An Upper Bound and Linear-Space Queries on the LZ-End Parsing

Dominik Kempa* Barna Saha†

Abstract

Lempel–Ziv (LZ77) compression is the most commonly used lossless compression algorithm. The basic idea is to greedily break the input string into blocks (called "phrases"), every time forming as a phrase the longest prefix of the unprocessed part that has an earlier occurrence. In 2010, Kreft and Navarro introduced a variant of LZ77 called LZ-End, that additionally requires the previous occurrence of each phrase to end at the boundary of an already existing phrase. Due to its excellent practical performance as a compression algorithm and a compressed index, they conjectured that it achieves a compression that can be provably upper-bounded in terms of the LZ77 size. Despite the recent progress in understanding such relation for other compression algorithms (e.g., the run-length encoded Burrows–Wheeler transform), no such result is known for LZ-End.

We prove that for any string of length n, the number z_e of phrases in the LZ-End parsing satisfies $z_e = \mathcal{O}(z \log^2 n)$, where z is the number of phrases in the LZ77 parsing. This puts LZ-End among the strongest dictionary compressors and solves a decade-old open problem of Kreft and Navarro. Using our techniques we also derive bounds for other variants of LZ-End and with respect to other compression measures. Our second contribution is a data structure that implements random access queries to the text in $\mathcal{O}(z_e)$ space and $\mathcal{O}(\text{polylog}\, n)$ time. This is the first linear-size structure on LZ-End that efficiently implements such queries. All previous data structures either incur a logarithmic penalty in the space or have slow queries. We also show how to extend these techniques to support longest-common-extension (LCE) queries.

1 Introduction

Lempel–Ziv (LZ77) [70] is the most commonly used lossless compression algorithm: In the Large Text Compression Benchmark [51], out of the 205 tested compressors, most (27.8%) are based on LZ77. It underlies the popular `gzip`, `7-zip`, and `png` formats, as well as modern compression formats `Brotli` [1], `LZ4` [18], and `zstd` [19], adopted by nearly all modern web browsers. Its impact was recognized with the Medal of Honor (the highest IEEE award) [61].

The basic idea of LZ77 is to greedily (left-to-right) break the input sequence $T[1\,..\,n]$ into blocks (called "phrases"), every time forming as a phrase the longest prefix of the unprocessed part $T[i\,..\,n]$ of the sequence that has an earlier occurrence. Each such phrase is encoded as a pair of integers (i', ℓ) consisting of a starting position $i' < i$ of some earlier occurrence of $T[i\,..\,i+\ell)$, and the phrase length $\ell > 0$. If there is no prefix of $T[i\,..\,n]$ occurring earlier in T, the next phrase is $T[i]$, and we encode it as a pair $(T[i], 0)$. This simple encoding is able to remove the redundancy caused not only by skewed symbol frequencies (its size, denoted by z, was shown to approach the kth order empirical entropy of T [44]), but also caused by repeated substrings: the greedy approach of LZ77 was proved to produce the *optimal* (i.e., of the smallest possible size) partition of the input sequence into phrases each having an earlier occurrence [49, Theorem 1]. The latter property is useful, as this type of redundancy is very common in modern applications. Examples of highly repetitive datasets include databases of same-species genomes [47], source code repositories (e.g., Github) [55], web crawls [22], and versioned documents (such as Wikipedia) [66]. The sizes of these datasets pose a computational challenge, e.g., the recently (2018) finished 100000 Human Genome Project [31] produced DNA sequences that in raw form take about 75TB when assembled. These datasets are, however, highly compressible: Github averages 20 versions per project [55], and two human genomes are known to be about 99.9% the same [63], making LZ77 the perfect tool to reduce their size.

Storage alone of these massive datasets is not enough, however. Many applications require the ability to quickly access or search the underlying (uncompressed) sequence [47]. Unfortunately, all off-the-shelf compressors produce a non-searchable output. This need to process highly repetitive datasets has been the driving force behind the

*Johns Hopkins University. Email: kempa@cs.jhu.edu. Supported by NIH HG011392 and NSF DBI-2029552.

†University of California, Berkeley. Email: barnas@berkeley.edu. Partially supported by an NSF CAREER Award 1652303, NSF 1909046, NSF HDR TRIPODS Grant 1934846 and an Alfred P. Sloan Fellowship.

development of the so-called *compressed indexes*: data structures capable of supporting queries (such as random access or pattern matching) on the uncompressed data, but using only space proportional to the size of compressed input. Due to its ideal properties, indexes based on LZ77 [2, 5, 7, 14, 21, 25, 36, 46, 58] and the closely related context-free grammars [8, 15, 16, 26, 32, 52, 68, 69] were the first to be developed. We refer to the recent surveys of Navarro for a comprehensive overview of these indexes [55] and the associated repetitiveness measures [54].

An important modification to the basic LZ77 compression scheme that has enabled the creation of one of the first LZ-based indexes [45] was to choose as the next phrase the longest prefix of the unprocessed part $T[i\,..\,n]$ that has an earlier occurrence *ending at the end of the already existing phrase*. The resulting "right-aligned" version of LZ77, proposed by Kreft and Navarro [45], is called LZ-End. Despite the excellent compression ratio (in practice only about 25% larger than LZ77 [39, 46]), its ease of use, and efficient construction algorithms [39, 40], little is known about the theoretical limits on its size. In their original paper [46], Kreft and Navarro show that similarly to LZ77, LZ-End parsing approaches the kth order entropy of the text. This bound, however, does not prove that LZ-End is suitable to store highly repetitive sequences, since entropy measures are insensitive to repetition of large blocks of data [54]. In their paper, Kreft and Navarro also showed a family of strings over large alphabet for which the ratio z_e/z is arbitrarily close to 2 (recently, such construction was also given for a binary alphabet [33]), and conjectured that the size z_e of the LZ-End parsing always satisfies $z_e \leq 2z$. In more than 10 years [45] no progress has been made on this problem and no relation to z is known, which is in contrast to nearly all other known dictionary compression algorithms [13, 27, 38, 41, 43, 64], whose sizes have been shown to either compress close to LZ77 (up to polylog n factors), or polynomially (i.e., by a n^ϵ factor) worse in the worst case. We thus formulate our first question as follows.

PROBLEM 1.1. *Can the size of LZ-End parsing be bounded in terms of LZ77 parsing?*

In addition to considering the *size* of the LZ-End parsing, we also study its *functionality*. In [53, Page 25], Navarro poses the following question: "Other equally fascinating questions can be asked about accessing and indexing strings: *What is the smallest reachable measure under which we can access and/or index the strings efficiently?* Right now, $\mathcal{O}(g_{\mathrm{rl}})$ is the best known limit for efficient access; it is unknown if one can access $T[i]$ efficiently within $\mathcal{O}(z_{\mathrm{no}})$ or $\mathcal{O}(r)$ space." Here, $\mathcal{O}(g_{\mathrm{rl}})$ refers to grammar compression that additionally permits special run-length productions [57]. We pose our second problem as a response to Navarro's question as follows.

PROBLEM 1.2. *Can we implement efficient random-access to the text using $\mathcal{O}(z_e)$ space?*

Our Contribution We positively answer both questions asked in Problem 1.1 and Problem 1.2. More precisely:

1. We prove that for any string of length n, it holds $z_e = \mathcal{O}(z \log^2 n)$[1], providing the first upper bound on the size of LZ-End parsing in terms of LZ77 parsing, and consequently establishing that LZ-End is a strong dictionary compression algorithm able to achieve size comparable to grammar compression [13], run-length encoded BWT [28], macro schemes [67], and collage systems [42]. This answers Problem 1.1, and thus provides the first non-trivial answer to the decade-old open problem of Kreft and Navarro [39]. After presenting the bound in its simplest form, we refine it as follows.
   - We show that with more careful analysis, we can improve and generalize it to other repetitiveness measures. In particular, we show $z_e = \mathcal{O}(\delta \log^2 \frac{n}{\delta})$, where $\delta \leq z$ is the *substring complexity*, a repetitiveness measure recently introduced by Kociumaka et al. [43]. As a corollary of this bound, we obtain the first bound $\mathcal{O}(\log^2 \frac{n}{\delta})$ relating the size of the LZ-End parsing for the string and its reverse.
   - We then show how to generalize our upper bound to the case of LZ-End parsing with the restriction t on the maximal phrase length. A fast algorithm to compute this variant of the LZ-End parsing was given in [39]. Its working space is proportional to the size of the output parsing plus the threshold t. Thus, we aim to use t as small as possible. The experimental results of [39] demonstrated that already with small values of t, the size of the resulting parsing was very close to LZ77, but this fact was entirely empirical. We show the first theoretical justification of this result: Letting $z_e(t)$ denote the size of the LZ-End parsing with the threshold value t, we prove that for any $t > 0$, it holds $z_e(t) = \mathcal{O}(\delta \log^2 m + \frac{n}{t} \log t)$, where $m = \min\left(\frac{n}{\delta}, t\right)$. Note that this bound is nearly optimal, since any parsing with threshold t has

[1] A similar bound $r = \mathcal{O}(z \log^2 n)$ has recently been achieved for run-length encoded BWT [38]. These two results, however, are obtained using different techniques.

at least $\lceil \frac{n}{t} \rceil$ phrases. On the other hand, $z_e(t) = \Omega(z) = \Omega(\delta)$ holds by [49, Theorem 1] and [43]. By the above result, it suffices to use a threshold t as small as $t = \frac{n \log n}{z}$ in the algorithm from [39], to guarantee that the restriction on the phrase length does not asymptotically affect the number of phrases in the resulting parsing. Since the largest value of $\frac{n}{z}$ in the experiments in [39] was less than 5000 and the datasets satisfied $\log n \leq 37$, this shows that the claim "For experiments, we fixed $t = 8 \times 2^{20}$, as it is small enough to not affect the RAM usage significantly, and big enough to have essentially no effect on the parsing size." [39] is now justified theoretically.

2. Our second main contribution focuses on Problem 1.2. We show that there exists a data structure of size $\mathcal{O}(z_e)$ that supports random access queries to T in $\mathcal{O}(\text{polylog}\, n)$ worst-case time, providing a non-trivial answer to the question of Navarro [53, Page 25] about linear-space (i.e., without logarithmic factors) random access queries on LZ-type compression. Similar (linear-space) result is known for grammar compression [8, 29]. No linear-space structure is known for LZ77 compression. The currently best structures for LZ77 (some of which generalize to LZ-End) incur a logarithmic penalty in the space and work either directly on LZ77 parsing [4, 5, 41] or using the earlier result [8, 29] on grammars whose size can be bounded in terms of LZ77 parsing [13, 34, 35, 64]. After presenting the structure for random access, we show how to generalize it to support Longest Common Extension (LCE) queries in $\mathcal{O}(\text{polylog}\, n)$ time and $\mathcal{O}(z_e)$ space. The LCE query $\text{LCE}(i, j)$, given two positions in a text T, returns the length of the longest common prefix of the suffixes $T[i\,..\,n]$ and $T[j\,..\,n]$ starting at positions i and j. These queries were introduced by Landau and Vishkin [48] in the context of approximate pattern matching. Since then, they have become one of the most commonly used tools in text processing [50]. The compressed structures for LCE queries achieve similar trade-offs as for random-access (e.g., [6, 30, 32, 57]).

Technical Overview The main idea in the basic variant $z_e = \mathcal{O}(z \log^2 n)$ of our upper bound (Theorem 3.1) is as follows. For any $j \in [1\,..\,z_e]$, let e_j denote the last position of the jth phrase, and let $\ell_j = e_j - e_{j-1}$ be its length (we assume $e_0 = 0$). We first observe that it suffices to only count phrases $T(e_{j-1}\,..\,e_j]$ satisfying either $j = z_e$, or $j \in [2\,..\,z_e)$ and $\ell_j \geq \lceil \frac{\ell_{j-1}}{2} \rceil$, since any consecutive $\lfloor \log n \rfloor + 2$ phrases must contain at least one such phrase. With each such phrase, we associate $\ell = e_j - e_{j-1}$ substrings of length 2^k of T (for some $2^k = \Theta(\ell)$), centered in the interval $(e_{j-1}\,..\,e_j]$. The value k is selected so that those substrings contain the preceding phrase. With such an assignment: (1) for every phrase, all associated substrings are distinct, and (2) each of the substrings is never assigned to more than two phrases. The proof of the first claim follows from the fact that 2^k is large enough so that all associated substrings cover the preceding phrase. This implies that if two of them were equal, the $(j-1)$th and jth phrase would both be contained in a highly periodic substring, which in turn would yield a longer candidate for jth phrase (see Fig. 2). The intuition for second claim is that when a substring X is assigned for the first time to some phrase $T(e_{i-1}\,..\,e_i]$, with the exception of the next phrase $T(e_i\,..\,e_{i+1}]$, X cannot be assigned to any other phrase to the right. The reason for this is that X now has an occurrence in T containing a phrase boundary in its right half. If X were again assigned to $T(e_{j-1}\,..\,e_j]$ for some $j > i+1$, this would yield a longer candidate (equal to the prefix of the initially assigned occurrence of X) for the preceding phrase $T(e_{j-2}\,..\,e_{j-1}]$ since the newly assigned occurrence of X is centered in the interval $(e_{j-1}\,..\,e_j]$ and completely covers the phrase $T(e_{j-2}\,..\,e_{j-1}]$. The last step of the proof is to apply an upper bound $z2^k$ on the number of distinct substrings of length 2^k. Since by the above argument each phrase is assigned a fraction of $\Theta(1/z)$ of all substrings of some length, considering all $\log n$ possible lengths, there cannot be more than $\mathcal{O}(z \log n)$ phrases in the assignment. Combining with the initial sparsification, we obtain the claimed upper bound $z_e = \mathcal{O}(z \log^2 n)$.

Our second contribution, a data structure implementing random access to T in $\mathcal{O}(z_e)$ space and $\mathcal{O}(\text{polylog}\, n)$ time can be thought of as a sparse version of the so-called *concentric exponential parse* (CEP) data structure [41, Theorem 5.3]. No such sparsification techniques were known before. The basic idea of CEP is as follows. Consider any LZ-like parsing of T. First, we store the leftmost and the rightmost symbol of each phrase. We then partition the remaining symbols of each phrase into at most $2 \log n$ blocks of exponentially increasing length from each end (i.e., we have blocks of size 1 on each end of the phrase, followed (towards the middle) by blocks of size two, four, and so on). If for each such block we store the pointer to its leftmost occurrence in T, we can compute $T[i]$ for any $i \in [1\,..\,n]$ in $\mathcal{O}(\log n)$ time by always mapping i to the corresponding position in the leftmost occurrence of the block that contains it. Since the leftmost occurrence of a substring is guaranteed to overlap a phrase boundary, in every step we halve the distance to the nearest phrase boundary. Consequently, after $\mathcal{O}(\log n)$ steps, we reach a position at the beginning or at the end of a phrase. We modify this basic structure as follows:

1. First, rather than storing the leftmost occurrence of each block $T[i\,..\,j]$, we store the pointer to the occurrence of $T[i\,..\,j]$ in T computed by repeatedly mapping the substring according to sources of phrases in the LZ-End parsing, until we reach an occurrence overlapping a phrase boundary. The resulting mapping does not increase the "R-distance", i.e., the distance to the nearest phrase boundary on the right (Lemma 4.1).
2. Second, rather than storing $\Theta(\log \ell)$ pointers for each length-ℓ phrase, we sample one of the blocks in its partition, and only store the pointer to its earlier occurrence (as defined above) for that block.
3. Finally, we store the pointer to the block containing the leftmost $\lfloor \frac{2}{3}\ell \rfloor$ symbols of each phrase.

The mapping with the above structure always first tries to map according to the sampled block. If i is not in the sampled block, we try to use the pointer for the leftmost $\lfloor \frac{2}{3}\ell \rfloor$ positions of the phrase. If position i is not among the leftmost $\lfloor \frac{2}{3}\ell \rfloor$ positions, we map according to the phrase source. Our main result is that with such mapping (requiring only 3 pointers for each phrase), whenever the sequence of mapped position crosses at least $\lfloor \log n \rfloor + 2$ sampled blocks, we are guaranteed that the R-distance is reduced by at least a factor of two-thirds (Lemma 4.4). We finally show that if we sample the blocks uniformly at random, with high probability for every position we will encounter at least one sampled block every $\Theta(\log^2 n)$ steps (Proposition 4.1). This ensures that taking $\Theta(\log^3 n)$ steps reduces $R(i)$ by at least $\frac{2}{3}$. Therefore, after $\mathcal{O}(\log^4 n)$ steps (each taking $\mathcal{O}(\log \log n)$ time due to a predecessor query), we reach a phrase boundary. Thus, the final query time is $\mathcal{O}(\log^4 n \cdot \log \log n) = \mathcal{O}(\text{polylog}\, n)$ (Corollary 4.2). In particular, this establishes the existence of a $\mathcal{O}(z_e)$-space data structure with worst-case $\mathcal{O}(\text{polylog}\, n)$-time queries (Theorem 4.1).

We point out that all previous applications of CEP (such as [27, 28, 37, 41]) simply store all $\Omega(\log n)$ pointers[2] for each phrase, and are deterministic. Our structure is the first to introduce the randomization, and show that further sparsification, combined with one extra "unbalanced" pointer for each phrase still lets us bound the number of steps during the query.

Related Work After the initial development of compressed indexes based on LZ77/LZ-End and grammar compression, more compression methods were adapted to allow efficient operations directly on compressed data. One of the most notable examples is the adaptation of the run-length compressed Burrows–Wheeler transform (BWT) [12]. The r-index, proposed by Gagie et al. [28] supports powerful suffix array and suffix tree queries, and has recently been proven to only take $\mathcal{O}(z \,\text{polylog}\, n)$ space [38]. Many improvements and enhancements of the r-index followed [3, 9, 10, 11, 17, 24, 59, 60]. A slightly different approach to BWT-based indexing taken by Sinha and Weinstein [65] is based on locally-decodable Move-to-Front (MTF) and also results in efficient queries.

Another family of indexes related to LZ77 is based on string attractors [41], combinatorial objects that generalize and whose bounds are strongly related to nearly all known dictionary compressors (except, until now, LZ-End). Due to this connection, the resulting indexes are called "universal". In [41], it was shown how to support random access on attractors. This was recently generalized to pattern matching [14, 56] and other queries [62].

The most recent addition to the set of repetitiveness measures and compressed indexes, are indexes whose space can be bounded in terms of δ, the *substring complexity*, a new measure recently introduced by Kociumaka et al. [43], defined as $\delta := \max_{m=1}^{n} \frac{1}{m}|\mathcal{S}_m|$, where $\mathcal{S}_m$ denotes the set of length-m substrings of T. As shown in [43], the value of δ lower-bounds nearly all other repetitiveness measures, and is insensitive to reversing the string (since the cardinality of $\mathcal{S}_m$ does not change after reversing). This makes it a useful compressibility measure.

2 Preliminaries

Let Σ be a nonempty set called the *alphabet*. For any finite string $S \in \Sigma^*$, we denote its length as $|S|$. We write $S[i\,..\,j]$, where $1 \le i, j \le |S|$, to denote a substring of S. If $i > j$, we assume $S[i\,..\,j]$ to be the empty string ε. An integer $p \in [1\,..\,|S|]$ is called a *period* of string S if $S[i] = S[i+p]$ holds for every $i \in [1\,..\,|S| - p]$. We write $[i\,..\,j)$ and $(i\,..\,j]$ to denote $[i\,..\,j-1]$ and $[i+1\,..\,j]$.

Throughout the paper, we consider a string $T[1\,..\,n] \in \Sigma^+$ such that $T[n] = \$ \in \Sigma$, where $\$$ is a special sentinel symbol that occurs only once in T. By $\text{lcp}(S_1, S_2)$ we denote the length of the longest common prefix of strings S_1 and S_2. For $j_1, j_2 \in [1\,..\,n]$, we let $\text{LCE}(j_1, j_2) = \text{lcp}(T[j_1\,..\,n], T[j_2\,..\,n])$.

An *LZ77-like factorization* of T is a factorization $T = F_1 \cdots F_f$ into non-empty *phrases* such that every phrase F_j with $|F_j| > 1$ has an earlier occurrence in T, i.e., letting $i = 1 + |F_1 \cdots F_{j-1}|$ and $\ell = |F_j|$, there exists $p \in [1\,..\,i)$ satisfying $\text{LCE}(p, i) \ge \ell$. The phrase $F_j = T[i\,..\,i+\ell)$ is encoded as a pair (p, ℓ). If there are multiple choices for

[2] Storing $\omega(\log n)$ pointers per phrase allows obtaining time-space tradeoffs.

p, we choose one arbitrarily. The occurrence $T[p\,..\,p+\ell)$ is called the *source* of F_j. If $\ell = 1$, $F_j = T[i]$ is encoded as a pair $(T[i], 0)$.

The LZ77 factorization [70] (or LZ77 parsing) of a string T is an LZ77-like factorization constructed by greedily parsing T from left to right into the longest possible phrases. More precisely, the jth phrase F_j is the longest substring starting at position $i = 1 + |F_1 \cdots F_{j-1}|$ that has an earlier occurrence in T. If there is no such substring, then $F_j = T[i]$. We denote the number of phrases in the LZ77 parsing by z. For example, the text $T = \texttt{bbabaababababaababa\$}$ has LZ77 factorization $\texttt{b} \cdot \texttt{b} \cdot \texttt{a} \cdot \texttt{ba} \cdot \texttt{aba} \cdot \texttt{bababa} \cdot \texttt{ababa} \cdot \texttt{\$}$ with $z = 8$ phrases, and is encoded as a sequence of pairs $(\texttt{b}, 0), (1, 1), (\texttt{a}, 0), (2, 2), (3, 3), (7, 6), (10, 5), (\texttt{\$}, 0)$. LZ77 parsing is known to be the LZ77-like parsing with the smallest possible number of phrases [49, Theorem 1].

The LZ-End factorization [46] of a string T is an LZ77-like factorization constructed by greedily parsing T from left to right into previous phrases, where at each step we choose the longest previous factor that has an earlier occurrence ending at the end of some already existing phrase. Formally, to compute the jth phrase starting at position $i = 1 + |F_1 \cdots F_{j-1}|$, we determine the largest $\ell \in [0\,..\,n-i+1]$ such the substring $T[i\,..\,i+\ell)$ is a suffix of the concatenation $F_1 \cdots F_{j'}$ of the first j' phrases for some $j' < j$. If $\ell > 0$, then the next phrase is $F_j = T[i\,..\,i+\ell)$, and we denote $\mathrm{src}(j) = j'$ (if there is more than one candidate for j', we choose one arbitrarily). Otherwise (i.e., if $\ell = 0$), the next phrase is $F_j = T[i]$.[3] The number of phrases in the LZ-End parsing is denoted by z_e. For example, the text $T = \texttt{bbabaababababaabaaba\$}$ has LZ-End factorization $\texttt{b} \cdot \texttt{b} \cdot \texttt{a} \cdot \texttt{ba} \cdot \texttt{aba} \cdot \texttt{baba} \cdot \texttt{baaba} \cdot \texttt{ba} \cdot \texttt{\$}$ with $z_e = 9$ phrases. By [49, Theorem 1], it holds $z \le z_e$.

Let q be a prime number and let $r \in [1\,..\,q)$ be chosen uniformly at random. The *Karp–Rabin fingerprint* of a substring $T(i\,..\,i+\ell]$, where $i \in [0\,..\,n]$ and $\ell \in [0\,..\,n-i]$, is defined as

$$\Phi(i, i+\ell) := \sum_{k=1}^{\ell} T[i+k] \cdot r^{\ell-k} \bmod q.$$

If $T(i\,..\,i+\ell] = T(j\,..\,j+\ell]$, then clearly $\Phi(i, i+\ell) = \Phi(j, j+\ell)$. On the other hand, if $T(i\,..\,i+\ell] \neq T(j\,..\,j+\ell]$, then with probability at least $1 - \ell/q$ [20], it holds $\Phi(i, i+\ell) \neq \Phi(j, j+\ell)$. In our algorithms, we are comparing only substrings of T of equal length. The number of different comparisons is thus less than n^3. Consequently, if for any positive constant c, we set q to be a prime number larger than n^{c+4} (but small enough to still fit in $\mathcal{O}(1)$ words), then by the union bound, the fingerprint function has no false positives (i.e., for all $i, j \in [0\,..\,n]$ and $\ell \in [1\,..\,n - \max(i, j)]$, the equality $T(i\,..\,i+\ell] = T(j\,..\,j+\ell]$ holds if and only if $\Phi(i, i+\ell) = \Phi(j, j+\ell)$) with probability at least $1 - n^{-c}$.

In our algorithm, we will utilize that for any i, j, k such that $i \le j \le k$, if we are given any two of the three Karp–Rabin fingerprints $\Phi(i, j), \Phi(j, k), \Phi(i, k)$, then we can quickly compute the third using one of the formulas:

$$\Phi(i, k) = \Phi(i, j) \cdot r^{k-j} + \Phi(j, k) \bmod q, \tag{2.1}$$

$$\Phi(i, j) = (\Phi(i, k) - \Phi(j, k)) \cdot r^{-(k-j)} \bmod q, \tag{2.2}$$

$$\Phi(j, k) = \Phi(i, k) - \Phi(i, j) \cdot r^{k-j} \bmod q, \tag{2.3}$$

each of which can be evaluated in $\mathcal{O}(\log n)$ time using exponentiation by squaring. The value of $r^{-1} \bmod q$ is precomputed once at the beginning with the extended Euclidean algorithm.

3 Upper Bounds

In this section, we present our first main result, i.e., a bound on the size of LZ-End compression in terms of LZ77.

The section is organized as follows. We start by presenting our bound in the simplest form $z_e = \mathcal{O}(z \log^2 n)$ (Section 3.1). We then show how to slightly improve the logarithmic factors and to generalize the bound to other repetitiveness measures (Section 3.2) (this results is the first bound relating the size of the LZ-End parsing of the string and its reverse). Finally, we show how to generalize the bound to variants of LZ-End parsing that impose a threshold on the phrase length (Section 3.3).

[3] This definition differs slightly from the original definition of Kreft and Navarro [46], who did not distinguish between the case $\ell > 0$ and $\ell = 0$, and always defined the next phrase as $F_j = T[i\,..\,i+\ell]$ (note that $T[i+\ell]$ is always defined, since by $T[n] = \texttt{\$}$, we always have $\ell \in [0\,..\,n-i]$). We show that our upper bound holds for both variants.

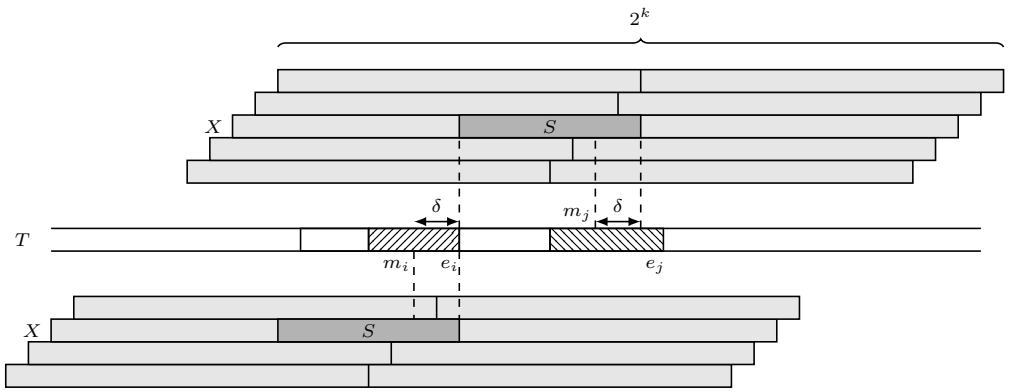

Figure 1: Illustration of the proof of Theorem 3.1: Every substring is associated with at most two phrases. If this is not true then, letting $X = T(m_i - 2^{k-1} \,..\, m_i + 2^{k-1}] = T(m_j - 2^{k-1} \,..\, m_j + 2^{k-1}]$ be the leftmost and the rightmost (respectively) occurrence of X that is associated with at least three phrases, we obtain that $i < j - 1$, and hence when the algorithm is adding the phrase $T(e_{j-2} \,..\, e_{j-1}]$ to the LZ-End parsing, the substring $S = T(e_{j-2} \,..\, m_j + \delta]$ has an earlier occurrence ending at the end of an already existing phrase $T(e_{i-1} \,..\, e_i]$. Since $m_j + \delta > e_{j-1}$, this contradicts $T(e_{j-2} \,..\, e_{j-1}]$ being in the parsing.

3.1 Basic Upper Bound

THEOREM 3.1. *Every string of length n satisfies $z_e = \mathcal{O}(z \log^2 n)$.*

Proof. Let T^∞ be an infinite string defined so that $T^\infty[i] = T[1 + (i-1) \bmod n]$ for $i \in \mathbb{Z}$; in particular, $T^\infty[1 \,..\, n] = T[1 \,..\, n]$. For any $m \geq 1$, let $\mathcal{S}_m = \{S \in \Sigma^m : S \text{ is a substring of } T^\infty\}$. Observe that it holds $|\mathcal{S}_m| \leq mz$, since every substring of length m of T^∞ has an occurrence overlapping or starting at some phrase boundary of the LZ77 parsing in T (this includes the substrings overlapping two copies of T, since $T[n] = \$$).

For any $j \in [1 \,..\, z_e]$, let e_j and ℓ_j denote (respectively) the last position and the length of the jth phrase in the LZ-End parsing of T. Letting $e_0 = 0$, we have $\ell_j = e_j - e_{j-1}$. We call the jth phrase $T(e_{j-1} \,..\, e_j]$ *special* if $j = z_e$, or $j \in [2 \,..\, z_e)$ and $\ell_j \geq \lceil \frac{\ell_{j-1}}{2} \rceil$. Let z'_e denote the number of special phrases. Observe that if $T(e_{j-1} \,..\, e_j]$ is not special for every $j \in [i \,..\, i + k]$, then $\ell_{i+k} \leq \frac{n}{2^k}$. Thus, any subsequence of $\lfloor \log n \rfloor + 2$ consecutive phrases contains a special phrase, and hence $z_e \leq z'_e \cdot (\lfloor \log n \rfloor + 2) = \mathcal{O}(z'_e \log n)$.

The basic idea of the proof is as follows. With each special phrase of length ℓ we associate ℓ distinct substrings of length 2^k, where $2^k < 12\ell$. We then show that every substring $X \in \mathcal{S}_{2^k}$, where $k \in [0 \,..\, \lceil \log n \rceil + 4]$, is associated with at most two phrases. Thus, by $|\mathcal{S}_{2^k}| \leq z2^k$, there are no more than $24z$ special phrases associated with strings in $\mathcal{S}_{2^k}$. Accounting all $k \in [0 \,..\, \lfloor \log n \rfloor + 4]$, this implies $z'_e = \mathcal{O}(z \log n)$, and hence $z_e = \mathcal{O}(z \log^2 n)$.

The assignment of substrings is done as follows. Let $j \in [1 \,..\, z_e]$ be such that $T(e_{j-1} \,..\, e_j]$ is a special phrase. Let $k \in [0 \,..\, \lfloor \log n \rfloor + 4]$ be the smallest integer such that $2^k \geq 6\ell$. With phrase $T(e_{j-1} \,..\, e_j]$ we associate substrings $X_i := T^\infty(i - 2^{k-1} \,..\, i + 2^{k-1}]$ where $i \in [e_{j-1} \,..\, e_j)$. We need to prove two things. First, that every substring is associated with at most two phrases, and second, that for every phrase, all associated ℓ substrings are distinct.

To show the first claim, suppose that for some $k \in [0 \,..\, \lfloor \log n \rfloor + 4]$, there exists $X \in \mathcal{S}_{2^k}$ associated with at least three special phrases. Let $T(e_{i-1} \,..\, e_i]$ be the leftmost, and $T(e_{j-1} \,..\, e_j]$ the rightmost such phrase in T. Note that since $T[n] = \$$ occurs in T only once, X cannot contain $\$$, and hence we have $j > 1$ and $\ell_j \geq \lceil \frac{\ell_{j-1}}{2} \rceil$. Let $m_i \in [e_{i-1} \,..\, e_i)$ and $m_j \in [e_{j-1} \,..\, e_j)$ be such that $X = T(m_i - 2^{k-1} \,..\, m_i + 2^{k-1}] = T(m_j - 2^{k-1} \,..\, m_j + 2^{k-1}]$ (note that since X does not contain $\$$, the two occurrences of X are inside T, and hence we do not need to write T^∞). Denote $\delta = e_i - m_i$. The situation is depicted in Fig. 1. Observe that $0 < \delta \leq 2^{k-1}$ (since $\delta \leq e_i - e_{i-1} \leq |X|/6$). Thus, $T(m_i - 2^{k-1} \,..\, m_i + \delta]$ is an occurrence of string $X[1 \,..\, 2^{k-1} + \delta]$ ending at the end of phrase $T(e_{i-1} \,..\, e_i]$. Since we assume at least three phrases are associated with X, we have $i < j - 1$, and hence the phrase $T(e_{j-2} \,..\, e_{j-1}]$ is to the right of $T(e_{i-1} \,..\, e_i]$. Recall, that we have $e_{j-1} - e_{j-2} \leq 2(e_j - e_{j-1})$. Thus, by $m_j - 2^{k-1} \leq m_j - 3(e_j - e_{j-1}) \leq e_j - 3(e_j - e_{j-1}) \leq e_{j-2}$ and $e_{j-1} \leq m_j$, the occurrence $T(m_j - 2^{k-1} \,..\, m_j + \delta] = X[1 \,..\, 2^{k-1} + \delta]$ contains the phrase $T(e_{j-2} \,..\, e_{j-1}]$. That, however, implies that when the algorithm is adding the phrase $T(e_{j-2} \,..\, e_{j-1}]$ to the LZ-End parsing, the substring $S = T(e_{j-2} \,..\, m_j + \delta]$ has an earlier occurrence (as a suffix of $X[1 \,..\, 2^{k-1} + \delta] = T(m_i - 2^{k-1} \,..\, m_i + \delta]$) ending at the end of an already

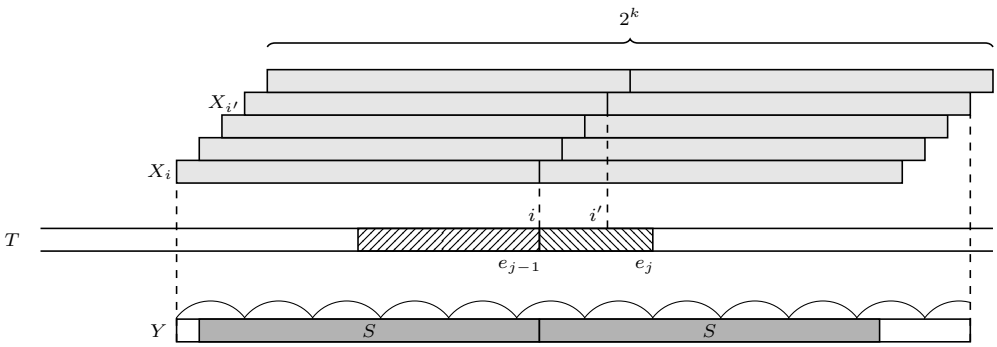

Figure 2: Illustration of the proof of Theorem 3.1: All $\ell = e_j - e_{j-1} = 5$ substrings associated with the phrase $T(e_{j-1}\,..\,e_j]$ are distinct, because if there exist $i, i' \in [e_{j-1}\,..\,e_j)$ such that $i < i'$ and $X_i = X_{i'}$, then $Y = T(i - 2^{k-1}\,..\,i' + 2^{k-1}]$ is periodic with period $p = i' - i < \ell$. This yields a substring S of length $|S| > \ell$ that starts at position $e_{j-1} + 1$ and has an earlier occurrence ending at a phrase boundary, contradicting $T(e_{j-1}\,..\,e_j]$ being in the parsing.

existing phrase $T(e_{i-1}\,..\,e_i]$. Since $m_j + \delta > e_{j-1}$, this substring is longer than $T(e_{j-2}\,..\,e_{j-1}]$, a contradiction.

To show the second claim, i.e., that for each special phrase $T(e_{j-1}\,..\,e_j]$, all associated ℓ strings are distinct, suppose that there exist $i, i' \in [e_{j-1}\,..\,e_j)$ such that $i < i'$ and $X_i = X_{i'}$; see Fig. 2. Note that by the uniqueness of $T[n] = \$$, we again have $j > 1$ and $\ell_j \geq \lceil \frac{\ell_{j-1}}{2} \rceil$. Denote $Y = T(i - 2^{k-1}\,..\,i' + 2^{k-1}]$ (since under the assumption that $X_i = X_{i'}$, the string Y does not contain $\$$, we again do not need to write T^∞). Since X_i is a prefix of Y, $X_{i'}$ is a suffix of Y, and it holds $X_i = X_{i'}$, it follows that Y has period p, where $p = i' - i < \ell$. By definition of a period, any substring of Y of length $2xp$, where $x \in \mathbb{Z}_{\geq 0}$, is a square (i.e., a string of the form SS, where $S \in \Sigma^*$). Consider thus the string $Y' := T(e_{j-1} - xp\,..\,e_{j-1} + xp]$, where $x = \lfloor (e_{j-1} - (i - 2^{k-1}))/p \rfloor$. Observe that:

- By definition of x, the position preceding Y' in T satisfies $e_{j-1} - xp \geq (i - 2^{k-1})$. On the other hand, $e_{j-1} + xp \leq e_{j-1} + 2^{k-1} - (i - e_{j-1}) \leq i' + 2^{k-1}$. Thus, Y' is a substring of Y.
- By $p < \ell$ and $2^{k-1} \geq 3\ell$, we have $xp > ((e_{j-1} - (i - 2^{k-1}))/p - 1)p = 2^{k-1} - p - (i - e_{j-1}) \geq \ell$.

By the above, the string Y' is a square, i.e., $Y' = SS$ and hence $T(e_{j-1} - xp\,..\,e_{j-1}] = T(e_{j-1}\,..\,e_{j-1} + xp]$. Moreover, it holds $xp > \ell$. This, however, contradicts the fact that $T(e_{j-1}\,..\,e_j]$ was selected as a phrase in the LZ-End parsing of T, since $T(e_{j-1}\,..\,e_{j-1} + xp]$ is a longer substring with a previous occurrence $T(e_{j-1} - xp\,..\,e_{j-1}]$ ending at a phrase boundary. Thus, we have shown that all ℓ strings associated with $T(e_{j-1}\,..\,e_j]$ are distinct. □

The above bound holds also for the original definition of the LZ-End parsing [46], and we prove this in the Appendix (Theorem A.1). In the rest of the paper, we adopt the above definition.

3.2 Tighter Upper Bound In this section, we show that with a more careful analysis, we can slightly improve the logarithmic factors in our upper bound. We also show how to generalize our bound to the measure δ [43], which lets us derive the first bound relating the size of the LZ-End parsing for the string T and its reverse.

LEMMA 3.1. *For any string of length n, it holds $z_e = \mathcal{O}(z'_e \log \frac{n}{z'_e})$, where z'_e is the number of special phrases.*

Proof. Let $(s_i)_{i \in [1\,..\,z'_e]}$ be an increasing sequence containing all $j \in [1\,..\,z_e]$ such that $T(e_{j-1}\,..\,e_j]$ is a special phrase (where again e_i is the last position of the ith phrase). Let $i \in [1\,..\,z'_e]$, and denote $x = s_i$ and $x' = s_{i-1}$. By definition, all but the last phrase overlapping $T(e_{x'}\,..\,e_x]$ are not special. Thus, looking at the lengths of those non-special phrases right-to-left, each at least doubles the length of the previous one, and hence $T(e_{x'}\,..\,e_x]$ overlaps at most $1 + \lceil \log(e_x - e_{x'}) \rceil$ phrases. Consequently, letting $s_0 = 0$ and $m_i = e_{s_i} - e_{s_{i-1}}$ for $i \in [1\,..\,z'_e]$, we obtain

$$z_e \leq z'_e + \sum_{i=1}^{z'_e} \lceil \log m_i \rceil \leq 2z'_e + \sum_{i=1}^{z'_e} \log m_i \leq 2z'_e + z'_e \log \tfrac{n}{z'_e},$$

where the last inequality follows from the log sum inequality[4] by observing that $\sum_{i=1}^{z'_e} m_i = n$. □

[4]For any $k \geq 1$ and any subset $\{a_1, \ldots, a_k, b_1, \ldots, b_k\} \subseteq \mathbb{R}_+$, it holds $\sum_{i=1}^k a_i \log \frac{b_i}{a_i} \leq a \log \frac{b}{a}$, where $a = \sum_{i=1}^k a_i$ and $b = \sum_{i=1}^k b_i$.

LEMMA 3.2. *For every string of length n, the number of special phrases satisfies $z'_e = \mathcal{O}(z \log \frac{n}{z})$.*

Proof. We perform the assignment of substrings to phrases as in the proof of Theorem 3.1, but only for special phrases of length $\ell \leq \frac{n}{z}$. The number of other special phrases is clearly bounded by z. Observe that for every special phrase of length $\ell \leq \frac{n}{z}$, the length of the associated substring in the proof of Theorem 3.1, satisfies $2^k \leq 12\ell \leq 12\frac{n}{z}$. Thus, to count the number of such phrases, rather than accounting all $k \in [0\,..\,\lfloor \log n \rfloor + 4]$, it suffices to account only $k \in [0\,..\,\lfloor \log \frac{n}{z} \rfloor + 4]$. Consequently, we obtain the upper bound $\mathcal{O}(z \log \frac{n}{z})$ on the number of special phrases of length $\ell \leq \frac{n}{z}$. Adding at most z special phrases of length $\ell > \frac{n}{z}$ yields the claim. □

COROLLARY 3.1. *Every string of length n satisfies $z_e = \mathcal{O}(z \log^2 \frac{n}{z})$.*

Proof. By Lemma 3.2, we have $z'_e = \mathcal{O}(z \log \frac{n}{z})$. Combining this with the upper bound $z_e = \mathcal{O}(z'_e \log \frac{n}{z'_e})$ shown in Lemma 3.1, and noting that as a function of z'_e, the expression $z'_e \log \frac{n}{z'_e}$ is increasing as long as $z'_e \leq \frac{n}{e}$, we obtain $z_e = \mathcal{O}(z \log \frac{n}{z} \log \frac{n}{z \log(n/z)}) = \mathcal{O}(z \log^2 \frac{n}{z})$. If $z'_e > \frac{n}{e}$, then $z_e = \Theta(z'_e) = \mathcal{O}(z \log \frac{n}{z})$. □

The above technique implies an upper bound on z_e in terms of another measure of repetitiveness: the *substring complexity* [43]. It is defined as $\delta := \max_{m=1}^{n} \frac{1}{m}|\mathcal{S}_m|$. The measure δ satisfies $\delta \leq z$ [43] and hence the following bound is always at least as good as $z_e = \mathcal{O}(z \log^2 \frac{n}{z})$.

THEOREM 3.2. *Every string of length n satisfies $z_e = \mathcal{O}(\delta \log^2 \frac{n}{\delta})$.*

Proof. We start by observing that δ can be equivalently defined as $\delta = \sup_{m=1}^{\infty} \frac{1}{m}|\mathcal{S}_m|$. This is because for any $m \geq 1$ we always have $|\mathcal{S}_m| \leq n$. Thus, for $m \geq n$, it holds $\frac{1}{m}|\mathcal{S}_m| \leq 1 \leq |\mathcal{S}_1|$. By this equivalent definition, for any $m \geq 1$, we have $\frac{1}{m}|\mathcal{S}_m| \leq \delta$, or equivalently, $|\mathcal{S}_m| \leq \delta m$. Plugging this bound on $|\mathcal{S}_m|$ into the proof of Lemma 3.2, we obtain that since every substring $X \in \mathcal{S}_{2^k}$, where $k \in [0\,..\,\lceil \log \frac{n}{\delta} \rceil + 4]$, is associated with at most two phrases, by $|\mathcal{S}_{2^k}| \leq \delta 2^k$, there is no more than 24δ special phrases associated with strings in $\mathcal{S}_{2^k}$. Accounting all $k \in [0\,..\,\lfloor \log \frac{n}{\delta} \rfloor + 4]$, this implies $z'_e = \mathcal{O}(\delta \log \frac{n}{\delta})$, and hence (via the same argument as in the proof of Corollary 3.1) $z_e = \mathcal{O}(\delta \log^2 \frac{n}{\delta})$. □

COROLLARY 3.2. *If z_e and $\overline{z_e}$ denote the size of the LZ-End parsing of a length-n text and its reverse, respectively, then $\overline{z_e}/z_e = \mathcal{O}(\log^2 \frac{n}{\delta})$.*

Proof. Note that the value of δ is the same for the string and its reverse. Thus, by Theorem 3.2, we obtain $\overline{z_e} = \mathcal{O}(\delta \log^2 \frac{n}{\delta})$. Combining this with the inequalities $\delta \leq z$ [43] and $z \leq z_e$ [49, Theorem 1], we obtain $\overline{z_e} = \mathcal{O}(z_e \log^2 \frac{n}{\delta})$. □

3.3 Upper Bound with Limit on Phrase Length In this section, we show how to generalize the upper bound of Theorem 3.2 to cover the case of LZ-End parsing with the threshold on the maximal phrase length.[5]

Let $z(t)$ and $z_e(t)$ denote the size of the LZ77 and the LZ-End variants with threshold t, respectively. The only modification in the definition of this variant is that the length of each phrase cannot exceed some predefined threshold t. Otherwise, the parsing is computed in the same way (i.e., greedily left to right). As a warmup, let us see how introducing the threshold affects the size of LZ77 parsing. The proof of the following fact is provided in the appendix.

FACT 3.1. *For any text of length n and any threshold $t \geq 1$, it holds $z(t) \leq z + \frac{n}{t}$.*

By the above, to achieve an increase in the space by no more than a constant factor, it suffices to impose the threshold t equal to the average phrase length. This bound is asymptotically tight, since any parsing with threshold t has at least $\lceil \frac{n}{t} \rceil$ phrases. On the other hand, by [49, Theorem 1], it cannot have less than z phrases.

Next, we show that a similar result holds for the LZ-End parsing. The strategy of the proof for this result is different than for the LZ77 parsing since, unlike LZ77, LZ-End is not the optimal parsing among the parsings satisfying the LZ-End property.

[5] Note that this modification is not the same as the one imposed, e.g., in `gzip`, which restricts the starting position i' of the earlier occurrence of each factor $T[i\,..\,i+\ell)$ in the parsing to start no further than w (the window size) positions earlier, i.e., that $i - i' \leq w$. Clearly, assuming the non-self-referential variant of LZ77, this also limits the lengths of all phrases by w, but is a much stronger assumption.

THEOREM 3.3. *For any text of length n and any $t \geq 1$, it holds $z_e(t) = \mathcal{O}(\delta \log^2 m + \frac{n}{t} \log t)$, where $m = \min\left(\frac{n}{\delta}, t\right)$.*

Proof. Let T^∞, $(e_i)_{i=0}^{z_e(t)}$, and $(\ell_i)_{i=1}^{z_e(t)}$ be defined analogously as in the proof of Theorem 3.1. We call a phrase $T(e_{j-1}\,..\,e_j]$ *long* if $\ell_j > \frac{1}{12}t$. Otherwise, a phrase is called *short*. A short phrase $T(e_{j-1}\,..\,e_j]$ is called *special* if $j = z_e(t)$, or $j \in [2\,..\,z_e(t))$ and $\ell_j \geq \lceil \frac{\ell_{j-1}}{2} \rceil$. Otherwise, a short phrase is called *regular*. By $z'_e(t)$ and $z''_e(t)$ we denote the number of special and the number of long phrases, respectively. The number of long phrases clearly satisfies $z''_e(t) \leq \lceil \frac{12n}{t} \rceil = \mathcal{O}(\frac{n}{t})$.

We first assume $t \leq \frac{n}{\delta}$. Observe, that if $T(e_{j-1}\,..\,e_j]$ is a regular phrase for every $j \in [i\,..\,i+k]$, then $\ell_{i+k} \leq \frac{t}{2^k}$. Thus, any subsequence of $\lfloor \log t \rfloor + 2$ consecutive phrases contains either a special or a long phrase, and hence

$$z_e(t) \leq (z'_e(t) + z''_e(t)) \cdot (\lfloor \log t \rfloor + 2) = \mathcal{O}(z'_e(t) \log t + z''_e(t) \log t).$$

To bound $z'_e(t)$, consider the assignment of substrings to phrases as in the proof of Theorem 3.1. To show that this assignment works also for the parsing with the limit on the phrase length, we observe that:

- Every substring is associated with at most two phrases. Let k, X, i, j, m_i, m_j, and δ be as in the proof of this claim in Theorem 3.1. Assuming that the claim is not true, we observed that this implies that the substring $T(e_{j-2}\,..\,m_j + \delta]$ has an earlier occurrence ending at the end of the already existing phrase, which by $(m_j + \delta) - e_{j-2} > e_{j-1} - e_{j-2}$ contradicts the phrase $T(e_{j-2}\,..\,e_{j-1}]$ being in the LZ-End parsing. We now additionally observe that since $T(e_{j-2}\,..\,m_j + \delta]$ is a substring of X and $T(e_{j-1}\,..\,e_j]$ is a special phrase, we have $m_j + \delta - e_{j-2} \leq |X| \leq 12(e_j - e_{j-1}) \leq t$, and consequently, $T(e_{j-2}\,..\,m_j + \delta]$ is a valid candidate for the phrase, even assuming we have a threshold t of the phrase length.
- For every special phrase $T(e_{j-1}\,..\,e_j]$, all associated $\ell = e_j - e_{j-1}$ substrings are pairwise distinct. Let i, x, and p be defined as in the proof of this claim in Theorem 3.1. Assuming the claim is not true, we observed that then $T(e_{j-1}\,..\,e_{j-1} + xp]$ has an earlier occurrence ending and the end of an already existing phrase, and $xp > \ell$ holds, contradicting $T(e_{j-1}\,..\,e_j]$ being in the LZ-End parsing. We now additionally observe that since $xp \leq 2^{k-1} \leq |X_i| \leq 12(e_j - e_{j-1}) \leq t$, the substring $T(e_{j-1}\,..\,e_{j-1} + xp]$ is again a valid candidate for the phrase, even assuming we have a threshold t of the phrase length.

By these two facts, and $|\mathcal{S}_{2^k}| \leq \delta 2^k$, there are no more than 24δ special phrases associated with strings in $\mathcal{S}_{2^k}$. Accounting all $k \in [0\,..\,\lfloor \log t \rfloor + 4]$, this implies $z'_e(t) = \mathcal{O}(\delta \log t)$.

Plugging this and the bound $z''_e(t) = \mathcal{O}(\frac{n}{t})$ into the above upper bound on $z_e(t)$, we obtain the claim $z_e(t) = \mathcal{O}(\delta \log^2 t + \frac{n}{t} \log t)$.

Let us now assume $t > \frac{n}{\delta}$. Let $(s_i)_{i \in [0\,..\,z'_e(t) + z''_e(t)]}$ be an increasing sequence such that $s_0 = 0$ and the remaining elements of the sequence contain all $j \in [1\,..\,z_e(t)]$ such that $T(e_{j-1}\,..\,e_j]$ is either a special or a long phrase. Let $i \in [1\,..\,z'_e(t) + z''_e(t)]$, and denote $x = s_i$ and $x' = s_{i-1}$. By definition, all but the last phrase overlapping $T(e_{x'}\,..\,e_x]$ are regular. Thus, looking at the lengths of those regular phrases right-to-left, each at least doubles the length of the previous one, and hence $T(e_{x'}\,..\,e_x]$ overlaps at most $1 + \lceil \log(e_x - e_{x'}) \rceil$ phrases. Consequently, letting $m_i = e_{s_i} - e_{s_{i-1}}$ for $i \in [1\,..\,z'_e(t) + z''_e(t)]$, we obtain

$$\begin{aligned} z_e(t) &\leq z'_e(t) + z''_e(t) + \sum_{i=1}^{z'_e(t)+z''_e(t)} \lceil \log m_i \rceil \leq 2(z'_e(t) + z''_e(t)) + \sum_{i=1}^{z'_e(t)+z''_e(t)} \log m_i \\ &\leq 2(z'_e(t) + z''_e(t)) + (z'_e(t) + z''_e(t)) \log \tfrac{n}{z'_e(t)+z''_e(t)} = \mathcal{O}(z'_e(t) \log \tfrac{n}{z'_e(t)} + z''_e(t) \log \tfrac{n}{z''_e(t)}) \end{aligned}$$

using the log sum inequality (utilizing that $\sum_{i=1}^{z'_e(t)+z''_e(t)} m_i = n$).

To bound $z'_e(t)$, we consider the assignment of substrings to phrases as in the proof of Theorem 3.1, except we perform the assignment only for special phrases of length $\ell \leq \frac{n}{12\delta}$. The number of other special phrases is clearly bounded by 12δ. To show that this assignment works also for the parsing with the limit on the phrase length, we recall from the above analysis of the case $t \leq \frac{n}{\delta}$, that it suffices to show that the phrases for which we perform the assignment are of length $\ell \leq \frac{1}{12}t$. This follows from $\ell \leq \frac{n}{12\delta}$ and $\frac{n}{\delta} < t$. Thus, there are no more than 24δ special phrases associated with strings in $\mathcal{S}_{2^k}$. Accounting all $k \in [0\,..\,\lfloor \log \frac{n}{12\delta} \rfloor + 4]$, this implies $z'_e(t) = \mathcal{O}(\delta \log \frac{n}{\delta})$.

Plugging this and the bound $z''_e(t) = \mathcal{O}(\frac{n}{t})$ into the above upper bound on $z_e(t)$, we obtain the claim $z_e(t) = \mathcal{O}(\delta \log \frac{n}{\delta} \log \frac{n}{\delta \log(n/\delta)} + \frac{n}{t} \log t) = \mathcal{O}(\delta \log^2 \frac{n}{\delta} + \frac{n}{t} \log t)$. □

4 Random Access in Linear Space

In this section, we present our structure that implements random access in $\mathcal{O}(\text{polylog}\, n)$ time and $\mathcal{O}(z_e)$ space.

The section is organized as follows. First (Section 4.1) we present the basic definitions. Next (Section 4.2), we prove the main combinatorial results used in our structure. In Section 4.3 we present our data structure. Finally (Section 4.4), we describe the query algorithm, prove its correctness, and analyze the running time.

4.1 Preliminaries Let $x \in [1\,..\,n]$, and let $j \in [1\,..\,z_e]$ be such that $x \in (e_{j-1}\,..\,e_j]$. We define the *left phrase offset* of x as $L(x) = x - e_{j-1}$. We let $L(n+1) = 1$. Analogously, we define the *right phrase offset* of x as $R(x) = e_j - x$. We denote $D(x) = \min(L(x), R(x)+1)$.

Let $b, e \in [0\,..\,n]$ be such that $b < e$ and let $\ell = e - b$. If there exists $j \in [1\,..\,z_e]$ such that $e_j \in (b\,..\,e]$, then we define $\text{pri}(b,e) = b$ and $\text{depth}(b,e) = 0$. Otherwise, let $j \in [1\,..\,z_e]$ be such that $e_{j-1} \le b < e < e_j$ (such j exists by the definition of the parsing). Then, we inductively define $\text{pri}(b,e) = \text{pri}(b',e')$ and $\text{depth}(b,e) = 1 + \text{depth}(b',e')$, where $j' = \text{src}(j)$ (it is defined since $e_j - e_{j-1} \ge 2$), b' is such that $e_{j'} - b' = e_j - b$, and $e' = b' + \ell$. Note that $\text{pri}(b,e)$ and $\text{depth}(b,e)$ are well defined since $j' < j$. In both cases, the fragment $T(\text{pri}(b,e)\,..\,\text{pri}(b,e)+\ell]$ is called the *primary occurrence* of $T(b\,..\,e]$. Informally, the primary occurrence is the occurrence of $T(b\,..\,e]$ that ends at a phrase boundary or overlaps two phrases, found by repeatedly mapping the fragment $T(b\,..\,e]$ to its copy induced by the source of the LZ-End phrase, and the depth of $T(b\,..\,e]$ is the number of steps needed to reach the primary occurrence.

4.2 Combinatorial Results

LEMMA 4.1. *Let $x \in [1\,..\,n]$ be a position in T, and let $b, e \in [0\,..\,n]$ be any integers such that $b < x \le e$. Then, the position $x' = \text{pri}(b,e) + (x-b)$ satisfies $T[x] = T[x']$ and $R(x') \le R(x)$.*

Proof. Induction on $\text{depth}(b,e)$. If $\text{depth}(b,e) = 0$, then $x' = x$, and thus we have $T[x'] = T[x]$ and $R(x') = R(x)$.

Let us assume $\text{depth}(b,e) > 0$. This implies that there exists $j \in [1\,..\,z_e]$ such that $e_{j-1} \le b < e < e_j$. Let $j' = \text{src}(j)$ (it is defined since $e_j - e_{j-1} \ge 2$), b' be such that $e_{j'} - b' = e_j - b$, and $e' = b' + (e-b)$. Observe then the position $y = b' + (x-b)$ satisfies $e_{j'} - y = e_j - x = R(x)$. Thus, $R(y) \le R(x)$. Moreover, since $T(b\,..\,e] = T(b'\,..\,e']$ and $y - b' = x - b$, we have $T[y] = T[x]$. By definition of the primary occurrence we then have $\text{pri}(b',e') = \text{pri}(b,e)$ and hence, from the inductive assumption applied to y and b', e', we obtain that for $x' = \text{pri}(b,e) + (x-b) = \text{pri}(b',e') + (y-b')$ it holds $T[x'] = T[y]$ and $R(x') \le R(y)$. Combining this with $T[y] = T[x]$ and $R(y) \le R(x)$, yields the claim. □

Consider a phrase $T(e_{j-1}\,..\,e_j]$ (where $j \in [1\,..\,z_e]$) in the LZ-End parsing of T. Partition the interval $(e_{j-1}\,..\,e_j]$ into blocks such that each block is a maximal contiguous subsequence of positions x with the same value of $\lfloor \log D(x) \rfloor$, and there is a block boundary in the middle of the phrase (if $e_j - e_{j-1}$ is even) or following the middle symbol (if $e_j - e_{j-1}$ is odd). More formally, let $\mathcal{B}_j$ be a set of pairwise disjoint integer intervals $\{\mathcal{I}_1, \ldots, \mathcal{I}_k\}$ (i.e., each $\mathcal{I}_i$ is a set of the form $(b\,..\,e] = \{b+1, b+2, \ldots, e\}$ for some $b, e \in [0\,..\,n]$ satisfying $b < e$) such that $\bigcup_{i=1}^{k} \mathcal{I}_i = (e_{j-1}\,..\,e_j]$, and for every $i \in [1\,..\,k]$, denoting $\mathcal{I}_i = (b\,..\,e]$, the following conditions are satisfied (recall that $\ell_j = e_j - e_{j-1}$):

- For every $x \in (b\,..\,e)$, it holds $\lfloor \log D(x) \rfloor = \lfloor \log D(x+1) \rfloor$,
- It holds $e_{j-1} + \lceil \ell_j/2 \rceil \notin (b\,..\,e)$,
- If $b \notin \{e_{j-1} + \lceil \ell_j/2 \rceil, e_{j-1}\}$, then $\lfloor \log D(b) \rfloor \ne \lfloor \log D(b+1) \rfloor$,
- If $e \notin \{e_{j-1} + \lceil \ell_j/2 \rceil, e_j\}$, then $\lfloor \log D(e) \rfloor \ne \lfloor \log D(e+1) \rfloor$.

The elements of the set $\mathcal{B}_j$ are called *bookmarks* of phrase $T(e_{j-1}\,..\,e_j]$. The set of all bookmarks is denoted $\mathcal{B} = \bigcup_{j=1}^{z_e} \mathcal{B}_j$. For any $q \in [1\,..\,|\mathcal{B}_j|]$, by $\mathcal{I}_{j,q}$ we denote the element $\mathcal{I} \in \mathcal{B}_j$ with the qth smallest starting position (it is well defined since all sets in $\mathcal{B}_j$ are disjoint).

LEMMA 4.2. *Let $x \in [1\,..\,n]$ and let $b, e \in [0\,..\,n]$ be such that $x \in (b\,..\,e] \in \mathcal{B}$. Then, the position $x' = \text{pri}(b,e) + (x-b)$ satisfies $L(x') \le L(x)/2$ or $R(x') \le R(x)/2$.*

Proof. Let $j \in [1\,..\,z_e]$ be such that $(b\,..\,e] \in \mathcal{B}_j$. Let us also denote $s = \lfloor \log D(x) \rfloor$, $h = \lceil \ell_j/2 \rceil$, and $h' = \lfloor \ell_j/2 \rfloor$.

Assume $x > e_{j-1} + h$. For any such x, it holds $L(x) > R(x)$, and hence $D(x) = R(x) + 1$. By definition of $\mathcal{B}_j$, we then have $s = \lfloor \log D(x) \rfloor = \lfloor \log D(e) \rfloor = \log D(e) = \log(R(e)+1)$, and hence $R(e) = 2^s - 1$. Since

there is a bookmark ending at position $e_{j-1}+h$ and $x > e_{j-1}+h$, we have $b \geq e_{j-1}+h$. Therefore, for any positions $y, y' \in (b\,..\,e]$ such that $y \neq y'$, we have $D(y) = R(y)+1 = e_j - y + 1 \neq e_j - y' + 1 = R(y')+1 = D(y')$. Thus, $e - b \leq 2^s$, and consequently, $e - x \leq 2^s - 1$. We now observe that by definition of $\mathrm{pri}(b,e)$, there exists $j' \in [1\,..\,z_e]$ such that $e_{j'} \in (\mathrm{pri}(b,e)\,..\,\mathrm{pri}(b,e)+(e-b)]$. Let $j'' \in [1\,..\,z_e]$ be the largest such j'. If $e_{j''} \geq x'$, then $R(x') \leq e_{j''} - x' \leq \mathrm{pri}(b,e)+(e-b)-x' = e-x$, and thus $2R(x') \leq 2(e-x) \leq e-x+2^s-1 = e-x+R(e) = R(x)$, or equivalently, $R(x') \leq R(x)/2$. On the other hand, if $e_{j''} < x'$, then it holds $L(x') = x' - e_{j''} < x' - \mathrm{pri}(b,e) = x-b$. This can be bounded as $x - b \leq e - b \leq e_j - e_{j-1} - h = \lfloor \ell_j/2 \rfloor \leq h$. Thus, $2L(x') \leq 2(x-b) \leq (x-b)+h \leq (x-(e_{j-1}+h))+h = (x-(e_{j-1}+h))+L(e_{j-1}+h) = L(x)$, or equivalently, $L(x') \leq L(x)/2$.

Consider now the other case, i.e., $x \leq e_{j-1}+h$. Analogously as before, for any such x, it holds $L(x) \leq R(x)+1$, and hence $D(x) = L(x)$. We also have $s = \lfloor \log D(x) \rfloor = \lfloor \log D(b+1) \rfloor = \log D(b+1) = \log L(b+1)$, and hence $L(b+1) = 2^s$. Similarly, since there is a bookmark ending at position $e_{j-1}+h$, and $x \leq e_{j-1}+h$, we have $e \leq e_{j-1}+h$. Since the value of $L(y)$ is different for every $y \in (b\,..\,e]$, it holds $e - b \leq 2^s$, and consequently, $x - b \leq 2^s - 1$. By definition of $\mathrm{pri}(b,e)$, there exists $j' \in [1\,..\,z_e]$ such that $e_{j'} \in (\mathrm{pri}(b,e)\,..\,\mathrm{pri}(b,e)+(e-b)]$. Let $j'' \in [1\,..\,z_e]$ be the largest such j'. If $e_{j''} < x'$, then it holds $L(x') = x' - e_{j''} \leq x' - (\mathrm{pri}(b,e)+1) = x-(b+1)$, and thus $2L(x') \leq 2(x-(b+1)) \leq (x-(b+1))+2^s = (x-(b+1))+L(b+1) = L(x)$, or equivalently, $L(x') \leq L(x)/2$. On the other hand, if $e_{j''} \geq x'$, then it holds $R(x') \leq e_{j''} - x' \leq \mathrm{pri}(b,e)+(e-b)-x' = e-x$. This can be bounded as $e - x \leq e - b - 1 \leq h - 1 \leq h'$. Thus, $2R(x') \leq 2(e-x) \leq (e-x)+h' \leq ((e_{j-1}+h)-x)+h' = ((e_{j-1}+h)-x)+R(e_{j-1}+h) = R(x)$, or equivalently, $R(x') \leq R(x)/2$. □

Let $(p_j)_{j=1}^{z_e}$ be a sequence of integers such that for any $j \in [1\,..\,z_e]$, it holds $p_j \in [1\,..\,|\mathcal{B}_j|]$. Any such sequence is called a *selector sequence*. Given a selector sequence (p_j), we define the function map : $[1\,..\,n] \to [1\,..\,n]$ as follows. Let $x \in [1\,..\,n]$, and let $j \in [1\,..\,z_e]$ be such that $x \in (e_{j-1}\,..\,e_j]$. If $x = e_j$, then we define $\mathrm{map}(x) = x$. Otherwise, i.e., when $x \in (e_{j-1}\,..\,e_j)$, let $(b\,..\,e] \in \mathcal{B}$ be the bookmark selected for phrase $T(e_{j-1}\,..\,e_j]$, i.e., such that it holds $(b\,..\,e] = \mathcal{I}_{j,p_j}$. Let $j' = \mathrm{src}(j)$ and note that $e_j - e_{j-1} \geq 2$. We define

$$\mathrm{map}(x) = \begin{cases} \mathrm{pri}(b,e) + (x-b) & \text{if } x \in \mathcal{I}_{j,p_j}, \\ \mathrm{pri}(e_{j-1}, e_{j-1} + \lfloor 2\ell_j/3 \rfloor) + (x - e_{j-1}) & \text{if } x \notin \mathcal{I}_{j,p_j} \text{ and } x \leq e_{j-1} + \lfloor 2\ell_j/3 \rfloor, \\ e_{j'} - (e_j - x) & \text{if } x \notin \mathcal{I}_{j,p_j} \text{ and } x > e_{j-1} + \lfloor 2\ell_j/3 \rfloor. \end{cases}$$

LEMMA 4.3. *For any $x \in [1\,..\,n]$, it holds $T[\mathrm{map}(x)] = T[x]$ and $R(\mathrm{map}(x)) \leq R(x)$.*

Proof. Let $j \in [1\,..\,z_e]$ be such that $x \in (e_{j-1}\,..\,e_j]$. If $x = e_j$, then $\mathrm{map}(x) = x$, and hence the claim holds trivially. Otherwise, we consider three cases:

1. If $x \in \mathcal{I}_{j,p_j}$ then, letting $(b\,..\,e] = \mathcal{I}_{j,p_j}$, we have $\mathrm{map}(x) = \mathrm{pri}(b,e) + (x-b)$. Thus, $T[\mathrm{map}(x)] = T[x]$ and $R(\mathrm{map}(x)) \leq R(x)$ follow by Lemma 4.1.
2. If $x \notin \mathcal{I}_{j,p_j}$ and $x \leq e_{j-1} + \lfloor 2\ell_j/3 \rfloor$, then both claims analogously follow by Lemma 4.1.
3. Finally, if $x \notin \mathcal{I}_{j,p_j}$ and $x > e_{j-1} + \lfloor 2\ell_j/3 \rfloor$ then, letting $j' = \mathrm{src}(j)$, we observe that $\mathrm{map}(x)$ satisfies $e_{j'} - \mathrm{map}(x) = e_j - x$. Thus, by definition of the parsing, $T[\mathrm{map}(x)] = T[x]$. This also implies that $R(\mathrm{map}(x)) \leq e_j - x = R(x)$. □

4.3 Data Structure Let $(p_j)_{j=1}^{z_e}$ be a selector sequence. Based on (p_j), we define the data structure implementing random access to text T as follows. The structure consists of five components:

1. First, we store an array $E[0\,..\,z_e]$ defined by $E[i] = e_i$, i.e., $E[i]$ contains the last position of each phrase. We augment E with a predecessor data structure. We use a structure from the full version of [23, Proposition 7], and hence the augmented E needs $\mathcal{O}(z_e)$ space and answers queries in $\mathcal{O}(\log\log n)$ worst-case time.
2. Second, we store an array $S[1\,..\,z_e]$ defined by $S[i] = \mathrm{src}(i)$ (see Section 2).
3. Third, we store an array $P[1\,..\,z_e]$ defined by $P[i] = \mathrm{pri}(e_{i-1}, e_{i-1} + \lfloor 2\ell_i/3 \rfloor)$.
4. Fourth, we store an array $B[1\,..\,z_e]$ defined by $B[i] = (b, e, \mathrm{pri}(b,e))$, where $(b\,..\,e] = \mathcal{I}_{i,p_i}$.
5. Finally, we store an array $C[1\,..\,z_e]$ defined by $C[i] = T[e_i]$.

4.4 Query Algorithm Using the above data structure, given any $x \in [1\,..\,n]$, we compute $T[x]$ as follows. First, observe that given any position x, we can compute $\mathrm{map}(x)$ in $\mathcal{O}(\log\log n)$ time. For this, using the predecessor on E we first in $\mathcal{O}(\log\log n)$ time determine $j = \min\{i \in [1\,..\,z_e] : e_i \geq x\}$. Then, $x \in (e_{j-1}\,..\,e_j]$. Note that we can

obtain $e_{j-1} = E[j-1]$ and $e_j = E[j]$ in $\mathcal{O}(1)$ time. If $x = e_j$, we immediately obtain $\mathrm{map}(x) = x$. Otherwise, letting $B[j] = (b, e, p)$, we check if $x \in (b\,..\,e]$. If so, we have $x \in \mathcal{I}_{j,p_j}$, and hence $\mathrm{map}(x) = p + (x - b)$. If $x \notin (b\,..\,e]$, we check if $x \le e_{j-1} + \lfloor 2\ell_j/3 \rfloor$. If so, we have $\mathrm{map}(x) = p' + (x - e_{j-1})$, where $p' = P[j] = \mathrm{pri}(e_{j-1}, e_{j-1} + \lfloor 2\ell_j/3 \rfloor)$. Finally, if $x > e_{j-1} + \lfloor 2\ell_j/3 \rfloor$, we have $\mathrm{map}(x) = e_{j'} - (e_j - x)$, where $j' = \mathrm{src}(j)$. The value $e_{j'}$ is obtained in $\mathcal{O}(1)$ time as $E[S[j]]$. In total, we have thus spent $\mathcal{O}(\log \log n)$ time.

Given the input position $x \in [1\,..\,n]$, the query algorithm repeatedly maps x to $\mathrm{map}(x)$ as long as the phrase $T(e_{j-1}\,..\,e_j]$ containing the symbol $T[x]$ satisfies $x \neq e_j$ (or equivalently, as long as $R(x) > 0$). When the algorithm reaches the value x such that $e_j = x$, we return the symbol $C[j] = T[x]$ as the answer. The correctness of this procedure follows from Lemma 4.3. To see that the algorithm always terminates, note that $x \neq e_j$ implies $\mathrm{map}(x) \in [1\,..\,x)$ and for $x = 1$ we have $e_j = 1 = x$. Thus, the procedure terminates after a finite number of steps.

To estimate the number of steps, we introduce the following definitions. Let $\mathcal{M} = \bigcup_{j=1}^{z_e} \mathcal{I}_{j,p_j}$ denote the set of positions in T that are covered by bookmarks selected by the sequence (p_j). Define $\mathrm{map}^{(0)}(x) = x$, and for any $s > 0$, $\mathrm{map}^{(s)}(x) = \mathrm{map}(\mathrm{map}^{(s-1)}(x))$.

LEMMA 4.4. *Let $x \in \mathcal{M}$. If $k \ge 0$ is such that $|\{s \in [0\,..\,k) : \mathrm{map}^{(s)}(x) \in \mathcal{M}\}| \ge \lfloor \log n \rfloor + 2$, then the position $y = \mathrm{map}^{(k)}(x)$ satisfies $R(y) \le \frac{2}{3}R(x)$.*

Proof. Let $(b_i)_{i=1}^{k}$ be a sequence defined by $b_i = |\{s \in [0\,..\,i) : \mathrm{map}^{(s)}(x) \in \mathcal{M}\}|$. To prove the lemma, we will show a more general claim, namely, that for any $i \in [1\,..\,k]$, it holds either $R(\mathrm{map}^{(i)}(x)) \le \frac{2}{3}R(x)$ or $L(\mathrm{map}^{(i)}(x)) \le R(x)/2^{b_i - 1}$. Since for any $y \in [1\,..\,n]$, it holds $L(y) \ge 1$, and we have $b_k \ge \lfloor \log n \rfloor + 2$, this implies that it holds $R(y) = R(\mathrm{map}^{(k)}(x)) \le \frac{2}{3}R(x)$, since $L(\mathrm{map}^{(k)}(x)) \le R(x)/2^{b_k - 1} \le n/2^{\lfloor \log n \rfloor + 1} < 1$ is impossible. The main intuition is that if after the first application of map we have $R(\mathrm{map}(x)) > \frac{2}{3}R(x)$, then $\mathrm{map}(x)$ must be in the left two-thirds of a phrase. We then show that for as long as it holds $R(\mathrm{map}^{(s)}(x)) > \frac{2}{3}R(x)$, the position $\mathrm{map}^{(s)}(x)$ must be in the left two-thirds of the phrase and thus each step must necessarily reduce the value of L for the current position by a factor of two (if we use a bookmark), or at least not increase it (if the second case in the definition of map holds). Eventually, however, the case $R(\mathrm{map}^{(s)}(x)) \le \frac{2}{3}R(x)$ must occur since the value of L cannot drop below 1. By Lemma 4.3, the value of R then stays below $\frac{2}{3}R(x)$ until we reach $y = \mathrm{map}^{(k)}(x)$.

The proof is by induction on $i \in [1\,..\,k]$. To show the claim for $i = 1$, let $u = x$ and $u' = \mathrm{map}(u)$. We have $u \in \mathcal{M}$. Let thus $j \in [1\,..\,z_e]$ and $b, e \in [0\,..\,n]$ be such that $u \in (b\,..\,e] = \mathcal{I}_{j,p_j} \in \mathcal{B}_j$. We then have $\mathrm{map}(u) = \mathrm{pri}(b, e) + (u - b)$. Denote $s = \lfloor \log D(u) \rfloor$, $h = \lceil \ell_j/2 \rceil$, and $h' = \lfloor \ell_j/2 \rfloor$. Similarly as in the proof of Lemma 4.2, we consider two cases:

- Let us first assume $u > e_{j-1} + h$. We then have $R(e) = 2^s - 1$ and $e - b \le 2^s$. By definition of $\mathrm{pri}(b, e)$, there exists $j' \in [1\,..\,z_e]$ such that $e_{j'} \in (\mathrm{pri}(b, e)\,..\,\mathrm{pri}(b, e) + (e - b)]$. Let $j'' \in [1\,..\,z_e]$ be the largest such j'. As observed in the proof of Lemma 4.2, if $e_{j''} \ge u'$, then $R(u') \le \frac{1}{2}R(u) \le \frac{2}{3}R(u) = \frac{2}{3}R(x)$. On the other hand, if $e_{j''} < u'$, then $L(u') \le e - b - 1 \le 2^s - 1 = R(e) \le R(u) = R(x)$. We have thus shown that either $R(\mathrm{map}(x)) \le \frac{2}{3}R(x)$ or $L(\mathrm{map}(x)) \le R(x)$.
- Consider now the other case, i.e., $u \le e_{j-1} + h$. Then, by definition of $\mathrm{pri}(b, e)$, there exists $j' \in [1\,..\,z_e]$ such that $e_{j'} \in (\mathrm{pri}(b, e)\,..\,\mathrm{pri}(b, e) + (e - b)]$. Let $j'' \in [1\,..\,z_e]$ be the largest such j'. If $e_{j''} < u'$, then, as shown in the proof of Lemma 4.2, it holds $L(u') \le u - b - 1 \le e - b - 1 \le h - 1 \le h' = R(e_{j-1} + h) \le R(u) = R(x)$. On the other hand, if $e_{j''} \ge u'$, then $R(u') \le \frac{1}{2}R(u) \le \frac{2}{3}R(u) = \frac{2}{3}R(x)$. Thus, we again have either $R(\mathrm{map}(x)) \le \frac{2}{3}R(x)$ or $L(\mathrm{map}(x)) \le R(x)$.

To show the induction step, let $i \in (1\,..\,k]$, and assume that the claim holds for all smaller i. Denote $u = \mathrm{map}^{(i-1)}(x)$ and $u' = \mathrm{map}^{(i)}(x) = \mathrm{map}(u)$. Let us assume that it holds $R(u') > \frac{2}{3}R(x)$. We will show that this implies $L(u') \le R(x)/2^{b_i - 1}$, implying the claim. We first observe that since the value of R never increases when applying map (Lemma 4.3), we must also have $R(u) > \frac{2}{3}R(x)$, and consequently, by the inductive assumption, $L(u) = L(\mathrm{map}^{(i-1)}(x)) \le R(x)/2^{b_{i-1} - 1}$. Let $j \in [1\,..\,z_e]$ be such that $u \in (e_{j-1}\,..\,e_j]$. Consider two cases:

- Assume first that $u \in \mathcal{M}$. Let $b, e \in [0\,..\,n]$ be such that $u \in (b\,..\,e] = \mathcal{I}_{j,p_j} \in \mathcal{B}_j$. By Lemma 4.2, the position $u' = \mathrm{map}(u) = \mathrm{pri}(b, e) + (u - b)$ then satisfies either $R(u') \le R(u)/2$ or $L(u') \le L(u)/2$. In the first case, we have $R(u') \le \frac{1}{2}R(u) \le \frac{1}{2}R(x)$, contradicting $R(u') > \frac{2}{3}R(x)$. We must thus have $L(u') \le \frac{1}{2}L(u)$. Since by $u \in \mathcal{M}$ we have $b_i = b_{i-1} + 1$, we obtain $L(\mathrm{map}^{(i)}(x)) \le \frac{1}{2}L(\mathrm{map}^{(i-1)}(x)) \le \frac{1}{2}R(x)/2^{b_{i-1} - 1} = R(x)/2^{b_i - 1}$.
- Assume now that $u \notin \mathcal{M}$. First, we observe that it holds $u \le e_{j-1} + \lfloor 2\ell_j/3 \rfloor$. To show this, note that $\ell_j = L(u) + R(u)$, and hence this inequality is equivalent to $L(u) \le 2R(u)$. To see that this holds for u, recall that we by the inductive assumption, we have $L(u) \le R(x)$. Combined with $R(u) > \frac{2}{3}R(x)$, we obtain $L(u) \le$

$R(x) < \frac{4}{3}R(x) < 2R(u)$. We thus obtained that $u' = \mathrm{map}(u) = \mathrm{pri}(e_{j-1}, e_{j-1} + \lfloor 2\ell_j/3 \rfloor) + (u - e_{j-1})$. Note that by $1 \le L(u) \le 2R(u)$ we obtain $R(u) > 0$, and consequently, $\ell_j > 1$. Denote $p = \mathrm{pri}(e_{j-1}, e_{j-1} + \lfloor 2\ell_j/3 \rfloor)$. By definition of p, there exists $j' \in [1\,..\,z_e]$ such that $e_{j'} \in (p\,..\,p + \lfloor 2\ell_j/3 \rfloor]$. Let $j'' \in [1\,..\,z_e]$ be the largest such j'. Observe that $e_{j''} \ge u'$ is impossible because, denoting $\delta = \ell_j - \lfloor 2\ell_j/3 \rfloor \ge \frac{1}{3}\ell_j$, this implies $R(u') \le e_{j''} - u' \le p + \lfloor 2\ell_j/3 \rfloor - u' = e_{j-1} + \lfloor 2\ell_j/3 \rfloor - u = e_j - \delta - u = R(u) - \delta \le R(u) - \frac{1}{3}\ell_j \le R(u) - \frac{1}{3}R(u) = \frac{2}{3}R(u) \le \frac{2}{3}R(x)$, contradicting our assumption $R(u') > \frac{2}{3}R(x)$. Thus, it holds $e_{j''} < u'$. But then, $L(u') = u' - e_{j''} \le u' - p = u - e_{j-1} = L(u)$. It remains to observe that $u \notin \mathcal{M}$ implies $b_i = b_{i-1}$. Hence, by the inductive assumption, $L(u') \le L(u) \le R(x)/2^{b_{i-1}-1} = R(x)/2^{b_i - 1}$. ☐

COROLLARY 4.1. *Let $x \in [1\,..\,n]$. If $k \ge 0$ is such that $|\{s \in [0\,..\,k) : \mathrm{map}^{(s)}(x) \in \mathcal{M}\}| \ge (\log_{3/2} n + 2)^2$, then the position $y = \mathrm{map}^{(k)}(x)$ satisfies $R(y) = 0$.*

Proof. Let $t = \lfloor \log_{3/2} n \rfloor + 2$. By Lemma 4.4, every time we use $\lfloor \log n \rfloor + 2 \le t$ bookmarks, we are guaranteed to reduce the value of R by a factor of $\frac{2}{3}$. Consequently, after $\lfloor \log_{3/2} n \rfloor + 1 \le t$ such events the value of R must be smaller than one. Thus, if we pass at least $(\log_{3/2} n + 2)^2 \ge t^2$ bookmarks overall, we have $R(y) = 0$. ☐

PROPOSITION 4.1. *Let $(p_j)_{j=1}^{z_e}$ be a selector sequence, such that each $p_j \in [1\,..\,|\mathcal{B}_j|]$ is chosen independently and uniformly at random. Let $k = 4\lceil \log n \rceil^2$. Then, with probability at least $1 - \frac{1}{n}$, all positions $x \in [1\,..\,n]$ satisfy $\{\mathrm{map}^{(s)}(x) : s \in [0\,..\,k)\} \cap \mathcal{M} \neq \emptyset$.*

Proof. Let $x \in [1\,..\,n]$. If there exists $s \in [0\,..\,k)$ such that $L(\mathrm{map}^{(s)}(x)) + R(\mathrm{map}^{(s)}(x)) = 2$, then the phrase $T(e_{j-1}\,..\,e_j]$ containing the position $\mathrm{map}^{(s)}(x)$ decomposes into a single bookmark, i.e., $|\mathcal{B}_j| = 1$, and hence we must have $\mathrm{map}^{(s)}(x) \in \mathcal{M}$.

Let us thus assume that for every $s \in [0\,..\,k)$, we have $L(\mathrm{map}^{(s)}(x)) + R(\mathrm{map}^{(s)}(x)) > 2$. It is easy to see that whenever $j \in [1\,..\,z_e]$ satisfies $e_j - e_{j-1} > 1$, we have $|\mathcal{B}_j| \le 2\lceil \log(e_j - e_{j-1}) \rceil \le 2\lceil \log n \rceil$. Since the bookmark for each phrase is selected uniformly at random, letting $T(e_{j-1}\,..\,e_j]$ be the phrase containing position $\mathrm{map}^{(s)}(x)$, we have $\mathbb{P}(\mathrm{map}^{(s)}(x) \in \mathcal{M}) = \frac{1}{|\mathcal{B}_j|} \ge \frac{1}{2\lceil \log n \rceil}$. Therefore, since phrases containing positions $\mathrm{map}^{(s)}(x)$ for $s \in [0\,..\,k)$ are all different,

$$\mathbb{P}\left(\{\mathrm{map}^{(s)}(x) : s \in [0\,..\,k)\} \cap \mathcal{M} = \emptyset\right) \le \left(1 - \tfrac{1}{2\lceil \log n \rceil}\right)^k \le \exp\left(-\tfrac{k}{2\lceil \log n \rceil}\right)$$
$$\le \exp\left(-2 \log n\right) = n^{-2/\ln(2)} \le 1/n^2.$$

By the union bound, the probability of the above event occurring for *some* $x \in [1\,..\,n]$ is at most $1/n$. Thus, with probability at least $1 - \frac{1}{n}$, all positions $x \in [1\,..\,n]$ satisfy $\{\mathrm{map}^{(s)}(x) : s \in [0\,..\,k)\} \cap \mathcal{M} \neq \emptyset$. ☐

COROLLARY 4.2. *Let $(p_j)_{j=1}^{z_e}$ be a selector sequence, such that each $p_j \in [1\,..\,|\mathcal{B}_j|]$ is chosen independently and uniformly at random. Then, with probability at least $1 - \frac{1}{n}$, the query algorithm returns the answer in $\mathcal{O}(\log^4 n \cdot \log\log n)$ time for all $x \in [1\,..\,n]$.*

Proof. Let $k' = 4\lceil \log n \rceil^2$, $k'' = \lceil (\log_{3/2} n + 2)^2 \rceil$, and $k = k'k''$. By Proposition 4.1, with probability at least $1 - \frac{1}{n}$, for all positions $x \in [1\,..\,n]$, it holds $|\{s \in [0\,..\,k) : \mathrm{map}^{(s)}(x) \in \mathcal{M}\}| \ge k'' \ge (\log_{3/2} n + 2)^2$. Thus, by Corollary 4.1, the position $y = \mathrm{map}^{(k)}(x)$ satisfies $R(y) = 0$. In other words, with probability at least $1 - \frac{1}{n}$, performing $k = \mathcal{O}(\log^4 n)$ applications of map suffices to reach the sampled text symbol. Each step takes $\mathcal{O}(\log\log n)$ time, which yields the claimed complexity. ☐

The above holds with nonzero probability. Thus, by the probabilistic method we obtain the following result.

THEOREM 4.1. *For any text T of length n, there exists a data structure of size $\mathcal{O}(z_e)$ that supports random access queries to T in $\mathcal{O}(\log^4 n \cdot \log\log n)$ worst-case time.*

Note that our result is existential and does not address the efficient construction of the structure. It is straightforward, however, to construct our structure with worst-case queries in expected $\mathcal{O}(\mathrm{poly}(n))$ time. Alternatively, one can easily achieve worst-case $\mathcal{O}(\mathrm{poly}(n))$ construction and expected-time queries. We leave improving the query time as an interesting future work, noting the inefficiency in the above construction (for the existential proof, it suffices to obtain a constant non-zero probability).

5 LCE Queries in Linear Space

In this section, we present a data structure that implements the longest common extension (LCE) queries on the text T in $\mathcal{O}(\text{polylog}\, n)$ time. The structure uses $\mathcal{O}(z_e)$ space.

Observe, that to support LCE queries on T, it suffices to support the computation of $\Phi(x, n)$ for any $x \in [0\,..\,n]$. We can then support the queries for Karp–Rabin fingerprints of substrings $T(x\,..\,y]$ using Eq. (2.2) as $\Phi(x, y) = (\Phi(x, n) - \Phi(y, n)) \cdot r^{y-n} \bmod q$. If the queries for suffix fingerprints $\Phi(x, n)$ are supported in time t_{KR} then, accounting for the exponentiation of r, the queries for substring fingerprints $\Phi(x, y)$ take $\mathcal{O}(t_{\mathrm{KR}} + \log n)$ time. With queries for substring fingerprints of T, the value $\mathrm{LCE}(i, j)$ for any $i, j \in [1\,..\,n]$ can then be determined using binary search in $\mathcal{O}((t_{\mathrm{KR}} + \log n) \cdot \log n)$ time. Note that this is easy to improve to $\mathcal{O}((t_{\mathrm{KR}} + \log n) \cdot \log \ell)$, where $\ell = \mathrm{LCE}(i, j)$: we first find the largest integer $k \geq 0$ such that $\mathrm{LCE}(i, j) \geq 2^k$, and then perform the binary search only in the range $[2^k\,..\,2^{k+1})$. The computation of k and the proper search for ℓ takes $\mathcal{O}((t_{\mathrm{KR}} + \log n) \cdot \log \ell)$ time.

5.1 Combinatorial Results

Consider some $x \in [0\,..\,n]$ and let $j \in [1\,..\,e_j]$ be such that $x \in (e_{j-1}\,..\,e_j]$. Observe that since we can precompute the values $\Phi(e_j, n)$ for all $j \in [1\,..\,z_e]$, the computation of $\Phi(x, n)$ can further be reduced to the computation of $\Phi(x, e_j)$ using Eq. (2.1): $\Phi(x, n) = \Phi(x, e_j) \cdot r^{n-e_j} + \Phi(e_j, n) \bmod q$. Thus, the main difficulty lies in computing fingerprints of phrase suffixes. We will show that, letting $x' = \mathrm{map}(x)$, the computation of the fingerprint $\Phi(x, e_j)$ can be reduced either to the computation of $\Phi(x', e_{j'})$ for some $j' \in [1\,..\,z_e]$ such that $x' \leq e_{j'}$, or to the computation of $\Phi(e_{j'}, x')$ for some $j' \in [1\,..\,z_e]$ such that $x' \geq e_{j'}$. The computation of fingerprints of phrase prefixes (i.e., $\Phi(e_j, x)$) is reduced analogously. Employing this strategy, by Corollary 4.2, after a bounded number of steps, we reach a query with the position at the end of some phrase. This case can be solved using precomputed fingerprints and hence terminates the query.

The following two lemmas show the generic reductions of the queries $\Phi(x, e_j)$ and $\Phi(e_j, x)$. The key property of these reductions is that they require a single precomputed fingerprint (per phrase), depending only on $\{b, b', e, e'\}$, but not on x.

LEMMA 5.1. *Let $j \in [1\,..\,z_e]$ and let b, e be such that $e_{j-1} \leq b < e \leq e_j$. Denote $b' = \mathrm{pri}(b, e)$, $e' = b' + (e - b)$, $\Phi_s = \Phi(e+1-L(e'+1), e_j)$, and $j' = \max\{i \in [1\,..\,z_e] : e_i \leq e'\}$. Then, for any $x \in (b\,..\,e]$, letting $x' = e' - (e - x)$, it holds*

$$\Phi(x, e_j) = \begin{cases} \Phi_s - \Phi(e_{j'}, x') \cdot r^{e_j - x} \bmod q & \text{if } x' > e_{j'}, \\ \Phi(x', e_{j'}) \cdot r^{(e_j - e) + (e' - e_{j'})} + \Phi_s \bmod q & \text{otherwise.} \end{cases}$$

Proof. By definition of the primary occurrence, there exists $i \in [1\,..\,z_e]$ such that $e_i \in (b'\,..\,e']$. Thus, it holds $e_{j'} \in (b'\,..\,e']$, and consequently, $L(e'+1) - 1 = e' - e_{j'}$. On the other hand, by definition, we have $e' - x' = e - x$. Consider now two cases (see Fig. 3):

- If $x' > e_{j'}$, then by $T(b'\,..\,e'] = T(b\,..\,e]$ we can write

$$\begin{aligned} T(e + 1 - L(e' + 1)\,..\,e_j] &= T(e + 1 - L(e' + 1)\,..\,e] \cdot T(e\,..\,e_j] \\ &= T(e_{j'}\,..\,e'] \cdot T(e\,..\,e_j] \\ &= T(e_{j'}\,..\,x'] \cdot T(x'\,..\,e'] \cdot T(e\,..\,e_j] \\ &= T(e_{j'}\,..\,x'] \cdot T(x\,..\,e] \cdot T(e\,..\,e_j] \\ &= T(e_{j'}\,..\,x'] \cdot T(x\,..\,e_j]. \end{aligned}$$

  Therefore, by Eq. (2.1), $\Phi(e + 1 - L(e' + 1), e_j) = \Phi(e_{j'}, x') \cdot r^{e_j - x} + \Phi(x, e_j) \bmod q$. Equivalently, after substituting Φ_s, we obtain, $\Phi(x, e_j) = \Phi_s - \Phi(e_{j'}, x') \cdot r^{e_j - x} \bmod q$.
- On the other hand, if $x' \leq e_{j'}$, then by $T(b'\,..\,e'] = T(b\,..\,e]$,

$$\begin{aligned} T(x\,..\,e_j] &= T(x\,..\,e] \cdot T(e\,..\,e_j] \\ &= T(x'\,..\,e'] \cdot T(e\,..\,e_j] \\ &= T(x'\,..\,e_{j'}] \cdot T(e_{j'}\,..\,e'] \cdot T(e\,..\,e_j] \\ &= T(x'\,..\,e_{j'}] \cdot T(e + 1 - L(e' + 1)\,..\,e] \cdot T(e\,..\,e_j] \\ &= T(x'\,..\,e_{j'}] \cdot T(e + 1 - L(e' + 1)\,..\,e_j]. \end{aligned}$$

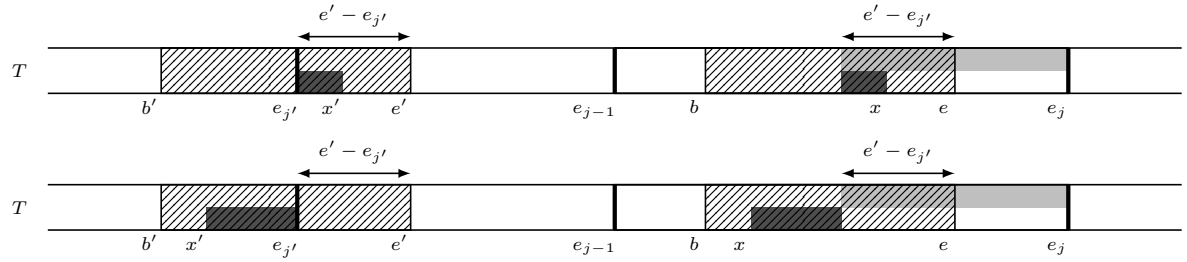

Figure 3: Illustration of Lemma 5.1. If we precompute and store the Karp–Rabin fingerprint $\Phi_s = \Phi(e_j - ((e_j - e) + (e' - e_{j'})), e_j) = \Phi(e + 1 - L(e' + 1), e_j)$ (light gray), then for any $x \in (b \,..\, e]$, the fingerprint $\Phi(x, e_j)$ can be obtained as a combination of Φ_s and $\Phi(e_{j'}, x')$ as $\Phi(x, e_j) = \Phi_s - \Phi(e_{j'}, x') \cdot r^{e_j - x} \bmod q$ (if $x' > e_{j'}$; depicted in the top panel), or as a combination of Φ_s and $\Phi(x', e_{j'})$ as $\Phi(x, e_j) = \Phi(x', e_{j'}) \cdot r^{(e_j - e) + (e' - e_{j'})} + \Phi_s \bmod q$ (if $x' \le e_{j'}$; depicted in the bottom). Bold vertical lines mark phrase boundaries.

Therefore, by Eq. (2.1), $\Phi(x, e_j) = \Phi(x', e_{j'}) \cdot r^{(e_j - e) + (e' - e_{j'})} + \Phi(e + 1 - L(e' + 1), e_j) \bmod q$. After substituting Φ_s, we thus obtain $\Phi(x, e_j) = \Phi(x', e_{j'}) \cdot r^{(e_j - e) + (e' - e_{j'})} + \Phi_s \bmod q$. □

LEMMA 5.2. *Let $j \in [0 \,..\, z_e)$ and let b, e be such that $e_j \le b < e \le e_{j+1}$. Denote $b' = \mathrm{pri}(b, e)$, $\Phi_p = \Phi(e_j, b + 1 + R(b' + 1))$, and $j' = \min\{i \in [1 \,..\, z_e] : e_i > b'\}$. Then, for any $x \in (b \,..\, e]$, letting $x' = b' + (x - b)$, it holds*

$$\Phi(e_j, x) = \begin{cases} (\Phi_p - \Phi(x', e_{j'})) \cdot r^{x' - e_{j'}} \bmod q & \text{if } x' \le e_{j'}, \\ \Phi_p \cdot r^{x' - e_{j'}} + \Phi(e_{j'}, x') \bmod q & \text{otherwise.} \end{cases}$$

Proof. By definition, there exists $i \in [1 \,..\, z_e]$ such that $e_i \in (b' \,..\, e']$. Thus, $e_{j'} \in (b' \,..\, e']$, and consequently, $b' + 1 + R(b' + 1) = e_{j'}$. On the other hand, by definition, we have $x' - b' = x - b$. Consider now two cases:

- If $x' \le e_{j'}$, then by $T(b' \,..\, e'] = T(b \,..\, e]$ we can write

$$\begin{aligned} T(e_j \,..\, b + 1 + R(b' + 1)] &= T(e_j \,..\, b] \cdot T(b \,..\, b + 1 + R(b' + 1)] \\ &= T(e_j \,..\, b] \cdot T(b' \,..\, e_{j'}] \\ &= T(e_j \,..\, b] \cdot T(b' \,..\, x'] \cdot T(x' \,..\, e_{j'}] \\ &= T(e_j \,..\, b] \cdot T(b \,..\, x] \cdot T(x' \,..\, e_{j'}] \\ &= T(e_j \,..\, x] \cdot T(x' \,..\, e_{j'}]. \end{aligned}$$

  Therefore, by Eq. (2.1), $\Phi(e_j, b + 1 + R(b' + 1)) = \Phi(e_j, x) \cdot r^{e_{j'} - x'} + \Phi(x', e_{j'}) \bmod q$. Equivalently, after substituting Φ_p, we obtain, $\Phi(e_j, x) = (\Phi_p - \Phi(x', e_{j'})) \cdot r^{x' - e_{j'}} \bmod q$.
- On the other hand, if $x' > e_{j'}$, then by $T(b' \,..\, e'] = T(b \,..\, e]$,

$$\begin{aligned} T(e_j \,..\, x] &= T(e_j \,..\, b] \cdot T(b \,..\, x] \\ &= T(e_j \,..\, b] \cdot T(b' \,..\, x'] \\ &= T(e_j \,..\, b] \cdot T(b' \,..\, e_{j'}] \cdot T(e_{j'} \,..\, x'] \\ &= T(e_j \,..\, b] \cdot T(b \,..\, b + 1 + R(b' + 1)] \cdot T(e_{j'} \,..\, x'] \\ &= T(e_j \,..\, b + 1 + R(b' + 1)] \cdot T(e_{j'} \,..\, x']. \end{aligned}$$

  Therefore, by Eq. (2.1), $\Phi(e_j, x) = \Phi(e_j, b + 1 + R(b' + 1)) \cdot r^{x' - e_{j'}} + \Phi(e_{j'}, x') \bmod q$. After substituting Φ_p, we thus obtain $\Phi(e_j, x) = \Phi_p \cdot r^{x' - e_{j'}} + \Phi(e_{j'}, x') \bmod q$. □

To use the above lemmas, we observe that whenever $x \in (e_{j-1} \,..\, e_j]$, there always exist positions b, e such that $e_{j-1} \le b < e \le e_j$ and $x' = b' + (b - x) = e' - (e - x) = \mathrm{map}(x)$ for $b' = \mathrm{pri}(b, e)$ and $e' = b' + (e - b)$. More precisely:

Function `SuffixFP`(j, x):
Input: integers $j \in [0\,..\,z_e]$ and $x \in [0\,..\,e_j]$.
Output: the fingerprint $\Phi(x, e_j)$.

if $x = E[j]$ or $x \le E[j-1]$ **then**
  $j' \leftarrow \min\{i \in [0\,..\,z_e] : E[i] \ge x\}$
  $f_2 \leftarrow (\Phi[j'] - \Phi[j]) \cdot r^{E[j]-n}$
  if $x \ne E[j']$ **then**
    $f_1 \leftarrow$ `SuffixFP`(j', x)
  else $f_1 \leftarrow 0$
  return $f_1 \cdot r^{E[j]-E[j']} + f_2 \bmod q$
$(b, e, b') \leftarrow B[j]$; $\ell_j \leftarrow E[j] - E[j-1]$
if $x \in (b\,..\,e]$ **then** ▶ **Case 1**
  $f_2 \leftarrow \Phi^B_s[j]$
  $e' \leftarrow b' + (e - b)$
elsif $x \le E[j-1] + 2\ell_j/3$ **then** ▶ **Case 2**
  $f_2 \leftarrow \Phi^P_s[j]$
  $e \leftarrow E[j-1] + 2\ell_j/3$; $e' = P[j] + 2\ell_j/3$
else ▶ **Case 3**
  $f_2 \leftarrow 0$
  $e \leftarrow E[j]$; $e' \leftarrow E[S[j]]$
$j' \leftarrow \max\{i \in [1\,..\,z_e] : E[i] \le e'\}$
$x' \leftarrow e' - (e - x)$ ▶ $x' = \mathrm{map}(x)$
if $x' > E[j']$ **then**
  $f_1 \leftarrow$ `PrefixFP`(j', x')
  return $f_2 - f_1 \cdot r^{E[j]-x} \bmod q$
else
  $f_1 \leftarrow$ `SuffixFP`(j', x')
  return $f_1 \cdot r^{(E[j]-e)+(e'-E[j'])} + f_2 \bmod q$

Function `PrefixFP`(j, x):
Input: integers $j \in [0\,..\,z_e]$ and $x \in [e_j\,..\,n]$.
Output: the fingerprint $\Phi(e_j, x)$.

if $x = E[j]$ or $x \ge E[j+1]$ **then**
  $j' \leftarrow \max\{i \in [0\,..\,z_e] : E[i] \le x\}$
  $f_1 \leftarrow (\Phi[j] - \Phi[j']) \cdot r^{E[j']-n}$
  if $x \ne E[j']$ **then**
    $f_2 \leftarrow$ `PrefixFP`(j', x)
  else $f_2 \leftarrow 0$
  return $f_1 \cdot r^{x-E[j']} + f_2 \bmod q$
$(b, e, b') \leftarrow B[j+1]$; $\ell_{j+1} \leftarrow E[j+1] - E[j]$
if $x \in (b\,..\,e]$ **then** ▶ **Case 1**
  $f_1 \leftarrow \Phi^B_p[j+1]$
elsif $x \le E[j] + 2\ell_{j+1}/3$ **then** ▶ **Case 2**
  $f_1 \leftarrow \Phi^P_p[j+1]$
  $b \leftarrow E[j]$; $b' \leftarrow P[j+1]$
else ▶ **Case 3**
  $f_1 \leftarrow \Phi^S_p[j+1]$
  $b \leftarrow E[j]$; $b' \leftarrow E[S[j+1]] - \ell_{j+1}$
$j' \leftarrow \min\{i \in [1\,..\,z_e] : E[i] > b'\}$
$x' \leftarrow b' + (x - b)$ ▶ $x' = \mathrm{map}(x)$
if $x' \le E[j']$ **then**
  $f_2 \leftarrow$ `SuffixFP`(j', x')
  return $(f_1 - f_2) \cdot r^{x'-E[j']} \bmod q$
else
  $f_2 \leftarrow$ `PrefixFP`(j', x')
  return $f_1 \cdot r^{x'-E[j']} + f_2 \bmod q$

Figure 4: Pseudocode of the functions for querying the values $\Phi(x, e_j)$ (left) and $\Phi(e_j, x)$ (right).

Case 1: If $x \in \mathcal{I}_{j,p_j}$, then b, e are such that $(b\,..\,e] = \mathcal{I}_{j,p_j}$,
Case 2: If $x \notin \mathcal{I}_{j,p_j}$ and $x \le e_{j-1} + \lfloor 2\ell_j/3 \rfloor$, then $b = e_{j-1}$ and $e = e_{j-1} + \lfloor 2\ell_j/3 \rfloor$,
Case 3: If $x \notin \mathcal{I}_{j,p_j}$ and $x > e_{j-1} + \lfloor 2\ell_j/3 \rfloor$, then $b = e_{j-1}$ and $e = e_j$.

Thus, when querying $\Phi(x, e_j)$ (resp. $\Phi(e_j, x)$) we can always apply Lemma 5.1 (resp. Lemma 5.2).

5.2 Data Structure With the above observation in mind, the data structure to compute $\Phi(x, n)$ for $x \in [0\,..\,n]$ is obtained by augmenting the structure from Section 4 as follows. We store the following six additional arrays with precomputed fingerprints. For $j \in [1\,..\,z_e]$, the arrays are defined as follows:

- $\Phi[j] = \Phi(e_j, n)$,
- $\Phi^B_p[j] = \Phi(e_{j-1}, b + 1 + R(b' + 1))$, where $(b, \cdot, b') = B[j]$ (defined as in Section 4),
- $\Phi^B_s[j] = \Phi(e + 1 - L(e' + 1), e_j)$, where $(b, e, b') = B[j]$ and $e' = b' + (e - b)$,
- $\Phi^P_p[j] = \Phi(e_{j-1}, e_{j-1} + 1 + R(b' + 1))$, where $b' = P[j]$ (defined as in Section 4),
- $\Phi^P_s[j] = \Phi(e + 1 - L(e' + 1), e_j)$, where $e = e_{j-1} + \lfloor 2\ell_j/3 \rfloor$ and $e' = P[j] + \lfloor 2\ell_j/3 \rfloor$,
- $\Phi^S_p[j] = \Phi(e_{j-1}, e_{j-1} + 1 + R(b' + 1))$, where $b' = e_{j'} - \ell_j$ and $j' = \mathrm{src}(j)$.

We also define $\Phi[0] = \Phi(e_0, n) = \Phi(0, n)$.

5.3 Query Algorithm The pseudocode of the query algorithm is given in Fig. 4. The two functions `SuffixFP` and `PrefixFP` compute the fingerprint of an arbitrary suffix (resp. prefix) of $T(0\,..\,e_j]$ (resp. $T(e_j\,..\,n]$). To compute the fingerprint of a text suffix, i.e., $\Phi(x, n)$ for some $x \in [0\,..\,n]$, the code is invoked, as $\Phi(x, n) :=$ `SuffixFP`(z_e, x). Note that when invoking `SuffixFP`(j, x) (resp. `PrefixFP`(j, x)), we do not require $x > e_{j-1}$ (resp. $x \le e_{j+1}$) as in Lemmas 5.1 and 5.2, but allow any $x \in [0\,..\,e_j]$ (resp. $x \in [e_j\,..\,n]$). The condition $x > e_{j-1}$ (resp. $x \le e_{j+1}$)

is ensured in Lines 1–7 (which also handle the case $x = e_j$) by using the precomputed fingerprints at phrase boundaries. Thus, when entering Line 8, we are guaranteed that $x \in (e_{j-1} \,..\, e_j)$ (resp. $x \in (e_j \,..\, e_{j+1})$).

In Lines 8–17 (resp. 8–16), we then determine the positions b, e satisfying $e_{j-1} \le b < e \le e_j$ (resp. $e_j \le b < e \le e_{j+1}$), and positions $b' = \mathrm{pri}(b, e)$ and $e' = b' + (e - b)$ such that $x' = \mathrm{map}(x) = b' + (x - b) = e' - (e - x)$ (see the three cases above; unused values are not computed).

Lastly, in Lines 18–25 (resp. 17–24), we apply Lemma 5.1 (resp. Lemma 5.2) to position x.

The correctness of the algorithm follows from Lemmas 5.1 and 5.2. To bound the query time, we first observe that when the condition in Line 1 evaluates true, then either $x = e_j$ and the query terminates, or $x \notin (e_{j-1} \,..\, e_j]$ (resp. $x \notin [e_j \,..\, e_{j+1})$) and the function performs a recursive call in Line 5. Note that a recursive call from Line 5 will never happen consecutively twice, since the definition of j' (Line 2) guarantees that $x \in (e_{j'-1} \,..\, e_{j'}]$ (resp. $x \in [e_{j'} \,..\, e_{j'+1})$). Thus, the number of function calls is determined by the number of applications of the map function to reach the last position of some phrase boundary. Thus, if we choose the selector sequence as in Corollary 4.2, then with high probability the number of function calls is bounded by $\mathcal{O}(\log^4 n)$. Accounting for the predecessor queries in Line 18 (resp. 17), and the exponentiation in Lines 22 and 25 (resp. 21 and 24), computing $\Phi(x, n)$ for any $x \in [0 \,..\, n]$ takes $t_{\mathrm{KR}} = \mathcal{O}(\log^5 n)$ time.

Thus, using the probabilistic method we have proved the following result.

THEOREM 5.1. *For any text T of length n, there exists a data structure of size $\mathcal{O}(z_e)$ that, for any $x, y \in [1 \,..\, n]$, computes $\ell = \mathrm{LCE}(x, y)$ in $\mathcal{O}(\log^5 n \cdot \log \ell)$ worst-case time.*

Acknowledgments

We thank the anonymous reviewers for their insightful comments and suggestions that helped to improve the presentation of our paper.

References

[1] Jyrki Alakuijala, Andrea Farruggia, Paolo Ferragina, Eugene Kliuchnikov, Robert Obryk, Zoltan Szabadka, and Lode Vandevenne. Brotli: A general-purpose data compressor. *ACM Trans. Inf. Syst.*, 37(1), 2018. doi:10.1145/3231935.

[2] Diego Arroyuelo, Gonzalo Navarro, and Kunihiko Sadakane. Stronger Lempel-Ziv based compressed text indexing. *Algorithmica*, 62(1-2):54–101, 2012. doi:10.1007/s00453-010-9443-8.

[3] Hideo Bannai, Travis Gagie, and Tomohiro I. Refining the r-index. *Theor. Comput. Sci.*, 812:96–108, 2020. doi:10.1016/j.tcs.2019.08.005.

[4] Djamal Belazzougui, Manuel Cáceres, Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Gonzalo Navarro, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Block trees. *J. Comput. Syst. Sci.*, 117:1–22, 2021. doi:10.1016/j.jcss.2020.11.002.

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

[6] Philip Bille, Anders Roy Christiansen, Patrick Hagge Cording, and Inge Li Gørtz. Finger search in grammar-compressed strings. *Theory Comput. Syst.*, 62(8):1715–1735, 2018. doi:10.1007/s00224-017-9839-9.

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

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

[9] Christina Boucher, Ondrej Cvacho, Travis Gagie, Jan Holub, Giovanni Manzini, Gonzalo Navarro, and Massimiliano Rossi. PFP compressed suffix trees. In *ALENEX*, pages 60–72, 2021. doi:10.1137/1.9781611976472.5.

[10] Christina Boucher, Travis Gagie, Tomohiro I, Dominik Köppl, Ben Langmead, Giovanni Manzini, Gonzalo Navarro, Alejandro Pacheco, and Massimiliano Rossi. PHONI: Streamed matching statistics with multi-genome references. In *DCC*, pages 193–202, 2021. doi:10.1109/DCC50243.2021.00027.

[11] Christina Boucher, Travis Gagie, Alan Kuhnle, Ben Langmead, Giovanni Manzini, and Taher Mun. Prefix-free parsing for building big BWTs. *Algorithms Mol. Biol.*, 14(1):13:1–13:15, 2019. doi:10.1186/s13015-019-0148-5.

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

[13] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *Trans. Inf. Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116.

[14] Anders Roy Christiansen, Mikko Berggren Ettienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Optimal-time dictionary-compressed indexes. *ACM Trans. Alg. 17*, 17(1), 2019. arXiv:1811.12779.

[15] Francisco Claude and Gonzalo Navarro. Self-indexed grammar-based compression. *Fundam. Informaticae*, 111(3):313–337, 2011. doi:10.3233/FI-2011-565.
[16] Francisco Claude, Gonzalo Navarro, and Alejandro Pacheco. Grammar-compressed indexes with logarithmic search time. *J. Comput. Syst. Sci.*, 118:53–74, 2021. doi:10.1016/j.jcss.2020.12.001.
[17] Dustin Cobas, Travis Gagie, and Gonzalo Navarro. A fast and small subsampled r-index. In *CPM*, pages 13:1–13:16, 2021. doi:10.4230/LIPIcs.CPM.2021.13.
[18] Yann Collet. LZ4-Extremely fast compression. http://lz4.org/. Accessed: 2021-03-18.
[19] Yann Collet and Murray S. Kucherawy. Zstandard compression and the application/zstd media type. *RFC8478*, 2018. https://tools.ietf.org/html/rfc8478.
[20] Martin Dietzfelbinger, Joseph Gil, Yossi Matias, and Nicholas Pippenger. Polynomial hash functions are reliable. In *ICALP*, pages 235–246, 1992. doi:10.1007/3-540-55719-9_77.
[21] Héctor Ferrada, Travis Gagie, Tommi Hirvola, and Simon J Puglisi. Hybrid indexes for repetitive datasets. *Philos. Trans. R. Soc. A*, 372, 2014.
[22] Paolo Ferragina and Giovanni Manzini. On compressing the textual web. In *WSDM*, pages 391–400, 2010. doi:10.1145/1718487.1718536.
[23] Johannes Fischer and Pawel Gawrychowski. Alphabet-dependent string searching with Wexponential search trees. In *CPM*, pages 160–171, 2015. Full version: arxiv.org/abs/1302.3347. doi:10.1007/978-3-319-19929-0_14.
[24] Travis Gagie. r-indexing Wheeler graphs, 2021. arXiv:2101.12341.
[25] Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. LZ77-based self-indexing with faster pattern matching. In *LATIN*, pages 731–742, 2014. doi:10.1007/978-3-642-54423-1_63.
[26] Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. A faster grammar-based self-index. In *LATA*, pages 240–251, 2012. doi:10.1007/978-3-642-28332-1_21.
[27] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. On the approximation ratio of Lempel-Ziv parsing. In *LATIN*, pages 490–503, 2018. doi:10.1007/978-3-319-77404-6_36.
[28] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM*, 67(1):1–54, 2020. doi:10.1145/3375890.
[29] Moses Ganardi, Artur Jez, and Markus Lohrey. Balancing straight-line programs. *J. ACM*, 68(4):27:1–27:40, 2021. doi:10.1145/3457389.
[30] Pawel Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Lacki, and Piotr Sankowski. Optimal dynamic strings. In *SODA*, pages 1509–1528, 2018. Full version: arxiv.org/abs/1511.02612. doi:10.1137/1.9781611975031.99.
[31] Genomics England. The 100000 Genomes Project. https://www.genomicsengland.co.uk/about-genomics-england/the-100000-genomes-project/. Accessed: 2021-03-18.
[32] Tomohiro I. Longest common extensions with recompression. In *CPM*, pages 18:1–18:15, 2017. doi:10.4230/LIPIcs.CPM.2017.18.
[33] Takumi Ideue, Takuya Mieno, Mitsuru Funakoshi, Yuto Nakashima, Shunsuke Inenaga, and Masayuki Takeda. On the approximation ratio of LZ-End to LZ77. In *SPIRE*, pages 114–126, 2021. doi:10.1007/978-3-030-86692-1_10.
[34] Artur Jeż. A really simple approximation of smallest grammar. *Theor. Comput. Sci.*, 616:141–150, 2016. doi:10.1016/j.tcs.2015.12.032.
[35] Artur Jeż. Recompression: A simple and powerful technique for word equations. *J. ACM*, 63(1):4:1–4:51, 2016. doi:10.1145/2743014.
[36] Juha Kärkkäinen. *Repetition-based Text Indexes*. PhD thesis, University of Helsinki, 1999.
[37] Dominik Kempa. Optimal construction of compressed indexes for highly repetitive texts. In *SODA*, pages 1344–1357, 2019. doi:10.1137/1.9781611975482.82.
[38] Dominik Kempa and Tomasz Kociumaka. Resolution of the Burrows-Wheeler transform conjecture. In *FOCS*, pages 1002–1013, 2020. doi:10.1109/FOCS46700.2020.00097.
[39] Dominik Kempa and Dmitry Kosolobov. LZ-End parsing in compressed space. In *DCC*, pages 350–359, 2017. doi:10.1109/DCC.2017.73.
[40] Dominik Kempa and Dmitry Kosolobov. LZ-End parsing in linear time. In *ESA*, pages 53:1–53:14, 2017. doi:10.4230/LIPIcs.ESA.2017.53.
[41] Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: String attractors. In *STOC*, pages 827–840, 2018. doi:10.1145/3188745.3188814.
[42] Takuya Kida, Tetsuya Matsumoto, Yusuke Shibata, Masayuki Takeda, Ayumi Shinohara, and Setsuo Arikawa. Collage system: A unifying framework for compressed pattern matching. *Theor. Comput. Sci.*, 298(1):253–272, 2003. doi:10.1016/S0304-3975(02)00426-7.
[43] Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Towards a definitive measure of repetitiveness. In *LATIN*, pages 207–219, 2020. doi:10.1007/978-3-030-61792-9_17.
[44] S. Rao Kosaraju and Giovanni Manzini. Compression of low entropy strings with Lempel-Ziv algorithms. *SIAM J.*

Comput., 29(3):893–911, 1999. doi:10.1109/SEQUEN.1997.666907.
[45] Sebastian Kreft and Gonzalo Navarro. LZ77-like compression with fast random access. In *DCC*, pages 239–248, 2010. doi:10.1109/DCC.2010.29.
[46] Sebastian Kreft and Gonzalo Navarro. On compressing and indexing repetitive sequences. *Theor. Comput. Sci.*, 483:115–133, 2013. doi:10.1016/j.tcs.2012.02.006.
[47] Alan Kuhnle, Taher Mun, Christina Boucher, Travis Gagie, Ben Langmead, and Giovanni Manzini. Efficient construction of a complete index for pan-genomics read alignment. *J. Comput. Biol.*, 27(4):500–513, 2020. doi:10.1089/cmb.2019.0309.
[48] Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *J. Comput. Syst. Sci.*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.
[49] Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Trans. Inf. Theory*, 22(1):75–81, 1976. doi:10.1109/TIT.1976.1055501.
[50] Moshe Lewenstein. LCP magic. In *CPM*, page 11, 2013. doi:10.1007/978-3-642-38905-4_2.
[51] Matt Mahoney. Large Text Compression Benchmark. http://mattmahoney.net/dc/text.html. Accessed: 2021-03-19.
[52] Shirou Maruyama, Masaya Nakahara, Naoya Kishiue, and Hiroshi Sakamoto. ESP-index: A compressed index based on edit-sensitive parsing. *J. Discrete Algorithms*, 18:100–112, 2013. doi:10.1016/j.jda.2012.07.009.
[53] Gonzalo Navarro. Indexing highly repetitive string collections, 2020. arXiv:2004.02781.
[54] Gonzalo Navarro. Indexing highly repetitive string collections, part I: Repetitiveness measures. *ACM Comput. Surv.*, 54(2):29:1–29:31, 2021. doi:10.1145/3434399.
[55] Gonzalo Navarro. Indexing highly repetitive string collections, part II: Compressed indexes. *ACM Comput. Surv.*, 54(2):26:1–26:32, 2021. doi:10.1145/3432999.
[56] Gonzalo Navarro and Nicola Prezza. Universal compressed text indexing. *Theor. Comput. Sci.*, 762:41–50, 2019. doi:10.1016/j.tcs.2018.09.007.
[57] Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Fully dynamic data structure for LCE queries in compressed space. In *MFCS*, pages 72:1–72:15, 2016. doi:10.4230/LIPIcs.MFCS.2016.72.
[58] Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Dynamic index and LZ factorization in compressed space. *Discret. Appl. Math.*, 274:116–129, 2020. doi:10.1016/j.dam.2019.01.014.
[59] Takaaki Nishimoto and Yasuo Tabei. Optimal-time queries on BWT-runs compressed indexes. In *ICALP*, pages 101:1–101:15, 2021. doi:10.4230/LIPIcs.ICALP.2021.101.
[60] Takaaki Nishimoto and Yasuo Tabei. R-enum: Enumeration of characteristic substrings in BWT-runs bounded space. In *CPM*, pages 21:1–21:21, 2021. doi:10.4230/LIPIcs.CPM.2021.21.
[61] Recipients of IEEE Medal of Honor. https://corporate-awards.ieee.org/recipients/current-recipients/, 2021.
[62] Nicola Prezza. Optimal rank and select queries on dictionary-compressed text. In *CPM*, pages 4:1–4:12, 2019. doi:10.4230/LIPIcs.CPM.2019.4.
[63] Molly Przeworski, Richard R. Hudson, and Anna Di Rienzo. Adjusting the focus on human variation. *Trends Genet.*, 16(7):296–302, 2000. doi:10.1016/S0168-9525(00)02030-8.
[64] Wojciech Rytter. Application of Lempel–Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1–3):211–222, 2003. doi:10.1016/S0304-3975(02)00777-6.
[65] Sandip Sinha and Omri Weinstein. Local decodability of the Burrows-Wheeler transform. In *STOC*, pages 744–755, 2019. doi:10.1145/3313276.3316317.
[66] Jouni Sirén, Niko Välimäki, Veli Mäkinen, and Gonzalo Navarro. Run-length compressed indexes are superior for highly repetitive sequence collections. In *SPIRE*, pages 164–175, 2008. doi:10.1007/978-3-540-89097-3_17.
[67] James A. Storer and Thomas G. Szymanski. The macro model for data compression. In *STOC*, pages 30–39, 1978. doi:10.1145/800133.804329.
[68] Yoshimasa Takabatake, Yasuo Tabei, and Hiroshi Sakamoto. Improved ESP-index: A practical self-index for highly repetitive texts. In *SEA*, pages 338–350, 2014. doi:10.1007/978-3-319-07959-2_29.
[69] Kazuya Tsuruta, Dominik Köppl, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Grammar-compressed self-index with Lyndon words, 2020. arXiv:2004.05309.
[70] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *Trans. Inf. Theory*, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.

A Deferred Proofs

THEOREM A.1. *Let $\widehat{z_e}$ denote the size of the LZ-End parsing, as defined in [46]. Every string of length n satisfies $\widehat{z_e} = \mathcal{O}(z \log^2 n)$.*

Proof. The proof follows closely the proof of Theorem 3.1. We observe, that the only two places where we used the definition of the LZ-End parsing are:

1. When proving that any substring $X \in \mathcal{S}_{2^k}$ is associated with at most two phrases, we used the fact that assuming this is not true, there exists $j \in [1\,..\,z_e]$ such that when the algorithm is adding the phrase $T(e_{j-2}\,..\,e_{j-1}]$ to the parsing, there exists a substring S of length $|S| > e_{j-1} - e_{j-2}$ occurring at position $e_{j-2}+1$, and that has an earlier occurrence in T ending at the end of an already existing phrase $T(e_{i-1}\,..\,e_i]$ for some $i < j-1$. This contradicts the definition of LZ-End parsing from Section 2. We now observe that this also contradicts the definition of LZ-End parsing from [46], since the algorithm in this case would create the phrase at least of length $|S|+1 > e_{j-1} - e_{j-2}$.
2. Similarly, when proving that all $\ell = e_j - e_{j-1}$ substrings associated with the phrase $T(e_{j-1}\,..\,e_j]$ are distinct, we observed that assuming the opposite, there exists a substring S of length $|S| > e_j - e_{j-1}$ occurring at position $e_{j-1}+1$, and that has an earlier occurrence ending at the end of (already existing) phrase $T(e_{j-2}\,..\,e_{j-1}]$. This again contradicts not only the definition from Section 2, but also from [46], since then the algorithm would also create a phrase of length at least $|S|+1 > e_j - e_{j-1}$. □

FACT 3.1. *For any text of length n and any threshold $t \geq 1$, it holds $z(t) \leq z + \frac{n}{t}$.*

Proof. We first prove that the parsing $T = F_1 \cdots F_{z(t)}$ computed by always selecting as F_i the longest prefix of $F_i \cdots F_{z(t)}$ having an earlier occurrence and whose length does not exceed t, produces the optimal (i.e., having the smallest possible number of phrases) parsing among LZ77-like parsings $T = F'_1 \cdots F'_f$ satisfying $\max_{i=1}^{f} |F'_i| \leq t$. Let $T = G_1 \cdots G_q$ be one of such smallest parsings. We will show that for every $i \in [1\,..\,z(t)]$, it holds $|F_1 \cdots F_i| \geq |G_1 \cdots G_i|$. This proves that $z(t) \leq q$, since otherwise we would have $|F_1 \cdots F_q| < n$ which by $n = |G_1 \cdots G_q|$ would imply $|F_1 \cdots F_q| < |G_1 \cdots G_q|$, contradicting the above claim for $i = q$.

To show the claim, suppose that there exists $i \in [1\,..\,z(t)]$ such that $|F_1 \cdots F_i| < |G_1 \cdots G_i|$ and let i be the smallest such index. Note that since we always have $|F_1| = |G_1| = 1$, it holds $i \geq 2$. We observe that since $|F_1 \cdots F_{i-1}| \geq |G_1 \cdots G_{i-1}|$, the factor G_i starts in T not later than F_i. Let G'_i be a suffix of G_i of length $|G_1 \cdots G_i| - |F_1 \cdots F_{i-1}|$. Note that: (1) $|G'_i| \leq |G_i| \leq t$, (2) G'_i has an earlier occurrence in T (as a suffix of an earlier occurrence of G_i), and (3) $|G'_i| = (|G_1 \cdots G_i| - |F_1 \cdots F_i|) + |F_i| > |F_i|$. This contradicts that F_i was selected as the longest prefix of $F_i \cdots F_{z(t)}$ having an earlier occurrence and a length not exceeding t. This proves the claim, and consequently, that $z(t) \leq q$.

We are now ready to prove the main claim. Let $H_1 \cdots H_z$ be the LZ77 parsing of T. We use it to construct an LZ77-like parsing $H'_1 \cdots H'_{z'}$ of T satisfying $\max_{i=1}^{z'} |H'_i| \leq t$ and $z' = \mathcal{O}(z + \frac{n}{t})$ as follows. Each phrase H_i satisfying $|H_i| \leq t$ occurs in the parsing $H'_1 \cdots H'_{z'}$ without change. However, if a phrase H_i satisfies $|H_i| > t$, we arbitrarily break it down into $\lceil |H_i|/t \rceil$ phrases of length not exceeding t. If $\ell_1, \ldots, \ell_k$ denotes the sequence of lengths for all phrases longer than t, then by $\sum_{i=1}^{k} \ell_i \leq n$ and $k \leq z$, we obtain $z' = z - k + \sum_{i=1}^{k} \lceil \ell_i/t \rceil \leq z + \frac{1}{t} \sum_{i=1}^{k} \ell_i \leq z + \frac{n}{t}$. By combining this with the above optimality result, we obtain $z(t) \leq z' \leq z + \frac{n}{t}$. □