^c$ by repeated calls to concat , and then builds $B_k(U)$ by traversing the parse tree of $\text{symb}_{\text{sig}}(U)$. Finally, it returns $\max(B_k(U) \cap [0.. \alpha_k])$.

As for the complexity analysis, we note Lemma 7.12 implies $|B_k(U)| = \mathcal{O}(\frac{1}{d_k}|U|) = \mathcal{O}(c)$, whereas Fact 7.13 implies $c = \mathcal{O}(h_k) = \mathcal{O}(\log^*(|T|\sigma))$. Consequently, the total running time does not exceed $\mathcal{O}(\log |T|(\log^*(|T|\sigma))^2(\log^*(|T|\sigma) + \frac{\log^2 \log g}{\log \log \log g}))$.

9 From Balanced Signature Parsing to Synchronizing Sets

Definition 9.1 (τ -runs). For a string $T \in \Sigma^+$ and an integer $\tau \in [1..n]$, we define the set $\text{RUNS}_{\tau}(T)$ of τ -runs in T that consists of all fragments $T[p..q]$ of length at least τ that satisfy $\text{per}(T[p..q]) \leq \frac{1}{3}\tau$ yet cannot be extended (in any direction) while preserving the shortest period.

By the Periodicity Lemma, distinct τ -runs γ, γ' satisfy $|\gamma \cap \gamma'| \leq \frac{2}{3}\tau$. Consequently, each τ -run γ contains at least $\frac{1}{3}\tau$ (trailing) positions which are disjoint from all τ -runs starting to the left of γ . Hence, $\text{RUNS}_{\tau}(T) \leq \frac{3n}{\tau}$.

⁷Recall that the function B_k is defined in Section 7.4 based on the signature function sig .

Construction 9.2. Consider the signature encoding of a string $T \in \Sigma^n$ with respect to a signature function sig and an integer $\tau \in \mathbb{Z}_+$. Based on the set $B_k(T)$ for the largest $k \in \mathbb{Z}_{\geq 0}$ with $\tau \geq 3\alpha_k$, we defined the set of $\mathbf{S}_{\text{sig}}(\tau, T)$ so that it consists of all positions $i \in [1..n - 2\tau + 1]$ that satisfy at least one of the following conditions:

- (1) $i + \tau - 1 \in B_k(T)$ and $T[i..i + 2\tau]$ is not contained in any τ -run,
- (2) $i = p - 1$ for some τ -run $T[p..q] \in \text{RUNS}_\tau(T)$,
- (3) $i = q - 2\tau + 2$ for some τ -run $T[p..q] \in \text{RUNS}_\tau(T)$.

Note that $\mathbf{S}_{\text{sig}}(T, \tau) = \emptyset$ if $\tau > \frac{1}{2}n$.

Lemma 9.3. For every $T \in \Sigma^n$ and $\tau \in [1.. \lfloor \frac{1}{2}n \rfloor]$, the set $\mathbf{S}_{\text{sig}}(\tau, T)$ obtained using Construction 9.2 is a τ -synchronizing set. Moreover, for every $i \in [1..n - 3\tau + 2]$, $|\mathbf{S}_{\text{sig}}(\tau, T) \cap [i..i + \tau]| = \mathcal{O}(\log^*(\tau\sigma))$.

Proof. First, suppose that $i, i' \in [1..n - 2\tau + 1]$ satisfy $T[i..i + 2\tau] = T[i'..i' + 2\tau]$ and $i \in \mathbf{S}_{\text{sig}}(\tau, T)$. We will show that if i satisfies conditions (1)–(3), then i' satisfies the same condition. If i satisfies condition (1), then $\text{per}(T[i'..i' + 2\tau]) = \text{per}(T[i..i + 2\tau]) > \frac{1}{3}\tau$, so $T[i'..i' + 2\tau]$ is not contained in any τ -run. At the same time, due to $T[i + \tau - 1 - \alpha_k..i + \tau - 1 + \beta_k] = T[i' + \tau - 1 - \alpha_k..i' + \tau + \beta_k]$, by Lemma 7.14, $i + \tau - 1 \in B_k(T)$ implies $i' + \tau - 1 \in B_k(T)$. Consequently, i' also satisfies condition (1). If i satisfies condition (2), then $\text{per}(T[i' + 1..i' + \tau]) = \text{per}(T[i + 1..i + \tau]) \leq \frac{1}{3}\tau < \text{per}(T[i..i + \tau]) = \text{per}(T[i'..i' + \tau])$. Hence, $T[i' + 1..i' + \tau]$ can be extended to a τ -run $T[p'..q']$ that starts at position $p' = i + 1$, and thus i' satisfies condition (2). Similarly, if i satisfies condition (3), then $\text{per}(T[i' + \tau - 1..i' + 2\tau - 2]) = \text{per}(T[i + \tau - 1..i + 2\tau - 2]) \leq \frac{1}{3}\tau < \text{per}(T[i + \tau - 1..i + 2\tau - 1]) = \text{per}(T[i' + \tau - 1..i' + 2\tau - 1])$. Hence, $T[i' + \tau - 1..i' + 2\tau - 2]$ can be extended to a τ -run $T[p'..q']$ that ends at position $q' = i + 2\tau - 2$, and thus i' satisfies condition (3).

For a proof of the density condition, consider a position $i \in [1..n - 3\tau + 2]$ with $[i..i + \tau] \cap \mathbf{S}_{\text{sig}}(\tau, T) = \emptyset$. We start by identifying a τ -run $T[p..q]$ with $p \leq i + \tau$ and $q \geq i + 2\tau - 2$. First, suppose that there exists a position $b \in [i + \tau - 1..i + 2\tau - 1] \cap B_k(T)$. Since $b - \tau + 1 \in [i..i + \tau]$ has not been added to $\mathbf{S}_{\text{sig}}(\tau, T)$, the fragment $T[b - \tau + 1..b + \tau]$ must be contained in a τ -run $T[p..q]$ that satisfies $p \leq b - \tau + 1 \leq i + \tau - 1$ and $q \geq b + \tau \geq i + 2\tau - 1$.

Next, suppose that $[i + \tau - 1..i + 2\tau - 1] \cap B_k(T) = \emptyset$. Then, $T[i + \tau - 1..i + 2\tau]$ is contained in a single phrase induced by T_k , and the length of this phrase is at least $\tau + 1$. If $k = 0$, this contradicts that all level-0 phrases are of length 1. Due to $\tau + 1 \geq 3\alpha_k + 1 > 3d_k$ (see Fact 7.13), so Fact 7.11 yields $\text{per}(T[i + \tau - 1..i + 2\tau]) \leq d_k \leq \alpha_k \leq \frac{1}{3}\tau$ (again by Fact 7.13). The run $T[p..q]$ extending $T[i + \tau - 1..i + 2\tau]$ satisfies $p \leq i + \tau - 1$ and $q \geq i + 2\tau$.

Note that $p - 1$ and $q - 2\tau + 2$ satisfy conditions (2) and (3), respectively. Due to $[i..i + \tau] \cap \mathbf{S}_{\text{sig}}(\tau, T) = \emptyset$, this implies $p \leq i$ and $q \geq i + 3\tau - 2$, which means that $\text{per}(T[i..i + 3\tau - 1]) \leq \frac{1}{3}\tau$ holds as claimed.

For the converse implication, note that if $s \in [i..i + \tau] \cap \mathbf{S}_{\text{sig}}(\tau, T)$, then $\text{per}(T[i..i + 3\tau - 1]) \geq \text{per}(T[s..s + 2\tau]) > \frac{1}{3}\tau$ because $T[s..s + 2\tau]$ is not contained in any τ -run (the latter observation is trivial if s satisfies condition (1); in the remaining two cases, it follows from the upper bound of $\frac{2}{3}\tau$ on the overlap of two τ -runs).

As for the size, observe that $s \in \mathbf{S}_{\text{sig}}(\tau, T)$ can be accounted to $s + \tau - 1 \in B_k(T)$ if it satisfies condition (1), to $\gamma \in \text{RUNS}_\tau(T)$ starting at position $s + 1$ if it satisfies condition (2), and to $\gamma \in \text{RUNS}_\tau(T)$ ending at position $s + 2\tau - 2$ if it satisfies condition (3). Since any two distinct τ -runs γ, γ' satisfy $|\gamma \cap \gamma'| \leq \frac{2}{3}\tau$, the number of positions $s \in \mathbf{S} \cap [i..i + \tau]$ satisfying (2) or (3) is $\mathcal{O}(1)$. By Lemma 7.12, the number of positions satisfying (1) is $\mathcal{O}(\frac{\tau}{d_k}) = \mathcal{O}(\frac{\alpha_k + 1}{d_k}) = \mathcal{O}(h_k) = \mathcal{O}(\log^*(k\sigma)) = \mathcal{O}(\log^*(\tau\sigma))$ by Facts 7.8 and 7.13. \square

An additional property of Construction 9.2 is consistency across different strings. The following observation can be proved by adapting the first paragraph of the proof of Lemma 9.3.

Observation 9.4. *Consider strings $T, T' \in \Sigma^+$ and positions $i \in [1..|T| - 2\tau + 1]$, $i' \in [1..|T'| - 2\tau + 1]$. If $T[i..i + 2\tau] = T'[i'..i' + 2\tau]$, then $i \in \mathbf{S}_{\text{sig}}(T, \tau)$ if and only if $i' \in \mathbf{S}_{\text{sig}}(\tau, T')$.*

Corollary 9.5. *Consider a string $T \in \Sigma^+$ and its substring $U = T[i..j]$. Then, $\mathbf{S}_{\text{sig}}(\tau, U) = \{s - i : s \in \mathbf{S}_{\text{sig}}(\tau, T) \cap (i..j - 2\tau + 1)\}$.*

Proposition 9.6. *The data structure of Section 8 can be extended so that, given $T \in \mathcal{W}$ and $\tau \in \mathbb{Z}_+$, the set $\text{RUNS}_\tau(T)$, with τ -runs ordered by their starting positions, can be constructed in $\mathcal{O}\left(\frac{|T| \log |T|}{\tau} (\log^*(|T|\sigma))^2 (\log^*(|T|\sigma) + \frac{\log^2 \log g}{\log \log \log g})\right)$ time.*

Proof. We partition T into blocks of length $\lfloor \frac{1}{3}\tau \rfloor$ (leaving up to $\lfloor \frac{1}{3}\tau \rfloor$ trailing characters behind). For any two consecutive blocks, we extract the corresponding fragment (using `split`) and apply `period` to retrieve its shortest period (provided that it does not exceed $\frac{1}{3}\tau$). If the period indeed does not exceed $\frac{1}{3}\tau$, we maximally extend the fragment while preserving its shortest period. This is implemented using an `lcp` query on two suffixes of T and an `lcs` queries on two prefixes of T . If the maximal fragment is of length at least τ , we include it in $\text{RUNS}_\tau(T)$. By the Periodicity Lemma, each τ -run is generated this way (perhaps multiple times). Moreover, if we process the block pairs in the left-to-right order, then the τ -runs are also generated in the left-to-right order. \square

Proposition 9.7. *The data structure of Section 8 can be extended so that, given $T \in \mathcal{W}$ and $\tau \in \mathbb{Z}_+$, the set $\mathbf{S}_{\text{sig}}(\tau, T)$ obtained using Construction 9.2 can be built in $\mathcal{O}\left(\frac{|T| \log |T|}{\tau} (\log^*(|T|\sigma))^2 \cdot (\log^*(|T|\sigma) + \frac{\log^2 \log g}{\log \log \log g})\right)$ time.*

Proof. We first compute the largest $k \in \mathbb{Z}_{\geq 0}$ with $2\tau \geq \alpha_k + \beta_k$ and construct T_k by traversing the parse tree $\mathcal{T}(\text{symb}_{\text{sig}}(T))$. In particular, this yields the set $B_k(T)$. Next, we generate $\text{RUNS}_\tau(T)$ using Proposition 9.6. To construct $\mathbf{S}_{\text{sig}}(\tau, T)$, we simultaneously scan $B_k(T)$ and $\text{RUNS}_\tau(T)$. For every position $i \in B_k(T) \cap [\alpha_k + 1..n - 2\tau - \alpha_k]$, we add $i - \tau + 1$ to $\mathbf{S}_{\text{sig}}(\tau, T)$ provided that $T[i - \tau..i - 2\tau]$ is not contained in any τ -run. Moreover, for every τ -run $T[p..q]$, we add to $\mathbf{S}_{\text{sig}}(\tau, T)$ positions $p - 1$ and $q - 2\tau + 2$ (provided that they are within $[1..n - 2\tau + 1]$). The query time is dominated by the cost of generating runs using Proposition 9.6. \square

10 Dynamic Text Implementation

In this section, we develop a data structure that maintains a dynamic text $T \in \Sigma^+$ subject to character insertions and deletions, as well as substring swaps (the ‘cut-paste’ operation), and supports queries specified in Assumptions 6.1, 6.3, 6.5, and 6.21.

For an alphabet Σ , we say that a *labelled string* over Σ is a string over $\Sigma \times \mathbb{Z}_{\geq 0}$. For $c := (a, \ell) \in \Sigma \times \mathbb{Z}_{\geq 0}$, we say that $\text{val}(c) := a$ is the *value* of c and $\text{label}(c) := \ell$ is the *label* of c . For a labelled string $S \in (\Sigma \times \mathbb{Z}_{\geq 0})^*$, we define the set of labels $L(S) = \{\text{label}(S[i]) : i \in [1..|S|]\}$ and the string of values $\text{val}(S) = \text{val}(S[1]) \cdots \text{val}(S[|S|])$.

Instead of maintaining a single labelled string representing T , our data structure internally allows maintaining multiple labelled strings (with character labels unique across the entire collection). This lets us decompose each update into smaller building blocks; for example, a `swap` (cut-paste) operation can be implemented using three splits followed by three concatenations. This internal interface

matches the setting considered in a large body of previous work on dynamic strings [MSU97, ABR00, GKK⁺18].

For a finite family $\mathcal{L} \subseteq (\Sigma \times \mathbb{Z}_{\geq 0})^*$, we set $L(\mathcal{L}) = \bigcup_{S \in \mathcal{L}} L(S)$ and $\|\mathcal{L}\| = \sum_{S \in \mathcal{L}} |S|$. We say \mathcal{L} is *uniquely labelled* if $|L(\mathcal{L})| = \|\mathcal{L}\|$; equivalently, for each label $\ell \in L(\mathcal{L})$ there exist unique $S \in \mathcal{L}$ and $i \in [1..|S|]$ such that $\ell = \text{label}(S[i])$.

Lemma 10.1. *There is a data structure maintaining a uniquely labelled family $\mathcal{L} \subseteq (\Sigma \times \mathbb{Z}_{\geq 0})^+$ using the following interface, where $\text{label}(S[1])$ is used as a reference to any string $S \in \mathcal{L}$:*

concat(R, S): *Given distinct $R, S \in \mathcal{L}$, set $\mathcal{L} := \mathcal{L} \setminus (\{R, S\}) \cup \{R \cdot S\}$.*

split(S, i): *Given $S \in \mathcal{L}$ and $i \in [1..|S|]$, set $\mathcal{L} := \mathcal{L} \setminus (\{S\}) \cup \{S[1..i], S(i..|S|)\}$.*

insert(a, ℓ): *Given $a \in \Sigma$ and $\ell \in \mathbb{Z}_{\geq 0} \setminus L(\mathcal{L})$, set $\mathcal{L} := \mathcal{L} \cup \{(a, \ell)\}$.*

delete(S): *Given $S \in \mathcal{L}$ with $|S| = 1$, set $\mathcal{L} := \mathcal{L} \setminus \{S\}$.*

label(S, i): *Given $S \in \mathcal{L}$ and $i \in [1..|S|]$, return $\text{label}(S[i])$.*

val(S, i): *Given $S \in \mathcal{L}$ and $i \in [1..|S|]$, return $\text{val}(S[i])$.*

unlabel(ℓ): *Given $\ell \in L(\mathcal{L})$, return (S, i) , where $S \in \mathcal{L}$ and $i \in [1..|S|]$ are such that $\ell = \text{label}(S[i])$.*

In the word RAM model with word size w satisfying $L(\mathcal{L}) \subseteq [0..2^w]$, each of these operations can be implemented in $\mathcal{O}(\log n)$ time, where $n = \|\mathcal{L}\|$.

Proof. Each string $S \in \mathcal{L}$ is stored as a balanced binary search tree (such as an AVL tree [AVL62]) whose in-order traversal yields S . Additionally, each node of the BST is augmented with the size of its subtree and the label of the leftmost node. We also maintain a node dictionary that maps each label $\ell \in L(\mathcal{L})$ to the node representing the character with label ℓ .

The **unlabel**(ℓ) operation uses the node dictionary to reach the node representing the character with label ℓ , and then traverses the path from ℓ to the root of the corresponding tree, using the subtree sizes to determine the number of nodes to the left of the path traversed. The remaining operations use the same approach to locate the roots of the trees representing strings $S \in \mathcal{L}$ represented by $\text{label}(S[1])$. Then, the **val** and **label** operations traverse the tree, using the subtree sizes to descend to the i th leftmost node, and they return the value and the label, respectively, of the corresponding character. The **split** operation also descends to the i th node of the tree representing S , but then it splits the tree. The **concat** operation joins two trees. The **delete** operation deletes a single-element tree and removes the corresponding entry from the node dictionary. The **insert** operation creates a single-element tree and inserts a new entry to the node dictionary. \square

As a warm-up, we show that Lemma 10.1 allows for efficient random access to a dynamic text.

Corollary 10.2. *A dynamic text $T \in \Sigma^n$ can be implemented so that initialization takes $\mathcal{O}(1)$ time, updates take $\mathcal{O}(\log n)$ time, and the following **access**(i) queries take $\mathcal{O}(\log n)$ time:*

access(i): *given $i \in [1..n]$, return $T[i]$.*

Proof. We maintain a uniquely labelled string family \mathcal{L} that satisfies the following invariant after each update is completed: $\mathcal{L} = \{S\}$ for a labelled string S such that $\text{val}(S) = T$. We also store a count c of the insertions performed so that we can assign a fresh label to every inserted character, and a label $\ell := \text{label}(S[1])$ needed to access the only string in \mathcal{L} .

To implement **initialize**(σ), we initialize an empty labelled family \mathcal{L} over $\Sigma = [0..\sigma)$ and perform an $\mathcal{L}.\text{insert}(\$, 0)$ operation to make sure that $\mathcal{L} = \{S\}$ with $\text{val}(S) = \$$. We also set $c := 0$ and $\ell := 0$.

As for the $T.\text{insert}(i, a)$ operation, we increment the counter c and perform the $\mathcal{L}.\text{insert}(a, c)$ operation adding to \mathcal{L} a new labelled string R with $\text{val}(R) = a$. If $i = 1$, we simply perform

$\mathcal{L}.\text{concat}(\ell, c)$ that results in $\mathcal{L} = \{R \cdot S\}$, and we update the label of the first character of the only string in \mathcal{L} , setting $\ell := c$. If $i > 1$, on the other hand, we retrieve the label $\ell_i := \mathcal{L}.\text{label}(\ell, i)$ of $S[i]$, and perform the following calls: $\mathcal{L}.\text{split}(\ell, i-1)$ (resulting in $\mathcal{L} = \{S[1..i], S[i..n], R\}$), $\mathcal{L}.\text{concat}(\ell, c)$ (resulting in $\mathcal{L} = \{S[1..i] \cdot R, S[i..n]\}$), and $\mathcal{L}.\text{concat}(\ell, \ell_i)$ (resulting in $\mathcal{L} = \{S[1..i] \cdot R \cdot S[i..n]\}$).

As for the $T.\text{delete}(i)$ operation, we retrieve the label $\ell_i := \mathcal{L}.\text{label}(\ell, i)$ of the character to be deleted and the label $\ell_{i+1} := \mathcal{L}.\text{label}(\ell, i+1)$ of the subsequent character. Next, we perform the $\mathcal{L}.\text{split}(\ell, i)$ operation that results in $\mathcal{L} = \{S[1..i], S(i..n)\}$. If $i = 1$, we then set $\ell := \ell_{i+1}$ because $S[1..i] = S[i]$ and $S(i..n) = S[1..i] \cdot S(i..n)$. Otherwise, we perform the $\mathcal{L}.\text{split}(\ell, i-1)$ operation (resulting in $\mathcal{L} = \{S[1..i], S[i], S(i..n)\}$) and the $\mathcal{L}.\text{concat}(\ell, \ell_{i+1})$ operation (resulting in $\mathcal{L} = \{S[1..i] \cdot S(i..n), S[i]\}$). In both cases, we conclude with a call $\mathcal{L}.\text{delete}(\ell_i)$ that removes $S[i]$ from \mathcal{L} and results in desired state $\mathcal{L} = \{S[1..i] \cdot S(i..n)\}$.

As for the $T.\text{swap}(i, j, k)$ operation, we first check whether $i < j < k$; otherwise, there is nothing to do. Next, we retrieve the labels $\ell_i := \mathcal{L}.\text{label}(\ell, i)$, $\ell_j := \mathcal{L}.\text{label}(\ell, j)$, and $\ell_k := \mathcal{L}.\text{label}(\ell, k)$ of $S[i]$, $S[j]$, and $S[k]$, respectively. Then, we perform the $\mathcal{L}.\text{split}(\ell, k-1)$ operation (resulting in $\mathcal{L} = \{S[1..k], S[k..n]\}$), the $\mathcal{L}.\text{split}(\ell, j-1)$ operation (resulting in $\mathcal{L} = \{S[1..j], S[j..k], S[k..n]\}$), and the $\mathcal{L}.\text{concat}(\ell, \ell_k)$ operation (resulting in $\mathcal{L} = \{S[1..j] \cdot S[k..n], S[j..k]\}$). If $i = 1$, we proceed with the $\mathcal{L}.\text{concat}(\ell_j, \ell)$ operation (resulting in $\mathcal{L} = \{S[j..k] \cdot S[1..j] \cdot S[k..n]\} = \{S[1..i] \cdot S[j..k] \cdot S[i..j] \cdot S[k..n]\}$ and set $\ell := \ell_j$ to update the label of the first character of the only string in \mathcal{L} .) Otherwise, we perform the $\mathcal{L}.\text{split}(\ell, i-1)$ operation (resulting in $\mathcal{L} = \{S[1..i], S[i..j] \cdot S[k..n], S[j..k]\}$), the $\mathcal{L}.\text{concat}(\ell, \ell_j)$ operation (resulting in $\mathcal{L} = \{S[1..i] \cdot S[j..k], S[i..j] \cdot S[k..n]\}$), and finally the $\mathcal{L}.\text{concat}(\ell, \ell_i)$ operation (resulting in $\mathcal{L} = \{S[1..i] \cdot S[j..k] \cdot S[i..j] \cdot S[k..n]\}$).

Finally, the $T.\text{access}(i)$ operation simply queries $\mathcal{L}.\text{access}(\ell, i)$ and forwards the obtained answer.

It is easy to see that the initialization takes $\mathcal{O}(1)$ time whereas the remaining operations take $\mathcal{O}(\log n)$ time. \square

10.1 Implementing Assumption 6.1

For a labelled family \mathcal{L} and an integer $q \in \mathbb{Z}_+$, we define

$$\text{CF}_q(\mathcal{L}) = \bigcup_{S \in \mathcal{L}} \{(\text{val}(S)^\infty[j..j+q], \text{label}(S[j])) : j \in [1..|S|]\}$$

Lemma 10.3. *For any fixed $q \in \mathbb{Z}_+$, the data structure of Lemma 10.1 can be augmented so that, given $r \in [1..|\mathcal{L}|]$, the r th smallest element $(X, \ell) \in \text{CF}_q(\mathcal{L})$ can be computed in $\mathcal{O}(q \log n)$ time, along with the counts $|\{(X', \ell') \in \text{CF}_q(\mathcal{L}) : X' \prec X\}|$ and $|\{(X', \ell') \in \text{CF}_q(\mathcal{L}) : X' \preceq X\}|$. This comes at the price of increasing the cost of all updates to $\mathcal{O}(q^2 \log n)$.*

Proof. We explicitly store $\text{CF}_q(\mathcal{L})$ in a balanced binary search tree (ordered lexicographically) and a dictionary that maps each label $\ell \in L(\mathcal{L})$ to the corresponding node $(X, \ell) \in \text{CF}_q(\mathcal{L})$. With this implementation, a query takes $\mathcal{O}(q \log n)$ time ($\mathcal{O}(\log n)$ comparisons in $\mathcal{O}(q)$ time each).

As for the $\text{delete}(S)$ operation, we just remove from $\text{CF}_q(\mathcal{L})$ the corresponding pair $(S^q, \text{label}(S))$, which takes $\mathcal{O}(q \log n)$ time. Symmetrically, $\text{insert}(a, \ell)$ inserts (a^q, ℓ) in $\mathcal{O}(q \log n)$ time.

The $\text{concat}(R, S)$ operation is implemented as follows: For each $j \in (\max(0, |R| - q + 1) .. |R|]$, we replace $(\text{val}(R)^\infty[j..j+q], \text{label}(R[j]))$ with $(\text{val}(RS)^\infty[j..j+q], \text{label}(R[j]))$ and, analogously, for each $j \in (\max(0, |S| - q + 1) .. |S|]$, we replace $(\text{val}(S)^\infty[j..j+q], \text{label}(S[j]))$ with $(\text{val}(RS)^\infty[j +$

$|R| \dots j + |R| + q$, $\text{label}(S[j])$). Each new entry can be constructed in $\mathcal{O}(q \log n)$ time using q calls to the val and label operations, for the overall running time of $\mathcal{O}(q^2 \log n)$.

The implementation of $\text{split}(S, i)$ is symmetric. For each $j \in (\max(0, i - q + 1) \dots i]$, we replace $(\text{val}(S)^\infty[j \dots j + q], \text{label}(S[j]))$ with $(\text{val}(S[1 \dots i])^\infty[j \dots j + q], \text{label}(S[j]))$, and, analogously, for each $j \in (\max(i, |S| - q + 1) \dots |S|]$, we replace $(\text{val}(S)^\infty[j \dots j + q], \text{label}(S[j]))$ with $(\text{val}(S(i \dots |S|))^\infty[j - i \dots j - i + q], \text{label}(S[j]))$. \square

Proposition 10.4. *A dynamic text $T \in \Sigma^n$ can be implemented so that initialization takes $\mathcal{O}(1)$ time, updates take $\mathcal{O}(\log n)$ time, and the queries of Assumption 6.1 take $\mathcal{O}(\log n)$ time.*

Proof. We proceed as in the proof of Corollary 10.2, but instead of implementing the uniquely labelled family \mathcal{L} using the vanilla version of Lemma 10.1, we apply the extension of Lemma 10.3 for $q = 16$. Due to $q = \mathcal{O}(1)$, this preserves the update times.

As for the query, we forward the argument i to the query of Lemma 10.3. This results in the i th smallest pair (X, ℓ) in the set $\{T^\infty[j \dots j + 16], \text{label}(S[j]) : j \in [1 \dots n]\}$, as well as the counts $|\{j \in [1 \dots n] : T^\infty[j \dots j + 16] \prec X\}|$ and $|\{j \in [1 \dots n] : T^\infty[j \dots j + 16] \preceq X\}|$. By Corollary 6.8, we have $X = T^\infty[\text{SA}[i] \dots \text{SA}[i] + 16]$, and thus the counts are equal to $\text{RangeBeg}_{16}(\text{SA}[i])$ and $\text{RangeEnd}_{16}(\text{SA}[i])$, respectively. Moreover, a position $j \in \text{Occ}_{16}(\text{SA}[i])$ can be retrieved using an $\text{unlabel}(\ell)$ call. \square

10.2 Common Tools for Assumptions 6.3, 6.5, and 6.21

For a family $\mathbf{L} \subseteq L(\mathcal{L})$ and a label $\ell \in \mathcal{L}$, we define $\text{pred}_{\mathbf{L}}(\ell)$ and $\text{succ}_{\mathbf{L}}(\ell)$ as follows. Let $(S, i) = \text{unlabel}(\ell)$. If $L(S[1 \dots i]) \cap \mathbf{L} = \emptyset$, then we set $\text{pred}_{\mathbf{L}}(\ell) = \perp$. Otherwise, $\text{pred}_{\mathbf{L}}(\ell) = \text{label}(S[\max\{t \in [1 \dots i] : \text{label}(S[t]) \in \mathbf{L}\}])$. Symmetrically, if $L(S[i \dots |S|]) \cap \mathbf{L} = \emptyset$, then we set $\text{succ}_{\mathbf{L}}(\ell) = \perp$. Otherwise, $\text{succ}_{\mathbf{L}}(\ell) = \text{label}(S[\min\{t \in [i \dots |S|] : \text{label}(S[t]) \in \mathbf{L}\}])$.

Lemma 10.5. *The data structure of Lemma 10.1 can be augmented to maintain a set $\mathbf{L} \subseteq L(\mathcal{L})$ of marked labels so that, given $\ell \in L(\mathcal{L})$, the values $\text{pred}_{\mathbf{L}}(\ell)$ and $\text{succ}_{\mathbf{L}}(\ell)$ can be computed in $\mathcal{O}(\log n)$ time. Moreover, marking and unmarking a label $\ell \in L(\mathcal{L})$ also costs $\mathcal{O}(\log n)$ time.*

Proof. Compared to the implementation in the proof of Lemma 10.1, each node ν stores two extra bits, specifying whether the corresponding label is marked and whether the subtree of ν contains a node with a marked label.

To determine the predecessor of $\text{pred}_{\mathbf{L}}(\ell)$, we first locate the node ν with label ℓ . Then, we check whether $\ell \in \mathbf{L}$; if so, we simply return ℓ . Otherwise, we traverse the path from ν the root of the corresponding tree. For ν as well as for each node that we reach from its left subtree, we check whether the right subtree contains a node with a marked label node. If so, we descend to the leftmost such node and return its label. If our traversal reaches the root without success, we report that $\text{pred}_{\mathbf{L}}(\ell) = \perp$. It is easy to see that this procedure is correct and takes $\mathcal{O}(\log n)$ time. A symmetric procedure computes $\text{succ}_{\mathbf{L}}(\ell)$.

As for marking and unmarking, we use the node dictionary to locate the node with label ℓ , and update its status with respect to marking. Then, we update the cumulative bits on the path towards the root; this costs $\mathcal{O}(\log n)$ time. As for the remaining updates, we recompute the cumulative bits for all nodes visited; this does not increase the asymptotic time of these updates. \square

Based on the function $\text{CF}^+(\cdot)$ defined for unlabeled string families in Section 8.7, let us denote $\text{CF}^+(\mathcal{L}) = \text{CF}^+(\{\text{val}(S) : S \in \mathcal{L}\})$.

Lemma 10.6. *The data structure of Lemma 10.1 can be augmented so that any two fragments in $\text{CF}^+(\mathcal{L})$ and any two fragments in $\text{CF}^+(\overline{\mathcal{L}})$ can be compared lexicographically in $\mathcal{O}(\log n \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$ time, where $m = \sigma + t$ and t is the total number of instruction that the data structure has performed so far. This comes at the price of increasing the cost of `concat` to $\mathcal{O}(\log n \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$, `split` to $\mathcal{O}(\log n \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$, and `insert` to $\mathcal{O}(\frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$.*

Proof. On top of the data structure of Lemma 10.1, we also store $\mathcal{W} := \{\text{val}(S) : S \in \mathcal{L}\}$ using dynamic strings of Section 8. The `insert`(a, ℓ) operation adds a new string a to \mathcal{W} using $\mathcal{W}.\text{insert}(a)$. The `split`(S, i) operation adds strings $\text{val}(S)[1..i]$ and $\text{val}(S)(i..|S|)$ to \mathcal{W} using $\mathcal{W}.\text{split}(\text{val}(S), i)$. The `concat`(R, S) operation adds the string $\text{val}(R) \cdot \text{val}(S)$ to \mathcal{W} using $\mathcal{W}.\text{concat}(\text{val}(R), \text{val}(S))$. As for lexicographic comparisons, we use the `compare` operation implemented in Section 8.7. \square

10.3 Implementing Assumption 6.3

Recall that, for a string T and an integer $\tau \in \mathbb{Z}_+$, we defined $\text{R}(\tau, T) = \{i \in [1..|T| - 3\tau + 2] : \text{per}(T[i..i + 3\tau - 2]) \leq \frac{1}{3}\tau\}$ and $\text{R}'(\tau, T) = \{i \in \text{R}(\tau, T) : i - 1 \notin \text{R}(\tau, T)\}$. We also define a symmetric set $\text{R}''(\tau, T) = \{i \in \text{R}(\tau, T) : i + 1 \notin \text{R}(\tau, T)\}$. For a uniquely labelled family \mathcal{L} , we generalize these notions as follows:

$$\begin{aligned} \text{R}(\tau, \mathcal{L}) &= \{\text{label}(S[i]) : S \in \mathcal{L} \text{ and } i \in \text{R}(\tau, \text{val}(S))\} \\ \text{R}'(\tau, \mathcal{L}) &= \{\text{label}(S[i]) : S \in \mathcal{L} \text{ and } i \in \text{R}'(\tau, \text{val}(S))\} \\ \text{R}''(\tau, \mathcal{L}) &= \{\text{label}(S[i]) : S \in \mathcal{L} \text{ and } i \in \text{R}''(\tau, \text{val}(S))\}. \end{aligned}$$

Lemma 10.7. *For any fixed $\tau \in \mathbb{Z}_+$, the data structure of Lemma 10.6 can be augmented so that, given $\ell \in L(\mathcal{L})$, we can check in $\mathcal{O}(\log n)$ time whether $\ell \in \text{R}(\tau, \mathcal{L})$. This comes at the price of increasing the cost of `concat` to $\mathcal{O}(\log n \frac{\log^2 \log m \cdot (\log^* m)^2}{\log \log \log m})$.*

Proof. On top of the data structure of Lemma 10.6, we store two instances of the component of Lemma 10.5, for $\text{R}'(\tau, \mathcal{L})$ and $\text{R}''(\tau, \mathcal{L})$, respectively.

In the query algorithm, we first find $\ell' := \text{pred}_{\text{R}'(\tau, \mathcal{L})}(\ell)$. If $\ell' = \perp$, we report that $\ell \notin \text{R}(\tau, \mathcal{L})$. Otherwise, we find $\ell'' := \text{succ}_{\text{R}''(\tau, \mathcal{L})}(\ell')$ (it is never \perp), and we use the `unlabel`(ℓ), `unlabel`(ℓ'), and `unlabel`(ℓ'') operations to determine the positions j, j' , and j'' corresponding to these labels (these are positions in the same string $S \in \mathcal{L}$). We report that $\ell \in \text{R}(\tau, \mathcal{L})$ if and only if $j' \leq j \leq j''$.

After executing the `concat`(R, S) operation of Lemma 10.6, we perform the following steps. First, we remove `label`($S[1]$) from $\text{R}'(\tau, \mathcal{L})$ and `label`($R[|R|]$) from $\text{R}''(\tau, \mathcal{L})$ (if present in the respective sets). Next, we compute the τ -runs in $\text{val}(U)$, where $U = R(\max(0, |R| - 3\tau + 1) .. |R|) \cdot S[1 .. \min(|S|, 3\tau - 1)]$. For this, we use `construct` $\text{val}(U)$ using `concat` and `split` operations on \mathcal{W} , and then run the algorithm of Proposition 9.6 on $\text{val}(U)$. We iterate over fragments $(R \cdot S)[x..y]$ corresponding to τ -runs of length at least $3\tau - 1$ in $\text{val}(U)$. For each such fragment, we add `label`(($R \cdot S$)[x]) to $\text{R}'(\tau, \mathcal{L})$ if $x > |R| - 3\tau + 2$, and we add `label`(($R \cdot S$)[y]) to $\text{R}''(\tau, \mathcal{L})$ if $y \leq |R| + 3\tau - 2$. The number of newly marked labels is $\mathcal{O}(1)$, so the extra cost is $\mathcal{O}(\log n \cdot \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$ time, dominated by the procedure of Proposition 9.6.

Before executing the `split`(S, i) operation of Lemma 10.6, we perform the following steps. First, we run the query algorithm to check whether `label`($S[i]$) $\in \text{R}(\tau, \mathcal{L})$ and `label`($S[i + 1]$) $\in \text{R}(\tau, \mathcal{L})$. If `label`($S[i]$) $\in \text{R}(\tau, \mathcal{L})$, we add `label`($S[i]$) to $\text{R}''(\tau, \mathcal{L})$, and if `label`($S[i + 1]$) $\in \text{R}(\tau, \mathcal{L})$, we add `label`($S[i + 1]$) $\in \text{R}'(\tau, \mathcal{L})$. Then, we remove from $\text{R}'(\tau, \mathcal{L})$ all labels ℓ with `unlabel`(ℓ) = (S, j) for

$j \in (i - 3\tau + 2 \dots i]$, and from $R''(\tau, \mathcal{L})$ all labels ℓ with $\text{unlabel}(\ell) = (S, j)$ for $j \in (i - 3\tau + 2 \dots i]$ for $j \in (i \dots i + 3\tau - 2]$. The labels to be removed are listed with $\mathcal{O}(\log n)$ -time delay using predecessor queries on $R'(\tau, \mathcal{L})$ and successor queries on $R''(\tau, \mathcal{L})$, respectively. The number of newly unmarked labels is $\mathcal{O}(1)$, so the extra cost compared to Lemma 10.6 is $\mathcal{O}(\log n)$, dominated by the original cost of $\mathcal{W}.\text{split}(S, i)$. \square

Proposition 10.8. *For any fixed $\ell \in \mathbb{Z}_+$, a dynamic text $T \in \Sigma^n$ can be implemented so that initialization takes $\mathcal{O}\left(\frac{\log^2 \log m \cdot \log^* m}{\log \log \log m}\right)$ time, updates take $\mathcal{O}\left(\log n \cdot \frac{\log^2 \log m \cdot (\log^* m)^2}{\log \log \log m}\right)$ time, and the queries of Assumption 6.3 take $\mathcal{O}(\log n)$ time, where $m = |\Sigma| + t$ and t is the total number of instructions that the data structure has performed so far.*

Proof. We proceed as in the proof of Corollary 10.2, but instead of implementing family \mathcal{L} using Lemma 10.1, we apply the extension of Lemma 10.7 for $\tau = \lfloor \frac{\ell}{3} \rfloor$. This increases the cost of initialize to $\mathcal{O}\left(\frac{\log^2 \log m \cdot \log^* m}{\log \log \log m}\right)$ and the cost of updates to $\mathcal{O}\left(\log n \cdot \frac{\log^2 \log m \cdot (\log^* m)^2}{\log \log \log m}\right)$ (dominated by $\mathcal{L}.\text{concat}$).

As for a query, note that $\mathcal{L} = \{S\}$, where S is a labelled string with $\text{val}(S) = T$. Given a position $i \in [1 \dots |T|]$, we compute $\text{label}(S[i])$ using a call $\mathcal{L}.\text{label}(S, i)$. Next, we apply the query algorithm of Lemma 10.7 to check whether $\text{label}(S[i]) \in R(\tau, \mathcal{L})$, which is equivalent to testing whether $i \in R(\tau, T)$. The query time is therefore $\mathcal{O}(\log n)$. \square

10.4 Implementing Assumption 6.5

Based on Construction 9.2, we define $\mathbf{S}_{\text{sig}}(\tau, \mathcal{L}) = \bigcup_{S \in \mathcal{L}} \{\text{label}(S[i]) : i \in \mathbf{S}_{\text{sig}}(\tau, \text{val}(S))\}$.

Lemma 10.9. *For any fixed $\tau \in \mathbb{Z}_{\geq 0}$, the data structure of Lemma 10.6 can be augmented so that, given $S \in \mathcal{L}$ and $j \in [1 \dots |S|]$, the value $\text{succ}_{\mathbf{S}_{\text{sig}}(\tau, \text{val}(S))}(j)$ can be computed in $\mathcal{O}(\log n)$ time, where sig is an (implicit) signature function. This comes at the price of increasing the cost of concat to $\mathcal{O}\left(\log n \cdot \frac{\log^2 \log m \cdot (\log^* m)^2}{\log \log \log m}\right)$.*

Proof. On top of the data structure of Lemma 10.6, we store the component of Lemma 10.5 with $\mathbf{L} := \mathbf{S}_{\text{sig}}(\tau, \mathcal{L})$. As for the query algorithm, we compute $\ell = \text{label}(S[j])$ and find $\ell' = \text{succ}_{\mathbf{L}}(\ell)$. If $\ell' = \perp$, then $\text{succ}_{\mathbf{S}_{\text{sig}}(\tau, \text{val}(S))}(j) = |S| - 2\tau + 2$ (see Section 6.2). Otherwise, $\text{succ}_{\mathbf{S}_{\text{sig}}(\tau, \text{val}(S))}(j) = j'$, where $(S, j') = \text{unlabel}(\ell')$. Thus, the query time is $\mathcal{O}(\log n)$.

While executing the $\text{split}(S, i)$ operation, we unmark all nodes corresponding to $S[i - 2\tau + 2 \dots i]$. For this, we repeatedly compute $\text{pred}_{\mathbf{S}_{\text{sig}}(\tau, \mathcal{L})}(\text{label}(S[i]))$ and, if the predecessor exists, use the unlabel operation to retrieve its position j . If $j \geq i - 2\tau + 2$, we unmark the corresponding label. Otherwise, we terminate the procedure. By Corollary 9.5, this correctly maintains the set of marked nodes. The running time of this step is $\mathcal{O}(\log n)$ per unmarked node and, by Lemma 9.3, $\mathcal{O}(\log n \log^*(\tau\sigma))$ in total. This cost is dominated by the cost $\mathcal{O}\left(\log n \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m}\right)$ of $\text{split}(\text{val}(S), i)$.

While executing the $\text{concat}(R, S)$ operation, by Corollary 9.5, we need to mark $\mathbf{S}_{\text{sig}}(U, \tau)$ in the tree representing $R \cdot S$, where $U = R(\max(0, |R| - 2\tau + 1) \dots |R|) \cdot S[1 \dots \min(2\tau, |S|)]$. For this, we construct $\text{val}(U)$ using split and concat operations on \mathcal{W} , and then apply Proposition 9.7 to derive $\mathbf{S}_{\text{sig}}(\text{val}(U), \tau)$. For each of the obtained positions, we locate the corresponding label using the $\text{label}(R \cdot S, j)$. The cost $\mathcal{O}\left(\log n \frac{\log^2 \log m \cdot (\log^* m)^2}{\log \log \log m}\right)$ is dominated by the application of Proposition 9.7. \square

For a uniquely labelled family \mathcal{L} , an integer $\tau \in \mathbb{Z}_+$, and a signature function sig , define

$$\mathcal{P}_{\text{sig}}(\tau, \mathcal{L}) = \bigcup_{S \in \mathcal{L}} \{(\overline{\text{val}(S)^\infty[i - 7\tau .. i]}, \text{val}(S)^\infty[i .. i + 7\tau], \text{label}(S[i])) : S \in \mathcal{S}_{\text{sig}}(\tau, \text{val}(S))\}$$

(cf. Definition 4.2). We interpret $\mathcal{P}_{\text{sig}}(\tau, \mathcal{L}) \subseteq \mathcal{X} \times \mathcal{Y} \times \mathbb{Z}$ as a family of labelled points with $\mathcal{X} = \text{CF}^+(\overline{\mathcal{L}})$ and $\mathcal{Y} = \text{CF}^+(\mathcal{L})$, both ordered lexicographically.

Lemma 10.10. *The data structure of Lemma 10.9 can be augmented so that range queries (Section 4) on $\mathcal{P}_{\text{sig}}(\tau, \mathcal{L})$ can be supported in time $\mathcal{O}(\log^3 n + \log^2 n \cdot \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$. This comes at the price of increasing the cost of `concat` and `split` to $\mathcal{O}(\log^2 n \cdot \frac{\log^2 \log m \cdot (\log^* m)^2}{\log \log \log m})$ time.*

Proof. Compared to the data structure of Lemma 10.9, we also maintain $\mathcal{P}_{\text{sig}}(\tau, \mathcal{L})$ in the data structure of Theorem 4.1. By Lemma 10.6, each comparison on \mathcal{X} and \mathcal{Y} costs $\mathcal{O}(\log n \cdot \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$ time. As a result, the cost of a range query is $\mathcal{O}(\log^3 n + \log^2 n \cdot \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$. As for updates, `insert` and `delete` do not incur updates to \mathcal{P} (strings of length 1 have empty synchronizing sets). On the other hand, `split` and `concat` may incur $\mathcal{O}(\log^* m)$ insertions and deletions, both corresponding to changes in the synchronizing set and to synchronizing positions whose context has changed; the latter can be generated by repeated calls to the $\text{succ}_{\mathcal{S}_{\text{sig}}(\tau, \mathcal{L})}$ operation. The running time is therefore increased to $\mathcal{O}(\log^2 n \cdot \frac{\log^2 \log m \cdot (\log^* m)^2}{\log \log \log m})$. \square

Proposition 10.11. *For any fixed $\ell \in \mathbb{Z}_+$, a dynamic text $T \in \Sigma^n$ can be implemented so that initialization takes $\mathcal{O}(\frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$ time, updates take $\mathcal{O}(\log^2 n \cdot \frac{\log^2 \log m \cdot (\log^* m)^2}{\log \log \log m})$ time, and the queries of Assumption 6.5 take $\mathcal{O}(\log^3 n + \log^2 n \cdot \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$ time, where $m = |\Sigma| + t$ and t is the total number of instructions that the data structure has performed so far.*

Proof. We proceed as in the proof of Corollary 10.2, but instead of implementing family \mathcal{L} using Lemma 10.1, we apply the extensions of Lemmas 10.9 and 10.10 for $\tau = \lfloor \frac{\ell}{3} \rfloor$. This increases the cost of `initialize` to $\mathcal{O}(\frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$ and the cost of updates to $\mathcal{O}(\log^2 n \cdot \frac{\log^2 \log m \cdot (\log^* m)^2}{\log \log \log m})$ (dominated by $\mathcal{L}.\text{concat}$ and $\mathcal{L}.\text{split}$).

We use $\mathcal{S} := \mathcal{S}_{\text{sig}}(\tau, T)$ as the synchronizing set of T . As for a $\text{succ}_{\mathcal{S}}(i)$ query for $i \in [1 .. n - 3\tau + 1]$, we simply forward the query to the component of Lemma 10.9. The cost of this query is $\mathcal{O}(\log n)$.

As for the string-string range queries on $\text{Points}_{7\tau}(T, \mathcal{S})$, we use the equivalent range queries on $\mathcal{P}_{\text{sig}}(\tau, \mathcal{L})$ instead. The only work needed is to convert the query arguments from indices to fragments in $\text{CF}^+(T)$ and $\text{CF}^+(\overline{T})$ (as specified in Problem 4.3). Moreover, the label returned by the range selection on $\mathcal{P}_{\text{sig}}(\tau, \mathcal{L})$ is converted to a position in T using the $\mathcal{L}.\text{unlabel}(\cdot)$ operation. The query time is $\mathcal{O}(\log^3 n + \log^2 n \cdot \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$, dominated by the cost of range queries of Lemma 10.10. \square

10.5 Implementing Assumption 6.21

Lemma 10.12. *For any fixed $\tau \in \mathbb{Z}_+$, the data structure of Lemma 10.7 can be augmented so that, given $S \in \mathcal{L}$ and $j \in \text{R}(\tau, \text{val}(S))$, in $\mathcal{O}(\log n)$ time we can compute $e(\tau, \text{val}(S), j)$, $|\text{root}_{f_{\text{sig}}}(\tau, \text{val}(S), j)|$, and $\text{head}_{f_{\text{sig}}}(\tau, \text{val}(S), j)$, where f_{sig} is the necklace-consistent function defined in Construction 8.2. This comes at the price of increasing the cost of `split` and `concat` to $\mathcal{O}(\log n \cdot \frac{\log^2 \log m \cdot (\log^* m)^2}{\log \log \log m})$.*

Proof. On top of the data structure of Lemma 10.7, for each $\ell \in R'(\tau, \mathcal{L})$, we store $|\text{root}_{f_{\text{sig}}}(\tau, \text{val}(S), j)|$ and $\text{head}_{f_{\text{sig}}}(\tau, \text{val}(S), j)$, where $(S, j) = \text{unlabel}(\ell)$.

In the query algorithm, we first compute $\ell = \text{label}(S[j])$. Then, we find $\ell' = \text{pred}_{R'(\tau, \mathcal{L})}(\ell)$ and $\ell'' = \text{succ}_{R''(\tau, \mathcal{L})}(\ell')$, and we use the $\text{unlabel}(\ell')$ and $\text{unlabel}(\ell'')$ operations to retrieve the corresponding positions j' and j'' in S . We note that $e(\tau, \text{val}(S), j) = e(\tau, \text{val}(S), j') = j'' + 3\tau - 2$. Additionally, we retrieve the stored values $p := |\text{root}_{f_{\text{sig}}}(\tau, \text{val}(S), j')|$ and $h := \text{head}_{f_{\text{sig}}}(\tau, \text{val}(S), j')$, and we return $|\text{root}_{f_{\text{sig}}}(\tau, \text{val}(S), j)| = p$ and $\text{head}_{f_{\text{sig}}}(\tau, \text{val}(S), j) = (h + j' - j) \bmod p$. The correctness of this procedure follows from Lemma 6.19, and the running time is clearly $\mathcal{O}(\log n)$.

In order to maintain the extra information stored for $\ell \in R'(\tau, \mathcal{L})$, we execute the following procedure whenever a label ℓ is added to $R'(\tau, \mathcal{L})$: first, we determine $(S, j) = \text{unlabel}(\ell)$. Next, we extract $\text{val}(S)[j..j + 3\tau - 1]$ using $\mathcal{W}.\text{split}$ operations, compute $p := \text{period}(\text{val}(S)[j..j + 3\tau - 1])$, and note that $|\text{root}_{f_{\text{sig}}}(\tau, \text{val}(S), j)| = p$. Then, we extract $\text{val}(S)[j..j + p]$ using another $\mathcal{W}.\text{split}$ operation, compute $h = \mathcal{W}.\text{canShift}(\text{val}(S)[j..j + p])$, and note that $\text{head}_{f_{\text{sig}}}(\tau, \text{val}(S), j) = h \bmod p$. Since $\text{split}(S, i)$ and $\text{concat}(R, S)$ involve $\mathcal{O}(1)$ insertions to $R'(\tau, \mathcal{L})$, the extra cost of these operations is $\mathcal{O}(\log n \cdot \frac{\log^2 \log m \cdot (\log^* m)^2}{\log \log \log m})$. The correctness follows from the fact that $\ell = \text{label}(S[j])$ is deleted from $R'(\tau, \mathcal{L})$ whenever $\text{split}(S, i)$ with $i \in [j..j + 3\tau - 2]$ is performed. \square

For a uniquely labelled family \mathcal{L} , an integer $\tau \in \mathbb{Z}_+$, a string H , and a necklace-consistent function f , we define (cf. Definitions 4.4 and 6.20):

$$\begin{aligned} \mathcal{P}_{f,H}^-(\tau, \mathcal{L}) &= \bigcup_{S \in \mathcal{L}} \{(d, \text{val}(S)^\infty[j..j + 7\tau], \text{label}(S[j])) : (j, d) \in E_{f,H}^-(\tau, \text{val}(S))\}, \\ \mathcal{P}_{f,H}^+(\tau, \mathcal{L}) &= \bigcup_{S \in \mathcal{L}} \{(d, \text{val}(S)^\infty[j..j + 7\tau], \text{label}(S[j])) : (j, d) \in E_{f,H}^+(\tau, \text{val}(S))\}, \\ \mathcal{I}_{f,H}^-(\tau, \mathcal{L}) &= \bigcup_{S \in \mathcal{L}} \{(a, b, \text{label}(S[j])) : (a, b, j) \in \mathcal{I}_{f,H}^-(\tau, \text{val}(S))\}, \\ \mathcal{I}_{f,H}^+(\tau, \mathcal{L}) &= \bigcup_{S \in \mathcal{L}} \{(a, b, \text{label}(S[j])) : (a, b, j) \in \mathcal{I}_{f,H}^+(\tau, \text{val}(S))\}. \end{aligned}$$

We interpret $\mathcal{P}_{f,H}^\pm(\tau, \mathcal{L})$ as a family of labelled points with $\mathcal{X} = \mathbb{Z}$ and $\mathcal{Y} = \text{CF}^+(\mathcal{L})$.

Lemma 10.13. *The data structure of Lemma 10.12 can be augmented so that, for each string H , given as a fragment of $\text{val}(S)$ for some $S \in \mathcal{L}$, range queries (Section 4) on $\mathcal{P}_{f_{\text{sig}},H}^\pm(\tau, \mathcal{L})$ and modular constraint queries (Section 5) on $\mathcal{I}_{f_{\text{sig}},H}^\pm(\tau, \mathcal{L})$ can be supported in time $\mathcal{O}(\log^3 n + \log^2 n \cdot \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$. This comes at the price of increasing the cost of concat and split to $\mathcal{O}(\log^2 n \cdot \frac{\log^2 \log m \cdot (\log^* m)^2}{\log \log \log m})$ time.*

Proof. The non-empty sets $\mathcal{P}_{f_{\text{sig}},H}^\pm(\tau, \mathcal{L})$ (henceforth denoted \mathcal{P}_H^\pm) are maintained in the data structure of Theorem 4.1, whereas the non-empty sets $\mathcal{I}_{f_{\text{sig}},H}^\pm(\tau, \mathcal{L})$ (henceforth denoted \mathcal{I}_H^\pm) are maintained in the data structure of Corollary 5.2. Pointers to these components are stored in a deterministic dynamic dictionary [FG15] indexed by $\text{symb}_{\text{sig}}(H)$.

The first step of the query algorithm is to compute $\text{symb}_{\text{sig}}(H)$ by extracting the appropriate fragment using $\mathcal{W}.\text{split}$. This lets us identify the data structure responsible for the query in $\mathcal{O}(\log n \cdot \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$ time. By Lemma 10.6, each comparison on $\text{CF}^+(\mathcal{L})$ costs $\mathcal{O}(\log n \cdot \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$

time. As a result, the cost of a range query is $\mathcal{O}(\log^3 n + \log^2 n \cdot \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$. On the other hand, the cost of a modular constraint query is $\mathcal{O}(\log^3 n)$.

Before executing the $\text{split}(S, i)$ operation of Lemma 10.7, we identify all maximal intervals $[j' \dots j''] \in \mathbf{R}(\tau, \text{val}(S))$ (with $j' \in \mathbf{R}'(\tau, \text{val}(S))$ and $j'' \in \mathbf{R}''(\tau, \text{val}(S))$) such that $i \in [j' \dots j'' + 7\tau)$ or $i + |S| \in [j' \dots j'' + 7\tau)$. There are $\mathcal{O}(1)$ such intervals, and they can be generated in $\mathcal{O}(1)$ time using the succ and pred operations on $\mathbf{R}'(\tau, \mathcal{L})$ and $\mathbf{R}''(\tau, \mathcal{L})$. For each identified interval $[j' \dots j'']$, we remove the entries with label $\text{label}(S[j'])$ from the sets \mathcal{P}_H^\pm and \mathcal{I}_H^\pm containing it, and we add $\text{label}(S[j'])$ to a temporary set A containing all labels $\ell \in \mathbf{R}'(\tau, \mathcal{L})$ which are missing their elements in \mathcal{P}_H^\pm and \mathcal{I}_H^\pm . Then, we execute the $\text{split}(S, i)$ operation of Lemma 10.7, updating the set A whenever a label is removed from or added to $\mathbf{R}'(\tau, \mathcal{L})$. Finally, we iterate over the set A to add the missing elements to \mathcal{P}_H^\pm and \mathcal{I}_H^\pm . For such a label $\ell \in A$, we retrieve $(T, j) = \text{unlabel}(\ell)$ and ask the query of Lemma 10.7 to determine $p := |\text{root}_{f_{\text{sig}}}(\tau, \text{val}(T), j)|$, $h := \text{head}_{f_{\text{sig}}}(\tau, \text{val}(T), j)$, and $e := e(\tau, \text{val}(T), j)$. This lets us extract $H = \text{val}(T)[j + h \dots j + h + p]$ via a $\mathcal{W}.\text{split}$ operation and, in particular identify $\text{symb}_{\text{sig}}(H)$. Moreover, we compute $\text{type}(\tau, \text{val}(T), j)$ by comparing $\text{val}(T)^\infty[j \dots]$ with $\text{val}(T)^\infty[j + p \dots]$ via Lemma 10.6 so that we know whether the elements should be inserted to \mathcal{P}_H^- and \mathcal{I}_H^- or to \mathcal{P}_H^+ and \mathcal{I}_H^+ . Finally, we determine $e' := e_{f_{\text{sig}}}^{\text{full}}(\tau, \text{val}(T), j) = e - (e - j - h) \bmod p$ and insert a point $(e', \text{val}(T)^\infty[e' \dots e' + 7\tau], \text{label}(j))$ to \mathcal{P}_H^\pm , and a tuple $(e' - e + 3\tau - 1, e' - j + 1, \text{label}(j))$ to \mathcal{I}_H^\pm . The overall running time is increased to $\mathcal{O}(\log^2 n \cdot \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$, dominated by the cost of $\mathcal{O}(1)$ insertions and deletions on the sets \mathcal{P}_H^\pm .

The implementation of the $\text{concat}(R, S)$ operation is symmetric. The only difference is that the auxiliary set A is initialized based on maximal intervals $[j' \dots j''] \in \mathbf{R}(\tau, \text{val}(R))$ with $|R| + 1 \in [j' \dots j'' + 7\tau)$ and on maximal intervals $[j' \dots j''] \in \mathbf{R}(\tau, \text{val}(S))$ with $|S| + 1 \in [j' \dots j'' + 7\tau)$. \square

Proposition 10.14. *For any fixed $\ell \in \mathbb{Z}_+$, a dynamic text $T \in \Sigma^n$ can be implemented so that initialization takes $\mathcal{O}(\frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$ time, updates take $\mathcal{O}(\log^2 n \cdot \frac{\log^2 \log m \cdot (\log^* m)^2}{\log \log \log m})$ time, and the queries of Assumption 6.21 take $\mathcal{O}(\log^3 n + \log^2 n \cdot \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$ time, where $m = |\Sigma| + t$ and t is the total number of instructions that the data structure has performed so far.*

Proof. We proceed as in the proof of Corollary 10.2, but instead of implementing family \mathcal{L} using Lemma 10.1, we apply the extensions of Lemmas 10.12 and 10.13 for $\tau = \lfloor \frac{\ell}{3} \rfloor$. This increases the cost of initialize to $\mathcal{O}(\frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$ and the cost of updates to $\mathcal{O}(\log^2 n \cdot \frac{\log^2 \log m \cdot (\log^* m)^2}{\log \log \log m})$ (dominated by $\mathcal{L}.\text{split}$ and $\mathcal{L}.\text{concat}$).

We use $f := f_{\text{sig}}$ as the necklace-consistent function. The queries asking to compute $|\text{root}_f(\tau, T, j)|$, $\text{head}_f(\tau, T, j)$, and $e(\tau, T, j)$ for $j \in \mathbf{R}(\tau, T)$ are answered in $\mathcal{O}(\log n)$ time directly using Lemma 10.12. As for the int-string range queries for the sets $\text{Points}_{7\tau}(T, E_{f,H}^\pm(\tau, T))$, we use the equivalent range queries on $\mathcal{P}_{f,H}^\pm(\tau, \mathcal{L})$ answered using Lemma 10.10. The only work needed is to convert the query arguments from indices to fragments in $\text{CF}^+(T)$ (as specified in Problem 4.5). Moreover, the label returned by the range selection on $\mathcal{P}_{\text{sig},H}^\pm(\tau, \mathcal{L})$ is converted to a position in T using the $\mathcal{L}.\text{unlabel}(\cdot)$ operation. The query time is $\mathcal{O}(\log^3 n + \log^2 n \cdot \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$, dominated by the cost of range queries of Lemma 10.13.

Finally, the modular constraint queries on the sets $\mathcal{I}_{f,H}^\pm(\tau, T)$ are implemented using the equivalent modular constraint queries on the set $\mathcal{I}_{f,H}^\pm(\tau, \mathcal{L})$. No transformation of arguments and query outputs are necessary here, and the overall query time is $\mathcal{O}(\log^3 n + \log^2 n \cdot \frac{\log^2 \log m \cdot \log^* m}{\log \log \log m})$. \square

10.6 Summary

We are now ready to describe the main result of our work: a dynamic text implementation that can answer suffix array queries. We start with a version with a bounded lifespan: it takes an additional parameter N at initialization time, and it is only able to handle N operations. Then, we use this solution as a black box to develop an ‘everlasting’ dynamic suffix array.

Proposition 10.15. *For any given integer $N \geq \sigma$, a dynamic text $T \in [0.. \sigma]^+$ can be implemented so that initialization takes $\mathcal{O}(\log N \cdot \frac{\log^2 \log N \cdot \log^* N}{\log \log \log N})$ time, updates take $\mathcal{O}(\log^3 N \cdot \frac{\log^2 \log N \cdot (\log^* N)^2}{\log \log \log N})$ time, the suffix array queries take $\mathcal{O}(\log^4 N)$ time, and the inverse suffix array queries take $\mathcal{O}(\log^5 N)$ time, provided that the total number of updates and queries does not exceed N .*

Proof. We maintain T using data structures of Proposition 10.4 and Lemma 10.6, as well as several instances of the data structures of Propositions 10.8, 10.11, and 10.14 for $\ell = 2^q$, where $q \in [4.. \lceil \log N \rceil]$. The initialization and each update operation needs to be replicated in all these components.

The suffix queries are implemented using Proposition 6.55. Due to the fact that $|T| \leq N$, the components maintained are sufficient to satisfy the assumption required in Proposition 6.55.

As for the inverse suffix array queries, we perform binary search. In each of the $\mathcal{O}(\log |T|) = \mathcal{O}(\log N)$ steps, we compare the specified suffix $T[j.. |T|]$ with the suffix $T[\text{SA}[i].. |T|]$; here, we use a suffix array query to determine $\text{SA}[i]$ and the lexicographic comparison (of Lemma 10.6) to compare the two suffixes lexicographically.

Recall that the running times of all the components are expressed in terms of parameters $n = |T|$ (which does not exceed N) and $m = \sigma + t$, where t is the total number of instructions performed so far by the respective component. This value may differ across components, but we bound it from above by the total number of instructions performed so far by all the components; let us call this value M . Note that each update and query costs $\mathcal{O}(\log^{\mathcal{O}(1)}(N + M))$ time, which means that $M = \mathcal{O}(\sigma + N \log^{\mathcal{O}(1)} N) = \mathcal{O}(N \log^{\mathcal{O}(1)} N)$, where the last step follows from the assumption $N \geq \sigma$.

Consequently, the initialization takes $\mathcal{O}(\log N \cdot \frac{\log^2 \log M \cdot \log^* M}{\log \log \log M}) = \mathcal{O}(\log N \cdot \frac{\log^2 \log N \cdot \log^* N}{\log \log \log N})$ time and the updates take $\mathcal{O}(\log N \cdot \log^2 n \cdot \frac{\log^2 \log M \cdot (\log^* M)^2}{\log \log \log M}) = \mathcal{O}(\log^3 N \cdot \frac{\log^2 \log N \cdot (\log^* N)^3}{\log \log \log N})$ time. By Proposition 6.55, suffix queries take $\mathcal{O}(\log n \cdot (\log^3 n + \log^2 n \frac{\log^2 \log M \cdot \log^* M}{\log \log \log M})) = \mathcal{O}(\log^4 N)$ time. On the other hand, the inverse suffix array queries cost $\mathcal{O}(\log N \cdot (\log^4 N + \log n \cdot \frac{\log^2 \log M \cdot \log^* M}{\log \log \log M})) = \mathcal{O}(\log^5 N)$ time. \square

Theorem 10.16. *A dynamic text $T \in [0.. \sigma]^n$ can be implemented so that initialization takes $\mathcal{O}(\log \sigma \cdot \frac{\log^2 \log \sigma \cdot \log^* \sigma}{\log \log \log \sigma})$ time, updates take $\mathcal{O}(\log^3(n\sigma) \cdot \frac{\log^2 \log(n\sigma) \cdot (\log^*(n\sigma))^2}{\log \log \log(n\sigma)})$ time, the suffix array queries take $\mathcal{O}(\log^4(n\sigma))$ time, and the inverse suffix array take queries $\mathcal{O}(\log^5(n\sigma))$ time.*

Proof. We first describe an amortized-time solution which performs a *reorganization* every $\Omega(n)$ operations. This reorganization takes $\mathcal{O}(n \cdot U(n, \sigma))$ time, where $U(n, \sigma) = \mathcal{O}(\log^3(n\sigma) \frac{\log^2 \log(n\sigma) \cdot (\log^*(n\sigma))^2}{\log \log \log(n\sigma)})$ is the update time.

The text T is stored using the data structures of both Corollary 10.2 and Proposition 10.15. Moreover, we maintain a counter t representing the number of operations that can be performed before reorganization. At initialization time, we set $t = 1$ and initialize both components, setting $N = \sigma$ for Proposition 10.15. The updates and queries are forwarded to the component of Proposition 10.15, but

we first perform reorganization (if $t = 0$) and decrement t (unconditionally). As for the reorganization, we set $t = \lceil \frac{1}{2}|T| \rceil$, discard the component of Proposition 10.15, and initialize a fresh copy using $N = \max(\sigma, \lceil \frac{3}{2}|T| \rceil - 1)$; we then insert characters of T one by one using the access operation of Corollary 10.2 and the insert operation of Proposition 10.15.

To prove that this implementation is correct, we must argue that each instance of Proposition 10.15 performs no more than N operations. The instance created at initialization time is limited to a single operation, which is no more than the allowance of $N = \sigma$ operations. On the other hand, an instance created during a reorganization performs $|T| - 1$ insertions during the reorganization, and is then limited to $\lceil \frac{1}{2}|T| \rceil$ operations. In total, this does not exceed the allowance of $N = \max(\sigma, \lceil \frac{3}{2}|T| \rceil - 1)$ operations.

It remains to analyze the time complexity. For this, we observe that, if $N > \sigma$, then $|T| \geq |T| - t \geq \lfloor \frac{1}{3}N \rfloor$ is preserved as an invariant. This means that $N = \mathcal{O}(\max(\sigma, |T|)) = \mathcal{O}(\sigma|T|)$, and thus the operation times of Proposition 10.15 can be expressed using $n\sigma$ instead of N . This also applies to the cost of reorganization, which uses initialization and $n - 1$ updates.

As for the deamortization, we use the standard technique of maintaining two instances of the above data structure. At any time, one them is active (handles updates and queries), whereas the other undergoes reorganization. The lifetime of the entire solution is organized into *epochs*. At the beginning of each epoch, the active instance is ready to handle $t \geq \frac{1}{2}n$ forthcoming operations, whereas the other instance needs to be reorganized. The epoch lasts for t operations. During the first half of the epoch, the reorganization is performed in the background and the updates are buffered in a queue. For each operation in the second half of the epoch, at most one update in buffered (none if the operation is a query) and two buffered updates are executed (unless there are already fewer updates in the buffer). Since the reorganization cost is bounded by $\mathcal{O}(t \cdot U(n, t))$ and since the query cost is larger than the update cost, the deamortized solution has the same asymptotic time complexity as the amortized one. \square

11 Conditional Lower Bound for Copy-Pastes

A natural extension of the dynamic text interface provided in Section 10 would be to support not only cut-pastes (that move fragments of T), but also a copy-pastes (that copy fragments of T). Such operation is readily supported by the underlying implementation of dynamic strings (Section 8), but it is incompatible with the concept of labelling characters – a single copy-paste may add multiple new characters, and we cannot afford to assign them labels one by one. In this section, we provide a conditional lower bound showing that this is not due to a limitation of our techniques, but rather due to inherent difficulty of the (inverse) suffix array queries in dynamic texts. Our lower bound is conditioned on the Online Matrix-Vector Multiplication Conjecture [HKNS15], which is often used in the context of dynamic algorithms. The underlying reduction resembles one from the Dynamic Internal Dictionary Matching problem [CKM⁺21], where the queries ask if a given fragment of a static text contains an occurrence of any pattern from a dynamic dictionary.

In the Online Boolean Matrix-Vector Multiplication (OMv) problem, we are given as input an $n \times n$ boolean matrix M . Then, a sequence n vectors v_1, \dots, v_n , each of size n , arrives in an online fashion. For each such vector v_i , we are required to output Mv_i before receiving v_{i+1} .

Conjecture 11.1 (OMv Conjecture [HKNS15]). *For any constant $\epsilon > 0$, there is no $\mathcal{O}(n^{3-\epsilon})$ -time algorithm that solves OMv correctly with probability at least $\frac{2}{3}$.*

We use the following simplified version of [HKNS15, Theorem 2.2].

Theorem 11.2 ([HKNS15]). *For all constants $\gamma, \epsilon > 0$, the OMv Conjecture implies that there is no algorithm that, given as input a $p \times q$ matrix M , with $p = \lfloor q^\gamma \rfloor$, preprocesses M in time polynomial in $p \cdot q$, and then, presented with a vector v of size q , computes Mv in time $\mathcal{O}(q^{1+\gamma-\epsilon})$ correctly with probability at least $\frac{2}{3}$.*

Theorem 11.3. *For all constants $\alpha, \beta > 0$ with $\alpha + \beta < 1$, the OMv Conjecture implies that there is no dynamic algorithm that preprocesses a text T in time polynomial in $|T|$, supports copy-pastes in $\mathcal{O}(|T|^\alpha)$ time, and inverse suffix array queries in $\mathcal{O}(|T|^\beta)$ time, with each answer correct with probability at least $\frac{2}{3}$.*

Proof. Let us suppose that there is such an algorithm and set $\gamma = \frac{1+\alpha-\beta}{1+\beta-\alpha}$. Given a $p \times q$ matrix M satisfying $p = \lfloor q^\gamma \rfloor$, we construct a text T of length $pq + 2p + 3$ over an alphabet $\{\$, a_0, \dots, a_p, \#\}$ (with $\$ \prec a_0 \prec \dots \prec a_p \prec \$$) using the following formula:

$$T = \left(\bigcirc_{j=1}^q \left(\bigcirc_{i=1}^p a_{i \cdot M[i,j]} \right) \right) \cdot \left(\bigcirc_{i=0}^p a_i \# \right) \cdot \$.$$

In other words, we first write down the columns of M in the increasing order, replacing each zero with a_0 and each one with a_i , where $i \in [1..p]$ is the row index of the underlying matrix entry. Then, we append $a_i \#$ for all $i \in [0..p]$, and finally we place $\$$ at the very end of T . At the preprocessing time, we also precompute, for each $i \in [1..p]$, the number c_i of occurrences of a_i in T .

Given a query vector $v \in \{0, 1\}^q$, we proceed as follows. For each $j \in [1..q]$ with $v_j = 1$, we copy $T((j-1)p..jp)$ to the end of T (setting $T := T[1..|T|] \cdot T((j-1)p..jp) \cdot T[|T|]$). Next, we construct the answer vector $w \in \{0, 1\}^p$, setting $w_i := 1$ if and only if $\text{ISA}[pq + 2i + 1] - \text{ISA}[pq + 2i - 1] > c_i$ for $i \in [1..p]$.

Let us prove that $w = Mv$ holds assuming that all the ISA queries are answered correctly. For this, note that, for $i \in [0..p]$, the value $\text{ISA}[pq + 2i + 1]$ represents total number of occurrences of symbols $\$, a_0, \dots, a_i$ in T ; that is because $T[pq + 2i + 1..pq + 2i + 2]$ is the unique occurrence of $a_i \#$ in T , and $\#$ is the largest symbol in Σ . In particular, $\text{ISA}[pq + 2i + 1] - \text{ISA}[pq + 2i - 1] > c_i$ holds if and only if one of the copy-pastes involved a substring $T((j-1)p..jp)$ containing a_i . This character could have only occurred at position $T[(j-1)p + i]$, indicating that $M[i, j] = 1$. Moreover, the whole copy-paste is executed if and only if $v_j = 1$. Consequently, $\text{ISA}[pq + 2i + 1] - \text{ISA}[pq + 2i - 1] > c_i$ holds if and only if there exists $j \in [1..q]$ with $M[i, j] = v_j = 1$; this is precisely when $(Mv)_i = 1$.

If the answers to ISA queries are correct with probability at least $\frac{2}{3}$, we can guarantee that the whole vector w is correct with probability at least $1 - n^{-\Omega(1)} \geq \frac{2}{3}$ by maintaining $\Theta(\log n)$ independent instances of the algorithm and taking the dominant answer to each ISA query. In total, we perform $\tilde{\mathcal{O}}(q)$ copy-pastes and $\tilde{\mathcal{O}}(p)$ ISA queries. Furthermore, the copy-pastes have disjoint sources, so the length of T increases at most twofold. Hence, the total time required is

$$\tilde{\mathcal{O}}\left(q|T|^\alpha + p|T|^\beta\right) = \tilde{\mathcal{O}}\left(p^\alpha q^{1+\alpha} + p^{1+\beta} q^\beta\right) = \tilde{\mathcal{O}}\left(q^{1+(1+\gamma)\alpha} + q^{\beta+\gamma(1+\beta)}\right) = \tilde{\mathcal{O}}\left(q^{\frac{1+\alpha+\beta}{1+\beta-\alpha}}\right).$$

In the light of Theorem 11.2, this would disprove Conjecture 11.1 due to $\frac{1+\alpha+\beta}{1+\beta-\alpha} < \frac{2}{1+\beta-\alpha} = 1+\gamma$. \square

Remark 11.4. The reduction behind Theorem 11.3 not only proves hardness of inverse suffix array queries, but also of the counting version of the dynamic indexing problem: It shows that the counting

queries are hard already for patterns of length one. Moreover, since each ISA query can be reduced to a logarithmic number of SA queries, we get the same lower bound for SA queries. (The underlying reduction is described in the proof of Proposition 10.15; the dynamic strings of Section 8 support lexicographic comparisons and copy-pastes in $\tilde{O}(1)$ time.)

A Omitted Proofs

Lemma 6.18. *Let $S \in \Sigma^k$, $\tau \in \mathbb{Z}_+$, and let f be any necklace-consistent function. If $j \in \mathbf{R}_{f,s,H}(\tau, S)$ then for any $j' \in [1..k]$, $\text{LCE}_S(j, j') \geq 3\tau - 1$ holds if and only if $j' \in \mathbf{R}_{f,s,H}(\tau, S)$. Moreover, if $j' \in \mathbf{R}_{f,s,H}(\tau, S)$, then:*

1. *If $\text{type}(\tau, S, j) = -1$ and $\text{type}(\tau, S, j') = +1$, then $S[j..] \prec S[j'..]$,*
2. *If $\text{type}(\tau, S, j) = \text{type}(\tau, S, j') = -1$ and $e(\tau, S, j) - j < e(\tau, S, j') - j'$, then $S[j..] \prec S[j'..]$,*
3. *If $\text{type}(\tau, S, j) = \text{type}(\tau, S, j') = +1$ and $e(\tau, S, j) - j > e(\tau, S, j') - j'$, then $S[j..] \prec S[j'..]$.*

Proof. Let $j' \in [1..k]$ be such that $\text{LCE}_S(j, j') \geq 3\tau - 1$. Then, by definition, $\text{root}_f(\tau, S, j') = \text{root}_f(S[j'..j'+3\tau-1]) = \text{root}_f(S[j..j+3\tau-1]) = \text{root}_f(\tau, S, j)$. To show $\text{head}_f(\tau, S, j') = s$, note that by $|H| \leq \tau$, the string $H'H^2$ (where H' is a length- s suffix of H) is a prefix of $S[j..j+3\tau-1] = S[j'..j'+3\tau-1]$. On the other hand, $\text{head}_f(\tau, S, j') = s'$ implies that $\widehat{H}'H^2$ (where \widehat{H}' is a length- s' suffix of H) is a prefix of $S[j'..j'+3\tau-1]$. Thus, by the synchronization property of primitive strings [CHL07, Lemma 1.11] applied to the two copies of H , we have $s' = s$, and consequently, $j' \in \mathbf{R}_{f,s,H}$.

For the converse implication, assume $j, j' \in \mathbf{R}_{f,s,H}$. This implies that both $S[j..e(\tau, S, j)]$ and $S[j'..e(\tau, S, j')]$ are prefixes of $H'H^\infty$ (where H' is as above). Thus, by $e(\tau, S, j) - j, e(\tau, S, j') - j' \geq 3\tau - 1$, we obtain $\text{LCE}_S(j, j') \geq 3\tau - 1$.

1. Let $Q = H'H^\infty$, where H' is a length- s suffix of H . We will prove $S[j..k] \prec Q \prec S[j'..k]$, which implies the claim. First, we note that $\text{type}(\tau, S, j) = -1$ implies that either $e(\tau, S, j) = k + 1$, or $e(\tau, S, j) \leq k$ and $S[e(\tau, S, j)] \prec S[e(\tau, S, j) - |H|]$. In the first case, $S[j..e(\tau, S, j)] = S[j..k]$ is a proper prefix of Q and hence $S[j..k] \prec Q$. In the second case, letting $\ell = e(\tau, S, j) - j$, we have $S[j..j+\ell] = Q[1..\ell]$ and $S[j+\ell] \prec S[j+\ell - |H|] = Q[1+\ell]$. Consequently, $S[j..k] \prec Q$. To show $Q \prec S[j'..k]$ we observe that $\text{type}(\tau, S, j') = +1$ implies $e(\tau, S, j') \leq k$. Thus, letting $\ell' = e(\tau, S, j') - j'$, we have $Q[1..\ell'] = S[j'..j'+\ell']$ and $Q[1+\ell'] = Q[1+\ell' - |H|] = S[j'+\ell' - |H|] \prec S[j'+\ell']$. Hence, we obtain $Q \prec S[j'..k]$.

2. Similarly as above, we consider two cases for $e(\tau, S, j)$. If $e(\tau, S, j) = k + 1$, then by $e(\tau, S, j) - j < e(\tau, S, j') - j'$, $S[j..e(\tau, S, j)] = S[j..k]$ is a proper prefix of $S[j'..e(\tau, S, j')]$ and hence $S[j..k] \prec S[j'..e(\tau, S, j')] \preceq S[j'..k]$. If $e(\tau, S, j) \leq k$, then letting $\ell = e(\tau, S, j) - j$, we have $S[j..j+\ell] = S[j'..j'+\ell]$ and by $e(\tau, S, j) - j < e(\tau, S, j') - j'$, $S[j+\ell] \prec S[j+\ell - |H|] = S[j'+\ell - |H|] = S[j'+\ell]$. Consequently, $S[j..k] \prec S[j'..k]$.

3. By $\text{type}(\tau, S, j') = +1$ we have $e(\tau, S, j') \leq k$. Thus, letting $\ell' = e(\tau, S, j') - j'$, by $e(\tau, S, j) - j > e(\tau, S, j') - j'$, we have $S[j..j+\ell'] = S[j'..j'+\ell']$ and $S[j+\ell'] = S[j+\ell' - |H|] = S[j'+\ell' - |H|] \prec S[j'+\ell']$. Consequently, $S[j..k] \prec S[j'..k]$. \square

Lemma 6.19. *Let $S \in \Sigma^+$, $\tau \in \mathbb{Z}_+$, and assume that f is a necklace-consistent function. For any position $j \in \mathbf{R}(\tau, S) \setminus \mathbf{R}'(\tau, S)$ it holds*

- $\text{root}_f(\tau, S, j-1) = \text{root}_f(\tau, S, j)$,
- $e(\tau, S, j-1) = e(\tau, S, j)$, and

- $\text{type}(\tau, S, j-1) = \text{type}(\tau, S, j)$.

Proof. Denote $p = \text{per}(S[j-1..j-1+3\tau-1])$ and $p' = \text{per}(S[j..j+3\tau-1])$. By $j-1, j \in \mathbf{R}(\tau, S)$ we have $p, p' \leq \frac{\tau}{3}$. Consider $Q = S[j..j+\tau]$. The string Q has both periods p and p' . If $p \neq p'$, then by the weak periodicity lemma, Q has a period $p'' = \text{gcd}(p, p') < p'$. Since $p'' \mid p'$ we obtain that $S[j..j+p']$ is not primitive, which contradicts $\text{per}(S[j..j+3\tau-1]) = p'$. Thus, $p = p'$. Then, by $p \leq \frac{\tau}{3}$, we have $S[j-1..j-1+p] = S[j-1+p..j-1+2p]$. Consequently, $\{S[j-1+t..j-1+t+p] : t \in [0..p]\} = \{S[j+t..j+t+p] : t \in [0..p]\}$. This implies that $S[j-1..j-1+p]$ and $S[j..j+p]$ are cyclically equivalent. Thus, it holds $\text{root}_f(\tau, S, j-1) = f(S[j-1..j-1+p]) = f(S[j..j+p]) = \text{root}_f(\tau, S, j)$.

By [KK19, Fact 3.2], $S[j-1..e(\tau, S, j-1)]$ (resp. $S[j..e(\tau, S, j)]$) is the longest substring starting at position $j-1$ (resp. j) with period p (resp. p'). Equivalently, $e(\tau, S, j-1) = j-1+p + \text{LCE}_S(j-1, j-1+p)$ and $e(\tau, S, j) = j+p' + \text{LCE}_S(j, j+p')$. Thus, by $p = p'$ and $S[j-1] = S[j-1+p]$, we have $e(\tau, S, j-1) = j-1+p + \text{LCE}_S(j-1, j-1+p) = j+p + \text{LCE}_S(j, j+p) = j+p' + \text{LCE}_S(j, j+p') = e(\tau, S, j)$.

The third claim follows from the definition of type and equalities $e(\tau, S, j-1) = e(\tau, S, j)$ and $p = p'$. \square

References

- [AB20] Amihod Amir and Itai Boneh. Update query time trade-off for dynamic suffix arrays. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 181 of *LIPIcs*, pages 63:1–63:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ISAAC.2020.63.
- [AB21] Amihod Amir and Itai Boneh. Dynamic suffix array with sub-linear update time and poly-logarithmic lookup time, 2021. arXiv:2112.12678.
- [ABM08] 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.
- [ABR00] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Pattern matching in dynamic texts. In David B. Shmoys, editor, *SODA*, pages 819–828. ACM/SIAM, 2000. URL: <http://dl.acm.org/citation.cfm?id=338219.338645>.
- [ACI⁺19] Mai Alzamel, Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Waleń, and Wiktor Zuba. Quasi-linear-time algorithm for longest common circular factor. In *CPM*, pages 25:1–25:14, 2019. doi:10.4230/LIPIcs.CPM.2019.25.
- [AJ22] Shyan Akmal and Ce Jin. Near-optimal quantum algorithms for string problems. In *SODA*, 2022. arXiv:2110.09696.
- [ANS12] Diego Arroyuelo, Gonzalo Navarro, and Kunihiko Sadakane. Stronger Lempel-Ziv based compressed text indexing. *Algorithmica*, 62(1-2):54–101, 2012. doi:10.1007/s00453-010-9443-8.
- [AVL62] G. M. Adel'son-Vel'skii and E. M. Landis. An algorithm for organization of information. *Dokl. Akad. Nauk SSSR*, 146(2):263–266, 1962.

- [BEGV18] 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.
- [BFG⁺17] Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, Tsvi Kopelowitz, and Pablo Montes. File maintenance: When in doubt, change the layout! In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1503–1522. SIAM, 2017. doi:10.1137/1.9781611974782.98.
- [BGG⁺15] Djamel 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.
- [BGP20] Or Birenzweig, Shay Golan, and Ely Porat. Locally consistent parsing for text indexing in small space. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 607–626. SIAM, 2020. doi:10.1137/1.9781611975994.37.
- [BLR⁺15] 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.
- [BW94] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994. URL: <http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf>.
- [CEK⁺21] 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.
- [Cha88] 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.
- [CHL07] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, Cambridge, UK, 2007. doi:10.1017/cbo9780511546853.
- [CHLS07] 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.
- [CHS⁺15] Yu-Feng Chien, Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Geometric BWT: compressed text indexing via sparse suffixes and range searching. *Algorithmica*, 71(2):258–278, 2015. doi:10.1007/s00453-013-9792-1.
- [CKM⁺21] Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal dictionary matching. *Algorithmica*, 83(7):2142–2169, 2021. doi:10.1007/s00453-021-00821-y.
- [CKPR21] Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski. Faster algorithms for longest common substring. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPICs*, pages 30:1–30:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ESA.2021.30.
- [CKW20] Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In *61st Annual IEEE Sym-*

- posium on Foundations of Computer Science, FOCS 2020*, pages 978–989, 2020. doi:10.1109/FOCS46700.2020.00095.
- [CN11] Francisco Claude and Gonzalo Navarro. Self-indexed grammar-based compression. *Fundam. Informaticae*, 111(3):313–337, 2011. doi:10.3233/FI-2011-565.
- [CNP21] Francisco Claude, Gonzalo Navarro, and Alejandro Pacheco. Grammar-compressed indexes with logarithmic search time. *J. Comput. Syst. Sci.*, 118:53–74, 2021. doi:10.1016/j.jcss.2020.12.001.
- [CV86] Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Inf. Control.*, 70(1):32–53, 1986. doi:10.1016/S0019-9958(86)80023-7.
- [DS87] Paul F. Dietz and Daniel Dominic Sleator. Two algorithms for maintaining order in a list. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 365–372. ACM, 1987. doi:10.1145/28395.28434.
- [EMOW11] Andrzej Ehrenfeucht, Ross M. McConnell, Nissa Osheim, and Sung-Wan Woo. Position heaps: A simple and dynamic text indexing data structure. *J. Discrete Algorithms*, 9(1):100–121, 2011. doi:10.1016/j.jda.2010.12.001.
- [FG15] 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.
- [FM05] Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.
- [GFB94] Ming Gu, Martin Farach, and Richard Beigel. An efficient algorithm for dynamic text indexing. In Daniel Dominic Sleator, editor, *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA*, pages 697–704. ACM/SIAM, 1994. URL: <http://dl.acm.org/citation.cfm?id=314464.314675>.
- [GGK⁺12] 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.
- [GGK⁺14] Travis Gagie, Paweł 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.
- [GKK⁺15] Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łacki, and Piotr Sankowski. Optimal dynamic strings, 2015. arXiv:1511.02612.
- [GKK⁺18] Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łacki, and Piotr Sankowski. Optimal dynamic strings. In *SODA*, pages 1509–1528, 2018. doi:10.1137/1.9781611975031.99.
- [GN09] Rodrigo González and Gonzalo Navarro. Rank/select on dynamic compressed sequences and applications. *Theor. Comput. Sci.*, 410(43):4414–4422, 2009. doi:10.1016/j.tcs.2009.07.022.

- [GNP20] 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.
- [Gus97] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.
- [GV05] 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.
- [Hag98] 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.
- [HKNS15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *47th Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.
- [HM10] Meng He and J. Ian Munro. Succinct representations of dynamic strings. In Edgar Chávez and Stefano Lonardi, editors, *String Processing and Information Retrieval - 17th International Symposium, SPIRE 2010, Los Cabos, Mexico, October 11-13, 2010. Proceedings*, volume 6393 of *Lecture Notes in Computer Science*, pages 334–346. Springer, 2010. doi:10.1007/978-3-642-16321-0_35.
- [Kär99] Juha Kärkkäinen. *Repetition-based Text Indexes*. PhD thesis, University of Helsinki, 1999. URL: <https://helda.helsinki.fi/bitstream/handle/10138/21348/repetiti.pdf>.
- [KK19] 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.
- [KK20] Dominik Kempa and Tomasz Kociumaka. Resolution of the Burrows-Wheeler transform conjecture. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1002–1013. IEEE, 2020. doi:10.1109/FOCS46700.2020.00097.
- [KK21] Dominik Kempa and Tomasz Kociumaka. Breaking the $\mathcal{O}(n)$ -barrier in the construction of compressed suffix arrays, 2021. arXiv 2106.12725. arXiv:2106.12725.
- [KLA⁺01] 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.
- [KN13] Sebastian Kreft and Gonzalo Navarro. On compressing and indexing repetitive sequences. *Theor. Comput. Sci.*, 483:115–133, 2013. doi:10.1016/j.tcs.2012.02.006.
- [Koc18] Tomasz Kociumaka. *Efficient Data Structures for Internal Queries in Texts*. PhD thesis, University of Warsaw, 2018.
- [KRRW15] Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. In *SODA*, pages 532–551, 2015. doi:10.1137/1.9781611973730.36.
- [LW82] George S. Lueker and Dan E. Willard. A data structure for dynamic range queries. *Inf. Process. Lett.*, 15(5):209–213, 1982. doi:10.1016/0020-0190(82)90119-3.

- [MBCT15] Veli Mäkinen, Djamel 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.
- [MM93] 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.
- [MN08] 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.
- [MNKS13] Shirou Maruyama, Masaya Nakahara, Naoya Kishiue, and Hiroshi Sakamoto. ESP-index: A compressed index based on edit-sensitive parsing. *J. Discrete Algorithms*, 18:100–112, 2013. doi:10.1016/j.jda.2012.07.009.
- [MNN20] 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.
- [MSU97] Kurt Mehlhorn, R. Sundar, and Christian Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, 1997. URL: <https://doi.org/10.1007/BF02522825>, doi:10.1007/BF02522825.
- [Nav16] Gonzalo Navarro. *Compact data structures: A practical approach*. Cambridge University Press, Cambridge, UK, 2016. doi:10.1017/cbo9781316588284.
- [NII⁺16] Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Fully dynamic data structure for LCE queries in compressed space. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016*, pages 72:1–72:15, 2016. doi:10.4230/LIPICs.MFCS.2016.72.
- [NII⁺20] 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.
- [NM07] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1):2, 2007. doi:10.1145/1216370.1216372.
- [NN14] Gonzalo Navarro and Yakov Nekrich. Optimal dynamic sequence representations. *SIAM J. Comput.*, 43(5):1781–1806, 2014. doi:10.1137/130908245.
- [NS14] Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Trans. Algorithms*, 10(3):16:1–16:39, 2014. doi:10.1145/2601073.
- [Ohl13] Enno Ohlebusch. *Bioinformatics algorithms: Sequence analysis, genome rearrangements, and phylogenetic reconstruction*. Oldenbusch Verlag, Ulm, Germany, 2013.
- [RNO08] Luís M. S. Russo, Gonzalo Navarro, and Arlindo L. Oliveira. Dynamic fully-compressed suffix trees. In Paolo Ferragina and Gad M. Landau, editors, *Combinatorial Pattern Matching, 19th Annual Symposium, CPM 2008, Pisa, Italy, June 18-20, 2008, Proceedings*, volume 5029 of *Lecture Notes in Computer Science*, pages 191–203. Springer, 2008. doi:10.1007/978-3-540-69068-9_19.
- [SLLM10] Mikael Salson, Thierry Lecroq, Martine Léonard, and Laurent Mouchard. Dynamic extended suffix arrays. *J. Discrete Algorithms*, 8(2):241–257, 2010. doi:10.1016/j.jda.2009.02.007.

- [SV94] Süleyman Cenk Sahinalp and Uzi Vishkin. Symmetry breaking for suffix tree construction. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 300–309. ACM, 1994. doi:[10.1145/195058.195164](https://doi.org/10.1145/195058.195164).
- [SV96] Süleyman Cenk Sahinalp and Uzi Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm (extended abstract). In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 320–328. IEEE Computer Society, 1996. doi:[10.1109/SFCS.1996.548491](https://doi.org/10.1109/SFCS.1996.548491).
- [TKN⁺20] Kazuya Tsuruta, Dominik Köppl, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Grammar-compressed self-index with Lyndon words, 2020. [arXiv:2004.05309](https://arxiv.org/abs/2004.05309).
- [TTS14] Yoshimasa Takabatake, Yasuo Tabei, and Hiroshi Sakamoto. Improved ESP-index: A practical self-index for highly repetitive texts. In *SEA*, pages 338–350, 2014. doi:[10.1007/978-3-319-07959-2_29](https://doi.org/10.1007/978-3-319-07959-2_29).
- [WL85] Dan E. Willard and George S. Lueker. Adding range restriction capability to dynamic data structures. *J. ACM*, 32(3):597–617, 1985. doi:[10.1145/3828.3839](https://doi.org/10.1145/3828.3839).