University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Jakub Radoszewski

Algorithmic and Combinatorial Problems Related to Enumeration of Repetitions in Words

PhD dissertation

Supervisor

prof. dr hab. Wojciech Rytter

Institute of Informatics
University of Warsaw

January 2012

Author's declaration:
aware of legal responsibility I hereby declare that I have written this dissertation myself and all the contents of the dissertation have been obtained by legal means.

January, 2012 .....

date *mgr Jakub Radoszewski*

Supervisor's declaration:
the dissertation is ready to be reviewed

January, 2012 .....

date *prof. dr hab. Wojciech Rytter*

Abstract

We present several new combinatorial and algorithmic results related to the repetitive structure of words. The main components of this structure are squares (factors of the type zz), higher powers and runs (called also maximal repetitions). The combinatorics of repetitions is quite difficult and not well understood despite a long research on the subject (the works of Thue date to the beginning of previous century), especially in case of squares. Here we concentrate on the strongly repetitive structure: factors which are higher powers or highly repetitive nonextendible segments (called cubic runs). We obtain simpler and more accurate bounds than for repetitions with long periods. The tools that we develop are used to provide an improved characterization of general runs.

Denote by $\mathsf{squares}(n)$ and $\mathsf{cubes}(n)$ the maximal number of different square factors and different cubic factors in a word of length n. By $\mathsf{runs}(n)$ and $\mathsf{cubic\text{-}runs}(n)$ we denote the maximal number of runs and cubic runs in a word of length n, and by $\mathsf{exp\text{-}runs}(n)$ and $\mathsf{exp\text{-}cubic\text{-}runs}(n)$ we denote the maximal sum of exponents of runs and cubic runs in such a word. All these functions are linear in terms of n, however the exact asymptotics are still unknown. We provide new bounds for highly periodic repetitions, that is, for cubes: $0.5\,n - 2\sqrt{n} \leqslant \mathsf{cubes}(n) \leqslant 0.8\,n$ and for cubic runs: $0.41\,n < \mathsf{cubic\text{-}runs}(n) < 0.5\,n$. Improved bounds for the sum of exponents of runs that we show, $2.035\,n < \mathsf{exp\text{-}runs}(n) < 4.1\,n$, contradict a conjecture by Kolpakov and Kucherov (1999) that $\mathsf{exp\text{-}runs}(n) \leqslant 2n$. We also give a better upper bound of $2.5\,n$ on the function $\mathsf{exp\text{-}cubic\text{-}runs}(n)$. Multiple lower bounds were obtained using extensive computer experiments.

New linear time algorithms for enumeration of repetitions are presented. The structure of runs in a word implies an algorithm finding the number of squares, cubes and powers with higher exponent, and also reporting all different powers. Another algorithm is linear time computation of local periods in a word. Our methods are considerably simpler than previously known linear time algorithms and show new structural relations between the notions of periodicity.

The results of this dissertation were presented in [17, 18, 21, 52]. Research supported by grant No. N206 568540 of the National Science Centre.

Key words: square in a word, cube in a word, run, cubic run, Lyndon word, local period, suffix array

AMS Classification: 68R15, 68P30, 68Q25

Streszczenie

W rozprawie przedstawiamy nowe wyniki kombinatoryczne i algorytmiczne dotyczące struktury powtórzeń w słowach. Głównymi elementami tej struktury są kwadraty (podsłowa typu zz), wyższe potęgi oraz maksymalne powtórzenia. Kombinatoryczne własności powtórzeń w słowach nie zostały jeszcze dokładnie opisane pomimo wieloletnich badań w tej dziedzinie (prace prekursora dziedziny, A. Thuego, pochodzą z początków ubiegłego wieku), jest to prawdziwe zwłaszcza w przypadku kwadratów. W rozprawie koncentrujemy się na strukturze powtórzeń silnie okresowych: podsłów będących wyższymi potęgami oraz maksymalnych powtórzeń sześciennych. Uzyskujemy łatwiejsze i dokładniejsze oszacowania niż w przypadku powtórzeń o długich okresach. Otrzymane narzędzia pozwalają sformułować nowe własności ogólnych maksymalnych powtórzeń.

Przez $\mathsf{squares}(n)$ oraz $\mathsf{cubes}(n)$ oznaczamy odpowiednio maksymalną liczbę różnych kwadratów oraz maksymalną liczbę różnych sześcianów w słowie długości n. Przez $\mathsf{runs}(n)$ oraz $\mathsf{cubic\text{-}runs}(n)$ oznaczamy maksymalną liczbę maksymalnych powtórzeń i, odpowiednio, maksymalnych powtórzeń sześciennych w słowie długości n, a przez $\mathsf{exp\text{-}runs}(n)$ oraz $\mathsf{exp\text{-}cubic\text{-}runs}(n)$ – maksymalną sumę wykładników odpowiednich powtórzeń. Wiadomo, że wszystkie podane funkcje są liniowe względem n, jednakże dokładne oszacowania asymptotyczne nie są jeszcze znane. W rozprawie otrzymaliśmy nowe oszacowania dotyczące powtórzeń silnie okresowych, tzn. sześcianów: $0{,}5\,n - 2\sqrt{n} \leqslant \mathsf{cubes}(n) \leqslant 0{,}8\,n$ oraz maksymalnych powtórzeń sześciennych: $0{,}41\,n < \mathsf{cubic\text{-}runs}(n) < 0{,}5\,n$. Przedstawione przez nas ulepszone oszacowania na maksymalną sumę wykładników maksymalnych powtórzeń, tj. $2{,}035\,n < \mathsf{exp\text{-}runs}(n) < 4{,}1\,n$, obalają hipotezę postawioną przez Kolpakova i Kucherova w 1999 roku, orzekającą, że $\mathsf{exp\text{-}runs}(n) \leqslant 2n$. Uzyskaliśmy także lepsze ograniczenie górne $2{,}5\,n$ na funkcję $\mathsf{exp\text{-}cubic\text{-}runs}(n)$. Większość podanych w pracy ograniczeń dolnych została otrzymana w wyniku eksperymentów z wykorzystaniem programów komputerowych.

Przedstawiamy także nowe liniowe algorytmy związane ze zliczaniem powtórzeń. Korzystając ze struktury maksymalnych powtórzeń w słowie, otrzymujemy algorytm znajdujący liczbę kwadratów, sześcianów i wyższych potęg oraz wyszukujący wszystkie różne potęgi. Kolejny algorytm pozwala na wyznaczanie wszystkich okresów lokalnych w słowie w czasie liniowym. Nasze metody są istotnie prostsze od znanych wcześniej liniowych algorytmów rozwiązujących podane problemy i pokazują nowe związki strukturalne pomiędzy różnymi typami powtórzeń.

Wyniki zawarte w rozprawie zostały wcześniej opublikowane w pracach [17, 18, 21, 52]. Projekt został sfinansowany ze środków Narodowego Centrum

Nauki, grant N206 568540.

Słowa kluczowe: kwadrat w słowie, sześcian w słowie, maksymalne powtórzenie, maksymalne powtórzenie sześcienne, słowo Lyndona, okres lokalny, tablica sufiksowa

Klasyfikacja tematyczna AMS: 68R15, 68P30, 68Q25

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Repetitions and periodicities in words are two of the fundamental topics in combinatorics on words [5, 23, 57] initiated by A. Thue [72]. These notions are widely used in many fields, such as combinatorics on words, pattern matching, automata theory, formal language theory, data compression, molecular biology etc. A deeper description of the motivation and related topics can be found in a survey by Crochemore et al. [14].

We consider several types of repetitions in a word: powers (squares, cubes etc.), runs and cubic runs. The k-th power of a word u, u^k, is composed of k juxtaposed occurrences of this word. The square and the cube are obviously the second and the third power. A word u is called periodic with the period p if $u_i = u_{i+p}$ holds for all i. A run, also called a maximal repetition, is a periodic factor of the word u in which the shortest period repeats at least twice. A run must be maximal, i.e., if extend it by one letter its period increases. A cubic run is similar, but is has to contain at least 3 occurrences of the period.

We study repetitions from a combinatorial point of view — finding bounds on the number of repetitions of a given type that a word may contain — and an algorithmic point of view — efficiently identifying factors being repetitions of different types.

1.1 Words and Repetitions

We consider *words* u over a finite alphabet Σ, $u \in \Sigma^*$. The empty word is denoted by ε. The positions in u are numbered from 1 to $|u|$. By Σ^n we denote the set of all words of length n from Σ^*. For $u = u_1 u_2 \dots u_n$, let us denote by $u[i\,..\,j]$ a *factor* of u equal to $u_i \dots u_j$ (in particular $u[i] = u[i\,..\,i]$). We denote by $u_{(i)}$ a prefix $u[1\,..\,i]$ and by $u^{(i)}$ a suffix $u[i\,..\,n]$.

We say that an integer p is the (shortest) *period* of a word $u = u_1u_2 \dots u_n$ (notation: $p = \mathsf{per}(u)$) if p is the smallest positive integer such that $u_i = u_{i+p}$ holds for all $1 \leqslant i \leqslant n - p$.

If $u = w^k$ (k is a non-negative integer), that is $u = ww \dots w$ (k times), then we say that u is the *k-th power* of the word w. A *square* is a second power of a non-empty word, and a *cube* is a third power of a non-empty word. By $\mathsf{squares}(u)$ we denote the number of different square factors of the word u, and by $\mathsf{cubes}(u)$ we denote the number of different cubic factors of u. For integer $n \geqslant 1$, by $\mathsf{squares}(n)$ and $\mathsf{cubes}(n)$ we denote the maximum of $\mathsf{squares}(u)$ and $\mathsf{cubes}(u)$ over all $u \in \Sigma^n$, see also Fig. 1.1.

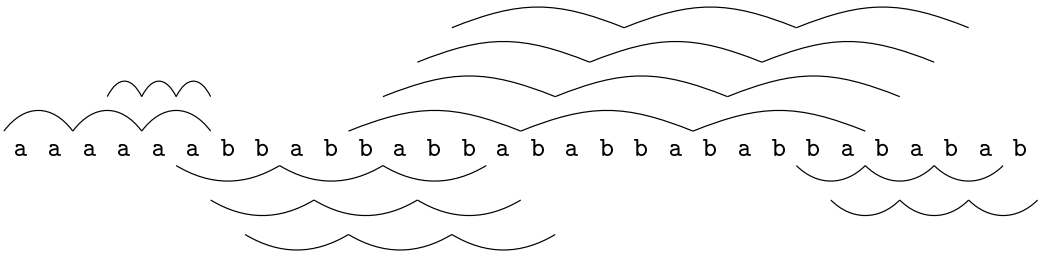


Figure 1.1: Example of a word with 11 different cubes. This is a word of length 30 with the maximum number of cubes among all binary words of the same length

A *run* (also called a maximal repetition) in a word u is an interval $[i, j]$ such that:

- the period p of the associated factor $u[i \,..\, j]$ satisfies $2p \leqslant j - i + 1$,
- the interval cannot be extended to the left nor to the right, without violating the previous property, that is, $u[i - 1] \neq u[i + p - 1]$ and $u[j - p + 1] \neq u[j + 1]$, provided that the respective letters exist.

Each run can be represented as a triple (i, j, p). The (fractional) *exponent* of a run v, denoted $\mathsf{exp}(v)$, is defined as $(j - i + 1)/p$. A run is called a *cubic run* if $\mathsf{exp}(v) \geqslant 3$. For simplicity, in the rest of the text we sometimes refer to runs or cubic runs as to occurrences of the corresponding factors of u.

By $\mathcal{R}(u)$ we denote the set of runs in u and by $\mathcal{CR}(u)$ we denote the set of cubic runs in u. We also introduce the following notation:

- $\mathsf{runs}(u) = |\mathcal{R}(u)|$ and $\mathsf{cubic\text{-}runs}(u) = |\mathcal{CR}(u)|$ are the number of runs and cubic runs in u respectively,

- exp-runs(u) and exp-cubic-runs(u) are the sum of exponents of runs and cubic runs in u respectively.

For a positive integer n, we use the same notations runs(n), cubic-runs(n), exp-runs(n) and exp-cubic-runs(n) to denote the maximal value of the respective function for a word of length n. By cubic-runs$_2(n)$ we denote the maximum over all such binary words.

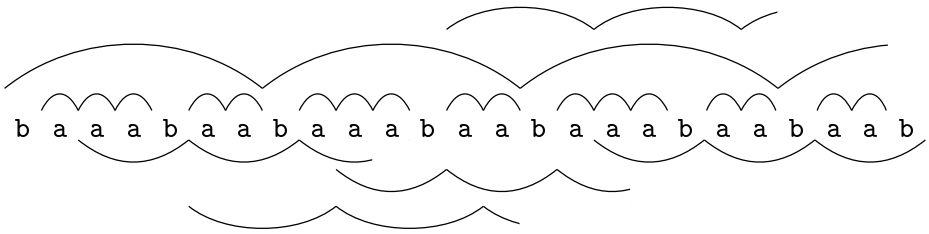


Figure 1.2: All runs present in the word baaabaabaaabaabaaabaabaab are indicated by wavy curves

Example 1.1. *Consider the word from Fig. 1.2. This word contains a run* $v = (1, 24, 7)$*, corresponding to the factor* baaabaabaaabaabaaabaabaa*. The exponent of this run is* exp$(v) = 3\frac{3}{7}$*, hence* v *is a cubic run. On the other hand, the run* $v' = (3, 10, 3)$*, corresponding to the factor* aabaabaa*, is not cubic,* exp$(v') = 2\frac{2}{3}$*.*

1.2 Previous Work on Squares and Powers

1.2.1 Combinatorial Results

The function squares(n) is known to satisfy squares$(n) = O(n)$, this means that all the different squares in a word can be represented in linear space. Particular bounds on this function were given in [31]: $n - o(n) \leqslant$ squares$(n) \leqslant 2n$. The upper bound is obtained by showing that each position in a word may hold at most two rightmost occurrences of squares in the word. Afterwards this bound was slightly improved [39, 40] to squares$(n) \leqslant 2n - \Theta(\log n)$. Recent attempts to reduce the gap between the bounds, basing on different approaches [25, 27], have not proved successful yet.

Obviously, the maximal number of occurrences of squares in a word of length n can be $\Theta(n^2)$, simply consider the word a^n. Several authors studied the maximal number of occurrences of primitively rooted squares [8] and

primitively rooted k-th powers for any $k \geqslant 2$ [10] and in both cases $\Theta(n \log n)$ bounds have been obtained.

The maximal number of squares was also considered for special families of words: Fibonacci words [32, 42] and, more generally, Sturmian words [24, 62, 63].

1.2.2 Algorithmic Results

Multiple algorithms for finding powers in a word have been presented. The largest part of the related literature deals with different approaches to searching for squares in a word. Most of the existing algorithms are rather complex. The first approach is to check if a word contains any square factor at all, or, otherwise, is square-free. Denote by n the length of the considered word. $O(n \log n)$ time algorithms for square-free testing are presented in [16, 60, 66] (the algorithm in [66] is randomized). The optimal $O(n)$ time algorithms are given in [9, 60].

For the problem of finding all different squares, there is a very complex linear time algorithm using suffix trees [37].

Another approach is to find all occurrences of primitively rooted squares in a word, that is, factors of the word in which the shortest period occurs exactly twice. A number of $O(n \log n)$ time algorithms reporting all such occurrences can be found in [3, 8, 51, 59, 71]. Due to the lower bound from [10] these algorithms are time optimal.

Yet another approach is to report simply all occurrences of squares in a word. Denote the number of such occurrences by z, recall that z could be $\Theta(n^2)$. Both $O(n \log n + z)$ time algorithms [54, 59, 71] and $O(n + z)$ time algorithms [37, 49] are known for this problem.

Finally, there are more recent results related to on-line square detection (that is, when the letters of the word are given one by one), improving the time complexity from $O(n \log^2 n)$ [56] to $O(n \log n)$ [44] and $O(n)$ [7].

1.3 Previous Work on Runs

1.3.1 Combinatorial Results

The concept of runs (also called maximal repetitions) has been introduced to represent all repetitions in a word in a succinct manner. The crucial property of runs is that their maximal number in a word of length n is $O(n)$, see Kolpakov and Kucherov [48]. This fact is the cornerstone of any algorithm computing all repetitions in a word of length n in $O(n)$ time.

Due to the work of many authors, tighter bounds on $\mathsf{runs}(n)$ have been obtained. The lower bound of $0.927\,n$ was first proved by Franek and Yang [34]. Afterwards, it was improved by Kusano et al. [53] to $0.944565\,n$ employing computer experiments, and recently by Simpson [70] to $0.944575712\,n$. On the other hand, the first explicit upper bound of $5\,n$ was settled by Rytter [67], afterwards it was systematically improved to $3.48\,n$ by Puglisi et al. [65], $3.44\,n$ by Rytter [69], $1.6\,n$ by Crochemore and Ilie [12, 13] and $1.52\,n$ by Giraud [36]. The best known result $\mathsf{runs}(n) \leqslant 1.029\,n$ is due to Crochemore et al. [15], but it is conjectured [48] that $\mathsf{runs}(n) < n$. The maximal number of runs was also studied for special types of words and tight bounds were established for Fibonacci words [42, 48, 68] and Sturmian words [4, 33, 62]. Cubic runs were also studied in the context of Sturmian words [64].

A stronger property of runs is that the maximal sum of their exponents in a word of length n is linear in terms of n, $\mathsf{exp\text{-}runs}(n) = O(n)$, see the final remarks in Kolpakov and Kucherov [50]. This fact has applications to the analysis of various algorithms, such as computing branching tandem repeats: the linearity of the sum of exponents solves a conjecture [37, 71] concerning the linearity of the number of maximal tandem repeats and implies that all can be found in linear time. The proof that $\mathsf{exp\text{-}runs}(n) < cn$ in Kolpakov and Kucherov's paper [50] is very complex and does not provide any particular value for the constant c. A bound can be derived from the proof of Rytter [67] but the paper mentions only that the obtained bound is "unsatisfactory" (it seems to be $25\,n$). The first explicit bound of $5.6\,n$ for $\mathsf{exp\text{-}runs}(n)$ was given by Crochemore and Ilie [13], who claim that it could be improved to $2.9\,n$ employing computer experiments. As for the lower bound on $\mathsf{exp\text{-}runs}(n)$, no exact values were previously known and it was conjectured [49, 50] that $\mathsf{exp\text{-}runs}(n) < 2n$.

More recently, some work has been done on "maximal repetitions" with exponent less than two [47].

1.3.2 Algorithmic Results

$O(n \log n)$ time algorithms computing all the runs in a word are present explicitly or implicitly in [3, 8, 16, 59, 71]. A first approach to linear time runs computation was made by Main [58], who showed how to compute leftmost occurrences of all runs in $O(n)$ time. A linear time algorithm computing the runs in Fibonacci words was proposed in [42]. Finally, a linear time algorithm computing all the runs in any word, basing on Main's algorithm, was given by Kolpakov and Kucherov [48, 49, 50].

The algorithm of Kolpakov and Kucherov uses Farach's linear time algorithm for suffix tree construction [28] to perform a Lempel-Ziv factor-

ization [9, 55] of the word. More practical algorithms compute the LZ-factorization directly from suffix arrays [1]. Several linear time algorithms finding all the runs in a word with efficient implementations are also given in [6].

1.4 Our Results

We present several new results related both to combinatorics and to algorithmics of repetitions in words. We investigate highly periodic repetitions: cubes and cubic runs. We show that one can obtain tighter estimations for $\mathsf{cubes}(n)$ than for $\mathsf{squares}(n)$:

$$0.5\,n - 2\sqrt{n} \leqslant \mathsf{cubes}(n) \leqslant 0.8\,n. \tag{1.1}$$

For the lower bound we construct an infinite family of binary words, whereas the upper bound follows from several periodic properties of cubes. The analysis of cubes can be found in Chapter 3.

We give the following general bounds on the function $\mathsf{cubic\text{-}runs}(n)$:

$$0.41\,n < \mathsf{cubic\text{-}runs}(n) < 0.5\,n. \tag{1.2}$$

Here we show that $\mathsf{cubic\text{-}runs}(n) = O(n)$ in a much simpler way than this property is proved for the function $\mathsf{runs}(n)$. For this we introduce so called handle functions, basing on properties of Lyndon words (this is described in Chapter 2). As for the lower bound, we again give an infinite family of binary words, this family was constructed using extensive computer experiments. We also set an improved $0.48\,n$ upper bound in the case of binary alphabet. Results related to cubic runs are presented in Chapter 4.

The handle function tool can also be used to improve the known proven upper bound on the maximal sum of exponents of runs in a word. We also bring down a conjecture by Kolpakov and Kucherov [49, 50] stating that $\mathsf{exp\text{-}runs}(n) < 2n$ by showing a family of binary words for which the sum of exponents of runs is greater than $2.035\,n$. We obtain the following bounds:

$$2.035\,n < \mathsf{exp\text{-}runs}(n) < 4.1\,n. \tag{1.3}$$

Additionally we provide an improved upper bound of $2.5\,n$ for the function $\mathsf{exp\text{-}cubic\text{-}runs}(n)$. Details can be found in Chapter 5.

On the algorithmic side, we show how to take advantage of linear upper bounds for the maximal number of different types of repetitions in a word to obtain efficient algorithms enumerating and counting repetitions. We provide linear time algorithms which, knowing all the runs in a word, find all different

squares or cubes in a word, count the number of all occurrences of squares and cubes, and find all local periods in a word. These algorithms are considerably simpler than the previously known linear time algorithms for these problems [37, 26], use data structures with smaller space demands (e.g., suffix arrays instead of suffix trees) and show new structural relations between the corresponding notions of periodicity. The algorithms are described in Chapter 6.

Most parts of this dissertation come from the following papers:

- *On the Maximal Number of Cubic Subwords in a String*, joint work with M. Kubica, W. Rytter and T. Waleń, preliminary version in *Proceedings of International Workshop on Combinatorial Algorithms IWOCA 2009* [52], final version under review at *European Journal of Combinatorics*

- *On the Maximal Number of Cubic Runs in a String*, joint work with M. Crochemore, C. S. Iliopoulos, M. Kubica, W. Rytter and T. Waleń, preliminary version in *Proceedings of Language and Automata Theory and Application LATA 2010* [18], final version in *Journal of Computer and System Sciences* [19]

- *On the Maximal Sum of Exponents of Runs in a String*, joint work with M. Crochemore, M. Kubica, W. Rytter and T. Waleń, preliminary version in *Proceedings of International Workshop on Combinatorial Algorithms IWOCA 2010* [21], final version in *Journal of Discrete Algorithms* [22]

- *Extracting Powers and Periods in a String from Its Runs Structure*, joint work with M. Crochemore, C. S. Iliopoulos, M. Kubica, W. Rytter and T. Waleń, preliminary version in *Proceedings of String Processing and Information Retrieval SPIRE 2010* [17].

Acknowledgements

Most importantly I would like to thank my advisor, Prof. Wojciech Rytter, for constant motivation and invaluable help during the process of writing this dissertation and for co-authoring all research papers included in it. I am also really grateful to my informal second advisor, Prof. Krzysztof Diks, for many kinds of guidance and support during the whole period of my PhD studies. I would like to express my gratitude to Prof. Maxime Crochemore and Prof. Costas Iliopoulos for hosting my Erasmus exchange at King's College London where I was able to perform a significant amount of this research.

Moreover I would like to thank all my co-authors: Michalis Christou, Marek Cygan, Tomasz Kociumaka, Tomasz Kulczyński, Jakub Łącki, Solon P. Pissis, Prof. Krzysztof Stencel and Bartosz Szreder. Special thanks go to Marcin Kubica and Tomasz Waleń who are co-authors of all publications my dissertation is based on.

Finally I would like to thank my friends for encouraging me and to thank my family, especially my mother, for love and strong support during my PhD studies and before.

Chapter 2

Combinatorial-Algorithmic Toolbox

In this chapter we present several crucial tools related to text processing and combinatorics on words that we use throughout the dissertation. We start by recalling important data structures related to text processing: suffix arrays with basic applications of Range Minimum Query data structure. Afterwards we recall the notions of primitive words and Lyndon words and introduce a relation between runs and Lyndon words. This relation, called the Lyndon representation of a run, is used both to improve the upper bounds on the quantity of repetitions in a word and to construct efficient algorithms for enumerating repetitions.

2.1 Suffix Arrays

The suffix array of a word u consists in three tables: SUF, LCP and RANK, see Table 2.1. The SUF array stores the list of positions in u sorted according to the increasing lexicographic order of suffixes starting at these positions, i.e.:

$$u[\mathsf{SUF}[1]\,..\,n] \;<\; u[\mathsf{SUF}[2]\,..\,n] \;<\; \ldots \;<\; u[\mathsf{SUF}[n]\,..\,n].$$

Thus, indices of SUF are ranks of the respective suffixes in the increasing lexicographic order. The LCP array is also indexed by the ranks of the suffixes, and stores the lengths of the longest common prefixes of consecutive suffixes in SUF. Denote by $lcp(i,j)$ the length of the longest common prefix between $u[i\,..\,n]$ and $u[j\,..\,n]$ (for $1 \leqslant i,j \leqslant n$). Then, we set $\mathsf{LCP}[1] = -1$ and, for $1 < i \leqslant n$, we have:

$$\mathsf{LCP}[i] = lcp(\mathsf{SUF}[i-1], \mathsf{SUF}[i]).$$

Finally, the RANK table is an inverse of the SUF array:

$$\mathsf{SUF}[\mathsf{RANK}[i]] = i \quad \text{for } i = 1, 2, \ldots, n.$$

All tables comprising the suffix array can be constructed in $O(n)$ time [11, 45, 46].

	b	a	a	a	b	a	a	b	a	a	a	b	a	a	b	a	a	a	b	a	a	b	a	a	b
index i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
RANK$[i]$	19	1	8	15	23	5	12	20	2	9	16	24	6	13	21	3	10	17	25	7	14	22	4	11	18

rank r	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
SUF$[r]$	2	9	16	23	6	13	20	3	10	17	24	7	14	21	4	11	18	25	1	8	15	22	5	12	19
LCP$[r]$	-1	16	9	2	3	12	5	6	15	8	1	2	11	4	5	14	7	0	1	17	10	3	4	13	6

Table 2.1: The suffix array of the word `baaabaabaaabaabaaabaabaab`

2.2 Range Minimum Queries

Define the range minimum query data structure (RMQ, in short) as follows. Assume that we are given an array $A[1\,..\,n]$ of integers. This array is preprocessed to answer the following form of queries: for an interval $[i, j]$ (where $1 \leqslant i \leqslant j \leqslant n$), find the minimum value $A[k]$ for $i \leqslant k \leqslant j$. The best known RMQ data structures have $O(n)$ preprocessing time and $O(1)$ query time [30, 38].

A common example of using RMQ data structure in text algorithms is computing the longest common extensions, i.e., longest common prefixes between any two suffixes of a word in $O(1)$ time, with $O(n)$ preprocessing time. In order to compute $lcp(i, j)$ we observe that the suffixes starting at the positions i and j are somehow located in the SUF array. Let x be $\min(\mathsf{RANK}[i], \mathsf{RANK}[j])$ and y be $\max(\mathsf{RANK}[i], \mathsf{RANK}[j])$. Then:

$$lcp(i, j) = \min\{\mathsf{LCP}[x + 1], \mathsf{LCP}[x + 2], \ldots, \mathsf{LCP}[y]\},$$

provided that $i \neq j$, see [11, 20, 41]. However, this is exactly the RMQ query over the LCP array which can be answered in $O(1)$ time.

2.3 Primitive Words and Lyndon Words

We start this section with a fundamental fact related to the notion of a period.

Lemma 2.1 (Periodicity lemma [29, 57])**.** *If a word of length n has two periods p and q, such that $p + q \leqslant n + \gcd(p, q)$, then $\gcd(p, q)$ is also a period of the word.*

Often it suffices to use a so called weak version of this lemma, where we only assume that $p + q \leqslant n$.

The *primitive root* of a word u, denoted $\mathsf{root}(u)$, is the shortest word w such that $w^k = u$ for some positive integer k. We call a word u *primitive* if $\mathsf{root}(u) = u$, otherwise it is called *non-primitive*.

By $rot(u, c)$ let us denote a *cyclic rotation* of the word $u \in \Sigma^n$, obtained by moving $(c \bmod n)$ first letters of u to its end (preserving the order of the letters). We say that the words u and $rot(u, c)$ are *cyclically equivalent*. Assume that Σ is totally ordered by $\leqslant$, which induces a lexicographical order on Σ^*, also denoted by $\leqslant$. We say that $\lambda \in \Sigma^*$ is a *minimal Lyndon word* if it is primitive and minimal in the class of words that are cyclically equivalent to it. Similarly, we define a *maximal Lyndon word* as a word that is maximal among all its cyclic rotations. By Lyndon words we mean both minimal and maximal Lyndon words.

Lyndon words admit several useful properties, see [57]. We will use the fact that a Lyndon word has no non-trivial prefix that is also its suffix.

Example 2.2. *The words* `a`*,* `b`*,* `abbbb` *and* `abababb` *are minimal Lyndon words, the words* `a`*,* `ba` *and* `bbabbaa` *are maximal Lyndon words, whereas the words* `aabaab` *and* `abaabb` *are not Lyndon words.*

It is a simple and well-known fact [5] that if u and v are cyclically equivalent then $|\mathsf{root}(u)| = |\mathsf{root}(v)|$. Hence, we can define the *Lyndon root* of a word u, $\mathsf{L\text{-}root}(u)$, as the (only) minimal Lyndon word cyclically equivalent to $\mathsf{root}(u)$, and the *maximal Lyndon root* of u, $\mathsf{L}^M\text{-}\mathsf{root}(u)$, as the (only) maximal Lyndon word cyclically equivalent to $\mathsf{root}(u)$.

2.4 Lyndon Representations

Now we extend the notions of Lyndon roots of words to runs. Let $u \in \Sigma^n$. Let $v = (i, j, p)$ be a run in u. We define the *Lyndon root* of v, denoted $\mathsf{L\text{-}root}(v)$, as $\mathsf{L\text{-}root}(u[i\,..\,i + p - 1])$, and the *maximal Lyndon root* of v, denoted $\mathsf{L}^M\text{-}\mathsf{root}(v)$, as $\mathsf{L}^M\text{-}\mathsf{root}(u[i\,..\,i + p - 1])$. Note that these notions are slightly different from the corresponding notions for words.

Example 2.3. *Consider the word from Fig. 1.2. This word contains a run $v = (1, 24, 7)$, corresponding to the factor* `baaabaabaaabaabaaabaabaa`*. We have $\mathsf{L\text{-}root}(v) =$* `aaabaab` *and $\mathsf{L}^M\text{-}\mathsf{root}(v) =$* `baabaaa`*.*

Each run v can be uniquely represented (*Lyndon representation*) in the following form:

$$v \doteq \lambda^{(a)} \cdot \lambda^m \cdot \lambda_{(b)} \tag{2.1}$$

where $\lambda = \mathsf{L\text{-}root}(v)$ and $0 \leqslant a, b < \mathsf{per}(v)$, see Fig. 2.1. We say that v is a λ-run.

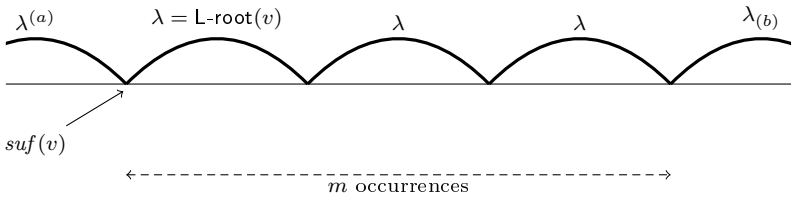


Figure 2.1: A graphical view of the Lyndon representation of a run $v = \lambda^{(a)} \cdot \lambda^m \cdot \lambda_{(b)}$

Example 2.4. *For the run v from Example 2.3, we have $\lambda =$* `aaabaab` *and the following Lyndon representation: $v \doteq \lambda^{(1)} \cdot \lambda^3 \cdot \lambda_{(2)}$.*

In this section we describe a linear time algorithm that groups all runs within u according to their Lyndon roots (Theorem 2.7).

For a run $v = (i, j, p)$, define $suf(v) = k$, where $k \geqslant i$ is the smallest index for which:

$$u[k \,..\, k + p - 1] = \lambda,$$

see Fig. 2.1. This parameter, together with the period $\mathsf{per}(v) = |\lambda|$, provides a unique indication of the Lyndon root of the run. Additionally, define $rank(v) = \mathsf{RANK}[suf(v)]$. In the following two lemmas we show how to compute the values of $suf(v)$ and $rank(v)$ and how to use these values to compare Lyndon roots of runs.

Lemma 2.5. *After $O(n)$ time preprocessing, for any run v in u the values of $suf(v)$ and $rank(v)$ can be computed in constant time.*

Proof. Let $v = (i, j, p)$ and let $\lambda = \mathsf{L\text{-}root}(v)$. The value of $rank(v)$ can be computed using Range Minimum Query on the interval $I = [i, i + p - 1]$ of the table RANK. Indeed, note that the prefixes of length p of the suffixes from the set $S = \{u[d \,..\, n] \ : \ d \in I\}$ are exactly all cyclic rotations of λ, more formally:

$$\{s_{(p)} \ : \ s \in S\} = \{rot(\lambda, c) \ : \ c = 0, 1, \ldots, p - 1\}.$$

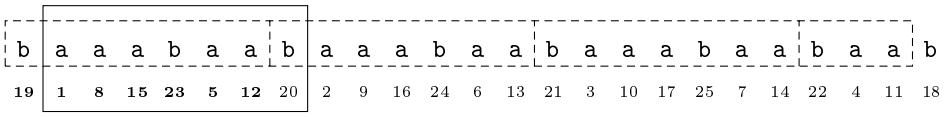


Figure 2.2: The word baaabaabaaabaabaaabaabaab with its RANK table, see also Table 2.1. The Lyndon root of the run $v = (\mathtt{baaabaa})^3\mathtt{baa}$ is indicated by the suffix of rank $\min\{19, 1, 8, 15, 23, 5, 12\} = 1$, hence $\mathsf{L\text{-}root}(v) = \mathtt{aaabaab}$

Hence, the lexicographically minimal element of the set S starts with the prefix λ. And this element can be identified using the aforementioned RMQ query, see also Fig. 2.2.

Recall that the suffix array of u can be computed in $O(n)$ time (see Section 2.1). Also recall that RMQ for an array of length n can be implemented with $O(n)$ preprocessing time and $O(1)$ query time (see Section 2.2). This concludes the computation of $rank(v)$ in the desired time complexity. Finally, $suf(v) = \mathsf{SUF}[rank(v)]$, which can be computed in constant time. □

Lemma 2.6. *After $O(n)$ time preprocessing, equality of Lyndon roots of runs in u can be tested in $O(1)$ time.*

Proof. The Lyndon roots of two runs v_1 and v_2 are equal if and only if $\mathsf{per}(v_1) = \mathsf{per}(v_2)$ and the longest common prefix of the suffixes of u at positions $suf(v_1)$ and $suf(v_2)$ is at least $\mathsf{per}(v_1)$. Recall from Section 2.2 that longest common prefixes of arbitrary suffixes can be computed using RMQ on the LCP array, which can be performed in $O(n)$ preprocessing time and $O(1)$ query time. Hence, the conclusion of the lemma follows from Lemma 2.5. □

We now show how to partition $\mathcal{R}(u)$ into subsets corresponding to different Lyndon roots of runs $\lambda_1, \lambda_2, \ldots, \lambda_t$.

Theorem 2.7. *The set $\mathcal{R}(u)$ of all runs within $u \in \Sigma^n$ can be partitioned in $O(n)$ time into pairwise disjoint classes $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_t$ corresponding to runs with Lyndon roots $\lambda_1, \lambda_2, \ldots, \lambda_t$, where $\lambda_i \neq \lambda_j$ for $i \neq j$.*

Proof. We start the proof of the theorem with the following claim.

Claim 2.8. *Let $L = v_1, v_2, \ldots, v_a$ be a list of all runs in u with period p ordered according to their ranks: $rank(v_1) < rank(v_2) < \ldots < rank(v_a)$. For a given minimal Lyndon word λ, $|\lambda| = p$, all runs in u with the Lyndon root λ form a sublist of L composed of a number of consecutive elements.*

Proof. Let us consider three runs v_{i_1}, v_{i_2} and v_{i_3}, $1 \leqslant i_1 < i_2 < i_3 \leqslant a$. Assume that $\mathsf{L\text{-}root}(v_{i_1}) = \mathsf{L\text{-}root}(v_{i_3}) = \lambda$. Then

$$lcp(suf(v_{i_1}), suf(v_{i_3})) \geqslant p.$$

Due to the rank inequalities we have

$$lcp(suf(v_{i_1}), suf(v_{i_3})) = \min(lcp(suf(v_{i_1}), suf(v_{i_2})),\ lcp(suf(v_{i_2}), suf(v_{i_3}))).$$

Therefore

$$lcp(suf(v_{i_1}), suf(v_{i_2})) \geqslant p$$

and consequently $\mathsf{L\text{-}root}(v_{i_2}) = \mathsf{L\text{-}root}(v_{i_1}) = \lambda$. □

Using Claim 2.8, the requested partition of $\mathcal{R}(u)$ can be obtained in $O(n)$ time in the following three steps, recall that $|\mathcal{R}(u)| = O(n)$.

1. Compute the values of $suf(v)$ and $rank(v)$ for all runs in $\mathcal{R}(u)$ — $O(n)$ time in total due to Lemma 2.5.
2. Represent all runs v in u as pairs $(\mathsf{per}(v), rank(v))$, sort all such pairs lexicographically — $O(n)$ time using Radix Sort.
3. Group runs with equal Lyndon roots — due to Claim 2.8, the groups consist in consecutive elements of the sorted list of pairs, and due to Lemma 2.6, equality of Lyndon roots of runs can be tested in $O(1)$ time with $O(n)$ preprocessing time. This yields $O(n)$ time complexity of this step. □

Define the *compact Lyndon representation* of a run $v = (i, j, p)$ as a tuple:

$$v \triangleq (i, j, p, a, m, b, \ell) \tag{2.2}$$

where ℓ is the length of v and a, m, b are defined as in the (ordinary) Lyndon representation (2.1).

Example 2.9. *For the run v from Example 2.4,* $v \triangleq (1, 24, 7, 1, 3, 2, 24)$.

Due to the following lemma, compact Lyndon representations of runs can be computed efficiently.

Lemma 2.10. *Let u be a word of length n. After $O(n)$ time preprocessing, compact Lyndon representations of runs in u, each run given as a triple (i, j, p), can be computed in constant time.*

Proof. For a run $v = (i, j, p)$, knowing the value of $suf(v)$ the compact Lyndon representation of v can be computed using the following additional formulas:

$$\ell = j - i + 1, \qquad a = suf(v) - i, \qquad m = \lfloor (\ell - a)/p \rfloor, \qquad b = \ell - a - mp.$$

Hence, the statement is a consequence of Lemma 2.5. □

2.5 Handles of Runs

For $u \in \Sigma^n$, we denote by $\mathcal{P} = \{p_1, p_2, \ldots, p_{n-1}\}$ the set of *inter-positions* in u that are located *between* pairs of consecutive letters of u. We show how to use combinatorial properties of Lyndon roots of runs to assign to each run a set of inter-positions from $\mathcal{P}$ called a *handle* of the run so that these sets for different runs are disjoint.

Definition 2.11. *We say that $F : \mathcal{R}(u) \to \mathit{subsets}(\mathcal{P})$ is a* handle function *for the runs in word u if the following condition holds:*

$$F(v_1) \cap F(v_2) = \emptyset \quad \textit{for any } v_1 \neq v_2. \tag{2.3}$$

We say that $F(v)$ is the set of handles *of the run v.*

We define a function $H : \mathcal{R}(u) \to \mathit{subsets}(\mathcal{P})$. Let $\lambda = \mathsf{L\text{-}root}(v)$ and $\lambda' = \mathsf{L}^M\text{-}\mathsf{root}(v)$. $H(v)$ is defined as follows, see Fig. 2.3 and 2.4:

(a) if $\lambda \neq \lambda'$ then $H(v)$ contains all inter-positions in the middle of any occurrence of λ^2 in v, and in the middle of any occurrence of $(\lambda')^2$ in v,

(b) if $\lambda = \lambda'$ then $H(v)$ contains all inter-positions within v.

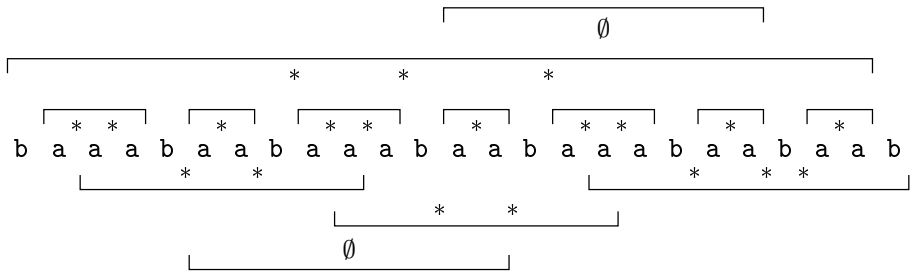


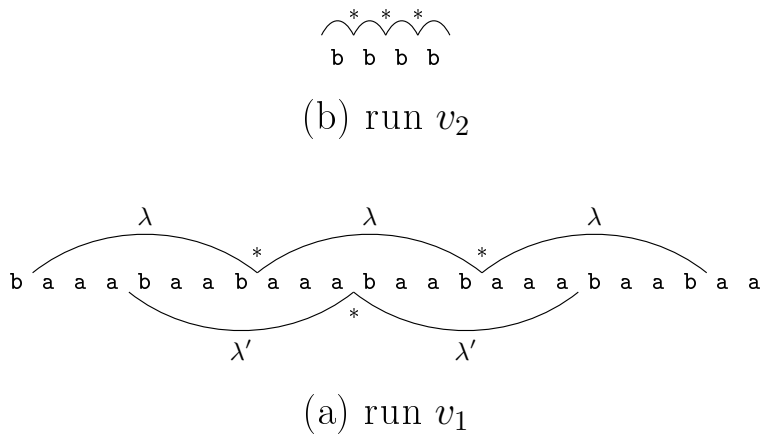
Figure 2.3: Handle sets $H(v)$ of all runs in the word from Fig. 1.2, handles denoted by stars. Note that each cubic run has at least 2 handles. For two runs with period 4 the handle set is empty

Case (b) of the above definition requires an additional explanation, stated in the following simple lemma.

Lemma 2.12. *If $\lambda = \lambda'$ then $|\lambda| = |\lambda'| = 1$.*

Proof. Assume, to the contrary, that $|\lambda| \geqslant 2$. The word λ is a (minimal) Lyndon word, therefore it must contain at least two distinct letters, let us

say: $a = \lambda[1]$ and $b = \lambda[i] \neq a$. If $b < a$ ($b > a$) then the cyclic rotation of λ by $i - 1$ letters is lexicographically smaller than λ (greater than λ) and therefore $\lambda \neq \lambda'$ — a contradiction. Hence, the above assumption is false and $|\lambda| = |\lambda'| = 1$. □



(b) run v_2

(a) run v_1

Figure 2.4: (a) For the run v_1 with period greater than 1 we have $\lambda \neq \lambda'$. (b) For the run v_2 we have $\lambda = \lambda' = \mathtt{b}$ (a single-letter word). The interpositions belonging to the sets $H(v_1)$ and $H(v_2)$ are indicated by stars, we have $|H(v_1)| = |H(v_2)| = 3$

A crucial property of the function H is that it is a handle function.

Theorem 2.13. *For any word $u \in \Sigma^*$, H is a handle function.*

Proof. We need to show that $H(v_1) \cap H(v_2) = \emptyset$ for any two different runs v_1 and v_2 in u. Assume, to the contrary, that $p_i \in H(v_1) \cap H(v_2)$ is a handle of two different runs v_1 and v_2. By the definition of H and Lemma 2.12, p_i is located in the middle of two squares of Lyndon words: λ_1^2 and λ_2^2, where $|\lambda_1| = \mathsf{per}(v_1)$ and $|\lambda_2| = \mathsf{per}(v_2)$. Note that $\lambda_1 \neq \lambda_2$, since otherwise runs v_1 and v_2 would be the same. Without the loss of generality, we can assume that $|\lambda_1| < |\lambda_2|$. Thus the word λ_1 is both a prefix and a suffix of λ_2 which contradicts the fact that λ_2 is a Lyndon word [57], see also Section 2.4. □

Additional properties of handles of runs are given in the next chapters: in Chapter 4 we show that each cubic run has at least 2 handles, and in Chapter 5 we show how $|H(v)|$ corresponds to $\mathsf{exp}(v)$ for $v \in \mathcal{R}(u)$.

Chapter 3

Number of Cubes

We consider asymptotic bounds for the function $\mathsf{cubes}(n)$, denoting the maximal number of different cubic factors in a word of length n. Note that we only consider different cubes, since the total number of occurrences of cubes can be $\Theta(n^2)$.

A trivial lower bound on the number of different cubic factors is the word a^n with $\left\lfloor \frac{n}{3} \right\rfloor$ different cubes, hence $\mathsf{cubes}(n) \geqslant \frac{n}{3}$ for infinitely many n.

As for the upper bound, for a given word u and real $k > 1$, take all repetitions of exponent k (i.e., powers with fractional exponent k) within u, and for each such repetition pick the starting position of its last occurrence within u. In the final remarks in the survey by Crochemore et al. [14] there is a proof that for every $k \geqslant 1+\frac{1+\sqrt{5}}{2}$, each position of u contains at most one such last occurrence of a repetition of exponent k. In particular, this conclusion holds for cubes with $k = 3$, which implies the bound $\mathsf{cubes}(n) \leqslant n - 2$.

We improve both these bounds. Notably, the difference between our upper bound $0.8\,n$ and lower bound $0.5\,n - o(n)$ for $\mathsf{cubes}(n)$ is more than three times smaller than the corresponding difference of known bounds for squares, i.e., $2n - \Theta(\log n)$ and $n - o(n)$.

3.1 Structure of Cubic Occurrences

Let us start with two combinatorial facts related to cubes. The following Lemma 3.1 is an auxiliary fact used to prove Lemma 3.2. Fig. 3.1 illustrates the situation from Lemma 3.1.

Lemma 3.1. *Assume that $x = w^3$ and $y = z^3$, and x is a factor of y starting at position i and ending at position j such that*

$$i \;\leqslant\; \left\lceil \frac{|\mathsf{root}(z)|}{2} \right\rceil + 1 \quad \text{and} \quad j > |z^2|.$$

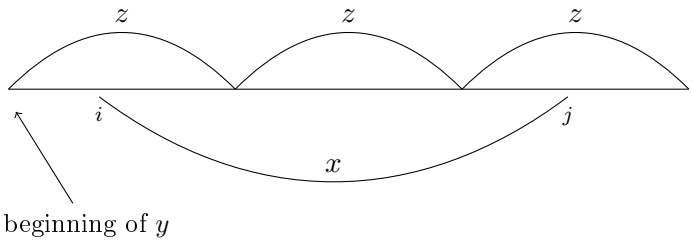


Figure 3.1: The situation from Lemma 3.1

Then, $|\mathit{root}(x)| = |\mathit{root}(y)|$.

Proof. Using the inequalities on i and j from the lemma, we obtain:

$$|x| \;=\; j - i + 1 \;\geqslant\; |z^2| + 1 - \left\lceil \frac{|\mathsf{root}(z)|}{2} \right\rceil - 1 + 1 \;\geqslant$$

$$\geqslant\; 2 \cdot |z| - \left\lceil \frac{|z|}{2} \right\rceil + 1 \;\geqslant\; 2 \cdot |z| - \frac{|z|}{2} \;=\; \frac{3}{2} \cdot |z|.$$

Let us also observe that $|\mathsf{root}(x)|$ and $|\mathsf{root}(y)|$ are both periods of x. Moreover:

$$|x| \;=\; |w^3| \;=\; |w| + \frac{2}{3} \cdot |x| \;\geqslant\; |w| + |z| \;\geqslant\; |\mathsf{root}(x)| + |\mathsf{root}(y)|.$$

From this, by the Periodicity Lemma, we obtain that $\gcd(|\mathsf{root}(x)|, |\mathsf{root}(y)|)$ is also a period of x. However, $\mathsf{root}(x)$ and $\mathsf{root}(y)$ are factors of x, so $|\mathsf{root}(x)| = |\mathsf{root}(y)|$, since in the opposite case one of the words $\mathsf{root}(x)$, $\mathsf{root}(y)$ would not be primitive. □

Lemma 3.2. *Let v^3 and w^3 be two cubes with last occurrences in a word u at positions k and i respectively, such that:*

$$k \;<\; i \;\leqslant\; k + \left\lceil \frac{|\mathit{root}(v)|}{2} \right\rceil.$$

Then:

$$|\mathit{root}(w)| = |\mathit{root}(v)| \quad \mathit{or} \quad |\mathit{root}(w)| \geqslant 2 \cdot |\mathit{root}(v)| - (i - k - 1).$$

Proof. Let us denote $p = |\mathsf{root}(v)|$, $q = |\mathsf{root}(w)|$, and let j be the position of the last letter of w^3. We consider two cases.

Case 1.
Assume that the last occurrence of w^3 is totally inside v^3. Observe that j must then be within the last of the three v's, since otherwise w^3 would occur in u at position $i+p$ or further (see also Fig. 3.1). Hence, due to Lemma 3.1, we obtain $q = p$.

Case 2.
In the opposite case, let x be the maximal prefix of w^3 that lays inside v^3. If $p \neq q$ then $p + q$ must be greater than $|x|$. Indeed, if $p + q \leqslant |x|$ then both $\mathsf{root}(v)$ and $\mathsf{root}(w)$ would be factors of x, so if $p \neq q$, then one of them would not be primitive due to the Periodicity Lemma. Therefore:

$$p + q > |x| = |v^3| - (i - k) \geqslant 3p - (i - k).$$

Consequently $q \geqslant 2p - (i - k) + 1$. □

3.2 Upper Bound

As we have observed in the beginning of the chapter, each position within u contains at most one last occurrence of a cube. The following notion of a *p-occurrence* provides a characterization of such a cube.

Definition 3.3. *Let $u \in \Sigma^n$. We say that there is a* p-occurrence *at position i of u if there exists a cube v^3 with primitive root of length p which occurs at position i in u and does not occur at any further position in u.*

It turns out that the primitive roots of cubes appearing close to each other cannot be arbitrary. It is formally expressed by the following lemma.

Lemma 3.4. *Let $a_1, a_2, \ldots, a_{p+1}$ be an increasing sequence of positions in a word u, such that $a_{j+1} \leqslant a_j + p$ for $j = 1, 2, \ldots, p$. It is not possible for all these positions to contain p-occurrences.*

Proof. Let us assume, to the contrary, that each of the positions $a_1, \ldots, a_{p+1}$ holds a p-occurrence. Observe that the inequalities from the hypothesis of the lemma imply that the primitive roots of cubes occurring at these positions are all cyclic rotations of each other. There are only p different rotations of such primitive roots; therefore, due to the pigeonhole principle, some two of them must be equal.

It suffices to show that all these cubes have the same length, because then some two of them are equal, and consequently one of them is not the last occurrence of the cube.

Assume to the contrary that not all of the considered cubes have equal length. Let a_j and a_{j+1} be two considered positions, such that cubes (v^3 and w^3 respectively) occurring at these positions have different lengths ($3kp$ and $3lp$ respectively, for $k \neq l$). Let us consider two cases. If $l < k$, then $3kp - 3lp \geqslant 3p$, and w^3 occurs in u at position $a_{j+1} + p$ or further. If $k < l$, then $3lp - 3kp \geqslant 3p$ and v^3 appears in u at position $a_j + p$ or further. In both cases we obtain a contradiction. Hence, it is not possible that the lengths of the cubes differ. □

From now on, instead of dealing with words, we will be working with sequences of cubic occurrences within words. For a word u, we introduce a sequence $C_u = (c_i)_{i=1}^{|u|}$ defined as: $c_i = 0$ if there are no last cubic occurrences in position u_i, and $c_i = q$ if there is a q-occurrence in position u_i. For an integer sequence c, by

$$R(c) = \frac{positive(c)}{|c|}$$

we denote the number of positive elements of c divided by the length of c. Now the desired upper bound (1.1) can be expressed as: for any word u, $R(C_u) \leqslant \frac{4}{5}$.

A position i in a sequence c is *zeros-rich* if $c_i > 0$ and there is a block of at least $\left\lceil \frac{c_i}{2} \right\rceil$ zeros following immediately this position in c.

Definition 3.5. *A nonempty finite sequence c composed of non-negative integers is called a* special sequence *if it satisfies all the following conditions:*

(a) If c_i and c_j are two consecutive nonzero elements of c (i.e., $i < j$, $c_i, c_j > 0$ and $c_{i+1} = \ldots = c_{j-1} = 0$) then $j - i \leqslant \left\lceil \frac{c_i}{2} \right\rceil$.

(b) If c_i and c_j are two consecutive nonzero elements of c, then either $c_i = c_j$ or $c_j \geqslant 2c_i - (j - i - 1)$.

(c) No $q + 1$ consecutive nonzero elements of c are equal to q.

(d) The last nonzero position in c, if exists, is zeros-rich.

Example 3.6. *The following sequences are special:*

$$1224444888888880000, \quad 2404044006000, \quad 000, \quad 100.$$

The following sequences are not special:

$$1202200, \quad 33000300, \quad 3330, \quad 122444470000.$$

Lemma 3.7. *For any word u, the sequence C_u admits a prefix which is a special sequence.*

Proof. Let $C_u = (c_i)_{i=1}^{|u|}$. If $c_1 = 0$ then the prefix of C_u of length 1 is a special sequence. From now on we assume that $c_1 > 0$.

First of all, let us prove that there exists a zeros-rich position within C_u. Let q be the maximum element of the sequence C_u and let i be the rightmost position in C_u such that $c_i = q$. From Lemma 3.2, $\left\lceil \frac{q}{2} \right\rceil$ positions following i contain elements equal to zero. Thus, the position i is a zeros-rich position.

Let k be the first zeros-rich position within C_u and let $c_k = q$. We prove that the prefix of C_u of length $l = k + \left\lceil \frac{q}{2} \right\rceil$, which we denote by c', is a special sequence. It suffices to prove that the sequence c' satisfies the conditions (a)-(d) from the above definition.

The condition (d) follows immediately from the choice of the position k. As for the condition (a), if $j - i > \left\lceil \frac{c'_i}{2} \right\rceil$, then the position i would be a zeros-rich position within c', hence also within C_u, preceding the position k, which is not possible. The fact that c' satisfies the condition (b) is implied by Lemma 3.2 and the condition (a). Finally, by Lemma 3.4 and due to (a) we have that no $q + 1$ consecutive positive elements of c' are equal to q, which implies point (c) of the definition of a special sequence. This concludes the proof that c' is a special sequence which is a prefix of C_u. □

Now we introduce a notion of a pessimistic sequence, which we claim maximizes the value of $R(c)$ among all special sequences (see the following Lemma 3.9).

Definition 3.8. *A nonempty sequence c composed of non-negative integers is called a* pessimistic sequence *if it is a special sequence admitting the following additional property: for any $q > 0$, either the element q does not occur in c or it occurs exactly q times, in q consecutive elements of c.*

$$c_1 = 3\,3\,3\;\;0\;\;\underbrace{5\ldots5}_{5\text{ times}}\;\;0\,0\;\;\underbrace{20\ldots20}_{20\text{ times}}\;\;\underbrace{0\ldots0}_{6\text{ times}}\;\;\underbrace{34\ldots34}_{34\text{ times}}\;\;\underbrace{0\ldots0}_{17\text{ times}}$$

$$c_2 = 1\;\;2\,2\;\;4\,4\,4\,4\;\;\underbrace{8\ldots8}_{8\text{ times}}\;\;\ldots\;\;\underbrace{2^k\ldots2^k}_{2^k\text{ times}}\;\;\underbrace{0\ldots0}_{2^{k-1}\text{ times}}$$

Figure 3.2: Examples of pessimistic sequences. The length of the sequence c_1 is 88 and it contains 62 positive elements. The ratio is $R(c_1) = 62/88 \approx 0.70 < 4/5$. For c_2, we have $|c_2| = 5 \cdot 2^{k-1} - 1$ and $\mathit{positive}(c_2) = 2^{k+1} - 1$, hence $R(c_2)$ tends to $\frac{4}{5}$ when $k \to \infty$

Lemma 3.9. *If c is a special sequence then there exists a pessimistic sequence c' for which $R(c') \geqslant R(c)$.*

Proof. Let c be a special sequence. Observe that if c contains such a pair of equal elements $c_i = c_j > 0$, that all the elements between them are equal zero, then all the elements between c_i and c_j can be removed from c without decreasing $R(c)$. Also, if c contains a subsequence of consecutive elements equal to q ($q > 0$) of length less than q then this subsequence can be extended to the length q without decreasing $R(c)$. Observe that none of these steps violates the properties (a)–(d) of a special sequence. By performing the described modification steps a sufficient number of times, we obtain a pessimistic sequence. □

Now we proceed with a proof of the key property of pessimistic sequences, which will enable us to prove the $\frac{4}{5}n$ upper bound on the number of different cubes in a word.

Lemma 3.10. *If c is a pessimistic sequence then $R(c) \leqslant \frac{4}{5}$.*

Proof. If c contains only zeros then the conclusion trivially holds. From now on we assume that c contains at least one positive element.

Define a *trimmed* pessimistic sequence as a prefix of a pessimistic sequence ending at its last zeros-rich position. For a trimmed pessimistic sequence c ending with an element p ($p > 0$), we define

$$R'(c) = \frac{\mathit{positive}(c)}{|c| + \frac{p}{2}}.$$

We will show by induction on the number of different positive elements of a trimmed pessimistic sequence c that $R'(c) \leqslant \frac{4}{5}$. This will imply that in each (ordinary) pessimistic sequence the ratio of positive elements is at most $\frac{4}{5}$.

If there is exactly one such positive element p then c is a sequence of p elements equal to p preceded by r elements equal to zero (possibly $r = 0$). Thus:

$$R'(c) = \frac{p}{r + p + \frac{p}{2}} \leqslant \frac{1}{\frac{3}{2}} = \frac{2}{3} < \frac{4}{5}.$$

Now assume that there are at least two different positive elements in c. Let p be the last element in c. Then c ends with: q elements equal to q, for some $q > 0$, l elements equal to 0, for some $0 \leqslant l < \left\lceil \frac{q}{2} \right\rceil$, and p elements equal to p:

$$c = \ldots \underbrace{q \ldots q}_{q \text{ times}} \underbrace{0 \ldots 0}_{l \text{ times}} \underbrace{p \ldots p}_{p \text{ times}}.$$

Let c' be a prefix of c ending at the last element equal to q. Hence, $|c| = |c'| + l + p$ and $\mathit{positive}(c) = \mathit{positive}(c') + p$. Note that c' is a trimmed pessimistic sequence. Hence, by the inductive hypothesis, we have:

$$R'(c') = \frac{\mathit{positive}(c')}{|c'| + \frac{q}{2}} \leqslant \frac{4}{5},$$

equivalently:

$$5 \cdot \mathit{positive}(c') \leqslant 4 \cdot |c'| + 2q. \tag{3.1}$$

Note that, by the condition (a) of the definition of a special sequence, we have $p \geqslant 2q - l$. Now from (3.1) we conclude that:

$$5 \cdot \mathit{positive}(c') \leqslant 4 \cdot |c'| + p + l$$

which implies that:

$$5(\mathit{positive}(c') + p) \leqslant 4\left(|c'| + l + \frac{3}{2}p\right).$$

From the latter inequality we conclude that:

$$\frac{4}{5} \geqslant \frac{\mathit{positive}(c') + p}{|c'| + l + p + \frac{p}{2}} = \frac{\mathit{positive}(c)}{|c| + \frac{p}{2}}.$$

This inequality is equivalent to $R'(c) \leqslant \frac{4}{5}$, which concludes the inductive proof. □

Thus we have proved that for any pessimistic sequence c, we have $R(c) \leqslant \frac{4}{5}$, hence, by Lemma 3.9, for any special sequence this ratio is bounded by $\frac{4}{5}$. Combining this observation with the fact that C_u starts with a special sequence for any word u, we obtain the aforementioned upper bound (1.1) on the maximal number of cubes in a word, as stated in the following theorem.

Theorem 3.11 (Upper Bound)**.**
The number of different nonempty cubes that occur in a word of length n is not greater than $\frac{4}{5}n$.

Proof. Due to Lemma 3.7, each sequence C_u is a concatenation of special sequences. Consequently $R(u) \leqslant 4/5$, since the frequency ratio in each component (special sequence) is at most 4/5. □

3.3 Lower Bound

Table 3.1 contains examples of several words with higher density of cubic factors. These words have been found using extensive computer experiments. We show a family of binary words which yields a lower bound of $\frac{1}{2}n - 2\sqrt{n}$ for the number of different cubic factors.

For $i \geqslant 1$, let q_i be the word $0^i10^{i+1}1$. Let r_m be the concatenation $q_1q_2 \ldots q_m$, i.e.

$$r_m = \prod_{i=1}^{m} 0^i10^{i+1}1.$$

E.g., $r_4 = 0100100100010001000010000100001000001$.

n	word	#cubes	ratio
20	01110101011011011000	7	0.35
30	000000110110110101101011010101	11	0.36
40	1101101101110111011100010001000100100100	16	0.40
50	11111111110010010010100101001010100101010010101000	20	0.40
60	10100101001010010101001010010101001010010101001010 1001010100	25	0.41
70	00000011011011010110101101010110101101010110101101 0101101010110101010111	30	0.42
80	11011011010110110101101101011010110101011010110101 01101011010101101010110101010111	34	0.42
90	11101101101110110110111011011011101101110110110111 01101110110110111011011101101110111011101110	40	0.44
100	10001010100101010010101001010010101001010010101001 01001010010101001010010100101010010100101001010111	44	0.44

Table 3.1: Examples of words with high density of different cubic factors

Lemma 3.12. *The length of r_m is $m^2 + 4m$.*

Proof. Clearly q_i contains $2i + 3$ bits, so

$$|r_m| \;=\; \sum_{i=1}^{m} 2i + 3 \;=\; m^2 + 4m. \qquad \square$$

Lemma 3.13. *The word r_m contains exactly*

$$\frac{m^2}{2} + \frac{m}{2} - 1 + \left\lfloor \frac{m+1}{3} \right\rfloor$$

different cubes.

Proof. Note that the concatenation $q_i q_{i+1} = \mathtt{0}^i\mathtt{1}\mathtt{0}^{i+1}\mathtt{1}\mathtt{0}^{i+1}\mathtt{1}\mathtt{0}^{i+2}\mathtt{1}$ contains the following $i+1$ cubes:

$$\left(\mathtt{0}^i\mathtt{1}\mathtt{0}\right)^3,\ \left(\mathtt{0}^{i-1}\mathtt{1}\mathtt{0}^2\right)^3,\ \ldots,\ \left(\mathtt{0}\mathtt{1}\mathtt{0}^i\right)^3,\ \left(\mathtt{1}\mathtt{0}^{i+1}\right)^3.$$

Apart from that, in r_m there are $\left\lfloor \frac{m+1}{3} \right\rfloor$ cubes of the form $\mathtt{0}^3, \mathtt{0}^6, \mathtt{0}^9, \ldots$ Thus far we obtained

$$\sum_{i=1}^{m-1}(i+1) + \left\lfloor \frac{m+1}{3} \right\rfloor \ = \ \frac{m^2}{2} + \frac{m}{2} - 1 + \left\lfloor \frac{m+1}{3} \right\rfloor$$

cubes.

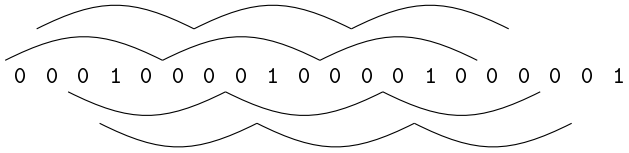


Figure 3.3: For $i = 3$ the word $q_i q_{i+1}$ contains 4 cubes of length $3i+6 = 15$

It remains to show that there are no more cubes in r_m. Note that we have considered all cubes u^3 for which the number of $\mathtt{1}$'s in u equals 0 or 1. On the other hand, if this number exceeds 1 then u would contain the factor $\mathtt{1}\mathtt{0}^i\mathtt{1}$ for some $i \geqslant 1$ and this is impossible, since for a given i such a factor appears within r_m at most twice. □

Theorem 3.14 (Lower Bound)**.**
For infinitely many positive integers n there exists a word of length n for which the number of cubes is greater than $\frac{n}{2} - 2\sqrt{n}$.

Proof. Due to Lemmas 3.12 and 3.13, for any word r_m we have:

$$\begin{aligned}\frac{|r_m|}{2} - \mathsf{cubes}(r_m) \ &= \ \frac{m^2}{2} + 2m - \frac{m^2}{2} - \frac{m}{2} + 1 - \left\lfloor \frac{m+1}{3} \right\rfloor = \\ &= \ \frac{3}{2}m - \left\lfloor \frac{m+1}{3} \right\rfloor + 1 \ \leqslant \ \frac{3}{2}m - \frac{m-1}{3} + 1 \ = \ \frac{7}{6}m + \frac{4}{3}.\end{aligned} \tag{3.2}$$

Note that the value of the expression (3.2) does not exceed $2\sqrt{|r_m|}$ for any $m \geqslant 1$:

$$\left(\frac{7}{6}m + \frac{4}{3}\right)^2 \ = \ \frac{49}{36}m^2 + \frac{28}{9}m + \frac{16}{9} \ < \ 4m^2 + 16m \ = \ 4 \cdot |r_m|.$$

We conclude that:

$$\frac{|r_m|}{2} - \mathsf{cubes}(r_m) < 2\sqrt{|r_m|} \quad \Rightarrow \quad \mathsf{cubes}(r_m) > \frac{|r_m|}{2} - 2\sqrt{|r_m|}. \qquad \square$$

Interestingly, the example of Fraenkel and Simpson [31] of a family of words that contain $n - o(n)$ squares is quite similar to our example, but instead of q_i it uses words of the form $q_i' = \mathtt{0}^{i+1}\mathtt{1}\mathtt{0}^{i}\mathtt{1}\mathtt{0}^{i+1}\mathtt{1}$.

Chapter 4

Number of Cubic Runs

In this section we prove an upper bound $0.5\,n$ and a lower bound $0.41\,n$ on the function cubic-runs(n). In the special case of a binary alphabet we improve the upper bound to $0.48\,n$. Both upper bounds require the notion of handles of runs, defined in Section 2.5. We start by describing the structure of cubic runs in Fibonacci words which will be useful in the construction of the aforementioned lower bound.

4.1 Cubic Runs in Fibonacci Words

Let us analyze the behavior of function cubic-runs for a very common benchmark in text algorithms, i.e., the Fibonacci words, defined recursively as:

$$F_0 = \mathtt{a}, \quad F_1 = \mathtt{ab}, \quad F_n = F_{n-1}F_{n-2} \quad \text{for} \;\; n \geqslant 2.$$

Denote by $\Phi_n = |F_n|$, the n-th Fibonacci number (we assume that for $n < 0$, $\Phi_n = 1$) and by g_n the word F_n with the last two letters removed.

Lemma 4.1. *[61, 68] Each run in F_n is of the form $F_k \cdot F_k \cdot g_{k-1}$ (short runs) or $F_k \cdot F_k \cdot F_k \cdot g_{k-1}$ (long runs), and has a period Φ_k.*

Obviously, in Lemma 4.1 only runs of the form $F_k^3 \cdot g_{k-1}$ are cubic runs.

Denote by $\#occ(u, v)$ the number of occurrences (as a factor) of a word u in a word v.

Lemma 4.2. *For every $k, n \geqslant 0$:*

$$\#occ(F_k^3 \cdot g_{k-1},\ F_n) \;=\; \#occ(F_k^3,\ F_n).$$

Proof. Each occurrence of F_k^3 within F_n must be followed by g_{k-1}, since otherwise it would form a run different from those specified in Lemma 4.1. □

Lemma 4.3. *For every $k \geqslant 2$ and $m \geqslant 0$:*

(a) $\#occ(F_k^3,\ F_{m+k}) = \#occ(\mathtt{aaba},\ F_m)$,

(b) $\#occ(\mathtt{aaba},\ F_m) = \Phi_{m-3} - 1$.

Proof. Recall the Fibonacci morphism φ:

$$\varphi(\mathtt{a}) = \mathtt{ab}, \quad \varphi(\mathtt{b}) = \mathtt{a}.$$

Recall that $F_n = \varphi^n(\mathtt{a})$. The following claim provides a useful tool for the proof of items (a) and (b).

Claim 4.4. *Assume $F_n = uvw$, where $u, v, w \in \{\mathtt{a}, \mathtt{b}\}^*$, $v[1] = \mathtt{a}$ and either $w[1] = \mathtt{a}$ or $w = \varepsilon$. Then there exist unique words u', v', w' such that:*

$$u = \varphi(u'),\ v = \varphi(v'),\ w = \varphi(w'),\ F_{n-1} = u'v'w'.$$

And conversely, if v' is a factor of some F_{n-1} and $v = \varphi(v')$ then v is a factor of F_n.

Proof. It is a straightforward consequence of the definition of φ and the fact that $F_n = \varphi(F_{n-1})$. □

Now we proceed to the actual proof of the lemma. We prove item (a) by induction on k. For $k = 2$ we show the following equalities:

$$\#occ(\mathtt{abaabaaba}, F_{m+2}) \ = \ \#occ(\mathtt{ababaa}, F_{m+1}) \ = \ \#occ(\mathtt{aaba}, F_m). \quad (4.1)$$

As for the first of the equalities (4.1), the occurrence of F_2^3 within F_{m+2} cannot be followed by the letter `a` (since this would imply a longer run, contradicting Lemma 4.1) and cannot be a suffix of F_{m+2} (since either F_4 or F_5 is a suffix of F_{m+2}). Thus:

$$\#occ(\mathtt{abaabaaba}, F_{m+2}) = \#occ(\mathtt{abaabaabab}, F_{m+2}) = \#occ(\mathtt{ababaa}, F_{m+1}).$$

The latter of the above equalities holds due to Claim 4.4, which applies here since no occurrence of `abaabaabab` in F_{m+2} can be followed by the letter `b` (`bb` is not a factor of any Fibonacci word).

To prove the second equality (4.1), we apply a very similar approach: `ababaa` is not a suffix of F_{m+1} and its occurrence cannot be followed by the letter `a`, since no Fibonacci word contains the factor `aaa`. Hence, by Claim 4.4:

$$\#occ(\mathtt{ababaa}, F_{m+1}) = \#occ(\mathtt{ababaab}, F_{m+1}) = \#occ(\mathtt{aaba}, F_m).$$

Finally, the inductive step for $k \geqslant 3$ also follows from Claim 4.4. Indeed, F_k^3 starts with the letter $\mathtt{a}$ and any of its occurrences in F_{m+k} is followed by the letter $\mathtt{a}$, since, by Lemma 4.1, it is a part of a longer run $F_k^3 \cdot g_{k-1}$. Thus:

$$\#occ(F_k^3,\ F_{m+k}) = \#occ(F_{k-1}^3,\ F_{m+k-1}).$$

The proof of item (b) goes by induction on m. For $m \leqslant 3$ one can easily check that $\#occ(\mathtt{aaba},\ F_m) = 0$, and there is exactly one occurrence of $\mathtt{aaba}$ in F_4. The inductive step is a conclusion of the fact that for $m \geqslant 5$ the word F_m contains all occurrences of $\mathtt{aaba}$ from F_{m-1} and F_{m-2} and one additional occurrence overlapping their concatenation:

$$\ldots \mathtt{ab}\,\underbrace{\mathtt{a} \mid \mathtt{aba}}\,\mathtt{ab} \ldots \qquad\qquad \ldots \mathtt{ab}\,\underbrace{\mathtt{aab} \mid \mathtt{a}}\,\mathtt{ba} \ldots$$

The case of $2 \nmid m$. The case of $2 \mid m$.

This concludes the proof of the lemma. □

Lemma 4.5. *For $n > 5$, the word F_n contains (see Fig. 4.1):*

- $\Phi_{n-5} - 1$ *cubic runs* $F_2^3 \cdot g_1$
- $\Phi_{n-6} - 1$ *cubic runs* $F_3^3 \cdot g_2$
  $\vdots$
- $\Phi_1 - 1$ *cubic runs* $F_{n-4}^3 \cdot g_{n-5}$.

Words $F_0, F_1, \ldots, F_5$ do not contain any cubic runs.

Proof. It is easy to check that words F_n for $n \leqslant 5$ do not contain any cubic runs. Let $n > 5$ and $k \in \{2, 3, \ldots, n-4\}$. Denote $m = n - k$. Combining the formulas from Lemmas 4.2 and 4.3, we obtain that:

$$\begin{aligned}\#occ(F_k^3 \cdot g_{k-1},\ F_n) &= \#occ(F_k^3 \cdot g_{k-1},\ F_{m+k}) = \#occ(F_k^3,\ F_{m+k}) \\ &= \#occ(\mathtt{aaba},\ F_m) = \Phi_{m-3} - 1 \\ &= \Phi_{n-k-3} - 1.\end{aligned}$$

□

We are now ready to describe the behaviour of the function $\mathsf{cubic\text{-}runs}(F_n)$. The following theorem not only provides an exact formula for it, but also shows a relationship between the number of cubic runs and the number of different cubes in Fibonacci words. This relationship is similar to the corresponding relationship between the number of (ordinary) runs and the number of (different) squares in Fibonacci words, which always differ exactly by 1, see [61, 68].

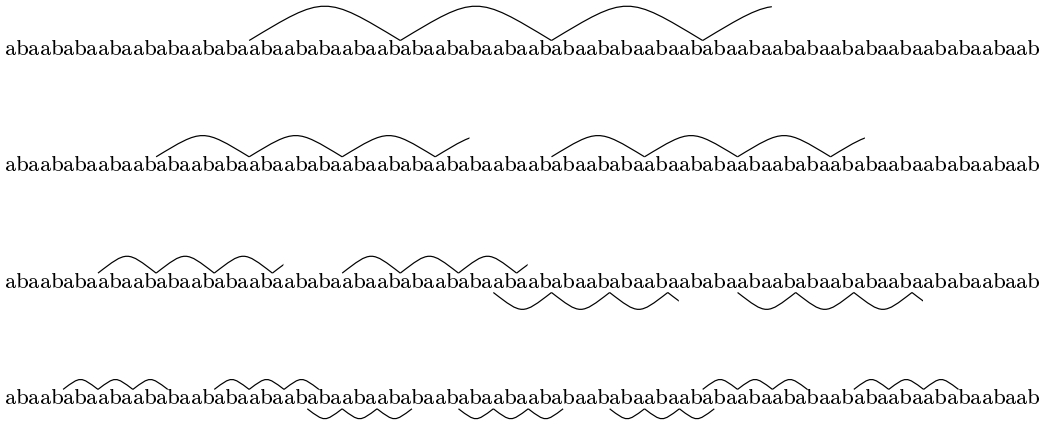


Figure 4.1: The structure of cubic runs in the Fibonacci word F_9. The cubic runs are distributed as follows: 1 run $F_5^3 \cdot g_4$, 2 runs $F_4^3 \cdot g_3$, 4 runs $F_3^3 \cdot g_2$, and 7 runs F_2^3

Theorem 4.6.

(a) $\mathsf{cubic\text{-}runs}(F_n) = \Phi_{n-3} - n + 2$.

(b) $\lim_{n\to\infty} \frac{\mathsf{cubic\text{-}runs}(F_n)}{|F_n|} = \frac{1}{\phi^3} \approx 0.2361$, *where* $\phi = \frac{1+\sqrt{5}}{2}$ *is the golden ratio.*

(c) $\mathsf{cubic\text{-}runs}(F_n) = \mathsf{cubes}(F_n)$.

Proof. (a) From Lemma 4.5 we obtain:

$$\mathsf{cubic\text{-}runs}(F_n) \;=\; \sum_{i=1}^{n-5}(\Phi_i - 1) \;=\; \Phi_{n-3} - 3 - (n-5) \;=\; \Phi_{n-3} - n + 2.$$

(b) It is a straightforward application of the formula from (a):

$$\lim_{n\to\infty} \frac{\mathsf{cubic\text{-}runs}(F_n)}{|F_n|} = \lim_{n\to\infty} \frac{\Phi_{n-3} - n + 2}{\Phi_n} = \frac{1}{\phi^3}.$$

(c) It suffices to note that the number of different cubes of length $3\Phi_{k+1}$ in $F_{k+1}^3 \cdot g_k$ is $|g_k| + 1 = \Phi_k - 1$. Therefore:

$$\mathsf{cubes}(F_n) = \sum_{k=1}^{n-5} (\Phi_k - 1) = \Phi_{n-3} - n + 2 = \mathsf{cubic\text{-}runs}(F_n). \qquad \square$$

4.2 Upper Bound

The general $0.5\,n$ upper bound on the maximal number of cubic runs in a word of length n is based on the handle function H, defined in Section 2.5. It suffices to note that each cubic run is guaranteed to have at least two handles.

Lemma 4.7. *If $v \in \mathcal{CR}(u)$ then $|H(v)| \geqslant 2$.*

Proof. Let $\lambda = \mathsf{L\text{-}root}(v)$ and $\lambda' = \mathsf{L}^M\text{-}\mathsf{root}(v)$. If $\lambda \neq \lambda'$ then v contains both λ^2 and $(\lambda')^2$ as factors, hence $|H(v)| \geqslant 2$. Otherwise ($\lambda = \lambda'$) each inter-position in v is a handle of v, and since the length of v is at least 3, we also have $|H(v)| \geqslant 2$. In both cases the conclusion holds. □

Theorem 4.8 (General Upper Bound)**.**
For any n, $\mathsf{cubic\text{-}runs}(n) < 0.5\,n$.

Proof. We use the fact that H is a handle function (Theorem 2.13). By Lemma 4.7, $|H(v)| \geqslant 2$ for any cubic run v. Thus we obtain:

$$n - 1 \geqslant \sum_{v \in \mathcal{CR}(u)} |H(v)| \geqslant 2 \cdot |\mathcal{CR}(u)| = 2 \cdot \mathsf{cubic\text{-}runs}(u).$$

The conclusion of the theorem follows. □

n	3	4	5	6	7	8	9	10	11
$\mathsf{cubic\text{-}runs}_2(n)$	1	1	1	2	2	2	3	3	3
n	12	13	14	15	16	17	18	19	20
$\mathsf{cubic\text{-}runs}_2(n)$	4	4	5	5	5	6	7	7	7
n	21	22	23	24	25	26	27	28	29
$\mathsf{cubic\text{-}runs}_2(n)$	8	8	8	9	9	10	10	10	11

Table 4.1: The maximum number $\mathsf{cubic\text{-}runs}_2(n)$ of cubic runs in a binary word of length n for $n = 3, \ldots, 29$. Example binary words for which the maximal number of cubic runs is attained are shown in the following Table 4.2

In the case of a binary alphabet, a better upper bound can be stated (see also see Tables 4.1 and 4.2). Let $u \in \{0,1\}^n$. Recall that $\mathcal{P} = \{p_1, p_2, \ldots, p_{n-1}\}$ is the set of all inter-positions of u. These are all candidates for handles of cubic runs from $\mathcal{CR}(u)$. As in the proof of Theorem 4.8, the maximal number of cubic runs would be obtained when there are $\frac{n-1}{2}$ cubic runs, and H assigns to each of them exactly two handles.

n	$\mathsf{cubic\text{-}runs}_2(n)$	u
3	1	aaa
6	2	aaabbb
9	3	aaabbbaaa
12	4	aaabaaabaaab
14	5	aaabaaabaaabbb
17	6	aaabaaabaaabbbaaa
18	7	aaabbbaaabbbaaabbb
21	8	aaabbbaaabbbaaabbbaaa
24	9	aaabbbaaabbbaaabbbaaabbb
26	10	aaabaaabaaabbbaaabbbaaabbb
29	11	aaabaaabaaabbbaaabbbaaabbbaaa

Table 4.2: Lexicographically smallest binary words $u \in \{\mathtt{a}, \mathtt{b}\}^n$, for which $\mathsf{cubic\text{-}runs}(u) = \mathsf{cubic\text{-}runs}_2(n)$ (see also Table 4.1)

Some cubic runs can have more than two handles. Some inter-positions can be not a handle of any cubic runs, such inter-positions are called here *free* inter-positions. The key to the improvement of the upper bound is the localizations of free inter-positions and cubic runs with more than two handles.

Denote:

$$Y = \{\, \mathtt{0},\ \mathtt{01},\ \mathtt{0001},\ \mathtt{0111},\ \mathtt{000111},\ \mathtt{1},\ \mathtt{10},\ \mathtt{1000},\ \mathtt{1110},\ \mathtt{111000} \,\}.$$

By an *internal factor* of a word w we mean any factor of w having an occurrence which is neither a prefix nor a suffix of w. An internal factor can also have an occurrence at the beginning or at the end of w. For example, ab is an internal factor of ababa, but not of abab.

Let X be the set of binary words w which satisfy at least one of the properties:

(1) w has an internal factor which is a non-cubic run containing a square of a word from Y,

(2) w has a factor which is a cube of a word from $Y \setminus \{\mathtt{0},\ \mathtt{1}\}$,

(3) w has a factor 0000 or 1111.

The words $x \in X$ have several useful properties. For example, if $x =$ 110001000101 then the center of the square 00010001 is a free inter-position

in x, since it could only be a handle of a cubic run with period 4, but the run with period 4 containing this square is not cubic. The word 1000100010 is a non-cubic run which is an internal factor of x.

On the other hand, if x contains a factor 000100010001 then it implies a cubic run with 3 handles — the centers of the squares 00010001 and 10001000 (0001 is the minimal rotation and 1000 is the maximal rotation of the period of the run).

The words in X can be checked to satisfy the following simple fact.

Observation 4.9. *Let $u \in \{0,1\}^n$.*

(a) If a factor $u[i\,..\,j]$ contains any factor satisfying point (1) of the definition of X then there is at least one free inter-position in u amongst $p_i, p_{i+1}, \ldots, p_{j-1}$.

(b) If a factor $u[i\,..\,j]$ contains any factor satisfying point (2) or (3) then there are at least 3 inter-positions in u amongst $p_i, p_{i+1}, \ldots, p_{j-1}$ which are handles of the same cubic run.

This implies the following result.

Theorem 4.10 (Improved Upper Bound)**.**
For any n, $\mathsf{cubic\text{-}runs}_2(n) \leqslant 0.48\,n$.

Proof. Each binary word of length 25 contains a factor from X. It has been shown experimentally by checking all binary words of size 25.

Let $u \in \{0,1\}^n$. Let us partition the word u into factors of length 25: $u[1\,..\,25], u[26\,..\,50], \ldots$ (possibly discarding at most 24 last letters of u). By Observation 4.9, it is possible to remove one inter-position from every one of these factors so that each cubic run in u has at least two handles in the set of remaining inter-positions.

The total number of inter-positions in u is $n-1$ and we have shown that at least $\left\lfloor \frac{n-1}{25} \right\rfloor$ of them can be removed and each cubic run will have at least two handles among remaining inter-positions. Hence:

$$\begin{aligned}
\mathsf{cubic\text{-}runs}(u) &\leqslant \frac{1}{2} \cdot \left(n - 1 - \left\lfloor \frac{n-1}{25} \right\rfloor \right) \\
&= \frac{1}{2} \cdot \left(\frac{24 \cdot (n-1)}{25} + \frac{n-1}{25} - \left\lfloor \frac{n-1}{25} \right\rfloor \right) \\
&\leqslant \frac{1}{2} \cdot \left(\frac{24 \cdot (n-1)}{25} + \frac{24}{25} \right) \;=\; 0.48\,n.
\end{aligned}$$

This completes the proof. □

4.3 Lower Bound

We start this section with a simple family of ternary words which yields a $0.4\,n$ lower bound on the function $\mathsf{cubic\text{-}runs}(n)$. Afterwards we improve the bound with a family of binary words.

Theorem 4.11 (Weak Lower Bound)**.**
For infinitely many n we have: $0.4\,n \leqslant \mathsf{cubic\text{-}runs}(n)$.

Proof. As for the lower bound, define:

$$u = \mathtt{0}^3\mathtt{1}^3,\ v = \mathtt{1}^3\mathtt{2}^3,\ w = \mathtt{2}^3\mathtt{0}^3,\ x_k\ =\ (\ u^2\ \mathtt{0}^3\ v^2\ \mathtt{1}^3\ w^2\ \mathtt{2}^3\ )^k.$$

Observe that for any $k \geqslant 1$, the word x_k contains at least $18k-1$ cubic runs. Indeed, we have $15k$ cubic runs with period 1, of the form $\mathtt{0}^3$, $\mathtt{1}^3$ or $\mathtt{2}^3$. Moreover, there are $3k-1$ cubic runs with period 6: $2k$ cubic runs of the form $(\mathtt{0}^3\mathtt{1}^3)^3$ or $(\mathtt{1}^3\mathtt{2}^3)^3$, fully contained within each occurrence of x_1 in $x_k = (x_1)^k$, and $k-1$ cubic runs of the form $(\mathtt{2}^3\mathtt{0}^3)^3$, overlapping the concatenations of consecutive x_1's.

Note that for $k \geqslant 3$, the whole word x_k forms an additional cubic run. Hence, in this case the word x_k has length $45k$ and contains at least $18k$ cubic runs. Thus:

$$\mathsf{cubic\text{-}runs}(x_k)\ \geqslant\ 0.4\,|x_k| \qquad \text{for } k \geqslant 3. \qquad \square$$

The lower bound can be improved in two ways: restricting words to be over binary alphabet and improving the coefficient from 0.4 to 0.41. For this, we use the following morphism, which was found experimentally using a genetic algorithm:

$$\psi(\mathtt{a}) = \mathtt{001110}, \quad \psi(\mathtt{b}) = \mathtt{0001110}.$$

Recall that F_n is the n-th Fibonacci word.

It appears that a sequence defined as $w_n = \psi(F_n)$ consists of cubic-run-rich words, see also Table 4.3. In particular, it can be checked experimentally that the word w_{20} (further denoted as w) of length 113 031 contains 46 348 cubic runs, hence $\mathsf{cubic\text{-}runs}(w) > 0.41\,|w|$. Below we show that for infinitely many words of the form w^k, the density of cubic runs is more than 0.41.

Theorem 4.12 (Improved Lower Bound)**.**
There are infinitely many binary words w^k, where $w = w_{20}$, such that:

$$\frac{r_k}{\ell_k} > 0.41,$$

where $r_k = \mathsf{cubic\text{-}runs}(w^k)$, $\ell_k = |w^k|$.

n	$\lvert w_n\rvert$	cubic-runs$(w_n)/\lvert w_n\rvert$	w_n
0	1	0.16667	$0^2 1^3 0$
1	3	0.23077	$0^2 1^3 0^4 1^3 0$
2	5	0.26316	$0^2 1^3 0^4 1^3 0^3 1^3 0$
3	10	0.31250	$0^2 1^3 0^4 1^3 0^3 1^3 0^3 1^3 0^4 1^3 0$
4	17	0.33333	$0^2 1^3 0^4 1^3 0^3 1^3 0^3 1^3 0^4 1^3 0^3 1^3 0^4 1^3 0^3 1^3 0$
5	30	0.36145	...
6	49	0.36567	
7	83	0.38249	

Table 4.3: Characteristics of a few first elements of the sequence (w_n)

Proof. We start the proof with the following claim, a similar property of the runs function (with different constants) was proved in [53].

Claim 4.13. *For any $k \geqslant 3$, $r_k = Ak - B$, where $A = r_4 - r_3$ and $B = 3r_4 - 4r_3$.*

Proof. We will first show that $r_{k+1} - r_k = r_4 - r_3$, i.e., that the increase of the number of cubic runs when concatenating w^k and w equals the corresponding increase when concatenating w^3 and w. Let $[i\,..\,j]$ be a cubic run in w^{k+1} ending within the last occurrence of w, that is, $j > k \cdot |w|$. In [53] it is proved (as Lemma 2) that the only run in w^{k+1} of length at least $2 \cdot |w|$ is the run equal to the word w^{k+1}. Hence, the cubic run $[i\,..\,j]$ either corresponds to the whole word w^{k+1} or satisfies $i > (k-2) \cdot |w|$. In both cases the cubic runs yield the same increase as when concatenating w to w^3. (Note that in the first case the cubic run forms only an extension of a cubic run already present in w^k, therefore it does not increase the number of cubic runs for any $k \geqslant 3$.)

This concludes that $r_{k+1} - r_k = r_4 - r_3$. From this formula we obtain that, for $k \geqslant 4$:

$$\begin{aligned} r_k &= r_{k-1} + r_4 - r_3 = r_{k-2} + 2 \cdot (r_4 - r_3) = \ldots \\ &= r_3 + (k-3) \cdot (r_4 - r_3) = k \cdot (r_4 - r_3) - (3r_4 - 4r_3). \end{aligned}$$

One can easily check that the same formula holds also for $k = 3$. □

Now we complete the proof of Theorem 4.12. Using an extensive computer experiment one can obtain that:

$$r_3 = 139\,083 \quad \text{and} \quad r_4 = 185\,450, \quad \text{and recall that } |w| = 113\,031.$$

By Claim 4.13, for $k > \frac{10^4 \cdot B}{|w|}$ we have:

$$\begin{aligned} \frac{r_k}{\ell_k} &= \frac{A \cdot k}{|w^k|} - \frac{B}{|w^k|} = \frac{r_4 - r_3}{|w|} - \frac{B}{|w| \cdot k} \\ &> \frac{185\,450 - 139\,083}{113\,031} - 0.0001 > 0.41. \end{aligned}$$

This concludes the proof of the theorem. □

Chapter 5

Sum of Exponents of Runs

In this chapter we provide an upper bound of $4.1\,n$ on the maximal sum of exponents of runs in a word of length n and also a stronger upper bound of $2.5\,n$ on the maximal sum of exponents of cubic runs in a word of length n. Thus we improve the best known proven upper bound of $5.6\,n$ on $\mathsf{exp\text{-}runs}(n)$ from [13]. Again, the main combinatorial tool used to obtain the upper bound are handles of runs (see Section 2.5). As for the lower bound, we bring down a conjecture by Kolpakov and Kucherov [49, 50], that $\mathsf{exp\text{-}runs}(n) < 2n$, by showing an infinite family of binary words for which the sum of exponents of runs is greater than $2.035\,n$.

5.1 Upper Bound for Runs and Cubic Runs

The proof of the upper bound for $\mathsf{exp\text{-}runs}(n)$ uses the properties of the handle function H. For $u \in \Sigma^*$, let $\mathcal{R}_1(u)$ and $\mathcal{R}_{\geqslant 2}(u)$ be the sets of runs in u with period 1 and at least 2, respectively.

Lemma 5.1.

(1) If $v \in \mathcal{R}_1(u)$ *then* $|H(v)| = \mathsf{exp}(v) - 1$.

(2) If $v \in \mathcal{R}_{\geqslant 2}(u)$ *then* $|H(v)| \geqslant 2 \cdot (\lceil \mathsf{exp}(v) \rceil - 3)$.

Proof. Part (1) follows from Lemma 2.12. Assume now that $v \in \mathcal{R}_{\geqslant 2}(u)$ and let w be a prefix of v of length $\mathsf{per}(v)$. The run v starts with a prefix being a k-th power w^k for $k = \lfloor \mathsf{exp}(v) \rfloor$, where $|w| = \mathsf{per}(v)$. Hence, both words λ^{k-1} and $(\lambda')^{k-1}$ are factors of v. Each of the words provides $k-2$ distinct handles for v. Consequently:

$$|H(v)| \geqslant 2 \cdot (\lfloor \mathsf{exp}(v) \rfloor - 2) \geqslant 2 \cdot (\lceil \mathsf{exp}(v) \rceil - 3),$$

which gives part (2) of the lemma. □

In the proof of the upper bound we use the bound $\mathsf{runs}(n) \leqslant 1.029\,n$ on the number of runs from Crochemore et al. [15].

Theorem 5.2 (Upper Bound for Runs)**.**
The sum of exponents of runs in a word of length n is less than $4.1\,n$.

Proof. Let u be a word of length n. Using parts (1) and (2) of Lemma 5.1, we obtain:

$$\begin{aligned}
\mathsf{exp\text{-}runs}(u) &= \sum_{v\in\mathcal{R}_1(u)} \mathsf{exp}(v) + \sum_{v\in\mathcal{R}_{\geqslant 2}(u)} \mathsf{exp}(v) \;\leqslant \\
&\leqslant \sum_{v\in\mathcal{R}_1(u)} (|H(v)|+1) + \sum_{v\in\mathcal{R}_{\geqslant 2}(u)} \left(\frac{|H(v)|}{2}+3\right) \;= \\
&= \sum_{v\in\mathcal{R}_1(u)} |H(v)| + |\mathcal{R}_1(u)| + \sum_{v\in\mathcal{R}_{\geqslant 2}(u)} \frac{|H(v)|}{2} + 3\cdot|\mathcal{R}_{\geqslant 2}(u)| \;\leqslant \\
&\leqslant 3\cdot|\mathcal{R}(u)| + \sum_{v\in\mathcal{R}_1(u)} |H(v)| + \sum_{v\in\mathcal{R}_{\geqslant 2}(u)} \frac{|H(v)|}{2} \;\leqslant \\
&\leqslant 3\cdot|\mathcal{R}(u)| + \sum_{v\in\mathcal{R}(u)} |H(v)|. \qquad (5.1)
\end{aligned}$$

By Theorem 2.13, H is a handle function, therefore $\sum_{v\in\mathcal{R}(u)} |H(v)| < n$. Combining this with (5.1), we obtain:

$$\mathsf{exp\text{-}runs}(u) < 3\cdot|\mathcal{R}(u)| + n \leqslant 3\cdot\mathsf{runs}(n) + n \leqslant 3\cdot 1.029\,n + n < 4.1\,n. \quad \square$$

A similar approach for cubic runs, this time using the upper bound of $0.5\,n$ for $\mathsf{cubic\text{-}runs}(n)$ from Theorem 4.8, enables us to immediately provide a stronger upper bound for the function $\mathsf{exp\text{-}cubic\text{-}runs}(n)$. For a word u, by $\mathcal{CR}_1(u)$ and $\mathcal{CR}_{\geqslant 2}(u)$ we denote the sets of cubic runs in u with period 1 and at least 2, respectively.

Theorem 5.3 (Upper Bound for Cubic Runs)**.**
The sum of exponents of cubic runs in a word of length n is less than $2.5\,n$.

Proof. Let u be a word of length n. From Lemma 5.1, we obtain:

$$\mathsf{exp\text{-}cubic\text{-}runs}(u) = \sum_{v\in\mathcal{CR}_1(u)} \mathsf{exp}(v) + \sum_{v\in\mathcal{CR}_{\geqslant 2}(u)} \mathsf{exp}(v) \;\leqslant$$

$$
\begin{aligned}
&\leqslant \sum_{v\in\mathcal{CR}_1(u)} (|H(v)|+1) + \sum_{v\in\mathcal{CR}_{\geqslant 2}(u)} \left(\frac{|H(v)|}{2}+3\right) = \\
&= \sum_{v\in\mathcal{CR}_1(u)} |H(v)| + |\mathcal{CR}_1(u)| + \\
&+ \sum_{v\in\mathcal{CR}_{\geqslant 2}(u)} \frac{|H(v)|}{2} + 3\cdot|\mathcal{CR}_{\geqslant 2}(u)| \leqslant \\
&\leqslant 3\cdot|\mathcal{CR}(u)| + \sum_{v\in\mathcal{CR}_1(u)} |H(v)| + \sum_{v\in\mathcal{CR}_{\geqslant 2}(u)} \frac{|H(v)|}{2} \leqslant \\
&\leqslant 3\cdot|\mathcal{CR}(u)| + \sum_{v\in\mathcal{CR}(u)} |H(v)|. \qquad (5.2)
\end{aligned}
$$

Again we may use the fact that H is a handle function, which implies $\sum_{v\in\mathcal{CR}(u)} |H(v)| < n$. Combining it with (5.2), we obtain:

$$\mathsf{exp\text{-}cubic\text{-}runs}(u) < 3\cdot|\mathcal{CR}(u)|+n \leqslant 3\cdot\mathsf{cubic\text{-}runs}(n)+n < \frac{3n}{2}+n = 2.5\,n. \ \square$$

5.2 Lower Bound

Let us start the analysis of the lower bound for $\mathsf{exp\text{-}runs}(n)$ by investigating the sum of exponents of runs of words from two known families that contain a large number of runs. We consider first the words defined by Franek and Yang [34], then the Padovan words defined by Simpson [70]. They give large sums of exponents, however below $2n$. Then we construct a new family of words which breaks the barrier of $2n$.

i	$\lvert x_i\rvert$	$\mathsf{runs}(x_i)/\lvert x_i\rvert$	$\mathsf{exp\text{-}runs}(x_i)$	$\mathsf{exp\text{-}runs}(x_i)/\lvert x_i\rvert$
1	6	0.3333	4.00	0.6667
2	27	0.7037	39.18	1.4510
3	116	0.8534	209.70	1.8078
4	493	0.9047	954.27	1.9356
5	2 090	0.9206	4 130.66	1.9764
6	8 855	0.9252	17 608.48	1.9885
7	37 512	0.9266	74 723.85	1.9920
8	158 905	0.9269	316 690.85	1.9930
9	673 134	0.9270	1 341 701.95	1.9932

Table 5.1: Number of runs and sum of exponents of runs in Franek and Yang's [34] words (x_i)

Let $\circ$ be a special concatenation operator defined as:

$$x[1\,..\,n]\circ y[1\,..\,m] = \begin{cases} x[1\,..\,n]y[2\,..\,m] = x[1\,..\,n-1]y[1\,..\,m] & \text{if } x[n] = y[1], \\ x[1\,..\,n-1]y[2\,..\,m] & \text{if } x[n] \neq y[1]. \end{cases}$$

Also let g be a morphism defined as:

$$g(x) = \begin{cases} \mathtt{010010} & \text{if } x = \mathtt{0}, \\ \mathtt{101101} & \text{if } x = \mathtt{1}, \\ g(x[1\,..\,n]) = g(x[1]) \circ g(x[2]) \circ \ldots \circ g(x[n]) & \text{if } |x| > 1. \end{cases}$$

Then $x_i = g^i(\mathtt{0})$ is the family of words described by Franek and Yang [34], which gives the lower bound $\mathsf{runs}(n) \geqslant 0.927\,n$, conjectured for some time to be optimal. The sums of exponents of runs of several first terms of the sequence x_i are listed in Table 5.1.

i	$\lvert y_i\rvert$	$\mathsf{runs}(y_i)/\lvert y_i\rvert$	$\mathsf{exp\text{-}runs}(y_i)$	$\mathsf{exp\text{-}runs}(y_i)/\lvert y_i\rvert$
1	13	0.6154	16.00	1.2308
6	69	0.7971	114.49	1.6593
11	287	0.8990	542.72	1.8910
16	1 172	0.9309	2 303.21	1.9652
21	4 781	0.9406	9 504.38	1.9879
26	19 504	0.9434	38 903.64	1.9946
31	79 568	0.9443	158 862.94	1.9966
36	324 605	0.9445	648 270.74	1.9971
41	1 324 257	0.9446	2 644 879.01	1.9973

Table 5.2: Number of runs and sum of exponents of runs in Simpson's [70] modified Padovan words (y_i)

Define a mapping $\delta(x) = R(f(x))$, where $R(x)$ is the reverse of x and f is the morphism

$$f(\mathtt{a}) = \mathtt{aacab},\ f(\mathtt{b}) = \mathtt{acab},\ f(\mathtt{c}) = \mathtt{ac}.$$

Let y'_i be a sequence of words defined for $i > 5$ recursively using $y'_{i+5} = \delta(y'_i)$. The first 5 elements of the sequence y'_i are:

$$\mathtt{b},\ \mathtt{a},\ \mathtt{ac},\ \mathtt{ba},\ \mathtt{aca}.$$

The words y'_i are called modified Padovan words. If we apply the following morphism h:

$$h(\mathtt{a}) = \mathtt{10100101100101001011010 0},$$

$$h(\mathtt{b}) = \mathtt{1010010110100}, \qquad h(\mathtt{c}) = \mathtt{10100101}$$

to y'_i, we obtain a sequence of run-rich words y_i defined by Simpson [70], which gives the best known lower bound $\mathsf{runs}(n) \geqslant 0.944575712\,n$. Table 5.2 lists the sums of exponents of runs of selected words from the sequence y_i.

The values in Tables 5.1 and 5.2 have been computed experimentally. They suggest that for the families of words x_i and y_i the maximal sum of exponents could be less than $2n$. We show, however, a lower bound for $\mathsf{exp\text{-}runs}(n)$ that is greater than $2n$.

i	$\lvert w_i\rvert$	$\mathsf{exp\text{-}runs}(w_i)$	$\mathsf{exp\text{-}runs}(w_i)/\lvert w_i\rvert$
1	31	47.10	1.5194
2	119	222.26	1.8677
3	461	911.68	1.9776
4	1 751	3 533.34	2.0179
5	6 647	13 498.20	2.0307
6	25 205	51 264.37	2.0339
7	95 567	194 470.30	2.0349
8	362 327	737 393.11	2.0352
9	1 373 693	2 795 792.39	2.0352
10	5 208 071	10 599 765.15	2.0353

Table 5.3: Sums of exponents of runs in words (w_i)

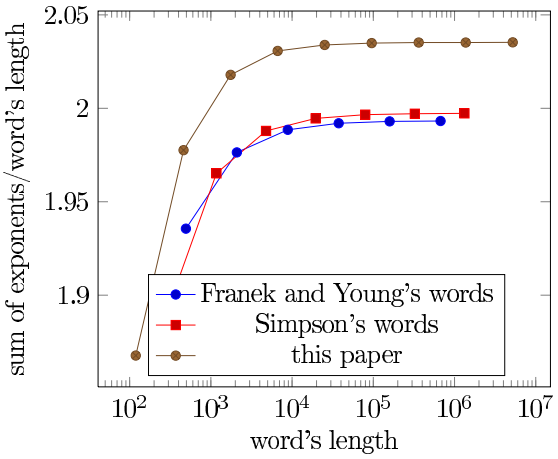


Figure 5.1: The sum of exponents of runs in selected families of words

Theorem 5.4 (Lower Bound)**.**
There are infinitely many binary words w such that:

$$\frac{\textit{exp-runs}(w)}{|w|} > 2.035.$$

Proof. Let us define two morphisms $\phi : \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mapsto \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$ and $\psi : \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mapsto \{\mathtt{0}, \mathtt{1}\}^*$ as follows:

$$\phi(\mathtt{a}) = \mathtt{baaba}, \quad \phi(\mathtt{b}) = \mathtt{ca}, \quad \phi(\mathtt{c}) = \mathtt{bca}$$

$$\psi(\mathtt{a}) = \mathtt{01011}, \quad \psi(\mathtt{b}) = \psi(\mathtt{c}) = \mathtt{01001011}.$$

We set $w_i = \psi(\phi^i(\mathtt{a}))$. Table 5.3 and Fig. 5.1 show the sum of exponents of runs in the words $w_1, \ldots, w_{10}$, computed experimentally.

In particular, the word w_8 has the length 362 327 and its sum of exponents of runs is 737 393.11 (which has been verified using a computer program). Hence, for any word $w = (w_8)^k$, $k \geqslant 1$, we have:

$$\frac{\textsf{exp-runs}(w)}{|w|} > 2.035. \qquad \square$$

Chapter 6

Algorithmic Applications of Runs

Recall that the number of runs in a word of length n is $O(n)$, more precisely, $\mathsf{runs}(n) \leqslant 1.029\,n$ [15]. Additionally, all the runs can be computed in $O(n)$ time [48, 49, 50] and practical and efficient implementations of this algorithm are known [6]. Applications of runs, despite their importance, were underrepresented in the existing literature (approximately one page in [48, 49]). In this chapter we show how to efficiently extract selected notions of periodicity: powers and local periods, from the runs structure of a word.

6.1 Extracting Powers

Denote by $\#powers_k(u)$ the total number of *different* k-th powers in a word $u \in \Sigma^n$. We present a linear time algorithm for computing this function as well as reporting the corresponding powers: for each different k-th power we provide a pair of indices denoting a corresponding factor within u. Recall that the total number of different squares, hence of powers of arbitrary exponent k, is $O(n)$ — the known upper bound for squares is roughly $2n$ [31, 39, 40] and, as we show in Chapter 3, for powers with larger exponent this number is even smaller. Afterwards, at the end of this section we give a formula for the number of all occurrences of all k-th powers in a word. We extensively use the Lyndon representations of runs, a notion developed in Section 2.4.

Each k-th power w^k (for $k \geqslant 2$) occurring in u extends to a run v containing this occurrence for which $\mathsf{per}(v) = |\mathsf{root}(w)|$, we say that w^k is *induced* by the run v. If $\mathsf{L\text{-}root}(w) = \lambda$ then we call w^k λ-*compatible*. Then, obviously, $\mathsf{L\text{-}root}(v) = \lambda$ and v is a λ-run. Hence, two runs may induce the same power of a word only if their Lyndon roots are equal.

For a λ-run v define $maxpower_k(v)$ as the maximal natural β such that some cyclic rotation of $\lambda^{k\beta}$ is induced by v. There exists an obvious formula for this value, as stated in the following observation.

Observation 6.1. *If v is a run of length ℓ with period p then $maxpower_k(v) = \lfloor \ell/(kp) \rfloor$.*

The following lemma shows a correspondence between the Lyndon representation of a run and the set of induced different k-th powers, see also Fig. 6.1.

Lemma 6.2. *Let v be a λ-run with period p and let $\beta = maxpower_k(v)$. Then all powers induced by v are:*

(a) all cyclic rotations of $\lambda^{k\alpha}$ for $\alpha < \beta$,

(b) cyclic rotations $rot(\lambda^{k\beta}, c)$ for $c \in \mathcal{I}_k(v)$, where $\mathcal{I}_k(v) \subseteq [0, p-1]$ is a union of at most two intervals.

The interval $\mathcal{I}_k(v)$ can be computed in $O(1)$ time.

Proof. Let $v \doteq \lambda^{(a)} \cdot \lambda^m \cdot \lambda_{(b)}$ be a run of length ℓ with period p.

Let us first note that, for a given natural α, the run v induces all cyclic rotations

$$rot(\lambda^{k\alpha}, c) \text{ for } c \in [p-a,\ p-a+\ell-kp\alpha]. \tag{6.1}$$

Indeed, the first of these rotations is the prefix w of v of length $kp\alpha$, which satisfies $rot(w, a) = \lambda^{k\alpha}$, hence

$$w = rot(\lambda^{k\alpha}, -a) = rot(\lambda^{k\alpha}, p-a).$$

The run v contains $\ell - kp\alpha + 1$ consecutive cyclic rotations of w: $rot(w, c)$ for $c \in [0, \ell - kp\alpha]$, which correspond exactly to the rotations of $\lambda^{k\alpha}$ as given in the formula (6.1).

As a consequence of (6.1), for $\alpha < \beta$ we obtain all different cyclic rotations of $\lambda^{k\alpha}$, since

$$\ell - kp\alpha \geqslant \ell - kp\left(\left\lfloor \frac{\ell}{(kp)} \right\rfloor - 1\right) \geqslant kp \geqslant p.$$

For $\alpha = \beta$, the interval $[p-a,\ p-a+\ell-kp\alpha]$ must be treated modulo p and forms either a single subinterval of $[0, p-1]$ or a sum of at most two intervals $\mathcal{I}_k(v)$. For $\alpha > \beta$, no cyclic rotation of the word $\lambda^{k\alpha}$ occurs in v, since $|\lambda^{k\alpha}| > |v|$. □

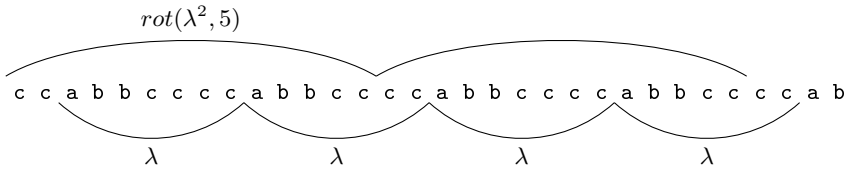


Figure 6.1: The run $\lambda^{(2)}\lambda^4\lambda_{(2)}$ with the Lyndon root $\lambda = \texttt{abbcccc}$ induces all possible different squares cyclically equivalent to λ^2 and 5 squares cyclically equivalent to λ^4, that is, $maxpower_2(v) = 2$ and $\mathcal{I}_k(v) = [0, 2] \cup [5, 6]$

Let λ be a minimal Lyndon word and let $\mathcal{R}_g$ be the set of λ-runs, assume that $\mathcal{R}_g \neq \emptyset$. Let $maxruns_k(u, \lambda)$ be the set of runs $v \in \mathcal{R}_g$ with maximal value of $maxpower_k(v)$. Denote by $\#powers_k(u, \lambda)$ the number of different λ-compatible k-th powers in u. The following lemma is a consequence of Lemma 6.2.

Lemma 6.3. *Let* $\beta_k(\lambda) = \max\{maxpower_k(v) \ : \ v \in \mathcal{R}_g\}$. *Then*

$$\#powers_k(u, \lambda) \ = \ (\beta_k(\lambda) - 1) \cdot |\lambda| + \left| \bigcup_{v \in maxruns_k(u,\lambda)} \mathcal{I}_k(v) \right|$$

$$\#powers_k(u) \ = \ \sum_{\lambda} \#powers_k(u, \lambda).$$

Theorem 6.4. *For a given word* $u \in \Sigma^n$, *the value* $\#powers_k(u)$ *can be computed and all different* k*-th powers in* u *can be reported in* $O(n)$ *time.*

Proof. The value $\#powers_k(u)$ can be computed using the formulas from Lemma 6.3. The main difficulty is to find the size of the union of the sets $\mathcal{I}_k(v)$ for a given group of λ-runs $v \in \mathcal{R}_g$ in $O(|\mathcal{R}_g|)$ time. We perform the following steps:

1. Compute the partition of $\mathcal{R}(u)$ from Theorem 2.7.
2. Compute the compact Lyndon representations of all the runs using Lemma 2.10.
3. For each group $\mathcal{R}_g$ of λ-runs, compute $\beta_k(\lambda)$ and the intervals from $\mathcal{I}_k(v)$ for all $v \in maxruns_k(u, \lambda)$ (using the formulas from the proof of Lemma 6.2).
4. Sort all intervals from $\mathcal{I}_k(v)$ across all the groups $\mathcal{R}_g$ using Radix Sort — we sort the intervals, treated as pairs, in non-descending order, storing the corresponding group numbers ($O(n)$ time).

5. Now the unions of the sets $\mathcal{I}_k(v)$ from a given group $\mathcal{R}_g$ can be computed by a simple left-to-right traversal in $O(|\mathcal{R}_g|)$ time.

Algorithm ReportPowers(u, k)
  $\{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_t\} \leftarrow$ Lyndon partition of $\mathcal{R}(u)$
  $\mathcal{J} \leftarrow \emptyset$;
  for $g \leftarrow 1$ **to** t **do**
    $\beta \leftarrow \max\{\mathit{maxpower}_k(v) : v \in \mathcal{R}_g\}$
    $v_0 \leftarrow \mathrm{maxarg}\{\mathit{maxpower}_k(v) : v \in \mathcal{R}_g\}$
    let $v_0 \triangleq (i, j, p, a, m, b, \ell)$
    for $\alpha \leftarrow 1$ **to** $\beta - 1$ **do**
      for $z \leftarrow i$ **to** $i + p - 1$ **do**
        $\mathit{report}(z,\ z + kp \cdot \alpha - 1)$
    for all v **in** $\mathcal{R}_g$ **do**
      if $\mathit{maxpower}_k(v) = \beta$ **then**
        for all I **in** Compute-$\mathcal{I}(v, k)$ **do**
          $\mathcal{J}.\mathit{insert}((I, v, g))$
  RadixSort$(\mathcal{J})$
  for all $(I, v, g) \in \mathcal{J}$ **do** $\mathcal{J}_g$.insert$((I, v))$
  for $g \leftarrow 1$ **to** t **do**
    $\mathit{pos} \leftarrow 0$
    for all $([\mathit{left}, \mathit{right}], v) \in \mathcal{J}_g$ **do**
      let $v \triangleq (i, j, p, a, m, b, \ell)$
      for $z \leftarrow \max(\mathit{left}, \mathit{pos})$ **to** right **do**
        if $z \geqslant p - a$ **then** $\mathit{start} \leftarrow i + z - p + a$
        else $\mathit{start} \leftarrow i + a + z$
        $\mathit{report}(\mathit{start},\ \mathit{start} + kp \cdot \mathit{maxpower}_k(v) - 1)$
      $\mathit{pos} \leftarrow \max(\mathit{pos}, \mathit{right} + 1)$

The algorithm reporting all different powers is a natural extension of the algorithm computing $\#\mathit{powers}_k(u)$ using the exact formulas from Lemma 6.2, see also the pseudocode of the ReportPowers(u, k) procedure. The structure of the pseudocode is as follows. For a given group of λ-runs, we first find any element $v_0 \in \mathit{maxruns}_k(u, \lambda)$, and using this element we report all powers being cyclic rotations of $\lambda^{k\alpha}$ for $\alpha < \mathit{maxpower}_k(v_0)$, that is, the powers corresponding to part (a) of Lemma 6.2. Afterwards we examine all the elements $v \in \mathit{maxruns}_k(u, \lambda)$, for each of them compute the intervals forming the set $\mathcal{I}_k(v)$ (using an auxiliary Compute-$\mathcal{I}(v, k)$ routine given in the following pseudocode). We populate all the intervals, together with the corresponding runs and group numbers, in a list $\mathcal{J}$. The list is then sorted using Radix Sort, as

```
Algorithm Compute-I(v, k)
    let v ≙ (i, j, p, a, m, b, ℓ)
    left ← p − a;  right ← left + ℓ − kp · β   {formula (6.1)}
    if right − left ⩾ p − 1 then
      I.insert(([0, p − 1], v, g))
    else if right < p then
      I.insert(([left, right], v, g))
    else
      if left < p then
        I.insert(([left, p − 1], v, g))
      I.insert(([0, right − p], v, g))
    return I
```

described above, and divided into parts $\mathcal{J}_g$ corresponding to different groups of λ-runs. Finally, intervals in each list $\mathcal{J}_g$ are summed from left to right, and different powers are reported each time the current union is extended with new elements. For this, simple formulas for the starting positions of the powers (the variable *start*) are derived from the formula (6.1). □

Denote by $\#\mathit{occ\text{-}powers}_k(u)$ the total number of *occurrences* of k-th powers in a word u. We end this section presenting a formula for $\#\mathit{occ\text{-}powers}_k(u)$ which can be evaluated in a straightforward manner to obtain an $O(n)$ time algorithm, where $n = |u|$. Note that the result of the formula can be $\Theta(n^2)$.

Theorem 6.5.

$$\#\mathit{occ\text{-}powers}_k(u) = \\ = \sum_{v=(i,j,p)\in\mathcal{R}(u),\, \ell=j-i+1} \Big(\beta_k(v)\cdot(\ell+1-kp/2) - \beta_k(v)^2\cdot kp/2\Big)$$

where $\beta_k(v)$ *denotes* $\mathit{maxpower}_k(v)$.

Proof. Let $v = (i, j, p)$ be a run of length ℓ in u and let $\lambda = \mathsf{L\text{-}root}(v)$. For each position $z = 1, 2, \ldots, \ell$ of v we count the number of occurrences of k-th powers induced by the run and ending at this position, see also Fig. 6.2:

- for $z \leqslant kp - 1$ there are no such powers,
- for $kp \leqslant z \leqslant 2kp - 1$ there is one such power cyclically equivalent to λ^k,
- for $2kp \leqslant z \leqslant 3kp - 1$ two such powers cyclically equivalent to λ^k and λ^{2k} respectively,

- ...
- for $(\beta-1)\cdot kp \leqslant z \leqslant \beta\cdot kp-1$ there are $\beta-1$ such powers, where $\beta = \left\lfloor \frac{\ell}{kp} \right\rfloor = maxpower_k(v)$,
- finally, for $\beta\cdot kp \leqslant z \leqslant \ell$ there are β such powers.

a	b	a	b	a	b	a	b	a	b	a	b	a	b	a	b	a	b	a
0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2	2	3	3

Figure 6.2: The number of cubes with primitive root cyclically equivalent to $\lambda = \mathtt{ab}$ ending at respective positions of the run $v = (\mathtt{ab})^9\mathtt{a}$. Here $\ell = 19$, $k = 3$, $p = 2$, hence $\beta_k(v) = 3$. By Theorem 6.5, this run induces $3\cdot(20-3) - 3^2\cdot 3 = 3\cdot 8 = 24$ cubes

Hence, the total number of occurrences equals:

$$\sum_{z=1}^{\beta-1}(z\cdot kp) + \beta\cdot(\ell-\beta\cdot kp+1) \;=\; kp\cdot\frac{(\beta-1)\beta}{2} + \beta(\ell+1) - \beta^2\cdot kp$$

$$= \beta\cdot(\ell+1-kp/2) - \beta^2\cdot kp/2. \qquad \square$$

Example 6.6. *Consider the word from Fig. 1.2 and $k = 2$ (counting squares). Here each run has exponent less than 4, i.e., $\ell \leqslant 4p$, therefore $\beta_2(v) = 1$ for each run. The formula from Theorem 6.5 reduces to:*

$$\sum_{(i,j,p)\in\mathcal{R}(u),\, \ell=j-i+1} (\ell+1-2p)\,.$$

6.2 Extracting Local Periods

Another application of the runs structure is the computation of local periods which are related to critical factorization of a word [23]. The known $O(n)$ time algorithm computing all local periods by Duval et al. [26] employs several different techniques modified in a non-trivial way. We present an equally efficient but simpler algorithm using the structure of runs and the solution of the Manhattan Skyline Problem. We start by recalling the definition of a local period.

Let $u \in \Sigma^n$. Recall that by $\mathcal{P} = \{p_1, p_2, \ldots, p_{n-1}\}$ we denote the set of inter-positions in u. We say that a square ww is centered at inter-position p_i of u if both of the following conditions hold, for $x = u_{(i)}$ and $y = u^{(n-i)}$:

- w is a suffix of x or x is a suffix of w,
- w is a prefix of y or y is a prefix of w.

We define the *local period* at inter-position p_i (notation: $\mathsf{localper}[i]$) as $|w|$, where ww is the shortest square centered at this inter-position, see also Fig. 6.3.

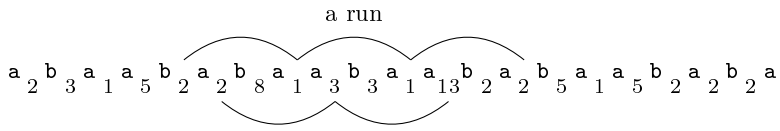


Figure 6.3: A Fibonacci word with local periods at all its inter-positions. Local period at inter-position p_9 of the word is 3, since the smallest period q of a run which completely covers the factor of the word corresponding to the interval $[9-q+1\,..\,9+q]$ equals 3

Clearly, for any p_i there are three possible cases:

Case A: $|w| \leqslant \min(|x|, |y|)$, i.e., ww is an *internal* square of u.

Case B: $\min(|x|, |y|) < |w| \leqslant \max(|x|, |y|)$, i.e., ww is a *left-external* square (if $|w| > |x|$) or a *right-external* square (if $|w| > |y|$).

Case C: $\max(|x|, |y|) < |w|$, i.e., ww is a *doubly-external* square.

We handle Cases A-C separately. In Case A we make use of the structure of runs in u and perform a reduction to the Manhattan Skyline Problem. In Cases B and C we use the border array from the Morris-Pratt algorithm, which is a simple alternative for a modified Boyer-Moore shift function used for this purpose in [26].

6.2.1 Internal Local Periods

The problem of finding internal minimal squares can be reduced in $O(n)$ time to a min-version of the well known Manhattan Skyline Problem, see Fig. 6.4.

Min-Variant of Manhattan Skyline Problem
Input:
a set $\mathcal{S}$ of $O(n)$ subintervals of $[1, n-1]$ with natural heights of size $O(n)$;
Output:
the table $f[t] = \min\{height([i,j]) : t \in [i,j],\ [i,j] \in \mathcal{S}\},\ t \in [1, n-1]$.

Indeed, note that any internal minimal square in u corresponds to one of the runs of u, see also Fig. 6.3. Each run $v = (i, j, q)$ in u induces such

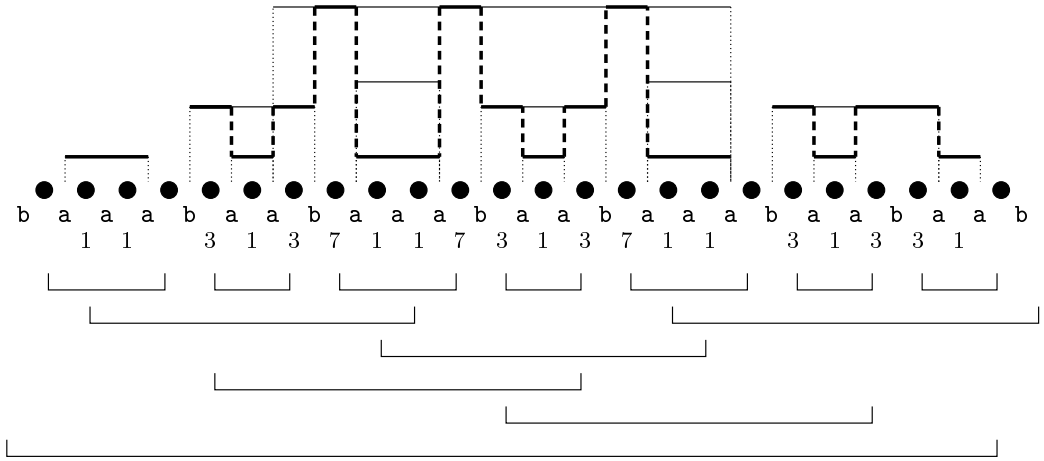


Figure 6.4: The word from Fig. 1.2 with all internal local periods (inter-positions are indicated with small black circles). Below the word is the runs structure. Above the word is the corresponding instance of the Min-Variant of the Manhattan Skyline Problem

squares with root q at inter-positions $p_{i+q-1}, p_{i+q}, \ldots, p_{j-q}$. Thus for each inter-position p_i we need to find the shortest period of a run (i.e., the smallest height of an interval from the Manhattan Skyline Problem) inducing a square at this inter-position.

Due to the following lemma, this reduction yields a linear time algorithm for computing internal minimal squares.

Lemma 6.7. *The Min-Variant of the Manhattan Skyline Problem can be solved in linear time.*

Proof. The solution works as follows.

Sort intervals from $\mathcal{S}$ according to their heights (in non-decreasing order)
Initialize $X = \emptyset$
for each interval $[i, j] \in \mathcal{S}$ (in the sorted order) **do**
  for each $t \in$ list-all-elements$([i, j] \setminus X)$ **do**
    $f[t] \leftarrow \mathit{height}([i, j])$
  $X \leftarrow X \cup [i, j]$

The key elementary operations in the above routine are listing elements in the set $[i, j] \setminus X$ and finding a union of sets $X \cup [i, j]$. In the following claim we show that these operations on sets can be implemented efficiently, so that the whole routine works in $O(n)$ time.

Claim 6.8. *Assume initially $X = \emptyset$. Let $\mathcal{I}$ be a family of $O(m)$ intervals $[i, j]$ from the universe $[1, m]$. Then the sequence of pairs of operations:*

$$\{ \quad \textit{list-all-elements}([i, j] \setminus X); \quad X \leftarrow X \cup [i, j]; \quad \} \tag{6.2}$$

for all $[i, j] \in \mathcal{I}$ can be implemented in $O(m)$ time.

Proof. The implementation uses a restricted version of the find/union data structure, in which we are allowed to union only adjacent subintervals. Thus the *structure* of union operations forms a static tree (here it is a path graph) and therefore $O(m)$ find/union operations can be performed in $O(m)$ time [35] (see also [43]).

In the algorithm the universe $[1, m + 1]$ (extended to the right by a sentinel) is partitioned into maximal segments of elements of X followed by a single element which is not in X: all elements in such a segment form a single find/union component which stores the index of its rightmost position. The operations (6.2) are implemented by traversing the components intersecting the interval $[i, j]$, reporting their rightmost elements and summing them one by one. □

This concludes the proof of the lemma. □

A pseudocode od the InternalLocalPeriods algorithm summarizes linear time computation of internal local periods.

```
Algorithm InternalLocalPeriods
    for all (i, j, q) ∈ R(u) do
        L.insert((q, i + q − 1, j − q))
    RadixSort(L)
    for i ← 1 to n do
        MakeSet(i);  Last[i] ← i
        localper[i] ← ∞
    for all (h_i, l_i, r_i) ∈ L do
        k ← Last[Find(l_i)]
        while k ⩽ r_i do
            localper[k] ← h_i
            Last[Find(k)] ← Last[Find(k + 1)]
            Union(k, k + 1)
            k ← Last[Find(k)]
```

Lemma 6.9. *Internal minimal squares can be computed in $O(n)$ time.*

6.2.2 Left and Right-External Local Periods

Recall that a word that is both a prefix and a suffix of a word u is called a *border* of the word u; a border of u is called proper if it is shorter than u. Denote by $\mathsf{border}[i]$, for $i = 1, 2, \ldots, n$, the length of the longest proper border of $u[1\,..\,i]$. Recall that the border array can be computed in $O(n)$ time, as in the Morris-Pratt algorithm [23].

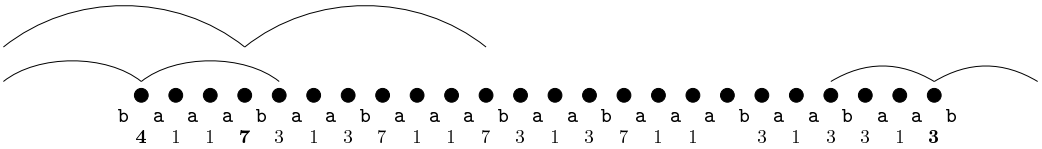


Figure 6.5: The word from Fig. 6.4 with two left-external minimal squares and one right-external minimal square. We have $\mathsf{border}[5] = 1$ and $\mathsf{localper}[1] = 4$, similarly $\mathsf{border}[11] = 4$ and $\mathsf{localper}[4] = 7$

The following lemma shows how the border array can be used to compute left-external minimal squares, see also Fig. 6.5. The case of right-external minimal squares is symmetric and can be treated similarly by considering the reversed word u.

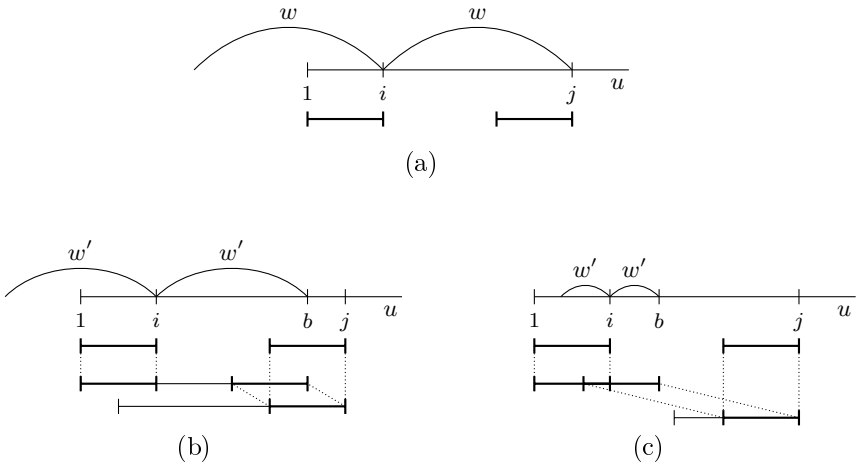


Figure 6.6: Illustration of the proof of Lemma 6.10

Lemma 6.10.

(1) If the shortest left-external minimal square at inter-position p_i is ww then there exists $j > i$ such that $\mathsf{border}[j] = i$ and $|w| = j - i$.

(2) If $\mathsf{border}[j] = i$ *for any* $j = 2, 3, \ldots, n$ *and* $i > 0$ *then* $\mathsf{localper}[i] \leqslant j - i$.

Proof. (1) Let ww be a left-external minimal square at inter-position p_i and let $j = i + |w|$. Note that there exists a border of $u[1\,..\,j]$ of length i, see Fig. 6.6a. It remains to show that $\mathsf{border}[j] = i$, i.e., that i is the longest proper border of $u[1\,..\,j]$.

Assume to the contrary that $b = \mathsf{border}[j] > i$. Then the word $u[1\,..\,i]$ would be a border of $u[1\,..\,b]$, what implies a square $w'w'$ centered at p_i with the root $b - i < j - i = \mathsf{localper}[i]$, which is either left-external or internal (see Fig. 6.6bc), a contradiction.

(2) It is a consequence of the fact that the condition $\mathsf{border}[j] = i$ implies a square centered at p_i with root $j - i$, either internal or left-external. □

Algorithm LeftExternalLocalPeriods updates the localper array constructed using internal minimal squares by considering left-external minimal squares. It performs a left-to-right traversal, basing on the properties (1) and (2) stated in Lemma 6.10. Clearly, it works in $O(n)$ time.

```
Algorithm LeftExternalLocalPeriods
    for j ← 1 to n do
        i ← border[j]
        if i > 0 then localper[i] ← min(localper[i], j − i)
```

6.2.3 Doubly-External Local Periods

Consider a doubly-external minimal square ww at inter-position p_i of u. Let b be the longest overlap between $u^{(n-i)}$ and $u_{(i)}$, i.e., the longest suffix of the former word which is also a prefix of the latter word. Then, clearly, $|w| = n-b$, see Fig. 6.7. Note that b is the length of the longest border of u which is not longer than $\min(i, n - i)$.

Example 6.11. *Consider the word* `baaabaabaaabaabaaabaabaab` *for which almost all local periods are given in Fig. 6.5. Only one inter-position,* p_{18}*, lacks an internal or a one-side-external minimal square centered at it. The longest overlap between* $u^{(7)} =$ `baabaab` *and* $u_{(18)} =$ `baaabaabaaabaabaaa` *equals 1 (a single letter* `b`*). Hence,* $\mathsf{localper}[18] = 24$.

Recall that the lengths of all proper borders of u are iterations of the form $\mathsf{border}^{(j)}[n]$. This concludes an $O(n)$ time algorithm DoublyExternalLocalPeriods which updates the localper array obtained after the previous

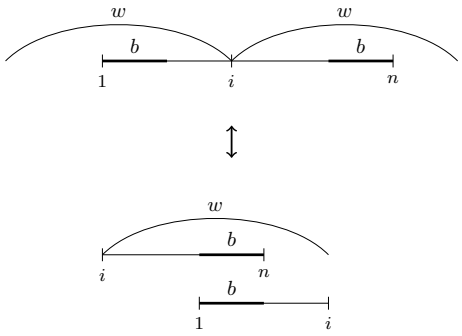


Figure 6.7: Doubly-external minimal squares and borders

cases considering all doubly-external minimal squares, filling the array from its middle to its sides.

```
Algorithm DoublyExternalLocalPeriods
    b ← border[n]
    i ← n div 2
    while i > 0 do
      while i ⩾ b do
        localper[i] ← min(localper[i], n − b)
        localper[n − i + 1] ← min(localper[n − i + 1], n − b)
        i ← i − 1
      if b > 0 then b ← border[b]
```

Combining the solutions to the internal, left- and right-external, and doubly-external minimal squares problems, we obtain the following result.

Theorem 6.12. *All local periods of a word u of length n can be computed in $O(n)$ time using the runs structure of u and the* border *array.*

Chapter 7

Conclusions

We presented several new or improved lower and upper bounds on the maximal number of repetitions in a word of a given length or the sum of exponents of such repetitions. Particularly tight bounds were given for highly periodic repetitions: cubes and cubic runs, for which the proof of linearity of the maximal number of repetitions turned out much easier than in the general case of squares and runs respectively.

The work on improving the bounds on the number of repetitions goes towards a better understanding of the structure of repetitions that any word may contain. Thanks to such bounds we can represent the repetitive structure of a word in a succinct way. This gives an idea why progress in this area is so important. Improving the upper bounds often requires an analysis of a large number of cases, which may be computer-assisted, as for the binary upper bound on $\mathsf{cubic\text{-}runs}(n)$ in this dissertation, and the best known lower bounds for most types of repetitions involved extensive computer experiments using heuristics of artificial intelligence.

A significant amount of work still needs to be done to verify several hypotheses related to the bounds on repetitions. In particular, the "squares" conjecture, that $\mathsf{squares}(n) < n$, and the "runs" conjecture, that $\mathsf{runs}(n) < n$, are the most urgent to verify. As a result of this dissertation, we formulate a "cubes" conjecture that $\mathsf{cubes}(n) < 0.5\,n$. Computer experiments of many authors, including ours, suggest validity of these conjectures, combinatorial arguments still need to be found.

We also proposed efficient algorithms for enumeration of repetitions basing on the linear upper bounds. Using the structure of runs in a word novel linear time algorithms for reporting distinct powers and computing local periods in a word were obtained. For the purpose of these algorithms, new structural relations between the notions of periodicity were developed. We believe that other algorithmic applications of runs are still to be discovered.

Bibliography

[1] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *J. Discrete Algorithms*, 2(1):53–86, 2004.

[2] A. Apostolico and Z. Galil, editors. *Combinatorial Algorithms on Words*, volume F12 of *NATO ASI Series*. Springer-Verlag, 1985.

[3] A. Apostolico and F. P. Preparata. Optimal off-line detection of repetitions in a string. *Theor. Comput. Sci.*, 22:297–315, 1983.

[4] P. Baturo, M. Piatkowski, and W. Rytter. The number of runs in Sturmian words. In O. H. Ibarra and B. Ravikumar, editors, *CIAA*, volume 5148 of *Lecture Notes in Computer Science*, pages 252–261. Springer, 2008.

[5] J. Berstel and J. Karhumäki. Combinatorics on words: a tutorial. *Bulletin of the EATCS*, 79:178–228, 2003.

[6] G. Chen, S. J. Puglisi, and W. F. Smyth. Fast and practical algorithms for computing all the runs in a string. In B. Ma and K. Zhang, editors, *CPM*, volume 4580 of *Lecture Notes in Computer Science*, pages 307–315. Springer, 2007.

[7] G.-H. Chen, J.-J. Hong, and H.-I. Lu. An optimal algorithm for online square detection. In A. Apostolico, M. Crochemore, and K. Park, editors, *CPM*, volume 3537 of *Lecture Notes in Computer Science*, pages 280–287. Springer, 2005.

[8] M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Inf. Process. Lett.*, 12(5):244–250, 1981.

[9] M. Crochemore. Transducers and repetitions. *Theor. Comput. Sci.*, 45(1):63–86, 1986.

[10] M. Crochemore, S. Z. Fazekas, C. S. Iliopoulos, and I. Jayasekera. Bounds on powers in strings. In M. Ito and M. Toyama, editors, *Developments in Language Theory*, volume 5257 of *Lecture Notes in Computer Science*, pages 206–215. Springer, 2008.

[11] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007.

[12] M. Crochemore and L. Ilie. Analysis of maximal repetitions in strings. In L. Kucera and A. Kucera, editors, *MFCS*, volume 4708 of *Lecture Notes in Computer Science*, pages 465–476. Springer, 2007.

[13] M. Crochemore and L. Ilie. Maximal repetitions in strings. *J. Comput. Syst. Sci.*, 74(5):796–807, 2008.

[14] M. Crochemore, L. Ilie, and W. Rytter. Repetitions in strings: Algorithms and combinatorics. *Theor. Comput. Sci.*, 410(50):5227–5235, 2009.

[15] M. Crochemore, L. Ilie, and L. Tinta. Towards a solution to the "runs" conjecture. In P. Ferragina and G. M. Landau, editors, *CPM*, volume 5029 of *Lecture Notes in Computer Science*, pages 290–302. Springer, 2008.

[16] M. Crochemore, C. S. Iliopoulos, M. Kubica, J. Radoszewski, W. Rytter, K. Stencel, and T. Walen. New simple efficient algorithms computing powers and runs in strings. In *Proceedings of the 15th Prague Stringology Conference, PSC*, pages 138–149, 2010.

[17] M. Crochemore, C. S. Iliopoulos, M. Kubica, J. Radoszewski, W. Rytter, and T. Walen. Extracting powers and periods in a string from its runs structure. In E. Chávez and S. Lonardi, editors, *SPIRE*, volume 6393 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2010.

[18] M. Crochemore, C. S. Iliopoulos, M. Kubica, J. Radoszewski, W. Rytter, and T. Walen. On the maximal number of cubic runs in a string. In A. H. Dediu, H. Fernau, and C. Martín-Vide, editors, *LATA*, volume 6031 of *Lecture Notes in Computer Science*, pages 227–238. Springer, 2010.

[19] M. Crochemore, C. S. Iliopoulos, M. Kubica, J. Radoszewski, W. Rytter, and T. Walen. The maximal number of cubic runs in a word. *J. Comput. System Sci.*, 2011, doi: 10.1016/j.jcss.2011.12.005.

[20] M. Crochemore, C. S. Iliopoulos, M. Kubica, W. Rytter, and T. Walen. Efficient algorithms for two extensions of LPF table: The power of suffix arrays. In J. van Leeuwen, A. Muscholl, D. Peleg, J. Pokorný, and B. Rumpe, editors, *SOFSEM*, volume 5901 of *Lecture Notes in Computer Science*, pages 296–307. Springer, 2010.

[21] M. Crochemore, M. Kubica, J. Radoszewski, W. Rytter, and T. Walen. On the maximal sum of exponents of runs in a string. In C. S. Iliopoulos and W. F. Smyth, editors, *IWOCA*, volume 6460 of *Lecture Notes in Computer Science*, pages 10–19. Springer, 2010.

[22] M. Crochemore, M. Kubica, J. Radoszewski, W. Rytter, and T. Walen. On the maximal sum of exponents of runs in a string. *Journal of Discrete Algorithms*, 2011, doi: 10.1016/j.jda.2011.12.016.

[23] M. Crochemore and W. Rytter. *Jewels of Stringology.* World Scientific, 2003.

[24] D. Damanik and D. Lenz. Powers in Sturmian sequences. *Eur. J. Comb.*, 24(4):377–390, 2003.

[25] A. Deza, F. Franek, and M. Jiang. A *d*-step approach for distinct squares in strings. In R. Giancarlo and G. Manzini, editors, *CPM*, volume 6661 of *Lecture Notes in Computer Science*, pages 77–89. Springer, 2011.

[26] J.-P. Duval, R. Kolpakov, G. Kucherov, T. Lecroq, and A. Lefebvre. Linear-time computation of local periods. *Theor. Comput. Sci.*, 326(1-3):229–240, 2004.

[27] K. Fan, S. J. Puglisi, W. F. Smyth, and A. Turpin. A new periodicity lemma. *SIAM J. Discrete Math.*, 20(3):656–668, 2006.

[28] M. Farach. Optimal suffix tree construction with large alphabets. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 137–143, 1997.

[29] N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16:109–114, 1965.

[30] J. Fischer and V. Heun. A new succinct representation of RMQ-information and improvements in the enhanced suffix array. In B. Chen, M. Paterson, and G. Zhang, editors, *ESCAPE*, volume 4614 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2007.

[31] A. S. Fraenkel and J. Simpson. How many squares can a string contain? *J. of Combinatorial Theory Series A*, 82:112–120, 1998.

[32] A. S. Fraenkel and J. Simpson. The exact number of squares in Fibonacci words. *Theor. Comput. Sci.*, 218(1):95–106, 1999.

[33] F. Franek, A. Karaman, and W. Smyth. Repetitions in Sturmian strings. *Theoretical Computer Science*, 249(2):289 – 303, 2000.

[34] F. Franek and Q. Yang. An asymptotic lower bound for the maximal number of runs in a string. *Int. J. Found. Comput. Sci.*, 19(1):195–203, 2008.

[35] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 246–251, 1983.

[36] M. Giraud. Not so many runs in strings. In C. Martín-Vide, F. Otto, and H. Fernau, editors, *LATA*, volume 5196 of *Lecture Notes in Computer Science*, pages 232–239. Springer, 2008.

[37] D. Gusfield and J. Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *J. Comput. Syst. Sci.*, 69(4):525–546, 2004.

[38] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[39] L. Ilie. A simple proof that a word of length n has at most $2n$ distinct squares. *J. of Combinatorial Theory Series A*, 112:163–164, 2005.

[40] L. Ilie. A note on the number of squares in a word. *Theoretical Computer Science*, 380:373–376, 2007.

[41] L. Ilie and L. Tinta. Practical algorithms for the longest common extension problem. In J. Karlgren, J. Tarhio, and H. Hyyrö, editors, *SPIRE*, volume 5721 of *Lecture Notes in Computer Science*, pages 302–309. Springer, 2009.

[42] C. S. Iliopoulos, D. Moore, and W. F. Smyth. A characterization of the squares in a Fibonacci string. *Theor. Comput. Sci.*, 172(1-2):281–291, 1997.

[43] A. Itai. Linear time restricted union/find. `http://www.cs.technion.ac.il/~itai/Courses/ds2/lectures/lecture.html`, 2006.

[44] J. Jansson and Z. Peng. Online and dynamic recognition of squarefree strings. In J. Jedrzejowicz and A. Szepietowski, editors, *MFCS*, volume 3618 of *Lecture Notes in Computer Science*, pages 520–531. Springer, 2005.

[45] J. Kärkkäinen and P. Sanders. Simple linear work suffix array construction. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 943–955. Springer, 2003.

[46] P. Ko and S. Aluru. Space efficient linear time construction of suffix arrays. In R. A. Baeza-Yates, E. Chávez, and M. Crochemore, editors, *CPM*, volume 2676 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2003.

[47] R. Kolpakov, G. Kucherov, and P. Ochem. On maximal repetitions of arbitrary exponent. *Inf. Process. Lett.*, 110(7):252–256, 2010.

[48] R. M. Kolpakov and G. Kucherov. Finding maximal repetitions in a word in linear time. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 596–604, 1999.

[49] R. M. Kolpakov and G. Kucherov. On maximal repetitions in words. *Journal of Discrete Algorithms*, 1:159–186, 1999.

[50] R. M. Kolpakov and G. Kucherov. On the sum of exponents of maximal repetitions in a word. *Tech. Report 99-R-034, LORIA*, 1999.

[51] S. R. Kosaraju. Computation of squares in a string (preliminary version). In M. Crochemore and D. Gusfield, editors, *CPM*, volume 807 of *Lecture Notes in Computer Science*, pages 146–150. Springer, 1994.

[52] M. Kubica, J. Radoszewski, W. Rytter, and T. Walen. On the maximal number of cubic subwords in a string. In J. Fiala, J. Kratochvíl, and M. Miller, editors, *IWOCA*, volume 5874 of *Lecture Notes in Computer Science*, pages 345–355. Springer, 2009.

[53] K. Kusano, W. Matsubara, A. Ishino, H. Bannai, and A. Shinohara. New lower bounds for the maximum number of runs in a string. *CoRR*, abs/0804.1214, 2008.

[54] G. M. Landau and J. P. Schmidt. An algorithm for approximate tandem repeats. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *CPM*, volume 684 of *Lecture Notes in Computer Science*, pages 120–133. Springer, 1993.

[55] A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976.

[56] H.-F. Leung, Z. Peng, and H.-F. Ting. An efficient online algorithm for square detection. In K.-Y. Chwa and J. I. Munro, editors, *COCOON*, volume 3106 of *Lecture Notes in Computer Science*, pages 432–439. Springer, 2004.

[57] M. Lothaire. *Combinatorics on Words*. Addison-Wesley, Reading, MA., U.S.A., 1983.

[58] M. G. Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25(1-2):145–153, 1989.

[59] M. G. Main and R. J. Lorentz. An O(n log n) algorithm for finding all repetitions in a string. *J. Algorithms*, 5(3):422–432, 1984.

[60] M. G. Main and R. J. Lorentz. Linear time recognition of squarefree strings. In Apostolico and Galil [2], pages 271–278.

[61] F. Mignosi and G. Pirillo. Repetitions in the Fibonacci infinite word. *ITA*, 26:199–204, 1992.

[62] M. Piatkowski. *Efficient algorithms related to combinatorial structure of words*. PhD thesis, Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, 2011.

[63] M. Piatkowski and W. Rytter. Asymptotic behaviour of the maximal number of squares in standard Sturmian words. In *Proceedings of the Prague Stringology Conference, PSC*, pages 237–248, 2009.

[64] M. Piatkowski and W. Rytter. Computing the number of cubic runs in standard Sturmian words. In *Proceedings of the Prague Stringology Conference, PSC*, pages 106–120, 2011.

[65] S. J. Puglisi, J. Simpson, and W. F. Smyth. How many runs can a string contain? *Theor. Comput. Sci.*, 401(1-3):165–171, 2008.

[66] M. O. Rabin. Discovering repetitions in strings. In Apostolico and Galil [2], pages 279–288.

[67] W. Rytter. The number of runs in a string: Improved analysis of the linear upper bound. In B. Durand and W. Thomas, editors, *STACS*, volume 3884 of *Lecture Notes in Computer Science*, pages 184–195. Springer, 2006.

[68] W. Rytter. The structure of subword graphs and suffix trees in Fibonacci words. *Theor. Comput. Sci.*, 363(2):211–223, 2006.

[69] W. Rytter. The number of runs in a string. *Inf. Comput.*, 205(9):1459–1469, 2007.

[70] J. Simpson. Modified Padovan words and the maximum number of runs in a word. *Australasian J. of Comb.*, 46:129–145, 2010.

[71] J. Stoye and D. Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree. *Theor. Comput. Sci.*, 270(1-2):843–856, 2002.

[72] A. Thue. Uber unendliche Zeichenreihen. *Norske Vid. Selsk. Skr. I Math-Nat.*, 7:1–22, 1906.