Tight Lower Bounds for Central String Queries in Compressed Space

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

In this work, we study limits of compressed data structures, i.e., data structures that support various queries on the input text $T \in \Sigma^n$ in space proportional to the size of T in compressed form. On the *upper bound* side, currently nearly all fundamental queries can be efficiently supported in $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space (where $\delta(T)$ is the *substring complexity* – a strong measure of compressibility that lower-bounds the optimal achievable space to represent the text [Kociumaka, Navarro, Prezza, IEEE Trans. Inf. Theory 2023]); this includes queries like random access, longest common extension, suffix array, longest common prefix array, and many others. In contrast, *lower bounds* for compressed data structures remained elusive, and currently they are known only for the basic random access problem. This work addresses this important gap and develops tight lower bounds for nearly all other fundamental queries:

- First, we prove that several core string-processing queries, including suffix array (SA) queries, inverse suffix array (SA^{-1}) queries, longest common prefix (LCP) array queries, and longest common extension (LCE) queries, all require $\Omega\big(\frac{\log n}{\log\log n}\big)$ time within $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space. All our lower bounds are asymptotically tight, i.e., we match the performance of known upper bounds.
- We then extend our framework to include other common queries that are currently supported in $\mathcal{O}(\log\log n)$ time and $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space. This includes Burrows–Wheeler Transform (BWT) queries, permuted longest common prefix array (PLCP) queries, Last-to-First (LF) queries, inverse LF (LF^{-1}) queries, lexicographical predecessor (Φ) queries, and inverse lexicographical predecessor (Φ^{-1}) queries, and again we prove that, within this space, they all *require* $\Omega(\log\log n)$ time, obtaining again a set of asymptotically tight lower bounds.

All our lower bounds hold even for texts over a binary alphabet. Our results establish a clean dichotomy: the optimal time complexity to support central string queries in compressed space is either $\Theta(\log n/\log\log n)$ or $\Theta(\log\log n)$. This completes the theoretical foundation of compressed indexing, closing a crucial gap between upper and lower bounds, and providing a clear target for future data structures: seeking either the optimal time in the smallest space, or fastest time in the optimal space (both of which are now known) for central string queries.

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

Query	Time Complexity	Lower Bound	Upper Bound
T	$\Theta(\frac{\log n}{\log\log n})$	[VY13]	[BCG+21]
LCP		Section 4.1.2	[GNP20][1]
SA		Section 4.2.2	[GNP20][1]
SA^{-1}		Section 4.3.2	[GNP20][1]
LCE		Section 4.4.2	[GNP20][2]
BWT	$\Theta(\log\log n)$	Section 5.1.2	[GNP20]
PLCP		Section 5.2.2	[GNP20]
LF		Section 5.3.2	[GNP20]
LF^{-1}		Section 5.4.2	Section 5.4.3
Φ		Section 5.5.2	[GNP20]
Φ^{-1}		Section 5.6.2	[GNP20]

Table 1: Query time lower and upper bounds for fundamental string queries in $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space for a text $T \in \Sigma^n$. All lower bounds hold even when $\Sigma = \{\mathtt{0}, \mathtt{1}\}$.

[1]These results are obtained through a cosmetic modification to the structures by Gagie, Navarro, and Prezza [GNP20, Theorems 5.4, 5.7, and 5.8] (originally using $\mathcal{O}(r(T)\log n)$ space and supporting the queries in $\mathcal{O}(\log n)$ time), resulting in $\mathcal{O}(r(T)\log^{1+\epsilon} n)$ space usage and $\mathcal{O}(\frac{\log n}{\log\log n})$ query time; see Theorem 2.28 and Remark 2.29. Combining with the upper bound $r(T) = \mathcal{O}(\delta(T)\log^2 n)$ [KK20] (see Theorem 2.35 and Remark 2.36), this results in suffix array, inverse suffix array, and LCP array queries in $\mathcal{O}(\delta(T)\log^{4+\epsilon} n) = \mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and $\mathcal{O}(\frac{\log n}{\log\log n})$ time.

[2]A data structure using $\mathcal{O}(z(T)\log n)$ (resp. $\mathcal{O}(\delta(T)\log n)$) space and answering LCE queries in $\mathcal{O}(\log n)$ time was presented in [I17] (resp. [KK23b]). A similar result, achieved via more general LCP RMQ queries (see Section B.1), was presented in [GNP20], resulting in LCE queries using $\mathcal{O}(r(T)\log n)$ space and $\mathcal{O}(\log n)$ time. In Section B, we explain in detail the modifications to the technique of Gagie, Navarro, and Prezza [GNP20] needed to achieve LCP RMQ queries in $\mathcal{O}(r(T)\log^{2+\epsilon} n)$ space and $\mathcal{O}(\frac{\log n}{\log\log n})$ time. By applying the upper bound $r(T) = \mathcal{O}(\delta(T)\log^2 n)$ [KK20] (see Theorem 2.35 and Remark 2.36), this results in LCE queries using $\mathcal{O}(\delta(T)\log^{4+\epsilon} n) = \mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and $\mathcal{O}(\frac{\log n}{\log\log n})$ time; see Section 4.4.3.

1 Introduction

Compressed indexing [Nav21b, Nav21a] is a recent paradigm in the design of data structures that combines elements of data compression, information theory, and combinatorial pattern matching to build data structures whose space usage is proportional to the size of the input text (string, sequence) in compressed form. For example, given a text $T \in \Sigma^n$ and a compression algorithm $C : \Sigma^* \to \hat{\Sigma}^*$, a compressed index supporting *random access* to T will typically use $\mathcal{O}(|C(T)| \cdot \log^{\mathcal{O}(1)} n)$ space and, given any position $i \in [1\,..\,n]$, return the symbol $T[i]$ in $\mathcal{O}(\log^{\mathcal{O}(1)} n)$ time.

In the past two decades, interest in compressed computation has spiked dramatically, as datasets in many applications have grown to such unwieldy sizes that they essentially cannot be handled otherwise. This includes source code storage systems (like GitHub), versioned databases (like Wikipedia), and computational biology, where some datasets already exceed petabytes [SLF+15], with estimates predicting growth to exabytes in the near future [Nat22]. These datasets are extremely repetitive/redundant, with some starting at 99.9% redundancy [PHR00] and increasing with dataset size [GNP20]. Crucially, *misrepresenting* any portion of this data is often infeasible – the presence of a disease could hinge on a single DNA mutation, and a bug might result from a single typo.

On the *upper bound* front, compressed indexing supports a rich repertoire of queries:

- Compressed indexes supporting random access were among the first [Ryt03, CLL+05]. They achieve $\mathcal{O}(\delta(T) \log^{\mathcal{O}(1)} n)$ space[3] and query time $\mathcal{O}(\log n)$. Subsequent works reduced space by a polylogarithmic factor [GJL21, BLR+15, KS22, KNP23] or query time to $\mathcal{O}(\frac{\log n}{\log\log n})$ [BCPT15, BCG+21, GJL21]. These indexes are typically used to simply replace the text.
- Compressed indexes also support more complex longest common extension (LCE) queries (see Section 2) using $\mathcal{O}(\delta(T) \log^{\mathcal{O}(1)} n)$ space and $\mathcal{O}(\log n)$ time [I17, GNP20, KK23b]. LCE queries are central in many applications, including approximate pattern matching [LV88, KNW24], BWT compression [KK20], and internal pattern matching [KRRW24].
- Compressed indexes supporting suffix array (SA) queries (which return $\mathrm{SA}_T[i]$, given any $i \in [1\,..\,n]$), inverse SA queries, and LCP array queries (Definitions 2.2, 2.5, and 2.11) are among the most powerful [GNP20, KK23b]. They use $\mathcal{O}(\delta(T) \log^{\mathcal{O}(1)} n)$ space, with query times ranging from $\mathcal{O}(\log n)$ [GNP20] to $\mathcal{O}(\log^{\mathcal{O}(1)} n)$ [KK23b]. SA/LCP queries solve myriad problems, including longest common substring [Gus97, CKPR21], LZ77 factorization [CIS08, OG11, KP13, GB13, KK24, PP18], maximal exact matches (MEMs) [Gus97], maximal unique matches (MUMs) [KPD+04], shortest unique substring [IS11], minimal absent word [BHMP14], sequence alignment [LS12, LD09], and many others [Gus97, ABM08].
- Compressed indexes also support faster algorithms for certain query types. Lexicographical predecessor Φ (Definition 2.6), LF mapping (Definition 2.8), BWT (Definition 2.26), and PLCP queries (Definition 2.12) can all be supported in $\mathcal{O}(\log\log n)$ time and $\mathcal{O}(\delta(T) \log^{\mathcal{O}(1)} n)$ space [GNP20, NT21].[4] Their applications include pattern listing [GNP20], suffix tree queries [Sad02, GNP20], and pattern counting [FM00, GV00, GNP20].

We refer to [Nav21b, Nav21a] for a further overview of *upper bound* results in compressed indexing. At this point, it is natural to ask: what can we say about compressed indexing on the *lower bound* front? The two fundamental questions are: (1) What is the *smallest achievable index space* that still supports efficient queries (e.g., in $\mathcal{O}(\log^{\mathcal{O}(1)} n)$ time)? Equally important is the complementary question: (2) What is the *best achievable query time* within near-optimal space (say, off by at most a polylogarithmic factor from the best possible)?

The first question has been largely resolved in recent years. Each of the indexes mentioned above achieves a specific space bound. For example, the indexes in [GNP20, NT21] use $\mathcal{O}(r(T))$ or $\mathcal{O}(r(T) \log n)$ space (where $r(T)$ is the size of the run-length Burrows–Wheeler transform [BW94] of T), while others use $\mathcal{O}(\delta(T) \log n)$ space [KNP23, KK23b] (where $\delta(T)$ is the substring complexity of T [KNP23]), $\mathcal{O}(z(T) \log n)$ space [Ryt03, CLL+05, GJL21, BCG+21, I17] (where $z(T)$ is the LZ77 size [ZL77]), or $\mathcal{O}(z_e(T))$ space [KS22]

[3] The value $\delta(T)$ is a fundamental measure of repetitiveness [KNP23], closely related to other core measures (such as the sizes of LZ77 factorization and run-length BWT compression). We discuss them all in more detail shortly.

[4] [NT21] can support LF and inverse lexicographical predecessor queries (see Sections 5.3.1 and 5.6.1) faster than $\mathcal{O}(\log\log n)$ if the query algorithm is given additional information (in addition to the index i) as input. This variant is useful since such a scenario occurs in some applications (e.g., pattern matching). The most general LF and inverse lexicographical predecessor queries (which are given only the index i) are supported in $\mathcal{O}(\log\log n)$ time by [GNP20].

(where $z_e(T)$ is the LZ-End parsing size [KN10]). However, in a series of works [Ryt03, CLL+05, KP18, GNP18, KK20, KS22, KNP23], *all* of these (and several other) repetitiveness measures and compression sizes have been shown to be within $\log^{\mathcal{O}(1)} n$ factors of each other in the worst case. Moreover, $\mathcal{O}(\delta(T)\log\frac{n\log\sigma}{\delta(T)\log n}) = \mathcal{O}(\delta(T)\log n)$[5] has been proven to be the optimal space [KNP23] and subsequently shown to be attainable for a range of queries (even for the most powerful queries like SA, inverse SA, LCP, and LCE queries) [KK23b],[6] without slowing down query times beyond $\log^{\mathcal{O}(1)} n$. For these reasons, it is reasonable to simplify space considerations and assume that by *compressed space* we mean space $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$.

The second question – regarding the limits of *query time* – has seen significantly less progress. To date, the only query among those mentioned above whose complexity is well understood is the most basic *random access* query: Verbin and Yu [VY13] demonstrated that in $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space, random access queries require $\Omega(\frac{\log n}{\log\log n})$ time, which matches the complexity of the best upper bounds. Currently, no lower bounds are known for any other fundamental queries (SA, SA^{-1}, LCP, LCE, BWT, PLCP, LF, LF^{-1}, Φ, Φ^{-1}). Given their central role and wealth of applications, we thus ask:

What are the lower bounds for fundamental string queries in compressed space?

Our Results We develop a series of new lower bounds that establish a clean dichotomy of fundamental string queries in $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space into two categories: those with time complexity $\Theta(\frac{\log n}{\log\log n})$ and those with time $\Theta(\log\log n)$ (cf. Table 1). In more detail, our results are as follows:

First, in Section 4.1, 4.2, 4.3, and 4.4, we prove that core string processing queries, including suffix array (SA), inverse suffix array (SA^{-1}), longest common prefix (LCP) array, and longest common extension (LCE) queries, require $\Omega(\frac{\log n}{\log\log n})$ time in $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space. These lower bounds are tight up to a constant factor, matching the performance of the r-index of Gagie, Navarro, and Prezza [GNP20], which supports these queries in $\mathcal{O}(r(T)\log n) = \mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and $\mathcal{O}(\log n)$ time and, with minor modifications, achieves $\mathcal{O}(\frac{\log n}{\log\log n})$ time and $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space; see Table 1. Our lower bounds (Theorems 4.2, 4.3, 4.5, and 4.7) hold even when the input text $T \in \Sigma^n$ is over a binary alphabet $\Sigma = \{\mathtt{0},\mathtt{1}\}$. In a single theorem, they can be summarized as follows:

Theorem 1.1. *There is no data structure that, for every text $T \in \{\mathtt{0},\mathtt{1}\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers any of the following queries:*

- *longest common prefix (LCP) array queries (Section 4.1),*
- *suffix array (SA) queries (Section 4.2),*
- *inverse suffix array (SA^{-1}) queries (Section 4.3), and*
- *longest common extension (LCE) queries (Section 4.4),*

in $o(\frac{\log n}{\log\log n})$ time.

Together with previous works [Ryt03, CLL+05, KP18, GNP18, KK20, KS22, KNP23] (which established relations between repetitiveness measures as well as the optimal *space* bounds for compressed indexing), our results (concerning the optimal *query time* bounds of compressed indexes) establish the two frontiers of optimization for future trade-offs, which can focus on either achieving the optimal query time in the smallest possible space, or the fastest possible query within the optimal space (and everything in between), *both* of which are now known for central string queries.

Lower Bounds for Faster Queries Our second contribution is extending our set of lower bounds to cover fundamental string queries that admit faster query times. This includes central queries such as BWT access, LF, PLCP, and lexicographical predecessor (Φ) queries (see Section 2), all of which can be supported in $\mathcal{O}(\log\log n)$ time using $\mathcal{O}(r(T)\log n) = \mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space [GNP20].

The $\Theta(\log\log n)$ runtime of these queries stems from the use of predecessor data structures, and until now, it was not known whether this was inherent or could be improved by further algorithmic advances. We show that, for all these (and related) problems, we can reverse the reductions and answer predecessor or

[5] Henceforth, σ denotes the alphabet size.

[6] LCP array queries are not directly addressed in [KK23b], but they follow as an immediate consequence. For every $T \in \Sigma^n$ and $i \in [2\,..\,n]$, it holds $\text{LCP}_T[i] = \text{LCE}_T(\text{SA}_T[i], \text{SA}_T[i-1])$ (see Section 2). In other words, if we can support SA and LCE queries, then we can also support LCP array queries in the same time (up to a constant factor).

colored predecessor queries (see Sections 2.5 and 2.6), which, in our regime of parameters, require $\Omega(\log\log n)$ time [PT06].

All our lower bounds again hold even when the input text $T \in \Sigma^n$ is over a binary alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. Our results (Theorems 5.4, 5.6, 5.8, 5.10, 5.20, and 5.24) can be summarized in a single theorem as follows:

Theorem 1.2. *There is no data structure that, for every text $T \in \{\mathtt{0}, \mathtt{1}\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers any of the following queries:*

- *Burrows–Wheeler Transform (BWT) queries (Section 5.1),*
- *permuted longest common prefix (PLCP) array queries (Section 5.2),*
- *last-to-first (LF) mapping queries (Section 5.3),*
- *inverse last-to-first (LF^{-1}) queries (Section 5.4),*
- *lexicographical predecessor (Φ) queries (Section 5.5), and*
- *inverse lexicographical predecessor (Φ^{-1}) queries (Section 5.6),*

in $o(\log\log n)$ time.

Related Work In this work, we focus on compressed indexes capable of strong (up to exponential) compression of the input text, which exploit redundancy arising from repeated substrings. Such redundancy is quantified using measures including substring complexity [KNP23], string attractor size [KP18], LZ77 size [ZL77], run-length BWT size [BW94, GNP20], grammar compression [Ryt03, CLL+05], LZ-End size [KN10, KS22], and others; see [Nav21a] for an overview.

Beyond highly repetitive texts, prior work has also addressed the *compact* or *entropy-bounded* setting, in which the text $T \in [0\,..\,\sigma)^n$ is indexed using $\mathcal{O}(n\log\sigma)$ bits of space (or, in some cases, space proportional to the empirical entropy $H_k(T)$). This is asymptotically precisely the space needed to represent any text $T \in [0\,..\,\sigma)^n$ in the worst case and improves upon classical indexes such as the suffix array [MM90] and the suffix tree [Wei73], which require $\Omega(n\log n)$ bits. The two main classes of space-efficient indexes—those based on repetitiveness and those based on entropy bounds—are largely incomparable, since entropy-based measures do not account for large-scale repetitions [KP18]. As a result, these approaches have been studied independently, using different techniques.

Many of the queries considered in this paper have been studied in the compact or entropy-bounded setting, including suffix array queries [FM00, GV00, KK23a], BWT and LF-mapping queries [FM00], lexicographic predecessor/successor queries [GV00], longest common extension (LCE) queries [KK19], and LCP/PLCP array queries [Sad02]. These foundational compact indexes introduced several unidirectional reductions, showing that, e.g., suffix array access can be reduced to rank queries over the BWT [FM00].

Recent work on compact indexes [KK23a, KK24, KK26] introduced an alternative framework based on *prefix range queries*. In particular, [KK23a] showed that many central indexing queries (such as SA and inverse SA queries) can be reduced to prefix select and prefix special rank queries (two specific types of prefix range queries). The resulting text indexes match the query time and space of state-of-the-art indexes [FM00, GV00] while also supporting sublinear-time construction in optimal working space. Subsequent work [KK26] has further extended these reductions, making them *bidirectional*, i.e., establishing that, up to a very small additive term in query time, prefix range queries are in fact *equivalent* to central string queries. This provides a unified view of previous reduction-based techniques and shows that, in the compact setting, many core string-processing queries can be equivalently expressed in terms of much simpler prefix range queries.

Reductions of this nature for *algorithms* are less common. A notable example is [KK25], which develops a broad reduction hierarchy including problems such as LZ77 factorization, BWT construction, batched ISA/LPF queries, longest common factor, and inversion counting, whose state-of-the-art algorithms achieve $\mathcal{O}(n\sqrt{\log n})$ time on inputs of n words (i.e., $n\log n$ bits).

Organization of the Paper First, in Section 2, we introduce the notation and definitions used in the paper. In Section 3, we then give an overview of our lower bounds. In Section 4, we describe our lower bounds for queries with complexity $\Theta\big(\frac{\log n}{\log\log n}\big)$, including longest common prefix (LCP) array queries (Section 4.1), suffix array (SA) queries (Section 4.2), inverse suffix array (SA^{-1}) queries (Section 4.3), and longest common extension (LCE) queries (Section 4.4). In Section 5, we describe our lower bounds for queries with complexity $\Theta(\log\log n)$, including Burrows–Wheeler Transform (BWT) queries (Section 5.1), permuted longest common

prefix (PLCP) array queries (Section 5.2), last-to-first (LF) mapping queries (Section 5.3), inverse last-to-first (LF^{-1}) queries (Section 5.4), lexicographical predecessor (Φ) queries (Section 5.5), and inverse lexicographical predecessor (Φ^{-1}) queries (Section 5.6).

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 $i \in [1\,..\,|S|]$,[7] 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 $1 \le i \le j \le |S|+1$. 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 $U \cdot V$. Furthermore, $S^k = \bigodot_{i=1}^{k} S$ is the concatenation of $k \in \mathbb{Z}_{\ge 0}$ 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 $U \in \Sigma^+$ and $k \ge 2$. 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 $i \in [1\,..\,\min(|U|,|V|)]$. For any string $S = c_1^{\ell_1}c_2^{\ell_2}\cdots c_h^{\ell_h}$, where $c_i \in \Sigma$ and $\ell_i > 0$ holds for $i \in [1\,..\,h]$, and $c_i \ne c_{i+1}$ holds for $i \in [1\,..\,h)$, we define the *run-length encoding* of S as a sequence $\mathrm{RL}(S) = ((c_1,\ell_1),\ldots,(c_h,\ell_h))$.

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

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

2.2 Suffix Arrays

Definition 2.2 (Suffix array)**.** For any string $T \in \Sigma^n$ of length $n \ge 1$, 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.

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 $\mathrm{Occ}(P,T) = \emptyset$.

Example 2.4. For the example text T in Fig. 1 and pattern $P = \texttt{ababa}$, it holds $\mathrm{Occ}(P,T) = \{15,10,8,6\} = \{\mathrm{SA}_T[i] : i \in (6\,..\,10]\}$, so $\mathrm{RangeBeg}(P,T) = 6$ and $\mathrm{RangeEnd}(P,T) = 10$.

Definition 2.5 (Inverse suffix array)**.** The *inverse suffix array* of a text $T \in \Sigma^n$ of length $n \ge 1$ is an array $\mathrm{SA}_T^{-1}[1\,..\,n]$ containing the inverse permutation of SA_T (Definition 2.2), i.e., $\mathrm{SA}_T^{-1}[j] = i$ holds if and only if $\mathrm{SA}_T[i] = j$. Equivalently, $\mathrm{SA}_T^{-1}[j]$ stores the lexicographic *rank* of $T[j\,..\,n]$ among the suffixes of T, that is, $\mathrm{SA}_T^{-1}[j] = 1 + \mathrm{RangeBeg}(T[j\,..\,n], T)$ (Definition 2.1). See Fig. 2 for an example.

Definition 2.6 (Lexicographical predecessor array Φ)**.** For any string $T \in \Sigma^n$ of length $n \ge 1$, we define $\Phi_T[1\,..\,n]$ so that $\Phi_T[\mathrm{SA}_T[1]] = \mathrm{SA}_T[n]$, and for every $j \in [1\,..\,n] \setminus \{\mathrm{SA}_T[1]\}$,

$$\Phi_T[j] = \mathrm{SA}_T[\mathrm{SA}_T^{-1}[j] - 1]$$

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

i	SA_T	LF_T	LF_T^{-1}	LCP_T	BWT_T	$T[\mathrm{SA}_T[i]\,..\,n]$
1	19	11	19	0	b	a
2	14	12	7	1	b	aababa
3	5	13	10	6	b	aababababaababa
4	17	14	11	1	b	aba
5	12	15	12	3	b	abaababa
6	3	16	13	8	b	abaababababaababa
7	15	2	14	3	a	ababa
8	10	17	15	5	b	ababaababa
9	8	18	17	5	b	abababaababa
10	6	3	18	7	a	abababababaababa
11	18	4	1	0	a	ba
12	13	5	2	2	a	baababa
13	4	6	3	7	a	baababababaababa
14	16	7	4	2	a	baba
15	11	8	5	4	a	babaababa
16	2	19	6	9	b	babaababababaababa
17	9	9	8	4	a	bababaababa
18	7	10	9	6	a	bababababaababa
19	1	1	16	1	a	bbabaababababaababa

Figure 1: A list of all sorted suffixes of $T =$ `bbabaababababaababa` along with the arrays SA_T (Definition 2.2), LF_T (Definition 2.8), LF_T^{-1} (Definition 2.9), LCP_T (Definition 2.11), and BWT_T (Definition 2.26).

$$
\begin{aligned}
i &= [1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ 11,\ 12,\ 13,\ 14,\ 15,\ 16,\ 17,\ 18,\ 19]\\
T &= [\mathtt{b},\ \mathtt{b},\ \mathtt{b},\ \mathtt{b},\ \mathtt{b},\ \mathtt{b},\ \mathtt{a},\ \mathtt{b},\ \mathtt{b},\ \mathtt{a},\ \mathtt{a},\ \mathtt{a},\ \mathtt{a},\ \mathtt{a},\ \mathtt{a},\ \mathtt{b},\ \mathtt{a},\ \mathtt{a},\ \mathtt{a}]\\
\mathrm{SA}_T^{-1} &= [19,\ 16,\ 6,\ 13,\ 3,\ 10,\ 18,\ 9,\ 17,\ 8,\ 15,\ 5,\ 12,\ 2,\ 7,\ 14,\ 4,\ 11,\ 1]\\
\Phi_T &= [7,\ 11,\ 12,\ 13,\ 14,\ 8,\ 9,\ 10,\ 2,\ 15,\ 16,\ 17,\ 18,\ 19,\ 3,\ 4,\ 5,\ 6,\ 1]\\
\Phi_T^{-1} &= [19,\ 9,\ 15,\ 16,\ 17,\ 18,\ 1,\ 6,\ 7,\ 8,\ 2,\ 3,\ 4,\ 5,\ 10,\ 11,\ 12,\ 13,\ 14]\\
\mathrm{PLCP}_T &= [1,\ 9,\ 8,\ 7,\ 6,\ 7,\ 6,\ 5,\ 4,\ 5,\ 4,\ 3,\ 2,\ 1,\ 3,\ 2,\ 1,\ 0,\ 0]
\end{aligned}
$$

Figure 2: The arrays SA_T^{-1} (Definition 2.5), Φ_T (Definition 2.6), Φ_T^{-1} (Definition 2.7), and PLCP_T (Definition 2.12) for the example string $T =$ `bbabaababababaababa` (the same as in Fig. 1).

(see Definitions 2.2 and 2.5). Equivalently, $\Phi_T[1\,..\,n]$ is the unique array that satisfies $\Phi_T[\mathrm{SA}_T[i]] = \mathrm{SA}_T[i-1]$ for every $i \in (1\,..\,n]$, and $\mathrm{SA}_T[\mathrm{SA}_T[1]] = \mathrm{SA}_T[n]$. See Fig. 2 for an example.

Definition 2.7 (Inverse lexicographical predecessor array Φ^{-1})**.** The *inverse lexicographical predecessor array* (or simply *lexicographical successor array*) of a text $T \in \Sigma^n$ of length $n \geq 1$ is an array $\Phi_T^{-1}[1\,..\,n]$ containing the inverse permutation of Φ_T (Definition 2.6), i.e., such that $\Phi_T^{-1}[j] = i$ holds if and only if $\Phi_T[i] = j$. Equivalently, $\Phi_T^{-1}[1\,..\,n]$ is the unique array that satisfies the equation $\Phi_T^{-1}[\mathrm{SA}_T[i]] = \mathrm{SA}_T[i+1]$ for every $i \in [1\,..\,n)$, and $\Phi_T^{-1}[\mathrm{SA}_T[n]] = \mathrm{SA}_T[1]$. See Fig. 2 for an example.

Definition 2.8 (Last-to-First mapping LF)**.** The *Last-to-First (LF) mapping* of a text $T \in \Sigma^n$ of length $n \geq 1$ is an array $\mathrm{LF}_T[1\,..\,n]$ defined so that, for every $i \in [1\,..\,n]$,

$$
\mathrm{LF}_T[i] := \begin{cases} \mathrm{SA}_T^{-1}[\mathrm{SA}_T[i]-1] & \text{if } \mathrm{SA}_T[i] \neq 1,\\ \mathrm{SA}_T^{-1}[n] & \text{otherwise.} \end{cases}
$$

Equivalently, $\mathrm{LF}_T[1\,..\,n]$ is the unique array that satisfies the equation $\mathrm{LF}_T[\mathrm{SA}_T^{-1}[j]] = \mathrm{SA}_T^{-1}[j-1]$ for every $j \in (1\,..\,n]$ and $\mathrm{LF}_T[\mathrm{SA}_T^{-1}[1]] = \mathrm{SA}_T^{-1}[n]$. See Fig. 1 for an example.

Definition 2.9 (First-to-Last mapping LF^{-1})**.** The *First-to-Last (inverse LF) mapping* of a text $T \in \Sigma^n$ of length $n \geq 1$ is an array $\mathrm{LF}_T^{-1}[1\,..\,n]$ containing the inverse permutation of LF_T (Definition 2.8), i.e., $\mathrm{LF}_T^{-1}[j] = i$ holds if and only if $\mathrm{LF}_T[i] = j$. Equivalently, $\mathrm{LF}_T^{-1}[1\,..\,n]$ is the unique array that satisfies the

equation $\mathrm{LF}_T^{-1}[\mathrm{SA}_T^{-1}[j]] = \mathrm{SA}_T^{-1}[j+1]$ for every $j \in [1\,..\,n)$ and $\mathrm{LF}_T^{-1}[\mathrm{SA}_T^{-1}[n]] = \mathrm{SA}_T^{-1}[1]$. See Fig. 1 for an example.

Remark 2.10. The inverse LF mapping (Definition 2.9) for text T is also denoted as Ψ_T [GV00].

2.3 LCP Arrays

Definition 2.11 (Longest common prefix (LCP) array)**.** For any string $T \in \Sigma^n$ of length $n \geq 1$, the *longest common prefix (LCP) array* $\mathrm{LCP}_T[1\,..\,n]$ is an array such that $\mathrm{LCP}_T[1] = 0$, and for every $i \in [2\,..\,n]$,

$$\mathrm{LCP}_T[i] = \mathrm{LCE}_T(\mathrm{SA}_T[i], \mathrm{SA}_T[i-1])$$

(see Section 2.1 and Definition 2.2). See Fig. 1 for an example.

Definition 2.12 (Permuted longest common prefix (PLCP) array)**.** For any string $T \in \Sigma^n$ of length $n \geq 1$, the *permuted longest common prefix (PLCP) array* $\mathrm{PLCP}_T[1\,..\,n]$ is defined so that $\mathrm{PLCP}_T[\mathrm{SA}_T[1]] = 0$ and, for every $j \in [1\,..\,n] \setminus \{\mathrm{SA}_T[1]\}$,

$$\mathrm{PLCP}_T[j] = \mathrm{LCE}_T(j, \Phi_T[j]) = \mathrm{LCE}_T(j, \mathrm{SA}_T[\mathrm{SA}_T^{-1}[j]-1])$$

(see Section 2.1 and Definitions 2.2, 2.5, and 2.6). Equivalently, PLCP_T is the unique array satisfying the equation $\mathrm{PLCP}_T[\mathrm{SA}_T[i]] = \mathrm{LCP}_T[i]$ for every $i \in [1\,..\,n]$ (see Definition 2.11). See Fig. 2 for an example.

2.4 Repetitiveness Measures

2.4.1 Lempel–Ziv Factorization

Definition 2.13 (LZ77-like factorization)**.** A partition $T = f_1 f_2 \dots f_k$ of a text $T \in \Sigma^*$ is called an *LZ77-like factorization* of T, if for every $i \in [1\,..\,k]$, it holds either $|f_i| = 1$ or, letting $j = |f_1 f_2 \dots f_{i-1}|$, there exists $j' \in [1\,..\,j)$ satisfying $\mathrm{LCE}_T(j, j') \geq |f_i|$. Each substring f_i in the LZ77-like factorization is called a *phrase*. If $|f_i| = 1$, then f_i is called a *literal* phrase. Otherwise, it is called a *repeat* phrase. If f_i is a repeat phrase, and $j' \in [1\,..\,j)$ (where $j = |f_1 \dots f_{i-1}|$) is an index satisfying $\mathrm{LCE}_T(j, j') \geq |f_i|$, then the substring $T[j'\,..\,j'+|f_i|)$ is called a *source* of phrase f_i.

Remark 2.14. Observe that if T has an LZ77-like factorization consisting of k phrases, then T can be encoded in $\mathcal{O}(k)$ space. In the underlying representation, the literal phrase f_i is encoded as a pair $(T[j+1], 0)$, where $j = |f_1 \cdots f_{i-1}|$. A repeat phrase f_i is encoded as $(j', |f_i|)$, where $j' \in [1\,..\,j)$ (with $j = |f_1 \dots f_{i-1}|$) is any index satisfying $\mathrm{LCE}_T(j, j') \geq |f_i|$.

Definition 2.15 (Longest previous factor (LPF) array)**.** For every text $T \in \Sigma^n$ of length $n \geq 1$, we define the *longest previous factor (LPF) array* $\mathrm{LPF}_T[1\,..\,n]$ such that $\mathrm{LPF}_T[1] = 0$, and for every $j \in [2\,..\,n]$, it holds (see Definition 2.1)

$$\mathrm{LPF}_T[j] = \max\{\ell \in [0\,..\,n-j+1] : \min \mathrm{Occ}(T[j\,..\,j+\ell), T) < j\}.$$

Definition 2.16 (Lempel–Ziv (LZ77) factorization [ZL77])**.** The *LZ77 factorization* of T is a *greedy* left-to-right LZ77-like factorization of $T = f_1 f_2 \dots f_k$ defined such that, for every $i \in [1\,..\,k]$,

$$|f_i| = \max(1, \mathrm{LPF}_T[j+1])$$

(see Definition 2.15), where $j = |f_1 \dots f_{i-1}|$. We denote the number of phrases in the LZ77 factorization by $z(T)$ (i.e.., $k = z(T)$).

Example 2.17. The text T in Fig. 1 has the LZ77 factorization $T = \mathtt{b} \cdot \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{ba} \cdot \mathtt{aba} \cdot \mathtt{bababa} \cdot \mathtt{ababa}$ with $z(T) = 7$ phrases, and its example representation is $(\mathtt{b}, 0), (1, 1), (\mathtt{a}, 0), (2, 2), (3, 3), (7, 6), (10, 5)$.

Theorem 2.18 ([LZ76, Theorem 1])**.** *For every LZ77-like factorization $T = f_1 f_2 \dots f_k$ of a text T, it holds $z(T) \leq k$.*

Observation 2.19. *For any text T satisfying $\mathrm{RL}(T) = k$, it holds $z(T) = \mathcal{O}(k)$.*

Proof. It is easy to write the LZ-like factorization of T consisting of $2k$ phrases. The claim thus follows by Theorem 2.18. □

2.4.2 Grammar Compression

Definition 2.20 (Context-free grammar (CFG))**.** A *context-free grammar* (CFG) is a tuple $G = (V, \Sigma, R, S)$, where V is a finite set of *nonterminals* (or *variables*), Σ is a finite set of *terminals*, and $R \subseteq V \times (V \cup \Sigma)^*$ is a set of *productions* (or *rules*). We assume $V \cap \Sigma = \emptyset$ and $S \in V$. The nonterminal S is called the *start symbol*. Nonterminals in $V \setminus \{S\}$ are called *secondary*. If $(N, \gamma) \in R$, then we write $N \to \gamma$. For $u, v \in (V \cup \Sigma)^*$, we write $u \Rightarrow v$ if there exist $u_1, u_2 \in (V \cup \Sigma)^*$ and a rule $N \to \gamma$ such that $u = u_1 N u_2$ and $v = u_1 \gamma u_2$. We say that u *derives* v and write $u \Rightarrow^* v$, if there exists a sequence $u_1, \ldots, u_k$, $k \geq 1$, such that $u = u_1$, $v = u_k$, and $u_i \Rightarrow u_{i+1}$ for $1 \leq i < k$. The *language* of grammar G is the set $L(G) := \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.

Definition 2.21 (Straight-line grammar (SLG))**.** A CFG $G = (V, \Sigma, R, S)$ is a *straight-line grammar (SLG)* if it satisfies the following two conditions:

1. there exists an ordering $(N_1, \ldots, N_{|V|})$ of all elements in V, such that, for every $(N, \gamma) \in R$, letting $i \in [1\,..\,|V|]$ be such that $N = N_i$, it holds $\gamma \in (\{N_{i+1}, \ldots, N_{|V|}\} \cup \Sigma)^*$, and
2. for every nonterminal $N \in V$, there exists exactly one $\gamma \in (V \cup \Sigma)^*$ such that $(N, \gamma) \in R$.

The unique string $\gamma \in (V \cup \Sigma)^*$ such that $(N, \gamma) \in R$ is called the *definition* of nonterminal N, and is denoted $\mathrm{rhs}_G(N)$. Note that in an SLG, for every $\alpha \in (V \cup \Sigma)^*$, there exists exactly one string $\gamma \in \Sigma^*$ satisfying $\alpha \Rightarrow^* \gamma$. Such γ is called the *expansion* of α, and is denoted $\exp_G(\alpha)$. In particular, in an SLG, we have $|L(G)| = 1$. We define the size of an SLG as $|G| = \sum_{N \in V} \max(|\mathrm{rhs}_G(N)|, 1)$.

Remark 2.22. Note that every SLG G of size $|G| = g$ can be encoded in $\mathcal{O}(g)$ space: we choose an ordering of nonterminals and write down the definitions of all variables, replacing nonterminals with their index in this order.

Definition 2.23 (Straight-line program (SLP))**.** An SLG $G = (V, \Sigma, R, S)$ is called a *straight-line program (SLP)* if for every $N \in V$, either $\mathrm{rhs}_G(N) = AB$ for some $A, B \in V$, or $\mathrm{rhs}_G(N) = c$ for some $c \in \Sigma$.

Definition 2.24 (Parse tree)**.** Let $G = (V, \Sigma, R, S)$ be an SLG. The *parse tree* of $A \in V \cup \Sigma$ is a rooted, ordered tree $\mathcal{T}_G(A)$, where each node v is associated with a symbol $s(v) \in V \cup \Sigma$. The root of $\mathcal{T}_G(A)$ is a node ρ such that $s(\rho) = A$. If $A \in \Sigma$, then ρ has no children. If $A \in V$ and $\mathrm{rhs}_G(A) = B_1 \cdots B_k$, then ρ has k children, and the subtree rooted at the ith child is (a copy of) $\mathcal{T}_G(B_i)$. The parse tree of G is defined as the parse tree $\mathcal{T}_G(S)$ of the start symbol S.

Definition 2.25 (Height of SLG)**.** Let $G = (V, \Sigma, R, S)$ be an SLG. The *height* of G is defined as the height of its parse tree.

2.4.3 Run-Length Burrows–Wheeler Transform

Definition 2.26 (Burrows–Wheeler transform (BWT) [BW94])**.** The *Burrows–Wheeler Transform* (BWT) of a text $T \in \Sigma^n$ of length $n \geq 1$, denoted $\mathrm{BWT}_T[1\,..\,n]$, is a permutation of the symbols in T such that, for every $i \in [1\,..\,n]$,

$$\mathrm{BWT}_T[i] := \begin{cases} T[\mathrm{SA}_T[i] - 1] & \text{if } \mathrm{SA}_T[i] > 1, \\ T[n] & \text{otherwise.} \end{cases}$$

Equivalently, $\mathrm{BWT}_T[i] = T^\infty[\mathrm{SA}_T[i] - 1]$. By $r(T) := |\mathrm{RL}(\mathrm{BWT}_T[1\,..\,n])|$, we denote the number of equal-letter runs in the BWT of T. See Fig. 1 for an example.

Example 2.27. For the example text T in Fig. 1, we have $r(T) = |\mathrm{RL}(\mathtt{b}^6\mathtt{a}^1\mathtt{b}^2\mathtt{a}^6\mathtt{b}^1\mathtt{a}^3)| = 6$.

Theorem 2.28 ([GNP20])**.** *Let $\epsilon \in (0, 1)$ be a constant. For every nonempty text $T \in \Sigma^n$, there exists a data structure of size $\mathcal{O}(r(T) \log^{1+\epsilon} n)$ (see Definition 2.26) that answers suffix array, inverse suffix array, and LCP array queries on T (that, given any $i \in [1\,..\,n]$, return $\mathrm{SA}_T[i]$, $\mathrm{SA}_T^{-1}[i]$, and $\mathrm{LCP}_T[i]$; see Definitions 2.2, 2.5, and 2.11) in $\mathcal{O}\big(\frac{\log n}{\log \log n}\big)$ time.*

Remark 2.29. Although the above results are not directly stated in [GNP20] (the data structures supporting SA, inverse SA, and LCP queries described in [GNP20, Theorems 5.4, 5.7, and 5.8] for a text $T \in \Sigma^n$ use $\mathcal{O}(r(T) \log n)$ space and support the queries in $\mathcal{O}(\log n)$ time), with a minor modification, the query time can

be improved to $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ to match the lower bound established in this paper (at the price of a small space increase).

Specifically, instead of storing two half-blocks around each BWT block boundary, we store $\tau = \Theta(\log^\epsilon n)$ blocks (for any constant $\epsilon > 0$), and adjust the block length to be a power of τ (rather than a power of 2). This is a standard technique, used for instance in [KP18, BCG+21].

The resulting data structures support SA, inverse SA, and LCP queries in $\mathcal{O}(\log_\tau n) = \mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ time and achieve the space stated in Theorem 2.28.

Theorem 2.30 ([GNP20]). *For every nonempty text $T \in \Sigma^n$, there exists a data structure of size $\mathcal{O}(r(T))$ (see Definition 2.26) that answers PLCP array queries on T (that, given any $j \in [1\,..\,n]$, return $\mathrm{PLCP}_T[j]$; see Definition 2.12) in $\mathcal{O}(\log\log n)$ time.*

2.4.4 Substring Complexity

Definition 2.31 (Substring counting function). For a string $T \in \Sigma^n$ and $\ell \in \mathbb{Z}_{>0}$, we denote the number of length-ℓ substrings by

$$\mathrm{d}_\ell(T) = |\{T[i\,..\,i+\ell) : i \in [1\,..\,n-\ell+1]\}|.$$

Note that for every $\ell > n$, $\mathrm{d}_\ell(T) = 0$.

Definition 2.32 (Substring complexity [KNP23]). The *substring complexity* of T is defined as $\delta(T) = \max_{\ell=1}^{n} \frac{1}{\ell}\mathrm{d}_\ell(T)$.

Lemma 2.33 ([KNP23, Lemma II.4]). *For every text $T \in [0\,..\,\sigma)^n$, it holds*

$$z(T) = \mathcal{O}\Big(\delta(T)\log\tfrac{n\log\sigma}{\delta(T)\log n}\Big).$$

Remark 2.34. A simplified statement of the upper bound from Lemma 2.33 is $z(T) = \mathcal{O}(\delta(T)\log n)$.

Theorem 2.35 ([KK20, Theorem III.7]). *For every text $T \in \Sigma^n$, it holds*

$$r(T) = \mathcal{O}\left(\big(\delta(T)\log\delta(T)\big)\cdot\max\left(1, \log\tfrac{n}{\delta(T)\log\delta(T)}\right)\right).$$

The above bound is asymptotically tight for all combinations of n, $r(T)$, and $\delta(T)$.

Remark 2.36. A simplified statement of the upper bound from Theorem 2.35 is $r(T) = \mathcal{O}(\delta(T)\log^2 n)$.

By combining [KP18, Lemma 3.7 and Theorem 3.9] and [KNP23, Lemma II.3], we obtain the following result.

Theorem 2.37 ([KP18, KNP23]). *For every text T, it holds*

1. $\delta(T) = \mathcal{O}(z(T))$,
2. $\delta(T) = \mathcal{O}(r(T))$.

By combining [VY13] with Lemma 2.33, we obtain the following result.

Theorem 2.38 ([VY13, KNP23]). *There is no data structure that, for every text $T \in \{\mathtt{0},\mathtt{1}\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers random access queries on T (that, given any $j \in [1\,..\,n]$, return $T[j]$) in $o(\frac{\log n}{\log\log n})$ time.*

The above lower bound is tight. By combining [BCG+21] with Lemma 2.33, we obtain the following result.

Theorem 2.39 ([BCG+21, KNP23]). *Let $\epsilon > 0$ be a positive constant. There exists a data structure that, for every $T \in \Sigma^n$, uses $\mathcal{O}(\delta(T)\log^{2+\epsilon} n)$ space and answers random access queries on T (that, given any $j \in [1\,..\,n]$, return $T[j]$) in $\mathcal{O}(\frac{\log n}{\log\log n})$ time.*

2.5 Predecessor Queries

Definition 2.40 (Predecessor)**.** Consider a nonempty set $A \subseteq \mathbb{Z}$. For every $x \in \mathbb{Z}$, we define

$$\mathsf{pred}_A(x) = \max\{y \in A : y < x\} \cup \{-\infty\}.$$

Example 2.41. Let $A = \{2, 5, 7, 8, 10, 12\}$. Then, it holds $\mathsf{pred}_A(9) = 8$, $\mathsf{pred}_A(5) = 2$, and $\mathsf{pred}_A(2) = -\infty$.

Theorem 2.42 (Y-fast tries [Wil83])**.** *Consider a sequence $(a_i)_{i\in[0\,..\,m]}$, where $m \geq 1$, such that $a_0 = -\infty$, $a_1 < a_2 < \cdots < a_m$, and there exists $u \in \mathbb{Z}_{>0}$ such that, for every $i \in [1\,..\,m]$, it holds $a_i \in [0\,..\,u]$. Then, there exists a data structure of size $\mathcal{O}(m)$ that, given any $x \in \mathbb{Z}$, in $\mathcal{O}(\log\log u)$ time returns $i \in [0\,..\,m]$ satisfying $a_i = \mathsf{pred}_A(x)$ (Definition 2.40), where $A = \{a_1, a_2, \ldots, a_m\}$.*

Theorem 2.43 (Fusion trees [FW93])**.** *Consider a sequence $(a_i)_{i\in[0\,..\,m]}$, where $m \geq 1$, such that $a_0 = -\infty$, $a_1 < a_2 < \cdots < a_m$, and, for every $i \in [1\,..\,m]$, it holds $a_i \in [0\,..\,u]$, where $u = 2^w$. In the word RAM model with w-bit word size, there exists a data structure of size $\mathcal{O}(m)$ that, given any $x \in \mathbb{Z}$, in $\mathcal{O}(1 + \log_w m)$ time returns $i \in [0\,..\,m]$ satisfying $a_i = \mathsf{pred}_A(x)$ (Definition 2.40), where $A = \{a_1, a_2, \ldots, a_m\}$.*

Theorem 2.44 ([PT06])**.** *There is no data structure that, for every nonempty set $A \subseteq [1\,..\,m^2]$ of size $|A| = m$, uses $\mathcal{O}(m \log^{\mathcal{O}(1)} m)$ space and answers predecessor queries on A (that given any $x \in \mathbb{Z}$, return $\mathsf{pred}_A(x)$; see Definition 2.40) in $o(\log\log m)$ time.*

2.6 Colored Predecessor Queries

Definition 2.45 (Colored predecessor)**.** Consider a nonempty set $A \subseteq \mathbb{Z}$. For every $x \in \mathbb{Z}$, we define[8]

$$\mathsf{pred\text{-}color}_A(x) = |\{y \in A : y < x\}| \bmod 2.$$

Example 2.46. Let $A = \{2, 5, 7, 8, 10, 12\}$. Then, it holds $\mathsf{pred\text{-}color}_A(9) = 0$ and $\mathsf{pred\text{-}color}_A(8) = 1$.

Theorem 2.47 ([PT06])**.** *There is no data structure that, for every nonempty set $A \subseteq [1\,..\,m^2]$ of size $|A| = m$, uses $\mathcal{O}(m \log^{\mathcal{O}(1)} m)$ space and answers colored predecessor queries on A (that given any $x \in \mathbb{Z}$, return $\mathsf{pred\text{-}color}_A(x)$; see Definition 2.45) in $o(\log\log m)$ time.*

2.7 Range Queries

Definition 2.48 (Range counting and selection queries)**.** 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

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

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 2.49 (Range minimum query (RMQ))**.** Let $A[1\,..\,m]$ be an array of $m \geq 0$ integers. For every $b, e \in [0\,..\,m]$ such that $b < e$, we define (see Remark 2.50)

$$\mathsf{rmq}_A(b, e) := \operatorname*{arg\,min}_{i\in(b\,..\,e]} A[i].$$

Remark 2.50. We assume that $\arg\min\{f(x) : x \in S\}$ returns the *smallest* $y \in S$ such that $f(y) = \min\{f(x) : x \in S\}$.

Example 2.51. Let $A = [5, 1, 2, 8, 4, 7, 6, 2, 9]$. Then, $\mathsf{range\text{-}count}_A(6, 4) = 4$, $\mathsf{range\text{-}select}_A(4, 5) = 7$, and $\mathsf{rmq}_A(2, 9) = 3$.

[8] This definition of the *colored predecessor problem* is equivalent to the original definition [PT06], in which the color of each integer is an arbitrary bit, and two consecutive integers may have the same color. It is easy to see that if two adjacent integers have the same color, the larger can be removed without affecting the query answer. Thus, we may assume adjacent elements have alternating colors, which is equivalent to defining the color of each element as the parity of its rank.

Theorem 2.52 ([FH11]). *For every array $A[1\,..\,m]$ of m integers, there exists a data structure of $\mathcal{O}(m)$ bits that answers range minimum queries on A in $\mathcal{O}(1)$ time and can be constructed in $\mathcal{O}(m)$ time.*

Theorem 2.53 ([Păt07]). *There is no data structure that, for every array $A[1\,..\,n]$ containing a permutation of $\{1,\ldots,n\}$, uses $\mathcal{O}(n\log^{\mathcal{O}(1)} n)$ space and answers range counting queries on A (given any $j \in [0\,..\,m]$ and $v \geq 0$, return $\mathsf{range\text{-}count}_A(j,v)$; see Definition 2.48) in $o(\frac{\log n}{\log\log n})$ time.*

Theorem 2.54 ([JL11]). *There is no data structure that, for every array $A[1\,..\,n]$ containing a permutation of $\{1,\ldots,n\}$, uses $\mathcal{O}(n\log^{\mathcal{O}(1)} n)$ space and answers range selection queries on A (given any $v \geq 0$ and $r \in [1\,..\,\mathsf{range\text{-}count}_A(m,v)]$, return $\mathsf{range\text{-}select}_A(r,v)$; see Definition 2.48) in $o(\frac{\log n}{\log\log n})$ time.*

2.8 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.

3 Technical Overview

3.1 Overview of the $\Omega(\frac{\log n}{\log\log n})$ Lower Bounds

In this section, we sketch the ideas of our lower bound for LCP array access queries showing that there is no data structure that, for every $T \in \{0,1\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers LCP array queries (that, given any $i \in [1\,..\,n]$, return $\mathrm{LCP}_T[i]$; see Definition 2.11 and Section 4.1.1) in $o(\frac{\log n}{\log\log n})$. This lower bound is asymptotically tight, since LCP queries can be answered in $\mathcal{O}(r(T)\log^{1+\epsilon} n) = \mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and $\mathcal{O}(\frac{\log n}{\log\log n})$ time (see Table 1).

It is known that there is no data structure that, for every array $A[1\,..\,m]$ containing a permutation of $\{1,\ldots,m\}$, uses near-linear (i.e., $\mathcal{O}(m\log^{\mathcal{O}(1)} m)$) space and answers range counting or selection queries in $o(\frac{\log m}{\log\log m})$ time [Păt07, JL11]; see Theorems 2.53 and 2.54.

The central idea in our reduction is to prove that if, for every binary text $T \in \{0,1\}^n$, there exists a data structure using $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answering LCP array queries in $t_{\mathrm{LCP}}(|T|) = o(\frac{\log|T|}{\log\log|T|})$ time, then we could break the lower bound for answering range select queries mentioned above; i.e., given any array $A[1\,..\,m]$ containing a permutation of $\{1,\ldots,m\}$, there exists a data structure of size $\mathcal{O}(m\log^{\mathcal{O}(1)} m)$ that answers range selection queries on A in $o(\frac{\log m}{\log\log m})$ time.

For an array $A[1\,..\,m]$ containing a permutation of $\{1,\ldots,m\}$, we define the following string:

$$T_A = \left(\bigodot_{i=1}^{m} (\mathtt{0}^{A[i]}\mathtt{1}^i)\right) \cdot \mathtt{0}^{m+1}\mathtt{1}^{m+1} \in \{\mathtt{0},\mathtt{1}\}^*.$$

Observation: The SA block corresponding to the suffixes of T_A prefixed with $\mathtt{0}^v\mathtt{1}$ can be used to answer queries $\mathsf{range\text{-}select}_A(r,v)$. Recall that the query $\mathsf{range\text{-}select}_A(r,v)$ returns the rth smallest element of the set $\mathcal{I} = \{i \in [1\,..\,m] : A[i] \geq v\}$. Observe that, by definition of T_A, it follows that $\mathtt{0}^v\mathtt{1}$ occurs within the substring $\mathtt{0}^{A[i]}\mathtt{1}^i$ of T_A only when $A[i] \geq v$. This implies that the set $\mathrm{Occ}(\mathtt{0}^v\mathtt{1}, T_A)$ captures precisely the set we are interested in (plus one extra occurrence within the suffix $\mathtt{0}^{m+1}\mathtt{1}^{m+1}$ of T_A). Let $(a_i)_{i\in[1\,..\,k]}$ be the increasing sequence containing all the elements of $\mathcal{I}$, and let $a_{k+1} = m+1$. Observe that since $\mathtt{0}^v\mathtt{1}^{a_1}\mathtt{0} \prec \mathtt{0}^v\mathtt{1}^{a_2}\mathtt{0} \prec \cdots \prec \mathtt{0}^v\mathtt{1}^{a_k}\mathtt{0} \prec \mathtt{0}^v\mathtt{1}^{m+1} = \mathtt{0}^v\mathtt{1}^{a_{k+1}}$, it follows that, for every $i \in [1\,..\,k+1]$, the ith element in the SA block corresponding to $\mathrm{Occ}(\mathtt{0}^v\mathtt{1}, T_A)$ is prefixed with $\mathtt{0}^v\mathtt{1}^{a_i}$ but not with $\mathtt{0}^v\mathtt{1}^{a_i+1}$; i.e., letting $b = \mathrm{RangeBeg}(\mathtt{0}^v\mathtt{1}, T_A)$, the suffix $T_A[\mathrm{SA}_{T_A}[b+i]\,..\,|T_A|]$ is prefixed with $\mathtt{0}^v\mathtt{1}^{a_i}$ but not $\mathtt{0}^v\mathtt{1}^{a_i+1}$. By definition of the LCP array, this implies that, for every $r \in [1\,..\,k]$, it holds

$$\mathrm{LCP}_{T_A}[b+r+1] = \mathrm{lcp}(T_A[\mathrm{SA}_{T_A}[b+r+1]\,..\,|T_A|], T_A[\mathrm{SA}_{T_A}[b+r]\,..\,|T_A|]) = v + a_r.$$

Consequently, we can compute a_r by asking the LCP query with index $b+r+1$ and subtracting v from the resulting value; see Lemma 4.1. Since $\mathsf{range\text{-}select}_A(r,v) = a_r$ holds by definition, we thus obtain a way to answer range select queries by LCP array queries on T_A.

To finish our argument, let us now assume that we can answer LCP array queries on any $T \in \{\mathtt{0},\mathtt{1}\}^*$ in $t_{\mathrm{LCP}}(|T|) = o(\frac{\log|T|}{\log\log|T|})$ time and using $\mathcal{O}(\delta(T)\log^c|T|)$ space (for some positive constant c). Using this structure, we design a structure for answering range selection queries on $A[1\,..\,m]$ so that it consists of the following components:

1. An array $R[1\,..\,m]$ defined so that $R[i] = \mathrm{RangeBeg}(\mathtt{0}^i\mathtt{1}, T_A)$. It needs $\mathcal{O}(m)$ space.
2. The above hypothetical data structure answering LCP queries on T_A. To bound the space for this structure, we need to show that $\delta(T_A) = \mathcal{O}(m\log^{\mathcal{O}(1)} m)$. To this end, observe that T_A can be run-length compressed to size $2(m+1)$. This implies that the LZ77 factorization of T_A has size $\mathcal{O}(m)$ (Observation 2.19), and hence, by the bound $\delta(S) = \mathcal{O}(z(S))$ (Theorem 2.37(1)) holding for all strings S, we obtain that $\delta(T_A) = \mathcal{O}(m)$. Note also that $|T_A| = \mathcal{O}(m^2)$. Thus, the data structure answering LCP queries on T_A needs $\mathcal{O}(\delta(T_A)\log^c|T_A|) = \mathcal{O}(m\log^c(m^2)) = \mathcal{O}(m\log^c m)$ space.

In total, our new data structure for range select queries needs $\mathcal{O}(m\log^c m) = \mathcal{O}(m\log^{\mathcal{O}(1)} m)$ space.

Let $v \geq 0$ and $r \in [1\,..\,\mathsf{range\text{-}count}_A(m,v)]$. To compute $\mathsf{range\text{-}select}_A(r,v)$ using the above hypothetical structure, we proceed as follows (assuming that $v \in [1\,..\,m]$; otherwise, we can answer the query immediately; see Theorem 4.2):

1. First, in $\mathcal{O}(1)$ time we fetch the value $b = R[v] = \mathrm{RangeBeg}(\mathtt{0}^v\mathtt{1}, T_A)$.
2. Then, in $\mathcal{O}(t_{\mathrm{LCP}}(|T_A|))$ time we compute $\ell = \mathrm{LCP}_{T_A}[b+r+1]$ and return $\ell - v$. By the above discussion, $\ell - v = \mathsf{range\text{-}select}_A(r,v)$.

In total, we spend $\mathcal{O}(t_{\mathrm{LCP}}(|T_A|)) = o(\frac{\log|T_A|}{\log\log|T_A|}) = o(\frac{\log(m^2)}{\log\log(m^2)}) = o(\frac{\log m}{\log\log m})$ time.

We have obtained a data structure answering range selection queries in $o(\frac{\log m}{\log\log m})$ time and using $\mathcal{O}(m\log^{\mathcal{O}(1)} m)$ space. This contradicts the lower bound from [JL11] (see also Theorem 2.54). Thus, the data structure for LCP queries we assumed at the beginning cannot exist.

3.2 Overview of the $\Omega(\log\log n)$ Lower Bounds

In this section, we sketch the key ideas of our lower bounds for faster queries, presented in Section 5.

It is known that there is no data structure that, for every $A \subseteq [1\,..\,m^2]$ of size $|A| = m$, uses near-linear (i.e., $\mathcal{O}(m\log^{\mathcal{O}(1)} m)$) space and answers $\mathsf{pred\text{-}color}_A(x)$ queries in $o(\log\log m)$ time [PT06] (see also Theorem 2.47). Note that this implies the same lower bound for any data structure that returns the index of the predecessor (see Theorem 2.44) since such a data structure could be used to answer $\mathsf{pred\text{-}color}_A(x)$ queries (it suffices to store the parity of each element's rank in an array and return it after finding the index of the predecessor).

Overview of the Lower Bound for BWT Queries We now give an overview of our lower bound for BWT queries presented in Section 5.1. Specifically, we outline how to show that there is no data structure that, for every $T \in \{\mathtt{0},\mathtt{1}\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers BWT queries (that, given any $i \in [1\,..\,n]$, return $\mathrm{BWT}_T[i]$; see Section 5.1.1) in $o(\log\log n)$ time. To this end, we will demonstrate that the existence of such a structure implies that, for every set A of m integers in the range $[1\,..\,m^2]$, there exists a data structure of size $\mathcal{O}(m\log^{\mathcal{O}(1)} m)$ that answers colored predecessor queries (that, given any $x \in \mathbb{Z}$, return $\mathsf{pred\text{-}color}_A(x)$) in $o(\log\log m)$ time, contradicting the above lower bound (Theorem 2.47).

Consider any set $A \subseteq [1\,..\,m^2]$ of size $|A| = m$. Denote $A = \{a_1,\ldots,a_m\}$ so that $a_1 < a_2 < \cdots < a_m$. We also define $a_0 = 0$ and $a_{m+1} = m^2$. Finally, let $(b_i)_{i\in[0\,..\,m]}$ be a sequence defined by $b_i = i \bmod 2$. We then define the following string:

$$T_A = \bigodot_{i=0}^{m} \left(b_i \cdot \mathrm{ebin}_k(i)\right)^{a_{i+1}-a_i} \in \{\mathtt{0},\mathtt{1}\}^*,$$

where $k = 1 + \lfloor\log m\rfloor$, $\mathrm{bin}_k(x)$ denotes a bitstring of length exactly k that contains the binary representation of integer x, with leading zeros, and $\mathrm{ebin}_k(x) = \mathtt{1}^{k+1}\cdot\mathtt{0}\cdot\mathrm{bin}_k(x)\cdot\mathtt{0}$.

Observation: The BWT of T_A can be used to answer queries $\mathsf{pred\text{-}color}_A(x)$. Consider the block $\mathrm{SA}_{T_A}[b\,..\,e]$ in the suffix array corresponding to $\mathrm{Occ}(\mathtt{1}^{k+1}\mathtt{0}, T)$. Note that $e - b = m^2$ since the string $\mathtt{1}^{k+1}\mathtt{0}$ occurs only at positions of the form $(2k+4)\cdot t - (2k+2)$ with $t \in [1\,..\,m^2]$. Moreover, observe that, for

every $x_1, x_2 \in [0\,..\,2^k)$ with $x_1 < x_2$, we have $\mathrm{ebin}_k(x_1) \prec \mathrm{ebin}_k(x_2)$. By the definition of T_A, this implies that the first $a_1 - a_0$ suffixes in $\mathrm{SA}_{T_A}(b\,..\,e]$ have $\mathrm{ebin}_k(1)$ as a prefix, the next $a_2 - a_1$ suffixes have $\mathrm{ebin}_k(2)$ as a prefix, and so on. Consequently, the first $a_1 - a_0$ symbols in $\mathrm{BWT}_{T_A}(b\,..\,e]$ are $b_1 = \mathtt{1}$, the next $a_2 - a_1$ symbols are $b_2 = \mathtt{0}$, and so on. More generally, we obtain $\mathrm{BWT}_{T_A}(b\,..\,e] = S$, where $S = b_0^{a_1-a_0} b_1^{a_2-a_1} \cdots b_m^{a_{m+1}-a_m} \in \{\mathtt{0},\mathtt{1}\}^{m^2}$. It remains to observe that S is the answer string for $\mathsf{pred\text{-}color}_A(x)$ queries, i.e., for every $x \in [1\,..\,m^2]$, it holds $\mathsf{pred\text{-}color}_A(x) = S[x]$. Thus, we obtain $\mathsf{pred\text{-}color}_A(x) = \mathrm{BWT}_{T_A}[b + x]$.

Using the above observation, we develop our lower bound as follows. Assume that we can answer BWT queries on any $T \in \{\mathtt{0},\mathtt{1}\}^n$ in $t_{\mathrm{BWT}}(|T|) = o(\log\log|T|)$ time and using $\mathcal{O}(\delta(T)\log^c|T|)$ space (for some positive constant c). Using this structure, we obtain a structure for answering colored predecessor queries on A by storing the position $b = \mathrm{RangeBeg}(\mathtt{1}^{k+1}\mathtt{0}, T_A)$, and the BWT structure for the string T_A.

To bound the space for the BWT structure for T_A, we need to upper bound $\delta(T_A)$. To this end, first note that the substring $(b_i \cdot \mathrm{ebin}_k(i))^{a_{i+1}-a_i}$ has an LZ77-like factorization (see Definition 2.13) of size $2k+5$ (the first $2k+4$ phrases are of length one, and the last phrase encodes the remaining suffix). This implies that T_A has an LZ77-like factorization of size $(m+1)(2k+5)$. Since the LZ77 factorization is the smallest LZ77-like factorization (Theorem 2.18), we thus obtain $z(T_A) \le (m+1)(2k+5) = \mathcal{O}(m\log m)$. By Theorem 2.37(1), we obtain $\delta(T_A) = \mathcal{O}(m\log m)$. It remains to observe that $|T_A| = \mathcal{O}(m^2\log m)$. The data structure for BWT queries applied on T_A hence needs

$$\mathcal{O}(\delta(T_A)\log^c|T_A|) = \mathcal{O}((m\log m)\cdot\log^c(m^2\log m)) = \mathcal{O}(m\log^{c+1} m) = \mathcal{O}(m\log^{\mathcal{O}(1)} m)$$

space. Given this data structure, we can compute $\mathsf{pred\text{-}color}_A(x)$ for any $x \in [1\,..\,m^2]$ (for other x, we can return the answer immediately) by issuing a BWT query to compute $\mathrm{BWT}_{T_A}[b+x]$ in $\mathcal{O}(t_{\mathrm{BWT}}(|T_A|)) = o(\log\log|T_A|) = o(\log\log(m^2\log m)) = o(\log\log m)$ time.

We have obtained a data structure answering colored predecessor queries in $o(\log\log m)$ time and using $\mathcal{O}(m\log^{\mathcal{O}(1)} m)$ space. This contradicts the lower bound from [PT06] (see also Theorem 2.47). Thus, the data structure for BWT queries we assumed at the beginning cannot exist.

Overview of the Lower Bound for PLCP Queries We now give an overview of a slightly different technique for proving a lower bound for PLCP queries presented in Section 5.2. Specifically, we outline how to show that there is no data structure that, for every $T \in \{\mathtt{0},\mathtt{1}\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers PLCP array queries (that, given any $j \in [1\,..\,n]$, return $\mathrm{PLCP}_T[j]$; see Section 5.2.1) in $o(\log\log n)$ time. To this end, we will demonstrate that the existence of such a structure implies that, for every set $A = \{a_1, a_2, \ldots, a_m\}$ of m integers (we assume $a_1 < a_2 < \cdots < a_m$, and also set $a_0 = -\infty$), in the range $[1\,..\,m^2]$, there exists a data structure of size $\mathcal{O}(m\log^{\mathcal{O}(1)} m)$ that, given any $x \in \mathbb{Z}$, in $o(\log\log m)$ time returns the index $i \in [0\,..\,m]$ satisfying $\mathsf{pred}_A(x) = a_i$. With an additional array whose ith entry contains the value a_i, this allows determining $\mathsf{pred}_A(x)$ in $o(\log\log m)$ time, contradicting the predecessor lower bound from [PT06] (Theorem 2.44).

Consider any set $A \subseteq [1\,..\,m^2]$ of size $|A| = m$. Denote $A = \{a_1, \ldots, a_m\}$, where $a_1 < a_2 < \cdots < a_m$. Let also $a_0 = -\infty$ and $a_{m+1} = m^2 + 1$. We then define the following string:

$$T_A = \left(\bigodot_{i=1}^{m} (\mathtt{0}^{a_i}\mathtt{1}^{m-i+2})\right)\cdot\left(\mathtt{0}^{m^2+1}\mathtt{1}\right)\cdot\left(\mathtt{0}^{m^2}\mathtt{1}^{m+2}\right) \in \{\mathtt{0},\mathtt{1}\}^*.$$

Observation: The PLCP array of T_A can be used to answer $\mathsf{pred}_A(x)$ queries. Assume $x \in [2\,..\,m^2]$ and consider the problem of computing $p = \min\{i \in [1\,..\,m+1] : a_i \ge x\}$; observe that $\mathsf{pred}_A(x) = a_{p-1}$ and $\mathsf{pred}_A(1) = -\infty$. Denote $\mathcal{I} = \{i \in [1\,..\,m+1] : a_i \ge x\}$, and note that, since $a_1 < a_2 < \cdots < a_m < a_{m+1}$, we have $\mathcal{I} = \{p, p+1, \ldots, m+1\}$. For every $i \in [1\,..\,m+1]$, $\mathtt{0}^x\mathtt{1}$ occurs within the substring $\mathtt{0}^{a_i}\mathtt{1}^{m-i+2}$ of T_A if and only if $a_i \ge x$. The set $\mathrm{Occ}(\mathtt{0}^x\mathtt{1}, T_A)$ thus captures the set $\mathcal{I}$ (plus one extra position corresponding to the occurrence of $\mathtt{0}^x\mathtt{1}$ within the substring $\mathtt{0}^{m^2}\mathtt{1}^{m+2}$). More precisely, (1) it holds $|\mathrm{Occ}(\mathtt{0}^x\mathtt{1}, T_A)| = m+3-p$, (2) for every $i \in \mathcal{I}$, there exists exactly one position in $\mathrm{Occ}(\mathtt{0}^x\mathtt{1}, T_A)$ having $\mathtt{0}^x\mathtt{1}^{m-i+2}\mathtt{0}$ as a prefix, and (3) there is exactly one position in $\mathrm{Occ}(\mathtt{0}^x\mathtt{1}, T_A)$ prefixed by $\mathtt{0}^x\mathtt{1}^{m+2}$. Since it holds

$$\mathtt{0}^x\mathtt{1}^{m+2}\mathtt{0} \succ \mathtt{0}^x\mathtt{1}^{m-p+2}\mathtt{0} \succ \mathtt{0}^x\mathtt{1}^{m-(p+1)+2}\mathtt{0} \succ \cdots \succ \mathtt{0}^x\mathtt{1}^{m-(m+1)+2}\mathtt{0} = \mathtt{0}^x\mathtt{1}\mathtt{0},$$

we conclude that, letting $b = \mathrm{RangeBeg}(\mathtt{0}^x\mathtt{1}, T_A)$, for every $i \in [1\,..\,m-p+2]$, the suffix $T_A[\mathrm{SA}_{T_A}[b+i]\,..\,|T_A|]$ has $\mathtt{0}^x\mathtt{1}^i\mathtt{0}$ as a prefix, and $T_A[\mathrm{SA}_{T_A}[b+(m+3-p)]\,..\,|T_A|]$ has $\mathtt{0}^x\mathtt{1}^{m+2}$ as a prefix. Since

$\mathtt{0}^x\mathtt{1}^{m+2}$ has only a single occurrence in T_A (at position $j = |T_A| - m - x - 1$), we obtain that

$$\mathrm{PLCP}_{T_A}[j] = \mathrm{lcp}(\mathtt{0}^x\mathtt{1}^{m+2}, \mathtt{0}^x\mathtt{1}^{m-p+2}\mathtt{0}) = x + m - p + 2.$$

Consequently, letting $y = \mathrm{PLCP}_{T_A}[j]$, it holds $p = (x + m + 2) - y$. Recall that $\mathsf{pred}_A(x) = a_{p-1}$. Thus, letting $i = (x + m + 1) - y = (x + m + 1) - \mathrm{PLCP}_{T_A}[|T_A| - m - x - 1]$, we have $\mathsf{pred}_A(x) = a_i$, i.e., we have reduced the computation of the predecessor index to a PLCP query on T_A.

Using the above observation, we develop our lower bound as follows. Assume that we can answer PLCP queries on any $T \in \{\mathtt{0},\mathtt{1}\}^n$ in $t_{\mathrm{PLCP}}(|T|) = o(\log\log|T|)$ time using $\mathcal{O}(\delta(T)\log^c|T|)$ space (for some positive constant c). Using this structure, we obtain a structure for answering predecessor queries on A by storing $|T_A|$, m, and the PLCP structure for the string T_A.

To bound the space for the PLCP structure on T_A, note that the string T_A consists of exactly $2(m+1)$ equal-letter runs. By Observation 2.19 and Theorem 2.37(1), we obtain $\delta(T_A) = \mathcal{O}(z(T_A)) = \mathcal{O}(m)$. It remains to observe that $|T_A| = \mathcal{O}(m^3)$. The data structure for PLCP queries on T_A thus needs $\mathcal{O}(\delta(T_A)\log^c|T_A|) = \mathcal{O}(m\log^c(m^3)) = \mathcal{O}(m\log^{c+1}m) = \mathcal{O}(m\log^{\mathcal{O}(1)}m)$ space. Given this data structure, we can compute $i \in [0\,..\,m]$ satisfying $\mathsf{pred}_A(x) = a_i$ for any $x \in [1\,..\,m^2]$ (for other x, we can return the answer immediately) by issuing a PLCP query to compute $y = \mathrm{PLCP}_{T_A}[|T_A| - m - x - 1]$ in $\mathcal{O}(t_{\mathrm{PLCP}}(|T_A|)) = o(\log\log|T_A|) = o(\log\log(m^3)) = o(\log\log m)$ time, and returning $i = (x + m + 1) - y$.

We have thus obtained a data structure answering predecessor queries in $o(\log\log m)$ time and using $\mathcal{O}(m\log^{\mathcal{O}(1)}m)$ space. This contradicts the lower bound from [PT06] (see also Theorem 2.44). Thus, the data structure for PLCP queries we assumed at the beginning cannot exist.

4 Queries with $\Theta(\frac{\log n}{\log\log n})$ Complexity

4.1 Longest Common Prefix (LCP) Queries

4.1.1 Problem Definition

INDEXING FOR LONGEST COMMON PREFIX (LCP) QUERIES

Input: A string $T \in \Sigma^n$.
Output: A data structure that, given any index $i \in [1\,..\,n]$, returns $\mathrm{LCP}_T[i]$ (Definition 2.11).

4.1.2 Lower Bound

Lemma 4.1. *Let $A[1\,..\,n]$ be a nonempty array containing a permutation of $\{1, 2, \ldots, n\}$. Let*

$$T = \Big(\bigodot_{i=1}^{n}(\mathtt{0}^{A[i]}\mathtt{1}^i)\Big)\cdot\mathtt{0}^{n+1}\mathtt{1}^{n+1} \in \{\mathtt{0},\mathtt{1}\}^*$$

(brackets added for clarity). Then, for every $v \in [1\,..\,n]$ and every $r \in [1\,..\,\mathsf{range\text{-}count}_A(n, v)]$ (see Definition 2.48), it holds

$$\mathsf{range\text{-}select}_A(r, v) = \mathrm{LCP}_T[b + r + 1] - v,$$

where $b = \mathrm{RangeBeg}(\mathtt{0}^v\mathtt{1}, T)$ (Definition 2.1).

Proof. Denote $k = \mathsf{range\text{-}count}_A(n, v)$. Let $(a_i)_{i\in[1\,..\,k+1]}$ be a sequence such that, for every $i \in [1\,..\,k]$,

$$a_i = \mathsf{range\text{-}select}_A(i, v),$$

and $a_{k+1} = n + 1$. By definition, we then have $a_1 < a_2 < \cdots < a_k < a_{k+1}$ and $\mathsf{range\text{-}select}_A(r, v) = a_r$. Next, let $(s_i)_{i\in[1\,..\,k+1]}$ be a sequence defined such that, for every $i \in [1\,..\,k]$,

$$s_i = \Big(\textstyle\sum_{t=1}^{a_i-1}(A[t] + t)\Big) + \Big(A[a_i] - v + 1\Big),$$

and $s_{k+1} = \big(\sum_{t=1}^{n}(A[t]+t)\big) + (n+2-v)$. Observe, that there is exactly $k+1$ occurrences of the substring $\mathtt{0}^v\mathtt{1}$ in T, and it holds $\mathrm{Occ}(\mathtt{0}^v\mathtt{1}, T) = \{s_i\}_{i\in[1\,..\,k+1]}$. Note also that for every $i \in [1\,..\,k]$, the suffix $T[s_i\,..\,|T|]$ has

the string $\mathtt{0}^v\mathtt{1}^{a_i}\mathtt{0}$ as a prefix. Moreover, $T[s_{k+1}\,..\,|T|]$ has $\mathtt{0}^v\mathtt{1}^{n+1}$ as a prefix. By $a_1 < a_2 < \cdots < a_k < a_{k+1}$, we thus have

$$T[s_1\,..\,|T|] \prec T[s_2\,..\,|T|] \prec \cdots \prec T[s_k\,..\,|T|] \prec T[s_{k+1}\,..\,|T|].$$

Consequently, by definition of $b = \mathrm{RangeBeg}(\mathtt{0}^v\mathtt{1}, T)$ and Remark 2.3, for every $i \in [1\,..\,k+1]$, it holds

$$\mathrm{SA}_T[b+i] = s_i.$$

Lastly, note that $\mathtt{0}^v\mathtt{1}^{a_i}\mathtt{0}$ being a prefix of $T[s_i\,..\,|T|]$ for $i \in [1\,..\,k]$, and $\mathtt{0}^v\mathtt{1}^{n+1}$ being a prefix of $T[s_{k+1}\,..\,|T|]$ imply that, for every $i \in [1\,..\,k]$, it holds $\mathrm{LCE}_T(s_i, s_{i+1}) = v + a_i$. Putting everything together, we thus obtain

$$\begin{aligned}
\mathrm{LCP}_T[b+r+1] &= \mathrm{LCE}_T(\mathrm{SA}_T[b+r], \mathrm{SA}_T[b+r+1]) \\
&= \mathrm{LCE}_T(s_r, s_{r+1}) \\
&= v + a_r \\
&= v + \mathsf{range\text{-}select}_A(r, v).
\end{aligned}$$

which is equivalent to the claim. □

Theorem 4.2. *There is no data structure that, for every text $T \in \{\mathtt{0},\mathtt{1}\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers LCP array queries on T (that, given any $i \in [1\,..\,n]$, return $\mathrm{LCP}_T[i]$; see Definition 2.11) in $o(\frac{\log n}{\log\log n})$ time.*

Proof. Suppose that the claim does not hold. Let D_{LCP} denote the hypothetical data structure answering LCP queries, and let $c = \mathcal{O}(1)$ be a positive constant such that, that for a text $T \in \{\mathtt{0},\mathtt{1}\}^n$, D_{LCP} uses $\mathcal{O}(\delta(T)\log^c n)$ space, and answers LCP queries on T in $t_{\mathrm{LCP}}(|T|) = o(\frac{\log|T|}{\log\log|T|})$ time. We will prove that this implies that there exists a data structure answering range selection queries that contradicts Theorem 2.54.

Consider any array $A[1\,..\,n]$ containing a permutation of $\{1, 2, \ldots, n\}$, where $n \geq 1$. Let $T_A = (\bigodot_{i=1}^{n}(\mathtt{0}^{A[i]}\mathtt{1}^i)) \cdot (\mathtt{0}^{n+1}\mathtt{1}^{n+1}) \in \{\mathtt{0},\mathtt{1}\}^*$ be a text defined as in Lemma 4.1. Let $R[1\,..\,n]$ be array defined so that, for every $i \in [1\,..\,n]$, $R[i] = \mathrm{RangeBeg}(\mathtt{0}^i\mathtt{1}, T_A)$ (Definition 2.1).

Let D_{select} denote the data structure consisting of the following components:

1. The array $R[1\,..\,n]$ stored in plain form. It uses $\mathcal{O}(n)$ space.
2. The data structure D_{LCP} for the string T_A. Note that

$$|T_A| = (\textstyle\sum_{i=1}^{n}(A[i]+i)) + 2(n+1) = (n+2)(n+1) \in \mathcal{O}(n^2).$$

   Moreover, note that $|\mathrm{RL}(T_A)| = 2(n+1)$. By applying Observation 2.19 and Theorem 2.37(1), we thus obtain $\delta(T_A) = \mathcal{O}(n)$. Consequently, D_{LCP} for T_A needs $\mathcal{O}(\delta(T_A)\log^c|T_A|) = \mathcal{O}(n\log^c n)$ space.

In total, D_{select} needs $\mathcal{O}(n\log^c n)$ space.

Given any $v \geq 0$ and $r \in [1\,..\,\mathsf{range\text{-}count}_A(n, v)]$, we compute $\mathsf{range\text{-}select}_A(r, v)$ as follows:

1. Note that since $A[1\,..\,n]$ contains a permutation of $\{1, \ldots, n\}$, the assumption $r \in [1\,..\,\mathsf{range\text{-}count}_A(n, v)]$ implies that $v \leq n$ (otherwise, $\mathsf{range\text{-}count}_A(n, v) = 0$). Additionally, note that if $v = 0$, then we can set $v = 1$ without changing the answer. Let us thus assume that $v \in [1\,..\,n]$. First, using array R, in $\mathcal{O}(1)$ time we compute $b = \mathrm{RangeBeg}(\mathtt{0}^v\mathtt{1}, T_A) = R[v]$.
2. Using D_{LCP}, in $\mathcal{O}(t_{\mathrm{LCP}}(|T_A|))$ time we compute $\ell = \mathrm{LCP}_{T_A}[b+r+1]$. By Lemma 4.1, it holds $\mathsf{range\text{-}select}_A(r, v) = \ell - v$. Thus, we return $\ell - v$ as the answer.

In total, the query time is

$$\mathcal{O}(t_{\mathrm{LCP}}(|T_A|)) = o\Big(\tfrac{\log|T_A|}{\log\log|T_A|}\Big) = o\Big(\tfrac{\log(n^2)}{\log\log(n^2)}\Big) = o\Big(\tfrac{\log n}{\log\log n}\Big).$$

We have thus proved that there exists a data structure that, for every array $A[1\,..\,n]$ containing a permutation of $\{1, \ldots, n\}$, uses $\mathcal{O}(n\log^c n) = \mathcal{O}(n\log^{\mathcal{O}(1)} n)$ space, and answers range selection queries on A in $o(\frac{\log n}{\log\log n})$ time. This contradicts Theorem 2.54. □

4.2 Suffix Array (SA) Queries

4.2.1 Problem Definition

INDEXING FOR SUFFIX ARRAY (SA) QUERIES

Input: A string $T \in \Sigma^n$.
Output: A data structure that, given any index $i \in [1\,..\,n]$, returns $\mathrm{SA}_T[i]$ (Definition 2.2).

4.2.2 Lower Bound

Theorem 4.3. *There is no data structure that, for every text $T \in \{0,1\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers suffix array queries on T (that, given any $i \in [1\,..\,n]$, return $\mathrm{SA}_T[i]$; see Definition 2.2) in $o(\frac{\log n}{\log\log n})$ time.*

Proof. Suppose that the claim does not hold. Let D_{SA} denote the hypothetical data structure answering suffix array queries, and let $c = \mathcal{O}(1)$ be a positive constant such that, that for a text $T \in \{0,1\}^n$, D_{SA} uses $\mathcal{O}(\delta(T)\log^c n)$ space, and answers suffix array queries on T in $t_{\mathrm{SA}}(|T|) = o(\frac{\log |T|}{\log\log |T|})$ time. We will prove that this implies that there exists a data structure answering LCP array queries that contradicts Theorem 4.2.

Let $T \in \{0,1\}^n$ be a nonempty text.

Let D_{LCP} denote the data structure consisting of the following components:

1. The data structure from Theorem 2.30 for text T. It uses $\mathcal{O}(r(T)) = \mathcal{O}(\delta(T)\log^2 n)$ space (see Theorem 2.35 and Remark 2.36).
2. The data structure D_{SA} for the string T. It uses $\mathcal{O}(\delta(T)\log^c n)$ space.

In total, D_{LCP} needs $\mathcal{O}(\delta(T)\log^c n + \delta(T)\log^2 n)$ space.

Given any $i \in [1\,..\,n]$, we compute $\mathrm{LCP}_T[i]$ (Definition 2.11) as follows:

1. Using D_{SA}, in $\mathcal{O}(t_{\mathrm{SA}}(n))$ time we compute $j = \mathrm{SA}_T[i]$.
2. Using the structure from Theorem 2.30, in $\mathcal{O}(\log\log n)$ time we compute and return $x = \mathrm{PLCP}_T[j]$. Note that by Definition 2.12, it holds $x = \mathrm{PLCP}_T[j] = \mathrm{PLCP}_T[\mathrm{SA}_T[i]] = \mathrm{LCP}_T[i]$.

In total, the query time is

$$\mathcal{O}(\log\log n + t_{\mathrm{SA}}(n)) = \mathcal{O}(\log\log n) + o\Big(\tfrac{\log n}{\log\log n}\Big) = o\Big(\tfrac{\log n}{\log\log n}\Big).$$

We have thus proved that there exists a data structure that, for every text $T \in \{0,1\}^n$ uses $\mathcal{O}(\delta(T)\log^c n + \delta(T)\log^2 n) = \mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space, and answers LCP array queries on T in $o(\frac{\log n}{\log\log n})$ time. This contradicts Theorem 4.2. □

4.3 Inverse Suffix Array (SA^{-1}) Queries

4.3.1 Problem Definition

INDEXING FOR INVERSE SUFFIX ARRAY (SA^{-1}) QUERIES

Input: A string $T \in \Sigma^n$.
Output: A data structure that, given any index $j \in [1\,..\,n]$, returns $\mathrm{SA}_T^{-1}[j]$ (Definition 2.5).

4.3.2 Lower Bound

Observation 4.4. *Let $T \in \{0,1\}^n$ be a nonempty text. Denote $n_0 = |\{j \in [1\,..\,n] : T[j] = 0\}|$. For every $j \in [1\,..\,n]$, $T[j] = 0$ holds if and only if $\mathrm{SA}_T^{-1}[j] \le n_0$ (Definition 2.5).*

Theorem 4.5. *There is no data structure that, for every text $T \in \{0,1\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers inverse SA queries on T (that is, given any $i \in [1\,..\,n]$, returns $\mathrm{SA}_T^{-1}[i]$; see Definition 2.5) in $o(\frac{\log n}{\log\log n})$ time.*

Proof. Suppose the claim does not hold. Let D_{ISA} denote the hypothetical data structure answering inverse SA queries, and let $c = \mathcal{O}(1)$ be a positive constant such that for a text $T \in \{\mathtt{0},\mathtt{1}\}^n$, D_{ISA} uses $\mathcal{O}(\delta(T)\log^c n)$ space, and answers inverse SA queries on T in time $t_{\mathrm{ISA}}(|T|) = o(\frac{\log|T|}{\log\log|T|})$. We will show that this implies the existence of a data structure answering random access queries, contradicting Theorem 2.38.

Let $T \in \{\mathtt{0},\mathtt{1}\}^n$, where $n \geq 1$, and let $n_0 = |\{j \in [1\,..\,n] : T[j] = \mathtt{0}\}|$.

Let D_{access} denote the data structure consisting of the following components:

1. The integer n_0, stored in $\mathcal{O}(1)$ space.
2. The data structure D_{ISA} for the string T, using $\mathcal{O}(\delta(T)\log^c n)$ space.

In total, D_{access} uses $\mathcal{O}(\delta(T)\log^c n)$ space.

Given any $j \in [1\,..\,n]$, we compute $T[j]$ as follows:

1. Use D_{ISA} to compute $i = \mathrm{SA}_T^{-1}[j]$ in $\mathcal{O}(t_{\mathrm{ISA}}(n))$ time.
2. Return $T[j] = \mathtt{0}$ if $i \leq n_0$ and $T[j] = \mathtt{1}$ otherwise. This is correct by Observation 4.4.

The total query time is $\mathcal{O}(t_{\mathrm{ISA}}(n)) = o(\frac{\log n}{\log\log n})$.

We have therefore constructed a data structure that, for every $T \in \{\mathtt{0},\mathtt{1}\}^n$, uses $\mathcal{O}(\delta(T)\log^c n) = \mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers random access queries in $o(\frac{\log n}{\log\log n})$ time. This contradicts Theorem 2.38. □

4.4 Longest Common Extension (LCE) Queries

4.4.1 Problem Definition

> Indexing for Longest Common Extension (LCE) Queries
>
> **Input:** A string $T \in \Sigma^n$.
> **Output:** A data structure that, given any $i, j \in [1\,..\,n]$, returns $\mathrm{LCE}_T(i,j)$ (Section 2.1).

4.4.2 Lower Bound

Lemma 4.6. *Let Σ be a nonempty set. For every nonempty text $T \in \Sigma^+$ and every $c \in \Sigma$, it holds $\delta(T') \leq \delta(T) + 1$ (see Definition 2.32), where $T' = T \cdot c$.*

Proof. Appending a symbol to T creates at most one new substring of any given length, i.e., for every $\ell \in \mathbb{Z}_{>0}$, it holds $\mathrm{d}_\ell(T') \leq \mathrm{d}_\ell(T) + 1$. This implies that

$$\begin{aligned}
\delta(T') &= \max_{\ell=1}^{|T|+1} \tfrac{1}{\ell}\mathrm{d}_\ell(T') \\
&\leq \max_{\ell=1}^{|T|+1} \tfrac{1}{\ell}(\mathrm{d}_\ell(T) + 1) \\
&\leq \max_{\ell=1}^{|T|+1} (\tfrac{1}{\ell}\mathrm{d}_\ell(T) + 1) \\
&= 1 + \max_{\ell=1}^{|T|+1} \tfrac{1}{\ell}\mathrm{d}_\ell(T) \\
&= 1 + \max_{\ell=1}^{|T|} \tfrac{1}{\ell}\mathrm{d}_\ell(T) \\
&= 1 + \delta(T).
\end{aligned}$$

□

Theorem 4.7. *There is no data structure that, for every text $T \in \{\mathtt{0},\mathtt{1}\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers LCE queries on T (that, given any $i, j \in [1\,..\,n]$, return $\mathrm{LCE}_T(i,j)$; see Section 2.1) in $o(\frac{\log n}{\log\log n})$ time.*

Proof. Suppose that the claim does not hold. Let D_{LCE} denote the hypothetical data structure answering LCE queries, and let $c = \mathcal{O}(1)$ be a positive constant such that, that for a text $T \in \{\mathtt{0},\mathtt{1}\}^n$, D_{LCE} uses $\mathcal{O}(\delta(T)\log^c n)$ space, and answers LCE queries on T in $t_{\mathrm{LCE}}(|T|) = o(\frac{\log|T|}{\log\log|T|})$ time. We will prove that this implies that there exists a data structure answering random access queries to T that contradicts Theorem 2.38.

Let $T \in \{\mathtt{0},\mathtt{1}\}^n$ be a nonempty text. Denote $T' = \mathtt{0} \cdot T \in \{\mathtt{0},\mathtt{1}\}^{n+1}$.

Let D_{access} denote the data structure consisting of a single component: the data structure D_{LCE} for the string T'. Note that $|T'| = n + 1$. Next, we give an upper bound for $\delta(T')$. To this end, note that, by

Definition 2.32, for every string S, it holds $\delta(S) = \delta(\overline{S})$ (where $\overline{S}$ denotes the reverse of S; see Section 2.1). By Lemma 4.6, we thus have

$$\delta(T') = \delta(\overline{T'}) = \delta(\overline{T} \cdot \mathtt{0}) = 1 + \delta(\overline{T}) = 1 + \delta(T).$$

Thus, D_{LCE} for T' (and hence also D_{access}) needs $\mathcal{O}(\delta(T') \log^c |T'|) = \mathcal{O}(\delta(T) \log^c n)$ space.

Given any $j \in [1\,..\,n]$, we compute $T[j]$ as follows:

1. Using D_{LCE}, in $\mathcal{O}(t_{\mathrm{LCE}}(|T'|))$ time we compute $\ell = \mathrm{LCE}_{T'}(1, j+1)$.
2. If $\ell \geq 1$, we return $T[j] = \mathtt{0}$. Otherwise, we return that $T[j] = \mathtt{1}$.

In total, the query time is

$$\mathcal{O}(t_{\mathrm{LCE}}(|T'|)) = o\Big(\tfrac{\log |T'|}{\log\log |T'|}\Big) = o\Big(\tfrac{\log(n+1)}{\log\log(n+1)}\Big) = o\Big(\tfrac{\log n}{\log\log n}\Big).$$

We have thus proved that there exists a data structure that, for every $T \in \{\mathtt{0},\mathtt{1}\}^n$ uses $\mathcal{O}(\delta(T) \log^c n) = \mathcal{O}(\delta(T) \log^{\mathcal{O}(1)} n)$ space, and answers random access queries on T in $o(\frac{\log n}{\log\log n})$ time. This contradicts Theorem 2.38. □

4.4.3 Upper Bound

Theorem 4.8. *Let $\epsilon \in (0,1)$ be a constant. For every nonempty text $T \in \Sigma^n$, there exists a data structure of size $\mathcal{O}(r(T) \log^{2+\epsilon} n)$ (see Definition 2.26) that answers LCE queries (that, given any $i, j \in [1\,..\,n]$, return $\mathrm{LCE}_T(i,j)$; see Section 2.1) in $\mathcal{O}(\frac{\log n}{\log\log n})$ time.*

Proof. The data structure consists of the following components:

1. The data structure from Theorem B.8 for text T. It needs $\mathcal{O}(r(T) \log^{2+\epsilon} n)$ space.
2. The data structure from Theorem 2.28 for text T. It needs $\mathcal{O}(r(T) \log^{1+\epsilon} n)$ space.

In total, the data structure needs $\mathcal{O}(r(T) \log^{2+\epsilon} n)$ space.

Implementation of queries Let $i, j \in [1\,..\,n]$. To compute $\mathrm{LCE}_T(i,j)$, we proceed as follows:

1. If $i = j$, then we return that $\mathrm{LCE}_T(i,j) = n - i + 1$. Let us thus assume that $i \neq j$.
2. Using Theorem 2.28, in $\mathcal{O}(\frac{\log n}{\log\log n})$ time we compute $b := \mathrm{SA}_T^{-1}[i]$ and $e = \mathrm{SA}_T^{-1}[j]$. If $b > e$, then we swap b and e. We then set $b := b - 1$.
3. Using Theorem B.8, in $\mathcal{O}(\frac{\log n}{\log\log n})$ time we compute $i_{\min} = \arg\min_{t \in (b\,..\,e]} \mathrm{LCP}_T[t]$.
4. Using Theorem 2.28, in $\mathcal{O}(\frac{\log n}{\log\log n})$ time we compute and return $\ell = \mathrm{LCP}_T[i_{\min}]$. By definition of suffix and LCP array, it holds $\ell = \mathrm{LCE}_T(i,j)$.

In total, the query algorithm takes $\mathcal{O}(\frac{\log n}{\log\log n})$ time. □

Theorem 4.9. *Let $\epsilon \in (0,1)$ be a constant. For every nonempty text $T \in \Sigma^n$, there exists a data structure of size $\mathcal{O}(\delta(T) \log^{4+\epsilon} n)$ (see Definition 2.32) that answers LCE queries (that, given any $i, j \in [1\,..\,n]$, return $\mathrm{LCE}_T(i,j)$; see Section 2.1) in $\mathcal{O}(\frac{\log n}{\log\log n})$ time.*

Proof. The result follows by combining Theorem 4.8 and Theorem 2.35 (see also Remark 2.36). □

5 Queries with $\Theta(\log\log n)$ Complexity

5.1 Burrows–Wheeler Transform (BWT) Queries

5.1.1 Problem Definition

Indexing for Burrows–Wheeler Transform (BWT) Queries

Input: A string $T \in \Sigma^n$.
Output: A data structure that, given any index $i \in [1\,..\,n]$, returns $\mathrm{BWT}_T[i]$ (Definition 2.26).

5.1.2 Lower Bound

Definition 5.1 (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).

Definition 5.2 (Extended binary mapping)**.** Let $k \in \mathbb{Z}_{>0}$. For every $x \in [0\,..\,2^k)$, define

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

where $\mathrm{bin}_k(x)$ is as in Definition 5.1.

Lemma 5.3. *Let $A \subseteq [1\,..\,m]$ be a nonempty set of size $|A| = m$. Denote $A = \{a_1, a_2, \ldots, a_m\}$, where $a_1 < a_2 < \cdots < a_m$, and let $a_0 = 0$ and $a_{m+1} = m^2$. Let $(b_i)_{i=0}^m$ be a sequence defined by $b_i = i \bmod 2$. Let (see Definition 5.2)*

$$T = \bigodot_{i=0}^{m} \Big(b_i \cdot \mathrm{ebin}_k(i)\Big)^{a_{i+1}-a_i} \in \{0,1\}^*$$

(brackets added for clarity), where $k = 1 + \lfloor \log m \rfloor$. Then, for every $x \in [1\,..\,m^2]$, it holds (see Definitions 2.26 and 2.45)

$$\mathsf{pred\text{-}color}_A(x) = \mathrm{BWT}_T[b + x],$$

where $b = \mathrm{RangeBeg}(1^{k+1}0, T)$ (see Definition 2.1).

Proof. Denote

$$S = b_0^{a_1-a_0} \cdot b_1^{a_2-a_1} \cdot b_2^{a_3-a_2} \cdot \ldots \cdot b_{m-1}^{a_m-a_{m-1}} \cdot b_m^{a_{m+1}-a_m} \in \{0,1\}^{m^2}.$$

The proof consists of two steps:

1. First, we prove that it holds $\mathrm{BWT}_T(b\,..\,b+m^2] = S$. Let $\delta = 2k+4$. For every $i \in [0\,..\,m]$, we define

$$P_i = \{\delta \cdot t - (2k+2) : t \in (a_i\,..\,a_{i+1}]\}.$$

Let also $P = \bigcup_{i=0}^m P_i = \{\delta \cdot t - (2k+2) : t \in [1\,..\,m^2]\}$. Observe that, it holds

$$\mathrm{Occ}(1^{k+1}0, T) = P.$$

On the other hand, note that since for every $i \in [0\,..\,m]$ and every $x \in P_i$, the suffix $T[x\,..\,|T|]$ has the string $1^{k+1} \cdot 0 \cdot \mathrm{bin}_k(i)$ as a prefix. This implies that if $i_1, i_2 \in [0\,..\,m]$ are such that $i_1 < i_2$, then for every $x_1 \in P_{i_1}$ and $x_2 \in P_{i_2}$, it holds $T[x_1\,..\,|T|] \prec T[x_2\,..\,|T|]$. Recall now that, by $b = \mathrm{RangeBeg}(1^{k+1}0, T)$ and Definition 2.1, it holds $\{\mathrm{SA}_T[b+t] : t \in [1\,..\,m^2]\} = P$. Combining this with the above observation about lexicographical order of suffixes starting at positions in P_{i_1} and P_{i_2}, we thus obtain that, for every $i \in [0\,..\,m]$, it holds

$$\{\mathrm{SA}_T[b+t] : t \in (a_i\,..\,a_{i+1}]\} = P_i.$$

Since for every $i \in [0\,..\,m]$ and $x \in P_i$, we have $T[x-1] = b_i$, it follows by Definition 2.26 that, for every $i \in [0\,..\,m]$,

$$\mathrm{BWT}_T(b + a_i\,..\,b + a_{i+1}] = b_i^{a_{i+1}-a_i}.$$

Hence, we obtain that it holds $\mathrm{BWT}_T(b\,..\,b+m^2] = b_0^{a_1-a_0} \cdot b_1^{a_2-a_1} \cdot \ldots \cdot b_m^{a_{m+1}-a_m} = S$.

2. In the second step, we observe that, by Definition 2.45, for every $x \in [1\,..\,m^2]$, it holds

$$\mathsf{pred\text{-}color}_A(x) = S[x].$$

Combining this observation with the above step, we thus obtain the claim. □

Theorem 5.4. *There is no data structure that, for every text $T \in \{0,1\}^n$, uses $\mathcal{O}(\delta(T) \log^{\mathcal{O}(1)} n)$ space and answers BWT queries on T (that, given any $i \in [1\,..\,n]$, return $\mathrm{BWT}_T[i]$; see Definition 2.26) in $o(\log \log n)$ time.*

Proof. Suppose that the claim does not hold. Let D_{BWT} denote the hypothetical data structure answering BWT queries, and let $c = \mathcal{O}(1)$ be a positive constant such that, that for a text $T \in \{0,1\}^n$, D_{BWT} uses $\mathcal{O}(\delta(T)\log^c n)$ space, and answers BWT queries on T in $t_{\mathrm{BWT}}(|T|) = o(\log\log|T|)$ time. We will prove that this implies that there exists a data structure answering colored predecessor queries, that contradicts Theorem 2.47.

Let $m \geq 1$ and let $A \subseteq [1\,..\,m^2]$ be a nonempty set of size m. Denote $A = \{a_1, a_2, \ldots, a_m\}$, where $a_1 < a_2 < \cdots < a_m$, and let $a_0 = 0$ and $a_{m+1} = m^2$. Let $(b_i)_{i=0}^m$ be a sequence defined by $b_i = i \bmod 2$. Let $k = 1 + \lfloor \log m \rfloor$. Let $T_A = \bigodot_{i=0}^m (b_i \cdot \mathrm{ebin}_k(i))^{a_{i+1}-a_i} \in \{0,1\}^*$ (see Definition 5.2) be a text defined as in Lemma 5.3. Finally, let $b = \mathrm{RangeBeg}(1^{k+1}0, T)$.

Let D_{pred} denote the data structure consisting of the following components:

1. The integer b stored in $\mathcal{O}(1)$ space.
2. The data structure D_{BWT} for the string T_A. Note that

$$|T_A| = \textstyle\sum_{i=0}^m (2k+4)\cdot(a_{i+1}-a_i) = (2k+4)\cdot\sum_{i=0}^m (a_{i+1}-a_i) = (2k+4)\cdot m^2 \in \mathcal{O}(m^2\log m).$$

   Next, observe that for every $i \in [0\,..\,m]$, the substring $(b_i \cdot \mathrm{ebin}_k(i))^{a_{i+1}-a_i}$ of T_A has an LZ77-like factorization consisting of $2k+5$ phrases: the first $2k+4$ phrases are of length one, and the remaining suffix is encoded with a single phrase. Thus, T_A has an LZ77-like factorization of size $(m+1)\cdot(2k+5)$. By Theorem 2.18, we thus obtain $z(T_A) \leq (m+1)\cdot(2k+5) = \mathcal{O}(m\log m)$. By Theorem 2.37(1), this in turn implies that $\delta(T_A) = \mathcal{O}(z(T_A)) = \mathcal{O}(m\log m)$. Consequently, D_{BWT} for T_A needs $\mathcal{O}(\delta(T_A)\log^c|T_A|) = \mathcal{O}(m\log^{c+1} m)$ space.

In total, D_{pred} needs $\mathcal{O}(m\log^{c+1} m)$ space.

Given any $x \in \mathbb{Z}$, we compute $\mathsf{pred\text{-}color}_A(x)$ (Definition 2.45) as follows:

1. If $x < 1$ (resp. $x > m^2$), we return $\mathsf{pred\text{-}color}_A(x) = 0$ (resp. $\mathsf{pred\text{-}color}_A(x) = m \bmod 2$) in $\mathcal{O}(1)$ time, and conclude the query algorithm. Let us thus assume that $x \in [1\,..\,m^2]$.
2. Using D_{BWT}, in $\mathcal{O}(t_{\mathrm{BWT}}(|T_A|))$ time we compute and return $c = \mathrm{BWT}_{T_A}[b+x]$ as the answer. By Lemma 5.3, it holds $c = \mathsf{pred\text{-}color}_A(x)$.

In total, the query takes $\mathcal{O}(t_{\mathrm{BWT}}(|T_A|)) = o(\log\log|T_A|) = o(\log\log(m^2\log m)) = o(\log\log m)$ time.

We have thus proved that there exists a data structure that, for every set $A \subseteq [1\,..\,m^2]$ of size $|A| = m$, uses $\mathcal{O}(m\log^{c+1} m) = \mathcal{O}(m\log^{\mathcal{O}(1)} m)$ space, and answers colored predecessor queries on A in $o(\log\log m)$ time. This contradicts Theorem 2.47. □

5.2 Permuted Longest Common Prefix (PLCP) Queries

5.2.1 Problem Definition

> Indexing for Permuted Longest Common Prefix (PLCP) Queries
>
> **Input:** A string $T \in \Sigma^n$.
> **Output:** A data structure that, given any index $j \in [1\,..\,n]$, returns $\mathrm{PLCP}_T[j]$ (Definition 2.12).

5.2.2 Lower Bound

Lemma 5.5. *Let $A \subseteq [1\,..\,m^2]$ be a nonempty set of size $|A| = m$. Denote $A = \{a_1, a_2, \ldots, a_m\}$, where $a_1 < a_2 < \cdots < a_m$, and let $a_0 = -\infty$. Let*

$$T = \left(\bigodot_{i=1}^m (0^{a_i}1^{m-i+2})\right)\cdot\left(0^{m^2+1}1\right)\cdot\left(0^{m^2}1^{m+2}\right) \in \{0,1\}^*$$

(brackets added for clarity). Then, for every $x \in [1\,..\,m^2]$, it holds (see Definition 2.40)

$$\mathsf{pred}_A(x) = a_i,$$

where $\Delta = |T| - (m^2+m+2)$ and $i = (x+m+1) - \mathrm{PLCP}_T[\Delta + m^2 - x + 1]$ (Definition 2.12).

Proof. Denote $j = \Delta + m^2 - x + 1$ and let $a_{m+1} = m^2 + 1$. Let

$$p = \min\{t \in [1\,..\,m+1] : a_t \geq x\}.$$

Note that p is well-defined, since $a_{m+1} = m^2 + 1$. For every $t \in [p\,..\,m+1]$, let $s_t = (\sum_{u=1}^{t-1}(a_u + (m - u + 2))) + (a_t - x + 1)$. Observe that, by definition of T, it holds $|\mathrm{Occ}(\mathtt{0}^x\mathtt{1}, T)| = n - p + 3$ and, moreover,

$$\mathrm{Occ}(\mathtt{0}^x\mathtt{1}, T) = \{s_t\}_{t\in[p\,..\,n+1]} \cup \{j\}.$$

Next, observe that:

- $T[j\,..\,|T|]$ has $\mathtt{0}^x\mathtt{1}^{m+2}$ as a prefix and
- for every $t \in [p\,..\,m+1]$, $T[s_t\,..\,|T|]$ has $\mathtt{0}^x\mathtt{1}^{m-t+2}\mathtt{0}$ as a prefix.

This implies that it holds

$$T[s_{m+1}\,..\,|T|] \prec T[s_m\,..\,|T|] \prec \cdots \prec T[s_p\,..\,|T|] \prec T[j\,..\,|T|].$$

Consequently, it holds $\Phi_T[j] = s_p$, and hence $\mathrm{PLCP}_T[j] = \mathrm{LCE}_T(j, \Phi_T[j]) = \mathrm{LCE}_T(j, s_p) = (x + m + 2) - p$. Combining with $\mathsf{pred}_A(x) = a_{p-1}$ (following from the definition of p), we thus obtain

$$\begin{aligned} i &= (x + m + 1) - \mathrm{PLCP}_T[\Delta + m^2 - x + 1] \\ &= (x + m + 1) - \mathrm{PLCP}_T[j] \\ &= (x + m + 1) - ((x + m + 2) - p) \\ &= p - 1. \end{aligned}$$

Thus, indeed we have $\mathsf{pred}_A(x) = a_{p-1} = a_i$. □

Theorem 5.6. *There is no data structure that, for every text $T \in \{\mathtt{0},\mathtt{1}\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers PLCP queries on T (that, given any $j \in [1\,..\,n]$, return $\mathrm{PLCP}_T[j]$; see Definition 2.12) in $o(\log\log n)$ time.*

Proof. Suppose that the claim does not hold. Let D_{PLCP} denote the hypothetical data structure answering PLCP queries, and let $c = \mathcal{O}(1)$ be a positive constant such that, that for a text $T \in \{\mathtt{0},\mathtt{1}\}^n$, D_{PLCP} uses $\mathcal{O}(\delta(T)\log^c n)$ space, and answers PLCP queries on T in $t_{\mathrm{PLCP}}(|T|) = o(\log\log|T|)$ time. We will prove that this implies that there exists a data structure answering predecessor queries, that contradicts Theorem 2.44.

Let $m \geq 1$ and let $A \subseteq [1\,..\,m^2]$ be a nonempty set of size m. Denote $A = \{a_1, a_2, \ldots, a_m\}$, where $a_1 < a_2 < \cdots < a_m$. Let $T_A = (\bigodot_{i=1}^{m}(\mathtt{0}^{a_i}\mathtt{1}^{m-i+2})) \cdot (\mathtt{0}^{m^2+1}\mathtt{1}) \cdot (\mathtt{0}^{m^2}\mathtt{1}^{m+2}) \in \{\mathtt{0},\mathtt{1}\}^*$ be a text defined as in Lemma 5.5. Denote $\Delta = |T_A| - (m^2 + m + 2)$. Let $A_{\mathrm{val}}[0\,..\,m]$ be an array such that $A_{\mathrm{val}}[0] = -\infty$ and, for every $i \in [1\,..\,m]$, $A_{\mathrm{val}}[i] = a_i$.

Let D_{pred} denote the data structure consisting of the following components:

1. The array $A_{\mathrm{val}}[0\,..\,m]$ stored in plain form. It needs $\mathcal{O}(m)$ space.
2. The integer Δ stored in $\mathcal{O}(1)$ space.
3. The data structure D_{PLCP} for the string T_A. Note that

$$|T_A| = \textstyle\sum_{i=1}^{m} a_i + \sum_{i=2}^{m+1} i + 2m^2 + m + 4 \leq m^3 + \frac{5}{2}m^2 + \frac{3}{2}m + 4 \in \mathcal{O}(m^3).$$

   Moreover, note that $|\mathrm{RL}(T_A)| = 2(m+2)$. By applying Observation 2.19 and Theorem 2.37(1), we thus obtain $\delta(T_A) = \mathcal{O}(m)$. Consequently, D_{PLCP} for T_A needs $\mathcal{O}(\delta(T_A)\log^c|T_A|) = \mathcal{O}(m\log^c m)$ space.

In total, D_{pred} needs $\mathcal{O}(m\log^c m)$ space.

Given any $x \in \mathbb{Z}$, we compute $\mathsf{pred}_A(x)$ as follows:

1. If $x < 1$ (resp. $x > m^2$), we return $\mathsf{pred}_A(x) = -\infty$ (resp. $\mathsf{pred}_A(x) = A_{\mathrm{val}}[m]$) in $\mathcal{O}(1)$ time, and conclude the query algorithm. Let us thus assume that $x \in [1\,..\,m^2]$.
2. Using D_{PLCP}, in $\mathcal{O}(t_{\mathrm{PLCP}}(|T_A|))$ time we compute $\ell = \mathrm{PLCP}_{T_A}[\Delta + m^2 - x + 1]$. We then let $i = m - (\ell - x) + 1$, and return $p = A_{\mathrm{val}}[i]$ as the answer. By Lemma 5.5, it holds $p = \mathsf{pred}_A(x)$.

In total, the query takes $\mathcal{O}(t_{\mathrm{PLCP}}(|T_A|)) = o(\log\log|T_A|) = o(\log\log(m^3)) = o(\log\log m)$ time.

We have thus proved that there exists a data structure that, for every set $A \subseteq [1\,..\,m^2]$ of size $|A| = m$, uses $\mathcal{O}(m\log^c m) = \mathcal{O}(m\log^{\mathcal{O}(1)} m)$ space, and answers predecessor queries on A in $o(\log\log m)$ time. This contradicts Theorem 2.44. □

5.3 Last-to-First (LF) Mapping Queries

5.3.1 Problem Definition

INDEXING FOR LAST-TO-FIRST (LF) MAPPING QUERIES

Input: A string $T \in \Sigma^n$.
Output: A data structure that, given any index $i \in [1\,..\,n]$, returns $\mathrm{LF}_T[i]$ (Definition 2.8).

5.3.2 Lower Bound

Observation 5.7. *Let $T \in \{0,1\}^n$, where $n \geq 1$. Denote $n_0 = |\{j \in [1\,..\,n] : T[j] = 0\}|$. For every $i \in [1\,..\,n]$, $\mathrm{BWT}_T[i] = 0$ (Definition 2.26) holds if and only if $\mathrm{LF}_T[i] \leq n_0$ (Definition 2.8).*

Proof. The proof consists of two steps.

1. First, we observe that $\mathrm{BWT}_T[i] = T[\mathrm{SA}_T[\mathrm{LF}_T[i]]]$. To see this, consider two cases:
   - If $\mathrm{SA}_T[i] = 1$, then by Definitions 2.8 and 2.26, we have $\mathrm{BWT}_T[i] = T[n]$ and $\mathrm{LF}_T[i] = \mathrm{SA}_T^{-1}[n]$. Putting the two together, we obtain $\mathrm{BWT}_T[i] = T[n] = T[\mathrm{SA}_T[\mathrm{LF}_T[i]]]$.
   - If $\mathrm{SA}_T[i] > 1$, then by Definitions 2.8 and 2.26, it holds $\mathrm{BWT}_T[i] = T[\mathrm{SA}_T[i]-1]$ and $\mathrm{SA}_T[\mathrm{LF}_T[i]] = \mathrm{SA}_T[i] - 1$. Thus, $\mathrm{BWT}_T[i] = T[\mathrm{SA}_T[i] - 1] = T[\mathrm{SA}_T[\mathrm{LF}_T[i]]]$.
2. By definition of the suffix array, for every $t \in [1\,..\,n]$, $T[\mathrm{SA}_T[t]] = 0$ holds if and only if $t \leq n_0$. Letting $t = \mathrm{LF}_T[i]$, we thus obtain the claim. □

Theorem 5.8. *There is no data structure that, for every text $T \in \{0,1\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers LF queries on T (that, given any $i \in [1\,..\,n]$, return $\mathrm{LF}_T[i]$; see Definition 2.8) in $o(\log\log n)$ time.*

Proof. Suppose that the claim does not hold. Let D_{LF} denote the hypothetical data structure answering LF queries, and let $c = \mathcal{O}(1)$ be a positive constant such that, that for a text $T \in \{0,1\}^n$, D_{LF} uses $\mathcal{O}(\delta(T)\log^c n)$ space, and answers LF queries on T in $t_{\mathrm{LF}}(|T|) = o(\log\log|T|)$ time. We will prove that this implies that there exists a data structure answering BWT queries contradicting Theorem 5.4.

Let $T \in \{0,1\}^n$, where $n \geq 1$. Let $n_0 = |\{j \in [1\,..\,n] : T[j] = 0\}|$.

Let D_{BWT} denote the data structure consisting of the following components:

1. The integer n_0 stored in $\mathcal{O}(1)$ space.
2. The data structure D_{LF} for the string T. If uses $\mathcal{O}(\delta(T)\log^c n)$. space.

In total, D_{BWT} needs $\mathcal{O}(\delta(T)\log^c n)$ space.

Given any $i \in [1\,..\,n]$, we compute $\mathrm{BWT}_T[i]$ (Definition 2.26) as follows:

1. First, in $\mathcal{O}(t_{\mathrm{LF}}(n))$ time we compute $i' = \mathrm{LF}_T[i]$.
2. We return that $\mathrm{BWT}_T[i] = 0$ (resp. $\mathrm{BWT}_T[i] = 1$) if $i' \leq n_0$ (resp. $i' > n_0$). This is correct by Observation 5.7.

In total, the query takes $\mathcal{O}(t_{\mathrm{LF}}(n)) = o(\log\log n)$ time.

We have thus proved that there exists a data structure that, for every $T \in \{0,1\}^n$, uses $\mathcal{O}(\delta(T)\log^c n) = \mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space, and answers BWT queries on the text T in $o(\log\log n)$ time. This contradicts Theorem 5.4. □

5.4 Inverse Last-to-First (LF^{-1}) Mapping Queries

5.4.1 Problem Definition

INDEXING FOR INVERSE LAST-TO-FIRST (LF^{-1}) MAPPING QUERIES

Input: A string $T \in \Sigma^n$.
Output: A data structure that, given any index $i \in [1\,..\,n]$, returns $\mathrm{LF}_T^{-1}[i]$ (Definition 2.9).

5.4.2 Lower Bound

Lemma 5.9. *Let $A \subseteq [1\,..\,m^2]$ be a nonempty set of size $|A| = m$. Denote $A = \{a_1, a_2, \ldots, a_m\}$, where $a_1 < a_2 < \cdots < a_m$, and let $a_0 = 0$ and $a_{m+1} = m^2$. Let $B = A \cup \{a_0\}$. Let (see Definition 5.2)*

$$T = \bigodot_{i=0}^{m} \Big(\mathtt{1} \cdot \mathrm{ebin}_k(i)\Big)^{a_{i+1}-a_i} \cdot \Big(\mathrm{ebin}_k(i)\Big)^{m^2-(a_{i+1}-a_i)} \in \{\mathtt{0},\mathtt{1}\}^*$$

(brackets added for clarity), where $k = 1 + \lfloor \log m \rfloor$. Then, for every $x \in [1\,..\,m^2]$, it holds (see Definition 2.40)

$$\mathsf{pred}_B(x) = a_i,$$

where $\alpha = \mathrm{RangeBeg}(\mathtt{1}^{k+2}\mathtt{0}, T)$ (Definition 2.1), $\beta = \mathrm{RangeBeg}(\mathtt{1}^{k+1}\mathtt{0}, T)$, and (see Definition 2.9)

$$i = \left\lceil \frac{\mathrm{LF}_T^{-1}[\alpha + x] - \beta}{m^2} \right\rceil - 1.$$

Proof. Denote $i_{\mathrm{arg}} = \alpha + x$. Let

$$p = \max\{t \in [0\,..\,m] : a_t < x\}.$$

Note that since $x \in [1\,..\,m^2]$ and $a_0 = 0$, it follows that p is well defined and it holds

$$\mathsf{pred}_B(x) = \mathsf{pred}_{A\cup\{a_0\}}(x) = a_p.$$

For every $t \in [0\,..\,m+1]$, denote $q_t = \beta + t \cdot m^2$ and $s_t = \alpha + a_t$. The proof proceeds in four steps:

1. First, we prove that, for $t \in [0\,..\,m]$, it holds $\mathrm{RangeBeg}(\mathrm{ebin}_k(t), T) = q_t$ and $\mathrm{RangeEnd}(\mathrm{ebin}_k(t), T) = q_{t+1}$. Observe that every occurrence of $\mathtt{1}^{k+1}\mathtt{0}$ in T coincides with an occurrence of $\mathrm{ebin}_k(t)$ for some $t \in [0\,..\,m]$. Formally, $\mathrm{Occ}(\mathtt{1}^{k+1}\mathtt{0}, T)$ is a disjoint union

   $$\mathrm{Occ}(\mathtt{1}^{k+1}\mathtt{0}, T) = \bigcup_{t=0}^{m} \mathrm{Occ}(\mathrm{ebin}_k(t), T).$$

   Next, note that for every $t \in [0\,..\,m]$, it holds $|\mathrm{Occ}(\mathrm{ebin}_k(t), T)| = m^2$. Finally, observe that by Definitions 5.1 and 5.2, it follows that for every $t_1, t_2 \in [0\,..\,m]$, $t_1 < t_2$ implies $\mathrm{ebin}_k(t_1) \prec \mathrm{ebin}_k(t_2)$. Since for all $t \in [0\,..\,m]$, $\mathrm{ebin}_k(t)$ is of the same length, it follows that for every $t_1, t_2 \in [0\,..\,m]$ such that $t_1 < t_2$, it holds $T[j_1\,..\,|T|] \prec T[j_2\,..\,|T|]$ for all $j_1 \in \mathrm{Occ}(\mathrm{ebin}_k(t_1), T)$ and $j_2 \in \mathrm{Occ}(\mathrm{ebin}_k(t_2), T)$. Consequently, the SA-interval $\mathrm{SA}_T(\beta\,..\,\beta + m^2]$ contains $\mathrm{Occ}(\mathrm{ebin}_k(0), T)$, $\mathrm{SA}_T(\beta + m^2\,..\,\beta + 2m^2]$ contains $\mathrm{Occ}(\mathrm{ebin}_k(1), T)$, etc. By definition of the sequence $(q_t)_{t\in[0\,..\,m+1]}$, we thus obtain that, for every $t \in [0\,..\,m]$, it holds $\mathrm{RangeBeg}(\mathrm{ebin}_k(t), T) = \beta + t \cdot m^2 = q_t$ and $\mathrm{RangeEnd}(\mathrm{ebin}_k(t), T) = \beta + (t+1) \cdot m^2 = q_{t+1}$.
2. In the second step, we prove that, for every $t \in [0\,..\,m]$, it holds $\mathrm{RangeEnd}(\mathtt{1} \cdot \mathrm{ebin}_k(t), T) = s_t$ and $\mathrm{RangeEnd}(\mathtt{1} \cdot \mathrm{ebin}_k(t), T) = s_{t+1}$. Observe that every occurrence of $\mathtt{1}^{k+2}\mathtt{0}$ in T coincides with an occurrence of $\mathtt{1} \cdot \mathrm{ebin}_k(t)$ for some $t \in [0\,..\,m]$. Formally, $\mathrm{Occ}(\mathtt{1}^{k+2}\mathtt{0}, T)$ is a disjoint union

   $$\mathrm{Occ}(\mathtt{1}^{k+2}\mathtt{0}, T) = \bigcup_{t=0}^{m} \mathrm{Occ}(\mathtt{1} \cdot \mathrm{ebin}_k(t), T).$$

   Next, note that for every $t \in [0\,..\,m]$, it holds $|\mathrm{Occ}(\mathtt{1} \cdot \mathrm{ebin}_k(t), T)| = a_{t+1} - a_t$. Finally, note that by the same argument as above, for every $t_1, t_2 \in [0\,..\,m]$ such that $t_1 < t_2$, it holds $T[j_1\,..\,|T|] \prec T[j_2\,..\,|T|]$ for all $j_1 \in \mathrm{Occ}(\mathtt{1} \cdot \mathrm{ebin}_k(t_1), T)$ and $j_2 \in \mathrm{Occ}(\mathtt{1} \cdot \mathrm{ebin}_k(t_2), T)$. Consequently, the SA-interval $\mathrm{SA}_T(\alpha + a_0\,..\,\alpha + a_1]$ contains $\mathrm{Occ}(\mathtt{1} \cdot \mathrm{ebin}_k(0), T)$, $\mathrm{SA}_T(\alpha + a_1\,..\,\alpha + a_2]$ contains $\mathrm{Occ}(\mathtt{1} \cdot \mathrm{ebin}_k(1), T)$, etc. By definition of the sequence $(s_t)_{t\in[0\,..\,m+1]}$, we thus obtain that, for every $t \in [0\,..\,m]$, it holds $\mathrm{RangeBeg}(\mathtt{1} \cdot \mathrm{ebin}_k(t), T) = \alpha + a_t = s_t$ $\mathrm{RangeEnd}(\mathtt{1} \cdot \mathrm{ebin}_k(t), T) = \alpha + a_{t+1} = s_{t+1}$.
3. In the third step, we prove that the suffix $T[\mathrm{SA}_T[i_{\mathrm{arg}}] + 1\,..\,|T|]$ has the string $\mathrm{ebin}_k(p)$ as a prefix. To this end, observe that, by definition of p, it holds $a_p < x \le a_{p+1}$. Consequently, it holds

   $$s_p = \alpha + a_p < \alpha + x \le \alpha + a_{p+1} = s_{p+1}.$$

   By the above, it holds $\mathrm{RangeBeg}(\mathtt{1} \cdot \mathrm{ebin}_k(p), T) = s_p$ and $\mathrm{RangeEnd}(\mathtt{1} \cdot \mathrm{ebin}_k(p), T) = s_{p+1}$. This implies that $\mathrm{SA}_T[\alpha + x] \in \mathrm{Occ}(\mathtt{1} \cdot \mathrm{ebin}_k(p), T)$, or equivalently, $\mathrm{SA}_T[i_{\mathrm{arg}}] \in \mathrm{Occ}(\mathtt{1} \cdot \mathrm{ebin}_k(p), T)$. Therefore, $T[\mathrm{SA}_T[i_{\mathrm{arg}}] + 1\,..\,|T|]$ has $\mathrm{ebin}_k(p)$ as a prefix.

4. We are now ready to finish the proof. By Definition 2.9, it holds $\mathrm{SA}_T[\mathrm{LF}_T^{-1}[i_{\mathrm{arg}}]] = \mathrm{SA}_T[i_{\mathrm{arg}}] + 1$ (note that here we utilize the fact that since $T[\mathrm{SA}_T[i_{\mathrm{arg}}] + 1\,..\,|T|]$ has $\mathrm{ebin}_k(p)$ as a prefix, we must have $\mathrm{SA}_T[i_{\mathrm{arg}}] < |T|$). We thus also have $T[\mathrm{SA}_T[\mathrm{LF}_T^{-1}[i_{\mathrm{arg}}]]\,..\,|T|] = T[\mathrm{SA}_T[i_{\mathrm{arg}}] + 1\,..\,|T|]$. Since above we proved that $T[\mathrm{SA}_T[i_{\mathrm{arg}}] + 1\,..\,|T|]$ has $\mathrm{ebin}_k(p)$ as a prefix, we thus obtain that $T[\mathrm{SA}_T[\mathrm{LF}_T^{-1}[i_{\mathrm{arg}}]]\,..\,|T|]$ has $\mathrm{ebin}_k(p)$ as a prefix as well. By the above characterization of values $\mathrm{RangeBeg}(\mathrm{ebin}_k(p), T)$ and $\mathrm{RangeEnd}(\mathrm{ebin}_k(p), T)$, it therefore follows that the index $\mathrm{LF}_T^{-1}[i_{\mathrm{arg}}]$ satisfies $q_p < \mathrm{LF}_T^{-1}[i_{\mathrm{arg}}] \leq q_{p+1}$. Plugging in the definition of q_p and q_{p+1}, we thus obtain that it holds

$$\beta + p \cdot m^2 < \mathrm{LF}_T^{-1}[i_{\mathrm{arg}}] \leq \beta + (p+1) \cdot m^2.$$

This is equivalent to $p \cdot m^2 < \mathrm{LF}_T^{-1}[i_{\mathrm{arg}}] - \beta \leq (p+1) \cdot m^2$, which implies that

$$\left\lceil \frac{\mathrm{LF}_T^{-1}[i_{\mathrm{arg}}] - \beta}{m^2} \right\rceil = p + 1.$$

Thus, by definition of i_{arg} and i we obtain

$$i = \left\lceil \frac{\mathrm{LF}_T^{-1}[\alpha + x] - \beta}{m^2} \right\rceil - 1 = \left\lceil \frac{\mathrm{LF}_T^{-1}[i_{\mathrm{arg}}] - \beta}{m^2} \right\rceil - 1 = p.$$

Since above we observed that $\mathsf{pred}_B(x) = a_p$, we thus obtain that $\mathsf{pred}_B(x) = a_p = a_i$, i.e., the claim. □

Theorem 5.10. *There is no data structure that, for every text $T \in \{\mathtt{0},\mathtt{1}\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers inverse LF queries on T (that, given any $j \in [1\,..\,n]$, return $\mathrm{LF}_T^{-1}[j]$; see Definition 2.9) in $o(\log\log n)$ time.*

Proof. Suppose that the claim does not hold. Let D_{ILF} denote the hypothetical data structure answering inverse LF queries, and let $c = \mathcal{O}(1)$ be a positive constant such that, that for a text $T \in \{\mathtt{0},\mathtt{1}\}^n$, D_{ILF} uses $\mathcal{O}(\delta(T)\log^c n)$ space, and answers inverse LF queries on T in $t_{\mathrm{ILF}}(|T|) = o(\log\log|T|)$ time. We will prove that this implies that there exists a data structure answering predecessor queries, that contradicts Theorem 2.44.

Let $m \geq 1$ and let $A \subseteq [1\,..\,m^2]$ be a nonempty set of size m. Let $k = 1 + \lfloor \log m \rfloor$. Denote $A = \{a_1, a_2, \ldots, a_m\}$, where $a_1 < a_2 < \cdots < a_m$. Let also $a_0 = 0$ and $a_{m+1} = m^2$. Let $T_A = \bigodot_{i=0}^{m} (\mathtt{1} \cdot \mathrm{ebin}_k(i))^{a_{i+1}-a_i} \cdot (\mathrm{ebin}_k(i))^{m^2-(a_{i+1}-a_i)} \in \{\mathtt{0},\mathtt{1}\}^*$ be a text defined as in Lemma 5.9. Denote $\alpha = \mathrm{RangeBeg}(\mathtt{1}^{k+2}\mathtt{0}, T)$ and $\beta = \mathrm{RangeBeg}(\mathtt{1}^{k+1}\mathtt{0}, T)$. Let $A_{\mathrm{val}}[1\,..\,m]$ be an array such that, for every $i \in [1\,..\,m]$, $A_{\mathrm{val}}[i] = a_i$.

Let D_{pred} denote the data structure consisting of the following components:

1. The array $A_{\mathrm{val}}[1\,..\,m]$ stored in plain form. It needs $\mathcal{O}(m)$ space.
2. The integers α and β stored in $\mathcal{O}(1)$ space.
3. The data structure D_{ILF} for the string T_A. Note that

$$\begin{aligned} |T_A| &= \textstyle\sum_{i=0}^{m} (2k+4) \cdot (a_{i+1} - a_i) + (2k+3) \cdot (m^2 - (a_{i+1} - a_i)) \\ &= \textstyle\sum_{i=0}^{m} (a_{i+1} - a_i) + \sum_{i=0}^{m} (2k+3) \cdot m^2 \\ &= m^2 + (2k+3)(m+1)m^2 \in \mathcal{O}(m^3 \log m). \end{aligned}$$

Next, observe that for every $i \in [0\,..\,m]$, the substring $(\mathtt{1} \cdot \mathrm{ebin}_k(i))^{a_{i+1}-a_i} \cdot (\mathrm{ebin}_k(i))^{m^2-(a_{i+1}-a_i)}$ of T_A has an LZ77-like factorization consisting of at most $4k+9$ phrases: the first $2k+4$ phrases of length one, then at most a single phrase corresponding to the substring $(\mathtt{1} \cdot \mathrm{ebin}_k(i))^{(a_{i+1}-a_i)-1}$, then $2k+3$ phrases of length one, and finally, at most a single phrase corresponding to the substring $(\mathrm{ebin}_k(i))^{(m^2-(a_{i+1}-a_i))-1}$. Thus, T_A has an LZ77-like factorization of size at most $(m+1) \cdot (4k+9)$. By Theorem 2.18, we thus obtain $z(T_A) \leq (m+1) \cdot (4k+9) = \mathcal{O}(m \log m)$. By Theorem 2.37(1), this in turn implies that $\delta(T_A) = \mathcal{O}(z(T_A)) = \mathcal{O}(m \log m)$. Consequently, D_{ILF} for T_A needs $\mathcal{O}(\delta(T_A)\log^c |T_A|) = \mathcal{O}(m \log^{c+1} m)$ space.

In total, D_{pred} needs $\mathcal{O}(m \log^{c+1} m)$ space.

Given any $x \in \mathbb{Z}$, we compute $\mathsf{pred}_A(x)$ as follows:

1. If $x < 1$ (resp. $x > m^2$), we return $\mathsf{pred}_A(x) = -\infty$ (resp. $\mathsf{pred}_A(x) = A_{\mathrm{val}}[m]$) in $\mathcal{O}(1)$ time, and conclude the query algorithm. Let us thus assume that $x \in [1\,..\,m^2]$.
2. Using D_{ILF}, in $\mathcal{O}(t_{\mathrm{ILF}}(|T_A|))$ time we compute $i = \lceil (\mathrm{LF}_{T_A}^{-1}[\alpha + x] - \beta)/m^2 \rceil - 1$. By Lemma 5.9, it holds $\mathsf{pred}_{A \cup \{a_0\}}(x) = a_i$. If $i = 0$, then we return $\mathsf{pred}_A(x) = -\infty$. Otherwise, we return $\mathsf{pred}_A(x) = a_i = A_{\mathrm{val}}[i]$.

In total, the query takes $\mathcal{O}(t_{\mathrm{ILF}}(|T_A|)) = o(\log\log|T_A|) = o(\log\log(m^3\log m)) = o(\log\log m)$ time.

We have thus proved that there exists a data structure that, for every set $A \subseteq [1\,..\,m^2]$ of size $|A| = m$, uses $\mathcal{O}(m\log^{c+1} m) = \mathcal{O}(m\log^{\mathcal{O}(1)} m)$ space, and answers predecessor queries on A in $o(\log\log m)$ time. This contradicts Theorem 2.44. □

5.4.3 Upper Bound

Lemma 5.11 ([GNP20, Lemma 5.2]). *Let $T \in \Sigma^n$ be a nonempty text such that $T[n]$ does not occur in $T[1\,..\,n)$. Then, for every $i \in [2\,..\,n]$ satisfying $\mathrm{BWT}_T[i-1] = \mathrm{BWT}_T[i]$ (Definition 2.26), it holds $\mathrm{LF}_T[i] = \mathrm{LF}_T[i-1] + 1$ (Definition 2.8).*

Lemma 5.12. *Let $T \in \Sigma^n$ be a nonempty text such that $T[n]$ does not occur in $T[1\,..\,n)$. Denote (see Definition 2.26)*

$$\mathcal{I} = \{i \in [2\,..\,n] : \mathrm{BWT}_T[i-1] \neq \mathrm{BWT}_T[i]\} \cup \{1\}.$$

Then, for every $j \in [1\,..\,n]$, it holds (see Definition 2.9)

$$\mathrm{LF}_T^{-1}[j] = \mathrm{LF}_T^{-1}[j_{\mathrm{prev}}] + (j - j_{\mathrm{prev}}),$$

where $\mathcal{J} = \{\mathrm{LF}_T[i] : i \in \mathcal{I}\}$ (Definition 2.8) and $j_{\mathrm{prev}} = \max\{t \in \mathcal{J} : t \leq j\}$.

Proof. We begin by showing that j_{prev} is well-defined. To this end, we will show that $1 \in \mathcal{J}$. Let $i_1 = \mathrm{LF}_T^{-1}[1]$. We claim that $i_1 \in \mathcal{I}$. Suppose that this does not hold. By definition of $\mathcal{I}$, we then have $i_1 > 1$ and $\mathrm{BWT}_T[i_1 - 1] = \mathrm{BWT}_T[i_1]$. By Lemma 5.11, we then have $\mathrm{LF}_T[i_1 - 1] = \mathrm{LF}_T[i_1] - 1 = 0$, which is not possible (see Definition 2.8). Thus, we must have $i_1 \in \mathcal{I}$, and hence $\mathrm{LF}_T[i_1] = 1 \in \mathcal{J}$.

We now prove the claim. If $j = j_{\mathrm{prev}}$, then we trivially obtain that $\mathrm{LF}_T^{-1}[j] = \mathrm{LF}_T^{-1}[j_{\mathrm{prev}}]$. Let us thus assume that $j_{\mathrm{prev}} < j$. Denote $i = \mathrm{LF}_T^{-1}[j]$ and $i_{\mathrm{prev}} = \mathrm{LF}_T^{-1}[j_{\mathrm{prev}}]$. Let also $i' = \max\{t \in \mathcal{I} : t \leq i\}$. The proof proceeds in three steps:

1. First, we prove that it holds $i' = i_{\mathrm{prev}}$. By definition of i', it holds $i' \in \mathcal{I}$ and $(i'\,..\,i] \cap \mathcal{I} = \emptyset$. Thus, for every $t \in (i'\,..\,i]$, $\mathrm{BWT}_T[t-1] = \mathrm{BWT}_T[t]$. Consequently, by Lemma 5.11, for every $t \in [i'\,..\,i]$, it holds $\mathrm{LF}_T[t] = \mathrm{LF}_T[t-1] + 1 = \mathrm{LF}_T[t-2] + 2 = \cdots = \mathrm{LF}_T[i'] + (t - i')$. In other words, the sequence of positions $\mathrm{LF}_T[i'], \mathrm{LF}_T[i'+1], \ldots, \mathrm{LF}_T[i]$ is increasing, and forms a contiguous block. On the other hand, note that $[i'\,..\,i] \cap \mathcal{I} = \{i'\}$ implies that $[\mathrm{LF}_T[i']\,..\,\mathrm{LF}_T[i]] \cap \mathcal{J} = \{\mathrm{LF}_T[i']\}$. Recalling that $\mathrm{LF}_T[i] = j$, we therefore obtain $j_{\mathrm{prev}} = \max\{t \in \mathcal{J} : t \leq j\} = \mathrm{LF}_T[i']$. This implies that $\mathrm{LF}_T^{-1}[j_{\mathrm{prev}}] = i'$, or equivalently, $i_{\mathrm{prev}} = i'$.
2. Next, we prove that $j - j_{\mathrm{prev}} = i - i_{\mathrm{prev}}$. Above we proved that, for every $t \in [i_{\mathrm{prev}}\,..\,i]$, it holds $\mathrm{LF}_T[t] = \mathrm{LF}_T[i_{\mathrm{prev}}] + (t - i_{\mathrm{prev}})$. For $t = i$, we thus obtain $\mathrm{LF}_T[i] = \mathrm{LF}_T[i_{\mathrm{prev}}] + (i - i_{\mathrm{prev}})$. Substituting $\mathrm{LF}_T[i_{\mathrm{prev}}] = j_{\mathrm{prev}}$ and $\mathrm{LF}_T[i] = j$, we thus obtain $j = j_{\mathrm{prev}} + (i - i_{\mathrm{prev}})$, or equivalently, $j - j_{\mathrm{prev}} = i - i_{\mathrm{prev}}$.
3. Putting together the two facts proved above, we obtain

$$\mathrm{LF}_T^{-1}[j] = i = i_{\mathrm{prev}} + (j - j_{\mathrm{prev}}) = \mathrm{LF}_T^{-1}[j_{\mathrm{prev}}] + (j - j_{\mathrm{prev}}).$$ □

Lemma 5.13. *Let $T \in [0\,..\,\sigma)^n$. Denote*

$$T' = \left(\bigodot_{i=1}^{n} (T[i] + 1)\right) \cdot 0 \in [0\,..\,\sigma+1)^{n+1}.$$

Let also $i_{\mathrm{first}} = \mathrm{SA}_T^{-1}[1]$ (Definition 2.5) and $i_{\mathrm{last}} = \mathrm{SA}_T^{-1}[n]$. Then, for every $i \in [1\,..\,n]$, it holds (see Definition 2.9)

$$\mathrm{LF}_T^{-1}[i] = \begin{cases} \mathrm{LF}_{T'}^{-1}[i+1] - 1 & \text{if } i \neq i_{\mathrm{last}}, \\ i_{\mathrm{first}} & \text{otherwise.} \end{cases}$$

Proof. We proceed in three steps:

1. First, we characterize $\mathrm{SA}_{T'}[1\,..\,|T'|]$. More precisely, we show that, for every $i \in [1\,..\,n]$, it holds $\mathrm{SA}_T[i] = \mathrm{SA}_{T'}[i+1]$. To see this, it suffices to note that $\mathrm{SA}_{T'}[1] = n+1$ and, for every $i, j \in [1\,..\,n]$, $T[i\,..\,|T|] \prec T[j\,..\,|T|]$ holds if and only if $T'[i\,..\,|T'|] \prec T'[j\,..\,|T'|]$.
2. Next, we observe that by the above characterization of $\mathrm{SA}_{T'}$, we also obtain that, for every $j \in [1\,..\,n]$, $\mathrm{SA}_T^{-1}[j] = \mathrm{SA}_{T'}^{-1}[j] - 1$.
3. We are now ready to prove the main claim. The equality $\mathrm{LF}_T^{-1}[i_{\mathrm{last}}] = i_{\mathrm{first}}$ follows by definition (see Definition 2.9). Let us thus consider any $i \in [1\,..\,n] \setminus \{i_{\mathrm{last}}\}$. By Definition 2.9, we then have

$$\begin{aligned}\mathrm{LF}_T^{-1}[i] &= \mathrm{SA}_T^{-1}[\mathrm{SA}_T[i]+1] \\ &= \mathrm{SA}_{T'}^{-1}[\mathrm{SA}_T[i]+1] - 1 \\ &= \mathrm{SA}_{T'}^{-1}[\mathrm{SA}_{T'}[i+1]+1] - 1 \\ &= \mathrm{LF}_{T'}^{-1}[i+1] - 1,\end{aligned}$$

   where in the second equality we used that $i \neq i_{\mathrm{last}}$ implies that $\mathrm{SA}_T[i]+1 \in [1\,..\,n]$. □

Lemma 5.14. *Let $T \in [0\,..\,\sigma)^n$. Denote*

$$T' = \left(\bigodot_{i=1}^{n}(T[i]+1)\right) \cdot \mathtt{0} \in [0\,..\,\sigma+1)^{n+1}.$$

Then, it holds $r(T') \leq r(T) + 3$ (Definition 2.26).

Proof. Observe that, for every $i, j \in [1\,..\,n]$, $T[i\,..\,n] \prec T[j\,..\,n]$ holds if and only if $T'[i\,..\,n+1] \prec T'[j\,..\,n+1]$. On the other hand, it holds $\mathrm{SA}_{T'}[1] = n+1$. This implies that, for every $i \in [2\,..\,n+1]$, it holds $\mathrm{SA}_{T'}[i] = \mathrm{SA}_T[i-1]$. Consequently, by Definition 2.26, we obtain

$$\mathrm{BWT}_{T'}[i] = \begin{cases} T[n]+1 & \text{if } i = 1, \\ \mathrm{BWT}_T[i-1]+1 & \text{if } i > 1 \text{ and } i-1 \neq \mathrm{SA}_T^{-1}[1], \\ \mathtt{0} & \text{if } i > 1 \text{ and } i-1 = \mathrm{SA}_T^{-1}[1]. \end{cases}$$

In other words, $\mathrm{BWT}_{T'}$ is obtained from BWT_T by

1. increasing all symbols by one (which does not increase the number of runs),
2. replacing the symbol at index $\mathrm{SA}_T^{-1}[1]$ with $\mathtt{0}$ (which can increase the number of runs by at most two), and
3. prepending the BWT with symbol $T[n]+1$ (which can increase the number of runs by at most one).

Thus, $r(T') \leq r(T) + 3$. □

Theorem 5.15. *For every text $T \in [0\,..\,\sigma)^n$ such that $T[n] = \mathtt{0}$, and $\mathtt{0}$ does not occur in $T[1\,..\,n)$, there exists a data structure of size $\mathcal{O}(r(T))$ (see Definition 2.26) that answers inverse LF queries on T (that, given any $i \in [1\,..\,n]$, return $\mathrm{LF}_T^{-1}[i]$; see Definition 2.9) in $\mathcal{O}(\log\log n)$ time.*

Proof. Let $\mathcal{I} = \{i \in [2\,..\,n] : \mathrm{BWT}_T[i-1] \neq \mathrm{BWT}_T[i]\} \cup \{1\}$ (Definition 2.26) and $\mathcal{J} = \{\mathrm{LF}_T[i] : i \in \mathcal{I}\}$ (Definition 2.8). Denote $m = |\mathcal{J}|$ and let $(p_j)_{j\in[1\,..\,m]}$ be an increasing sequence containing all elements of $\mathcal{J}$, i.e., such that $\{p_1, \ldots, p_m\} = \mathcal{J}$ and $p_1 < p_2 < \cdots < p_m$. We also set $p_0 = -\infty$. Let $A_{\mathcal{J}}[0\,..\,m]$ be an array defined by $A_{\mathcal{J}}[i] = p_i$ for $i \in [0\,..\,m]$. Finally, let $A_{\mathrm{ILF}}[1\,..\,m]$ be an array defined by $A_{\mathrm{ILF}}[i] = \mathrm{LF}_T^{-1}[p_i]$ for $i \in [1\,..\,m]$.

The data structure to answer inverse LF queries on T consists of the following components:

1. The predecessor data structure from Theorem 2.42 for the sequence $(p_i)_{i\in[0\,..\,m]}$. Since for every $i \in [1\,..\,m]$, it holds $p_i \in [0\,..\,n]$, the structure answers queries in $\mathcal{O}(\log\log n)$ time. To bound its space, observe that $m = |\mathcal{J}| = |\mathcal{I}| = r(T)$ (see Definition 2.26). Thus, the structure needs $\mathcal{O}(m) = \mathcal{O}(r(T))$ space.
2. The array $A_{\mathcal{J}}[0\,..\,m]$ stored in plain form. It needs $\mathcal{O}(m) = \mathcal{O}(r(T))$ space.
3. The array $A_{\mathrm{ILF}}[1\,..\,m]$ stored in plain form. It needs $\mathcal{O}(m) = \mathcal{O}(r(T))$ space.

In total, the structure needs $\mathcal{O}(r(T))$ space.

Let $j \in [1\,..\,n]$. To compute $\mathrm{LF}_T^{-1}[j]$ using the above data structure, we proceed as follows:

1. Using the structure from Theorem 2.42, in $\mathcal{O}(\log\log n)$ time, we compute the index $k \in [0\,..\,m]$ satisfying $p_k = \mathsf{pred}_{\mathcal{J}}(j)$ (Definition 2.40). If $k+1 \leq m$ and $p_{k+1} = j$, then we set $j_{\mathrm{prev}} = A_{\mathcal{J}}[k+1]$ and $x = A_{\mathrm{ILF}}[k+1]$. Otherwise, we set $j_{\mathrm{prev}} = A_{\mathcal{J}}[k]$ and $x = A_{\mathrm{ILF}}[k]$. Note that at this point, we have $j_{\mathrm{prev}} = \max\{t \in \mathcal{J} : t \leq j\}$ and $x = \mathrm{LF}_T^{-1}[j_{\mathrm{prev}}]$.
2. In $\mathcal{O}(1)$ time, we set $y = x + (j - j_{\mathrm{prev}})$. By Lemma 5.12, it holds $y = \mathrm{LF}_T^{-1}[j]$. Thus, we return y as the answer.

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

Theorem 5.16. *For every nonempty text $T \in [0\,..\,\sigma)^n$, there exists a data structure of size $\mathcal{O}(r(T))$ (see Definition 2.26) that answers inverse LF queries on T (that, given any $i \in [1\,..\,n]$, return $\mathrm{LF}_T^{-1}[i]$; see Definition 2.9) in $\mathcal{O}(\log\log n)$ time.*

Proof. Let $T' = (\bigodot_{i=1}^{n}(T[i]+1)) \cdot 0 \in [0\,..\,\sigma+1)^{n+1}$. Let also $i_{\mathrm{first}} = \mathrm{SA}_T^{-1}[1]$ and $i_{\mathrm{last}} = \mathrm{SA}_T^{-1}[n]$ (Definition 2.5).

The data structure to answer inverse LF queries on T consists of the following components:

1. The integers i_{first} and i_{last} stored in $\mathcal{O}(1)$ space.
2. The data structure from Theorem 5.15 for text T'. Note that, by Lemma 5.14, it holds $r(T') \leq r(T)+3$. Thus, the structure needs $\mathcal{O}(r(T')) = \mathcal{O}(r(T))$ space.

In total, the structure needs $\mathcal{O}(r(T))$ space.

Let $i \in [1\,..\,n]$. To compute $\mathrm{LF}_T^{-1}[i]$ using the above data structure, we proceed as follows:

1. If $i = i_{\mathrm{last}}$, then in $\mathcal{O}(1)$ time we return $\mathrm{LF}_T^{-1}[i] = i_{\mathrm{first}}$ (this holds by Definition 2.9), and conclude the query algorithm. Let us thus assume that $i \neq i_{\mathrm{last}}$.
2. Using the structure from Theorem 5.15, in $\mathcal{O}(\log\log n)$ time, we compute $x = \mathrm{LF}_{T'}^{-1}[i+1]$.
3. In $\mathcal{O}(1)$ time, we set $y = x - 1$. By Lemma 5.13, it holds $y = \mathrm{LF}_T^{-1}[i]$. Thus, we return y as the answer.

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

Theorem 5.17. *For every nonempty text $T \in \Sigma^n$, there exists a data structure of size $\mathcal{O}(r(T))$ (see Definition 2.26) that answers inverse LF queries on T (that, given any $i \in [1\,..\,n]$, return $\mathrm{LF}_T^{-1}[i]$; see Definition 2.9) in $\mathcal{O}(\log\log n)$ time.*

Proof. Let Σ_T denote the effective alphabet of T, i.e., $\Sigma_T = \{T[i] : i \in [1\,..\,n]\}$. Let $T_{\mathrm{rank}}[1\,..\,n]$ denote a string defined so that, for every $i \in [1\,..\,n]$, $T_{\mathrm{rank}}[i]$ is the rank of $T[i]$ in Σ_T, i.e., $T_{\mathrm{rank}}[i] = |\{c \in \Sigma_T : c \prec T[i]\}|$.

The data structure to answer inverse LF queries on T consists of a single component: the structure from Theorem 5.16 for text T_{rank}. Since $r(T) = r(T_{\mathrm{rank}})$, the structure from Theorem 5.16 (and hence the whole structure) needs $\mathcal{O}(r(T))$ space.

Let $i \in [1\,..\,n]$. To compute $\mathrm{LF}_T^{-1}[i]$ using the above data structure, we simply compute and return $\mathrm{LF}_{T_{\mathrm{rank}}}^{-1}[i] = \mathrm{LF}_T^{-1}[i]$ in $\mathcal{O}(\log\log n)$ using the structure from Theorem 5.16. □

Theorem 5.18. *For every nonempty text $T \in \Sigma^n$, there exists a data structure of size $\mathcal{O}(\delta(T)\log^2 n)$ (see Definition 2.32) that answers inverse LF queries on T (that, given any $i \in [1\,..\,n]$, return $\mathrm{LF}_T^{-1}[i]$; see Definition 2.9) in $\mathcal{O}(\log\log n)$ time.*

Proof. The claim follows by combining Theorem 5.17 and Theorem 2.35 (see also Remark 2.36). □

5.5 Lexicographical Predecessor (Φ) Queries

5.5.1 Problem Definition

Indexing for Lexicographical Predecessor (Φ) Queries

Input: A string $T \in \Sigma^n$.
Output: A data structure that, given any index $j \in [1\,..\,n]$, returns $\Phi_T[j]$ (Definition 2.6).

5.5.2 Lower Bound

Lemma 5.19. *Let $A \subseteq [1\,..\,m^2]$ be a nonempty set of size $|A| = m$. Denote $A = \{a_1, a_2, \ldots, a_m\}$, where $a_1 < a_2 < \cdots < a_m$, and let $a_0 = -\infty$. Let*

$$T = \left(\bigodot_{i=1}^{m} (\mathtt{0}^{a_i}\mathtt{1}^{m^2-a_i+2})\right) \cdot \left(\mathtt{0}^{m^2+1}\mathtt{1}\right) \cdot \left(\mathtt{0}^{m^2}\mathtt{1}^{m^2+2}\right) \in \{\mathtt{0},\mathtt{1}\}^*$$

(brackets added for clarity). Then, for every $x \in [1\,..\,m^2]$, it holds (see Definition 2.40)

$$\mathsf{pred}_A(x) = a_i,$$

where $\Delta = |T| - 2(m^2+1)$ and $i = \left\lceil \frac{\Phi_T[\Delta + m^2 - x + 1]}{m^2+2} \right\rceil - 1$ (Definition 2.6).

Proof. Denote $j = \Delta + m^2 - x + 1$ and let $a_{m+1} = m^2 + 1$. Let $p = \min\{t \in [1\,..\,m+1] : a_t \geq x\}$. Note that that p is well-defined, since $a_{m+1} = m^2+1$. For every $t \in [p\,..\,m+1]$, denote $s_t = (t-1)(m^2+2) + (a_t - x + 1)$. By definition of T, it holds $|\mathrm{Occ}(\mathtt{0}^x\mathtt{1}, T)| = n - p + 3$ and, moreover, $\mathrm{Occ}(\mathtt{0}^v\mathtt{1}, T) = \{s_t\}_{t \in [p\,..\,n+1]} \cup \{j\}$. Next, observe that:

- $T[j\,..\,|T|]$ has $\mathtt{0}^x\mathtt{1}^{m^2+2}$ as a prefix and
- for every $t \in [p\,..\,m+1]$, $T[s_t\,..\,|T|]$ has $\mathtt{0}^x\mathtt{1}^{m^2-a_t+2}\mathtt{0}$ as a prefix.

This implies that it holds $T[s_{m+1}\,..\,|T|] \prec T[s_m\,..\,|T|] \prec \cdots \prec T[s_p\,..\,|T|] \prec T[j\,..\,|T|]$. Consequently, it holds $\Phi_T[j] = s_p$. It remains to note that for every $t \in [p\,..\,m+1]$, it holds $\lceil s_t/(m^2+2) \rceil = t$. Combining this with $\mathsf{pred}_A(x) = a_{p-1}$ (following from the definition of p), we obtain

$$i = \left\lceil \frac{\Phi_T[\Delta + m^2 - x + 1]}{m^2+2} \right\rceil - 1 = \left\lceil \frac{\Phi_T[j]}{m^2+2} \right\rceil - 1 = \left\lceil \frac{s_p}{m^2+2} \right\rceil - 1 = p - 1.$$

Thus, indeed we have $\mathsf{pred}_A(x) = a_{p-1} = a_i$. □

Theorem 5.20. *There is no data structure that, for every text $T \in \{\mathtt{0},\mathtt{1}\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers Φ queries on T (that, given any $j \in [1\,..\,n]$, return $\Phi_T[j]$; see Definition 2.6) in $o(\log\log n)$ time.*

Proof. Suppose that the claim does not hold. Let D_Φ denote the hypothetical data structure answering Φ queries, and let $c = \mathcal{O}(1)$ be a positive constant such that, that for a text $T \in \{\mathtt{0},\mathtt{1}\}^n$, D_Φ uses $\mathcal{O}(\delta(T)\log^c n)$ space, and answers Φ queries on T in $t_\Phi(|T|) = o(\log\log|T|)$ time. We will prove that this implies that there exists a data structure answering predecessor queries, that contradicts Theorem 2.44.

Let $m \geq 1$ and let $A \subseteq [1\,..\,m^2]$ be a nonempty set of size m. Denote $A = \{a_1, a_2, \ldots, a_m\}$, where $a_1 < a_2 < \cdots < a_m$. Let $T_A = (\bigodot_{i=1}^{m}(\mathtt{0}^{a_i}\mathtt{1}^{m^2-a_i+2})) \cdot (\mathtt{0}^{m^2+1}\mathtt{1}) \cdot (\mathtt{0}^{m^2}\mathtt{1}^{m^2+2}) \in \{\mathtt{0},\mathtt{1}\}^*$ be a text defined as in Lemma 5.19. Denote $\Delta = |T_A| - 2(m^2+1)$. Let $A_{\mathrm{val}}[0\,..\,m]$ be an array such that $A_{\mathrm{val}}[0] = -\infty$ and, for every $i \in [1\,..\,m]$, $A_{\mathrm{val}}[i] = a_i$.

Let D_{pred} denote the data structure consisting of the following components:

1. The array $A_{\mathrm{val}}[0\,..\,m]$ stored in plain form. It needs $\mathcal{O}(m)$ space.
2. The integer Δ stored in $\mathcal{O}(1)$ space.
3. The data structure D_Φ for the string T_A. Note that

$$|T_A| = m^3 + 3m^2 + 2m + 3 \in \mathcal{O}(m^3).$$

   Moreover, note that $|\mathrm{RL}(T_A)| = 2(m+2)$. By applying Observation 2.19 and Theorem 2.37(1), we thus have $\delta(T_A) = \mathcal{O}(m)$. Consequently, D_Φ for T_A needs $\mathcal{O}(\delta(T_A)\log^c|T_A|) = \mathcal{O}(m\log^c m)$ space.

In total, D_{pred} needs $\mathcal{O}(m\log^c m)$ space.

Given any $x \in \mathbb{Z}$, we compute $\mathsf{pred}_A(x)$ as follows:

1. If $x < 1$ (resp. $x > m^2$), we return $\mathsf{pred}_A(x) = -\infty$ (resp. $\mathsf{pred}_A(x) = A_{\mathrm{val}}[m]$) in $\mathcal{O}(1)$ time, and conclude the query algorithm. Let us thus assume that $x \in [1\,..\,m^2]$.
2. Using D_Φ, in $\mathcal{O}(t_\Phi(|T_A|))$ time we compute $j = \Phi_{T_A}[\Delta + m^2 - x + 1]$. We then let $i = \lceil j/(m^2+2) \rceil - 1$, and return $p = A_{\mathrm{val}}[i]$ as the answer. By Lemma 5.19, it holds $p = \mathsf{pred}_A(x)$.

In total, the query takes $\mathcal{O}(t_\Phi(|T_A|)) = o(\log\log|T_A|) = o(\log\log(m^3)) = o(\log\log m)$ time.

We have thus proved that there exists a data structure that, for every set $A \subseteq [1\,..\,m^2]$ of size $|A| = m$, uses $\mathcal{O}(m\log^c m) = \mathcal{O}(m\log^{\mathcal{O}(1)} m)$ space, and answers predecessor queries on A in $o(\log\log m)$ time. This contradicts Theorem 2.44. □

5.6 Inverse Lexicographical Predecessor (Φ^{-1}) Queries

5.6.1 Problem Definition

INDEXING FOR INVERSE LEXICOGRAPHICAL PREDECESSOR (Φ^{-1}) QUERIES

Input: A string $T \in \Sigma^n$.
Output: A data structure that, given any index $j \in [1\,..\,n]$, returns $\Phi_T^{-1}[j]$ (Definition 2.7).

5.6.2 Lower Bound

Lemma 5.21. *Let $\sigma \geq 1$ and $T \in [0\,..\,\sigma)^n$ be a nonempty text. Let*

$$T' = \left(\bigodot_{i=1}^{n} \left(\mathtt{001} \cdot (\sigma - 1 - T[i]) \cdot \mathtt{1}\right)\right) \cdot \mathtt{1} \in [0\,..\,\sigma)^*$$

(brackets added for clarity). Then:

- *For every $j \in [1\,..\,n] \setminus \{\mathrm{SA}_T[1]\}$, it holds (see Definitions 2.6 and 2.7)*
$$\Phi_T[j] = \tfrac{1}{5}\left(\Phi_{T'}^{-1}[1 + 5(j-1)] - 1\right) + 1.$$
- *For every $j \in [1\,..\,n] \setminus \{\mathrm{SA}_T[n]\}$, it holds*
$$\Phi_T^{-1}[j] = \tfrac{1}{5}\left(\Phi_{T'}[1 + 5(j-1)] - 1\right) + 1.$$

Proof. Let $n' = |T'| = 5n+1$. Let $(p_j)_{j\in[1\,..\,n]}$ be a sequence defined by $p_j = 1+5(j-1)$. For every $a \in [0\,..\,\sigma)$, denote $s(a) = \mathtt{001} \cdot (\sigma - 1 - a) \cdot \mathtt{1} \in [0\,..\,\sigma)^*$. Observe that for every $j \in [1\,..\,n]$, we then have

$$T'[p_j\,..\,n'] = s(T[j])s(T[j+1]) \cdots s(T[n]) \cdot \mathtt{1}.$$

Finally, let $\alpha = \mathrm{RangeBeg}(\mathtt{00}, T')$.

The proof proceeds in three steps:

1. First, we prove that, for every $j_1, j_2 \in [1\,..\,n]$, $T[j_1\,..\,n] \prec T[j_2\,..\,n]$ implies that $T'[p_{j_1}\,..\,n'] \succ T'[p_{j_2}\,..\,n']$. We consider two cases:
   - First, assume that $T[j_1\,..\,n]$ is a proper prefix of $T[j_2\,..\,n]$. Note that, letting $\ell = n - j_1 + 1$, we then have $j_2 < j_1$ and
   $$\begin{aligned} T'[p_{j_1}\,..\,p_{j_1} + 5\ell) &= s(T[j_1])s(T[j_1+1]) \cdots s(T[n]) \\ &= s(T[j_2])s(T[j_2+1]) \cdots s(T[j_2+\ell-1]) \\ &= T'[p_{j_2}\,..\,p_{j_2} + 5\ell). \end{aligned}$$
   Moreover, by $p_{j_1} + 5\ell = 1 + 5(j_1 - 1) + 5(n - j_1 + 1) = 5n + 1 = n'$, we then have $T'[p_{j_1} + 5\ell] = \mathtt{1}$. On the other hand, $p_{j_2} + 5\ell = 1 + 5(j_2 - 1) + 5(n - j_1 + 1) = 5n + 1 - 5(j_1 - j_2) = n' - 5(j_1 - j_2) < n'$ implies that $T'[p_{j_2} + 5\ell] = \mathtt{0}$ (this follows since, for every $a \in [0\,..\,\sigma)$, the first symbol of $s(a)$ is $\mathtt{0}$). Thus, we have $T'[p_{j_1}\,..\,n'] \succ T'[p_{j_2}\,..\,n']$.
   - Let us now assume that there exists $\ell \in [0\,..\,\min(n - j_1 + 1, n - j_2 + 1))$ such that $T[j_1\,..\,j_1 + \ell) = T[j_2\,..\,j_2 + \ell)$ and $T[j_1 + \ell] \prec T[j_2 + \ell]$. By definition of T', we then have
   $$\begin{aligned} T'[p_{j_1}\,..\,p_{j_1} + 5\ell) &= s(T[j_1])s(T[j_1+1]) \cdots s(T[j_1+\ell-1]) \\ &= s(T[j_2])s(T[j_2+1]) \cdots s(T[j_2+\ell-1]) \\ &= T'[p_{j_2}\,..\,p_{j_2} + 5\ell). \end{aligned}$$
   Moreover, by $T[j_1 + \ell] \prec T[j_2 + \ell]$, we then have
   $$\begin{aligned} T'[p_{j_1} + 5\ell\,..\,p_{j_1} + 5(\ell+1)) &= s(T[j_1 + \ell]) \\ &\succ s(T[j_2 + \ell]) \\ &= T'[p_{j_2} + 5\ell\,..\,p_{j_2} + 5(\ell+1)). \end{aligned}$$
   Thus, we obtain $T'[p_{j_1}\,..\,n'] \succ T'[p_{j_2}\,..\,n']$.

2. In the second step, we prove that, for every $i \in [1\,..\,n]$, it holds

$$\mathrm{SA}_{T'}[\alpha + i] = p_{\mathrm{SA}_T[(n-i)+1]}.$$

Recall that, by definition of the suffix array, it holds $T[\mathrm{SA}_T[1]\,..\,n] \prec \cdots \prec T[\mathrm{SA}_T[n]\,..\,n]$. By the above step, we thus have

$$T'[p_{\mathrm{SA}_T[n]}\,..\,n'] \prec \cdots \prec T'[p_{\mathrm{SA}_T[1]}\,..\,n'].$$

It remains to observe that $\mathrm{Occ}(\mathtt{00}, T') = \{p_1, p_2, \ldots, p_n\}$. Recalling that $\alpha = \mathrm{RangeBeg}(\mathtt{00}, T')$, we thus have $\mathrm{SA}_{T'}[\alpha + i] = p_{\mathrm{SA}_T[(n-i)+1]}$ for every $i \in [1\,..\,n]$.

3. We are now ready to show the claims. Let $j \in [1\,..\,n] \setminus \{\mathrm{SA}_T[1]\}$. Let $i = \mathrm{SA}_T^{-1}[j]$. By Definition 2.6, we then have $i > 1$ and $\Phi_T[j] = \mathrm{SA}_T[i-1]$. In the previous step, we proved that $\Phi_{T'}^{-1}[p_{\mathrm{SA}_T[i]}] = p_{\mathrm{SA}_T[i-1]}$. Substituting $\mathrm{SA}_T[i]$ with j and $\mathrm{SA}_T[i-1]$ with $\Phi_T[j]$, we thus obtain that $\Phi_{T'}^{-1}[p_j] = p_{\Phi_T[j]}$. Expanding the definition of p_j and $p_{\Phi_T[j]}$, we therefore have

$$\Phi_{T'}^{-1}[1 + 5(j-1)] = 1 + 5 \cdot (\Phi_T[j] - 1).$$

By rewriting for $\Phi_T[j]$, we obtain $\Phi_T[j] = \frac{1}{5}(\Phi_{T'}^{-1}[1 + 5(j-1)] - 1) + 1$, i.e., the first claim. Let us now consider $j \in [1\,..\,n] \setminus \{\mathrm{SA}_T[n]\}$. Let again $i = \mathrm{SA}_T^{-1}[j]$. By Definition 2.7, we have $i < n$ and $\Phi_T^{-1}[j] = \mathrm{SA}_T[i+1]$. In the previous step, we proved that $\Phi_{T'}[p_{\mathrm{SA}_T[i]}] = p_{\mathrm{SA}_T[i+1]}$. Substituting $\mathrm{SA}_T[i]$ with j and $\mathrm{SA}_T[i+1]$ with $\Phi_T^{-1}[j]$, we thus obtain that $\Phi_{T'}[p_j] = p_{\Phi_T^{-1}[j]}$. Expanding the definition of p_j and $p_{\Phi_T^{-1}[j]}$, we therefore have

$$\Phi_{T'}[1 + 5(j-1)] = 1 + 5 \cdot (\Phi_T^{-1}[j] - 1).$$

By rewriting for $\Phi_T^{-1}[j]$, we obtain $\Phi_T^{-1}[j] = \frac{1}{5}(\Phi_{T'}[1 + 5(j-1)] - 1) + 1$, i.e., the second claim. □

Lemma 5.22. *Let Σ be a nonempty set, and let $f : \Sigma \to \Sigma^k$, where $k \in \mathbb{Z}_{>0}$, be any function. Then, for every nonempty text $T \in \Sigma^n$, it holds (see Definition 2.32)*

$$\delta(T') = \mathcal{O}(k \cdot \delta(T) \cdot \log n),$$

where $T' = \bigodot_{i=1}^{n} f(T[i])$.

Proof. The proof proceeds in two steps:

1. First, we show that $z(T') \leq k \cdot z(T)$. To this end, we observe that T' has an LZ77-like factorization consisting of $k \cdot z(T)$ phrases, which is constructed as follows:
   - If $T[i]$ in the LZ77 factorization is encoded as a literal phrase, then in the LZ77-like parsing of T', we encode symbols in $T'(k(i-1)\,..\,ki]$ as k literal phrases.
   - If the substring $T(i\,..\,j]$ is encoded as a repeat phrase in the LZ77 factorization of T, with a source starting at $i' < i$, then in the LZ77-like parsing of T', we encode the substring $T(ki\,..\,kj]$ as a repeat phrase, with a source starting at position $1 + k(i'-1)$.

   It remains to apply Theorem 2.18 to obtain $z(T') \leq k \cdot z(T)$.
2. We are now ready to complete the proof. By Lemma 2.33 (see also Remark 2.34), it holds $z(T) = \mathcal{O}(\delta(T) \cdot \log n)$. On the other hand, by Theorem 2.37(1), we have $\delta(T') = \mathcal{O}(z(T'))$. Putting these two observations together with the previous step, we thus obtain

$$\delta(T') = \mathcal{O}(z(T')) = \mathcal{O}(k \cdot z(T)) = \mathcal{O}(k \cdot \delta(T) \cdot \log n).$$ □

Theorem 5.23. *Let $\sigma \geq 1$ and $T \in [0\,..\,\sigma)^n$ be a nonempty text. The $\Phi_T[j]$ and $\Phi_T^{-1}[j]$ queries (Definitions 2.6 and 2.7) are related in the following sense: If there exists a data structure using $\mathcal{O}(\delta(T)\log^c n)$ space (where $c = \mathcal{O}(1)$ is a positive constant) that answers one of these two types of queries in $t_{\mathrm{query}}(n)$ time, then there exists a data structure answering the other type of queries also in $t_{\mathrm{query}}(5n+1)$ time and using $\mathcal{O}(\delta(T)\log^{c+1} n)$ space.*

Proof. For concreteness, we prove the equivalence in one direction. The proof in the opposite direction follows symmetrically (relying on the symmetry of Lemma 5.21). Let D_Φ be a data structure that, for every $T \in [0\,..\,\sigma)^n$, uses $\mathcal{O}(\delta(T)\log^c n)$ space (where $c = \mathcal{O}(1)$ is a positive constant) and, given any $j \in [1\,..\,n]$, returns $\Phi_T[j]$ in $t_{\text{query}}(n)$ time. We will present a data structure that uses $\mathcal{O}(\delta(T)\log^{c+1} n)$ space and, given any $j \in [1\,..\,n]$, returns $\Phi_T^{-1}[j]$ in $t_{\text{query}}(5n+1)$ time.

Let $T \in [0\,..\,\sigma)^n$ and $T' = (\bigodot_{i=1}^{n}(\mathtt{001}\cdot(\sigma-1-T[i])\cdot\mathtt{1}))\cdot\mathtt{1} \in [0\,..\,\sigma)^*$ be a text defined as in Lemma 5.21. Let $j_{\text{lexfirst}} = \mathrm{SA}_T[1]$ and $j_{\text{lexlast}} = \mathrm{SA}_T[n]$.

Let $D_{\Phi^{-1}}$ denote a data structure consisting of the following components:

1. The integers j_{lexfirst} and j_{lexlast} using $\mathcal{O}(1)$ space.
2. The data structure D_Φ for the string T'. Note that $|T'| = 5n+1$. To show an upper bound for $\delta(T')$, we define T'_{pref} as a string satisfying $T' = T'_{\text{pref}}\cdot\mathtt{1}$. By Lemma 5.22, we have $\delta(T'_{\text{pref}}) = \mathcal{O}(\delta(T)\log n)$. On the other hand, by Lemma 4.6, it holds $\delta(T') = \mathcal{O}(\delta(T'_{\text{pref}}))$. Thus, we obtain $\delta(T') = \mathcal{O}(\delta(T)\log n)$. Consequently, D_Φ for T' needs $\mathcal{O}(\delta(T')\log^c|T'|) = \mathcal{O}((\delta(T)\log n)\cdot\log^c|T'|) = \mathcal{O}((\delta(T)\log n)\cdot\log^c n) = \mathcal{O}(\delta(T)\log^{c+1} n)$ space.

In total, $D_{\Phi^{-1}}$ needs $\mathcal{O}(\delta(T)\log^{c+1} n)$ space.

Given any $j \in [1\,..\,n]$, we compute $\Phi_T^{-1}[j]$ as follows:

1. If $j = j_{\text{lexlast}}$, then in $\mathcal{O}(1)$ time we return $\Phi_T^{-1}[j] = j_{\text{lexfirst}}$. Let us now assume that $j \neq j_{\text{lexlast}}$.
2. Using D_Φ, in $t_{\text{query}}(|T'|) = t_{\text{query}}(5n+1)$ time we compute $j' = \Phi_{T'}[1+5(j-1)]$. We then set $j'' = \frac{1}{5}(j'-1)+1$. By Lemma 5.21, it holds $\Phi_T^{-1}[j] = j''$. Thus, we return j'' as the answer.

In total, the query takes $\mathcal{O}(1) + t_{\text{query}}(5n+1) = t_{\text{query}}(5n+1)$ time. □

Theorem 5.24. *There is no data structure that, for every text $T \in \{\mathtt{0},\mathtt{1}\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers Φ^{-1} queries on T (that, given any $j \in [1\,..\,n]$, return $\Phi_T^{-1}[j]$; see Definition 2.7) in $o(\log\log n)$ time.*

Proof. Suppose that the claim does not hold. Let $D_{\Phi^{-1}}$ denote the hypothetical data structure answering Φ^{-1} queries, and let $c = \mathcal{O}(1)$ be a positive constant such that, that for a text $T \in \{\mathtt{0},\mathtt{1}\}^n$, $D_{\Phi^{-1}}$ uses $\mathcal{O}(\delta(T)\log^c n)$ space, and answers Φ^{-1} queries on T in $t_{\Phi^{-1}}(|T|) = o(\log\log|T|)$ time. By Theorem 5.23, this implies that, for every text $T \in \{\mathtt{0},\mathtt{1}\}^n$, there exists a data structure of size $\mathcal{O}(\delta(T)\log^{c+1} n) = \mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ that answers Φ queries on T in $t_{\Phi^{-1}}(5|T|+1) = o(\log\log(5n+1)) = o(\log\log n)$ time. This contradicts Theorem 5.20. □

A Stronger Lower Bound for Inverse Suffix Array (SA^{-1}) Queries

In this section, we present a reduction from range counting queries to inverse suffix array queries. While this does not change the main result of the paper — we can already prove an $\Omega\left(\frac{\log n}{\log\log n}\right)$ lower bound for inverse suffix array queries by reducing from random access queries (which in turn are proved via a reduction from range counting *parity* [VY13]) — the reduction below (which is of independent interest) establishes a strictly stronger result: namely, that inverse suffix array can be used to answer full range counting queries.

Lemma A.1. *Let $A[1\,..\,n]$ be a nonempty array containing a permutation of $\{1,2,\ldots,n\}$. Let*

$$T = \left(\bigodot_{i=1}^{n}(\mathtt{0}^{A[i]}\mathtt{1}^i)\right)\cdot\mathtt{0}^{n+1}\mathtt{1}^{n+1}\cdot\left(\bigodot_{i=1}^{n+1}(\mathtt{0}^{n+1}\mathtt{1}^i)\right) \in \{\mathtt{0},\mathtt{1}\}^*$$

(brackets added for clarity). Then, for every $j \in [0\,..\,n]$ and every $v \in [1\,..\,n]$, it holds (see Definitions 2.5 and 2.48)

$$\mathsf{range\text{-}count}_A(j,v) = \mathrm{SA}_T^{-1}[j'] - (b+j+1),$$

where

- $b = \mathrm{RangeBeg}(\mathtt{0}^v\mathtt{1}, T)$ *(Definition 2.1),*
- $\ell_1 = n\cdot(n+1)$,
- $\ell_2 = 2\cdot(n+1)$,
- $\delta = j\cdot(n+1+\frac{j+1}{2}) + (n+2-v)$, *and*
- $j' = \ell_1 + \ell_2 + \delta$.

Proof. Denote $r = \mathsf{range\text{-}count}_A(j, v)$ and $k = \mathsf{range\text{-}count}_A(n, v)$. Let $(a_i)_{i\in[1\,..\,k]}$ be a sequence such that, for every $i \in [1\,..\,k]$, $a_i = \mathsf{range\text{-}select}_A(r, v)$. By definition, we then have $a_1 < a_2 < \cdots < a_r \leq j < a_{r+1} < \cdots < a_k$. Next, let $(s_i)_{i\in[1\,..\,k]}$ be a sequence defined such that for every $i \in [1\,..\,k]$,

$$s_i = \Big(\textstyle\sum_{t=1}^{a_i-1}(A[t]+t)\Big) + \Big(A[a_i] - v + 1\Big).$$

Let also

$$u = \ell_1 + (n + 2 - v).$$

Next, denote $q = n + 1$, and let $(t_i)_{i\in[1\,..\,q]}$ be a sequence defined so that for every $i \in [1\,..\,q]$,

$$t_i = \ell_1 + \ell_2 + i \cdot (n + 1 + \tfrac{i+1}{2}) + (n + 2 - v).$$

Observe that by definition of T, it follows that $|\mathrm{Occ}(\mathtt{0}^v\mathtt{1}, T)| = k + 1 + q$ and, moreover, $\mathrm{Occ}(\mathtt{0}^v\mathtt{1}, T)$ is a disjoint union

$$\mathrm{Occ}(\mathtt{0}^v\mathtt{1}, T) = \{s_i\}_{i\in[1\,..\,k]} \cup \{u\} \cup \{t_i\}_{i\in[1\,..\,q]}.$$

Finally, note that it holds

$$T[j'\,..\,|T|] = \mathtt{0}^v\mathtt{1}^{j+1} \cdot \Big(\textstyle\bigodot_{i=j+2}^{n+1}(\mathtt{0}^{n+1}\mathtt{1}^i)\Big).$$

The rest of the proof proceeds in four steps:

1. First, we prove that $|\{i \in [1\,..\,k] : T[s_i\,..\,|T|] \prec T[j'\,..\,|T|]\}| = r$. Observe that for every $i \in [1\,..\,k]$, the suffix $T[s_i\,..\,|T|]$ has the string $\mathtt{0}^v\mathtt{1}^{a_1}\mathtt{0}$ as a prefix. By $a_1 < \cdots < a_r \leq j$ and since $T[j'\,..\,|T|]$ has $\mathtt{0}^v\mathtt{1}^{j+1}$ as a prefix, we obtain $T[s_1\,..\,|T|] \prec \cdots \prec T[s_r\,..\,|T|] \prec T[j'\,..\,|T|]$. We now show that for every $i \in [r + 1\,..\,k]$, it holds $T[j'\,..\,|T|] \prec T[s_i\,..\,|T|]$. Consider any $i \in [r + 1\,..\,k]$. Note that the existence of such i implies that $j < n$, and hence also that $T[j'\,..\,|T|]$ has $\mathtt{0}^v\mathtt{1}^{j+1}\mathtt{0}^{n+1}\mathtt{1}$ as a prefix. Consider two cases.
   - First, assume that $i \in [r+2\,..\,k]$. Recall that $T[s_i\,..\,|T|]$ has $\mathtt{0}^v\mathtt{1}^{a_i}\mathtt{0}$ as a prefix. By $j+1 \leq a_{r+1} < a_i$, we thus immediately obtain $T[j'\,..\,|T|] \prec T[s_i\,..\,|T|]$.
   - Let us now consider the case $i = r+1$. Recall that by definition of r, we have $j+1 \leq a_{r+1}$. If $j+1 < a_{r+1}$, then we immediately obtain $T[j'\,..\,|T|] \prec T[s_{r+1}\,..\,|T|]$, since $T[j'\,..\,|T|]$ (resp. $T[s_{r+1}\,..\,|T|]$) has $\mathtt{0}^v\mathtt{1}^{j+1}\mathtt{0}$ (resp. $\mathtt{0}^v\mathtt{1}^{a_{r+1}}\mathtt{0}$) as a prefix. Let us thus assume that $j + 1 = a_{r+1}$. If $a_{r+1} < n$, then again we obtain $T[j'\,..\,|T|] \prec T[s_{r+1}\,..\,|T|]$, since then $T[j'\,..\,|T|]$ (resp. $T[s_{r+1}\,..\,|T|]$) has $\mathtt{0}^v\mathtt{1}^{j+1}\mathtt{0}^{n+1}\mathtt{1}$ (resp. $\mathtt{0}^v\mathtt{1}^{a_{r+1}}\mathtt{0}^{A[a_{r+1}+1]}\mathtt{1} = \mathtt{0}^v\mathtt{1}^{j+1}\mathtt{0}^{A[a_{r+1}+1]}\mathtt{1}$) as a prefix (recall that $A[t] \leq n$ holds for all $t \in [1\,..\,n]$). Let us finally consider the remaining case $j + 1 = a_{r+1} = n$. In this case, we have $T[j'\,..\,|T|] = \mathtt{0}^v\mathtt{1}^n\mathtt{0}^{n+1}\mathtt{1}^{n+1}$ and $T[s_{r+1}\,..\,|T|]$ has $\mathtt{0}^v\mathtt{1}^{a_{r+1}}\mathtt{0}^{n+1}\mathtt{1}^{n+1}\mathtt{0} = \mathtt{0}^v\mathtt{1}^n\mathtt{0}^{n+1}\mathtt{1}^{n+1}\mathtt{0}$ as a prefix. Thus, $T[j'\,..\,|T|]$ is a prefix of $T[s_{r+1}\,..\,|T|]$, and hence $T[j'\,..\,|T|] \prec T[s_{r+1}\,..\,|T|]$.
2. Second, we show that $T[j'\,..\,|T|] \prec T[u\,..\,|T|]$. Consider two cases:
   - First, assume that $j < n$. Then, $T[j'\,..\,|T|]$ has $\mathtt{0}^v\mathtt{1}^{j+1}\mathtt{0}$ as a prefix. Since, on the other hand, $T[u\,..\,|T|]$ has $\mathtt{0}^v\mathtt{1}^{n+1}$ as a prefix, we thus obtain $T[j'\,..\,|T|] \prec T[u\,..\,|T|]$.
   - Let us now assume that $j = n$. Note that then $T[j'\,..\,|T|] = \mathtt{0}^v\mathtt{1}^{n+1}$. Since $T[u\,..\,|T|]$ has $\mathtt{0}^v\mathtt{1}^{n+1}\mathtt{0}$ as a prefix, we thus again obtain $T[j'\,..\,|T|] \prec T[u\,..\,|T|]$.
3. Next, we prove that $|\{i \in [1\,..\,q] : T[t_i\,..\,|T|] \prec T[j'\,..\,|T|]\}| = j$. We proceed in three steps.
   - We start by observing that, for every $i \in [1\,..\,q-1]$, the suffix $T[t_i\,..\,|T|]$ has the string $\mathtt{0}^v\mathtt{1}^i\mathtt{0}$ as a prefix. Since $T[j'\,..\,|T|]$ has $\mathtt{0}^v\mathtt{1}^{j+1}$ as a prefix, we obtain that for all $i \in [1\,..\,j]$, it holds $T[t_i\,..\,|T|] \prec T[j'\,..\,|T|]$.
   - Next, observe that $j' = t_{j+1}$. Thus, it does not hold that $T[t_{j+1}\,..\,|T|] \prec T[j'\,..\,|T|]$.
   - Finally, let us consider any $i \in [j + 2\,..\,q]$. Note that the existence of such i implies that $j + 1 < j + 2 \leq q = n + 1$. Thus, $T[j'\,..\,|T|]$ then has $\mathtt{0}^v\mathtt{1}^{j+1}\mathtt{0}$ as a prefix. On the other hand, $T[t_i\,..\,|T|]$ has $\mathtt{0}^v\mathtt{1}^i$ as a prefix. By $j + 1 < i$, we thus obtain $T[j'\,..\,|T|] \prec T[t_i\,..\,|T|]$.
4. We now put everything together. First, note that by the three facts proved above, and the earlier observation that $|\mathrm{Occ}(\mathtt{0}^v\mathtt{1}, T)| = k + 1 + q$ and $\mathrm{Occ}(\mathtt{0}^v\mathtt{1}, T) = \{s_i\}_{i\in[1\,..\,k]} \cup \{u\} \cup \{t_i\}_{i\in[1\,..\,q]}$, we obtain that

$$|\{t \in \mathrm{Occ}(\mathtt{0}^v\mathtt{1}, T) : T[t\,..\,|T|] \prec T[j'\,..\,|T|]\}| = j + r.$$

By $j' \in \mathrm{Occ}(\mathtt{0}^v\mathtt{1}, T)$, it follows that $\mathrm{RangeBeg}(T[j'\,..\,|T|], T) = \mathrm{RangeBeg}(\mathtt{0}^v\mathtt{1}, T) + |\{t \in \mathrm{Occ}(\mathtt{0}^v\mathtt{1}, T) : T[t\,..\,|T|] \prec T[j'\,..\,|T|]\}|$. Consequently, by definition of $\mathrm{SA}_T^{-1}[j']$, we have

$$\begin{aligned}\mathrm{SA}_T^{-1}[j'] &= 1 + \mathrm{RangeBeg}(T[j'\,..\,|T|], T)\\ &= 1 + \mathrm{RangeBeg}(\mathtt{0}^v\mathtt{1}, T) + |\{t \in \mathrm{Occ}(\mathtt{0}^v\mathtt{1}, T) : T[t\,..\,|T|] \prec T[j'\,..\,|T|]\}|\\ &= 1 + b + j + r\\ &= (1 + b + j) + \mathsf{range\text{-}count}_A(j, v),\end{aligned}$$

which is equivalent to the claim. □

Theorem A.2. *There is no data structure that, for every text $T \in \{\mathtt{0},\mathtt{1}\}^n$, uses $\mathcal{O}(\delta(T)\log^{\mathcal{O}(1)} n)$ space and answers inverse SA queries on T (that, given any $i \in [1\,..\,n]$, return $\mathrm{SA}_T^{-1}[i]$; see Definition 2.5) in $o(\frac{\log n}{\log\log n})$ time.*

Proof. Suppose that the claim does not hold. Let D_{ISA} denote the hypothetical data structure answering inverse SA queries, and let $c = \mathcal{O}(1)$ be a positive constant such that, that for a text $T \in \{\mathtt{0},\mathtt{1}\}^n$, D_{ISA} uses $\mathcal{O}(\delta(T)\log^c n)$ space, and answers inverse SA queries on T in $t_{\mathrm{ISA}}(|T|) = o(\frac{\log|T|}{\log\log|T|})$ time. We will prove that this implies that there exists a data structure answering range counting queries that contradicts Theorem 2.53.

Consider any array $A[1\,..\,n]$ containing a permutation of $\{1, 2, \ldots, n\}$, where $n \geq 1$. Let $T_A = (\bigodot_{i=1}^{n}(\mathtt{0}^{A[i]}\mathtt{1}^i)) \cdot \mathtt{0}^{n+1}\mathtt{1}^{n+1} \cdot (\bigodot_{i=1}^{n+1}(\mathtt{0}^{n+1}\mathtt{1}^i)) \in \{\mathtt{0},\mathtt{1}\}^*$ be a text defined as in Lemma A.1. Let $R[1\,..\,n]$ be an array defined so that, for every $v \in [1\,..\,n]$, $R[i] = \mathrm{RangeBeg}(\mathtt{0}^i\mathtt{1}, T_A)$ (Definition 2.1).

Let D_{count} denote the data structure consisting of the following components:

1. The array $R[1\,..\,n]$ stored in plain form. It uses $\mathcal{O}(n)$ space.
2. The data structure D_{ISA} for the string T_A. Note that

$$\begin{aligned}|T_A| &= (\textstyle\sum_{i=1}^{n}(A[i] + i)) + 2(n+1) + (\sum_{i=1}^{n+1}(n + 1 + i))\\ &= (\tfrac{5}{2}n + 4)(n+1) \in \mathcal{O}(n^2).\end{aligned}$$

   Moreover, note that $|\mathrm{RL}(T_A)| = 4(n+1)$. By applying Observation 2.19 and Theorem 2.37(1), we thus obtain $\delta(T_A) = \mathcal{O}(n)$. Consequently, D_{ISA} for T_A needs $\mathcal{O}(\delta(T_A)\log^c|T_A|) = \mathcal{O}(n\log^c n)$ space.

In total, D_{count} needs $\mathcal{O}(n\log^c n)$ space.

Given any $j \in [0\,..\,n]$ and $v \geq 0$, we compute $\mathsf{range\text{-}count}_A(j, v)$ as follows:

1. If $v < 1$ (resp. $v > n$), then in $\mathcal{O}(1)$ time we return that $\mathsf{range\text{-}count}_A(j, v) = j$ (resp. $\mathsf{range\text{-}count}_A(j, v) = 0$), and conclude the query algorithm. Let us thus assume that $v \in [1\,..\,n]$.
2. Using array R, in $\mathcal{O}(1)$ time we compute $b = \mathrm{RangeBeg}(\mathtt{0}^v\mathtt{1}, T_A) = R[v]$. In $\mathcal{O}(1)$ time we also compute $\ell_1 = n \cdot (n+1)$, $\ell_2 = 2 \cdot (n+1)$ $\delta = j \cdot (n + 1 + \frac{j+1}{2}) + (n + 2 - v)$, and $j' = \ell_1 + \ell_2 + \delta$.
3. Using D_{ISA}, in $\mathcal{O}(t_{\mathrm{ISA}}(|T_A|))$ time we compute $x = \mathrm{SA}_{T_A}^{-1}[j']$.
4. In $\mathcal{O}(1)$ time we set $y = x - (b + j + 1)$. By Lemma A.1, it holds $y = \mathsf{range\text{-}count}_A(j, v)$. Thus, we return y as the answer.

In total, the query time is

$$\mathcal{O}(t_{\mathrm{ISA}}(|T_A|)) = o\Big(\tfrac{\log|T_A|}{\log\log|T_A|}\Big) = o\Big(\tfrac{\log(n^2)}{\log\log(n^2)}\Big) = o\Big(\tfrac{\log n}{\log\log n}\Big).$$

We have thus proved that there exists a data structure that, for every array $A[1\,..\,n]$ containing a permutation of $\{1, \ldots, n\}$, uses $\mathcal{O}(n\log^c n) = \mathcal{O}(n\log^{\mathcal{O}(1)} n)$ space, and answers range counting queries on A in $o(\frac{\log n}{\log\log n})$ time. This contradicts Theorem 2.53. □

B Faster Longest Common Prefix Range Minimum Queries (LCP RMQ)

B.1 Problem Definition

INDEXING FOR LONGEST COMMON PREFIX RANGE MINIMUM QUERIES (LCP RMQ)

Input: A string $T \in \Sigma^n$.
Output: A data structure that, given any $b, e \in [0\,..\,n]$ such that $b < e$, returns the position $\arg\min_{i \in (b\,..\,e]} \mathrm{LCP}_T[i]$ (see Definition 2.11 and Remark 2.50).

B.2 Upper Bound

In this section, we present the modifications of the data structure of Gagie, Navarro, and Prezza [GNP20], that enable answering LCP RMQ queries in $\mathcal{O}(\frac{\log n}{\log\log n})$ time using $\mathcal{O}(r(T)\log^{2+\epsilon} n)$ space, for any constant $\epsilon \in (0,1)$ and any text $T \in \Sigma^n$ (see Definition 2.26). These improvements build upon the original structure described in [GNP20, Section 6.3], which achieves $\mathcal{O}(r(T)\log n)$ space and $\mathcal{O}(\log n)$ query time.

Theorem B.1 ([GNP20, Lemma 6.6]). *Let $T \in \Sigma^n$ be a nonempty text. Let $A[1\,..\,n]$ be an array defined by setting $A[1] = \mathrm{LCP}_T[1]$ and, for every $i \in [2\,..\,n]$, $A[i] = \mathrm{LCP}_T[i] - \mathrm{LCP}_T[i-1]$ (see Definition 2.11). There exists an SLP G (Definition 2.23) of size $|G| = \mathcal{O}(r(T)\log^2 n)$ (see Definition 2.26) and height $h = \mathcal{O}(\log n)$ (Definition 2.25) that expands to A, i.e., $L(G) = \{A\}$.*

Remark B.2. We remark that [GNP20, Lemma 6.6] proves a stronger result than above, i.e., it describes the construction of a so-called *run-length SLP (RLSLP)* G' of size $|G'| = \mathcal{O}(r(T)\log n)$ that expands to the differentially encoded LCP array A. However, this implies the existence of an equivalent (i.e., expanding to the same string) SLP of size $\mathcal{O}(r(T)\log^2 n)$. This expansion may increase the height of the grammar, but it can be rebalanced to height $\mathcal{O}(\log n)$ without asymptotically increasing its size by using [GJL21].

Corollary B.3. *Let $\epsilon \in (0,1)$ be a constant and $T \in \Sigma^n$ be a nonempty text. Let $A[1\,..\,n]$ be an array defined by setting $A[1] = \mathrm{LCP}_T[1]$ and, for every $i \in [2\,..\,n]$, $A[i] = \mathrm{LCP}_T[i] - \mathrm{LCP}_T[i-1]$ (see Definition 2.11). There exists an SLG $G = (V, \Sigma, R, S)$ (Definition 2.21) of size $|G| = \mathcal{O}(r(T)\log^{2+\epsilon} n)$ (see Definition 2.26) and height $h = \mathcal{O}(\frac{\log n}{\log\log n})$ (Definition 2.25) that expands to A, i.e., $L(G) = \{A\}$, and such that for some $\ell = \mathcal{O}(\log^\epsilon n)$ it holds $|\mathrm{rhs}_G(N)| \le \ell$ for all $N \in V$.*

Proof. Let $G_{\mathrm{in}} = (V, \Sigma, R_{\mathrm{in}}, S)$ be an SLP from Theorem B.1. Based on G_{in}, we will construct an SLG $G = (V, \Sigma, R, S)$ with the required properties. Our construction follows the technique described in [BCPT15, Theorem 2]. Let $k = \epsilon \cdot \log\log n$. For every $N \in V$, we define $\mathrm{rhs}_G(N)$ to be the string obtained by starting with $\mathrm{rhs}_{G_{\mathrm{in}}}(N)$, and iteratively expanding all variables k times (or less, if the expansion reaches a terminal symbol). Since G_{in} is an SLP, each step of the expansion at most doubles the length of the current string, and hence it holds $|\mathrm{rhs}_G(N)| \le \ell$, where $\ell = 2 \cdot 2^k = 2 \cdot \log^\epsilon n = \mathcal{O}(\log^\epsilon n)$. The resulting SLG G expands to the same string as G_{in}, the size of G satisfies $|G| = \mathcal{O}(|G_{\mathrm{in}}| \cdot \ell) = \mathcal{O}(r(T)\log^{2+\epsilon} n)$, and its height is equal to height of G_{in} divided by k, i.e., $\mathcal{O}(\frac{\log n}{\log\log n})$. □

Lemma B.4 (Based on [GNP20, Section 6.3]). *Denote $\Sigma = \mathbb{Z}$ and let $G = (V, \Sigma, R, S)$ be an SLG of size g (Definition 2.21) and height h (Definition 2.25) such that, for every $N \in V$, it holds $|\mathrm{rhs}_G(V)| \le \ell$. Denote $n = |\exp_G(S)|$, $k = |V|$, and $V = \{N_1, \ldots, N_k\}$. In the word RAM model with word size $w = \Omega(\log n)$, there exists a data structure of size $\mathcal{O}(g)$ that, given any $x \in [1\,..\,k]$ and $p \in [1\,..\,|\exp_G(N_x)|]$, in $\mathcal{O}(h \cdot (1 + \log_w \ell))$ time returns three values:*

- $B[1] + \cdots + B[p]$,
- $\min_{t \in [1\,..\,p]}(B[1] + \cdots + B[t])$, *and*
- $\arg\min_{t \in [1\,..\,p]}(B[1] + \cdots + B[t])$ *(see Remark 2.50),*

where $B = \exp_G(N_x)$.

Proof. We use the following definitions. For every $i \in [1\,..\,k]$, let $\ell_i = |\mathrm{rhs}_G(N_i)|$, and let $R_i[1\,..\,\ell_i]$ and $R'_i[1\,..\,\ell_i]$ denote arrays defined such that for every $j \in [1\,..\,\ell_i]$, $R_i[j] = \mathrm{rhs}_G(N_i)[j]$ and $R'_i[j] = \mathtt{0}$ holds if $\mathrm{rhs}_G(N_i)[j] \in \Sigma$, and $R_i[j] = j'$ and $R'_i[j] = \mathtt{1}$ holds if $\mathrm{rhs}_G(N_i)[j] \in V$ (and $j' \in \{1, \ldots, k\}$ is such that $\mathrm{rhs}_G(N_i)[j] = N_{j'}$). For every $i \in [1\,..\,k]$, we define $P_i^{\mathrm{len}}[1\,..\,\ell_i]$, $P_i^{\mathrm{sum}}[1\,..\,\ell_i]$, $P_i^{\mathrm{min}}[1\,..\,\ell_i]$, and $P_i^{\mathrm{pos}}[1\,..\,\ell_i]$, so that, for every $\delta \in [1\,..\,\ell_i]$, letting $B = \mathrm{rhs}_G(N_i)[1\,..\,\delta)$ and $E = \exp_G(B)$, it holds

$$\begin{aligned} P_i^{\mathrm{len}}[\delta] &= |E|, \\ P_i^{\mathrm{sum}}[\delta] &= E[1] + \cdots + E[|E|], \\ P_i^{\mathrm{min}}[\delta] &= \min\{E[1] + \cdots + E[t] : t \in [1\,..\,|E|]\} \cup \{\infty\}. \end{aligned}$$

If $P_i^{\mathrm{min}}[\delta] \neq \infty$, then we also define

$$P_i^{\mathrm{pos}}[\delta] = \arg\min_{t \in [1\,..\,|E|]}(E[1] + \cdots + E[t]).$$

Otherwise (i.e., if $\delta = 1$), we leave $P_i^{\mathrm{pos}}[\delta]$ undefined. For every $i \in [1\,..\,k]$, we define a sequence $(b_{i,j})_{j \in [0\,..\,\ell_i]}$ so that $b_{i,0} = -\infty$ and, for every $j \in [1\,..\,\ell_i]$, it holds

$$b_{i,j} = |\exp_G(\mathrm{rhs}_G(N_i)[1\,..\,j))| = P_i^{\mathrm{len}}[j].$$

Components The data structure consists of the following components:

1. For every $i \in [1\,..\,k]$, we store the predecessor data structure from Theorem 2.43 for the sequence $(b_{i,j})_{j \in [0\,..\,\ell_i]}$. The structure for $i \in [1\,..\,k]$ needs $\mathcal{O}(\ell_i)$ space (recall that $\ell_i \geq 1$), and hence in total all structures need $\mathcal{O}(\sum_{i \in [1\,..\,k]} \ell_i) = \mathcal{O}(g)$ space.
2. For every $i \in [1\,..\,k]$, we store the arrays $P_i^{\mathrm{len}}[1\,..\,\ell_i]$, $P_i^{\mathrm{sum}}[1\,..\,\ell_i]$, $P_i^{\mathrm{min}}[1\,..\,\ell_i]$, $P_i^{\mathrm{pos}}[1\,..\,\ell_i]$, $R_i[1\,..\,\ell_i]$, and $R'_i[1\,..\,\ell_i]$ in plain form. In total, they need $\mathcal{O}(\sum_{i \in [1\,..\,k]} \ell_i) = \mathcal{O}(g)$ space.

In total, the structure uses $\mathcal{O}(g)$ space.

Implementation of queries Let $x \in [1\,..\,k]$, $B = \exp_G(N_x)$, and $p \in [1\,..\,|B|]$. We now show how, given x and p, to compute the following three values:

- $s_{x,p} = B[1] + \cdots + B[p]$,
- $v_{x,p} = \min_{t \in [1\,..\,p]}(B[1] + \cdots + B[t])$, and
- $\delta_{x,p} = \arg\min_{t \in [1\,..\,p]}(B[1] + \cdots + B[t])$

in $\mathcal{O}(h \cdot (1 + \log_w \ell))$ time. Let $(a_i)_{i \in [0\,..\,q]}$ be the sequence of symbol identifiers corresponding to the nodes in the parse tree $\mathcal{T}_G(N_x)$ (see Definition 2.24) on the path from the root to the pth leftmost leaf, and $(\delta_i)_{i \in [1\,..\,q]}$ is such that, for every $i \in [1\,..\,q]$, node i on the path is the δ_ith leftmost child of node $i-1$ on the path. Formally, $(a_i)_{i \in [0\,..\,q]} \in \mathbb{Z}^{q+1}$ and $(\delta_i)_{i \in [1\,..\,q]} \in \mathbb{Z}^q_{>0}$, where $q \geq 1$, are the unique sequences satisfying the following conditions:

- $a_0 = x$,
- for every $i \in [1\,..\,q)$, $\mathrm{rhs}_G(N_{a_{i-1}})[\delta_i] = N_{a_i}$,
- $\mathrm{rhs}_G(N_{a_{q-1}})[\delta_q] = a_q \in \Sigma$,
- $\left(\sum_{i=1}^{q} |\exp_G(\mathrm{rhs}_G(N_{a_{i-1}})[1\,..\,\delta_i))|\right) + 1 = p$.

Note that then it holds:

$$\exp_G(N_x)[1\,..\,p] = \left(\bigodot_{i=1}^{q} \exp_G(\mathrm{rhs}_G(N_{a_{i-1}})[1\,..\,\delta_i))\right) \cdot a_q.$$

By definition of arrays P_i^{sum}, we thus have:

$$s_{x,p} = \left(\sum_{j \in [1\,..\,q]} P_{a_{j-1}}^{\mathrm{sum}}[\delta_j]\right) + a_q.$$

Moreover, by definition of arrays P_i^{sum} and P_i^{min}, we also have:

$$v_{x,p} = \min\left\{\min_{i \in [1\,..\,q]}\left(\sum_{j \in [1\,..\,i)} P_{a_{j-1}}^{\mathrm{sum}}[\delta_j]\right) + P_{a_{i-1}}^{\mathrm{min}}[\delta_i], s_{x,p}\right\}.$$

Finally, to compute the smallest prefix length $\delta_{x,p}$ that minimizes the prefix sum, i.e., the value $\arg\min_{t \in [1\,..\,p]}(B[1] + \cdots + B[t])$, we observe that:

- If $s_{x,p} < \min_{i\in[1\,..\,q]}(\sum_{j\in[1\,..\,i)} P^{\mathrm{sum}}_{a_{j-1}}[\delta_j]) + P^{\mathrm{min}}_{a_{i-1}}[\delta_i]$, then $\delta_{x,p} = p$.
- Otherwise, letting i' be the smallest index in $[1\,..\,q]$ such that $(\sum_{j\in[1\,..\,i')} P^{\mathrm{sum}}_{a_{j-1}}[\delta_j]) + P^{\mathrm{min}}_{a_{i'-1}}[\delta_{i'}] = v_{x,p}$, it holds by definition of arrays P^{len}_i and P^{pos}_i that

$$\delta_{x,p} = \Big(\sum_{j\in[1\,..\,i')} P^{\mathrm{len}}_{a_{j-1}}[\delta_j]\Big) + P^{\mathrm{pos}}_{a_{i'-1}}[\delta_{i'}].$$

We are now ready to present the query algorithm. Given x and p, we proceed as follows:

1. In the first step, we compute the arrays $a[0\,..\,q]$ and $\delta[1\,..\,q]$ containing the sequences $(a_i)_{i\in[0\,..\,q]}$ and $(\delta_i)_{i\in[1\,..\,q]}$ defined above. We initialize the current position $p_{\mathrm{cur}} := p$, current nonterminal number $x_{\mathrm{cur}} := x$, and current depth $d_{\mathrm{cur}} := 0$ in the parse tree $\mathcal{T}_G(N_x)$. We also set $a[d_{\mathrm{cur}}] := x_{\mathrm{cur}}$. We then repeat the following sequence of steps:
   (a) In $\mathcal{O}(1)$ time, we set $d_{\mathrm{cur}} := d_{\mathrm{cur}} + 1$.
   (b) Denote $i = x_{\mathrm{cur}}$. Using the structure from Theorem 2.43 for the sequence $(b_{i,j})_{j\in[0\,..\,\ell_i]}$, in $\mathcal{O}(1 + \log_w \ell_i) = \mathcal{O}(1 + \log_w \ell)$ time we compute $\delta \in [1\,..\,\ell_i]$ satisfying $b_{i,\delta} = \mathsf{pred}_C(p_{\mathrm{cur}})$, where $C = \{b_{i,j}\}_{j\in[1\,..\,\ell_i]}$ (note that $\delta \geq 1$ follows by $p_{\mathrm{cur}} \geq 1$ and $b_{i,1} = 0$). Note that the p_{cur}th leftmost leaf in the parse tree $\mathcal{T}_G(N_{x_{\mathrm{cur}}})$ is then in the subtree rooted in the δth leftmost child of the root of $\mathcal{T}_G(N_{x_{\mathrm{cur}}})$.
   (c) In $\mathcal{O}(1)$ time, we set $\delta[d_{\mathrm{cur}}] := \delta$.
   (d) In $\mathcal{O}(1)$ time, we determine if $\mathrm{rhs}_G(N_{x_{\mathrm{cur}}})[\delta] \in \Sigma$ by checking if $R'_{x_{\mathrm{cur}}}[\delta] = \mathtt{0}$. We then consider two cases:
      - If $R'_{x_{\mathrm{cur}}}[\delta] = \mathtt{0}$, then in $\mathcal{O}(1)$ time we first compute $c := \mathrm{rhs}_G(N_{x_{\mathrm{cur}}})[\delta] = R_{x_{\mathrm{cur}}}[\delta]$ and set $a[d_{\mathrm{cur}}] := c$. We then terminate the query algorithm with $q = d_{\mathrm{cur}}$.
      - Otherwise, in $\mathcal{O}(1)$ time we compute the index x_{next} satisfying $\mathrm{rhs}_G(N_{x_{\mathrm{cur}}})[\delta] = N_{x_{\mathrm{next}}}$ by looking up $x_{\mathrm{next}} = R_{x_{\mathrm{cur}}}[\delta]$. We then set $a[d_{\mathrm{cur}}] := x_{\mathrm{next}}$. Finally, in preparation for the next iteration, we set $p_{\mathrm{cur}} := p_{\mathrm{cur}} - P^{\mathrm{len}}_{x_{\mathrm{cur}}}[\delta]$ and $x_{\mathrm{cur}} := x_{\mathrm{next}}$.

   Each of the iterations takes $\mathcal{O}(1 + \log_w \ell)$. In total we thus spend $\mathcal{O}(q \cdot (1 + \log_w \ell))$ time.
2. In the second step, we compute the value $s_{x,p}$, i.e., $B[1] + \cdots + B[p]$.
   (a) First, we compute an array $A_{\mathrm{sum}}[0\,..\,q]$ defined by $A_{\mathrm{sum}}[i] = \sum_{j\in[1\,..\,i]} P^{\mathrm{sum}}_{a_{j-1}}[\delta_j]$. This is easily done in $\mathcal{O}(q)$ time: first set $A_{\mathrm{sum}}[0] = 0$, and then, for every $j \in [1\,..\,q]$, $A_{\mathrm{sum}}[j] = A_{\mathrm{sum}}[j-1] + P^{\mathrm{sum}}_{a[j-1]}[\delta[j]]$.
   (b) We then return that $s_{x,p} = A_{\mathrm{sum}}[q] + a[q]$.

   In total, this takes $\mathcal{O}(q)$ time.
3. In the third step, we compute $\min_{t\in[1\,..\,p]}(B[1] + \cdots + B[t])$. To this end, in $\mathcal{O}(q)$ time we compute the value of the expression

$$\min\Big\{A_{\mathrm{sum}}[0] + P^{\mathrm{min}}_{a[0]}[\delta[1]], \ldots, A_{\mathrm{sum}}[q-1] + P^{\mathrm{min}}_{a[q-1]}[\delta[q]], s_{x,p}\Big\}.$$

   By the above discussion, the result is equal to $v_{x,p}$, i.e., $\min_{t\in[1\,..\,p]}(B[1] + \cdots + B[t])$. Computing $v_{x,p}$ thus takes $\mathcal{O}(q)$ time.
4. In the fourth step, we compute $\delta_{x,p}$, i.e., $\arg\min_{t\in[1\,..\,p]}(B[1] + \cdots + B[t])$. First, in $\mathcal{O}(q)$ time we compute $v = \min\{A_{\mathrm{sum}}[0] + P^{\mathrm{min}}_{a[0]}[\delta[1]], \ldots, A_{\mathrm{sum}}[q-1] + P^{\mathrm{min}}_{a[q-1]}[\delta[q]]\}$ (where A_{sum} is the array computed above).
   - If $v > s_{x,p}$, then by the above discussion we have $\arg\min_{t\in[1\,..\,p]}(B[1] + \cdots + B[t]) = p$.
   - Let us now assume that $v \leq s_{x,p}$. In $\mathcal{O}(q)$ time, we compute the smallest index $i' \in [1\,..\,q]$ that satisfies $A_{\mathrm{sum}}[i'-1] + P^{\mathrm{min}}_{a[i'-1]}[\delta[i']] = v_{x,p}$ (the latter value was computed above). In $\mathcal{O}(q)$ time, we then obtain $\delta_{x,p} = \big(\sum_{j\in[1\,..\,i')} P^{\mathrm{len}}_{a[j-1]}[\delta[j]]\big) + P^{\mathrm{pos}}_{a[i'-1]}[\delta[i']]$. The correctness of this computation follows by the above discussion.

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

In total, the computation of $s_{x,p}$, $v_{x,p}$, and $\delta_{x,p}$ takes $\mathcal{O}(q \cdot (1 + \log_w \ell)) = \mathcal{O}(h \cdot (1 + \log_w \ell))$ time. □

Remark B.5. The next result shows how to compute the largest prefix sum on the suffix of the expansion of the nonterminal in a grammar. Note that although the data structure is similar to the one in Lemma B.4, it is not symmetric. This is because this time we are investigating the *suffix* of a nonterminal expansion, but still ask about the largest *prefix* sum of that substring, making the details of the query slightly different.

Lemma B.6 (Based on [GNP20, Section 6.3]). *Denote $\Sigma = \mathbb{Z}$ and let $G = (V, \Sigma, R, S)$ be an SLG of size g (Definition 2.21) and height h (Definition 2.25) such that, for every $N \in V$, it holds $|\mathrm{rhs}_G(V)| \leq \ell$. Denote $n = |\exp_G(S)|$, $k = |V|$, and $V = \{N_1, \ldots, N_k\}$. In the word RAM model with word size $w = \Omega(\log n)$, there exists a data structure of size $\mathcal{O}(g)$ that, given any $x \in [1\,..\,k]$ and $p \in [1\,..\,m]$ (where $m = |\exp_G(N_x)|$), in $\mathcal{O}(h \cdot (1 + \log_w \ell))$ time returns three values:*

- $B[m-p+1] + \cdots + B[m]$,
- $\min_{t \in [1\,..\,p]}(B[m-p+1] + \cdots + B[m-p+t])$, *and*
- $\arg\min_{t \in [1\,..\,p]}(B[m-p+1] + \cdots + B[m-p+t])$ *(see Remark 2.50),*

where $B = \exp_G(N_x)$.

Proof. We use the following definitions. For every $i \in [1\,..\,k]$, let $\ell_i = |\mathrm{rhs}_G(N_i)|$, and let $R_i[1\,..\,\ell_i]$ and $R'_i[1\,..\,\ell_i]$ denote arrays defined such that for every $j \in [1\,..\,\ell_i]$, $R_i[j] = \mathrm{rhs}_G(N_i)[j]$ and $R'_i[j] = \mathtt{0}$ holds if $\mathrm{rhs}_G(N_i)[j] \in \Sigma$, and $R_i[j] = j'$ and $R'_i[j] = \mathtt{1}$ holds if $\mathrm{rhs}_G(N_i)[j] \in V$ (and $j' \in \{1, \ldots, k\}$ is such that $\mathrm{rhs}_G(N_i)[j] = N_{j'}$). For every $i \in [1\,..\,k]$, we define $S_i^{\mathrm{len}}[1\,..\,\ell_i]$, $S_i^{\mathrm{sum}}[1\,..\,\ell_i]$, $S_i^{\mathrm{min}}[1\,..\,\ell_i]$, and $S_i^{\mathrm{pos}}[1\,..\,\ell_i]$, so that, for every $\delta \in [1\,..\,\ell_i]$, letting $B = \mathrm{rhs}_G(N_i)(\delta\,..\,|\mathrm{rhs}_G(N_i)|]$ and $E = \exp_G(B)$, it holds

$$\begin{aligned} S_i^{\mathrm{len}}[\delta] &= |E|, \\ S_i^{\mathrm{sum}}[\delta] &= E[1] + \cdots + E[|E|], \\ S_i^{\mathrm{min}}[\delta] &= \min\{E[1] + \cdots + E[t] : t \in [1\,..\,|E|]\} \cup \{\infty\}. \end{aligned}$$

If $S_i^{\mathrm{min}}[\delta] \neq \infty$, then we also define

$$S_i^{\mathrm{pos}}[\delta] = \arg\min_{t \in [1\,..\,|E|]}(E[1] + \cdots + E[t]).$$

Otherwise (i.e., if $\delta = |\mathrm{rhs}_G(N_i)|$), we leave $S_i^{\mathrm{pos}}[\delta]$ undefined. For every $i \in [1\,..\,k]$, we define a sequence $(b_{i,j})_{j \in [0\,..\,\ell_i]}$ so that $b_{i,0} = -\infty$ and, for every $j \in [1\,..\,\ell_i]$, it holds

$$b_{i,j} = |\exp_G(\mathrm{rhs}_G(N_i)[1\,..\,j))| = S_i^{\mathrm{len}}[j].$$

We define $A_{\mathrm{explen}}[1\,..\,k]$ so that, for every $i \in [1\,..\,k]$, $A_{\mathrm{explen}}[i] = |\exp_G(N_i)|$.

Components The data structure consists of the following components:

1. For every $i \in [1\,..\,k]$, we store the predecessor data structure from Theorem 2.43 for the sequence $(b_{i,j})_{j \in [0\,..\,\ell_i]}$. The structure for $i \in [1\,..\,k]$ needs $\mathcal{O}(\ell_i)$ space (recall that $\ell_i \geq 1$), and hence in total all structures need $\mathcal{O}(\sum_{i \in [1\,..\,k]} \ell_i) = \mathcal{O}(g)$ space.
2. For every $i \in [1\,..\,k]$, we store the arrays $S_i^{\mathrm{len}}[1\,..\,\ell_i]$, $S_i^{\mathrm{sum}}[1\,..\,\ell_i]$, $S_i^{\mathrm{min}}[1\,..\,\ell_i]$, $S_i^{\mathrm{pos}}[1\,..\,\ell_i]$, $R_i[1\,..\,\ell_i]$, and $R'_i[1\,..\,\ell_i]$ in plain form. In total, they need $\mathcal{O}(\sum_{i \in [1\,..\,k]} \ell_i) = \mathcal{O}(g)$ space.
3. We store the array $A_{\mathrm{explen}}[1\,..\,k]$ in plan form using $\mathcal{O}(k) = \mathcal{O}(g)$ space.

In total, the structure uses $\mathcal{O}(g)$ space.

Implementation of queries Let $x \in [1\,..\,k]$, $B = \exp_G(N_x)$, $m = |B|$, and $p \in [1\,..\,m]$. We now show how, given x and p, to compute the following three values:

- $s_{x,p} = B[m-p+1] + \cdots + B[m]$,
- $v_{x,p} = \min_{t \in [1\,..\,p]}(B[m-p+1] + \cdots + B[m-p+t])$, and
- $\delta_{x,p} = \arg\min_{t \in [1\,..\,p]}(B[m-p+1] + \cdots + B[m-p+t])$

in $\mathcal{O}(h \cdot (1 + \log_w \ell))$ time. Let $(a_i)_{i \in [0\,..\,q]}$ be the sequence of symbol identifiers corresponding to the nodes in the parse tree $\mathcal{T}_G(N_x)$ (see Definition 2.24) on the path from the root to the pth rightmost leaf, and $(\delta_i)_{i \in [1\,..\,q]}$ is such that, for every $i \in [1\,..\,q]$, node i on the path is the δ_ith leftmost child of node $i-1$ on the path. Formally, $(a_i)_{i \in [0\,..\,q]} \in \mathbb{Z}^{q+1}$ and $(\delta_i)_{i \in [1\,..\,q]} \in \mathbb{Z}^q_{>0}$, where $q \geq 1$, are the unique sequences satisfying the following conditions:

- $a_0 = x$,

- for every $i \in [1\,..\,q)$, $\mathrm{rhs}_G(N_{a_{i-1}})[\delta_i] = N_{a_i}$,
- $\mathrm{rhs}_G(N_{a_{q-1}})[\delta_q] = a_q \in \Sigma$,
- $\big(\sum_{i=1}^{q} |\exp_G(\mathrm{rhs}_G(N_{a_{i-1}})(\delta_i\,..\,|\mathrm{rhs}_G(N_{a_{i-1}})|])|\big) + 1 = p$.

Note that then it holds:

$$\exp_G(N_x)(m-p\,..\,m] = a_q \cdot \Big(\bigodot\nolimits_{i=q,\ldots,1} \exp_G(\mathrm{rhs}_G(N_{a_{i-1}})(\delta_i\,..\,|\mathrm{rhs}_G(N_{a_{i-1}})|])\Big).$$

By definition of arrays S_i^{sum}, we thus have:

$$s_{x,p} = \Big(\textstyle\sum_{j\in[1\,..\,q]} S^{\mathrm{sum}}_{a_{j-1}}[\delta_j]\Big) + a_q.$$

Moreover, by definition of arrays S_i^{sum} and S_i^{min}, we also have:

$$v_{x,p} = \min\Big\{a_q, \min_{i\in[1\,..\,q]}\Big(a_q + \textstyle\sum_{j\in(i\,..\,q]} S^{\mathrm{sum}}_{a_{j-1}}[\delta_j]\Big) + S^{\mathrm{min}}_{a_{i-1}}[\delta_i]\Big\}.$$

Finally, to compute the smallest prefix length $\delta_{x,p}$ that minimizes the prefix sum, i.e., the value $\arg\min_{t\in[1\,..\,p]}(B[m-p+1]+\cdots+B[m-p+t])$, we observe that:

- If $a_q < \min_{i\in[1\,..\,q]}(a_q + \sum_{j\in(i\,..\,q]} S^{\mathrm{sum}}_{a_{j-1}}[\delta_j]) + S^{\mathrm{min}}_{a_{i-1}}[\delta_i]$, then $\delta_{x,p} = 1$.
- Otherwise, letting i' be the largest index in $[1\,..\,q]$ such that $(a_q + \sum_{j\in(i\,..\,q]} S^{\mathrm{sum}}_{a_{j-1}}[\delta_j]) + S^{\mathrm{min}}_{a_{i-1}}[\delta_i] = v_{x,p}$, it holds by definition of arrays S_i^{len} and S_i^{pos} that

$$\delta_{x,p} = 1 + \Big(\textstyle\sum_{j\in(i'\,..\,q]} S^{\mathrm{len}}_{a_{j-1}}[\delta_j]\Big) + S^{\mathrm{pos}}_{a_{i'-1}}[\delta_{i'}].$$

We are now ready to present the query algorithm. Given x and p, we proceed as follows:

1. In the first step, we compute the arrays $a[0\,..\,q]$ and $\delta[1\,..\,q]$ containing the sequences $(a_i)_{i\in[0\,..\,q]}$ and $(\delta_i)_{i\in[1\,..\,q]}$ defined above. The query algorithm proceeds the same as in the proof of Lemma B.6, except at the beginning of the algorithm, we initialize $p_{\mathrm{cur}} := A_{\mathrm{explen}}[x] - p + 1$. The computation of the two arrays takes $\mathcal{O}(q\cdot(1+\log_w \ell))$ time.
2. In the second step, we compute the value $s_{x,p}$, i.e., $B[m-p+1]+\cdots+B[m]$.
   (a) First, we compute an array $A_{\mathrm{sum}}[0\,..\,q]$ defined by $A_{\mathrm{sum}}[i] = \sum_{j\in(i\,..\,q]} S^{\mathrm{sum}}_{a_{j-1}}[\delta_j]$. This is easily done in $\mathcal{O}(q)$ time: first set $A_{\mathrm{sum}}[q] = 0$, and then, for every $i = q-1,\ldots,0$, set $A_{\mathrm{sum}}[i] = A_{\mathrm{sum}}[i+1] + S^{\mathrm{sum}}_{a[i]}[\delta[i+1]]$.
   (b) We then return that $s_{x,p} = A_{\mathrm{sum}}[0] + a[q]$.

   In total, this takes $\mathcal{O}(q)$ time.
3. In the third step, we compute $\min_{t\in[1\,..\,p]}(B[m-p+1]+\cdots+B[m-p+t])$. To this end, in $\mathcal{O}(q)$ time we compute the value of the expression

$$\min\Big\{a[q], a[q]+A_{\mathrm{sum}}[1]+S^{\mathrm{min}}_{a[0]}[\delta[1]], \ldots, a[q]+A_{\mathrm{sum}}[q]+S^{\mathrm{min}}_{a[q-1]}[\delta[q]], \Big\}.$$

   By the above discussion, the result is equal to $v_{x,p}$, i.e., $\min_{t\in[1\,..\,p]}(B[m-p+1]+\cdots+B[m-p+t])$. Computing $v_{x,p}$ thus takes $\mathcal{O}(q)$ time.
4. In the fourth step, we compute $\delta_{x,p}$, i.e., $\arg\min_{t\in[1\,..\,p]}(B[m-p+1]+\cdots+B[m-p+t])$. First, in $\mathcal{O}(q)$ time we compute $v = \min\{a[q]+A_{\mathrm{sum}}[1]+S^{\mathrm{min}}_{a[0]}[\delta[1]], \ldots, a[q]+A_{\mathrm{sum}}[q]+S^{\mathrm{min}}_{a[q-1]}[\delta[q]]\}$ (where A_{sum} is the array computed above).
   - If $a[q] < v$, then by the above discussion we have $\arg\min_{t\in[1\,..\,p]}(B[m-p+1]+\cdots+B[m-p+t]) = 1$.
   - Let us now assume that $a[q] \geq v$. In $\mathcal{O}(q)$ time, we compute the largest index $i' \in [1\,..\,q]$ that satisfies $a[q] + A_{\mathrm{sum}}[i'] + S^{\mathrm{min}}_{a[i'-1]}[\delta[i']] = v_{x,p}$ (the latter value was computed above). In $\mathcal{O}(q)$ time, we then obtain $\delta_{x,p} = 1 + \big(\sum_{j\in(i'\,..\,q]} S^{\mathrm{len}}_{a[j-1]}[\delta[j]]\big) + S^{\mathrm{pos}}_{a[i'-1]}[\delta[i']]$. The correctness of this computation follows by the above discussion.

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

In total, the computation of $s_{x,p}$, $v_{x,p}$, and $\delta_{x,p}$ takes $\mathcal{O}(q\cdot(1+\log_w \ell)) = \mathcal{O}(h\cdot(1+\log_w \ell))$ time. □

Lemma B.7 (Based on [GNP20, Section 6.3]). *Denote $\Sigma = \mathbb{Z}$ and let $G = (V, \Sigma, R, S)$ be an SLG of size g (Definition 2.21) and height h (Definition 2.25) such that, for every $N \in V$, it holds $|\mathrm{rhs}_G(V)| \leq \ell$. Denote*

$A = \exp_G(S)$ and $n = |A|$. In the word RAM model with word size $w = \Omega(\log n)$, there exists a data structure of size $\mathcal{O}(g)$ that, given any $b, e \in [0\,..\,n]$ satisfying $b < e$, in $\mathcal{O}(h \cdot (1 + \log_w \ell))$ time returns (see Remark 2.50)

$$\arg\min_{i \in (b\,..\,e]} (A[1] + \cdots + A[i]).$$

Proof. We use the following definitions. Denote $k = |V|$ and $V = \{N_1, \ldots, N_k\}$. For every $i \in [1\,..\,k]$, let $\ell_i = |\mathrm{rhs}_G(N_i)|$, and let $R_i[1\,..\,\ell_i]$ denote an array defined such that for every $j \in [1\,..\,\ell_i]$, $R_i[j] = \mathrm{rhs}_G(N_i)[j]$ holds if $\mathrm{rhs}_G(N_i)[j] \in \Sigma$, and $R_i[j] = j'$ if $\mathrm{rhs}_G(N_i)[j] \in V$ (and $j' \in \{1, \ldots, k\}$ is such that $\mathrm{rhs}_G(N_i)[j] = N_{j'}$). For every $i \in [1\,..\,k]$, we define $P_i^{\mathrm{len}}[1\,..\,\ell_i + 1]$ and $P_i^{\mathrm{sum}}[1\,..\,\ell_i + 1]$ so that, for every $\delta \in [1\,..\,\ell_i + 1]$, letting $B = \mathrm{rhs}_G(N_i)[1\,..\,\delta)$ and $E = \exp_G(B)$, it holds:

$$P_i^{\mathrm{len}}[\delta] = |E|,$$
$$P_i^{\mathrm{sum}}[\delta] = E[1] + \cdots + E[|E|].$$

For every $i \in [1\,..\,k]$, we define a sequence $(b_{i,j})_{j \in [0\,..\,\ell_i + 1]}$ so that $b_{i,0} = -\infty$ and, for every $j \in [1\,..\,\ell_i + 1]$, it holds

$$b_{i,j} = |\exp_G(\mathrm{rhs}_G(N_i)[1\,..\,j))| = P_i^{\mathrm{len}}[j].$$

For every $i \in [1\,..\,k]$, we define $M_i^{\min}[1\,..\,\ell_i]$ and $M_i^{\mathrm{pos}}[1\,..\,\ell_i]$ so that, for every $\delta \in [1\,..\,\ell_i]$, letting $E = \exp_G(N_i)$, $\ell_{\mathrm{beg}} = b_{i,\delta}$, and $\ell_{\mathrm{end}} = b_{i,\delta+1}$, it holds:

$$M_i^{\min}[\delta] = \min_{t \in (\ell_{\mathrm{beg}}\,..\,\ell_{\mathrm{end}}]} E[1] + \cdots + E[t],$$
$$M_i^{\mathrm{pos}}[\delta] = \arg\min_{t \in (\ell_{\mathrm{beg}}\,..\,\ell_{\mathrm{end}}]} E[1] + \cdots + E[t].$$

Let $s \in [1\,..\,k]$ be such that $S = N_s$ is the start symbol of G.

Components The data structure consists of the following components:

1. For every $i \in [1\,..\,k]$, we store the predecessor data structure from Theorem 2.43 for the sequence $(b_{i,j})_{j \in [0\,..\,\ell_i + 1]}$. The structure for $i \in [1\,..\,k]$ needs $\mathcal{O}(\ell_i)$ space (recall that $\ell_i \geq 1$), and hence in total all structures need $\mathcal{O}(\sum_{i=1}^{k} \ell_i) = \mathcal{O}(g)$ space.
2. For every $i \in [1\,..\,k]$, we store the RMQ data structure from Theorem 2.52 for the array $M_i^{\min}[1\,..\,\ell_i]$. The structure for $i \in [1\,..\,k]$ needs $\mathcal{O}(\ell_i)$ space, and hence in total all structures need $\mathcal{O}(\sum_{i=1}^{k} \ell_i) = \mathcal{O}(g)$ space.
3. For every $i \in [1\,..\,k]$, we store the arrays $R_i[1\,..\,\ell_i]$, $P_i^{\mathrm{len}}[1\,..\,\ell_i + 1]$, $P_i^{\mathrm{sum}}[1\,..\,\ell_i + 1]$, $M_i^{\min}[1\,..\,\ell_i]$, and $M_i^{\mathrm{pos}}[1\,..\,\ell_i]$ in plain form. In total, they need $\mathcal{O}(\sum_{i=1}^{k} \ell_i) = \mathcal{O}(g)$ space.
4. The data structure from Lemma B.4 for the SLG G. It uses $\mathcal{O}(g)$ space.
5. The data structure from Lemma B.6 for the SLG G. It also uses $\mathcal{O}(g)$ space.

In total, the structure uses $\mathcal{O}(g)$ space.

Implementation of queries Let $b, e \in [0\,..\,n]$ be such that $b < e$. We now show how, given b and e, to compute the position $\arg\min_{i \in (b\,..\,e]} (A[1] + \cdots + A[i])$ in $\mathcal{O}(h \cdot (1 + \log_w \ell))$ time. We proceed as follows:

1. In the first step, we perform a traversal in the parse tree of G to decompose the input interval $(b\,..\,e]$ into simpler form. We initialize $b' := b$, $e' := e$, and $x := s$. We then repeat the following sequence of steps:
   (a) Using Theorem 2.43 for the sequence $(b_{x,t})_{t \in [1\,..\,\ell_x + 1]}$, in $\mathcal{O}(1 + \log_w \ell_x) = \mathcal{O}(1 + \log_w \ell)$ time we compute $i \in [0\,..\,\ell_x + 1]$ satisfying $b_{x,i} = \mathsf{pred}_C(b')$, where $C = \{b_{x,t}\}_{t \in [1\,..\,\ell_x + 1]}$.
   (b) Analogously, in $\mathcal{O}(1 + \log_w \ell)$ time compute $j \in [0\,..\,\ell_x + 1]$ satisfying $b_{x,j} = \mathsf{pred}_C(e' + 1)$.
   (c) We then consider two cases:
   - If $i = j$, then it holds $b_{x,i} < b' < e' < b_{x,i+1}$ and hence we continue the traversal. Note that this implies tat $i \geq 1$ and $\mathrm{rhs}_G(N_x)[i] \in V$. In preparation for the next iteration, we set $b' := b' - b_{x,i} = b' - P_x^{\mathrm{len}}[i]$, $e' := e' - b_{x,i} = e' - P_x^{\mathrm{len}}[i]$, and $x := R_x[i]$.
   - If $i < j$, we stop the downward traversal. Observe that at the end of the traversal, the variables b', e', x, i, and j satisfy the following conditions:

- $0 \le b' < e' \le |\exp_G(N_x)|$,
- $\exp_G(N_x)(b' \,..\, e'] = A(b \,..\, e]$,
- $0 \le i < j \le \ell_x + 1$,
- $b_{x,i} < b' \le b_{x,i+1}$,
- $b_{x,j} \le e' < b_{x,j+1}$.

In each step, we spend $\mathcal{O}(1 + \log_w \ell)$ time, and descend one level in the parse tree of G. Thus, the above traversal takes $\mathcal{O}(h \cdot (1 + \log_w \ell))$ time.

2. In the next step, we compute $\arg\min_{i \in (b. \,..\, e]} (A[1] + \cdots + A[i])$. The central properties that we will use are:

   - First, we observe that by $\exp_G(N_x)(b' \,..\, e'] = A(b \,..\, e]$, letting $B = \exp_G(N_x)(b' \,..\, e']$, it holds

   $$\arg\min_{t \in (b. \,..\, e]} (A[1] + \cdots + A[t]) = b + \arg\min_{t \in [1. \,..\, e' - b']} (B[1] + \cdots + B[t]).$$

   This holds because each of the prefix sums on the right can be mapped to exactly one prefix sum on the left such that they differ by exactly $A[1] + \cdots + A[b]$. Thus, the position of the minimum is not affected.
   - Second, note that we can write the interval $(b' \,..\, e']$ as a disjoint union (with either of the three subintervals possibly empty)

   $$(b' \,..\, e'] = (b' \,..\, b_{x,i+1}] \cup (b_{x,i+1} \,..\, b_{x,j}] \cup (b_{x,j} \,..\, e'].$$

   The query proceeds as follows:

   (a) In $\mathcal{O}(1)$ time we initialize the minimum prefix sum to $v_{\text{cur}} = \infty$, the length of the prefix corresponding to this prefix sum to $\delta_{\text{cur}} = 0$, and the sum and length of the prefix examined so far to $s_{\text{cur}} = 0$ and $p_{\text{cur}} = 0$. We also set $p_L := P_x^{\text{len}}[i+1] - b'$, $p_M := P_x^{\text{len}}[j] - P_x^{\text{len}}[i+1]$, and $p_R := e' - P_x^{\text{len}}[j]$ to be lengths of the three subintervals in the above decomposition.

   (b) If $p_L = 0$, then we skip this step. Let us thus assume that $p_L > 0$. Observe that since above we noted that $b_{x,i} < b' \le b_{x,i+1}$, it follows by $p_L > 0$ that $|\exp_G(N_x)[i]| = b_{x,i+1} - b_{x,i} = (b_{x,i+1} - b') + (b' - b_{x,i}) = p_L + (b' - b_{x,i}) \ge 2$, and hence $\text{rhs}_G(N_x)[i] \in V$.

      i. In $\mathcal{O}(1)$ time we compute $x_L := R_x[i]$. We then have $\text{rhs}_G(N_x)[i] = N_{x_L}$. Denote $B_L = \exp_G(N_{x_L})$ and $m_L = |B_L|$.

      ii. Using Lemma B.6 and x_L and p_L and input, in $\mathcal{O}(h \cdot (1 + \log_w \ell))$ time we compute three values:

      - $s_L = B_L[m_L - p_L + 1] + \cdots + B_L[m_L]$,
      - $v_L = \min_{t \in [1. \,..\, p_L]} B_L[m_L - p_L + 1] + \cdots + B_L[m_L - p_L + t]$,
      - $\delta_L = \arg\min_{t \in [1. \,..\, p_L]} B_L[m_L - p_L + 1] + \cdots + B_L[m_L - p_L + t]$.

      iii. We then update the values recording the currently smallest prefix sum and the auxiliary statistics by setting $v_{\text{cur}} := v_L$, $\delta_{\text{cur}} = \delta_L$, $s_{\text{cur}} := s_L$, and $p_{\text{cur}} := p_L$.

   In total, above steps take $\mathcal{O}(h \cdot (1 + \log_w \ell))$ time.

   (c) If $p_M = 0$, then we skip this step. Let us thus assume that $p_M > 0$. Note that this implies that $i < j - 1$.

      i. Using Theorem 2.52, in $\mathcal{O}(1)$ time we compute $t_{\text{child}} = \arg\min_{t \in (i. \,..\, j-1]} M_x^{\min}[t]$.

      ii. In $\mathcal{O}(1)$ time we set $s_M := P_x^{\text{sum}}[j] - P_x^{\text{sum}}[i+1]$, $v_M := M_x^{\min}[t_{\text{child}}] - P_x^{\text{sum}}[i+1]$, and $\delta_M := M_x^{\text{pos}}[t_{\text{child}}] - P_x^{\text{len}}[i+1]$. Denoting $B_M = \exp_G(\text{rhs}_G(N_x)(i \,..\, j-1])$ and $m_M = |B_M|$, we then have:

      - $s_M = B_M[1] + \cdots + B_M[m_M]$,
      - $v_M = \min_{t \in [1. \,..\, m_M]}(B_M[1] + \cdots + B_M[t])$,
      - $\delta_M = \arg\min_{t \in [1. \,..\, m_M]}(B_M[1] + \cdots + B_M[t])$.

      iii. In $\mathcal{O}(1)$ time we update the values corresponding to the smallest prefix. More precisely, if $s_{\text{cur}} + v_M < v_{\text{cur}}$, then we set $v_{\text{cur}} := s_{\text{cur}} + v_M$ and $\delta_{\text{cur}} := p_{\text{cur}} + \delta_M$.

      iv. We update the remaining variables by setting $s_{\text{cur}} := s_{\text{cur}} + s_M$ and $p_{\text{cur}} := p_{\text{cur}} + p_M$.

In total, above steps take $\mathcal{O}(1)$ time.

(d) If $p_R = 0$, then we skip this step. Let us thus assume that $p_R > 0$. Observe that since above we noted that $b_{x,j} \leq e' < b_{x,j+1}$, it follows by $p_R > 0$ that $|\exp_G(N_x)[j]| = b_{x,j+1} - b_{x,j} = (b_{x,j+1} - e') + (e' - b_{x,j}) = (b_{x,j+1} - e') + p_R \geq 2$, and hence $\mathrm{rhs}_G(N_x)[j] \in V$.

i. In $\mathcal{O}(1)$ time we compute $x_R := R_x[j]$. We then have $\mathrm{rhs}_G(N_x)[j] = N_{x_R}$. Denote $B_R = \exp_G(N_{x_R})$ and $m_R = |B_R|$.

ii. Using Lemma B.4 and x_R and p_R and input, in $\mathcal{O}(h \cdot (1 + \log_w \ell))$ time we compute three values:

- $s_R = B_R[1] + \cdots + B_R[p_R]$,
- $v_R = \min_{t \in [1\,..\,p_R]} B_R[1] + \cdots + B_R[t]$,
- $\delta_R = \arg\min_{t \in [1\,..\,p_R]} B_R[1] + \cdots + B_R[t]$.

iii. In $\mathcal{O}(1)$ time we update the values corresponding to the smallest prefix. More precisely, if $s_{\mathrm{cur}} + v_R < v_{\mathrm{cur}}$, then we set $v_{\mathrm{cur}} := s_{\mathrm{cur}} + v_R$ and $\delta_{\mathrm{cur}} := p_{\mathrm{cur}} + \delta_R$.

In total, above steps take $\mathcal{O}(h \cdot (1 + \log_w \ell))$ time.

(e) At this point, we have $\delta_{\mathrm{cur}} = \arg\min_{t \in [1\,..\,e'-b']} (B[1] + \cdots + B[t])$, where B is defined as above, i.e., $B = \exp_G(N_x)(b'\,..\,e']$. Thus, by the above discussion, in $\mathcal{O}(1)$ we return that $\arg\min_{t \in (b\,..\,e]}(A[1] + \cdots + A[t]) = b + \delta_{\mathrm{cur}}$.

In total, the above steps take $\mathcal{O}(h \cdot (1 + \log_w \ell))$ time.

In total, the whole query algorithm takes $\mathcal{O}(h \cdot (1 + \log_w \ell))$ time. □

Theorem B.8 (Based on [GNP20, Section 6.3]). *Let $\epsilon \in (0, 1)$ be a constant. For every nonempty set $T \in \Sigma^n$, there exists a data structure of size $\mathcal{O}(r(T) \log^{2+\epsilon})$ (see Definition 2.26) that answers LCP RMQ queries on T (that, given any $b, e \in [0\,..\,n]$ satisfying $b < e$, return $\arg\min_{i \in (b\,..\,e]} \mathrm{LCP}_T[i]$; see Definition 2.11 and Remark 2.50) in $\mathcal{O}(\frac{\log n}{\log \log n})$ time.*

Proof. We use the following definitions. Let $A[1\,..\,n]$ be an array defined such that $A[1] = \mathrm{LCP}_T[1]$ and, for every $i \in [2\,..\,n]$, $A[i] = \mathrm{LCP}_T[i] - \mathrm{LCP}_T[i-1]$. Let $G = (V, \Sigma, R, S)$ denote the SLG from Corollary B.3. Recall that G has size $g = \mathcal{O}(r(T) \log^{2+\epsilon} n)$, it has height $h = \mathcal{O}(\frac{\log n}{\log \log n})$, it expands to A (i.e., $L(G) = \{A\}$), and there exists $\ell = \mathcal{O}(\log^{\epsilon} n)$ such that, for every $N \in V$, it holds $|\mathrm{rhs}_G(V)| \leq \ell$.

Components The data structure to answer LCP RMQ queries consists of a single component: the data structure from Lemma B.7 for the SLG G. The data structure from Lemma B.7 (and hence the entire structure for LCP RMQ queries) needs $\mathcal{O}(g) = \mathcal{O}(r(T) \log^{2+\epsilon} n)$ space.

Implementation of queries Let $b, e \in [0\,..\,n]$ be such that $b < e$. To compute $\arg\min_{i \in (b\,..\,e]} \mathrm{LCP}_T[i]$, given the indexes b and e, we use Lemma B.7 to compute $i_{\min} := \arg\min_{i \in (b\,..\,e]}(A[1] + \cdots + A[i])$ in

$$\mathcal{O}(h \cdot (1 + \log_w \ell)) = \mathcal{O}(h \cdot (1 + \tfrac{\log \ell}{\log w})) = \mathcal{O}(h) = \mathcal{O}(\tfrac{\log n}{\log \log n})$$

time. We then return $i_{\min}$ as the answer. The correctness follows by observing that, for every $i \in [1\,..\,n]$, $\sum_{j=1}^{i} A[j] = \mathrm{LCP}_T[1] + (\mathrm{LCP}_T[2] - \mathrm{LCP}_T[1]) + \cdots + (\mathrm{LCP}_T[i] - \mathrm{LCP}_T[i-1]) = \mathrm{LCP}_T[i]$. Thus, $i_{\min} = \arg\min_{i \in (b\,..\,e]} \mathrm{LCP}_T[i]$. □

References

[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.

[BCG+21] Djamal Belazzougui, Manuel Cáceres, Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Gonzalo Navarro, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Block trees. *Journal of Computer and System Sciences*, 117:1–22, 2021. doi:10.1016/j.jcss.2020.11.002.

[BCPT15] Djamal Belazzougui, Patrick Hagge Cording, Simon J. Puglisi, and Yasuo Tabei. Access, rank, and select in grammar-compressed strings. In Nikhil Bansal and Irene Finocchi, editors, *23rd Annual European Symposium on Algorithms, ESA 2025*, volume 9294 of *LNCS*, pages 142–154. Springer, 2015. doi:10.1007/978-3-662-48350-3_13.

[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.

[BLR+15] Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. *SIAM Journal on Computing*, 44(3):513–539, 2015. doi:10.1137/130936889.

[BW94] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994. URL: https://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf.

[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.

[CLL+05] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116.

[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.

[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.

[FW93] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993. doi:10.1016/0022-0000(93)90040-4.

[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.

[GJL21] Moses Ganardi, Artur Jeż, and Markus Lohrey. Balancing straight-line programs. *Journal of the ACM*, 68(4):27:1–27:40, 2021. doi:10.1145/3457389.

[GNP18] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. On the approximation ratio of Lempel-Ziv parsing. In Michael A. Bender, Martin Farach-Colton, and Miguel A. Mosteiro, editors, *13th Latin American Symposium on Theoretical Informatics, LATIN 2018*, volume 10807 of *LNCS*, pages 490–503. Springer, 2018. doi:10.1007/978-3-319-77404-6_36.

[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.

[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.

[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.

[I17] Tomohiro I. Longest common extensions with recompression. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, volume 78 of *LIPIcs*, pages 18:1–18:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.18.

[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.

[JL11] Allan Grønlund Jørgensen and Kasper Green Larsen. Range selection and median: Tight cell probe lower bounds and adaptive data structures. In Dana Randall, editor, *32nd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011*, pages 805–813. SIAM, 2011. doi:10.1137/1.9781611973082.63.

[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.

[KK20] Dominik Kempa and Tomasz Kociumaka. Resolution of the Burrows-Wheeler Transform conjecture. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 1002–1013. IEEE Computer Society, 2020. doi:10.1109/FOCS46700.2020.00097.

[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, Santa Cruz, CA, USA, November 6-9, 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 *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. Explaining the inherent tradeoffs for suffix array functionality: Equivalences between string problems and prefix range queries. In *Proceedings of the 2026 ACM-SIAM Symposium on Discrete Algorithms, SODA 2026*, 2026. To appear.

[KN10] Sebastian Kreft and Gonzalo Navarro. LZ77-like compression with fast random access. In *2010 Data Compression Conference*, pages 239–248. IEEE Computer Society, 2010. doi:10.1109/DCC.2010.29.

[KNP23] Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Towards a definitive compressibility measure for repetitive sequences. *IEEE Transactions on Information Theory*, 69(4):2074–2092, 2023. doi:10.1109/TIT.2022.3224382.

[KNW24] Tomasz Kociumaka, Jakob Nogler, and Philip Wellnitz. On the communication complexity of approximate pattern matching. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *56th Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 1758–1768. ACM, 2024. doi:10.1145/3618260.3649604.

[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.

[KP18] Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: String attractors. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 827–840. ACM, 2018. doi:10.1145/3188745.3188814.

[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.

[KRRW24] Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. *SIAM Journal on Computing*, 53(5):1524–1577, 2024. doi:10.1137/23M1567618.

[KS22] Dominik Kempa and Barna Saha. An upper bound and linear-space queries on the LZ-end parsing. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *33rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 2847–2866. SIAM, 2022. doi:10.1137/1.9781611977073.111.

[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.

[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.

[LZ76] Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976. doi:10.1109/TIT.1976.1055501.

[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.

[Nat22] National Human Genome Research Institute (NIH). Genomic data science, 2022. Accessed March 30, 2023. URL: https://www.genome.gov/about-genomics/fact-sheets/Genomic-Data-Science.

[Nav21a] Gonzalo Navarro. Indexing highly repetitive string collections, part I: Repetitiveness measures. *ACM Computing Surveys*, 54(2):29:1–29:31, 2021. doi:10.1145/3434399.

[Nav21b] Gonzalo Navarro. Indexing highly repetitive string collections, part II: Compressed indexes. *ACM Computing Surveys*, 54(2):26:1–26:32, 2021. doi:10.1145/3432999.

[NT21] Takaaki Nishimoto and Yasuo Tabei. Optimal-time queries on BWT-runs compressed indexes. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 101:1–101:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.101.

[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.

[Păt07] Mihai Pătraşcu. Lower bounds for 2-dimensional range counting. In David S. Johnson and Uriel Feige, editors, *39th Annual ACM Symposium on Theory of Computing, STOC 2007*, pages 40–46. ACM, 2007. doi:10.1145/1250790.1250797.

[PHR00] Molly Przeworski, Richard R. Hudson, and Anna Di Rienzo. Adjusting the focus on human variation. *Trends in Genetics*, 16(7):296–302, 2000. doi:10.1016/S0168-9525(00)02030-8.

[PP18] Alberto Policriti and Nicola Prezza. LZ77 computation based on the run-length encoded BWT. *Algorithmica*, 80(7):1986–2011, 2018. doi:10.1007/S00453-017-0327-Z.

[PT06] Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In Jon M. Kleinberg, editor, *38th Annual ACM Symposium on Theory of Computing, STOC 2006*, pages 232–240. ACM, 2006. doi:10.1145/1132516.1132551.

[Ryt03] Wojciech Rytter. Application of Lempel–Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302(1–3):211–222, 2003. doi:10.1016/S0304-3975(02)00777-6.

[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.

[SLF+15] Zachary D Stephens, Skylar Y Lee, Faraz Faghri, Roy H Campbell, Chengxiang Zhai, Miles J Efron, Ravishankar Iyer, Michael C Schatz, Saurabh Sinha, and Gene E Robinson. Big data: astronomical or genomical? *PLoS biology*, 13(7):e1002195, 2015. doi:10.1371/journal.pbio.1002195.

[VY13] Elad Verbin and Wei Yu. Data structure lower bounds on random access to grammar-compressed strings. In Johannes Fischer and Peter Sanders, editors, *24th Annual Symposium on Combinatorial Pattern Matching, CPM 2013*, volume 7922 of *LNCS*, pages 247–258. Springer, 2013. doi:10.1007/978-3-642-38905-4_24.

[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.

[Wil83] Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Information Processing Letters*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.

[ZL77] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.