Explaining the Inherent Tradeoffs for Suffix Array Functionality: Equivalences between String Problems and Prefix Range Queries

Dominik Kempa*
Department of Computer Science,
Stony Brook University,
Stony Brook, NY, USA
kempa@cs.stonybrook.edu

Tomasz Kociumaka
Max Planck Institute for Informatics,
Saarland Informatics Campus,
Saarbrücken, Germany
tomasz.kociumaka@mpi-inf.mpg.de

Abstract

We address a fundamental question in string processing: how much time is needed to access suffix array entries when there is insufficient space to store it in plain form? The suffix array $\mathrm{SA}_T[1\,..\,n]$ of a text T of length n is a permutation of $\{1,\ldots,n\}$ that encodes the lexicographic ordering of the suffixes of T. While its canonical application is as a pattern matching index, the suffix array's simplicity, space efficiency, and theoretical guarantees have made it a cornerstone of string processing over the past 35 years, with applications spanning data compression, bioinformatics, and information retrieval.

Prior work has developed unidirectional reductions, showing how suffix array queries (given $i \in [1\,..\,n]$, return $\mathrm{SA}_T[i]$) can be reduced, among others, to rank queries over the Burrows–Wheeler Transform (BWT). More recently, an alternative family of *prefix queries* was introduced, along with a reduction that transforms a simple tradeoff for prefix queries—requiring little more than a page to develop—into a suffix array tradeoff that matches all known space and query time bounds, while achieving sublinear construction time. For a binary text $T \in \{\mathtt{0},\mathtt{1}\}^n$, this tradeoff achieves space usage $S(n) = \mathcal{O}(n)$ bits, preprocessing time $P_t(n) = \mathcal{O}(n/\sqrt{\log n})$, preprocessing space $P_s(n) = \mathcal{O}(n)$ bits, and query time $Q(n) = \mathcal{O}(\log^\epsilon n)$ for any constant $\epsilon > 0$. Despite this progress, a key question remains: can any of these complexities be improved using fundamentally different techniques?

In this work, we address this question as follows:

- We establish the first bidirectional reduction, proving that suffix array queries are, up to an additive $\mathcal{O}(\log\log n)$ term in query time, *equivalent* to prefix select queries in all four aspects. For a sequence $W[1\,..\,m]$ of $\mathcal{O}(\log m)$-bit strings, a prefix select query, given $i \in [1\,..\,m]$ and a string X, returns the position j of the ith leftmost string $W[j]$ in W with prefix X. Our reduction unifies previous approaches and offers a structured framework for deriving new tradeoffs without sacrificing generality: before this work, prefix select queries constituted merely one approach to efficient suffix array queries, whereas we prove that, up to the $\mathcal{O}(\log\log n)$ additive overhead in query time, they capture all possible suffix array representations.
- We further show that other core string-processing problems—such as inverse suffix array queries, pattern ranking, lexicographic range queries, and pattern SA-interval queries—are also computationally equivalent to distinct types of prefix queries. In total, we identify *six* fundamental problem pairs, each consisting of a string problem and a corresponding prefix problem, and prove analogous equivalences.

Taken together, our reductions provide a general method for analyzing the efficiency of suffix array and related queries in terms of prefix queries, demonstrating that further improvements to fundamental string problems can be achieved by focusing solely on this recently introduced class of abstract queries.

*Partially funded by the NSF CAREER Award 2337891 and the Simons Foundation Junior Faculty Fellowship.

1 Introduction

The suffix array (SA) [MM90] is one of the most fundamental and widely used text indexing structures in computer science. Given a string $T \in \Sigma^n$ of length n over an alphabet Σ, its suffix array $\mathrm{SA}_T[1\,..\,n]$ is a permutation of $\{1, 2, \ldots, n\}$ that orders the suffixes of T lexicographically, i.e., so that $T[\mathrm{SA}_T[1]\,..\,n] \prec T[\mathrm{SA}_T[2]\,..\,n] \prec \cdots \prec T[\mathrm{SA}_T[n]\,..\,n]$. This simple yet powerful structure enables fast pattern matching: given a length-m pattern $P \in \Sigma^m$, we can determine whether P appears in T via binary search over SA_T in $\mathcal{O}(m \log n)$ time.

Shortly after it was introduced, researchers recognized that the suffix array has applications far beyond pattern matching. Due to its simplicity, space efficiency, strong theoretical guarantees, and excellent practical performance, the suffix array has, over the past 35 years, become one of the most widely used data structures [Gus97, ABM08, AKO04], with applications in string processing, data compression, and information retrieval:

- Suffix arrays can be augmented [AKO04] to implement the full suffix tree functionality [Wei73], enabling efficient solutions to classical problems such as longest common substring [Gus97, CKPR21], longest common extension (LCE) queries [LV88, KK19], longest repeating substring [Gus97], shortest unique substring [IS11], and minimal absent word [BHMP14].
- Suffix arrays underlie widely used compression algorithms, such as the Burrows–Wheeler Transform (BWT) [BW94, OS09, KK19] and the Lempel–Ziv (LZ77) factorization [CIS08, OG11, KP13, GB13]. This, in turn, determines the complexity of finding runs (maximal repeats) [Mai89, KK99, CPS07, CI08], approximate repetitions [KK03], local periods [DKK+04], and seeds [KKR+20].
- Suffix arrays solve numerous central bioinformatics problems, such as sequence alignment [LS12, LD09], maximal exact matches (MEMs) [Gus97], maximal unique matches (MUMs) [KPD+04], tandem repeats [GS04], genome assembly [GK12, OG10], sequence clustering [HL11], k-mer counting [KNSW08], error correction [IFI11], and many others [Gus97, Ohl13, Gro11].
- Suffix arrays, in addition to basic exact pattern matching [MM90], solve numerous other variants, including approximate pattern matching [Nav01, LV88], gapped pattern matching [CPZ20], parameterized pattern matching [FNI+19], pattern matching with "don't care" symbols (also known as *wildcards*) [MB91], regular expression pattern matching [Gus97, BYRN99, ABG+14], and compressed pattern matching [GNP20].

Numerous variants and generalizations of suffix arrays have been developed, including sparse SAs [Pre18, GK17], multi-string SAs [Shi96], dynamic SAs [KK22], on-disk SAs [GMC+14], distributed SAs [FA19], enhanced SAs [AKO04], gapped SAs [CT10], compact SAs [Mäk03], succinct SAs [MN05], entropy-compressed SAs [GV05, FM05, KK19], dictionary-compressed SAs [GNP20, KK23b], SAs for circular strings [HLST11], SAs for weighted strings [CILP20], SAs for matrices [KKP03], SAs for alignments [NPL+13], SAs for tries [FLMM05], and SAs for graphs [BCFR18].

In the majority of the above applications, suffix array construction and query time determine the complexity of the entire algorithm. Unsurprisingly, understanding the tradeoff between the space needed to represent the suffix array, its query time, and its construction time and working space, is one of the most studied problems in stringology [GV05, FLMM05, GGV03, Sad02, HSS09, Mäk03, HLSS03, LSSY02, MN05, FGGV06, ABM08, Gro11, Bel14, BN14, GNF14, GNP15, MNN17, NM07, Kem19, GNP20, MNN20, GHN20, MSST20, CGN21, KK23a, KK23b].

In this paper, we address the above question in its most fundamental and widely studied variant: What are the optimal query time, space usage, and construction time/space for a data structure that, for any static text $T \in [0\,..\,\sigma)^n$, can answer suffix array (SA) queries for the text T (that is, given any $i \in [1\,..\,n]$, return $\mathrm{SA}_T[i]$)?

The standard suffix array (SA) implementation [MM90] requires $\Theta(n \log n)$ bits, storing each entry $\mathrm{SA}_T[i]$ with $\lceil \log n \rceil$ bits. A major breakthrough, discovered independently by Ferragina and Manzini [FM00] and by Grossi and Vitter [GV00], demonstrated SA representations using $\mathcal{O}(n \log \sigma)$ bits, equivalent to $\mathcal{O}(n/\log_\sigma n)$ machine words, with query support in $\mathcal{O}(\log^\epsilon n)$ time for any constant $\epsilon > 0$. These structures, known as the *FM-index* [FM00] and the *Compressed Suffix Array (CSA)* [GV00], respectively, match the asymptotic space of storing T, where each symbol requires $\Theta(\log \sigma)$ bits.

Following these results, researchers turned their attention to optimizing the remaining aspects of the

suffix array, with the most pressing being the time and working space usage of algorithms constructing data structures with SA functionality. Given that both the input text and the output SA representation use only $\mathcal{O}(n \log \sigma)$ bits, an optimal $\mathcal{O}(n/\log_\sigma n)$ construction time is feasible in principle. The first efficient construction [HSS09] achieved $\mathcal{O}(n \log\log \sigma)$ time while preserving the $\mathcal{O}(n \log \sigma)$-bit working space. Subsequent improvements reduced this time to randomized $\mathcal{O}(n)$ [Bel14], deterministic $\mathcal{O}(n)$ [MNN17], and finally $\mathcal{O}(n \min(1, \log \sigma/\sqrt{\log n}))$ [KK23a]. The latest structure in [KK23a] achieves the optimal size of $\mathcal{O}(n \log \sigma)$ bits, answers SA queries on T in $\mathcal{O}(\log^\epsilon n)$ time, and, given the text T represented in $\mathcal{O}(n \log \sigma)$ bits, can be built in $\mathcal{O}(n \min(1, \log \sigma/\sqrt{\log n}))$ time and within the optimal $\mathcal{O}(n \log \sigma)$-bit working space. For example, when $\sigma = 2$, the structure uses $\mathcal{O}(n)$ bits (that is, $\mathcal{O}(n/\log n)$ machine words) and the construction time is $\mathcal{O}(n/\sqrt{\log n})$.

Despite all these advances, the tradeoff between SA space, query time, and construction time/working space is still not fully understood. Each of the existing SA structures employs distinct techniques: the FM-index relies on the Burrows–Wheeler Transform (BWT) and rank queries [FM00], while the CSA is built on the Ψ function [GV00]. Recent work introduces yet another paradigm, reducing SA queries to *prefix rank* and *select queries* [KK23a]. However, the lack of bidirectional reductions between these approaches leaves open a fundamental question: Are there other, still undiscovered techniques that could further reduce the space, query time, or construction time of space-efficient SAs, or have we reached the theoretical limit? Answering this question has immediate consequences for dozens of other problems and is key to understanding more complex variants of suffix arrays. We thus pose our question as:

Is there a query whose complexity governs the time/space tradeoff for space-efficient SA representation?

Our Results After 35 years since the discovery of the suffix array, we prove the first bidirectional reduction, showing that suffix array queries are nearly perfectly equivalent to *prefix select* queries [KK23a] in all four aspects (space usage, query time, construction time, and construction working space). In other words, while the reduction in [KK23a] proved that prefix rank/select queries provide *one* way to achieve efficient SA queries, we establish here that, up to a very small additive term in the query time, they in fact capture *every possible* approach to SA structures and their constructions. Hence, to obtain future tradeoffs, it essentially suffices to focus on prefix select queries.

Our reduction distills the core difficulty of the SA indexing problem—which involves handling complex situations such as suffixes sharing long prefixes—into a nearly perfectly computationally equivalent, yet much more straightforward, problem on short bitstrings. Such reductions are important because they reshape our understanding of the problem, allowing researchers to focus on significantly simpler tasks without losing essentially any generality. Indeed, as demonstrated by prior work [KK23a], the tradeoff for prefix select queries that leads to the state-of-the-art parameters of suffix array representations takes only around *one page* to develop (see [KK23a, Proposition 4.3]).

To state our results in more detail, let us, for simplicity, consider the case of a binary alphabet ($\sigma = 2$).[1] We now formally define the problems studied in this paper. As in previous works [HSS09, Bel14, MNN17, KK23a], we assume that strings in size-N instances of both problems are represented in the *packed* form, in which a string $S \in [0\,..\,\sigma)^n$ is stored using $\Theta(\lceil n/\log_\sigma N \rceil)$ machine words, with each word storing $\mathcal{O}(\log_\sigma N)$ characters from the alphabet $[0\,..\,\sigma)$.

Indexing for Suffix Array Queries**:** Given a binary text $T \in \{0,1\}^n$, represented using $\mathcal{O}(n)$ bits, construct a data structure that, given any $i \in [1\,..\,n]$, returns $\mathrm{SA}_T[i]$, i.e., the starting position of the lexicographically ith smallest suffix of T. We define the input size as $N := n$ bits, which matches the length of the text T.

Indexing for Prefix Select Queries**:** Given a sequence $W[1\,..\,m]$ of $m \geq 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$ (each represented using $\mathcal{O}(\ell)$ bits), construct a data structure that, given an $\mathcal{O}(\ell)$-bit representation of any string $X \in \{0,1\}^{\leq \ell}$ and any integer $r \in [1\,..\,|\{j \in [1\,..\,m] : X \text{ is a prefix of } W[j]\}|]$, returns the rth smallest element of the set $\{j \in [1\,..\,m] : X \text{ is a prefix of } W[j]\}$. We define the input size as $N := m \cdot \ell$ bits.

Our main result can be summarized in the following theorem:

[1] We focus on strings over a binary alphabet $\Sigma = \{0,1\}$, because, as proved later in this paper, any larger alphabet can be efficiently reduced to the binary case; see, e.g., Sections 4.2.3 and 4.3.3.

String Problem	Prefix Problem	Reference
Suffix Array Queries	Prefix Select Queries	Section 4
Inverse Suffix Array Queries	Prefix Special Rank Queries	Section 5
Pattern Ranking Queries	Prefix Rank Queries	Section 6
Lex-Range Reporting Queries	Prefix Range Reporting Queries	Section 7
Lex-Range Emptiness Queries	Prefix Range Emptiness Queries	Section 8
Lex-Range Minimum Queries	Prefix Range Minimum Queries	Section 9

Table 1: Near-perfect equivalences established in this paper between central string-processing problems (left) and fundamental prefix problems (right). Additionally, in Section 10, we prove that the problem of indexing for Pattern Ranking Queries is perfectly equivalent to the problem of Pattern SA-Interval Queries.

Theorem 1.1 (Main result of this paper)**.** *The problems of* Indexing for Suffix Array Queries *and* Indexing for Prefix Select Queries *are equivalent in the following sense. Consider any data structure that, for an input of at most N bits to one of the problems, achieves the following complexities:*

- *space usage of $S(N)$ bits,*
- *preprocessing time $P_t(N)$,*
- *preprocessing space of $P_s(N)$ bits (working space),*
- *query time $Q(N)$.*

Then, for every integer N', there exists $N = \mathcal{O}(N')$ such that, given an input instance of at most N' bits to the other problem, we can construct, in $\mathcal{O}(P_t(N))$ time and using $\mathcal{O}(P_s(N))$ bits of working space, a data structure of size $\mathcal{O}(S(N))$ bits that answers queries in $\mathcal{O}(\log\log N + Q(N))$ time.

Beyond Suffix Array Queries The above reduction raises natural questions: Are other fundamental string queries nearly equivalent to certain prefix queries? Conversely, are basic prefix queries nearly equivalent to natural string queries? For example, one may ask these questions for *Inverse Suffix Array* (ISA) queries (given a position $j \in [1\,..\,n]$, return the index $i \in [1\,..\,n]$ such that $\mathrm{SA}_T[i] = j$) and for *prefix rank* queries (given $i \in [1\,..\,m]$ and $X \in \{0,1\}^{\le \ell}$, compute $\mathsf{prefix\text{-}rank}_W(i, X) := |\{j \in [1\,..\,i] : X \text{ is a prefix of } W[j]\}|$).

Remarkably, the answer to both questions is affirmative. In total, we establish *six* equivalences between fundamental string queries, such as SA and ISA, and basic prefix queries, including prefix select and prefix rank. These six problems and their corresponding prefix formulations are summarized in Table 1.

Perhaps surprisingly, although ISA queries were previously answered via a reduction to prefix rank queries [KK23a], we do not obtain the converse reduction. Instead, we show that ISA queries correspond to what we call *Prefix Special Rank* queries. A prefix special rank query is defined on the same instance as prefix rank and select queries; given an index $i \in [1\,..\,m]$ and a position $p \in [0\,..\,\ell]$, it returns $\mathsf{prefix\text{-}rank}_W(i, W[i][1\,..\,p])$ (see Definition 2.8). This equivalence is analogous to Theorem 1.1, incurring only an additional $\mathcal{O}(\log\log N)$ factor in query time.

By contrast, the fully general prefix rank queries are equivalent to another fundamental problem, *Pattern Ranking*: computing the lexicographic rank of a pattern $P \in \Sigma^m$ among all text suffixes (i.e., counting suffixes lexicographically smaller than P). This equivalence is more subtle because the input pattern may occupy $\omega(1)$ machine words (equivalently, $\omega(\log N)$ bits) under the standard word-RAM model with word size $\Theta(\log N)$. We therefore state both problems more formally below to clarify how this input-size issue affects the reductions.

Indexing for Pattern Ranking Queries**:** Given a binary text $T \in \{0,1\}^n$, represented using $\mathcal{O}(n)$ bits, construct a data structure that, given any pattern $P \in \{0,1\}^m$ represented in $\mathcal{O}(m)$ bits, returns $|\{i \in [1\,..\,n] : T[i\,..\,n] \prec P\}|$. We define the input size as $N := n$ bits and the pattern size as $M := m$ bits.

Indexing for Prefix Rank Queries**:** Given a sequence $W[1\,..\,m]$ of $m \ge 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$ (each represented using $\mathcal{O}(\ell)$ bits), construct a data structure that, given any $i \in [0\,..\,m]$ and an $\mathcal{O}(\ell)$-bit representation of any $X \in \{0,1\}^{\le \ell}$, returns $\mathsf{prefix\text{-}rank}_W(i, X) = |\{j \in [1\,..\,i] :$

X is a prefix of $W[j]\}|$. We define the input size as $N := m \cdot \ell$ bits.

Unlike the earlier two-step reductions, our reduction sequence involves *three* problems, with a special variant of pattern ranking as the intermediate problem. In more detail:

1. We prove that, given any data structure for prefix rank queries with complexities $S(N)$, $P_t(N)$, $P_s(N)$, and $Q(N)$ (defined as in Theorem 1.1), we can derive a structure for pattern ranking with complexities $S'(N) = \mathcal{O}(S(N'))$, $P'_t(N) = \mathcal{O}(P_t(N'))$, $P'_s(N) = \mathcal{O}(P_s(N'))$, and $Q'(N, M) = \mathcal{O}(\log\log N + M/\log N + Q(N'))$ for some $N' = \mathcal{O}(N)$, where $Q'(N, M)$ is the time complexity of answering ranking queries on M-bit patterns.
2. We then prove that, given any structure for pattern ranking queries *restricted for patterns of length* $\mathcal{O}(\log N)$ that achieves complexities $S(N)$, $P_t(N)$, $P_s(N)$, and $Q(N)$, we can achieve $S'(N) = \mathcal{O}(S(N'))$, $P'_t(N) = \mathcal{O}(P_t(N'))$, $P'_s(N) = \mathcal{O}(P_s(N'))$, and $Q'(N) = \mathcal{O}(Q(N'))$ for prefix rank for some $N' = \mathcal{O}(N)$.
3. To complete the cycle, we observe that, by definition, if we can answer pattern ranking queries in $Q(N, M) = \mathcal{O}(M/\log N + Q(N))$ time for patterns of any length M, then we can answer them for patterns of length $M = \mathcal{O}(\log N)$ in $\mathcal{O}(Q(N))$ time.

This cycle of reductions thus shows that: (1) the pattern ranking problem achieves maximal hardness (that is, the overhead beyond the time necessary to read the query input) already for patterns of length $\mathcal{O}(\log N)$, and (2) this restricted pattern ranking problem is nearly perfectly equivalent to prefix rank queries.

More Complex Prefix Queries Having established the string problems corresponding to the prefix rank, prefix select, and prefix special rank problems, we turn our attention to more complex prefix *range* queries. Let again $W[1\,..\,m]$ be a sequence of m binary strings of length $\ell := 1 + \lfloor \log m \rfloor$, and let $b, e \in [0\,..\,m]$ and $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$. The next two types of queries we study ask about the set $\mathcal{J} := \{j \in (b\,..\,e] : X \text{ is a prefix of } W[j]\}$ described using the positions b, e and the string X. Specifically, in *Prefix Range Reporting*, we ask to report all elements of the set $\mathcal{J}$. In *Prefix Range Emptiness*, we simply ask whether $\mathcal{J} \neq \emptyset$. In addition to these two prefix range queries, recent work on sublinear-time LZ77 factorization [KK24] defined yet another, more complex, variant of prefix queries: *Prefix Range Minimum Queries (Prefix RMQ)*. In the prefix RMQ problem, in addition to the sequence $W[1\,..\,m]$ of m strings, we are also given an array $A[1\,..\,m]$ of m integers. At query time, given any $b, e \in [0\,..\,m]$ and $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$, we ask to return $\arg\min\{A[j] : j \in \mathcal{J}\}$. In other words, we seek the position of the smallest value in the array A restricted to indices $j \in (b\,..\,e]$ for which the corresponding string $W[j]$ has X as a prefix.

In this paper, we prove that each of these three types of prefix range queries is nearly perfectly equivalent (again, up to an additional $\mathcal{O}(\log\log N)$ term in the query time) to a distinct fundamental string problem in which, given two patterns $P_1, P_2 \in \{\mathtt{0},\mathtt{1}\}^*$, we ask about the set $\mathrm{LexRange}(P_1, P_2, T) := \{j \in [1\,..\,n] : P_1 \preceq T[j\,..\,n] \prec P_2\}$ (Definition 2.4), that is, the contiguous block in the suffix array of T containing all suffixes that are lexicographically between P_1 and P_2 (see Remark 2.5). These types of problems arise, for example, in algorithms for suffix sorting and in the construction of the Burrows–Wheeler Transform (BWT) [KKP14, FGM12, KKP15]. We show that:

- Prefix range reporting is nearly perfectly equivalent to enumerating $\mathrm{LexRange}(P_1, P_2, T)$ (Section 7).
- Prefix range emptiness is nearly perfectly equivalent to checking if $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$ (Section 8).
- Prefix range minimum is nearly perfectly equivalent to computing $\min \mathrm{LexRange}(P_1, P_2, T)$ (Section 9).

These six near-perfect equivalences summarized in Table 1 unveil a remarkably clean structure underlying many central string problems, which, at their core, all reduce to much simpler rank/select and range queries over short bitstrings.

Alphabet Reductions As an auxiliary result of the above reductions, we establish a very useful fact: All twelve of the studied problems, which naturally generalize to strings over an integer alphabet $[0\,..\,\sigma)$, already reach maximal computational hardness for the binary alphabet ($\sigma = 2$). Moreover, any instance with a larger alphabet can be efficiently reduced to the binary case.

To illustrate this reduction principle, let us consider the most fundamental of the twelve problems: Indexing a string $T \in [0\,..\,\sigma)^n$ for suffix array queries. In [KK23a], it was shown that, given the $\mathcal{O}(n \log \sigma)$-bit representation of a text $T \in [0\,..\,\sigma)^n$ and any constant $\epsilon > 0$, we can construct a data structure of size $\mathcal{O}(n \log \sigma)$ bits in $\mathcal{O}((n \log \sigma)/\sqrt{\log n})$ time and using $\mathcal{O}(n \log \sigma)$ bits of working space. This structure allows

us to retrieve $\mathrm{SA}_T[i]$ for any $i \in [1\,..\,n]$ in $\mathcal{O}(\log^{\epsilon} n)$ time. For $\sigma = 2$, we thus obtain an $\mathcal{O}(n)$-bit structure which can be constructed in $\mathcal{O}(n/\sqrt{\log n})$ time using $\mathcal{O}(n)$ bits of working space. In this paper, we show that to achieve the same tradeoff for any alphabet size $\sigma \geq 2$, it would suffice to develop the above result only for $\sigma = 2$. More precisely, in Section 4.2.3 we prove that, given the packed representation of $T \in [0\,..\,\sigma)^n$, we can compute, in $\mathcal{O}(n/\log_\sigma n)$ time, integers α, β, γ, μ, and the packed representation of a binary string T_{bin} of length $|T_{\mathrm{bin}}| = \Theta(n \log \sigma)$ such that, for every $i \in [1\,..\,n]$, it holds

$$\mathrm{SA}_T[i] = \frac{\mathrm{SA}_{T_{\mathrm{bin}}}[\alpha + i] - \beta}{\gamma} + \mu.$$

Thus, even if the construction from [KK23a] were available only for the binary alphabet, our reduction would allow us to extend it to larger alphabets in a black-box manner, significantly simplifying the original algorithm. In Sections 4.3.3, 5.2.2, 5.3.2, 6.2.2, 6.3.2, 7.2.2, 7.3.2, 8.2.2, 8.3.2, 9.2.3, and 9.3.2, we also establish analogous results for the remaining problems analyzed in this paper.

Related Work Many of the queries studied in this paper have also been examined in the compressed setting, where the input string is highly repetitive, and the goal is to answer queries using space close to the size of the text in compressed form. Parallel work [KK26] significantly advances the understanding of string queries in this setting, showing that in $\mathcal{O}(\delta(T) \log^{\mathcal{O}(1)} n)$ space (a well-studied and widely accepted space bound for compressed data structures), central string queries fall into two categories: those with complexity $\Theta(\frac{\log n}{\log \log n})$ (including suffix array (SA), inverse suffix array (ISA), longest common prefix (LCP) array, and longest common extension (LCE) queries) and those with complexity $\Theta(\log \log n)$ (including the Burrows–Wheeler transform (BWT), permuted longest common prefix (PLCP) array, last-to-first (LF) mapping, inverse last-to-first (LF^{-1}) mapping, lexicographical predecessor (Φ), and inverse lexicographical predecessor (Φ^{-1}) queries).

Another fundamental data structure in string processing that has been extensively studied in terms of trade-offs between space usage, query time, and construction efficiency is the suffix tree—a predecessor of the suffix array. Over the years, numerous (unidirectional) reductions have been demonstrated. For example, Sadakane [Sad02] expresses the runtime of his *Compressed Suffix Tree (CST)* operations primarily in terms of the so-called Φ and Ψ functions. Russo, Navarro, and Oliveira [RNO11] achieve different trade-offs using the same functions. Fischer, Mäkinen, and Navarro [FMN09] use a different set of operations for their CST, expressing all runtimes in terms of suffix array queries, range minimum queries (RMQ), next smaller value (NSV), and previous smaller value (PSV) queries. Gagie, Navarro, and Prezza [GNP20] use a similar set of operations, additionally augmented with the LF-mapping query and access to the longest common prefix (LCP) array.

Similar reductions for *algorithms* are less common. A recent step in this direction is [KK25], which develops a comprehensive hierarchy among many string and combinatorial problems (such as LZ77 factorization, BWT construction, batched ISA/LPF queries, longest common factor, and inversion counting) whose state-of-the-art algorithms run in $\mathcal{O}(n/\sqrt{\log n})$ time for inputs of n bits (i.e., $\Theta(n/\log n)$ machine words).

Organization of the Paper First, in Section 2, we formally introduce the notation and definitions used throughout the paper. In Section 3, we provide an overview of our techniques. Next, in Sections 4 to 9, we present our main equivalences (see Table 1). Finally, in Section 10, we present an additional equivalence between Pattern Ranking Queries and Pattern SA-Interval Queries.

2 Preliminaries

2.1 Basic Definitions

A *string* is a finite sequence of characters from a given *alphabet* Σ. The length of a string S is denoted $|S|$. For $i \in [1\,..\,|S|]$,[2] the ith character of S is denoted $S[i]$. A *substring* or a *factor* of S is a string of the form $S[i\,..\,j) = S[i]S[i+1]\cdots S[j-1]$ for some $1 \leq i \leq j \leq |S|+1$. Substrings of the form $S[1\,..\,j)$ and $S[i\,..\,|S|+1)$ are called *prefixes* and *suffixes*, respectively. We use $\overline{S}$ to denote the *reverse* of S, i.e., $S[|S|]\cdots S[2]S[1]$. We denote the *concatenation* of two strings U and V, that is, the string $U[1]\cdots U[|U|]V[1]\cdots V[|V|]$, by UV or

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

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

Figure 1: A list of all sorted suffixes of $T = \texttt{bbabaabababababaababa}$ along with the suffix array.

$U \cdot V$. Furthermore, $S^k = \bigodot_{i=1}^{k} S$ is the concatenation of $k \in \mathbb{Z}_{\geq 0}$ copies of S; note that $S^0 = \varepsilon$ is the *empty string*. A nonempty string S is said to be *primitive* if it cannot be written as $S = U^k$, where $k \geq 2$. An integer $p \in [1\,..\,|S|]$ is a *period* of S if $S[i] = S[i+p]$ holds for every $i \in [1\,..\,|S| - p]$. We denote the smallest period of S as $\mathrm{per}(S)$. For every $S \in \Sigma^+$, we define the infinite power S^∞ so that $S^\infty[i] = S[1 + (i-1) \bmod |S|]$ for $i \in \mathbb{Z}$. In particular, $S = S^\infty[1\,..\,|S|]$. By $\mathrm{lcp}(U, V)$ we denote the length of the longest common prefix of U and V. For any string $S \in \Sigma^*$ and any $j_1, j_2 \in [1\,..\,|S|]$, we denote $\mathrm{LCE}_S(j_1, j_2) = \mathrm{lcp}(S[j_1\,..\,|S|], S[j_2\,..\,|S|])$. We use $\preceq$ to denote the order on Σ, extended to the *lexicographic* order on Σ^* so that $U, V \in \Sigma^*$ satisfy $U \preceq V$ if and only if either (a) U is a prefix of V, or (b) $U[1\,..\,i) = V[1\,..\,i)$ and $U[i] \prec V[i]$ holds for some $i \in [1\,..\,\min(|U|, |V|)]$.

Definition 2.1 (Pattern occurrences and SA-interval)**.** For any $P \in \Sigma^*$ and $T \in \Sigma^*$, we define

$$\begin{aligned}
\mathrm{Occ}(P, T) &= \{j \in [1\,..\,|T|] : j + |P| \leq |T| + 1 \text{ and } T[j\,..\,j+|P|) = P\},\\
\mathrm{RangeBeg}(P, T) &= |\{j \in [1\,..\,|T|] : T[j\,..\,|T|] \prec P\}|,\\
\mathrm{RangeEnd}(P, T) &= \mathrm{RangeBeg}(P, T) + |\mathrm{Occ}(P, T)|.
\end{aligned}$$

2.2 Suffix Array

Definition 2.2 (Suffix array and its inverse [MM93])**.** For any string $T \in \Sigma^n$ of length $n \geq 1$, the *suffix array* $\mathrm{SA}_T[1\,..\,n]$ of T is a permutation of $[1\,..\,n]$ such that $T[\mathrm{SA}_T[1]\,..\,n] \prec T[\mathrm{SA}_T[2]\,..\,n] \prec \cdots \prec T[\mathrm{SA}_T[n]\,..\,n]$, i.e., $\mathrm{SA}_T[i]$ is the starting position of the lexicographically ith suffix of T; see Fig. 1 for an example. The *inverse suffix array* $\mathrm{ISA}_T[1\,..\,n]$ is the inverse permutation of SA_T, i.e., $\mathrm{ISA}_T[j] = i$ holds if and only if $\mathrm{SA}_T[i] = j$. Intuitively, $\mathrm{ISA}_T[j]$ stores the lexicographic *rank* of $T[j\,..\,n]$ among the suffixes of T.

Remark 2.3. Note that the two values $\mathrm{RangeBeg}(P, T)$ and $\mathrm{RangeEnd}(P, T)$ (Definition 2.1) are the endpoints of the so-called *SA-interval* representing the occurrences of P in T, i.e.,

$$\mathrm{Occ}(P, T) = \{\mathrm{SA}_T[i] : i \in (\mathrm{RangeBeg}(P, T)\,..\,\mathrm{RangeEnd}(P, T)]\}$$

holds for every $T \in \Sigma^+$ and $P \in \Sigma^*$, including when $P = \varepsilon$ and when $\mathrm{Occ}(P, T) = \emptyset$.

Definition 2.4 (Lexicographical range)**.** For any $P_1, P_2 \in \Sigma^*$ and $T \in \Sigma^*$, we define

$$\mathrm{LexRange}(P_1, P_2, T) = \{j \in [1\,..\,|T|] : P_1 \preceq T[j\,..\,|T|] \prec P_2\}.$$

Remark 2.5. Observe that for any $P_1, P_2 \in \Sigma^*$ and any nonempty $T \in \Sigma^+$, it holds

$$\mathrm{LexRange}(P_1, P_2, T) = \{\mathrm{SA}_T[i] : i \in (\mathrm{RangeBeg}(P_1, T)\,..\,\mathrm{RangeBeg}(P_2, T)]\}.$$

2.3 Rank and Selection Queries

Definition 2.6 (Rank and selection queries)**.** For a string $S \in \Sigma^n$, we define:

$\mathsf{rank}_S(j, a)$**:** Given $a \in \Sigma$ and $j \in [0\,..\,n]$, compute $|\{i \in [1\,..\,j] : S[i] = a\}|$.

$\mathsf{special\text{-}rank}_S(j)$**:** Given $j \in [1\,..\,n]$, compute $|\{i \in [1\,..\,j] : S[i] = S[j]\}|$.

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

Theorem 2.7 (Rank and selection queries in bitvectors [BGKS15, Cla98, Jac89, MNV16])**.** *For every string $S \in \{0, 1\}^*$, there exists a data structure of $\mathcal{O}(|S|)$ bits answering rank and selection queries in $\mathcal{O}(1)$ time. Moreover, given the packed representations of m binary strings of total length n, the data structures for all these strings can be constructed in $\mathcal{O}(m + n/\log n)$ time.*

2.4 Prefix Queries

Definition 2.8 (Prefix rank and selection queries [KK23a])**.** For a sequence $S \in (\Sigma^*)^m$ of strings over alphabet Σ, we define:

$\mathsf{prefix\text{-}rank}_S(j, X)$**:** Given $X \in \Sigma^*$ and $j \in [0\,..\,m]$, compute $|\{i \in [1\,..\,j] : X \text{ is a prefix of } S[i]\}|$.

$\mathsf{prefix\text{-}special\text{-}rank}_S(j, p)$**:** Given $j \in [1\,..\,m]$ and $p \in [0\,..\,|S[j]|]$, compute $\mathsf{prefix\text{-}rank}_S(j, S[j][1\,..\,p])$.

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

Definition 2.9 (Prefix range minimum queries (prefix RMQ) [KK24])**.** Let $A \in \mathbb{Z}^m_{\geq 0}$ be a sequence of m nonnegative integers and $S \in (\Sigma^*)^m$ be a sequence of m strings over alphabet Σ. For every $b, e \in [0\,..\,m]$ and $X \in \Sigma^*$ we define[3]

$$\mathsf{prefix\text{-}rmq}_{A,S}(b, e, X) := \begin{cases} \arg\min_{i \in \mathcal{I}} A[i] & \text{if } \mathcal{I} \neq \emptyset, \\ \infty & \text{otherwise,} \end{cases}$$

where $\mathcal{I} = \{i \in (b\,..\,e] : X \text{ is a prefix of } S[i]\}$.

2.5 Range Queries

Definition 2.10 (Range counting and selection)**.** Let $A[1\,..\,m]$ be an array of $m \geq 0$ nonnegative integers. For every $j \in [0\,..\,m]$ and $v \geq 0$, we define

$$\mathsf{range\text{-}count}_A(j, v) := |\{j \in (0\,..\,j] : A[i] \geq v\}|.$$

For every $v \geq 0$ and every $r \in [1\,..\,\mathsf{range\text{-}count}_A(m, v)]$, by $\mathsf{range\text{-}select}_A(r, v)$ we define the rth smallest element of $\{i \in [1\,..\,m] : A[i] \geq v\}$.

Definition 2.11 (Range minimum queries (RMQ))**.** Let $A[1\,..\,m]$ be a sequence of m integers. For every $b, e \in [0\,..\,m]$, we define

$$\mathsf{rmq}_A(b, e) := \begin{cases} \arg\min_{i \in (b\,..\,e]} A[i] & \text{if } b < e, \\ \infty & \text{otherwise.} \end{cases}$$

Definition 2.12 (Three-sided RMQ)**.** Let $A[1\,..\,m]$ and $B[1\,..\,m]$ be two arrays of m integers. For every $b, e \in [0\,..\,m]$ and $v \in \mathbb{Z}$, we define

$$\mathsf{three\text{-}sided\text{-}rmq}_{A,B}(b, e, v) := \begin{cases} \arg\min_{i \in \mathcal{I}} A[i] & \text{if } \mathcal{I} \neq \emptyset, \\ \infty & \text{otherwise,} \end{cases}$$

where $\mathcal{I} = \{i \in (b\,..\,e] : B[i] \geq v\}$.

Theorem 2.13 ([FH11])**.** *For every array $A[1\,..\,m]$ of m integers, there is a data structure of $\mathcal{O}(m)$ bits that answers range minimum queries over A in $\mathcal{O}(1)$ time and can be constructed in $\mathcal{O}(m)$ time.*

[3] We assume that $\arg\min_{x \in S} f(x)$ returns the *smallest* $y \in S$ such that $f(y) = \min\{f(x) : x \in S\}$.

Corollary 2.14. *An array $A[1\,..\,m]$ of m integers can be preprocessed in $\mathcal{O}(m)$ time, so that given any $b, e \in [0\,..\,m]$ and $v \geq 0$, we can compute the set $\mathcal{I} = \{i \in (b\,..\,e] : A[i] \geq v\}$ in $\mathcal{O}(1 + |\mathcal{I}|)$ time.*

Proof. The structure consists of a single component: the data structure from Theorem 2.13 using $O(m)$ bits.

The query algorithm is a recursive function that, given $b, e \in [0\,..\,m]$ and $v \geq 0$, returns $\{i \in (b\,..\,e] : A[i] \geq v\}$. If $b \geq e$, we immediately exit. Otherwise, compute $i_{\min} = \mathsf{rmq}_A(b, e)$. If $A[i_{\min}] \geq v$, report $i_{\min}$ and recurse on $(b\,..\,i-1]$ and $(i\,..\,e]$. If $A[i_{\min}] < v$, we again exit. In total, this takes $\mathcal{O}(1 + |\mathcal{I}|)$ time.

The construction of the data structure takes $\mathcal{O}(m)$ time (see Theorem 2.13). □

Theorem 2.15 ([KK23a])**.** *An array $A[1\,..\,m']$ of $m' \leq m$ nonnegative integers such that $\sum_{i=1}^{m'} A[i] = \mathcal{O}(m \log m)$ can be preprocessed in $\mathcal{O}(m)$ time, so that two-sided range counting and range selection queries (Definition 2.10) on A can be answered in $\mathcal{O}(\log \log m)$ time and $\mathcal{O}(1)$ time, respectively.*

Theorem 2.16 ([KK24])**.** *Any two arrays $A[1\,..\,m']$ and $B[1\,..\,m']$ of $m' \leq m$ nonnegative satisfying $\max_{i=1}^{m'} A[i] = \mathcal{O}(m \log m)$ and $\sum_{i=1}^{m'} B[i] = \mathcal{O}(m \log m)$ can be preprocessed in $\mathcal{O}(m)$ time, so that three-sided RMQ queries (Definition 2.12) on A and B can be answered in $\mathcal{O}(\log \log m)$ time.*[4]

2.6 String Synchronizing Sets

String synchronizing sets [KK19] allow for a locally-consistent selection of positions in a given text T. The underlying parameter τ governs the context size (with respect to which the selection is consistent) and the achievable size of the synchronizing set.

Definition 2.17 (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 $\mathsf{S} \subseteq [1\,..\,n - 2\tau + 1]$ is called a *τ-synchronizing set* of T if it satisfies the following *consistency* and *density* conditions:

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

$$\mathsf{R}(\tau, T) := \{i \in [1\,..\,n - 3\tau + 2] : \mathrm{per}(T[i\,..\,i + 3\tau - 2]) \leq \tfrac{1}{3}\tau\}.$$

Remark 2.18. In most applications, we want to minimize $|\mathsf{S}|$. Note, however, that the density condition imposes a lower bound $|\mathsf{S}| = \Omega(\frac{n}{\tau})$ for strings of length $n \geq 3\tau - 1$ that do not contain substrings of length $3\tau - 1$ with period at most $\frac{1}{3}\tau$. Thus, we cannot hope to achieve an upper bound improving in the worst case upon the following ones.

Theorem 2.19 ([KK19, Proposition 8.10])**.** *For every string T of length n and parameter $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$, there exists a τ-synchronizing set S of size $|\mathsf{S}| = \mathcal{O}\left(\frac{n}{\tau}\right)$. Moreover, if $T \in [0\,..\,\sigma)^n$, where $\sigma = n^{\mathcal{O}(1)}$, such S can be deterministically constructed in $\mathcal{O}(n)$ time.*

Theorem 2.20 ([KK19, Theorem 8.11])**.** *For every constant $\mu < \frac{1}{5}$, given the packed representation of a text $T \in [0\,..\,\sigma)^n$ and a positive integer $\tau \leq \mu \log_\sigma n$, one can deterministically construct in $\mathcal{O}(\frac{n}{\tau})$ time a τ-synchronizing set of T of size $\mathcal{O}(\frac{n}{\tau})$.*

2.7 Model of Computation

We use the standard word RAM model of computation [Hag98] with w-bit *machine words*, where $w \geq \log n$, and all standard bitwise and arithmetic operations take $\mathcal{O}(1)$ time. Unless explicitly stated otherwise, we measure space complexity in machine words.

In the RAM model, strings are usually represented as arrays, with each character occupying one memory cell. A single character, however, only needs $\lceil \log \sigma \rceil$ bits, which might be much less than w. We can therefore store (the *packed representation* of) a text $T \in [0\,..\,\sigma)^n$ using $\mathcal{O}(\lceil \frac{n \log \sigma}{w} \rceil)$ words.

[4] The structure in [KK24] assumes that the arguments b, e, v satisfy $\{i \in (b\,..\,e] : B[i] \geq v\} \neq \emptyset$. To adapt the structure from [KK24] to the more general definition of prefix RMQ queries (Definition 2.9), we simply include the structure from Theorem 2.15 as a component of the structure from Theorem 2.16. At the beginning of the query, we check if $\{i \in (b\,..\,e] : B[i] \geq v\} \neq \emptyset$ in $\mathcal{O}(\log \log m)$ time using two range counting queries.

3 Technical Overview

3.1 Equivalence of Suffix Array and Prefix Select Queries

In this section, we outline the equivalence between indexing for suffix array (SA) queries and indexing for prefix select queries. Our focus is on the reduction showing how prefix select queries can be answered using suffix array queries. The reduction in the opposite direction (from suffix array to prefix select queries), which motivated our research, was proved in [KK23a]. In Section 4.3, we augment this result with the alphabet reduction for prefix select queries (Section 4.3.3), obtaining a clean reduction between the two problems on a binary alphabet (in particular, without requiring an extra alphabet symbol to serve as a sentinel at the end of the text).

Reducing Prefix Select to Suffix Array Queries Let $W[1\,..\,m]$ be an array of $m \geq 1$ bitstrings of length $\ell = 1 + \lfloor \log m \rfloor$, each represented in $\mathcal{O}(1)$ machine words; that is, the input to the prefix select indexing problem (see Section 4.1). Consider the string $T = \bigodot_{i=1}^{m} \overline{W[i]} \cdot \mathtt{4} \cdot (i+4)$ over the alphabet $\{\mathtt{0}, \mathtt{1}, \ldots, m+4\}$, where $\overline{Y}$ denotes the reversal of Y.

Consider any $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$, and let $q = \mathsf{prefix\text{-}rank}_W(m, X)$ be the number of indices $i \in [1\,..\,m]$ such that X is a prefix of $W[i]$. Moreover, let $(p_1, p_2, \ldots, p_q)$ be the sequence of these indices in increasing order; formally, $p_i = \mathsf{prefix\text{-}select}_W(i, X)$. Our reduction is based on the following observation:

Observation: The lexicographically sorted sequence of suffixes starting in $\mathrm{Occ}(\overline{X} \cdot \mathtt{4}, T)$ *lets us identify the sequence* $(p_1, p_2, \ldots, p_q)$. There are exactly m occurrences of the symbol $\mathtt{4}$ in T. Thus, every occurrence of $\overline{X} \cdot \mathtt{4}$ aligns with the end of $\overline{W[i]} \cdot \mathtt{4}$ for some $i \in [1\,..\,m]$. The existence of such an occurrence is equivalent to X being a prefix of $W[i]$. Hence, $\mathrm{Occ}(\overline{X} \cdot \mathtt{4}, T) = \{p_i \cdot \delta - (1 + |X|) : i \in [1\,..\,q]\}$, where $\delta = \ell + 2$. Since $p_1 < p_2 < \cdots < p_q$, the elements of $\mathrm{Occ}(\overline{X} \cdot \mathtt{4}, T)$ appear in the same order in T and in its suffix array. Denoting $\mathrm{Occ}(\overline{X} \cdot \mathtt{4}, T) = \{a_1, a_2, \ldots, a_q\}$ with $a_1 < a_2 < \cdots < a_q$, we thus have $a_i = p_i \cdot \delta - (1 + |X|)$ for all $i \in [1\,..\,q]$, and also $\mathrm{SA}_T(b\,..\,b+q] = (a_1, \ldots, a_q)$, where $b = \mathrm{RangeBeg}(\overline{X} \cdot \mathtt{4}, T)$. Consequently, $\mathrm{SA}_T[b+i] = a_i = p_i \cdot \delta - (1 + |X|)$ holds for every $i \in [1\,..\,q]$. Since $\mathsf{prefix\text{-}select}_W(i, X) = p_i$ by definition, it follows that

$$\mathsf{prefix\text{-}select}_W(i, X) = p_i = \frac{\mathrm{SA}_T[b+i] + (1 + |X|)}{\delta} = \left\lceil \frac{\mathrm{SA}_T[b+i]}{\delta} \right\rceil,$$

where the last equality follows from $1 \leq 1 + |X| < \delta$.

The first challenge in this reduction is that the resulting string T is over a large alphabet Σ of size $|\Sigma| = \Theta(m)$. Any approach that maps each symbol to $\Theta(\log |\Sigma|)$ bits would produce a string of length $\Theta(|T| \cdot \log m) = \Theta(m \log^2 m)$, which is too large. This issue is resolved by slightly redefining T. For $i \in [0\,..\,m]$, define $s(i)$ as a string of length $k := 1 + \lfloor \log m \rfloor$, obtained by replacing every occurrence of $\mathtt{0}$ with $\mathtt{2}$ and every occurrence of $\mathtt{1}$ with $\mathtt{3}$ in the length-k binary representation of the integer i. (We include leading zeros to pad the string to the required length k.) Then, define

$$T = \bigodot_{i=1}^{m} \overline{W[i]} \cdot \mathtt{4} \cdot s(i-1).$$

The resulting string has an alphabet of size only five and retains all the properties required in the above analysis, except that the value of δ in the formula for $\mathsf{prefix\text{-}select}_W(i, X)$ becomes $\delta = \ell + k + 1$.

The next challenge in the reduction is that, to answer a prefix select query via a suffix array query (using the formula above), we need to quickly obtain the value $b = \mathrm{RangeBeg}(\overline{X} \cdot \mathtt{4}, T)$. Since the number of possible strings $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$ is $\Theta(2^\ell) = \Theta(m)$, we can store this information for all X. In general, it is not known how to compute all such values for an arbitrary text T. However, the symbol $\mathtt{4}$ in T provides separation that reduces the problem to: (1) computing the frequencies of all $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$ as prefixes of strings in $W[1\,..\,m]$, and then (2) computing a prefix sum over these frequencies during a postorder traversal of a complete binary tree of height ℓ. These computations can be performed in $\mathcal{O}(m)$ time; see Proposition 4.11.

The final challenge is reducing the string T to a binary alphabet. We now sketch a more general reduction proving that, given the packed representation of $T \in [0\,..\,\sigma)^n$, we can compute the packed representation of a binary string T_{bin} of length $|T_{\mathrm{bin}}| = \Theta(n \log \sigma)$ and integers α, β, γ, and μ such that, for every $i \in [1\,..\,n]$,

$$\mathrm{SA}_T[i] = \frac{\mathrm{SA}_{T_{\mathrm{bin}}}[\alpha + i] - \beta}{\gamma} + \mu.$$

For any $k \in \mathbb{Z}_{>0}$ and $x \in [0\,..\,2^k)$, let $\mathrm{bin}_k(x) \in \{\mathtt{0},\mathtt{1}\}^k$ denote the length-k string containing the binary representation of x (with leading zeros). For a string S over $\Sigma = [0\,..\,2^k)$, we then let $\mathrm{bin}_k(S) := \bigodot_{i=1}^{|S|} \mathrm{bin}_k(S[i])$. For the same k, x, and S, we define

$$\mathrm{ebin}_k(x) := \mathtt{1}^{k+1} \cdot \mathtt{0} \cdot \mathrm{bin}_k(x) \cdot \mathtt{0}, \qquad \text{and} \qquad \mathrm{ebin}_k(S) := \bigodot_{i=1}^{|S|} \mathrm{ebin}_k(S[i]).$$

With these definitions, we can state our alphabet reduction. For any $T \in [0\,..\,\sigma)^n$, we define $T_{\mathrm{bin}} = \mathrm{ebin}_k(T) = \bigodot_{i=1}^{n} \mathrm{ebin}_k(T[i])$, where $k = \lceil \log \sigma \rceil$. To emulate suffix array queries on T using T_{bin}, note that, for any $S_1, S_2 \in [0\,..\,\sigma)^*$, $S_1 \prec S_2$ holds if and only if $\mathrm{ebin}_k(S_1) \prec \mathrm{ebin}_k(S_2)$. This implies that the suffixes of T_{bin} starting at positions in the set $\mathcal{P} = \{1 + (i-1)\gamma : i \in [1\,..\,n]\}$ (where $\gamma = 2\lceil \log \sigma \rceil + 3$) occur in the same relative order as the corresponding suffixes in T. Moreover, these are precisely the suffixes prefixed with $\mathtt{1}^{k+1}$, so they occupy a contiguous block of $\mathrm{SA}_{T_{\mathrm{bin}}}$. It is easy to see that this block is located at the very end of the suffix array. Putting everything together, letting $\alpha = |T_{\mathrm{bin}}| - |T|$, we have for every $i \in [1\,..\,n]$ that $\mathrm{SA}_{T_{\mathrm{bin}}}[\alpha + i] = (\mathrm{SA}_T[i] - 1)\gamma + 1$. By rewriting this, we obtain $\mathrm{SA}_T[i] = (\mathrm{SA}_{T_{\mathrm{bin}}}[\alpha + i] - 1)/\gamma + 1$. Thus, we obtain our formula with $\beta = \mu = 1$, and α and γ defined as above. Finally, note that the string T_{bin} can be constructed from T using lookup tables in $\mathcal{O}(n/\log_\sigma n)$ time (see Proposition 4.15).

3.2 Equivalence of Lex-Range Reporting and Prefix Range Reporting Queries

In this section, we provide an overview of a novel equivalence between lex-range reporting and prefix range reporting queries. This reduction differs from similar reductions in earlier works (e.g., the reduction from pattern ranking to prefix rank presented in [KK23a]), where the query algorithm proceeds differently depending on whether the input pattern $P \in \Sigma^m$ is highly periodic or not. In the lex-range reporting queries considered here, the input consists of *two* patterns, P_1 and P_2, which may differ in their periodicity (e.g., only one may be periodic), leading to more complex query algorithms.

Reducing Lex-Range Reporting to Prefix Range Reporting Queries Consider a data structure answering prefix range reporting queries that, for any sequence W of $m \geq 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$, uses $S(m)$ space, takes $P_t(m)$ time and $P_s(m)$ working space to build, and, given any $b, e \in [0\,..\,m]$ and the packed representation of $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$, returns $\mathcal{I} := \{i \in (b\,..\,e] : X \text{ is a prefix of } W[i]\}$ in $Q_{\mathrm{base}}(m) + |\mathcal{I}| \cdot Q_{\mathrm{elem}}(m)$ time. In this section, we outline the proof that, for every $T \in \{\mathtt{0},\mathtt{1}\}^n$, there exists $m = \Theta(n/\log n)$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log n + P_t(m))$ time and using $\mathcal{O}(n/\log n + P_s(m))$ working space construct a data structure of size $\mathcal{O}(n/\log n + S(m))$ that, given the packed representations of any $P_1, P_2 \in \{\mathtt{0},\mathtt{1}\}^*$, computes $\mathcal{J} := \mathrm{LexRange}(P_1, P_2, T)$ (see Definition 2.4) in $\mathcal{O}(\log\log n + Q_{\mathrm{base}}(m) + |\mathcal{J}| \cdot Q_{\mathrm{elem}}(m) + |P_1|/\log n + |P_2|/\log n)$ time.

To simplify the reduction, we assume that $T[n] = \mathtt{2}$, i.e., that there is a sentinel symbol at the end of T. This increases the alphabet size, but since at the very end we reduce the alphabet back to binary, adding the sentinel does not affect the parameters of the reduction. We set $\sigma = 3$. We also assume that $n > \sigma^{13}$ and denote $\tau = \mu \log_\sigma n \geq 1$, where $\mu < \frac{1}{12}$ is a constant (such μ and τ exist by our assumption about n).

We say that a pattern $P \in [0\,..\,\sigma)^m$ is *τ-periodic* if $m \geq 3\tau - 1$ and $\mathrm{per}(P[1\,..\,3\tau - 1]) \leq \frac{1}{3}\tau$. Otherwise, P is *τ-nonperiodic*. For simplicity, below we outline how to reduce the computation of $\mathrm{LexRange}(P_1, P_2, T)$ to prefix range reporting when one of the following three cases holds:

- $|P_1| \leq 3\tau - 1$ and $|P_2| \leq 3\tau - 1$,
- P_1 and P_2 are τ-nonperiodic and $\mathrm{lcp}(P_1, P_2) \geq 3\tau - 1$,
- P_1 and P_2 are τ-periodic and $\mathrm{lcp}(P_1, P_2) \geq 3\tau - 1$.

Our reduction also works for all other combinations. In those cases, we decompose the set $\mathrm{LexRange}(P_1, P_2, T)$ into at most three subsets, each of which is computed using a variant of one of the above three cases; see the proof of Proposition 7.60 for details.

Short Patterns Consider $P_1, P_2 \in [0\,..\,\sigma)^*$ satisfying $|P_1| \leq 3\tau - 1$ and $|P_2| \leq 3\tau - 1$. Denote $\mathcal{S}_{\mathrm{short}} = \{T[i\,..\,\min(n+1, i+3\tau-1)) : i \in [1\,..\,n]\}$. Note that $|\mathcal{S}_{\mathrm{short}}| = \mathcal{O}(\sigma^{3\tau-1}) = \mathcal{O}(\sigma^{3\mu \log_\sigma n}) = \mathcal{O}(n^{3\mu}) = \mathcal{O}(\sqrt{n})$.

Let $k = |\mathcal{S}_{\text{short}}|$ and let $A_{\text{short}}[1\,..\,k]$ be an array containing all elements of $\mathcal{S}_{\text{short}}$ in lexicographic order. Finally, let $B_{3\tau-1}[1\,..\,n]$ be a bitvector marking every index $i \in [1\,..\,n]$ such that either $i = n$, or $i < n$ and $\mathrm{LCE}_T(\mathrm{SA}_T[i], \mathrm{SA}_T[i+1]) < 3\tau - 1$. In other words, $B_{3\tau-1}$ marks boundaries between blocks of suffixes of T sharing the same string from $\mathcal{S}_{\text{short}}$. Assume that $B_{3\tau-1}$ has been augmented with support for $\mathcal{O}(1)$-time rank and select queries using Theorem 2.7. Note that, thanks to the sentinel symbol at the end of T, the set $\mathcal{S}_{\text{short}}$ is prefix-free (i.e., no string in $\mathcal{S}_{\text{short}}$ is a proper prefix of another string in $\mathcal{S}_{\text{short}}$).

We enumerate the set $\mathrm{LexRange}(P_1, P_2, T)$ as follows. First, using lookup tables, we compute $b_i = \mathrm{RangeBeg}(P_i, T)$ for $i \in \{1, 2\}$ (see Proposition 7.10). Observe that for every $i \in \{1, 2\}$, $b_i > 0$ implies that $B_{3\tau-1}[b_i] = 1$. Thus, using rank/select queries on $B_{3\tau-1}$, we can compute $i_1, i_2 \in [0\,..\,k]$ such that

$$\mathrm{LexRange}(P_1, P_2, T) = \bigcup_{i \in (i_1\,..\,i_2]} \mathrm{Occ}(A_{\text{short}}[i], T).$$

We have thus reduced computing $\mathrm{LexRange}(P_1, P_2, T)$ to enumerating $\mathrm{Occ}(X, T)$ for $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$; see Proposition 7.12.

To solve the latter problem, we split the text T into $\mathcal{O}(\frac{n}{\tau})$ blocks of size $2\ell - 1$, where $\ell = 3\tau - 1$, so that each adjacent block overlaps the next by exactly $\ell - 1$ symbols. We then sort all the blocks (represented by their starting positions) lexicographically and store the resulting sequence in an array $A_{\text{blk}}[1\,..\,m]$. We also prepare a lookup table L_{blocks} that, for every $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, stores a list $\mathrm{blocks}'(X)$ of pointers to *distinct* blocks in the above list that contain at least one occurrence of X among the first ℓ positions. Along with each pointer, we also keep a bitmask marking the occurrences of X within the first ℓ positions of that block. At query time, we first obtain the list of distinct blocks containing X from L_{blocks}. For each distinct block in $\mathrm{blocks}'(X)$, we then obtain from A_{blk} the list of all occurrences of that block in the text. Altogether, this computation provides enough information to retrieve all occurrences of X in T in time linear in their number. See Proposition 7.11 for details.

Nonperiodic Patterns Let $P_1, P_2 \in [0\,..\,\sigma)^*$ be τ-nonperiodic patterns (Definition 7.5) satisfying $\mathrm{lcp}(P_1, P_2) \geq 3\tau - 1$. Denote $P' = P_1[1\,..\,3\tau - 1] = P_2[1\,..\,3\tau - 1]$. To compute $\mathrm{LexRange}(P_1, P_2, T)$, we first check using lookup tables (Proposition 7.10) whether $\mathrm{Occ}(P', T) = \emptyset$. If so, we conclude that $\mathrm{LexRange}(P_1, P_2, T) = \emptyset$. Let us therefore assume that $\mathrm{Occ}(P', T) \neq \emptyset$. Let S be a τ-synchronizing set of T computed in $\mathcal{O}(n/\log_\sigma n)$ time using Theorem 2.20. Denote $n' = |\mathsf{S}| = \mathcal{O}(\frac{n}{\tau})$.

Observation: The computation of $\mathrm{LexRange}(P_1, P_2, T)$ *can be reduced to prefix range reporting.* Since P is τ-nonperiodic (i.e., $\mathrm{per}(P') > \frac{1}{3}\tau$), it follows from the consistency and density conditions of S (Definition 2.17) that $\mathrm{succ}_{\mathsf{S}}(i) - i = \mathrm{succ}_{\mathsf{S}}(i') - i' < \tau$ holds for every $i, i' \in \mathrm{Occ}(P', T)$. Let $\delta_{\text{text}} \geq 0$ denote this common offset, i.e., such that for every $i \in \mathrm{Occ}(P', T)$, it holds $\mathrm{succ}_{\mathsf{S}}(i) - i = \delta_{\text{text}}$. Let $(s_i)_{i \in [1\,..\,n']}$ be the sequence containing positions in S, sorted according to the lexicographic order of the corresponding suffixes in T. Let $P'_k = P_k(\delta_{\text{text}}\,..\,|P_k|]$, where $k \in \{1, 2\}$. By the above observation, to identify $\mathrm{LexRange}(P_1, P_2, T)$, we need to find the range of positions in the sequence $(s_i)_{i \in [1\,..\,n']}$ containing suffixes of the text that are lexicographically between P'_1 and P'_2. Such computation can be done in $\mathcal{O}(\log\log n + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time (see Proposition 7.21). Denote $b_k = |\{i \in [1\,..\,n'] : T[s_i\,..\,n] \prec P'_k\}|$ for $k \in \{1, 2\}$. After computing b_1 and b_2, it remains to select the positions from $(s_i)_{i \in (b_1\,..\,b_2]}$ that correspond to suffixes preceded by $P'[1\,..\,\delta_{\text{text}}]$ in T. To implement this, define an array $A_{\text{str}}[1\,..\,n']$ such that $A_{\text{str}}[i] = \overline{T^\infty[s_i - \tau\,..\,s_i + 2\tau)}$. Then, letting $D = P'[1\,..\,\delta_{\text{text}} + 2\tau]$, it holds $\mathrm{LexRange}(P_1, P_2, T) = \{s_i - \delta_{\text{text}} : i \in (b_1\,..\,b_2] \text{ and } \overline{D} \text{ is a prefix of } A_{\text{str}}[i]\}$; see Lemmas 7.23 and 7.24. In other words, the remaining step in the computation of $\mathrm{LexRange}(P_1, P_2, T)$ is a prefix range reporting query. See Proposition 7.25 for the complete algorithm.

Periodic Patterns Let $P_1, P_2 \in [0\,..\,\sigma)^*$ be τ-periodic patterns (Definition 7.5) such that $\mathrm{lcp}(P_1, P_2) \geq 3\tau - 1$. Denote $P' = P_1[1\,..\,3\tau - 1] = P_2[1\,..\,3\tau - 1]$. Since, by Definition 7.5, we have $\mathrm{per}(P') \leq \frac{1}{3}\tau$, and since every $j \in \mathrm{LexRange}(P_1, P_2, T)$ satisfies $j \in \mathrm{Occ}(P', T)$, we obtain $\mathrm{LexRange}(P_1, P_2, T) \subseteq \mathsf{R}(\tau, T) = \{j \in [1\,..\,n - 3\tau + 2] : \mathrm{per}(T[j\,..\,j + 3\tau - 1)) \leq \frac{1}{3}\tau\}$ (Definition 2.17).

Observation: $\mathrm{LexRange}(P_1, P_2, T)$ *can be expressed using maximal periodic runs in* T. To efficiently compute $\mathrm{LexRange}(P_1, P_2, T)$ in this case, we group positions in $\mathsf{R}(\tau, T)$ into maximal blocks corresponding to

τ-*runs*, i.e., maximal substrings $T[i\,..\,j)$ satisfying $j-i \geq 3\tau-1$ and $\mathrm{per}(T[i\,..\,j)) \leq \frac{1}{3}\tau$. The gap between $|Y|$ and $\mathrm{per}(Y)$ ensures that τ-runs overlap by fewer than $\frac{2}{3}\tau$ symbols, so the number of τ-runs is $\mathcal{O}(\frac{n}{\tau}) = \mathcal{O}(n/\log_\sigma n)$ (see Lemma 7.40). To efficiently process τ-runs, we introduce the following definitions. Let $T[x\,..\,y)$ be a τ-run. Since $y-x \geq 3\tau-1$ and $p := \mathrm{per}(T[x\,..\,y)) \leq \frac{1}{3}\tau$, we can uniquely write $T[x\,..\,y) = H'H^kH''$, where $H = \min\{T[x+\delta\,..\,x+\delta+p) : \delta \in [0\,..\,p)\}$ is the *Lyndon root* of the run, and H' (resp. H'') is a proper suffix (resp. prefix) of H. We then denote $\mathrm{head}(x,\tau,T) = |H'|$, $\mathrm{tail}(x,\tau,T) = |H''|$, $\mathrm{root}(x,\tau,T) = H$, and $\exp(x,\tau,T) = k$. Using these properties, we partition $\mathsf{R}(\tau,T)$ as well as all τ-runs into classes that allow us to reduce the search space during query time (see Definitions 7.33 and 7.34). Specifically, using the above notation, we can express certain ranges in the suffix array in terms of τ-runs (Lemma 7.54). This characterization is then gradually extended into a complete characterization (as well as an efficient enumeration) of the set $\mathrm{LexRange}(P_1,P_2,T)$; see Propositions 7.55 to 7.59.

The above three cases are combined in Proposition 7.60. Recall, however, that we added a sentinel symbol at the end of T to ensure that every suffix of T has exactly one occurrence in T. To reduce the alphabet back to binary, we therefore need to perform an alphabet reduction for prefix range reporting. To this end, observe that if, given any sequence $W[1\,..\,m]$ of $m \geq 1$ strings of length $\ell \geq 1$ over the alphabet $[0\,..\,\sigma)$, we set $k = \lceil \log \sigma \rceil$ and let $W_{\mathrm{bin}}[1\,..\,m]$ be any sequence of binary strings such that, for every $j \in [1\,..\,m]$, $\mathrm{bin}_k(W[j])$ (Definition 4.1) is a prefix of $W_{\mathrm{bin}}[j]$, then for every $X \in [0\,..\,\sigma)^{\leq \ell}$ and $b,e \in [0\,..\,m]$, it holds that

$$\{i \in (b\,..\,e] : X \text{ is a prefix of } W[i]\} = \{i \in (b\,..\,e] : X_{\mathrm{bin}} \text{ is a prefix of } W_{\mathrm{bin}}[i]\},$$

where $X_{\mathrm{bin}} = \mathrm{bin}_k(X)$; see Lemma 4.19. Moreover, the reduction can be implemented in time optimal in the bit-encoding of W; see Proposition 7.61.

Reducing Prefix Range Reporting to Lex-Range Reporting Queries Let $W[1\,..\,m]$ be an array of $m \geq 1$ bitstrings of length $\ell = 1 + \lfloor \log m \rfloor$, each represented in $\mathcal{O}(1)$ machine words, i.e., the input to the problem of indexing for prefix range reporting queries (see Section 7.1). The basic idea of the reduction is similar to that in Section 3.1; namely, the text T is defined as $T = \bigodot_{i=1}^{m} \overline{W[i]} \cdot \mathtt{4} \cdot s(i-1)$, where $s(i)$ is a length-k binary encoding of $i \in [0\,..\,m]$, with each $\mathtt{0}$ (resp. $\mathtt{1}$) replaced by $\mathtt{2}$ (resp. $\mathtt{3}$), where $k := 1 + \lfloor \log m \rfloor$. We then prove (see Lemma 7.1) that, for every $b,e \in [0\,..\,m]$ and $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$, it holds

$$\{i \in (b\,..\,e] : X \text{ is a prefix of } W[i]\} = \{\lceil j/\beta \rceil : j \in \mathrm{LexRange}(P_1,P_2,T)\},$$

where $P_1 = \overline{X} \cdot \mathtt{4} \cdot s(b)$, $P_2 = \overline{X} \cdot \mathtt{4} \cdot s(e)$, and $\beta = \ell + k + 1$. Thus, an algorithm for lex-range reporting can be used for prefix range reporting.

As in other problems, the above reduction increases the size of the alphabet. To reduce the alphabet back to binary, in Section 7.2.2, we prove a general alphabet reduction for lex-range reporting. Specifically, we show that, given the packed representation of $T \in [0\,..\,\sigma)^n$, we can, in $\mathcal{O}(n/\log_\sigma n)$ time, construct the packed representation of a string $T_{\mathrm{bin}} \in \{\mathtt{0},\mathtt{1}\}^*$ satisfying $|T_{\mathrm{bin}}| = \Theta(n \log \sigma)$, as well as integers α, γ, and δ, and a data structure that, given the packed representation of any $P_1,P_2 \in [0\,..\,\sigma)^{\leq m}$, in $\mathcal{O}(1 + m/\log_\sigma n)$ time, returns the packed representations of $P_1',P_2' \in \{\mathtt{0},\mathtt{1}\}^*$ such that $|P_i'| = |P_i| \cdot (2\lceil \log \sigma \rceil + 3)$ (for $i \in \{1,2\}$) and

$$\mathrm{LexRange}(P_1,P_2,T) = \left\{ \frac{j-\alpha}{\gamma} + \delta : j \in \mathrm{LexRange}(P_1',P_2',T_{\mathrm{bin}}) \right\};$$

see Proposition 7.3. Together, these two reductions yield Theorem 7.4.

4 Equivalence of Suffix Array and Prefix Select Queries

4.1 Problem Definitions

Indexing for Suffix Array Queries
Input: The packed representation of a string $T \in \{\mathtt{0},\mathtt{1}\}^n$.

Output: A data structure that, given any $i \in [1\,..\,n]$, returns $\mathrm{SA}_T[i]$, i.e., the starting position of the lexicographically ith smallest suffix of T.

INDEXING FOR PREFIX SELECT QUERIES

Input: A sequence $W[1\,..\,m]$ of $m \geq 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$, with all strings represented in the packed form.

Output: A data structure that, given the packed representation of any $X \in \{0,1\}^{\leq \ell}$ and any $i \in [1\,..\,\mathsf{prefix\text{-}rank}_W(m, X)]$ (see Definition 2.8), returns $\mathsf{prefix\text{-}select}_W(i, X)$, i.e., the ith smallest element of the set $\{j \in [1\,..\,m] : X \text{ is a prefix of } W[j]\}$.

4.2 Reducing Prefix Select to Suffix Array Queries

4.2.1 Preliminaries

Definition 4.1 (Binary mapping)**.** For any $k \in \mathbb{Z}_{>0}$ and $x \in [0\,..\,2^k)$, by $\mathrm{bin}_k(x) \in \{0,1\}^k$ we denote the length-k string containing the binary representation of x (with leading zeros). For any string S over alphabet $\Sigma = [0\,..\,2^k)$, we then let $\mathrm{bin}_k(S) := \bigodot_{i=1}^{|S|} \mathrm{bin}_k(S[i])$.

Proposition 4.2. *Let $u \geq 1$ and σ be such that $2 \leq \sigma < u^{\mathcal{O}(1)}$. Denote $k = \lceil \log \sigma \rceil$. In $\mathcal{O}(u/\log_\sigma u)$ time we can construct a data structure that, given the packed representation of any $S \in [0\,..\,\sigma)^*$, computes the packed representation of $\mathrm{bin}_k(S)$ (Definition 4.1) in $\mathcal{O}(1 + |S|/\log_\sigma u)$ time.*

Proof. The proof proceeds similarly as the proof of [KK25, Proposition 5.47]. If $\sigma > u^{1/4}$, then we do not need any preprocessing. Otherwise, in $\mathcal{O}(u/\log_\sigma u)$ time we construct a lookup table that implements the mapping for substring of length $\leq \ell$ for some $\ell = \Theta(\log_\sigma u)$ in $\mathcal{O}(1)$ time, yielding an overall query time of $\mathcal{O}(1 + |S|/\log_\sigma u)$. □

Definition 4.3 (Extended binary mapping)**.** Let $k \in \mathbb{Z}_{>0}$. For every $x \in [0\,..\,2^k)$, define

$$\mathrm{ebin}_k(a) := \mathtt{1}^{k+1} \cdot \mathtt{0} \cdot \mathrm{bin}_k(x) \cdot \mathtt{0},$$

where $\mathrm{bin}_k(x)$ is as in Definition 4.1. For any string S over alphabet $\Sigma = [0\,..\,2^k)$, we then let

$$\mathrm{ebin}_k(S) := \bigodot_{i=1}^{|S|} \mathrm{ebin}_k(S[i]).$$

Lemma 4.4 ([KK25])**.** *Let $k \in \mathbb{Z}_{>0}$ and let $\sigma = 2^k$. For every $S_1, S_2 \in [0\,..\,\sigma)^*$, $S_1 \prec S_2$ holds if and only if $\mathrm{ebin}_k(S_1) \prec \mathrm{ebin}_k(S_2)$.*

Proposition 4.5 ([KK25])**.** *Let $u \geq 1$ and σ be such that $2 \leq \sigma < u^{\mathcal{O}(1)}$. Denote $k = \lceil \log \sigma \rceil$. In $\mathcal{O}(u/\log_\sigma u)$ time we can construct a data structure that, given the packed representation of any $S \in [0\,..\,\sigma)^*$, computes the packed representation of $\mathrm{ebin}_k(S)$ (Definition 4.3) in $\mathcal{O}(1 + |S|/\log_\sigma u)$ time.*

Definition 4.6 (Symbol-substitute function)**.** Let $u, v \in \Sigma^+$ and $c \in \Sigma$. We define $\mathrm{sub}(u, c, v)$ as a string obtained by replacing all occurrences of c in u with v. Formally, $\mathrm{sub}(u, c, v) = \bigodot_{i=1,\ldots,|u|} f(u[i])$, where $f : \Sigma \to \Sigma^*$ is such that for every $a \in \Sigma$:

$$f(a) = \begin{cases} a & \text{if } a \neq c, \\ v & \text{otherwise.} \end{cases}$$

Definition 4.7 (Sequence-to-string mapping)**.** Let $W[1\,..\,m]$ be a sequence of $m \geq 1$ strings of length $\ell \geq 1$ over alphabet $\{0,1\}$. Denote $k = 1 + \lfloor \log m \rfloor$. For any $i \in [0\,..\,m]$, let $s(i) = \mathrm{sub}(\mathrm{sub}(\mathrm{bin}_k(i), \mathtt{0}, \mathtt{2}), \mathtt{1}, \mathtt{3})$ (see Definitions 4.1 and 4.6). We then define

$$\mathrm{SeqToString}_\ell(W) := \bigodot_{i=1}^{m} \overline{W[i]} \cdot \mathtt{4} \cdot s(i-1).$$

Definition 4.8 (Integer mapping for binary strings)**.** For every $X \in \{0,1\}^*$, we let $\mathrm{BinStrToInt}(X) = 1$ (if $X = \varepsilon$) and $\mathrm{BinStrToInt}(X) = 2^k + x$, where $k = |X|$ and $x \in [0\,..\,2^k)$ is such that $\mathrm{bin}_k(x) = X$ (if $X \neq \varepsilon$); see Definition 4.1.

Remark 4.9. Note that the mapping in Definition 4.8 is injective, since given $y = \mathrm{BinStrToInt}(X)$ we can decode X by first computing the length of $|X|$ (using the position of the most significant bit in y), and then interpreting the least significant $|X|$ bits as symbols of X. Alternatively, this follows by noting that $\mathrm{BinStrToInt}(X)$ is the identifier of a node with a root-to-node label X in a trie of the set $\{0,1\}^*$, in which nodes are numbered as in a heap (with 1 as the identifier of the root).

Lemma 4.10. *Let $W[1\,..\,m]$ be a sequence of $m \geq 1$ strings of length $\ell \geq 1$ over alphabet $\{0,1\}$. Let $T = \mathrm{SeqToString}_\ell(W)$ (Definition 4.7), $k = 1 + \lfloor \log m \rfloor$, and $\beta = \ell + k + 1$. Finally, let $X \in \{0,1\}^{\leq \ell}$ and $\mathcal{P} = \{i \in [1\,..\,m] : X$ is a prefix of $W[i]\}$. Then, it holds*

$$\mathrm{Occ}(\overline{X} \cdot 4, T) = \{i \cdot \beta - (k + |X|) : i \in \mathcal{P}\}.$$

Proof. Denote $A = \{i \cdot \beta - (k + |X|) : i \in \mathcal{P}\}$.

First, we show the inclusion $\mathrm{Occ}(\overline{X} \cdot 4, T) \subseteq A$. Let $j \in \mathrm{Occ}(\overline{X} \cdot 4, T)$. Note that $\mathrm{Occ}(4, T) = \{i \cdot \beta - k : i \in [1\,..\,m]\}$. Thus, $j \in \mathrm{Occ}(\overline{X} \cdot 4, T)$ implies that for some $i \in [1\,..\,m]$, it holds $j = i \cdot \beta - (k + |X|)$. Since $T[i \cdot \beta - k\,..\,|T|]$ is preceded in T with $\overline{W[i]}$, it follows that $\overline{X}$ is a suffix of $\overline{W[i]}$, or equivalently, X is a prefix of $W[i]$. This implies that $i \in \mathcal{P}$, and hence, by definition of A, $i \cdot \beta - (k + |X|) \in A$. It remains to note that $i \cdot \beta - (k + |X|) = j$.

We now show the opposite inclusion. Let $j \in A$. Then, there exists $i \in \mathcal{P}$ such that $j = i \cdot \beta - (k + |X|)$. By $i \in \mathcal{P}$, it holds $i \in [1\,..\,m]$ and X is a prefix of $W[i]$. Note that, by Definition 4.7, $T(i \cdot \beta\,..\,|T|]$ is preceded in T with $\overline{W[i]} \cdot 4 \cdot s(i-1)$. Since X is a prefix of $W[i]$, or equivalently, $\overline{X}$ is a suffix of $\overline{W[i]}$, we thus obtain that $T(i \cdot \beta\,..\,|T|]$ is preceded in T with $\overline{X} \cdot 4 \cdot s(i-1)$. Thus, $i \cdot \beta - (k + |X|) \in \mathrm{Occ}(\overline{X} \cdot 4, T)$. By $j = i \cdot \beta - (k + |X|)$, we thus obtain $j \in \mathrm{Occ}(\overline{X} \cdot 4, T)$. □

Proposition 4.11. *Let $W[1\,..\,m]$ be an array of $m \geq 1$ binary string of length $\ell = 1 + \lfloor \log m \rfloor$ and let $T = \mathrm{SeqToString}_\ell(W)$ (Definition 4.7). Given the array W, with all strings represented in the packed form, we can in $\mathcal{O}(m)$ time construct a data structure that, given the packed representation of any $X \in \{0,1\}^{\leq \ell}$ (i.e., a pair (k, x), where $k = |X|$ and $x \in [0\,..\,2^k)$ is such that $\mathrm{bin}_k(x) = X$), returns $\mathrm{RangeBeg}(\overline{X} \cdot 4, T)$ in $\mathcal{O}(1)$ time.*

Proof. **Components** The data structure consists of a single component: an array $R[1\,..\,2^{\ell+1})$ defined such that for every $X \in \{0,1\}^{\leq \ell}$, $R[\mathrm{BinStrToInt}(X)] = \mathrm{RangeBeg}(\overline{X} \cdot 4, T)$ (see Definition 4.8). The array R, and hence the whole data structure, needs $\mathcal{O}(2^\ell) = \mathcal{O}(m)$ space.

Implementation of queries The queries are implemented as follows. Suppose that we are given the packed representation of X, i.e., a pair (k, x), where $k = |X|$ and $x \in [0\,..\,2^k)$ is such that $\mathrm{bin}_k(x) = X$. First, in $\mathcal{O}(1)$ time we compute $y := \mathrm{BinStrToInt}(X) = 2^k + x$. We then return $R[y]$.

Construction algorithm The data structure is constructed as follows:

1. First, we compute an auxiliary array $C[1\,..\,2^{\ell+1})$ defined such that, for every $X \in \{0,1\}^{\leq \ell}$, it holds $C[\mathrm{BinStrToInt}(X)] = |\mathrm{Occ}(X \cdot 4, T)|$, i.e., C stores the frequency in T of every string of the form $X \cdot 4$. We begin by initializing all entries of C to zero. We then compute $C[\mathrm{BinStrToInt}(X)]$ for all $X \in \{0,1\}^\ell$. To this end, it suffices to scan the array W. We then iterate $k = \ell - 1, \ell - 2, \ldots, 1$, and for each k we compute $C[\mathrm{BinStrToInt}(X)]$ for all $X \in \{0,1\}^k$ using the equality $C[\mathrm{BinStrToInt}(X)] = C[\mathrm{BinStrToInt}(0 \cdot X)] + C[\mathrm{BinStrToInt}(1 \cdot X)]$. More precisely, given $k \in (\ell\,..\,1]$ and some $x \in [0\,..\,2^k)$ such that $\mathrm{bin}_k(x) = X$, we have $\mathrm{BinStrToInt}(X) = 2^k + x$, $\mathrm{BinStrToInt}(0 \cdot X) = 2^{k+1} + x$, and $\mathrm{BinStrToInt}(1 \cdot X) = 2^{k+1} + 2^k + x$, and hence we set $C[2^k + x] = C[2^{k+1} + x] + C[2^{k+1} + 2^k + x]$. Finally, we set $C[\mathrm{BinStrToInt}(\varepsilon)] = C[1] = m$. In total, the computation of C takes $\mathcal{O}(2^\ell) = \mathcal{O}(m)$ time.

2. Next, we compute an auxiliary array $R'[1\,..\,2^{\ell+1})$ defined such that for every $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$, it holds $R'[\mathrm{BinStrToInt}(X)] = \mathrm{RangeBeg}(X \cdot \mathtt{4}, T)$. The case $X = \varepsilon$ is easy to handle, and we set $R'[\mathrm{BinStrToInt}(\varepsilon)] = R'[1] = |T| - m$. To compute the remaining values, observe that since the set $\{X \cdot \mathtt{4} : X \in \{\mathtt{0},\mathtt{1}\}^*\}$ is prefix-free (i.e., no string in the set is a prefix of another), it follows that for every $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell} \setminus \{\varepsilon\}$, it holds

$$\mathrm{RangeBeg}(X \cdot \mathtt{4}, T) = \sum_{X' \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell} \setminus \{\varepsilon\} : X' \cdot \mathtt{4} \prec X \cdot \mathtt{4}} |\mathrm{Occ}(X' \cdot \mathtt{4}, T)|.$$

   Consequently, to compute R', it suffices to enumerate all strings in the set $\{X \cdot \mathtt{4} : X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell} \setminus \{\varepsilon\}\}$ in lexicographical order, while keeping the cumulative frequency of smaller strings. To perform this enumeration, we observe that such order corresponds to a post-order traversal of a trie of $\{\mathtt{0},\mathtt{1}\}^{\ell}$. Such traversal is easily implemented as a recursive function. To navigate during this traversal, we maintain the packed representation of the root-to-node label for the currently visited node, i.e., for a node labeled X, we keep $k = |X|$ and an integer $x \in [0\,..\,2^k)$ such that $\mathrm{bin}_k(x) = X$. The corresponding representation for the left (resp. right) node is then $(k+1, 2x)$ (resp. $(k+1, 2x+1)$). Using this representation, we can always compute $|\mathrm{Occ}(X \cdot \mathtt{4}, T)| = C[\mathrm{BinStrToInt}(X)] = C[2^k + x]$ to update the cumulative frequency, and write the current cumulative frequency to $R'[\mathrm{BinStrToInt}(X)]$. In total, we spend $\mathcal{O}(2^\ell) = \mathcal{O}(m)$ time.
3. Let $\ell' = \lceil \ell/2 \rceil$. In this step, we compute an auxiliary lookup table $L_{\mathrm{rev}}[1\,..\,2^{\ell'+1})$ defined such that for every $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell'} \setminus \{\varepsilon\}$, letting $k = |X|$ and $y \in [0\,..\,2^k)$ be such that $\mathrm{bin}_k(y) = \overline{X}$, it holds $L_{\mathrm{rev}}[\mathrm{BinStrToInt}(X)] = y$. The computation of y for any $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell'} \setminus \{\varepsilon\}$ takes $\mathcal{O}(|X|) = \mathcal{O}(\ell')$ time, and hence in total we spend $\mathcal{O}(2^{\ell'}\ell') = \mathcal{O}(\sqrt{m}\log m) = \mathcal{O}(m)$ time. Given L_{rev} and the packed representation of any $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$ (i.e., the length $k = |X|$ and an integer $x \in [0\,..\,2^k)$ satisfying $\mathrm{bin}_k(x) = X$), we can in $\mathcal{O}(1)$ time compute $y \in [0\,..\,2^k)$ such that $\mathrm{bin}_k(y) = \overline{X}$. Consequently, with such input we can compute $\mathrm{BinStrToInt}(\overline{X})$.
4. In the last step, we compute the array R. To this end, we iterate over all $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$. Given the packed representation of X, in $\mathcal{O}(1)$ time we first compute $\mathrm{BinStrToInt}(X)$ and $\mathrm{BinStrToInt}(\overline{X})$, and then write $R[\mathrm{BinStrToInt}(X)] := R'[\mathrm{BinStrToInt}(\overline{X})]$. In total, this takes $\mathcal{O}(2^\ell) = \mathcal{O}(m)$ time.

In total, the construction takes $\mathcal{O}(m)$ time. □

Proposition 4.12. *Let $W[1\,..\,m]$ be an array of $m \geq 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$. Given array W, with all strings represented in the packed form, we can in $\mathcal{O}(m)$ time compute the packed representation of the string* $\mathrm{SeqToString}_\ell(W)$ *(Definition 4.7).*

Proof. The algorithm proceeds as follows:

1. Let $\ell' = \lceil \ell/2 \rceil$. We compute an auxiliary lookup table L_{rev} such that for every $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell'}$, L_{rev} maps X to the packed representation of $\overline{X}$ (over alphabet $\{\mathtt{0},\ldots,\mathtt{4}\}$). When accessing L_{rev}, we map X to $\mathrm{BinStrToInt}(X)$ (Definition 4.8). The computation takes $\mathcal{O}(\sqrt{m}\log m) = \mathcal{O}(m)$ time. Given the packed representation of any $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$ (over alphabet $\{\mathtt{0},\mathtt{1}\}$), we can then in $\mathcal{O}(1)$ time compute the packed representation of $\overline{X}$ (over $\{\mathtt{0},\ldots,\mathtt{4}\}$).
2. Let ℓ' be defined as above. In this step, we compute an auxiliary lookup table L_{map} defined such that for every $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell'}$, L_{map} maps X to the packed representation of $\mathrm{sub}(\mathrm{sub}(X, \mathtt{0}, \mathtt{2}), \mathtt{1}, \mathtt{3})$ (over alphabet $\{\mathtt{0},\ldots,\mathtt{4}\}$). Similarly as above, the construction takes $\mathcal{O}(\sqrt{m}\log m) = \mathcal{O}(m)$ time. Given $i \in [0\,..\,m]$, we can then compute the packed representation of $s(i)$ (defined as in Lemma 4.13) over alphabet $\{\mathtt{0},\ldots,\mathtt{4}\}$ in $\mathcal{O}(1)$ time.
3. With the above lookup tables, the construction of the packed representation of $\mathrm{SeqToString}_\ell(W)$ takes $\mathcal{O}(m)$ time.

In total, we spend $\mathcal{O}(m)$ time. □

4.2.2 Problem Reduction

Lemma 4.13. *Let $W[1\,..\,m]$ be an array of $m \geq 1$ strings of length $\ell \geq 1$ over alphabet $\{0,1\}$. Denote $T = \mathrm{SeqToString}_\ell(W)$ (Definition 4.7). For every $X \in \{0,1\}^{\leq \ell}$ and $i \in [1\,..\,\mathsf{prefix\text{-}rank}_W(m,X)]$, it holds*

$$\mathsf{prefix\text{-}select}_W(i,X) = \left\lceil \frac{\mathrm{SA}_T[\alpha+i]}{\beta} \right\rceil,$$

where $\alpha = \mathrm{RangeBeg}(\overline{X}\cdot 4, T)$ and $\beta = \ell + \lfloor \log m \rfloor + 2$.

Proof. Let k and $s(i)$ (where $i \in [0\,..\,m]$) be as in Definition 4.7. Denote $q = \mathsf{prefix\text{-}rank}_W(m,X)$. Let $(a_i)_{i\in[1\,..\,q]}$ and $(b_i)_{i\in[1\,..\,q]}$ be sequences defined by $a_i = \mathsf{prefix\text{-}select}_W(i,X)$ and $b_i = a_i\cdot\beta - (k+|X|)$. The proof proceeds in three steps:

1. First, observe that by Lemma 4.10, it holds $\mathrm{Occ}(\overline{X}\cdot 4, T) = \{b_1,\ldots,b_q\}$.
2. Next, we prove that $T[b_1\,..\,|T|] \prec T[b_2\,..\,|T|] \prec \cdots \prec T[b_q\,..\,|T|]$. To this end, it suffices to note that for every $i \in [1\,..\,q]$, the suffix $T[b_i\,..\,|T|]$ starts with the string $\overline{X}\cdot 4\cdot s(a_i-1)$. Since by definition we have $a_1 < a_2 < \cdots < a_q$ and $s(0) \prec s(1) \prec \cdots \prec s(m-1)$, it follows that $\overline{X}\cdot 4\cdot s(a_i) \prec \overline{X}\cdot 4\cdot s(a_{i+1})$ holds for every $i \in [1\,..\,q)$. This immediately yields $T[b_i\,..\,|T|] \prec T[b_{i+1}\,..\,|T|]$.
3. By the above two steps, and the definition of α, it follows that for every $i \in [1\,..\,q]$, it holds $\mathrm{SA}_T[\alpha+i] = b_i$. Noting that for every $i \in [1\,..\,q]$, it holds $\lceil b_i/\beta \rceil = a_i$, we thus obtain

$$\mathsf{prefix\text{-}select}_W(i,X) = a_i = \left\lceil \frac{b_i}{\beta} \right\rceil = \left\lceil \frac{\mathrm{SA}_T[\alpha+i]}{\beta} \right\rceil. \qquad \square$$

4.2.3 Alphabet Reduction for Suffix Array Queries

Lemma 4.14. *Let $\sigma \geq 2$ and $T \in [0\,..\,\sigma)^+$. For every $i \in [1\,..\,|T|]$, it holds*

$$\mathrm{SA}_T[i] = \frac{\mathrm{SA}_{T'}[\Delta+i]-1}{\delta} + 1,$$

where $k = \lceil \log\sigma \rceil$, $T' = \mathrm{ebin}_k(T)$ (Definition 4.3), $\Delta = |T'| - |T|$, and $\delta = 2k+3$.

Proof. By Lemma 4.4, for every $j_1, j_2 \in [1\,..\,n]$, $T[j_1\,..\,|T|] \prec T[j_2\,..\,|T|$ holds if and only if $T'[j_1'\,..\,|T'|] \prec T'[j_2'\,..\,|T'|]$, where $j_k' = (j_k-1)\delta + 1$ ($k \in \{1,2\}$). On the other hand, since 1^{k+2} does not occur in T', it follows that $\{\mathrm{SA}_{T'}[\Delta+i] : i \in [1\,..\,|T|]\} = \{(j-1)\delta+1 : j \in [1\,..\,|T|]\}$. Putting these together implies that for every $i \in [1\,..\,|T|]$, it holds $\mathrm{SA}_{T'}[\Delta+i] = (\mathrm{SA}_T[i]-1)\delta+1$. Rewriting this formula yields the claim. $\square$

Proposition 4.15. *Let $T \in [0\,..\,\sigma)^n$ be a nonempty string, where $2 \leq \sigma < n^{\mathcal{O}(1)}$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time compute integers α, β, γ, μ, and the packed representation of a string $T_{\mathrm{bin}} \in \{0,1\}^+$ such that $|T_{\mathrm{bin}}| = \Theta(n\log\sigma)$, and for every $i \in [1\,..\,n]$, it holds*

$$\mathrm{SA}_T[i] = \frac{\mathrm{SA}_{T_{\mathrm{bin}}}[\alpha+i]-\beta}{\gamma} + \mu.$$

Proof. Let $k = \lceil \log\sigma \rceil$. The algorithm proceeds as follows:

1. In $\mathcal{O}(n/\log_\sigma n)$ time we construct the data structure from Proposition 4.5 with $u = n$. Given the packed representation of any $S \in [0\,..\,\sigma)^*$, we can then compute the packed representation of $\mathrm{ebin}_k(S)$ (Definition 4.3) in $\mathcal{O}(1+|S|/\log_\sigma n)$ time.
2. Using the above structure, we compute the packed representation of the string $T_{\mathrm{bin}} = \mathrm{ebin}_k(T)$. This takes $\mathcal{O}(n/\log_\sigma n)$ time.
3. In $\mathcal{O}(1)$ time we set $\alpha = |T_{\mathrm{bin}}| - |T|$, $\beta = 1$, $\gamma = 2k+3$, and $\mu = 1$.

In total, the above algorithm takes $\mathcal{O}(n/\log_\sigma n)$ time.

The correctness of the construction follows by Lemma 4.14. $\square$

4.2.4 Summary

Theorem 4.16. *Consider a data structure answering suffix array queries (see Section 4.1) that, for any text $T \in \{0,1\}^n$, achieves the following complexities (where in the preprocessing we assume that we are given as input the packed representation of T):*

- *space usage $S(n)$,*
- *preprocessing time $P_t(n)$,*
- *preprocessing space $P_s(n)$,*
- *query time $Q(n)$.*

For every sequence $W[1\,..\,m]$ of $m \geq 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$, there exists $n = \mathcal{O}(m \log m)$ such that, given the sequence W with all strings represented in the packed form, we can in $\mathcal{O}(m + P_t(n))$ time and $\mathcal{O}(m+P_s(n))$ working space build a data structure of size $\mathcal{O}(m+S(n))$ that, given the packed representation of any $X \in \{0,1\}^{\leq \ell}$ and any $i \in [1\,..\,\mathsf{prefix\text{-}rank}_W(m, X)]$, in $\mathcal{O}(Q(n))$ time computes $\mathsf{prefix\text{-}select}_W(i, X)$.

Proof. We use the following definitions. Let $T_{\mathrm{aux}} = \mathrm{SeqToString}_\ell(W)$ (Definition 4.7). Denote $n_{\mathrm{aux}} = |T_{\mathrm{aux}}| = m \cdot (2\ell+1) = \Theta(m \log m)$. Let α, β, γ, μ, and T_{bin} denote the four integers and the text resulting from applying Proposition 4.15 to text T_{aux}. Denote $n = |T_{\mathrm{bin}}|$. By Proposition 4.15, we have $n = \Theta(n_{\mathrm{aux}}) = \Theta(m \log m)$, and for every $i \in [1\,..\,n_{\mathrm{aux}}]$, it holds $\mathrm{SA}_{T_{\mathrm{aux}}}[i] = (\mathrm{SA}_{T_{\mathrm{bin}}}[\alpha + i] - \beta)/\gamma + \mu$.

Components The data structure consists of the following components:

1. The integers α, β, γ, and μ defined as above.
2. The data structure from the claim for the text T_{bin}. It needs $\mathcal{O}(S(n))$ space.
3. The structure resulting from applying Proposition 4.11 to the array W. It needs $\mathcal{O}(m)$ space.

In total, the structure needs $\mathcal{O}(m + S(n))$ space.

Implementation of queries Given the packed representation of any string $X \in \{0,1\}^{\leq \ell}$ and any index $i \in [1\,..\,\mathsf{prefix\text{-}rank}_W(m, X)]$, we compute $\mathsf{prefix\text{-}select}_W(i, X)$ as follows:

1. Using Proposition 4.11, in $\mathcal{O}(1)$ time we compute $\alpha_{\mathrm{aux}} := \mathrm{RangeBeg}(\overline{X} \cdot 4, T_{\mathrm{aux}})$.
2. Using the structure from the claim constructed for text T_{bin}, in $\mathcal{O}(Q(n))$ time we compute $j := \mathrm{SA}_{T_{\mathrm{bin}}}[\alpha + \alpha_{\mathrm{aux}} + i]$. In $\mathcal{O}(1)$ time we set $j_{\mathrm{aux}} = (j - \beta)/\gamma + \mu$. By Proposition 4.15 (and the discussion above), it holds $j_{\mathrm{aux}} = \mathrm{SA}_{T_{\mathrm{aux}}}[\alpha_{\mathrm{aux}} + i]$.
3. In $\mathcal{O}(1)$ time we compute $s := \lceil j_{\mathrm{aux}}/\beta_{\mathrm{aux}} \rceil$, where $\beta_{\mathrm{aux}} = 2\ell + 1$. By Lemma 4.13, it holds $s = \mathsf{prefix\text{-}select}_W(i, X)$. Thus, we return s as the answer.

In total, the query takes $\mathcal{O}(Q(n))$ time.

Construction algorithm The components of the data structure are constructed as follows:

1. To compute α, β, γ, and μ (defined above), we proceed as follows. First, in $\mathcal{O}(m)$ time we apply Proposition 4.12 to the sequence W to compute the packed representation of the string T_{aux} (defined above). Then, we apply Proposition 4.15 for T_{aux} to compute the packed representation of T_{bin} (as defined above), together with values α, β, γ, and μ. Since T_{aux} is over alphabet $\{0, \ldots, 4\}$, the application of Proposition 4.15 takes $\mathcal{O}(|T_{\mathrm{aux}}|/\log |T_{\mathrm{aux}}|) = \mathcal{O}(m)$ time.
2. In the second step, we apply the preprocessing from the claim to string T_{bin}. This takes $\mathcal{O}(P_t(n))$ time and uses $\mathcal{O}(P_s(n))$ working space.
3. Finally, we apply Proposition 4.11 to the sequence W. This takes $\mathcal{O}(m)$ time.

In total, the construction takes $\mathcal{O}(m + P_t(n))$ time and uses $\mathcal{O}(m + P_s(n))$ working space. □

4.3 Reducing Suffix Array to Prefix Select Queries

4.3.1 Preliminaries

Observation 4.17. *Let $W[1\,..\,m]$ be a sequence of $m \geq 1$ strings of length $\ell \geq 1$ over alphabet Σ and let $W'[1\,..\,m']$ be any sequence of $m' \geq m$ strings of length $\ell' \geq \ell$ over alphabet Σ such that for every $j \in [1\,..\,m]$, $W[j]$ is a prefix of $W'[j]$. Consider any string $X \in \Sigma^{\leq \ell}$. Then,*

- *For every $i \in [0\,..\,m]$, it holds*

$$\mathsf{prefix\text{-}rank}_W(i, X) = \mathsf{prefix\text{-}rank}_{W'}(i, X).$$

- *For every $i \in [1\,..\,\mathsf{prefix\text{-}rank}_W(m, X)]$, it holds*

$$\mathsf{prefix\text{-}select}_W(i, X) = \mathsf{prefix\text{-}select}_{W'}(i, X).$$

Furthermore, for every $i \in [1\,..\,m]$ and every $p \in [0\,..\,\ell]$, it holds

$$\mathsf{prefix\text{-}special\text{-}rank}_W(i, p) = \mathsf{prefix\text{-}special\text{-}rank}_{W'}(i, p).$$

4.3.2 Problem Reduction

Theorem 4.18 ([KK23a])**.** *Consider a data structure that, given any sequence W of k strings of length ℓ over alphabet $[0\,..\,\sigma)$ achieves the following complexities (where the input strings during construction are given in the packed representation, and at query time we are given the packed representation of any $X \in [0\,..\,\sigma)^{\leq \ell}$ and any $i \in [1\,..\,\mathsf{prefix\text{-}rank}_W(k, X)]$, and we return $\mathsf{prefix\text{-}select}_W(i, X)$; see Section 2.4):*

- *space usage $S(k, \ell, \sigma)$,*
- *preprocessing time $P_t(k, \ell, \sigma)$,*
- *preprocessing space $P_s(k, \ell, \sigma)$,*
- *query time $Q(k, \ell, \sigma)$.*

For every $T \in [0\,..\,\sigma)^n$ such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$, there exist positive integers $k = \Theta(n/\log_\sigma n)$ and $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n + P_t(k, \ell, \sigma))$ time and using $\mathcal{O}(n/\log_\sigma n + P_s(k, \ell, \sigma))$ working space build a data structure of size $\mathcal{O}(n/\log_\sigma n + S(k, \ell, \sigma))$ that answers suffix array queries on T (see Section 2) in $\mathcal{O}(\log\log n + Q(k, \ell, \sigma))$ time.

Proof. The above result, with cosmetically different parameters, was proved in [KK23a, Theorem 5.2]. To obtain the above claim, we make the following minor changes:

- First, we note that the original statement mentions both prefix rank and select queries. To obtain the above result it suffices to observe that during SA queries only prefix select queries are used (see [KK23a, Proposition 5.5]).
- In the original claim it holds $k = \mathcal{O}(n/\log_\sigma n)$. To obtain the claim with $k = \Theta(n/\log_\sigma n)$, we simply pad the input sequence of strings into length $k = \max(k', k'') = \Theta(n/\log_\sigma n)$, where $k' = \mathcal{O}(n/\log_\sigma n)$ is the original parameter, and $k'' = \lceil n/\log_\sigma n \rceil$. By Observation 4.17, the queries on the padded sequence are the same as on the original sequence.
- In the above claim, we have $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ and $2 \leq \sigma < n^{1/13}$. However, the constraints in the original claim [KK23a] are $\ell = \mathcal{O}(\log_\sigma n)$ and $2 \leq \sigma < n^{1/7}$. To obtain the claim with modified parameters, we recall that in the proof of the original claim, we have $\ell = 3\tau$, where $\tau = \lfloor \mu \log_\sigma n \rfloor$ and μ is any positive constant smaller than $\frac{1}{6}$ such that $\tau \geq 1$ (such μ exists by the assumption $\sigma < n^{1/7}$). Here we modify the value of τ and instead use $\tau = \lfloor \mu' \log_\sigma n \rfloor$, where μ' is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$ (such μ' exists by the assumption $\sigma < n^{1/13}$). This change does not affect the correctness of the data structure, since it makes τ smaller, and the only constraints on τ in [KK23a] are upper bounds. To show that $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$, we proceed as follows:
  - First, observe that by the assumption $2 \leq \sigma < n^{1/13}$ implies that n is sufficiently large so that $k^2 \geq k''^2 \geq (n/\log n)^2 \geq n$. Consequently, $\log n \leq 2 \log k$.
  - On the other hand, note that for every $x \geq 1$, it holds $\lceil x \rceil \leq 2x$.

  Putting the above together, we obtain:

$$\ell = 3\lfloor \mu' \log_\sigma n \rfloor \leq \lfloor 3\mu' \log_\sigma n \rfloor \leq \left\lfloor \tfrac{1}{4} \tfrac{\log n}{\log \sigma} \right\rfloor \leq \left\lfloor \tfrac{1}{2} \tfrac{\log k}{\log \sigma} \right\rfloor \leq \left\lfloor \tfrac{\log k}{\lceil \log \sigma \rceil} \right\rfloor \leq \left\lfloor \tfrac{1 + \lfloor \log k \rfloor}{\lceil \log \sigma \rceil} \right\rfloor \leq \tfrac{1 + \lfloor \log k \rfloor}{\lceil \log \sigma \rceil}. \qquad \square$$

4.3.3 Alphabet Reduction for Prefix Select Queries

Lemma 4.19. *Let $\sigma \geq 2$ and let $W[1\,..\,m]$ be an array of $m \geq 1$ strings of length $\ell \geq 1$ over alphabet $[0\,..\,\sigma)$. Let $k = \lceil \log \sigma \rceil$, and $W_{\mathrm{bin}}[1\,..\,m]$ be any sequence of binary strings such that for every $j \in [1\,..\,m]$, $\mathrm{bin}_k(W[j])$ (Definition 4.1) is a prefix of $W_{\mathrm{bin}}[j]$.*

1. *Let $X \in [0\,..\,\sigma)^{\leq \ell}$ and $X_{\mathrm{bin}} = \mathrm{bin}_k(X)$. Then:*
   (a) *For every $i \in [0\,..\,m]$, it holds*
   $$\mathsf{prefix\text{-}rank}_W(i, X) = \mathsf{prefix\text{-}rank}_{W_{\mathrm{bin}}}(i, X_{\mathrm{bin}}).$$
   (b) *For every $i \in [1\,..\,\mathsf{prefix\text{-}rank}_W(m, X)]$, it holds*
   $$\mathsf{prefix\text{-}select}_W(i, X) = \mathsf{prefix\text{-}select}_{W_{\mathrm{bin}}}(i, X_{\mathrm{bin}}).$$
   (c) *For every $b, e \in [0\,..\,m]$, it holds*
   $$\{i \in (b\,..\,e] : X \text{ is a prefix of } W[i]\} = \{i \in (b\,..\,e] : X_{\mathrm{bin}} \text{ is a prefix of } W_{\mathrm{bin}}[i]\}.$$
2. *For every $i \in [1\,..\,m]$ and every $p \in [0\,..\,\ell]$, it holds*
   $$\mathsf{prefix\text{-}special\text{-}rank}_W(i, p) = \mathsf{prefix\text{-}special\text{-}rank}_{W_{\mathrm{bin}}}(i, p \cdot k).$$

Proof. The proof follows by observing that for every $A, B \in [0\,..\,\sigma)^*$, A is a prefix of B if and only if $\mathrm{bin}_k(A)$ is a prefix of $\mathrm{bin}_k(B)$. □

Proposition 4.20. *Consider a data structure answering prefix select queries (see Section 4.1) that, for any sequence of $m \geq 1$ binary strings of length $1 + \lfloor \log m \rfloor$, achieves the following complexities (where both the string at query time as well as input strings during construction are given in the packed representation):*

- *space usage $S(m)$,*
- *preprocessing time $P_t(m)$,*
- *preprocessing space $P_s(m)$,*
- *query time $Q(m)$.*

Let $W[1\,..\,m']$ be a sequence of $m' \geq 1$ equal-length strings over alphabet $[0\,..\,\sigma)$ (where $\sigma \geq 2$) of length $\ell \leq (1 + \lfloor \log m' \rfloor)/\lceil \log \sigma \rceil$, Given the sequence W, with all strings represented in the packed form, we can in $\mathcal{O}(m' + P_t(m'))$ time and using $\mathcal{O}(m' + P_s(m'))$ working space construct a data structure of size $\mathcal{O}(m' + S(m'))$ that, given the packed representation of any $X \in [0\,..\,\sigma)^{\leq \ell}$ and any $i \in [1\,..\,\mathsf{prefix\text{-}rank}_W(m', X)]$, returns $\mathsf{prefix\text{-}select}_W(i, X)$ in $\mathcal{O}(Q(m'))$ time.

Proof. Let $k = \lceil \log \sigma \rceil$. Let $W_{\mathrm{bin}}[1\,..\,m']$ denote a sequence of binary strings of length $\ell_{\mathrm{bin}} = 1 + \lfloor \log m' \rfloor$ defined so that for every $j \in [1\,..\,m']$, $W_{\mathrm{bin}}[j]$ is a prefix of length ℓ_{bin} of the string $\mathrm{bin}_k(W[j]) \cdot 0^\infty$. Note that since we assumed $\ell \leq (1 + \lfloor \log m' \rfloor)/\lceil \log \sigma \rceil$, it follows (see Definition 4.1) that $\ell \cdot k \leq 1 + \lfloor \log m' \rfloor = \ell_{\mathrm{bin}}$, i.e., for every $j \in [1\,..\,m']$, $\mathrm{bin}_k(W[j])$ is a prefix of $W_{\mathrm{bin}}[j]$.

Components The data structure consists of the following components:

1. The data structure from Proposition 4.2 constructed for $u = m'$. Given the packed representation of any $S \in [0\,..\,\sigma)^*$, we can then compute the packed representation of $\mathrm{bin}_k(S)$ (Definition 4.1) in $\mathcal{O}(1 + |S|/\log_\sigma m')$ time. It needs $\mathcal{O}(m'/\log_\sigma m') = \mathcal{O}(m')$ space.
2. The data structure from the claim constructed for $W_{\mathrm{bin}}[1\,..\,m']$. It needs $\mathcal{O}(S(m'))$ space.

In total, the data structure needs $\mathcal{O}(m' + S(m'))$ space.

Implementation of queries The queries are answered as follows. Assume that we are given the packed representation of some $X \in [0\,..\,\sigma)^{\leq \ell}$ and some position $i \in [1\,..\,\mathsf{prefix\text{-}rank}_W(m', X)]$. The value $\mathsf{prefix\text{-}select}_W(i, X)$ is computed as follows:

1. Using Proposition 4.2, in $\mathcal{O}(1 + |X|/\log_\sigma m') = \mathcal{O}(1 + \ell/\log_\sigma m') = \mathcal{O}(1)$ time we compute the packed representation of the string $X_{\mathrm{bin}} = \mathrm{bin}_k(X)$.

2. Using the structure from the claim constructed for W_{bin}, we compute $j = \mathsf{prefix\text{-}select}_{W_{\text{bin}}}(i, X_{\text{bin}})$ in $\mathcal{O}(Q(m'))$ time. Note that $|X_{\text{bin}}| \leq |X| \cdot k \leq \ell \cdot k \leq \ell_{\text{bin}}$. Thus, we can indeed ask such query for W_{bin}. Moreover, note that by Lemma 4.19(1a), it holds $\mathsf{prefix\text{-}rank}_W(m', X) = \mathsf{prefix\text{-}rank}_{W_{\text{bin}}}(m', X_{\text{bin}})$. Thus, $i \in [1\,..\,\mathsf{prefix\text{-}rank}_{W_{\text{bin}}}(m', X_{\text{bin}})]$, i.e., $\mathsf{prefix\text{-}select}_{W_{\text{bin}}}(i, X_{\text{bin}})$ is well-defined. Finally, note that by Lemma 4.19(1b), it also holds $\mathsf{prefix\text{-}select}_W(i, X) = \mathsf{prefix\text{-}select}_{W_{\text{bin}}}(i, X_{\text{bin}})$. We thus return j as the answer.

In total, the query takes $\mathcal{O}(Q(m'))$ time.

Construction algorithm The components of the data structure are constructed as follows:

1. In $\mathcal{O}(m'/\log_\sigma m') = \mathcal{O}(m')$ time we construct the structure from Proposition 4.2 for $u = m'$.
2. To construct the second component, we first construct the sequence $W_{\text{bin}}[1\,..\,m']$ using Proposition 4.2. Computing $\mathrm{ebin}_k(W[j])$ for a single position $j \in [1\,..\,m']$ takes $\mathcal{O}(1 + |W[j]|/\log_\sigma m') = \mathcal{O}(1 + \ell/\log_\sigma m') = \mathcal{O}(1)$ time. Padding with zeros to the length ℓ_{bin} also takes $\mathcal{O}(1)$ time. Thus, over all $j \in [1\,..\,m']$, we spend $\mathcal{O}(m')$ time. We then apply the preprocessing from the claim to the sequence $W_{\text{bin}}[1\,..\,m']$. It takes $\mathcal{O}(m' + P_t(m'))$ time and uses $\mathcal{O}(m' + P_s(m'))$ working space.

In total, the construction takes $\mathcal{O}(m' + P_t(m'))$ time and uses $\mathcal{O}(m' + P_s(m'))$ working space. □

4.3.4 Summary

Theorem 4.21. *Consider a data structure answering prefix select queries (see Section 4.1) that, for any sequence of $m \geq 1$ binary strings of length $1 + \lfloor \log m \rfloor$, achieves the following complexities (where both the string at query time as well as input strings during construction are given in the packed representation):*

- *space usage $S(m)$,*
- *preprocessing time $P_t(m)$,*
- *preprocessing space $P_s(m)$,*
- *query time $Q(m)$.*

For every $T \in \{0, 1\}^n$, there exists $m = \Theta(n/\log n)$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log n + P_t(m))$ time and using $\mathcal{O}(n/\log n + P_s(m))$ working space construct a data structure of size $\mathcal{O}(n/\log n + S(m))$ that answers suffix array queries on T (see Section 4.1) in $\mathcal{O}(\log\log n + Q(m))$ time.

Proof. Assume that $n > 3^{13} - 1$ (otherwise, the claim holds trivially).

We use the following definitions. Let $\sigma = 3$ and let $T' \in [0\,..\,\sigma)^{n+1}$ be a string defined so that $T'[1\,..\,n] = T$ and $T'[n+1] = 2$. Denote $n' = |T'|$. Note that $T'[n']$ does not occur in $T'[1\,..\,n')$ and it holds $2 \leq \sigma < (n')^{1/13}$. Let D denote the data structure resulting from applying Proposition 4.20 with alphabet size σ to the structure (answering prefix select queries on binary strings) form the claim. Given any sequence $W[1\,..\,m']$ of $m' \geq 1$ equal-length strings over alphabet $[0\,..\,\sigma)$ of length $\ell' \leq (1 + \lfloor \log m' \rfloor)/\lceil \log \sigma \rceil$ as input (with all strings represented in the packed form), the structure D achieves the following complexities (where at query time it is given the packed representation of any $X \in [0\,..\,\sigma)^{\leq \ell'}$ and any $i \in [1\,..\,\mathsf{prefix\text{-}rank}_W(m', X)]$, and returns $\mathsf{prefix\text{-}select}_W(i, X)$):

- space usage $S'(m', \ell', \sigma) = \mathcal{O}(m' + S(m'))$,
- preprocessing time $P'_t(m', \ell', \sigma) = \mathcal{O}(m' + P_t(m'))$,
- preprocessing space $P'_s(m', \ell', \sigma) = \mathcal{O}(m' + P_s(m'))$,
- query time $Q'(m', \ell', \sigma) = \mathcal{O}(Q(m'))$,

where the complexities $S(m')$, $P_t(m')$, $P_s(m')$, and $Q(m')$ refer to the structure from the claim (answering prefix select for binary strings).

Components The data structure answering suffix array queries consists of a single component: the data structure from Theorem 4.18 applied for text T' and with the structure D as the structure answering prefix select queries. Note that we can use D, since it supports prefix select queries for the combination of parameters required in Theorem 4.18, i.e., for sequences over alphabet $[0\,..\,\sigma)$, with the length ℓ of all strings satisfying $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ (where k is the length of the input sequence). To bound the space usage of this component, note that by Theorem 4.18 and the above discussion, there exists $m = \Theta(n'/\log_\sigma n') = \Theta(n/\log n)$

(recall that $\sigma = 3$) such that the structure needs $\mathcal{O}(n'/\log_\sigma n' + S'(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + S(m)) = \mathcal{O}(n/\log n + S(m))$ space.

Implementation of queries The suffix array queries are answered as follows. Let $i \in [1\,..\,n]$. Note that by definition of T', it follows that $\mathrm{SA}_T[i] = \mathrm{SA}_{T'}[i]$. By Theorem 4.18 and the above discussion, computing $\mathrm{SA}_{T'}[i]$ takes $\mathcal{O}(\log\log n' + Q'(m,\ell,\sigma)) = \mathcal{O}(\log\log n + Q(m))$ time.

Construction algorithm By Theorem 4.18 and the above discussion, construction of the above data structure answering suffix array queries on T' (and hence also on T) takes $\mathcal{O}(n'/\log_\sigma n' + P'_t(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + P_t(m)) = \mathcal{O}(n/\log n + P_t(m))$ time and uses $\mathcal{O}(n'/\log_\sigma n' + P'_s(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + P_s(m)) = \mathcal{O}(n/\log n + P_s(m))$ working space. □

5 Equivalence of Inverse Suffix Array and Prefix Special Rank Queries

5.1 Problem Definitions

Indexing for Inverse Suffix Array Queries

Input: The packed representation of a string $T \in \{0,1\}^n$.
Output: A data structure that, given any $i \in [1\,..\,n]$, returns $\mathrm{ISA}_T[i]$, i.e., the value

$$|\{j \in [1\,..\,n] : T[j\,..\,n] \preceq T[i\,..\,n]\}|.$$

Indexing for Prefix Special Rank Queries

Input: A sequence $W[1\,..\,m]$ of $m \geq 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$, with all strings represented in the packed form.
Output: A data structure that, given any index $i \in [1\,..\,m]$ and any length $p \in [0\,..\,\ell]$, returns $\mathsf{prefix\text{-}special\text{-}rank}_W(i,p)$ (Definition 2.8), i.e., the value

$$|\{j \in [1\,..\,i] : X \text{ is a prefix of } W[j]\}|,$$

where $X = W[i][1\,..\,p]$.

5.2 Reducing Prefix Special Rank to Inverse Suffix Array Queries

5.2.1 Problem Reduction

Lemma 5.1. *Let $W[1\,..\,m]$ be an array of $m \geq 1$ strings of length $\ell \geq 1$ over alphabet $\{0,1\}$. Denote $T = \mathrm{SeqToString}_\ell(W)$ (Definition 4.7). Then, for every $i \in [1\,..\,m]$ and every $p \in [0\,..\,\ell]$, it holds*

$$\mathsf{prefix\text{-}special\text{-}rank}_W(i,p) = \mathrm{ISA}_T[(i\beta - (k+p)] - \alpha,$$

where X is a prefix of $W[i]$ of length p, $\alpha = \mathrm{RangeBeg}(\overline{X}\cdot 4, T)$, and $\beta = \ell + \lfloor \log m \rfloor + 2$.

Proof. Let k and $s(i)$ (where $i \in [0\,..\,m]$) be as in Definition 4.7. By definition of the prefix special rank query, $\mathsf{prefix\text{-}special\text{-}rank}_W(i,p) = j$ holds if and only if $j \in [1\,..\,\mathsf{prefix\text{-}rank}_W(m,X)]$ and $\mathsf{prefix\text{-}select}_W(j,X) = i$. We will prove the claim by showing that the latter two conditions are satisfied for $j = \mathrm{ISA}_T[i\beta - (k+p)] - \alpha$.

- First, we prove that $j \in [1\,..\,\mathsf{prefix\text{-}rank}_W(m,X)]$. To show $j \geq 1$, observe that the suffix $T(i\beta\,..\,|T|]$ is preceded with a string $\overline{W[i]}\cdot 4 \cdot s(i-1)$. Since X is a prefix of $W[i]$ of length p (which is equivalent to $\overline{X}$ being a suffix of length p of $\overline{W[i]}$), it follows that $T[i\beta - (k+p)\,..\,|T|]$ has the substring $\overline{X}\cdot 4$ as a prefix. Consequently, letting $\alpha' = \mathrm{RangeEnd}(\overline{X}\cdot 4, T)$, we have $\mathrm{ISA}_T[i\beta - (k+p)] \in (\alpha\,..\,\alpha']$. In

particular, $j = \mathrm{ISA}_T[i\beta - (k+p)] - \alpha \geq 1$. To show $j \leq \mathsf{prefix\text{-}rank}_W(m, X)$, recall that in the proof of Lemma 4.13, we showed that $|\mathrm{Occ}(\overline{X} \cdot \mathtt{4}, T)| = \mathsf{prefix\text{-}rank}_W(m, X)$. Thus, $j = \mathrm{ISA}_T[i\beta - (k+p) - \alpha \leq \alpha' - \alpha = |\mathrm{Occ}(\overline{X} \cdot \mathtt{4}, T)| = \mathsf{prefix\text{-}rank}_W(m, X)$.
- To prove that $\mathsf{prefix\text{-}select}_W(j, X) = i$, we apply Lemma 4.13. More precisely, it holds

$$\begin{aligned}\mathsf{prefix\text{-}select}_W(j, X) &= \mathsf{prefix\text{-}select}_W(\mathrm{ISA}_T[i\beta - (k+p)] - \alpha, X)\\ &= \left\lceil \frac{\mathrm{SA}_T[\alpha + \mathrm{ISA}_T[i\beta - (k+p)] - \alpha]}{\beta} \right\rceil\\ &= \left\lceil \frac{\mathrm{SA}_T[\mathrm{ISA}_T[i\beta - (k+p)]]}{\beta} \right\rceil = \left\lceil \frac{i\beta - (k+p)}{\beta} \right\rceil = i. \qquad \square\end{aligned}$$

5.2.2 Alphabet Reduction for Inverse Suffix Array Queries

Proposition 5.2 ([KK25]). *Let $T \in [0\,..\,\sigma)^n$ be a nonempty string, where $\sigma < n^{\mathcal{O}(1)}$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time compute integers Δ and δ, and the packed representation of a string $T' \in \{\mathtt{0},\mathtt{1}\}^+$ such that $|T'| = \Theta(n \log \sigma)$, and for every $j \in [1\,..\,n]$, it holds*

$$\mathrm{ISA}_T[j] = \mathrm{ISA}_{T'}[(j-1)\delta + 1] - \Delta.$$

5.2.3 Summary

Theorem 5.3. *Consider a data structure answering inverse suffix array queries (see Section 5.1) that, for any text $T \in \{\mathtt{0},\mathtt{1}\}^n$, achieves the following complexities (where in the preprocessing we assume that we are given as input the packed representation of T):*

- *space usage $S(n)$,*
- *preprocessing time $P_t(n)$,*
- *preprocessing space $P_s(n)$,*
- *query time $Q(n)$.*

For every sequence $W[1\,..\,m]$ of $m \geq 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$, there exists $n = \mathcal{O}(m \log m)$ such that, given the sequence W with all strings represented in the packed form, we can in $\mathcal{O}(m + P_t(n))$ time and using $\mathcal{O}(m + P_s(n))$ working space construct a data structure of size $\mathcal{O}(m + S(n))$ that, given any $i \in [1\,..\,m]$ and $p \in [0\,..\,\ell]$, computes $\mathsf{prefix\text{-}special\text{-}rank}_W(i, p)$ in $\mathcal{O}(Q(n))$ time.

Proof. We use the following definitions. Let $T_{\mathrm{aux}} = \mathrm{SeqToString}_\ell(W)$ (Definition 4.7). Denote $n_{\mathrm{aux}} = |T_{\mathrm{aux}}| = m \cdot (2\ell + 1) = \Theta(m \log m)$. Let Δ, δ, and T_{bin} denote the two integers and the text resulting from applying Proposition 5.2 to text T_{aux}. Denote $n = |T_{\mathrm{bin}}|$. By Proposition 5.2, we have $n = \Theta(n_{\mathrm{aux}}) = \Theta(m \log m)$, and for every $j \in [1\,..\,n_{\mathrm{aux}}]$, it holds $\mathrm{ISA}_{T_{\mathrm{aux}}}[j] = \mathrm{ISA}_{T_{\mathrm{bin}}}[(j-1)\delta + 1] - \Delta$.

Components The data structure consists of the following components:

1. The integers Δ and δ defined as above.
2. The data structure from the claim for the text T_{bin}. It needs $\mathcal{O}(S(n))$ space.
3. The structure resulting from applying Proposition 4.11 to the array W. It needs $\mathcal{O}(m)$ space.

In total, the structure needs $\mathcal{O}(m + S(n))$ space.

Implementation of queries Given any $i \in [1\,..\,m]$ and $p \in [0\,..\,\ell]$, we compute $\mathsf{prefix\text{-}special\text{-}rank}_W(i, p)$ as follows:

1. In $\mathcal{O}(1)$ time compute the packed representation of X, defined as a length-p prefix of $W[i]$.
2. Using Proposition 4.11, in $\mathcal{O}(1)$ time compute $\alpha_{\mathrm{aux}} := \mathrm{RangeBeg}(\overline{X} \cdot \mathtt{4}, T_{\mathrm{aux}})$.
3. Set $j_{\mathrm{aux}} := i\beta - (\ell + p)$, where $\beta = 2\ell + 1$. Using the structure from the claim constructed for text T_{bin}, in $\mathcal{O}(Q(n))$ time compute $i_{\mathrm{aux}} := \mathrm{ISA}_{T_{\mathrm{bin}}}[(j_{\mathrm{aux}} - 1)\delta + 1] - \Delta$. By Proposition 5.2 (and the discussion above), $i_{\mathrm{aux}} = \mathrm{ISA}_{T_{\mathrm{aux}}}[j_{\mathrm{aux}}] = \mathrm{ISA}_{T_{\mathrm{aux}}}[i\beta - (\ell + p)]$.
4. In $\mathcal{O}(1)$ time compute $r := i_{\mathrm{aux}} - \alpha_{\mathrm{aux}}$. By Lemma 5.1, it holds $r = \mathsf{prefix\text{-}special\text{-}rank}_W(i, p)$. Thus, we return r as the answer.

In total, the query takes $\mathcal{O}(Q(n))$ time.

Construction algorithm The components of the data structure are constructed as follows:

1. To compute Δ and δ (defined above), we proceed as follows. First, in $\mathcal{O}(m)$ time we apply Proposition 4.12 to the sequence W to compute the packed representation of the string T_{aux} (defined above). Then, we apply Proposition 5.2 for T_{aux} to compute the packed representation of T_{bin} (as defined above), together with values Δ and δ. Since T_{aux} is over alphabet $\{0, \ldots, 4\}$, the application of Proposition 5.2 takes $\mathcal{O}(|T_{\mathrm{aux}}|/\log|T_{\mathrm{aux}}|) = \mathcal{O}(m)$ time.
2. In the second step, we apply the preprocessing from the claim to string T_{bin}. This takes $\mathcal{O}(P_t(n))$ time and uses $\mathcal{O}(P_s(n))$ working space.
3. Finally, we apply Proposition 4.11 to the sequence W. This takes $\mathcal{O}(m)$ time.

In total, the construction takes $\mathcal{O}(m + P_t(n))$ time and uses $\mathcal{O}(m + P_s(n))$ working space. □

5.3 Reducing Inverse Suffix Array to Prefix Special Rank Queries

5.3.1 Problem Reduction

Theorem 5.4 ([KK23a]). *Consider a data structure that, given any sequence W of k strings of length ℓ over alphabet $[0\,..\,\sigma)$ achieves the following complexities (where the input strings during construction are given in the packed representation, and at query time we are given any $i \in [1\,..\,k]$ and any $p \in [0\,..\,\ell]$, and we return $\mathsf{prefix\text{-}special\text{-}rank}_W(i,p)$; see Section 2.4):*

- *space usage $S(k,\ell,\sigma)$,*
- *preprocessing time $P_t(k,\ell,\sigma)$,*
- *preprocessing space $P_s(k,\ell,\sigma)$,*
- *query time $Q(k,\ell,\sigma)$.*

For every $T \in [0\,..\,\sigma)^n$ such that $2 \le \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$, there exist positive integers $k = \Theta(n/\log_\sigma n)$ and $\ell \le (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n + P_t(k,\ell,\sigma))$ time and using $\mathcal{O}(n/\log_\sigma n + P_s(k,\ell,\sigma))$ working space build a data structure of size $\mathcal{O}(n/\log_\sigma n + S(k,\ell,\sigma))$ that answers inverse suffix array queries on T (see Section 2) in $\mathcal{O}(\log\log n + Q(k,\ell,\sigma))$ time.

Proof. The proof is almost identical to the proof of Theorem 4.18, and reduces to performing cosmetic fixes to [KK23a, Theorem 5.2]. The main difference here is to observe that during inverse suffix array queries, the structure from [KK23a] performs only prefix special rank queries; see [KK23a, Proposition 5.4]. Observe also that performing the padding of the input for prefix special rank queries does not change the answers by Observation 4.17. □

5.3.2 Alphabet Reduction for Prefix Special Rank Queries

Proposition 5.5. *Consider a data structure answering prefix special rank queries (see Section 5.1) that, for any sequence of $m \ge 1$ binary strings of length $1 + \lfloor \log m \rfloor$, achieves the following complexities (where the input strings during construction are given in the packed representation):*

- *space usage $S(m)$,*
- *preprocessing time $P_t(m)$,*
- *preprocessing space $P_s(m)$,*
- *query time $Q(m)$.*

Let $W[1\,..\,m']$ be a sequence of $m' \ge 1$ equal-length strings over alphabet $[0\,..\,\sigma)$ (where $\sigma \ge 2$) of length $\ell \le (1 + \lfloor \log m' \rfloor)/\lceil \log \sigma \rceil$, Given the sequence W, with all strings represented in the packed form, we can in $\mathcal{O}(m' + P_t(m'))$ time and using $\mathcal{O}(m' + P_s(m'))$ working space construct a data structure of size $\mathcal{O}(m' + S(m'))$ that, given any $i \in [1\,..\,m']$ and $p \in [0\,..\,\ell]$, returns $\mathsf{prefix\text{-}special\text{-}rank}_W(i,p)$ in $\mathcal{O}(Q(m'))$ time.

Proof. Let $k = \lceil \log \sigma \rceil$. Let $W_{\mathrm{bin}}[1\,..\,m']$ denote a sequence of binary strings of length $\ell_{\mathrm{bin}} = 1 + \lfloor \log m' \rfloor$ defined so that for every $j \in [1\,..\,m']$, $W_{\mathrm{bin}}[j]$ is a prefix of length ℓ_{bin} of the string $\mathrm{bin}_k(W[j]) \cdot \mathtt{0}^\infty$. Note that

since we assumed $\ell \leq (1 + \lfloor \log m' \rfloor)/\lceil \log \sigma \rceil$, it follows (see Definition 4.1) that $\ell \cdot k \leq 1 + \lfloor \log m' \rfloor = \ell_{\text{bin}}$, i.e., for every $j \in [1\,..\,m']$, $\text{bin}_k(W[j])$ is a prefix of $W_{\text{bin}}[j]$.

Components The data structure answering prefix special rank queries on $W[1\,..\,m']$ consist of single component: the data structure from the claim constructed for $W_{\text{bin}}[1\,..\,m']$. It needs $\mathcal{O}(S(m'))$ space.

Implementation of queries The queries are answered as follows. Assume that we are given some $i \in [1\,..\,m']$ and $p \in [0\,..\,\ell]$. To compute the value of $\mathsf{prefix\text{-}special\text{-}rank}_W(i, p)$, we simply return the value $\mathsf{prefix\text{-}special\text{-}rank}_{W_{\text{bin}}}(i, p \cdot k)$ in $\mathcal{O}(Q(m'))$ time. This is correct by Lemma 4.19(2).

Construction algorithm To construct the above data structure, we proceed as follows. First, in $\mathcal{O}(m'/\log_\sigma m') = \mathcal{O}(m')$ time we construct the structure from Proposition 4.2 for $u = m'$. We then construct the sequence $W_{\text{bin}}[1\,..\,m']$ using Proposition 4.2. Computing $\text{ebin}_k(W[j])$ for a single position $j \in [1\,..\,m']$ takes $\mathcal{O}(1 + |W[j]|/\log_\sigma m') = \mathcal{O}(1 + \ell/\log_\sigma m') = \mathcal{O}(1)$ time. Padding with zeros to the length ℓ_{bin} also takes $\mathcal{O}(1)$ time. Thus, over all $j \in [1\,..\,m']$, we spend $\mathcal{O}(m')$ time. We then apply the preprocessing from the claim to the sequence $W_{\text{bin}}[1\,..\,m']$. It takes $\mathcal{O}(m' + P_t(m'))$ time and uses $\mathcal{O}(m' + P_s(m'))$ working space. In total, the construction takes $\mathcal{O}(m' + P_t(m'))$ time and uses $\mathcal{O}(m' + P_s(m'))$ working space. □

5.3.3 Summary

Theorem 5.6. *Consider a data structure answering prefix special rank queries (see Section 5.1) that, for any sequence of $m \geq 1$ binary strings of length $1 + \lfloor \log m \rfloor$, achieves the following complexities (where the input strings during construction are given in the packed representation):*

- *space usage $S(m)$,*
- *preprocessing time $P_t(m)$,*
- *preprocessing space $P_s(m)$,*
- *query time $Q(m)$.*

For every $T \in \{\mathtt{0}, \mathtt{1}\}^n$, there exists $m = \Theta(n/\log n)$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log n + P_t(m))$ time and using $\mathcal{O}(n/\log n + P_s(m))$ working space construct a data structure of size $\mathcal{O}(n/\log n + S(m))$ that answers inverse suffix array queries on T (see Section 5.1) in $\mathcal{O}(\log\log n + Q(m))$ time.

Proof. Assume that $n > 3^{13} - 1$ (otherwise, the claim holds trivially).

We use the following definitions. Let $\sigma = 3$ and let $T' \in [0\,..\,\sigma)^{n+1}$ be a string defined so that $T'[1\,..\,n] = T$ and $T'[n+1] = \mathtt{2}$. Denote $n' = |T'|$. Note that $T'[n']$ does not occur in $T'[1\,..\,n')$ and it holds $2 \leq \sigma < (n')^{1/13}$. Let D denote the data structure resulting from applying Proposition 5.5 with alphabet size σ to the structure (answering prefix special rank queries on binary strings) form the claim. Given any sequence $W[1\,..\,m']$ of $m' \geq 1$ equal-length strings over alphabet $[0\,..\,\sigma)$ of length $\ell' \leq (1 + \lfloor \log m' \rfloor)/\lceil \log \sigma \rceil$ as input (with all strings represented in the packed form), the structure D achieves the following complexities (where at query time it is given some $i \in [1\,..\,m']$ and $p \in [0\,..\,\ell]$, and returns the value $\mathsf{prefix\text{-}special\text{-}rank}_W(i, p)$):

- space usage $S'(m', \ell', \sigma) = \mathcal{O}(m' + S(m'))$,
- preprocessing time $P'_t(m', \ell', \sigma) = \mathcal{O}(m' + P_t(m'))$,
- preprocessing space $P'_s(m', \ell', \sigma) = \mathcal{O}(m' + P_s(m'))$,
- query time $Q'(m', \ell', \sigma) = \mathcal{O}(Q(m'))$,

where the complexities $S(m')$, $P_t(m')$, $P_s(m')$, and $Q(m')$ refer to the structure from the claim (answering prefix special rank for binary strings).

Components The data structure answering inverse suffix array queries consists of a single component: the data structure from Theorem 5.4 applied for text T' and with the structure D as the structure answering prefix special rank queries. Note that we can use D, since it supports prefix special rank queries for the combination of parameters required in Theorem 5.4, i.e., for sequences over alphabet $[0\,..\,\sigma)$, with the length ℓ of all strings satisfying $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ (where k is the length of the input sequence). To bound the space usage of this component, note that by Theorem 5.4 and the above discussion, there

exists $m = \Theta(n'/\log_\sigma n') = \Theta(n/\log n)$ (recall that $\sigma = 3$) such that the structure needs $\mathcal{O}(n'/\log_\sigma n' + S'(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + S(m)) = \mathcal{O}(n/\log n + S(m))$ space.

Implementation of queries The inverse suffix array queries are answered as follows. Let $i \in [1\,..\,n]$. Note that by definition of T', it follows that $\mathrm{ISA}_T[i] = \mathrm{ISA}_{T'}[i]$. By Theorem 5.4 and the above discussion, computing $\mathrm{ISA}_{T'}[i]$ takes $\mathcal{O}(\log\log n' + Q'(m,\ell,\sigma)) = \mathcal{O}(\log\log n + Q(m))$ time.

Construction algorithm By Theorem 5.4 and the above discussion, construction of the above data structure answering inverse suffix array queries on T' (and hence also on T) takes $\mathcal{O}(n'/\log_\sigma n' + P'_t(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + P_t(m)) = \mathcal{O}(n/\log n + P_t(m))$ time and uses $\mathcal{O}(n'/\log_\sigma n' + P'_s(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + P_s(m)) = \mathcal{O}(n/\log n + P_s(m))$ working space. □

6 Equivalence of Pattern Ranking and Prefix Rank Queries

6.1 Problem Definitions

Indexing for Pattern Ranking Queries

Input: The packed representation of a string $T \in \{0,1\}^n$.
Output: A data structure that, given the packed representation of any $P \in \{0,1\}^*$, returns

$$\mathrm{RangeBeg}(P,T) = |\{j \in [1\,..\,n] : T[j\,..\,n] \prec P\}|.$$

Indexing for Prefix Rank Queries

Input: A sequence $W[1\,..\,m]$ of $m \geq 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$, with all strings represented in the packed form.
Output: A data structure that, given the packed representation of any $X \in \{0,1\}^{\leq \ell}$ and any $i \in [0\,..\,m]$, returns $\mathsf{prefix\text{-}rank}_W(i,X)$, i.e., the value $|\{j \in [1\,..\,i] : X \text{ is a prefix of } W[j]\}|$.

6.2 Reducing Prefix Rank to Pattern Ranking Queries

6.2.1 Problem Reduction

Lemma 6.1. *Let $W[1\,..\,m]$ be an array of $m \geq 1$ strings of length $\ell \geq 1$ over alphabet $\{0,1\}$. Denote $T = \mathrm{SeqToString}_\ell(W)$ (Definition 4.7). Let $s(i)$ (where $i \in [0\,..\,m]$) be as in Definition 4.7. For every $X \in \{0,1\}^{\leq \ell}$ and $i \in [0\,..\,m]$, it holds*

$$\mathsf{prefix\text{-}rank}_W(i,X) = \mathrm{RangeBeg}(X',T) - \alpha,$$

where $X' = \overline{X} \cdot 4 \cdot s(i)$ and $\alpha = \mathrm{RangeBeg}(\overline{X} \cdot 4, T)$.

Proof. Denote

$$A = \{j \in \mathrm{Occ}(\overline{X} \cdot 4, T) : T[j\,..\,n] \prec X'\}.$$

Observe that because $\overline{X} \cdot 4$ is a prefix of X', it follows that $\mathrm{RangeBeg}(X',T) = \mathrm{RangeBeg}(\overline{X} \cdot 4, T) + |A| = \alpha + |A|$. This implies that we only need to show that $\mathsf{prefix\text{-}rank}_W(i,X) = |A|$.

Let $k = 1 + \lfloor \log m \rfloor$ and $\beta = \ell + k + 1$. Denote $q = \mathsf{prefix\text{-}rank}_W(m,X)$. Let $(a_i)_{i \in [1\,..\,q]}$ and $(b_i)_{i \in [1\,..\,q]}$ be sequences defined by $a_i = \mathsf{prefix\text{-}select}_W(i,X)$ and $b_i = a_i \cdot \beta - (k + |X|)$. Recall that in the proof of Lemma 4.13, we showed that $\mathrm{Occ}(\overline{X} \cdot 4, T) = \{b_1, \ldots, b_q\}$ and $T[b_1\,..\,|T|] \prec T[b_2\,..\,|T|] \prec \cdots \prec T[b_q\,..\,|T|]$. Denote $r = \mathsf{prefix\text{-}rank}_W(i,X)$, and consider three cases:

- First, assume that $0 < r < q$. By definition of the prefix rank query, this implies that $a_r \leq i < a_{r+1}$. Thus, we also have $s(a_r - 1) \prec s(i) \preceq s(a_{r+1} - 1)$. Observe now (see also the proof of Lemma 4.13), that for every $i \in [1\,..\,q]$, $\overline{X} \cdot 4 \cdot s(a_i - 1)$ is a prefix of $T[b_i\,..\,|T|]$. In particular $\overline{X} \cdot 4 \cdot s(a_r - 1)$ is a prefix of $T[b_r\,..\,|T|]$

and $\overline{X}\cdot\mathtt{4}\cdot s(a_{r+1}-1)$ is a prefix of $T[b_{r+1}\,..\,|T|]$. By $X'=\overline{X}\cdot\mathtt{4}\cdot s(i)$ and $s(a_r-1)\prec s(i)\preceq s(a_{r+1}-1)$ (and the fact that all strings $s(a_r-1)$, $s(i)$, and $s(a_{r+1}-1)$ are of the same length), we thus obtain that $T[b_r\,..\,|T|]\prec X'\preceq T[b_{r+1}\,..\,|T|]$. By $T[b_1\,..\,|T|]\prec T[b_2\,..\,|T|]\prec\cdots\prec T[b_q\,..\,|T|]$, this implies that $A=\{b_1,\ldots,b_r\}$, i.e., $|A|=r$.
- Let us now assume that $r=0$. This implies that $i<a_0$. Hence, $s(i)\preceq s(a_0-1)$, and thus $X'\preceq T[b_1\,..\,|T|]$. We thus have $A=\emptyset$, and hence $|A|=r$.
- Let us now assume that $r=q$. We then have $a_q\leq i$. Thus, $s(a_q-1)\prec s(i)$, and hence $T[b_q\,..\,|T|]\prec X'$. We thus have $A=\{b_1,\ldots,b_q\}$, and hence $|A|=r$.

We have thus proved that in all cases it holds $|A|=r=\mathsf{prefix\text{-}rank}_W(i,X)$. By the above discussion, this concludes the proof. □

6.2.2 Alphabet Reduction for Pattern Ranking Queries

Lemma 6.2. *Let $\sigma\geq 2$, $T\in[0\,..\,\sigma)^*$, and $P\in[0\,..\,\sigma)^*$. Then, it holds*

$$\mathrm{RangeBeg}(P,T)=\mathrm{RangeBeg}(P',T')-\Delta,$$

where $k=\lceil\log\sigma\rceil$, $P'=\mathrm{ebin}_k(P)$ (Definition 4.3), $T'=\mathrm{ebin}_k(T)$, and $\Delta=|T'|-|T|$.

Proof. Denote

$$A=\{j\in\mathrm{Occ}(\mathtt{1}^{k+1},T'):T'[j\,..\,|T'|]\prec P'\}.$$

Observe that since $\mathtt{1}^{k+1}$ is a prefix of P', it follows that $\mathrm{RangeBeg}(P',T')=\mathrm{RangeBeg}(\mathtt{1}^{k+1},T')+|A|=\Delta+|A|$. Thus, it suffices to prove that $\mathrm{RangeBeg}(P,T)=|A|$.

First, note that $\mathrm{Occ}(\mathtt{1}^{k+1},T')=\{1+\delta\cdot(j-1):j\in[1\,..\,|T|]\}$, where $\delta=2k+3$. This implies that, letting $\mathcal{S}=\{\mathrm{ebin}_k(T[j\,..\,|T|]):j\in[1\,..\,|T|]\}$, we can equivalently write $\mathcal{S}=\{T'[j\,..\,|T'|]:j\in\mathrm{Occ}(\mathtt{1}^{k+1},T')\}$. By Lemma 4.4 and the definition of $\mathrm{RangeBeg}(P,T)$, we thus have

$$\begin{aligned}\mathrm{RangeBeg}(P,T)&=|\{j\in[1\,..\,|T|]:T[j\,..\,|T|]\prec P\}|\\&=|\{S\in\mathcal{S}:S\prec P'\}|\\&=|\{j\in\mathrm{Occ}(\mathtt{1}^{k+1},T):T'[j\,..\,|T'|]\prec P'\}|=|A|.\end{aligned}$$

□

Proposition 6.3. *Let $T\in[0\,..\,\sigma)^n$ be a nonempty string, where $2\leq\sigma<n^{\mathcal{O}(1)}$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct the packed representation of a text $T_{\mathrm{bin}}\in\{\mathtt{0},\mathtt{1}\}^*$ satisfying $|T_{\mathrm{bin}}|=\Theta(n\log\sigma)$, an integer α, and a data structure that, given the packed representation of any $P\in[0\,..\,\sigma)^m$, in $\mathcal{O}(1+m/\log_\sigma n)$ time returns the packed representation of a string $P_{\mathrm{bin}}\in\{\mathtt{0},\mathtt{1}\}^*$ satisfying $|P_{\mathrm{bin}}|=m\cdot(2\lceil\log\sigma\rceil+3)$ and*

$$\mathrm{RangeBeg}(P,T)=\mathrm{RangeBeg}(P_{\mathrm{bin}},T_{\mathrm{bin}})-\alpha$$

Proof. Let $k=\lceil\log\sigma\rceil$. To compute α and the packed representation of T_{bin}, we proceed as follows:

1. In $\mathcal{O}(n/\log_\sigma n)$ time we construct the data structure from Proposition 4.5 for $u=n$. Given the packed representation of any $S\in[0\,..\,\sigma)^*$, we can then compute the packed representation of $\mathrm{ebin}_k(S)$ (Definition 4.3) in $\mathcal{O}(1+|S|/\log_\sigma n)$ time.
2. Using the above structure, we compute the packed representation of $T_{\mathrm{bin}}=\mathrm{ebin}_k(T)$ in $\mathcal{O}(n/\log_\sigma n)$ time.
3. In $\mathcal{O}(1)$ time we set $\alpha=|T_{\mathrm{bin}}|-|T|$.

In total, the construction of the packed representation of T_{bin} and α takes $\mathcal{O}(n/\log_\sigma n)$. By Definition 4.3, it holds $|T_{\mathrm{bin}}|=n\cdot(2k+3)=\Theta(n\log\sigma)$.

Next, we address the construction of the data structure from the claim. It consists of a single component: the data structure from Proposition 4.5 for $u=n$. It uses $\mathcal{O}(n/\log_\sigma n)$ space.

The queries are implemented as follows. Given the packed representation of $P\in[0\,..\,\sigma)^m$, we compute the packed representation of P_{bin} defined as $P_{\mathrm{bin}}=\mathrm{ebin}_k(P)$. Using the structure from Proposition 4.5, this takes $\mathcal{O}(1+m/\log_\sigma n)$ time. By Definition 4.3, it holds $|P_{\mathrm{bin}}|=m\cdot(2k+3)$, and by Lemma 6.2, we have $\mathrm{RangeBeg}(P,T)=\mathrm{RangeBeg}(P_{\mathrm{bin}},T_{\mathrm{bin}})-\alpha$.

By Proposition 4.5, construction of the data structure takes $\mathcal{O}(n/\log_\sigma n)$ time. □

6.2.3 Summary

Theorem 6.4. *Consider a data structure answering pattern ranking queries (see Section 6.1) that, for any text $T \in \{0,1\}^n$, achieves the following complexities (where in the preprocessing we assume that we are given as input the packed representation of T, and at query time we are given a packed representation of $P \in \{0,1\}^{\leq k}$):*

- *space usage $S(n)$,*
- *preprocessing time $P_t(n)$,*
- *preprocessing space $P_s(n)$,*
- *query time $Q(n,k)$.*

For every sequence $W[1\,..\,m]$ of $m \geq 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$, there exists $n = \mathcal{O}(m \log m)$ and $k = \mathcal{O}(\log m)$ such that, given the sequence W with all strings represented in the packed form, we can in $\mathcal{O}(m + P_t(n))$ time and using $\mathcal{O}(m + P_s(n))$ working space construct a data structure of size $\mathcal{O}(m + S(n))$ that, given any $i \in [0\,..\,m]$ and the packed representation of any $X \in \{0,1\}^{\leq \ell}$, computes $\mathsf{prefix\text{-}rank}_W(i, X)$ in $\mathcal{O}(Q(n,k))$ time.

Proof. We use the following definitions. Let $T_{\mathrm{aux}} = \mathrm{SeqToString}_\ell(W)$ (Definition 4.7). Denote $n_{\mathrm{aux}} = |T_{\mathrm{aux}}| = m \cdot (2\ell + 1) = \Theta(m \log m)$. Let T_{bin} and α_{bin} denote the text and an integer obtained by applying Proposition 6.3 to text T_{aux}. Denote $n = |T_{\mathrm{bin}}|$. Since T_{aux} is over alphabet $\{0,\ldots,4\}$, by Proposition 6.3 we have $n = \Theta(|T_{\mathrm{aux}}|) = \Theta(m \log m)$. Denote $\ell' = \lceil \ell/2 \rceil$ and $k = 18\lfloor \log m \rfloor + 27 = \mathcal{O}(\log m)$.

Components The data structure consists of the following components:

1. The structure from Proposition 6.3 applied to text T_{aux}. Since T_{aux} is over alphabet $\{0,\ldots,4\}$, the structure needs $\mathcal{O}(|T_{\mathrm{aux}}|/\log|T_{\mathrm{aux}}|) = \mathcal{O}(m)$ space.
2. The data structure from the claim (i.e., answering pattern ranking queries) for the text T_{bin}. The structure needs $\mathcal{O}(S(n))$ space.
3. The structure resulting from applying Proposition 4.11 to the array W. It needs $\mathcal{O}(m)$ space.
4. A lookup table L_{rev} such that for every $X \in \{0,1\}^{\leq \ell'}$, L_{rev} maps X to the packed representation of $\overline{X}$ (over alphabet $\{0,\ldots,4\}$). When accessing L_{rev}, we map X to $\mathrm{BinStrToInt}(X)$ (Definition 4.8). Thus, L_{rev} needs $\mathcal{O}(2^{\ell'}) = \mathcal{O}(\sqrt{n})$ space. Given the packed representation of any $X \in \{0,1\}^{\leq \ell}$ (over alphabet $\{0,1\}$), we can then in $\mathcal{O}(1)$ time compute the packed representation of $\overline{X}$ (over $\{0,\ldots,4\}$).
5. A lookup table L_{map} defined such that for every $X \in \{0,1\}^{\leq \ell'}$, L_{map} maps X to the packed representation of $\mathrm{sub}(\mathrm{sub}(X,0,2),1,3)$ (over alphabet $\{0,\ldots,4\}$). Similarly as above, L_{map} needs $\mathcal{O}(\sqrt{n})$ space. Given $i \in [0\,..\,m]$, we can then compute the packed representation (over alphabet $\{0,\ldots,4\}$) of the string $s(i)$ (Definition 4.7) in $\mathcal{O}(1)$ time.
6. The integer α_{bin} (defined above).

In total, the structure needs $\mathcal{O}(m + S(n))$ space.

Implementation of queries The queries are answered as follows. Assume that we are given some $i \in [0\,..\,m]$ and the packed representation of some $X \in \{0,1\}^{\leq \ell}$. The query algorithm proceeds as follows:

1. Using L_{rev} and L_{map}, in $\mathcal{O}(1)$ time we compute the packed representation of the string $X_{\mathrm{aux}} = \overline{X} \cdot 4 \cdot s(i)$ (over alphabet $\{0,\ldots,4\}$). Note that $|X_{\mathrm{aux}}| = |X| + 1 + \ell \leq 2\ell + 1 \leq 2\lfloor \log m \rfloor + 3$.
2. Using Proposition 4.11, in $\mathcal{O}(1)$ time we compute $\alpha_{\mathrm{aux}} := \mathrm{RangeBeg}(\overline{X} \cdot 4, T_{\mathrm{aux}})$.
3. Using Proposition 6.3, in $\mathcal{O}(1 + |X_{\mathrm{aux}}|/\log n_{\mathrm{aux}}) = \mathcal{O}(1 + \ell/\log m) = \mathcal{O}(1)$ time, we compute the packed representation of a string $X_{\mathrm{bin}} \in \{0,1\}^*$ of length $|X_{\mathrm{bin}}| = 9 \cdot |X_{\mathrm{aux}}| \leq 18\lfloor \log m \rfloor + 27 \leq k$ such that
$$\mathrm{RangeBeg}(X_{\mathrm{aux}}, T_{\mathrm{aux}}) = \mathrm{RangeBeg}(X_{\mathrm{bin}}, T_{\mathrm{bin}}) - \alpha_{\mathrm{bin}}.$$
4. Using the structure from the claim, we compute $b_{\mathrm{bin}} = \mathrm{RangeBeg}(X_{\mathrm{bin}}, T_{\mathrm{bin}})$. By $|X_{\mathrm{bin}}| \leq k$, this takes $\mathcal{O}(Q(n,k))$ time.
5. In $\mathcal{O}(1)$ time, we compute $b_{\mathrm{aux}} = b_{\mathrm{bin}} - \alpha_{\mathrm{bin}}$. By the above, $b_{\mathrm{aux}} = \mathrm{RangeBeg}(X_{\mathrm{aux}}, T_{\mathrm{aux}})$.
6. In $\mathcal{O}(1)$ time we compute and return $r = b_{\mathrm{aux}} - \alpha_{\mathrm{aux}}$. By Lemma 6.1, $\mathsf{prefix\text{-}rank}_W(i, X) = r$.

In total, the query takes $\mathcal{O}(Q(n,k))$ time.

Construction algorithm The components of the data structure are constructed as follows:

1. In $\mathcal{O}(m)$ time we apply Proposition 4.12 to the sequence W to compute the packed representation of the string T_{aux}. We then apply Proposition 6.3 to T_{aux}. Since T_{aux} is over alphabet $\{0, \ldots, 4\}$, this takes $\mathcal{O}(|T_{\text{aux}}|/\log|T_{\text{aux}}|) = \mathcal{O}(m)$ time. Note that Proposition 6.3, in addition to the data structure, also returns the integer α_{bin} and the packed representation of the string T_{bin} (defined above).
2. We apply the preprocessing from the claim to the string T_{bin}. This takes $\mathcal{O}(P_t(|T_{\text{bin}}|)) = \mathcal{O}(P_t(n))$ time and uses $\mathcal{O}(P_s(|T_{\text{bin}}|)) = \mathcal{O}(P_s(n))$ working space.
3. We apply Proposition 4.11 to the array W. This takes $\mathcal{O}(m)$ time.
4. The lookup table L_{rev} is computed in $\mathcal{O}(m)$ time similarly as in the proof of Proposition 4.12.
5. Next, we compute the lookup table L_{map}. Similarly as above, we proceed as in Proposition 4.12, and spend $\mathcal{O}(m)$ time.
6. Lastly, we store the integer α_{bin}, which was already computed above.

In total, the construction takes $\mathcal{O}(m + P_t(n))$ time and uses $\mathcal{O}(m + P_s(n))$ working space. □

6.3 Reducing Pattern Ranking to Prefix Rank Queries

6.3.1 Problem Reduction

Theorem 6.5 ([KK23a]). *Consider a data structure that, given any sequence W of k strings of length ℓ over alphabet $[0\,..\,\sigma)$ achieves the following complexities (where the input strings during construction are given in the packed representation, and at query time we are given the packed representation of any $X \in [0\,..\,\sigma)^{\leq \ell}$ and any $i \in [0\,..\,k]$, and we return $\mathsf{prefix\text{-}rank}_W(i, X)$; see Section 2.4):*

- *space usage $S(k, \ell, \sigma)$,*
- *preprocessing time $P_t(k, \ell, \sigma)$,*
- *preprocessing space $P_s(k, \ell, \sigma)$,*
- *query time $Q(k, \ell, \sigma)$.*

For every $T \in [0\,..\,\sigma)^n$ such that $\sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$, there exist positive integers $k = \Theta(n/\log_\sigma n)$ and $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n + P_t(k, \ell, \sigma))$ time and using $\mathcal{O}(n/\log_\sigma n + P_s(k, \ell, \sigma))$ working space build a data structure of size $\mathcal{O}(n/\log_\sigma n + S(k, \ell, \sigma))$ that, given the packed representation of any $P \in [0\,..\,\sigma)^*$, returns $\mathrm{RangeBeg}(P, T)$ (see Section 2) in $\mathcal{O}(\log\log n + Q(k, \ell, \sigma) + |P|/\log_\sigma n)$ time.

Proof. The proof is almost identical to the proof of Theorem 4.18, and reduces to performing cosmetic fixes to [KK23a, Theorem 6.3]. Note that performing the padding of the input for prefix rank queries does not change the answers by Observation 4.17. □

6.3.2 Alphabet Reduction for Prefix Rank Queries

Proposition 6.6. *Consider a data structure answering prefix rank queries (see Section 6.1) that, for any sequence of $m \geq 1$ binary strings of length $1 + \lfloor \log m \rfloor$, achieves the following complexities (where both the string at query time as well as input strings during construction are given in the packed representation):*

- *space usage $S(m)$,*
- *preprocessing time $P_t(m)$,*
- *preprocessing space $P_s(m)$,*
- *query time $Q(m)$.*

Let $W[1\,..\,m']$ be a sequence of $m' \geq 1$ equal-length strings over alphabet $[0\,..\,\sigma)$ (where $\sigma \geq 2$) of length $\ell \leq (1 + \lfloor \log m' \rfloor)/\lceil \log \sigma \rceil$, Given the sequence W, with all strings represented in the packed form, we can in $\mathcal{O}(m' + P_t(m'))$ time and using $\mathcal{O}(m' + P_s(m'))$ working space construct a data structure of size $\mathcal{O}(m' + S(m'))$ that, given the packed representation of any $X \in [0\,..\,\sigma)^{\leq \ell}$ and any $i \in [0\,..\,m']$, returns $\mathsf{prefix\text{-}rank}_W(i, X)$ in $\mathcal{O}(Q(m'))$ time.

Proof. The above reduction is essentially the same as the one presented in Proposition 4.20, except we only need Lemma 4.19(1a). □

6.3.3 Summary

Theorem 6.7. *Consider a data structure answering prefix rank queries (see Section 6.1) that, for any sequence of $m \geq 1$ binary strings of length $1 + \lfloor \log m \rfloor$, achieves the following complexities (where both the string at query time as well as input strings during construction are given in the packed representation):*

- *space usage $S(m)$,*
- *preprocessing time $P_t(m)$,*
- *preprocessing space $P_s(m)$,*
- *query time $Q(m)$.*

For every $T \in \{0,1\}^n$, there exists $m = \Theta(n/\log n)$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log n + P_t(m))$ time and using $\mathcal{O}(n/\log n + P_s(m))$ working space construct a data structure of size $\mathcal{O}(n/\log n + S(m))$ that answers pattern ranking queries, i.e., given the packed representation of any $P \in \{0,1\}^*$, returns $\mathrm{RangeBeg}(P,T)$ *(see Section 6.1) in $\mathcal{O}(\log\log n + Q(m) + |P|/\log n)$ time.*

Proof. Assume that $n > 3^{13} - 1$ (otherwise, the claim holds trivially).

We use the following definitions. Let $\sigma = 3$ and let $T' \in [0\,..\,\sigma)^{n+1}$ be a string defined so that $T'[1\,..\,n] = T$ and $T'[n+1] = 2$. Denote $n' = |T'|$. Note that $T'[n']$ does not occur in $T'[1\,..\,n')$ and it holds $2 \leq \sigma < (n')^{1/13}$. Let D denote the data structure resulting from applying Proposition 6.6 with alphabet size σ to the structure (answering prefix rank queries on binary strings) form the claim. Given any sequence $W[1\,..\,m']$ of $m' \geq 1$ equal-length strings over alphabet $[0\,..\,\sigma)$ of length $\ell' \leq (1 + \lfloor \log m' \rfloor)/\lceil \log \sigma \rceil$ as input (with all strings represented in the packed form), the structure D achieves the following complexities (where at query time it is given the packed representation of any $X \in [0\,..\,\sigma)^{\leq \ell'}$ and any $i \in [0\,..\,m']$, and returns $\mathsf{prefix\text{-}rank}_W(i,X)$):

- space usage $S'(m',\ell',\sigma) = \mathcal{O}(m' + S(m'))$,
- preprocessing time $P'_t(m',\ell',\sigma) = \mathcal{O}(m' + P_t(m'))$,
- preprocessing space $P'_s(m',\ell',\sigma) = \mathcal{O}(m' + P_s(m'))$,
- query time $Q'(m',\ell',\sigma) = \mathcal{O}(Q(m'))$,

where the complexities $S(m')$, $P_t(m')$, $P_s(m')$, and $Q(m')$ refer to the structure from the claim (answering prefix rank for binary strings).

Components The data structure answering pattern ranking queries consists of a single component: the data structure from Theorem 6.5 applied for text T' and with the structure D as the structure answering prefix rank queries. Note that we can use D, since it supports prefix rank queries for the combination of parameters required in Theorem 6.5, i.e., for sequences over alphabet $[0\,..\,\sigma)$, with the length ℓ of all strings satisfying $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ (where k is the length of the input sequence). To bound the space usage of this component, note that by Theorem 6.5 and the above discussion, there exists $m = \Theta(n'/\log_\sigma n') = \Theta(n/\log n)$ (recall that $\sigma = 3$) such that the structure needs $\mathcal{O}(n'/\log_\sigma n' + S'(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + S(m)) = \mathcal{O}(n/\log n + S(m))$ space.

Implementation of queries The pattern ranking queries are answered as follows. Let $P \in \{0,1\}^*$. Note that by definition of T', it follows that $\mathrm{RangeBeg}(P,T) = \mathrm{RangeBeg}(P,T')$. By Theorem 6.5 and the above discussion, computing $\mathrm{RangeBeg}(P,T')$ takes $\mathcal{O}(\log\log n' + Q'(m,\ell,\sigma) + |P|/\log_\sigma n') = \mathcal{O}(\log\log n + Q(m) + |P|/\log n)$ time.

Construction algorithm By Theorem 6.5 and the above discussion, construction of the above data structure answering pattern ranking queries on T' (and hence also on T) takes $\mathcal{O}(n'/\log_\sigma n' + P'_t(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + P_t(m)) = \mathcal{O}(n/\log n + P_t(m))$ time and uses $\mathcal{O}(n'/\log_\sigma n' + P'_s(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + P_s(m)) = \mathcal{O}(n/\log n + P_s(m))$ working space. □

7 Equivalence of Lex-Range Reporting and Prefix Range Reporting Queries

7.1 Problem Definitions

INDEXING FOR LEX-RANGE REPORTING QUERIES

Input: The packed representation of a string $T \in \{0,1\}^n$.
Output: A data structure that, given the packed representation of any $P_1, P_2 \in \{0,1\}^*$, returns the set $\mathrm{LexRange}(P_1, P_2, T)$ (Definition 2.4).

INDEXING FOR PREFIX RANGE REPORTING QUERIES

Input: A sequence $W[1\,..\,m]$ of $m \geq 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$, with all strings represented in the packed form.
Output: A data structure that, given the packed representation of any $X \in \{0,1\}^{\leq \ell}$ and any $b, e \in [0\,..\,m]$, returns the set

$$\{i \in (b\,..\,e] : X \text{ is a prefix of } W[i]\}.$$

7.2 Reducing Prefix Range Reporting to Lex-Range Reporting Queries

7.2.1 Problem Reduction

Lemma 7.1. *Let $W[1\,..\,m]$ be a sequence of $m \geq 1$ strings of length $\ell \geq 1$ over alphabet $\{0,1\}$. Let $s(i)$ (where $i \in [0\,..\,m]$) be as in Definition 4.7. Denote $T = \mathrm{SeqToString}_\ell(W)$ (Definition 4.7). Consider a string $X \in \{0,1\}^{\leq \ell}$ and let $b, e \in [0\,..\,m]$. Denote $P_1 = \overline{X} \cdot 4 \cdot s(b)$, $P_2 = \overline{X} \cdot 4 \cdot s(e)$, and $\beta = \ell + \lfloor \log m \rfloor + 2$. Let also $\mathcal{P} = \{i \in (b\,..\,e] : X \text{ is a prefix of } W[i]\}$ and $\mathcal{Q} = \mathrm{LexRange}(P_1, P_2, T)$. Then, it holds:*

1. $\mathcal{P} = \{\lceil j/\beta \rceil : j \in \mathcal{Q}\}$,
2. $|\mathcal{P}| = |\mathcal{Q}|$.

Proof. Denote $k = 1 + \lfloor \log m \rfloor$ and $\mathcal{Q}' = \{\lceil j/\beta \rceil : j \in \mathcal{Q}\}$.

First, we prove the inclusion $\mathcal{P} \subseteq \mathcal{Q}'$. Let $i \in \mathcal{P}$. Then, $i \in (b\,..\,e]$ and X is a prefix of $W[i]$. Denote $j = i \cdot \beta - (k + |X|)$. By Definition 4.7, the string $T[i \cdot \beta + 1\,..\,|T|]$ is preceded in T with $\overline{W[i]} \cdot 4 \cdot s(i-1)$. Since X is a prefix of $W[i]$, we thus obtain that $T[j\,..\,|T|]$ has $\overline{X} \cdot 4 \cdot s(i-1)$ as a prefix. The assumption $b < i \leq e$, or equivalently, $b \leq i - 1 < e$, implies that $s(b) \preceq s(i-1) \prec s(e)$. We thus obtain that $P_1 \preceq \overline{X} \cdot 4 \cdot s(i-1) \prec P_2$, and hence $P_1 \preceq T[j\,..\,|T|] \prec P_2$. We have thus proved that $j \in \mathrm{LexRange}(P_1, P_2, T)$. Since $\lceil j/\beta \rceil = i$, we thus obtain $i \in \mathcal{Q}'$.

We now prove the second inclusion. Let $q \in \mathcal{Q}'$. Then, there exists $j \in \mathrm{LexRange}(P_1, P_2, T)$ such that $\lceil j/\beta \rceil = q$. The assumption $j \in \mathrm{LexRange}(P_1, P_2, T)$ implies that $P_1 \preceq T[j\,..\,|T|] \prec P_2$. By definition of P_1 and P_2, this implies that $T[j\,..\,|T|]$ has the string $\overline{X} \cdot 4$ as a prefix. Consequently, by Lemma 4.10, there exists $i \in [1\,..\,m]$ such that X is a prefix of $W[i]$ and $j = i \cdot \beta - (k + |X|)$. By Definition 4.7, the suffix $T(i \cdot \beta - k\,..\,|T|]$ has the string $s(i-1)$ as a prefix. Putting together with the previous observation, we thus obtain that $T[j\,..\,|T|]$ has $\overline{X} \cdot 4 \cdot s(i-1)$ as a prefix. From the definition of P_1 and P_2, and the assumption $P_1 \preceq T[j\,..\,|T|] \prec P_2$, this implies that $s(b) \preceq s(i-1) \prec s(e)$, and hence $b \leq i - 1 < e$, or equivalently, $b < i \leq e$. Combining this with X being a prefix of $W[i]$ (see above), we obtain $i \in \mathcal{P}$. It remains to note that $j = i \cdot \beta - (k + |X|)$ implies that $q = \lceil j/\beta \rceil = i$. We have thus proved that $q \in \mathcal{P}$.

It remains to show that $|\mathcal{P}| = |\mathcal{Q}|$. Observe that since above we proved that $|\mathcal{P}| = |\mathcal{Q}'|$, it suffices to show that $|\mathcal{Q}| = |\mathcal{Q}'|$. To show this, first note that, by definition of $\mathcal{Q}'$, it immediately follows that $|\mathcal{Q}'| \leq |\mathcal{Q}|$. To show $|\mathcal{Q}| \leq |\mathcal{Q}'|$, consider any $j_1, j_2 \in Q$ such that $j_1 \neq j_2$. Above we observed that this implies that there exist i_1, i_2 such that $j_1 = i_1 \cdot \beta - (k + |X|)$ and $j_2 = i_2 \cdot \beta - (k + |X|)$. Since we assumed that $j_1 \neq j_2$, we thus must have $i_1 \neq i_2$. It remains to note that for $x \in \{1, 2\}$, it holds $\lceil j_x/\beta \rceil = \lceil (i_x \cdot \beta - (k + |X|))/\beta \rceil = i_x$. We have thus proved that the mapping $f : \mathcal{Q} \to \mathcal{Q}'$ defined by $f(j) = \lceil j/\beta \rceil$ is injective. Thus, $|\mathcal{Q}| \leq |\mathcal{Q}'|$, and hence we obtain $|\mathcal{Q}| = |\mathcal{Q}'| = |\mathcal{P}|$, i.e., the last claim. □

7.2.2 Alphabet Reduction for Lex-Range Reporting Queries

Lemma 7.2. *Let $\sigma \geq 2$ and $T, P_1, P_2 \in [0\,..\,\sigma)^*$. Then, it holds (see Definition 2.4)*

$$\mathrm{LexRange}(P_1', P_2', T') = \{(j-1)\delta + 1 : j \in \mathrm{LexRange}(P_1, P_2, T)\},$$

where $k = \lceil \log \sigma \rceil$, $\delta = 2k+3$, $P_1' = \mathrm{ebin}_k(P_1)$, $P_2' = \mathrm{ebin}_k(P_2)$, and $T' = \mathrm{ebin}_k(T)$ (Definition 4.3).

Proof. Denote $A = \{(j-1)\delta + 1 : j \in \mathrm{LexRange}(P_1, P_2, T)\}$.

First, we prove the inclusion $A \subseteq \mathrm{LexRange}(P_1', P_2', T')$. Let $j' \in A$. Then, there exists a position $j \in \mathrm{LexRange}(P_1, P_2, T)$ such that $j' = (j-1)\delta + 1$. By Definition 2.4, we have $P_1 \preceq T[j\,..\,|T|] \prec P_2$. By Definition 4.3, it holds $T'[j'\,..\,|T'|] = \bigodot_{t=j}^{|T|} \mathrm{ebin}_k(T[t]) = \mathrm{ebin}_k(T[j\,..\,|T|])$. By Lemma 4.4, $P_1 \preceq T[j\,..\,|T|]$ holds if and only if $\mathrm{ebin}_k(P_1) \preceq \mathrm{ebin}_k(T[j\,..\,|T|])$. Thus, we have $P_1' \preceq T'[j'\,..\,|T'|]$. Analogously, $T'[j'\,..\,|T'|] \prec P_2'$. Thus, we have $j' \in \mathrm{LexRange}(P_1', P_2', T')$.

We now prove the opposite inclusion. Let $j' \in \mathrm{LexRange}(P_1', P_2', T')$. This implies that $P_1' \preceq T'[j'\,..\,|T'|] \prec P_2'$. Note that both P_1' and P_2' have the string $\mathtt{1}^{k+1}$ as a prefix. Thus, $\mathtt{1}^{k+1}$ is also a prefix of $T'[j'\,..\,|T'|]$. By $\mathrm{Occ}(\mathtt{1}^{k+1}, T') = \{(t-1)\delta + 1 : t \in [1\,..\,|T|]\}$, it follows that there exists $j \in [1\,..\,|T|]$ such that $j' = (j-1)\delta + 1$. By definition of T', it follows that $T'[j'\,..\,|T'|] = \bigodot_{t=j}^{|T|} \mathrm{ebin}_k(T[t]) = \mathrm{ebin}_k(T[j\,..\,|T|])$. Observe that by Lemma 4.4, $P_1 \preceq T[j\,..\,|T|]$ holds if and only if $P_1' \preceq \mathrm{ebin}_k(T[j\,..\,|T|])$. By $\mathrm{ebin}_k(T[j\,..\,|T|]) = T'[j'\,..\,|T'|]$ and the assumption $P_1' \preceq T'[j'\,..\,|T'|]$, we thus obtain $P_1 \preceq T[j\,..\,|T|]$. Analogously, it holds $T[j\,..\,|T|] \prec P_2$. Thus, $j \in \mathrm{LexRange}(P_1, P_2, T)$, and hence we obtain $j' \in A$. □

Proposition 7.3. *Let $T \in [0\,..\,\sigma)^n$ be a nonempty string, where $2 \leq \sigma < n^{\mathcal{O}(1)}$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct the packed representation of a text $T_{\mathrm{bin}} \in \{\mathtt{0},\mathtt{1}\}^*$ satisfying $|T_{\mathrm{bin}}| = \Theta(n \log \sigma)$, integers α, γ, and δ, and a data structure that, given the packed representation of any $P_1, P_2 \in [0\,..\,\sigma)^{\leq m}$, in $\mathcal{O}(1 + m/\log_\sigma n)$ time returns the packed representation of $P_1', P_2' \in \{\mathtt{0},\mathtt{1}\}^*$ such that $|P_i'| = |P_i| \cdot (2\lceil \log \sigma \rceil + 3)$ (where $i \in \{1,2\}$) and*

$$\mathrm{LexRange}(P_1, P_2, T) = \left\{ \frac{j-\alpha}{\gamma} + \delta : j \in \mathrm{LexRange}(P_1', P_2', T_{\mathrm{bin}}) \right\}.$$

Proof. Let $k = \lceil \log \sigma \rceil$. To compute α, γ, δ, and T_{bin}, we proceed as follows:

1. In $\mathcal{O}(n/\log_\sigma n)$ time we construct the data structure from Proposition 4.5 for $u = n$. Given the packed representation of any $S \in [0\,..\,\sigma)^*$, we can then compute the packed representation of $\mathrm{ebin}_k(S)$ (Definition 4.3) in $\mathcal{O}(1 + |S|/\log_\sigma n)$ time.
2. Compute the packed representation of $T_{\mathrm{bin}} = \mathrm{ebin}_k(T)$ in $\mathcal{O}(n/\log_\sigma n)$ time.
3. In $\mathcal{O}(1)$ time set $\alpha = 1$, $\gamma = 2k+3$, and $\delta = 1$.

In total, the construction of the packed representation of T_{bin} and integers α, γ, and δ takes $\mathcal{O}(n/\log_\sigma n)$. By Definition 4.3, $|T_{\mathrm{bin}}| = n \cdot (2k+3) = \Theta(n \log \sigma)$.

Next, we address the construction of the data structure from the claim. It consists of a single component: the data structure from Proposition 4.5 for $u = n$. It uses $\mathcal{O}(n/\log_\sigma n)$ space.

The queries are implemented as follows. Given the packed representation of $P_1, P_2 \in [0\,..\,\sigma)^{\leq m}$, we compute the packed representation of P_1' and P_2' defined as $P_i' = \mathrm{ebin}_k(P_i)$. Using the structure from Proposition 4.5, this takes $\mathcal{O}(1 + m/\log_\sigma n)$ time. By Definition 4.3, it holds $|P_i'| = |P_i| \cdot (2k+3)$ for $i \in \{1,2\}$, and by Lemma 7.2, $\mathrm{LexRange}(P_1, P_2, T) = \{(j-\alpha)/\gamma + \delta : j \in \mathrm{LexRange}(P_1', P_2', T_{\mathrm{bin}})\}$.

By Proposition 4.5, construction of the data structure takes $\mathcal{O}(n/\log_\sigma n)$ time. □

7.2.3 Summary

Theorem 7.4. *Consider a data structure answering lex-range reporting queries (see Section 7.1) that, for any text $T \in \{\mathtt{0},\mathtt{1}\}^n$, achieves the following complexities (where in the preprocessing we assume that we are given as input the packed representation of T, and at query time we are given a packed representation of $P_1, P_2 \in \{\mathtt{0},\mathtt{1}\}^{\leq k}$, and we denote $p = |\mathrm{LexRange}(P_1, P_2, T)|$):*

- *space usage $S(n)$,*
- *preprocessing time $P_t(n)$,*

- *preprocessing space* $P_s(n)$,
- *query time* $Q_{\text{base}}(n,k) + p \cdot Q_{\text{occ}}(n,k)$.

For every sequence $W[1\,..\,m]$ *of* $m \geq 1$ *binary strings of length* $\ell = 1 + \lfloor \log m \rfloor$*, there exists* $n = \mathcal{O}(m \log m)$ *and* $k = \mathcal{O}(\log m)$ *such that, given the sequence* W*, with all strings represented in the packed form, we can in* $\mathcal{O}(m + P_t(n))$ *time and using* $\mathcal{O}(m + P_s(n))$ *working space construct a data structure of size* $\mathcal{O}(m + S(n))$ *that, given the packed representation of any* $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$ *and any pair of integers* $b, e \in [0\,..\,m]$*, computes the set* $\mathcal{P} = \{i \in (b\,..\,e] : X \text{ is a prefix of } W[i]\}$ *in* $\mathcal{O}(Q_{\text{base}}(n,k) + p \cdot Q_{\text{occ}}(n,k))$ *time, where* $p = |\mathcal{P}|$.

Proof. We use the following definitions. Let $T_{\text{aux}} = \text{SeqToString}_\ell(W)$ (Definition 4.7). Denote $n_{\text{aux}} = |T_{\text{aux}}| = m \cdot (2\ell + 1) = \Theta(m \log m)$. Let T_{bin} and α_{bin}, γ_{bin}, and δ_{bin} denote the text and the three integers obtained by applying Proposition 7.3 to text T_{aux}. Denote $n = |T_{\text{bin}}|$. Since T_{aux} is over alphabet $\{\mathtt{0},\ldots,\mathtt{4}\}$, by Proposition 7.3 we have $n = \Theta(|T_{\text{aux}}|) = \Theta(m \log m)$. Lastly, denote $k = 18\lfloor \log m \rfloor + 27 = \mathcal{O}(\log m)$.

Components The data structure consists of the following components:

1. The structure from Proposition 7.3 applied to text T_{aux}. Since T_{aux} is over alphabet $\{\mathtt{0},\ldots,\mathtt{4}\}$, the structure needs $\mathcal{O}(|T_{\text{aux}}|/\log|T_{\text{aux}}|) = \mathcal{O}(m)$ space.
2. The data structure from the claim (i.e., answering lex-range reporting queries) for the text T_{bin}. The structure needs $\mathcal{O}(S(n))$ space.
3. A lookup table L_{rev} defined as in Theorem 6.4. It needs $\mathcal{O}(\sqrt{n}) = \mathcal{O}(m)$ space.
4. A lookup table L_{map} defined as in Theorem 6.4. It also needs $\mathcal{O}(\sqrt{n}) = \mathcal{O}(m)$ space.
5. The integers α_{bin}, γ_{bin}, and δ_{bin}.

In total, the structure needs $\mathcal{O}(m + S(n))$ space.

Implementation of queries The queries are answered as follows. Assume that we are given the packed representation of some $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$ and two integers $b, e \in [0\,..\,m]$. The query algorithm proceeds as follows:

1. Using L_{rev} and L_{map}, in $\mathcal{O}(1)$ time we compute the packed representation of strings $P_1 = \overline{X} \cdot \mathtt{4} \cdot s(b)$ and $P_2 = \overline{X} \cdot \mathtt{4} \cdot s(e)$ (over alphabet $\{\mathtt{0},\ldots,\mathtt{4}\}$). Note that $|P_1| = |P_2| = |X| + 1 + \ell \leq 2\ell + 1 \leq 2\lfloor \log m \rfloor + 3$.
2. Using Proposition 7.3, in $\mathcal{O}(1 + |P_1|/\log n_{\text{aux}} + |P_2|/\log n_{\text{aux}}) = \mathcal{O}(1 + \ell/\log m) = \mathcal{O}(1)$ time, we compute the packed representation of strings $P_1', P_2' \in \{\mathtt{0},\mathtt{1}\}^*$ of length $|P_1'| = |P_2'| = 9 \cdot |P_1| \leq 18\lfloor \log m \rfloor + 27 \leq k$ such that
$$\text{LexRange}(P_1, P_2, T_{\text{aux}}) = \{(j - \alpha_{\text{bin}})/\gamma_{\text{bin}} + \delta_{\text{bin}} : j \in \text{LexRange}(P_1', P_2', T_{\text{bin}})\}.$$
3. Using the structure from the claim, we compute the set $\mathcal{Q}_{\text{bin}} = \text{LexRange}(P_1', P_2', T_{\text{bin}})$. Letting $p = |\mathcal{Q}_{\text{bin}}|$, by $|P_1'|, |P_2'| \leq k$, this takes $\mathcal{O}(Q_{\text{base}}(n,k) + p \cdot Q_{\text{occ}}(n,k))$ time.
4. In $\mathcal{O}(p)$ time we compute the set $\mathcal{Q}_{\text{aux}} = \{(j - \alpha_{\text{bin}})/\gamma_{\text{bin}} + \delta_{\text{bin}} : j \in \mathcal{Q}_{\text{bin}}\}$. Note that since each element of $\mathcal{Q}_{\text{bin}}$ yields a different element of $\mathcal{Q}_{\text{aux}}$, we have $p = |\mathcal{Q}_{\text{bin}}| = |\mathcal{Q}_{\text{aux}}|$. By the above equation, it holds $\mathcal{Q}_{\text{aux}} = \text{LexRange}(P_1, P_2, T_{\text{aux}})$.
5. In $\mathcal{O}(p)$ time we compute the set $\mathcal{P} = \{\lceil j/\beta \rceil : j \in \mathcal{Q}_{\text{aux}}\}$, where $\beta = \ell + \lfloor \log m \rfloor + 2 = 2\ell + 1$. By Lemma 7.1, it holds $\mathcal{P} = \{i \in (b\,..\,e] : X \text{ is a prefix of } W[i]\}$. Moreover, it holds $|\mathcal{P}| = |\mathcal{Q}_{\text{aux}}|$, which implies $|\mathcal{P}| = p$. We thus return $\mathcal{P}$ as the answer.

In total, the query takes $\mathcal{O}(Q_{\text{base}}(n,k) + p \cdot Q_{\text{occ}}(n,k))$ time.

Construction algorithm The components of the data structure are constructed as follows:

1. The first component of the structure is computed as follows:
   (a) In $\mathcal{O}(m)$ time we apply Proposition 4.12 to the sequence W to compute the packed representation of the string $T_{\text{aux}} = \text{SeqToString}_\ell(W)$ (Definition 4.7).
   (b) We apply Proposition 7.3 to T_{aux}. Since the string T_{aux} is over alphabet $\{\mathtt{0},\ldots,\mathtt{4}\}$, this takes $\mathcal{O}(|T_{\text{aux}}|/\log|T_{\text{aux}}|) = \mathcal{O}(m)$ time. Note that Proposition 7.3, in addition to the data structure, also returns integers α_{bin}, γ_{bin}, and δ_{bin}, and the packed representation of the string T_{bin} (defined above).
2. We apply the preprocessing from the claim to the string T_{bin}. This takes $\mathcal{O}(P_t(|T_{\text{bin}}|)) = \mathcal{O}(P_t(n))$ time and uses $\mathcal{O}(P_s(|T_{\text{bin}}|)) = \mathcal{O}(P_s(n))$ working space.

3. The lookup table L_{rev} is computed in $\mathcal{O}(m)$ time similarly as in the proof of Proposition 4.12.
4. Next, we compute the lookup table L_{map}. Similarly as above, we proceed as in Proposition 4.12, and spend $\mathcal{O}(m)$ time.
5. Lastly, we store integers α_{bin}, γ_{bin}, and δ_{bin}, which were already computed above.

In total, the construction takes $\mathcal{O}(m + P_t(n))$ time and uses $\mathcal{O}(m + P_s(n))$ working space. □

7.3 Reducing Lex-Range Reporting to Prefix Range Reporting Queries

7.3.1 Problem Reduction

7.3.1.1 Preliminaries

Definition 7.5 (τ-periodic and τ-nonperiodic patterns)**.** Let $P \in \Sigma^m$ and $\tau \geq 1$. We say that P is *τ-periodic* if it holds $m \geq 3\tau - 1$ and $\mathrm{per}(P[1\,..\,3\tau-1]) \leq \frac{1}{3}\tau$. Otherwise, it is called *τ-nonperiodic*.

Definition 7.6 (String-to-integer mapping)**.** Let $m > 0$, $\sigma > 1$. For every $X \in [0\,..\,\sigma)^{\leq m}$, by $\mathrm{int}(m, \sigma, X)$ we denote an integer constructed by appending $2m - 2|X|$ zeros to X and $|X|$ cs (where $c = \sigma - 1$) to X, and then interpreting the resulting string as a base-σ representation of a number in $[0\,..\,\sigma^{2m})$.

Lemma 7.7 ([KK24])**.** *Let $m > 0$ and $\sigma > 1$. For all $X, X' \in [0\,..\,\sigma)^{\leq m}$, $X \prec X'$ implies $\mathrm{int}(m, \sigma, X) < \mathrm{int}(m, \sigma, X')$. In particular, $X \neq X'$ implies $\mathrm{int}(m, \sigma, X) \neq \mathrm{int}(m, \sigma, X')$.*

Proposition 7.8. *Let $\sigma \geq 2$ and $\tau \geq 1$. Given the values of σ and τ, we can in $\mathcal{O}(\sigma^{3\tau} \cdot \tau^2)$ time construct a data structure that, given the packed representation of any $P \in [0\,..\,\sigma)^{3\tau-1}$, in $\mathcal{O}(1)$ time determines if P is τ-periodic (Definition 7.5).*

Proof. The data structure consists of a lookup table L_{per}. Its definition, space requirement, usage, and construction is as described in Section 5.1.1 of [KK23a], except to reduce its size, we index the lookup table using simply the packed representation of P (and hence reduce its size to $\mathcal{O}(\sigma^{3\tau})$). □

7.3.1.2 The Short Patterns

Preliminaries

Definition 7.9 (Indicator vector of a set)**.** For any $\ell \in \mathbb{Z}_{>0}$ and any set $A \subseteq [1\,..\,\ell]$, by $\mathrm{IndVec}_\ell(A)$ we denote a bitvector $V[1\,..\,\ell]$ defined so that for $i \in [1\,..\,\ell]$, $V[i] = 1$ holds if and only if $i \in A$.

Basic Navigational Primitives

Proposition 7.10. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that, given the packed representation of any $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, returns the pair $(\mathrm{RangeBeg}(X, T), \mathrm{RangeEnd}(X, T))$ (see Definition 2.1) in $\mathcal{O}(1)$ time.*

Proof. The data structure consists of the lookup table L_{range}. Its definition, space requirement, usage, and construction is as described in Section 5.1 of [KK23a]. □

Algorithms

Proposition 7.11. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that given the packed representation of any $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, returns $\mathcal{J} := \mathrm{Occ}(X, T)$ (Definition 2.1) in $\mathcal{O}(1 + |\mathcal{J}|)$ time.*

Proof. We use the following definitions. Let $\ell = 3\tau - 1$. For any $i \in [1\,..\,n]$, denote $B(i) := T[i\,..\,\min(n + 1, i + 2\ell - 1))$. Denote $m = \lceil \frac{n}{\ell} \rceil$, and let $(c_j)_{j \in [1\,..\,m]}$ be a sequence containing a permutation of the set

$\{1+(i-1)\ell : i \in [1\,..\,m]\}$ such that, for every $j, j' \in [1\,..\,m]$, $j < j'$ implies that it holds $B(c_j) \prec B(c_{j'})$, or $B(c_j) = B(c_{j'})$ and $c_j < c_{j'}$. Let $A_{\mathrm{blk}}[1\,..\,m]$ be an array defined by $A_{\mathrm{blk}}[i] = c_i$. Let

$$\mathcal{F} := \{1\} \cup \{i \in [2\,..\,m] : B(c_i) \neq B(c_{i-1})\}.$$

For every $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, we define (see Definition 7.9)

$$\begin{aligned}\mathrm{blocks}(X) &:= \{i \in \mathcal{F} : \mathrm{Occ}(X, B(c_i)) \cap [1\,..\,\ell] \neq \emptyset\},\\ \mathrm{blocks}'(X) &:= \{(i, \mathrm{IndVec}_\ell(\mathrm{Occ}(X, B(c_i)) \cap [1\,..\,\ell]) : i \in \mathrm{blocks}(X)\}.\end{aligned}$$

Let L_{blocks} be a mapping such that, for every $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, L_{blocks} maps X to the set $\mathrm{blocks}'(X)$ defined above. Finally, let L_{bit} be a mapping such that, for every $x \in [1\,..\,2^\ell)$, L_{bit} maps x to the position of the least significant 1-bit in the binary encoding of x.

Components The data structure consists of the following components:

1. The array $A_{\mathrm{blk}}[1\,..\,m]$ in plain form using $\mathcal{O}(m) = \mathcal{O}(\frac{n}{\ell}) = \mathcal{O}(n/\log_\sigma n)$ space.
2. The lookup table L_{blocks}, where for every $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, $L_{\mathrm{blocks}}[X]$ stores the pointer to the head of a linked list containing all elements of the set $\mathrm{blocks}'(X)$. Each element $(i, V) \in \mathrm{blocks}'(X)$ is stored as a pair of integers, with the bitvector V encoded as a number in the range $[0\,..\,2^\ell)$. When accessing $L_{\mathrm{blocks}}[X]$, the string $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$ is represented as a number $\mathrm{int}(3\tau, \sigma, X) \in [0\,..\,\sigma^{6\tau})$ (Definition 7.6). Thus, the range of indexes for L_{blocks} is bounded by $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(\sqrt{n}) = \mathcal{O}(n/\log_\sigma n)$. To bound the total length of all lists, i.e., $\sum_{X \in [0\,..\,\sigma)^{\leq 3\tau-1}} |\mathrm{blocks}(X)|$, we proceed as follows:
   - First, observe that by definition of the sequence $(c_j)_{j \in [1\,..\,m]}$ and the set $\mathcal{F}$, for every $i, i' \in \mathcal{F}$, $i \neq i'$ implies that $B(c_i) \neq B(c_{i'})$. Since for every $j \in [1\,..\,n]$, $B(j) \in [0\,..\,\sigma)^{\leq 2\ell-1}$, it follows that $|\mathcal{F}| = \mathcal{O}(\sigma^{2\ell}) = \mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(\sqrt{n})$.
   - Second, note that $|\mathrm{blocks}(\varepsilon)| = |\mathcal{F}| = \mathcal{O}(\sqrt{n})$.
   - Finally, observe that if for some $i \in \mathcal{F}$ and $X \in [0\,..\,\sigma)^{\leq \ell} \setminus \{\varepsilon\}$, it holds $i \in \mathrm{blocks}(X)$, then for some $\delta \in [1\,..\,\ell]$ satisfying $\delta + |X| \leq |B(c_i)| + 1$, it holds $B(c_i)[\delta\,..\,\delta + |X|) = X$. This implies that i can occur in $\mathrm{blocks}(X)$ for at most ℓ^2 distinct nonempty strings X. Consequently, $\sum_{X \in [0\,..\,\sigma)^{\leq 3\tau-1} \setminus \{\varepsilon\}} |\mathrm{blocks}(X)| \leq |\mathcal{F}| \cdot \ell^2$.

   Putting everything together, we thus obtain that

$$\sum_{X \in [0\,..\,\sigma)^{\leq 3\tau-1}} |\mathrm{blocks}(X)| = \mathcal{O}(|\mathcal{F}| \cdot \ell^2) = \mathcal{O}(\sqrt{n} \log^2 n) = \mathcal{O}(n/\log_\sigma n).$$

3. The lookup table L_{bit} using $\mathcal{O}(2^\ell) = \mathcal{O}(\sigma^\ell) = \mathcal{O}(\sqrt{n}) = \mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n)$ space.

Implementation of queries Given the packed representation of any $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, we compute $\mathrm{Occ}(X, T)$ as follows:

1. Initialize the output set $\mathcal{R} := \emptyset$.
2. Using the lookup table L_{blocks}, in $\mathcal{O}(1)$ time obtain the pointer to head of a linked list containing all the elements of the set $\mathrm{blocks}'(X)$. If the list is empty, then observe that, by definition of the sequence $(c_i)_{i \in [1\,..\,m]}$ and the sets $\mathrm{blocks}(X)$ and $\mathrm{blocks}'(X)$, we then have $\mathrm{Occ}(X, T) = \emptyset$. Thus, in this case we skip the rest of this step. Let us thus assume that $\mathrm{blocks}'(X) \neq \emptyset$. In $\mathcal{O}(1 + |\mathrm{blocks}'(X)|)$ time, we then iterate over all elements of $\mathrm{blocks}'(X)$. For every $(i, V) \in \mathrm{blocks}'(X)$, we proceed as follows:
   (a) Compute the set containing the positions of 1-bits in V, i.e., the set $\Delta_i := \mathrm{Occ}(X, B(c_i)) \cap [1\,..\,\ell]$. Note that $\Delta_i \neq \emptyset$. With the help of L_{bit}, computing Δ_i takes $\mathcal{O}(|\Delta_i|)$ time.
   (b) By scanning the array A_{blk} from index i, compute

$$i' = \min\{j \in (i\,..\,m] : B(c_j) \neq B(c_i)\} \cup \{m+1\}.$$

   Denote $\mathcal{J}_i = [i\,..\,i')$ and note that $\mathcal{J}_i \neq \emptyset$. Denote also $\mathcal{C}_i = \{c_j : j \in [i\,..\,i')\}$. The computation of i' takes $\mathcal{O}(|\mathcal{J}_i|) = \mathcal{O}(|\mathcal{C}_i|)$ time, since given any $p \in [1\,..\,n]$, and the packed representation of T, we can in $\mathcal{O}(1)$ time obtain the packed representation of $B(p)$.

(c) In $\mathcal{O}(|\mathcal{C}_i| \cdot |\Delta_i|)$ time add the set

$$\mathcal{P}_i := \{A_{\text{blk}}[t] + \delta - 1 : (t, \delta) \in \mathcal{J}_i \times \Delta_i\} = \{c + \delta - 1 : (c, \delta) \in \mathcal{C}_i \times \Delta_i\}$$

to $\mathcal{R}$. Note that since $\Delta_i \subseteq [1\,..\,\ell]$, and every position $c \in \mathcal{C}_i$ is of the form $c = 1 + t \cdot \ell$, where $t \in \mathbb{Z}_{\geq 0}$, it holds $|\mathcal{P}_i| = |\mathcal{C}_i| \cdot |\Delta_i|$,

In total, we spend $\mathcal{O}(|\Delta_i| + |\mathcal{C}_i| + |\mathcal{C}_i| \cdot |\Delta_i|) = \mathcal{O}(|\mathcal{C}_i| \cdot |\Delta_i|) = \mathcal{O}(|\mathcal{P}_i|)$ time. Over all $(i, V) \in \text{blocks}'(X)$, we spend $\mathcal{O}(\sum_{i \in \text{blocks}(X)} |\mathcal{P}_i|)$ time (recall that $|\text{blocks}(X)| = |\text{blocks}'(X)|$, and for every $(i, V) \in \text{blocks}'(X)$, it holds $i \in \text{blocks}(X)$). By definition of the sequence $(c_j)_{j \in [1\,..\,m]}$ and sets $\text{blocks}(X)$ and $\text{blocks}'(X)$, the set $\text{Occ}(X, T)$ is a disjoint union

$$\text{Occ}(X, T) = \bigcup_{i \in \text{blocks}(X)} \mathcal{P}_i.$$

Thus, at the end of the algorithm, we have $\mathcal{R} = \text{Occ}(X, T)$, and the total time of this step is $\mathcal{O}(\sum_{i \in \text{blocks}(X)} |\mathcal{P}_i|) = \mathcal{O}(|\text{Occ}(X, T)|)$.

3. We return the set $\mathcal{R} = \text{Occ}(X, T)$ as the answer in $\mathcal{O}(1 + |\text{Occ}(X, T)|)$ time.

In total, the query takes $\mathcal{O}(1 + |\text{Occ}(X, T)|)$ time.

Construction algorithm The components of the structure are constructed as follows:

1. The construction of the array $A_{\text{blk}}[1\,..\,m]$ proceeds as follows:
   (a) In $\mathcal{O}(\frac{n}{\ell}) = \mathcal{O}(n/\log_\sigma n)$ time construct the array $A_{\text{sort}}[1\,..\,m]$ such that, for every $i \in [1\,..\,m]$, $A_{\text{sort}}[i] = (\text{int}(2\ell, \sigma, B(1 + (i-1)\ell)), 1 + (i-1)\ell)$ (see Definition 7.6).
   (b) Sort the array $A_{\text{sort}}[1\,..\,m]$ lexicographically. By Definition 7.6, the first coordinate is in the range $[0\,..\,\sigma^{4\ell}) \subseteq [0\,..\,\sigma^{12\tau}) \subseteq [0\,..\,n^{12\mu}) \subseteq [0\,..\,n)$, and the second coordinate is in the range $[1\,..\,n]$. We use a 4-round radix sort to achieve $\mathcal{O}(m + \sqrt{n}) = \mathcal{O}(n/\log_\sigma n)$ sorting time. By Lemma 7.7, the resulting array contains the sequence $(c_j)_{j \in [1\,..\,m]}$ on the second coordinate. With one more scan, in $\mathcal{O}(m) = \mathcal{O}(n/\log_\sigma n)$ time we construct $A_{\text{blk}}[1\,..\,m]$.

   In total, constructing $A_{\text{blk}}[1\,..\,m]$ takes $\mathcal{O}(n/\log_\sigma n)$ time.
2. The construction of the second component proceeds as follows:
   (a) First, we compute an array $A_{\mathcal{F}}[1\,..\,m]$ such that for every $i \in [1\,..\,m]$, $A_{\mathcal{F}}[i] = 1$ holds if and only if $i \in \mathcal{F}$. With the help of the array $A_{\text{blk}}[1\,..\,m]$ (computed above), and the packed representation of T, computing $A_{\mathcal{F}}[1\,..\,m]$ takes $\mathcal{O}(m)$ time.
   (b) Next, we compute a linked list containing the elements of the set $\text{blocks}'(\varepsilon)$. To this end, it suffices to iterate over $A_{\mathcal{F}}$ and add the pair $(i, 1^\ell)$ (where 1^ℓ is represented as integer $2^\ell - 1$) to $\text{blocks}'(\varepsilon)$ for every $i \in [1\,..\,m]$ such that $A_{\mathcal{F}}[i] = 1$. Lastly, we store the pointer to the head of the list in $L_{\text{blocks}}[\varepsilon]$. This takes $\mathcal{O}(m)$ time.
   (c) To compute lists containing sets $\text{blocks}'(X)$ for all $X \in [0\,..\,\sigma)^{\leq 3\tau - 1} \setminus \{\varepsilon\}$, we first initialize $L_{\text{blocks}}[X]$ for all such X to empty lists. For every $i \in [1\,..\,m]$ such that $A_{\mathcal{F}}[i] = 1$, we then perform the following steps:
      i. Denote $B = B(c_i) = B(A_{\text{blk}}[i])$. We compute the set

      $$\mathcal{S}_i := \{X \in [0\,..\,\sigma)^{\leq \ell} \setminus \{\varepsilon\} : \text{Occ}(X, B) \cap [1\,..\,\ell] \neq \emptyset\},$$

      with each of the strings X in the set represented as $\text{int}(\ell, \sigma, X)$ (Definition 7.6). To this end, first create a multiset containing all substrings $B[s\,..\,s+t)$, where $1 \leq s < s + t \leq |B| + 1$, $t \leq \ell$, and $s \leq \ell$, and then remove the duplicates. This is easily implemented in $\mathcal{O}(\ell^3)$ time. Note that $|\mathcal{S}_i| \leq \ell^2$.
      ii. For every $X \in \mathcal{S}_i$, compute the bitvector $V = \text{IndVec}_\ell(\text{Occ}(X, B) \cap [1\,..\,\ell])$ (Definition 7.9), with V represented as an integer in $[0\,..\,2^\ell)$. Then, add the pair (X, V) to the list whose head is currently stored in $L_{\text{blocks}}[X]$. Using a naive algorithm, the computation of V takes $\mathcal{O}(\ell^2)$. Over all $X \in \mathcal{S}_i$, we thus spend $\mathcal{O}(\ell^4)$ time.

   In total, the computation of $\text{blocks}'(X)$ for all $X \in [0\,..\,\sigma)^{\leq 3\tau - 1} \setminus \{\varepsilon\}$ takes $\mathcal{O}(m + |\mathcal{F}| \cdot \ell^4) = \mathcal{O}(m + \sqrt{n} \log^4 n)$ time.

In total, the construction of the second component takes $\mathcal{O}(m + \sqrt{n}\log^4 n) = \mathcal{O}(n/\log_\sigma n)$ time.

3. Computing $L_{\mathrm{bit}}[x]$ for any $x \in [1\,..\,2^\ell)$ takes $\mathcal{O}(\ell)$ time. Thus, over all $x \in [1\,..\,2^\ell)$, we spend $\mathcal{O}(2^\ell \cdot \ell) = \mathcal{O}(\sigma^\ell \cdot \ell) = \mathcal{O}(\sqrt{n}\log n) = \mathcal{O}(n/\log_\sigma n)$ time.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n)$ time. □

Proposition 7.12. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \le \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \ge 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that answers the following queries:*

1. *Given the packed representation of any $X_1, X_2 \in [0\,..\,\sigma)^{\le 3\tau-1}$, return $\mathcal{J} := \mathrm{LexRange}(X_1, X_2, T)$ (see Definition 2.4) in $\mathcal{O}(1 + |\mathcal{J}|)$ time.*
2. *Given the packed representation of any $X \in [0\,..\,\sigma)^{\le 3\tau-1}$, return $\mathcal{J} := \mathrm{LexRange}(X, Xc^\infty, T)$ (where $c = \sigma - 1$) in $\mathcal{O}(1 + |\mathcal{J}|)$ time.*
3. *Given the packed representation of any $X_1, X_2 \in [0\,..\,\sigma)^{\le 3\tau-1}$, return $\mathcal{J} := \mathrm{LexRange}(X_1c^\infty, X_2, T)$ (where $c = \sigma - 1$) in $\mathcal{O}(1 + |\mathcal{J}|)$ time.*

Proof. We use the following definitions. Let $B_{3\tau-1}[1\,..\,n]$ be a bitvector defined so that for every $i \in [1\,..\,n]$, $B_{3\tau-1}[i] = \mathtt{1}$ holds if and only if $i = n$, or $i < n$ and $\mathrm{LCE}_T(\mathrm{SA}_T[i], \mathrm{SA}_T[i{+}1]) < 3\tau{-}1$. Let $k = \mathsf{rank}_{B_{3\tau-1}}(n, \mathtt{1})$. Let $A_{\mathrm{str}}[1\,..\,k]$ be an array defined such that, for every $i \in [1\,..\,k]$, it holds $A_{\mathrm{str}}[i] = T[\mathrm{SA}_T[i']\,..\,\min(n + 1, \mathrm{SA}_T[i'] + 3\tau - 1))$, where $i' = \mathsf{select}_{B_{3\tau-1}}(i, \mathtt{1})$. Observe that $k = \mathcal{O}(\sigma^{3\tau-1}) = \mathcal{O}(\sqrt{n})$.

Components The data structure consists of the following components:

1. The data structure from Proposition 7.10. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
2. The bitvector $B_{3\tau-1}[1\,..\,n]$ augmented with the support for $\mathcal{O}(1)$-time rank and select queries using Theorem 2.7. The bitvector together with the augmentation of Theorem 2.7 needs $\mathcal{O}(n/\log n) = \mathcal{O}(n/\log_\sigma n)$ space.
3. The array $A_{\mathrm{str}}[1\,..\,k]$ stored in plain form using $\mathcal{O}(k) = \mathcal{O}(\sqrt{n}) = \mathcal{O}(n/\log_\sigma n)$ space.
4. The structure from Proposition 7.11 for text T. It needs $\mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n)$ space.

Implementation of queries The queries are answered as follows:

1. Let $X_1, X_2 \in [0\,..\,\sigma)^{\le 3\tau-1}$. Given the packed representation of strings X_1 and X_2, we compute the set $\mathrm{LexRange}(X_1, X_2, T)$ as follows:
   (a) Using Proposition 7.10, in $\mathcal{O}(1)$ time compute $b_k = \mathrm{RangeBeg}(X_k, T)$ for $k \in \{1, 2\}$. Observe that then
   $$\mathrm{LexRange}(X_1, X_2, T) = \{\mathrm{SA}_T[i]\}_{i \in (b_1\,..\,b_2]}$$
   (see Remark 2.5). If $b_1 \ge b_2$, then $\mathrm{LexRange}(X_1, X_2, T) = \emptyset$. Let us thus assume that $b_1 < b_2$. Note that, by definition of $B_{3\tau-1}$, if $b_k > 0$, then $B_{3\tau-1}[b_k] = \mathtt{1}$ holds for $k \in \{1, 2\}$.
   (b) If $b_1 = 0$, then we set $i_1 = 0$. Otherwise, using Theorem 2.7, in $\mathcal{O}(1)$ time we compute $i_1 = \mathsf{rank}_{B_{3\tau-1}}(b_1, \mathtt{1})$. Analogously we compute i_2. Observe that we then have
   $$\mathrm{LexRange}(X_1, X_2, T) = \bigcup_{i \in (i_1\,..\,i_2]} \mathrm{Occ}(A_{\mathrm{str}}[i], T).$$
   (c) Initialize the set $\mathcal{P} := \emptyset$.
   (d) For every $i \in (i_1\,..\,i_2]$, using Proposition 7.11, compute the set $\mathcal{J}_i := \mathrm{Occ}(A_{\mathrm{str}}[i], T)$ in $\mathcal{O}(1 + |\mathcal{J}_i|)$ time, and add to $\mathcal{P}$. In total, this takes $\mathcal{O}((i_2{-}i_1){+}\sum_{i \in (i_1\,..\,i_2]} |\mathcal{J}_i|) = \mathcal{O}(1{+}|\mathrm{LexRange}(X_1, X_2, T)|)$ time, where we used that since for every $i \in (i_1\,..\,i_2]$, it holds $\mathcal{J}_i \neq \emptyset$, it follows that $i_2 - i_1 = \mathcal{O}(\sum_{i \in (i_1\,..\,i_2]} |\mathcal{J}_i|)$. Note also that the set of strings in $A_{\mathrm{str}}[1\,..\,k]$ is prefix-free (i.e., no string is a prefix of another), which implies that for every $i, j \in [1\,..\,k]$, $i \neq j$ implies that $\mathrm{Occ}(A_{\mathrm{str}}[i], T) \cap \mathrm{Occ}(A_{\mathrm{str}}[j], T) = \emptyset$, and hence $|\mathrm{LexRange}(X_1, X_2, T)| = \sum_{i \in (i_1\,..\,i_2]} |\mathcal{J}_i|$. We then return $\mathcal{P}$ as the answer.

   In total, we spend $\mathcal{O}(1 + |\mathrm{LexRange}(X_1, X_2, T)|)$ time.

2. Let us now consider $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, where $c = \sigma - 1$. Given the packed representation of X, we compute $\mathrm{LexRange}(X, Xc^\infty, T)$ by setting $X_1 = X$ and $X_2 = Xc^\infty$, and using the same algorithm as above, except to compute $b_2 = \mathrm{RangeBeg}(Xc^\infty, T)$, we observe that $\mathrm{RangeBeg}(Xc^\infty, T) = \mathrm{RangeEnd}(X, T)$. Thus, we can determine b_2 using Proposition 7.10 in $\mathcal{O}(1)$ time.
3. Let $X_1, X_2 \in [0\,..\,\sigma)^{\leq 3\tau-1}$. Given the packed representation of X_1 and X_2, we compute the set $\mathrm{LexRange}(X_1c^\infty, X_2, T)$ (where $c = \sigma - 1$) similarly as above, except to compute the value $b_1 = \mathrm{RangeBeg}(X_1c^\infty, T)$, we use the fact that $\mathrm{RangeBeg}(X_1c^\infty, T) = \mathrm{RangeEnd}(X_1, T)$. Thus, we can compute b_1 using Proposition 7.10 in $\mathcal{O}(1)$ time.

Construction algorithm The components of the data structure are constructed as follows:

1. The first component is constructed using Proposition 7.10 in $\mathcal{O}(n/\log_\sigma n)$ time.
2. To construct the second component, we first compute the bitvector $B_{3\tau-1}[1\,..\,n]$ as described in [KK23a, Proposition 5.3] in $\mathcal{O}(n/\log_\sigma n)$ time. We then augment $B_{3\tau-1}$ with support for $\mathcal{O}(1)$-time rank/select queries using Theorem 2.7 in $\mathcal{O}(n/\log n) = \mathcal{O}(n/\log_\sigma n)$ time.
3. The array $A_{\mathrm{str}}[1\,..\,k]$ (with all strings stored in the packed representation) is constructed as described in [KK23a, Proposition 5.3] in $\mathcal{O}(n/\log_\sigma n)$ time.
4. The last component is constructed using Proposition 7.11 in $\mathcal{O}(n/\log_\sigma n)$ time.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n)$ time. □

7.3.1.3 The Nonperiodic Patterns

Preliminaries

Definition 7.13 (Sequence of positions in lex-order)**.** Let $T \in \Sigma^n$, $Q \subseteq [1\,..\,n]$, and $q = |Q|$. By $\mathrm{LexSorted}(Q, T)$, we denote a sequence $(a_i)_{1\in[1\,..\,q]}$ containing all positions from Q such that for every $i, j \in [1\,..\,q]$, $i < j$ implies $T[a_i\,..\,n] \prec T[a_j\,..\,n]$.

Definition 7.14 (Sequence of positions in text-order)**.** Consider any finite set $Q \subseteq \mathbb{Z}$, and let $q = |Q|$. By $\mathrm{Sort}(Q)$, we denote a sequence $(a_i)_{1\in[1\,..\,q]}$ containing all positions from Q such that for every $i, j \in [1\,..\,q]$, $i < j$ implies $a_i < a_j$.

Definition 7.15 (Successor)**.** Consider any totally ordered set $\mathcal{U}$, and let $A \subseteq \mathcal{U}$ be a nonempty subset of $\mathcal{U}$. For every $x \in \mathcal{U}$ satisfying $x \leq \max A$, we denote $\mathrm{succ}_A(x) = \min\{x' \in A : x' \succeq x\}$.

Definition 7.16 (Distinguishing prefix of a suffix)**.** Let $T \in \Sigma^n$. Let $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$, and let S be a τ-synchronizing set of T. For every $j \in [1\,..\,n - 3\tau + 2] \setminus \mathsf{R}(\tau, T)$, we denote (see Definition 7.15)

$$\mathrm{DistPrefix}(j, \tau, T, \mathsf{S}) := T[j\,..\,\mathrm{succ}_{\mathsf{S}}(j) + 2\tau).$$

We then let

$$\mathcal{D}(\tau, T, \mathsf{S}) := \{\mathrm{DistPrefix}(j, \tau, T, \mathsf{S}) : j \in [1\,..\,n - 3\tau + 2] \setminus \mathsf{R}(\tau, T)\}.$$

Remark 7.17. Note that $\mathrm{succ}_{\mathsf{S}}(j)$ in Definition 7.16 is well-defined for every $j \in [1\,..\,n - 3\tau + 2] \setminus \mathsf{R}(\tau, T)$, because by Definition 2.17(2), for such j we have $[j\,..\,j + \tau) \cap \mathsf{S} \neq \emptyset$.

Lemma 7.18 ([KK24])**.** *Let $T \in \Sigma^n$. Let $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$ and let S be a τ-synchronizing set of T. Then:*

1. *It holds $\mathcal{D}(\tau, T, \mathsf{S}) \subseteq [0\,..\,\sigma)^{\leq 3\tau-1}$.*
2. *$\mathcal{D}(\tau, T, \mathsf{S})$ is prefix-free, i.e., for $D, D' \in \mathcal{D}(\tau, T, \mathsf{S})$, $D \neq D'$ implies that D is not a prefix of D'.*

Lemma 7.19. *Let $T \in \Sigma^n$. Let $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$ and let S be a τ-synchronizing set of T. Let $P \in \Sigma^m$ be a τ-nonperiodic pattern (Definition 7.5) such that $m \geq 3\tau - 1$ and $\mathrm{Occ}(P[1\,..\,3\tau - 1], T) \neq \emptyset$. Then, there exists a unique $D \in \mathcal{D}(\tau, T, \mathsf{S})$ (Definition 7.16) that is a prefix of P.*

Proof. Denote $P' = P[1\,..\,3\tau - 1]$. Consider any $j \in \mathrm{Occ}(P', T)$ (such position exists by the assumption $\mathrm{Occ}(P', T) \neq \emptyset$). Since P' is τ-nonperiodic, it follows by $|P'| = 3\tau - 1$ and Definition 7.5 that $j \in [1\,..\,n - 3\tau + 2]$ and $\mathrm{per}(T[j\,..\,j + 3\tau - 1)) = \mathrm{per}(P') > \frac{1}{3}\tau$. By Definition 2.17, we thus have $j \in [1\,..\,n - 3\tau + 2] \setminus \mathsf{R}(\tau, T)$

and $[j\,..\,j+\tau) \cap \mathsf{S} \neq \emptyset$. Consequently, letting $j' = \mathrm{succ}_{\mathsf{S}}(j)$ (Definition 7.15), it holds $j' - j < \tau$. Therefore, letting $D = T[j\,..\,j'+2\tau)$, it holds $|D| = j' - j + 2\tau \leq 3\tau - 1 = |P'|$, and hence D is a prefix of P' (and thus also P). On the other hand, by Definition 7.16, we have $D \in \mathcal{D}(\tau, T, \mathsf{S})$. It remains to observe that since $\mathcal{D}(\tau, T, \mathsf{S})$ is prefix-free (Lemma 7.18(2)), no other string from $\mathcal{D}(\tau, T, \mathsf{S})$ can be a prefix of P. □

Definition 7.20 (Distinguishing prefix of a pattern)**.** Let $T \in \Sigma^n$. Let $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$, and let S be a τ-synchronizing set of T. For every τ-nonperiodic pattern $P \in \Sigma^m$ satisfying $m \geq 3\tau - 1$ and $\mathrm{Occ}(P[1\,..\,3\tau - 1], T) \neq \emptyset$, by $\mathrm{DistPrefix}(P, \tau, T, \mathsf{S})$ we denote the unique string $D \in \mathcal{D}(\tau, T, \mathsf{S})$ (Definition 7.16) that is a prefix of P (such D exists by Lemma 7.19).

Basic Navigational Primitives

Proposition 7.21 ([KK24, Proposition 5.13])**.** *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. Let S be a τ-synchronizing set of T satisfying $|\mathsf{S}| = \mathcal{O}(\frac{n}{\tau})$. Denote $(s_i)_{i \in [1\,..\,n']} = \mathrm{LexSorted}(\mathsf{S}, T)$ (Definition 7.13). Given the set S and the packed representation of the text T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure, denoted $\mathrm{NavNonperiodic}(\mathsf{S}, \tau, T)$, that supports the following queries:*

1. *Given the packed representation of any string $X \in [0\,..\,\sigma)^{\leq 3\tau - 1}$, in $\mathcal{O}(1)$ time return the packed representation of string $\overline{X}$ (see Section 2).*
2. *Let $P \in \Sigma^m$ be a τ-nonperiodic pattern satisfying $m \geq 3\tau - 1$ and $\mathrm{Occ}(P[1\,..\,3\tau - 1], T) \neq \emptyset$. Denote $D = \mathrm{DistPrefix}(P, \tau, T, \mathsf{S})$ (Definition 7.20), $\delta_{\mathrm{text}} = |D| - 2\tau$, and $P' = P(\delta_{\mathrm{text}}\,..\,m]$.*
   (a) *Given the packed representation of P, in $\mathcal{O}(1)$ time compute the packed representation of D.*
   (b) *Given the packed representation of P, in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time compute a pair of integers (b, e) such that:*
   - *$b = |\{i \in [1\,..\,n'] : T[s_i\,..\,n] \prec P'\}|$ and*
   - *$e = |\{i \in [1\,..\,n'] : T[s_i\,..\,n] \prec P'c^\infty\}|$, where $c = \sigma - 1$.*

Proof. Note that in [KK24, Proposition 5.13], we assumed that $\mathrm{Occ}(P, T) \neq \emptyset$. It is easy to check that this is not in fact required, and the structure works even if only $\mathrm{Occ}(P[1\,..\,3\tau - 1], T) \neq \emptyset$ holds. Note also that in the original claim, the value e is defined so that it holds $e - b = |\{i \in [1\,..\,n'] : P' \text{ is a prefix of } T[s_i\,..\,n]\}|$. Here we use a slightly different but equivalent definition. □

Theorem 7.22 ([KK19, Theorem 4.3])**.** *Given the packed representation of text $T \in [0\,..\,\sigma)^n$ and its τ-synchronizing set S of size $|\mathsf{S}| = \mathcal{O}(\frac{n}{\tau})$ for $\tau = \mathcal{O}(\log_\sigma n)$, we can compute the sequence $\mathrm{LexSorted}(\mathsf{S}, T)$ (Definition 7.13) in $\mathcal{O}(\frac{n}{\tau})$ time.*

Combinatorial Properties

Lemma 7.23. *Let $T \in \Sigma^n$ and $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$. Let S be a τ-synchronizing set of T. Denote $(s_i)_{i \in [1\,..\,n']} = \mathrm{LexSorted}(\mathsf{S}, T)$ (Definition 7.13). Let $D \in \mathcal{D}(\tau, T, \mathsf{S})$ (Definition 7.16) and let $P_1, P_2 \in \Sigma^+$ be τ-nonperiodic patterns (Definition 7.5) both having D as a prefix. Denote $\delta_{\mathrm{text}} = |D| - 2\tau$. For $k \in \{1, 2\}$, let $P'_k = P_k(\delta_{\mathrm{text}}\,..\,|P_k|]$, and $b_k = |\{i \in [1\,..\,n'] : T[s_i\,..\,n] \prec P'_k\}|$. Then, it holds (see Definition 2.4)*

$$\mathrm{LexRange}(P_1, P_2, T) = \{s_i - \delta_{\mathrm{text}} : i \in (b_1\,..\,b_2] \text{ and } s_i - \delta_{\mathrm{text}} \in \mathrm{Occ}(D, T)\}.$$

Proof. Denote $A = \{s_i - \delta_{\mathrm{text}} : i \in (b_1\,..\,b_2] \text{ and } s_i - \delta_{\mathrm{text}} \in \mathrm{Occ}(D, T)\}$. Observe that since D is a prefix of P_1 and P_2, for every $j \in \mathrm{LexRange}(P_1, P_2, T)$, it holds $j \in \mathrm{Occ}(D, T)$. Note also that, by definition of b_k for $k \in \{1, 2\}$, it follows that for every $i \in [1\,..\,n']$, $s_i \in \mathrm{LexRange}(P'_1, P'_2, T)$ holds if and only if $i \in (b_1\,..\,b_2]$.

First, we prove that $\mathrm{LexRange}(P_1, P_2, T) \subseteq A$. Let $j \in \mathrm{LexRange}(P_1, P_2, T)$ and let $j' = j + \delta_{\mathrm{text}}$.

- First, observe that by the above discussion, we have $j \in \mathrm{Occ}(D, T)$.
- Second, note that since $\mathrm{lcp}(P_1, P_2) \geq |D| \geq \delta_{\mathrm{text}}$, and since for $k \in \{1, 2\}$ it holds $\delta_{\mathrm{text}} + |P'_k| = |P_k|$, it follows by $j \in \mathrm{LexRange}(P_1, P_2, T)$ that $j' \in \mathrm{LexRange}(P'_1, P'_2, T)$.

- Finally, we prove that $j' \in \mathsf{S}$. By Definition 7.16, there exists $i \in [1\,..\,n-3\tau+2] \setminus \mathsf{R}(\tau,T)$ such that $i \in \mathrm{Occ}(D,T)$ and $\mathrm{succ}_{\mathsf{S}}(i) = i + |D| - 2\tau = i + \delta_{\text{text}}$. Thus, $i + \delta_{\text{text}} \in \mathsf{S}$. Since above we proved that $j \in \mathrm{Occ}(D,T)$, it follows that $T[j+\delta_{\text{text}}\,..\,j+|D|) = T[i+\delta_{\text{text}}\,..\,i+|D|)$, and hence by the consistency of S (Definition 2.17(1)), $j' = j + \delta_{\text{text}} \in \mathsf{S}$.

By $j' \in \mathsf{S}$, there exists $i \in [1\,..\,n']$ such that $s_i = j'$. Above we proved that $j' \in \mathrm{LexRange}(P_1', P_2', T)$. Thus, by $s_i = j'$ and the earlier characterization of the set $\{s_t\}_{t\in(b_1\,..\,b_2]}$, we obtain $i \in (b_1\,..\,b_2]$. It remains to note that above we also proved that $j = j' - \delta_{\text{text}} = s_i - \delta_{\text{text}} \in \mathrm{Occ}(D,T)$. Putting everything together, we have thus proved that there exists $i \in (b_1\,..\,b_2]$ such that $j = s_i - \delta_{\text{text}}$ and $s_i - \delta_{\text{text}} \in \mathrm{Occ}(D,T)$, i.e., $j \in A$.

We now prove that $A \subseteq \mathrm{LexRange}(P_1, P_2, T)$. Let $j \in A$. To prove $j \in \mathrm{LexRange}(P_1,P_2,T)$, by Definition 2.4, we need to show that $j \in [1\,..\,n]$ and $P_1 \preceq T[j\,..\,n] \prec P_2$. We proceed as follows:

- First, note that by $j \in A$, there exists $i \in (b_1\,..\,b_2]$ such that $j = s_i - \delta_{\text{text}}$ and $s_i - \delta_{\text{text}} \in \mathrm{Occ}(D,T)$. In particular, by Definition 2.1, this implies $j = s_i - \delta_{\text{text}} \in [1\,..\,n]$.
- Next, observe that by $i \in (b_1\,..\,b_2]$ and the above characterization of $\{s_t\}_{t\in(b_1\,..\,b_2]}$, it follows that $s_i \in \mathrm{LexRange}(P_1', P_2', T)$, i.e., $P_1' \preceq T[s_i\,..\,n] \prec P_2'$. By $s_i = j+\delta_{\text{text}}$, we thus have $P_1' \preceq T[j+\delta_{\text{text}}\,..\,n] \prec P_2'$. Recall now that we assumed that both P_1 and P_2 are prefixed with D. Moreover, above we observed that $j \in \mathrm{Occ}(D,T)$. Since $|D| \geq \delta_{\text{text}}$, we thus have $P_1[1\,..\,\delta_{\text{text}}] = P_2[1\,..\,\delta_{\text{text}}] = T[j\,..\,j+\delta_{\text{text}}) = D[1\,..\,\delta_{\text{text}}]$. Combining with $P_1' \preceq T[j+\delta_{\text{text}}\,..\,n] \prec P_2'$, we thus obtain $P_1 \preceq T[j\,..\,n] \prec P_2$.

This concludes the proof of $j \in \mathrm{LexRange}(P_1,P_2,T)$. □

Lemma 7.24. *Let $T \in \Sigma^n$, $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$, and assume that $T[n]$ does not occur in $T[1\,..\,n)$. Let S be a τ-synchronizing set of T. Denote $(s_i)_{i\in[1\,..\,n']} = \mathrm{LexSorted}(\mathsf{S},T)$ (Definition 7.13). Let $A_{\mathsf{S}}[1\,..\,n']$ and $A_{\text{str}}[1\,..\,n']$ be defined by*

- $A_{\mathsf{S}}[i] = s_i$,
- $A_{\text{str}}[i] = \overline{D_i}$, *where* $D_i = T^{\infty}[s_i - \tau\,..\,s_i + 2\tau)$.

Let $D \in \mathcal{D}(\tau,T,\mathsf{S})$ (Definition 7.16) and let $P_1, P_2 \in \Sigma^+$ be τ-nonperiodic patterns (Definition 7.5) both having D as a prefix. Denote $\delta_{\text{text}} = |D| - 2\tau$. For $k \in \{1,2\}$, let $P_k' = P_k(\delta_{\text{text}}\,..\,|P_k|]$, and $b_k = |\{i \in [1\,..\,n'] : T[s_i\,..\,n] \prec P_k'\}|$. Then, it holds (see Definition 2.4)

$$\mathrm{LexRange}(P_1,P_2,T) = \{A_{\mathsf{S}}[i] - \delta_{\text{text}} : i \in (b_1\,..\,b_2] \text{ and } \overline{D} \text{ is a prefix of } A_{\text{str}}[i]\}.$$

Proof. Observe that for every $i \in [1\,..\,n']$, $s_i - \delta_{\text{text}} \in \mathrm{Occ}(D,T)$ holds if and only if $\overline{D}$ is a prefix of $A_{\text{str}}[i]$. The proof of this equivalence follows as in [KK24, Lemma 5.15] (full version).

By putting together the above equivalence and Lemma 7.23, we obtain

$$\begin{aligned}\mathrm{LexRange}(P_1,P_2,T) &= \{s_i - \delta_{\text{text}} : i \in (b_1\,..\,b_2] \text{ and } s_i - \delta_{\text{text}} \in \mathrm{Occ}(D,T)\}\\ &= \{A_{\mathsf{S}}[i] - \delta_{\text{text}} : i \in (b_1\,..\,b_2] \text{ and } \overline{D} \text{ is a prefix of } A_{\text{str}}[i]\}.\end{aligned}$$ □

Algorithms

Proposition 7.25. *Consider a data structure answering prefix range reporting queries that, given any sequence $W[1\,..\,k]$ strings of length ℓ over alphabet $[0\,..\,\sigma)$ achieves the following complexities (where the input strings during construction are given in the packed representation, and at query time we are given any $b,e \in [0\,..\,k]$ and the packed representation of any $X \in [0\,..\,\sigma)^{\leq \ell}$, and we return the set $\mathcal{I} := \{i \in (b\,..\,e] : X$ is a prefix of $W[i]\}$):*

- *space usage $S(k,\ell,\sigma)$,*
- *preprocessing time $P_t(k,\ell,\sigma)$,*
- *preprocessing space $P_s(k,\ell,\sigma)$,*
- *query time $Q_{\text{base}}(k,\ell,\sigma) + |\mathcal{I}| \cdot Q_{\text{elem}}(k,\ell,\sigma)$.*

Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. There exist positive integers $k = \Theta(n/\log_\sigma n)$ and $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n + P_t(k,\ell,\sigma))$ time and using $\mathcal{O}(n/\log_\sigma n + P_s(k,\ell,\sigma))$ working space construct a data structure of size $\mathcal{O}(n/\log_\sigma n + S(k,\ell,\sigma))$ that answers the following queries:

1. *Given the packed representation of a τ-nonperiodic (Definition 7.5) patterns $P_1, P_2 \in [0\,..\,\sigma)^+$ such that $\mathrm{lcp}(P_1, P_2) \geq 3\tau - 1$, computes the set $\mathcal{J} := \mathrm{LexRange}(P_1, P_2, T)$ (see Definition 2.4) in $\mathcal{O}(\log\log n + Q_{\mathrm{base}}(k, \ell, \sigma) + |\mathcal{J}| \cdot Q_{\mathrm{elem}}(k, \ell, \sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.*
2. *Given the packed representation of a τ-nonperiodic pattern $P \in [0\,..\,\sigma)^+$ satisfying $|P| \geq 3\tau - 1$, computes the set $\mathcal{J} := \mathrm{LexRange}(P, P'c^\infty, T)$ (where $P' = P[1\,..\,3\tau - 1]$ and $c = \sigma - 1$) in $\mathcal{O}(\log\log n + Q_{\mathrm{base}}(k, \ell, \sigma) + |\mathcal{J}| \cdot Q_{\mathrm{elem}}(k, \ell, \sigma) + |P|/\log_\sigma n)$ time.*

Proof. We use the following definitions. Let S be a τ-synchronizing set of T of size $|\mathsf{S}| = \mathcal{O}(\frac{n}{\tau})$ constructed using Theorem 2.20. Denote $n' = |\mathsf{S}|$. Denote $(s_i)_{i\in[1\,..\,n']} = \mathrm{LexSorted}(\mathsf{S}, T)$ (Definition 7.13). Let $A_{\mathsf{S}}[1\,..\,n']$ be an array defined by $A_{\mathsf{S}}[i] = s_i$. Let $k = \max(n', k') = \Theta(n/\log_\sigma n)$, where $k' = \lceil n/\log_\sigma n\rceil$, and let $A_{\mathrm{str}}[1\,..\,k]$ be an array defined so that for every $i \in [1\,..\,n']$, $A_{\mathrm{str}}[i] = \overline{D_i}$, where $D_i = T^\infty[s_i - \tau\,..\,s_i + 2\tau)$. We leave the remaining element initialized arbitrarily. Denote $\ell = 3\tau$, and note that by the same analysis as in Theorem 4.18, it holds $\ell \leq (1 + \lfloor\log k\rfloor)/\lceil\log\sigma\rceil$.

Components The data structure consists of the following components:

1. The structure from Proposition 7.10 for text T. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
2. The structure $\mathrm{NavNonperiodic}(\mathsf{S}, \tau, T)$ from Proposition 7.21. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
3. The structure from the claim (answering prefix range reporting queries) for the sequence $A_{\mathrm{str}}[1\,..\,k]$. It needs $\mathcal{O}(S(k, \ell, \sigma))$ space.
4. The array $A_{\mathsf{S}}[1\,..\,n']$ in plain form using $\mathcal{O}(n') = \mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n + S(k, \ell, \sigma))$ space.

Implementation of queries The queries are answered as follows.

1. Let $P_1, P_2 \in [0\,..\,\sigma)^+$ be τ-nonperiodic patterns satisfying $\mathrm{lcp}(P_1, P_2) \geq 3\tau - 1$. Denote $P' = P_1[1\,..\,3\tau - 1] = P_2[1\,..\,3\tau - 1]$. Given the packed representation of P_1 and P_2, we compute $\mathrm{LexRange}(P_1, P_2, T)$ as follows:
   (a) First, using Proposition 7.10, in $\mathcal{O}(1)$ time we compute $\mathrm{RangeBeg}(P', T)$ and $\mathrm{RangeEnd}(P', T)$. This lets us determine $|\mathrm{Occ}(P', T)|$. If $\mathrm{Occ}(P', T) = \emptyset$, then it holds $\mathrm{LexRange}(P_1, P_2, T) = \emptyset$ (since any element of this set would have P' as a prefix), and we finish the query algorithm. Let us now assume that $\mathrm{Occ}(P', T) \neq \emptyset$.
   (b) Using Proposition 7.21(2a), in $\mathcal{O}(1)$ time compute the packed representation of the string $D = \mathrm{DistPrefix}(P_1, \tau, T, \mathsf{S})$ (Definition 7.20). Note that since $\mathrm{lcp}(P_1, P_2) \geq 3\tau - 1 \geq |D|$, D is also a prefix of P_2 (and by Lemma 7.19 no other string in $\mathcal{D}(\tau, T, \mathsf{S})$ is a prefix of P_2). In $\mathcal{O}(1)$ time we then calculate $\delta_{\mathrm{text}} = |D| - 2\tau$.
   (c) Using Proposition 7.21(1), in $\mathcal{O}(1)$ time compute the packed representation of $\overline{D}$.
   (d) Denote $P'_k = P_k(\delta_{\mathrm{text}}\,..\,|P_k|]$, where $k \in \{1, 2\}$. Using Proposition 7.21(2b), compute the value $b_k = |\{i \in [1\,..\,n'] : T[s_i\,..\,n] \prec P'_k\}|$ for $k \in \{1, 2\}$. This takes $\mathcal{O}(\log\log n + |P'_1|/\log_\sigma n + |P'_2|/\log_\sigma n) = \mathcal{O}(\log\log n + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.
   (e) Using the data structure from the claim, compute the elements of the set $\mathcal{I} = \{i \in (b_1\,..\,b_2] : \overline{D}$ is a prefix of $A_{\mathrm{str}}[i]\}$ in $\mathcal{O}(Q_{\mathrm{base}}(k, \ell, \sigma) + |\mathcal{I}| \cdot Q_{\mathrm{elem}}(k, \ell, \sigma))$ time. In $\mathcal{O}(1 + |\mathcal{I}|)$ time we then compute and return the set $\mathcal{J} = \{A_{\mathsf{S}}[i] - \delta_{\mathrm{text}} : i \in \mathcal{I}\}$ as the answer. By Lemma 7.24, it holds $\mathcal{J} = \mathrm{LexRange}(P_1, P_2, T)$. Note also that $|\mathcal{I}| = |\mathcal{J}|$. Finally, observe that the array A_{str} defined here is padded with extra strings compared to the array in Lemma 7.24, but this does not affect the result of the query, since $b_1, b_2 \leq n'$.

   In total, we spend $\mathcal{O}(\log\log n + Q_{\mathrm{base}}(k, \ell, \sigma) + |\mathcal{J}| \cdot Q_{\mathrm{elem}}(k, \ell, \sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time, where $\mathcal{J} = \mathrm{LexRange}(P_1, P_2, T)$.
2. Let $P \in [0\,..\,\sigma)^+$ be a τ-nonperiodic pattern satisfying $|P| \geq 3\tau - 1$. To compute the elements of the set $\mathrm{LexRange}(P, P'c^\infty, T)$ (where $P' = P[1\,..\,3\tau - 1]$ and $c = \sigma - 1$), we set $P_1 = P$ and $P_2 = P'c^\infty$, and then proceed similarly as above, except we compute $b_2 = |\{i \in [1\,..\,n'] : T[s_i\,..\,n] \prec P'_2\}|$ using the second integer computed in Proposition 7.21(2b) (which requires only the packed representation of P'). The whole query takes $\mathcal{O}(\log\log n + Q_{\mathrm{base}}(k, \ell, \sigma) + |\mathcal{J}| \cdot Q_{\mathrm{elem}}(k, \ell, \sigma) + |P|/\log_\sigma n)$ time, where $\mathcal{J} = \mathrm{LexRange}(P, P'c^\infty, T)$.

Construction algorithm The components of the structure are constructed as follows:

1. We apply Proposition 7.10. This uses $\mathcal{O}(n/\log_\sigma n)$ time and working space.
2. Using Theorem 2.20, we compute the τ-synchronizing set S satisfying $|\mathsf{S}| = \mathcal{O}(\frac{n}{\tau}) = \mathcal{O}(n/\log_\sigma n)$ in $\mathcal{O}(n/\log_\sigma n)$ time. Then, using S and the packed representation of T as input, we construct NavNonperiodic(S, τ, T) in $\mathcal{O}(n/\log_\sigma n)$ time using Proposition 7.21.
3. To construct the next component of the structure, we proceed as follows:
   (a) Using Theorem 7.22, in $\mathcal{O}(\frac{n}{\tau}) = \mathcal{O}(n/\log_\sigma n)$ time we first compute the sequence $(s_i)_{i\in[1\,..\,n']} = \text{LexSorted}(\mathsf{S}, T)$ (Definition 7.13).
   (b) Using Proposition 7.21(1), in $\mathcal{O}(n') = \mathcal{O}(n/\log_\sigma n)$ time we compute the elements of the array A_{str} at indexes $i \in [1\,..\,n']$, i.e., $A_{\text{str}}[i] = \overline{D_i}$, where $D_i = T^\infty[s_i - \tau\,..\,s_i + 2\tau)$. In $\mathcal{O}(k - n') = \mathcal{O}(k) = \mathcal{O}(n/\log_\sigma n)$ time we then pad the remaining elements of A_{str} at indexes $i \in (n'\,..\,k]$.
   (c) Finally, we apply the preprocessing from the claim to the array A_{str}. This takes $\mathcal{O}(P_t(k, \ell, \sigma))$ time and uses $\mathcal{O}(P_s(k, \ell, \sigma))$ working space.

   In total, constructing this component of the structure takes $\mathcal{O}(n/\log_\sigma n + P_t(k, \ell, \sigma))$ time and uses $\mathcal{O}(n/\log_\sigma n + P_s(k, \ell, \sigma))$ working space.
4. To construct the last component, we simply store the sequence $(s_i)_{i\in[1\,..\,n']} = \text{LexSorted}(\mathsf{S}, T)$ (computed above in $\mathcal{O}(n/\log_\sigma n)$ time) in array $A_{\mathsf{S}}[1\,..\,n']$.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n + P_t(k, \ell, \sigma))$ time and uses $\mathcal{O}(n/\log_\sigma n + P_s(k, \ell, \sigma))$ working space. □

7.3.1.4 The Periodic Patterns

Preliminaries

Definition 7.26 (Basic properties of τ-periodic patterns)**.** Let $\tau \geq 1$ and let $P \in \Sigma^+$ be a τ-periodic pattern (Definition 7.5). Denote $p = \text{per}(P[1\,..\,3\tau - 1])$. We then define

- $\text{root}(P, \tau) := \min\{P[1 + t\,..\,1 + t + p) : t \in [0\,..\,p)\}$,
- $e(P, \tau) := 1 + p + \text{lcp}(P[1\,..\,|P|], P[1 + p\,..\,|P|])$.

Furthermore, we then let

$$\text{type}(P, \tau) = \begin{cases} +1 & \text{if } e(P, \tau) \leq |P| \text{ and } P[e(P, \tau)] \succ P[e(P, \tau) - p], \\ -1 & \text{otherwise.} \end{cases}$$

Remark 7.27. Observe that in Definition 7.26, we can write $P[1\,..\,e(P, \tau)) = H'H^kH''$, where $H = \text{root}(P, \tau)$, and H' (resp. H'') is a proper suffix (resp. proper prefix) of H. This factorization is unique, since the opposite would contradict the synchronization property of primitive strings [CHL07, Lemma 1.11].

Definition 7.28 (Run properties in a τ-periodic pattern)**.** Let $\tau \geq 1$ and $P \in \Sigma^+$ be a τ-periodic pattern. Let $P[1\,..\,e(P, \tau)) = H'H^kH''$ be such that $H = \text{root}(P, \tau)$, and H' (resp. H'') is a proper suffix (resp. proper prefix) of H (see Remark 7.27). We then define:

- $\text{head}(P, \tau) := |H'|$,
- $\exp(P, \tau) := k$,
- $\text{tail}(P, \tau) := |H''|$,
- $e^{\text{full}}(P, \tau) := 1 + |H'| + k \cdot |H| = e(P, \tau) - \text{tail}(P, \tau)$.

Lemma 7.29 ([KK23b])**.** *Let $\tau \geq 1$ and $P \in \Sigma^+$ be a τ-periodic pattern. For every $P' \in \Sigma^+$, $\text{lcp}(P, P') \geq 3\tau - 1$ holds if and only if P' is τ-periodic, $\text{root}(P', \tau) = \text{root}(P, \tau)$, and $\text{head}(P', \tau) = \text{head}(P, \tau)$. Moreover, if $e(P, \tau) \leq |P|$ and $\text{lcp}(P, P') \geq e(P, \tau)$ (which holds, in particular, when P is a prefix of P'), then:*

- $e(P', \tau) = e(P, \tau)$,
- $\text{tail}(P', \tau) = \text{tail}(P, \tau)$,
- $e^{\text{full}}(P', \tau) = e^{\text{full}}(P, \tau)$,
- $\exp(P', \tau) = \exp(P, \tau)$,

- $\mathrm{type}(P', \tau) = \mathrm{type}(P, \tau)$.

Lemma 7.30 ([KK23b]). *Let $\tau \geq 1$ and $P_1, P_2 \in \Sigma^+$ be τ-periodic patterns such that $\mathrm{root}(P_1, \tau) = \mathrm{root}(P_2, \tau)$ and $\mathrm{head}(P_1, \tau) = \mathrm{head}(P_2, \tau)$. Denote $t_1 = e(P_1, \tau) - 1$ and $t_2 = e(P_2, \tau) - 1$. Then, it holds $\mathrm{lcp}(P_1, P_2) \geq \min(t_1, t_2)$. Moreover:*

1. *If $\mathrm{type}(P_1, \tau) \neq \mathrm{type}(P_2, \tau)$ or $t_1 \neq t_2$, then $P_1 \neq P_2$ and $\mathrm{lcp}(P_1, P_2) = \min(t_1, t_2)$,*
2. *If $\mathrm{type}(P_1, \tau) \neq \mathrm{type}(P_2, \tau)$, then $P_1 \prec P_2$ if and only if $\mathrm{type}(P_1, \tau) < \mathrm{type}(P_2, \tau)$,*
3. *If $\mathrm{type}(P_1, \tau) = -1$, then $t_1 < t_2$ implies $P_1 \prec P_2$,*
4. *If $\mathrm{type}(P_1, \tau) = +1$, then $t_1 < t_2$ implies $P_1 \succ P_2$,*
5. *If $\mathrm{type}(P_1, \tau) = \mathrm{type}(P_2, \tau) = -1$ and $t_1 \neq t_2$, then $t_1 < t_2$ if and only if $P_1 \prec P_2$,*
6. *If $\mathrm{type}(P_1, \tau) = \mathrm{type}(P_2, \tau) = +1$ and $t_1 \neq t_2$, then $t_1 < t_2$ if and only if $P_1 \succ P_2$.*

Definition 7.31 (Properties of τ-periodic positions)**.** Let $T \in \Sigma^n$, $\tau \in [1 \,..\, \lfloor \frac{n}{2} \rfloor]$, and $j \in \mathsf{R}(\tau, T)$. Letting $P = T[j \,..\, n]$, we define:

- $\mathrm{root}(j, \tau, T) := \mathrm{root}(P, \tau)$,
- $\mathrm{head}(j, \tau, T) := \mathrm{head}(P, \tau)$,
- $\exp(j, \tau, T) := \exp(P, \tau)$,
- $\mathrm{tail}(j, \tau, T) := \mathrm{tail}(P, \tau)$,
- $e(j, \tau, T) := j + e(P, \tau) - 1$,
- $e^{\mathrm{full}}(j, \tau, T) := j + e^{\mathrm{full}}(P, \tau) - 1$,
- $\mathrm{type}(j, \tau, T) := \mathrm{type}(P, \tau)$.

Remark 7.32. Note that applying the notation for τ-periodic patterns (Definition 7.5) to P in Definition 7.31 is well-defined since it is easy to see that for every $j \in \mathsf{R}(\tau, T)$ (Definition 2.17), the string $T[j \,..\, n]$ is τ-periodic. Note also that, letting $s = \mathrm{head}(j, \tau, T)$, $H = \mathrm{root}(j, \tau, T)$, $p = |H|$, and $k = \exp(j, \tau, T)$, it holds:

- $e(j, \tau, T) = j + p + \mathrm{LCE}_T(j, j + p)$,
- $e^{\mathrm{full}}(j, \tau, T) = j + s + kp = e(j, \tau, T) - \mathrm{tail}(j, \tau, T)$.

Definition 7.33 (Subsets of $\mathsf{R}(\tau, T)$)**.** Let $T \in \Sigma^n$ and $\tau \in [1 \,..\, \lfloor \frac{n}{2} \rfloor]$. For every $H \in \Sigma^+$, $s \in \mathbb{Z}_{\geq 0}$, and $k \in \mathbb{Z}_{>0}$, we define the following subsets of $\mathsf{R}(\tau, T)$:

- $\mathsf{R}^-(\tau, T) := \{j \in \mathsf{R}(\tau, T) : \mathrm{type}(j, \tau, T) = -1\}$,
- $\mathsf{R}^+(\tau, T) := \mathsf{R}(\tau, T) \setminus \mathsf{R}^-(\tau, T)$,
- $\mathsf{R}_H(\tau, T) := \{j \in \mathsf{R}(\tau, T) : \mathrm{root}(j, \tau, T) = H\}$,
- $\mathsf{R}^-_H(\tau, T) := \mathsf{R}^-(\tau, T) \cap \mathsf{R}_H(\tau, T)$,
- $\mathsf{R}^+_H(\tau, T) := \mathsf{R}^+(\tau, T) \cap \mathsf{R}_H(\tau, T)$,
- $\mathsf{R}_{s,H}(\tau, T) := \{j \in \mathsf{R}_H(\tau, T) : \mathrm{head}(j, \tau, T) = s\}$,
- $\mathsf{R}^-_{s,H}(\tau, T) := \mathsf{R}^-(\tau, T) \cap \mathsf{R}_{s,H}(\tau, T)$,
- $\mathsf{R}^+_{s,H}(\tau, T) := \mathsf{R}^+(\tau, T) \cap \mathsf{R}_{s,H}(\tau, T)$,
- $\mathsf{R}_{s,k,H}(\tau, T) := \{j \in \mathsf{R}_{s,H}(\tau, T) : \exp(j, \tau, T) = k\}$,
- $\mathsf{R}^-_{s,k,H}(\tau, T) := \mathsf{R}^-(\tau, T) \cap \mathsf{R}_{s,k,H}(\tau, T)$,
- $\mathsf{R}^+_{s,k,H}(\tau, T) := \mathsf{R}^+(\tau, T) \cap \mathsf{R}_{s,k,H}(\tau, T)$.

Definition 7.34 (Starting positions of τ-runs)**.** Let $T \in \Sigma^n$ and $\tau \in [1 \,..\, \lfloor \frac{n}{2} \rfloor]$. The set of starting positions of maximal blocks in $\mathsf{R}(\tau, T)$ is denoted

$$\mathsf{R}'(\tau, T) := \{j \in \mathsf{R}(\tau, T) : j - 1 \notin \mathsf{R}(\tau, T)\}.$$

For every $H \in \Sigma^+$, we then define:

- $\mathsf{R}'^-(\tau, T) := \mathsf{R}'(\tau, T) \cap \mathsf{R}^-(\tau, T)$,
- $\mathsf{R}'^+(\tau, T) := \mathsf{R}'(\tau, T) \cap \mathsf{R}^+(\tau, T)$,
- $\mathsf{R}'^-_H(\tau, T) := \mathsf{R}'(\tau, T) \cap \mathsf{R}^-_H(\tau, T)$,
- $\mathsf{R}'^+_H(\tau, T) := \mathsf{R}'(\tau, T) \cap \mathsf{R}^+_H(\tau, T)$.

Lemma 7.35 ([KK23b]). *Let $T \in \Sigma^n$ and $\tau \in [1 \,..\, \lfloor \frac{n}{2} \rfloor]$. For every $j \in \mathsf{R}(\tau, T)$ such that $j - 1 \in \mathsf{R}(\tau, T)$, it holds*

- $\mathrm{root}(j-1,\tau,T)=\mathrm{root}(j,\tau,T)$,
- $e(j-1,\tau,T)=e(j,\tau,T)$,
- $\mathrm{tail}(j-1,\tau,T)=\mathrm{tail}(j,\tau,T)$,
- $e^{\mathrm{full}}(j-1,\tau,T)=e^{\mathrm{full}}(j,\tau,T)$,
- $\mathrm{type}(j-1,\tau,T)=\mathrm{type}(j,\tau,T)$.

Lemma 7.36 ([KK23b]). *Let $T\in\Sigma^n$ and $\tau\in[1\,..\,\lfloor\frac{n}{2}\rfloor]$. For every $j\in\mathsf{R}(\tau,T)$, it holds*

$$e(j,\tau,T)=\max\{j'\in[j\,..\,n]:[j\,..\,j']\subseteq\mathsf{R}(\tau,T)\}+3\tau-1.$$

Lemma 7.37 ([KK23b]). *Let $T\in\Sigma^n$, $\tau\in[1\,..\,\lfloor\frac{n}{2}\rfloor]$, and $j\in[1\,..\,n]$.*

1. *Let $P\in\Sigma^+$ be a τ-periodic pattern. Then, the following conditions are equivalent:*
   - $\mathrm{lcp}(P,T[j\,..\,n])\geq 3\tau-1$,
   - $j\in\mathsf{R}(\tau,T)$, $\mathrm{root}(j,\tau,T)=\mathrm{root}(P,\tau)$, *and* $\mathrm{head}(j,\tau,T)=\mathrm{head}(P,\tau)$.

   Moreover, if, letting $t=e(P,\tau)-1$, it holds $\mathrm{lcp}(P,T[j\,..\,n])>t$, then:
   - $e(P,\tau)-1=e(j,\tau,T)-j$,
   - $\mathrm{tail}(P,\tau)=\mathrm{tail}(j,\tau,T)$,
   - $e^{\mathrm{full}}(P,\tau)-1=e^{\mathrm{full}}(j,\tau,T)-j$,
   - $\exp(P,\tau)=\exp(j,\tau,T)$,
   - $\mathrm{type}(P,\tau)=\mathrm{type}(j,\tau,T)$.
2. *Let $j'\in\mathsf{R}(\tau,T)$. Then, the following conditions are equivalent:*
   - $\mathrm{LCE}_T(j',j)\geq 3\tau-1$,
   - $j\in\mathsf{R}(\tau,T)$, $\mathrm{root}(j,\tau,T)=\mathrm{root}(j',\tau,T)$, *and* $\mathrm{head}(j',\tau,T)=\mathrm{head}(j,\tau,T)$.

   Moreover, if letting $t=e(j,\tau,T)-j$, it holds $\mathrm{LCE}_T(j,j')>t$, then:
   - $e(j',\tau,T)-j'=e(j,\tau,T)-j$,
   - $\mathrm{tail}(j',\tau,T)=\mathrm{tail}(j,\tau,T)$,
   - $e^{\mathrm{full}}(j',\tau,T)-j'=e^{\mathrm{full}}(j,\tau,T)-j$,
   - $\exp(j',\tau,T)=\exp(j,\tau,T)$,
   - $\mathrm{type}(j',\tau,T)=\mathrm{type}(j,\tau,T)$.

Lemma 7.38 ([KK23b]). *Let $T\in\Sigma^n$, $\tau\in[1\,..\,\lfloor\frac{n}{2}\rfloor]$, and $j\in\mathsf{R}(\tau,T)$. Let $j'\in\mathsf{R}(\tau,T)$ be such that $\mathrm{root}(j',\tau,T)=\mathrm{root}(j,\tau,T)$ and $\mathrm{head}(j',\tau,T)=\mathrm{head}(j,\tau,T)$. Then, letting $t_1=e(j,\tau,T)-j$ and $t_2=e(j',\tau,T)-j'$, it holds $\mathrm{LCE}_T(j,j')\geq\min(t_1,t_2)$ and:*

1. *If $\mathrm{type}(j,\tau,T)\neq\mathrm{type}(j',\tau,T)$ or $t_1\neq t_2$, then $\mathrm{LCE}_T(j,j')=\min(t_1,t_2)$,*
2. *If $\mathrm{type}(j,\tau,T)\neq\mathrm{type}(j',\tau,T)$, then $T[j\,..\,n]\prec T[j'\,..\,n]$ iff $\mathrm{type}(j,\tau,T)<\mathrm{type}(j',\tau,T)$,*
3. *If $\mathrm{type}(j,\tau,T)=\mathrm{type}(j',\tau,T)=-1$ and $t_1\neq t_2$, then $t_1<t_2$ iff $T[j\,..\,n]\prec T[j'\,..\,n]$,*
4. *If $\mathrm{type}(j,\tau,T)=\mathrm{type}(j',\tau,T)=+1$ and $t_1\neq t_2$, then $t_1<t_2$ iff $T[j\,..\,n]\succ T[j'\,..\,n]$.*

Lemma 7.39 ([KK23b]). *Let $T\in\Sigma^n$ and $\tau\in[1\,..\,\lfloor\frac{n}{2}\rfloor]$. Let $j,j',j''\in[1\,..\,n]$ be such that $j,j''\in\mathsf{R}(\tau,T)$, $j'\notin\mathsf{R}(\tau,T)$, and $j<j'<j''$. Then, it holds $e(j,\tau,T)\leq j''+\tau-1$.*

Lemma 7.40 ([KK23a, Section 5.3.2]). *For every $T\in\Sigma^n$ and $\tau\in[1\,..\,\lfloor\frac{n}{2}\rfloor]$, it holds $|\mathsf{R}'(\tau,T)|\leq\frac{2n}{\tau}$ and $\sum_{i\in\mathsf{R}'(\tau,T)}e(i,\tau,T)-i\leq 2n$.*

Lemma 7.41 ([KK23b]). *Let $T\in\Sigma^n$ and $\tau\in[1\,..\,\lfloor\frac{n}{2}\rfloor]$. For any $j,j'\in\mathsf{R}'(\tau,T)$, $j\neq j'$ implies $e^{\mathrm{full}}(j,\tau,T)\neq e^{\mathrm{full}}(j',\tau,T)$.*

Lemma 7.42 ([KK23b]). *Let $T\in\Sigma^n$ and $\tau\in[1\,..\,\lfloor\frac{n}{2}\rfloor]$. Let $H\in\Sigma^+$, $p=|H|$, $s\in[0\,..\,p)$, and $k_{\min}=\lceil\frac{3\tau-1-s}{p}\rceil-1$. For every $k\in(k_{\min}\,..\,n]$, it holds*

$$\mathsf{R}^-_{s,k,H}(\tau,T)=\{e^{\mathrm{full}}(j,\tau,T)-s-kp:j\in\mathsf{R}'^-_H(\tau,T)\text{ and }s+kp\leq e^{\mathrm{full}}(j,\tau,T)-j\}.$$

Lemma 7.43 ([KK25]). *Let $T\in\Sigma^n$ and $\tau\in[1\,..\,\lfloor\frac{n}{2}\rfloor]$. Let $H\in\Sigma^+$, $p=|H|$, $s\in[0\,..\,p)$, and $k_{\min}=\lceil\frac{3\tau-1-s}{p}\rceil-1$. For every $k\in(k_{\min}\,..\,n)$, it holds*

$$|\mathsf{R}_{s,k,H}(\tau,T)|\geq|\mathsf{R}_{s,k+1,H}(\tau,T)|.$$

Definition 7.44 (Anchored lex-orderings of τ-runs)**.** Let $T \in \Sigma^n$ and $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$.

- Letting $m = |\mathsf{R}'^{-}(\tau, T)|$, by $\mathrm{RunsLexSorted}^{-}(\tau, T)$, we denote a sequence $(a_i)_{i\in[1\,..\,m]}$ containing all elements of $\mathsf{R}'^{-}(\tau, T)$ such that for every $i, j \in [1\,..\,m]$, $i < j$ implies $\mathrm{root}(a_i, \tau, T) \prec \mathrm{root}(a_j, \tau, T)$, or $\mathrm{root}(a_i, \tau, T) = \mathrm{root}(a_j, \tau, T)$ and $T[e^{\mathrm{full}}(a_i, \tau, T)\,..\,n] \prec T[e^{\mathrm{full}}(a_j, \tau, T)\,..\,n]$.
- For every $H \in \Sigma^+$, letting $m_H = |\mathsf{R}'^{-}_H(\tau, T)|$, by $\mathrm{RunsLexSorted}^{-}_H(\tau, T)$, we denote a sequence $(a_i)_{i\in[1\,..\,m_H]}$ containing all elements of $\mathsf{R}'^{-}_H(\tau, T)$ such that for every $i, j \in [1\,..\,m_H]$, $i < j$ implies $T[e^{\mathrm{full}}(a_i, \tau, T)\,..\,n] \prec T[e^{\mathrm{full}}(a_j, \tau, T)\,..\,n]$.

Definition 7.45 (Text orderings of τ-runs)**.** Let $T \in \Sigma^n$ and $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$.

- Letting $m = |\mathsf{R}'^{-}(\tau, T)|$, by $\mathrm{RunsTextSorted}^{-}(\tau, T)$, we denote a sequence $(a_i)_{i\in[1\,..\,m]}$ containing all elements of $\mathsf{R}'^{-}(\tau, T)$ such that for every $i, j \in [1\,..\,m]$, $i < j$ implies $\mathrm{root}(a_i, \tau, T) \prec \mathrm{root}(a_j, \tau, T)$, or $\mathrm{root}(a_i, \tau, T) = \mathrm{root}(a_j, \tau, T)$ and $a_i < a_j$.
- For every $H \in \Sigma^+$, letting $m_H = |\mathsf{R}'^{-}_H(\tau, T)|$, by $\mathrm{RunsTextSorted}^{-}_H(\tau, T)$, we denote a sequence $(a_i)_{i\in[1\,..\,m_H]}$ containing all elements of $\mathsf{R}'^{-}_H(\tau, T)$ such that for every $i, j \in [1\,..\,m_H]$, $i < j$ implies $a_i < a_j$.

Definition 7.46 (Length-based ordering of τ-runs)**.** Let $T \in \Sigma^n$ and $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$. Let $m = |\mathsf{R}'^{-}(\tau, T)|$. By $\mathrm{RunsLengthSorted}^{-}(\tau, T)$, we denote a sequence $(a_i)_{i\in[1\,..\,m]}$ containing all elements of $\mathsf{R}'^{-}(\tau, T)$ such that for every $i, j \in [1\,..\,m]$, $i < j$ implies that one of the following three conditions hold:

- $\mathrm{root}(a_i, \tau, T) \prec \mathrm{root}(a_j, \tau, T)$,
- $\mathrm{root}(a_i, \tau, T) = \mathrm{root}(a_j, \tau, T)$ and $e^{\mathrm{full}}(a_i, \tau, T) - a_i < e^{\mathrm{full}}(a_j, \tau, T) - a_j$,
- $\mathrm{root}(a_i, \tau, T) = \mathrm{root}(a_j, \tau, T)$, $e^{\mathrm{full}}(a_i, \tau, T) - a_i = e^{\mathrm{full}}(a_j, \tau, T) - a_j$, and $a_i < a_j$.

Definition 7.47 (Exponent-block indicator bitvector)**.** Let $T \in \Sigma^n$ and $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$. By

$$\mathrm{ExpBlockBitvector}^{-}(\tau, T),$$

we denote a bitvector $B[1\,..\,n]$ defined such that for every $i \in [1\,..\,n]$, $B[i] = \mathtt{0}$ holds if and only if $\mathrm{SA}_T[i] \in [1\,..\,n] \setminus \mathsf{R}^{-}(\tau, T)$, or $i < n$ and the positions $j = \mathrm{SA}_T[i]$ and $j' = \mathrm{SA}_T[i+1]$ satisfy $j, j' \in \mathsf{R}^{-}_{s,k,H}(\tau, T)$ for some $s \in \mathbb{Z}_{\geq 0}$, $k \in \mathbb{Z}_{>0}$, and $H \in \Sigma^+$.

Definition 7.48 (Leftmost representatives of τ-run occurrences)**.** For every $T \in \Sigma^n$ and $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$, we define

$$\mathsf{R}^{-}_{\min}(\tau, T) := \{j \in \mathsf{R}^{-}(\tau, T) : j = \min \mathrm{Occ}(T[j\,..\,e(j, \tau, T)), T) \cap \mathsf{R}^{-}(\tau, T)\}.$$

Definition 7.49 (Leftmost τ-run indicator bitvectors)**.** Let $T \in \Sigma^n$ and $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$. By

$$\mathrm{MinPosBitvector}^{-}(\tau, T),$$

we denote a bitvector $B[1\,..\,n]$ defined such that for every $i \in [1\,..\,n]$,

$$B[i] = \begin{cases} 1 & \text{if } \mathrm{SA}_T[i] \in \mathsf{R}^{-}_{\min}(\tau, T), \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, for every $H \in \Sigma^+$, $s \in [0\,..\,|H|)$, and $k \in \mathbb{Z}_{>0}$, we then denote

$$\mathrm{MinPosBitvector}^{-}_{s,H}(\tau, T) = B(x\,..\,y],$$
$$\mathrm{MinPosBitvector}^{-}_{s,k,H}(\tau, T) = B(x'\,..\,y'],$$

where $x, y, x', y' \in [0\,..\,n]$ are such that

$$\{\mathrm{SA}_T[i] : i \in (x\,..\,y]\} = \mathsf{R}^{-}_{s,H}(\tau, T),$$
$$\{\mathrm{SA}_T[i] : i \in (x'\,..\,y']\} = \mathsf{R}^{-}_{s,k,H}(\tau, T).$$

Remark 7.50. To show that x and y in Definition 7.49 are well-defined, note that by Lemma 7.37(2), the set $\mathsf{R}_{s,H}(\tau,T)$ occupies a contiguous block of positions in SA_T. Moreover, by Lemma 7.38(2), all positions $j \in \mathsf{R}_{s,H}(\tau,T)$ satisfying $\mathrm{type}(j,\tau,T) = -1$ precede positions $j' \in \mathsf{R}_{s,H}(\tau,T)$ satisfying $\mathrm{type}(j',\tau,T) = +1$. In order words, positions in $\mathsf{R}^{-}_{s,H}(\tau,T)$ occupy a contiguous block in SA_T. Thus, x, y are indeed well-defined.

To show that x' and y' in Definition 7.49 are well-defined, observe now that letting $a, b \in [0\,..\,n]$ be such that $\{\mathrm{SA}_T[i] : i \in (a\,..\,b]\} = \mathsf{R}^{-}_{s,H}(\tau,T)$, it follows by Lemma 7.38(3) that for every $i \in (a\,..\,b)$, it holds $e(\mathrm{SA}_T[i],\tau,T) - \mathrm{SA}_T[i] \leq e(\mathrm{SA}_T[i+1],\tau,T) - \mathrm{SA}_T[i+1]$. By $\mathrm{head}(\mathrm{SA}_T[i],\tau,T) = \mathrm{head}(\mathrm{SA}_T[i+1],\tau,T) = s$, we thus obtain

$$\begin{aligned}\exp(\mathrm{SA}_T[i],\tau,T) &= \lfloor \tfrac{e(\mathrm{SA}_T[i],\tau,T) - \mathrm{SA}_T[i] - s}{p} \rfloor \\ &\leq \lfloor \tfrac{e(\mathrm{SA}_T[i+1],\tau,T) - \mathrm{SA}_T[i+1] - s}{p} \rfloor \\ &= \exp(\mathrm{SA}_T[i+1],\tau,T).\end{aligned}$$

Thus, all positions from $\mathsf{R}^{-}_{s,k,H}(\tau,T)$ occupy a contiguous block in SA_T. The values x', y' are thus indeed well-defined.

Lemma 7.51 ([KK24]). *Let $T \in \Sigma^n$ and $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$. Let $H \in \Sigma^+$, $s \in \mathbb{Z}_{\geq 0}$, and $b, e \in [0\,..\,n]$ be such that $b < e$ and $\{\mathrm{SA}_T[i] : i \in (b\,..\,e]\} = \mathsf{R}^{-}_{s,H}(\tau,T)$. Let $B_{\min}[1\,..\,n] = \mathrm{MinPosBitvector}^{-}(\tau,T)$ (Definition 7.49). If $B_{\min}[i] = 1$ for some $i \in (b\,..\,e]$, then for every $i' \in (i\,..\,e]$, it holds*

$$\mathrm{SA}_T[i'] > \mathrm{SA}_T[i].$$

Basic Navigational Primitives

Proposition 7.52 ([KK23a, KK24]). *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct:*

1. *the packed representation of bitvector $\mathrm{ExpBlockBitvector}^{-}(\tau,T)$ (Definition 7.47),*
2. *the packed representation of bitvector $\mathrm{MinPosBitvector}^{-}(\tau,T)$ (Definition 7.49),*
3. *the sequence $\mathrm{RunsLexSorted}^{-}(\tau,T)$ (Definition 7.44),*

Proof. The construction proceeds as follows:

1. This follows by combining Proposition 5.3 and Proposition 5.15 from [KK23a].
2. This follows by combining Proposition 5.40 and Proposition 5.71 from [KK24].
3. This follows again by combining Proposition 5.3 and Proposition 5.15 from [KK23a]. □

Proposition 7.53. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. Denote $q = |\mathsf{R}'^{-}(\tau,T)|$ and $(a_i)_{i\in[1\,..\,q]} = \mathrm{RunsLexSorted}^{-}(\tau,T)$ (Definition 7.44). Let also $A_{\mathrm{pos}}[1\,..\,q]$ and $A_{\mathrm{len}}[1\,..\,q]$ be arrays defined by*

- $A_{\mathrm{pos}}[i] = e^{\mathrm{full}}(a_i,\tau,T)$,
- $A_{\mathrm{len}}[i] = e^{\mathrm{full}}(a_i,\tau,T) - a_i$.

Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure, denoted $\mathrm{NavPeriodic}(\tau,T)$, that answers the following queries:

1. *Given any $i \in [1\,..\,n]$ such that $\mathrm{SA}_T[i] \in \mathsf{R}(\tau,T)$, compute $\mathrm{SA}_T[i]$ in $\mathcal{O}(\log\log n)$ time.*
2. *Given any $j \in \mathsf{R}(\tau,T)$, in $\mathcal{O}(1)$ time compute:*
   - $\mathrm{head}(j,\tau,T)$,
   - $|\mathrm{root}(j,\tau,T)|$,
   - $\exp(j,\tau,T)$,
   - $e^{\mathrm{full}}(j,\tau,T)$,
   - *integers $x, y, z \in [0\,..\,n]$ satisfying $x \leq y \leq z$, $\{\mathrm{SA}_T[i]\}_{i\in(x\,..\,y]} = \mathsf{R}^{-}_{s,H}(\tau,T)$, and $\{\mathrm{SA}_T[i]\}_{i\in(y\,..\,z]} = \mathsf{R}^{+}_{s,H}(\tau,T)$, where $s = \mathrm{head}(j,\tau,T)$ and $H = \mathrm{root}(j,\tau,T)$.*

3. *Given any* $j \in \mathsf{R}^-(\tau, T)$*, in* $\mathcal{O}(1)$ *time compute*
   - *index* $i \in [1\,..\,q]$ *satisfying* $e^{\text{full}}(j, \tau, T) = e^{\text{full}}(a_i, \tau, T)$*,*
   - *integers* $x, y \in [0\,..\,q]$ *satisfying* $x = \sum_{H' \in [0\,..\,\sigma)^+ : H' \prec H} |\mathsf{R}'^-_{H'}(\tau, T)|$ *and* $y = x + |\mathsf{R}'^-_H(\tau, T)|$*, where* $H = \text{root}(j, \tau, T)$*.*
4. *Given the packed representation of any* τ*-periodic (Definition 7.5) pattern* $P \in [0\,..\,\sigma)^+$*, return the pair*
$$(\text{RangeBeg}(P, T), \text{RangeEnd}(P, T))$$
   in $\mathcal{O}(\log\log n + |P|/\log_\sigma n)$ *time.*
5. *Given any* $b, e \in [0\,..\,q]$ *and* $v \geq 0$ *such that* $\mathsf{three\text{-}sided\text{-}rmq}_{A_{\text{pos}}, A_{\text{len}}}(b, e, v) \neq \infty$ *(see Definition 2.12), return*
$$A_{\text{pos}}[\mathsf{three\text{-}sided\text{-}rmq}_{A_{\text{pos}}, A_{\text{len}}}(b, e, v)]$$
   in $\mathcal{O}(\log\log n)$ *time.*
6. *Given any* $b, e \in [0\,..\,q]$ *and* $v \geq 0$*, return*
$$\mathcal{I} = \{i \in (b\,..\,e] : e^{\text{full}}(a_i, \tau, T) - a_i \geq v\}$$
   in $\mathcal{O}(1 + |\mathcal{I}|)$ *time.*

Proof. Let L_{runs} be a mapping defined such that for every $H \in [0\,..\,\sigma)^{\leq\tau}$ satisfying $\mathsf{R}^-_H(\tau, T) \neq \emptyset$, L_{runs} maps X to the pair of integers (x, y) defined by $x = \sum_{H' \in [0\,..\,\sigma)^+ : H' \prec H} |\mathsf{R}'^-_{H'}(\tau, T)|$ and $y = x + |\mathsf{R}'^-_H(\tau, T)|$. Denote $q_{\max} = 2n/\tau$.

Components The structure consists of the following components:

1. The components of the indexes (CSA, CST, and pattern matching index) from Sections 5.3.2, 6.3.2, 7.3.1 in [KK23a]. All these structures need $\mathcal{O}(n/\log_\sigma n)$ space.
2. The lookup table L_{runs}. When accessing L_{runs}, each $H \in [0\,..\,\sigma)^{\leq\tau}$ is converted to the integer $\text{int}(\tau, \sigma, H)$ (Definition 7.6). By $\text{int}(\tau, \sigma, H) \in [0\,..\,\sigma^{2\tau})$, this component needs $\mathcal{O}(\sigma^{2\tau}) = \mathcal{O}(n^{2\mu}) = \mathcal{O}(n^{1/6}) = \mathcal{O}(n/\log_\sigma n)$ space.
3. The plain representations of arrays $A_{\text{pos}}[1\,..\,q]$ and $A_{\text{len}}[1\,..\,q]$ augmented with the data structure from Theorem 2.16. Observe that we can use Theorem 2.16 since, by Lemma 7.40, it holds $q \leq q_{\max}$, $\max_{i=1}^{q} A_{\text{pos}}[i] \leq n = \mathcal{O}(q_{\max}\log q_{\max})$, and $\sum_{i=1}^{q} A_{\text{len}}[i] \leq 2n = \mathcal{O}(q_{\max}\log q_{\max})$. The arrays need $\mathcal{O}(q) = \mathcal{O}(\frac{n}{\tau}) = \mathcal{O}(n/\log_\sigma n)$ space, and the structure from Theorem 2.16 needs $\mathcal{O}(q_{\max}) = \mathcal{O}(n/\log_\sigma n)$ space.
4. The array $A_{\text{len}}[1\,..\,q]$ augmented with the data structure from Corollary 2.14. By the above analysis, we have $\sum_{i=1}^{q} A_{\text{len}}[i] = \mathcal{O}(q_{\max}\log q_{\max})$, where $q \leq q_{\max} = \mathcal{O}(n/\tau)$. Thus, this component needs $\mathcal{O}(q_{\max}) = \mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n)$ space.

Implementation of queries The queries are answered as follows:

1. Given any $i \in [1\,..\,n]$ such that $\text{SA}_T[i] \in \mathsf{R}(\tau, T)$, we can compute $\text{SA}_T[i]$ in $\mathcal{O}(\log\log n)$ time using the first component of the above structure ([KK23a, Proposition 5.14]).
2. The queries are standard navigational queries for periodic positions; see [KK23a, Proposition 5.7]. The indices x, y, z are computed with the help of rank/select queries on the bitvector $\text{ExpBlockBitvector}^-(\tau, T)$ (Definition 7.47) and its symmetric version (adapted for positions in $\mathsf{R}^+(\tau, T)$); see the proof of [KK23a, Proposition 5.9].
3. The first query is again a simple application of the first component (see the proof of [KK23a, Proposition 5.10] for a similar computation). The second query is answered using the lookup table L_{runs} (defined above).
4. Given the packed representation of any τ-periodic pattern $P \in [0\,..\,\sigma)^+$, we compute the pair $(\text{RangeBeg}(P, T), \text{RangeEnd}(P, T))$ in $\mathcal{O}(\log\log n + |P|/\log_\sigma n)$ time using the first component of the structure, as described in [KK23a, Proposition 6.12].

5. Given $b, e \in [0\,..\,q]$ and $v \geq 0$ such that $\mathsf{three\text{-}sided\text{-}rmq}_{A_{\mathrm{pos}},A_{\mathrm{len}}}(b, e, v) \neq \infty$, we compute the value $A_{\mathrm{pos}}[\mathsf{three\text{-}sided\text{-}rmq}_{A_{\mathrm{pos}},A_{\mathrm{len}}}(b, e, v)]$ using Theorem 2.16 in $\mathcal{O}(\log\log q) = \mathcal{O}(\log\log n)$ time.
6. Given any $b, e \in [0\,..\,q]$ and $v \geq 0$, we compute the set $\mathcal{I} = \{i \in (b\,..\,e] : e^{\mathrm{full}}(a_i, \tau, T) - a_i \geq v\}$ using Corollary 2.14 (applied to array A_{len}) in $\mathcal{O}(1 + |\mathcal{I}|)$ time.

Construction algorithm The components of the structure are constructed as follows:

1. The first component is constructed by combining Propositions 5.3, 5.15, 6.3, 6.13, 7.7, and 7.23 from [KK23a]. In total, this takes $\mathcal{O}(n/\log_\sigma n)$ time.
2. The lookup table is constructed in $\mathcal{O}(n/\log_\sigma n)$ time similarly as the closely related lookup table L_{runs} in the proof of [KK23a, Proposition 5.15].
3. The third component is constructed as follows:
   (a) Using Proposition 7.52(3), compute the sequence $(a_i)_{i\in[1\,..\,q]} = \mathrm{RunsLexSorted}^-(\tau, T)$ (Definition 7.44) in $\mathcal{O}(n/\log_\sigma n)$ time.
   (b) With the help of the first component, we can then compute arrays $A_{\mathrm{pos}}[1\,..\,q]$ and $A_{\mathrm{len}}[1\,..\,q]$ in $\mathcal{O}(q) = \mathcal{O}(n/\log_\sigma n)$ time.
   (c) Apply Theorem 2.16 to arrays $A_{\mathrm{pos}}[1\,..\,q]$ and $A_{\mathrm{len}}[1\,..\,q]$ in $\mathcal{O}(q_{\max}) = \mathcal{O}(n/\log_\sigma n)$ time.

   In total, the construction of the third component takes $\mathcal{O}(n/\log_\sigma n)$ time.
4. First, compute the array $A_{\mathrm{len}}[1\,..\,q]$ as described above. Then, apply Corollary 2.14 to array $A_{\mathrm{len}}[1\,..\,q]$ in $\mathcal{O}(q_{\max}) = \mathcal{O}(n/\log_\sigma n)$ time.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n)$ time. □

Combinatorial Properties

Lemma 7.54. *Let $T \in \Sigma^n$ and $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$. Let $i_1, i_2 \in [1\,..\,n]$ be such that $i_1 \leq i_2$ and $\{\mathrm{SA}_T[i]\}_{i\in[i_1\,..\,i_2]} \subseteq \mathsf{R}^-_{s,k,H}(\tau, T)$ for some $s \in \mathbb{Z}_{\geq 0}$, $k \in \mathbb{Z}_{>0}$, and $H \in \Sigma^+$. Denote $p = |H|$, $\delta_{\mathrm{text}} = s + kp$, and $(a_j)_{j\in[1\,..\,q]} = \mathrm{RunsLexSorted}^-(\tau, T)$ (Definition 7.44). Let $j_1, j_2 \in [1\,..\,q]$ be such that for $k \in \{1, 2\}$, $e^{\mathrm{full}}(\mathrm{SA}_T[i_k], \tau, T) = e^{\mathrm{full}}(a_{j_k}, \tau, T)$. Then, it holds*

$$\{\mathrm{SA}_T[i]\}_{i\in[i_1\,..\,i_2]} = \{e^{\mathrm{full}}(a_j, \tau, T) - \delta_{\mathrm{text}} : j \in [j_1\,..\,j_2] \text{ and } e^{\mathrm{full}}(a_j, \tau, T) - a_j \geq \delta_{\mathrm{text}}\}.$$

Proof. Denote $A = \{e^{\mathrm{full}}(a_j, \tau, T) - \delta_{\mathrm{text}} : j \in [j_1\,..\,j_2] \text{ and } e^{\mathrm{full}}(a_j, \tau, T) - a_j \geq \delta_{\mathrm{text}}\}$. Before proving the claim, i.e., that $\{\mathrm{SA}_T[i]\}_{i\in[i_1\,..\,i_2]} = A$, we first establish some auxiliary properties:

- First, we show that for $k \in \{1, 2\}$, it holds $a_{j_k} \leq \mathrm{SA}_T[i_k]$ and $[a_{j_k}\,..\,\mathrm{SA}_T[i_k]] \subseteq \mathsf{R}^-(\tau, T)$. Let $k \in \{1, 2\}$ and $t \in \mathsf{R}'^-(\tau, T)$ be such that $[t\,..\,\mathrm{SA}_T[i_k]] \subseteq \mathsf{R}^-(\tau, T)$. If $t \neq a_{j_k}$, then by Lemma 7.41, we have $e^{\mathrm{full}}(t, \tau, T) \neq e^{\mathrm{full}}(a_{j_k}, \tau, T)$. On the other hand, by Lemma 7.35, $e^{\mathrm{full}}(t, \tau, T) = e^{\mathrm{full}}(\mathrm{SA}_T[i_k], \tau, T)$. Thus, we obtain $e^{\mathrm{full}}(\mathrm{SA}_T[i_k], \tau, T) \neq e^{\mathrm{full}}(a_{j_k}, \tau, T)$, which contradicts the assumption the claim. We thus must have $t = a_{j_k}$, i.e., $a_{j_k} \leq \mathrm{SA}_T[i_k]$ and
$$[a_{j_k}\,..\,\mathrm{SA}_T[i_k]] \subseteq \mathsf{R}^-(\tau, T).$$
- Second, observe that by the above property and Lemma 7.35, for $k \in \{1, 2\}$ it holds:
  - $\mathrm{root}(a_{j_k}, \tau, T) = \mathrm{root}(\mathrm{SA}_T[i_k], \tau, T)$,
  - $e(a_{j_1}, \tau, T) = e(\mathrm{SA}_T[i_1], \tau, T)$,
  - $e^{\mathrm{full}}(a_{j_1}, \tau, T) = e^{\mathrm{full}}(\mathrm{SA}_T[i_1], \tau, T)$.
- Next, observe that for every $i \in [i_1\,..\,i_2]$, the assumption $\mathrm{SA}_T[i] \in \mathsf{R}^-_{s,k,H}(\tau, T)$ implies that
$$e^{\mathrm{full}}(\mathrm{SA}_T[i], \tau, T) = \mathrm{SA}_T[i] + s + kp = \mathrm{SA}_T[i] + \delta_{\mathrm{text}}.$$
- Next, we prove that it holds $\mathrm{tail}(a_{j_1}, \tau, T) + \delta_{\mathrm{text}} \geq 3\tau - 1$. First, observe that, by definition, for every $y \in \mathsf{R}(\tau, T)$, it holds $e(y, \tau, T) - y \geq 3\tau - 1$. In particular, $e(\mathrm{SA}_T[i_1], \tau, T) - \mathrm{SA}_T[i_1] \geq 3\tau - 1$. By the above, we can rewrite this as $e(a_{j_1}, \tau, T) - \mathrm{SA}_T[i_1] \geq 3\tau - 1$. Substituting $\mathrm{SA}_T[i_1] = e^{\mathrm{full}}(\mathrm{SA}_T[i_1], \tau, T) - \delta_{\mathrm{text}} = e^{\mathrm{full}}(a_{j_1}, \tau, T) - \delta_{\mathrm{text}}$ (see above), we thus obtain $e(a_{j_1}, \tau, T) - (e^{\mathrm{full}}(a_{j_1}, \tau, T) - \delta_{\mathrm{text}}) \geq 3\tau - 1$, which simplifies to
$$\mathrm{tail}(a_{j_1}, \tau, T) + \delta_{\mathrm{text}} \geq 3\tau - 1.$$

- Finally, we prove that $\mathrm{tail}(\mathrm{SA}_T[i_1],\tau,T) \le \mathrm{tail}(\mathrm{SA}_T[i_1+1],\tau,T) \le \cdots \le \mathrm{tail}(\mathrm{SA}_T[i_2],\tau,T)$. By Lemma 7.38(3), $e(\mathrm{SA}_T[i_1],\tau,T)-\mathrm{SA}_T[i_1] \le e(\mathrm{SA}_T[i_1+1],\tau,T)-\mathrm{SA}_T[i_1+1]) \le \cdots \le e(\mathrm{SA}_T[i_2],\tau,T)-\mathrm{SA}_T[i_2]$. On the other hand, note that for every $i \in [i_1\,..\,i_2]$, $\mathrm{SA}_T[i] \in \mathsf{R}^-_{s,k,H}(\tau,T)$ implies that $\mathrm{tail}(\mathrm{SA}_T[i],\tau,T) = (e(\mathrm{SA}_T[i],\tau,T)-\mathrm{SA}_T[i])-\delta_{\mathrm{text}}$. This yields

$$\mathrm{tail}(\mathrm{SA}_T[i_1],\tau,T) \le \mathrm{tail}(\mathrm{SA}_T[i_1+1],\tau,T) \le \cdots \le \mathrm{tail}(\mathrm{SA}_T[i_2],\tau,T).$$

We now return to the proof of the main claim. First, we prove that $\{\mathrm{SA}_T[i]\}_{i\in[i_1\,..\,i_2]} \subseteq A$. Let $x \in \{\mathrm{SA}_T[i]\}_{i\in[i_1\,..\,i_2]}$, and let $i \in [i_1\,..\,i_2]$ be such that $x = \mathrm{SA}_T[i]$. Let $x' \in \mathsf{R}'^-(\tau,T)$ be such that $[x'\,..\,x] \subseteq \mathsf{R}^-(\tau,T)$. Since $(a_j)_{j\in[1\,..\,q]}$ contains all elements of $\mathsf{R}'^-(\tau,T)$ (see Definition 7.44), there exists $j \in [1\,..\,q]$ such that $x' = a_j$. We will now prove that $j \in [j_1\,..\,j_2]$. We proceed in two steps:

1. Above we observed that $\mathrm{root}(a_{j_1},\tau,T) = \mathrm{root}(\mathrm{SA}_T[i_1],\tau,T) = H = \mathrm{root}(\mathrm{SA}_T[i_2],\tau,T) = \mathrm{root}(a_{j_2},\tau,T)$. On the other hand, $[a_j\,..\,\mathrm{SA}_T[i]] \subseteq \mathsf{R}^-(\tau,T)$ and $\mathrm{SA}_T[i] \in \mathsf{R}^-_{s,k,H}(\tau,T)$ imply by Lemma 7.35 that $\mathrm{root}(a_j,\tau,T) = \mathrm{root}(\mathrm{SA}_T[i],\tau,T) = H$. Thus,

$$\mathrm{root}(a_{j_1},\tau,T) = \mathrm{root}(a_j,\tau,T) = \mathrm{root}(a_{j_2},\tau,T) = H.$$

2. Second, we prove that $T[e^{\mathrm{full}}(a_{j_1},\tau,T)\,..\,n] \preceq T[e^{\mathrm{full}}(a_j,\tau,T)\,..\,n] \preceq T[e^{\mathrm{full}}(a_{j_2},\tau,T)\,..\,n]$. To this end, first observe that by $\mathrm{SA}_T[i_1],\mathrm{SA}_T[i],\mathrm{SA}_T[i_2] \in \mathsf{R}^-_{s,k,H}(\tau,T)$, it follows that suffixes $T[\mathrm{SA}_T[i_1]\,..\,n]$, $T[\mathrm{SA}_T[i]\,..\,n]$, and $T[\mathrm{SA}_T[i_2]\,..\,n]$ share a common prefix of length $s+kp = \delta_{\mathrm{text}}$. Thus, by $i_1 \le i \le i_2$ and the definition of the suffix array, we have

$$T[\mathrm{SA}_T[i_1]+\delta_{\mathrm{text}}\,..\,n] \preceq T[\mathrm{SA}_T[i]+\delta_{\mathrm{text}}\,..\,n] \preceq T[\mathrm{SA}_T[i_2]+\delta_{\mathrm{text}}\,..\,n].$$

   On the other hand, note that $\mathrm{SA}_T[i_1]+\delta_{\mathrm{text}} = e^{\mathrm{full}}(\mathrm{SA}_T[i_1],\tau,T)$, and similarly, $\mathrm{SA}_T[i]+\delta_{\mathrm{text}} = e^{\mathrm{full}}(\mathrm{SA}_T[i],\tau,T)$, and $\mathrm{SA}_T[i_2]+\delta_{\mathrm{text}} = e^{\mathrm{full}}(\mathrm{SA}_T[i_2],\tau,T)$. Combining this with the assumptions $e^{\mathrm{full}}(\mathrm{SA}_T[i_k],\tau,T) = e^{\mathrm{full}}(a_{j_k},\tau,T)$ $(k \in \{1,2\})$ and the fact that $e^{\mathrm{full}}(\mathrm{SA}_T[i],\tau,T) = e^{\mathrm{full}}(a_j,\tau,T)$ (following from Lemma 7.35), we obtain that $\mathrm{SA}_T[i_k]+\delta_{\mathrm{text}} = e^{\mathrm{full}}(a_{j_k},\tau,T)$ $(k \in \{1,2\})$ and $\mathrm{SA}_T[i]+\delta_{\mathrm{text}} = e^{\mathrm{full}}(a_j,\tau,T)$. Thus, the above inequalities can be equivalent rewritten as $T[e^{\mathrm{full}}(a_{j_1},\tau,T)\,..\,n] \preceq T[e^{\mathrm{full}}(a_j,\tau,T)\,..\,n] \preceq T[e^{\mathrm{full}}(a_{j_2},\tau,T)\,..\,n]$.

Combining the above properties with the definition of $(a_j)_{j\in[1\,..\,q]}$ (see Definition 7.44), we immediately obtain that $j_1 \le j \le j_2$. It remains to recall that by $a_j \le \mathrm{SA}_T[i]$, $e^{\mathrm{full}}(\mathrm{SA}_T[i],\tau,T)-\mathrm{SA}_T[i] = \delta_{\mathrm{text}}$, it follows that $e^{\mathrm{full}}(a_j,\tau,T)-a_j \ge e^{\mathrm{full}}(a_j,\tau,T)-\mathrm{SA}_T[i] = e^{\mathrm{full}}(\mathrm{SA}_T[i],\tau,T)-\mathrm{SA}_T[i] = \delta_{\mathrm{text}}$. Thus, by $e^{\mathrm{full}}(a_j,\tau,T)-\delta_{\mathrm{text}} = e^{\mathrm{full}}(\mathrm{SA}_T[i],\tau,T)-\delta_{\mathrm{text}} = \mathrm{SA}_T[i] = x$, we obtain $x \in A$. This concludes the proof of the inclusion $\{\mathrm{SA}_T[i]\}_{i\in[i_1\,..\,i_2]} \subseteq A$.

We now prove that $A \subseteq \{\mathrm{SA}_T[i]\}_{i\in[i_1\,..\,i_2]}$. Let $x \in A$, and let $j \in [j_1\,..\,j_2]$ be such that $e^{\mathrm{full}}(a_j,\tau,T)-a_j \ge \delta_{\mathrm{text}}$ and $x = e^{\mathrm{full}}(a_j,\tau,T)-\delta_{\mathrm{text}}$. We proceed as follows:

1. In the first step, we prove that $a_j \le x$ and that it holds $[a_j\,..\,x] \subseteq \mathsf{R}(\tau,T)$. To this end, first note that by Lemma 7.36, the maximal block of positions from the set $\mathsf{R}(\tau,T)$ containing any $y \in \mathsf{R}'(\tau,T)$ is $[y\,..\,e(y,\tau,T)-(3\tau-1)]$. Thus, to show $[a_j\,..\,x] \subseteq \mathsf{R}(\tau,T)$, we need to prove that $a_j \le x$ and $x \le e(a_j,\tau,T)-(3\tau-1)$. Recall that $x = e^{\mathrm{full}}(a_j,\tau,T)-\delta_{\mathrm{text}}$, and that a_j satisfies $e^{\mathrm{full}}(a_j,\tau,T)-\delta_{\mathrm{text}} \ge a_j$. This immediately yields $x \ge a_j$. To show $x \le e(a_j,\tau,T)-(3\tau-1)$, recall that above we proved that $\mathrm{tail}(a_j,\tau,T) \ge \mathrm{tail}(a_{j_1},\tau,T)$, and $\mathrm{tail}(a_{j_1},\tau,T)+\delta_{\mathrm{text}} \ge 3\tau-1$. Putting these inequalities together with $x = e^{\mathrm{full}}(a_j,\tau,T)-\delta_{\mathrm{text}}$, we obtain $e(a_j,\tau,T)-x = e(a_j,\tau,T)-(e^{\mathrm{full}}(a_j,\tau,T)-\delta_{\mathrm{text}}) = \mathrm{tail}(a_j,\tau,T)+\delta_{\mathrm{text}} \ge \mathrm{tail}(a_{j_1},\tau,T)+\delta_{\mathrm{text}} \ge 3\tau-1$ or equivalently, $x \le e(a_j,\tau,T)-(3\tau-1)$. Thus, we indeed obtain

$$[a_j\,..\,x] \subseteq [a_j\,..\,e(a_j,\tau,T)-(3\tau-1)] \subseteq \mathsf{R}(\tau,T).$$

2. In this step, we prove that $x \in \mathsf{R}^-_{s,k,H}(\tau,T)$. Above we showed that $a_j \le x$ and $[a_j\,..\,x] \subseteq \mathsf{R}(\tau,T)$. By Lemma 7.35, this implies that $e^{\mathrm{full}}(x,\tau,T) = e^{\mathrm{full}}(a_j,\tau,T)$, $\mathrm{root}(x,\tau,T) = \mathrm{root}(a_j,\tau,T) = H$, and $\mathrm{type}(x,\tau,T) = \mathrm{type}(a_j,\tau,T) = -1$. Moreover, we then have $\mathrm{head}(x,\tau,T) = (e^{\mathrm{full}}(x,\tau,T)-x) \bmod |H| = (e^{\mathrm{full}}(a_j,\tau,T)-x) \bmod |H| = \delta_{\mathrm{text}} = s$ and $\exp(x,\tau,T) = ((e^{\mathrm{full}}(x,\tau,T)-x)-\mathrm{head}(x,\tau,T))/|H| = (\delta_{\mathrm{text}}-s)/|H| = k$. We thus obtain

$$x \in \mathsf{R}^-_{s,k,H}(\tau,T).$$

3. In the next step, we show that $T[\mathrm{SA}_T[i_1]\,..\,n] \preceq T[x\,..\,n] \preceq T[\mathrm{SA}_T[i_2]\,..\,n]$. Let us focus on showing $T[\mathrm{SA}_T[i_1]\,..\,n] \preceq T[x\,..\,n]$. Observe that by $x, \mathrm{SA}_T[i_1] \in \mathsf{R}^-_{s,k,H}(\tau, T)$, we have $e^{\mathrm{full}}(x,\tau,T) - x = e^{\mathrm{full}}(\mathrm{SA}_T[i_1],\tau,T) - \mathrm{SA}_T[i_1]$ and
$$T[\mathrm{SA}_T[i_1]\,..\,e^{\mathrm{full}}(\mathrm{SA}_T[i_1],\tau,T)) = T[x\,..\,e^{\mathrm{full}}(x,\tau,T)).$$
On the other hand, observe that the assumption $j_1 \le j$ combined with the above observation $\mathrm{root}(a_{j_1},\tau,T) = \mathrm{root}(a_j,\tau,T)$ and the definition of the sequence $(a_j)_{j\in[1\,..\,q]}$ (see Definition 7.44), imply that $T[e^{\mathrm{full}}(a_{j_1},\tau,T)\,..\,n] \preceq T[e^{\mathrm{full}}(a_j,\tau,T)\,..\,n]$. Recalling that we have $e^{\mathrm{full}}(a_{j_1},\tau,T) = e^{\mathrm{full}}(\mathrm{SA}_T[i_1],\tau,T)$ and $e^{\mathrm{full}}(a_j,\tau,T) = e^{\mathrm{full}}(x,\tau,T)$, we can equivalently write this as
$$T[e^{\mathrm{full}}(\mathrm{SA}_T[i_1],\tau,T)\,..\,n] \preceq T[e^{\mathrm{full}}(x,\tau,T)\,..\,n].$$
Putting the above two observations together yields $T[\mathrm{SA}_T[i_1]\,..\,n] \preceq T[x\,..\,n]$. The proof of $T[x\,..\,n] \preceq T[\mathrm{SA}_T[i_2]\,..\,n]$ is analogous.
4. We are now ready to finish the proof. By the previous step and the definition of the suffix array, there exists $i \in [i_1\,..\,i_2]$ such that $x = \mathrm{SA}_T[i]$. Thus, $x \in \{\mathrm{SA}_T[i]\}_{i\in[i_1\,..\,i_2]}$. This concludes the proof of the inclusion $A \subseteq \{\mathrm{SA}_T[i]\}_{i\in[i_1\,..\,i_2]}$. □

Algorithms

Proposition 7.55. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \le \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \ge 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that given any $j \in \mathsf{R}^-_{s,H}(\tau,T)$ (where $H \in [0\,..\,\sigma)^+$ and $s \in [0\,..\,|H|)$) and some $k > k_{\min}$, where $k_{\min} = \lceil \frac{3\tau-1-s}{|H|} \rceil - 1$, returns the set*
$$\mathsf{R}^-_{s,k,H}(\tau,T)$$
in $\mathcal{O}(1 + |\mathsf{R}^-_{s,k,H}(\tau,T)|)$ time.

Proof. Let $q = |\mathsf{R}'^-(\tau,T)|$ and $(b_i)_{i\in[1\,..\,q]} = \mathrm{RunsLengthSorted}^-(\tau,T)$ (Definition 7.46). Let $A_{\mathrm{runs}}[1\,..\,q]$ be an array defined by $A_{\mathrm{runs}}[i] = b_i$.

Components The data structure consists of the following components:

1. The data structure from Proposition 7.53 for text T. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
2. The array $A_{\mathrm{runs}}[1\,..\,q]$ in plain form. By Lemma 7.40, it needs $\mathcal{O}(n/\tau) = \mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n)$ space.

Implementation of queries The queries are answered as follows. Let $j \in \mathsf{R}^-_{s,H}(\tau,T)$, where $H \in [0\,..\,\sigma)^+$, $p = |H|$, and $s \in [0\,..\,p)$. Denote $k_{\min} = \lceil \frac{3\tau-1-s}{p} \rceil - 1$. Given j and any $k > k_{\min}$, we compute $\mathsf{R}^-_{s,k,H}(\tau,T)$ as follows:

1. Using Proposition 7.53(3), in $\mathcal{O}(1)$ time compute integers $x, z \in [0\,..\,q]$ such that it holds $x = \sum_{H'\in[0\,..\,\sigma)^+ : H' \prec H} |\mathsf{R}'^-_{H'}(\tau,T)|$ and $z = x + |\mathsf{R}'^-_H(\tau,T)|$. Note that we then have $\{b_i\}_{i\in(x\,..\,z]} = \mathsf{R}'^-_H(\tau,T)$.
2. Denote $\delta_{\mathrm{text}} = s + kp$. In the next step, we compute an index $y \in [x\,..\,z]$ such that, for every $i \in (x\,..\,z]$, $e^{\mathrm{full}}(b_i,\tau,T) - b_i \ge \delta_{\mathrm{text}}$ holds if and only if $i \in (y\,..\,z]$. To this end, we scan the array $A_{\mathrm{runs}}(x\,..\,z]$ right-to-left. For each $i \in (x\,..\,z]$, in $\mathcal{O}(1)$ time we first compute $e = e^{\mathrm{full}}(A_{\mathrm{runs}}[i],\tau,T)$ using Proposition 7.53(2). We then check if $e - A_{\mathrm{runs}}[i] \ge \delta_{\mathrm{text}}$. Once we reach $i = x$, or we find $i \in (x\,..\,z]$ such that $e - A_{\mathrm{runs}}[i] < \delta_{\mathrm{text}}$, we stop the scan. The correctness of this algorithm follows by Definition 7.46, i.e., we use that for all $i_1, i_2 \in (x\,..\,z]$ such that $i_1 < i_2$, it holds $e^{\mathrm{full}}(b_{i_1},\tau,T) - b_{i_1} \le e^{\mathrm{full}}(b_{i_2},\tau,T) - b_{i_2}$. The computation of y takes $\mathcal{O}(1 + (z-y))$ time. By Lemma 7.42, it holds
$$\mathsf{R}^-_{s,k,H}(\tau,T) = \{e^{\mathrm{full}}(b_i,\tau,T) - \delta_{\mathrm{text}} : i \in (y\,..\,z]\}.$$
By Lemma 7.41, $z - y = |\mathsf{R}^-_{s,k,H}(\tau,T)|$, and hence this step takes $\mathcal{O}(1 + |\mathsf{R}^-_{s,k,H}(\tau,T)|)$ time.

3. We perform one more scan of $A_{\text{runs}}(y\,..\,z]$, and with the help of Proposition 7.53(2), we compute and return the set $\mathcal{P} = \{e^{\text{full}}(A_{\text{runs}}[i], \tau, T) - \delta_{\text{text}} : i \in (y\,..\,z]\}$. This takes $\mathcal{O}(1 + (z - y)) = \mathcal{O}(1 + |\mathsf{R}^-_{s,k,H}(\tau, T)|)$ time, and by the above discussion, we have $\mathcal{P} = \mathsf{R}^-_{s,k,H}(\tau, T)$.

In total, we spend $\mathcal{O}(1 + |\mathsf{R}^-_{s,k,H}(\tau, T)|)$ time.

Construction algorithm The components of the structure are constructed as follows:

1. The construction of the first component takes $\mathcal{O}(n/\log_\sigma n)$ time using Proposition 7.53.
2. To construct the second component, we proceed as follows:
   (a) First, using Proposition 7.52(3), compute the sequence $(a_i)_{i\in[1\,..\,q]} = \text{RunsLexSorted}^-(\tau, T)$ (Definition 7.44) in $\mathcal{O}(n/\log_\sigma n)$ time.
   (b) With the help of Proposition 7.53(2), create an array $A_{\text{sort}}[1\,..\,q]$ of triples defined by $A_{\text{sort}}[i] = (\text{int}(\tau, \sigma, \text{root}(a_i, \tau, T)), e^{\text{full}}(a_i, \tau, T) - a_i, a_i)$ (see Definition 7.6). This takes $\mathcal{O}(q) = \mathcal{O}(n/\log_\sigma n)$ time.
   (c) We sort the array $A_{\text{sort}}[1\,..\,q]$ lexicographically. Since the first coordinate is an integer smaller than $\sigma^{2\tau} = \mathcal{O}(n^{1/6})$, and the other two coordinates are numbers in $[1\,..\,n]$, it suffices to use a 5-round radix sort to achieve $\mathcal{O}(q + \sqrt{n}) = \mathcal{O}(n/\log_\sigma n)$ time. The last coordinate of the resulting array contains the sequence $(b_i)_{i\in[1\,..\,q]} = \text{RunsLengthSorted}^-(\tau, T)$ (Definition 7.46), i.e., the elements of the array $A_{\text{runs}}[1\,..\,q]$. The correctness of this step follows by Lemma 7.7 and Definition 7.46.

   In total, construction of the second component takes $\mathcal{O}(n/\log_\sigma n)$ time.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n)$ time. □

Proposition 7.56. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \le \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \ge 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that given any $b, e \in [0\,..\,n]$ such that $b < e$ and $\{\text{SA}_T[i]\}_{i\in(b\,..\,e]} \subseteq \mathsf{R}^-_{s,k,H}(\tau, T)$ holds for some $s \in \mathbb{Z}_{\ge 0}$, $H \in [0\,..\,\sigma)^+$, and $k \in \mathbb{Z}_{>0}$, returns the set*

$$\{\text{SA}_T[i]\}_{i\in(b\,..\,e]}$$

in $\mathcal{O}((e - b) + \log\log n)$ time.

Proof. Let $q = |\mathsf{R}'^-(\tau, T)|$ and $(a_i)_{i\in[1\,..\,q]} = \text{RunsLexSorted}^-(\tau, T)$ (Definition 7.44). Let $A_{\text{runs}}[1\,..\,q]$ be an array defined by $A_{\text{runs}}[i] = a_i$.

Components The data structure consists of the following components:

1. The data structure from Proposition 7.53 for text T. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
2. The array $A_{\text{runs}}[1\,..\,q]$ in plain form. By Lemma 7.40, it needs $\mathcal{O}(n/\tau) = \mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n)$ space.

Implementation of queries The queries are answered as follows. Let $b, e \in [0\,..\,n]$ be such that $b < e$ and $\{\text{SA}_T[i]\}_{i\in(b\,..\,e]} \subseteq \mathsf{R}^-_{s,k,H}(\tau, T)$ for some $s \in \mathbb{Z}_{\ge 0}$, $H \in [0\,..\,\sigma)^+$, and $k \in \mathbb{Z}_{>0}$. Given b, e, we compute the set $\{\text{SA}_T[i]\}_{i\in(b\,..\,e]}$ as follows:

1. Denote $i_1 = b + 1$ and $i_2 = e$. Using Proposition 7.53(1), in $\mathcal{O}(\log\log n)$ time compute $p_k = \text{SA}_T[i_k]$ for $k \in \{1, 2\}$.
2. Using Proposition 7.53(2), in $\mathcal{O}(1)$ time compute $s = \text{head}(p_1, \tau, T)$, $p = |\text{root}(p_1, \tau, T)|$, and $k = \exp(p_1, \tau, T)$. We then set $\delta_{\text{text}} = s + kp$.
3. Using Proposition 7.53(3), in $\mathcal{O}(1)$ time compute $j_k \in [1\,..\,q]$ such that $e^{\text{full}}(p_k, \tau, T) = e^{\text{full}}(a_{j_k}, \tau, T)$ for $k \in \{1, 2\}$.
4. Using Proposition 7.53(6), compute the set $\mathcal{J} = \{j \in [j_1\,..\,j_2] : e^{\text{full}}(a_j, \tau, T) - a_j \ge \delta_{\text{text}}\}$ in $\mathcal{O}(1 + |\mathcal{J}|)$ time. Observe now that by Lemma 7.54, it holds

$$\{\text{SA}_T[i]\}_{i\in(b\,..\,e]} = \{\text{SA}_T[i]\}_{i\in[i_1\,..\,i_2]} = \{e^{\text{full}}(a_j, \tau, T) - \delta_{\text{text}} : j \in \mathcal{J}\}.$$

   By Lemma 7.41, $|\mathcal{J}| = i_2 - i_1 + 1 = e - b$. Hence, this step takes $\mathcal{O}(1 + |\mathcal{J}|) = \mathcal{O}(1 + (e - b))$ time.

5. Using Proposition 7.53(2), compute and return the set $\{e^{\text{full}}(A_{\text{runs}}[j], \tau, T) - \delta_{\text{text}} : j \in \mathcal{J}\}$ in $\mathcal{O}(1+|\mathcal{J}|) = \mathcal{O}(1 + (e-b))$ time. By the above formula, this is precisely $\{\text{SA}_T[i]\}_{i\in(b\,..\,e]}$.

In total, we spend $\mathcal{O}((e-b) + \log\log n)$ time.

Construction algorithm The components of the structure are constructed as follows:

1. The construction of the first component takes $\mathcal{O}(n/\log_\sigma n)$ time using Proposition 7.53.
2. The second component is constructed using Proposition 7.52(3) in $\mathcal{O}(n/\log_\sigma n)$ time.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n)$ time. □

Proposition 7.57. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \le \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \ge 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that given any $b, e \in [0\,..\,n]$ such that $b < e$ and $\{\text{SA}_T[i]\}_{i\in(b\,..\,e]} \subseteq \mathsf{R}^-_{s,H}(\tau, T)$ holds for some $s \in \mathbb{Z}_{\ge 0}$ and $H \in [0\,..\,\sigma)^+$, returns the set*

$$\{\text{SA}_T[i]\}_{i\in(b\,..\,e]}$$

in $\mathcal{O}((e-b) + \log\log n)$ time.

Proof. We use the following definitions. Denote $(a_i)_{i\in[1\,..\,q]} = \text{RunsLexSorted}^-(\tau, T)$ (Definition 7.44). Let also $B_{\text{exp}} = \text{ExpBlockBitvector}^-(\tau, T)$ (Definition 7.47).

Components The data structure consists of the following components:

1. The data structure from Proposition 7.53 for text T. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
2. The data structure from Proposition 7.55 for text T. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
3. The data structure from Proposition 7.56 for text T. It also needs $\mathcal{O}(n/\log_\sigma n)$ space.
4. The bitvector B_{exp} augmented with the support for $\mathcal{O}(1)$-time rank and select queries using Theorem 2.7. The bitvector together with the augmentation of Theorem 2.7 needs $\mathcal{O}(n/\log n) = \mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n)$ space.

Implementation of queries The queries are answered as follows. Let $b, e \in [0\,..\,n]$ be such that $b < e$ and $\{\text{SA}_T[i]\}_{i\in(b\,..\,e]} \subseteq \mathsf{R}^-_{s,H}(\tau, T)$ holds for some $s \in \mathbb{Z}_{\ge 0}$ and $H \in [0\,..\,\sigma)^+$. Given b, e, we compute the set $\{\text{SA}_T[i]\}_{i\in(b\,..\,e]}$ as follows.

1. In $\mathcal{O}(1)$ time compute $x = \mathsf{select}_{B_{\text{exp}}}(\mathsf{rank}_{B_{\text{exp}}}(b, 1) + 1, 1)$. If $x \ge e$, then by Definition 7.47, there exists $k \in \mathbb{Z}_{>0}$ such that $\{\text{SA}_T[i]\}_{i\in(b\,..\,e]} \subseteq \mathsf{R}^-_{s,k,H}(\tau, T)$. In this case, we compute $\{\text{SA}_T[i]\}_{i\in(b\,..\,e]}$ using Proposition 7.56 in $\mathcal{O}((e-b) + \log\log n)$ time, and this completes the query algorithm. Let us now assume that $x < e$.
2. In $\mathcal{O}(1)$ time, compute $y = \mathsf{select}_{B_{\text{exp}}}(\mathsf{rank}_{B_{\text{exp}}}(e-1, 1), 1)$. Note that then $b < x \le y < e$ and, by Definition 7.47, there exist $k_1, k_2 \in \mathbb{Z}_{>0}$ such that $k_1 < k_2$, and it holds:
   - $\{\text{SA}_T[i]\}_{i\in(b\,..\,x]} \subseteq \mathsf{R}^-_{s,k_1,H}(\tau, T)$,
   - $\{\text{SA}_T[i]\}_{i\in(x\,..\,y]} = \bigcup_{k\in(k_1\,..\,k_2)} \mathsf{R}^-_{s,k,H}(\tau, T)$,
   - $\{\text{SA}_T[i]\}_{i\in(y\,..\,e]} \subseteq \mathsf{R}^-_{s,k_2,H}(\tau, T)$.

   We enumerate the elements of each of the above sets as follows:
   - The set $\{\text{SA}_T[i]\}_{i\in(b\,..\,x]}$ is computed using Proposition 7.56 in $\mathcal{O}((x-b) + \log\log n)$ time.
   - Next, we check if $x = y$. If so, we skip this step. Let us thus assume that $x < y$. Then, it holds $k_1 + 1 < k_2$. Moreover, note that $k_1 + 1 > k_{\min}$, where $k_{\min} = \lceil \frac{3\tau-1-s}{|H|} \rceil - 1$. To see this, note that by $b < x$, it follows that $\mathsf{R}^-_{s,k_1,H}(\tau, T) \neq \emptyset$. For any element j of this set, it holds, by definition,

   $$k_1 = \exp(j, \tau, T) = \left\lfloor \tfrac{e(j,\tau,T)-j-s}{|H|} \right\rfloor \ge \left\lfloor \tfrac{3\tau-1-s}{|H|} \right\rfloor \ge \left\lceil \tfrac{3\tau-1-s}{|H|} \right\rceil - 1 = k_{\min}.$$

   Thus, $k_1 + 1 > k_{\min}$. To compute $\{\text{SA}_T[i]\}_{i\in(x\,..\,y]} = \bigcup_{k\in(k_1\,..\,k_2)} \mathsf{R}^-_{s,k,H}(\tau, T)$, we proceed as follows:

   (a) Using Proposition 7.53(1), in $\mathcal{O}(\log\log n)$ time compute $j_{\text{first}} = \text{SA}_T[x+1]$ and $j_{\text{last}} = \text{SA}_T[y]$.

(b) Using Proposition 7.53(2), in $\mathcal{O}(1)$ time compute integers $k_{\text{first}} = \exp(j_{\text{first}}, \tau, T)$ and $k_{\text{last}} = \exp(j_{\text{last}}, \tau, T)$. Observe that it holds $k_1 + 1 = k_{\text{first}}$ and $k_2 - 1 = k_{\text{last}}$.

(c) We apply Proposition 7.55 to position j_1 and $k = k_{\text{first}}, k_{\text{first}} + 1, \ldots, k_{\text{last}}$ to compute all elements of the set $\bigcup_{k \in (k_1 \,..\, k_2)} \mathsf{R}^-_{s,k,H}(\tau, T)$. In total, this takes $\mathcal{O}((k_2 - k_1) + (y - x))$. To simplify this complexity, note that by Lemma 7.43, it holds

$$|\mathsf{R}^-_{s,k_1+1,H}(\tau, T)| \geq \ldots \geq |\mathsf{R}^-_{s,k_2-1,H}(\tau, T)|.$$

Since by $j_{\text{last}} \in \mathsf{R}^-_{s,k_2-1,H}(\tau, T)$, we have $\mathsf{R}^-_{s,k_2-1,H}(\tau, T) \neq \emptyset$, we thus obtain that $k_2 - k_1 = \mathcal{O}(\sum_{k \in (k_1 \,..\, k_2)} |\mathsf{R}^-_{s,k,H}(\tau, T)|) = \mathcal{O}(y - x)$, we can thus simplify the above runtime to $\mathcal{O}(y - x)$.

In total, computing $\{\mathrm{SA}_T[i]\}_{i \in (x \,..\, y]} = \bigcup_{k \in (k_1 \,..\, k_2)} \mathsf{R}^-_{s,k,H}(\tau, T)$ takes $\mathcal{O}((y - x) + \log\log n)$ time.

- The set $\{\mathrm{SA}_T[i]\}_{i \in (y \,..\, e]}$ is computed using Proposition 7.56 in $\mathcal{O}((e - y) + \log\log n)$ time.

In total, we spend $\mathcal{O}((x - b) + (y - x) + (e - y) + \log\log n) = \mathcal{O}((e - b) + \log\log n)$ time.

In total, the query takes $\mathcal{O}((e - b) + \log\log n)$ time.

Construction algorithm The components of the structure are constructed as follows:

1. By Proposition 7.53, the construction of the first component takes $\mathcal{O}(n/\log_\sigma n)$ time.
2. The second component is constructed using Proposition 7.55 in $\mathcal{O}(n/\log_\sigma n)$ time.
3. The third component is constructed using Proposition 7.56 in $\mathcal{O}(n/\log_\sigma n)$ time.
4. To construct the last component, first, in $\mathcal{O}(n/\log_\sigma n)$ time we construct the bitvector $B_{\text{exp}} = \text{ExpBlockBitvector}^-(\tau, T)$ (Definition 7.47) using Proposition 7.52(1). We then augment B_{exp} using Theorem 2.7 in $\mathcal{O}(n/\log n) = \mathcal{O}(n/\log_\sigma n)$ time.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n)$ time. □

Proposition 7.58. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that given any $b, e \in [0\,..\,n]$ such that $b < e$ and $\{\mathrm{SA}_T[i]\}_{i \in (b \,..\, e]} \subseteq \mathsf{R}_{s,H}(\tau, T)$ for some $s \in \mathbb{Z}_{\geq 0}$ and $H \in [0\,..\,\sigma)^+$, returns the set*

$$\{\mathrm{SA}_T[i]\}_{i \in (b \,..\, e]}.$$

in $\mathcal{O}((e - b) + \log\log n)$ time.

Proof. **Components** The data structure consists of the following components:

1. The structure from Proposition 7.53 for text T. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
2. The structure from Proposition 7.57 for text T. It also needs $\mathcal{O}(n/\log_\sigma n)$ space.
3. The symmetric version of the structure from Proposition 7.57 adapted to positions in $\mathsf{R}^+(\tau, T)$ (see Definition 7.33). It also needs $\mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n)$ space.

Implementation of queries The queries are answered as follows. Let $b, e \in [0\,..\,n]$ be such that $b < e$ and $\{\mathrm{SA}_T[i]\}_{i \in (b \,..\, e]} \subseteq \mathsf{R}_{s,H}(\tau, T)$ for some $s \in \mathbb{Z}_{\geq 0}$ and $H \in [0\,..\,\sigma)^+$. Given b, e, we compute $\{\mathrm{SA}_T[i]\}_{i \in (b \,..\, e]}$ as follows:

1. Using Proposition 7.53(1), in $\mathcal{O}(\log\log n)$ time we compute $j = \mathrm{SA}_T[e]$.
2. Using Proposition 7.53(2), in $\mathcal{O}(1)$ time we then compute $x, y, z \in [0\,..\,n]$ such that $x \leq y \leq z$, $\{\mathrm{SA}_T[i]\}_{i \in (x \,..\, y]} = \mathsf{R}^-_{s,H}(\tau, T)$, and $\{\mathrm{SA}_T[i]\}_{i \in (y \,..\, z]} = \mathsf{R}^+_{s,H}(\tau, T)$, where $s = \text{head}(j, \tau, T)$ and $H = \text{root}(j, \tau, T)$. Note that then $x \leq b < e \leq z$.
3. In $\mathcal{O}(1)$ time initialize $\mathcal{M} = \emptyset$.
4. If $b < y$, then using Proposition 7.57, in $\mathcal{O}((\min(e, y) - b) + \log\log n)$ time compute

$$\{\mathrm{SA}_T[i]\}_{i \in (b \,..\, \min(e,y)]},$$

and add to $\mathcal{M}$. Note that $\{\mathrm{SA}_T[i]\}_{i \in (b \,..\, \min(e,y)]} \subseteq \mathsf{R}^-_{s,H}(\tau, T)$.

5. If $e > y$, then using the symmetric version of the data structure from Proposition 7.57, in $\mathcal{O}((e - \max(b, y)) + \log\log n)$ time compute

$$\{\mathrm{SA}_T[i]\}_{i\in(\max(b,y),e]},$$

and add to $\mathcal{M}$. Note that $\{\mathrm{SA}_T[i]\}_{i\in(\max(b,y)..e]} \subseteq \mathsf{R}^+_{s,H}(\tau, T)$.
6. Note that at this point, we have $\{\mathrm{SA}_T[i]\}_{i\in(b..e]} = \mathcal{M}$. Thus, we return $\mathcal{M}$ as the answer.

In total, answering the query takes $\mathcal{O}((e - b) + \log\log n)$ time.

Construction algorithm The components of the data structure are constructed as follows:

1. By Proposition 7.53, the construction of the first component takes $\mathcal{O}(n/\log_\sigma n)$ time.
2. By Proposition 7.57, the construction of the second component takes $\mathcal{O}(n/\log_\sigma n)$ time.
3. Since the third component is a symmetric version of the second component, its construction also takes $\mathcal{O}(n/\log_\sigma n)$ time.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n)$ time. □

Proposition 7.59. *Let $T \in [0..\sigma)^n$ be such that $2 \le \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1..n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \ge 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that answers the following queries:*

1. *Given the packed representation of τ-periodic (Definition 7.5) patterns $P_1, P_2 \in [0..\sigma)^+$ such that $\mathrm{lcp}(P_1, P_2) \ge 3\tau - 1$, computes the set $\mathcal{J} := \mathrm{LexRange}(P_1, P_2, T)$ (Definition 2.4) in $\mathcal{O}(\log\log n + |\mathcal{J}| + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.*
2. *Given the packed representation of a τ-periodic pattern $P \in [0..\sigma)^+$ satisfying $|P| \ge 3\tau - 1$, computes the set $\mathcal{J} := \mathrm{LexRange}(P, P'c^\infty, T)$ (where $P' = P[1..3\tau - 1]$ and $c = \sigma - 1$) in $\mathcal{O}(\log\log n + |\mathcal{J}| + |P|/\log_\sigma n)$ time.*

Proof. **Components** The data structure consists of the following components:

1. The data structure from Proposition 7.53 using $\mathcal{O}(n/\log_\sigma n)$ space.
2. The data structure from Proposition 7.58 using $\mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n)$ space.

Implementation of queries The queries are answered as follows:

1. Let $P_1, P_2 \in [0..\sigma)^+$ be τ-periodic patterns such that $\mathrm{lcp}(P_1, P_2) \ge 3\tau - 1$. Given the packed representation of P_1 and P_2, we compute $\mathrm{LexRange}(P_1, P_2, T)$ as follows:
   (a) Using Proposition 7.53(4), in $\mathcal{O}(\log\log n + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time we compute $b_k = \mathrm{RangeBeg}(P_k, T)$, where $k \in \{1, 2\}$. If $b_1 \ge b_2$, then $\mathrm{LexRange}(P_1, P_2, T) = \emptyset$ (see Remark 2.5). Let us now assume that $b_1 < b_2$. By Remark 2.5, we then have

$$\mathrm{LexRange}(P_1, P_2, T) = \{\mathrm{SA}_T[i]\}_{i\in(b_1..b_2]}.$$

   By $\mathrm{lcp}(P_1, P_2) \ge 3\tau - 1$, for every $j_1, j_2 \in \mathrm{LexRange}(P_1, P_2, T)$, it holds $\mathrm{LCE}_T(j_1, j_2) \ge 3\tau - 1$. Thus, by Lemma 7.37(2), there exists $s \in \mathbb{Z}_{\ge 0}$ and $H \in [0..\sigma)^+$ such that $\{\mathrm{SA}_T[i]\}_{i\in(b_1..b_2]} \subseteq \mathsf{R}_{s,H}(\tau, T)$.
   (b) Using Proposition 7.58, in $\mathcal{O}((b_2 - b_1) + \log\log n)$ time compute and return all elements of the set $\{\mathrm{SA}_T[i]\}_{i\in(b_1..b_2]}$.

   In total, we spend

$$\mathcal{O}(\log\log n + (b_2 - b_1) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n) =$$
$$\mathcal{O}(\log\log n + |\mathrm{LexRange}(P_1, P_2, T)| + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$$

   time.

2. Let us now consider a τ-periodic pattern $P \in [0\,..\,\sigma)^+$ that satisfies $|P| \geq 3\tau-1$, where $P' = P[1\,..\,3\tau-1]$ and $c = \sigma - 1$. Given the packed representation of P, we compute $\mathrm{LexRange}(P, P'c^\infty, T)$ similarly as above, letting $P_1 = P$ and $P_2 = P'c^\infty$. The main difference is that to compute $b_2 = \mathrm{RangeBeg}(P_2, T)$, we observe that $\mathrm{RangeBeg}(P'c^\infty, T) = \mathrm{RangeEnd}(P', T)$. Thus, we can determine the value of b_2 using Proposition 7.53(4) in $\mathcal{O}(\log\log n + |P'|/\log_\sigma n) = \mathcal{O}(\log\log n)$ time.

Construction algorithm The components of the structure are constructed as follows:

1. The first component is constructed using Proposition 7.53 in $\mathcal{O}(n/\log_\sigma n)$ time.
2. The second component is constructed in the same time using Proposition 7.58.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n)$ time. □

7.3.1.5 Summary

Proposition 7.60. *Consider a data structure answering prefix range reporting queries (see Section 7.1) that, for any sequence W of k strings of length ℓ over alphabet $[0\,..\,\sigma)$ achieves the following complexities (where the input strings during construction are given in the packed representation, and at query time we are given any $b, e \in [0\,..\,k]$ and the packed representation of any $X \in [0\,..\,\sigma)^{\leq \ell}$, and we return the set $\mathcal{I} := \{i \in (b\,..\,e] : X \text{ is a prefix of } W[i]\}$):*

- *space usage $S(k, \ell, \sigma)$,*
- *preprocessing time $P_t(k, \ell, \sigma)$,*
- *preprocessing space $P_s(k, \ell, \sigma)$,*
- *query time $Q_{\mathrm{base}}(k, \ell, \sigma) + |\mathcal{I}| \cdot Q_{\mathrm{elem}}(k, \ell, \sigma)$.*

Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. There exist positive integers $k = \Theta(n/\log_\sigma n)$ and $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n + P_t(k, \ell, \sigma))$ time and using $\mathcal{O}(n/\log_\sigma n + P_s(k, \ell, \sigma))$ working space construct a data structure of size $\mathcal{O}(n/\log_\sigma n + S(k, \ell, \sigma))$ that, given the packed representation of any patterns $P_1, P_2 \in [0\,..\,\sigma)^*$, computes the set $\mathcal{J} := \mathrm{LexRange}(P_1, P_2, T)$ (see Definition 2.4) in $\mathcal{O}(\log\log n + Q_{\mathrm{base}}(k, \ell, \sigma) + |\mathcal{J}| \cdot Q_{\mathrm{elem}}(k, \ell, \sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.

Proof. We use the following definitions. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$ (such τ exists by the assumption $\sigma < n^{1/13}$). Let k and ℓ be integers resulting from the application of Proposition 7.25 to text T, with the structure from the above claim to answer prefix range reporting queries. Note that they satisfy $k = \Theta(n/\log_\sigma n)$ and $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$.

Components The data structure consists of the following components:

1. The structure from Proposition 7.8 for σ and τ. If needs $\mathcal{O}(\sigma^{3\tau} \cdot \tau^2) = \mathcal{O}(n/\log_\sigma n)$ space.
2. The structure from Proposition 7.10 for τ and T. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
3. The structure from Proposition 7.12 for τ and T. It uses $\mathcal{O}(n/\log_\sigma n)$ space.
4. The structure from Proposition 7.25 for τ and T, and using the structure from the above claim to answer prefix range reporting queries. It uses $\mathcal{O}(n/\log_\sigma n + S(k, \ell, \sigma))$ space, where k and ℓ are defined above.
5. The structure from Proposition 7.59 for τ and T. It uses $\mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n + S(k, \ell, \sigma))$ space.

Implementation of queries The queries are answered as follows. Let $P_1, P_2 \in [0\,..\,\sigma)^*$. Given the packed representation of P_1 and P_2, we compute $\mathrm{LexRange}(P_1, P_2, T)$ as follows. First, in $\mathcal{O}(1 + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time we check if $P_1 \prec P_2$. If not, then we have $\mathrm{LexRange}(P_1, P_2, T) = \emptyset$ (see Definition 2.4), and the query algorithm is complete. Let us thus assume that it holds $P_1 \prec P_2$. In $\mathcal{O}(1)$ time we use the packed representation of P_1 and P_2 to compute the packed representation of prefixes $X_1 = P_1[1\,..\,\min(3\tau - 1, |P_1|)]$ and $X_2 = P_2[1\,..\,\min(3\tau - 1, |P_2|)]$. Note that the assumption $P_1 \prec P_2$ implies that $X_1 \preceq X_2$. In $\mathcal{O}(1)$ time we check if $X_1 = X_2$. In $\mathcal{O}(1)$ time we also check if X_1 is a prefix of X_2. We then consider three cases:

- First, assume that $X_1 = X_2$. Note that $|X_1| = |X_2| = 3\tau - 1$, since otherwise $P_1 = X_1 = X_2 = P_2$, and the algorithm would have already finished. Using Proposition 7.8, in $\mathcal{O}(1)$ time we check if X_1 is τ-periodic (Definition 7.5). Consider two cases:
  - If X_1 is τ-periodic, then so are X_2, P_1, and P_2. We compute $\mathcal{J} := \mathrm{LexRange}(P_1, P_2, T)$ using Proposition 7.59(1) in $\mathcal{O}(\log\log n + |\mathcal{J}| + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.
  - Otherwise, P_1 and P_2 are τ-nonperiodic. Since $\mathrm{lcp}(P_1, P_2) \geq 3\tau - 1$, we then compute $\mathcal{J} := \mathrm{LexRange}(P_1, P_2, T)$ in $\mathcal{O}(\log\log n + Q_{\mathrm{base}}(k,\ell,\sigma) + |\mathcal{J}| \cdot Q_{\mathrm{elem}}(k,\ell,\sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time using Proposition 7.25(1).

  In total, we spend $\mathcal{O}(\log\log n + |\mathcal{J}| + Q_{\mathrm{base}}(k,\ell,\sigma) + |\mathcal{J}| \cdot Q_{\mathrm{elem}}(k,\ell,\sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n) = \mathcal{O}(\log\log n + Q_{\mathrm{base}}(k,\ell,\sigma) + |\mathcal{J}| \cdot Q_{\mathrm{elem}}(k,\ell,\sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.
- Let us now assume that $X_1 \neq X_2$ and X_1 is a prefix of X_2. Note that then X_1 must be a proper prefix of X_2, and hence $|X_1| < |X_2|$. Thus, we must have $P_1 = X_1$. Combining this with $X_1 \preceq X_2$ (see above), we obtain $P_1 \preceq X_2 \preceq P_2$. Consequently, we can write $\mathrm{LexRange}(P_1, P_2, T)$ as a disjoint union:

$$\mathrm{LexRange}(P_1, P_2, T) = \mathrm{LexRange}(P_1, X_2, T) \cup \mathrm{LexRange}(X_2, P_2, T).$$

  We will separately enumerate each of the two sets. Using Proposition 7.12(1), we compute $\mathcal{J}_1 := \mathrm{LexRange}(X_1, X_2, T)$ (recall that $P_1 = X_1$) in $\mathcal{O}(1 + |\mathcal{J}_1|)$ time. Let us now focus on computing $\mathcal{J}_2 := \mathrm{LexRange}(X_2, P_2, T)$. If $|P_2| < 3\tau - 1$, then again we compute $\mathcal{J}_2$ using Proposition 7.12(1) in $\mathcal{O}(1 + |\mathcal{J}_2|)$ time. Let us now assume that $|P_2| \geq 3\tau - 1$. Using Proposition 7.8, in $\mathcal{O}(1)$ time we check if P_2 is τ-periodic (Definition 7.5). Consider two cases:
  - If P_2 is τ-periodic, then using Proposition 7.59(1), we compute $\mathcal{J}_2$ in $\mathcal{O}(\log\log n + |\mathcal{J}_2| + |P_2|/\log_\sigma n)$ time. Note that we can use this because $X_2 = P_2[1\,..\,3\tau - 1]$.
  - If P_2 is τ-nonperiodic, then using Proposition 7.25(1), we compute $\mathcal{J}_2$ in $\mathcal{O}(\log\log n + Q_{\mathrm{base}}(k,\ell,\sigma) + |\mathcal{J}_2| \cdot Q_{\mathrm{elem}}(k,\ell,\sigma) + |P_2|/\log_\sigma n)$ time.

  In total, we spend $\mathcal{O}(\log\log n + |\mathcal{J}_1| + |\mathcal{J}_2| + Q_{\mathrm{base}}(k,\ell,\sigma) + |\mathcal{J}_2| \cdot Q_{\mathrm{elem}}(k,\ell,\sigma) + |P_2|/\log_\sigma n) = \mathcal{O}(\log\log n + Q_{\mathrm{base}}(k,\ell,\sigma) + |\mathcal{J}| \cdot Q_{\mathrm{elem}}(k,\ell,\sigma) + |P_2|/\log_\sigma n)$ time, where

$$\mathcal{J} := \mathrm{LexRange}(P_1, P_2, T) = \mathcal{J}_1 \cup \mathcal{J}_2.$$

- Let us now assume that $X_1 \neq X_2$ and X_1 is not a prefix of X_2. Observe that, letting $c = \sigma - 1$, we then have $X_1 c^\infty \prec X_2$. Combining this with $P_1 \preceq X_1 c^\infty$ and $X_2 \preceq P_2$, we can thus write $\mathrm{LexRange}(P_1, P_2, T)$ as a disjoint union:

$$\begin{aligned}\mathrm{LexRange}(P_1, P_2, T) = {} & \mathrm{LexRange}(P_1, X_1 c^\infty, T) \cup \\ & \mathrm{LexRange}(X_1 c^\infty, X_2, T) \cup \\ & \mathrm{LexRange}(X_2, P_2, T).\end{aligned}$$

  We will separately enumerate the elements of each of these three sets. We begin with the set $\mathcal{J}_1 := \mathrm{LexRange}(P_1, X_1 c^\infty, T)$. If $|P_1| \leq 3\tau - 1$, then $P_1 = X_1$, and hence we compute $\mathcal{J}_1$ in $\mathcal{O}(1 + |\mathcal{J}_1|)$ time using Proposition 7.12(2). Let us now assume that $|P_1| > 3\tau - 1$. In $\mathcal{O}(1)$ time we check if P_1 is τ-periodic using Proposition 7.8. We then consider two cases:
  - If P_1 is τ-periodic, then we compute $\mathcal{J}_1$ using Proposition 7.59(2) in $\mathcal{O}(\log\log n + |\mathcal{J}_1| + |P_1|/\log_\sigma n)$ time.
  - Otherwise (i.e., if P_1 is τ-nonperiodic), we compute $\mathcal{J}_1$ using Proposition 7.25(2) in $\mathcal{O}(\log\log n + Q_{\mathrm{base}}(k,\ell,\sigma) + |\mathcal{J}_1| \cdot Q_{\mathrm{elem}}(k,\ell,\sigma) + |P_1|/\log_\sigma n)$ time.

  Next, we compute $\mathcal{J}_2 := \mathrm{LexRange}(X_1 c^\infty, X_2, T)$. Note that $\mathrm{RangeBeg}(X_1 c^\infty, T) = \mathrm{RangeEnd}(X_1, T)$. We thus proceed as follows. Using Proposition 7.10, in $\mathcal{O}(1)$ time we compute $e_1 = \mathrm{RangeEnd}(X_1, T) = \mathrm{RangeBeg}(X_1 c^\infty, T)$ and $b_2 = \mathrm{RangeBeg}(X_2, T)$. If $e_1 \geq b_2$, then by Remark 2.5, we have $\mathcal{J}_2 = \emptyset$. Let us thus assume that $e_1 < b_2$. Then, we compute $\mathcal{J}_2$ using Proposition 7.12(3) in $\mathcal{O}(1 + |\mathcal{J}_2|)$ time. Finally, we address the problem of computing $\mathcal{J}_3 := \mathrm{LexRange}(X_2, P_2, T)$. If $|P_2| \leq 3\tau - 1$, then $P_2 = X_2$, and hence we have $\mathcal{J}_3 = \emptyset$. Let us now assume that $|P_2| > 3\tau - 1$. In $\mathcal{O}(1)$ time we check if P_2 is τ-periodic using Proposition 7.8. We then consider two cases:

- If P_2 is τ-nonperiodic, then we compute $\mathcal{J}_3$ using Proposition 7.25(1) in $\mathcal{O}(\log\log n + Q_{\text{base}}(k,\ell,\sigma) + |\mathcal{J}_3| \cdot Q_{\text{elem}}(k,\ell,\sigma) + |P_2|/\log_\sigma n)$ time.
- Otherwise (i.e., if P_2 is τ-periodic), we compute the elements of the set $\mathcal{J}_3$ using Proposition 7.59(1) in $\mathcal{O}(\log\log n + |\mathcal{J}_3| + |P_2|/\log_\sigma n)$ time.

In total, we spend $\mathcal{O}(\log\log n + |\mathcal{J}_1| + |\mathcal{J}_2| + |\mathcal{J}_3| + Q_{\text{base}}(k,\ell,\sigma) + (|\mathcal{J}_1| + |\mathcal{J}_3|) \cdot Q_{\text{elem}}(k,\ell,\sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n) = \mathcal{O}(\log\log n + Q_{\text{base}}(k,\ell,\sigma) + |\mathcal{J}| \cdot Q_{\text{elem}}(k,\ell,\sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time in the above step, where $\mathcal{J} = \mathcal{J}_1 \cup \mathcal{J}_2 \cup \mathcal{J}_3$.

In total, we spend $\mathcal{O}(\log\log n + Q_{\text{base}}(k,\ell,\sigma) + |\mathcal{J}| \cdot Q_{\text{elem}}(k,\ell,\sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time, where $\mathcal{J} = \text{LexRange}(P_1, P_2, T)$.

Construction algorithm The components of the structure are constructed as follows:

1. In $\mathcal{O}(\sigma^{3\tau} \cdot \tau^2) = \mathcal{O}(n/\log_\sigma n)$ time we apply Proposition 7.8.
2. In $\mathcal{O}(n/\log_\sigma n)$ time we apply Proposition 7.10 to text T.
3. In the same time as above we apply Proposition 7.12 to text T.
4. In $\mathcal{O}(n/\log_\sigma n + P_t(k,\ell,\sigma))$ time and using $\mathcal{O}(n/\log_\sigma n + P_s(k,\ell,\sigma))$ working space we apply Proposition 7.25 to text T.
5. In $\mathcal{O}(n/\log_\sigma n)$ time we apply Proposition 7.59 to text T.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n + P_t(k,\ell,\sigma))$ time uses $\mathcal{O}(n/\log_\sigma n + P_s(k,\ell,\sigma))$ working space. □

7.3.2 Alphabet Reduction for Prefix Range Reporting Queries

Proposition 7.61. *Consider a data structure answering prefix range reporting queries (see Section 7.1) that, for and any sequence W of m binary strings of length $\ell = 1 + \lfloor \log m \rfloor$ (where $m \geq 1$), achieves the following complexities (where both the input strings during construction are given in the packed representation, and at query time, we are given some $b, e \in [0\,..\,m]$, and the packed representation of some $X \in \{0,1\}^{\leq \ell}$, and we return the set $\mathcal{I} := \{i \in (b\,..\,e] : X \text{ is a prefix of } W[i]\}$):*

- *space usage $S(m)$,*
- *preprocessing time $P_t(m)$,*
- *preprocessing space $P_s(m)$,*
- *query time $Q_{\text{base}}(m) + |\mathcal{I}| \cdot Q_{\text{elem}}(m)$.*

Let $m' \geq 1$ and let $W'[1\,..\,m']$ be a sequence of m' equal-length strings over alphabet $[0\,..\,\sigma)$ (where $\sigma \geq 2$) of length $\ell' \leq (1 + \lfloor \log m' \rfloor)/\lceil \log \sigma \rceil$. Given the array W', with all strings represented in the packed form, we can in $\mathcal{O}(m' + P_t(m'))$ time and using $\mathcal{O}(m' + P_s(m'))$ working space construct a data structure of size $\mathcal{O}(m' + S(m'))$ that, given the packed representation of any $X' \in [0\,..\,\sigma)^{\leq \ell'}$ and any $b, e \in [0\,..\,m']$, computes

$$\mathcal{I}' := \{i \in (b\,..\,e] : X' \text{ is a prefix of } W'[i]\}$$

in $\mathcal{O}(Q_{\text{base}}(m') + |\mathcal{I}'| \cdot Q_{\text{elem}}(m'))$ time.

Proof. The above reduction is essentially the same as the one presented in Proposition 4.20, except we only need Lemma 4.19(1c). □

7.3.3 Summary

Theorem 7.62. *Consider a data structure answering prefix range reporting queries (see Section 7.1) that, for any sequence W of $m \geq 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$, achieves the following complexities (where the input strings during construction are given in the packed representation, and at query time we are given any $b, e \in [0\,..\,m]$, and the packed representation of any $X \in \{0,1\}^{\leq \ell}$, and we return $\mathcal{I} := \{i \in (b\,..\,e] : X \text{ is a prefix of } W[i]\}$):*

- *space usage $S(m)$,*
- *preprocessing time $P_t(m)$,*
- *preprocessing space $P_s(m)$,*
- *query time $Q_{\text{base}}(m) + |\mathcal{I}| \cdot Q_{\text{elem}}(m)$.*

For every $T \in \{0,1\}^n$, there exists $m = \Theta(n/\log n)$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log n + P_t(m))$ time and using $\mathcal{O}(n/\log n + P_s(m))$ working space construct a data structure of size $\mathcal{O}(n/\log n + S(m))$ that, given the packed representation of any $P_1, P_2 \in \{0,1\}^*$, computes $\mathcal{J} := \mathrm{LexRange}(P_1, P_2, T)$ (see Definition 2.4) in $\mathcal{O}(\log\log n + Q_{\mathrm{base}}(m) + |\mathcal{J}| \cdot Q_{\mathrm{elem}}(m) + |P_1|/\log n + |P_2|/\log n)$ time.

Proof. Assume that $n > 3^{13} - 1$ (otherwise, the claim holds trivially).

We use the following definitions. Let $\sigma = 3$ and let $T' \in [0\,..\,\sigma)^{n+1}$ be a string defined so that $T'[1\,..\,n] = T$ and $T'[n+1] = 2$. Denote $n' = |T'|$. Note that $T'[n']$ does not occur in $T'[1\,..\,n')$ and it holds $2 \le \sigma < (n')^{1/13}$. Let D denote the data structure resulting from applying Proposition 7.61 with alphabet size σ to the structure (answering prefix range reporting queries on binary strings) form the claim. Given any sequence $W[1\,..\,m']$ of $m' \ge 1$ equal-length strings over alphabet $[0\,..\,\sigma)$ of length $\ell' \le (1 + \lfloor \log m' \rfloor)/\lceil \log \sigma \rceil$ as input (with all strings represented in the packed form), the structure D achieves the following complexities (where at query time it is given the packed representation of any $X \in [0\,..\,\sigma)^{\le \ell'}$ and any $b, e \in [0\,..\,m']$, and computes $\mathcal{I} := \{i \in (b\,..\,e] : X \text{ is a prefix of } W[i]\}$):

- space usage $S'(m', \ell', \sigma) = \mathcal{O}(m' + S(m'))$,
- preprocessing time $P'_t(m', \ell', \sigma) = \mathcal{O}(m' + P_t(m'))$,
- preprocessing space $P'_s(m', \ell', \sigma) = \mathcal{O}(m' + P_s(m'))$,
- query time $Q'_{\mathrm{base}}(m', \ell', \sigma) = \mathcal{O}(Q_{\mathrm{base}}(m'))$ and $Q'_{\mathrm{elem}}(m', \ell', \sigma) = \mathcal{O}(Q_{\mathrm{elem}}(m'))$,

where the complexities $S(m')$, $P_t(m')$, $P_s(m')$, $Q_{\mathrm{base}}(m')$, and $Q_{\mathrm{elem}}(m')$ refer to the structure from the claim (answering prefix range reporting queries for binary strings).

Components The data structure answering lex-range reporting queries consists of a single component: the data structure from Proposition 7.60 applied for text T' and with the structure D as the structure answering prefix range reporting queries. Note that we can use D, since it supports prefix range reporting queries for the combination of parameters required in Proposition 7.60, i.e., for sequences over alphabet $[0\,..\,\sigma)$, with the length ℓ of all strings satisfying $\ell \le (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ (where k is the length of the input sequence). To bound the space usage of this component, note that by Proposition 7.60 and the above discussion, there exists $m = \Theta(n'/\log_\sigma n') = \Theta(n/\log n)$ (recall that $\sigma = 3$) such that the structure needs $\mathcal{O}(n'/\log_\sigma n' + S'(m, \ell, \sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + S(m)) = \mathcal{O}(n/\log n + S(m))$ space.

Implementation of queries The lex-range reporting queries are answered as follows. Let $P_1, P_2 \in \{0,1\}^*$. Note that by definition of T', it follows that $\mathrm{LexRange}(P_1, P_2, T) = \mathrm{LexRange}(P_1, P_2, T')$. By Proposition 7.60 and the above discussion, enumerating the set $\mathcal{J} := \mathrm{LexRange}(P_1, P_2, T')$ takes $\mathcal{O}(\log\log n' + Q'_{\mathrm{base}}(m, \ell, \sigma) + |\mathcal{J}| \cdot Q'_{\mathrm{elem}}(m, \ell, \sigma) + |P_1|/\log_\sigma n' + |P_2|/\log_\sigma n') = \mathcal{O}(\log\log n + Q_{\mathrm{base}}(m) + |\mathcal{J}| \cdot Q_{\mathrm{elem}}(m) + |P_1|/\log n + |P_2|/\log n)$ time.

Construction algorithm By Proposition 7.60 and the above discussion, construction of the above data structure answering lex-range reporting queries on T' (and hence also on T) takes $\mathcal{O}(n'/\log_\sigma n' + P'_t(m, \ell, \sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + P_t(m)) = \mathcal{O}(n/\log n + P_t(m))$ time and uses $\mathcal{O}(n'/\log_\sigma n' + P'_s(m, \ell, \sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + P_s(m)) = \mathcal{O}(n/\log n + P_s(m))$ working space. □

8 Equivalence of Lex-Range Emptiness and Prefix Range Emptiness

8.1 Problem Definitions

Indexing for Lex-Range Emptiness Queries

Input: The packed representation of a string $T \in \{0,1\}^n$.
Output: A data structure that, given the packed representation of any $P_1, P_2 \in \{0,1\}^*$, returns a YES/NO answer indicating whether the set $\mathrm{LexRange}(P_1, P_2, T)$ (Definition 2.4) is empty.

Indexing for Prefix Range Emptiness Queries

Input: A sequence $W[1\,..\,m]$ of $m \geq 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$, with all strings represented in the packed form.

Output: A data structure that, given the packed representation of any $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$ and any $b, e \in [0\,..\,m]$, returns a YES/NO answer indicating whether $\mathsf{prefix\text{-}rank}_W(e, X) - \mathsf{prefix\text{-}rank}_W(b, X) > 0$, i.e., whether there exists $i \in (b\,..\,e]$ such that X is a prefix of $W[i]$.

8.2 Reducing Prefix Range Emptiness to Lex-Range Emptiness Queries

8.2.1 Problem Reduction

Lemma 8.1. *Let $W[1\,..\,m]$ be a sequence of $m \geq 1$ strings of length $\ell \geq 1$ over alphabet $\{\mathtt{0},\mathtt{1}\}$. Let $s(i)$ (where $i \in [0\,..\,m]$) be as in Definition 4.7. Denote $T = \mathrm{SeqToString}_\ell(W)$ (Definition 4.7). Consider a string $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$ and let $b, e \in [0\,..\,m]$ be such that $b < e$. Denote $P_1 = \overline{X} \cdot \mathtt{4} \cdot s(b)$ and $P_2 = \overline{X} \cdot \mathtt{4} \cdot s(e)$. Then, it holds (see Definition 2.4)*

$$\mathsf{prefix\text{-}rank}_W(e, X) - \mathsf{prefix\text{-}rank}_W(b, X) = |\mathrm{LexRange}(P_1, P_2, T)|.$$

Proof. By Lemma 6.1, letting $\alpha = \mathrm{RangeBeg}(\overline{X} \cdot \mathtt{4}, T)$, it holds

$$\begin{aligned}\mathsf{prefix\text{-}rank}_W(e, X) - \mathsf{prefix\text{-}rank}_W(b, X) &= (\mathrm{RangeBeg}(P_2, T) - \alpha) - (\mathrm{RangeBeg}(P_1, T) - \alpha)\\ &= \mathrm{RangeBeg}(P_2, T) - \mathrm{RangeBeg}(P_1, T)\\ &= |\mathrm{LexRange}(P_1, P_2, T)|,\end{aligned}$$

where in the last equality we use that $P_1 \prec P_2$ (which holds due to $b < e$); see also Remark 2.5. □

8.2.2 Alphabet Reduction for Lex-Range Emptiness Queries

Lemma 8.2. *Let $\sigma \geq 2$ and $T, P_1, P_2 \in [0\,..\,\sigma)^*$. Then $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$ holds if and only if $\mathrm{LexRange}(P_1', P_2', T') \neq \emptyset$, where $k = \lceil \log \sigma \rceil$, $P_1' = \mathrm{ebin}_k(P_1)$, $P_2' = \mathrm{ebin}_k(P_2)$, and $T' = \mathrm{ebin}_k(T)$ (Definition 4.3).*

Proof. The proof follows immediately by Lemma 7.2. □

Proposition 8.3. *Let $T \in [0\,..\,\sigma)^n$ be a nonempty string, where $2 \leq \sigma < n^{\mathcal{O}(1)}$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct the packed representation of a text $T_{\mathrm{bin}} \in \{\mathtt{0},\mathtt{1}\}^*$ satisfying $|T_{\mathrm{bin}}| = \Theta(n \log \sigma)$, and a data structure that, given the packed representation of any $P_1, P_2 \in [0\,..\,\sigma)^{\leq m}$, in $\mathcal{O}(1 + m/\log_\sigma n)$ time returns the packed representation of $P_1', P_2' \in \{\mathtt{0},\mathtt{1}\}^*$ satisfying $|P_i'| = |P_i| \cdot (2\lceil \log \sigma \rceil + 3)$ (where $i \in \{1, 2\}$) such that $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$ holds if and only if $\mathrm{LexRange}(P_1', P_2', T_{\mathrm{bin}}) \neq \emptyset$.*

Proof. Let $k = \lceil \log \sigma \rceil$. To compute T_{bin}, we proceed as follows:

1. In $\mathcal{O}(n/\log_\sigma n)$ time we construct the data structure from Proposition 4.5 for $u = n$. Given the packed representation of any $S \in [0\,..\,\sigma)^*$, we can then compute the packed representation of $\mathrm{ebin}_k(S)$ (Definition 4.3) in $\mathcal{O}(1 + |S|/\log_\sigma n)$ time.
2. Compute the packed representation of $T_{\mathrm{bin}} = \mathrm{ebin}_k(T)$ in $\mathcal{O}(n/\log_\sigma n)$ time.

In total, the construction of the packed representation of T_{bin} takes $\mathcal{O}(n/\log_\sigma n)$. By Definition 4.3, it holds $|T_{\mathrm{bin}}| = n \cdot (2k + 3) = \Theta(n \log \sigma)$.

Next, we address the construction of the data structure from the claim. It consists of a single component: the data structure from Proposition 4.5 for $u = n$. It uses $\mathcal{O}(n/\log_\sigma n)$ space.

The queries are implemented as follows. Given the packed representation of $P_1, P_2 \in [0\,..\,\sigma)^{\leq m}$, we compute the packed representation of P_1' and P_2' defined as $P_i' = \mathrm{ebin}_k(P_i)$. Using the structure from Proposition 4.5, this takes $\mathcal{O}(1 + m/\log_\sigma n)$ time. By Definition 4.3, it holds $|P_i'| = |P_i| \cdot (2k + 3)$ for $i \in \{1, 2\}$, and by Lemma 8.2, $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$ holds if and only if $\mathrm{LexRange}(P_1', P_2', T_{\mathrm{bin}}) \neq \emptyset$.

By Proposition 4.5, construction of the data structure takes $\mathcal{O}(n/\log_\sigma n)$ time. □

8.2.3 Summary

Theorem 8.4. *Consider a data structure answering lex-range emptiness queries (see Section 8.1) that, for any text $T \in \{0,1\}^n$, achieves the following complexities (where in the preprocessing we assume that we are given as input the packed representation of T, and at query time we are given a packed representation of $P_1, P_2 \in \{0,1\}^{\leq k}$):*

- *space usage $S(n)$,*
- *preprocessing time $P_t(n)$,*
- *preprocessing space $P_s(n)$,*
- *query time $Q(n,k)$.*

For every sequence $W[1\,..\,m]$ of $m \geq 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$, there exists $n = \mathcal{O}(m \log m)$ and $k = \mathcal{O}(\log m)$ such that, given the sequence W, with all strings represented in the packed form, we can in $\mathcal{O}(m + P_t(n))$ time and using $\mathcal{O}(m + P_s(n))$ working space construct a data structure of size $\mathcal{O}(m + S(n))$ that, given the packed representation of any $X \in \{0,1\}^{\leq \ell}$ and any pair of integers $b, e \in [0\,..\,m]$, determines if there exists $i \in (b\,..\,e]$ such that X is a prefix of $W[i]$ (i.e., if $\mathsf{prefix\text{-}rank}_W(e, X) - \mathsf{prefix\text{-}rank}_W(b, X) > 0$) in $\mathcal{O}(Q(n,k))$ time.

Proof. We use the following definitions. Let $T_{\mathrm{aux}} = \mathrm{SeqToString}_\ell(W)$ (Definition 4.7). Denote $n_{\mathrm{aux}} = |T_{\mathrm{aux}}| = m \cdot (2\ell+1) = \Theta(m \log m)$. Let T_{bin} denote the text obtained by applying Proposition 8.3 to text T_{aux}. Denote $n = |T_{\mathrm{bin}}|$. Since T_{aux} is over alphabet $\{0,\ldots,4\}$, by Proposition 8.3, we have $n = \Theta(|T_{\mathrm{aux}}|) = \Theta(m \log m)$. Denote $k = 18\lfloor \log m \rfloor + 27 = \mathcal{O}(\log m)$.

Components The data structure consists of the following components:

1. The structure from Proposition 8.3 applied to text T_{aux}. Since T_{aux} is over alphabet $\{0,\ldots,4\}$, the structure needs $\mathcal{O}(|T_{\mathrm{aux}}|/\log|T_{\mathrm{aux}}|) = \mathcal{O}(m)$ space.
2. The data structure from the claim (i.e., answering lex-range emptiness queries) for the text T_{bin}. The structure needs $\mathcal{O}(S(n))$ space.
3. A lookup table L_{rev} defined as in Theorem 6.4. It needs $\mathcal{O}(\sqrt{n}) = \mathcal{O}(m)$ space.
4. A lookup table L_{map} defined as in Theorem 6.4. It also needs $\mathcal{O}(\sqrt{n}) = \mathcal{O}(m)$ space.

In total, the structure needs $\mathcal{O}(m + S(n))$ space.

Implementation of queries The queries are answered as follows. Assume that we are given the packed representation of some $X \in \{0,1\}^{\leq \ell}$ and two integers $b, e \in [0\,..\,m]$. If $b \geq e$, then we immediately return that $\mathsf{prefix\text{-}rank}_W(e, X) - \mathsf{prefix\text{-}rank}_W(b, X) \leq 0$. Let us thus assume that $b < e$. The query algorithm proceeds as follows:

1. Using L_{rev} and L_{map}, in $\mathcal{O}(1)$ time we compute the packed representation of strings $P_1 = \overline{X} \cdot 4 \cdot s(b)$ and $P_2 = \overline{X} \cdot 4 \cdot s(e)$ (over alphabet $\{0,\ldots,4\}$). Note that $|P_1| = |P_2| = |X| + 1 + \ell \leq 2\ell + 1 \leq 2\lfloor \log m \rfloor + 3$.
2. Using Proposition 8.3, in $\mathcal{O}(1 + |P_1|/\log n_{\mathrm{aux}} + |P_2|/\log n_{\mathrm{aux}}) = \mathcal{O}(1 + \ell/\log m) = \mathcal{O}(1)$ time, we compute the packed representation of strings $P_1', P_2' \in \{0,1\}^*$ of length $|P_1'| = |P_2'| = 9 \cdot |P_1| \leq 18\lfloor \log m \rfloor + 27 \leq k$ such that $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$ if and only if $\mathrm{LexRange}(P_1', P_2', T_{\mathrm{bin}}) \neq \emptyset$.
3. Using the structure from the claim, we determine whether it holds $\mathrm{LexRange}(P_1', P_2', T_{\mathrm{bin}}) \neq \emptyset$. By $|P_1'|, |P_2'| \leq k$, this takes $\mathcal{O}(Q(n,k))$ time. By the observation in the previous step and Lemma 8.1, this answer also determines whether $\mathsf{prefix\text{-}rank}_W(e, X) - \mathsf{prefix\text{-}rank}_W(b, X) > 0$. Thus we return that bit as the output of the query.

In total, the query takes $\mathcal{O}(Q(n,k))$ time.

Construction algorithm The components of the data structure are constructed as follows:

1. The first component of the structure is computed as follows:
   (a) In $\mathcal{O}(m)$ time we apply Proposition 4.12 to the sequence W to compute the packed representation of the string $T_{\mathrm{aux}} = \mathrm{SeqToString}_\ell(W)$ (Definition 4.7).
   (b) We apply Proposition 8.3 to T_{aux}. Since the string T_{aux} is over alphabet $\{0,\ldots,4\}$, this takes $\mathcal{O}(|T_{\mathrm{aux}}|/\log|T_{\mathrm{aux}}|) = \mathcal{O}(m)$ time. Note that Proposition 8.3, in addition to the data structure,

also returns the packed representation of the string T_{bin} (defined above).

2. We apply the preprocessing from the claim to the string T_{bin}. This takes $\mathcal{O}(P_t(|T_{\text{bin}}|)) = \mathcal{O}(P_t(n))$ time and uses $\mathcal{O}(P_s(|T_{\text{bin}}|)) = \mathcal{O}(P_s(n))$ working space.
3. The lookup table L_{rev} is computed in $\mathcal{O}(m)$ time similarly as in the proof of Proposition 4.12.
4. Next, we compute the lookup table L_{map}. Similarly as above, we proceed as in Proposition 4.12, and spend $\mathcal{O}(m)$ time.

In total, the construction takes $\mathcal{O}(m + P_t(n))$ time and uses $\mathcal{O}(m + P_s(n))$ working space. □

8.3 Reducing Lex-Range Emptiness to Prefix Range Emptiness Queries

8.3.1 Problem Reduction

8.3.1.1 The Short Patterns

Proposition 8.5. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that answers the following queries:*

1. *Given the packed representation of any $X_1, X_2 \in [0\,..\,\sigma)^{\leq 3\tau-1}$, check whether $\text{LexRange}(X_1, X_2, T) \neq \emptyset$ in $\mathcal{O}(1)$ time.*
2. *Given the packed representation of any $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, check whether $\text{LexRange}(X, Xc^\infty, T) \neq \emptyset$ (where $c = \sigma - 1$) in $\mathcal{O}(1)$ time.*

Proof. **Components** The data structure consists of a single component: the structure from Proposition 7.10. It needs $\mathcal{O}(n/\log_\sigma n)$ space.

Implementation of queries The queries are answered as follows:

1. Let $X_1, X_2 \in [0\,..\,\sigma)^{\leq 3\tau-1}$. Using Proposition 7.10 and the packed representations of X_1 and X_2, in $\mathcal{O}(1)$ time we compute the values $b_1 = \text{RangeBeg}(X_1, T)$ and $b_2 = \text{RangeBeg}(X_2, T)$. If $b_1 < b_2$, we have $\text{LexRange}(X_1, X_2, T) \neq \emptyset$. Otherwise, we return that $\text{LexRange}(X_1, X_2, T) = \emptyset$. The correctness follows by Remark 2.5. In total, we spend $\mathcal{O}(1)$ time.
2. Let $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$. Observe that it holds $\text{RangeBeg}(Xc^\infty, T) = \text{RangeEnd}(X, T)$. Thus, the query is answered as above, except b_2 is computed as $\text{RangeEnd}(X, T)$ using Proposition 7.10. In total, we again spend $\mathcal{O}(1)$ time.

Construction algorithm The construction of the structure takes $\mathcal{O}(n/\log_\sigma n)$ time by Proposition 7.10. □

8.3.1.2 The Nonperiodic Patterns

Combinatorial Properties

Lemma 8.6. *Let $T \in \Sigma^n$, $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$, and assume that $T[n]$ does not occur in $T[1\,..\,n)$. Let S be a τ-synchronizing set of T. Denote $(s_i)_{i \in [1\,..\,n']} = \text{LexSorted}(\mathsf{S}, T)$ (Definition 7.13). Let $A_{\mathsf{S}}[1\,..\,n']$ and $A_{\text{str}}[1\,..\,n']$ be defined by*

- $A_{\mathsf{S}}[i] = s_i$,
- $A_{\text{str}}[i] = \overline{D_i}$, *where* $D_i = T^\infty[s_i - \tau\,..\,s_i + 2\tau)$.

Let $D \in \mathcal{D}(\tau, T, \mathsf{S})$ (Definition 7.16) and let $P_1, P_2 \in \Sigma^+$ be τ-nonperiodic patterns (Definition 7.5) both having D as a prefix. Denote $\delta_{\text{text}} = |D| - 2\tau$. For $k \in \{1, 2\}$, let $P'_k = P_k(\delta_{\text{text}}\,..\,|P_k|]$, and $b_k = |\{i \in [1\,..\,n'] : T[s_i\,..\,n] \prec P'_k\}|$. Then, the following conditions are equivalent:

- $\text{LexRange}(P_1, P_2, T) \neq \emptyset$,
- $\{i \in (b_1\,..\,b_2] : \overline{D}$ *is a prefix of* $A_{\text{str}}[i]\} \neq \emptyset$.

Proof. The result follows from Lemma 7.24. □

Algorithms

Proposition 8.7. *Consider a data structure answering prefix range emptiness queries (see Section 8.1) that, given any sequence W of k strings of length ℓ over alphabet $[0\,..\,\sigma)$ achieves the following complexities (where the input strings during construction are given in the packed representation, and at query time we are given any $b, e \in [0\,..\,k]$ and the packed representation of any $X \in [0\,..\,\sigma)^{\leq \ell}$, and we return a bit indicating whether $\{i \in (b\,..\,e] : X \text{ is a prefix of } W[i]\} \neq \emptyset$):*

- *space usage $S(k, \ell, \sigma)$,*
- *preprocessing time $P_t(k, \ell, \sigma)$,*
- *preprocessing space $P_s(k, \ell, \sigma)$,*
- *query time $Q(k, \ell, \sigma)$.*

Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. There exist positive integers $k = \Theta(n/\log_\sigma n)$ and $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n + P_t(k, \ell, \sigma))$ time and using $\mathcal{O}(n/\log_\sigma n + P_s(k, \ell, \sigma))$ working space construct a data structure of size $\mathcal{O}(n/\log_\sigma n + S(k, \ell, \sigma))$ that answers the following queries:

1. *Given the packed representation of any τ-nonperiodic (Definition 7.5) patterns $P_1, P_2 \in [0\,..\,\sigma)^+$ satisfying $\operatorname{lcp}(P_1, P_2) \geq 3\tau - 1$, determines whether $\operatorname{LexRange}(P_1, P_2, T) \neq \emptyset$ (Definition 2.4) in $\mathcal{O}(\log\log n + Q(k, \ell, \sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.*
2. *Given the packed representation of a τ-nonperiodic pattern $P \in [0\,..\,\sigma)^+$ satisfying $|P| \geq 3\tau - 1$, determines whether $\operatorname{LexRange}(P, P'c^\infty, T) \neq \emptyset$ (where $P' = P[1\,..\,3\tau - 1]$ and $c = \sigma - 1$) in $\mathcal{O}(\log\log n + Q(k, \ell, \sigma) + |P|/\log_\sigma n)$ time.*

Proof. We use the following definitions. Let S be a τ-synchronizing set of T of size $|\mathsf{S}| = \mathcal{O}(\frac{n}{\tau})$ constructed using Theorem 2.20. Denote $n' = |\mathsf{S}|$. Denote $(s_i)_{i \in [1\,..\,n']} = \operatorname{LexSorted}(\mathsf{S}, T)$ (Definition 7.13). Let $k = \max(n', k') = \Theta(n/\log_\sigma n)$, where $k' = \lceil n/\log_\sigma n \rceil$, and let $A_{\mathrm{str}}[1\,..\,k]$ be an array defined so that for every $i \in [1\,..\,n']$, $A_{\mathrm{str}}[i] = \overline{D_i}$, where $D_i = T^\infty[s_i - \tau\,..\,s_i + 2\tau)$. We leave the remaining element initialized arbitrarily. Denote $\ell = 3\tau$, and note that by the same analysis as in Theorem 4.18, it holds $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$.

Components The data structure consists of the following components:

1. The structure from Proposition 7.10 for text T. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
2. The structure $\operatorname{NavNonperiodic}(\mathsf{S}, \tau, T)$ from Proposition 7.21. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
3. The structure from the claim (answering prefix range emptiness queries) for the sequence $A_{\mathrm{str}}[1\,..\,k]$. It needs $\mathcal{O}(S(k, \ell, \sigma))$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n + S(k, \ell, \sigma))$ space.

Implementation of queries The queries are answered as follows.

1. Let $P_1, P_2 \in [0\,..\,\sigma)^+$ be τ-nonperiodic patterns satisfying $\operatorname{lcp}(P_1, P_2) \geq 3\tau - 1$. Denote $P' = P_1[1\,..\,3\tau - 1] = P_2[1\,..\,3\tau - 1]$. Given the packed representation of P_1 and P_2, we determine if $\operatorname{LexRange}(P_1, P_2, T) \neq \emptyset$ as follows:
   (a) First, using Proposition 7.10, in $\mathcal{O}(1)$ time we compute the values of $\operatorname{RangeBeg}(P', T)$ and $\operatorname{RangeEnd}(P', T)$. This lets us determine $|\operatorname{Occ}(P', T)|$. If $\operatorname{Occ}(P', T) = \emptyset$, then we return that $\operatorname{LexRange}(P_1, P_2, T) = \emptyset$ (since any element of this set would have P' as a prefix), and finish the query algorithm. Let us now assume that $\operatorname{Occ}(P', T) \neq \emptyset$.
   (b) Using Proposition 7.21(2a), in $\mathcal{O}(1)$ time compute the packed representation of the string $D = \operatorname{DistPrefix}(P_1, \tau, T, \mathsf{S})$ (Definition 7.20). Note that since $\operatorname{lcp}(P_1, P_2) \geq 3\tau - 1 \geq |D|$, D is also a prefix of P_2 (and by Lemma 7.19 no other string in $\mathcal{D}(\tau, T, \mathsf{S})$ is a prefix of P_2). In $\mathcal{O}(1)$ time we then calculate $\delta_{\mathrm{text}} = |D| - 2\tau$.
   (c) Using Proposition 7.21(1), in $\mathcal{O}(1)$ time compute the packed representation of $\overline{D}$.
   (d) Denote $P'_k = P_k(\delta_{\mathrm{text}}\,..\,|P_k|]$, where $k \in \{1, 2\}$. Using Proposition 7.21(2b), compute the value $b_k = |\{i \in [1\,..\,n'] : T[s_i\,..\,n] \prec P'_k\}|$ for $k \in \{1, 2\}$. This takes $\mathcal{O}(\log\log n + |P'_1|/\log_\sigma n + |P'_2|/\log_\sigma n) = \mathcal{O}(\log\log n + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.

(e) Using the structure from the claim, in $\mathcal{O}(Q(k,\ell,\sigma))$ time we check if there exists $i \in (b_1\,..\,b_2]$ such that $\overline{D}$ is a prefix of $A_{\mathrm{str}}[i]$. By Lemma 8.6, this is equivalent to checking if $\mathrm{LexRange}(P_1,P_2,T) \neq \emptyset$. Note that the array defined here is padded with extra strings compared to the array in Lemma 8.6, but this does not affect the result of the above query, since $b_1, b_2 \leq n'$.

In total, we spend $\mathcal{O}(\log\log n + Q(k,\ell,\sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.

2. Let $P \in [0\,..\,\sigma)^+$ be a τ-nonperiodic pattern satisfying $|P| \geq 3\tau - 1$. To determine whether it holds $\mathrm{LexRange}(P, P'c^\infty, T) \neq \emptyset$ (where $P' = P[1\,..\,3\tau-1]$ and $c = \sigma - 1$), we set $P_1 = P$ and $P_2 = P'c^\infty$, and then proceed similarly as above, except we compute $b_2 = |\{i \in [1\,..\,n'] : T[s_i\,..\,n] \prec P_2'\}|$ using the second integer computed in Proposition 7.21(2b) (which requires only the packed representation of P'). The whole query takes $\mathcal{O}(\log\log n + Q(k,\ell,\sigma) + |P|/\log_\sigma n)$ time.

Construction algorithm The components of the structure are constructed as follows:

1. We apply Proposition 7.10. This uses $\mathcal{O}(n/\log_\sigma n)$ time and working space.
2. Using Theorem 2.20, we compute the τ-synchronizing set S satisfying $|\mathsf{S}| = \mathcal{O}(\frac{n}{\tau}) = \mathcal{O}(n/\log_\sigma n)$ in $\mathcal{O}(n/\log_\sigma n)$ time. Then, using S and the packed representation of T as input, we construct $\mathrm{NavNonperiodic}(\mathsf{S},\tau,T)$ in $\mathcal{O}(n/\log_\sigma n)$ time using Proposition 7.21.
3. To construct the next component of the structure, we proceed as follows:
   (a) Using Theorem 7.22, in $\mathcal{O}(\frac{n}{\tau}) = \mathcal{O}(n/\log_\sigma n)$ time we first compute the sequence $(s_i)_{i\in[1\,..\,n']} = \mathrm{LexSorted}(\mathsf{S},T)$ (Definition 7.13).
   (b) Using Proposition 7.21(1), in $\mathcal{O}(n') = \mathcal{O}(n/\log_\sigma n)$ time we compute the elements of the array A_{str} at indexes $i \in [1\,..\,n']$, i.e., $A_{\mathrm{str}}[i] = \overline{D_i}$, where $D_i = T^\infty[s_i - \tau\,..\,s_i + 2\tau)$. In $\mathcal{O}(k-n') = \mathcal{O}(k) = \mathcal{O}(n/\log_\sigma n)$ time we then pad the remaining elements of A_{str} at indexes $i \in (n'\,..\,k]$.
   (c) Finally, we apply the preprocessing from the claim to the array A_{str}. This takes $\mathcal{O}(P_t(k,\ell,\sigma))$ time and uses $\mathcal{O}(P_s(k,\ell,\sigma))$ working space.

   In total, constructing this component of the structure takes $\mathcal{O}(n/\log_\sigma n + P_t(k,\ell,\sigma))$ time and uses $\mathcal{O}(n/\log_\sigma n + P_s(k,\ell,\sigma))$ working space.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n + P_t(k,\ell,\sigma))$ time and uses $\mathcal{O}(n/\log_\sigma n + P_s(k,\ell,\sigma))$ working space. □

8.3.1.3 The Periodic Patterns

Proposition 8.8. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that answers the following queries:*

1. *Given the packed representation of τ-periodic (Definition 7.5) patterns $P_1, P_2 \in [0\,..\,\sigma)^+$ satisfying $\mathrm{lcp}(P_1,P_2) \geq 3\tau - 1$, determines if $\mathrm{LexRange}(P_1,P_2,T) \neq \emptyset$ (Definition 2.4) in $\mathcal{O}(\log\log n + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.*
2. *Given the packed representation of a τ-periodic pattern $P \in [0\,..\,\sigma)^+$, determines whether it holds $\mathrm{LexRange}(P, P'c^\infty, T) \neq \emptyset$ (where $P' = P[1\,..\,3\tau-1]$ and $c = \sigma - 1$) in $\mathcal{O}(\log\log n + |P|/\log_\sigma n)$ time.*

Proof. **Components** The data structure consists of a single component: the structure from Proposition 7.53. It needs $\mathcal{O}(n/\log_\sigma n)$ space.

Implementation of queries The queries are answered as follows:

1. Let $P_1, P_2 \in [0\,..\,\sigma)^+$ be τ-periodic patterns. Using Proposition 7.53(4) and the packed representation of P_1 and P_2, in $\mathcal{O}(\log\log n + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time we compute the values $b_1 = \mathrm{RangeBeg}(P_1,T)$ and $b_2 = \mathrm{RangeBeg}(P_2,T)$. If $b_1 < b_2$, we have $\mathrm{LexRange}(P_1,P_2,T) \neq \emptyset$. Otherwise, we return that $\mathrm{LexRange}(P_1,P_2,T) = \emptyset$. The correctness follows by Remark 2.5. In total, we spend $\mathcal{O}(\log\log n + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.
2. Let $P \in [0\,..\,\sigma)^+$ be a τ-periodic pattern. Observe that it holds $\mathrm{RangeBeg}(P'c^\infty,T) = \mathrm{RangeEnd}(P',T)$. Thus, the query is answered as above, except the value b_2 is computed as $\mathrm{RangeEnd}(P',T)$ using

Proposition 7.53(4). In total, we spend $\mathcal{O}(\log\log n+|P|/\log_\sigma n+|P'|/\log_\sigma n)=\mathcal{O}(\log\log n+|P|/\log_\sigma n)$ time.

Construction algorithm The construction of the structure takes $\mathcal{O}(n/\log_\sigma n)$ time by Proposition 7.53. □

8.3.1.4 Summary

Proposition 8.9. *Consider a data structure answering prefix range emptiness queries (see Section 8.1) that, for any sequence W of k strings of length ℓ over alphabet $[0\,..\,\sigma)$ achieves the following complexities (where the input strings during construction are given in the packed representation, and at query time we are given any $b,e\in[0\,..\,k]$ and the packed representation of any $X\in[0\,..\,\sigma)^{\leq\ell}$, and we return a bit indicating whether $\{i\in(b\,..\,e]:X$ is a prefix of $W[i]\}\neq\emptyset$):*

- *space usage $S(k,\ell,\sigma)$,*
- *preprocessing time $P_t(k,\ell,\sigma)$,*
- *preprocessing space $P_s(k,\ell,\sigma)$,*
- *query time $Q(k,\ell,\sigma)$.*

Let $T\in[0\,..\,\sigma)^n$ be such that $2\leq\sigma<n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. There exist positive integers $k=\Theta(n/\log_\sigma n)$ and $\ell\leq(1+\lfloor\log k\rfloor)/\lceil\log\sigma\rceil$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n+P_t(k,\ell,\sigma))$ time and using $\mathcal{O}(n/\log_\sigma n+P_s(k,\ell,\sigma))$ working space construct a data structure of size $\mathcal{O}(n/\log_\sigma n+S(k,\ell,\sigma))$ that, given the packed representation of any patterns $P_1,P_2\in[0\,..\,\sigma)^*$, determines if $\mathrm{LexRange}(P_1,P_2,T)\neq\emptyset$ (see Definition 2.4) in $\mathcal{O}(\log\log n+Q(k,\ell,\sigma)+|P_1|/\log_\sigma n+|P_2|/\log_\sigma n)$ time.

Proof. We use the following definitions. Let $\tau=\lfloor\mu\log_\sigma n\rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau\geq1$ (such τ exists by the assumption $\sigma<n^{1/13}$). Let k and ℓ be integers resulting from the application of Proposition 8.7 to text T, with the structure from the above claim to answer prefix range emptiness queries. Note that they satisfy $k=\Theta(n/\log_\sigma n)$ and $\ell\leq(1+\lfloor\log k\rfloor)/\lceil\log\sigma\rceil$.

Components The data structure consists of the following components:

1. The structure from Proposition 7.8 for σ and τ. It needs $\mathcal{O}(\sigma^{3\tau}\cdot\tau^2)=\mathcal{O}(n/\log_\sigma n)$ space.
2. The structure from Proposition 7.10 for τ and T. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
3. The structure from Proposition 8.5 for τ and T. It uses $\mathcal{O}(n/\log_\sigma n)$ space.
4. The structure from Proposition 8.7 for τ and T, and using the structure from the above claim to answer prefix range emptiness queries. It uses $\mathcal{O}(n/\log_\sigma n+S(k,\ell,\sigma))$ space, where k and ℓ are defined above.
5. The structure from Proposition 8.8 for τ and T. It uses $\mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n+S(k,\ell,\sigma))$ space.

Implementation of queries The queries are answered as follows. Let $P_1,P_2\in[0\,..\,\sigma)^*$. Given the packed representation of P_1 and P_2, we determine if $\mathrm{LexRange}(P_1,P_2,T)\neq\emptyset$ as follows. First, in $\mathcal{O}(1+|P_1|/\log_\sigma n+|P_2|/\log_\sigma n)$ time we check if $P_1\prec P_2$. If not, then we return that $\mathrm{LexRange}(P_1,P_2,T)=\emptyset$ (see Definition 2.4), and the query algorithm is complete. Let us thus assume that it holds $P_1\prec P_2$. In $\mathcal{O}(1)$ time we use the packed representation of P_1 and P_2 to compute the packed representation of prefixes $X_1=P_1[1\,..\,\min(3\tau-1,|P_1|)]$ and $X_2=P_2[1\,..\,\min(3\tau-1,|P_2|)]$. Note that the assumption $P_1\prec P_2$ implies that $X_1\preceq X_2$. In $\mathcal{O}(1)$ time we check if $X_1=X_2$. In $\mathcal{O}(1)$ time we also check if X_1 is a prefix of X_2. We then consider three cases:

- First, assume that $X_1=X_2$. Note that $|X_1|=|X_2|=3\tau-1$, since otherwise $P_1=X_1=X_2=P_2$, and the algorithm would have already finished. Using Proposition 7.8, in $\mathcal{O}(1)$ time we check if X_1 is τ-periodic (Definition 7.5). Consider two cases:
  - If X_1 is τ-periodic, then so are X_2, P_1, and P_2. We then check if $\mathrm{LexRange}(P_1,P_2,T)\neq\emptyset$ using Proposition 8.8(1) in $\mathcal{O}(\log\log n+|P_1|/\log_\sigma n+|P_2|/\log_\sigma n)$ time.
  - Otherwise, P_1 and P_2 are τ-nonperiodic. Since $\mathrm{lcp}(P_1,P_2)\geq3\tau-1$, we then check whether it holds $\mathrm{LexRange}(P_1,P_2,T)\neq\emptyset$ in $\mathcal{O}(\log\log n+Q(k,\ell,\sigma)+|P_1|/\log_\sigma n+|P_2|/\log_\sigma n)$ time using Proposition 8.7(1).

In total, we spend $\mathcal{O}(\log\log n + Q(k,\ell,\sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.

- Let us now assume that $X_1 \neq X_2$ and X_1 is a prefix of X_2. Note that then X_1 must be a proper prefix of X_2, and hence $|X_1| < |X_2|$. Thus, we must have $P_1 = X_1$. Combining this with $X_1 \preceq X_2$ (see above), we obtain $P_1 \preceq X_2 \preceq P_2$. Consequently, we can write $\mathrm{LexRange}(P_1, P_2, T)$ as a disjoint union:

$$\mathrm{LexRange}(P_1, P_2, T) = \mathrm{LexRange}(P_1, X_2, T) \cup \mathrm{LexRange}(X_2, P_2, T).$$

  We will separately check the emptiness of each of the two sets. Based on this, we can easily determine if $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$. Using Proposition 8.5(1), we check if $\mathrm{LexRange}(X_1, X_2, T) \neq \emptyset$ (recall that $P_1 = X_1$) in $\mathcal{O}(1)$ time. Let us now focus on checking if $\mathrm{LexRange}(X_2, P_2, T) \neq \emptyset$. If $|P_2| < 3\tau - 1$, then again we check if $\mathrm{LexRange}(X_2, P_2, T) \neq \emptyset$ using Proposition 8.5(1) in $\mathcal{O}(1)$ time. Let us now assume that $|P_2| \geq 3\tau - 1$. Using Proposition 7.8, in $\mathcal{O}(1)$ time we check if P_2 is τ-periodic (Definition 7.5). Consider two cases:

  - If P_2 is τ-periodic, then using Proposition 8.8(1), we check if $\mathrm{LexRange}(X_2, P_2, T) \neq \emptyset$ in $\mathcal{O}(\log\log n + |P_2|/\log_\sigma n)$ time. Note that we can use this because $X_2 = P_2[1\,..\,3\tau - 1]$.
  - If P_2 is τ-nonperiodic, then using Proposition 8.7(1), we check if $\mathrm{LexRange}(X_2, P_2, T) \neq \emptyset$ in $\mathcal{O}(\log\log n + Q(k,\ell,\sigma) + |P_2|/\log_\sigma n)$ time.

  In total, we spend $\mathcal{O}(\log\log n + Q(k,\ell,\sigma) + |P_2|/\log_\sigma n)$ time.
- Let us now assume that $X_1 \neq X_2$ and X_1 is not a prefix of X_2. Observe that, letting $c = \sigma - 1$, we then have $X_1 c^\infty \prec X_2$. Combining this with $P_1 \preceq X_1 c^\infty$ and $X_2 \preceq P_2$, we can thus write $\mathrm{LexRange}(P_1, P_2, T)$ as a disjoint union:

$$\begin{aligned}\mathrm{LexRange}(P_1, P_2, T) = {} & \mathrm{LexRange}(P_1, X_1 c^\infty, T) \cup \\ & \mathrm{LexRange}(X_1 c^\infty, X_2, T) \cup \\ & \mathrm{LexRange}(X_2, P_2, T).\end{aligned}$$

  We will separately check the emptiness of each of the three sets. Based on this, we can easily determine if $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$. We begin with the set $\mathrm{LexRange}(P_1, X_1 c^\infty, T)$. If $|P_1| \leq 3\tau - 1$, then $P_1 = X_1$, and hence we check if $\mathrm{LexRange}(P_1, X_1 c^\infty, T) \neq \emptyset$ in $\mathcal{O}(1)$ time using Proposition 8.5(2). Let us now assume that $|P_1| > 3\tau - 1$. In $\mathcal{O}(1)$ time we check if P_1 is τ-periodic using Proposition 7.8. We then consider two cases:

  - If P_1 is τ-periodic, then we check if $\mathrm{LexRange}(P_1, X_1 c^\infty, T) \neq \emptyset$ using Proposition 8.8(2) in $\mathcal{O}(\log\log n + |P_1|/\log_\sigma n)$ time.
  - Otherwise (i.e., if P_1 is τ-nonperiodic), we check if $\mathrm{LexRange}(P_1, X_1 c^\infty, T) \neq \emptyset$ using Proposition 8.7(2) in $\mathcal{O}(\log\log n + Q(k,\ell,\sigma) + |P_1|/\log_\sigma n)$ time.

  Next, we check if $\mathrm{LexRange}(X_1 c^\infty, X_2, T) \neq \emptyset$. To this end, note that $\mathrm{RangeBeg}(X_1 c^\infty, T) = \mathrm{RangeEnd}(X_1, T)$. We thus proceed as follows. Using Proposition 7.10, in $\mathcal{O}(1)$ time we compute $e_1 = \mathrm{RangeEnd}(X_1, T) = \mathrm{RangeBeg}(X_1 c^\infty, T)$ and $b_2 = \mathrm{RangeBeg}(X_2, T)$. If $e_1 < b_2$, we return that $\mathrm{LexRange}(X_1 c^\infty, X_2, T) \neq \emptyset$. Otherwise, we have $\mathrm{LexRange}(X_1 c^\infty, X_2, T) = \emptyset$. The correctness of this follows by Remark 2.5. Finally, we address the problem of checking if $\mathrm{LexRange}(X_2, P_2, T) \neq \emptyset$. If $|P_2| \leq 3\tau - 1$, then $P_2 = X_2$, and hence we have $\mathrm{LexRange}(X_2, P_2, T) = \emptyset$. Let us now assume that $|P_2| > 3\tau - 1$. In $\mathcal{O}(1)$ time we check if P_2 is τ-periodic using Proposition 7.8. We then consider two cases:

  - If P_2 is τ-nonperiodic, then we check if $\mathrm{LexRange}(X_2, P_2, T) \neq \emptyset$ using Proposition 8.7(1) in $\mathcal{O}(\log\log n + Q(k,\ell,\sigma) + |P_2|/\log_\sigma n)$ time.
  - Otherwise (i.e., if P_2 is τ-periodic), we check if $\mathrm{LexRange}(X_2, P_2, T) \neq \emptyset$ using Proposition 8.8(1) in $\mathcal{O}(\log\log n + |P_2|/\log_\sigma n)$ time.

  We spend $\mathcal{O}(\log\log n + Q(k,\ell,\sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time in the above step.

Over all steps, we spend $\mathcal{O}(\log\log n + Q(k,\ell,\sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.

Construction algorithm The components of the structure are constructed as follows:

1. In $\mathcal{O}(\sigma^{3\tau} \cdot \tau^2) = \mathcal{O}(n/\log_\sigma n)$ time we apply Proposition 7.8.

2. In $\mathcal{O}(n/\log_\sigma n)$ time we apply Proposition 7.10 to text T.
3. In the same time as above we apply Proposition 8.5 to text T.
4. In $\mathcal{O}(n/\log_\sigma n + P_t(k,\ell,\sigma))$ time and using $\mathcal{O}(n/\log_\sigma n + P_s(k,\ell,\sigma))$ working space we apply Proposition 8.7 to text T.
5. In $\mathcal{O}(n/\log_\sigma n)$ time we apply Proposition 8.8 to text T.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n + P_t(k,\ell,\sigma))$ time uses $\mathcal{O}(n/\log_\sigma n + P_s(k,\ell,\sigma))$ working space. □

8.3.2 Alphabet Reduction for Prefix Range Emptiness Queries

Proposition 8.10. *Consider a data structure answering prefix range emptiness queries (see Section 8.1) that, for any sequence of $m \geq 1$ binary strings of length $1 + \lfloor \log m \rfloor$, achieves the following complexities (where both the string at query time as well as input strings during construction are given in the packed representation):*

- *space usage $S(m)$,*
- *preprocessing time $P_t(m)$,*
- *preprocessing space $P_s(m)$,*
- *query time $Q(m)$.*

Let $W[1\,..\,m']$ be a sequence of $m' \geq 1$ equal-length strings over alphabet $[0\,..\,\sigma)$ (where $\sigma \geq 2$) of length $\ell \leq (1 + \lfloor \log m' \rfloor)/\lceil \log \sigma \rceil$. Given the sequence W, with all strings represented in the packed form, we can in $\mathcal{O}(m' + P_t(m'))$ time and using $\mathcal{O}(m' + P_s(m'))$ working space construct a data structure of size $\mathcal{O}(m' + S(m'))$ that, given the packed representation of any $X \in [0\,..\,\sigma)^{\leq \ell}$ and any $b, e \in [0\,..\,m']$, checks if there exists $i \in (b\,..\,e]$ such that X is a prefix of $W[i]$, in $\mathcal{O}(Q(m'))$ time.

Proof. The above reduction is essentially the same as the one presented in Proposition 4.20, except we only need Lemma 4.19(1c). □

8.3.3 Summary

Theorem 8.11. *Consider a data structure answering prefix range emptiness queries (see Section 8.1) that, for any sequence of $m \geq 1$ binary strings of length $1 + \lfloor \log m \rfloor$, achieves the following complexities (where both the string at query time as well as input strings during construction are given in the packed representation):*

- *space usage $S(m)$,*
- *preprocessing time $P_t(m)$,*
- *preprocessing space $P_s(m)$,*
- *query time $Q(m)$.*

For every $T \in \{\mathtt{0},\mathtt{1}\}^n$, there exists $m = \Theta(n/\log n)$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log n + P_t(m))$ time and using $\mathcal{O}(n/\log n + P_s(m))$ working space construct a data structure of size $\mathcal{O}(n/\log n + S(m))$ that, given the packed representation of any $P_1, P_2 \in \{\mathtt{0},\mathtt{1}\}^*$, determines whether $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$ (see Definition 2.4) in $\mathcal{O}(\log\log n + Q(m) + |P_1|/\log n + |P_2|/\log n)$ time.

Proof. Assume that $n > 3^{13} - 1$ (otherwise, the claim holds trivially).

We use the following definitions. Let $\sigma = 3$ and let $T' \in [0\,..\,\sigma)^{n+1}$ be a string defined so that $T'[1\,..\,n] = T$ and $T'[n+1] = \mathtt{2}$. Denote $n' = |T'|$. Note that $T'[n']$ does not occur in $T'[1\,..\,n')$ and it holds $2 \leq \sigma < (n')^{1/13}$. Let D denote the data structure resulting from applying Proposition 8.10 with alphabet size σ to the structure (answering prefix range emptiness queries on binary strings) form the claim. Given any sequence $W[1\,..\,m']$ of $m' \geq 1$ equal-length strings over alphabet $[0\,..\,\sigma)$ of length $\ell' \leq (1 + \lfloor \log m' \rfloor)/\lceil \log \sigma \rceil$ as input (with all strings represented in the packed form), the structure D achieves the following complexities (where at query time it is given the packed representation of any $X \in [0\,..\,\sigma)^{\leq \ell'}$ and any $b, e \in [0\,..\,m']$, and determines whether there exists $i \in (b\,..\,e]$ such that X is a prefix of $W[i]$):

- space usage $S'(m', \ell', \sigma) = \mathcal{O}(m' + S(m'))$,
- preprocessing time $P'_t(m', \ell', \sigma) = \mathcal{O}(m' + P_t(m'))$,
- preprocessing space $P'_s(m', \ell', \sigma) = \mathcal{O}(m' + P_s(m'))$,
- query time $Q'(m', \ell', \sigma) = \mathcal{O}(Q(m'))$,

where the complexities $S(m')$, $P_t(m')$, $P_s(m')$, and $Q(m')$ refer to the structure from the claim (answering prefix range emptiness queries for binary strings).

Components The data structure answering lex-range emptiness queries consists of a single component: the data structure from Proposition 8.9 applied for text T' and with the structure D as the structure answering prefix range emptiness queries. Note that we can use D, since it supports prefix range emptiness queries for the combination of parameters required in Proposition 8.9, i.e., for sequences over alphabet $[0\,..\,\sigma)$, with the length ℓ of all strings satisfying $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ (where k is the length of the input sequence). To bound the space usage of this component, note that by Proposition 8.9 and the above discussion, there exists $m = \Theta(n'/\log_\sigma n') = \Theta(n/\log n)$ (recall that $\sigma = 3$) such that the structure needs $\mathcal{O}(n'/\log_\sigma n' + S'(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + S(m)) = \mathcal{O}(n/\log n + S(m))$ space.

Implementation of queries The lex-range emptiness queries are answered as follows. Let $P_1, P_2 \in \{\mathtt{0},\mathtt{1}\}^*$. Note that by definition of T', it follows that $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$ if and only if $\mathrm{LexRange}(P_1, P_2, T') \neq \emptyset$. By Proposition 8.9 and the above discussion, determining if $\mathrm{LexRange}(P_1, P_2, T') \neq \emptyset$ takes $\mathcal{O}(\log\log n' + Q'(m,\ell,\sigma) + |P_1|/\log_\sigma n' + |P_2|/\log_\sigma n') = \mathcal{O}(\log\log n + Q(m) + |P_1|/\log n + |P_2|/\log n)$ time.

Construction algorithm By Proposition 8.9 and the above discussion, construction of the above data structure answering lex-range emptiness queries on T' (and hence also on T) takes $\mathcal{O}(n'/\log_\sigma n' + P'_t(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + P_t(m)) = \mathcal{O}(n/\log n + P_t(m))$ time and uses $\mathcal{O}(n'/\log_\sigma n' + P'_s(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + P_s(m)) = \mathcal{O}(n/\log n + P_s(m))$ working space. □

9 Equivalence of Lex-Range Minimum and Prefix RMQ

9.1 Problem Definitions

Indexing for Lex-Range Minimum Queries

Input: The packed representation of a nonempty string $T \in \{\mathtt{0},\mathtt{1}\}^n$.
Output: A data structure that, given the packed representation of any $P_1, P_2 \in \{\mathtt{0},\mathtt{1}\}^*$, returns the value (see Definition 2.4)

$$\min \mathrm{LexRange}(P_1, P_2, T) \cup \{\infty\}.$$

Indexing for Prefix RMQ Queries

Input: A nonempty sequence $A[1\,..\,m]$ of m nonnegative integers (with $A[i] \in [1\,..\,m]$ for $i \in [1\,..\,m]$) and a sequence $W[1\,..\,m]$ of m binary strings of length $\ell = 1 + \lfloor \log m \rfloor$, with all strings represented in the packed form.
Output: A data structure that, given the packed representation of any $X \in \{\mathtt{0},\mathtt{1}\}^{\leq \ell}$ and any indexes $b, e \in [0\,..\,m]$, returns $\mathsf{prefix\text{-}rmq}_{A,W}(b, e, X)$ (see Definition 2.9).

9.2 Reducing Prefix RMQ to Lex-Range Minimum Queries

9.2.1 Preliminaries

Definition 9.1 (Permuted sequence-to-string mapping)**.** Let $W[1\,..\,m]$ be a sequence of $m \geq 1$ strings of length $\ell \geq 1$ over alphabet $\{\mathtt{0},\mathtt{1}\}$. Let $\pi[1\,..\,m]$ be an array containing a permutation of $[1\,..\,m]$. Denote $k = 1 + \lfloor \log m \rfloor$. For any $i \in [0\,..\,m]$, let $s(i) = \mathrm{sub}(\mathrm{sub}(\mathrm{bin}_k(i), \mathtt{0}, \mathtt{2}), \mathtt{1}, \mathtt{3})$ (see Definitions 4.1 and 4.6). We then define

$$\mathrm{PermSeqToString}_\ell(\pi, W) := \bigodot_{i=1}^{m} \overline{W[\pi[i]]} \cdot \mathtt{4} \cdot s(\pi[i] - 1).$$

Lemma 9.2. *Let $W[1\,..\,m]$ be a sequence of $m \geq 1$ strings of length $\ell \geq 1$ over alphabet $\{\mathtt{0},\mathtt{1}\}$. Let $\pi[1\,..\,m]$ be an array containing a permutation of $[1\,..\,m]$, and let $\pi^{-1}[1\,..\,m]$ be the inverse permutation (i.e., such*

that for every $i \in [1\,..\,m]$, $\pi[\pi^{-1}[i]] = i$). Let $T = \mathrm{PermSeqToString}_\ell(\pi, W)$ (Definition 9.1), $k = 1 + \lfloor \log m \rfloor$, and $\beta = \ell + k + 1$. Finally, let $X \in \{0,1\}^{\le \ell}$ and $\mathcal{P} = \{i \in [1\,..\,m] : X$ is a prefix of $W[i]\}$. Then, it holds

$$\mathrm{Occ}(\overline{X} \cdot 4, T) = \{\pi^{-1}[i] \cdot \beta - (k + |X|) : i \in \mathcal{P}\}.$$

Proof. Denote $A = \{\pi^{-1}[i] \cdot \beta - (k + |X|) : i \in \mathcal{P}\}$.

First, we show the inclusion $\mathrm{Occ}(\overline{X} \cdot 4, T) \subseteq A$. Let $j \in \mathrm{Occ}(\overline{X} \cdot 4, T)$. Note that $\mathrm{Occ}(4, T) = \{i \cdot \beta - k : i \in [1\,..\,m]\}$. Thus, $j \in \mathrm{Occ}(\overline{X} \cdot 4, T)$ implies that for some $i \in [1\,..\,m]$, it holds $j = i \cdot \beta - (k + |X|)$. Since $T[i \cdot \beta - k\,..\,|T|]$ is preceded in T with $\overline{W[\pi[i]]}$, it follows that $\overline{X}$ is a suffix of $\overline{W[\pi[i]]}$, or equivalently, X is a prefix of $W[\pi[i]]$. This implies that $\pi[i] \in \mathcal{P}$, and hence, by definition of A, $\pi^{-1}[\pi[i]] \cdot \beta - (k + |X|) \in A$. It remains to note that $\pi^{-1}[\pi[i]] \cdot \beta - (k + |X|) = i \cdot \beta - (k + |X|) = j$.

We now show the opposite inclusion. Let $j \in A$. Then, there exists $i \in \mathcal{P}$ such that $j = \pi^{-1}[i] \cdot \beta - (k + |X|)$. By $i \in \mathcal{P}$, it holds $i \in [1\,..\,m]$ and X is a prefix of $W[i]$. Note that, by Definition 9.1, $T(\pi^{-1}[i] \cdot \beta\,..\,|T|]$ is preceded in T with $\overline{W[\pi[\pi^{-1}[i]]]} \cdot 4 \cdot s(\pi[\pi^{-1}[i]] - 1) = \overline{W[i]} \cdot 4 \cdot s(i-1)$. Since X is a prefix of $W[i]$, or equivalently, $\overline{X}$ is a suffix of $\overline{W[i]}$, we thus obtain that $T(\pi^{-1}[i] \cdot \beta\,..\,|T|]$ is preceded in T with $\overline{X} \cdot 4 \cdot s(i-1)$. Thus, $\pi^{-1}[i] \cdot \beta - (k + |X|) \in \mathrm{Occ}(\overline{X} \cdot 4, T)$. By $j = \pi^{-1}[i] \cdot \beta - (k + |X|)$, we thus obtain $j \in \mathrm{Occ}(\overline{X} \cdot 4, T)$. □

Lemma 9.3. *Let $W[1\,..\,m]$ be a sequence of $m \ge 1$ strings of length $\ell \ge 1$ over alphabet $\{0,1\}$. Let $\pi[1\,..\,m]$ be an array containing a permutation of $[1\,..\,m]$. Let $s(i)$ (where $i \in [0\,..\,m]$) be as in Definition 9.1. Denote $T = \mathrm{PermSeqToString}_\ell(\pi, W)$ (Definition 9.1). Consider a string $X \in \{0,1\}^{\le \ell}$ and let $b, e \in [0\,..\,m]$. Denote $P_1 = \overline{X} \cdot 4 \cdot s(b)$, $P_2 = \overline{X} \cdot 4 \cdot s(e)$, and $\beta = \ell + \lfloor \log m \rfloor + 2$. Finally, let*

- $\mathcal{P} = \{i \in (b\,..\,e] : X$ *is a prefix of* $W[i]\}$,
- $\mathcal{Q} = \{\lceil j/\beta \rceil : j \in \mathrm{LexRange}(P_1, P_2, T)\}$ *(Definition 2.4).*

Then, it holds

$$\mathcal{P} = \{\pi[q] : q \in \mathcal{Q}\}.$$

Proof. Denote $k = 1 + \lfloor \log m \rfloor$.

First, we prove the inclusion $\mathcal{P} \subseteq \{\pi[q] : q \in \mathcal{Q}\}$. Let $i \in \mathcal{P}$. Then, $i \in (b\,..\,e]$ and X is a prefix of $W[i]$. Let $\pi^{-1}[1\,..\,m]$ be an array containing the inverse permutation of π, i.e., such that for every $i \in [1\,..\,m]$, it holds $\pi^{-1}[\pi[i]] = i$. Denote $i' = \pi^{-1}[i]$ and let $j = i' \cdot \beta - (k + |X|)$. Observe that, by Definition 9.1, the string $T[i' \cdot \beta + 1\,..\,|T|]$ is preceded in T with $\overline{W[\pi[i']]} \cdot 4 \cdot s(\pi[i'] - 1) = \overline{W[i]} \cdot 4 \cdot s(i-1)$. Since X is a prefix of $W[i]$, we thus obtain that $T[j\,..\,|T|]$ has $\overline{X} \cdot 4 \cdot s(i-1)$ as a prefix. The assumption $b < i \le e$, or equivalently, $b \le i - 1 < e$, implies that $s(b) \preceq s(i-1) \prec s(e)$. We thus obtain that $P_1 \preceq \overline{X} \cdot 4 \cdot s(i-1) \prec P_2$, and hence $P_1 \preceq T[j\,..\,|T|] \prec P_2$. We have thus proved that $j \in \mathrm{LexRange}(P_1, P_2, T)$. Since $\lceil j/\beta \rceil = i'$, we thus obtain $i' \in \mathcal{Q}$, and hence $\pi[i'] = \pi[\pi^{-1}[i]] = i \in \{\pi[q] : q \in \mathcal{Q}\}$.

We now prove the second inclusion. Let $q' \in \{\pi[q] : q \in \mathcal{Q}\}$. Let $q \in \mathcal{Q}$ be such that $q' = \pi[q]$. Then, there exists $j \in \mathrm{LexRange}(P_1, P_2, T)$ such that $\lceil j/\beta \rceil = q$. The assumption $j \in \mathrm{LexRange}(P_1, P_2, T)$ implies that $P_1 \preceq T[j\,..\,|T|] \prec P_2$. By definition of P_1 and P_2, this implies that $T[j\,..\,|T|]$ has the string $\overline{X} \cdot 4$ as a prefix. Consequently, by Lemma 9.2, there exists $i \in [1\,..\,m]$ such that X is a prefix of $W[i]$ and $j = \pi^{-1}[i] \cdot \beta - (k + |X|)$. By Definition 9.1, the suffix $T(\pi^{-1}[i] \cdot \beta - k\,..\,|T|]$ has the string $s(\pi[\pi^{-1}[i]] - 1) = s(i-1)$ as a prefix. Putting together with the previous observation, we thus obtain that $T[j\,..\,|T|]$ has $\overline{X} \cdot 4 \cdot s(i-1)$ as a prefix. From the definition of P_1 and P_2, and the assumption $P_1 \preceq T[j\,..\,|T|] \prec P_2$, this implies that $s(b) \preceq s(i-1) \prec s(e)$, and hence $b \le i - 1 < e$, or equivalently, $b < i \le e$. Combining this with X being a prefix of $W[i]$ (see above), we obtain $i \in \mathcal{P}$. It remains to note that $j = \pi^{-1}[i] \cdot \beta - (k + |X|)$ implies that $q = \lceil j/\beta \rceil = \pi^{-1}[i]$. Hence, $q' = \pi[q] = \pi[\pi^{-1}[i]] = i$. We have thus proved that $q' \in \mathcal{P}$. □

Proposition 9.4. *Let $W[1\,..\,m]$ be an array of $m \ge 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$ and let $\pi[1\,..\,m]$ be an array containing a permutation of $[1\,..\,m]$. Given arrays W and π, with all strings in W represented in the packed form, we can in $\mathcal{O}(m)$ time compute the packed representation of the string $\mathrm{PermSeqToString}_\ell(\pi, W)$ (Definition 9.1).*

Proof. The construction proceeds as in Proposition 4.12, except in the last step, we additionally use the permutation π. □

9.2.2 Problem Reduction

Lemma 9.5. *Let $A[1\,..\,m]$ be an array of $m \geq 1$ integers and let $W[1\,..\,m]$ be a sequence of m strings of length $\ell \geq 1$ over alphabet $\{0,1\}$. Let $s(i)$ (where $i \in [0\,..\,m]$) be as in Definition 9.1. Let $\pi[1\,..\,m]$ be a permutation of $[1\,..\,m]$ such that*

$$A[\pi[1]] \leq A[\pi[2]] \leq \cdots \leq A[\pi[m]].$$

Denote $T = \mathrm{PermSeqToString}_\ell(\pi, W)$ (Definition 9.1). Then, for every $X \in \{0,1\}^{\leq \ell}$ and every $b, e \in [0\,..\,m]$, letting $P_1 = \overline{X} \cdot 4 \cdot s(b)$ and $P_2 = \overline{X} \cdot 4 \cdot s(e)$, the following two conditions are equivalent:

- $\mathsf{prefix\text{-}rmq}_{A,W}(b, e, X) \neq \infty$ *(Definition 2.9),*
- $\min \mathrm{LexRange}(P_1, P_2, T) \cup \{\infty\} \neq \infty$ *(Definition 2.4).*

Moreover, if $\min \mathrm{LexRange}(P_1, P_2, T) \cup \{\infty\} \neq \infty$*, then*

$$\mathsf{prefix\text{-}rmq}_{A,W}(b, e, X) = \pi\left[\left\lceil \frac{\min \mathrm{LexRange}(P_1, P_2, T)}{\beta} \right\rceil\right],$$

where $\beta = \ell + \lfloor \log m \rfloor + 2$.

Proof. Denote

- $\mathcal{P} = \{i \in (b\,..\,e] : X \text{ is a prefix of } W[i]\}$,
- $\mathcal{Q} = \{\lceil j/\beta \rceil : j \in \mathrm{LexRange}(P_1, P_2, T)\}$.

To show the first claim, observe that the following four conditions are equivalent.

1. $\mathsf{prefix\text{-}rmq}_{A,W}(b, e, X) \neq \infty$,
2. $\mathcal{P} \neq \emptyset$,
3. $\mathcal{Q} \neq \emptyset$,
4. $\min \mathrm{LexRange}(P_1, P_2, T) \cup \{\infty\} \neq \infty$.

The equivalence of Conditions 1 and 2 follows by Definition 2.9. The equivalence for 2 and 3 follows by Lemma 9.3. Finally, the equivalence of Conditions 3 and 4 follows simply by definition of Q. Thus, we obtain the equivalence of Conditions 1 and 4, i.e., the claim.

We now show the second claim. Assume that it holds $\min \mathrm{LexRange}(P_1, P_2, T) \cup \{\infty\} \neq \infty$ and $\mathsf{prefix\text{-}rmq}_{A,W}(b, e, X) \neq \infty$. Denote $j_{\min} = \min \mathrm{LexRange}(P_1, P_2, T)$ and let $i = \pi[\lceil j_{\min}/\beta \rceil]$. To show that it holds $i = \mathsf{prefix\text{-}rmq}_{A,W}(b, e, X)$, we need to prove (see Definition 2.9) that $i \in (b\,..\,e]$, X is a prefix of $W[i]$, and for that every $i' \in (b\,..\,e]$ such that X is a prefix of $W[i']$, it holds $A[i] \leq A[i']$.

- By definition of $\mathcal{Q}$, we have $\lceil j_{\min}/\beta \rceil \in \mathcal{Q}$. Consequently, by Lemma 9.3, it holds $\pi[\lceil j_{\min}/\beta \rceil] \in \mathcal{P}$, i.e., $i \in \mathcal{P}$. By definition of $\mathcal{P}$, we thus have $i \in (b\,..\,e]$ and X is a prefix of $W[i]$.
- To show the remaining claim, consider any $i' \in (b\,..\,e]$ such that X is a prefix of $W[i']$. By the definition of $\mathcal{P}$, we have $i' \in \mathcal{P}$. By Lemma 9.3, there exist $j' \in \mathrm{LexRange}(P_1, P_2, T)$ such that $\pi[\lceil j'/\beta \rceil] = i'$. By definition of $j_{\min}$, we have $j_{\min} \leq j'$. Thus, $\lceil j_{\min}/\beta \rceil \leq \lceil j'/\beta \rceil$. By the assumption $A[\pi[1]] \leq A[\pi[2]] \leq \cdots \leq A[\pi[m]]$, we thus obtain that $A[\pi[\lceil j_{\min}/\beta \rceil]] \leq A[\pi[\lceil j'/\beta \rceil]]$. In other words, $A[i] \leq A[i']$. □

9.2.3 Alphabet Reduction for Lex-Range Minimum Queries

Lemma 9.6. *Let $\sigma \geq 2$, $T \in [0\,..\,\sigma)^*$, $P_1 \in [0\,..\,\sigma)^*$, and $P_2 \in [0\,..\,\sigma)^*$. Let $k = \lceil \log \sigma \rceil$, $\delta = 2k + 3$, $P_1' = \mathrm{ebin}_k(P_1)$, $P_2' = \mathrm{ebin}_k(P_2)$, and $T' = \mathrm{ebin}_k(T)$ (Definition 4.3). Then, the following conditions are equivalent (see Definition 2.4):*

- $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$,
- $\mathrm{LexRange}(P_1', P_2', T') \neq \emptyset$.

Moreover, if $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$*, then*

$$\min \mathrm{LexRange}(P_1, P_2, T) = \frac{\min \mathrm{LexRange}(P_1', P_2', T') - 1}{\delta} + 1.$$

Proof. The first claim (i.e., the equivalence) follows immediately by Lemma 7.2. To prove the second claim, note that, letting $\mathrm{LexRange}(P_1, P_2, T) = \{a_1, a_2, \ldots, a_k\}$ be such that $a_1 < a_2 < \cdots < a_k$, and $(b_i)_{i \in [1\,..\,k]}$ be a sequence defined by $b_i = (a_i - 1)\delta + 1$, by Lemma 7.2 it holds $\mathrm{LexRange}(P'_1, P'_2, T') = \{b_1, b_2, \ldots, b_k\}$. Clearly, we have $b_1 < b_2 < \cdots < b_k$. Thus, $\min \mathrm{LexRange}(P_1, P_2, T) = a_1 = (b_1 - 1)/\delta + 1 = (\min \mathrm{LexRange}(P'_1, P'_2, T') - 1)/\delta + 1$. □

Proposition 9.7. *Let $T \in [0\,..\,\sigma)^n$ be a nonempty string, where $2 \le \sigma < n^{\mathcal{O}(1)}$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct the packed representation of a text $T_{\mathrm{bin}} \in \{\mathtt{0},\mathtt{1}\}^*$ satisfying $|T_{\mathrm{bin}}| = \Theta(n \log \sigma)$, integers α, β, and γ, and a data structure that, given the packed representation of any $P_1, P_2 \in [0\,..\,\sigma)^{\le m}$, in $\mathcal{O}(1 + m/\log_\sigma n)$ time returns the packed representation of $P'_1, P'_2 \in \{\mathtt{0},\mathtt{1}\}^*$ satisfying $|P'_i| = |P_i| \cdot (2\lceil \log \sigma \rceil + 3)$ (where $i \in \{1, 2\}$) and such that the following conditions are equivalent:*

- $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$,
- $\mathrm{LexRange}(P'_1, P'_2, T) \neq \emptyset$.

Moreover, if $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$, then it holds

$$\min \mathrm{LexRange}(P_1, P_2, T) = \frac{\min \mathrm{LexRange}(P'_1, P'_2, T_{\mathrm{bin}}) - \alpha}{\beta} + \gamma.$$

Proof. The proof proceeds the same as in Proposition 7.3, except the correctness of the formula follows by Lemma 9.6 (rather than Lemma 7.2). □

9.2.4 Summary

Theorem 9.8. *Consider a data structure answering lex-range minimum queries (see Section 9.1) that, for any text $T \in \{\mathtt{0},\mathtt{1}\}^n$, achieves the following complexities (where in the preprocessing we assume that we are given as input the packed representation of T, and at query time we are given a packed representation of $P_1, P_2 \in \{\mathtt{0},\mathtt{1}\}^{\le k}$):*

- *space usage $S(n)$,*
- *preprocessing time $P_t(n)$,*
- *preprocessing space $P_s(n)$,*
- *query time $Q(n, k)$.*

For every sequence $W[1\,..\,m]$ of $m \ge 1$ binary strings of length $\ell = 1 + \lfloor \log m \rfloor$ and any array $A[1\,..\,m]$ of integers (where $A[i] \in [1\,..\,m]$ for $i \in [1\,..\,m]$), there exists $n = \mathcal{O}(m \log m)$ and $k = \mathcal{O}(\log m)$ such that, given the sequences A and W, with all strings in W represented in the packed form, we can in $\mathcal{O}(m + P_t(n))$ time and using $\mathcal{O}(m + P_s(n))$ working space construct a data structure of size $\mathcal{O}(m + S(n))$ that, given the packed representation of any $X \in \{\mathtt{0},\mathtt{1}\}^{\le \ell}$ and any pair of integers $b, e \in [0\,..\,m]$, computes $\mathsf{prefix\text{-}rmq}_{A,W}(b, e, X)$ in $\mathcal{O}(Q(n, k))$ time.

Proof. We use the following definitions. Let $\pi[1\,..\,m]$ be a permutation of $[1\,..\,m]$ such that for every $i, j \in [1\,..\,m]$ satisfying $i < j$, it holds $A[\pi[i]] < A[\pi[j]]$, or $A[\pi[i]] = A[\pi[j]]$ and $\pi[i] < \pi[j]$. Note that, in particular, we then have $A[\pi[1]] \le A[\pi[2]] \le \cdots \le A[\pi[m]]$. Let $T_{\mathrm{aux}} = \mathrm{PermSeqToString}_\ell(\pi, W)$ (Definition 9.1). Denote $n_{\mathrm{aux}} = |T_{\mathrm{aux}}| = m \cdot (2\ell + 1) = \Theta(m \log m)$. Let T_{bin} and α_{bin}, β_{bin}, and γ_{bin} denote the text and the three integers obtained by applying Proposition 9.7 to text T_{aux}. Denote $n = |T_{\mathrm{bin}}|$. Since T_{aux} is over alphabet $\{\mathtt{0}, \ldots, \mathtt{4}\}$, by Proposition 9.7 we have $n = \Theta(|T_{\mathrm{aux}}|) = \Theta(m \log m)$. Denote $k = 18\lfloor \log m \rfloor + 27 = \mathcal{O}(\log m)$.

Components The data structure consists of the following components:

1. The structure from Proposition 9.7 applied to text T_{aux}. Since T_{aux} is over alphabet $\{\mathtt{0}, \ldots, \mathtt{4}\}$, the structure needs $\mathcal{O}(|T_{\mathrm{aux}}|/\log |T_{\mathrm{aux}}|) = \mathcal{O}(m)$ space.
2. The data structure from the claim (i.e., answering lex-range minimum queries) for the text T_{bin}. The structure needs $\mathcal{O}(S(n))$ space.
3. The array $\pi[1\,..\,m]$ storing the permutation π defined above. It needs $\mathcal{O}(m)$ space.
4. A lookup table L_{rev} defined as in Theorem 6.4. It needs $\mathcal{O}(\sqrt{n}) = \mathcal{O}(m)$ space.
5. A lookup table L_{map} defined as in Theorem 6.4. It also needs $\mathcal{O}(\sqrt{n}) = \mathcal{O}(m)$ space.
6. The integers α_{bin}, β_{bin}, and γ_{bin}.

In total, the structure needs $\mathcal{O}(m + S(n))$ space.

Implementation of queries The queries are answered as follows. Assume that we are given the packed representation of some $X \in \{0,1\}^{\leq \ell}$ and two integers $b, e \in [0\,..\,m]$. The algorithm to compute $\mathsf{prefix\text{-}rmq}_{A,W}(b, e, X)$ proceeds as follows:

1. Using L_{rev} and L_{map}, in $\mathcal{O}(1)$ time we compute the packed representation of strings $P_1 = \overline{X} \cdot 4 \cdot s(b)$ and $P_2 = \overline{X} \cdot 4 \cdot s(e)$ (over alphabet $\{0, \ldots, 4\}$). Note that $|P_1| = |P_2| = |X| + 1 + \ell \leq 2\ell + 1 \leq 2\lfloor \log m \rfloor + 3$.
2. Using Proposition 9.7, in $\mathcal{O}(1+|P_1|/\log n_{\mathrm{aux}}+|P_2|/\log n_{\mathrm{aux}}) = \mathcal{O}(1+\ell/\log m) = \mathcal{O}(1)$ time, we compute the packed representation of strings $P_1', P_2' \in \{0,1\}^*$ of length $|P_1'| = |P_2'| = 9 \cdot |P_1| \leq 18\lfloor \log m \rfloor + 27 \leq k$ such that $\mathrm{LexRange}(P_1, P_2, T_{\mathrm{aux}}) \neq \emptyset$ holds if and only if $\mathrm{LexRange}(P_1', P_2', T_{\mathrm{bin}}) \neq \emptyset$, and moreover, if $\mathrm{LexRange}(P_1, P_2, T_{\mathrm{aux}}) \neq \emptyset$, then it holds
$$\min \mathrm{LexRange}(P_1, P_2, T_{\mathrm{aux}}) = \frac{\min \mathrm{LexRange}(P_1', P_2', T_{\mathrm{bin}}) - \alpha_{\mathrm{bin}}}{\beta_{\mathrm{bin}}} + \gamma_{\mathrm{bin}}.$$
3. Using the structure from the claim, we compute $p'_{\min} = \min \mathrm{LexRange}(P_1', P_2', T_{\mathrm{bin}}) \cup \{\infty\}$. By $|P_1'|, |P_2'| \leq k$, this takes $\mathcal{O}(Q(n,k))$ time.
4. If $p'_{\min} = \infty$ (which is equivalent to $\mathrm{LexRange}(P_1', P_2', T_{\mathrm{bin}}) = \emptyset$), then by the above, it holds $\mathrm{LexRange}(P_1, P_2, T_{\mathrm{aux}}) = \emptyset$, or equivalently, $\min \mathrm{LexRange}(P_1, P_2, T_{\mathrm{aux}}) \cup \{\infty\} = \infty$. By Lemma 9.5, this is equivalent to $\mathsf{prefix\text{-}rmq}_{A,W}(b, e, X) = \infty$. We thus return ∞, and conclude the query algorithm. Let us now assume that $p'_{\min} \neq \infty$. We then have $\mathrm{LexRange}(P_1', P_2', T_{\mathrm{bin}}) \neq \emptyset$, $\mathrm{LexRange}(P_1, P_2, T_{\mathrm{aux}}) \neq \emptyset$, and $\min \mathrm{LexRange}(P_1, P_2, T_{\mathrm{aux}}) \cup \{\infty\} \neq \infty$.
5. In $\mathcal{O}(1)$ time, we compute the value $p_{\min} = (p'_{\min} - \alpha_{\mathrm{bin}})/\beta_{\mathrm{bin}} + \gamma_{\mathrm{bin}}$. By the above, it holds $p_{\min} = \min \mathrm{LexRange}(P_1, P_2, T_{\mathrm{aux}})$.
6. In $\mathcal{O}(1)$ time compute $i = \lceil p_{\min}/\beta \rceil$, where $\beta = 2\ell + 1$. Then, compute and return the value $r = \pi[i]$. By Lemma 9.5, it holds $\mathsf{prefix\text{-}rmq}_{A,W}(b, e, X) = r$.

In total, the query takes $\mathcal{O}(Q(n,k))$ time.

Construction algorithm The components of the data structure are constructed as follows:

1. The first component of the structure is computed as follows:
   (a) First, we compute the array containing the permutation π. To this end, in $\mathcal{O}(m)$ time we initialize an array $A_{\mathrm{perm}}[1\,..\,m]$ defined by $A_{\mathrm{perm}}[i] = (A[i], i)$. We then sort this array lexicographically. Utilizing that all numbers are in the range $[1\,..\,m]$, we use a 2-round radix sort and achieve $\mathcal{O}(m)$ time. The resulting array contains the permutation $\pi[1\,..\,m]$ on the second coordinate.
   (b) In $\mathcal{O}(m)$ time we apply Proposition 9.4 to the permutation π and the sequence W to compute the packed representation of the string $T_{\mathrm{aux}} = \mathrm{PermSeqToString}_{\ell}(\pi, W)$ (Definition 9.1).
   (c) Lastly, we apply Proposition 9.7 to T_{aux}. Since T_{aux} is over alphabet $\{0, \ldots, 4\}$, this takes $\mathcal{O}(|T_{\mathrm{aux}}|/\log |T_{\mathrm{aux}}|) = \mathcal{O}(m)$ time. Note that Proposition 9.7, in addition to the data structure, also returns integers α_{bin}, β_{bin}, and γ_{bin}, and the packed representation of the string T_{bin} (defined above).
2. We apply the preprocessing from the claim to the string T_{bin}. This takes $\mathcal{O}(P_t(|T_{\mathrm{bin}}|)) = \mathcal{O}(P_t(n))$ time and uses $\mathcal{O}(P_s(|T_{\mathrm{bin}}|)) = \mathcal{O}(P_s(n))$ working space.
3. We store the permutation π (computed above) in $\mathcal{O}(m)$ time.
4. The lookup table L_{rev} is computed in $\mathcal{O}(m)$ time similarly as in the proof of Proposition 4.12.
5. Next, we compute the lookup table L_{map}. Similarly as above, we proceed as in Proposition 4.12, and spend $\mathcal{O}(m)$ time.
6. Lastly, we store integers α_{bin}, β_{bin}, and γ_{bin}, which were already computed above.

In total, the construction takes $\mathcal{O}(m + P_t(n))$ time and uses $\mathcal{O}(m + P_s(n))$ working space. □

9.3 Reducing Lex-Range Minimum to Prefix RMQ

9.3.1 Problem Reduction

9.3.1.1 The Short Patterns

Proposition 9.9. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that given the packed representation of any $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$ satisfying $\mathrm{Occ}(X,T) \neq \emptyset$, returns $\min \mathrm{Occ}(X,T)$ in $\mathcal{O}(1)$ time.*

Proof. The data structure consists of a single component: the lookup table L_{minocc}. Its definition, space requirement, usage, and construction is as described in Section 5.2 of [KK24].[5] □

Proposition 9.10. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that answers the following queries:*

1. *Given the packed representation of any $X_1, X_2 \in [0\,..\,\sigma)^{\leq 3\tau-1}$, return $\min \mathrm{LexRange}(X_1, X_2, T) \cup \{\infty\}$ (see Definition 2.4) in $\mathcal{O}(1)$ time.*
2. *Given the packed representation of any $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, return $\min \mathrm{LexRange}(X, Xc^\infty, T) \cup \{\infty\}$ (where $c = \sigma - 1$) in $\mathcal{O}(1)$ time.*
3. *Given the packed representation of any $X_1, X_2 \in [0\,..\,\sigma)^{\leq 3\tau-1}$, return $\min \mathrm{LexRange}(X_1 c^\infty, X_2, T) \cup \{\infty\}$ (where $c = \sigma - 1$) in $\mathcal{O}(1)$ time.*

Proof. We use the following definitions. Let $B_{3\tau-1}[1\,..\,n]$ be a bitvector defined so that for every $i \in [1\,..\,n]$, $B_{3\tau-1}[i] = 1$ holds if and only if $i = n$, or $i < n$ and $\mathrm{LCE}_T(\mathrm{SA}_T[i], \mathrm{SA}_T[i{+}1]) < 3\tau{-}1$. Let $k = \mathsf{rank}_{B_{3\tau-1}}(n, 1)$. Let $A_{\min}[1\,..\,k]$ be an array of positive integers such that for every $i \in [1\,..\,k]$, $A_{\min}[i] = \min \mathrm{Occ}(X,T)$, where $i' = \mathsf{select}_{B_{3\tau-1}}(i, 1)$ and $X = T[\mathrm{SA}_T[i']\,..\,\min(n{+}1, \mathrm{SA}_T[i']{+}3\tau{-}1))$. Observe that $k = \mathcal{O}(\sigma^{3\tau-1}) = \mathcal{O}(\sqrt{n})$.

Components The data structure consists of the following components:

1. The data structure from Proposition 7.10. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
2. The bitvector $B_{3\tau-1}[1\,..\,n]$ augmented with the support for $\mathcal{O}(1)$-time rank and select queries using Theorem 2.7. The bitvector together with the augmentation of Theorem 2.7 needs $\mathcal{O}(n/\log n) = \mathcal{O}(n/\log_\sigma n)$ space.
3. The array $A_{\min}[1\,..\,k]$ augmented with support for $\mathcal{O}(1)$-time RMQ queries using Theorem 2.13. The array together with the augmentation of Theorem 2.13 needs $\mathcal{O}(k) = \mathcal{O}(\sqrt{n}) = \mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n)$ space.

Implementation of queries The queries are answered as follows:

1. Let $X_1, X_2 \in [0\,..\,\sigma)^{\leq 3\tau-1}$. Given the packed representation of strings X_1 and X_2, we compute $\min \mathrm{LexRange}(X_1, X_2, T) \cup \{\infty\}$ as follows:
   (a) Using Proposition 7.10, in $\mathcal{O}(1)$ time compute $b_k = \mathrm{RangeBeg}(X_k, T)$ for $k \in \{1, 2\}$. Observe that then
   $$\mathrm{LexRange}(X_1, X_2, T) = \{\mathrm{SA}_T[i]\}_{i \in (b_1\,..\,b_2]}$$
   (see Remark 2.5). If $b_1 \geq b_2$, then it holds $\mathrm{LexRange}(X_1, X_2, T) = \emptyset$, and hence we return $\min \mathrm{LexRange}(X_1, X_2, T) \cup \{\infty\} = \infty$. Let us thus assume that $b_1 < b_2$. Note that, by definition of $B_{3\tau-1}$, if $b_k > 0$, then $B_{3\tau-1}[b_k] = 1$ holds for $k \in \{1, 2\}$.
   (b) If $b_1 = 0$, then we set $i_1 = 0$. Otherwise, using Theorem 2.7, in $\mathcal{O}(1)$ time we compute $i_1 = \mathsf{rank}_{B_{3\tau-1}}(b_1, 1)$. Analogously we compute i_2.
   (c) Using Theorem 2.13, in $\mathcal{O}(1)$ time we compute $i_{\min} = \mathsf{rmq}_{A_{\min}}(i_1, i_2)$. Observe that by definition of $B_{3\tau-1}$ and $A_{\min}$, we then have $\min\{\mathrm{SA}_T[i]\}_{i \in (b_1\,..\,b_2]} = A_{\min}[i_{\min}]$. Thus, we return $A_{\min}[i_{\min}]$ as the answer.

   In total, we spend $\mathcal{O}(1)$ time.
2. Let us now consider $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, where $c = \sigma - 1$. Given the packed representation of X, we compute $\min \mathrm{LexRange}(X, Xc^\infty, T) \cup \{\infty\}$ by setting $X_1 = X$ and $X_2 = Xc^\infty$, and using the same

[5] The construction is originally described only for $X \in [0\,..\,\sigma)^{<3\tau-1}$, but it is easy to check that without modifications it works for $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$.

algorithm as above, except to compute $b_2 = \mathrm{RangeBeg}(Xc^\infty, T)$, we observe that $\mathrm{RangeBeg}(Xc^\infty, T) = \mathrm{RangeEnd}(X, T)$. Thus, we can determine b_2 using Proposition 7.10 in $\mathcal{O}(1)$ time.

3. Let $X_1, X_2 \in [0\,..\,\sigma)^{\leq 3\tau-1}$. Given the packed representation of strings X_1 and X_2, we compute $\min \mathrm{LexRange}(X_1c^\infty, X_2, T) \cup \{\infty\}$ (where $c = \sigma - 1$) similarly as above, except to compute $b_1 = \mathrm{RangeBeg}(X_1c^\infty, T)$, we use the fact that $\mathrm{RangeBeg}(X_1c^\infty, T) = \mathrm{RangeEnd}(X_1, T)$. Thus, we can compute b_1 using Proposition 7.10 in $\mathcal{O}(1)$ time.

Construction algorithm The components of the data structure are constructed as follows:

1. The first component is constructed using Proposition 7.10 in $\mathcal{O}(n/\log_\sigma n)$ time.
2. To construct the second component, we first compute the bitvector $B_{3\tau-1}[1\,..\,n]$ as described in [KK23a, Proposition 5.3] in $\mathcal{O}(n/\log_\sigma n)$ time. We then augment $B_{3\tau-1}$ with support for $\mathcal{O}(1)$-time rank/select queries using Theorem 2.7 in $\mathcal{O}(n/\log n) = \mathcal{O}(n/\log_\sigma n)$ time.
3. To describe the construction of $A_{\min}[1\,..\,k]$, we first introduce as auxiliary array $A_{\mathrm{str}}[1\,..\,k]$ defined such that, for every $i \in [1\,..\,k]$, it holds $A_{\mathrm{str}}[i] = T[\mathrm{SA}_T[i']\,..\,\min(n+1, \mathrm{SA}_T[i'] + 3\tau - 1))$, where $i' = \mathsf{select}_{B_{3\tau-1}}(i, 1)$. Observe that for every $i \in [1\,..\,k]$, we then have $A_{\min}[i] = \min \mathrm{Occ}(A_{\mathrm{str}}[i], T)$. The construction of $A_{\min}[1\,..\,k]$ proceeds as follows:
   (a) In $\mathcal{O}(n/\log_\sigma n)$ time construct the array $A_{\mathrm{str}}[1\,..\,k]$ (with all strings stored in the packed representation) as described in [KK23a, Proposition 5.3].
   (b) In $\mathcal{O}(n/\log_\sigma n)$ time construct the data structure from Proposition 9.9.
   (c) For every $i \in [1\,..\,k]$, compute $A_{\min}[i] = \min \mathrm{Occ}(A_{\mathrm{str}}[i], T)$ in $\mathcal{O}(1)$ time using Proposition 9.9. In total, we spend $\mathcal{O}(k)$ time.

   In total, the construction of $A_{\min}[1\,..\,k]$ takes $\mathcal{O}(n/\log_\sigma n + k) = \mathcal{O}(n/\log_\sigma n)$ time.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n)$ time. □

9.3.1.2 The Nonperiodic Patterns

Combinatorial Properties

Lemma 9.11. *Let $T \in \Sigma^n$, $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$, and assume that $T[n]$ does not occur in $T[1\,..\,n)$. Let S be a τ-synchronizing set of T. Denote $(s_i)_{i\in[1\,..\,n']} = \mathrm{LexSorted}(\mathsf{S}, T)$ (Definition 7.13). Let $A_{\mathsf{S}}[1\,..\,n']$ and $A_{\mathrm{str}}[1\,..\,n']$ be defined by*

- $A_{\mathsf{S}}[i] = s_i$,
- $A_{\mathrm{str}}[i] = \overline{D_i}$, *where* $D_i = T^\infty[s_i - \tau\,..\,s_i + 2\tau)$.

Let $D \in \mathcal{D}(\tau, T, \mathsf{S})$ (Definition 7.16) and let $P_1, P_2 \in \Sigma^+$ be τ-nonperiodic patterns (Definition 7.5) both having D as a prefix. Denote $\delta_{\mathrm{text}} = |D| - 2\tau$. For $k \in \{1, 2\}$, let $P'_k = P_k(\delta_{\mathrm{text}}\,..\,|P_k|]$, and $b_k = |\{i \in [1\,..\,n'] : T[s_i\,..\,n] \prec P'_k\}|$. Then, the following conditions are equivalent:

1. $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$ *(see Definition 2.4),*
2. $\mathsf{prefix\text{-}rmq}_{A_{\mathsf{S}}, A_{\mathrm{str}}}(b_1, b_2, \overline{D}) \neq \infty$ *(see Definition 2.9).*

Moreover, if $\mathrm{LexRange}(P_1, P_2, T) \neq \emptyset$, *then*

$$\min \mathrm{LexRange}(P_1, P_2, T) = A_{\mathsf{S}}[\mathsf{prefix\text{-}rmq}_{A_{\mathsf{S}}, A_{\mathrm{str}}}(b_1, b_2, \overline{D})] - \delta_{\mathrm{text}}.$$

Proof. The first claim (i.e., the equivalence) follows by combining Definition 2.9 and Lemma 8.6.

To prove the second claim, we combine Lemma 7.24 and Definition 2.9 to obtain:

$$\begin{aligned}
\min \mathrm{LexRange}(P_1, P_2, T) &= \min\{A_{\mathsf{S}}[i] - \delta_{\mathrm{text}} : i \in (b_1\,..\,b_2] \text{ and } \overline{D} \text{ is a prefix of } A_{\mathrm{str}}[i]\} \\
&= \min\{A_{\mathsf{S}}[i] : i \in (b_1\,..\,b_2] \text{ and } \overline{D} \text{ is a prefix of } A_{\mathrm{str}}[i]\} - \delta_{\mathrm{text}} \\
&= A_{\mathsf{S}}[\mathsf{prefix\text{-}rmq}_{A_{\mathsf{S}}, A_{\mathrm{str}}}(b_1, b_2, \overline{D})] - \delta_{\mathrm{text}}.
\end{aligned}$$

□

Lemma 9.12. *Let $T \in \Sigma^n$, $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$, and assume that $T[n]$ does not occur in $T[1\,..\,n)$. Let S be a τ-synchronizing set of T. Denote $(s_i^{\mathrm{lex}})_{i\in[1\,..\,n']} = \mathrm{LexSorted}(\mathsf{S}, T)$ (Definition 7.13), $(s_i^{\mathrm{text}})_{i\in[1\,..\,n']} = \mathrm{Sort}(\mathsf{S})$*

(Definition 7.14), and let $\pi[1\,..\,n']$ be a permutation of $[1\,..\,n']$ such that for every $i \in [1\,..\,n']$, it holds $s_i^{\text{lex}} = s_{\pi[i]}^{\text{text}}$. Let $A_\pi[1\,..\,n']$, $A_{\mathsf{S}}[1\,..\,n']$, and $A_{\text{str}}[1\,..\,n']$ be defined by

- $A_\pi[i] = \pi[i]$,
- $A_{\mathsf{S}}[i] = s_i^{\text{text}}$,
- $A_{\text{str}}[i] = \overline{D_i}$, *where* $D_i = T^\infty[s_i - \tau\,..\,s_i + 2\tau)$.

Let $D \in \mathcal{D}(\tau, T, \mathsf{S})$ (Definition 7.16) and let $P_1, P_2 \in \Sigma^+$ be τ-nonperiodic patterns (Definition 7.5) both having D as a prefix. Denote $\delta_{\text{text}} = |D| - 2\tau$. For $k \in \{1, 2\}$, let $P'_k = P_k(\delta_{\text{text}}\,..\,|P_k|]$, and $b_k = |\{i \in [1\,..\,n'] : T[s_i^{\text{lex}}\,..\,n] \prec P'_k\}|$. Then, the following conditions are equivalent:

1. $\text{LexRange}(P_1, P_2, T) \neq \emptyset$ *(see Definition 2.4),*
2. $\mathsf{prefix\text{-}rmq}_{A_\pi, A_{\text{str}}}(b_1, b_2, \overline{D}) \neq \infty$ *(see Definition 2.9).*

Moreover, if $\text{LexRange}(P_1, P_2, T) \neq \emptyset$, *then*

$$\min \text{LexRange}(P_1, P_2, T) = A_{\mathsf{S}}[A_\pi[\mathsf{prefix\text{-}rmq}_{A_\pi, A_{\text{str}}}(b_1, b_2, \overline{D})]] - \delta_{\text{text}}.$$

Proof. To show the first claim, note that whether $\mathsf{prefix\text{-}rmq}_{A,S}(b, e, X) \neq \infty$ holds in Definition 2.9 does not depend on the array A. Thus, the first claim follows by Lemma 9.11.

To prove the second claim, observe that by definition of the permutation π, for every $i, j \in [1\,..\,n']$ such that $i \neq j$, $s_i^{\text{lex}} < s_j^{\text{lex}}$ holds if and only if $s_{\pi[i]}^{\text{text}} < s_{\pi[j]}^{\text{text}}$. On the other hand, by Definition 7.14, $s_{\pi[i]}^{\text{text}} < s_{\pi[j]}^{\text{text}}$ holds if and only if $\pi[i] < \pi[j]$. Thus, $s_i^{\text{lex}} < s_j^{\text{lex}}$ holds if and only if $\pi[i] < \pi[j]$. This implies that for every $Q \subseteq [1\,..\,n']$, $\arg\min_{j \in Q} s_j^{\text{lex}} = \arg\min_{j \in Q} \pi[j]$. In particular, letting $i = \arg\min_{j \in Q} \pi[j]$, it holds $\min\{s_j^{\text{lex}} : j \in Q\} = s_i = s_{\pi[i]}^{\text{text}}$. Consequently, combining Lemma 7.23 and Definition 2.9, we obtain:

$$\begin{aligned}
\min \text{LexRange}(P_1, P_2, T) &= \min\{s_i^{\text{lex}} - \delta_{\text{text}} : i \in (b_1\,..\,b_2] \text{ and } \overline{D} \text{ is a prefix of } A_{\text{str}}[i]\} \\
&= \min\{s_i^{\text{lex}} : i \in (b_1\,..\,b_2] \text{ and } \overline{D} \text{ is a prefix of } A_{\text{str}}[i]\} - \delta_{\text{text}} \\
&= A_{\mathsf{S}}[A_\pi[\mathsf{prefix\text{-}rmq}_{A_\pi, A_{\text{str}}}(b_1, b_2, \overline{D})]] - \delta_{\text{text}}.
\end{aligned}$$

□

Remark 9.13. The reason for introducing the array A_π in Lemma 9.12 is so that the first array used in prefix RMQ queries has values between 1 and the size of the array. This is used in Proposition 9.14.

Algorithms

Proposition 9.14. *Consider a data structure that, given any array of integers $A[1\,..\,k]$ (where $A[i] \in [1\,..\,k]$ for $i \in [1\,..\,k]$) and a sequence $W[1\,..\,k]$ strings of length ℓ over alphabet $[0\,..\,\sigma)$ achieves the following complexities (where the input strings during construction are given in the packed representation, and at query time we are given any $b, e \in [0\,..\,k]$ and the packed representation of any $X \in [0\,..\,\sigma)^{\leq \ell}$, and we return the position $\mathsf{prefix\text{-}rmq}_{A,W}(b, e, X)$; see Definition 2.9):*

- *space usage $S(k, \ell, \sigma)$,*
- *preprocessing time $P_t(k, \ell, \sigma)$,*
- *preprocessing space $P_s(k, \ell, \sigma)$,*
- *query time $Q(k, \ell, \sigma)$.*

Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. There exist positive integers $k = \Theta(n/\log_\sigma n)$ and $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n + P_t(k, \ell, \sigma))$ time and using $\mathcal{O}(n/\log_\sigma n + P_s(k, \ell, \sigma))$ working space construct a data structure of size $\mathcal{O}(n/\log_\sigma n + S(k, \ell, \sigma))$ that answers the following queries:

1. *Given the packed representation of a τ-nonperiodic (Definition 7.5) patterns $P_1, P_2 \in [0\,..\,\sigma)^+$ such that $\text{lcp}(P_1, P_2) \geq 3\tau - 1$, computes $\min \text{LexRange}(P_1, P_2, T) \cup \{\infty\}$ (see Definition 2.4) in $\mathcal{O}(\log\log n + Q(k, \ell, \sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.*
2. *Given the packed representation of a τ-nonperiodic pattern $P \in [0\,..\,\sigma)^+$ satisfying $|P| \geq 3\tau - 1$, computes $\min \text{LexRange}(P, P'c^\infty, T) \cup \{\infty\}$ (where $P' = P[1\,..\,3\tau - 1]$ and $c = \sigma - 1$) in $\mathcal{O}(\log\log n + Q(k, \ell, \sigma) + |P|/\log_\sigma n)$ time.*

Proof. We use the following definitions. Let S be a τ-synchronizing set of T of size $|\mathsf{S}| = \mathcal{O}(\frac{n}{\tau})$ constructed using Theorem 2.20. Denote $n' = |\mathsf{S}|$. Denote $(s_i^{\text{lex}})_{i\in[1..n']} = \text{LexSorted}(\mathsf{S}, T)$ (Definition 7.13), $(s_i^{\text{text}})_{i\in[1..n']} = \text{Sort}(\mathsf{S})$ (Definition 7.14), and let $\pi[1..n']$ be a permutation of $[1..n']$ such that for every $i \in [1..n']$ it holds $s_i^{\text{lex}} = s_{\pi[i]}^{\text{text}}$. Let $k = \max(n', k') = \Theta(n/\log_\sigma n)$, where $k' = \lceil n/\log_\sigma n\rceil$. Let $A_\pi[1..n']$ be defined so that for every $i \in [1..n']$, $A_\pi[i] = \pi[i]$. Let $A_{\mathsf{S}}[1..k]$ and $A_{\text{str}}[1..k]$ be arrays defined so that for every $i \in [1..n']$, $A_{\mathsf{S}}[i] = s_i^{\text{text}}$ and $A_{\text{str}}[i] = \overline{D_i}$, where $D_i = T^\infty[s_i - \tau..s_i + 2\tau)$. The remaining elements of the arrays are initialized arbitrarily (they are not used). Denote $\ell = 3\tau$, and note that by the same analysis as in Theorem 4.18, it holds $\ell \le (1 + \lfloor\log k\rfloor)/\lceil\log\sigma\rceil$.

Components The data structure consists of the following components:

1. The structure from Proposition 7.10 for text T. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
2. The structure NavNonperiodic(S, τ, T) from Proposition 7.21. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
3. The structure from the claim (answering prefix range minimum queries) for sequences $A_\pi[1..k]$ and $A_{\text{str}}[1..k]$. It needs $\mathcal{O}(S(k, \ell, \sigma))$ space.
4. The plain representation of the array $A_{\mathsf{S}}[1..k]$. It needs $\mathcal{O}(k) = \mathcal{O}(n/\log_\sigma n)$ space.
5. The plain representation of the array $A_\pi[1..n']$ using $\mathcal{O}(n') = \mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n + S(k, \ell, \sigma))$ space.

Implementation of queries The queries are answered as follows.

1. Let $P_1, P_2 \in [0..\sigma)^+$ be τ-nonperiodic patterns satisfying $\text{lcp}(P_1, P_2) \ge 3\tau - 1$. Denote $P' = P_1[1..3\tau - 1] = P_2[1..3\tau-1]$. Given the packed representation of P_1 and P_2, we compute $\min \text{LexRange}(P_1, P_2, T) \cup \{\infty\}$ as follows:
   (a) First, using Proposition 7.10, in $\mathcal{O}(1)$ time we compute $\text{RangeBeg}(P', T)$ and $\text{RangeEnd}(P', T)$. This lets us determine $|\text{Occ}(P', T)|$. If $\text{Occ}(P', T) = \emptyset$, then we have $\text{LexRange}(P_1, P_2, T) = \emptyset$ (since any element of this set would have P' as a prefix), and hence we return $\min \text{LexRange}(P_1, P_2, T) \cup \{\infty\} = \infty$, and finish the query algorithm. Let us now assume that $\text{Occ}(P', T) \ne \emptyset$.
   (b) Using Proposition 7.21(2a), in $\mathcal{O}(1)$ time compute the packed representation of the string $D = \text{DistPrefix}(P_1, \tau, T, \mathsf{S})$ (Definition 7.20). Note that since $\text{lcp}(P_1, P_2) \ge 3\tau - 1 \ge |D|$, D is also a prefix of P_2 (and by Lemma 7.19 no other string in $\mathcal{D}(\tau, T, \mathsf{S})$ is a prefix of P_2). In $\mathcal{O}(1)$ time we then calculate $\delta_{\text{text}} = |D| - 2\tau$.
   (c) Using Proposition 7.21(1), in $\mathcal{O}(1)$ time compute the packed representation of $\overline{D}$.
   (d) Denote $P'_k = P_k(\delta_{\text{text}}..|P_k|]$, where $k \in \{1, 2\}$. Using Proposition 7.21(2b), compute the value $b_k = |\{i \in [1..n'] : T[s_i..n] \prec P'_k\}|$ for $k \in \{1, 2\}$. This takes $\mathcal{O}(\log\log n + |P'_1|/\log_\sigma n + |P'_2|/\log_\sigma n) = \mathcal{O}(\log\log n + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.
   (e) Using the structure from the claim, we compute $j = \mathsf{prefix\text{-}rmq}_{A_\pi, A_{\text{str}}}(b_1, b_2, \overline{D})$ (Definition 2.9) in $\mathcal{O}(Q(k, \ell, \sigma))$ time. If $j = \infty$, then by Lemma 9.12, $\text{LexRange}(P_1, P_2, T) = \emptyset$, and hence we return $\min \text{LexRange}(P_1, P_2, T) \cup \{\infty\} = \infty$. Otherwise (i.e., if $j \ne \infty$), by Lemma 9.12 we have $A_{\mathsf{S}}[A_\pi[j]] = \min \text{LexRange}(P_1, P_2, T)$. We thus return $A_{\mathsf{S}}[A_\pi[j]]$ as the answer. Note that the array defined here is padded with extra strings compared to the array in Lemma 9.12, but this does not affect the result of the above query, since $b_1, b_2 \le n'$.

   In total, we spend $\mathcal{O}(\log\log n + Q(k, \ell, \sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.
2. Let us now consider a τ-nonperiodic pattern $P \in [0..\sigma)^+$ such that it holds $|P| \ge 3\tau - 1$. To compute $\min \text{LexRange}(P, P'c^\infty, T) \cup \{\infty\}$ (where $P' = P[1..3\tau - 1]$ and $c = \sigma - 1$), we set $P_1 = P$ and $P_2 = P'c^\infty$, and then proceed similarly as above, except we compute $b_2 = |\{i \in [1..n'] : T[s_i..n] \prec P'_2\}|$ using the second integer computed in Proposition 7.21(2b) (which requires only the packed representation of P'). The whole query takes $\mathcal{O}(\log\log n + Q(k, \ell, \sigma) + |P|/\log_\sigma n)$ time.

Construction algorithm The components of the structure are constructed as follows:

1. We apply Proposition 7.10. This uses $\mathcal{O}(n/\log_\sigma n)$ time and working space.
2. Using Theorem 2.20, we compute the τ-synchronizing set S satisfying $|\mathsf{S}| = \mathcal{O}(\frac{n}{\tau}) = \mathcal{O}(n/\log_\sigma n)$ in $\mathcal{O}(n/\log_\sigma n)$ time. Then, using S and the packed representation of T as input, we construct

NavNonperiodic(S, τ, T) in $\mathcal{O}(n/\log_\sigma n)$ time using Proposition 7.21.

3. To construct the next component of the structure, we proceed as follows:
   (a) Using Theorem 7.22, in $\mathcal{O}(\frac{n}{\tau}) = \mathcal{O}(n/\log_\sigma n)$ time we first compute the sequence $(s_i)_{i\in[1\,..\,n']} = \text{LexSorted}(\mathsf{S}, T)$ (Definition 7.13).
   (b) Using Proposition 7.21(1), in $\mathcal{O}(n') = \mathcal{O}(n/\log_\sigma n)$ time we compute the elements of the array A_{str} at indexes $i \in [1\,..\,n']$, i.e., $A_{\text{str}}[i] = \overline{D_i}$, where $D_i = T^\infty[s_i - \tau\,..\,s_i + 2\tau)$.
   (c) Finally, we apply the preprocessing from the claim to the array A_{str}. This takes $\mathcal{O}(P_t(k, \ell, \sigma))$ time and uses $\mathcal{O}(P_s(k, \ell, \sigma))$ working space.

   In total, the construction of this component of the structure takes $\mathcal{O}(n/\log_\sigma n + P_t(k, \ell, \sigma))$ time and uses $\mathcal{O}(n/\log_\sigma n + P_s(k, \ell, \sigma))$ working space.
4. The sequence Sort(S) (and hence the array A_{S}) is easily obtained from S (computed above) using a 2-round radix sort in $\mathcal{O}(|\mathsf{S}| + \sqrt{n}) = \mathcal{O}(n/\log_\sigma n)$ time.
5. To compute the last component, first recall that the sequences $(s_i^{\text{lex}})_{i\in[1\,..\,n']}$ and $(s_i^{\text{text}})_{i\in[1\,..\,n']}$ have already been constructed above. To compute the permutation $\pi[1\,..\,n']$, we create an array $P[1\,..\,n']$ of pairs defined by $P[i] = (s_i^{\text{lex}}, i)$. We then sort the array P lexicographically. Using a 4-round radix sort, we achieve $\mathcal{O}(n' + \sqrt{n}) = \mathcal{O}(n/\log_\sigma n)$ time. Let $(b_i)_{i\in[1\,..\,n']}$ be the sequence containing the second coordinates in the resulting sequence of pairs. For every $i \in [1\,..\,n']$, we set $A_\pi[b_i] = i$. In total, construction of $A_\pi[1\,..\,n']$ takes $\mathcal{O}(n/\log_\sigma n)$ time.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n + P_t(k, \ell, \sigma))$ time and uses $\mathcal{O}(n/\log_\sigma n + P_s(k, \ell, \sigma))$ working space. □

9.3.1.3 The Periodic Patterns

Combinatorial Properties

Lemma 9.15. *Let $T \in \Sigma^n$ and $\tau \in [1\,..\,\lfloor\frac{n}{2}\rfloor]$. Let $(a_j)_{j\in[1\,..\,q]} = \text{RunsLexSorted}^-(\tau, T)$ (Definition 7.44), and let $A_{\text{pos}}[1\,..\,q]$ and $A_{\text{len}}[1\,..\,q]$ be arrays defined by*

- $A_{\text{pos}}[i] = e^{\text{full}}(a_i, \tau, T)$,
- $A_{\text{len}}[i] = e^{\text{full}}(a_i, \tau, T) - a_i$.

Let $i_1, i_2 \in [1\,..\,n]$ be such that $i_1 \le i_2$ and $\{\text{SA}_T[i]\}_{i\in[i_1\,..\,i_2]} \subseteq \mathsf{R}^-_{s,k,H}(\tau, T)$ for some $s \in \mathbb{Z}_{\ge 0}$, $k \in \mathbb{Z}_{>0}$, and $H \in \Sigma^+$. Denote $p = |H|$ and $\delta_{\text{text}} = s + kp$. Let $j_1, j_2 \in [1\,..\,q]$ be such that for $k \in \{1, 2\}$, it holds $e^{\text{full}}(\text{SA}_T[i_k], \tau, T) = e^{\text{full}}(a_{j_k}, \tau, T)$. Then, there exists $j \in (j_1 - 1\,..\,j_2]$ such that $A_{\text{len}}[j] \ge \delta_{\text{text}}$. Moreover, it holds (see Definition 2.12):

$$\min\{\text{SA}_T[i]\}_{i\in[i_1\,..\,i_2]} = A_{\text{pos}}[\textsf{three-sided-rmq}_{A_{\text{pos}},A_{\text{len}}}(j_1 - 1, j_2, \delta_{\text{text}})] - \delta_{\text{text}}.$$

Proof. The first claim follows by $i_1 \le i_2$ and Lemma 7.54. To obtain the second claim, we apply Definition 2.12 and Lemma 7.54, resulting in:

$$\begin{aligned}\min\{\text{SA}_T[i]\}_{i\in[i_1\,..\,i_2]} &= \min\{A_{\text{pos}}[j] - \delta_{\text{text}} : j \in [j_1\,..\,j_2] \text{ and } A_{\text{len}}[j] \ge \delta_{\text{text}}\}\\ &= \min\{A_{\text{pos}}[j] : j \in [j_1\,..\,j_2] \text{ and } A_{\text{len}}[j] \ge \delta_{\text{text}}\} - \delta_{\text{text}}\\ &= A_{\text{pos}}[\textsf{three-sided-rmq}_{A_{\text{pos}},A_{\text{len}}}(j_1 - 1, j_2, \delta_{\text{text}})] - \delta_{\text{text}}.\end{aligned}$$

□

Lemma 9.16. *Let $T \in \Sigma^n$ and $\tau \in [1\,..\,\lfloor\frac{n}{2}\rfloor]$. Let $s \in \mathbb{Z}_{\ge 0}$, $k \in \mathbb{Z}_{>0}$, and $H \in \Sigma^+$ be such that $\mathsf{R}^-_{s,k,H}(\tau, T) \neq \emptyset$. Then, $\text{Occ}(1, \text{MinPosBitvector}^-_{s,k,H}(\tau, T)) = \emptyset$ (see Definition 7.49) implies that $\mathsf{R}^-_{s,k+1,H}(\tau, T) \neq \emptyset$ and*

$$\min \mathsf{R}^-_{s,k+1,H}(\tau, T) < \min \mathsf{R}^-_{s,k,H}(\tau, T).$$

Proof. Denote $p = |H|$. By Definition 7.49, the assumption $\text{Occ}(1, \text{MinPosBitvector}^-_{s,k,H}(\tau, T)) = \emptyset$ implies that $\mathsf{R}^-_{s,k,H}(\tau, T) \cap \mathsf{R}^-_{\min}(\tau, T) = \emptyset$. Denote $j = \min \mathsf{R}^-_{s,k,H}(\tau, T)$, and let $X = T[j\,..\,e(j, \tau, T))$. It holds $j \in \text{Occ}(X, T) \cap \mathsf{R}^-(\tau, T)$, but since $j \notin \mathsf{R}^-_{\min}(\tau, T)$, the position $j' = \min \text{Occ}(X, T) \cap \mathsf{R}^-(\tau, T)$ must satisfy $j' < j$; see Definition 7.48. Position j' also satisfies the following properties:

- Note that $|X| \geq 3\tau - 1$, and hence by Lemma 7.37(2), $j' \in \mathsf{R}^{-}_{s,H}(\tau, T)$. Moreover, $e(j', \tau, T) - j' \geq |X|$, and hence $\exp(j', \tau, T) = \lfloor ((e(j', \tau, T) - j') - s)/p \rfloor \geq \lfloor (|X| - s)/p \rfloor = k$.
- Next, we prove $j' \in \mathsf{R}^{-}_{\min}(\tau, T)$. Suppose this is not the case. Denote $X' = T[j' \,..\, e(j', \tau, T))$. Since $j' \in \mathrm{Occ}(X', T)$ and $j' \in \mathsf{R}^{-}(\tau, T)$, the assumption $j' \notin \mathsf{R}^{-}_{\min}(\tau, T)$ implies that there exists $j'' < j'$ such that $j'' \in \mathrm{Occ}(X', T)$ and $j'' \in \mathsf{R}^{-}(\tau, T)$. Since X is a prefix of X', we thus have $j'' \in \mathrm{Occ}(X, T) \cap \mathsf{R}^{-}(\tau, T)$. We thus obtain a contradiction with the assumption $j' = \min \mathrm{Occ}(X, T) \cap \mathsf{R}^{-}(\tau, T)$.

Denote $k' = \exp(j', \tau, T)$. Above we established that $k' \geq k$. However, since we also proved that $j' \in \mathsf{R}^{-}_{\min}(\tau, T)$ and $j' \in \mathsf{R}^{-}_{s,H}(\tau, T)$, and earlier we assumed that $\mathsf{R}^{-}_{s,k,H}(\tau, T) \cap \mathsf{R}^{-}_{\min}(\tau, T) = \emptyset$, we must therefore have $k' > k$. Denote

$$x = e^{\mathrm{full}}(j', \tau, T) - (s + (k+1)p).$$

Observe that x satisfies the following properties:

- First, we show that $j' \leq x$. To see this, recall that $j' \in \mathsf{R}^{-}_{s,k',H}(\tau, T)$, where $k' \geq (k+1)$. Thus, $e^{\mathrm{full}}(j', \tau, T) - j' = s + k' \cdot p \geq s + (k+1) \cdot p = e^{\mathrm{full}}(j', \tau, T) - x$. Equivalently, $j' \leq x$.
- Next, we prove that $j' \leq e(j', \tau, T) - (3\tau - 1)$. First, note that by $j \in \mathsf{R}^{-}_{s,k,H}(\tau, T)$, it holds

$$s + (k+1)p > s + kp + \mathrm{tail}(j, \tau, T) = e(j, \tau, T) - j \geq 3\tau - 1.$$

  Thus, $x = e^{\mathrm{full}}(j', \tau, T) - (s + (k+1)p) \leq e(j', \tau, T) - (s + (k+1)p) \leq e(j', \tau, T) - (3\tau - 1)$.
- In this step, we prove that $x \in \mathsf{R}^{-}_{s,k+1,H}(\tau, T)$. Recall that by Lemma 7.36, $[j' \,..\, e(j', \tau, T) - (3\tau - 1)]$ is a maximal block of positions in $\mathsf{R}(\tau, T)$ containing j'. Above we proved that $x \in [j' \,..\, e(j', \tau, T) - (3\tau - 1)]$. Thus, $x \in \mathsf{R}(\tau, T)$. Moreover, by Lemma 7.35, $\mathrm{root}(x, \tau, T) = \mathrm{root}(j', \tau, T) = H$, $\mathrm{type}(x, \tau, T) = \mathrm{type}(j', \tau, T) = -1$, and $e^{\mathrm{full}}(x, \tau, T) = e^{\mathrm{full}}(j', \tau, T)$. This, in turn, yields $\mathrm{head}(x, \tau, T) = (e^{\mathrm{full}}(x, \tau, T) - x) \bmod p = (e^{\mathrm{full}}(j', \tau, T) - x) \bmod p = (s + (k+1)p) \bmod p = s$ and $\exp(x, \tau, T) = \lfloor (e^{\mathrm{full}}(x, \tau, T) - x)/p \rfloor = \lfloor (e^{\mathrm{full}}(j', \tau, T) - x)/p \rfloor = \lfloor (s + (k+1)p)/p \rfloor = k + 1$. Thus, we obtain $x \in \mathsf{R}^{-}_{s,k+1,H}(\tau, T)$. In particular, this proves that $\mathsf{R}^{-}_{s,k+1,H}(\tau, T) \neq \emptyset$, i.e., the first part of the claim.
- Lastly, we prove that $x < j$. Recall that $j' < j$. We consider two cases:
  - First, assume that $[j' \,..\, j] \subseteq \mathsf{R}(\tau, T)$. By Lemma 7.35, we have $e^{\mathrm{full}}(j', \tau, T) = e^{\mathrm{full}}(j, \tau, T)$. On the other hand, $j \in \mathsf{R}^{-}_{s,k,H}(\tau, T)$ implies that $e^{\mathrm{full}}(j, \tau, T) - j = s + kp$. Equivalently, $j = e^{\mathrm{full}}(j, \tau, T) - (s + kp) = e^{\mathrm{full}}(j', \tau, T) - (s + kp) = x + p$. Thus, $x < j$.
  - Let us now assume that $[j' \,..\, j] \not\subseteq \mathsf{R}(\tau, T)$. Then, there exists $y \in [1 \,..\, n] \setminus \mathsf{R}(\tau, T)$ such that $j' < y < j$. By Lemma 7.39, it holds $e(j', \tau, T) - j < \tau$, or equivalently, $e(j', \tau, T) - \tau < j$. On the other hand, recall that above we proved that $s + (k+1)p \geq 3\tau - 1$. Thus, we obtain

$$\begin{aligned} x &= e^{\mathrm{full}}(j', \tau, T) - (s + (k+1)p) \\ &\leq e(j', \tau, T) - (s + (k+1)p) \\ &\leq e(j', \tau, T) - (3\tau - 1) \\ &< e(j', \tau, T) - \tau < j. \end{aligned}$$

  In both cases, we thus obtain $x < j$.

To sum up, we proved that $x \in \mathsf{R}^{-}_{s,k+1,H}(\tau, T)$ and $x < j$. Recalling that $j = \min \mathsf{R}^{-}_{s,k,H}(\tau, T)$, we thus obtain the second part of the claim, i.e.,

$$\min \mathsf{R}^{-}_{s,k+1,H}(\tau, T) \leq x < j = \min \mathsf{R}^{-}_{s,k,H}(\tau, T). \qquad \square$$

Lemma 9.17. *Let $T \in \Sigma^n$ and $\tau \in [1 \,..\, \lfloor \frac{n}{2} \rfloor]$. Let $b, e \in [0 \,..\, n]$ be such that $\{\mathrm{SA}_T[i]\}_{i \in (b \,..\, e]} = \mathsf{R}^{-}_{s,k,H}(\tau, T)$ for some $s \in \mathbb{Z}_{\geq 0}$, $k \in \mathbb{Z}_{>0}$, and $H \in \Sigma^+$ satisfying $\mathsf{R}^{-}_{s,k,H}(\tau, T) \neq \emptyset$. Denote $B_{\min} = \mathrm{MinPosBitvector}^{-}(\tau, T)$ (Definition 7.49). If $\mathsf{rank}_{B_{\min}}(b, 1) \neq \mathsf{rank}_{B_{\min}}(e, 1)$, then, letting $z = \mathsf{select}_{B_{\min}}(\mathsf{rank}_{B_{\min}}(b, 1) + 1, 1)$, it holds*

$$\min \mathsf{R}^{-}_{s,k,H}(\tau, T) = \mathrm{SA}_T[z].$$

Proof. Observe that by the assumption $\mathsf{rank}_{B_{\min}}(b, 1) \neq \mathsf{rank}_{B_{\min}}(e, 1)$ and the definition of rank and select queries, it holds $z \in (b \,..\, e]$ and $B_{\min}[z] = 1$. Consequently (see Definition 7.49), $\mathrm{SA}_T[z] \in \mathsf{R}^{-}_{\min}(\tau, T)$ (Definition 7.48), and hence, letting $Z = T[\mathrm{SA}_T[z] \,..\, e(\mathrm{SA}_T[z], \tau, T))$, it holds

$$\mathrm{SA}_T[z] = \min \mathrm{Occ}(Z, T) \cap \mathsf{R}^{-}(\tau, T).$$

Note also that we then have $\mathrm{Occ}(1, B_{\min}(b\,..\,z)) = \emptyset$. The proof consists of two steps:

- First, we prove that for every $x \in (b\,..\,z)$, it holds $\mathrm{SA}_T[x] > \mathrm{SA}_T[z]$. Suppose that this is not the case, and there exists $x \in (b\,..\,z)$ such that $\mathrm{SA}_T[x] < \mathrm{SA}_T[z]$. We then proceed as follows.
  - Denote $X = T[\mathrm{SA}_T[x]\,..\,e(\mathrm{SA}_T[x], \tau, T))$. First, observe that it holds $3\tau - 1 \leq |X| < |Z|$. The inequality $3\tau - 1 \leq |X|$ follows by definition. To show the other inequality, note that if $|X| \geq |Z|$, then by $\mathrm{SA}_T[x], \mathrm{SA}_T[z] \in \mathsf{R}^-_{s,H}(\tau, T)$, Z would be a prefix of X. Consequently, we would have $\mathrm{SA}_T[x] \in \mathrm{Occ}(Z, T) \cap \mathsf{R}^-(\tau, T)$. Since $\mathrm{SA}_T[x] < \mathrm{SA}_T[z]$, this contradicts $\mathrm{SA}_T[z] = \min \mathrm{Occ}(Z, T) \cap \mathsf{R}^-(\tau, T)$.
  - Denote $j = \min \mathrm{Occ}(X, T) \cap \mathsf{R}^-(\tau, T)$ and $J = T[j\,..\,e(j, \tau, T))$. Next, we show that $|X| \leq |J| < |Z|$. On the one hand, by $3\tau - 1 \leq |X|$ and Lemma 7.37(2), we have $j \in \mathsf{R}^-_{s,H}(\tau, T)$. By $\mathrm{SA}_T[x] \in \mathsf{R}^-_{s,H}(\tau, T)$, this implies that $|X| \leq |J|$. On the other hand, recall that we also have $\mathrm{SA}_T[z] \in \mathsf{R}^-_{s,H}(\tau, T)$. Thus, by $j \leq \mathrm{SA}_T[x] < \mathrm{SA}_T[z]$, we must have $|J| < |Z|$, since otherwise we would obtain a contradiction with $\mathrm{SA}_T[z] = \min \mathrm{Occ}(Z, T) \cap \mathsf{R}^-(\tau, T)$.
  - Lastly, observe that it holds $j \in \mathsf{R}^-_{\min}(\tau, T)$. To see this i.e., that we have $j = \min \mathrm{Occ}(J, T) \cap \mathsf{R}^-(\tau, T)$, note that a position $j' < j$ satisfying $j' \in \mathrm{Occ}(J, T) \cap \mathsf{R}^-(\tau, T)$ would contradict that $j = \min \mathrm{Occ}(X, T) \cap \mathsf{R}^-(\tau, T)$ (since X is a prefix of J).

  We have thus proved that the position j satisfies $j \in \mathsf{R}^-_{s,H}(\tau, T)$, and

  $$e(\mathrm{SA}_T[x], \tau, T) - \mathrm{SA}_T[x] \leq e(j, \tau, T) - j < e(\mathrm{SA}_T[z], \tau, T) - \mathrm{SA}_T[z].$$

  Since $\mathrm{SA}_T[x], \mathrm{SA}_T[z] \in \mathsf{R}^-_{s,k,H}(\tau, T)$, this implies that we also have $j \in \mathsf{R}^-_{s,k,H}(\tau, T)$. Moreover, since by Lemma 7.38(3) it holds

  $$e(\mathrm{SA}_T[b+1], \tau, T) - \mathrm{SA}_T[b+1] \leq e(\mathrm{SA}_T[b+2], \tau, T) - \mathrm{SA}_T[b+2] \leq \cdots \leq e(\mathrm{SA}_T[e], \tau, T) - \mathrm{SA}_T[e],$$

  it follows that there exists $x' \in (b\,..\,z)$ such that $j = \mathrm{SA}_T[x']$. Since we also proved that $j \in \mathsf{R}^-_{\min}(\tau, T)$, we have $B_{\min}[x'] = 1$. This contradicts that $\mathrm{Occ}(1, B_{\min}(b\,..\,z)) = \emptyset$, which concludes the proof that for every $x \in (b\,..\,z)$, we have $\mathrm{SA}_T[x] > \mathrm{SA}_T[z]$.
- Second, observe that by Lemma 7.51, for every $x \in (z\,..\,e]$, it holds $\mathrm{SA}_T[x] > \mathrm{SA}_T[z]$.

We have proved that for every $x \in (b\,..\,e] \setminus \{z\}$, it holds $\mathrm{SA}_T[x] > \mathrm{SA}_T[z]$. This implies that

$$\min \mathsf{R}^-_{s,k,H}(\tau, T) = \min\{\mathrm{SA}_T[i]\}_{i \in (b\,..\,e]} = \mathrm{SA}_T[z]. \qquad \square$$

Lemma 9.18. *Consider a text $T \in \Sigma^n$ and let $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$. Let $b, e \in [0\,..\,n]$ be such that $\{\mathrm{SA}_T[i]\}_{i \in (b\,..\,e]} = \bigcup_{k \in [k_1\,..\,k_2]} \mathsf{R}^-_{s,k,H}(\tau, T)$ for some $s \in \mathbb{Z}_{\geq 0}$, $k_1, k_2 \in \mathbb{Z}_{>0}$, and $H \in \Sigma^+$ satisfying $k_1 \leq k_2$, $\mathsf{R}^-_{s,k_1,H}(\tau, T) \neq \emptyset$, and $\mathsf{R}^-_{s,k_2,H}(\tau, T) \neq \emptyset$. Let $B_{\min} = \mathrm{MinPosBitvector}^-(\tau, T)$ (Definition 7.49).*

1. *If $\mathsf{rank}_{B_{\min}}(b, 1) \neq \mathsf{rank}_{B_{\min}}(e, 1)$, then, letting $z = \mathsf{select}_{B_{\min}}(\mathsf{rank}_{B_{\min}}(b, 1) + 1, 1)$,*

   $$\min\{\mathrm{SA}_T[i]\}_{i \in (b\,..\,e]} = \mathrm{SA}_T[z].$$

2. *If $\mathsf{rank}_{B_{\min}}(b, 1) = \mathsf{rank}_{B_{\min}}(e, 1)$, then*

   $$\min\{\mathrm{SA}_T[i]\}_{i \in (b\,..\,e]} = \min \mathsf{R}^-_{s,k_2,H}(\tau, T).$$

Proof. We prove each of the claims separately:

1. The assumption $\mathsf{rank}_{B_{\min}}(b, 1) \neq \mathsf{rank}_{B_{\min}}(e, 1)$ and the definition of rank and select queries imply that $z \in (b\,..\,e]$. Thus, there exists $k' \in [k_1\,..\,k_2]$ such that $\mathrm{SA}_T[z] \in \mathsf{R}^-_{s,k',H}(\tau, T)$. Let $b', e' \in [b\,..\,e]$ be such that $\{\mathrm{SA}_T[i]\}_{i \in (b'\,..\,e']} = \mathsf{R}^-_{s,k',H}(\tau, T)$ (see Remark 7.50). Note that then $z \in (b'\,..\,e']$. By Lemma 7.37, we also have $\{\mathrm{SA}_T[i]\}_{i \in (b\,..\,b']} = \bigcup_{k \in [k_1\,..\,k')} \mathsf{R}^-_{s,k,H}(\tau, T)$ and $\{\mathrm{SA}_T[i]\}_{i \in (e'\,..\,e]} = \bigcup_{k \in (k'\,..\,k_2]} \mathsf{R}^-_{s,k,H}(\tau, T)$, and hence

   $$B_{\min}(b\,..\,b'] = \bigodot_{k \in [k_1\,..\,k')} \mathrm{MinPosBitvector}^-_{s,k,H}(\tau, T)$$
   $$B_{\min}(e'\,..\,e] = \bigodot_{k \in (k'\,..\,k_2]} \mathrm{MinPosBitvector}^-_{s,k,H}(\tau, T).$$

   Finally, note that by definition of z, we have $B_{\min}[z] = 1$ and $\mathrm{Occ}(1, B_{\min}(b\,..\,z)) = \emptyset$. We are now ready to prove the claim.

- First, we prove that for every $i \in (b\,..\,b']$, it holds $\mathrm{SA}_T[i] > \min \mathsf{R}^-_{s,k',H}(\tau, T)$. If $k' = k_1$, then $b' = b$, and the claim holds. Let us thus assume that $k_1 < k'$, and recall that $\mathsf{R}^-_{s,k_1,H}(\tau, T) \neq \emptyset$. Observe that $z > b'$ and $\mathrm{Occ}(\mathtt{1}, B_{\min}(b\,..\,z)) = \emptyset$ imply that $\mathrm{Occ}(\mathtt{1}, \mathrm{MinPosBitvector}^-_{s,k_1,H}(\tau, T)) = \emptyset$. By Lemma 9.16, we then have $\mathsf{R}^-_{s,k_1+1,H}(\tau, T) \neq \emptyset$ and
$$\min \mathsf{R}^-_{s,k_1+1,H}(\tau, T) < \min \mathsf{R}^-_{s,k_1,H}(\tau, T).$$
If $k' = k_1 + 1$, then this proves the claim, because then we have $\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,b']} = \mathsf{R}^-_{s,k_1,H}(\tau, T)$, and by the above, for every $x \in \mathsf{R}^-_{s,k_1,H}(\tau, T) = \{\mathrm{SA}_T[i]\}_{i\in(b\,..\,b']}$, it holds $x \geq \min \mathsf{R}^-_{s,k_1,H}(\tau, T) > \min \mathsf{R}^-_{s,k_1+1,H}(\tau, T) = \min \mathsf{R}^-_{s,k',H}(\tau, T)$. Let us thus assume that $k' > k_1 + 1$. Using again $z > b'$ and $\mathrm{Occ}(\mathtt{1}, B_{\min}(b\,..\,z)) = \emptyset$, we obtain that $\mathrm{Occ}(\mathtt{1}, \mathrm{MinPosBitvector}^-_{s,k_1+1,H}(\tau, T)) = \emptyset$. By Lemma 9.16, we then obtain that it holds $\mathsf{R}^-_{s,k_1+2,H}(\tau, T) \neq \emptyset$ and $\min \mathsf{R}^-_{s,k_1+2,H}(\tau, T) < \min \mathsf{R}^-_{s,k_1+1,H}(\tau, T)$. If $k' = k_1 + 2$, then we obtain the claim, because we now have
$$\min \mathsf{R}^-_{s,k_1+2,H}(\tau, T) < \min \mathsf{R}^-_{s,k_1+1,H}(\tau, T) < \min \mathsf{R}^-_{s,k_1,H}(\tau, T),$$
which implies that for every $x \in \mathsf{R}^-_{s,k_1,H}(\tau, T) \cup \mathsf{R}^-_{s,k_1+1,H}(\tau, T) = \{\mathrm{SA}_T[i]\}_{i\in(b\,..\,b']}$, we have $x \geq \min \mathsf{R}^-_{s,k_1+1,H}(\tau, T) > \min \mathsf{R}^-_{s,k_1+2,H}(\tau, T) = \min \mathsf{R}^-_{s,k',H}(\tau, T)$. Continuing analogously, we obtain that no matter the value of k', for every $x \in \{\mathrm{SA}_T[i]\}_{i\in(b\,..\,b']}$, it holds $x > \min \mathsf{R}^-_{s,k',H}(\tau, T)$, or equivalently, that for every $i \in (b\,..\,b']$, $\mathrm{SA}_T[i] > \min \mathsf{R}^-_{s,k',H}(\tau, T)$.
- Next, we prove that for every $i \in (e'\,..\,e]$, it holds $\mathrm{SA}_T[i] > \min \mathsf{R}^-_{s,k',H}(\tau, T)$. First, observe that since
$$\mathsf{R}^-_{s,k',H}(\tau, T) \subseteq \textstyle\bigcup_{k\in[k_1\,..\,k_2]} \mathsf{R}^-_{s,k,H}(\tau, T) \subseteq \mathsf{R}^-_{s,H}(\tau, T),$$
it follows that, letting $\hat{b}, \hat{e} \in [0\,..\,n]$ be such that $\{\mathrm{SA}_T[i]\}_{i\in(\hat{b}\,..\,\hat{e}]} = \mathsf{R}^-_{s,H}(\tau, T)$ (see Remark 7.50), it holds
$$\hat{b} \leq b \leq b' < z \leq e' \leq e \leq \hat{e}.$$
Recall that $z \in (b'\,..\,e']$ and $B_{\min}[z] = \mathtt{1}$. Thus, by Lemma 7.51, it follows that for every $i \in (z\,..\,\hat{e}]$, it holds $\mathrm{SA}_T[i] > \mathrm{SA}_T[z]$. In particular, this includes all $i \in (e'\,..\,e]$, i.e., for such i, we have $\mathrm{SA}_T[i] > \mathrm{SA}_T[z] \geq \min \mathsf{R}^-_{s,k',H}(\tau, T)$.
- Finally, we prove that $\min \mathsf{R}^-_{s,k',H}(\tau, T) = \mathrm{SA}_T[z]$. First, note that $z \in (b'\,..\,e']$, $B_{\min}[z] = \mathtt{1}$, and $\{\mathrm{SA}_T[i]\}_{i\in(b'\,..\,e']} = \mathsf{R}^-_{s,k',H}(\tau, T)$ imply that $\mathsf{R}^-_{s,k',H}(\tau, T) \neq \emptyset$ and $\mathsf{rank}_{B_{\min}}(b', \mathtt{1}) \neq \mathsf{rank}_{B_{\min}}(e', \mathtt{1})$. One the other hand, note that $\mathrm{Occ}(\mathtt{1}, B_{\min}(b'\,..\,z)) = \emptyset$ implies that $z = \mathsf{select}_{B_{\min}}(\mathsf{rank}_{B_{\min}}(b', \mathtt{1}) + 1, \mathtt{1})$. Thus, by Lemma 9.17, we obtain $\min \mathsf{R}^-_{s,k',H}(\tau, T) = \mathrm{SA}_T[z]$.

By the above, for every $i \in (b\,..\,e] \setminus (b'\,..\,e']$, it holds $\mathrm{SA}_T[i] > \min \mathsf{R}^-_{s,k',H}(\tau, T) = \mathrm{SA}_T[z]$. On the other hand, since we have $\{\mathrm{SA}_T[i]\}_{i\in(b'\,..\,e']} = \mathsf{R}^-_{s,k',H}(\tau, T)$ and $\min \mathsf{R}^-_{s,k',H}(\tau, T) = \mathrm{SA}_T[z]$, we obtain that $\mathrm{SA}_T[i] > \mathrm{SA}_T[z]$ also for $i \in (b'\,..\,e'] \setminus \{z\}$. Thus, $\min\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]} = \mathrm{SA}_T[z]$.

2. The assumption $\mathsf{rank}_{B_{\min}}(b, \mathtt{1}) = \mathsf{rank}_{B_{\min}}(e, \mathtt{1})$ implies that $\mathrm{Occ}(\mathtt{1}, B_{\min}(b\,..\,e]) = \emptyset$. Consequently, for every $k \in [k_1\,..\,k_2]$, it holds $\mathrm{Occ}(\mathtt{1}, \mathrm{MinPosBitvector}^-_{s,k,H}(\tau, T)) = \emptyset$. Recall now that we assumed that $\mathsf{R}^-_{s,k_1,H}(\tau, T) \neq \emptyset$. Thus, by repeatedly applying Lemma 9.16 for $k = k_1, k_1 + 1, \ldots, k_2 - 1$, we obtain that for every $k \in [k_1\,..\,k_2]$, it holds $\mathsf{R}^-_{s,k,H}(\tau, T) \neq \emptyset$ and
$$\min \mathsf{R}^-_{s,k_1,H}(\tau, T) > \min \mathsf{R}^-_{s,k_1+1,H}(\tau, T) > \cdots > \min \mathsf{R}^-_{s,k_2,H}(\tau, T).$$
This implies $\min \bigcup_{k\in[k_1\,..\,k_2]} \mathsf{R}^-_{s,k,H}(\tau, T) = \min \mathsf{R}^-_{s,k_2,H}(\tau, T)$, or equivalently, $\min\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]} = \min \mathsf{R}^-_{s,k_2,H}(\tau, T)$, i.e., the claim. □

Algorithms

Proposition 9.19. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that given any $b, e \in [0\,..\,n]$ such that $b < e$ and $\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]} \subseteq \mathsf{R}^-_{s,H}(\tau, T)$ for some $s \in \mathbb{Z}_{\geq 0}$ and $H \in [0\,..\,\sigma)^+$, in $\mathcal{O}(\log \log n)$ time returns*
$$\min\{\mathrm{SA}_T[i] : i \in (b\,..\,e]\}.$$

Proof. Let $B_{\exp} = \text{ExpBlockBitvector}^-(\tau, T)$ (Definition 7.47) and $B_{\min} = \text{MinPosBitvector}^-(\tau, T)$ (Definition 7.49). Denote also $(a_i)_{i\in[1\,..\,q]} = \text{RunsLexSorted}^-(\tau, T)$ (Definition 7.44).

Components The data structure consists of the following components:

1. The data structure from Proposition 7.53 for text T. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
2. The bitvector $B_{\exp}$ augmented with the support for $\mathcal{O}(1)$-time rank and select queries using Theorem 2.7. The bitvector together with the augmentation of Theorem 2.7 needs $\mathcal{O}(n/\log n) = \mathcal{O}(n/\log_\sigma n)$ space.
3. The bitvector $B_{\min}$ augmented with Theorem 2.7 similarly as above. The bitvector together with the augmentation also needs $\mathcal{O}(n/\log n) = \mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n)$ space.

Implementation of queries The queries are answered as follows. Let $b, e \in [0\,..\,n]$ be such that $b < e$ and $\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]} \subseteq \mathsf{R}^-_{s,H}(\tau, T)$ for some $s \in \mathbb{Z}_{\geq 0}$ and $H \in [0\,..\,\sigma)^+$. We gradually develop an algorithm that, given b, e, returns $\min\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]}$.

First, let us assume that there exists $k \in \mathbb{Z}_{>0}$, such that $\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]} \subseteq \mathsf{R}^-_{s,k,H}(\tau, T)$. In this case, we compute $\min\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]}$ as follows:

1. Denote $i_1 = b + 1$ and $i_2 = e$. Using Proposition 7.53(1), in $\mathcal{O}(\log\log n)$ time compute $p_k = \mathrm{SA}_T[i_k]$, where $k \in \{1, 2\}$.
2. Using Proposition 7.53(2), in $\mathcal{O}(1)$ time compute $s = \mathrm{head}(p_1, \tau, T)$, $p = |\mathrm{root}(p_1, \tau, T)|$, and $k = \exp(p_1, \tau, T)$. We then set $\delta_{\text{text}} = s + pk$.
3. Using Proposition 7.53(3), in $\mathcal{O}(1)$ time compute indexes $j_k \in [1\,..\,q]$ such that $e^{\text{full}}(p_k, \tau, T) = e^{\text{full}}(a_{j_k}, \tau, T)$, where $k \in \{1, 2\}$. Note that then, by Lemma 9.15, there exists $j \in (j_1 - 1\,..\,j_2]$ such that $A_{\text{len}}[j] \geq \delta_{\text{text}}$.
4. Using Proposition 7.53(5), in $\mathcal{O}(\log\log n)$ time compute $x = A_{\text{pos}}[\mathsf{prefix\text{-}rmq}_{A_{\text{pos}},A_{\text{len}}}(j_1 - 1, j_2, \delta_{\text{text}})]$. By Lemma 9.15, it holds $\min\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]} = x - \delta_{\text{text}}$. Thus, we return $x - \delta_{\text{text}}$ as the answer.

In total, we spend $\mathcal{O}(\log\log n)$ time.

Let us now consider a general case, where $\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]} \subseteq \mathsf{R}^-_{s,H}(\tau, T)$. We then compute the value $\min\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]}$ as follows:

1. In $\mathcal{O}(1)$ time compute $x = \mathsf{select}_{B_{\exp}}(\mathsf{rank}_{B_{\exp}}(b, 1) + 1, 1)$. If $x \geq e$, then by Definition 7.47, there exists $k \in \mathbb{Z}_{>0}$ such that $\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]} \subseteq \mathsf{R}^-_{s,k,H}(\tau, T)$. Thus, in this case we compute $\min\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]}$ using the algorithm presented above in $\mathcal{O}(\log\log n)$ time. Let us now assume that $x < e$.
2. In $\mathcal{O}(1)$ time, compute $y = \mathsf{select}_{B_{\exp}}(\mathsf{rank}_{B_{\exp}}(e - 1, 1), 1)$. Note that then $b < x \leq y < e$ and, by Definition 7.47, there exist $k_1, k_2 \in \mathbb{Z}_{>0}$ such that $k_1 < k_2$, and it holds:
   - $\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,x]} \subseteq \mathsf{R}^-_{s,k_1,H}(\tau, T)$,
   - $\{\mathrm{SA}_T[i]\}_{i\in(x\,..\,y]} = \bigcup_{k\in(k_1\,..\,k_2)} \mathsf{R}^-_{s,k,H}(\tau, T)$,
   - $\{\mathrm{SA}_T[i]\}_{i\in(y\,..\,e]} \subseteq \mathsf{R}^-_{s,k_2,H}(\tau, T)$.

   Initialize the set $\mathcal{M} = \emptyset$, and then compute the minimum of each of the above sets as follows:
   - The computation of $m_1 = \min\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,x]}$ proceeds using the algorithm described above and takes $\mathcal{O}(\log\log n)$ time. We then add m_1 to the set $\mathcal{M}$.
   - Next, we check if $x = y$. If so, we skip this step. Let us thus assume that $x < y$. Then, it holds $k_1 + 1 < k_2$. In $\mathcal{O}(1)$ time we check if $\mathsf{rank}_{B_{\min}}(x, 1) \neq \mathsf{rank}_{B_{\min}}(y, 1)$. Consider two cases:
     - First, assume that $\mathsf{rank}_{B_{\min}}(x, 1) \neq \mathsf{rank}_{B_{\min}}(y, 1)$. In $\mathcal{O}(1)$ time compute the value $z = \mathsf{select}_{B_{\min}}(\mathsf{rank}_{B_{\min}}(x, 1) + 1, 1)$. Using Proposition 7.53(1), in $\mathcal{O}(\log\log n)$ time compute $m_2 = \mathrm{SA}_T[z]$. By Lemma 9.18, $\min\{\mathrm{SA}_T[i]\}_{i\in(x\,..\,y]} = m_2$. Add m_2 to $\mathcal{M}$.
     - Let us now assume that $\mathsf{rank}_{B_{\min}}(x, 1) = \mathsf{rank}_{B_{\min}}(y, 1)$. In $\mathcal{O}(1)$ time compute the position $y' = \mathsf{select}_{B_{\exp}}(\mathsf{rank}_{B_{\exp}}(y - 1, 1), 1)$. By Definition 7.47, we then have $x \leq y' < y$ and $\{\mathrm{SA}_T[i]\}_{i\in(y'\,..\,y]} = \mathsf{R}^-_{s,k_2-1,H}(\tau, T)$. Using the algorithm described above, in $\mathcal{O}(\log\log m)$ time we compute $m_2 = \min\{\mathrm{SA}_T[i]\}_{i\in(y'\,..\,y]}$, and add m_2 to $\mathcal{M}$. Note that by Lemma 9.18, it holds $\min\{\mathrm{SA}_T[i]\}_{i\in(x\,..\,y]} = \min \mathsf{R}^-_{s,k_2-1,H}(\tau, T) = m_2$.
   - The computation of $m_3 = \min\{\mathrm{SA}_T[i]\}_{i\in(y\,..\,e]}$ proceeds as above and takes $\mathcal{O}(\log\log n)$ time. We

then add m_3 to the set $\mathcal{M}$.

By the above discussion, we have $2 \leq |\mathcal{M}| \leq 3$ and $\min\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]} = \min \mathcal{M}$. Thus, in $\mathcal{O}(1)$ time we compute and return $\min \mathcal{M}$.

In total, the query in the general case takes $\mathcal{O}(\log\log n)$ time.

Construction algorithm The components of the structure are constructed as follows:

1. By Proposition 7.53, the construction of the first component takes $\mathcal{O}(n/\log_\sigma n)$ time.
2. To construct the second component, first, in $\mathcal{O}(n/\log_\sigma n)$ time we construct the bitvector $B_{\mathrm{exp}} = \mathrm{ExpBlockBitvector}^-(\tau, T)$ (Definition 7.47) using Proposition 7.52(1). We then augment B_{exp} using Theorem 2.7 in $\mathcal{O}(n/\log n) = \mathcal{O}(n/\log_\sigma n)$ time.
3. The third component of the data structure is constructed similarly as above, by combining Proposition 7.52(2) and Theorem 2.7.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n)$ time. □

Proposition 9.20. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that given any $b, e \in [0\,..\,n]$ such that $b < e$ and $\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]} \subseteq \mathsf{R}_{s,H}(\tau, T)$ for some $s \in \mathbb{Z}_{\geq 0}$ and $H \in [0\,..\,\sigma)^+$, in $\mathcal{O}(\log\log n)$ time returns*

$$\min\{\mathrm{SA}_T[i] : i \in (b\,..\,e]\}.$$

Proof. **Components** The data structure consists of the following components:

1. The structure from Proposition 7.53 for text T. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
2. The structure from Proposition 9.19 for text T. It also needs $\mathcal{O}(n/\log_\sigma n)$ space.
3. The symmetric version of the structure from Proposition 9.19 adapted to positions in $\mathsf{R}^+(\tau, T)$ (see Definition 7.33). It also needs $\mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n)$ space.

Implementation of queries The queries are answered as follows. Let $b, e \in [0\,..\,n]$ be such that $b < e$ and $\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]} \subseteq \mathsf{R}_{s,H}(\tau, T)$ for some $s \in \mathbb{Z}_{\geq 0}$ and $H \in [0\,..\,\sigma)^+$. Given b, e, we compute $\min\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]}$ as follows:

1. Using Proposition 7.53(1), in $\mathcal{O}(\log\log n)$ time we compute $j = \mathrm{SA}_T[e]$.
2. Using Proposition 7.53(2), in $\mathcal{O}(1)$ time we then compute $x, y, z \in [0\,..\,n]$ such that $x \leq y \leq z$, $\{\mathrm{SA}_T[i]\}_{i\in(x\,..\,y]} = \mathsf{R}^-_{s,H}(\tau, T)$, and $\{\mathrm{SA}_T[i]\}_{i\in(y\,..\,z]} = \mathsf{R}^+_{s,H}(\tau, T)$, where $s = \mathrm{head}(j, \tau, T)$ and $H = \mathrm{root}(j, \tau, T)$. Note that then $x \leq b < e \leq z$.
3. In $\mathcal{O}(1)$ time initialize $\mathcal{M} = \emptyset$.
4. If $b < y$, then using Proposition 9.19, in $\mathcal{O}(\log\log n)$ time compute
$$m^- = \min\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,\min(e,y)]},$$
and add m^- to $\mathcal{M}$. Note that $\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,\min(e,y)]} \subseteq \mathsf{R}^-_{s,H}(\tau, T)$.
5. If $e > y$, then using the symmetric version of the structure from Proposition 9.19, in $\mathcal{O}(\log\log n)$ time compute
$$m^+ = \min\{\mathrm{SA}_T[i]\}_{i\in(\max(b,y),e]},$$
and add m^+ to $\mathcal{M}$. Note that $\{\mathrm{SA}_T[i]\}_{i\in(\max(b,y)\,..\,e]} \subseteq \mathsf{R}^+_{s,H}(\tau, T)$.
6. Note that at this point, we have $1 \leq |\mathcal{M}| \leq 2$, and $\min\{\mathrm{SA}_T[i]\}_{i\in(b\,..\,e]} = \min \mathcal{M}$. Thus, in $\mathcal{O}(1)$ time we compute and return $\min \mathcal{M}$ as the answer.

In total, answering the query takes $\mathcal{O}(\log\log n)$ time.

Construction algorithm The components of the data structure are constructed as follows:

1. By Proposition 7.53, the construction of the first component takes $\mathcal{O}(n/\log_\sigma n)$ time.
2. By Proposition 9.19, the construction of the second component takes $\mathcal{O}(n/\log_\sigma n)$ time.
3. Since the third component is a symmetric version of the second component, its construction also takes $\mathcal{O}(n/\log_\sigma n)$ time.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n)$ time. □

Proposition 9.21. *Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$. Given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n)$ time construct a data structure that answers the following queries:*

1. *Given the packed representation of τ-periodic (Definition 7.5) patterns $P_1, P_2 \in [0\,..\,\sigma)^+$ such that $\mathrm{lcp}(P_1, P_2) \geq 3\tau - 1$, computes $\min \mathrm{LexRange}(P_1, P_2, T) \cup \{\infty\}$ (Definition 2.4) in $\mathcal{O}(\log\log n + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.*
2. *Given the packed representation of a τ-periodic pattern $P \in [0\,..\,\sigma)^+$ satisfying $|P| \geq 3\tau - 1$, computes $\min \mathrm{LexRange}(P, P'c^\infty, T) \cup \{\infty\}$ (where $P' = P[1\,..\,3\tau - 1]$ and $c = \sigma - 1$) in $\mathcal{O}(\log\log n + |P|/\log_\sigma n)$ time.*

Proof. **Components** The data structure consists of the following components:

1. The data structure from Proposition 7.53 using $\mathcal{O}(n/\log_\sigma n)$ space.
2. The data structure from Proposition 9.20 using $\mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n)$ space.

Implementation of queries The queries are answered as follows:

1. Let $P_1, P_2 \in [0\,..\,\sigma)^+$ be τ-periodic patterns such that $\mathrm{lcp}(P_1, P_2) \geq 3\tau - 1$. Given the packed representation of P_1 and P_2, we compute $\min \mathrm{LexRange}(P_1, P_2, T) \cup \{\infty\}$ as follows:
   (a) Using Proposition 7.53(4), in $\mathcal{O}(\log\log n + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time we compute $b_k = \mathrm{RangeBeg}(P_k, T)$, where $k \in \{1, 2\}$. If $b_1 \geq b_2$, then $\mathrm{LexRange}(P_1, P_2, T) = \emptyset$ (see Remark 2.5), and hence we return $\min \mathrm{LexRange}(P_1, P_2, T) \cup \{\infty\} = \infty$. Let us now assume that $b_1 < b_2$. By Remark 2.5, we then have
   $$\mathrm{LexRange}(P_1, P_2, T) = \{\mathrm{SA}_T[i]\}_{i \in (b_1\,..\,b_2]}.$$
   By $\mathrm{lcp}(P_1, P_2) \geq 3\tau - 1$, for every $j_1, j_2 \in \mathrm{LexRange}(P_1, P_2, T)$, it holds $\mathrm{LCE}_T(j_1, j_2) \geq 3\tau - 1$. Thus, by Lemma 7.37(2), there exists $s \in \mathbb{Z}_{\geq 0}$ and $H \in [0\,..\,\sigma)^+$ such that $\{\mathrm{SA}_T[i]\}_{i \in (b_1\,..\,b_2]} \subseteq \mathsf{R}_{s,H}(\tau, T)$.
   (b) Using Proposition 9.20, in $\mathcal{O}(\log\log n)$ time compute and return $\min\{\mathrm{SA}_T[i]\}_{i \in (b_1\,..\,b_2]}$.

   In total, we spend $\mathcal{O}(\log\log n + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.
2. Let us now consider a τ-periodic pattern $P \in [0\,..\,\sigma)^+$ that satisfies $|P| \geq 3\tau - 1$, where $P' = P[1\,..\,3\tau - 1]$ and $c = \sigma - 1$. Given the packed representation of P, we compute $\min \mathrm{LexRange}(P, P'c^\infty, T) \cup \{\infty\}$ similarly as above, letting $P_1 = P$ and $P_2 = P'c^\infty$. The main difference is that to compute $b_2 = \mathrm{RangeBeg}(P_2, T)$, we observe that $\mathrm{RangeBeg}(P'c^\infty, T) = \mathrm{RangeEnd}(P', T)$. Thus, we can determine the value of b_2 using Proposition 7.53(4) in $\mathcal{O}(\log\log n + |P'|/\log_\sigma n) = \mathcal{O}(\log\log n)$ time.

Construction algorithm The components of the structure are constructed as follows:

1. The first component is constructed using Proposition 7.53 in $\mathcal{O}(n/\log_\sigma n)$ time.
2. The second component is constructed in the same time using Proposition 9.20.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n)$ time. □

9.3.1.4 Summary

Proposition 9.22. *Consider a data structure answering prefix RMQ (see Section 9.1) that, for any array of integers $A[1\,..\,k]$ (where $A[i] \in [1\,..\,k]$ for $i \in [1\,..\,k]$) and a sequence $W[1\,..\,k]$ strings of length ℓ over alphabet*

$[0\,..\,\sigma)$ achieves the following complexities (where the input strings at query time and during construction are given in the packed representation):

- *space usage $S(k,\ell,\sigma)$,*
- *preprocessing time $P_t(k,\ell,\sigma)$,*
- *preprocessing space $P_s(k,\ell,\sigma)$,*
- *query time $Q(k,\ell,\sigma)$.*

Let $T \in [0\,..\,\sigma)^n$ be such that $2 \leq \sigma < n^{1/13}$ and $T[n]$ does not occur in $T[1\,..\,n)$. There exist positive integers $k = \Theta(n/\log_\sigma n)$ and $\ell \leq (1+\lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n + P_t(k,\ell,\sigma))$ time and using $\mathcal{O}(n/\log_\sigma n + P_s(k,\ell,\sigma))$ working space construct a data structure of size $\mathcal{O}(n/\log_\sigma n + S(k,\ell,\sigma))$ that, given the packed representation of any patterns $P_1, P_2 \in [0\,..\,\sigma)^*$, returns $\min \mathrm{LexRange}(P_1,P_2,T) \cup \{\infty\}$ (see Definition 2.4) in $\mathcal{O}(\log\log n + Q(k,\ell,\sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.

Proof. We use the following definitions. Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is a positive constant smaller than $\frac{1}{12}$ such that $\tau \geq 1$ (such τ exists by the assumption $\sigma < n^{1/13}$). Let k and ℓ be integers resulting from the application of Proposition 9.14 to text T, with the structure from the above claim to answer prefix RMQ. Note that they satisfy $k = \Theta(n/\log_\sigma n)$ and $\ell \leq (1+\lfloor \log k \rfloor)/\lceil \log \sigma \rceil$.

Components The data structure consists of the following components:

1. The structure from Proposition 7.8 for σ and τ. If needs $\mathcal{O}(\sigma^{3\tau}\cdot\tau^2) = \mathcal{O}(n/\log_\sigma n)$ space.
2. The structure from Proposition 7.10 for τ and T. It needs $\mathcal{O}(n/\log_\sigma n)$ space.
3. The structure from Proposition 9.10 for τ and T. It uses $\mathcal{O}(n/\log_\sigma n)$ space.
4. The structure from Proposition 9.14 for τ and T, and using the structure from the above claim to answer prefix RMQ. It uses $\mathcal{O}(n/\log_\sigma n + S(k,\ell,\sigma))$ space, where k and ℓ are defined above.
5. The structure from Proposition 9.21 for τ and T. It uses $\mathcal{O}(n/\log_\sigma n)$ space.

In total, the structure needs $\mathcal{O}(n/\log_\sigma n + S(k,\ell,\sigma))$ space.

Implementation of queries The queries are answered as follows. Let $P_1, P_2 \in [0\,..\,\sigma)^*$. Given the packed representation of P_1 and P_2, we compute $\min \mathrm{LexRange}(P_1,P_2,T) \cup \{\infty\}$ as follows. First, in $\mathcal{O}(1 + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time we check if $P_1 \prec P_2$. If not, then we have $\mathrm{LexRange}(P_1,P_2,T) = \emptyset$. We then return $\min \mathrm{LexRange}(P_1,P_2,T) \cup \{\infty\} = \infty$, and the query algorithm is complete. Let us thus assume that it holds $P_1 \prec P_2$. In $\mathcal{O}(1)$ time we use the packed representation of P_1 and P_2 to compute the packed representation of prefixes $X_1 = P_1[1\,..\,\min(3\tau-1, |P_1|)]$ and $X_2 = P_2[1\,..\,\min(3\tau-1,|P_2|)]$. Note that the assumption $P_1 \prec P_2$ implies that $X_1 \preceq X_2$. In $\mathcal{O}(1)$ time we check if $X_1 = X_2$. In $\mathcal{O}(1)$ time we also check if X_1 is a prefix of X_2. We then consider three cases:

- First, assume that $X_1 = X_2$. We then must have $|X_1| = |X_2| = 3\tau - 1$, since otherwise we would have $P_1 = X_1 = X_2 = P_2$, contradicting $P_1 \prec P_2$. Using Proposition 7.8, in $\mathcal{O}(1)$ time we check if X_1 is τ-periodic (Definition 7.5). Consider two cases:
  - If X_1 is τ-periodic, then so are P_1 and P_2. The value $\min \mathrm{LexRange}(P_1,P_2,T) \cup \{\infty\}$ is computed using Proposition 9.21(1) in $\mathcal{O}(\log\log n + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.
  - Otherwise, the strings P_1 and P_2 are τ-nonperiodic. Since $\mathrm{lcp}(P_1,P_2) \geq 3\tau - 1$, we then compute $\min \mathrm{LexRange}(P_1,P_2,T) \cup \{\infty\}$ in $\mathcal{O}(\log\log n + Q(k,\ell,\sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time using Proposition 9.14(1).

  In total, we spend $\mathcal{O}(\log\log n + Q(k,\ell,\sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.
- Let us now assume that $X_1 \neq X_2$ and X_1 is a prefix of X_2. Note that then X_1 must be a proper prefix of X_2, and hence $|X_1| < |X_2|$. Thus, we must have $P_1 = X_1$. Combining this with $X_1 \preceq X_2$ (see above), we obtain $P_1 \preceq X_2 \preceq P_2$. Consequently, we can write $\mathrm{LexRange}(P_1,P_2,T)$ as a disjoint union:

$$\mathrm{LexRange}(P_1,P_2,T) = \mathrm{LexRange}(P_1,X_2,T) \cup \mathrm{LexRange}(X_2,P_2,T).$$

  We will separately handle each of the two sets, and then combine the result. Using Proposition 9.10(1), we compute $\min \mathrm{LexRange}(X_1,X_2,T) \cup \{\infty\}$ (recall that $P_1 = X_1$) in $\mathcal{O}(1)$ time. Let us now

focus on computing $\min \mathrm{LexRange}(X_2, P_2, T) \cup \{\infty\}$. If $|P_2| < 3\tau - 1$, then again we compute $\min \mathrm{LexRange}(X_2, P_2, T) \cup \{\infty\}$ using Proposition 9.10(1) in $\mathcal{O}(1)$ time. Let us now assume that $|P_2| \geq 3\tau - 1$. Using Proposition 7.8, in $\mathcal{O}(1)$ time we check if P_2 is τ-periodic (Definition 7.5). Consider two cases:

- If P_2 is τ-periodic, then using Proposition 9.21(1), we compute $\min \mathrm{LexRange}(X_2, P_2, T) \cup \{\infty\}$ in $\mathcal{O}(\log\log n + |P_2|/\log_\sigma n)$ time. We can use this because $X_2 = P_2[1\,..\,3\tau - 1]$.
- If P_2 is τ-nonperiodic, then using Proposition 9.14(1), compute $\min \mathrm{LexRange}(X_2, P_2, T) \cup \{\infty\}$ in $\mathcal{O}(\log\log n + Q(k, \ell, \sigma) + |P_2|/\log_\sigma n)$ time.

In total, we spend $\mathcal{O}(\log\log n + Q(k, \ell, \sigma) + |P_2|/\log_\sigma n)$ time.

- Let us now assume that $X_1 \neq X_2$ and X_1 is not a prefix of X_2. Observe that, letting $c = \sigma - 1$, we then have $X_1 c^\infty \prec X_2$. Combining this with $P_1 \preceq X_1 c^\infty$ and $X_2 \preceq P_2$, we can thus write $\mathrm{LexRange}(P_1, P_2, T)$ as a disjoint union:

$$\begin{aligned}\mathrm{LexRange}(P_1, P_2, T) = {} & \mathrm{LexRange}(P_1, X_1 c^\infty, T) \cup \\ & \mathrm{LexRange}(X_1 c^\infty, X_2, T) \cup \\ & \mathrm{LexRange}(X_2, P_2, T).\end{aligned}$$

We will separately compute the minimum of each of the three sets. Based on this, we can determine $\min \mathrm{LexRange}(P_1, P_2, T) \cup \{\infty\}$. We begin with the set $\mathrm{LexRange}(P_1, X_1 c^\infty, T)$. If $|P_1| \leq 3\tau - 1$, then $P_1 = X_1$, and hence we compute $\min \mathrm{LexRange}(P_1, X_1 c^\infty, T) \cup \{\infty\}$ in $\mathcal{O}(1)$ time using Proposition 9.10(2). Let us now assume that $|P_1| > 3\tau - 1$. In $\mathcal{O}(1)$ time we check if P_1 is τ-periodic using Proposition 7.8. We then consider two cases:

- If P_1 is τ-periodic, then we compute $\min \mathrm{LexRange}(P_1, X_1 c^\infty, T) \cup \{\infty\}$ using Proposition 9.21(2) in $\mathcal{O}(\log\log n + |P_1|/\log_\sigma n)$ time.
- Otherwise (i.e., if P_1 is τ-nonperiodic), we compute $\min \mathrm{LexRange}(P_1, X_1 c^\infty, T) \cup \{\infty\}$ using Proposition 9.14(2) in $\mathcal{O}(\log\log n + Q(k, \ell, \sigma) + |P_1|/\log_\sigma n)$ time.

Next, we compute $\min \mathrm{LexRange}(X_1 c^\infty, X_2, T) \cup \{\infty\}$ using Proposition 9.10(3) in $\mathcal{O}(1)$ time. Finally, we address the problem of computing $\min \mathrm{LexRange}(X_2, P_2, T) \cup \{\infty\}$. If $|P_2| \leq 3\tau - 1$, then $P_2 = X_2$, and hence we have $\mathrm{LexRange}(X_2, P_2, T) = \emptyset$. We thus obtain $\min \mathrm{LexRange}(X_2, P_2, T) \cup \{\infty\} = \infty$. Let us now assume that $|P_2| > 3\tau - 1$. In $\mathcal{O}(1)$ time we check if P_2 is τ-periodic using Proposition 7.8. We then consider two cases:

- If P_2 is τ-nonperiodic, then we compute $\min \mathrm{LexRange}(X_2, P_2, T) \cup \{\infty\}$ using Proposition 9.14(1) in $\mathcal{O}(\log\log n + Q(k, \ell, \sigma) + |P_2|/\log_\sigma n)$ time.
- Otherwise (i.e., if P_2 is τ-periodic), we compute $\min \mathrm{LexRange}(X_2, P_2, T) \cup \{\infty\}$ using Proposition 9.21(1) in $\mathcal{O}(\log\log n + |P_2|/\log_\sigma n)$ time.

We spend $\mathcal{O}(\log\log n + Q(k, \ell, \sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time in the above step.

Over all steps, we spend $\mathcal{O}(\log\log n + Q(k, \ell, \sigma) + |P_1|/\log_\sigma n + |P_2|/\log_\sigma n)$ time.

Construction algorithm The components of the structure are constructed as follows:

1. In $\mathcal{O}(\sigma^{3\tau} \cdot \tau^2) = \mathcal{O}(n/\log_\sigma n)$ time we apply Proposition 7.8.
2. In $\mathcal{O}(n/\log_\sigma n)$ time we apply Proposition 7.10 to text T.
3. In the same time as above we apply Proposition 9.10 to text T.
4. In $\mathcal{O}(n/\log_\sigma n + P_t(k, \ell, \sigma))$ time and using $\mathcal{O}(n/\log_\sigma n + P_s(k, \ell, \sigma))$ working space we apply Proposition 9.14 to text T.
5. In $\mathcal{O}(n/\log_\sigma n)$ time we apply Proposition 9.21 to text T.

In total, the construction takes $\mathcal{O}(n/\log_\sigma n + P_t(k, \ell, \sigma))$ time uses $\mathcal{O}(n/\log_\sigma n + P_s(k, \ell, \sigma))$ working space. □

9.3.2 Alphabet Reduction for Prefix RMQ

Proposition 9.23. *Consider a data structure answering prefix RMQ queries (see Section 9.1) that, for any array of m integers in the range $[1\,..\,m]$, and any sequence of m binary strings of length $1 + \lfloor \log m \rfloor$ (where*

$m \geq 1$), achieves the following complexities (where both the string at query time as well as input strings during construction are given in the packed representation):

- *space usage $S(m)$,*
- *preprocessing time $P_t(m)$,*
- *preprocessing space $P_s(m)$,*
- *query time $Q(m)$.*

Let $m' \geq 1$ and let $A[1\,..\,m']$ be a sequence of m' integers (where for every $i \in [1\,..\,m']$, it holds $A[i] \in [1\,..\,m']$) and $W[1\,..\,m']$ be a sequence of m' equal-length strings over alphabet $[0\,..\,\sigma)$ (where $\sigma \geq 2$) of length $\ell \leq (1 + \lfloor \log m' \rfloor)/\lceil \log \sigma \rceil$. Given the arrays A and W, with all strings represented in the packed form, we can in $\mathcal{O}(m' + P_t(m'))$ time and using $\mathcal{O}(m' + P_s(m'))$ working space construct a data structure of size $\mathcal{O}(m' + S(m'))$ that, given the packed representation of any $X \in [0\,..\,\sigma)^{\leq \ell}$ and any $b, e \in [0\,..\,m']$, computes $\mathsf{prefix\text{-}rmq}_{A,W}(b, e, X)$ (Definition 2.9) in $\mathcal{O}(Q(m'))$ time.

Proof. The above reduction is essentially the same as the one presented in Proposition 4.20, except we only need Lemma 4.19(1c). □

9.3.3 Summary

Theorem 9.24. *Consider a data structure answering prefix RMQ (see Section 9.1) that, for any sequence of m integers in the range $[1\,..\,m]$, and any sequence of m binary strings of length $1 + \lfloor \log m \rfloor$ (where $m \geq 1$), achieves the following complexities (where both the string at query time as well as input strings during construction are given in the packed representation):*

- *space usage $S(m)$,*
- *preprocessing time $P_t(m)$,*
- *preprocessing space $P_s(m)$,*
- *query time $Q(m)$.*

For every $T \in \{0,1\}^n$, there exists $m = \Theta(n/\log n)$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log n + P_t(m))$ time and using $\mathcal{O}(n/\log n + P_s(m))$ working space construct a data structure of size $\mathcal{O}(n/\log n + S(m))$ that, given the packed representation of any $P_1, P_2 \in \{0,1\}^*$, computes $\min \mathrm{LexRange}(P_1, P_2, T) \cup \{\infty\}$ (see Definition 2.4) in $\mathcal{O}(\log\log n + Q(m) + |P_1|/\log n + |P_2|/\log n)$ time.

Proof. Assume that $n > 3^{13} - 1$ (otherwise, the claim holds trivially).

We use the following definitions. Let $\sigma = 3$ and let $T' \in [0\,..\,\sigma)^{n+1}$ be a string defined so that $T'[1\,..\,n] = T$ and $T'[n+1] = 2$. Denote $n' = |T'|$. Note that $T'[n']$ does not occur in $T'[1\,..\,n')$ and it holds $2 \leq \sigma < (n')^{1/13}$. Let D denote the data structure resulting from applying Proposition 9.23 with alphabet size σ to the structure (answering prefix RMQ on binary strings) form the claim. Given any sequence $A[1\,..\,m']$ of m' integers in $[1\,..\,m']$, and any sequence $W[1\,..\,m']$ of m' equal-length strings over alphabet $[0\,..\,\sigma)$ of length $\ell' \leq (1 + \lfloor \log m' \rfloor)/\lceil \log \sigma \rceil$ (where $m' \geq 1$) as input (with all strings represented in the packed form), the structure D achieves the following complexities (where at query time it is given the packed representation of any $X \in [0\,..\,\sigma)^{\leq \ell'}$ and any $b, e \in [0\,..\,m']$, and returns $\mathsf{prefix\text{-}rmq}_{A,W}(b, e, X)$):

- space usage $S'(m', \ell', \sigma) = \mathcal{O}(m' + S(m'))$,
- preprocessing time $P'_t(m', \ell', \sigma) = \mathcal{O}(m' + P_t(m'))$,
- preprocessing space $P'_s(m', \ell', \sigma) = \mathcal{O}(m' + P_s(m'))$,
- query time $Q'(m', \ell', \sigma) = \mathcal{O}(Q(m'))$,

where the complexities $S(m')$, $P_t(m')$, $P_s(m')$, and $Q(m')$ refer to the structure from the claim (answering prefix RMQ for binary strings).

Components The data structure answering lex-range minimum queries consists of a single component: the data structure from Proposition 9.22 applied for text T' and with the structure D as the structure answering prefix RMQ queries. Note that we can use D, since it supports prefix RMQ queries for the combination of parameters required in Proposition 9.22, i.e., for sequences of strings over alphabet $[0\,..\,\sigma)$, with the length ℓ of all strings satisfying $\ell \leq (1 + \lfloor \log k \rfloor)/\lceil \log \sigma \rceil$ (where k is the length of the input sequence). To bound the space usage of this component, note that by Proposition 9.22 and the above

discussion, there exists $m = \Theta(n'/\log_\sigma n') = \Theta(n/\log n)$ (recall that $\sigma = 3$) such that the structure needs $\mathcal{O}(n'/\log_\sigma n' + S'(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + S(m)) = \mathcal{O}(n/\log n + S(m))$ space.

Implementation of queries The lex-range minimum queries are answered as follows. Let $P_1, P_2 \in \{\mathtt{0},\mathtt{1}\}^*$. Note that by definition of T', it follows that $\min \mathrm{LexRange}(P_1,P_2,T)\cup\{\infty\} = \min \mathrm{LexRange}(P_1,P_2,T')\cup\{\infty\}$. By Proposition 9.22 and the above discussion, computing the value of $\min \mathrm{LexRange}(P_1,P_2,T') \cup \{\infty\}$ takes $\mathcal{O}(\log\log n' + Q'(m,\ell,\sigma) + |P_1|/\log_\sigma n' + |P_2|/\log_\sigma n') = \mathcal{O}(\log\log n + Q(m) + |P_1|/\log n + |P_2|/\log n)$ time.

Construction algorithm By Proposition 9.22 and the above discussion, construction of the above data structure answering lex-range minimum queries on T' (and hence also on T) takes $\mathcal{O}(n'/\log_\sigma n' + P'_t(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + P_t(m)) = \mathcal{O}(n/\log n + P_t(m))$ time and uses $\mathcal{O}(n'/\log_\sigma n' + P'_s(m,\ell,\sigma)) = \mathcal{O}(n'/\log_\sigma n' + m + P_s(m)) = \mathcal{O}(n/\log n + P_s(m))$ working space. □

10 Equivalence of Pattern Ranking and Pattern SA-Interval Queries

10.1 Problem Definitions

Indexing for Pattern SA-Interval Queries

Input: The packed representation of a nonempty string $T \in \{\mathtt{0},\mathtt{1}\}^n$.
Output: A data structure that, given the packed representation of any $P \in \{\mathtt{0},\mathtt{1}\}^*$, returns the pair $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ (Definition 2.1).

10.2 Problem Reduction

Definition 10.1 (Alphabet inversion)**.** Let $\sigma \geq 2$. For every $S \in [0\,..\,\sigma)^*$, by $\mathrm{InvString}_\sigma(S)$ we denote a string $S_{\mathrm{inv}} \in [0\,..\,\sigma)^*$ of length $|S_{\mathrm{inv}}| = |S|$ such that for every $i \in [1\,..\,|S|]$, it holds $S_{\mathrm{inv}}[i] = \sigma - 1 - S[i]$.

Lemma 10.2. *Let $\sigma \geq 2$ and let $S_1, S_2 \in [0\,..\,\sigma)^*$ be such that S_1 is not a prefix of S_2. Then, $S_1 \succ S_2$ holds if and only if $\mathrm{InvString}_\sigma(S_1) \prec \mathrm{InvString}_\sigma(S_2)$ (see Definition 10.1).*

Proof. Denote $S'_k = \mathrm{InvString}_\sigma(S_k)$, where $k \in \{1,2\}$.

First, assume that $S_1 \succ S_2$. This implies that for some $\ell \in [0\,..\,\min(|S_1|,|S_2|))$, it holds $S_1[1\,..\,\ell] = S_2[1\,..\,\ell]$ and $S_1[\ell+1] \succ S_2[\ell+1]$. By Definition 10.1, we then have $\ell < \min(|S'_1|,|S'_2|)$, for every $j \in [1\,..\,\ell]$, $S'_1[j] = \sigma - 1 - S_1[j] = \sigma - 1 - S_2[j] = S'_2[j]$, and $S'_1[\ell+1] = \sigma - 1 - S_1[\ell+1] \prec \sigma - 1 - S_2[\ell+1] = S'_2[\ell+1]$. Thus, $S'_1 \prec S'_2$.

The proof of the opposite implication follows almost identically by noting that $\mathrm{InvString}_\sigma(S'_k) = S_k$ holds for $k \in \{1,2\}$. □

Remark 10.3. Note that the assumption that S_1 is not a prefix of S_2 is necessary in Lemma 10.2, since without it, it is possible that $S_1 \prec S_2$ and $\mathrm{InvString}_\sigma(S_1) \prec \mathrm{InvString}_\sigma(S_2)$.

Lemma 10.4. *Let $\sigma \geq 2$, $T \in [0\,..\,\sigma)^*$ and $P \in [0\,..\,\sigma)^*$. Then, it holds*

$$\mathrm{RangeEnd}(P,T) = |T| - \mathrm{RangeBeg}(P_{\mathrm{inv}}, T_{\mathrm{inv}}),$$

where $P_{\mathrm{inv}} = \mathrm{InvString}_\sigma(P)$ and $T_{\mathrm{inv}} = \mathrm{InvString}_\sigma(T)$ (see Definition 10.1).

Proof. Denote $n = |T|$, $\mathcal{S} = \{T[j\,..\,n] : j \in [1\,..\,n]\}$, and let

$$\begin{aligned}
\mathcal{A} &= \{T[j\,..\,n] : j \in [1\,..\,n] \text{ and } T[j\,..\,n] \prec P\},\\
\mathcal{B} &= \{T[j\,..\,n] : j \in \mathrm{Occ}(P,T)\},\\
\mathcal{C} &= \{T[j\,..\,n] : j \in [1\,..\,n] \setminus \mathrm{Occ}(P,T) \text{ and } T[j\,..\,n] \succ P\}.
\end{aligned}$$

By definition, $\mathrm{RangeBeg}(P,T) = |\mathcal{A}|$ and $\mathcal{S}$ is a disjoint union $\mathcal{S} = \mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$. Thus, by $\mathrm{RangeEnd}(P,T) = \mathrm{RangeBeg}(P,T) + |\mathrm{Occ}(P,T)|$, we obtain that $\mathrm{RangeEnd}(P,T) = |\mathcal{A}| + |\mathcal{B}| = n - |\mathcal{C}|$. Consequently, it suffices to show that $|\mathcal{C}| = \mathrm{RangeBeg}(P_{\mathrm{inv}}, T_{\mathrm{inv}})$.

Denote $\mathcal{C}_{\mathrm{inv}} = \{T_{\mathrm{inv}}[j\,..\,n] : j \in [1\,..\,n] \text{ and } T_{\mathrm{inv}}[j\,..\,n] \prec P_{\mathrm{inv}}\}$. By definition, $\mathrm{RangeBeg}(P_{\mathrm{inv}}, T_{\mathrm{inv}}) = |\mathcal{C}_{\mathrm{inv}}|$. Note that for every $j \in [1\,..\,n]$, $T_{\mathrm{inv}}[j\,..\,n] \prec P_{\mathrm{inv}}$ implies that $j \notin \mathrm{Occ}(P_{\mathrm{inv}}, T_{\mathrm{inv}})$. Moreover, $\mathrm{Occ}(P,T) = \mathrm{Occ}(P_{\mathrm{inv}}, T_{\mathrm{inv}})$. Thus, by Lemma 10.2,

$$\begin{aligned}
|\mathcal{C}| &= |\{T[j\,..\,n] : j \in [1\,..\,n] \setminus \mathrm{Occ}(P,T) \text{ and } T[j\,..\,n] \succ P\}| \\
&= |\{T[j\,..\,n] : j \in [1\,..\,n] \setminus \mathrm{Occ}(P,T) \text{ and } \mathrm{InvString}_\sigma(T[j\,..\,n]) \prec \mathrm{InvString}_\sigma(P)\}| \\
&= |\{T[j\,..\,n] : j \in [1\,..\,n] \setminus \mathrm{Occ}(P,T) \text{ and } T_{\mathrm{inv}}[j\,..\,n] \prec P_{\mathrm{inv}}\}| \\
&= |\{T_{\mathrm{inv}}[j\,..\,n] : j \in [1\,..\,n] \setminus \mathrm{Occ}(P,T) \text{ and } T_{\mathrm{inv}}[j\,..\,n] \prec P_{\mathrm{inv}}\}| \\
&= |\{T_{\mathrm{inv}}[j\,..\,n] : j \in [1\,..\,n] \setminus \mathrm{Occ}(P_{\mathrm{inv}}, T_{\mathrm{inv}}) \text{ and } T_{\mathrm{inv}}[j\,..\,n] \prec P_{\mathrm{inv}}\}| \\
&= |\{T_{\mathrm{inv}}[j\,..\,n] : j \in [1\,..\,n] : T_{\mathrm{inv}}[j\,..\,n] \prec P_{\mathrm{inv}}\}| \\
&= |\mathcal{C}_{\mathrm{inv}}| = \mathrm{RangeBeg}(P_{\mathrm{inv}}, T_{\mathrm{inv}}).
\end{aligned}$$

□

Proposition 10.5. *For every $u \geq 1$ we can in $\mathcal{O}(\sqrt{u}\log u)$ time construct a data structure of size $\mathcal{O}(\sqrt{u})$ that, given the packed representation of any string $S \in \{0,1\}^*$, returns the packed representation of* $\mathrm{InvString}_2(S)$ *(Definition 10.1) in $\mathcal{O}(1 + |S|/\log u)$ time.*

Proof. Let $\ell = \lceil (\log(u+1))/2 \rceil$. Let L_{inv} be a lookup table such that, for every $X \in \{0,1\}^{\leq \ell}$, L_{inv} maps X to the packed representation of $\mathrm{InvString}_2(X)$.

Components The data structure consists of a single component: the lookup table L_{inv}. When accessing L_{inv}, we map $X \in \{0,1\}^{\leq \ell}$ to $\mathrm{BinStrToInt}(X)$ (Definition 4.8). The packed representation of $X_{\mathrm{inv}} = \mathrm{InvString}_2(X)$ is also stored as $\mathrm{BinStrToInt}(X_{\mathrm{inv}})$. Thus, L_{inv} needs $\mathcal{O}(2^\ell) = \mathcal{O}(\sqrt{u})$ space.

Implementation of queries To answer queries, observe that using L_{inv}, given the packed representation of any $S \in \{0,1\}^*$, we can compute $\mathrm{InvString}_2(S)$ in $\mathcal{O}(1 + |S|/\ell) = \mathcal{O}(1 + |S|/\log u)$ time.

Construction algorithm The computation of a single entry of the L_{inv} table takes $\mathcal{O}(\ell)$ time. Thus, construction of L_{inv} takes $\mathcal{O}(2^\ell \cdot \ell) = \mathcal{O}(\sqrt{u}\log u)$ time in total. □

10.3 Summary

Theorem 10.6. *Consider a data structure answering pattern ranking queries (see Section 10.1) that, for any text $T \in \{0,1\}^n$, achieves the following complexities (where in the preprocessing we assume that we are given as input the packed representation of T, and at query time we are given a packed representation of $P \in \{0,1\}^{\leq k}$):*

- *space usage $S(n)$,*
- *preprocessing time $P_t(n)$,*
- *preprocessing space $P_s(n)$,*
- *query time $Q(n,k)$.*

Then, for every $T \in \{0,1\}^n$, we can in $\mathcal{O}(n/\log n + P_t(n))$ time and using $\mathcal{O}(n/\log n + P_s(n))$ working space construct a data structure of size $\mathcal{O}(n/\log n + S(n))$ that, given the packed representation of any $P \in \{0,1\}^{\leq k}$, returns $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ *in $\mathcal{O}(k/\log n + Q(n,k))$ time.*

Proof. Denote $T_{\mathrm{inv}} = \mathrm{InvString}_2(T)$ (Definition 10.1).

Components The data structure consists of the following components:

1. The data structure from Proposition 10.5 for $u = n$. It needs $\mathcal{O}(\sqrt{n})$ space.
2. The data structure from the claim for T. It needs $\mathcal{O}(S(n))$ space.
3. The data structure from the claim for T_{inv}. Since $|T_{\mathrm{inv}}| = |T| = n$, it also needs $\mathcal{O}(S(n))$ space.

In total, the structure needs $\mathcal{O}(\sqrt{n} + S(n)) = \mathcal{O}(n/\log n + S(n))$ space.

Implementation of queries The queries are answered as follows. Let $P \in \{0,1\}^{\leq k}$, and assume that we are given the packed representation of P. To compute $(\text{RangeBeg}(P,T), \text{RangeEnd}(P,T))$ we proceed as follows:

1. Using the structure from the claim, compute $b = \text{RangeBeg}(P,T)$ in $\mathcal{O}(Q(n,k))$ time.
2. Using Proposition 10.5, in $\mathcal{O}(1+|P|/\log n) = \mathcal{O}(1+k/\log n)$ time we compute the packed representation of $P_{\text{inv}} = \text{InvString}_2(P)$.
3. Using the structure from the claim (built for T_{inv}), compute $e = n - \text{RangeBeg}(P_{\text{inv}}, T_{\text{inv}})$ in $\mathcal{O}(Q(n,k))$ time. By Lemma 10.4, $e = \text{RangeEnd}(P,T)$. Thus, we return (b,e) as the answer.

In total, the query takes $\mathcal{O}(k/\log n + Q(n,k))$ time.

Construction algorithm The components of the structure are constructed as follows:

1. In $\mathcal{O}(\sqrt{n}\log n) = \mathcal{O}(n/\log n)$ time we apply Proposition 10.5.
2. The structure from the claim is built for T in $\mathcal{O}(P_t(n))$ time and using $\mathcal{O}(P_s(n))$ space.
3. In analogous time and space we construct the structure from the claim for T_{inv}.

In total, we spend $\mathcal{O}(n/\log n + P_t(n))$ time and use $\mathcal{O}(n/\log n + P_s(n))$ working space. □

Theorem 10.7. *The problems*

- Indexing for Pattern Ranking Queries *(Section 6.1) and*
- indexing for Pattern SA-Interval Queries *(Section 10.1)*

are equivalent in the following sense. If, given the packed representation of a nonempty text $T \in \{0,1\}^n$, we can in $P_t(n)$ time and using $P_s(n)$ working space construct a data structure of size $S(n)$ that, given a packed representation of $P \in \{0,1\}^{\leq k}$, answers queries to one of the problems in $Q(n,k)$ time, then in $\mathcal{O}(n/\log n + P_t(n))$ time and using $\mathcal{O}(n/\log n + P_s(n))$ working space we can construct a data structure of size $\mathcal{O}(n/\log n + S(n))$ that answers queries for the other problem in $\mathcal{O}(k/\log n + Q(n,k))$ time.

Proof. The reduction in one direction follows by definition. The other direction follows by Theorem 10.6. □

References

[ABG+14] Diego Arroyuelo, Carolina Bonacic, Veronica Gil-Costa, Mauricio Marín, and Gonzalo Navarro. Distributed text search using suffix arrays. *Parallel Computing*, 40(9):471–495, 2014. doi:10.1016/J.PARCO.2014.06.007.

[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.

[AKO04] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004. doi:10.1016/S1570-8667(03)00065-0.

[BCFR18] Nieves R. Brisaboa, Diego Caro, Antonio Fariña, and M. Andrea Rodríguez. Using compressed suffix-arrays for a compact representation of temporal-graphs. *Information Sciences*, 465:459–483, 2018. doi:10.1016/J.INS.2018.07.023.

[Bel14] Djamal Belazzougui. Linear time construction of compressed text indices in compact space. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing, STOC 2014*, pages 148–193. ACM, 2014. doi:10.1145/2591796.2591885.

[BGKS15] Maxim A. Babenko, Paweł Gawrychowski, Tomasz Kociumaka, and Tatiana Starikovskaya. Wavelet trees meet suffix trees. In Piotr Indyk, editor, *26th Annual ACM-SIAM Symposium on*

Discrete Algorithms, SODA 2015, pages 572–591. SIAM, 2015. doi:10.1137/1.9781611973730.39.

[BHMP14] Carl Barton, Alice Héliou, Laurent Mouchard, and Solon P. Pissis. Linear-time computation of minimal absent words using suffix array. *BMC Bioinformatics*, 15:388, 2014. doi:10.1186/S12859-014-0388-9.

[BN14] Djamal Belazzougui and Gonzalo Navarro. Alphabet-independent compressed text indexing. *ACM Transactions on Algorithms*, 10(4):23:1–23:19, 2014. doi:10.1145/2635816.

[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: https://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern information retrieval*. ACM Press New York, 1999.

[CGN21] Dustin Cobas, Travis Gagie, and Gonzalo Navarro. A fast and small subsampled r-index. In Pawel Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching, CPM 2021*, volume 191 of *LIPIcs*, pages 13:1–13:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CPM.2021.13.

[CHL07] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, Cambridge, UK, 2007. doi:10.1017/cbo9780511546853.

[CI08] Maxime Crochemore and Lucian Ilie. Computing longest previous factor in linear time and applications. *Information Processing Letters*, 106(2):75–80, 2008. doi:10.1016/J.IPL.2007.10.006.

[CILP20] Panagiotis Charalampopoulos, Costas S. Iliopoulos, Chang Liu, and Solon P. Pissis. Property suffix array with applications in indexing weighted sequences. *ACM Journal of Experimental Algorithmics*, 25:1–16, 2020. doi:10.1145/3385898.

[CIS08] Maxime Crochemore, Lucian Ilie, and William F. Smyth. A simple algorithm for computing the Lempel Ziv factorization. In *2008 Data Compression Conference, DCC 2008*, pages 482–488. IEEE Computer Society, 2008. doi:10.1109/DCC.2008.36.

[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*, volume 204 of *LIPIcs*, pages 30:1–30:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.30.

[Cla98] David R. Clark. *Compact Pat Trees*. PhD thesis, University of Waterloo, 1998.

[CPS07] Gang Chen, Simon J. Puglisi, and William F. Smyth. Fast and practical algorithms for computing all the runs in a string. In Bin Ma and Kaizhong Zhang, editors, *18th Annual Symposium on Combinatorial Pattern Matching, CPM 2007*, volume 4580 of *LNCS*, pages 307–315. Springer, 2007. doi:10.1007/978-3-540-73437-6_31.

[CPZ20] Manuel Cáceres, Simon J. Puglisi, and Bella Zhukova. Fast indexes for gapped pattern matching. In Alexander Chatzigeorgiou, Riccardo Dondi, Herodotos Herodotou, Christos A. Kapoutsis, Yannis Manolopoulos, George A. Papadopoulos, and Florian Sikora, editors, *46th International Conference on Current Trends in Theory and Practice of Informatics, SOFSEM 2020*, volume 12011 of *LNCS*, pages 493–504. Springer, 2020. doi:10.1007/978-3-030-38919-2_40.

[CT10] Maxime Crochemore and German Tischler. The gapped suffix array: A new index structure for fast approximate matching. In Edgar Chávez and Stefano Lonardi, editors, *17th International Symposium on String Processing and Information Retrieval, SPIRE 2010*, volume 6393 of *LNCS*, pages 359–364. Springer, 2010. doi:10.1007/978-3-642-16321-0_37.

[DKK+04] Jean-Pierre Duval, Roman Kolpakov, Gregory Kucherov, Thierry Lecroq, and Arnaud Lefebvre. Linear-time computation of local periods. *Theoretical Computer Science*, 326(1-3):229–240, 2004. doi:10.1016/J.TCS.2004.06.024.

[FA19] Patrick Flick and Srinivas Aluru. Distributed enhanced suffix arrays: efficient algorithms for construction and querying. In Michela Taufer, Pavan Balaji, and Antonio J. Peña, editors, *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019*, pages 72:1–72:17. ACM, 2019. doi:10.1145/3295500.3356211.

[FGGV06] Luca Foschini, Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. When indexing equals compression: Experiments with compressing suffix arrays and applications. *ACM Transactions on Algorithms*, 2(4):611–639, 2006. doi:10.1145/1198513.1198521.

[FGM12] Paolo Ferragina, Travis Gagie, and Giovanni Manzini. Lightweight data indexing and compression in external memory. *Algorithmica*, 63(3):707–730, 2012. doi:10.1007/S00453-011-9535-0.

[FH11] Johannes Fischer and Volker Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM Journal on Computing*, 40(2):465–492, 2011. doi:10.1137/090779759.

[FLMM05] Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Structuring labeled trees for optimal succinctness, and beyond. In *46th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2005*, pages 184–196. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.69.

[FM00] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000*, pages 390–398. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892127.

[FM05] Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *Journal of the ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.

[FMN09] Johannes Fischer, Veli Mäkinen, and Gonzalo Navarro. Faster entropy-bounded compressed suffix trees. *Theoretical Computer Science*, 410(51):5354–5364, 2009. doi:10.1016/J.TCS.2009.09.012.

[FNI+19] Noriki Fujisato, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Direct linear time construction of parameterized suffix and LCP arrays for constant alphabets. In Nieves R. Brisaboa and Simon J. Puglisi, editors, *26th International Symposium on String Processing and Information Retrieval, SPIRE 2019*, volume 11811 of *LNCS*, pages 382–391. Springer, 2019. doi:10.1007/978-3-030-32686-9_27.

[GB13] Keisuke Goto and Hideo Bannai. Simpler and faster Lempel Ziv factorization. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2013 Data Compression Conference, DCC 2013*, pages 133–142. IEEE, 2013. doi:10.1109/DCC.2013.21.

[GGV03] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *14th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2003*, pages 841–850. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644250.

[GHN20] Younan Gao, Meng He, and Yakov Nekrich. Fast preprocessing for optimal orthogonal range reporting and range successor with applications to text indexing. In *28th Annual European Symposium on Algorithms, ESA 2020*, volume 173 of *LIPIcs*, pages 54:1–54:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ESA.2020.54.

[GK12] Giorgio Gonnella and Stefan Kurtz. Readjoiner: a fast and memory efficient string graph-based sequence assembler. *BMC Bioinformatics*, 13:82, 2012. doi:10.1186/1471-2105-13-82.

[GK17] Paweł Gawrychowski and Tomasz Kociumaka. Sparse suffix tree construction in optimal time and space. In Philip N. Klein, editor, *28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 425–439. SIAM, 2017. doi:10.1137/1.9781611974782.27.

[GMC+14] Simon Gog, Alistair Moffat, J. Shane Culpepper, Andrew Turpin, and Anthony Wirth. Large-scale pattern search using reduced-space on-disk suffix arrays. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1918–1931, 2014. doi:10.1109/TKDE.2013.129.

[GNF14] Rodrigo González, Gonzalo Navarro, and Héctor Ferrada. Locally compressed suffix arrays. *ACM Journal of Experimental Algorithmics*, 19(1), 2014. doi:10.1145/2594408.

[GNP15] Simon Gog, Gonzalo Navarro, and Matthias Petri. Improved and extended locating functionality on compressed suffix arrays. *Journal of Discrete Algorithms*, 32:53–63, 2015. doi:10.1016/J.JDA.2015.01.006.

[GNP20] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *Journal of the ACM*, 67(1):2:1–2:54, 2020. doi:10.1145/3375890.

[Gro11] Roberto Grossi. A quick tour on suffix arrays and compressed suffix arrays. *Theoretical Computer Science*, 412(27):2964–2973, 2011. doi:10.1016/J.TCS.2010.12.036.

[GS04] Dan Gusfield and Jens Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *Journal of Computer and System Sciences*, 69(4):525–546, 2004. doi:10.1016/j.jcss.2004.03.004.

[Gus97] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, UK, 1997. doi:10.1017/cbo9780511574931.

[GV00] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract). In F. Frances Yao and Eugene M. Luks, editors, *32nd Annual ACM Symposium on Theory of Computing, STOC 2000*, pages 397–406. ACM, 2000. doi:10.1145/335305.335351.

[GV05] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 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.

[HL11] Scott Hazelhurst and Zsuzsanna Lipták. Kaboom! a new suffix array based algorithm for clustering expression data. *Bioinformatics*, 27(24):3348–3355, 2011. doi:10.1093/bioinformatics/btr560.

[HLSS03] Wing-Kai Hon, Tak Wah Lam, Kunihiko Sadakane, and Wing-Kin Sung. Constructing compressed suffix arrays with large alphabets. In Toshihide Ibaraki, Naoki Katoh, and Hirotaka Ono, editors, *14th International Symposium on Algorithms and Computation, ISAAC 2003*, volume 2906 of *LNCS*, pages 240–249. Springer, 2003. doi:10.1007/978-3-540-24587-2_26.

[HLST11] Wing-Kai Hon, Chen-Hua Lu, Rahul Shah, and Sharma V. Thankachan. Succinct indexes for circular patterns. In Takao Asano, Shin-Ichi Nakano, Yoshio Okamoto, and Osamu Watanabe, editors, *22nd International Symposium on Algorithms and Computation, ISAAC 2011*, volume 7074 of *LNCS*, pages 673–682. Springer, 2011. doi:10.1007/978-3-642-25591-5_69.

[HSS09] Wing-Kai Hon, Kunihiko Sadakane, and Wing-Kin Sung. Breaking a time-and-space barrier in constructing full-text indices. *SIAM Journal on Computing*, 38(6):2162–2178, 2009. doi:10.1137/070685373.

[IFI11] Lucian Ilie, Farideh Fazayeli, and Silvana Ilie. Hitec: accurate error correction in high-throughput sequencing data. *Bioinformatics*, 27(3):295–302, 2011. doi:10.1093/bioinformatics/btq653.

[IS11] Lucian Ilie and William F. Smyth. Minimum unique substrings and maximum repeats. *Fundamenta Informaticae*, 110(1-4):183–195, 2011. doi:10.3233/FI-2011-536.

[Jac89] Guy Jacobson. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science, FOCS 1989*, pages 549–554. IEEE Computer Society, 1989. doi:10.1109/SFCS.1989.63533.

[Kem19] Dominik Kempa. Optimal construction of compressed indexes for highly repetitive texts. In Timothy M. Chan, editor, *30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1344–1357. SIAM, 2019. doi:10.1137/1.9781611975482.82.

[KK99] Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science, FOCS 1999*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.

[KK03] Roman M. Kolpakov and Gregory Kucherov. Finding approximate repetitions under Hamming distance. *Theoretical Computer Science*, 303(1):135–156, 2003. doi:10.1016/S0304-3975(02)00448-6.

[KK19] Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: Sublinear-time BWT construction and optimal LCE data structure. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 756–767. ACM, 2019. doi:10.1145/3313276.3316368.

[KK22] Dominik Kempa and Tomasz Kociumaka. Dynamic suffix array with polylogarithmic queries and updates. In Stefano Leonardi and Anupam Gupta, editors, *54th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2022*, pages 1657–1670. ACM, 2022. doi:10.1145/3519935.3520061.

[KK23a] Dominik Kempa and Tomasz Kociumaka. Breaking the $O(n)$-barrier in the construction of compressed suffix arrays and suffix trees. In Nikhil Bansal and Viswanath Nagarajan, editors, *34th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 5122–5202. SIAM, 2023. doi:10.1137/1.9781611977554.ch187.

[KK23b] Dominik Kempa and Tomasz Kociumaka. Collapsing the hierarchy of compressed data structures: Suffix arrays in optimal compressed space. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, pages 1877–1886. IEEE, 2023. doi:10.1109/FOCS57990.2023.00114.

[KK24] Dominik Kempa and Tomasz Kociumaka. Lempel-Ziv (LZ77) factorization in sublinear time. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024*, pages 2045–2055. IEEE, 2024. Full version: arXiv:2409.12146. doi:10.1109/FOCS61266.2024.00122.

[KK25] Dominik Kempa and Tomasz Kociumaka. On the hardness hierarchy for the $O(n\sqrt{\log n})$ complexity in the word RAM. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025*, pages 290–300. ACM, 2025. doi:10.1145/3717823.3718291.

[KK26] Dominik Kempa and Tomasz Kociumaka. Tight lower bounds for central string queries in compressed space. In *Proceedings of the 2026 ACM-SIAM Symposium on Discrete Algorithms, SODA 2026*, 2026. To appear.

[KKP03] Dong Kyue Kim, Yoo Ah Kim, and Kunsoo Park. Generalizations of suffix arrays to multi-dimensional matrices. *Theoretical Computer Science*, 302(1-3):223–238, 2003. doi:10.1016/S0304-3975(02)00828-9.

[KKP14] Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi. String range matching. In Alexander S. Kulikov, Sergei O. Kuznetsov, and Pavel A. Pevzner, editors, *25th Annual Symposium on Combinatorial Pattern Matching, CPM 2014*, volume 8486 of *LNCS*, pages 232–241. Springer, 2014. doi:10.1007/978-3-319-07566-2_24.

[KKP15] Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi. Parallel external memory suffix sorting. In Ferdinando Cicalese, Ely Porat, and Ugo Vaccaro, editors, *26th Annual Symposium on Combinatorial Pattern Matching, CPM 2015*, volume 9133 of *Lecture Notes in Computer Science*, pages 329–342. Springer, 2015. doi:10.1007/978-3-319-19929-0_28.

[KKR+20] Tomasz Kociumaka, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. A linear-time algorithm for seeds computation. *ACM Transactions on Algorithms*, 16(2):27:1–27:23, 2020. doi:10.1145/3386369.

[KNSW08] Stefan Kurtz, Apurva Narechania, Joshua C Stein, and Doreen Ware. A new method to compute k-mer frequencies and its application to annotate large repetitive plant genomes. *BMC Genomics*, 9(1):1–18, 2008. doi:10.1186/1471-2164-9-517.

[KP13] Dominik Kempa and Simon J. Puglisi. Lempel-Ziv factorization: Simple, fast, practical. In Peter Sanders and Norbert Zeh, editors, *15th Meeting on Algorithm Engineering and Experiments, ALENEX 2013*, pages 103–112. SIAM, 2013. doi:10.1137/1.9781611972931.9.

[KPD+04] Stefan Kurtz, Adam Phillippy, Arthur L. Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L. Salzberg. Versatile and open software for comparing large genomes. *Genome Biology*, 5:1–9, 2004. doi:10.1186/gb-2004-5-2-r12.

[LD09] Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009. doi:10.1093/bioinformatics/btp324.

[LS12] Ben Langmead and Steven L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357, 2012. doi:10.1038/nmeth.1923.

[LSSY02] Tak Wah Lam, Kunihiko Sadakane, Wing-Kin Sung, and Siu-Ming Yiu. A space and time efficient algorithm for constructing compressed suffix arrays. In Oscar H. Ibarra and Louxin Zhang, editors, *8th Annual International Conference on Computing and Combinatorics, COCOON 2002*, volume 2387 of *LNCS*, pages 401–410. Springer, 2002. doi:10.1007/3-540-45655-4_43.

[LV88] Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *Journal of Computer and System Sciences*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.

[Mai89] Michael G Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25(1-2):145–153, 1989. doi:10.1016/0166-218X(89)90051-6.

[Mäk03] Veli Mäkinen. Compact suffix array: A space-efficient full-text index. *Fundamenta Informaticae*, 56(1-2):191–210, 2003.

[MB91] Udi Manber and Ricardo A. Baeza-Yates. An algorithm for string matching with a sequence of don't cares. *Information Processing Letters*, 37(3):133–136, 1991. doi:10.1016/0020-0190(91)90032-D.

[MM90] Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. In David S. Johnson, editor, *1st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1990*, pages 319–327. SIAM, 1990. URL: http://dl.acm.org/citation.cfm?id=320176.320218.

[MM93] Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993. doi:10.1137/0222058.

[MN05] Veli Mäkinen and Gonzalo Navarro. Succinct suffix arrays based on run-length encoding. *Nordic Journal of Computing*, 12(1):40–66, 2005. doi:10.1007/11496656_5.

[MNN17] J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Space-efficient construction of compressed indexes in deterministic linear time. In Philip N. Klein, editor, *28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 408–424. SIAM, 2017. doi:10.1137/1.9781611974782.26.

[MNN20] J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Text indexing and searching in sublinear time. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020*, volume 161 of *LIPIcs*, pages 24:1–24:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.CPM.2020.24.

[MNV16] J. Ian Munro, Yakov Nekrich, and Jeffrey Scott Vitter. Fast construction of wavelet trees. *Theoretical Computer Science*, 638:91–97, 2016. doi:10.1016/j.tcs.2015.11.011.

[MSST20] Kotaro Matsuda, Kunihiko Sadakane, Tatiana Starikovskaya, and Masakazu Tateshita. Compressed orthogonal search on suffix arrays with applications to range LCP. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020*, volume 161 of *LIPIcs*, pages 23:1–23:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.CPM.2020.23.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001. doi:10.1145/375360.375365.

[NM07] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):2:1–2:61, 2007. doi:10.1145/1216370.1216372.

[NPL+13] Joong Chae Na, Heejin Park, Sunho Lee, Minsung Hong, Thierry Lecroq, Laurent Mouchard, and Kunsoo Park. Suffix array of alignment: A practical index for similar data. In Oren Kurland, Moshe Lewenstein, and Ely Porat, editors, *20th International Symposium on String Processing and Information Retrieval, SPIRE 2013*, volume 8214 of *LNCS*, pages 243–254. Springer, 2013. doi:10.1007/978-3-319-02432-5_27.

[OG10] Enno Ohlebusch and Simon Gog. Efficient algorithms for the all-pairs suffix-prefix problem and the all-pairs substring-prefix problem. *Information Processing Letters*, 110(3):123–128, 2010. doi:10.1016/J.IPL.2009.10.015.

[OG11] Enno Ohlebusch and Simon Gog. Lempel-Ziv factorization revisited. In Raffaele Giancarlo and Giovanni Manzini, editors, *22nd Annual Symposium on Combinatorial Pattern Matching, CPM 2011*, volume 6661 of *LNCS*, pages 15–26. Springer, 2011. doi:10.1007/978-3-642-21458-5_4.

[Ohl13] Enno Ohlebusch. *Bioinformatics algorithms: Sequence analysis, genome rearrangements, and phylogenetic reconstruction*. Oldenbusch Verlag, Ulm, Germany, 2013.

[OS09] Daisuke Okanohara and Kunihiko Sadakane. A linear-time Burrows–Wheeler transform using induced sorting. In Jussi Karlgren, Jorma Tarhio, and Heikki Hyyrö, editors, *16th International Symposium on String Processing and Information Retrieval, SPIRE 2009*, volume 5721 of *LNCS*, pages 90–101. Springer, 2009. doi:10.1007/978-3-642-03784-9_9.

[Pre18] Nicola Prezza. In-place sparse suffix sorting. In Artur Czumaj, editor, *29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1496–1508. SIAM, 2018. doi:10.1137/1.9781611975031.98.

[RNO11] Luís M. S. Russo, Gonzalo Navarro, and Arlindo L. Oliveira. Fully compressed suffix trees. *ACM Transactions on Algorithms*, 7(4):53:1–53:34, 2011. doi:10.1145/2000807.2000821.

[Sad02] Kunihiko Sadakane. Succinct representations of LCP information and improvements in the compressed suffix arrays. In *13th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2002*, pages 225–232. ACM/SIAM, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545410.

[Shi96] Fei Shi. Suffix arrays for multiple strings: a method for on-line multiple string searches. In Joxan Jaffar and Roland H. C. Yap, editors, *2nd Asian Conference on Computing Science, ASIAN 1996*, volume 1179 of *LNCS*, pages 11–22. Springer, 1996. doi:10.1007/bfb0027775.

[Wei73] Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, SWAT (FOCS) 1973*, pages 1–11. IEEE Computer Society, 1973. doi:10.1109/SWAT.1973.13.