University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Wiktor Zuba

Efficient enumerations in words

PhD dissertation

Supervisor:
prof. dr hab. Wojciech Rytter
Institute of Informatics
University of Warsaw

June 2021

Author's declaration:
I hereby declare that this dissertation is my own work.

June 8, 2021

Wiktor Zuba
.....
Wiktor Zuba

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

June 8, 2021

W Rytter
.....
Wojciech Rytter

Abstract

The dissertation presented here consists of a set of results in text processing algorithms from four different scientific publications. The papers focus on algorithms and data structures that allow efficient enumeration and counting of certain classes of factors. Most of the results are based on different variations of the same key property - regularity of the set of occurrences of factors of specific types.

In "Efficient Representation and Counting of Antipower Factors in Words" we focused on a class of k-antipowers (words consisting of k pairwise different segments of the same length). The results of the paper consist of effective algorithms for enumerating all occurrences, counting all occurrences and counting different k-antipower factors in a string, working in $O(nk \log k + \text{output})$, $O(nk \log k)$ and $O(nk^4 \log k \log n)$ time, respectively.

The paper "Counting Distinct Patterns in Internal Dictionary Matching" deals with a problem in which we are given an input word T and its internal dictionary D. We want to answer questions of the form: "How many distinct words from D are the factors of $T[i..j]$?". We have shown few algorithms and data structures, that allows us to do that effectively. The algorithms differ in exploited properties and complexities (each one can be more effective from the other depending on the ratio between T and D).

In "The Number of Repetitions in 2D-Strings" we focused on the extensions of the notions of squares and runs to the case of two-dimensional strings. We gave new bounds on the maximal number of their occurrences in strings and effective algorithms for enumerating them.

In the fourth paper titled "Efficient Enumeration of Distinct Factors Using Package Representations", we proposed a new compact representation of string factors. It allows us to obtain new effective algorithms for enumeration and counting of distinct factors it contains. In particular it allowed us to obtain a new simple algorithm for finding distinct squares and a faster algorithm for finding all distinct antipowers in a string (improving one of the results from the first of enclosed papers).

Keywords: algorithm, data structure, enumeration, counting, factor, subword, antipower, runs, two-dimensional string, internal dictionary

Efektywne zliczanie w słowach.

Streszczenie

Przedstawiona rozprawa stanowi zbiór wyników prac na temat algorytmów przetwarzających dane tekstowe. Prace skupiają się na zaprezentowaniu algorytmów i struktur danych pozwalających na efektywne znajdowanie i zliczanie podsłów należących do konkretnych klas podsłów. Innym elementem wspólnym poniższych prac jest to, że wykorzystują one regularności w słowach związane z wystąpieniami podsłów szczególnego typu.

W pracy "Efficient Representation and Counting of Antipower Factors in Words" zajęliśmy się klasą k-antypotęg (słów składających się z k parami różnych członów o tej samej długości). Rezultatem pracy są efektywne algorytmy wypisywania wszystkich wystąpień, zliczania wszystkich wystąpień oraz zliczania różnych k-antypotęg w słowie, działające w czasach odpowiednio $O(nk \log k +$ wynik), $O(nk \log k)$ i $O(nk^4 \log k \log n)$.

Praca "Counting Distinct Patterns in Internal Dictionary Matching" przedstawia problem w którym mając dane słowo wejściowe T oraz słownik jego podsłów D chcielibyśmy móc efektywnie odpowiadać na pytania "Ile różnych słów ze słownika D jest podsłowami słowa $T[i\,..\,j]$?". Przedstawiliśmy kilka algorytmów/struktur danych, które pozwalają na rozwiązanie tego problemu. Algorytmy różnią się wykorzystywanymi założeniami oraz złożonościami (każdy może być lepszy od pozostałych w zależności od zastosowania, oraz stosunku wielkości T i D).

W pracy "The Number of Repetitions in 2D-Strings" zajęliśmy się uogólnieniami pojęć kwadratów i maksymalnych powtórzeń na teksty dwuwymiarowe. Podaliśmy nowe ograniczenia na maksymalną ich ilość w tekstach, oraz efektywne algorytmy ich wyznaczania.

Na końcu przedstawiam pracę "Efficient Enumeration of Distinct Factors Using Package Representations", w której wprowadziliśmy nowy, skompresowany sposób reprezentacji wystąpień wielu podsłów. Pozwala ona na otrzymanie nowych, efektywnych algorytmów wyznaczania i zliczania różnych słów przez nią reprezentowanych. W szczególności pozwoliło nam to na otrzymanie nowego prostego algorytmu wyznaczania kwadratów i szybszego algorytmu wyznaczania antypotęg (poprawiając jeden z wyników pierwszej z przedstawionych prac).

Słowa kluczowe: algorytm, struktura danych, wyliczanie, zliczanie, podsłowo, antypotęga, maksymalne powtórzenie, słowo dwuwymiarowe, słownik wewnętrzny

Contents

Chapter 1

Extended abstract

1.1 Introduction

1.1.1 Motivation

The study of regular words and especially of algorithms finding and counting such words in texts is one of the most popular areas of text algorithms. There are many algorithms, which work differently on periodic and aperiodic parts of strings, as such parts may show distinct properties. For example, high periodicity of a word results in a very good compression rate - we can describe it using only its period and length. On the other hand it may make searching (for it or in it) much more difficult, as an aperiodic word of length m may occur in a word of length n at most $2\frac{n}{m}$ times. That is not true if the word is periodic - for instance a word $aaaaaa$ appears in $aaaaaaaaaaaaaaaa$ $n - m + 1 = 11$ times. The number of such (distinct) structures in a word can by itself serve as a good measure of the words regularity or complexity.

The presented dissertation is composed of publications in which (together with my coauthors) we have concerned a few classes of such regular (or irregular in a specific sense) subwords. In each of the papers we have shown new, effective algorithms for enumerating and counting (all occurrences or all distinct) factors from a specific class.

1.1.2 Contents

The dissertation is composed of four articles written together with my coauthors during the course of my doctoral studies at the Faculty of Mathematics, Informatics and Mechanics of the University of Warsaw:

1. Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszyński, Tomasz Waleń, Wiktor Zuba: *E cient Representation and Counting of Antipower Factors in Words.* LATA 2019: 421-433

The paper is included in the version accepted for publication in the Special Issue of Information & Computation derived from LATA 2019 conference.

2. Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszyński, Tomasz Waleń, Wiktor Zuba: *Counting Distinct Patterns in Internal Dictionary Matching.* CPM 2020: 8:1-8:15

3. Panagiotis Charalampopoulos, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, Wiktor Zuba: *The Number of Repetitions in 2D-Strings.* ESA 2020: 32:1-32:18

4. Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, Wiktor Zuba: *E cient Enumeration of Distinct Factors Using Package Representations.* SPIRE 2020: 247-261

All those papers contain original results in text algorithms. They were written by the authors representing the University of Warsaw together with scientists from King's College London (Panagiotis Charalampopoulos had a long term stay in Warsaw) and Tomasz Kociumaka, who changed his affiliation from the University of Warsaw to Bar-Ilan University in the meantime.

In section 1.2 of this abstract I introduce the basic notions - regular classes of words and factors, which are used throughout the presented papers.

In section 1.3 I show a detailed overview of our results together with sketches of methods used to obtain them.

In section 1.4 I list the publications I coauthored during the course of my doctoral studies, which however are not included as a part of this dissertation.

Chapter 2 contains a polish version of this extended abstract, while chapters 3, 4, 5 and 6 contain the papers included in the dissertation. The bibliography at the end contains references to papers cited in this abstract.

1.2 Preliminaries

To present the results we need to define the string regularities they use or search for first. Many of them are used by more then just one of the papers included in this dissertation.

Definition 1.1. Let us assume that $x = y_0 y_1 \cdots y_{k-1}$ where $k \quad 2$ and y_i are words of length d. We say that:

- x is a k-**power** if all y_i's are the same (x is a **square** if in addition $k = 2$);
- x is a k-**antipower** if all y_i's are pairwise distinct;
- x is a **weak** k-**power** if it is not a k-antipower, that is, if $y_i = y_j$ for some $i = j$;
- x is a **gapped square** if $y_0 = y_{k-1}$.

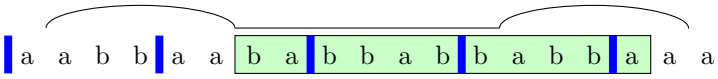


Figure 1.1: Regularities in standard strings. A run (with the period equal to 3) is depicted by a green background. Blue lines mark an occurrence of a 4-antipower. Notice that the words of length 16 starting one or two positions later are not 4-antipowers as they are gapped squares generated by a maximal 3-gapped repeat.

Definition 1.2. We say that:

- p is a **period** of a word w if $w[i] = w[i+p]$ for all i $[0\,..\,|w|-p)$ (by the period of a word we usually mean the shortest one);
- a fragment $w[i\,..\,j]$ is a **run** (a maximal repetition) if its length is equal to at least a double of its shortest period and it cannot be extended by a one position to the left or to the right without breaking the period;
- a fragment $w[i\,..\,j]$ is a **maximal α-gapped repeat** (for α 1) if it is of a form uvu for $p = |uv|$ $\alpha|u|$ and it cannot be extended by a one position to the left or to the right without breaking the period.

Definition 1.3. For two-dimensional texts we define repetitions analogously.

- A **tandem** is a two-dimensional text of even width, in which its left and right halves are equal as texts (extension of square to the case where instead of letters we have columns of letters).
- A **quartic** is a two-dimensional text Q of even height and width, such that Q and Q are tandems (it is composed of four identical texts in a 2×2 grid).
- A **2D-run** is a $i \times j$ fragment of a text such that its vertical period is smaller or equal to $\frac{i}{2}$ and its horizontal period is smaller or equal to $\frac{j}{2}$. Moreover it cannot be extended by a row or a column from any side (if such exist in the text) without breaking any of those periods.

By a square generated by a run I mean a square fully contained in it with length equal to exactly double of its period. All such squares can be easily obtained from a run, and every square is generated by a run. The generation of gapped-squares by runs and maximal α-gapped repeats and the generation of quartics by 2D-runs work analogously.

A square is primitively rooted if its root (half) is not a k-power for any $k > 1$. Analogously a quartic is primitively rooted if its root (quarter) is not a k-power for any $k > 1$ neither horizontally nor diagonally.

Definition 1.4. By an **internal dictionary** we mean a set of factors of a given word represented by the starting and ending positions of a single occurrence of each factor.

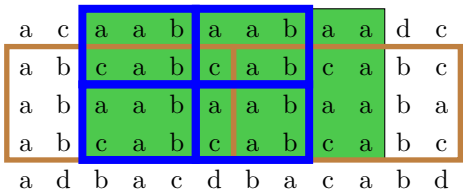


Figure 1.2: Different types of repetitions in 2D-strings. A tandem is marked by brown rims, a quartic by the blue ones, while the green background depicts a 2D-run.

1.3 Presentation of the main results

1.3.1 E□cient Representation and Counting of Antipower Factors in Words

The paper presented here was presented on LATA 2019 conference and published in conference proceedings. Extended version of that paper was later accepted for publication (it is not published yet) in the Special Issue of Information & Computation journal derived from that conference. It is included in this dissertation in the mentioned extended version (in chapter 3).

Introduction

Badkobeh et al. in [5] presented a lower and upper bound $\Theta(n^2/k)$ for the maximal number of occurrences of k-antipowers in a string. Furthermore they presented an $O(n^2/k)$ time algorithm that finds all those occurrences. Due to the lower bound on the maximal size of the output the algorithm is optimal in the pessimistic case. It does not mean however, that nothing more in this field can be done.

In our publication we have focused on the problems that are not directly influenced by those bounds - counting of antipowers and enumerating them with complexity parameterized by the size of the output (more effective if a word contains a small number of antipowers).

More formally - we presented algorithms, which for a word of length n count all the occurrences of k-antipower factors in $O(nk \log k)$ time, find all those occurrences in $O(nk \log k + \text{output})$ time, and count all distinct k-antipowers in a word in $O(nk^4 \log k \log n)$ time. We also presented a data structure that allows us to check efficiently if a given subword is a k-antipower.

In (the most common) case when k is small in comparison to the length of the word (smaller than $\sqrt{n}$) our algorithm has a better asymptotic performance time than the one previously known.

Weak powers and a compact representation

Since antipowers itself are very irregular it is not that easy to effectively find them. Due to that, we focused on computing a representation of all the other

words, that is of all weak k-powers. Every weak k-power on the other hand is a consequence of the existence of a gapped square (possibly for a smaller k, but certainly for the same d).

Observation 1.1.

(a) Each (gapped) square is generated by a run with (not necessarily shortest) period $(q+1) \cdot d$ or a maximal $(q+1)$-gapped repeat with a period equal to $(q+1) \cdot d$ for some $q \quad k-2$.

(b) Each run and each maximal α-gapped repeat generate a single interval of positions in which (gapped) squares of a given length start, moreover this interval can be computed in constant time.

(c) In a word of length n there are at most $O(n)$ general runs and their representation can be computed in $O(n)$ time ([6]).

(d) There are only $O(n\alpha)$ maximal α-gapped repeats in a word and their representation can be computed in $O(n\alpha)$ time ([11, 12]).

With the use of those properties we can compute the representation of weak k-powers for all d's as $O(nk)$ interval chains (set of intervals of the same lengths and beginnings forming an arithmetic progression) in total. Then, using geometric methods we can determine their set-theoretic sum in $O(nk \log k)$ total time. This immediately gives us the number of all weak k-powers, hence also of all k-antipower factors. Instead of counting the size of the complement of this set we can enumerate through all of the positions it does not contain and thus obtain an algorithm that finds all k-antipowers in $O(nk \log k + \text{output})$ time.

Antipower queries

Paper [5] included descriptions of two data structures, that after being constructed answer queries "Is a subword $w[i..j]$ a k-antipower?". The first structure requires $O(n)$ space and answers those questions in $O(k)$ time, while the other answers them in a constant time, but requires storing information of $O(n^2)$ size.

In our paper we have shown a new structure parameterized by a value r $[1, n]$. After being built in $O(n^2/r)$ time our structure stores information of size $O(n^2/r)$ and answers such questions in $O(r)$ time.

If $r < k$ then we know that the base of the power d equals at most $n/r > n/k$. To answer the questions we store for each $d \quad n/r$ separately a data structure based on the range minimum queries structure, that is of $O(n)$ size and answers questions in constant time. Otherwise $(r \quad k)$ to obtain the right complexities it is enough to just use the mentioned data structure from [5].

Counting distinct k-antipowers

In the journal version of the paper we have added a new result - an algorithm for counting distinct k-antipowers in a word.

Due to a potentially large number of all occurrences of antipowers, the methods enumerating through those cannot obtain a satisfactory computation time.

To obtain such a result we made use of the connection between antipowers and weak powers once again.

The number of distinct k-antipower factors can be obtained as a subtraction of two other values. The first one is equal to the number of (all) distinct factors of the word of length divisible by k. We can compute it in $O(n)$ time with the use of a suffix tree.

The second value needed is the number of distinct weak k-powers. To obtain that we have strengthened the definition of weak powers, so that each such subword can be generated only in one way. Then with the use of a reduction to a graph problem we have acquired an algorithm computing that number in $O(nk^4 \log k \log n)$ total time.

1.3.2 Counting Distinct Patterns in Internal Dictionary Matching

In [8] my coauthors introduced a problem of pattern matching with internal dictionary - for a given word T and its internal dictionary D they presented a data structure that allows effective answering of questions for subwords $T[i..j]$:

- does $T[i..j]$ contain a word from the dictionary?
- return all subwords of $T[i..j]$ that belong to the dictionary
- return all distinct factors from D contained in $T[i..j]$
- count all subwords of $T[i..j]$ that belong to the dictionary
- count the number of distinct factors from D contained in $T[i..j]$

The structure requires $\tilde{O}(n+d)$ space and construction time (where $n = |T|$ denotes the length of the word, $d = |D|$ the number of factors contained in the dictionary, and $\tilde{O}$ is the asymptotic notation ignoring polilogarithmic factors). It answers the first four questions in $\tilde{O}(1 + \text{output})$ time.

Unfortunately for the last question only a data structure answering those question $O(\log n)$ approximately was shown.

Space	Preprocessing time	Query time	Variant
$\tilde{O}(n+d)$	$\tilde{O}(n+d)$	$\tilde{O}(1)$	2-approximation
$\tilde{O}(n^2/m^2+d)$	$\tilde{O}(n^2/m+d)$	$\tilde{O}(m)$	exact
$\tilde{O}(nd/m+d)$	$\tilde{O}(nd/m+d)$	$\tilde{O}(m)$	exact
$O(n\log^2 n)$	$O(n\log^2 n)$	$O(\log n)$	D = squares, exact

Table 1.1: Our results for Count Distinct queries ($m \quad [1,n]$ is an arbitrarily chosen parameter).

2-approximation

To improve the result from the previous paper we have designed a data structure which stores all the exact results for the base words - that is for all subwords of length $(1+\delta)^p$ for all natural p's and a fixed value $\delta = \frac{1}{9}$. To obtain

the result for any given subword $T[i..j]$ we make use of two maximal length base words $T[i..i']$ and $T[j'..j]$ contained in $T[i..j]$ to obtain its division into three parts $F_1F_2F_3$ ($F_1F_2 = T[i..i']$, $F_2F_3 = T[j'..j]$).

The result is obtained by counting all the factors from the dictionary, which have an occurrence starting in F_1 and ending in F_3, which at the same time occurs in neither of the two base words. We can do that effectively thanks to a large ratio of the length of F_2 to the lengths of F_1 and F_3. To the obtained value we add the results stored for the two base words, which however results in only a 2-approximate result.

There are $O(n \log n)$ base words in a word of length n. The exact values for them can be counted in $O(n \log^{1+\varepsilon} n + d)$ total time (for any constant $\varepsilon > 0$) with the use of the property that we can efficiently update the result after a single position shift. The structure used to count the words described earlier costs us additional $O(n + d \log n)$ space and $O(n \log n / \log\log n + d \log^{3/2} n)$ construction time. It answers the questions in $O(\log^2 n / \log\log n)$ time.

Exact counting

Instead of the base words we can focus on storing the results for all the words of the form $T[c_1 m + 1..c_2 m]$ for a chosen m and all the values $c_1 < c_2$ to obtain a data structure of $O(n^2/m^2 + n + d)$ size construable in $O((n^2 \log n)/m + n\sqrt{\log n} + d)$ time. We can update the result obtained after extending a subword by a one position in $O(\log n)$ time, and to obtain the exact result for any chosen factor we only require up to $O(m)$ such operations (we extend one of the words for which the result is stored).

We can also approach the problem differently, namely, we can inspect the structure of the dictionary.

If the dictionary does not contain a set of k distinct prefixes of the same word, then at any given position only up to $k - 1$ dictionary words can start. In this case we can store for each of the dictionary words the set of positions in T, where such a word occurs. We can construct such a data structure of $O(nk \log n)$ size in $O(nk \log n)$ time. In return, it allows us to respond to our queries in $O(\log n)$ time.

To obtain a dictionary of such a form we can simply remove from it all such large groups of prefixes (building new dictionaries). There are at most d/k such groups, and for each of those we can answer the queries in $O(\log n)$ time (for any $\varepsilon > 0$) using a bounded LCP structure ([14]) of $O(n\sqrt{\log n})$ size.

For $k = \frac{d}{m}$ this gives us a query time equal to $O(m \log n + \log n)$.

Counting distinct squares factors

For the popular case, where we want to count the number of distinct square factors in a given subword we have described a yet different algorithm.

In a word of length n there can be up to $O(n^2)$ squares (only $O(n)$ distinct ones), however thanks to the runs (only $O(n)$ such can appear in the word, and all such can be computed in $O(n)$ time) we can distinguish $m = O(n \log n)$ occurrences which really affect our results. With the use of a geometrical data

structure of $O(m \log m)$ size and construction time from [13] we can answer the queries for the number of distinct squares in a subword in $O(\log m) = O(\log n)$ time.

1.3.3 The Number of Repetitions in 2D-Strings

In the paper from chapter 5 we have focused on the repetitions in two-dimensional texts. The table 1.2 contains a summary of the current knowledge about the maximal possible number of 2D-runs, tandems and quartics (distinct or all occurrences of primitively rooted ones) in a two-dimensional string. It also shows the time complexity of the fastest known algorithms which find those structures in a text. Notice, that even though the maximal number of 2D-runs in a string was previously studied the gap between the known lower and upper bounds was linear. In our paper we have reduced this gap to a polilogarithmic one.

	Bounds on the number in an $n \times n$ string	Computation time
2D-runs	$\Omega(n^2), O(n^3)$[2] $O(n^2 \log^2 n)$	$O(n^2 \log n$+output)[2] $O(n^2 \log^2 n)$
Occurrences of primitively rooted quartics	$\Omega(n^2 \log^2 n)$ [3]	$O(n^2 \log^2 n)$
Distinct quartics	$O(n^2 \log^2 n)$	$O(n^2 \log^2 n)$
Occurrences of primitively rooted tandems	$\Omega(n^3 \log n)$[3]	$O(n^3 \log n)$[4]
Distinct tandems	$\Omega(n^2); O(n^4)$(trivial) $\Omega(n^3)$	$O(n^3)$

Table 1.2: Overview of the previous knowledge and of our results (written in bold).

2D-runs

To obtain our upper bound on the number of 2D-runs we made use of a lemma from [1], to assign to every such 2D-run a maximal horizontal repetition (no condition on vertical periodicity or maximality) of height equal to a power of two (largest possible not exceeding the height of the 2D-run). Such repetition meets three important conditions as well - its upper left or lower left corner equals the respective corner of the 2D-run, its horizontal period is equal to the horizontal period of the 2D-run, and its width equals at least the width of the 2D-run (see figure 1.3)

With the use of those properties, of the periodicity lemma, and of the three squares lemma we proved that the horizontal repetitions occupying the rows from i to $i + 2^k - 1$ can be assigned to at most $O(n \log n)$ distinct 2D-runs in total. By multiplying this value by the number of feasible choices of i and k we

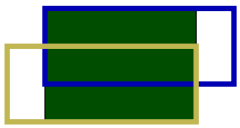


Figure 1.3: Assignment of a maximal horizontal repetition of height 2^k to a 2D-run.

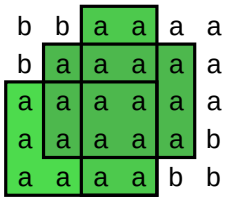


Figure 1.4: Many overlapping 2D-runs

obtain an upper bound on the number of 2D-runs in an $n \times n$ string equal to $O(n^2 \log^2 n)$.

Our bound immediately shows that the algorithm finding all the 2D-runs in a 2D-string from [2] works in $O(n^2 \log^2 n)$ time (their algorithm has time complexity parameterized by the size of the output).

Primitively rooted quartics

In standard strings we can easily extract all the occurrences of primitively rooted squares from the runs - it is enough to take all the subwords of the runs of length equal to the double of their period.

In 2D situation it is somewhat similar - each primitively rooted quartic is a rectangle of width equal to the double of the horizontal period and height equal to the double of the vertical period of a 2D-run, fully contained in it. However, while in one dimension two runs with the same period cannot have a big enough overlap to generate the same square, in 2D many such overlaps can easily appear (see figure 1.4), hence simple adaptation of such a one-dimensional algorithm would result in reporting some of the quartics multiple times.

This problem can be solved with the use of a sweeping line technique ([7]), which allows us to count set-theoretic sums of many families of rectangles on an $n \times n$ grid in $O(n + r + \text{output})$ total time (r denotes the number of all the rectangles).

By dividing all the 2D-runs according to their periods, for each such group of runs we count the set-theoretic sum of the rectangles containing the starting positions of quartics generated by each 2D-run. This way we obtain the desired algorithm.

Due to our bound on r from the previous paragraph (the number of 2D-runs), and the $O(n^2 \log^2 n)$ bound on the number of the occurrences of primitively rooted quartics from [3] our algorithm runs in $O(n^2 \log^2 n)$ total time (optimal

in the pessimistic case by the mentioned bound).

Distinct quartics

In an $n \times n$ unary string there are $\Theta(n^4)$ occurrences of general quartics. Due to that in the pessimistic case a trivial algorithm finding all such occurrences in an optimal time. A more interesting problems are the one of bounding the maximal number of distinct quartics (differing as texts and not only by their positioning) and the one of enumerating through all of them.

[3] already gave us a bound on the number of distinct primitively rooted quartics. All the other ones are composed of many adjacent occurrences of the primitively rooted ones forming a larger rectangle. In other words, while a primitively rooted quartic is constructed from occurrences of a primitive text W arranged in a 2×2 grid, in the case of the other ones it is a $k \times l$ grid ($k > 1$ or $l > 1$). We divide such quartics into the thin ones ($k = 1$ or $l = 1$), and the thick ones (both $k, l > 1$).

Then again we perform a yet another division - we group quartics for which their primitive string W has size in $[2^a, 2^{a+1}) \times [2^b, 2^{b+1})$ for the same values $a, b \in \log n$.

Due to the periodicity lemma and the three squares lemma any position can be the top left corner of the rightmost occurrence (same quartic does not occur anywhere further to the right) of at most four distinct thin quartics from a single group (it works analogously to a proof of a bound of the number of distinct squares in a standard string [10]).

In the case of the thick quartics the multitude of the grid points in which an occurrence of the W string begins in a large power of W allows us to uniquely assign such points to every smaller power of W. Moreover, due to the periodicity and the three squares lemmas such points cannot belong to a grid for a power of a different string W' which belongs to the same group.

Due to this assignments of the points in $[1, n] \times [1, n]$ to the quartics and due to the fact, that there are only $\log^2 n$ choices of a and b we obtain an upper bound on the number of distinct quartics of size $O(n^2 \log^2 n)$.

To compute all the distinct quartics in a text we use our previous algorithm to find all the occurrences of primitively rooted quartics, and then we group them by their roots.

Since other quartics are constructed from the occurrences of the primitive ones (with the same root) we can distinguish the top left corners of all such occurrences, and then find the largest possible grids they form.

This problem can be solved with the sweep line approach to obtain an algorithm working in $O(n^2 \log^2 n)$ time.

Distinct tandems

In the same paper we also considered the related problem of finding all the distinct tandems in a string (as with the quartics, there can be $\Theta(n^4)$ occurrences of all the tandems in a 2D-string).

In this problem we can easily identify the tandems occupying the rows from i to j with squares in a standard string by assigning single letters to all the columns of height $j - i + 1$. In a standard word of length n there can be up to $\Theta(n)$ distinct squares, hence the same bound applies to the number of distinct tandems occupying the distinguished rows. Multiplying this value by a number of possible choices of i and j we obtain $O(n^3)$ as the upper bound on the number of distinct tandems.

$\Theta(n^3)$ as a lower bound can be shown just as easy - it is obtained for a text in which every row is a different unary word.

The assignment of standard letters to the columns of height equal to $j - i + 1$ can be performed constructively. Thanks to that, we can obtain an $O(n^3)$ time algorithm for finding all distinct tandems in a 2D-string, with the use of an algorithm finding distinct squares in a standard string. Due to our lower bound on the maximal size of the output our algorithm is optimal in the pessimistic case.

1.3.4 Efficient Enumeration of Distinct Factors Using Package Representations

In the last of the described papers we introduced a new representation of sets of subwords, which in many cases gives a compact representation, and at the same time allows effective operations on the set.

By a package we mean a set of subwords of the same length, whose starting positions form an interval. Our representation is composed of many such packages represented as triples (starting position, length of words, length of interval). Our greatest interest in this representation lies in the property, that the set of the distinct factors it represents can be found efficiently.

More formally, we are interested in the set

$$Factors(F) = \{T[j..j + l) : j \in [i, i + k] \text{ and } (i, l, k) \in F\}.$$

There are many families of subwords, which can be effectively represented in this way - for example k-powers (when each package corresponds to a run) and k-antipowers (when we can use the representation from section 1.3.1). The obtained representations have size and computation time equal to $O(n)$ and $O(nk^2)$ respectively (each interval chain can be represented by at most k ordinary intervals).

To enumerate through $Factors(F)$, and compute $|Factors(F)|$ most effectively we consider two cases. The simpler one, called special, is the one in which if a factor belongs to $Factors(F)$, then each of its occurrences must be contained in a package from F (it cannot be that some occurrences are represented, and some are not).

In this case for a word of length n and F composed of m packages we make use of a data structure for the longest previous factors ([9]) to obtain efficient enumeration and counting of the result in $O(n + m + output)$ and $O(n + m)$ time respectively. In the case of aforementioned distinct k-powers we obtain a

new, simple algorithm computing them in $O(n)$ time, and in the case of distinct k-antipowers we obtain a new algorithm working in $O(nk^2 + \text{output})$ time. This means, that the newly obtained algorithm has a better performance time than the algorithm described in section 1.3.1, which counts those antipower factors in $O(nk^4 \log k \log n)$ time, and is much more complicated.

The second case, called general does not require this extra assumption, which however makes solving it much more difficult. To do that we adapt a method we used in the paper about k-antipowers (counting of distinct such factors) to obtain algorithms finding and counting Factors(F) in $O(n \log^2 n + m \log n + \text{output})$ and $O(n \log^2 n + m \log n)$ time respectively.

1.4 Other publications

During the course of my doctoral studies several other papers coauthored by me have been published. I decided not to include them in this dissertation, since they are not directly connected to its topic.

1. Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Faster Recovery of Approximate Periods over Edit Distance. SPIRE 2018: 233-240

2. Wojciech Rytter, Wiktor Zuba: Syntactic View of Sigma-Tau Generation of Permutations. LATA 2019: 447-459
   Extended version of the paper was accepted for publication in Theoretical Computer Science journal.

3. Mai Alzamel, Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Quasi-Linear-Time Algorithm for Longest Common Circular Factor. CPM 2019: 25:1-25:14

4. Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Circular Pattern Matching with k Mismatches. FCT 2019: 213-228

5. Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Weighted Shortest Common Supersequence Problem Revisited. SPIRE 2019: 221-238

6. Maxime Crochemore, Costas S. Iliopoulos, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Shortest Covers of All Cyclic Shifts of a String. WALCOM 2020: 69-80

7. Panagiotis Charalampopoulos, Solon P. Pissis, Jakub Radoszewski, Tomasz Walen, Wiktor Zuba: Unary Words Have the Smallest Levenshtein k-Neighbourhoods. CPM 2020: 10:1-10:12

8. Maxime Crochemore, Costas S. Iliopoulos, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Internal Quasiperiod Queries. SPIRE 2020: 60-75

9. Maxime Crochemore, Costas S. Iliopoulos, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Shortest covers of all cyclic shifts of a string. Theor. Comput. Sci. 866: 70-81 (2021)

10. Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Circular pattern matching with k mismatches. J. Comput. Syst. Sci. 115: 73-85 (2021)

Chapter 2

Autoreferat

2.1 Wst¦p

2.1.1 Motywacja

Badanie regularnych fragmentów sªów, w szczególno±ci ich znajdowanie i zliczanie jest jedn¡ z najbardziej popularnych cz¦±ci algorytmiki tekstowej. Wiele algorytmów inaczej dziaªa na fragmentach sªów które s¡ okresowe ni» na takich w których ta wªasno±¢ nie wyst¦puje, jako »e fragmenty te cechuj¡ cz¦sto inne wªasno±ci. Przykªadowo du»a okresowo±¢ sªowa pozwala na ªatw¡ jego kompresj¦ - opisanie go przy pomocy samego okresu i dªugo±ci, z drugiej strony mo»e to utrudnia¢ wyszukiwanie takiego sªowa - je±li sªowo dªugo±ci n jest nieokresowe, to w tek±cie dªugo±ci n mo»e si¦ pojawi¢ najwy»ej $2\frac{n}{m}$ razy, nie jest to jednak prawd¡ w przypadku sªów okresowych (na przykªad sªowo aaaaaa pojawia si¦ w sªowie aaaaaaaaaaaaaaaaaa $n - m + 1 = 11$ razy). Sama ilo±¢ takich (ró»nych) struktur w sªowie mo»e sªu»y¢ za miar¦ jego regularno±ci lub skomplikowania.

W pracach wchodz¡cych w skªad doktoratu wraz z wspóªautorami zaj¦li±my si¦ kilkoma klasami podsªów regularnych lub takich, które mo»na przedstawi¢ w zwarty sposób. W ka»dej z prac przedstawili±my nowe, efektywne algorytmy wypisywania i zliczania wszystkich wyst¡pie«, lub wszystkich ró»nych podsªów pewnej klasy.

2.1.2 Skªad pracy

Rozprawa skªada si¦ z czterech artykuªów, napisanych podczas moich studiów doktorskich na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego we wspóªpracy z innymi autorami:

1. Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszy«ski, Tomasz Wale«, Wiktor Zuba: E cient Representation and Counting of Antipower Factors in Words. LATA 2019: 421-433

W wersji przyj¦tej do druku w Special Issue konferencji LATA 2019, które ma by¢ wydane w czasopi±mie Information & Computation.

2. Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszy«ski, Tomasz Wale«, Wiktor Zuba: Counting Distinct Patterns in Internal Dictionary Matching. CPM 2020: 8:1-8:15

3. Panagiotis Charalampopoulos, Jakub Radoszewski, Wojciech Rytter, Tomasz Wale«, Wiktor Zuba: The Number of Repetitions in 2D-Strings. ESA 2020: 32:1-32:18

4. Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Tomasz Wale«, Wiktor Zuba: E cient Enumeration of Distinct Factors Using Package Representations. SPIRE 2020: 247-261

Wszystkie prace zawieraj¡ oryginalne wyniki z algorytmiki na s^{a}owach. Prace zosta^{a}y napisane przez autorów reprezentuj¡cych Uniwersytet Warszawski we wspó^{a}pracy z naukowcami z King's College London (Panagiotis Charalampopoulos przebywa^a przez d^{a}u»szy czas w Warszawie) oraz Tomaszem Kociumak¡, który w mi¦dzyczasie zmieni^a a liacj¦ z Uniwersytetu Warszawskiego na Uniwersytet Bar-Ilan.

W sekcji 2.2 niniejszego autoreferatu przedstawiam podstawowe poj¦cia - de nicje regularnych klas s^{a}ów i pods^{a}ów, które s¡ wykorzystywane w przedstawianych pracach.

W sekcji 2.3 pokazuj¦ dok^{a}adny przegl¡d wyników za^a¡czonych prac oraz szkice metod u»ytych do ich uzyskania.

W sekcji 2.4 wymieniam publikacje których jestem wspó^{a}autorem i które zosta^{a}y opublikowane podczas moich studiów doktoranckich, które jednak nie zosta^{a}y w^a¡czone w sk^{a}ad tej rozprawy.

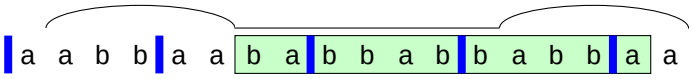
W rozdziale 1 zamieszczone jest to samo streszczenie w j¦zyku angielskim, za± w rozdzia^{a}ach 3, 4, 5 oraz 6 kolejne prace wchodz¡ce w sk^{a}ad rozprawy. Bibliogra a na ko«cu zawiera referencje u»yte w niniejszym streszczeniu.

2.2 Preliminaria

Aby przedstawi¢ wyniki za^a¡czonych prac nale»y zacz¡¢ od przedstawienia typów regularno±ci w s^{a}owach, przez nie wykorzystywanych. Wiele z tych poj¦¢ pojawia si¦ w wi¦cej ni» jednej z tych prac.

De nicja 2.1. Niech $x = y_0 y_1 \ldots y_{k-1}$ dla $k \geq 2$ i s^{a}ów y_i o tej samej d^{a}ugo±ci d. Wtedy mówimy, »e:

- x jest k-pot¦g¡ je±li wszystkie y_i s¡ takie same (x jest kwadratem je±li dodatkowo $k = 2$);
- x jest k-antypot¦g¡ je±li wszystkie y_i s¡ parami ró»ne;



Rysunek 2.1: Regularno±ci w s^{a}owach jednowymiarowych. Zielonym t^{a}em zaznaczono maksymalne powtórzenie (o okresie 3). Niebieskie linie pokazuj¡ wyst¡pienie 4-antypot¦gi. Zauwa»my, »e jedn¡ i dwie pozycje dalej nie wyst¦puj¡ 4-antypot¦gi o tej samej podstawie poniewa» te wyrazy d^{a}ugo±ci 16 s¡ przerywanymi kwadratami generowanymi przez zaznaczone nad s^{a}owem maksymalne 3-przerywane wyst¡pienie.

- x jest s^{a}ab¡ k-pot¦g¡ je±li nie jest k-antypot¦g¡, to jest je±li $y_i = y_j$ dla pewnych $i \neq j$;
- x jest przerywanym kwadratem je±li $y_0 = y_{k-1}$.

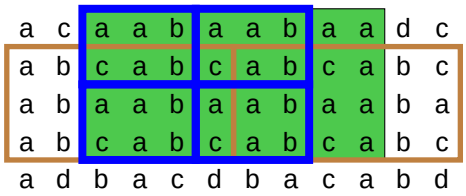
De nicja 2.2. Mówimy, »e:

- p jest okresem s^{a}owa w je±li $w[i] = w[i+p]$ dla wszystkich $i \in [0..|w|-p)$ (przez okres s^{a}owa najcz¦±ciej rozumiemy jego najkrótszy okres);
- fragment s^{a}owa $w[i..j]$ jest maksymalnym powtórzeniem je±li jego d^{a}ugo±¢ jest równa co najmniej dwóm jego najkrótszym okresom i nie mo»na go rozszerzy¢ o jedn¡ pozycj¦ w lewo ani w prawo bez zmiany tego okresu;
- fragment s^{a}owa $w[i..j]$ jest maksymalnym α-przerywanym powtórzeniem (dla $\alpha \geq 1$) je±li jest postaci uvu dla $p = |uvj \neg juj$ i nie mo»na go rozszerzy¢ o jedn¡ pozycj¦ w lewo ani w prawo bez zmiany okresu.

De nicja 2.3. Dla tekstów dwuwymiarowych de niujemy analogiczne powtórzenia.

- Tandemem nazywamy dwuwymiarowy tekst o parzystej d^{a}ugo±ci, w którym lewa i prawa po^{a}owa tekstu s¡ takie same (uogólnienie kwadratu na przypadek gdzie pojedyncz¡ liter¡ jest ca^{a}a kolumna).
- Kwartyk¡ nazywamy dwuwymiarowy tekst K o parzystej d^{a}ugo±ci i szeroko±ci, taki »e zarówno K jak i $K^?$ s¡ tandemami (cztery identyczne teksty u^{a}o»one w krat¦ 2 2).
- maksymalnym 2D-powtórzeniem nazywamy fragment tekstu dwuwymiarowego o wymiarach i, j, taki »e jego okres poziomy jest wielko±ci co najwy»ej $\frac{j}{2}$ a pionowy co najwy»ej $\frac{i}{2}$. Co wi¦cej powi¦kszenie tego fragmentu o dodatkow¡ kolumn¦ lub wiersz z dowolnej strony (o ile takie istnieje w ca^{a}ym tek±cie) skutkowa^{a}oby zmian¡ tych okresów.

Przez kwadrat generowany przez maksymalne powtórzenie rozumiem kwadrat zawarty w tym powtórzeniu o d^{a}ugo±ci równej dok^{a}adnie dwóm okresom powtórzenia.



Rysunek 2.2: Ró»ne rodzaje powtórze« w tekstach dwuwymiarowych. Kolorem br¡zowym zaznaczono tandem, niebieskim kwartyk¦, za± zielonym t ªem maksymalne 2D-powtórzenie.

Wszystkie takie kwadraty mo»na ªatwo wyznaczy¢ z powtórzenia i ka»dy kwadrat jest generowany przez jakie± maksymalne powtórzenie. Analogicznie wygl¡da sytuacja generowania przerywanych kwadratów przez powtórzenia i -przerywane powtórzenia oraz generowania kwartyk przez 2D-powtórzenia.

Kwadrat ma pierwiastek pierwotny gdy jego pierwiastek (po ªowa) nie jest k-pot¦g¡ dla »adnego $k > 1$. Analogicznie dzia ªa to w przypadku kwartyk, dla których jego pierwiastek (¢wiartka) nie jest k-pot¦g¡ dla $k > 1$ ani poziomo ani pionowo.

De nicja 2.4. S ªownikiem wewn¦trznym nazywamy zbiór pozycji pocz¡tkowych i ko«cowych wybranych pods ªów zadanego s ªowa.

2.3 Przegl¡d wyników

2.3.1 E cient Representation and Counting of Antipower Factors in Words

Zaprezentowana tutaj praca zosta ªa zaprezentowana na konferencji LATA 2019 i opublikowana w sprawozdaniu z konferencji. Rozszerzona wersja tej pracy zosta ªa pó¹niej przyj¦ta do druku w Special Issue tej konferencji, wydawanym w w czasopi±mie Information & Computation. Prac¦ w ªa±nie w tej formie (przyj¦tej do druku, lecz jeszcze nie opublikowanej) za ª¡czy ªem do niniejszej rozprawy (w rozdziale 3).

Wst¦p

W pracy [5] przedstawione zosta ªo górne i dolne ograniczenie $\Theta(n^2=k)$ na maksymaln¡ liczb¦ wyst¡pie« k-antypot¦g w s ªowie. Praca ta poda ªa równie» algorytm wyznaczaj¡cy te wyst¡pienia w czasie $O(n^2=k)$. Ze wzgl¦du na ograniczenia na wielko±¢ wyj±cia algorytm ten jest optymalny w przypadku pesymistycznym. Nie oznacza to jednak, »e nic w tej dziedzinie nie mo»na poprawi¢.

W naszej publikacji skupili±my si¦ na problemach które nie podlegaj¡ temu ograniczeniu - zliczaniu antypot¦g oraz ich wyznaczaniu ze z ªo»ono±ci¡ zale»n¡ od rozmiaru wyj±cia (efektywniejsz¡ gdy rozwa»ane s ªowo zawiera ma ªo antypot¦g).

Pisz¡c dokªadniej - zaprezentowali±my algorytmy, które dla sªowa dªugo±ci zliczaj¡ wszystkie podsªowa b¦d¡ce k-antypot¦gami w czasie $O(nk \log k)$, wyznaczaj¡ wszystkie te podsªowa w czasie $O(nk \log k$+rozmiar wyniku), oraz wyznaczaj¡ liczb¦ ró»nych k-antypot¦g w sªowie w czasie $O(nk^4 \log k \log n)$. Zaprezentowali±my równie» now¡ struktur¦ danych pozwalaj¡c¡ na szybkie sprawdzenie, czy dane podsªowo jest k-antypot¦g¡.

Jak wida¢ w (najcz¦±ciej wykorzystywanym) przypadku gdy rozwa»ane k jest maªe w stosunku do dªugo±ci sªowa (mniejsze ni» $\sqrt{n}$) nasz algorytm uzyskuje lepszy asymptotyczny czas dziaªania od poprzednio znanych.

Sªabe pot¦gi i zwarta reprezentacja

Ze wzgl¦du na nieregularn¡ natur¦ antypot¦g efektywne ich znajdowanie nie jest proste. Dlatego te», skupili±my si¦ na wyznaczeniu reprezentacji tych podsªów, które nimi nie s¡, czyli sªabych k-pot¦g. Jednocze±nie ka»da sªaba k-pot¦ga jest konsekwencj¡ wyst¡pienia przerywanego kwadratu (by¢ mo»e dla mniejszego k, lecz tego samego d).

Obserwacja 2.1.

(a) Ka»dy (przerywany) kwadrat jest generowany przez maksymalne powtórzenie o (nie koniecznie najkrótszym) okresie q $(q + 1)$ d lub maksymalne $(q + 1)$-przerywane powtórzenie o okresie q $(q + 1)$ d dla $q \neg k$ 2.

(b) Ka»de maksymalne powtórzenie i ka»de przerywane powtórzenie generuje pojedynczy przedziaª pozycji w którym pojawiaj¡ si¦ (przerywane) kwadraty okre±lonej dªugo±ci, co wi¦cej mo»emy ten przedziaª wyznaczy¢ w czasie staªym.

(c) Wszystkich ogólnych maksymalnych powtórze« w sªowie o dªugo±ci n jest $O(n)$ i ich reprezentacj¦ mo»na wyliczy¢ w czasie $O(n)$ ([6]).

(d) Wszystkich maksymalnych α-przerywanych powtórze« w sªowie jest $O(n^{\alpha})$ i ich reprezentacj¦ mo»na wyliczy¢ w czasie $O(n^{\alpha})$ ([11, 12]).

Korzystaj¡c z tych wªasno±ci potra my wyliczy¢ reprezentacj¦ sªabych k-pot¦g dla wszystkich d w postaci ª¡cznie $O(nk)$ ªa«cuchów przedziaªów (zbiór przedziaªów o tej samej dªugo±ci i pocz¡tkach tworz¡cych post¦p arytmetyczny). Nast¦pnie korzystaj¡c z metod geometrycznych potra my wyznaczy¢ sum¦ teoriomnogo±ciow¡ tych ªa«cuchów w ª¡cznym czasie $O(nk \log k)$. Daje nam to bezpo±rednio ilo±¢ sªabych k-pot¦g dla ka»dego d, a wi¦c i ilo±¢ k-antypot¦g. Zamiast liczy¢ wielko±¢ zbioru pozycji pokrytych przez reprezentacj¦ wyst¡pie« sªabych k-pot¦g mo»emy wyznaczy¢ wszystkie pozycje niepokryte i otrzyma¢ algorytm wyznaczaj¡cy k-antypot¦gi w czasie $O(nk \log k$ + wynik).

Pytania o podsªowa

W pracy [5] zaprezentowano równie» dwie struktury danych które po zbudowaniu pozwalaj¡ odpowiada¢ na pytania "Czy dane podsªowo $w[i..j]$ jest k-antypot¦g¡?". Pierwsza struktura o rozmiarze $O(n)$ odpowiada na pytania

w czasie $O(k)$, za± druga robi to w czasie sta^{a}ym, lecz wymaga zapami¦tania informacji rozmiaru $O(n^2)$.

W naszej pracy przedstawili±my now¡ struktur¦, parametryzowan¡ przez warto±¢ $r \in [1; n]$. Po zbudowaniu w czasie $O(n^2=r)$ struktura ma rozmiar $O(n^2=r)$ i odpowiada na pytania w czasie $O(r)$.

Je±li $r < k$, to wiemy, »e podstawa pot¦gi d wynosi co najwy»ej $n=r > n=k$. Aby odpowiada¢ na pytania dla ka»dego d ¬ $n=r$ osobno budujemy struktur¦ danych opart¡ na range minimum queries, o rozmiarze $O(n)$ i czasie zapytania $O(1)$. W komplementarnym przypadku aby uzyska¢ po»¡dany rezultat wystarczy skorzysta¢ ze wspomnianej struktury z pracy [5].

Zliczanie ró»nych k-antypot¦g

W wersji do czasopisma dodali±my kolejny wynik - zliczanie ró»nych k-antypot¦g w s^{a}owie.

Ze wzgl¦du na potencjaln¡ du»¡ ilo±¢ wszystkich wyst¡pie« antypot¦g metody wykorzystuj¡ce ich wyznaczenie nie pozwalaj¡ na osi¡gni¦cie satysfakcjonuj¡cej szybko±ci. Aby uzyska¢ taki wynik ponownie skorzystali±my z powi¡zania antypot¦g z bardziej regularnymi s^{a}abymi pot¦gami.

Liczb¦ ró»nych k-antypot¦g mo»emy uzyska¢ odejmuj¡c od siebie dwie inne warto±ci. Pierwsz¡ warto±ci¡ jest ilo±¢ (wszystkich) ró»nych pods^{a}ów o d^{a}ugo±ci podzielnej przez k. Mo»emy j¡ ^{a}atwo uzyska¢ w czasie $O(n)$ przy u»yciu drzewa su ksowego.

Drug¡ potrzebn¡ warto±ci¡ jest liczba ró»nych s^{a}abych k-pot¦g. Aby j¡ uzyska¢ najpierw wzmocnili±my de nicj¦ s^{a}abych pot¦g, tak aby ka»da by^{a}a generowana na dok^{a}adnie jeden sposób. Nast¦pnie przez redukcj¦ do problemu grafowego uda^{a}o nam si¦ wykona¢ liczenie ró»nych takich konstrukcji w ^{a}¡cznym czasie $O(nk^4 \log k \log n)$.

2.3.2 Counting Distinct Patterns in Internal Dictionary Matching

W pracy [8] moi wspó^{a}autorzy wprowadzili problem wyszukiwania przy u»yciu s^{a}ownika wewn¦trznego - dla zadanego s^{a}owa T i jego s^{a}ownika wewn¦trznego D przedstawili konstrukcj¦ struktury danych pozwalaj¡c¡ efektywnie odpowiada¢ na pytania dla danego pods^{a}owa $T[i..j]$:

- czy $T[i..j]$ zawiera jakie± s^{a}owo ze s^{a}ownika?
- zwróci¢ wszystkie pods^{a}owa s^{a}owa $T[i..j]$ które nale»¡ do s^{a}ownika
- zwróci¢ wszystkie s^{a}owa ze s^{a}ownika D które s¡ zawarte w $T[i..j]$
- zwróci¢ liczb¦ pods^{a}ów $T[i..j]$ które nale»¡ do s^{a}ownika
- zwróci¢ liczb¦ ró»nych s^{a}ów ze s^{a}ownika D które s¡ zawarte w $T[i..j]$

Przedstawiona struktura danych mia^{a}a rozmiar i czas konstrukcji $\tilde{O}(n+d)$ (gdzie $n = |T|$ oznacza d^{a}ugo±¢ s^{a}owa T, $d = |D|$ liczb¦ s^{a}ów w s^{a}owniku, za± $\tilde{O}$ notacj¦ asymptotyczn¡ pomijaj¡c¡ czynniki polilogarytmiczne) i odpowiadaj¡c¡ na pierwsze cztery pytania w czasie $\tilde{O}(1 + \text{wynik})$.

W przypadku ostatniego pytania tylko $O(\log n)$ aproksymacja wyniku zosta³a zaprezentowana.

Rozmiar	Czas konstrukcji	Czas zapytania	Wariant
$\Theta(n + d)$	$\Theta(n + d)$	$\Theta(1)$	2-aproksymacja
$\Theta(n^2/m^2 + d)$	$\Theta(n^2/m + d)$	$\Theta(m)$	dok³adny
$\Theta(nd/m + d)$	$\Theta(nd/m + d)$	$\Theta(m)$	dok³adny
$O(n \log^2 n)$	$O(n \log^2 n)$	$O(\log n)$	D = kwadraty, dok³adny

Tabela 2.1: Nasze rezultaty dla zapytañ policz ró¿ne$_r(n)$ ($r \in [1; n]$ jest dowolnie wybranym parametrem).

2-aproksymacja

Aby poprawiæ wynik z poprzedniej pracy stworzyli¶my strukturê danych, która pamiêta wyniki dla pods³ów bazowych, czyli wszystkich pods³ów d³ugo¶ci $b(1 + \varepsilon)^p c$ dla wszystkich liczb naturalnych p i ustalonego $\varepsilon = \frac{1}{9}$. Aby otrzymaæ wynik dla dowolnego pods³owa $T[i..j]$ korzystamy z maksymalnych s³ów bazowych postaci $T[i..i']$ oraz $T[j'..j]$ zawartych w $T[i..j]$ i otrzymujemy jego podzia³ na trzy czê¶ci $F_1F_2F_3$ ($F_1F_2 = T[i..i']$, $F_2F_3 = T[j'..j]$).

Wynik uzyskujemy poprzez zliczenie s³ów ze s³ownika, które maj¹ wyst¹pienie zaczynaj¹ce siê w F_1 i ko«cz¹ce w F_3, jednak nie wystêpuj¹ce w ¿adnym z wymienionych s³ów bazowych, co udaje nam siê zrobiæ efektywnie dziêki du¿emu stosunkowi d³ugo¶ci F_2 do F_1 i F_3. Do tego dodajemy ilo¶ci s³ów wystêpuj¹cych w obu s³owach bazowych, co jednak skutkuje uzyskaniem jedynie 2-aproksymacji ze wzglêdu na mo¿liwe powtórzenia.

Wszystkich s³ów bazowych jest $O(n \log n)$, za¶ warto¶ci dla nich potrafimy policzyæ w ³¹cznym czasie $O(n \log^{1+\varepsilon} n + d)$ (dla dowolnego sta³ego $\varepsilon > 0$) wykorzystuj¹c to, ¿e wynik po przesuniêciu o jedn¹ pozycjê mo¿na zaktualizowaæ niewielkim kosztem. Struktura danych u¿ywana do zliczania s³ów opisanych w poprzednim akapicie kosztuje nas dodatkowe $O(n \log n/\log\log n + d \log^{3/2} n)$ czasu preprocessingu oraz zajmuje $O(n + d \log n)$ pamiêci. Pozwala ona odpowiadaæ na pytania w czasie $O(\log^2 n/\log\log n)$.

Obliczanie dok³adne

Je¶li zamiast dla s³ów bazowych zapamiêtamy wyniki dla wszystkich pods³ów postaci $T[c_1 m + 1..c_2 m]$ dla wybranego m oraz wszystkich $c_1 < c_2$ otrzymamy strukturê danych o rozmiarze $O(n^2/m^2 + n + d)$ obliczan¹ w czasie $O((n^2 \log n)/m + n\sqrt{\log n} + d)$. Rozszerzaj¹c takie s³owo po jednej pozycji mo¿emy uzyskaæ wynik dla dowolnego s³owa przy u¿yciu $O(m)$ operacji o koszcie $O(\log n)$.

Innym podej¶ciem do problemu jest zag³êbienie siê w strukturê s³ownika.

Je¶li s³ownik nie zawiera zbioru k ró¿nych prefiksów tego samego s³owa, to na dowolnej pozycji mo¿e siê zaczynaæ co najwy¿ej $k - 1$ ró¿nych s³ów z tego

sªownika. W takim przypadku mo»emy dla ka»dego sªowa ze sªownika zapami¦ta¢ wszystkie pozycje na których ono wyst¦puje. Daje nam to struktur¦ danych wielko±ci $O(nk \log n)$, obliczaln¡ w czasie $O(nk \log n)$ i pozwalaj¡c¡ odpowiada¢ na pytania w czasie $O(\log n)$.

Aby uzyska¢ sªownik w takiej postaci wydzielamy z niego (jako nowe sªowniki) sªowa niespeªniaj¡ce tej wªasno±ci, to jest du»e grupy sªów b¦d¡cych preksami tego samego sªowa. Takich grup (o wielko±ci co najmniej k) jest co najwy»ej $d=k$, za± dla ka»dej takiej grupy potra my efektywnie odpowiada¢ na pytania przy pomocy struktury do pyta« o ograniczone LCP ([14]) o rozmiarze $O(n^{\sqrt{\log n}})$ w czasie $O(\log n)$ dla dowolnego > 0.

Dla $k = d\frac{d}{m}e$ daje nam to w sumie czas zapytania wielko±ci $O(m \log n + \log n)$.

Sªownik kwadratów

Dla popularnego przypadku obliczania liczby kwadratów w podsªowie opisali±my odr¦bny algorytm. Dzi¦ki szczególnym wªasno±ciom takich podsªów mo»emy otrzyma¢ lepsze wyniki.

Wszystkich wyst¡pie« kwadratów w sªowie mo»e by¢ $O(n^2)$ (ró»nych kwadratów tylko $O(n)$), jednak dzi¦ki skorzystaniu z maksymalnych powtórze«, których w caªym sªowie jest $O(n)$ (i które mo»na obliczy¢ w czasie $O(n)$) potra my wyznaczy¢ $m = O(n \log n)$ istotnych wyst¡pie« kwadratów, które s¡ wystarczaj¡co reprezentatywne dla naszych potrzeb. Dzi¦ki skorzystaniu z geometrycznej struktury danych o rozmiarze i czasie konstrukcji $O(m \log m)$ z pracy [13] potra my odpowiedzie¢ na pytanie o liczb¦ ró»nych takich sªów w podsªowie w czasie $O(\log m) = O(\log n)$.

2.3.3 The Number of Repetitions in 2D-Strings

W pracy zamieszczonej w rozdziale 5 zaj¦li±my si¦ problemem powtórze« w tekstach dwuwymiarowych. W tabeli 2.2 przedstawiªem podsumowanie naszych wyników oraz dotychczasowej wiedzy na temat mo»liwej ilo±ci maksymalnych 2D-powtórze« jak i maksymalnej liczby ró»nych tandemów i kwartyk oraz wszystkich wyst¡pie« tych o pierwotnym pierwiastku. Przedstawiªem równie» czasy algorytmów pozwalaj¡cych wyznaczy¢ te struktury w tek±cie. Warto zauwa»y¢, »e ilo±¢ 2D-powtórze« byªa ju» wcze±niej badana, jednak granica pomi¦dzy dolnym i górnym ograniczeniem pozostawaªa liniowa. W przedstawianej pracy udaªo nam si¦ zredukowa¢ t¡ wielko±¢ do polilogarytmicznej.

maksymalne 2D-powtórzenia

Aby uzyska¢ ograniczenie na liczb¦ maksymalnych 2D-powtórze« skorzystali±my z lematu z pracy [1] aby przypisa¢ ka»de takie powtórzenie do maksymalnego powtórzenia poziomego (brak wymagania pionowej okresowo±ci i maksymalno±ci) o wysoko±ci b¦d¡cej pot¦g¡ dwójki (najwi¦ksz¡ mo»liw¡, nie wi¦ksz¡ ni» wysoko±¢ 2D-powtórzenia). Takie powtórzenie poziome speªnia równie» trzy istotne warunki - jego lewy górny lub lewy dolny naro»nik jest równy odpowiadaj¡cemu naro»nikowi 2D-powtórzenia, jego okres jest równy okresowi poziomemu

	Ograniczenia na ilo±¢ w tek±cie n n	Czas wyznaczania
maksymalne 2D-powtórzenia	$(n^2), O(n^3)$[2] $O(n^2 \log^2 n)$	$O(n^2 \log n$+wynik)[2] $O(n^2 \log^2 n)$
Wyst¡pienia kwartyk o pierwotnych pierwiastkach	$(n^2 \log^2 n)$ [3]	$O(n^2 \log^2 n)$
Ró»ne kwartyki	$O(n^2 \log^2 n)$	$O(n^2 \log^2 n)$
Wyst¡pienia tandemów o pierwotnych pierwiastkach	$(n^3 \log n)$[3]	$O(n^3 \log n)$[4]
Ró»ne tandemy	$(n^2); O(n^4)$(trywialne) (n^3)	$O(n^3)$

Tabela 2.2: Przegl¡d wcze±niejszych wyników i naszych rezultatów (pogrubione).

Rysunek 2.3: Przypisanie maksymalnemu 2D-powtórzeniu maksymalnego powtórzenia poziomego o wysoko±ci 2^k

powi¡zanego 2D-powtórzenia oraz jego d³ugo±¢ jest co najmniej taka jak d³ugo±¢ 2D-powtórzenia (rysunek 2.3).

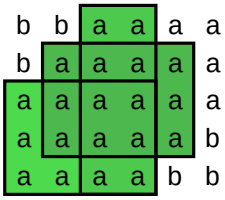
Korzystaj¡c z tych w³asno±ci, lematu o okresowo±ci oraz lematu o trzech kwadratach dowiedli±my, »e wszystkim poziomym powtórzeniom okupuj¡cym wiersze od i do $i + 2^k - 1$ mo»e zosta¢ przypisane ³¡cznie $O(n \log n)$ ró»nych maksymalnych 2D-powtórze«. Mno»¡c t¡ warto±¢, przez ilo±¢ mo»liwych wyborów i oraz k otrzymujemy ograniczenie $O(n^2 \log^2 n)$ na liczb¦ maksymalnych 2D-powtórze« w tek±cie rozmiaru n.

Nasze ograniczenie na ilo±¢ maksymalnych 2D-powtórze« automatycznie daje nowe ograniczenie na czas dzia³ania algorytmu z pracy [2] (algorytm ten dzia³a w czasie zale»nym od rozmiaru wyniku).

Kwartyki o pierwotnych pierwiastkach

W przypadku jednowymiarowym wszystkie kwadraty o pierwotnych pierwiastkach mo»na bardzo ³atwo wyznaczy¢ znaj¡c maksymalne powtórzenia w s³owie - wystarczy wzi¡¢ wszystkie s³owa o d³ugo±ci dwóch okresów maksymalnego powtórzenia, które s¡ w nim zawarte.

W przypadku dwuwymiarowym jest podobnie - ka»da kwartyka o pierwotnym pierwiastku jest prostok¡tem o d³ugo±ci dwóch okresów poziomych i wysoko±ci dwóch okresów pionowych pewnego maksymalnego 2D-powtórzenia, ca³kowicie w nim zawartym. Pojawia si¦ jednak pewien problem - o ile w przypadku jednowymiarowym dwa maksymalne powtórzenia o tym samym okresie nie mog¡



Rysunek 2.4: Wiele przecinaj¡cych si¦ maksymalnych 2D-powtórze« o tych samych okresach

mie¢ du»ego przeci¦cia i w szczególno±ci nie mog¡ generowa¢ tego samego kwadratu, o tyle w 2D co± takiego mo»e bardzo ªatwo zachodzi¢ (rysunek 2.4), przez to zwykªe u»ycie takiego algorytmu mogªoby skutkowa¢ wielokrotnym zwracaniem tych samych kwartyk.

Rozwi¡zaniem tego problemu jest metoda zamiatania - algorytm z pracy [7], który pozwala na liczenie sum teoriomnogo±ciowych kilku ró»nych rodzin prostok¡tów na kracie $n \times n$ w ª¡cznym czasie $O(n + r + \text{wynik})$, gdzie r to liczba wszystkich prostok¡tów.

Dziel¡c wszystkie maksymalne 2D-powtórzenia na grupy o takich samych okresach i dla ka»dej takiej bior¡c prostok¡ty wyznaczaj¡ce lewe górne rogi generowanych kwartyk otrzymujemy po»¡dany algorytm.

Dzi¦ki naszemu ograniczeniu na liczb¦ r z poprzedniej podsekcji oraz ograniczeniu $O(n^2 \log^2 n)$ na liczb¦ kwartyk o pierwotnym pierwiastku z pracy [3] algorytm dziaªa w ª¡cznym czasie $O(n^2 \log^2 n)$ (optymalnym ze wzgl¦du na dolne ograniczenie z tej samej pracy).

Ró»ne kwartyki

W tek±cie rozmiaru $n \times n$ zªo»onym z samych litera zawarte jest $\Theta(n^4)$ wyst¡pie« dowolnych kwartyk, przez co trywialny algorytm znajduje wszystkie takie wyst¡pienia w optymalnym pesymistycznym czasie. Dlatego te» ciekawszym problemem jest wyznaczenie ró»nych kwartyk (fragmenty ró»ni¡ce si¦ nie tylko poªo»eniem, ale te» jako teksty).

Wynik z pracy [3] daje ograniczenie na liczb¦ kwartyk o pierwiastku pierwotnym. Inne kwartyki zªo»one s¡ z wielu przylegaj¡cych do siebie wyst¡pie« takich samych pierwotnych kwartyk tworz¡cych wi¦kszy prostok¡t. Innymi sªowy o ile w przypadku tych poprzednich kwartyk wyst¡pienia pierwotnego tekstu W tworzyªy krat¦ 2×2, to w przypadku tych pozostaªych tworz¡ kraty $2k \times 2l$, gdzie $k > 1$ lub $l > 1$. Kwartyki takie dzielimy na w¡skie, czyli takie dla których $k = 1$ lub $l = 1$ i szerokie dla których $k, l > 1$.

Nast¦pnie dokonujemy kolejnego podziaªu - kwartyki dla których sªowo pierwotne W ma rozmiary w przedziaªach $[2^a; 2^{a+1}) \times [2^b; 2^{b+1})$ dla pewnych $a, b \in \log n$ rozwa»amy razem.

Ze wzgl¦du na lematy o okresowo±ci oraz o trzech kwadratach ka»dy punkt mo»e by¢ lewym górnym rogiem ostatniego wyst¡pienia (w tym samym wierszu, bardziej na prawo w tek±cie nie wyst¦puje taka sama kwartyka) co najwy»ej

czterech ró»nych kwartyk w¡skich z jednej grupy (analogicznie dzia³a dowód na ograniczenie liczby ró»nych kwadratów w standardowym tek±cie [10]).

W przypadku kwartyk szerokich dzi¦ki mnogo±ci punktów kratowych w których zaczyna si¦ wyst¡pienie s³owa W ka»dej takiej kwartyce b¦d¡cej pot¦g¡ W mo»emy przypisa¢ unikalny punkt na tej kracie. Dodatkowo dzi¦ki lematowi o okresowo±ci taki punkt nie mo»e nale»e¢ do kraty dla kwartyki b¦d¡cej pot¦g¡ innego $W' \neq W$ o rozmiarze nale»¡cym do $[2^a; 2^{a+1}) \times [2^b; 2^{b+1})$.

Dzi¦ki przypisywaniu punktów z $[1 ; n] \times [1; n]$ do kwartyk oraz $\log^2 n$ wyborom $a; b$ otrzymujemy ograniczenia $O(n^2 \log^2 n)$ na ilo±¢ ró»nych kwartyk ka»dego z typów.

W celu wyznaczenia wszystkich unikalnych kwartyk w tek±cie korzystamy z naszego wcze±niejszego algorytmu by wyznaczy¢ wszystkie wyst¡pienia kwartyk o pierwiastkach pierwotnych i grupujemy je wed³ug pierwiastków.

Kwartyki b¦d¡ce pot¦gami s³owa pierwotnego W powstaj¡ z po³¡czenia wielu kwartyk pierwotnych o tym samym pierwiastku. Mo»emy wi¦c wyznaczy¢ lewe górne rogi wszystkich wyst¡pie« takich kwartyk i wyszuka¢ maksymalne kraty utworzone przez takie punkty.

Problem ten rozwi¡zujemy u»ywaj¡c ponownie metody zamiatania by uzyska¢ algorytm dzia³aj¡cy w czasie $O(n^2 \log^2 n)$.

Ró»ne tandemy

Jako powi¡zanym problemem zaj¦li±my si¦ te» wyznaczaniem ró»nych tandemów (podobnie jak w przypadku kwartyk wszystkich wyst¡pie« tandemów mo»e by¢ $\Omega(n^4)$).

W wypadku tego problemu mo»na ³atwo uto»sami¢ tandemy zaczynaj¡ce si¦ w wierszu i oraz ko«cz¡ce w wierszu j z kwadratami w standardowym s³owie (jednowymiarowym), poprzez przypisanie kolumnom wysoko±ci $j - i + 1$ standardowych liter. W standardowym s³owie d³ugo±ci n mo»e by¢ $\Omega(n)$ ró»nych kwadratów, st¡d te» takie samo ograniczenie obowi¡zuje tandemy okupuj¡ce wyznaczone wiersze. Mno»¡c t¦ warto±¢ przez liczb¦ wyborów warto±ci i oraz j otrzymujemy ograniczenie $O(n^3)$ na liczb¦ ró»nych tandemów.

Dolne ograniczenie $\Omega(n^3)$ na maksymaln¡ liczb¦ takich tandemów otrzymujemy równie prosto - otrzymujemy je na przyk³ad dla tekstu w którym ka»dy wiersz jest s³owem nad alfabetem jednoliterowym, dla ka»dego wiersza innym.

Przypisanie kolumnom wysoko±ci $j - i + 1$ standardowych liter mo»na zrobi¢ konstrukcyjnie. Dzi¦ki temu i dzi¦ki skorzystaniu z algorytmów wyszukuj¡cych ró»ne kwadraty w standardowym s³owie otrzymujemy algorytm wyznaczaj¡cy ró»ne tandemy w s³owie dwuwymiarowym w czasie $O(n^3)$. Ze wzgl¦du na dolne ograniczenie maksymalnego rozmiaru wyniku »aden bardziej skomplikowany algorytm nie pozwala na uzyskanie lepszego rezultatu w pesymistycznym przypadku.

2.3.4 E cient Enumeration of Distinct Factors Using Package Representations

W ostatniej z opisywanych prac zaj¦li±my si¦ now¡ reprezentacj¡ podzbiorów pods²ów która w wielu przypadkach zapewnia zwi¦z²¡ reprezentacj¦, która jednocze±nie pozwala na efektywne operowanie opisywanym podzbiorem.

Przez pakiet rozumiemy zbiór pods²ów tej samej d²ugo±ci wyst¦puj¡cych na kolejnych pozycjach w s²owie. Nasza reprezentacja sk²ada si¦ ze zbioru takich pakietów reprezentowanych jako trójki (pierwsza pozycja pocz¡tkowa, d²ugo±¢ s²ów, d²ugo±¢ przedzia²u). Rzecz¡, która interesuje nas szczególnie jest zbiór wszystkich pods²ów które nale»¡ do przynajmniej jednego z tych pakietów.

Bardziej formalnie

$$Factors(F) = \{T[j..j + l) : j \in [i, i + k] \text{ and } (i, l, k) \in F\}.$$

Przyk²adami rodzin pods²ów, dla których ²atwo mo»na uzyska¢ tak¡ reprezentacj¦ s¡ pot¦gi (kiedy to ka»dy pakiet reprezentuje bezpo±rednio jedno maksymalne powtórzenie) orazk-antypot¦gi (gdy mo»emy wykorzysta¢ bezpo±rednio ich reprezentacj¦ z pracy opisanej w sekcji 2.3.1). Reprezentacje te maj¡ rozmiar (i czas obliczania) równy odpowiednio$O(n)$ i $O(nk^2)$ (ka»dy ²a«cuch przedzia²ów reprezentuje co najwy»ejk zwyk²ych przedzia²ów).

W celu znalezienia zbioru$Factors(F)$, czyli ró»nych pods²ów nale»¡cych do reprezentacji oraz$|Factors(F)|$, czyli ilo±ci takich s²ów rozpatrujemy dwa przypadki. Pierwszy prostszy przypadek, nazywany w pracy specjalnym, to taki w którym je±li pods²owo nale»y do$Factors(F)$, to ka»de jego wyst¡pienie w rozwa»anym s²owie nale»y do którego± z pakietów nale»¡cych do$F$.

W tym przypadku dla s²owa d²ugo±ci$n$ i F sk²adaj¡cej si¦ z m pakietów korzystaj¡c ze struktury danych najd²u»szych poprzednich pods²ów (longest previous factors [9]) otrzymujemy wyznaczenie i zliczanie wyniku w czasie odpowiednio $O(n + m + wynik)$ i $O(n + m)$. W przypadku przedstawionych przyk²adów daje nam to nowy, prosty algorytm wyznaczania ró»nych pot¦g w s²owie, dzia²aj¡cy w czasie$O(n)$, oraz nowy algorytm wyznaczania ró»nychk-antypot¦g dzia²aj¡cy w czasie$O(nk^2 + wynik)$, a wi¦c lepszy od naszego algorytmu z pracy opisanej w sekcji 2.3.1, który zwraca ilo±¢ ró»nychk-antypot¦g, dzia²a w czasie $O(nk^4 \log k \log n)$ i jest du»o bardziej skomplikowany.

Drugi rozwa»any przypadek, nazywany w pracy ogólnym nie wymaga tego dodatkowego za²o»enia, co jednak czyni go istotnie trudniejszym do rozwi¡zania. Aby go rozwi¡za¢ adaptujemy metod¦ u»yt¡ poprzednio w pracy o k-antypot¦gach (zliczanie ró»nych k-antypot¦g) i uzyskujemy algorytmy wyznaczania i zliczania wyniku $Factors(F)$, dzia²aj¡ce w czasie odpowiednio$O(n \log^2 n + m \log n + wynik)$ i $O(n \log^2 n + m \log n)$.

2.4 Pozosta²e publikacje

Podczas moich studiów doktorskich zosta²y opublikowane równie» inne prace których jestem wspó²autorem. Zdecydowa²em si¦ nie do²¡cza¢ ich do niniejszej

rozprawy ze wzgl¦du na niedostateczne dopasowanie do jej tematu.

1. Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Faster Recovery of Approximate Periods over Edit Distance. SPIRE 2018: 233-240

2. Wojciech Rytter, Wiktor Zuba: Syntactic View of Sigma-Tau Generation of Permutations. LATA 2019: 447-459
   Rozszerzona wersja pracy zosta³a przyj¦ta do druku w czasopi±mie Theoretical Computer Science.

3. Mai Alzamel, Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Quasi-Linear-Time Algorithm for Longest Common Circular Factor. CPM 2019: 25:1-25:14

4. Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Circular Pattern Matching with k Mismatches. FCT 2019: 213-228

5. Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Weighted Shortest Common Supersequence Problem Revisited. SPIRE 2019: 221-238

6. Maxime Crochemore, Costas S. Iliopoulos, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Shortest Covers of All Cyclic Shifts of a String. WALCOM 2020: 69-80

7. Panagiotis Charalampopoulos, Solon P. Pissis, Jakub Radoszewski, Tomasz Walen, Wiktor Zuba: Unary Words Have the Smallest Levenshtein k-Neighbourhoods.CPM 2020: 10:1-10:12

8. Maxime Crochemore, Costas S. Iliopoulos, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Internal Quasiperiod Queries. SPIRE 2020: 60-75

9. Maxime Crochemore, Costas S. Iliopoulos, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Shortest covers of all cyclic shifts of a string. Theor. Comput. Sci. 866: 70-81 (2021)

10. Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, Wiktor Zuba: Circular pattern matching with k mismatches. J. Comput. Syst. Sci. 115: 73-85 (2021)

Chapter 3

E cient Representation and Counting of Antipower Factors in Words

Chapter 4

Counting Distinct Patterns in Internal Dictionary Matching

Chapter 5

The Number of Repetitions in 2D-Strings

Chapter 6

E cient Enumeration of Distinct Factors Using Package Representations

Bibliography

[1] Amihood Amir, Gad M. Landau, Shoshana Marcus, and Dina Sokol. Two-dimensional maximal repetitions. In 26th Annual European Symposium on Algorithms, ESA 2018, volume 112 of LIPIcs, pages 2:1{2:14. Schloss Dagstuhl - Leibniz-Zentrum fur Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.2.

[2] Amihood Amir, Gad M. Landau, Shoshana Marcus, and Dina Sokol. Two-dimensional maximal repetitions. Theoretical Computer Science, 812:49{61, 2020. doi:10.1016/j.tcs.2019.07.006 .

[3] Alberto Apostolico and Valentin E. Brimkov. Fibonacci arrays and their two-dimensional repetitions. Theoretical Computer Science, 237(1-2):263{273, 2000. doi:10.1016/S0304-3975(98)00182-0 .

[4] Alberto Apostolico and Valentin E. Brimkov. Optimal discovery of repetitions in 2D. Discrete Applied Mathematics, 151(1-3):5{20, 2005. doi:10.1016/j.dam.2005.02.019 .

[5] Golnaz Badkobeh, Gabriele Fici, and Simon J. Puglisi. Algorithms for anti-powers in strings. Information Processing Letters, 137:57{60, 2018. doi:10.1016/j.ipl.2018.05.003 .

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

[7] Jon Louis Bentley. Algorithms for Klee's rectangle problems. Unpublished notes, Computer Science Department, Carnegie Mellon University, 1977.

[8] Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, and Tomasz Wale«. Internal dictionary matching. In 30th International Symposium on Algorithms and Computation, ISAAC 2019, volume 149 of LIPIcs, pages 22:1{22:17. Schloss Dagstuhl{Leibniz-Zentrum fur Informatik, 2019. arXiv:1909.11577 , doi:10.4230/LIPIcs.ISAAC.2019.22 .

[9] Maxime Crochemore and Lucian Ilie. Computing longest previous factor in linear time and applications. *Inf. Process. Lett.*, 106(2):75–80, 2008. doi:10.1016/j.ipl.2007.10.006.

[10] Aviezri S. Fraenkel and Jamie Simpson. How many squares can a string contain? *J. Comb. Theory, Ser. A*, 82(1):112–120, 1998. doi:10.1006/jcta.1997.2843.

[11] Paweł Gawrychowski, Tomohiro I, Shunsuke Inenaga, Dominik Köppl, and Florin Manea. Tighter bounds and optimal algorithms for all maximal α-gapped repeats and palindromes - finding all maximal α-gapped repeats and palindromes in optimal worst case time on integer alphabets. *Theory of Computing Systems*, 62(1):162–191, 2018. doi:10.1007/s00224-017-9794-5.

[12] Tomohiro I and Dominik Köppl. Improved upper bounds on all maximal α-gapped repeats and palindromes. *Theoretical Computer Science*, 753:1–15, 2019. doi:10.1016/j.tcs.2018.06.033.

[13] Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Efficient colored orthogonal range counting. *SIAM Journal on Computing*, 38(3):982–1011, 2008. doi:10.1137/070684483.

[14] Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. In *26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 532–551. SIAM, 2015. doi:10.1137/1.9781611973730.36.