# Counting Distinct Square Substrings in Sublinear Time

**Panagiotis Charalampopoulos**
King's College London, UK

**Manal Mohamed**
Birkbeck, University of London, UK

**Jakub Radoszewski**
University of Warsaw, Poland

**Wojciech Rytter**
University of Warsaw, Poland

**Tomasz Waleń**
University of Warsaw, Poland

**Wiktor Zuba**
University of Warsaw, Poland

## Abstract

We show that the number of distinct squares in a packed string of length $n$ over an alphabet of size $\sigma$ can be computed in $\mathcal{O}(n/\log_\sigma n)$ time in the word-RAM model of computation. This paper is the first to introduce a sublinear time algorithm for the packed version of squares counting. The packed representation of a string of length $n$ over an alphabet of size $\sigma$ is given as a sequence of $\mathcal{O}(n/\log_\sigma n)$ machine words in the word-RAM model (a machine word consists of $\omega \geq \log_2 n$ bits).

Previously it was known how to count distinct squares in $\mathcal{O}(n)$ time [Gusfield and Stoye, JCSS 2004], even for a string over an integer alphabet, see [Crochemore et al., TCS 2014; Bannai et al., CPM 2017; Charalampopoulos et al., SPIRE 2020]. We use techniques of squares extraction from runs described by Crochemore et al. [TCS 2014]. However, the packed model requires novel approaches. In particular, we need an $\mathcal{O}(n/\log_\sigma n)$ sized representation of all *long-period* runs (runs with periods that are $\Omega(\log_\sigma n)$) which guarantees sublinear time counting of potentially linearly-many implied squares. The long-period runs with a string period that is periodic itself (called *layer runs*) are an obstacle, since their number can be $\Omega(n)$. Fortunately, the number of all other long-period runs is $\mathcal{O}(n/\log_\sigma n)$ and we can construct an implicit representation of all long-period runs in $\mathcal{O}(n/\log_\sigma n)$ time by adopting the insights of Amir et al. [ESA 2019], combined with sublinear time tools provided by the `PILLAR` model of computations in case of packed strings. We count squares in layer runs in sublinear time by exploiting combinatorial properties of types of pyramidally-shaped groups of layer runs. As a by-product, we discover several new structural properties of runs.

Another difficulty is to compute, in sublinear time, locations of Lyndon roots of runs in packed strings, which is needed for grouping of runs that can generate equal squares. To overcome this difficulty, we introduce *sparse-Lyndon roots* which are based on the notion of string synchronizers proposed by Kempa and Kociumaka [STOC 2019].

**2012 ACM Subject Classification** Theory of computation → Pattern matching

**Keywords and phrases** square in a string, packed model, run (maximal repetition), Lyndon word

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2025.36

**Related Version** *Full Version*: `http://arxiv.org/abs/2508.03930`

**Funding** *Jakub Radoszewski*: Supported by the Polish National Science Center, grant no. 2022/46/E/ST6/00463.

50th International Symposium on Mathematical Foundations of Computer Science (MFCS 2025).
Editors: Paweł Gawrychowski, Filip Mazowiecki, and Michał Skrzypczak; Article No. 36; pp. 36:1–36:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

# 1 Introduction

We consider a problem of counting distinct squares (and more generally, powers) in a string. Such problems are important not only from a purely theoretical point of view, but are also relevant in some applications in bioinformatics (see the book [27]). Strings of the form $X^2 = XX$, for a non-empty string $X$, called *squares* (or tandem repeats), are the most natural type of repetition.

A fundamental algorithmic problem related to squares is checking if a given string of length $n$ is *square-free*, that is, if it avoids square substrings. Thue's construction of an infinite ternary square-free string [44] can be viewed as the beginning of combinatorics on words. The first $\mathcal{O}(n \log n)$-time algorithm for checking square-freeness was given by Main and Lorentz [40]. An $\mathcal{O}(n)$-time algorithm for this problem, for the case of a constant-sized alphabet, was proposed by Crochemore [18]. Subsequently, $\mathcal{O}(n)$-time algorithms for square-freeness over an integer alphabet and over a general ordered alphabet follow from Kolpakov and Kucherov's [36] and Ellert and Fischer's [23] algorithms for computing runs under these assumptions, respectively. Most recently, Ellert, Gawrychowski, and Gourdel [24] obtained an $\mathcal{O}(n \log \sigma)$-time algorithm for testing square-freeness of a string over a general unordered alphabet of size $\sigma$; they also showed that the algorithm is optimal under these assumptions. Square-freeness was also studied in on-line [29, 37], parallel [2, 3, 20] and dynamic [1] settings.

A much more challenging problem than testing square-freeness, that has received significant attention, is computing the number of *distinct* square substrings of a given string. Fraenkel and Simpson were the first to show that a string of length $n$ contains $\mathcal{O}(n)$ dictinct squares [26]. Brlek and Li very recently, using arguments from linear algebra and graph theory, improved the $2n$ upper bound of Fraenkel and Simpson to just $n$; see [11, 10].

Linear-time algorithms for counting distinct square substrings were proposed by Gusfield and Stoye [28], Crochemore et al. [19], Bannai, Inenaga, and Köppl [6], and Charalampopoulos et al. [14]; notably, the last three results work for a string over an integer (generally, linearly sortable) alphabet. As already mentioned, testing square-freeness is a simpler problem than counting distinct squares. In particular, for a general (ordered) alphabet, element distinctness can be reduced in linear time to counting squares[1], and the latter problem is hard: it requires $\Omega(n \log n)$ time in the comparison model [8].

In the word-RAM model of computation with word size $\Theta(\log n)$, we may store up to $\Omega(\log_\sigma n)$ string characters in a single machine word, where $\sigma$ is the size of the alphabet. The *packed representation* of a length-$n$ string $S$ over an integer alphabet $[0\mathinner{.\,.}\sigma)$ is a sequence of $\mathcal{O}(n/\log_\sigma n)$ integers, each encoding a fragment of $S$ of length $\mathcal{O}(\log_\sigma n)$.

A recent line of work has yielded $o(n)$-time solutions for several basic stringology problems in the setting where the input string(s) is/are given in packed form (the *packed setting*). These include pattern matching [7] and indexing [31, 41], computing the LZ factorization and BWT [22, 30, 31, 32], the longest common substring [13], the longest palindromic substring [16], the Lyndon array [4], and covers of a string [42]. While for some of the discussed problems, such as pattern matching, optimal $\mathcal{O}(n/\log_\sigma n)$-time algorithms exist, for several others, such as BWT construction, the best known algorithms run in time $\mathcal{O}(n\sqrt{\log n}/\log_\sigma n)$. A recent work of Kempa and Kociumaka [33] shed light on the source of difficulty of several stringology problems for which the state-of-the-art solutions take $\mathcal{O}(n\sqrt{\log n}/\log_\sigma n)$ time.

[1] As for the reduction, a sequence $a_1, \ldots, a_n$ contains a repeating element, if and only if, string $a_1^2 b_1 a_2^2 b_2 \cdots a_n^2 b_n$, for a square-free string $b_1 b_2 \ldots b_n$ (say, a prefix of the infinite ternary square free string [44]), contains less than $n$ distinct square substrings.

We are the first to study the problem of counting squares in the packed setting. The problem is formally defined as follows.

Packed Counting Of Distinct Squares
**Input:** A string $T$ of length $n$ over alphabet $[0\,..\,\sigma)$ given in a packed representation.
**Output:** $|\mathsf{squares}(T)|$, where $\mathsf{squares}(T)$ denotes the set of all squares that are equal to some substring of $T$.

▶ **Example 1.** Consider string $T = (\mathtt{ab})^{1000}(\mathtt{ba})^{1000}$; see Figure 1 for a comparison. This string contains 2000 distinct squares. We have

$$\mathsf{squares}(T) = \{\mathtt{b}(\mathtt{ab})^i\mathtt{b}(\mathtt{ab})^i : 0 \le i \le 999\} \cup \{(\mathtt{ab})^{2i} : 1 \le i \le 500\} \cup \{(\mathtt{ba})^{2i} : 1 \le i \le 500\}.$$

We settle the time complexity of the Packed Counting Of Distinct Squares problem, classifiying it as one of the elementary stringology problems that admit an $\mathcal{O}(n/\log_\sigma n)$-time solution in the packed setting.

▶ **Theorem 2.** *The Packed Counting Of Distinct Squares problem can be solved in $\mathcal{O}(n/\log_\sigma n)$ time.*

Moreover, our algorithm can report $k$ distinct squares, for any $k$ between 0 and the actual number of distinct squares in the string, in $\mathcal{O}(n/\log_\sigma n + k)$ time. Our algorithm generalizes readily to powers with higher exponent: for any integer $t \ge 2$, a string of length $n$ contains at most $n/(t-1)$ powers with exponent $t$ [39] and we can compute the actual number of those in a packed string in $\mathcal{O}(n/\log_\sigma n)$ time.

**Other related work.** A string of length $n$ contains $\mathcal{O}(n \log n)$ substrings that are primitively rooted squares and they can all be computed in $\mathcal{O}(n \log n)$ time; see [17, 21, 43]. The same representation is computed by Apostolico and Breslauer's parallel algorithm in [3]. We note that these algorithms compute *all occurrences* of primitively rooted squares and are not concerned with whether any two computed substrings correspond to the same square.

## Technical Overview

Let $T$ be a string of length $n$ over alphabet $[0\,..\,\sigma)$. It was already observed by Crochemore et al. [19] that distinct squares in $T$ can be computed from runs (maximal periodic fragments). This is because a square $U^2$ can be extended to a unique run with the same period as the primitive root of $U^2$ and a string of length $n$ contains $\mathcal{O}(n)$ runs that can be computed in $\mathcal{O}(n)$ time [36, 5, 23]. Amir et al. [1] (see also [12]) proposed an algorithm that efficiently maintains a representation of runs in a dynamic string. We note that their algorithm works in the $\mathtt{PILLAR}$ model of Charalampopoulos, Kociumaka, and Wellnitz [15] and it can be used to compute a representation of all runs in $T$ with period at least $\log_\sigma n$ in $\mathcal{O}(n/\log_\sigma n)$ time. All the remaining runs in $T$ either fit in a machine word or are so-called $\tau$-runs (see [31]); the number of the latter is $\mathcal{O}(n/\log_\sigma n)$. As a result, a representation of all runs in $T$ can be computed in $\mathcal{O}(n/\log_\sigma n)$ time and space. This representation can be used to compute the longest square in a string in $\mathcal{O}(n/\log_\sigma n)$ time in a straightforward way.

Computing the number of distinct squares from this representation is much more challenging. The first obstacle towards achieving this goal is the difficulty in grouping the runs with respect to their Lyndon roots in packed strings. We overcome this obstacle by introducing a version of Lyndon roots more suitable for the packed model, called here *sparse* Lyndon roots. Positions of such nonstandard roots are based on synchronizing sets of positions (see Kempa and Kociumaka [31]).

Let us call squares $U^2$ whose root $U$ is both primitive and highly periodic (here: containing at least 4 occurrences of the period) *special*, while remaining squares are called *plain*. Plain squares can be efficiently counted by combining tabulation with the approach of [19] applied on a selected $\mathcal{O}(n/\log_\sigma n)$-sized subset of the runs of $T$, after grouping these runs by their Lyndon roots (for small-period runs) or sparse Lyndon roots (for long-period runs). Counting special squares is significantly more challenging. We tackle this problem by processing certain families of runs. The runs corresponding to special squares with large period (larger than $\log_\sigma n$) are called here *layer-runs*. The crucial point is that these layer-runs can be grouped in $\mathcal{O}(n/\log_\sigma n)$ groups called "pyramids", though they can contain together $\Omega(n)$ layer-runs. These "pyramids" are very regular and counting squares in them can be done in batches in sublinear time. Let us provide a trivial yet illustrative example.

▶ **Example 3.** Consider string $S = (\mathtt{ab})^m(\mathtt{ba})^m$. For each $i \in [0\mathinner{.\,.} m)$, we have a square of the form $\mathtt{b}(\mathtt{ab})^i\mathtt{b}(\mathtt{ab})^i$ that occurs at (0-based) position $2m-1-2i$; this square is primitively rooted and, in fact, $S[2m-1-2i\mathinner{.\,.}2m+2i]$ is a run. These are the only primitively rooted squares in $S$ other than primitively rooted squares $\mathtt{abab}$ and $\mathtt{baba}$; see Figure 1.

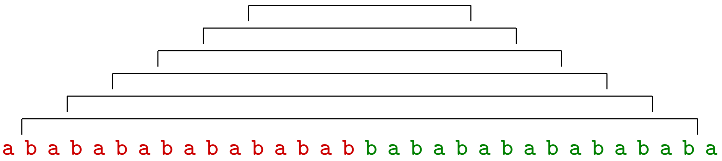
a b a b a b a b a b a b a b a b b a b a b a b a b a b a b a b a

**Figure 1** The pyramidal-shaped structure of six special squares contained in $(\mathtt{ab})^8(\mathtt{ba})^8$.

Using the approach of Crochemore et al. [19], we can count all plain squares in the string from Example 3 by processing a constant number of runs: $(\mathtt{ab})^m$, $(\mathtt{ba})^m$, $\mathtt{bb}$, and $(\mathtt{b}(\mathtt{ab})^i)^2$ for $i \in \{1,2,3\}$. However, there are $\Theta(n)$ runs of the form $\mathtt{b}(\mathtt{ab})^i\mathtt{b}(\mathtt{ab})^i$ for $i \geq 4$, each corresponding to a special square, and we clearly cannot afford to iterate over these runs in $o(n)$ time. All such special squares in our example have their first half in run $(\mathtt{ab})^m$ and their second half in run $(\mathtt{ba})^m$, and that these two runs have the same Lyndon root $(\mathtt{ab})$.

We employ an $\mathcal{O}(n/\log_\sigma n)$-sized representation of all runs that generate special squares. As presented intuitively in Example 3, the representation relies on $\mathcal{O}(n/\log_\sigma n)$ pairs $T[a\mathinner{.\,.}c]$ and $T[b\mathinner{.\,.}d]$ of runs that have the same Lyndon root $\lambda$ and satisfy $b \in (c-|\lambda|+1\mathinner{.\,.}c+1]$. Counting special squares from said representation is reduced to appropriately grouping the computed pairs of runs.

## 2 Preliminaries

For a string $S$, its positions are numbered as $S[0],\dots,S[|S|-1]$. If $|S|=0$, $S$ is called the empty string. A string consisting of characters $S[i],S[i+1],\dots,S[j]$ is a substring of $S$. A fragment of $S$ is a positioned substring; that is, a fragment $S[i\mathinner{.\,.}j]$ represents the substring $S[i],S[i+1],\dots,S[j]$. Where possible, we treat fragments and substrings as equivalent. For two fragments $F=S[a\mathinner{.\,.}b]$ and $F'=S[a'\mathinner{.\,.}b']$, we write $F \subseteq F'$ if $[a\mathinner{.\,.}b] \subseteq [a'\mathinner{.\,.}b']$. Two fragments $S[a\mathinner{.\,.}b]$ and $S[a'\mathinner{.\,.}b']$ are called *neighboring* if $[a-1\mathinner{.\,.}b+1]\cap[a'\mathinner{.\,.}b'] \neq \emptyset$. For two neighboring fragments $F=S[a\mathinner{.\,.}b]$ and $F'=S[a'\mathinner{.\,.}b']$, by $F\cup F'$ we denote the fragment $S[\min(a,a')\mathinner{.\,.}\max(b,b')]$ and by $F\cap F'$ we denote the fragment $S[\max(a,a')\mathinner{.\,.}\min(b,b')]$.

We say that a positive integer $p$ is a period of string $S$ if $p \leq |S|$ and $S[i] = S[i+p]$ for all $i \in [0\mathinner{.\,.}|S|-p)$; equivalently, $S[0\mathinner{.\,.}|S|-p) = S[p\mathinner{.\,.}|S|)$. Fine and Wilf's periodicity lemma [25] asserts that if a string of length $n$ has periods $p$ and $q$ such that $p+q \leq n$, then the string has period $\gcd(p,q)$. By $\mathsf{per}(S)$ we denote the smallest period of $S$ to which we refer as *the period* of $S$.

A non-empty string $S$ is called primitive if the equality $S = U^t$ for a positive integer $t$ implies that $t = 1$. By $\mathsf{rot}_c(S)$ we denote a cyclic rotation of the string $S$, obtained by moving the $c$ first characters of $S$ to its end. A string is called a *Lyndon string* if it is primitive and lexicographically minimal in the class of its cyclic rotations.

We say that a string $S$ is *periodic* if $\mathsf{per}(S) \leq \frac{1}{2}|S|$ and *highly periodic* if $\mathsf{per}(S) \leq \frac{1}{4}|S|$. The *Lyndon root* of a periodic string $S$, denoted by $\mathsf{Lroot}(S)$, is the lexicographically smallest rotation of $S[0\mathinner{.\,.}\mathsf{per}(S))$. For example, $\mathsf{Lroot}((\mathtt{abaaa})^3\,\mathtt{aba}) = \mathtt{aaaab}$.

▶ **Definition 4.** *The* Lyndon representation *of a periodic string $U$ is a quadruple $\mathsf{Lrepr}(U) = (\lambda, e, \alpha, \beta)$ such that:*

- *$\lambda = \mathsf{Lroot}(U)$, and*
- *$U = P\lambda^e S$ with $|P| = \alpha < |\lambda|$ and $|S| = \beta < |\lambda|$. ($P$ and/or $S$ can be the empty string.)*

▶ **Example 5.** We have $\mathsf{Lrepr}(U) = (\mathtt{aaaab}, 3, 2, 1)$ for

$$U = (\mathtt{abaaa})^3\,\mathtt{aba} = \mathtt{abaaa\,abaaa\,abaaa\,aba} = \mathtt{ab}\,(\mathtt{aaaab})^3\,\mathtt{a}.$$

A *run* in a string $T$ is a fragment $F = T[a\mathinner{.\,.}b]$ that is periodic, that is, $p := \mathsf{per}(F) \leq \frac{1}{2}|F|$, and inclusion-maximal, that is,

- $a = 0$ or $T[a-1] \neq T[a-1+p]$ and
- $b = |T|-1$ or $T[b+1] \neq T[b+1-p]$.

We denote the set of runs in $T$ by $\mathsf{Runs}(T)$. A string of length $n$ contains at most $n$ runs and they can be computed in $\mathcal{O}(n)$ time [5], even if the string is over an arbitrary ordered alphabet [23].

The *Lyndon position* of a run $R = T[a\mathinner{.\,.}b]$ with Lyndon root $\lambda$ is the unique position $i \in [a\mathinner{.\,.}a+\mathsf{per}(R))$ such that $T[i\mathinner{.\,.}i+\mathsf{per}(R)) = \lambda$.

A square is a string of the form $X^2 = XX$ for a non-empty string $X$. A square $X^2$ is called *primitively rooted* if $X$ is primitive and *non-primitively-rooted* otherwise.

We say that a square $X^2$ is *generated* by a periodic string $U$ if $X^2$ is contained in $U$ and $\mathsf{per}(X^2) = \mathsf{per}(U)$. By the periodicity lemma, in this case $|X|$ is a multiple of $\mathsf{per}(U)$. We denote by frag-squares$(U)$ the set of squares generated by a periodic string $U$.

▶ **Example 6.** For the underlined run $R = T[1\mathinner{.\,.}13]$ with period 2 in $T = \mathtt{c\underline{ababababababa}d}$, we have frag-squares$(R) = \{(\mathtt{ab})^2, (\mathtt{ba})^2, (\mathtt{abab})^2, (\mathtt{baba})^2, (\mathtt{ababab})^2\}$.

For a set $\mathcal{X}$ of periodic fragments of $T$ we denote

$$\text{frag-squares}(\mathcal{X}) = \bigcup_{U \in \mathcal{X}} \text{frag-squares}(U).$$

The following observation states that each square in a string is generated by a run.

▶ **Observation 7** ([19]). *$\mathsf{squares}(T)$ = frag-squares$(\mathsf{Runs}(T))$.*

Unfortunately, from the point of view of counting, the same square string can be generated by many runs. Dealing with squares whose first half is both primitive and (highly) periodic is the most challenging. The next section is devoted to runs that generate such squares.

The next fact follows by using radix sort (with bucket sort).

▶ **Fact 8.** *A list of $\mathcal{O}(n/\log_\sigma n)$ $k$-tuples of integers in $[0\mathinner{.\,.}n)$, for any constant $k > 0$, can be sorted lexicographically in a stable manner in $\mathcal{O}(n/\log_\sigma n)$ time.*

## 3 Pyramids of Runs

In this section we describe the structure of runs that generate special squares.

▶ **Definition 9** (Subperiodic strings). *For a periodic string $U$ we define*

$$\mathsf{subper}(U) \;=\; \min\{\, \mathsf{per}(X) \;:\; X^2 \in \text{frag-squares}(U)\,\}.$$

*A periodic string $U$ is called* subperiodic *if $\mathsf{subper}(U) \le \mathsf{per}(U)/4$.*

▶ **Example 10.** The string $U = \texttt{ab}\,(\texttt{babababab})^5\,\texttt{ba}$ with period 5 generates a subperiodic square $(\texttt{babababab})^2$ of length $2 \cdot \mathsf{per}(U)$. Thus $U$ is subperiodic, and $\mathsf{subper}(U) = 2$. Observe that all the remaining squares generated by $U$, in particular, $(\texttt{ababababb})^2$ and $(\texttt{babababab})^4$, are not subperiodic.

A special square is a square which is primitively rooted and subperiodic. All other squares are called plain.

For two neighboring runs $F, F'$ with equal period $p$ in $T$, we have $|F \cap F'| \le p - 1$ [36]. Two such runs can induce a collection of runs with subperiod $p$. We formalize this structure in the following definition and provide illustrations in Figures 2 and 3.

▶ **Definition 11.** *Let $F$ and $F'$ be neighboring runs in $T$ with period $p$ and equal Lyndon roots. A* pyramid $\mathbf{P}(F, F')$ *of runs is the set*

$$\{R \,:\, R \textit{ is a subperiodic run},\ \mathsf{subper}(R) = p,\ R \cap (F \cup F') \textit{ is periodic with period } \mathsf{per}(R)\}.$$

*If $R \in \mathbf{P}(F, F')$, run $R$ is called a* layer-run *(or a* layer *for brevity).*

▶ Remark 12. We show that all layers in $\mathbf{P}(F, F')$ are contained in $F \cup F'$ except possibly the longest layer (for example the red layer in Figure 3).

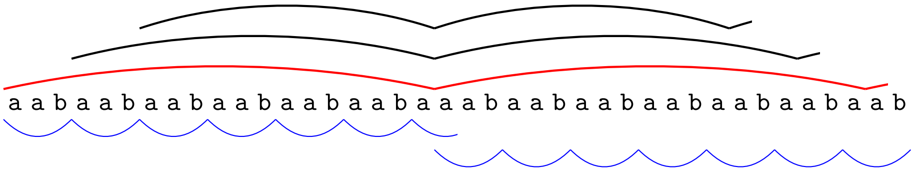
a a b a a b a a b a a b a a b a a b a a a b a a b a a b a a b a a b a a b a a b

**Figure 2** The runs $F$ and $F'$ with period 3 (at the bottom, in blue) imply a pyramid $\mathbf{P}(F, F')$ containing three layer-runs $(\texttt{aab})^i\texttt{a}\,(\texttt{aab})^i\texttt{aa}$, for $i \in \{4, 5, 6\}$ (above).

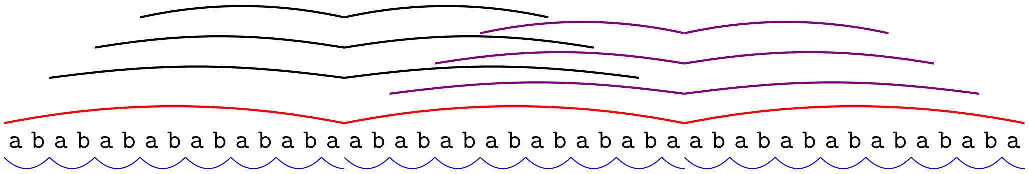
a b a b a b a b a b a b a b a a b a b a b a b a b a b a b a a b a b a b a b a b a b a b a

**Figure 3** The subsequent runs $F, F', F''$ with period 2 each corresponding to string $(\texttt{ab})^7\texttt{a}$ (at the bottom, blue) imply pyramids $\mathbf{P}(F, F')$ and $\mathbf{P}(F', F'')$ (above). The longest layer (in red) corresponding to string $((\texttt{ab})^7\texttt{a})^3$ is common to both pyramids.

▶ **Lemma 13.** *If $U^2 \in \text{frag-squares}(R \cap (F \cup F'))$ for some layer $R$ in pyramid $\mathbf{P}(F, F')$, then the first half of $U^2$ is contained in $F$ and the other half in $F'$.*

**Proof.** Let $p = \mathsf{per}(F) = \mathsf{per}(F')$. Layer $R$ is subperiodic with $\mathsf{subper}(R) = p$ and thus must have period at least $4p$, so $|U| \geq 4p$.

First we show that $U^2$ cannot be a fragment of $F$ (or of $F'$). Indeed, this would mean that $U^2$ has period $p$ as well as period $|U|$. By the periodicity lemma applied to $U^2$, $p$ would divide $|U|$. Consequently, $p$ would be a period of $R$, a contradiction.

Let us now show, by contradiction, that each half of $U^2$ is contained in $F$ or in $F'$. Suppose that this is not the case. One of the two halves is fully contained in one of $F$ and $F'$ and hence has period $p$, while the other half contains a position that is in $F$ but not in $F'$ and a position that is in $F'$ but not in $F$, and hence has period greater than $p$ due to the maximality of runs $F$ and $F'$. We have thus obtained a contradiction. ◀

Next we obtain a combinatorial characterization of all runs in a pyramid.

▶ **Definition 14.** *A layer with a maximal period in a pyramid is called a* max-layer*. We denote by $\mathbf{RegP}(F, F')$ the set of layer-runs in $\mathbf{P}(F, F')$ without the max-layer. The elements of $\mathbf{RegP}(F, F')$ are called* regular *layers.*

▶ **Example 15.** In Figure 2, there are two regular layers and one max-layer. In Figure 3, the first pyramid contains three regular layers while the second pyramid contains two; there is one max-layer that is common to all the pyramids.

Consider a pyramid $\mathbf{P}(F, F')$ and let $p = \mathsf{per}(F)$. A *canonical representation* of pyramid $\mathbf{P}(F, F')$ consists of the (endpoints of) runs $F$ and $F'$, its max-layer, and sequences specifying the starting positions, ending positions, and periods of its regular layers. In a canonical representation, the end positions of regular layers form an arithmetic progression with difference $p$, whereas the starting positions form an arithmetic progression with difference $-p$. Moreover, the periods of all regular layers form an arithmetic progression with difference $p$. The lemma below is proved in the following Lemmas 17 and 18.

▶ **Lemma 16.** *Any non-empty pyramid admits a canonical representation.*

We use the following notation for a fixed pyramid $\mathbf{P}(F, F')$. Let $F = T[a\mathinner{..}b]$, $F' = T[a'\mathinner{..}b']$, assuming without loss of generality that $a < a'$, and $p = \mathsf{per}(F) = \mathsf{per}(F')$. Let the Lyndon positions of $F$ and $F'$ be $\ell$ and $\ell'$, respectively, and define $\delta := (\ell' - \ell) \bmod p$. For each $k \in \mathbb{Z}$, we denote $a'_k := a' - k \cdot p - \delta$ and $b_k = b + k \cdot p + \delta$.

▶ **Lemma 17.** *The set $\mathcal{R} := \{T[x\mathinner{..}y] \in \mathbf{P}(F, F') : x, y \in (a\mathinner{..}b')\}$ is equal to*

$$\mathcal{K} := \{T[a'_k\mathinner{..}b_k] : k \in K\}, \text{ where } K = \{k \in \mathbb{Z} \;:\; k \geq 4,\; a'_k > a,\; b_k < b'\}.$$

*For each $k \in K$, the period of run $T[a'_k\mathinner{..}b_k]$ is $k \cdot p + \delta$.*

**Proof.** First, let us argue that $\mathsf{per}(T[a'_k\mathinner{..}b_k]) = k \cdot p + \delta$ for each $k \geq 2$. We note that $T[a'_k\mathinner{..}b] = T[a'\mathinner{..}b_k]$ (by the definition of $\delta$ and the fact that the two strings are contained in $F$ and $F'$, respectively), so $a'_k - a'$ is a period of $T[a'_k\mathinner{..}b_k]$ by definition, and $a'_k - a' = k \cdot p + \delta$. Observe that $T[a'_k\mathinner{..}b]$ has period $p$ and hence it cannot have an occurrence starting before position $a'$ and ending after position $b$ – as all fragments satisfying these conditions do not have period $p$ by the maximality of $F$ and $F'$. Thus, $T[a'_k\mathinner{..}b_k]$ does not have any period smaller than $k \cdot p + \delta$.

**$\mathcal{K} \subseteq \mathcal{R}$:** Let $R = T[a'_k \mathinner{.\,.} b_k]$. We have

$$|R| = |R \cap (F \cup F')| = b_k - a'_k + 1 = b - a' + 1 + 2kp + 2\delta \geq 2kp + 2\delta = 2 \cdot \mathsf{per}(R)$$

as $F$ and $F'$ are neighboring. In particular, $R$ is periodic. Let us show that $R$ is a maximal fragment with period $\mathsf{per}(R)$. As $a'_k > a$, the periodicities of $F$ and $R$ and left maximality of $F'$ imply that

$$T[a'_k - 1] = T[a'_k - 1 + p] = T[a'_k - 1 + p + \mathsf{per}(R)] \neq T[a'_k - 1 + \mathsf{per}(R)] = T[a' - 1],$$

which shows the left maximality of $R$. A symmetric argument yields right maximality. Hence, $R$ is indeed a run.

The prefix $T[a'_k \mathinner{.\,.} b]$ of $R$ has length $|R| - \mathsf{per}(R) \geq \mathsf{per}(R) = kp + \delta \geq 2p$ and period $p$, so $R$ is subperiodic and $\mathsf{subper}(R) \leq p$. Moreover, $\mathsf{subper}(R)$ cannot be smaller than $p$, as then there would be a run with period smaller than $p$ overlapping one of runs $F, F'$ on at least $kp + \delta \geq 4p$ positions, which is impossible due to the periodicity lemma.

**$\mathcal{R} \subseteq \mathcal{K}$:** Let us fix some $R = T[\alpha \mathinner{.\,.} \beta] \in \mathcal{R}$. As $R$ is subperiodic, $\mathsf{per}(R) \geq 4p$, and by the definition of $\mathcal{R}$, $\alpha > a$ and $\beta < b'$. Consider a square $T[x \mathinner{.\,.} x + 2 \cdot \mathsf{per}(R)) \in \text{frag-squares}(R)$. By Lemma 13, $T[x \mathinner{.\,.} x + \mathsf{per}(R)) \subseteq F$ and $T[x + \mathsf{per}(R) \mathinner{.\,.} x + 2 \cdot \mathsf{per}(R)) \subseteq F'$. Since primitive strings do not match non-trivial rotations of themselves, we have that $T[x \mathinner{.\,.} x + p)$ occurs only at positions $y$ of $T$ contained in $F'$ such that $x - \ell \equiv y - \ell' \pmod{p}$. This implies that $\mathsf{per}(R)$ has to be equivalent to $y - x \equiv \ell' - \ell \equiv \delta \pmod{p}$. Then, for some $k \in \mathbb{Z}$, $\mathsf{per}(R) = k \cdot p + \delta$. By the definition of $\mathcal{R}$, $k \in K$. Finally, there can be at most one run with period $\mathsf{per}(R)$ that contains at least $\mathsf{per}(R)$ positions of $F$ and $F'$ and we have shown the existence of such a run in $\mathcal{K}$, so $R \in \mathcal{K}$. ◀

▶ **Lemma 18.** *For the set $\mathcal{R}$ defined in Lemma 17, there is at most one run $R \in \mathbf{P}(F, F') \setminus \mathcal{R}$.*

**Proof.** Consider the case when there exists a run $R \in \mathbf{P}(F, F')$ containing position $a$. We then know that $R \cap (F \cup F')$ has subperiod $p$ and is periodic with period $\mathsf{per}(R)$. Consider the square $T[a \mathinner{.\,.} a + 2 \cdot \mathsf{per}(R)) \in \text{frag-squares}(R \cap (F \cup F'))$. By Lemma 13, we know that the first half of this square is contained in $F$, while the second half is contained in $F'$. Hence, the square has subperiod $p$. This means that $\mathsf{per}(R) \in [a' - a \mathinner{.\,.} b - a]$, which is an interval of length at most $p - 1$. Now, $\mathsf{per}(R)$ also has to be equivalent to $\delta \pmod{p}$, so there is a single possible value for it, say $y$.

Observe that if $b'$ is not contained in $R$, then $F$ is a common prefix of $T[a \mathinner{.\,.} |T|)$ and $T[a + y \mathinner{.\,.} |T|)$, which means that $a' + |F| - 1 \leq a + y + |F| - 1 < b'$ and hence $|F'| > |F|$. Now, a run $R' \in \mathbf{P}(F, F')$ containing position $b'$ would have period in $[b' - b \mathinner{.\,.} b' - a']$ by symmetric arguments to those above. Then, we would have $\mathsf{per}(R') \geq b' - b > |F|$, a contradiction to the fact that a square with period $\mathsf{per}(R')$ generated by $R'$ would have its first half in $F$.

Finally, if our attempt to compute a run containing position $a$ fails, we perform symmetric computations to find a run $R \in \mathbf{P}(F, F')$ that contains position $b'$, if one exists. ◀

## 4 Computing a Representation of All Runs

In this section we show that a representation of all runs in a string of length $n$ over an alphabet $[0 \mathinner{.\,.} \sigma)$ can be computed in $\mathcal{O}(n / \log_\sigma n)$ time.

First we show how to compute runs with large periods. Some of these runs are grouped in pyramids.

Amir et al. [1] showed how to compute squares and runs in a dynamic string. Their techniques can be interpreted in the so-called PILLAR model, introduced by Charalampopoulos, Kociumaka, and Wellnitz [15]. Recent optimal data structures for LCP queries [31] and IPM queries [35] in the packed setting imply that any problem on strings of total length $n$ that can be solved in $\mathcal{O}(f(n))$ time in the `PILLAR` model, can be solved in $\mathcal{O}(n/\log_\sigma n + f(n))$ time in the packed setting. All in all, we obtain the following fact whose proof closely follows [1, 12]; for completeness, it is provided in the full version.

▶ **Fact 19** (see [1, 12]). *Let $T \in [0\,..\,\sigma)^n$ be a string given in packed form. For any constant $c > 0$, in time $\mathcal{O}(n/\log_\sigma n)$, we can compute a set $\mathcal{X}$ of runs such that none of them is a regular layer of any pyramid of $T$ and a set $\mathcal{Y}$ of pyramids given by their canonical representations, such that $|\mathcal{X}|, |\mathcal{Y}| = \mathcal{O}(n/\log_\sigma n)$, and, for $\mathcal{Z} := \bigcup_{(F,F')\in\mathcal{Y}} \mathbf{RegP}(F, F')$, we have that*

- *$\mathcal{X} \cup \mathcal{Z}$ is a superset of all runs in $T$ of period at least $c\log_\sigma n$, and*
- *$\mathcal{X} \cap \mathcal{Z} = \emptyset$.*

We do not include max-layers in set $\mathcal{Z}$ as they can be common to many pyramids.

▶ Remark 20. As shown in the proof of Fact 19 (given in the full version) and implicitly in [1, 12], for any parameter $q$, the number of both max-layers with period at least $q$ and non-layer-runs with period at least $q$ in a length-$n$ string is $\mathcal{O}(n/q)$. We note that the number of *all* layer runs with period at least $q$ can be $\Omega(n)$: for any $q \geq 3$, the string $S$ from Example 3 has at least $(n/2 - q - 1)/2$ layer runs with period at least $q$.

▶ **Definition 21** (Clusters of runs). *For a set of runs $\mathcal{X}$ in $T$ and a set of integers $D$, we define a* cluster of runs*:*

$$\mathbf{Cluster}(\mathcal{X}, D) = \{T[a+d\,..\,b+d] \,:\, T[a\,..\,b] \in \mathcal{X},\, d \in D,\, T[a\,..\,b] = T[a+d\,..\,b+d]\}.$$

*The* size *of a cluster of runs is defined as $|\mathcal{X}| + |D|$.*

In this work, in all considered clusters of runs, we have $0 \in D$.

▶ **Example 22.** The string $S = \#\texttt{ababaabaab}\$$ contains runs $S[1\,..\,5] = \texttt{ababa}$, $S[5\,..\,6] = S[8\,..\,9] = \texttt{aa}$, $S[3\,..\,10] = \texttt{abaabaab}$. Thus the following string

$$T = \#\texttt{ababaabaab}\$\,\#\texttt{ababaabaab}\$\,\#\texttt{edcbaedcba}\$\,\#\texttt{ababaabaab}\$$$

contains a cluster of runs $\mathbf{Cluster}(\,\{T[1\,..\,5], T[5\,..\,6], T[8\,..\,9], T[3\,..\,10]\},\ \{0, 12, 36\}\,)$.

A *$\tau$-run* $R$ is a run of length at least $3\tau - 1$ with period at most $\frac{1}{3}\tau$.

▶ **Lemma 23** ([31, Section 6.1.2],[13, Lemma 10]). *For a positive integer $\tau$, a string $T \in [0\,..\,\sigma)^n$ contains $\mathcal{O}(n/\tau)$ $\tau$-runs. Moreover, if $\tau \leq \frac{1}{9}\log_\sigma n$, given a packed representation of $T$, we can compute all $\tau$-runs in $T$ in $\mathcal{O}(n/\tau)$ time. Within the same complexity, we can compute the Lyndon position of each $\tau$-run.*

The next lemma can be proved using tabulation; a proof can be found in the full version.

▶ **Lemma 24.** *Given a string $T$ in packed form and an integer $\tau \leq \frac{1}{9}\log_\sigma n$, we can compute all runs of length smaller than $3\tau - 1$ and period at most $\frac{1}{3}\tau$, represented as $\mathcal{O}(n/\log_\sigma n)$ clusters of runs, in $\mathcal{O}(n/\log_\sigma n)$ time. The sum of lengths of lists $\mathcal{X}$ across all clusters of runs is $\tilde{\mathcal{O}}(n^{7/18})$.*

Putting everything together, we obtain the following proposition. We may need to trim arithmetic progressions of regular layers to avoid double reporting runs with small periods.

▶ **Proposition 25.** *A representation of all runs in a string $T \in [0\,..\,\sigma)^n$ consisting of a disjoint union of $\mathcal{O}(n/\log_\sigma n)$ runs, regular layers of pyramids, and clusters of runs, can be computed in $\mathcal{O}(n/\log_\sigma n)$ time.*

## 5 Grouping Runs via Lyndon Roots and Sparse-Lyndon Roots

The next observation is crucial in counting distinct square substrings of a string. Let us denote by $\mathsf{Runs}(T,\lambda)$ and $\mathsf{squares}(T,\lambda)$ the sets of runs and squares in $T$ with Lyndon root $\lambda$.

▶ **Observation 26** ([19]). *Consider two runs $R$ and $R'$ in a string $T$. Then, $\text{frag-squares}(R) \cap \text{frag-squares}(R') \neq \emptyset$ implies that $\mathsf{Lroot}(R) = \mathsf{Lroot}(R')$. In particular, for any Lyndon string $\lambda$, we have*

$$\mathsf{squares}(T,\lambda) = \bigcup_{R \in \mathsf{Runs}(T,\lambda)} \text{frag-squares}(R).$$

Crochemore et al. [19] considered all runs in the string in groups consisting of runs with equal Lyndon root. The algorithm for grouping of runs that they used consists of the following three steps:

1. Computing a Lyndon position for each run.
2. Sorting runs with equal periods in the order of the suffixes starting at Lyndon positions. It is guaranteed that runs from the same group are listed consecutively.
3. Partitioning the sorted list of runs obtained for each period into groups by issuing an LCP-query for each pair of subsequent runs in the list.

We use Proposition 25 to compute an $\mathcal{O}(n/\log_\sigma n)$-sized representation of all the runs in $T$. For runs with small periods, we use the aforementioned approach combined with tabulation (for runs that are not $\tau$-runs) and the following fact for $\tau$-runs.

▶ **Fact 27** ([31, Section 6.1.2],[13, Lemma 10]). *If $\tau \le \frac{1}{9}\log_\sigma n$, given a packed representation of $T$, all $\tau$-runs in $T$ can be sorted by their Lyndon roots in $\mathcal{O}(n/\tau)$ time.*

A proof of Lemma 28 is given in the full version.

▶ **Lemma 28.** *All runs in $T$ with periods at most $\tau$, for a given $\tau \le \frac{1}{9}\log_\sigma n$, can be grouped by equal Lyndon roots in $\mathcal{O}(n/\log_\sigma n)$ time. Among possibly many runs corresponding to equal substrings, at least one needs to be reported, but not necessarily all.*

One issue with adapting the aforementioned approach to grouping runs with large periods is that we do not know how to compute the Lyndon positions of $\mathcal{O}(n/\log_\sigma n)$ runs in $T$ if their period is greater than $c\log_\sigma n$ for a constant $c$, in $\mathcal{O}(n/\log_\sigma n)$ time.

Using cyclic equivalence queries of Kociumaka et al. [35] that allow to check if two substrings of $T$ are cyclic rotations of each other, we can check if two runs have the same Lyndon root in $\mathcal{O}(1)$ time after $\mathcal{O}(n/\log_\sigma n)$-time preprocessing, but this is not sufficient for grouping the runs by Lyndon roots. Moreover, it is unknown whether minimal cyclic rotation queries can be implemented efficiently in the packed model. The fastest known solution, by Kociumaka [34], answers minimal cyclic rotation queries in $\mathcal{O}(1)$ time but requires $\Theta(n)$ preprocessing; improving the preprocessing time to sublinear here seems to be challenging.

Instead, we use a string synchronizing set, as defined by Kempa and Kociumaka [31], to select a unique position within each long-period run (that might not be the Lyndon position) in a consistent way that allows us to group such runs by Lyndon roots.

▶ **Definition 29** (Synchronizing set [31]). *For a length-$n$ string $T$ and a positive integer $\tau \le \frac{1}{2}n$, a set $\mathbf{Sync} \subseteq [0\mathinner{.\,.} n-2\tau]$ is a* $\tau$-synchronizing set *of $T$ if it satisfies the following two conditions:*

1. *Consistency: If $T[i\mathinner{.\,.} i+2\tau) = T[j\mathinner{.\,.} j+2\tau)$, then $i \in \mathbf{Sync}$ if and only if $j \in \mathbf{Sync}$.*
2. *Density: For $i \in [0\mathinner{.\,.} n-3\tau+1]$,*
   *$\mathbf{Sync} \cap [i\mathinner{.\,.} i+\tau) = \emptyset$ if and only if $\mathsf{per}(T[i\mathinner{.\,.} i+3\tau-2]) \le \frac{1}{3}\tau$.*

▶ Remark 30. Informally, in the simpler case that $T$ is cube-free, a $\tau$-synchronizing set of $T$ is an $\mathcal{O}(n/\tau)$-sized set of synchronizing positions in $T$ such that each length-$\tau$ fragment of $T$ (except for the end of the string) contains at least one synchronizing position, and the leftmost synchronizing positions within two length-$3\tau$ matching fragments of $T$ are consistent.

Crucially, string synchronizing sets for small values of $\tau$ can be constructed in optimal time in the packed setting.

▶ **Theorem 31** ([31, Proposition 8.10, Theorem 8.11]). *For a string $T \in [0\mathinner{.\,.}\sigma)^n$ with $\sigma = n^{\mathcal{O}(1)}$ and $\tau \le \frac{1}{5}\log_\sigma n$, there exists a $\tau$-synchronizing set of size $\mathcal{O}(n/\tau)$ that can be constructed in $\mathcal{O}(n/\tau)$ time, if $T$ is given in a packed representation.*

Henceforth, we fix $\tau := \lfloor \frac{1}{18}\log_\sigma n \rfloor$ and a $\tau$-synchronizing set $\mathbf{Sync}$ for $T$ computed in $\mathcal{O}(n/\log_\sigma n)$ time using Theorem 31. We next define *sparse-Lyndon positions*, noting that their existence is only guaranteed for runs whose periods are long enough, and use them to group runs by Lyndon roots.

▶ **Definition 32** (Sparse-Lyndon position). *Position $i$ is a* sparse-Lyndon position *for a periodic fragment $U = T[a\mathinner{.\,.}b]$ with period $p$ if $T[i\mathinner{.\,.}n)$ is the lexicographically minimal string among $\{T[j\mathinner{.\,.}n) : j \in [a\mathinner{.\,.}a+p) \cap \mathbf{Sync}\}$.*

*If a periodic fragment $U$ with period $p$ has a sparse-Lyndon position $i$, we call $T[i\mathinner{.\,.}i+p)$ the* sparse-Lyndon root *of $U$ and denote it as $\mathsf{sLroot}(U)$.*

The following key lemma shows that Lyndon positions can indeed be replaced by sparse-Lyndon positions for the sake of grouping runs by Lyndon roots.

▶ **Lemma 33.** *Both of the following hold.*

**(a)** *If a run $R$ has period $p \ge 2\tau - 1$, then $R$ has a unique sparse-Lyndon root.*

**(b)** *Two runs $R_1$ and $R_2$ with period $p \ge 2\tau$ have the same Lyndon root if and only if they have the same sparse-Lyndon root.*

**Proof.** (a) Let $R = T[a\mathinner{.\,.}b]$. We will first show that $[a\mathinner{.\,.}a+p) \cap \mathbf{Sync}$ is non-empty. String $T[a\mathinner{.\,.}a+p+\tau)$ has period $p$ as $p \ge \tau$. By the periodicity lemma [25], if $T[a\mathinner{.\,.}a+p+\tau)$ had a period at most $\frac{1}{3}\tau$, then the string would have a period $p'$ that is smaller than $p$ and divides $p$, which is not possible as this would imply that $R$ has a period $p'$. We have $p+\tau \ge 3\tau - 1$. String $T[a\mathinner{.\,.}a+p+\tau)$ contains a fragment of length $3\tau - 1$ with period greater than $\frac{1}{3}\tau$; indeed, otherwise the periods of all such fragments would be equal by the periodicity lemma and this would imply that $T[a\mathinner{.\,.}a+p+\tau)$ has a period at most $\frac{1}{3}\tau$. Let $T[i\mathinner{.\,.}i+3\tau-2]$ be such a fragment with period greater than $\frac{1}{3}\tau$. By density, $\mathbf{Sync} \cap [i\mathinner{.\,.}i+\tau) \ne \emptyset$. We have $a \le i$ and $i + 3\tau - 2 < a + p + \tau$, so $[i\mathinner{.\,.}i+\tau) \subseteq [a\mathinner{.\,.}a+p)$. Hence, indeed, $[a\mathinner{.\,.}a+p) \cap \mathbf{Sync} \ne \emptyset$.

This shows the existence of a sparse-Lyndon root of $R$. As no two distinct suffixes are equal, the sparse-Lyndon root of $R$ is unique.

(b) By part (a), the sparse-Lyndon roots of both runs are well-defined.

The implication "$\Leftarrow$" is obvious. As for the implication "$\Rightarrow$", let $R_1 = T[a\mathinner{.\,.}b]$ and $R_2 = T[a'\mathinner{.\,.}b']$ and assume that $i$ is the sparse-Lyndon position of $R_1$. By the assumption, there exists $i' \in [a'\mathinner{.\,.}a'+p)$ such that $T[i'\mathinner{.\,.}i'+p) = T[i\mathinner{.\,.}i+p)$. By the fact that $p \ge 2\tau$, we have $T[i'\mathinner{.\,.}i'+2\tau) = T[i\mathinner{.\,.}i+2\tau)$, so by consistency, $i' \in \mathbf{Sync}$.

To prove that $i'$ is the sparse-Lyndon position of $R_2$, assume to the contrary that there exists a position $j' \in ([a'\mathinner{.\,.}a'+p) \cap \mathbf{Sync}) \setminus \{i'\}$ such that $T[j'\mathinner{.\,.}j'+n) < T[i'\mathinner{.\,.}i'+n)$. Consequently, $T[j'\mathinner{.\,.}j'+p) < T[i'\mathinner{.\,.}i'+p)$ as no two cyclic rotations of a primitive string are equal. By the assumption, there exists $j \in [a\mathinner{.\,.}a+p)$ such that $T[j\mathinner{.\,.}j+p) = T[j'\mathinner{.\,.}j'+p)$. By the fact that $p \ge 2\tau$ and consistency, $j \in \mathbf{Sync}$. We have

$$T[j\mathinner{.\,.}j+p) = T[j'\mathinner{.\,.}j'+p) < T[i'\mathinner{.\,.}i'+p) = T[i\mathinner{.\,.}i+p), \text{ so } T[j\mathinner{.\,.}j+n) < T[i\mathinner{.\,.}i+n),$$

which contradicts the assumption that $T[i\mathinner{.\,.}i+p)$ is the sparse-Lyndon root of $R_1$. ◀

▶ **Definition 34.** *The* sparse-Lyndon representation *of a periodic fragment $U$ of $T$ is a quadruple $(\lambda, e, \alpha, \beta)$ such that:*

- $\lambda = \mathsf{sLroot}(U)$, *and*
- $U = P\lambda^e S$ *with* $|P| = \alpha < |\lambda|$ *and* $|S| = \beta < |\lambda|$.

We use the next lemma to obtain the main result of this section.

▶ **Lemma 35** ([31, Theorem 4.3])**.** *Given the packed representation of a text $T \in [0\,..\,\sigma)^n$ and a $t$-synchronizing set $\mathcal{S}$ of $T$ of size $\mathcal{O}(n/t)$ for $t = \mathcal{O}(\log_\sigma n)$, we can compute in $\mathcal{O}(n/t)$ time the lexicographic order of all suffixes of $T$ starting at positions in $\mathcal{S}$.*

▶ **Proposition 36.** *All runs in $T$ computed as in Proposition 25, except for the regular layers of pyramids, can be grouped by equal Lyndon roots in $\mathcal{O}(n/\log_\sigma n)$ time. For runs with period at most $2\left\lfloor \frac{1}{18}\log_\sigma n \right\rfloor$, we compute their Lyndon representations, and for the remaining runs, we compute their sparse-Lyndon representations.*

**Proof.** Recall that $\tau = \left\lfloor \frac{1}{18}\log_\sigma n \right\rfloor$. Runs with periods at most $2\tau$ are grouped by their Lyndon roots using Lemma 28. The remaining runs are grouped by their sparse-Lyndon roots, and thus by Lyndon roots due to Lemma 33, using Lemma 35 as follows.

Let $\mathbf{Sync} = \{s_1, \dots, s_{|\mathbf{Sync}|}\}$, with $s_1 < \dots < s_{|\mathbf{Sync}|}$, be a $\tau$-synchronizing set of $T$ constructed as in Theorem 31. By Lemma 35, in $\mathcal{O}(n/\log_\sigma n)$ time we can construct an array $\mathsf{SparseRANK}[1\,..\,|\mathbf{Sync}|]$ ("sparse RANK" array) such that

$$\mathsf{SparseRANK}[i] = |\{j \in [1\,..\,|\mathbf{Sync}|] \,:\, T[s_j\,..\,n) \leq T[s_i\,..\,n)\}|.$$

Then, in $\mathcal{O}(|\mathsf{SparseRANK}|)$ time, we construct a data structure that can answer range minimum queries over $\mathsf{SparseRANK}$ in $\mathcal{O}(1)$ time [9].

Let $s_0 = -1$ and $s_{|\mathbf{Sync}|+1} = n$ be sentinels. Let $\Pi$ denote the set of all runs with period $p \geq 2\tau$ that are not regular layers of any pyramid. By Fact 19, set $\Pi$ can be computed in $\mathcal{O}(n/\log_\sigma n)$ time. For each run $T[a\,..\,b] \in \Pi$, we need to compute an interval $[u\,..\,v]$ such that $s_{u-1} < a \leq s_u$ and $s_v < a + p \leq s_{v+1}$. By Lemma 33(b), this interval is not empty and hence $u \leq v$. The sparse-Lyndon position of each such run can then be computed in $\mathcal{O}(1)$ time as the argmin of a range minimum query over $\mathsf{SparseRANK}[u\,..\,v]$. The positions $u$ and $v$ are computed for all runs simultaneously in $\mathcal{O}(n/\log_\sigma n)$ time by bucket sorting the set $\{x : x = a \text{ or } x = a + p - 1 \text{ for a run } T[a\,..\,b] \in \Pi\}$ and merging the obtained sorted list with the synchronizing set $\mathbf{Sync}$ in a merge-sort fashion.

The remainder of the algorithm mimics steps 2 and 3; see the discussion after Observation 26. Namely, in $\mathcal{O}(n/\log_\sigma n)$ time, we bucket sort the runs with large periods by pairs $(p, \mathsf{SparseRANK}[i])$, where $i$ is the sparse-Lyndon position and $p$ is the period of the run. Runs with equal sparse-Lyndon roots form consecutive sublists of the sorted lists. The equality of sparse-Lyndon roots of consecutive runs in the sorted list can be checked in $\mathcal{O}(1)$ time using longest common extension queries after an $\mathcal{O}(n/\log_\sigma n)$-time preprocessing [31, Theorem 5.4]. Thus, the grouping is performed in $\mathcal{O}(n/\log_\sigma n)$ time. ◀

## 6 Squares Generated by Pyramids

We show that a special square (that is, a square with a primitive and highly periodic half) is always generated by a layer of a pyramid. The proof of the lemma uses the assumption of at least 4 occurrences of the period in a special square half.

▶ **Lemma 37.** *Let $U^2$ be a fragment of $T$. Then $U^2$ is a special square if and only if there exists a pyramid $\mathbf{P}(F, F')$ in $T$ and a layer $R$ such that $U^2 \in \text{frag-squares}(R \cap (F \cup F'))$ and $\mathsf{per}(U) = \mathsf{per}(F)$.*

**Proof.** ($\Rightarrow$) Let $U^2$ be a special square fragment of $T$ and $p = \mathsf{per}(U)$. Let $F$ and $F'$ be runs with period $p = \mathsf{per}(U)$ that contain the first and the second half of the considered occurrence of $U^2$ in $T$, respectively. We have $F \neq F'$, as otherwise $U^2$ would have period $p$ and, by the periodicity lemma, $U$ would not be primitive.

By Observation 7, there exists a run $R$ in $T$ such that $U^2 \in \text{frag-squares}(R)$. By definition, we have $4p < |U| = \mathsf{per}(R)$. Moreover, $R$ is a subperiodic run with $\mathsf{per}(R) = |U|$ and $\mathsf{subper}(R) \leq p$. If we had $\mathsf{subper}(R) = p' < p$, then there would exist a run $G$ in $T$ with period $p'$ that overlaps $F$ or $F'$ – say, $F$ – on at least $|U|/2$ positions. The overlap length would be greater than $p + p'$, so by the periodicity lemma, the overlap would have period $q := \gcd(p, p') < p$ that divides $p$, so $F$ would have period $q$; a contradiction.

Clearly, the runs $F, F'$ are neighboring. We have $R \in \mathbf{P}(F, F')$ or $R$ is a max-layer of some pyramid $\mathbf{P}(F'', F''')$ with $\mathsf{subper}(R) < p$. In either case, $U^2 \subseteq F \cup F'$ and $U^2 \in \text{frag-squares}(R \cap (F \cup F'))$, as required.

($\Leftarrow$) Let $R$ be a layer of some pyramid with

$$\mathsf{per}(F) = \mathsf{per}(F') = \mathsf{per}(U) = p \text{ and } U^2 \in \text{frag-squares}(R \cap (F \cup F')).$$

We have $|U| \geq \mathsf{per}(R) > 4p$ since $R$ is subperiodic. By Lemma 13, one half of $U^2$ is contained in $F$ and the other in $F'$.

Period $p$ does not divide $|U|$ as otherwise we would have $F = F'$. Moreover, by the periodicity lemma, $U$ does not have a period $q$ that would divide $U$. Thus, $U$ is primitive and highly periodic, which means that $U^2$ is a special square. ◀

▶ **Definition 38** (Pyramid type). *Let $F, F'$ be neighboring runs with period $p$ in $T$. We define the* type *of the pyramid $\mathbf{P}(F, F')$ as a triad $\mathsf{type}(F, F') = (ov, X, Y)$ where (see Figure 4):*

$$ov = |F \cap F'|, \quad X = F[|F| - p \mathinner{.\,.} |F|), \quad Y = F'[0 \mathinner{.\,.} p).$$

▶ Remark 39. The strings $X$ and $Y$ are cyclically equivalent if $\mathbf{P}(F, F')$ is non-empty.

▶ **Example 40.** Let $T = T[0 \mathinner{.\,.} 60] = (\texttt{aaaab})^5\texttt{a}(\texttt{aaaab})^7$, $F = (\texttt{aaaab})^5\texttt{aaaa} = T[0 \mathinner{.\,.} 28]$ and $F' = (\texttt{aaaab})^7 = T[26 \mathinner{.\,.} 60]$. Then $\mathsf{type}(F, F') = (3, \texttt{baaaa}, \texttt{aaaab})$.

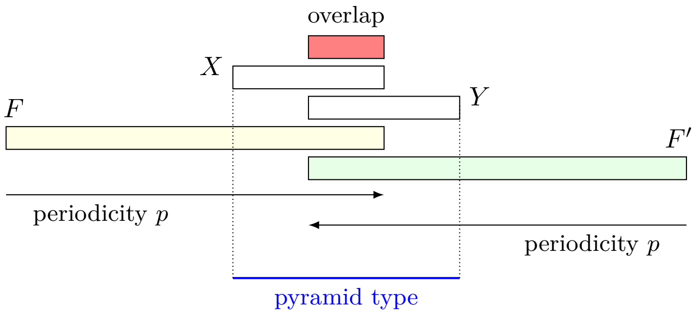

**Figure 4** Illustration of $\mathsf{type}(F, F') = (ov, X, Y)$, for two runs $F, F'$ with the same period $p$. We have $|X| = |Y| = p$.

We extend the notation frag-squares to pyramids as follows.

▶ **Definition 41** (Special squares generated by pyramids).

$$\text{frag-squares}(\mathbf{P}(F, F')) := \bigcup_{R \in \mathbf{P}(F, F')} \text{frag-squares}(R \cap (F \cup F')).$$

*We say that the elements of* frag-squares$(\mathbf{P}(F, F'))$ *are* generated *by the pyramid* $\mathbf{P}(F, F')$.

▶ **Lemma 42.** *The sets of squares generated by two pyramids of different types are disjoint.*

**Proof.** Let $F_1, F'_1$ and $F_2, F'_2$ be pairs of neighboring runs with equal periods such that $\mathsf{type}(F_1, F'_1) \neq \mathsf{type}(F_2, F'_2)$. We will show that the sets $\text{frag-squares}(\mathbf{P}(F_1, F'_1))$ and $\text{frag-squares}(\mathbf{P}(F_2, F'_2))$ are disjoint.

Assume there exists $U^2 \in \text{frag-squares}(\mathbf{P}(F_1, F'_1)) \cap \text{frag-squares}(\mathbf{P}(F_2, F'_2))$. We have

$$U^2 \in \text{frag-squares}(R_1 \cap (F_1 \cup F'_1)) \cap \text{frag-squares}(R_2 \cap (F_2 \cup F'_2)),$$

for some runs $R_1 \in \mathbf{P}(F_1, F'_1)$ and $R_2 \in \mathbf{P}(F_2, F'_2)$. Let $\mathsf{type}(F_1, F'_1) = (ov_1, X_1, Y_1)$ and $\mathsf{type}(F_2, F'_2) = (ov_2, X_2, Y_2)$. By Lemma 37, $U^2$ is a special square with $\mathsf{per}(U) = \mathsf{per}(F_1) = \mathsf{per}(F'_1) = \mathsf{per}(F_2) = \mathsf{per}(F'_2)$. Let $p = \mathsf{per}(U)$. Square $U^2$ does not have period $p$ (as $U$ is primitive). Hence, we can define

- $i$ as the smallest position in $U^2$ such that $\mathsf{per}(U^2[0\,..\,i]) > p$;
- $j$ as the largest position in $U^2$ such that $\mathsf{per}(U^2[j\,..\,|U^2|)) > p$.

We have $j < |U| \leq i$. Then, we have

$$X_1 = U^2[i-p\,..\,i-1] = X_2,\ Y_1 = U^2[j+1\,..\,j+p] = Y_2,\ \text{and } ov_1 = i-j-1 = ov_2,$$

so $\mathsf{type}(F_1, F'_1) = \mathsf{type}(F_2, F'_2)$. This contradiction concludes the proof. ◀

By $\text{frag-squares}(\mathbf{RegP}(F, F'))$ we denote the set of (special) squares generated by regular layers of $\mathbf{RegP}(F, F')$.

▶ **Lemma 43.** *Both of the following hold:*

**(a)** *If $\mathbf{P}(F, F')$ is a pyramid, then*

$$|\text{frag-squares}(\mathbf{RegP}(F, F'))| = |\mathbf{RegP}(F, F')| \cdot (|F \cap F'| + 1).$$

**(b)** *If $\mathbf{P}(F_1, F'_1)$ and $\mathbf{P}(F_2, F'_2)$ are pyramids such that $\mathsf{type}(F_1, F'_1) = \mathsf{type}(F_2, F'_2)$, then*

$$|\mathbf{RegP}(F_1, F'_1)| < |\mathbf{RegP}(F_2, F'_2)| \ \Rightarrow\ \text{frag-squares}(\mathbf{P}(F_1, F'_1)) \subset \text{frag-squares}(\mathbf{P}(F_2, F'_2)).$$

**Proof.** Let $F = T[a\,..\,b]$, $F' = T[a'\,..\,b']$ be neighboring runs with period $p$ and $a < a'$. Due to Lemma 18, the runs in $\mathbf{RegP}(F, F')$ are all layers in the set $\mathcal{R} := \{T[x\,..\,y] \in \mathbf{P}(F, F') : x, y \in (a\,..\,b')\}$ defined in Lemma 17, apart, possibly, from the one with the largest period.

**Proof of (a).** Each run $R \in \mathbf{RegP}(F, F')$ generates $|R| - 2 \cdot \mathsf{per}(R) + 1$ squares. By Lemma 17, for run $R = T[a'_k\,..\,b_k]$, this number of squares equals

$$b_k - a'_k + 2 - 2 \cdot \mathsf{per}(R) = b - a' + 2 + 2kp + 2\delta - 2 \cdot \mathsf{per}(R) = b - a' + 2 = |F \cap F'| + 1.$$

**Proof of (b).** An application of Lemma 17 to $(F_1, F'_1)$ and for $(F_2, F'_2)$ produces equal runs for subsequent values of $k$ if $\mathsf{type}(F_1, F'_1) = \mathsf{type}(F_2, F'_2)$. Thus, if $|\mathbf{RegP}(F_1, F'_1)| \leq |\mathbf{RegP}(F_2, F'_2)|$, then for each run in $\mathbf{RegP}(F_1, F'_1)$, an equal run is present in $\mathbf{RegP}(F_2, F'_2)$. For the max-layer $R_1$ of $\mathbf{P}(F_1, F'_1)$ and the regular layer $R_2 \in \mathbf{RegP}(F_2, F'_2)$ with the same period, we have $\text{frag-squares}(R_1 \cap (F_1 \cup F'_1)) \subseteq \text{frag-squares}(R_2 \cap (F_2 \cup F'_2))$. Runs in a pyramid have different periods, so they generate disjoint sets of squares. The max-layer of $\mathbf{P}(F_2, F'_2)$ thus generates a special square that is not generated by $\mathbf{P}(F_1, F'_1)$. ◀

## 7 Counting Squares

For brevity, primitively rooted squares are called *p-squares* and non-primitively rooted squares are called *np-squares* (see [38]). We note that special squares are, in particular, p-squares.

## 7.1 Counting Plain Squares

Recall that a square is plain unless it is special; that is, $U^2$ is plain if it is an np-square or it is not highly periodic. The next lemma follows from [19] (and Fact 8); see full version.

▶ **Lemma 44** (see [19, Theorem 13]). *Assume we are given $r$ periodic fragments in $T$ grouped by their Lyndon roots and that the Lyndon representations of all these periodic fragments are available. The numbers of distinct p-squares and distinct np-squares generated by these periodic fragments can be computed in $\mathcal{O}(r+\sqrt{n})$ time.*

*The same conclusion holds if we are given $r$ periodic fragments in $T$ grouped by their sparse-Lyndon roots and that their sparse-Lyndon representations are available.*

*In each case, any $k$ distinct corresponding squares can be reported in $\mathcal{O}(k+r+\sqrt{n})$ time.*

▶ **Lemma 45.** *The number of np-squares in $T$ can be computed in $\mathcal{O}(n/\log_\sigma n)$ time.*

**Proof.** We use Lemma 44 for counting np-squares generated by all runs that are not regular layers, grouped as in Proposition 36. For runs with small periods, we use Lyndon representations, and for the remaining runs we use sparse-Lyndon representations. There are $\mathcal{O}(n/\log_\sigma n)$ such runs, so np-squares are counted in $\mathcal{O}(n/\log_\sigma n)$ time. ◀

▶ **Lemma 46.** *The number of plain p-squares in $T$ can be computed in $\mathcal{O}(n/\log_\sigma n)$ time.*

**Proof.** By Lemma 37, plain p-squares are generated by runs grouped as in Proposition 36. For each run computed in Proposition 36, we check if it is also reported as a max-layer in Proposition 25. This can be checked globally for all runs in $\mathcal{O}(n/\log_\sigma n)$ time using bucket sort. The runs that turned out to be max-layers are cut into smaller periodic fragments that generate plain p-squares (to avoid counting of special squares) as shown below.

Consider a max-layer $R = T(x\mathinner{.\,.} y)$ with $\mathsf{subper}(R) = p$. Let $R_0,\dots,R_g$ be the sequence of runs with period $p$, sorted with respect to their starting positions, such that $R$ is a max-layer of $\mathbf{P}(R_i,R_{i+1})$ for all $i\in[0\mathinner{.\,.} g)$. Further, let $R_i = T[x_i\mathinner{.\,.} y_i]$ for each $i\in[0\mathinner{.\,.} g]$.

For convenience, for all $i\in\mathbb{Z}\setminus[0\mathinner{.\,.} g]$, set $x_i=\infty$ and $y_i=-\infty$. For $i\in[0\mathinner{.\,.} g]$, let us denote $Y_i := T(\max\{x, y_i-\mathsf{per}(R)+1\}\mathinner{.\,.}\min\{x_{i+2}+\mathsf{per}(R)-1, y\})$. Due to the periodicity of $R$, for any $i,j\in[1\mathinner{.\,.} g-3]$, we have $Y_i=Y_j$. Additionally, any occurrence of a plain p-square generated by $R$ in $R$ is contained in some $Y_i$. Further, each $Y_i$ does not generate any special square of length $2\cdot\mathsf{per}(R)$, as it only contains a single maximal periodic fragment with period $\mathsf{subper}(R)$ that is of length at least $\mathsf{per}(R)$.

Hence, it suffices to use the strings among $Y_0$, $Y_1$, and $Y_{g-2}$ that are of length at least $2\cdot\mathsf{per}(R)$ instead of $R$. Those strings are periodic and their (sparse-) Lyndon representations can be inferred in $\mathcal{O}(1)$ time from the (sparse-) Lyndon representation of the max-layer $R$.

We obtain $\mathcal{O}(n/\log_\sigma n)$ runs that are not max-layers from Proposition 36 and $\mathcal{O}(n/\log_\sigma n)$ periodic fragments constructed as described above from max-layers ($\mathcal{O}(1)$ periodic fragments from each max-layer). By Lemma 44, plain p-squares can be counted in $\mathcal{O}(n/\log_\sigma n)$ time. ◀

## 7.2 Counting Special Squares

By Lemma 37, special squares are only generated by layers of pyramids.

▶ **Lemma 47.** *All pyramids $\mathbf{P}(F,F')$ can be grouped by their types in $\mathcal{O}(n/\log_\sigma n)$ time.*

**Proof.** Recall that the type of a pyramid $\mathbf{P}(F,F')$ is $\mathsf{type}(F,F') = (ov, X, Y)$ where $ov = |F\cap F'|$, $X$ is a length-$p$ suffix of run $F$, $Y$ is a length-$p$ prefix of run $F'$ and $p=\mathsf{per}(F)=\mathsf{per}(F')$. By Proposition 36, if $p\le 2\left\lfloor\frac{1}{18}\log_\sigma n\right\rfloor$, we know the Lyndon roots of $F, F'$, and otherwise, we know their sparse-Lyndon roots.

The Lyndon roots of $F$ and $F'$ are the same. We have $X = \mathsf{rot}_{c_X}(\lambda)$ and $Y = \mathsf{rot}_{c_Y}(\lambda)$ for the common Lyndon root $\lambda$ and some values $c_X, c_Y$ that can be computed from the Lyndon representations of $F, F'$ in $\mathcal{O}(1)$ time. Instead of grouping pyramids by triads $(ov, X, Y)$, it suffices to group them by quadruples $(\lambda, ov, c_X, c_Y)$. Grouping by Lyndon roots $\lambda$ is performed in Proposition 36. The remaining elements of quadruples are integers in $[0\mathinner{.\,.} n)$, so we can bucket sort the quadruples by them in $\mathcal{O}(n/\log_\sigma n)$ time in a stable way (so that we do not break the grouping by Lyndon roots) using Fact 8.

The same argument, with sparse-Lyndon roots instead of Lyndon roots, applies for grouping pyramids by types in case the period of runs $F$, $F'$ is greater than $2\left\lfloor\frac{1}{18}\log_\sigma n\right\rfloor$. ◀

▶ **Lemma 48.** *The number of special squares in $T$ can be computed in $\mathcal{O}(n/\log_\sigma n)$ time.*

**Proof.** We group the pyramids by their types using Lemma 47. By Lemma 42, special squares generated by layers from each group can be considered separately. By Lemma 43(b), we can remove all pyramids of the same type that are not of maximal size (in terms of the number of layers). Among the remaining pyramids, all special squares generated by regular layers are counted using Lemma 43(a). Special squares generated by max-layers are counted by partitioning each max-layer into periodic fragments generating only special squares as in the proof of Lemma 46 and then counting all squares generated by such periodic fragments using Lemma 44. ◀

A combination of Lemmas 45, 46, and 48 implies our main result (Theorem 2).

## 8 Final Remarks

Any $k$ squares, for positive $k$ up to the number of distinct squares in $T$, can be listed as follows. Lemma 44 allows to report subsequent squares. For special squares, we list squares generated by subsequent runs in the tallest pyramid of each type.

▶ **Theorem 49.** *Given a string $T$ of length $n$ over alphabet $[0\mathinner{.\,.}\sigma)$ in packed form and integer $1 < k \le |\mathsf{squares}(T)|$, we can output $k$ distinct squares in $T$ in $\mathcal{O}(n/\log_\sigma n + k)$ time.*

Our algorithms generalize to powers with any given exponent $t > 2$ in the same time complexity. In this case, we do not need to consider regular layers, as they generate no powers of exponent greater than 2. Thus an analogue of Lemma 44 suffices.

▶ **Theorem 50.** *Given a string $T$ of length $n$ over alphabet $[0\mathinner{.\,.}\sigma)$ in packed form and integer $t > 2$, we can compute in $\mathcal{O}(n/\log_\sigma n)$ time the number of distinct $t$-th powers in $T$.*

## References

1 Amihood Amir, Itai Boneh, Panagiotis Charalampopoulos, and Eitan Kondratovsky. Repetition detection in a dynamic string. In *27th Annual European Symposium on Algorithms, ESA 2019*, volume 144 of *LIPIcs*, pages 5:1–5:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ESA.2019.5.

2 Alberto Apostolico. Optimal parallel detection of squares in strings. *Algorithmica*, 8(4):285–319, 1992. doi:10.1007/BF01758848.

3 Alberto Apostolico and Dany Breslauer. An optimal O(log log n)-time parallel algorithm for detecting all squares in a string. *SIAM Journal on Computing*, 25(6):1318–1331, 1996. doi:10.1137/S0097539793260404.

4 Hideo Bannai and Jonas Ellert. Lyndon arrays in sublinear time. In *31st Annual European Symposium on Algorithms, ESA 2023*, volume 274 of *LIPIcs*, pages 14:1–14:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.14.

5 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.

6 Hideo Bannai, Shunsuke Inenaga, and Dominik Köppl. Computing all distinct squares in linear time for integer alphabets. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, volume 78 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.CPM.2017.22.

7 Oren Ben-Kiki, Philip Bille, Dany Breslauer, Leszek Gasieniec, Roberto Grossi, and Oren Weimann. Towards optimal packed string matching. *Theoretical Computer Science*, 525:111–129, 2014. doi:10.1016/J.TCS.2013.06.013.

8 Michael Ben-Or. Lower bounds for algebraic computation trees (preliminary report). In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, STOC 1983*, pages 80–86. ACM, 1983. doi:10.1145/800061.808735.

9 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

10 Srečko Brlek and Shuo Li. On the number of distinct squares in finite sequences: Some old and new results. In *Combinatorics on Words - 14th International Conference, WORDS 2023*, volume 13899 of *Lecture Notes in Computer Science*, pages 35–44. Springer, 2023. doi:10.1007/978-3-031-33180-0_3.

11 Srečko Brlek and Shuo Li. On the number of squares in a finite word. *Combinatorial Theory*, 5(1), 2025. doi:10.5070/C65165014.

12 Panagiotis Charalampopoulos. *Data structures for strings in the internal and dynamic settings*. PhD thesis, King's College London, UK, 2020. URL: https://kclpure.kcl.ac.uk/ws/portalfiles/portal/155221105/2021_Charalampopoulos_Panagiotis_1559341_ethesis.pdf.

13 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski. Faster algorithms for longest common substring. In *29th Annual European Symposium on Algorithms, ESA 2021*, volume 204 of *LIPIcs*, pages 30:1–30:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ESA.2021.30.

14 Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. Efficient enumeration of distinct factors using package representations. In *String Processing and Information Retrieval - 27th International Symposium, SPIRE 2020*, volume 12303 of *Lecture Notes in Computer Science*, pages 247–261. Springer, 2020. doi:10.1007/978-3-030-59212-7_18.

15 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 978–989. IEEE, 2020. doi:10.1109/FOCS46700.2020.00095.

16 Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Longest palindromic substring in sublinear time. In *33rd Annual Symposium on Combinatorial Pattern Matching, CPM 2022*, volume 223 of *LIPIcs*, pages 20:1–20:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.CPM.2022.20.

17 Maxime Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12(5):244–250, 1981. doi:10.1016/0020-0190(81)90024-7.

18 Maxime Crochemore. Transducers and repetitions. *Theoretical Computer Science*, 45(1):63–86, 1986. doi:10.1016/0304-3975(86)90041-1.

19 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Extracting powers and periods in a word from its runs structure. *Theoretical Computer Science*, 521:29–41, 2014. doi:10.1016/J.TCS.2013.11.018.

20 Maxime Crochemore and Wojciech Rytter. Efficient parallel algorithms to test square-freeness and factorize strings. *Information Processing Letters*, 38(2):57–60, 1991. doi:10.1016/0020-0190(91)90223-5.

21 Maxime Crochemore and Wojciech Rytter. Squares, cubes, and time-space efficient string searching. *Algorithmica*, 13(5):405–425, 1995. doi:10.1007/BF01190846.

22 Jonas Ellert. Sublinear time Lempel-Ziv (LZ77) factorization. In *String Processing and Information Retrieval - 30th International Symposium, SPIRE 2023*, volume 14240 of *Lecture Notes in Computer Science*, pages 171–187. Springer, 2023. doi:10.1007/978-3-031-43980-3_14.

23 Jonas Ellert and Johannes Fischer. Linear time runs over general ordered alphabets. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 63:1–63:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.63.

24 Jonas Ellert, Paweł Gawrychowski, and Garance Gourdel. Optimal square detection over general alphabets. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 5220–5242. SIAM, 2023. doi:10.1137/1.9781611977554.CH189.

25 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. doi:10.2307/2034009.

26 Aviezri S. Fraenkel and Jamie Simpson. How many squares can a string contain? *Journal of Combinatorial Theory A*, 82(1):112–120, 1998. doi:10.1006/JCTA.1997.2843.

27 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology.* Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.

28 Dan Gusfield and Jens Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *Journal of Computer and System Sciences*, 69(4):525–546, 2004. doi:10.1016/J.JCSS.2004.03.004.

29 Jin-Ju Hong and Gen-Huey Chen. Efficient on-line repetition detection. *Theoretical Computer Science*, 407(1-3):554–563, 2008. doi:10.1016/J.TCS.2008.08.038.

30 Dominik Kempa. Optimal construction of compressed indexes for highly repetitive texts. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1344–1357. SIAM, 2019. doi:10.1137/1.9781611975482.82.

31 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 756–767. ACM, 2019. doi:10.1145/3313276.3316368.

32 Dominik Kempa and Tomasz Kociumaka. Lempel-Ziv (LZ77) factorization in sublinear time. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024*, pages 2045–2055. IEEE, 2024. doi:10.1109/FOCS61266.2024.00122.

33 Dominik Kempa and Tomasz Kociumaka. On the hardness hierarchy for the O(n√log n) complexity in the word RAM. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025*, pages 290–300. ACM, 2025. doi:10.1145/3717823.3718291.

34 Tomasz Kociumaka. Minimal suffix and rotation of a substring in optimal time. In *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016*, volume 54 of *LIPIcs*, pages 28:1–28:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.CPM.2016.28.

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

36 Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science, FOCS 1999*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.

37 Dmitry Kosolobov. Online detection of repetitions with backtracking. In *Combinatorial Pattern Matching - 26th Annual Symposium, CPM 2015*, volume 9133 of *Lecture Notes in Computer Science*, pages 295–306. Springer, 2015. doi:10.1007/978-3-319-19929-0_25.

38 Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. On the maximum number of cubic subwords in a word. *European Journal of Combinatorics*, 34(1):27–37, 2013. doi:10.1016/J.EJC.2012.07.012.

39 Shuo Li, Jakub Pachocki, and Jakub Radoszewski. A note on the maximum number of k-powers in a finite word. *Electronic Journal of Combinatorics*, 31(3), 2024. doi:10.37236/11270.

40 Michael G. Main and Richard J. Lorentz. An $O(n \log n)$ algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5(3):422–432, 1984. doi:10.1016/0196-6774(84)90021-X.

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

42 Jakub Radoszewski and Wiktor Zuba. Computing string covers in sublinear time. In *String Processing and Information Retrieval - 31st International Symposium, SPIRE 2024*, volume 14899 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2024. doi:10.1007/978-3-031-72200-4_21.

43 Jens Stoye and Dan Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree. *Theoretical Computer Science*, 270(1-2):843–856, 2002. doi:10.1016/S0304-3975(01)00121-9.

44 A. Thue. Über unendliche Zeichenreihen. *Norske Videnskabers Selskabs Skrifter Mat.-Nat. Kl.*, 7:1–22, 1906.