University of Verona

Department of
Computer Science

Graduate School of
Natural Sciences and Engineering

Doctoral Program in
Computer Science

Cycle: XXXV, 2019

Sensitivity of the Burrows-Wheeler Transform to Small Modifications, and Other Problems on String Compressors in Bioinformatics

S.S.D. INF/01

Author:
Sara Giuliani

Supervisor:
Zsuzsanna Lipták

Sensitivity of the Burrows-Wheeler Transform to Small Modifications, and
Other Problems on String Compressors in Bioinformatics
Sara Giuliani
Ph.D. Thesis
Verona, Italy, September 14, 2023

Abstract

Extensive amount of data is produced in textual form nowadays, especially in bioinformatics. Several algorithms exist to store and process this data efficiently in compressed space. In this thesis, we focus on both combinatorial and practical aspects of two of the most widely used algorithms for compressing text in bioinformatics: the Burrows-Wheeler Transform (BWT) and Lempel-Ziv compression (LZ77).

In the first part, we focus on combinatorial aspects of the BWT. Given a word v, $r = r(v)$ denotes the number of maximal equal-letter runs in $\mathrm{BWT}(v)$.

First, we investigate the relationship between r of a word and r of its reverse. We prove that there exist words for which these two values differ by a logarithmic factor in the length of the word. In other words, although the repetitiveness in the two words is preserved, the number of runs can change by a non-constant factor. This suggests that the number of runs may not be an ideal repetitiveness measure.

The second combinatorial aspect we are interested in is how small alterations in a word may affect its BWT in a relevant way. We prove that the number of runs of the BWT of a word can change (increase or decrease) by up to a logarithmic factor in the length of the word by just adding, removing, or substituting a single character.

We then consider the special character \$ used in real-life applications to mark the end of a word. We investigate the impact of this character on words with respect to the BWT. We characterize positions in a word where \$ can be inserted in order to turn it into the BWT of a \$-terminated word over the same alphabet. We show that, whether and where \$ is allowed, depends entirely on the structure of a specific permutation of the indices of the word, which is called the *standard permutation* of the word.

The final part of this thesis treats more applied aspects of text compressors. In bioinformatics, BWT-based compressed data structures are widely used for pattern matching. We give an algorithm based on the BWT to find Maximal Unique Matches (MUMs) of a pattern with respect to a reference text in compressed space, extending an existing tool called PHONI [Boucher et. al, DCC 2021].

Finally, we study some aspects of the Lempel-Ziv 77 (LZ77) factorization of a word. Modeling DNA short reads, we provide a bound on the compression size of the concatenation of regular samples of a word.

Acknowledgment

Firstly, I want to express my gratitude to my parents, Annamaria and Mimmo, who have always supported my choices and stood by me through the consequences. They have consistently done everything in their power to make things as smooth as possible for me.

A special thanks goes to my supervisor, Zsuzsanna Lipták, who patiently guided me in my initial steps toward research. She has always been there and willing to help whenever I needed it.

I am thankful to all my co-authors from whom I learned a lot. They have provided me with technical support and emotional encouragement, especially Simon, Marinella, Golnaz, and Giuseppe. I would also like to express my gratitude to Hideo Bannai and Johannes Fischer for dedicating their time and effort to review the manuscript. I genuinely appreciate all the valuable comments and suggestions that helped improve the quality of the thesis.

I am glad I had the opportunity to participate in the Monday Meetings, and would like to thank each member of this group, with whom I probably spent the longest but also the most interesting Mondays ever.

I would like to extend my thanks to all my colleagues, particularly Davide, who shared the entire Ph.D. journey with me, from the very beginning to the end. I thank everyone in office 1.70, especially Francesco and Alberto, who have a remarkable ability to uplift my spirits even on the worst days.

I want to express my appreciation to all my friends, both the ones I have known for a long time and the new friends I have made through climbing. I have always felt emotionally spotted from the former and also literally from the latter. I am thankful for each and every one of them.

Lastly, I want to thank my beloved Andrei, who is always there to encourage me when I feel exhausted, to help me find a way when I feel lost, to dry my tears when I need it (I cry a lot, like A LOT!), and to believe in me especially when I do not. I sincerely could not make it without him.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis consists of 152 pages and approximately 60 000 words. When read in the traditional western way, left-to-right and top-down, the sequence of words in these pages has a meaning and carries specific information. Assume the we are interested in some particular properties of the text of this thesis. For example, does it give some information if read back-to-front, too? Does a decomposition of the text in specific parts suggest some property of it? Do repeated sentences in the text show any regularity? Studies of this kind on textual data fall in the *combinatorics on words* field of research.

On the other hand, during the process of writing down this manuscript, a periodic commit on GitHub was done. This resulted in tens of versions of this thesis, all differing very little from each other, hence contained in an extremely *repetitive* repository.

Similarly, one may need to store hundreds of textual documents. They would need a lot of storage, and some sort of textual compression would be necessary. Additionally, one may also want to search for some term in one or all these documents, or in many of them, therefore, being able to look for specific terms in them would be helpful. There exist *algorithms* and *data structures* for efficiently solving problems such as: compressing and decompressing textual data, answering whether a pattern occurs in the text, locating the pattern, or counting how many times it occurs.

Let us explain all this more in detail.

Combinatorics on words is a field of discrete mathematics studying *words*. Words (or *texts*) are defined as finite or infinite sequences of symbols taken from a set. The first systematic collection of results in this field can be found in the book *"Combinatorics on words"* [55] written by a group of mathematicians under the pseudonym of M. Lothaire in 1983. Before this book, works on words, such as [3, 66, 67, 68], were usually more for supporting the research in other areas (e.g. Mathematics and Computer Science), rather than the main topic. Even when the theory on words became an area of its own, the area of combinatorics on words has remained strictly connected to that of computer science when automata, formal languages, and textual data are involved, in particular, to the branch of *theoretical computer science*.

Another aspect of computer science related to textual data is the need to compress and use the data in compressed form. As a matter of fact, the applications where the data is growing the fastest nowadays, thus requiring the most to be handled in compressed form, are those dealing with highly repetitive textual data (as the simple example of the GitHub repository of this thesis, but with much larger amounts of textual data). Usual statistical compression methods for collections are based on the statistical entropy defined by Shannon [79]. However, measures defined on Shannon's entropy are not able to capture the repetitiveness of the text. Thus, methods that can measure the repetitiveness of textual data have been attracting considerable attention both from a theoretical and combinatorial point of view, as well as from the more applied research communities interested in exploiting string repetitiveness for more efficient compression. Two widely used text compression techniques that can also be used to measure the repetitiveness of a text to evaluate its compressibility are the *Burrows-Wheeler Transform* and *Lempel-Ziv factorization*. (A detailed overview of these and other string repetitiveness measures can be found in [69].) We will explain them in the following.

The Burrows-Wheeler Transform (BWT) was introduced in 1994 by M. Burrows and D.J. Wheeler. This transform produces a permutation of the characters of the text which tends to be easier to compress than the input. The input text can be easily recovered, and this is one of the reasons why the BWT has become fundamental for string compression and indexing. The key point of the transform is that the characters are rearranged in such a way that occurrences of the same characters tend to be grouped together. Groups of equal consecutive characters are usually called *equal-letter runs* or just *runs*. For instance, the BWT of the word `abracadabra` is `rdarcaaaabb`. All the occurrences of `b` appear together, thus producing one run, while the occurrences of `r` remain separated, producing one run each. In this way, the number of runs goes from 11 of `abracadabra` (equal to its length) to 7 of its BWT `rdarcaaaabb`. The clusterization happens because the transform sorts the rotations of the input text, and extracts and concatenates the preceding character (thus the last) of each sorted rotation. This is easier to notice when the BWT is seen as the last column of a matrix consisting of all lexicographically sorted rotations of the word, as in Figure 1.1. Due to the sorting, rotations starting with the same repetition occurring within the text are grouped together (e.g. rotations starting with `bra` in Figure 1.1). Since repetitions often have a common preceding character, long runs in the BWT of the text are created (in the example, both occurrences of `bra` are preceded by `a`). However, usually, not all occurrences of a character are actually moved one after the other in the BWT. One can notice in the example that most occurrences of `a` appear together in the BWT, but one occurrence remains separated, while both `b`'s are moved one after the other. This effect is commonly known as *clustering effect* of the BWT.

rotations of `abracadabra`	BWT
`aabracadabr`	`r`
`abraabracad`	`d`
`abracadabra`	`a`
`acadabraabr`	`r`
`adabraabrac`	`c`
`braabracada`	`a`
`bracadabraa`	`a`
`cadabraabra`	`a`
`dabraabraca`	`a`
`raabracadab`	`b`
`racadabraab`	`b`

Figure 1.1: The BWT matrix of the word `abracadabra`.

Originally, the BWT was presented as a preprocessing step to the move-to-front compression technique, but its tendency of clustering together equal characters in runs makes it suitable for *run-length encoding*, (RLE). This compression scheme consists of replacing each run with a pair containing a single character and the number of consecutive occurrences of the character in that run. For example, the RLE applied on the example `rdarcaaaabb` results in $(\mathtt{r}, 1)(\mathtt{d}, 1)(\mathtt{a}, 1)(\mathtt{r}, 1)(\mathtt{c}, 1)(\mathtt{a}, 4)(\mathtt{b}, 2)$. Due to the tendency of the BWT to have long runs of the same character, the RLE applied on the BWT of a text usually results in a higher compression than applying it directly on the text itself. This can already be seen in the short example, where the RLE applied on `abracadabra` would produce three more pairs than when applied on its BWT.

The other compression technique we discuss was introduced by A. Lempel and J. Ziv in 1977 (LZ77). The idea is to exploit repetitions in the text and substitute new occurrences with a pointer to a previous occurrence. Analogously to the BWT, this technique works particularly well when the input is very repetitive. In particular, very long repetitions in the word allow replacing many characters with just one pointer character-position referring to a previous occurrence of that repetition. Many variants of this compression scheme have been studied and developed so far [45, 70, 72, 76, 83, 85], to name just a few.

One field that deals with extremely redundant textual data, and where both BWT and LZ are widely used, is *bioinformatics*. Bioinformatics tackles problems of collecting, storing, analyzing, and disseminating biological data.

Often, compression schemes are also used as an index for large texts, exploiting the high rate of repetitions in them. An *index* is a data structure that allows finding a specific pattern in a larger text, locating the pattern or counting the number of its occurrences, depending on the type of the index. The index is built beforehand and permits speeding up search queries in the original text. In fact, BWT and LZ are not only used to compress data, but also to index texts to allow easier and faster queries of a pattern on them.

A breakthrough for BWT was the introduction of the FM-index [26] which is the base of some of the widely used bioinformatics tools for genome alignment, such as `bwa` [51], `bowtie` [49], and SOAP2 [48]. Although the BWT can be stored and queried in compressed space [60], the size of the FM-index may grow with the length of the uncompressed text. Later, the *r-index* [30] was introduced. The r-index is also a BWT-based index able to handle hundreds of human genomes, and whose size grows with r of the input.

1.1 Contributions

This thesis is mainly devoted to the study of combinatorial properties of words with respect to their BWT, and of words that are the BWT of some other word. We also treat an application of BWT in bioinformatics, and a combinatorial study on LZ77 related to bioinformatics.

The first research direction we discuss deals with the BWT in relation to a comparison between a text and its reverse. The BWT is known to tend to cluster together equal characters of the input text in long runs of the same character, especially when these occurrences appear in parts of the text repeated many times. This tendency makes the BWT, and in particular the number of runs in the BWT, suitable to measure the number of repetitions in the input text: the fewer and longer the runs in the BWT, the higher the repetitiveness of the input. Intuitively, a word and its reverse are equally repetitive, i.e. a word read left-to-right contains the same number of repetitions as the word read right-to-left.

The first research results of this thesis are in this direction, showing that, even though the repetitiveness is preserved when a word is reversed, r may not be. We focus on the parameter ρ, or *runs-ratio*. This is the ratio between the number r of runs of the BWT of a text and that of the BWT of its reverse. The first upper bound $\rho = \mathcal{O}(\log^2 n)$ was given in [43]. Two questions that remained open were whether this bound was tight, and whether there existed examples with $\rho = \omega(1)$, i.e. ρ non-constant. We give a first answer to these questions by exhibiting an infinite family of binary words whose members satisfy $\rho = \Theta(\log n)$, where n is the length of the word.

Another direction we took is related to how r changes after a one-character modification is performed on the word. This question was already treated for LZ78 [47], where it was shown that prepending just one character to a word may significantly increase the size of the LZ78 factorization of the word. This event is known as the *one-bit catastrophe*. We used this notion loosely considering one-*character* changes rather than one-*bit* changes.

In close relation to this, one could think that such small changes in the word do not affect much the repetitiveness of the word and thus the number of runs of its BWT. We show that there exist words for which prepending, appending, or inserting a single character increases r by a multiplicative logarithmic factor in the length n of the word, going from 2 to $\Theta(\log n)$ runs. Moreover, also deleting or substituting one character can produce the same effect.

Usually, in bioinformatics applications of the BWT, a so-called *sentinel character* \$ is involved. This character is set to be smaller than any character of the word, and it is implicitly added at the end of each sequence. It permits marking the end of the sequence, and it also allows computing the BWT sorting the suffixes of the sequence instead of sorting its rotations. Our results on the non-constant increment of r when a character smaller than all characters of the word is added may be of particular interest in this context.

Another topic treated in this thesis combines the essential role of the character \$ and the interest in deciding whether, given a data structure, there exists a word on which that data structure is built. Characterizations of words that are BWT of some other word over the same alphabet already exist, both for binary alphabets [64], and for general alphabets [52], and show that whether a word w is a BWT depends on a specific permutation of the indices of the word. This permutation is called the *standard permutation* of the word. In this context, we characterized positions in a word w that allow inserting \$ to turn w in a BWT of some \$-terminated word, and we call these *nice positions*. In other words, those are positions that change the standard permutation of w in such a way that it becomes the standard permutation of the BWT of a \$-terminated word.

Additionally, we provide an efficient algorithm that, given a word, returns all nice positions, and we use it to produce some statistics on the number of words with a given number of nice positions.

We finally focus on *fully clustered words* over a binary alphabet. As the name suggests, fully clustered words are words in which all occurrences of a character appear together. The study of nice positions of these words attracted our interest because fully clustered words are the simplest form of BWT output. They were treated in several works [25, 64, 80], where words whose BWT is a fully clustered word are characterized. In [64], the authors show that binary fully clustered words are exactly the BWT of powers of rotations of standard words. In [25, 80] larger alphabets were considered. We investigate the number of nice positions of a fully clustered binary word w, and the number of words which have k nice positions, for a given length n.

Finally, we present two applications of string compressors in bioinformatics.

The first is an application of the BWT. With the advent of third-generation sequencing, the quality of assembled genomes drastically increased, and with it, the availability of these data grew. One important step to enable the use of these high-quality assembled genomes is to build a multiple sequence alignment of the genomes. Tools like MUMmer [46, 65], and Mauve [18] proposed a solution to the original problem of multiple sequence alignment by using

Maximal Unique Matches (MUMs) between two input sequences as prospective anchors for an alignment. MUMs are long stretches of the genomes that are equal in both genomes and occur only once in each. MUMs have also been proven useful for strain level read quantification [86], and as a computationally efficient genomic distance measure [21]. We present an extension of an already existing algorithm to compute Maximal Exact Matches (MEMs), called PHONI [8]. We store $\mathcal{O}(r)$ samples of the LCP array additionally with respect to the original implementation. With the additional samples, we are able to compute enhanced Matching Statistics that permit computing MUMs in the same asymptotic time and space as MEMs in PHONI.

For the second application we used LZ77. We give a model for DNA short reads sequencing, and we give a bound for the size of its LZ77 factorization. The wide adoption of high-throughput sequencing in medical and evolutionary biology over the last decade has made short read data sets abundant and very large. The de facto standard in most labs and large institutions is to compress such files with the `gzip` all-purpose file compressor, which usually leads to a still relatively large file. However, to the best of our knowledge, no careful analysis of the compressibility of short read data sets—even in an idealized setting—has been undertaken. We consider the problem of compressing a set of substrings regularly sampled from a string: Given a string X and two integer parameters m and d, we sample the string X extracting substrings of length m at a regular distance d one to the other. We give an upper bound on the size of the LZ77 factorization of the concatenation of the extracted samples in terms of the length of X, the two parameters m and d, and the compression size of the original string X. Finally, we also show a different upper bound on the size of the factorization that holds regardless of the order in which the samples are concatenated.

1.2 Organization of the thesis

Chapter 2 We first give the basic definition on words and permutations that will be used in the manuscript. We then give the definition of the BWT, explaining some of its properties, such as how the original word can be recovered from its BWT. We also introduce a widely studied family of words presenting a particular form of BWT. We then briefly explain two variants of LZ, i.e. LZ77 and LZ78, and we show how to encode and decode strings via these compressors.

Chapter 3 We answer the question of how much the number of runs of the BWT of a text may differ from the BWT of the same text read back-to-front, showing in detail the structure of the BWT of both the forward and the reverse of a particular family of words whose number of runs changes logarithmically in its length.

Chapter 4 We state that the BWT suffers from the so-called *one-bit catastrophe*, showing that the number of runs in the BWT of a word may increase by a logarithmic factor in the length of the word when any kind of edit operation is conducted on the word: inserting, deleting or substituting a single character.

Chapter 5 This chapter is devoted to the problem of inserting the special character \$ in a word in order to make it the BWT of another word over the same alphabet. The \$ is the so-called *sentinel character*, used in many applications involving the BWT, to indicate the end of the input text. In this chapter, we show whether and where a \$ can be inserted in a text to make it the BWT of some text.

Chapter 6 We give two examples of applications, one for BWT and one for LZ77. In Section 6.1, we describe MUM-PHINDER, an algorithm to compute Maximal Unique Extensions (MUMs) of a query pattern over an index of hundreds of human genomes.

In Section 6.2 we present a model of DNA short reads. In our model, the reads fully cover the genome, and they have a known and fixed distance.

Chapter 7 Finally, we give a brief summary, and discuss some future ideas linked to this research field.

1.3 Publications

Several parts of this thesis have been published in conference proceedings or international journals.

1. Parts of Chapter 3 have appeared in
   S. Giuliani, S. Inenaga, Zs. Lipták, N. Prezza, M. Sciortino, and A. Toffanello, *Novel Results on the Number of Runs of the Burrows-Wheeler-Transform*, International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2021) [32],
   where the paper won the best paper award.

2. The contents of Chapter 4 is pubblished in
   S. Giuliani, S. Inenaga, Zs. Lipták, G. Romana, M. Sciortino, and C. Urbina, *Bit catastrophes for the Burrows-Wheeler Transform*, Developments in Language Theory (DLT 2023) [33].

3. The contents of Chapter 5 have led to three publications:

   i) the very first results on pseudo-cycles and the algorithm is published in
   S. Giuliani, Zs. Lipták, R. Rizzi, *When a Dollar Makes a BWT*, Italian Conference on Theoretical Computer Science (ICTCS 2019) [36],

   ii) an extended version appeared in the international journal Theoretical Computer Science S. Giuliani, Zs. Lipták, F. Masillo, R. Rizzi, *When a Dollar Makes a BWT*, Theoretical Computer Science, 2021 [35],

   iii) the narrowed definition of essential pseudo-cycles and the study on fully clustered words appeared in
   S. Giuliani, Zs. Lipták, F. Masillo, *When a Dollar in a Fully Clustered Word Makes a BWT*, Italian Conference on Theoretical Computer Science (ICTCS 2022) [34]

4. The contents of Chapter 6, Section 6.1 have been published in
   S. Giuliani, G. Romana, M. Rossi, *Computing Maximal Unique Matches with the r-index*, Symposium on Experimental Algorithms (SEA 2022) [37].

5. The contents of Chapter 6, Section 6.2 have been published in
   G. Badkobeh, S. Giuliani, Zs. Lipták, S. J. Puglisi, *On Compressing Collections of Substring Samples*, Italian Conference on Theoretical Computer Science (ICTCS 2022) [4].

Chapter 2

Technical Background

In this chapter, we introduce the basic concepts needed in the thesis. Most of the terminology used concerning words refers to the books [54, 55].

2.1 Strings and permutations

In the following, we give the necessary terminology and notation on strings and permutations.

Strings Let $\Sigma = \{c_1, c_2, \ldots, c_\sigma\}$ be a finite ordered alphabet. The elements of the alphabet are called *characters*, and σ is its size. A *string* (or *word*) $v = v[0]v[1] \cdots v[n-1]$ (or just $v = v[0..n-1]$) over Σ is a sequence of n characters from Σ. We denote by $|v|$ the length of the word v, and by $|v|_c$ the number of occurrences in v of character $c \in \Sigma$. The unique string of length 0 is denoted by ε. We denote by Σ^* the set of all words over the alphabet. For two strings $x, y \in \Sigma^*, v = x \cdot y$ (or just $v = xy$) denotes the concatenation of the words x and y. Given $v = xuy$, x, u, y are called *factors* (or *substrings*) of v. In particular, x is a *prefix* and y is a *suffix* of v. We refer to the suffix of v starting in position i as $\mathit{suf}_i(v)$, and to the prefix of v ending in position j as $\mathit{pref}_j(v)$. Maximal substrings of equal consecutive characters in a word are called *runs*.

The *lexicographic order* on Σ^* is defined by: $v < v'$ if either v is a proper prefix of v', or there exists a $u \in \Sigma^*$ and characters $c, c' \in \Sigma$ such that uc is a prefix of v and uc' is a prefix of w, where $c < c'$. Given two words v and v' of length n such that $v[0] = v'[n-1], v[1] = v'[n-2], \ldots, v[n-1] = v'[0]$, then v' is the *reverse* of v, and we write $v' = v^{\text{rev}}$. A *palindrome* is a word v such that $v = v^{\text{rev}}$. Given two words $v = xy$ and $v' = yx$, then we say that v and v' are *conjugate*, and we write $\mathit{conj}_{|x|}(v) = v'$. We can also refer to v' as a *rotation* of v. For instance, the word `ationrot` is the third conjugate of the word `rotation`, i.e. $\mathit{conj}_3(\texttt{rotation}) = \texttt{ationrot}$. Conjugacy between words is an equivalence relation over Σ^*. A word u is a *circular factor* of v if it is the prefix of some conjugate of v. In a word v, a circular factor u is called *left-special* if, for some $c, c' \in \Sigma$, both cu and $c'u$ occur as circular factors of v.

For an integer $k \geq 1, u^k = u \cdots u$ is the kth power of u. A word v is called *primitive* if $v = u^k$ implies $k = 1$. A word v is primitive if and only if it has exactly $|v|$ distinct conjugates. Let v be primitive and the lexicographically smallest of its rotations, then v is called a *Lyndon word*. In the earlier example, the conjugate `ationrot` is the Lyndon conjugate of the word `rotation`.

In many applications, a special character is used to mark the end of the string, commonly denoted \$. The \$-character is assumed not to occur anywhere else in the string, and it is set to be smaller than all other characters. It will be explicitly indicated when a string ends with \$, and we call such a string *\$-terminated*.

The *longest common prefix* (lcp) of two words v, v' is the maximum length word u such that u is a prefix both of v and v'. Additionally, $\text{LCP}(v, v') = |u|$ gives the length of the lcp of v and v'. Finally, we are interested in the lcp length of two rotations of the same word v. Let $v = xuyuz$ such that $y[0] \neq z[0]$, and $|x| = i - 1, |xuy| = i' - 1$, then $\text{lcp}_v(i, i') = |u|$. Finally, we need an array of length $|v|$ containing the length of the longest common prefix between consecutive suffixes of v in lexicographic order, that is $\text{LCP}_v[i] = \text{lcp}_v(j, j')$ where $suf_j(v)$ and $suf_{j'}(v)$ are the i^{th} respectively the $(i+1)^{st}$ suffixes of v lexicographically. As an example, let $v =$ `rotation` and $v' =$ `rotated`. Then, $\text{lcp}(v, v') =$ `rotat` and $\text{LCP}(v, v') = 4$, while $\text{LCP}_v[6] = \text{lcp}_v(2, 4) = 1$ because $\text{lcp}(suf_2(v), suf_4(v)) =$ `t`, and $suf_2(v) =$ `tation` is the 6^{th} suffix of v in lexicographic order of v and $suf_4(v) =$ `tion` is the 7^{th}.

Permutations Let n be a positive integer. A *permutation* of n is a bijection from $\{0, 1, \ldots, n-1\}$ to itself. Permutations are often written using the two-line notation $\left(\begin{smallmatrix} 0 & 1 & \cdots & n-1 \\ \pi(0) & \pi(1) & \cdots & \pi(n-1) \end{smallmatrix}\right)$, or the one line notation $[\pi(0), \pi(1), \ldots, \pi(n-1)]$. A *cycle* in a permutation π is a minimal subset $C \subseteq \{0, \ldots, n-1\}$ with the property that $\pi(C) = C$. A cycle of length 1 is called a *fixpoint*, and one of length 2 a *transposition*. Every permutation can be decomposed uniquely into disjoint cycles, giving rise to the *cycle representation* of a permutation π, i.e. as a composition of the cycles in the cycle decomposition of π. For example, $\pi = [3, 1, 4, 5, 2, 0] = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 3 & 1 & 4 & 5 & 2 & 0 \end{smallmatrix}\right) = (0\ 3\ 5)(1)(2\ 4)$. Permutations whose cycle decomposition consists of just one cycle are called *cyclic*.

A fundamental theorem about permutations says that every permutation π can be written as a product (composition) of transpositions, and that the number of any sequence of transpositions whose product is π is either always even or always odd: this is called the *parity* of the permutation. The *sign* $sgn(\pi)$ of a permutation π is defined as 1 if π is even, and as (-1) if it is odd; equivalently, $sgn(\pi) = (-1)^m$, where $\pi = \prod_{i=1}^m \tau_i$ for some transpositions τ_i.

The sign $sgn(C)$ of a cycle C of m elements is $(-1)^{m-1}$, since any cycle $C = (x_0, \ldots, x_{m-1})$ can be written as $C = (x_0, x_1)(x_1, x_2) \cdots (x_{m-2}, x_{m-1})$, thus C is the product of $m - 1$ transpositions. Moreover, if $\pi = \prod_{i=1}^c C_i$ is the cycle decomposition of permutation π of $\{0, \ldots, n-1\}$, then $sgn(\pi) = \prod_{i=1}^{c} sgn(C_i) = (-1)^{n-c}$.

2.1.1 Fundamental permutations on strings

There are three fundamental permutations on strings that we will frequently encounter in this thesis. The *standard permutation* π_v of the word v is a permutation of length $|v|$ defined as follows: $\pi_v[i] < \pi_v[j]$ if and only if either $v[i] < v[j]$ or $v[i] = v[j]$ and $i < j$. For example, the standard permutation of the word `thisisathesis` is $\pi = [11, 2, 4, 7, 5, 8, 0, 12, 3, 1, 9, 6, 10]$.

The *conjugate array* CA_v of the word v is the array of length $|v|$ containing all indices of v according to the lexicographic order of the conjugates of v. More precisely, $\mathrm{CA}_v[i] < \mathrm{CA}_v[j]$ if and only if $conj_i(v) < conj_j(v)$ or $conj_i(v) = conj_j(v)$ and $i < j$. Clearly, if v is not primitive, then at least two conjugates are equal (second case of the definition). The conjugate array of `thisisathesis` is $\mathrm{CA} = [6, 9, 8, 1, 4, 2, 11, 5, 3, 10, 12, 7, 0]$.

The third permutation we are going to use is the *suffix array* SA_v of the word v. It is defined as follows $\mathrm{SA}_v[i] < \mathrm{SA}_v[j]$ if and only if $suf_i(v) < suf_j(v)$. The suffix array of `thisisathesis` is $\mathrm{SA} = [6, 9, 8, 1, 11, 4, 2, 12, 5, 10, 3, 7, 0]$. Note that there are cases when $\mathrm{CA} = \mathrm{SA}$, but it is not true in general, as one can see in the example. We will see later in Section 2.2.1 that if v is \$-terminated, then $\mathrm{CA} = \mathrm{SA}$.

When the word v is clear for the context, then we refer to π_v, CA_v and SA_v by π, CA, and SA, respectively.

Using the SA, we can express $\mathrm{LCP}_v[i]$ as the lcp between the suffix of v starting in position $SA[i]$ and the immediately smaller suffix: $\mathrm{LCP}_v[i] = \mathrm{lcp}_v(suf_{\mathrm{SA}[i]}, suf_{\mathrm{SA}[i-1]})$.

2.2 The Burrows-Wheeler Transform

The *Burrows-Wheeler Transform* (BWT), was introduced by Burrows and Wheeler as a first step for a lossless text compression algorithm in a technical report in 1994 [9]. It is at the heart of several lossless data compressors, such as *bzip* [78], and of several text indices, especially in bioinformatics. An example is the FM-index [26] on which some of the most commonly used bioinformatics tools are based, such as `bwa` [51], `bowtie` [49], and SOAP2 [48].

The BWT is a reversible transform of the input string producing a permutation of the input characters. The transform tends to reorganize the characters of the input in such a way that it makes it easier to compress the output than the original string. Occurrences of the same characters tend to be grouped together, and this works well especially with highly repetitive texts. Due to its reversibility, it is possible to recover the original string, and this can be done in linear time.

Let us define the BWT. For a word v of length n, the output of the BWT consists of a word w and an index i such that w is the concatenation of the last character of the rotations sorted lexicographically, and i is the rank of the original word v in the list of the sorted rotations. Formally,

Definition 1 *Let v be a word of length n over the alphabet Σ. The* Burrows-Wheeler Transform *(BWT) of v is a pair (i, w) consisting of an integer i and a word of length n such that for $0 \leq j \leq n-1$, $w[j] = v[(CA_v[j]-1) \bmod n]$, and v is the i^{th} conjugate in lexicographic order.*

The BWT of the word v can also be visualized as follows. Consider the matrix M of size $n \times n$ consisting of all conjugates of v sorted lexicographically (see Figure 2.1). We call this matrix the *BWT matrix* of the word v. In the figure, on the left of the matrix, the indices of the starting position in the original word of each conjugate are shown (i.e. the conjugate array, or CA). Considering the rows and the columns in the matrix, they are all permutations of the characters of the word. In particular, each row is a conjugate of the input word, and the first column consists of all characters of the word sorted lexicographically. The output of the BWT of the word v is the word w and the index i such that w is the last column of the BWT matrix, and i is the rank of the row containing v. Recall that maximal substrings of consecutive equal characters in a word are called *runs*. The fact that the BWT of a word tends to have fewer and longer runs than the original word is the key point of the transform. We define $runs(v)$ as the number of runs of the word v and $r(v) = runs(\text{BWT}(v))$ the number of runs of the BWT of v.

Note that, the index i is needed to recover the exact rotation of v from its BWT (as explained in detail in Section 2.2.2). In this thesis we are interested in recovering v up to rotation, therefore we do not need to store the index i. We are going to refer as BWT(v) only to the characters permutation representing the last column of the BWT matrix, ignoring the index indicating the rank of v in CA.

It can be noticed in Figure 2.1 that occurrences of the same character tend to be grouped together in the BWT of v. This is the so-called *clustering effect* of the BWT. This effect has an easy explanation: consider the repetition `s` occurring in the example. That is a suffix of length 1 of the repetition `is` of length 2 also occurring in the example. All rotations starting with some occurrence of `s` will be grouped together in the BWT matrix by construction. Those `s` that are suffixes of `is` are the prefix of length 1 of rotations preceded by the character `i`. If some occurrence of `s` is preceded by some other character, this may break the run of `i` in the BWT of the block of rotations starting with `s`. In the example, the occurrence of `es` breaks the runs of `i` in the last column, but most of the `s` still appear together, producing longer runs of `i` in the BWT than in the original word.

Note that, when a word containing occurrences of exactly two distinct characters is considered, the number of runs in its BWT must be even. The

	CA	`thisisathesis`	BWT
0	6	`athesisthisis`	`s`
1	9	`esisthisisath`	`h`
2	8	`hesisthisisat`	`t`
3	1	`hisisathesist`	`t`
4	4	`isathesisthis`	`s`
5	2	`isisathesisth`	`h`
6	11	`isthisisathes`	`s`
7	5	`sathesisthisi`	`i`
8	3	`sisathesisthi`	`i`
9	10	`sisthisisathe`	`e`
10	12	`sthisisathesi`	`i`
11	7	`thesisthisisa`	`a`
12	0	`thisisathesis`	`s`

Figure 2.1: The BWT matrix of the word `thisisathesis`.

reason is that the BWT of these words cannot start with the smallest character in the alphabet (let us say `a`), nor end with the largest (`b`). Let us consider a word v of length n of this type (i.e. consisting of occurrences of `a` and `b`), with $w = \text{BWT}(v)$, and $w[0] = \texttt{a}$. This would imply that there exists a conjugate $conj_i(v) = v[i..n-1]v[0..i-1]$ starting with $v[i] = w[0] = \texttt{a}$, and circularly followed by the lexicographically smallest conjugate of v. But this cannot happen because the $conj_{i+1}(v) = v[i+1..n-1]v[0..i]$ ending with $v[i] = w[0] = \texttt{a}$ cannot be smaller that $conj_i(v) = \texttt{a}v[i+1..n-1]v[0..i-1]$. The analogous reasoning can be done for $w[n-1] = \texttt{b}$.

2.2.1 The role of the \$ character

Often, in real-life applications, a special character is used to mark the end of the string. In the area of data structures for bioinformatics, the character used is \$, since it does not occur in the strings considered. It is set to be smaller than any character of the alphabet. The unique occurrence of \$ guarantees that no rotation occurs twice (the word is primitive), and also that no suffix is a prefix of some other prefix. In fact, when v is \$-terminated the lexicographic order of the conjugates of the word is equal to the lexicographic order of the suffixes, therefore $\text{CA}_v = \text{SA}_v$. For this reason, the BWT can also be built from the SA of a word.

Let us now define $\text{BWT}(v)$ over the suffixes as follows:

Definition 2 *Let v be a word of length n over the alphabet. The* Burrows-Wheeler Transform *of v is a word $w = BWT(v)$ such that $w[i] = v[n-1]$ if $SA[i] = 0$, otherwise $w[i] = v[SA[i]-1]$.*

In Figure 2.2, we see that the BWT matrices of the word `thisisathesis` built over the CA and the SA are the same when \$ is appended at the end of the word.

	CA	`thisisathesis$`	BWT
0	13	`$thisisathesis`	`s`
1	6	`athesis$thisis`	`s`
2	9	`esis$thisisath`	`h`
3	8	`hesis$thisisat`	`t`
4	1	`hisisathesis$t`	`t`
5	11	`is$thisisathes`	`s`
6	4	`isathesis$this`	`s`
7	2	`isisathesis$th`	`h`
8	12	`s$thisisathesi`	`i`
9	5	`sathesis$thisi`	`i`
10	10	`sis$thisisathe`	`e`
11	3	`sisathesis$thi`	`i`
12	7	`thesis$thisisa`	`a`
13	0	`thisisathesis$`	`$`

(a)

	SA	`thisisathesis$`	BWT
0	13	`$`	`s`
1	6	`athesis$`	`s`
2	9	`esis$`	`h`
3	8	`hesis$`	`t`
4	1	`hisisathesis$`	`t`
5	11	`is$`	`s`
6	4	`isathesis$`	`s`
7	2	`isisathesis$`	`h`
8	12	`s$`	`i`
9	5	`sathesis$`	`i`
10	10	`sis$`	`e`
11	3	`sisathesis$`	`i`
12	7	`thesis$`	`a`
13	0	`thisisathesis$`	`$`

(b)

Figure 2.2: When the \$ is appended at the end of the word `thisisathesis` the CA and the SA coincide, and the corresponding BWT matrices as well.

In contrast, considering `shttshsiieias`, which is the BWT of the word `thisisathesis` (see Figure 2.1), it does not have a clear relationship with the word `sshttsshiieia`, which is the result of removing the \$ from the BWT of `thisisathesis$` (see Figure 2.2).

Additionally, when a word ends with \$, the output of the BWT is just the word w from Definition 2. The original word can be recovered starting from the position where the \$ character occurs in w.

2.2.2 Recovering the original word

From the BWT of a word, it is possible to produce the original word in linear time [9]. To recover the word, some properties of the BWT are essential, which are described in the following:

Proposition 1 (BWT properties [9]) *Given a word v of length n, and the BWT-matrix of v, let F be the string consisting of the concatenation of the characters in the first column of the matrix, and L the concatenation of the characters in the last. Then*

i) F is the list of the characters of v sorted lexicographically,

ii) L is the output string,

iii) each row of the matrix is a rotation of v,

iv) for all $i < n$, $F[i]$ is cyclically preceded by $L[i]$ in v,

v) for any character c, the i^{th} occurrence of c in F corresponds to the i^{th} occurrence of c in L (LF-property). For example, in Figure 2.1 the *second* `s` *in the L (i.e. the BWT), is the character preceding the suffix* `athesis$`*, and the second* `s` *in F is also preceding the suffix* `athesis$`*.*

The LF-property produces the so-called *LF-mapping* that gives a correspondence between characters from the *L*ast and the *F*irst column of the BWT matrix. This mapping is the standard permutation π_L of $L = \text{BWT}(v)$. Recalling from Section 2.1.1, π_w is a permutation of the indices of w such that $\pi_w[i] < \pi_w[j]$ if and only if either $w_i < w_j$ or $w_i = w_j$ and $i < j$.

The idea behind the reconstruction of the word is described in the following, and works the same with or without the index of the correct rotation of the word we want, and with or without $. In case we stored the index i indicating the desired rotation of the original word, we start recovering the word from position i in L. Otherwise (i.e. we did not store i), we can start recovering it from any character of the BWT. In case the $ character is used, we start from the position of $. Once we have the starting character for recovering the word, we start building it back-to-front. This means that the character will be the last character $v[n-1]$ of the rotation of the word we are recovering. Now, we use the LF-mapping (i.e. π_w) to know the position of $v[n-1]$ in the lexicographically sorted characters of the word, thus in the string F. Once we get that position, let us say j, we can deduce the preceding character. This is clear from the BWT matrix, where each row is a rotation of the word, and therefore the first character of a row is preceded by the character in the BWT in the same row (see Proposition 1). Going on with this procedure, a rotation of the original word will be recovered.

Let us show with the example $v =$ `thisisathesis$` in Figure 2.2a how to recover the word from its BWT $w =$ `sshttsshiieia$`. The standard permutation of w tells us in which position $i = \pi[j]$ in F we can find the character in position j in BWT: $\pi_w = [8, 9, 3, 12, 13, 10, 11, 4, 5, 6, 2, 7, 1, 0]$. Since we have the end-of-string indicator, we can start recovering the word back-to-front from position 13 where we have $v[13] = \$$. The character $ is the smallest character in the word, therefore we can easily recover the previous character in v, namely the very first character $w[0] =$ `s` in w. So now we have $v[12..13] =$ `s$`. We can now exploit $\pi[0]$, which tells us where to find in F the occurrence of `s` that we just wrote, therefore which rotation in the BWT matrix starts with that occurrence of `s`. The rotation we are looking for is the $\pi[0] = 8^{th}$ one, which ends with $w[8] =$ `i`. We have now the third last character of v. Then we go on and we can write:

$$\begin{aligned}
v[10] &= w[\pi[8]] = w[5] = \texttt{s}\\
v[9] &= w[\pi[5]] = w[10] = \texttt{e}\\
v[8] &= w[\pi[10]] = w[2] = \texttt{h}\\
v[7] &= w[\pi[2]] = w[3] = \texttt{t}\\
v[6] &= w[\pi[3]] = w[12] = \texttt{a}\\
v[5] &= w[\pi[12]] = w[1] = \texttt{s}\\
v[4] &= w[\pi[1]] = w[9] = \texttt{i}\\
v[3] &= w[\pi[9]] = w[6] = \texttt{s}\\
v[2] &= w[\pi[6]] = w[11] = \texttt{i}\\
v[1] &= w[\pi[11]] = w[7] = \texttt{h}\\
v[0] &= w[\pi[7]] = w[4] = \texttt{t}
\end{aligned}$$

As one can see, if we do another round of this and we check which character precedes the last `t` we wrote in v, it is actually $v[n-1] = w[\pi[4]] = w[13] = \$$.

2.2.3 Fully clustered words and BWT

The power of the BWT is the tendency to cluster together occurrences of the same character in runs. Usually, the more repetitive the input string, the fewer the runs. There are words for which the BWT tends to cluster the characters better than other words.

Words consisting of exactly one run for each character appearing in the word are called *fully clustered words*. More formally, let $alph(w)$ be the set of distinct characters in w, then w is fully clustered if $runs(w) = |alph(w)|$.

For instance, the word $\texttt{bbbaaaaa} = \texttt{b}^3\texttt{a}^5$ is a binary fully clustered word, and $\texttt{cbbbbaaaaa} = \texttt{cb}^4\texttt{a}^5$ is a fully clustered word over an alphabet of size 3. Those words are of special interest for their property of being particularly easy to compress with RLE-based compressors. Words whose BWT is a fully clustered word have been studied under different names in [25, 73, 74, 80, . . .]. In particular, Simpson and Puglisi [80] treat ternary words v whose BWT is a fully clustered word $w = BWT(v)$. They divide the pre-images v in 3 types on the basis of the form of their BWT: $\texttt{c}^i\texttt{b}^j\texttt{a}^k$ is of *Type I*, $\texttt{c}^i\texttt{a}^k\texttt{b}^j$ is of *Type II*, $\texttt{b}^j\texttt{c}^i\texttt{a}^k$ is of *Type III*. They show that there exists a characterization via morphisms of words whose BWT is of Type I. In [74], a combinatorial property (circular palindromic richness) was shown to be necessary but not sufficient condition for words having a fully clustered BWT of Type I over arbitrary alphabets of size σ, i.e. for words whose BWT has the form $x_\sigma^{m_\sigma} x_{\sigma-1}^{m_{\sigma-1}} \cdots x_1^{m_1}$, with $x_1 < \ldots < x_{\sigma-1} < x_\sigma \in \Sigma$. Finally, a characterization of words with fully clustered BWT words over arbitrary alphabet was given in [25] in terms of interval exchanges.

2.2.3.1 BWT of standard words

Of particular interest for this thesis are binary fully clustered words. In fact, these words are precisely the BWT of a widely studied family of words called *standard words* (see e.g. [64, 74]). Next, we introduce in detail standard words, and we follow [57] to discuss these words.

Definition 3 (Standard words) *Given an infinite sequence of integers $(d_0, d_1, d_2, \ldots)$, with $d_0 \geq 0, d_i > 0$ for all $i > 0$, called a* directive sequence, *define a sequence of words $(s_i)_{i\geq 0}$ of increasing length as follows: $s_0 = \mathtt{b}, s_1 = \mathtt{a}$, and for $i \geq 1$, $s_{i+1} = s_i^{d_{i-1}} s_{i-1}$. The words s_i are called* standard words, *and the index i is referred to as the* order *of s_i.*

In the following, we present some properties of standard words that will be used later in this thesis (see [6, 7, 56, 58]).

Proposition 2 (Some known properties of standard words) *Let s_i be a standard word of with directive sequence $d = (d_0, \ldots, d_{i-2})$, and $i > 2$. The following properties hold:*

1. *let s'_i be a standard word with directive sequence $d = (0, d_0, \ldots, d_{i-2})$, then s'_i can be obtained from s_i by interchanging the character $\mathtt{a}$ with the character $\mathtt{b}$,*

2. *for all $k \geq 1$, $s_{2k} = x_{2k}\mathtt{ab}$ and $s_{2k+1} = x_{2k+1}\mathtt{ba}$, where x_{2k} and x_{2k+1} are palindromes ($x_2 = \varepsilon$) and are called* central words,

3. *for all $k \geq 2$,*
   - $s_{2k} = x_{2k-1}\mathtt{ba}x_{2k-2}\mathtt{ab} = x_{2k-2}\mathtt{ab}x_{2k-1}\mathtt{ab}$,
   - $s_{2k+1} = x_{2k}\mathtt{ab}x_{2k-1}\mathtt{ba} = x_{2k-1}\mathtt{ba}x_{2k}\mathtt{ba}$.

Note that, by Proposition 2, part 1, we can restrict ourselves to the case of $d_0 > 0$.

Standard words are used for the construction of infinite Sturmian words, in the sense that every characteristic Sturmian word is the limit of a sequence of standard words (cf. Chapter 2 of [54]). These words have many interesting combinatorial properties and appear as an extreme case in a great range of contexts [10, 11, 12, 44, 56, 58, 64, 77]. A fundamental result in connection with the BWT is the following: $\mathrm{BWT}(w) = \mathtt{b}^q\mathtt{a}^p$ with p, q co-prime if and only if w is a conjugate of a standard word [64]. This result implies that the BWT of standard words is a fully clustered word, and all fully clustered words over a binary alphabet are conjugates of some standard word.

Fibonacci words are a particular case of standard words, which are given by a directive sequence consisting of only ones $(1, 1, 1, \ldots)$, and of which the first few elements are as follows:

$$\begin{aligned}
s_0 &= \texttt{b}\\
s_1 &= \texttt{a}\\
s_2 &= s_1 s_0 = \texttt{ab}\\
s_3 &= s_2 s_1 = \texttt{aba}\\
s_4 &= s_3 s_2 = \texttt{abaab}\\
s_5 &= s_4 s_3 = \texttt{abaababa}\\
s_6 &= s_5 s_4 = \texttt{abaababaabaab}\\
s_7 &= s_6 s_5 = \texttt{abaababaabaababaababa}\\
s_8 &= s_7 s_6 = \texttt{abaababaabaababaababaabaababaabaab}\\
&\dots
\end{aligned}$$

Note that $|s_i| = F_i$, where F_i is the i^{th} Fibonacci number of the sequence defined by $F_0 = F_1 = 1$ and $F_{i+1} = F_i + F_{i-1}$. Moreover, the number of occurrences of the character $\texttt{a}$ in s_i is $|s_i|_{\texttt{a}} = F_{i-1}$ and the number of occurrences of the character $\texttt{b}$ is $|s_i|_{\texttt{b}} = F_{i-2}$, for $i \geq 2$.

Proposition 2 holds for Fibonacci words as well. In the following, we present some additional properties of Fibonacci words, some of which can be deduced from more general properties that hold for all standard words (see [6, 7, 56, 58]).

Proposition 3 (Additional known properties of Fibonacci words) *Let v_k be a Fibonacci word of order k*

1. *for $k > 2$, the standard word s with directive sequence $(1, 1, \dots, 1, 2)$ of length $k-2$ is a conjugate of the Fibonacci word of order k and directive sequence $(1, 1, \dots, 1)$ of length $k - 1$,*

2. *for $k > 1$, $s = x_{2k}\texttt{ba}$ is the standard word from part 1, where x_{2k} is the central word from Proposition 2, i.e. with directive sequence $(d_0, \dots, d_{2k-3})$ of length $2k - 2$, and $d_0 = \dots = d_{2k-4} = 1$, $d_{2k-3} = 2$,*

3. *for $k \geq 1$, $s = x_{2k+1}\texttt{ab}$ is the standard word from part 1, where x_{2k+1} is the central word from Proposition 2, i.e. with directive sequence $(d_0, \dots, d_{2k-2})$ of length $2k - 1$, and $d_{2k-2} = 2$, and for $k > 1$, $d_0 = \dots = d_{2k-3} = 1$,*

4. *for all $k \geq 2$, $\texttt{a}x_k\texttt{b}$ is a Lyndon word, where x_k is the palindrome from Proposition 2 .*

5. *for all circular factors y, z of s_i with $|y| = |z|$, and for each $c \in \Sigma$, one has that $||y|_c - |z|_c| \leq 1$ (Balancedness Property).*

Standard and Fibonacci words will be key concepts in Chapter 3 and Chapter 4.

2.3 The Lempel-Ziv factorization

In 1977, Lempel and Ziv introduced a new text compression technique based on a dictionary compression scheme [87]. The very next year, a different variant was published by the same authors [88], which was then followed by many others in the following years [45, 70, 72, 76, 83, 85, ...]. Lempel-Ziv based algorithms are at the basis of many widely used lossless data compressors, such as *GIF, PNG, ZIP, 7-Zip*.

The main idea, which slightly changes among variants, is to output a decomposition of the input text into non-empty factors that are either a character that never appeared before, or the longest substring that appeared already. The already occurred factors are thus encoded as a pointer to (one of) the other occurrence(s) position. The result of the algorithm is then an encoded factorization of the input string. For example, in the string `thisisathesis`, the substring `sis` occurs twice, once in position 3 and once in position 10. The occurrence in position 10 may be referred to as the pair $(3, 3)$, where the first 3 is the position in `thisisathesis` of the other occurrence of `sis`, and the second 3 is the length of the substring to encode. Clearly, the longer the repetitions in the string, the higher the compression.

The decompression simply consists of decoding each factor in string order one by one.

2.3.1 Variants

We now briefly explain the two variants that we will encounter later in the thesis, namely LZ77 [87] and LZ78 [88]. Given a string v, they both produce a factorization of the string where each factor f is either one character that never occurred before, or a pair used as a pointer to previously occurring factors. Let us see in detail how the pair is defined in the two versions.

2.3.1.1 LZ77

In this version of the compression algorithm [87], the longest substrings that already occurred in the string are encoded as a pair position-length. This pair is used as a pointer to the previous occurrence, and stores the position where the substring already occurred and the length of the repetition. Formally:

Definition 4 (LZ77 Factorization [87]) *The LZ factorization of v is a factorization $v = f_1 f_2 \dots f_z$ of v into* phrases *such that each phrase f_i (a substring of v) is either*

1. *a letter that does not occur in $f_1 \cdots f_{i-1}$, or*
2. *the longest substring that occurs at least twice in $f_1 \cdots f_i$.*

The number z of the phrases is the length of the LZ factorization of v.

Every factor f_i starting at position j in the string v is encoded with a pointer consisting of pair $(c, 0)$ if the character c has never occurred before, or of a pair $(\texttt{pos}, \texttt{len})$, such that `pos` is the position of a previous occurrence of f_i, and `len` is the length of f_i. The occurrence of f_i in position j is called *phrase*, while the previous occurrence appearing in position `pos` is called *source*. In other words, the pair $(\texttt{pos}, \texttt{len})$ is a pointer from the phrase to its source.

Let us encode our example `thisisathesis`. Every not-yet-seen character c is encoded with a pair $(0, c)$. So the first four factors will be

$$\begin{aligned} f_1 &= (0, \texttt{t}) \\ f_2 &= (0, \texttt{h}) \\ f_3 &= (0, \texttt{i}) \\ f_4 &= (0, \texttt{s}) \end{aligned}$$

We can now start exploiting repetitions. The following factor will encode the substring `is`, which already occurred starting in position 2 in the string and the length of the repetition is 2.

$$f_5 = (2, 2)$$

We can go on this way until the end of the string.

$$\begin{aligned} f_6 &= (0, \texttt{a}) \\ f_7 &= (0, 2) \\ f_8 &= (0, \texttt{e}) \\ f_9 &= (3, 3) \end{aligned}$$

Decoding the string does not require any additional information. The concatenation of each factor will produce the original string. Let us show how to recover the original string with the example.

The first four factors are just characters, so we can decode v by concatenating those.

$$\begin{aligned} v[0] &= f_1 = \texttt{t} \\ v[1] &= f_2 = \texttt{h} \\ v[2] &= f_3 = \texttt{i} \\ v[3] &= f_4 = \texttt{s} \end{aligned}$$

f_i	pair	phrase	source	prefix
f_1	$(0, \texttt{t})$	$v[0]$		`t`
f_2	$(0, \texttt{h})$	$v[1]$		`th`
f_3	$(0, \texttt{i})$	$v[2]$		`thi`
f_4	$(0, \texttt{s})$	$v[3]$		`this`
f_5	$(2, 2)$	$v[4..5]$	$v[2..3]$	`thisis`
f_6	$(0, \texttt{a})$	$v[6]$		`thisisa`
f_7	$(0, 2)$	$v[7..8]$	$v[0..1]$	`thisisath`
f_8	$(0, \texttt{e})$	$v[9]$		`thisisathe`
f_9	$(3, 3)$	$v[10..12]$	$v[3..5]$	`thisisathesis`

Figure 2.3: Sum up of the LZ77 factorization of `thisisathesis`.

For each factor representing a repetition (i.e. f_5), the decoding of the substring can be done by looking at the already decoded prefix of the string. The index in the pair (first element) gives us a position in the prefix of the string, and the length (second element) tells us how long is the substring to copy starting from the given position in the text. In our example:

$$v[4..5] = f_5 = (2, 2) = v[2..3] = \texttt{is}$$

We can go on this way until the end of the string. In Table 2.3 we summarize the example.

LZ77 will be a key point in Chapter 6 (Section 6.2), where also more details about it are given.

2.3.1.2 LZ78

In the version published in 1978 [88] the definition of factorization is slightly different. The pair encoding each factor consists of a position in the list of already encoded factors (i.e. the dictionary), and a character. The new factor can be decoded as the already seen factor pointed to in the dictionary by the pair (the position) concatenated with the character stored in the pair. The idea is that the prefix of the encoded factor f_i of length $|f_i| - 1$ already has an occurrence in some position p, and $v[p..p + |f_i| - 2] = f_i[0..|f_i| - 2]$.

Definition 5 (LZ78 Factorization [88]) *The LZ factorization of v is a factorization $v = f_1 f_2 \dots f_{z_{78}}$ of v into* phrases *such that each phrase f_i (a substring of v) is either*

1. *a letter that does not occur in $f_1 \cdots f_{i-1}$, or*
2. *the longest substring never occurred before, such that $f_i[0..|f_i| - 2]$ has already occurred as a phrase in $f_1 \cdots f_{i-1}$.*

The number z_{78} of the phrases is the length of the LZ factorization of v.

	pair	dictionary $D[i]$
0	$(0, \varepsilon)$	ε
1	$(0, \texttt{t})$	`t`
2	$(0, \texttt{h})$	`h`
3	$(0, \texttt{i})$	`i`
4	$(0, \texttt{s})$	`s`
5	$(2, \texttt{s})$	`is`
6	$(0, \texttt{a})$	`a`
7	$(0, \texttt{h})$	`th`
8	$(0, \texttt{e})$	`e`
9	$(4, \texttt{i})$	`si`
10	$(0, \texttt{s})$	`s`

Figure 2.4: The LZ78 factorization for `thisisathesis`. For example, consider the fifth factor $f_5 = (2, \texttt{s})$. It encoded the substring of the word `is` whose prefix `i` has a previous occurrence encoded in the 3nd element of the dictionary, and it is followed by the character `s`.

We show the LZ78 factorization for the example `thisisathesis` in Table 2.4. The decoding works in the same way as that for LZ77.

Chapter 3

The Number of Runs in the BWT as a Repetitiveness Measure

In this chapter, we discuss the number r of runs of the BWT as a reflection of the factor complexity of words, and we show why it may not be ideal to use it as a repetitiveness measure.

The number of runs of the BWT of a word is known to be related to the repetitiveness of the word. In particular, usually the more repetitive the word, the fewer the runs in its BWT. This is because repetitions in the word generate rotations of the word starting with the same prefix, and they will appear one after the other in the BWT matrix of the word. As a consequence, rotations starting with a suffix of a repetition produce blocks in the matrix starting with a common prefix and ending with the same character.

In other words, if we consider a large text v with a substring $v[i_1] \cdots v[j_1]$ repeated many times, for example, $v[i_1..j_1] = v[i_2..j_2] = \ldots = v[i_k..j_k]$, then we have a block in the BWT matrix containing all rotations prefixed by this substring. Additionally, we have other blocks of consecutive rotations in the BWT matrix, all prefixed by some suffix of this substring. Clearly, many of these rotations will be preceded by the same character (those starting with the ending part of the whole $v[i_1..j_1]$ substring), creating long runs of the same character in the last column of the BWT matrix, namely in the BWT of the word. For this reason, the number of runs in the BWT of a word may be considered a word-repetitiveness measure.

As mentioned before, great interest has arisen recently in repetitiveness measures in general to evaluate the compressibility of repetitive strings.

In this chapter, we focus on the parameter ρ, or *runs ratio*, which is the ratio between the number r of runs of the BWT of a word and that of the BWT of its reverse. The first upper bound $\rho = \mathcal{O}(\log^2 n)$ was given in [43]. It was still open whether this bound was tight, and it was also open the question of whether examples with $\rho = \omega(1)$ existed, i.e. ρ non-constant.

We give a first answer to this question by exhibiting an infinite family of binary words whose members satisfy $\rho = \Theta(\log n)$. We prove that words that are specific extensions of standard words have $\rho = \mathcal{O}(\log n)$, and, in particular, extensions of Fibonacci words have $\rho = \Theta(\log n)$. We call these words *standard-plus* words (respectively, *Fibonacci-plus* words). In showing this result, we also present the BWT matrix of such words, discussing how the lexicographic order of the rotations of Fibonacci-plus words changes with respect to the original Fibonacci words.

The contents of this chapter have partially appeared in [32].

3.1 Number of runs in the BWT of the forward and the backward word

If we consider a word v and its reverse $v' = v^{rev}$, for each circular factor u in v there exists a unique circular factor u^{rev} in v^{rev}. In particular, the number of distinct circular factors in the two words is the same, and the number of occurrences of u in v is equal to the number of occurrences of u^{rev} in v^{rev}. If we consider the word `repetition`, then `r,e,p,t,i,o,n` are the 7 factors of length 1, `re, ep, pe, et, ti, it, io, on, nr` are the 9 factors of length 2, and so on. Consider now its reverse `noititeper`, the factors of length 1 are the same, and those of length 2 are `no, oi, it, ti, te, ep, pe, er, rn`, which are the reverse of the 2-length factors of the forward word.

On the other hand, the number of runs is known to decrease with the repetitiveness of the word. For this reason, one could think that the number of runs of the BWT of a word may be preserved in its reverse since its repetitiveness is.

Although there are words for which this is true and the number of runs is the same in the two directions, this is not the case in general. For example, the BWT of `repetition` is `rpttoienie` with 9 runs, while the BWT of its reverse is `tptorneeii` with 8 runs.

In this chapter, we are going to show that there exists a family of infinite words for which the number of runs of the reverse increases by a logarithmic factor in the length of the word with respect to that of the original word.

We first define the *runs-ratio* as a measure of comparison of the number of runs of a word and its reverse. We discuss this ratio for standard words. Additionally, we introduce extensions of standard words, and we show an increment by a logarithmic factor in the length of the word of the runs from the BWT of the forward word and the BWT of the reverse.

3.2 The runs-ratio parameter

Let us start with the definition of the runs-ratio, the measure that gives an idea of the difference in the repetitiveness of a word and its reverse. Recall that, given a word v, we indicate with $runs(v)$ the number of maximal substrings of consecutive equal characters in v, while $r(v) = runs(BWT(v))$ is the number of runs in the BWT of v.

Definition 6 *We define the* runs-ratio $\rho(v)$ of a word v *as*

$$\rho(v) = \max\left(\frac{runs(BWT(v))}{runs(BWT(v^{rev}))}, \frac{runs(BWT(v^{rev}))}{runs(BWT(v))}\right) = \max\left(\frac{r(v)}{r(v^{rev})}, \frac{r(v^{rev})}{r(v)}\right),$$

We are also interested in the maximum number of runs $\rho(n)$ among all words of length n over the alphabet.

Definition 7 *Let n be an integer. We define the* runs-ratio over a length $\rho(n)$ *as* $\rho(n) = \max\{\rho(v) : |v| = n\}$.

Note that $\rho(v) \geq 1$ holds by definition. Since $r(v) = r(v^{\text{rev}})$ for all v with $|v| \leq 6$, we have $\rho(n) = 1$ for $n < 7$. In Table 3.1, we give the values of $\rho(n)$ for $n = 7, \ldots, 30$ (computed with a computer program by exhaustively computing the number of runs of the words for each length from 7 to 30).

n	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$\rho(n)$	1.5	1.5	2	2	2	2	2	2	2	2	2.5	2.5	2.5	2.5	3	2.5	3	3	2.67	3	3	3	3	3

Table 3.1: The values of $\rho(n)$ for $n = 7, \ldots, 30$.

3.2.1 Runs-ratio of standard words

We recall standard words (see Section 2.2.3). Given a so-called directive sequence of integers $(d_0, d_1, d_2, \ldots)$, with $d_0 \geq 0, d_i > 0$ for all $i > 0$, a standard word s_i, with $i \geq 0$, is a word such that $s_0 = \mathtt{b}, s_1 = \mathtt{a}, s_{i+1} = s_i^{d_{i-1}} s_{i-1}$, for $i \geq 1$. The index i is referred to as the *order* of s_i. Fibonacci words are standard words with directive sequence consisting of only ones. Recall that, by Proposition 2, part 1 (page 25), we can restrict ourselves to the case of $d_0 > 0$.

Standard words have many interesting properties, for instance, the BWT of any standard word consists of just two runs [64]: all the `b`'s followed by all the `a`'s. A key property of standard words that is important in the context of ρ is that the reverse of any standard word is a conjugate of the word itself. Therefore, they both have the same BWT, and thus $\rho = 1$. For example, `aaabaaaabaaaab` is the standard word with directive sequence $d = (3, 1, 2)$. Its reverse is `baaaabaaaabaaa`, which is the rotation of `aaabaaaabaaaab` starting in position 3. The BWT of both words is `bbbaaaaaaaaaaa`.

3.3 Fibonacci-plus words

In this section, we introduce a one-character extension of Fibonacci words. We show that such words have $\rho = \Theta(\log n)$, where n is the length of the word, meaning that the number r of runs of the BWT of these words differs by a logarithmic factor from r of the reverse. In other words, there exist cases in which r changes by a non-constant factor. This suggests that r may not be an ideal measure for the repetitiveness of words.

Let us define the one-character extension of Fibonacci words as follows.

Definition 8 *A word s is called a* Fibonacci-plus *word if it is either of the form $s\mathtt{b}$, where s is a Fibonacci word of even order $2k$, $k \geq 2$, or of the form $s\mathtt{a}$, where s is a Fibonacci word of odd order $2k+1$, $k \geq 2$. In the first case, s is* of even order*, otherwise* of odd order*.*

We start showing that the number of runs of the BWT of Fibonacci-plus words is constant. We then show that r of the reverse is logarithmic in the length of the word. We conclude with Theorem 1 claiming the result on ρ.

Proposition 4 *Let v be a Fibonacci-plus word. Then $r(v) = 4$. In particular,*

1. *if $v = s_{2k}\mathtt{b}$, then $BWT(v) = \mathtt{b}^{F_{2k-2}}\mathtt{a}^{F_{2k-1}-1}\mathtt{ba}$, and*
2. *if $v = s_{2k+1}\mathtt{a}$, then $BWT(v) = \mathtt{bab}^{F_{2k-1}-1}\mathtt{a}^{F_{2k}}$.*

Proof. We give the proof for even order only. The proof for odd order is analogous.

Let us write $v = s\mathtt{b}$, with $s = s_{2k}$, and $n = |s|$. Since Fibonacci words are standard words, it follows that $\mathrm{BWT}(s) = \mathtt{b}^{F_{2k-2}}\mathtt{a}^{F_{2k-1}}$ (see Section 2.2.3). By Proposition 2, part 2 (page 25), s can be written as $s = x\mathtt{ab}$ for a palindrome x; moreover, it follows from the specific form of x (Proposition 2, part 3, page 25) that both $x\mathtt{ab}$ and $x\mathtt{ba}$ are conjugates. It is further known that the lexicographically smallest conjugate is $\mathtt{a}x\mathtt{b}$ and the largest is $\mathtt{b}x\mathtt{a}$ [7, 64].

Now consider the conjugates of $v = s\mathtt{b}$. We will show that the conjugates of s retain their relative order after the insertion of the new $\mathtt{b}$, i.e. that if $conj_i(s) < conj_j(s)$ then also $conj_i(v) < conj_j(v)$. We further show that the new conjugate $conj_n(v)$ is the penultimate one in the lexicographic order of the conjugates of v. Since $conj_n(v)$ ends in $\mathtt{b}$, while the other conjugates $conj_i(v)$ end in the same character as $conj_i(s)$, for $i = 1, \ldots, n-1$, the claim follows.

Let $conj_i(s) < conj_j(s)$ be lexicographically consecutive conjugates of s. If $i < j$, then the new $\mathtt{b}$ appears earlier in $conj_j(s)$ than in $conj_i(s)$, therefore $conj_i(v) < conj_j(v)$ clearly holds. Now let $i > j$. It is known [7] that two lexicographically consecutive conjugates of s have the form $u\mathtt{ab}u'$ and $u\mathtt{ba}u'$, where $u'u = x$ is the palindrome from Proposition 2, part 2 (page 25). From $s_{2k} = x_{2k-1}\mathtt{ba}x_{2k-2}\mathtt{ab} = x_{2k-2}\mathtt{ab}x_{2k-1}\mathtt{ab}$, it follows that $x_{2k} = x_{2k-1}\mathtt{ba}x_{2k-2} = x_{2k-2}\mathtt{ab}x_{2k-1}$, and we deduce that $x = x_{2k}$ has exactly two occurrences in s as a circular factor. Therefore, $conj_i(v) = u\mathtt{abb}u'$, and the new $\mathtt{b}$ appears in $conj_j(v)$ within the suffix u'. This implies $u = lcp(conj_i(v), conj_j(v))$, and thus $conj_i(v) < conj_j(v)$.

Now note that $conj_n(v) = conj_{n-1}(s)\mathtt{b} = \mathtt{b}x\mathtt{ab}$. We know that $conj_{n-1}(s) = \mathtt{b}x\mathtt{a}$ is the lexicographically largest conjugate of s; let $conj_i(s)$ be the one immediately preceding it lexicographically. Then, for some u, u', we have $conj_i(s) = u\mathtt{ab}u'$ and $conj_{n-1}(s) = u\mathtt{ba}u'$, and therefore $conj_i(v) = u\mathtt{abb}u' < u\mathtt{ba}u'\mathtt{b} = conj_n(v)$. On the other hand, $conj_n(v) < conj_{n-1}(v) = \mathtt{bb}x\mathtt{a}$.

This completes the proof. □

We now show that r of the reverse of Fibonacci-plus words depends on the order of the respective Fibonacci word. In particular, it increases linearly with the order, and therefore it increases logarithmically with the length of the word.

Proposition 5 *Let v be a Fibonacci-plus word of order $2k$. Then $r(v^{rev}) = 2k$. In particular,*

1. *if v is of even order, i.e. $v = s_{2k}\mathtt{b}$ for some $k \geq 1$, then $BWT(v^{rev}) = \mathtt{b}^{F_{2k-2}-k+1}\mathtt{a}^{F_0}\mathtt{ba}^{F_2}\mathtt{ba}^{F_4}\mathtt{b}\cdots\mathtt{a}^{F_{2k-4}}\mathtt{bba}^{F_{2k-2}}$,*

2. *if v is of odd order, i.e. $v = s_{2k+1}\mathtt{a}$ for some $k \geq 1$, then $BWT(v^{rev}) = \mathtt{b}^{F_{2k-2}}\mathtt{aab}^{F_{2k-4}}\mathtt{ab}^{F_{2k-6}}\mathtt{a}\cdots\mathtt{b}^{F_2}\mathtt{ab}^{F_0}\mathtt{a}^{F_{2k}-k+1}$.*

Example 1 In Figure 3.1 we display the BWT matrices of the Fibonacci-plus word $v = s_8\mathtt{b}$ of length 35 and of its reverse.

Now consider the first few conjugates of v^{rev}. Since $v = s_{2k}\mathtt{b} = x_{2k}\mathtt{abb}$, we have $v^{\mathrm{rev}} = \mathtt{bba}x_{2k}$, noting that x_{2k} is a palindrome. Thus

$$\begin{aligned}
conj_0(v^{\mathrm{rev}}) &= \mathtt{bba}x_{2k},\\
conj_1(v^{\mathrm{rev}}) &= \mathtt{ba}x_{2k}\mathtt{b},\\
conj_2(v^{\mathrm{rev}}) &= \mathtt{a}x_{2k}\mathtt{bb},\\
conj_3(v^{\mathrm{rev}}) &= x_{2k}\mathtt{bba}.
\end{aligned}$$

Since Fibonacci words have no occurrence of $\mathtt{bb}$, the conjugate $conj_0(v^{\mathrm{rev}}) = v^{\mathrm{rev}}$ is the last row of the matrix. Moreover, by Proposition 3, part 4 (page 26), $\mathtt{a}x_{2k}\mathtt{b}$ is a Lyndon word, and therefore $conj_2(v^{\mathrm{rev}})$, having only an extra $\mathtt{b}$ at the end, is also Lyndon, and thus can be found in the first row. The relative order of the other two conjugates is also clear since x_{2k} begins with an $\mathtt{a}$, thus we have

$$\mathtt{a}x_{2k}\mathtt{bb} < x_{2k}\mathtt{bba} < \mathtt{ba}x_{2k}\mathtt{b} < \mathtt{bba}x_{2k}.$$

We will now subdivide the BWT matrix into three parts, according to the positions of these conjugates, and we will call these *top part*, *middle part*, and *bottom part*. The conjugates $\mathtt{a}x_{2k}\mathtt{bb}$, $x_{2k}\mathtt{bba}$ and $\mathtt{ba}x_{2k}\mathtt{b}$ are the first row of the top part, middle part, and bottom part, respectively. We use this to partition the BWT into the three corresponding parts $\mathrm{BWT}(v^{\mathrm{rev}})_{\mathrm{top}}, \mathrm{BWT}(v^{\mathrm{rev}})_{\mathrm{mid}}$, and $\mathrm{BWT}(v^{\mathrm{rev}})_{\mathrm{bot}}$. Thus we have

$$\mathrm{BWT}(v^{\mathrm{rev}}) = \mathrm{BWT}(v^{\mathrm{rev}})_{\mathrm{top}} \cdot \mathrm{BWT}(v^{\mathrm{rev}})_{\mathrm{mid}} \cdot \mathrm{BWT}(v^{\mathrm{rev}})_{\mathrm{bot}}.$$

BW array		rotations of $v =$ abaababaabaababaababaabaababaabaab$\underline{b}$	BWT(v)
1	21	aabaababaabaab$\underline{b}$abaababaabaababaabab	b
2	8	aabaababaababaabaababaabaab$\underline{b}$abaabab	b
3	29	aabaab$\underline{b}$abaababaabaababaababaabaabab	b
4	16	aababaabaababaabaab$\underline{b}$abaababaabaabab	b
5	3	aababaabaababaababaabaababaabaab$\underline{b}$ab	b
6	24	aababaabaab$\underline{b}$abaababaabaababaababaab	b
7	11	aababaababaabaababaabaab$\underline{b}$abaababaab	b
8	32	aab$\underline{b}$abaababaabaababaababaabaababaab	b
9	19	abaabaababaabaab$\underline{b}$abaababaabaababaab	b
10	6	abaabaababaababaabaababaabaab$\underline{b}$abaab	b
11	27	abaabaab$\underline{b}$abaababaabaababaababaabaab	b
12	14	abaababaabaababaabaab$\underline{b}$abaababaabaab	b
13	1	abaababaabaababaababaabaababaabaab$\underline{b}$	$\underline{b}$
14	22	abaababaabaab$\underline{b}$abaababaabaababaababa	a
15	9	abaababaababaabaababaabaab$\underline{b}$abaababa	a
16	30	abaab$\underline{b}$abaababaabaababaababaabaababa	a
17	17	ababaabaababaabaab$\underline{b}$abaababaabaababa	a
18	4	ababaabaababaababaabaababaabaab$\underline{b}$aba	a
19	25	ababaabaab$\underline{b}$abaababaabaababaababaaba	a
20	12	ababaababaabaababaabaab$\underline{b}$abaababaaba	a
21	33	ab$\underline{b}$abaababaabaababaababaabaababaaba	a
22	20	baabaababaabaab$\underline{b}$abaababaabaababaaba	a
23	7	baabaababaababaabaababaabaab$\underline{b}$abaaba	a
24	28	baabaab$\underline{b}$abaababaabaababaababaabaaba	a
25	15	baababaabaababaabaab$\underline{b}$abaababaabaaba	a
26	2	baababaabaababaababaabaababaabaab$\underline{b}$a	a
27	23	baababaabaab$\underline{b}$abaababaabaababaababaa	a
28	10	baababaababaabaababaabaab$\underline{b}$abaababaa	a
29	31	baab$\underline{b}$abaababaabaababaababaabaababaa	a
30	18	babaabaababaabaab$\underline{b}$abaababaabaababaa	a
31	5	babaabaababaababaabaababaabaab$\underline{b}$abaa	a
32	26	babaabaab$\underline{b}$abaababaabaababaababaabaa	a
33	13	babaababaabaababaabaab$\underline{b}$abaababaabaa	a
34	35	$\underline{b}$abaababaabaababaababaabaababaabaab	b
35	34	b$\underline{b}$abaababaabaababaababaabaababaaba	a

BW array		rotations of $v^{rev} =$ $\underline{b}$baabaababaabaababaababaabaababaaba	BWT(v^{rev})
1	3	aabaababaabaababaababaabaababaaba$\underline{b}$b	b
2	11	aabaababaababaabaababaaba$\underline{b}$baabaabab	b
3	24	aabaababaaba$\underline{b}$baabaababaabaababaabab	b
4	6	aababaabaababaababaabaababaaba$\underline{b}$baab	b
5	19	aababaabaababaaba$\underline{b}$baabaababaabaabab	b
6	14	aababaababaabaababaaba$\underline{b}$baabaababaab	b
7	27	aababaaba$\underline{b}$baabaababaabaababaababaab	b
8	32	aaba$\underline{b}$baabaababaabaababaababaabaabab	b
9	9	abaabaababaababaabaababaaba$\underline{b}$baabaab	b
10	22	abaabaababaaba$\underline{b}$baabaababaabaababaab	b
11	4	abaababaabaababaababaabaababaaba$\underline{b}$ba	a
12	17	abaababaabaababaaba$\underline{b}$baabaababaabaab	b
13	12	abaababaababaabaababaaba$\underline{b}$baabaababa	a
14	25	abaababaaba$\underline{b}$baabaababaabaababaababa	a
15	30	abaaba$\underline{b}$baabaababaabaababaababaabaab	b
16	7	ababaabaababaababaabaababaaba$\underline{b}$baaba	a
17	20	ababaabaababaaba$\underline{b}$baabaababaabaababa	a
18	15	ababaababaabaababaaba$\underline{b}$baabaababaaba	a
19	28	ababaaba$\underline{b}$baabaababaabaababaababaaba	a
20	33	aba$\underline{b}$baabaababaabaababaababaabaababa	a
21	35	a$\underline{b}$baabaababaabaababaababaabaababaab	b
22	2	baabaababaabaababaababaabaababaaba$\underline{b}$	$\underline{b}$
23	10	baabaababaababaabaababaaba$\underline{b}$baabaaba	a
24	23	baabaababaaba$\underline{b}$baabaababaabaababaaba	a
25	5	baababaabaababaababaabaababaaba$\underline{b}$baa	a
26	18	baababaabaababaaba$\underline{b}$baabaababaabaaba	a
27	13	baababaababaabaababaaba$\underline{b}$baabaababaa	a
28	26	baababaaba$\underline{b}$baabaababaabaababaababaa	a
29	31	baaba$\underline{b}$baabaababaabaababaababaabaaba	a
30	8	babaabaababaababaabaababaaba$\underline{b}$baabaa	a
31	21	babaabaababaaba$\underline{b}$baabaababaabaababaa	a
32	16	babaababaabaababaaba$\underline{b}$baabaababaabaa	a
33	29	babaaba$\underline{b}$baabaababaabaababaababaabaa	a
34	34	ba$\underline{b}$baabaababaabaababaababaabaababaa	a
35	1	$\underline{b}$baabaababaabaababaababaabaababaaba	a

Figure 3.1: BWT matrices of the Fibonacci-plus word $v = s_8$b of length 35 and its reverse, underlined the added b.

3.3.1 Bottom part

We show that the bottom part consists of all conjugates starting with a b. By construction, only one conjugate ends also with a b, while all the others end with an a.

Proposition 6 *$BWT(v^{rev})_{\text{bot}} =$* ba$^{F_{2k-2}}$.

Proof. By definition, the bottom part starts with the conjugate $conj_1(v) =$ bax_{2k}b. Since ax_{2k}bb is Lyndon (Proposition 3, part 4, page 26), it is smaller than all other conjugates, and therefore, bax_{2k}b is smaller than all other conjugates starting with b. Thus, the bottom part consists exactly of all conjugates starting with b. The number of b's in v, and thus in v^{rev} is $F_{2k-2} + 1$. Since s_{2k} has no occurrence of bb, every b in v^{rev} except the one in position 1 is preceded by an a, thus bax_{2k}b is the only conjugate ending in b. This proves the claim. □

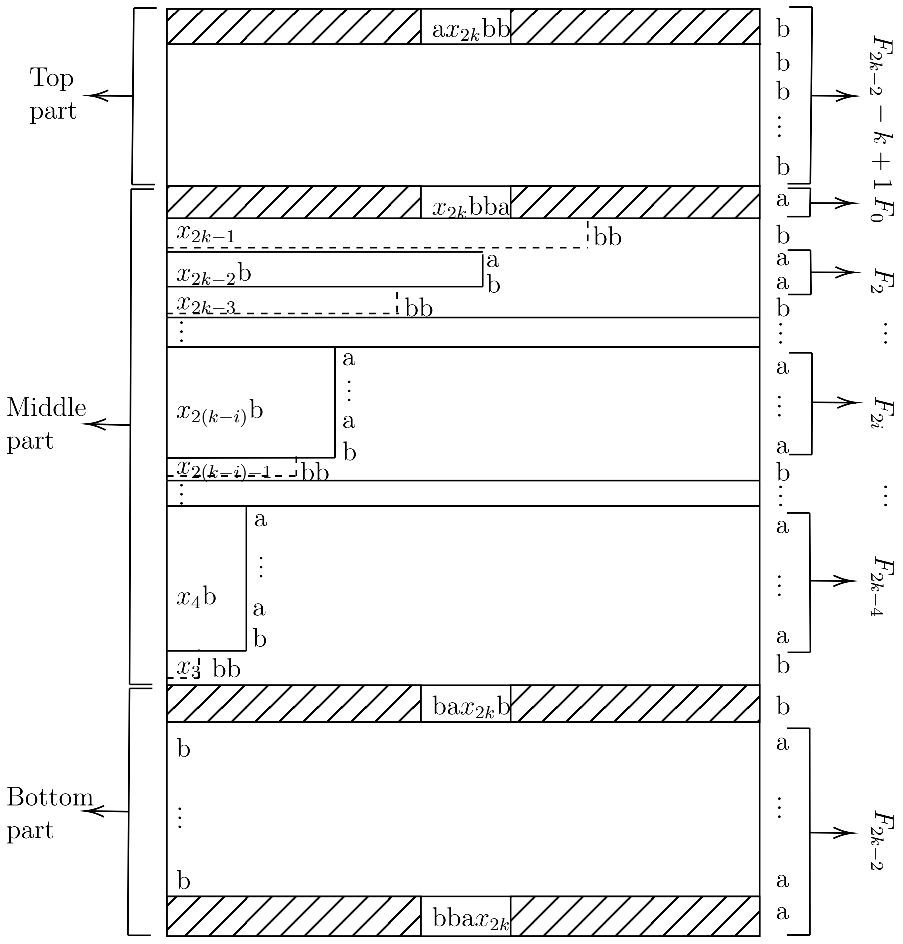


Figure 3.2: A sketch of the BWT matrix structure of v^{rev} where v is a Fibonacci-plus word.

3.3.2 Middle part

The middle part is the most complex one. It consists of blocks of conjugates prefixed by some x_h, h odd, which are all preceded by an $\mathtt{a}$ except for the largest one in each block, which is preceded by a $\mathtt{b}$.

We first need to show the following auxiliary lemmas.

Lemma 1 *The left-special circular factors of v^{rev} are exactly the prefixes of $x_{2k-1}\mathtt{b}$ and the prefixes of $\mathtt{ba}x_{2k-2}$.*

Proof. From Proposition 2, part 3 (page 25), $v^{\text{rev}} = \mathtt{bba}x_{2k} = \mathtt{bba}x_{2k-1}\mathtt{ba}x_{2k-2} = \mathtt{bba}x_{2k-2}\mathtt{ab}x_{2k-1}$. Let u be a left-special circular factor of v^{rev}. Since $\mathtt{bb}$ occurs only once, u does not contain $\mathtt{bb}$ as a factor. Moreover, from combinatorial properties of standard words (see [7]), it is known that for each $0 \leq h \leq F_{2k} - 2$, there is exactly one left-special circular factor of $\mathtt{ba}x_{2k}$ having length h, and it is a prefix of x_{2k}. Since $x_{2k-1}\mathtt{ba}$ (that is a prefix of x_{2k}) occurs exactly once in v^{rev} and $\mathtt{ba}x_{2k-2}$ has exactly two occurrences (one preceded by $\mathtt{b}$ and followed by $\mathtt{a}$, the other one preceded by $\mathtt{a}$ and followed by $\mathtt{b}$), either u is a prefix of $x_{2k-1}\mathtt{b}$ or it is a prefix of $\mathtt{ba}x_{2k-2}$. □

Lemma 2 *Let s_{2k} be a Fibonacci word of even order. Then, for all $i = 0, \ldots, k-2$, $\mathtt{a}x_{2(k-i)}\mathtt{b}$ and $\mathtt{a}x_{2(k-i)-1}\mathtt{b}$ have F_{2i} and F_{2i+1} occurrences, respectively, as circular factors of s_{2k}.*

Proof. The statement can be proved by induction on i. For $i = 0$, the statement follows from the fact that $\mathtt{a}x_{2k}\mathtt{b}$ and $\mathtt{a}x_{2k-1}\mathtt{b}$ have just $1 = F_0 = F_1$ occurrence. Let us suppose the statement is true for all $j \leq i$. Note that $\mathtt{a}x_{2(k-i)-2}\mathtt{b}$ appears as suffix of $\mathtt{a}x_{2(k-i)}\mathtt{b}$ and as suffix of $\mathtt{a}x_{2(k-i)-1}\mathtt{b}$. Moreover, such two occurrences are distinct because $\mathtt{a}x_{2(k-i)-1}\mathtt{b}$ is not a suffix of $\mathtt{a}x_{2(k-i)}\mathtt{b}$. This means that, by using the inductive hypothesis, the number of occurrences of $\mathtt{a}x_{2(k-i)-2}\mathtt{b}$ is $F_{2i} + F_{2i+1} = F_{2i+2}$. Analogously, $\mathtt{a}x_{2(k-i)-3}\mathtt{b}$ appears as prefix of $\mathtt{a}x_{2(k-i)-1}\mathtt{b}$ and as prefix of $\mathtt{a}x_{2(k-i)-2}\mathtt{b}$. Moreover, such two occurrences are distinct because $\mathtt{a}x_{2(k-i)-2}\mathtt{b}$ is not a prefix of $\mathtt{a}x_{2(k-i)-1}\mathtt{b}$. This means that the number of occurrences of $\mathtt{a}x_{2(k-i)-3}\mathtt{b}$ is $F_{2i+1} + F_{2i+2} = F_{2i+3}$. □

Proposition 7 *$BWT(v^{rev})_{\text{mid}} = \mathtt{a}^{F_0}\mathtt{ba}^{F_2}\mathtt{b} \ldots \mathtt{a}^{F_{2k-4}}\mathtt{b}$.*

Proof. For all $2 \leq i < j$, x_i is a prefix (and also a suffix) of x_j. This means that the rotations starting with $x_i\mathtt{bb}$ are lexicographically greater than $x_j\mathtt{bb}$. Note that, if $k = 2$, $v = x_4\mathtt{abb} = x_3\mathtt{ba}x_2\mathtt{abb}$ where $x_4 = \mathtt{aba}$, $x_3 = \mathtt{a}$, $x_2 = \varepsilon$ by Proposition 2 (page 25). Therefore, $\text{BWT}(v^{\text{rev}})_{\text{mid}} = \mathtt{a}^{F_0}\mathtt{b}$ since the rotations involved start with $x_4\mathtt{bb}$ and $x_3\mathtt{bb}$, respectively. Let us suppose $k \geq 3$. By Proposition 2, part 3 (page 25), $x_{2k-1}\mathtt{b}$ is a prefix of $v = s_{2k}\mathtt{b}$, as well as

$x_{2t}\mathtt{ab}$ for $2 \leq t \leq k-1$. This means that, for $1 \leq i \leq k-2$, $x_{2(k-i)}\mathtt{b}$ is not a prefix of $x_{2k-1}\mathtt{b}$. Thus, by Lemma 1, $x_{2(k-i)}\mathtt{b}$ is not left-special. Therefore, each occurrence of $x_{2(k-i)}\mathtt{b}$ is preceded by the same character; this character must be $\mathtt{a}$, since otherwise, both $\mathtt{b}x_{2(k-i)}\mathtt{b}$ and $\mathtt{a}x_{2(k-i)}\mathtt{a}$ would be factors, contradicting the fact that s_{2k}^{rev} is balanced (Proposition 3, part 5, page 26). Therefore, all occurrences of $x_{2(k-i)}\mathtt{b}$ correspond to a run of $\mathtt{a}$'s in the BWT. The length of this run is F_{2i} by Lemma 2. The claim follows from the fact that each $x_{2(k-i)-1}\mathtt{bb}$ occurs exactly once, and it is preceded by $\mathtt{b}$. □

3.3.3 Top part

We finally show the first run of $\mathtt{b}$'s of the BWT, namely the top part of the BWT matrix.

Lemma 3 *Let i be such that $conj_i(v^{rev}) < x_{2k}\mathtt{bba}$. Then the last character of $conj_i(v^{rev})$ is $\mathtt{b}$.*

Proof. Let $u = lcp(conj_i(v^{\text{rev}}), x_{2k}\mathtt{bba})$. Then u is a proper prefix of x_{2k-1}. This is because there are only two occurrences of x_{2k-1}, one followed by $\mathtt{ba}$, this is the prefix of $x_{2k}\mathtt{bba}$, and the other followed by $\mathtt{bb}$, thus greater than $x_{2k}\mathtt{bba}$. Therefore, $u' = u\mathtt{a}$ is a prefix of $conj_i(v^{\text{rev}})$ but not of x_{2k-1}, and thus by Lemma 1 it is not left-special. Now assume that $conj_i(v^{\text{rev}})$ ends with $\mathtt{a}$. Then $\mathtt{a}u\mathtt{a}$ is a factor of v^{rev}, and since u does not contain $\mathtt{bb}$, it is thus also a factor of s_{2k}^{rev}. On the other hand, $u\mathtt{b}$ is left-special, since it is a prefix of $x_{2k-1}\mathtt{b}$ (Lemma 1), therefore both $\mathtt{b}u\mathtt{b}$ and $\mathtt{a}u\mathtt{a}$ are factors of v^{rev}, and again, of s_{2k}^{rev}. This implies that both $\mathtt{a}u^{\text{rev}}\mathtt{a}$ and $\mathtt{b}u^{\text{rev}}\mathtt{b}$ are factors of s_{2k}. This is a contradiction, since s_{2k} is balanced (Proposition 3, part 5, page 26). □

Proposition 8 *$BWT(v^{rev})_{\text{top}} = \mathtt{b}^{F_{2k-2}-k+1}$.*

Proof. By Lemma 3, $\text{BWT}(v^{\text{rev}})_{\text{top}}$ consists of $\mathtt{b}$'s only. The number of $\mathtt{b}$'s of v is $F_{2k-2}+1$, of which we have accounted for k (since 1 is contained in $\text{BWT}(v^{\text{rev}})_{\text{bot}}$ and $k-1$ in $\text{BWT}(v^{\text{rev}})_{\text{mid}}$), there remaining exactly $F_{2k-2}-k+1$ $\mathtt{b}$'s. □

3.3.4 Fibonacci-plus runs-ratio proof

We recall Proposition 5 and we show its proof.

Proposition 5 *Let v be a Fibonacci-plus word. Then $r(v^{rev}) = 2k$. In particular,*

1. *if v is of even order, i.e. $v = s_{2k}\mathtt{b}$ for some $k \geq 1$, then $BWT(v^{rev}) = \mathtt{b}^{F_{2k-2}-k+1}\mathtt{a}^{F_0}\mathtt{b}\mathtt{a}^{F_2}\mathtt{b}\mathtt{a}^{F_4}\mathtt{b}\cdots\mathtt{a}^{F_{2k-4}}\mathtt{b}\mathtt{b}\mathtt{a}^{F_{2k-2}}$,*

2. *if v is of odd order, i.e. $v = s_{2k+1}\mathtt{a}$ for some $k \geq 1$, then $BWT(v^{rev}) = \mathtt{b}^{F_{2k-2}}\mathtt{a}\mathtt{a}\mathtt{b}^{F_{2k-4}}\mathtt{a}\mathtt{b}^{F_{2k-6}}\mathtt{a}\cdots\mathtt{b}^{F_2}\mathtt{a}\mathtt{b}^{F_0}\mathtt{a}^{F_{2k}-k+1}$.*

Proof. The statement for even order Fibonacci-plus words follows from Propositions 6, 7, and 8. The statement for odd order Fibonacci-plus words can be proved analogously. □

Theorem 1 *Let v be a Fibonacci-plus word, and let $|v| = n$. Then $\rho(v) = \Theta(\log n)$.*

Proof. From Propositions 4 and 5, we have that $\rho(v) = 2k/4 = k/2$. On the other hand, $n = |v| = F_{2k} + 1$, if v is of even order $2k$, $n = |v| = F_{2k+1} + 1$ if v is of odd order $2k + 1$. Thus, by the properties of the Fibonacci numbers, $2k = \Theta(\log n)$, implying that $\rho(v) = k/2 = \Theta(\log n)$. □

3.4 Standard-plus words: a generalization of Fibonacci-plus words

In fact, Fibonacci words are a specific case of standard words. In particular, Fibonacci words are standard words with directive sequence consisting of only 1's.

In this section, we generalize Fibonacci-plus words to *standard-plus* words. We prove that r of a standard-plus word may change up to a logarithmic factor with respect to its reverse. This result combined with that of the previous section implies that Fibonacci-plus words are maximal with respect to ρ among all standard words.

3.4.1 Standard-plus words have $\rho = \mathcal{O}(\log n)$

Recall that we can restrict ourselves to the case of $d_0 > 0$, which means that the word has more $\mathtt{a}$'s than $\mathtt{b}$'s. Otherwise, we could consider the word obtained by exchanging $\mathtt{a}$'s and $\mathtt{b}$'s and the result would still be true.

Definition 9 *A word v is called* standard-plus *if it is either of the form $s\mathtt{b}$, where s is a standard word of even order $2k$, $k \geq 2$, or of the form $s\mathtt{a}$, where s is a standard word of odd order $2k+1$, $k \geq 2$. In the first case, v is* of even order*, otherwise* of odd order.

Proposition 9 *Let $v = s_{2k}\mathtt{b}$ be a standard-plus word of even order. Then $r(v) = 4$.*

The proof of Proposition 9 is analogous to that of Proposition 4.

Proposition 10 *Let $v = s_{2k}\mathtt{b}$ be a standard-plus word of even order $2k$, where s_{2k} is the standard word that is obtained by using the directive sequence $(d_0, d_1, \ldots, d_{2k-2})$ of length $2k-1$, where $d_0 \geq 1$. If $d_0 = 1$, then $r(v^{rev}) = 2k$. Otherwise, $r(v^{rev}) = 2k+2$.*

Analogously, let $v = s_{2k+1}\mathtt{a}$ be a standard-plus word of even order $2k+1$, where s_{2k+1} is the standard word obtained by using the directive sequence $(d_0, d_1, \ldots, d_{2k-1})$ of length $2k$, where $d_0 \geq 1$. Then $r(v^{rev}) = 2k$.

Proof. (Sketch)

We sketch the proof for even order. Similar to what happens with Fibonacci's words (see Proposition 2, page 25), it is known that $s_{2k} = C\mathtt{ab}$, where C is a palindrome, the conjugate $\mathtt{a}C\mathtt{b}$ is a Lyndon word (see [6, 58]). Then $v^{\text{rev}} = \mathtt{bba}C$ and, in order to lexicographically sort the conjugates of v^{rev}, we can consider its Lyndon rotation $\mathtt{a}C\mathtt{bb}$. One can verify that $C \in \{\mathtt{a}^{d_0}\mathtt{b}, \mathtt{a}^{d_0+1}\mathtt{b}\}^*$. It is possible to see that $\text{BWT}(v^{\text{rev}})$ ends with $\mathtt{ba}^{|s_{2k}|_\mathtt{b}}$, since $\mathtt{ba}C\mathtt{b}$ is the smallest rotation starting with $\mathtt{b}$. Moreover, since $t = \mathtt{b}(\mathtt{a}^{d_0}\mathtt{b})^{d_1}\mathtt{b}$ is a suffix of $\mathtt{a}C\mathtt{bb}$, all rotations of v^{rev} starting with the first occurrence of $\mathtt{a}$ in each run $\mathtt{a}^{d_0}$ in t determine d_1 consecutive $\mathtt{b}$'s in $\text{BWT}(v^{\text{rev}})$. If $d_0 = 1$ such rotations are followed by the rotation $\mathtt{ba}C\mathtt{b}$, otherwise several rotations preceded by $\mathtt{a}$ (including the rotations starting with the other $\mathtt{a}$'s of t) are in between. So, if $d_0 = 1$, the last run of $\mathtt{b}$'s has length d_1+1, otherwise the last two runs of $\mathtt{b}$'s have length d_1 and 1, respectively.

Finally, when d_i (with odd i) is used to generate standard words, a set of consecutive rotations starting with $(\mathtt{a}^{d_0}\mathtt{b})^{d_1}\mathtt{a}^{d_0+1}\mathtt{b}$ and preceded by $\mathtt{b}$ is produced. This means that the other runs of $\mathtt{b}$'s have length $d_3, d_5, \ldots, d_{2k-3}, |s_{2k}|_\mathtt{b} - (d_1 + d_3 + \ldots + d_{2k-3})$. □

Theorem 2 *Let v be a standard-plus word of even order with $|v| = n$. Then $\rho(v) = \mathcal{O}(\log n)$.*

Proof. By definition, $v = s_{2k}b$ where s_{2k} is a standard word of order $2k$ for some positive k. Then $|s_{2k}| \geq F_{2k}$, and by Proposition 9 and 10, $\rho(v) \leq \frac{k+1}{2} \in \mathcal{O}(\log n)$. □

This last proposition of the section shows that Fibonacci-plus words are maximal with respect to ρ among all standard-plus words.

Proposition 11 *Let v be a Fibonacci-plus word, and v' a standard-plus word s.t. $|v| = |v'|$. Then $\rho(v) \geq \rho(v')$.*

Proof. Follows directly from Proposition 9 and 10, and from the fact that Fibonacci words have the longest directive sequence among all standard words of the same length. □

3.5 Runs-ratio and palindromic richness

Another interesting property of standard words is that they contain the highest possible number of palindromes when seen circularly. For this reason, we decided to investigate this aspect in standard-plus words, too.

In [23] the authors showed that any word v contains at most $|v|+1$ distinct palindromic factors, including the empty string. Words with exactly $|v|+1$ distinct palindromic factors are said to be *palindromically rich* (or just *rich*). A word v is said to be *strongly rich* if all conjugates of v are rich.

It is shown in [74] that, if BWT(v) is fully clustered, then v is strongly rich. Therefore, standard words are strongly rich since their BWT is fully clustered.

We are going to show that standard-plus words are palindromically rich but not strongly rich. This means that there exists at least one conjugate of any standard-plus word v having less than $|v|+1$ distinct palindromic factors. We will show this by exhibiting a non-rich conjugate of a standard-plus word.

Proposition 12 *Let v be a standard-plus word with $d = (d_0, \ldots, d_{2k})$, then v is rich, but it is not strongly rich.*

Proof. We will show the statement for standard words of even order with a further $\mathtt{b}$ at the end. The proof is analogous for standard words of odd order with a further $\mathtt{a}$.

We first prove that v is rich. Then we show that $conj_{d_0+1}(v)$ is not, and these statements together will prove the claim.

To show that v is rich it is enough to consider that we are adding a $\mathtt{b}$ to a standard word, which is already rich. Since the factor $\mathtt{bb}$ does not occur in standard words of even order with $d_0 > 0$, appending a character $\mathtt{b}$ results in extending the word by one character, and gaining the new palindromic factor $\mathtt{bb}$. This makes the standard-plus word palindromically rich since it has $|v|+1$ palindromic factors.

On the other hand, $conj_{d_0+1}(v)$ is not palindromically rich. We will show that $conj_{d_0+1}(v)$ loses d_0+1 palindromic factors, and gains only d_0 palindromic factors with respect to v.

Let us rewrite v as $s_{2k}\mathtt{b} = s_{2k-1}^{d_{2k}} s_{2k-2}\mathtt{b}$. The word $conj_{d_0+1}(v)$ starts with a suffix of s_{2k-1} since $|s_{2k-1}| > d_0$, followed by either another occurrence of s_{2k-1} or by one of s_{2k-2} ($d_0 > 1$ respectively $d_0 = 1$). Since $d_0 + 1 = |s_2|$, thus $conj_{d_0+1}(v)$ ends with $s_{2k-2}\mathtt{b}s_2$. Therefore, all palindromic factors contained in s_{2k-2} are preserved, while some contained in v are not, as shown in the following. For every standard word s it holds that $s_{2i} = x_{2i}\mathtt{ab}$ and $s_{2i-1} = x_{2i-1}\mathtt{ba}$, with x_{2i} and x_{2i-1} palindrome (see Proposition 2, Chapter 2, page 25). Therefore, v can be written as $v = x_{2k}\mathtt{abb}$, and the factors $x_{2k}, x_{2k}[1..|x_{2k}|-2], \ldots, x_{2k}[d_0..|x_{2k}|-d_0-1]$ are missing in $conj_{d_0+1}(v)$. This is because the first occurrence of s_2 in v now occurs as a suffix of the word instead of as a prefix. Altogether, $conj_{d_0+1}(v)$ loses $|s_2| = d_0 + 1$ palindromic factors with respect to v.

Let us now consider the suffix $s_{2k-2}\mathtt{b}s_2$ of $conj_{d_0+1}(v)$. There is exactly one occurrence of $\mathtt{bb}$ in v, and it is between $x_2\mathtt{a}$ (as a suffix of s_{2k-2} without the last $\mathtt{b}$) and $s_2 = x_2\mathtt{ab} = \mathtt{a}^{d_0-1}\mathtt{ab}$. Therefore, $x_2\mathtt{abb}x_2\mathtt{ab} = \mathtt{a}^{d_0-1}\mathtt{abba}^{d_0-1}\mathtt{ab}$ is a suffix of v, and $x_2\mathtt{abb}x_2\mathtt{a}$ is a new palindromic factor, as well as all the other $d_0 - 1 = |s_2| - 2$ factors of it sharing $\mathtt{bb}$ as center. In total, those are d_0 new palindromic factors. The factor $x_2\mathtt{abb}x_2\mathtt{a}$ is prefixed by an $\mathtt{a}$ by construction and followed by a $\mathtt{b}$, then there cannot be any more additional palindromic factors in $conj_{d_0+1}(v)$ with respect to v involving the unique occurrence of $\mathtt{bb}$. Combining the $d_0 + 1$ missing palindromic factors to the d_0 gained, the word is not rich. □

3.6 Are there words with higher runs-ratio?

We showed in the previous sections that, for standard-plus words, the number of runs in their BWT differs by up to a logarithmic factor in the length of the word with respect to their reverse. In particular, Fibonacci-plus words are maximal among standard-plus words with $\rho = \Theta(\log n)$. The question we asked ourselves was whether there exist binary words with a higher increase in r between the forward word and its reverse. In case of a positive answer, the gap between our bound and the upper bound of $\rho = \mathcal{O}(\log^2 n)$ given by Kempa and Kociumaka in [43] would be narrowed. The answer unfortunately is *we do not know yet*.

We experimentally produced binary words w with 4 or 6 runs such that $w = BWT(v)$, for some word v, and $\rho(v)$ is strictly greater than $\rho(s)$ for any standard-plus word s of the same length. We could not give a characterization for such words, leaving us with the question of whether ρ of these words is asymptotically greater than that of standard-plus words, or if it is just a matter of constants.

Since we were interested in words with high ρ, we started our study from words that are BWT of some other word over the alphabet, and are as clustered as possible. We were looking for those with r much greater than r of their reverse, i.e. words $w = BWT(v)$ and $r(v^{rev})$ much larger than $r(v)$.

However, we discarded BWT words with just two runs, because those are the BWTs of standard words [64], for which a full characterization with respect to r already exists. We recall that the BWT of binary words always has an even number of runs. For this reason, we first focused on words with four runs. We then extended our search to words with six runs. We exhaustively produced words v up to length $n = 30$ such that $\rho(v) = \rho(n)$, having then the greatest ρ for that length. We noticed that for certain lengths ($n = 23, 28$) we only have one such string (up to rotation and reverse), and that the BWT of both words and of both reverse words have a similar form, as follows:

- $n = 23$: BWT(v) = $\mathtt{b}^3\mathtt{a}^2\mathtt{baba}^5\mathtt{b}^3\mathtt{a}^2\mathtt{baba}^3$, BWT$(v^{rev})$ = $\mathtt{b}^5\mathtt{a}^{10}\mathtt{b}^5\mathtt{a}^3$
- $n = 28$: BWT(v) = $\mathtt{b}^3\mathtt{a}^2\mathtt{baba}^{10}\mathtt{b}^3\mathtt{a}^2\mathtt{baba}^3$, BWT$(v^{rev})$ = $\mathtt{b}^5\mathtt{a}^{15}\mathtt{b}^5\mathtt{a}^3$

We then further studied words having the same number of b's in their 4-run BWT. Let $w = \mathtt{b}^p\mathtt{a}^q\mathtt{b}^p\mathtt{a}^t$ be a BWT word of length n. We list the (p, q, t) triplets in Table A.1 in the Appendix, together with the length of the word w = BWT(v) and $\rho(v)$. For example, consider the Fibonacci-plus word u of order 10. The BWT of the forward word has 4 runs and the BWT of the reverse has 10 runs, therefore $\rho(u) = 2.5$. Consider now the word $BWT(v) = \mathtt{b}^{19}\mathtt{a}^{45}\mathtt{b}^{19}\mathtt{a}^7$ (which is the word of the same length 90 in Table A.1 in the Appendix). The BWT of the reverse of v has 18 runs, and therefore $\rho(v) = 4.5$.

We extended our experiments in the following way. For lengths n of words in Table A.1 (in the Appendix), we generated words w of 4 or 6 runs such that w = BWT(v), for some v, and $\rho(v) > \rho(v')$, where v' is a word from Table A.1 in the Appendix. In Table 3.2 we partially list these words. Among the words we computed in such a way, only words with the highest ρ for each length are reported in Table 3.2. For the full list, we refer the reader to Table A.2 in the Appendix. For example, the word $\mathtt{b}^{19}\mathtt{a}^{45}\mathtt{b}^{19}\mathtt{a}^7$ mentioned before is not in the new tables because there exist words whose BWT has 6 runs and having $\rho > 4.5$. One of these words is v whose BWT is $\mathtt{b}^2\mathtt{a}^5\mathtt{b}^{20}\mathtt{a}^{17}\mathtt{b}^{31}\mathtt{a}^{15}$, with $\rho(v) = 4.67$.

As opposed to the words from Table A.1, the runs of b's of these new words are not necessarily all equal (as in the example word). There is no such word for $n < 45$.

n	runs	r(reverse)	ρ	highest no. of palindromic factors
45	2, 5, 8, 11, 13, 6	22	3.67	29
45	2, 5, 10, 15, 8, 5	22	3.67	26
45	5, 8, 15, 10, 5, 2	22	3.67	26
45	6, 13, 11, 8, 5, 2	22	3.67	29
45	7, 3, 4, 13, 11, 7	22	3.67	33
45	7, 11, 13, 4, 3, 7	22	3.67	33
47	6, 7, 22, 12	16	4	22
47	6, 19, 10, 12	16	4	20
47	7, 16, 13, 11	16	4	36
47	11, 13, 16, 7	16	4	36
47	12, 10, 19, 6	16	4	20
47	12, 22, 7, 6	16	4	22
53	6, 13, 22, 12	18	4.5	24
53	12, 22, 13, 6	18	4.5	24
55	6, 11, 26, 12	18	4.5	25
55	12, 26, 11, 6	18	4.5	25
68	7, 13, 33, 15	18	4.5	25
68	8, 14, 31, 15	18	4.5	25

n	runs	r(reverse)	ρ	highest no. of palindromic factors
68	9, 19, 25, 15	18	4.5	29
68	15, 25, 19, 9	18	4.5	29
68	15, 31, 14, 8	18	4.5	25
68	15, 33, 13, 7	18	4.5	25
78	7, 16, 36, 19	18	4.5	27
78	9, 16, 36, 17	18	4.5	25
78	10, 22, 29, 17	18	4.5	29
78	12, 22, 13, 31	18	4.5	35
78	12, 26, 11, 29	18	4.5	35
78	17, 29, 22, 10	18	4.5	29
78	17, 36, 16, 9	18	4.5	25
78	19, 36, 16, 7	18	4.5	27
78	29, 11, 26, 12	18	4.5	35
78	31, 13, 22, 12	18	4.5	35
89	12, 26, 28, 23	20	5	25
89	23, 28, 26, 12	20	5	25
90	2, 5, 20, 17, 31, 15	28	4.67	39
90	2, 5, 22, 31, 19, 11	28	4.67	32
90	2, 5, 25, 31, 19, 8	28	4.67	31
90	4, 7, 18, 29, 23, 9	28	4.67	34
90	6, 10, 11, 37, 20, 6	28	4.67	38
90	6, 14, 37, 11, 16, 6	28	4.67	40
90	6, 16, 11, 37, 14, 6	28	4.67	40
90	6, 20, 37, 11, 10, 6	28	4.67	38
90	7, 10, 13, 42, 11, 7	28	4.67	35
90	7, 11, 42, 13, 10, 7	28	4.67	35
90	8, 19, 31, 25, 5, 2	28	4.67	31
90	9, 23, 29, 18, 7, 4	28	4.67	34
90	11, 18, 24, 6, 18, 13	28	4.67	40
90	11, 19, 31, 22, 5, 2	28	4.67	32
90	13, 18, 6, 24, 18, 11	28	4.67	40
90	13, 28, 10, 9, 11, 19	28	4.67	38
90	15, 31, 17, 20, 5, 2	28	4.67	39
90	18, 13, 5, 13, 23, 18	28	4.67	53
90	18, 23, 13, 5, 13, 18	28	4.67	53
90	19, 11, 9, 10, 28, 13	28	4.67	38
93	12, 20, 24, 5, 18, 14	30	5	39
93	14, 18, 5, 24, 20, 12	30	5	39
93	14, 30, 22, 27	20	5	18
93	27, 22, 30, 14	20	5	18
95	14, 30, 23, 28	20	5	18
95	18, 34, 19, 24	20	5	34
95	24, 19, 34, 18	20	5	34
95	28, 23, 30, 14	20	5	18
106	2, 9, 22, 19, 37, 17	30	5	47
106	6, 10, 14, 44, 23, 9	30	5	32
106	9, 23, 44, 14, 10, 6	30	5	32
106	12, 16, 12, 20, 27, 19	30	5	33
106	14, 16, 49, 27	20	5	22
106	15, 23, 34, 8, 7, 19	30	5	40
106	15, 35, 31, 25	20	5	39
106	17, 37, 19, 22, 9, 2	30	5	47

n	runs	r(reverse)	ρ	highest no. of palindromic factors
106	17, 39, 26, 24	20	5	36
106	19, 7, 8, 34, 23, 15	30	5	40
106	19, 27, 20, 12, 16, 12	30	5	33
106	24, 26, 39, 17	20	5	36
106	25, 31, 35, 15	20	5	39
106	27, 49, 16, 14	20	5	22

Table 3.2: Examples of words with higher ρ than standard-plus words. For a larger list, see A.2 in the Appendix.

Chapter 4

Bit Catastrophes: How Small Changes in a Word Affect its BWT

In this chapter, we show that the BWT is sensitive to one-character changes in the input word. We present a family of binary words for which edit operations (of insertion, deletion, and substitution) involving just one character may affect the number of runs r in the BWT of the word by a logarithmic factor in the length of the word.

This phenomenon was first studied for LZ78. Considering an optimally compressing word, Lutz and other authors [50, 53] posed the question of whether the word would still be reasonably well compressible after a one-bit change in it. This question was eventually answered in [47], where the authors showed that prepending a character to a word may cause a so-called *one-bit catastrophe* by reducing its compressibility. They additionally specified that this phenomenon is not a *tragedy* for LZ78, showing that, when it happens, it may produce an incompressible word only when the original word was already poorly compressible.

The interest in these so-called bit-catastrophes has recently spread to other string compressors and repetitiveness measures; an overview can be found in [1].

Here we focus on the effect on the BWT, and we are using the term "one-bit catastrophe" in a looser meaning, namely simply to denote the effect that a one-character edit operation may significantly change the compression size. The increase should be such that $r(w'_n) = \omega(r(w_n))$, for an infinite family $(w_n)_{n>0}$, where w'_n is the word resulting from applying a single edit operation to w_n. Considering the same families of words used in the previous chapter, we show that the reverse of Fibonacci-plus words are an example of a logarithmic increase from $\mathcal{O}(1)$ runs for the BWT of the reverse of Fibonacci words to $\Theta(\log n)$ runs. In other words, a one-bit catastrophe can be caused by prepending a character to the reverse of a Fibonacci word. Additionally, we show that prepending, appending or inserting a character can produce the

	CA	rotations of abaababaabaab	
0	7	aabaababaabab	b
1	2	aababaabaabab	b
2	10	aababaababaab	b
3	5	abaabaababaab	b
4	0	abaababaabaab	b
5	8	abaababaababa	a
6	3	ababaabaababa	a
7	11	ababaababaaba	a
8	6	baabaababaaba	a
9	1	baababaabaaba	a
10	9	baababaababaa	a
11	4	babaabaababaa	a
12	12	babaababaabaa	a

(a) Fibonacci word of order 6

	CA	rotations of abaababbaabaab	
0	8	aabaababaababb	b
1	11	aababaababbaab	b
2	2	aababbaabaabab	b
3	9	abaababaababba	a
4	0	abaababbaabaab	b
5	12	ababaababbaaba	a
6	3	ababbaabaababa	a
7	5	abbaabaababaab	b
8	7	baabaababaabab	b
9	10	baababaababbaa	a
10	1	baababbaabaaba	a
11	13	babaababbaabaa	a
12	4	babbaabaababaa	a
13	6	bbaabaababaaba	a

(b) Insertion

	CA	rotations of abaababaabaa	
0	10	aaabaababaab	b
1	7	aabaaabaabab	b
2	11	aabaababaaba	a
3	2	aababaabaaab	b
4	8	abaaabaababa	a
5	5	abaabaaabaab	b
6	0	abaababaabaa	a
7	3	ababaabaaaba	a
8	9	baaabaababaa	a
9	6	baabaaabaaba	a
10	1	baababaabaaa	a
11	4	babaabaaabaa	a

(c) Deletion

	CA	rotations of abaabbbaabaab	
0	7	aabaababaabbb	b
1	10	aababaabbbaab	b
2	2	aabbbaabaabab	b
3	8	abaababaabbba	a
4	0	abaabbbaabaab	b
5	11	ababaabbbaaba	a
6	3	abbbaabaababa	a
7	6	baabaababaabb	b
8	9	baababaabbbaa	a
9	1	baabbbaabaaba	a
10	12	babaabbbaabaa	a
11	5	bbaabaababaab	b
12	4	bbbaabaababaa	a

(d) Substitution

Figure 4.1: The BWT matrix of the Fibonacci word of order 6 (a), and that of the result for 3 bit-catastrophes: (b) inserting a character in position $6 = F_{6-1} - 2$, (c) deleting the last character (d) substituting the character in position $5 = F_{6-1} - 3$.

same effect. Finally, we show that deleting and inserting single characters can produce this effect, as well. (See Figure 4.1 for an example) To show this, we will use some results from the previous chapter, in particular, Lemma 2, Proposition 5, 6 and 9.

The contents of this chapter have been published at Developments in Language Theory (DLT 2023) [33].

4.1 Appending, prepending, inserting a character

In this section, we show that, for BWT, prepending, appending or inserting a character within a word may increase r by a logarithmic factor. To do so, we consider the reverse of Fibonacci words.

Proposition 13 *Let v be the reverse of a Fibonacci word. If v^{rev} is of even order $2k$, then $r(v\mathtt{b}) = 2k$. If v^{rev} is of odd order $2k+1$, then $r(v\mathtt{a}) = 2k$.*

Proof. This follows from the fact that $v\mathtt{b}$ is a conjugate of $\mathtt{b}v$, and thus they have the same BWT. Since $\mathtt{b}v$ is the reverse of a Fibonacci-plus word, by Proposition 5 (Chapter 3, page 35), then $r(\mathtt{b}v) = 2k$. The same reasoning holds for Fibonacci words of odd order $2k-1$ and the additional character $\mathtt{a}$. □

The following proposition can be deduced from previous results, and shows that there exist at least two positions in a Fibonacci word of even order, where adding a character causes a logarithmic increase of r. Recall properties of Fibonacci words presented in Proposition 2 and 3, (Chapter 2, pages 25 and 26). In particular, we will need the decomposition of standard words $s_{2k} = x_{2k}\mathtt{ab}$, $s_{2k+1} = x_{2k+1}\mathtt{ba}$, where x_{2k}, x_{2k+1} are palindromes, and the property of standard words that s with directive sequence of length k of the form $(1,\dots,1,2)$ is a conjugate of the Fibonacci word of order $k+2$.

Proposition 14 *Let s be the Fibonacci word of even order $2k$. If we add*

i) a $\mathtt{b}$ in position $F_{2k-1}-2$ or

ii) an $\mathtt{a}$ in position $F_{2k}-2$,

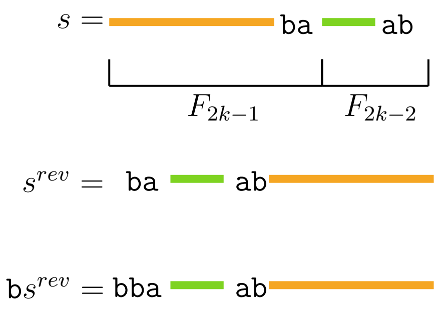
then the BWT of the resulting word has $\Theta(\log n)$ runs, where $n = |s|$.

Proof. Let us rewrite the word s as $s = x_{2k}\mathtt{ab} = x_{2k-1}\mathtt{ba}x_{2k-2}\mathtt{ab}$. We will first prove i), then ii).

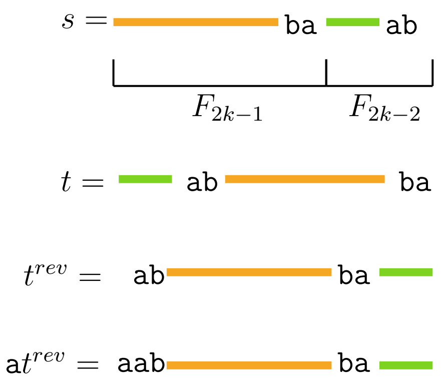
i) The reverse of s is its conjugate $conj_{F_{2k-1}-2}(s) = \mathtt{ba}x_{2k-2}\mathtt{ab}x_{2k-1}$. Prepending a $\mathtt{b}$ to $conj_{F_{2k-1}-2}(s)$ is equivalent to adding a $\mathtt{b}$ in position $F_{2k-1}-2$ in s. Clearly, the word $\mathtt{b}conj_{F_{2k-1}-2}(s)$ is the reverse of the Fibonacci-plus word of order $2k$, and by Proposition 5 (Chapter 3, page 35), its BWT has $2k$ runs. See Figure 4.2a for an illustration.

ii) On the other hand, $t = x_{2k}\mathtt{ba}$ is also a conjugate of s, and it is the standard word of odd order $2k-1$. It is easy to verify that $t^{rev} = \mathtt{ab}x_{2k} = conj_{F_{2k}-2}(s)$. By Proposition 10 (Chapter 3, page 41), $r(\mathtt{a}t^{rev}) = 2k-2 = \Theta(\log n)$, and for similar considerations as above, prepending an $\mathtt{a}$ to t^{rev} is equivalent to inserting $\mathtt{a}$ at position $F_{2k}-2$ in s. See Figure 4.2b for an illustration. □

Note that, by using similar arguments as in Proposition 13, it is possible to prove that adding a further $\mathtt{b}$ at the end of the reverse of a Fibonacci word of even order has the same effect as adding a character $\mathtt{c}$ greater than $\mathtt{b}$ and not



(a) Insert a b in position $F_{2k-1} - 2$.



(b) Insert an a in position $F_{2k} - 2$

Figure 4.2: In the figure, two positions where inserting a character in the reverse of a Fibonacci word of order $2k$ that produce a one-bit catastrophe (a b in position $F_{2k-1} - 2$, and an a in position $F_{2k} - 2$) are shown. The word s is represented as the decomposition in the palindromic central words (colored in orange and green) and ab, ba depending on the order of the central words. The decomposition of the word s, the position where the character is inserted and the conjugates used are chosen with respect to the proof of Proposition 14.

present in the word. This is because in both cases a new factor is introduced in the word, namely bb respectively c, see Figure 4.3 for an example. Both these factors are greater than all the other factors of the word, and they are the only changes in the word. The analogous thing happens when adding a further a to the reverse of a Fibonacci word of odd order or a character smaller than a, say \$ (see Figure 4.4). These results are formalized in the following two propositions.

Proposition 15 *Let v be the reverse of the Fibonacci word of even order $2k$ and $c >$ b, then $r(vc) = \Theta(k)$.*

Proof. Recall that $\text{BWT}(vc) = \text{BWT}(cv)$. For the proof, we consider the conjugate cv. Let us consider the reverse of the Fibonacci-plus word of the same order $2k$, $v' =$ bv. Then $v = v'[1..n-1]$, and cv and v' differ only for the characters in position 0. Moreover, $cv[0] = c$b is the lexicographically greatest 2-length substring of v, and $v'[0..1] =$ bb is the lexicographically greatest 2-length substring of v. In particular, the relative lexicographic order of the conjugates of v and $v'[1..n-1]$ is the same, and therefore the BWT of cv and v' differs only by the character preceding the conjugates starting in position 1. By Proposition 6 (Chapter 3), the character preceding $conj_1(v')$ is the last character of a run of b's, therefore the character c which precedes $conj_0(v)$ adds only one run, namely the 1-length run of c. Since $r(v') = 2k$, $\text{BWT}(cv)$ has the same $2k$ runs, plus the further run consisting of the single c. Therefore, $r(vc) = r(cv) = \Theta(k)$ □

Analogously, a similar argument can be made when a character $c < \texttt{a}$ is prepended to the reverse of a Fibonacci word of odd order, as shown in the following.

Proposition 16 *Let v be the reverse of the Fibonacci word of odd order $2k+1$ and $c < \texttt{a}$, then $r(vc) = \Theta(k)$.*

Proof. As in the previous proof, we note that the conjugate cv has the same BWT as vc, i.e. $\text{BWT}(cv) = \texttt{b}^{F_k 2-2}\texttt{aab}^{F_{2k-4}} \cdots \texttt{b}^{F_2}\texttt{aba}^{F_{2k}-k+1}$ by Proposition 5 (Chapter 3, page 35). Let us consider the reverse of the Fibonacci-plus word of the same order, $v' = \texttt{a}v$ of length n. Then $v = v'[1..n-1]$, and cv and v' differ only by the characters in position 0. In particular, the relative lexicographic order of the conjugates of v and $v'[1..n-1]$ is the same. The additional rotation cv starts with $c = \$$, and it is now the smallest among all rotations, therefore it will be at the very beginning of the matrix. This rotation is preceded by $\texttt{a}$. Since cv ends with $\texttt{a}$ and the lexicographically following rotations end with $\texttt{b}$, it increments the number of runs by at most 1 with respect to the BWT of v'.

Additionally, the rotation ending with $\$$ contributes to r with at most 2 more runs. This is because it either falls in between runs of two distinct characters, or within a run of a single character. In total, $\text{BWT}(cv)$ has at most $2k+3$ runs, and thus $r(cv) = r(vc) = \Theta(k)$. ☐

	CA	rotations of baabaababaaba	BWT	CA	rotations of baabaababaababb	BWT	CA	rotations of baabaababaabac	BWT
0	1	aabaababaabab	b	1	aabaababaababb	b	1	aabaababaabacb	b
1	9	aababaabaabab	b	4	aababaababbaab	b	4	aababaabacbaab	b
2	4	aababaababaab	b	9	aababbaabaabab	b	9	aabacbaabaabab	b
3	12	abaabaababaab	b	2	abaababaababba	a	2	abaababaabacba	a
4	7	abaababaabaab	b	7	abaababbaabaab	b	7	abaabacbaabaab	b
5	2	abaababaababa	a	5	ababaababbaaba	a	5	ababaabacbaaba	a
6	10	ababaabaababa	a	10	ababbaabaababa	a	10	abacbaabaababa	a
7	5	ababaababaaba	a	12	abbaabaababaab	b	12	acbaabaababaab	b
8	0	baabaababaaba	a	0	baabaababaabab	b	0	baabaababaabac	c
9	8	baababaabaaba	a	3	baababaababbaa	a	3	baababaabacbaa	a
10	3	baababaababaa	a	8	baababbaabaaba	a	8	baabacbaabaaba	a
11	11	babaabaababaa	a	6	babaababbaabaa	a	6	babaabacbaabaa	a
12	6	babaababaabaa	a	11	babbaabaababaa	a	11	bacbaabaababaa	a
13				13	bbaabaababaaba	a	13	cbaabaababaaba	a

Figure 4.3: The BWT matrices of the reverse of the Fibonacci word of order 6 and that of the same word with a further character $\texttt{b}$ (resp. $\texttt{c}$) at the end. The inserted character is highlighted in red. The lexicographic order of the rotations is the same when a $\texttt{b}$ or a $\texttt{c}$ is appended. The BWT coincides as well, except for the additional characters $\texttt{b}$ respectively $\texttt{c}$, which are in the same position in the two BWTs.

The previous proposition implies that words that are \$-terminated may have $\Theta(\log n)$ more runs in their BWT with respect to the same word without the end-of-string character, as shown in the following corollary. The insight of this corollary is that the bit-catastrophe should be taken into consideration when a terminator symbol (\$) is used. In fact, this single additional character

	CA	rotations of	BWT	CA	rotations of	BWT	CA	rotations of	BWT
		ababaababaabaababaaba			ababaababaabaababaabaa			ababaababaabaababaaba$	
0	9	aabaababaabaababaabab	b	20	aaababaababaabaababaab	b	21	$ababaababaabaababaaba	a
1	17	aabaababaababaabaabab	b	17	aabaababaababaabaabab	b	20	a$ababaababaabaababaab	b
2	4	aababaabaababaabaabab	b	9	aabaababaabaababaabab	b	17	aaba$ababaababaabaabab	b
3	12	aababaabaababaababaab	b	12	aababaabaababaababaab	b	9	aabaababaaba$ababaabab	b
4	20	aababaababaabaababaab	b	4	aababaabaababaabaaabab	b	12	aababaaba$ababaababaab	b
5	7	abaabaababaabaababaab	b	21	aababaababaabaababaaba	a	4	aababaabaababaaba$abab	b
6	15	abaabaababaababaabaab	b	18	abaababaababaabaababa	a	15	abaaba$ababaababaabaab	b
7	2	abaababaabaababaabaab	b	15	abaabaababaababaabaab	b	7	abaabaababaaba$ababaab	b
8	10	abaababaabaababaababa	a	7	abaabaababaabaababaab	b	10	abaababaaba$ababaababa	a
9	18	abaababaababaabaababa	a	10	abaababaabaababaababa	a	2	abaababaabaababaaba$ab	b
10	5	ababaabaababaabaababa	a	2	abaababaabaababaabaaab	b	13	ababaaba$ababaababaaba	a
11	13	ababaabaababaababaaba	a	13	ababaabaababaababaaba	a	5	ababaabaababaaba$ababa	a
12	0	ababaababaabaababaaba	a	5	ababaabaababaabaaababa	a	0	ababaababaabaababaaba$	$
13	8	baabaababaabaababaaba	a	0	ababaababaabaababaabaa	a	19	ba$ababaababaabaababaa	a
14	16	baabaababaababaabaaba	a	19	baaababaababaabaababaa	a	16	baaba$ababaababaabaaba	a
15	3	baababaabaababaabaaba	a	16	baabaababaababaabaaba	a	8	baabaababaaba$ababaaba	a
16	11	baababaabaababaababaa	a	8	baabaababaabaababaaba	a	11	baababaaba$ababaababaa	a
17	19	baababaababaabaababaa	a	11	baababaabaababaababaa	a	3	baababaabaababaaba$aba	a
18	6	babaabaababaabaababaa	a	3	baababaabaababaabaaaba	a	14	babaaba$ababaababaabaa	a
19	14	babaabaababaababaabaa	a	14	babaabaababaababaabaa	a	6	babaabaababaaba$ababaa	a
20	1	babaababaabaababaabaa	a	6	babaabaababaabaaababaa	a	1	babaababaabaababaaba$a	a
21				1	babaababaabaababaabaaa	a			

Figure 4.4: The BWT matrices of the reverse of the Fibonacci word of order 7 and that of the same word with a further character `a` (resp. \$) at the end. The inserted character is highlighted in red. Except for two rotations, the lexicographic order of other rotations is the same when a `a` or a \$ is appended.

may cause an asymptotically relevant increase in the number of runs of the BWT of the text.

Corollary 1 *There exist an infinite family of words v such that $r_\$(v)/r(v) = \Theta(\log n)$, where $n = |v|$.*

Proof. Consider the word $v\$$, where v is the reverse of a Fibonacci word of odd order. $\square$

4.2 Deleting a character

We next show that deleting a character can result in a logarithmic increment in r. In particular, we consider a Fibonacci word of even order and show the form of its BWT, as claimed in the following proposition.

We recall the notation $\widehat{s} = s[0..|s|-2]$, i.e. $\widehat{s}$ is the word s without the character in the last position.

Proposition 17 *Let s be the Fibonacci word of length n and order $2k > 4$, and $\widehat{s} = s[0..n-2]$. Then $BWT(\widehat{s})$ has the following form:*

$$BWT(\widehat{s}) = \mathtt{b}^{k-1}\mathtt{ab}^{F_{2k-3}-k+1}\mathtt{ab}^{F_{2k-5}}\cdots\mathtt{b}^{F_5}\mathtt{ab}^{F_3}\mathtt{aba}^{F_{2k-1}-k+1}.$$

In particular, $BWT(\widehat{s})$ has $2k$ runs.

To give the proof, we divide the BWT matrix of the word v in three parts: *top*, *middle* and *bottom part*, showing the form of each part separately:

$$\begin{aligned}\mathrm{BWT}(\widehat{s})_{\mathrm{top}} &= \mathtt{b}^{k-1}\mathtt{ab}^{F_{2k-3}-k+1}, \text{ consisting of 3 runs,}\\ \mathrm{BWT}(\widehat{s})_{\mathrm{mid}} &= \mathtt{ab}^{F_{2k-5}}\mathtt{ab}^{F_{2k-7}}\cdots\mathtt{ab}, \text{ consisting in } 2(k-2) \text{ runs,}\\ \mathrm{BWT}(\widehat{s})_{\mathrm{bot}} &= \mathtt{a}^{F_{2k-1}-k+1}, \text{ consisting in just one run.}\end{aligned}$$

Altogether, we then have $3 + 2(k-2) + 1 = 2k$ runs.

We identify 3 conjugates of the word of length $n-1$ that delimit the 3 parts of the BWT-matrix of the word:

$$\begin{aligned}conj_{n-2}(\widehat{s}) &= \mathtt{aa}x_{2k-2}\mathtt{ab}x_{2k-3}\mathtt{ba}\cdots\mathtt{ab}\\ conj_{n-4}(\widehat{s}) &= \mathtt{abaaab}\cdots x_3\mathtt{ba}\\ conj_0(\widehat{s}) &= x_{2k}\mathtt{a}\end{aligned}$$

and they are such that $conj_{n-2}(\widehat{s}) < conj_{n-4}(\widehat{s}) < conj_0(\widehat{s})$. In particular, the first mentioned rotation is the smallest rotation in the matrix due to the unique aaa prefix. The second rotation indicates the beginning of the middle part, and it is the smallest rotation starting with ab. Finally, the word itself $v = x_{2k}\mathtt{a}$ determines the beginning of the bottom part, namely the last long run of a's in the BWT.

Top part The top part of the matrix consists of all rotations of the word starting with aa. We are going to show that only one of these rotations ends with an a, and we show where the a in the BWT of the top part breaks the run of b's.

Proposition 18 (Top part) *Given $\widehat{s}$, the Fibonacci word of order $2k$ without the last character, then the first k rotations in the BWT matrix are* $\mathtt{aaa}\cdots\mathtt{b} < \mathtt{a}x_4\mathtt{aa}\cdots\mathtt{b} < \mathtt{a}x_6\mathtt{aa}\cdots\mathtt{b} < \ldots < \mathtt{a}x_{2k-2}\mathtt{aa}\cdots\mathtt{b} < \mathtt{a}x_{2k}$. *All other $F_{2k-3} - k + 1$ rotations starting with* aa *ends with a* b.

Proof. Overall, there are $F_{2k-3} + 1$ occurrences of aa in v. This is because there are F_{2k-1} a's, of which $F_{2k-2} - 1$ are followed by a b, therefore there are $F_{2k-1} - F_{2k-2} + 1 = F_{2k-3} + 1$ occurrences of a followed by an a.

Among all rotations starting by aa, the k smallest ones are those starting with the rightmost occurrence of $\mathtt{a}x_h$ for each even h in increment order of h. This is because of the single occurrence of aaa following the rightmost occurrence of x_3. Finally, only the largest $\mathtt{a}x_h$ is preceded by an a, therefore the k smallest rotations of $\widehat{s}$ are all preceded by a b except for the largest of them which is preceded by an a. This shows that the k smallest rotations in the BWT-matrix form two runs: $\mathtt{b}^{k-1}\mathtt{a}$.

There are not any other occurrences of aaa, therefore all the remaining $F_{2k-3} - k + 1$ rotations starting with aa are preceded by b by construction. Therefore, we have $\mathtt{b}^{k-1}\mathtt{ab}^{F_{2k-3}-k+1}$ in the top part of the BWT matrix of $\widehat{s}$.

□

Middle part In Figure 4.5 the structure of the middle part of the BWT matrix, when the last character is deleted, is shown. To prove the number of runs in this part of the matrix, we first draw attention to the fact that, for each x_h, all rotations starting with x_h appear together in the BWT matrix. In particular, they occur in blocks such that rotations starting with $x_h\mathtt{a}$, with h odd, are immediately lexicographically preceded by the unique rotation starting with $x_{h-1}\mathtt{aa}$, and immediately followed by the rotation starting with $x_{h+1}\mathtt{aa}$. This is because $x_h\mathtt{a}$, h odd, is prefixed by $x_{h-1}\mathtt{ab}$ (Lemma 4 and 5).

The word $\widehat{s}$ separates the middle and the bottom part. Note that every two lexicographically consecutive rotations of a Fibonacci word s differ by just two characters [7]. Therefore, for all $conj_i(\widehat{s})$ such that $\mathrm{lcp}(\widehat{s}, conj_i(\widehat{s})) = x_h$ for some h, where $conj_i(\widehat{s})$ is prefixed by $x_h\mathtt{a}$ while $\widehat{s}$ is prefixed by $x_h\mathtt{b}$, then $conj_i(\widehat{s})$ is in the middle part (i.e. smaller than $\widehat{s}$).

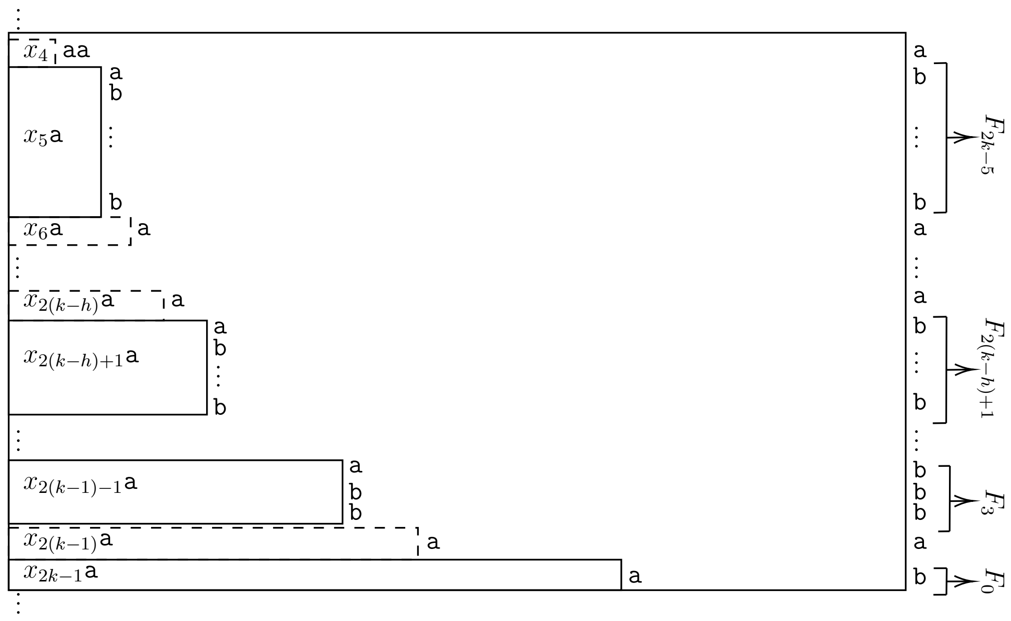


Figure 4.5: In the figure, the middle part $\mathrm{BWT}_{mid}(\widehat{s})$ of the BWT matrix for the deletion of the last character of a Fibonacci word of even order $2k$ is shown. In particular, for all $x_{2(k-h)+1}$, rotations starting with that factor occur in blocks such that rotations prefixed by $x_{2(k-h)+1}\mathtt{a}$ are all preceded by a $\mathtt{b}$. Moreover, they are immediately lexicographically preceded by the unique rotation starting with $x_{2(k-h)}\mathtt{aa}$, which is preceded by an $\mathtt{a}$, instead.

Lemma 4 *For all $h \leq 2k$, if h is even, then the rotation starting with the rightmost occurrence of x_h is preceded by* $\mathtt{a}$ *and is the smallest rotation among those starting with x_h.*

Proof. Each x_h is both a prefix and a suffix of x_{2k}. By properties of Fibonacci words, the rightmost occurrence of x_h is preceded by $\mathtt{a}$ if h is even, and by $\mathtt{b}$ otherwise. Additionally, the rightmost occurrence of x_h is the only occurrence of x_h circularly followed by $\mathtt{aa}$. Therefore, this is the smallest rotation starting with x_h. □

Lemma 5 *For $h \leq 2k$, rotations starting with some $x_{h-1}\mathtt{aa}$ are smaller than rotations starting with $x_h\mathtt{a}$. In particular, the word $\widehat{s} = x_{2k}\mathtt{a}$ is greater than any of the aforementioned rotations.*

Proof. We are showing that $x_{h-1}\mathtt{aa} < x_h\mathtt{a}$ for every $h \leq 2k$. Every x_{h-1} is a prefix of x_h. Since there is exactly one circular occurrence of $\mathtt{aaa}$ in $\widehat{s}$, then $x_h\mathtt{a}$ is either prefixed by $x_{h-1}\mathtt{ab}$ or by $x_{h-1}\mathtt{ba}$, i.e. the $\mathtt{aaa}$ factor occurs earlier in $x_{h-1}\mathtt{aa}$. In either case, the claim holds. □

Lemma 6 *There are $F_{2h} - 1$ occurrences of $\mathtt{b}x_{2(k-h)}\mathtt{a}$ and F_{2h+1} occurrences of $\mathtt{b}x_{2(k-h)-1}\mathtt{a}$ as circular factors in $\widehat{s} = x_{2k}\mathtt{a}$*

Proof. The claim can be proved by induction. For $h = 1$, the statement follows from the fact that there is one occurrence of $\mathtt{b}x_{2k}\mathtt{a}$ in the Fibonacci word of order $2k$, therefore there are $F_0 - 1 = 0$ occurrences in $\widehat{s}$ because of the missing $\mathtt{b}$ at the end. There are $F_1 = 1$ occurrences of $\mathtt{b}x_{2k-1}\mathtt{a}$ in both words. Note that the occurrence of any $\mathtt{b}x_{2h}\mathtt{a}$ in position $F_{2k} - 1$ of the Fibonacci word of order $2k$ is missing in $\widehat{s}$ due to the missing $\mathtt{b}$ at the end of the word.

Let us suppose the statement holds for all $i \leq h$. The factor $\mathtt{b}x_{2(k-h)-2}\mathtt{a}$ appears as a prefix of $\mathtt{b}x_{2(k-h)-1}\mathtt{a}$ and as a prefix of $\mathtt{b}x_{2(k-h)}\mathtt{a}$. Moreover, the mentioned occurrences are distinct because $\mathtt{b}x_{2(k-h)-1}\mathtt{a}$ is not a prefix of $\mathtt{b}x_{2(k-h)}\mathtt{a}$. Therefore, by induction, the number of occurrences of $\mathtt{b}x_{2(k-h)}\mathtt{a}$ is equal to the sum of the number of occurrences of $\mathtt{b}x_{2(k-h)-2}\mathtt{a}$ and those of $\mathtt{b}x_{2(k-h)-1}\mathtt{a}$: $F_{2h} - 1 + F_{2h+1} = F_{2h+2} - 1$. On the other, $\mathtt{b}x_{2(k-h)-3}\mathtt{a}$ appears as a suffix of $\mathtt{b}x_{2(k-h)-2}\mathtt{a}$ and as a suffix of $\mathtt{b}x_{2(k-h)-1}\mathtt{a}$. Moreover, the mentioned occurrences are distinct because $\mathtt{b}x_{2(k-h)-2}\mathtt{a}$ is not a suffix of $\mathtt{b}x_{2(k-h)-1}\mathtt{a}$. Finally, $\mathtt{b}x_{2(k-h)-3}\mathtt{a}$ appears once also as suffix of $\mathtt{a}x_{2(k-h)-2}\mathtt{a}$, starting in position $F_{2k} - 2$ of $\widehat{s}$. Therefore, by induction, the number of occurrences of $\mathtt{b}x_{2(k-h)-3}\mathtt{a}$ is equal to the sum of the number of occurrences of $\mathtt{b}x_{2(k-h)-2}\mathtt{a}$ and those of $\mathtt{b}x_{2(k-h)-1}\mathtt{a}$ plus one: $F_{2h+2}-1+F_{2h+1}+1 = F_{2h+3}$.□

Proposition 19 (Middle part) *The middle part contributes to $r(\widehat{s})$ with $2(k-2)$ runs in the following form:* $\mathtt{ab}^{F_{2k-5}}\mathtt{ab}^{F_{2k-7}} \cdots \mathtt{ab}^{F_3}\mathtt{ab}$

Proof. By Lemma 4, among rotations starting with the same x_h, $h \geq 4$ even, the smallest one is preceded by $\mathtt{a}$. All the following F_{2k-h-1} rotations starting with $x_{h+1}\mathtt{a}$ are preceded by $\mathtt{b}$ (Lemma 6). The fact that there exist exactly $k-2$ such x_h proves the claim. □

Bottom part The rotations that divide the middle part from the bottom part are the two rotations prefixed by the two occurrences of x_{2k-1}. By properties of Fibonacci words, one rotation is prefixed by $x_{2k-1}\mathtt{a}$ (end of middle part) and the other by $x_{2k-1}\mathtt{b}$ (beginning of bottom part). The latter follows the first in lexicographic order. Note that the rotation starting with $x_{2k-1}\mathtt{b}$ is $\widehat{s}$, namely $x_{2k-1}\mathtt{ba}x_{2k-2}\mathtt{a}$.

Proposition 20 (Bottom part) *All rotations greater than* $\widehat{s} = x_{2k-1}\mathtt{ba}x_{2k-2}\mathtt{a}$ *end with* $\mathtt{a}$ *.*

Proof. From Lemma 18 we have that $k-1+F_{2k-3}-k+1$ rotations ending with $\mathtt{b}$ have already appeared in the matrix, and from Lemma 19 $F_{2k-5}+\ldots+F_3+F_1$ rotations ending with $\mathtt{b}$ have already appeared in the matrix. Summing the number of $\mathtt{b}$'s we have $k-1+F_{2k-3}-k+1+F_{2k-5}+\ldots+F_3+F_1 = F_{2k-3}+F_{2k-5}+\ldots+F_3+F_1$. We can decompose each odd Fibonacci number F_{2x+1} in the sum $F_{2x}+F_{2x-1}$. Therefore, the previous sum becomes $F_{2k-4}+F_{2k-5}+F_{2k-6}+F_{2k-7}\ldots+F_2+F_1+F_1$. For every Fibonacci number F_x ,it holds that $F_x = F_{x-2}+F_{x-3}+F_{x-4}+\ldots+F_2+F_1+2$. Therefore, $F_{2k-4}+F_{2k-5}+F_{2k-6}+F_{2k-7}\ldots+F_2+F_1+F_1 = F_{2k-2}-1$, which is exactly the number of $\mathtt{b}$'s in $\widehat{s}$. Therefore, all the remaining rotations must end with $\mathtt{a}$. □

In the following, we report Proposition 17 and we explicitly state its proof.

Proposition 17 *Let s be the Fibonacci word of length n and order $2k > 4$, and $\widehat{s} = s[0..n-2]$. Then $BWT(\widehat{s})$ has the following form:*

$$BWT(\widehat{s}) = \mathtt{b}^{k-1}\mathtt{ab}^{F_{2k-3}-k+1}\mathtt{ab}^{F_{2k-5}}\cdots\mathtt{b}^{F_5}\mathtt{ab}^{F_3}\mathtt{aba}^{F_{2k-1}-k+1}.$$

In particular, $BWT(\widehat{s})$ has $2k$ runs.

Proof. The statement follows directly from Propositions 18, 19, 20. □

We have thus shown that appending or deleting a single character can substantially increase the parameter r. This implies the following known result:

Corollary 2 *The measure r is not monotone.*

4.3 Substituting a character

Next, we show how to increment r by a logarithmic factor by the edit operation of substituting a character. Consider a modification of the reverse of Fibonacci words of even order in which we replace the last $\mathtt{a}$ by a $\mathtt{b}$. More formally, let $s = x_{2k}\mathtt{ab}$ be the Fibonacci word of order $2k$, and consider $v = \mathtt{ba}\widehat{x_{2k}}\mathtt{b}$. Note that replacing the last $\mathtt{a}$ in s^{rev} with a $\mathtt{b}$ is equivalent to replace in the Fibonacci word s the character $\mathtt{a}$ at position $F_{2k-1} - 3$ by $\mathtt{b}$. This is because the conjugate $conj_{F_{2k-1}-2}(s) = \mathtt{ba}x_{2k-2}\mathtt{ab}x_{2k-1}$ is exactly the reverse of s.

Then $\text{BWT}(v)$ has $\Theta(\log n)$ runs, where n is the length of the word, as shown in the following proposition.

Proposition 21 *Let s be the Fibonacci word of order $2k$ with a $\mathtt{b}$ instead of the first $\mathtt{a}$, and $v = s^{rev} = \mathtt{ba}\widehat{x_{2k}}\mathtt{b}$. Then $BWT(v)$ has the form*

$$BWT(v) = \mathtt{b}^{F_{2k-2}-k+1}\mathtt{a}^{F_0}\mathtt{ba}^{F_2}\mathtt{b}\cdots\mathtt{a}^{F_{2k-4}}\mathtt{ba}^{F_{2k-2}-2}\mathtt{ba}.$$

In particular, $BWT(v)$ has $2k$ runs.

As for the deletion of a character, we divide the BWT matrix into three parts, and we prove the structure of each part separately. The BWT in the 3 parts is as follows:

$$\begin{aligned}
\text{BWT}(v)_{\text{top}} &= \mathtt{b}^{F_{2k-2}-k+1}, \text{ consisting in one run}\\
\text{BWT}(v)_{\text{mid}} &= \mathtt{a}^{F_0}\mathtt{ba}^{F_2}\mathtt{b}\cdots\mathtt{a}^{F_{2k-4}}\mathtt{b}, \text{ consisting in } 2(k-1) \text{ runs},\\
\text{BWT}(v)_{\text{bot}} &= \mathtt{a}^{F_{2k-2}-2}\mathtt{ba}, \text{ consisting in just 3 runs.}
\end{aligned}$$

Altogether, we have $1 + 2(k-1) + 3 = 2k + 2$ runs.

In this case, the rotations we identify to divide the BWT matrix are

$$\begin{aligned}
conj_0(v) &= \mathtt{ba}\widehat{x_{2k}}\mathtt{b}\\
conj_1(v) &= \mathtt{a}\widehat{x_{2k}}\mathtt{bb}\\
conj_2(v) &= \widehat{x_{2k}}\mathtt{bba}\\
conj_{n-2}(v) &= \mathtt{bbba}\widehat{x_{2k}}[0..n-2]
\end{aligned}$$

and they are such that $conj_1(v) < conj_2(v) < v < conj_{n-2}(v)$. Note that, as it is shown in the following, by properties of Fibonacci words, $conj_1(v)$ is smaller than any other rotation of v starting with $\mathtt{a}$, and consequently v is the smallest rotation starting with $\mathtt{b}$. In particular, $conj_1(v)$ is the smallest rotation in the top part, $conj_2(v)$ and v are the smallest respectively the largest rotation in the middle part, and $conj_{n-2}(v)$ is the largest rotation in the bottom part.

Lemma 7 *The conjugate $conj_1(v)$ is the Lyndon conjugate of v.*

Proof. By properties of Fibonacci words, $\mathtt{a}x_{2k}\mathtt{b}$ is the Lyndon conjugate of the Fibonacci word s_{2k} of order $2k$. Note that $\mathtt{a}x_{2k}\mathtt{b}$ is not the reverse of s_{2k}, and the reverse of any Fibonacci word is a conjugate of the Fibonacci word itself. We are claiming that the corresponding conjugate $\mathtt{a}\widehat{x_{2k}}\mathtt{bb}$ in v is Lyndon as well. Substituting the last $\mathtt{a}$ in v with the $\mathtt{b}$, we create the unique occurrence of $\mathtt{bbb}$ at the end of the string. Doing this, no conjugate $conj_i(v)$ is produced such that $conj_i(v) < \mathtt{a}x_{2k}\mathtt{b}$. Let us assume the contrary, i.e. that there exists a conjugate of v $conj_i(v)$ lexicographically smaller than $\mathtt{a}x_{2k}\mathtt{b}$. This means that $\mathrm{lcp}(\mathtt{a}x_{2k}\mathtt{b}, conj_i(v)) = y$ for some factor y of v, and $conj_i(v)$ starts with $y\mathtt{a}$, while $\mathtt{a}x_{2k}\mathtt{b}$ starts with $y\mathtt{b}$. The factor y must be of length $|x_{2k}|$, otherwise also $conj_i((s_{2k})^{rev})$ would be lexicographically smaller than $\mathtt{a}x_{2k}\mathtt{b}$, which is a contradiction with $\mathtt{a}x_{2k}\mathtt{b}$ being the Lyndon rotation of Fibonacci word s_{2k} of order $2k$. Therefore, $conj_1(v) = \mathtt{a}\widehat{x_{2k}}\mathtt{bb}$ is the Lyndon rotation of v. □

Corollary 3 *The word v is the smallest conjugate in v starting with* $\mathtt{b}$.

Proof. From Lemma 7 $conj_1(v) = \mathtt{a}\widehat{x_{2k}}\mathtt{bb}$ is the Lyndon rotation of v. It follows that $\mathtt{ba}\widehat{x_{2k}}\mathtt{b}$ is lexicographically smaller than all other conjugate in v starting with $\mathtt{b}$. □

Bottom part The bottom part consists in all conjugates lexicographically greater than $v = \mathtt{ba}\widehat{x_{2k}}\mathtt{b}$. Such rotations must start with a $\mathtt{b}$, and, as shown in the following, only one of such conjugates also ends with $\mathtt{b}$. Since the $\mathtt{b}$-ending conjugate does not appear as the very last nor the very first row in the bottom part of the matrix, then it breaks a long run of $\mathtt{a}$'s generating 3 runs.

Lemma 8 *The two conjugates $conj_{n-1}(v)$, $conj_{n-2}(v)$ are respectively the largest and the second largest conjugates of v, and they end with* $\mathtt{a}$ *respectively* $\mathtt{b}$.

Proof. The conjugate $conj_{n-2}(v)$ starts with the unique occurrence of $\mathtt{bbb}$, therefore it is the lexicographically greatest conjugate of v, and it is preceded by an $\mathtt{a}$. On the other hand, $conj_{n-1}(v)$ is the unique conjugate starting with $\mathtt{bba}$, therefore it is the immediate smaller conjugate than $conj_{n-2}(v)$ and it is preceded by $\mathtt{b}$. □

Lemma 9 *All conjugates $conj_i(v)$ such that $v < conj_i(v) < conj_{n-1}(v)$ end with* $\mathtt{a}$.

Proof. All such conjugates $conj_i(v)$ start with b. There is exactly one occurrence of bbb in v, therefore exactly two occurrences of b are preceded by b. One such occurrence is in position 0, i.e. at the beginning of v. The other occurrence is in position $n-1$, i.e. at the beginning of the second largest conjugate of v. Therefore, all conjugates lexicographically greater than v and larger than $conj_{n-1}(v)$ must be preceded by a. □

Proposition 22 (Bottom part) $BWT(v)_{bot} = \mathtt{a}^{F_{2k-2}-2}\mathtt{ba}$.

Proof. By Lemma 8 and 9, we have that $\mathrm{BWT}(v)_{bot}$ consists in a long run of a's followed by ba. The length of the run of a's is the number of conjugates of v such that $v < conj_i(v) < conj_{n-1}(v)$. The number of b's in v is $F_{2k-2}+1$. Therefore, the number of conjugates $v < conj_i(v) < conj_{n-1}(v)$ is exactly $F_{2k-2}+1-3 = F_{2k-2}-2$. □

Middle part A picture of the structure of the middle part of the BWT matrix is shown in Figure 4.6. To prove the form of $\mathrm{BWT}(v)_{mid}$ we need Lemma 2 from Chapter 3, that state that for the Fibonacci words s_{2k} of order $2k$ for all $i = 0, \ldots, k-2$, $ax_{2(k-i)}b$ and $ax_{2(k-i)-1}b$ have F_{2i} and F_{2i+1} occurrences, respectively, as circular factors of s_{2k}. Note that, exactly one occurrence of $\mathtt{a}x_{2(k-i)}\mathtt{b}$ and one of $\mathtt{a}x_{2(k-i)-1}\mathtt{b}$ of s_{2k}^{rev} are replaced by $\mathtt{a}\widehat{x_{2(k-i)}}\mathtt{bb}$ respectively $\mathtt{a}\widehat{x_{2(k-i)-1}}\mathtt{bb}$ in the word $v = \mathtt{ba}\widehat{x_{2k}}\mathtt{b}$. Thus, in v we have $F_{2i}-1$ and $F_{2i+1}-1$ occurrences of $\mathtt{a}x_{2(k-i)}\mathtt{b}$ respectively $\mathtt{a}x_{2(k-i)-1}\mathtt{b}$, and one occurrence of $\mathtt{a}\widehat{x_{2(k-i)}}\mathtt{bb}$ and $\mathtt{a}\widehat{x_{2(k-i)-1}}\mathtt{bb}$.

Lemma 10 *The left-special circular factors of v are exactly the prefixes of $\widehat{x_{2k-1}}$ and the prefixes of x_{2k-2}.*

Proof. Let u be a left-special circular factor in $v = \mathtt{ba}\widehat{x_{2k}}\mathtt{b}$. The factor bbb occurs exactly once in v, therefore u does not contain bbb. By combinatorial properties of Fibonacci words, $v = \mathtt{ba}x_{2k-3}\mathtt{ba}x_{2k-2}\mathtt{ba}\widehat{x_{2k-2}}\mathtt{b} = \mathtt{ba}x_{2k-2}\mathtt{ab}x_{2k-2}\mathtt{ab}\widehat{x_{2k-3}}\mathtt{b}$. Moreover, for each $0 \le h \le F_{2k}-2$, there exists exactly one left-special circular factor of $\mathtt{ba}x_{2k}$ having length h, and it is a prefix of x_{2k}. Since $\widehat{x_{2k-1}}$ occurs twice in v (once preceded and followed by a, once preceded and followed by b), and x_{2k-2} (once preceded by a and once preceded by b), and both $\widehat{x_{2k-1}}$ and x_{2k-2} are prefixes of x_{2k}, then either u is a prefix of $\widehat{x_{2k-1}}$ or it is a prefix of x_{2k-2}. □

To prove the middle part of the BWT matrix, we consider the blocks of lexicographically consecutive rotations prefixed by $x_{2h}\mathtt{b}$. We are showing that such rotations are always preceded by an a. In addition, the immediate smaller rotation is the largest rotation prefixed by $x_{2h}\mathtt{ab}$, i.e. that prefixed by $\widehat{x_{2h+1}}\mathtt{bb}$ and preceded by b. This is formalized in the following lemma.

Proposition 23 (Middle part) $BWT(v)_{mid} = \mathtt{a}^{F_0}\mathtt{ba}^{F_2}\mathtt{b}\cdots\mathtt{a}^{F_{2k-4}}\mathtt{b}$.

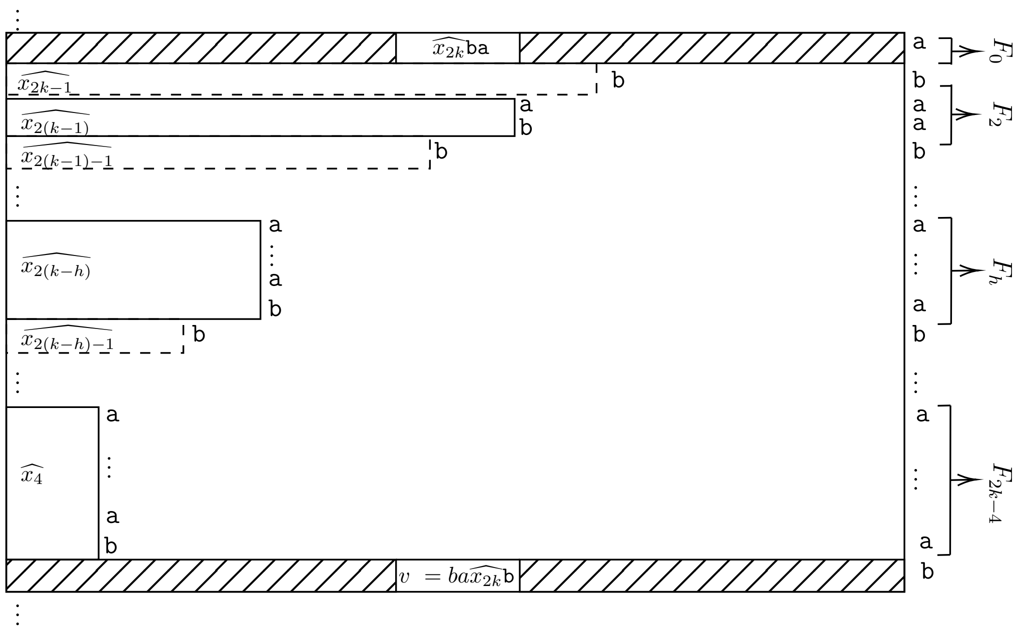


Figure 4.6: In the figure, the structure for the middle part $\mathrm{BWT}_{mid}(v)$ f the BWT matrix of the reverse of a Fibonacci word of order $2k$ when the last character is substituted. For each $\widehat{x_{2(k-h)}}$, all rotations prefixed by that factor appear together in the BWT. These rotations are all preceded by an a, and lexicographically followed by the unique rotation starting with $\widehat{x_{2(k-h)-1}}\mathtt{b}$, which is instead preceded by a b.

Proof. For $2 \leq i < j$, rotations starting with $\widehat{x_i}\mathtt{b}$ are lexicographically greater than x_i. For $1 \leq i < k$, $x_{2(k-i)}\mathtt{b}$ is not a prefix of $\widehat{x_{2k}}$. Thus, rotations starting with $x_{2(k-i)}\mathtt{b}$ are lexicographically greater than $\widehat{x_{2(k-i)+1}}$. Moreover, $x_{2(k-i)}\mathtt{b}$ is not a prefix of $x_{2k-2}\mathtt{a}$, as well. By Lemma 10, only factors that are prefixes of $x_{2k-2}\mathtt{a}$ or $\widehat{x_{2k}}$ occur in v both preceded by a and by b. Note that $v = \mathtt{ba}\widehat{x_{2k}}\mathtt{b}$ can also be written as $v = \mathtt{ba}x_{2k-3}\mathtt{ba}x_{2k-2}\mathtt{ba}\widehat{x_{2k-2}}\mathtt{b}$. Every $\mathtt{a}x_{2(k-i)}\mathtt{b}$ is a suffix of $x_{2(k-i)+1}\mathtt{b}$, therefore the character preceding any other $x_{2(k-i)}\mathtt{b}$ must also be an a. There are exactly $F_{2i} - 1$ occurrences of $x_{2(k-i)}\mathtt{b}$, and they are lexicographically immediately followed by $\widehat{x_{2(k-i)}}\mathtt{b}$. Together, rotations prefixed by such factors correspond to a run of a's in $\mathrm{BWT}(v)_{mid}$ of length F_{2i}. The claim follows from the fact that each $\widehat{x_{2(k-i)-1}}\mathtt{b}$ occurs exactly once, and it is preceded by b. □

Top part The top part of the matrix consists of the remaining b's of the word.

Proposition 24 (Top part) *$BWT_{top}(v) = \mathtt{b}^{F_{2k-2}-k+1}$.*

Proof. In total, in v there are $F_{2k-2} + 1$ b's. So far, we encountered $k - 1$ b's in the middle part of the BWT matrix, and 1 b in the bottom part. Therefore, we have exactly $F_{2k-2} + 1 - (k - 1) - 1 = F_{2k-2} - k + 1$ b's in the top part of the matrix. □

In the following, we rewrite Proposition 21, and we explicitly state its proof.

Proposition 21 *Let s be the Fibonacci word of order $2k$ with a* $\mathtt{b}$ *instead of the first* $\mathtt{a}$, *and $v = s^{rev} = \mathtt{ba}\widehat{x}\mathtt{b}$. Then $BWT(v)$ has the form*

$$BWT(v) = \mathtt{b}^{F_{2k-2}-k+1}\mathtt{a}^{F_0}\mathtt{ba}^{F_2}\mathtt{b}\cdots\mathtt{a}^{F_{2k-4}}\mathtt{ba}^{F_{2k-2}-2}\mathtt{ba}.$$

In particular, $BWT(v)$ has $2k+2$ runs.

Proof. The statement follows directly from Propositions 22, 23, 24. □

4.4 Shall we call these phenomena “tragedies”?

In this chapter, we showed how small changes in a word may affect r, presenting words of length n for which a single-character edit operation has the power of increasing r from 2 to $\Theta(\log n)$. Now, the question is whether these “catastrophes” affect the compressibility of the word. To answer this, we need a definition of compressibility.

The usual statistical compression methods for collections are based on the statistical entropy defined by Shannon [79]. An adaptation for finite strings is the *empirical entropy* (see [17]). However, the measures defined on Shannon’s entropy define the compressibility of a text based on the frequency of a character to follow a certain context of fixed length. This way, they are not able to capture the repetitiveness of the text and exploit it to evaluate its compressibility. Compressors that better reflect the repetitiveness of a text have been introduced and studied as an alternative to statistical compression methods.

Examples are compressors based on the number r of runs of the BWT respectively and those based on the number z of factors of some LZ factorization variant. Both r and z are usually considered good measures of the repetitiveness of the word. A high number of repetitions leads to longer and fewer runs in the BWT resulting in texts that are easier to compress than the original one. Analogously repeated substrings lead to longer and fewer phrases in the LZ factorization of the word, which is again easier to compress than the original word.

In this context, Lagarde and Perifel [47] define the compressibility of a string based on the size of its LZ78 factorization with respect to the length of the input word. They first show that prepending a character to a word can decrease its compressibility up to a non-constant factor, resulting in the so-called “one-bit catastrophe” for LZ78. However, they prove that a word can become incompressible only when it was poorly compressible from the beginning, and those that are well compressible cannot be turned into incompressible words. For this reason, they state that the one-bit catastrophe cannot be called a *tragedy*.

In our work, we refer as "bit catastrophe" to *any* change in one character which results in an increase of r by a non-constant factor. However, for a further discussion on "tragedies" for r, we would need to decide for one of the different definitions of BWT-compressibility currently in use. In particular, a string w may be called *compressible* with respect to the BWT

i) if $r = \mathcal{O}(n/polylog(n))$ [43],

ii) if $r(w)/runs(w) \rightarrow 0$ [27],

iii) if $runs(w) > r(w)$ [62].

If we consider as "tragedy" if a one-character edit operation on a compressible word turns it into an incompressible one, then the events we described in this chapter are not tragedies for any of the above-mentioned definitions. In particular,

i) after the edit operation we have $\Theta(\log n)$ runs in the BWT, and $\Theta(\log n) = \mathcal{O}(n/polylog(n))$, thus the produced word is compressible,

ii) since the runs in Fibonacci words are never longer than 2, then we still have $r(w)/runs(w) \rightarrow 0$ after the edit operation,

iii) the number of runs of the original word is strictly greater than r even after the edit operation is performed, and therefore the catastrophe produces a compressible word.

However, Fibonacci words are precisely those words with the best possible compressibility (r for these words is 2), and we showed that r can increase by multiplicative logarithmic factor after a one-character edit operation. In [33], another infinite family of words showing a non-constant change in r after a single-character edit operation is shown. Those are words of length n whose r is $\Theta(\sqrt{n})$, and their BWT can gain *additive* $\Theta(\sqrt{n})$ runs. Even though the upper bound for the multiplicative increase of r is proved to be $\mathcal{O}(\log n \log r)$ [1], as for now, no word is known with the property that r is non-constant, and r also presents a multiplicative non-constant increase after a one-character edit operation is performed.

Chapter 5

When a Dollar Makes a BWT

In this chapter we give a characterization of the positions of a word where the special character \$ can be inserted, obtaining the BWT of a \$-terminated word.

The BWT has always raised interest from a combinatorial point of view. A common combinatorial question on words related to some data structure built on words is deciding whether, given a data structure, a word on which that data structure is built exists, and sometimes even inferring it. For example, finding a word having a given permutation as its suffix array, or a given graph as its directed acyclic word graph (DAWG) is possible as presented in [40]. Similar questions were answered for prefix tables [15], LCP-arrays [42], Lyndon arrays [20], and suffix trees [13, 40, 82]. The problem was also studied for the BWT. A partial answer to a similar question to the one presented in this chapter can be found in [63], where a characterization for fixpoints of the BWT is given.

On the other hand, the BWT also lies at the heart of some of the most powerful compressed indexes in terms of query time and functionality, such as the well-known *FM-index* [26], and the more recent *RLBWT* [61] and *r-index* [5, 30, 31]. For this reason, the BWT has an important role in bioinformatics, where those data structures are mostly used, and the data are extremely repetitive. In general, in all real-life applications, special characters are used to indicate the end of the file. In bioinformatics applications, the end-of-file character is usually \$.

As a result, the research community interested in the BWT is divided in the two main branches mentioned: data structures applied on real-life bioinformatics problems and combinatorics on words. The former uses \$, the latter usually does not.

We are interested in what differences, if any, may arise when \$ is used with respect to when it is not. We already saw in Chapter 4 that, for Fibonacci words of odd order, inserting a character smaller than the characters of the word (such as \$) increases the number of runs by up to a logarithmic factor. In particular, this happens also when \$ is appended at the end of the string. What about whether a word is the BWT of some other word over the alphabet? Does the \$ have an impact on this? Can it turn a BWT word into a non-BWT? And vice versa?

We answer these questions in this chapter, giving a characterization of positions where \$ can be inserted in a word and turn it in the BWT of a \$-terminated word. We call such positions *nice*, and we show that whether a position is nice depends on the indices of the original word. In particular, we show that just a subset of the indices of the word is enough to determine the positions in the word that are nice. We additionally give an algorithm that efficiently computes all nice positions of a word. We finally give some bounds on the number of nice positions in a word, and some properties of such positions. We devote a section to the nice positions of fully clustered words, i.e. words having all occurrences of the same character clustered together.

The contents of this chapter have been published in [34, 35, 37].

5.1 Nice positions

In this section, we define the positions where \$ can be inserted in a word in order to make it a BWT. To do so, we first present some known results showing that whether a word is a BWT of some other word over the same alphabet depends on the structure of its standard permutation. We recall the standard permutation π_w of a word w, defined in Chapter 2, which is an array of length $|w|$ defined as follows: $\pi_w[i] < \pi_w[j]$ if and only if either $w_i < w_j$ or $w_i = w_j$ and $i < j$.

The following statement was proved for binary alphabets in [64], and stated in generalized form for larger alphabets in [52]:

Theorem 3 (Mantaci et al., 2003 [64], Likhomanov and Shur, 2011 [52]) *For a string $v \in \Sigma^*$, $BWT(v) = a_0^c \cdots a_{m-1}^c$ for some $c \geq 1$, if and only if $v = u^c$ with $BWT(u) = a_0 \cdots a_{m-1}$.*

From this, the authors of [52] obtain the following result:

Theorem 4 (Likhomanov and Shur, 2011 [52]) *A word $w \in \Sigma^*$ is a BWT image if and only if the number of cycles of π_w equals the greatest common divisor of the runlengths of w.*

In the following, we will need the explicit form of the standard permutation of BWT images.

Corollary 4 *If w is the BWT of a word $v \in \Sigma^*$ then π_w has the following form, where $c \geq 1$ and $m = n/c$, for some c:*

$$\pi_w = (0, e_0, \ldots, e_{m-1})(1, e_0{+}1, \ldots, e_{m-1}{+}1) \ldots (c{-}1, e_0{+}c{-}1, \ldots, e_{m-1}{+}c{-}1). \tag{5.1}$$

Moreover, for $c = 1$, it holds that w is the BWT of a primitive word if and only if π_w is cyclic.

Proof. Let c be the greatest common divisor of the runlengths of w. By Theorem 3, there exists a primitive word u s.t. $v = u^c$. If $c = 1$, then, by Theorem 4, π_w is cyclic, as claimed. Otherwise, let $0 \leq i \leq |w| - 1$ and $i - 1 = kc + r$, with $0 \leq r < c$ be the unique decomposition of $i - 1$ modulo c. It follows from the definition of the standard permutation that

$$\pi_w(i) = \pi_w(kc + r + 1) = \pi_w(kc + 1) + r, \tag{5.2}$$

since $w[i] = w[i-1] = \ldots = w[kc + 1]$, i.e. position i and position $kc + 1$ lie in the same run. But this implies that the standard permutation of w has the form (5.1), as claimed.

For $c = 1$, the reverse implication follows from applying the BWT reversal algorithm: if π_w is cyclic, then the output is a word of length $|w|$. Note that this direction is not true for $c > 1$: e.g. $(0, 2)(1, 3)$ is the standard permutation of *bbaa*, but also of *cdab*, and the latter is not a BWT image. $\square$

Example 2 *Consider the word* `cbccabaa`*. Its standard permutation is* $\left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 3 & 6 & 7 & 0 & 4 & 1 & 2 \end{smallmatrix}\right) = (0, 5, 4)(1, 3, 7, 2, 6)$*, and has two cycles and the* gcd *of its run-lengths is 1, therefore it is not the BWT of any other word. Contrarily, the BWT of* `cbccabaa` *is* `bcaaccab`*, whose standard permutation is* $\left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 5 & 0 & 1 & 6 & 7 & 2 & 4 \end{smallmatrix}\right) = (0, 3, 1, 5, 7, 4, 6, 2)$ *with 1 cycle, and the* gcd *of its run-lengths is also 1. Finally,* $BWT((\texttt{cbccabaa})^3) =$ `bbbcccaaaaaaccccccaaabbb` *is the "blow-up" of the BWT of* `bcaaccab` *as stated in Theorem 3, and its standard permutation* $\left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 9 & 10 & 11 & 15 & 16 & 17 & 0 & 1 & 2 & 3 & 4 & 5 & 18 & 19 & 20 & 21 & 22 & 23 & 6 & 7 & 8 & 12 & 13 & 14 \end{smallmatrix}\right) =$ $(0, 9, 3, 15, 21, 12, 18, 6)(1, 10, 4, 16, 22, 13, 19, 7)(2, 11, 5, 17, 23, 14, 20, 8)$ *has the form as in Corollary 4.*

Let $w \in \Sigma^*$, and $0 \leq i \leq n$. We denote by $dol(w, i)$ the $(n + 1)$-length word $w[0..i - 1]\$w[i..n - 1]$, i.e. the word which results from inserting \$ in w at position i.

Whenever w is clear from the context, we denote by π_i the standard permutation of $dol(w, i)$. Additionally, we define $\Sigma^*_\$$ the set of all words over Σ with an additional \$ at the end, and $\text{BWT}(\Sigma^*_\$)$ the set of words that are the BWT of a \$-terminated word. Clearly, all words in $\Sigma^*_\$$ are primitive.

We can now define *nice positions* as follows:

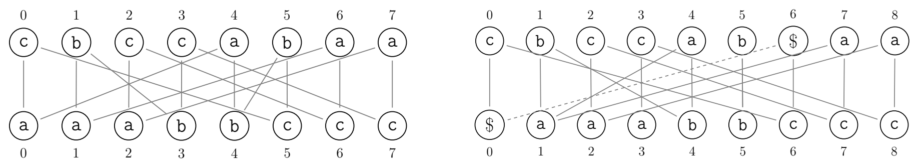
Definition 10 *Let* $w \in \Sigma^*$*,* $|w| = n$*, we call a position* i *nice if* $0 \leq i \leq n$ *and* $dol(w, i) \in BWT(\Sigma^*_\$)$*, i.e. if there exists a word* $v \in \Sigma^*$*,* $|v| = n$*, such that* $BWT(v\$) = dol(w, i)$*.*

Since, for every i, the character \$ appears exactly once in $dol(w, i)$, from Theorem 4 we immediately get the following:

Corollary 5 *For* $w \in \Sigma^n$ *and* $0 \leq i \leq n$*,* i *is nice if and only if* π_i *is cyclic.*

Figure 5.1: Standard permutation for $w =$ `cbccabaa` from Example 2 with $\sigma_w = \left(\begin{smallmatrix}0&1&2&3&4&5&6&7\\5&3&6&7&0&4&1&2\end{smallmatrix}\right)$ and $\sigma_6 = \left(\begin{smallmatrix}0&1&2&3&4&5&6&7&8\\6&4&7&8&1&5&0&2&3\end{smallmatrix}\right)$.

(a) Standard permutation of w **(b)** Standard permutation of $dol(w,7)$



Lemma 11 *Let $w \in \Sigma^n$, $0 \le i \le n$, π_w the standard permutation of w, and π_i the standard permutation of $dol(w,i)$. Then*

$$\pi_i(j) = \begin{cases} \pi_w(j) + 1 & \text{if } j < i, \\ 1 & \text{if } j = i, \text{ and} \\ \pi_w(j-1) + 1 & \text{if } j > i. \end{cases}$$

Proof. Immediate from the definition. □

We use a bipartite graph G_w to visualize the standard permutation of w (see Fig. 5.1). The top row corresponds to w, and the bottom row to the characters of w in alphabetical order. When w is a BWT, then this implies that the top row corresponds to the last column of a BWT matrix M, and the bottom row to the first. (This graph is therefore sometimes called BWT-graph.) Let us refer to the nodes in the top row as $x_0, \ldots, x_{n-1}$ and to those in the bottom row as $y_0, \ldots, y_{n-1}$. Nodes x_i are labeled by character $w[i]$, and nodes y_i are labeled by the characters of w in alphabetical order. We connect (x_i, y_j) if and only if $i = j$ (parallel edges) or $j = \pi_w(i)$ (oblique edges). It is easy to see that the node set of any cycle S in G_w has the form $\{x_k, y_k \mid k \in \mathcal{I}\}$ for some $\mathcal{I} \subseteq \{0, \ldots, n-1\}$, and that S is a cycle in G_w if and only if $\mathcal{I}$ is a cycle in π.

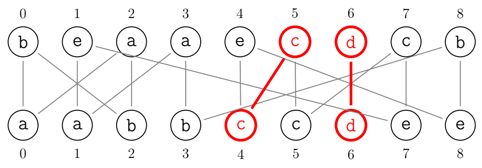
Now observe what happens when we insert a dollar into w in position i (see Fig. 5.1). For positions j which are smaller than i, their image is incremented by one; i is mapped to 0; and for positions j on the right of i, both j and its image $\pi(j)$ are shifted to the right by one. This is what is formally said in Lemma 11.

5.2 Characterization of nice positions via essential pseudo-cycles

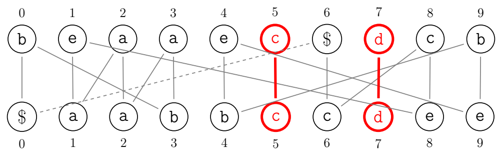
In this section, we give the characterization of nice positions, namely which position i in a word w is such that $dol(w,i) = BWT(v\$)$ for some v over the same alphabet.

Figure 5.2: Standard permutation for $w = \texttt{beaaecdcb}$ with $\sigma_w = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 7 & 0 & 1 & 8 & 4 & 6 & 5 & 3 \end{smallmatrix}\right)$ and $\sigma_7 = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 8 & 1 & 2 & 9 & 5 & 0 & 7 & 6 & 4 \end{smallmatrix}\right)$. Red edges become fixpoints in σ_6.

(a) Standard permutation of w



(b) Standard permutation of $dol(w, 6)$



Our first result is that every word that is the BWT of some other word over the alphabet has at least one nice position.

Theorem 5 *Let $w \in BWT(\Sigma^n)$, and c the number of cycles of π_w. Then c is nice.*

Proof. By Corollary 4, π_w has the form

$$\pi_w = (0, e_0, \ldots, e_{m-1})(1, e_0{+}1, \ldots, e_{m-1}{+}1) \ldots (c{-}1, e_0{+}c{-}1, \ldots, e_{m-1}{+}c{-}1).$$

By Lemma 11 we have that $\pi_c(c) = 0$. Note that each cycle of π_w has exactly one element which is smaller than c, therefore for exactly one element $j < c$ of each cycle we have that $\pi_c(j) = \pi_w(j)$, by Lemma 11. These elements are $\pi_c(0) = e_0, \pi_c(1) = e_0{+}1, \ldots, \pi_c(c{-}1) = e_0{+}c{-}1$. For all other elements $j > c$ we have that $\pi_c(j) = \pi_w(j{-}1)$, i.e. $\pi_c(e_0) = \pi_w(e_0){+}1 = e_1, \pi_c(e_1{+}1) = \pi_w(e_1) + 1 = e_2, \ldots, \pi_c(e_i + c - 1) = \pi_w(e_i + c - 2) + 1 = e_{i+1} + c - 1$. The resulting form of π_c is the following.

$$\pi_c = (0, e_0{+}1, \ldots, e_{m-1}{+}1, 1, e_0{+}2, \ldots, e_{m-1}{+}2, \ldots, c{-}1, e_0{+}c, \ldots, e_{m-1}{+}c, c),$$

and is thus cyclic, therefore, by Corollary 4, c is nice. □

Proposition 25 *Let $w = BWT(v\$)$ such that $w[1] = \$$, i.e. \$ is in the second position of w. Then v is Lyndon.*

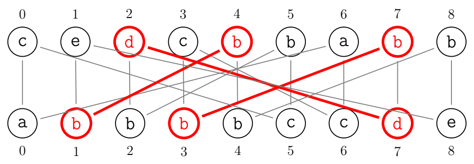
Proof. Let $v' = v\$$. The smallest rotation of v' is $\$v$, since \$ is smaller than all other characters; while $v\$$ is the second smallest rotation, since $w[1] = \$$. Therefore, every proper suffix u of v is lexicographically strictly larger than v, implying that v is Lyndon. □

In order to see which positions are nice, we want to understand how cycles in π_i are created. See the example $w = \texttt{beaaecdcb}$ and $i = 6$ in Figure 5.2. Position 6 is not nice, and π_6 has two fixpoints.

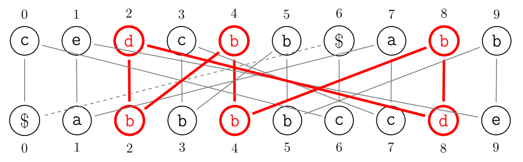
In general, if j is a fixpoint in π_w and $i \leq j$, then $j+1$ will be a fixpoint in π_i. Similarly, if $\pi(j) = j - 1$ and $j < i$, then j is a fixpoint in π_i. These two

Figure 5.3: Standard permutation of $w =$ `cedcbbabb` with $\sigma = \begin{pmatrix} 0&1&2&3&4&5&6&7&8 \\ 5&8&7&6&1&2&0&3&4 \end{pmatrix}$ and of $dol(w, 6)$ with $\sigma_6 = \begin{pmatrix} 0&1&2&3&4&5&6&7&8&9 \\ 6&9&8&7&2&3&0&1&4&5 \end{pmatrix}$.

(a) Pseudo-cycle in the standard permutation of w



(b) Cycle in the standard permutation of $dol(w, 6)$



cases are illustrated in Fig. 5.2, where 6 is a fixpoint in π_w and $\pi_w(5) = 4$, so the insertion of \$ in position $i = 6$ leads to the two fixpoints 5 and 7 in π_6. Therefore, position 6 is not nice.

Indeed, the previous observation can be generalized: if S is a cycle in π_w, then no position $i \leq \min S$ is nice. Similarly, if S is such that $\pi_w(S) = S - 1 = \{j - 1 \mid j \in S\}$, then no position $i > \max S$ is nice; in both cases, the insertion of \$ in such a position would turn S into a cycle. However, the situation can also be more complex, as illustrated in Figure 5.3. In Theorem 6 we will give a necessary and sufficient condition for creating a proper cycle by inserting a \$ in some position. First, we need another definition.

We recall that not all words are the BWT of some other word, and that whether a word is a BWT or not depends on its standard permutation. Clearly, also whether a position is nice with respect to a given word depends on the standard permutation of the word. To characterize such positions, we introduce the concept of *pseudo-cycle* and the correlated *critical interval* in the following.

Definition 11 (Pseudo-cycle) *Let τ be a permutation of n. A non-empty subset $S \subseteq \{0, 1, \ldots, n-1\}$ is called a* pseudo-cycle *if it can be partitioned into two sets S_{left} and S_{right}, where $S_{left} < S_{right}$ (elementwise), such that $\tau(S) = \{x - 1 \mid x \in S_{left}\} \cup S_{right}$.*

Definition 12 (Critical interval) *The* critical interval *of the pseudo-cycle S is $R_S = [a + 1, b]$, where $a = \max S_{left}$ and $b = \min S_{right}$. If S_{left} is empty, we set $a = -1$, and if S_{right} is empty, we set $b = n$. Let $S_1, \ldots, S_k$ be all pseudo-cycles of the word w, then $R_w = R_1 \cup \ldots \cup R_k$ is the union of the critical intervals of all pseudo-cycles in w.*

Definition 13 (Additional definitions on pseudo-cycle) *Let $S = S_{left} \cup S_{right}$ be a pseudo-cycle, then*

i) If $S_{left} = \emptyset$ then S is called right-only*, if $S_{right} = \emptyset$ then S is called* left-only.

ii) If $S_{left} \neq \emptyset$, then we refer to $a = \max S_{left}$ as the boundary *of S.*

iii) A right-only pseudo-cycle is called minimal *if no proper subset is a cycle.*

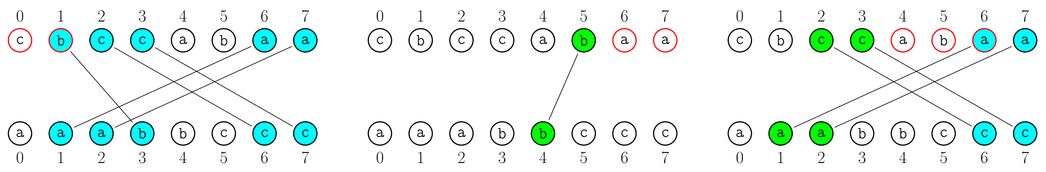
Example 3 (continued from page 65) *Consider the word* `cbccabaa`*. The pseudo-cycle $S = \{1,2,3,6,7\}$, with $\pi(S) = \{1,2,3,6,7\}$ is a right-only pseudo-cycle with critical interval $R_S = [0,1]$, while $S' = \{5\}$, with $\pi(S') = \{4\}$ is a left-only pseudo-cycle with boundary 4 and critical interval $R_{S'} = [6,8]$. Finally, $S'' = \{2,3,6,7\}$, with $\pi(S'') = \{1,2,6,7\}$ has boundary 3 and critical interval $R_{S''} = [4,6]$. This last pseudo-cycle can be divided in $S''_{left} = \{2,3\}$ and $S''_{right} = \{6,7\}$ (See Figure 5.4).*

Figure 5.4: The pseudo-cycles of the word `cbccabaa` mentioned in Example 3 S, S', S'' (S_{left} in green, S_{right} in blue, critical intervals in red).

(a) Right-only pseudo-cycle $S = \{1,2,3,6,7\}$, with $\pi(S) = \{1,2,3,6,7\}$ and critical interval $R_S = [0,1]$

(b) Left-only pseudo-cycle $S' = \{5\}$, with $\pi(S') = \{4\}$, with boundary 5 and critical interval $R_{S'} = [6,8]$

(c) $S'' = \{2,3,6,7\}$, with $\pi(S'') = \{1,2,6,7\}$, boundary 3 and critical interval $R_{S''} = [4,6]$



Lemma 12 *Let S and S' be two pseudo-cycles with boundary i. Then $S \cap S'$ is also a pseudo-cycle with boundary i.*

Proof. If $S \subseteq S'$ or $S' \subseteq S$, then the statement is trivial. Otherwise, we have to show that (1) $i \in S \cap S'$, and (2) if $x \in S \cap S'$ then $x - 1 \in \pi(S \cap S')$ if $x \leq i$, and $x \in \pi(S \cap S')$ if $x > i$. (1) follows from the fact that both S and S' have boundary i. Ad (2): Let $x \in S \cap S'$, $x \leq i$. Since S is a pseudo-cycle with boundary i, it follows that $\pi^{-1}(x-1) \in S$. Similarly, since S' is a pseudo-cycle with boundary i, $\pi^{-1}(x-1) \in S'$, thus $\pi^{-1}(x-1) \in S \cap S'$. Now let $x \in S \cap S'$, $x > i$. Then it follows that $\pi^{-1}(x) \in S$ and $\pi^{-1}(x) \in S'$, thus $\pi^{-1}(x) \in S \cap S'$. □

Note that not every i is the boundary of some pseudo-cycle. But with Lemma 12, we can now define a unique pseudo-cycle for each i which is a boundary, as follows.

Definition 14 *Let $0 \leq i \leq n-1$. A pseudo-cycle $S = S_{left} \cup S_{right}$, with $S_{left} \neq \emptyset$ is called* i-essential *if i is the boundary of S and every pseudo-cycle S' with boundary i is a superset of S. A pseudo-cycle is called* essential *if it is i-essential for some i.*

Clearly, for every i, there exists at most one i-essential pseudo-cycle. This implies that altogether there are at most $n-2$ essential pseudo-cycles. See Figure B.1 in the Appendix.

Example 4 (continued from page 69) *Consider again the word* `cbccabaa`. $S = \{3, 7\}$ *is the 3-essential pseudo-cycle with* $\pi(S) = \{2, 7\}$ *and critical interval* $R = [4, 7]$. *In this pseudo-cycle we have* $S_{\mathit{left}} = \{3\}$ *and* $S_{\mathit{right}} = \{7\}$.

Proposition 26 *Given an i-essential pseudo-cycle S, its critical interval R_S is maximal w.r.t. the boundary i. In other words, if S' is a pseudo-cycle with the same boundary i, then $R_{S'} \subseteq R_S$.*

Proof. Follows immediately from Lemma 12. □

Corollary 6 *Let w be a word. Then R_w equals the union of critical intervals of those pseudo-cycles which are either minimal right-only or essential.*

Definition 15 *Let $0 \leq i \leq n$ and $S \subseteq \{0, 1, \ldots, n-1\}$. We define*

$$\mathit{shift}(S, i) = \{x \mid x \in S \text{ and } x < i\} \cup \{x + 1 \mid x \in S \text{ and } x \geq i\}, \text{ and}$$
$$\mathit{unshift}(S, i) = \{x \mid x \in S \text{ and } x < i\} \cup \{x - 1 \mid x \in S \text{ and } x > i\}.$$

Lemma 13 *Let $w \in \Sigma^*$ and $\pi = \pi_w$. Let $0 \leq i \leq n$, and $U \subseteq \{0, 1, \ldots, n\} \setminus \{i\}$. Then U is a cycle in the permutation π_i if and only if $S = \mathit{unshift}(U, i)$ is a pseudo-cycle w.r.t. π, and i belongs to the critical interval of S.*

Proof. Let $U_1 = \{x \in U \mid x < i\}$, $U_2 = \{x \in U \mid x > i\}$. Then $S = U_1 \cup (U_2 - 1)$. We have to show that U is a cycle if and only if S is a pseudo-cycle, with $S_{\mathit{left}} = U_1$ and $S_{\mathit{right}} = U_2 - 1$. Note that this implies that i is contained in the critical interval of S.

First, let S be a pseudo-cycle with $S_{\mathit{left}} = U_1$ and $S_{\mathit{right}} = U_2 - 1$, and let $x \in U$. We have to show that $x \in \pi_i(U)$, which implies the claim. If $x \in U_1$, then $x \in S_{\mathit{left}}$, and there is a $y \in S$ s.t. $\pi(y) = x - 1$. If $y \in S_{\mathit{left}}$, then $y \in U_1$ and $\pi_i(y) = x$ by Lemma 11, thus $x \in \pi_i(U)$. Else $y \in S_{\mathit{right}}$, then $y + 1 \in U_2$ and $\pi_i(y + 1) = x$, again by Lemma 11, and thus $x \in \pi_i(U)$.

Now let $x \in U_2$. Then $x - 1 \in S_{\mathit{right}}$ and there is a $y \in S$ s.t. $\pi(y) = x - 1$. If $y \in S_{\mathit{left}}$, then $y \in U_1$ and $x = \pi(y) + 1 = \pi_i(y)$ by Lemma 11, thus $x \in \pi_i(U)$. Else $y \in S_{\mathit{right}}$, then $y + 1 \in U_2$ and $\pi_i(y + 1) = x$, again by Lemma 11, and thus $x \in \pi_i(U)$.

Conversely, let U be a cycle, set $S_{\mathit{left}} = U_1$ and $S_{\mathit{right}} = U_2 - 1$. Let $x \in S$. We will show that if $x \in S_{\mathit{left}}$, then $x - 1 \in \pi(S)$, and if $x \in S_{\mathit{right}}$, then $x \in \pi(S)$, proving that S is a pseudo-cycle. The claim follows with analogous arguments as above and noting that $\pi(j) = \pi_i(j) - 1$ if $j < i$, and $\pi_i(j+1) - 1$

if $j \geq i$. □

What the latter lemma says is that the \$ can be inserted in some position i which will not turn a pseudo-cycle of w into a cycle in $dol(w, i)$. This is because words that are \$-terminated must be cyclic in order to be the BWT of some other word (see Theorem 4). Such "*forbidden*" positions are those contained in the critical interval of some pseudo-cycle. Therefore, we can now characterize nice positions via pseudo-cycles as follows.

Theorem 6 *Let w be a word of length n over Σ, and $0 \leq i \leq n$. Then i is nice if and only if there is no pseudo-cycle S w.r.t. the standard permutation $\pi = \pi_w$ whose critical interval contains i.*

Proof. "$\Rightarrow$": We will assume that π contains a pseudo-cycle with i in its critical interval, and show that then i is not nice. Let S be a pseudo-cycle w.r.t. π, R its critical interval and $i \in R$. By Lemma 13, *shift*(S, i) is a cycle in π_i not containing i. Therefore, π_i has at least two cycles, implying that $dol(w, i) \notin \text{BWT}(\Sigma^*_\$)$. "$\Leftarrow$": For the converse, assume that i is not nice. Then π_i contains at least two cycles, and thus it contains at least one cycle $C \subseteq \{0, 1, \ldots, n\} \setminus \{i\}$. By Lemma 13, this implies that *unshift*(C, i) is a pseudo-cycle in π, and its critical interval contains i. □

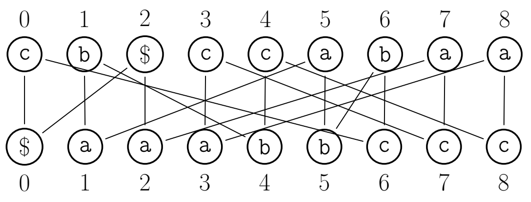


Figure 5.5: The standard permutation of `cb$ccabaa` is cyclic: $\begin{pmatrix} 0&1&2&3&4&5&6&7&8 \\ 6&4&0&7&8&1&5&2&3 \end{pmatrix} = (0, 6, 5, 1, 4, 8, 3, 7, 2)$. Position 2 is nice in $w =$ `cbccabaa` because it is not contained in R_w.

5.3 Computing nice positions

Given a word w, it is easy to compute all nice positions of w, by inserting \$ in each position i and running the BWT reversal algorithm, in a total of $\mathcal{O}(n^2)$ time. Here we present an $\mathcal{O}(n \log n)$ time algorithm for the problem. The underlying idea is that, if we know π_i, the standard permutation of $dol(w, i)$, then it is not too difficult to compute π_{i+1}.

Note that, in this section, we compute nice positions without considering pseudo-cycles (presented in Section 5.2). However, an algorithm exploiting them exists [34]. We will discuss later the advantages of choosing one algorithm over the other.

Lemma 14 *Let $w \in \Sigma^n$, and $0 \leq i \leq n-1$. Then*

i) $\pi_0(0) = 0$ *and for* $i > 0$, $\pi_0(i) = \pi_w(i-1) + 1$, *and*

ii) $\pi_{i+1} = (0, \pi_i(i+1)) \cdot \pi_i$.

In particular, the standard permutation π_{i+1} is the result of applying a single transposition to π_i.

Proof.

i) Follows by applying Lemma 11 to $i = 0$.

ii) First notice that for all $j \neq i, i+1$, $\pi_i(j) = \pi_{i+1}(j)$, by Lemma 11, since j is either smaller than both i and $i+1$, or larger than both i and $i+1$.

   In the first case (i.e. $j < i, i+1$), we have that $\pi_i(j) = \pi_w(j) + 1 = \pi_{i+1}(j)$. In the second case (i.e. $j > i, i+1$), we have that $\pi_i(j) = \pi_w(j-1)+1 = \pi_{i+1}(j)$. We have $\pi_i(i) = 0 = \pi_{i+1}(i+1)$, and $\pi_i(i+1) = \pi_w(i) + 1 = \pi_{i+1}(i)$, again by Lemma 11.

□

As we show next, applying a transposition to a permutation has either the effect of splitting a cycle, or of merging two cycles.

Lemma 15 *Let $\pi = C_1 \cdots C_k$ be the cycle decomposition of the permutation π, $x \neq y$, and $\pi' = (\pi(x), \pi(y)) \cdot \pi$.*

i) If x and y are in the same cycle C_i, then this cycle is split into two. In particular, let $C_i = (c_1, c_2, \ldots, c_j, \ldots, c_m)$, with $c_m = x$ and $c_j = y$. Then $\pi' = (c_1, c_2, \ldots, c_{j-1}, y)(c_{j+1} \ldots c_{m-1}, x) \prod_{\ell \neq i} C_\ell$.

ii) If x and y are in different cycles C_i and C_j, then these two cycles are merged. In particular, let $C_i = (c_1, c_2, \ldots, c_m)$, with $c_m = x$, and $C_j = (c'_1, c'_2, \ldots, c'_r)$, with $c'_r = y$, then $\pi' = (c_1, \ldots, c_{m-1}, x, c'_1, \ldots, c'_{r-1}, y) \prod_{\ell \neq i,j} C_\ell$.

Proof. Let $\tau = (\pi(x), \pi(y))$. First, note that $\pi'(z) = \tau(\pi(z)) = \pi(z)$ for all $z \neq x, y$.

i) $\pi'(x) = \tau(\pi(x)) = \pi(y) = c_{j+1}$ and $\pi'(y) = \tau(\pi(y)) = \pi(x) = c_1$, which proves the claim.

ii) Follows analogously. □

(See Example 5 for an example.)

Example 5 *We report $\sigma = \sigma_w$, followed by σ_i for $i = 0, \ldots, 10$. Changes from one permutation to the next are highlighted in red, and cyclic σ_i, i.e. nice positions i, are marked with a box. On the right, we note whether a merge or a split has taken place. By Lemma 15, this depends on whether i and $i+1$ are in distinct cycles or in the same cycle.*

$w = \texttt{acccbccbab}$

$\sigma = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 0 & 5 & 6 & 7 & 2 & 8 & 9 & 3 & 1 & 4 \end{smallmatrix}\right) = (0)(1,5,8)(2,6,9,4)(3,7)$

no. cycles

$$\sigma_0 = \left(\begin{smallmatrix} \mathbf{0} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \mathbf{0} & 1 & 6 & 7 & 8 & 3 & 9 & 10 & 4 & 2 & 5 \end{smallmatrix}\right) = (\mathbf{0})(\mathbf{1})(2,6,9)(3,7,10,5)(4,8)$$ *5*

merge

$$\sigma_1 = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \mathbf{1} & \mathbf{0} & 6 & 7 & 8 & 3 & 9 & 10 & 4 & 2 & 5 \end{smallmatrix}\right) = (0,\mathbf{1})(\mathbf{2},6,9)(3,7,10,5)(4,8)$$ *4*

merge

$$\sigma_2 = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & \mathbf{6} & \mathbf{0} & 7 & 8 & 3 & 9 & 10 & 4 & 2 & 5 \end{smallmatrix}\right) = (0,1,6,9,\mathbf{2})(\mathbf{3},7,10,5)(4,8)$$ *3*

merge

$$\sigma_3 = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 6 & \mathbf{7} & \mathbf{0} & 8 & 3 & 9 & 10 & 4 & 2 & 5 \end{smallmatrix}\right) = (0,1,6,9,2,7,10,5,\mathbf{3})(\mathbf{4},8)$$ *2*

merge

$$\boxed{\sigma_4 = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 6 & 7 & \mathbf{8} & \mathbf{0} & 3 & 9 & 10 & 4 & 2 & 5 \end{smallmatrix}\right) = (0,1,6,9,2,7,10,\mathbf{5},3,8,\mathbf{4})}$$ *1*

split

$$\sigma_5 = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 6 & 7 & 8 & \mathbf{3} & \mathbf{0} & 9 & 10 & 4 & 2 & 5 \end{smallmatrix}\right) = (0,1,\mathbf{6},9,2,7,10,\mathbf{5})(3,8,4)$$ *2*

split

$$\sigma_6 = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 6 & 7 & 8 & 3 & \mathbf{9} & \mathbf{0} & 10 & 4 & 2 & 5 \end{smallmatrix}\right) = (0,1,\mathbf{6})(9,2,\mathbf{7},10,5)(3,8,4)$$ *3*

merge

$$\sigma_7 = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 6 & 7 & 8 & 3 & 9 & \mathbf{10} & \mathbf{0} & 4 & 2 & 5 \end{smallmatrix}\right) = (0,1,6,10,5,9,2,\mathbf{7})(3,\mathbf{8},4)$$ *2*

merge

$$\boxed{\sigma_8 = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 6 & 7 & 8 & 3 & 9 & 10 & \mathbf{4} & \mathbf{0} & 2 & 5 \end{smallmatrix}\right) = (0,1,6,10,5,\mathbf{9},2,7,4,3,\mathbf{8})}$$ *1*

split

$$\sigma_9 = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 6 & 7 & 8 & 3 & 9 & 10 & 4 & \mathbf{2} & \mathbf{0} & 6 \end{smallmatrix}\right) = (0,1,6,\mathbf{10},5,\mathbf{9})(2,7,4,3,8)$$ *2*

split

$$\sigma_{11} = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 6 & 7 & 8 & 3 & 9 & 10 & 4 & 2 & \mathbf{5} & \mathbf{0} \end{smallmatrix}\right) = (0,1,6,10)(5,9)(2,7,4,3,8)$$ *3*

5.3.1 High-level description of the algorithm

The algorithm first computes the standard permutation $\pi = \pi_w$ of w and initializes a counter c with the number of cycles in π. It then computes π_0 according to Lemma 14, part 1. It increments counter c by 1 for $i = 0$, since 0 is always a fixpoint in π_0. Then the algorithm iteratively computes the new permutation π_{i+1}, updating c at each iteration. By Lemma 15, c either increases or decreases by 1 at every iteration: it increases if $i+1$ is in the same cycle as i, and it decreases otherwise. Whenever c equals 1, the algorithm reports the current value i. See Algorithm 1 for the pseudocode.

Algorithm 1: FINDNICEPOSITIONS(w)

Given a word w, return a set $\mathcal{I}$ of positions in which the \$-character can be inserted to turn w into a BWT image.

```
$n \leftarrow |w|$
$\pi \leftarrow$ standard permutation of $w$        // variant of counting sort
$c \leftarrow$ number of cycles of $\pi$
$\mathcal{I} \leftarrow \emptyset$
for $i \leftarrow n$ down to $1$ do                  // compute $\pi_0$ from $\pi$
    $\pi(i) \leftarrow \pi(i-1)$
$\pi(0) \leftarrow 0$
$c \leftarrow c+1$                         // $\pi_0$ has one more cycle than $\pi$
for $i \leftarrow 0$ to $n-1$ do
    $C \leftarrow$ cycle of $\pi$ which contains $0$         // $C$ also contains $i$
    if $i+1 \in C$ then
        $c \leftarrow c+1$                                   // case split
    else
        $c \leftarrow c-1$                                   // case merge
    $\pi \leftarrow$ UPDATE($\pi, i$)                        // now $\pi = \pi_{i+1}$
    if $c = 1$ then
        $\mathcal{I} \leftarrow \mathcal{I} \cup \{i+1\}$
return $\mathcal{I}$
procedure UPDATE($\pi, i$):                                  // $\pi(i) = 0$
    $\pi(i) \leftarrow \pi(i+1)$
    $\pi(i+1) \leftarrow 0$
    return $\pi$
```

5.3.2 Implementation with splay trees

For the algorithm's implementation, we need an appropriate data structure for maintaining and updating the current permutation π_i. Using an array to keep π_i would allow us to update it in constant time in each step, but would not give us the possibility to efficiently decide whether $i + 1 \in C$ in line 11. On the other hand, a standard balanced binary search tree would require an auxiliary data structure that, for each element, keeps track of the cycle where the element is. This auxiliary data structure would need to be updated at each split and merge operation in $\mathcal{O}(n)$ in the worst case. Thus, we need a data structure to maintain the cycles of π_i. The functionalities we seek are (a) deciding whether two elements are in the same cycle, (b) splitting two cycles, (c) merging two cycles. The data structure we have chosen is a forest of splay trees [81]. This data structure supports the above operations in amortized $\mathcal{O}(\log n)$ time.

Splay trees are self-adjusting binary search trees. They are not necessarily balanced, but they have the property that at every access-operation, the element x accessed is moved to the root and the tree is adjusted in such a way as to move nodes on the path from the root to x closer to the root, thus reducing access-time to these nodes for future operations. The basic operation, called *splaying*, consists of a series of the usual edge rotations in binary search trees. Which rotations are applied depends on the position of the node with respect to its parent and grandparent (the cases are referred to as *zig*, *zig-zig*, and *zig-zag*). Splay trees can implement the standard operations on binary search trees, such as *access, insert, delete, join, split* in amortized logarithmic time, in the total number of nodes involved. We refer the reader to the original article [81] for more details.

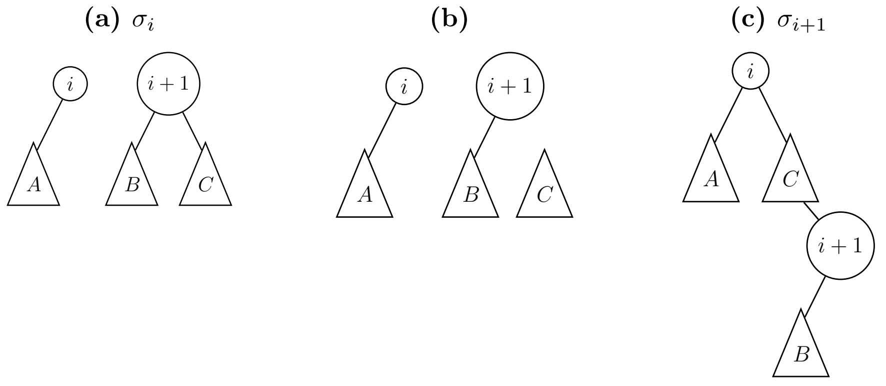
We represent the current permutation π_i as a forest of splay trees, where each tree corresponds to a cycle of π_i. Let $(c_0, c_2, \ldots, c_{k-1})$ be an arbitrary rotation of a cycle in π_i. We consider the cycle as a ranked list of the elements from c_0 to c_{k-1} and assign to the element c_j its position j as key. This way, the in-order traversal of the tree leads to the cycle. For a node v of the tree, the elements of the list that come before v are contained in the left subtree of v, and the elements that come after v are contained in the right subtree of v. Note that we always have 0 as the left-most node and i as the right-most node in the first cycle of π_i.

We now explain how to update the data structure.

If i and $i + 1$ are in distinct cycles of π_i, then the transposition of $0 = \pi_i(i)$ and $\pi_i(i + 1)$ leads to the merge of their cycles (Lemma 15). The implementation of the merge-step using splay trees is shown in Fig. 5.6. Let the two cycles have the form $(0, A, i)$ and $(B, i + 1, C)$, respectively, with A, B, C sequences of numbers. First, with the access operations on i and $i + 1$, these two elements move to the roots of their respective trees. This situation is displayed in (a). Now we have a *split* of the right subtree of node $i + 1$, the result of which is shown in (b). Next, a *join* operation links the C

subtree to the node i as its right child, and another *join* operation links node $i+1$, together with its left subtree B, as the right child of the rightmost node of C.

Figure 5.6: The implementation of the merge-step with splay trees.



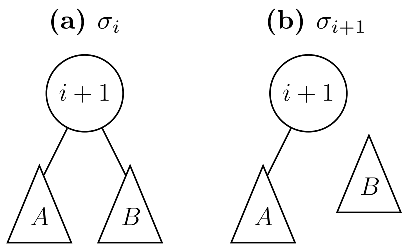
If i and $i+1$ are in the same cycle of π_i, then the transposition of $0 = \pi_i(i)$ and $\pi_i(i+1)$ leads to a split of the cycle (Lemma 15). The implementation of this operation with splay trees is shown in Fig. 5.7. Let the cycle have the form $(0, A, i+1, B, i)$. The corresponding splay tree after *access* $i+1$ is shown in (a). The *split* operation cuts the right subtree of $i+1$ producing the two new trees in (b).

5.3.3 Analysis

We now show that Algorithm 1 takes $\mathcal{O}(n \log n)$ time in the worst case.

Computing the standard permutation of w takes $\mathcal{O}(n)$ time (using a variant of Counting Sort [16], and noting that the alphabet of the string has cardinality at most n). The computation of π_0 (lines 5 to 7) takes $\mathcal{O}(n)$ time. All steps in one iteration of the for-loop (lines 9 to 17) take constant time, except deciding whether $i+1 \in C$ (line 11), and updating π (line 15). For deciding whether $i+1 \in C$, we *access* $i+1$. If the answer is YES, we will have a split-step: this is a *split*-operation on the tree for C (Fig. 5.7). If the answer is NO, then we *access* i, and merge the two trees (Fig. 5.6); the implementation of this consists of one *split*- and two *join*-operations on the trees.

Figure 5.7: The implementation of the split-step with splay trees.



Therefore, in one iteration of the for-loop, we either have one *access* and one *split* operation (for a split-step), or two *access*-, one *split*-, and two *join*-operations (merge-step), thus in either case, at most five operations. There are n iterations of the for-loop, thus at most $5n$ operations. Together with the initial insertion of the $n+1$ nodes, we get a total of $6n$ operations. We report the relevant theorem from [81]:

Theorem 7 (Balance Theorem with Updates, Theorem 6 in [81]) *A sequence of m arbitrary operations on a collection of initially empty splay trees takes $\mathcal{O}(m + \sum_{j=1}^{m} \log n_j)$ time, where n_j is the number of items in the tree or trees involved in operation j.*

For our algorithm, we have $m = \mathcal{O}(n)$ operations altogether, each involving no more than $n+1$ nodes, thus Theorem 7 guarantees that the total time spent on the splay trees is $\mathcal{O}(n + n \log n)$. Adding to this the computation of π_0 and the initialization of the splay trees, each in $\mathcal{O}(n)$ time, and of the constant-time operations within the for-loop, we get altogether $\mathcal{O}(n \log n)$ time. Memory usage is $\mathcal{O}(n)$, since the forest of splay trees consists of $n+1$ vertices in total. Summarizing, we have

Theorem 8 *Algorithm 1 runs in $\mathcal{O}(n \log n)$ time and uses $\mathcal{O}(n)$ space, for an input string of length n.*

Note that this algorithm does not exploit our characterization of nice positions. It is possible to retrieve the nice positions of a word by computing all the essential pseudo-cycles and the right-only pseudo-cycles of the word. An algorithm that does so is presented in [34], and it computes those pseudo-cycles in quadratic time. Even though the latter algorithm is slower than Algorithm 1, it is essential when information about the structure of the pseudo-cycles of the word is required.

5.4 Bounds and properties on nice positions

In this section, we study the number of nice positions of a given string w.

Definition 16 *For $w \in \Sigma^n$, let $h(w)$ denote the number of nice positions of w.*

We will first show that all nice positions of a word w have the same parity.

Theorem 9 *Let w be a word over Σ. Then either all nice positions are even, or all nice positions are odd. In particular, let c be the number of cycles in the standard permutation π_w; if c is even, then all nice positions are even, and if c is odd, then all nice positions are odd.*

Proof. Let us assume that $i < j$ are both nice, thus π_i and π_j are cyclic. This implies that $sgn(\pi_i) = (-1)^n = sgn(\pi_j)$, since both cycles consist of $n+1$ elements. By Lemma 14, $\pi_{i+1} = \tau_i \cdot \pi_i$, where $\tau_i = (0, \pi_i(i+1))$. Thus, $\pi_j = \tau_{j-1} \cdots \tau_i \cdot \pi_i$, so $sgn(\pi_j) = (-1)^{j-i} sgn(\pi_i)$, and therefore $sgn(\pi_j) = sgn(\pi_i)$ if and only if $j - i$ is even.

Given a cycle $C = (e_0, \ldots, e_{m-1})$, let $C' = (e_0 + 1, \ldots, e_{m-1} + 1)$. Now let $\pi_w = \prod_{j=1}^{c} C_j$ be the cycle decomposition of π_w. By Lemma 14, $\pi_0 = (0) \prod_{j=1}^{c} C'_j$. Therefore, $sgn(\pi_0) = (-1)^{n+1-(c+1)} = (-1)^{n-c}$. On the other hand, again by Lemma 14, $\pi_i = \tau_{i-1} \cdots \tau_1 \cdot \pi_0$, thus $sgn(\pi_i) = (-1)^{n-c+i-0}$. But this equals $(-1)^n$ if and only if c and i have the same parity. □

Corollary 7 *Let $w \in \Sigma^n$. Then $h(w) \leq \lfloor \frac{n+1}{2} \rfloor$.*

Proof. Follows from Theorem 9 and the fact that π_0 is not nice. □

Given the cycle decomposition of $\pi_w = \prod_{j=1}^{c} C_j$, let ℓ_j denote the minimum element of C_j, and $L = \max_{j=1,\ldots,c} \ell_j$.

Proposition 27 *If i is nice, then $i \geq L$. In particular, $i \geq c$, where c is the number of cycles of π_w.*

Proof. Note that every cycle C_j is a pseudo-cycle, where $S_{left} = \emptyset$ and $S_{right} = C_j$, with critical interval $[0, \ell_j]$. Therefore, by Theorem 6, no $i < L$ can be nice. The second claim follows since $L \geq \ell_c \geq c$. □

Corollary 8 *Let $w \in \Sigma^n$. Then $h(w) \leq \lceil \frac{n-L+1}{2} \rceil$.*

Proof. Follows from Theorem 9 and Proposition 27. □

We next derive some properties of nice positions from Algorithm 1. We show that, given π_i, there may exist a cycle C in π_i that blocks all positions before or after i, depending on the elements of C and i (see Proposition 29). We additionally show that we can count the number of specific pairs of indices of π_0 to derive a lower bound for the smallest nice position in the word (see Proposition 30).

Proposition 28 *Let c_i be the number of cycles of π_i. If j is nice and $j > i$, then $j \geq i + c_i$.*

Proof. The permutation π_j is computed from π_i by $j-i$ iterations of the for-loop of Algorithm 1 (lines 9-17), each of which either results in incrementing (split) or decrementing (merge) the number of cycles. Therefore, at least c_i steps are needed to arrive at a cyclic permutation. (Since $c_0 = c+1$, this implies in particular that for every nice position j, $j \geq c$, as already seen in Proposition 27.) □

Definition 17 *Let C be a cycle of π_i. We call C a* bad cycle w.r.t. *i if $i \notin [\min C, \max C]$.*

Example 6 (continued from p. 73) *The cycle $(4,8)$ is a bad cycle w.r.t. 3, and $(2,7,4,3,8)$ is a bad cycle w.r.t. 9.*

Proposition 29 *If C is a bad cycle w.r.t. i, then*

i) if $i < \min C$, then no $j \leq i$ is nice,

ii) if $i > \max C$, then no $j \geq i$ is nice.

Proof.

i) Let $i < \min C$. Then i is not nice, since π_i has at least two cycles. Now let $j < i$, thus $\pi_i = \tau_{i-1} \dots \tau_j \cdot \pi_j$, where $\tau_k = (0, \pi_k(k))$. Since $j \leq i < \min C$, it follows that each τ_k is disjoint from C, and since C is a cycle of π_i, therefore it is also a cycle of π_j. Since $[\min C, \max C] \neq \{0, \dots, n\}$, this implies that j is not nice.

ii) Analogously, if $i > \max C$, this implies that all π_j for $j \geq i$ have C as a cycle, implying that j is not nice.

□

Definition 18 *Let $\pi_w = \prod_{j=1}^{c} C_j$ be the cycle decomposition of π_w and $\ell_j = \min C_j$ for $j = 1, \dots, c$, where the cycles are in increasing order w.r.t. their minima, i.e. $\ell_1 < \dots < \ell_c$. We call a pair $(\ell_j, \ell_j + 1)$* bad pair *if $j < c$ and $\ell_j + 1 \in C_j$.*

Example 7 *Given the permutation $(0)(1,5,2,6,8,3)(4,7,9)$ with 3 cycles, the pair $(1,2)$ in the second cycle is a bad pair because 1 is the smallest element of its own cycle, and 2 is in the same cycle too. That is the only bad pair.*

Proposition 30 *Let b be the number of bad pairs in π_w. If i is a nice position of w, then $i \geq 2b + c$.*

Proof. We will count the number of iterations of the for-loop (lines 9-17) of Algorithm 1 before arriving at a cyclic permutation. Let us refer to the iterations as either merge- or split-steps. As we saw before, we need at least c merge-steps, since π_0 has $c+1$ cycles. It should also be clear that every additional split-step will necessitate a further merge-step. Therefore, it suffices to show that every bad pair results in a distinct split-step.

Let (ℓ_j, ℓ_j+1) be a bad pair. By Lemma 11, $\pi_0 = (0)\prod C'_j$, where $\min C'_j = \ell_j + 1$. Therefore, C'_j is a bad cycle w.r.t. ℓ_j, and thus is present in all π_i for $i \leq \ell_j$. Since $\ell_j \notin C'_j$, step ℓ_j is a merge-step. Now $\ell_j + 2$ is still in C'_j, so step $\ell_j + 1$ is a split-step. □

Example 8 (continued from page 79) *The standard permutation of the word* `abbababbbaa` *is* $(0)(1,5,2,6,8,3)(4,7,9)$. *There are two nice positions,* 5 *and* 7, *both greater or equal* $5 = 2+3 = 2b+c$.

We summarize:

Theorem 10 *Let w be a word over Σ and $\pi_w = \prod_{j=1}^{c} C_j$ the cycle decomposition of it standard permutation π_w.*

i) If i is nice, then $i \geq \max\{L+1, 2b+c\}$, where $L = \max_j \min C_j$, and b is the number of bad pairs in π_w.

ii) Let c_i be the number of cycles of π_i. If j is nice, then $j \geq i + c_i - 1$. Moreover, if π_i has a bad cycle C s.t. $i > \max C$, then no $j \geq i$ is nice.

5.5 Experimental results

In the following, we give some examples (Sec. 5.5.1), followed by some statistics on the number of nice positions (Sec. 5.5.2).

5.5.1 Examples

We list all words over $\{\mathtt{a}, \mathtt{b}\}$ of length 2, 3, 4, and 5 (Tables 5.1 to 5.4). For each word w (first column), we give the lexicographically smallest v such that $\mathrm{BWT}(v) = w$, if such a v exists, dashes otherwise (second column); the standard permutation $\pi = \pi_w$ (third column); and the number $h(w)$ of nice positions for w (fourth column). For each word w with $h(w) > 0$, we also list every $dol(w,i)$ with i nice, giving the analogous information, and specify i in the last (fifth) column.

In Table 5.5, the same information is shown about several longer strings over alphabets of sizes 2 and 3. They are ordered by string length ($n = 10, 13, 15, 18$). We chose these strings in order to give examples of as many different cases as possible.

w	v	π	$h(w)$	i
`aa`	$(\mathtt{a})^2$	$(0)(1)$	1	
`aa$`	`aa$`	$(0,1,2)$		2
`ab`	-	$(0)(1)$	1	
`ab$`	`ba$`	$(0,1,2)$		2

w	v	π	$h(w)$	i
`ba`	`ab`	$(0,1)$	1	
`b$a`	`ab$`	$(0,1,2)$		1
`bb`	$(\mathtt{b})^2$	$(0)(1)$	1	
`bb$`	`bb$`	$(0,1,2)$		2

Table 5.1: All strings w of length 2 over a binary alphabet. See text for details.

There are words that are the BWT of primitive words (strings 1, 7, 8, 12, 13, 20, 21, 22), some of which have the maximum number of nice positions according to their length (strings 7, 12, 20); two words have only one nice position (8, 13); string 1 is an example that shows that, once a position is nice, not necessarily all following positions with the same parity are also nice. Similarly, string 21 shows that there are no further nice positions after position 12 due to a bad cycle with respect to 13 containing only elements strictly smaller than 13.

We have BWTs of powers of primitive words (strings 2, 14, 15, 23, 24, 25), but only one of these has the maximum number of nice positions (string 2).

The table also contains strings that are not BWT (strings 3, 4, 5, 6, 9, 10, 11, 16, 17, 18, 19). Three of these have no nice positions (strings 6, 11, 19). Sometimes the parity of the number of cycles equals the parity of the smallest element of the last cycle (strings 3, 4, 5, 16, 18, 19), but this does not always happen (strings 6, 17).

w	v	π	$h(w)$	i
`aaa`	`aaa`	$(0)(1)(2)$	1	
`aaa$`	`aaa$`	$(0,1,2,3)$		3
`aab`	-	$(0)(1)(2)$	1	
`aab$`	`baa$`	$(0,1,2,3)$		3
`aba`	-	$(0)(1,2)$	1	
`ab$a`	`aba$`	$(0,1,3,2)$		2
`abb`	-	$(0)(1)(2)$	1	
`abb$`	`bba$`	$(0,1,2,3)$		3

w	v	π	$h(w)$	i
`baa`	`aab`	$(0,2,1)$	1	
`b$aa`	`aab$`	$(0,3,2,1)$		1
`bab`	-	$(0,1)(2)$	0	
`bba`	`abb`	$(0,1,2)$	2	
`b$ba`	`abb$`	$(0,2,3,1)$		1
`bba$`	`bab$`	$(0,2,1,3)$		3
`bbb`	`bbb`	$(0)(1)(2)$	1	
`bbb$`	`bbb$`	$(0,1,2,3)$		3

Table 5.2: All strings w of length 3 over a binary alphabet. See text for details.

w	v	π	$h(w)$	i
`aaaa`	`aaaa`	$(0)(1)(2)(3)$	1	
`aaaa$`	`aaaa$`	$(0,1,2,3,4)$		4
`aaab`	-	$(0)(1)(2)(3)$	1	
`aaab$`	`baaa$`	$(0,1,2,3,4)$		4
`aaba`	-	$(0)(1)(2,3)$	1	
`aab$a`	`abaa$`	$(0,1,2,4,3)$		3
`aabb`	-	$(0)(1)(2)(3)$	1	
`aabb$`	`bbaa$`	$(0,1,2,3,4)$		4
`abaa`	-	$(0)(1,3,2)$	1	
`ab$aa`	`aaba$`	$(0,1,4,3,2)$		2
`abab`	-	$(0)(1,2)(3)$	0	
`abba`	-	$(0)(1,2,3)$	2	
`ab$ba`	`abba$`	$(0,1,3,4,2)$		2
`abba$`	`baba$`	$(0,1,3,2,4)$		4
`abbb`	-	$(0)(1)(2)(3)$	1	
`abbb$`	`bbba$`	$(0,1,2,3,4)$		4

w	v	π	$h(w)$	i
`baaa`	`aaab`	$(0,3,2,1)$	1	
`b$aaa`	`aaab$`	$(0,4,3,2,1)$		1
`baab`	-	$(0,2,1)(3)$	0	
`baba`	`aabb`	$(0,2,3,1)$	1	
`b$aba`	`aabb$`	$(0,3,4,2,1)$		1
`babb`	-	$(0,1)(2)(3)$	0	
`bbaa`	`abab`	$(0,2)(1,3)$	2	
`bb$aa`	`abab$`	$(0,3,1,4,2)$		2
`bbaa$`	`baab$`	$(0,3,2,1,4)$		4
`bbab`	-	$(0,1,2)(3)$	1	
`bbab$`	`bbab$`	$(0,2,1,3,4)$		4
`bbba`	`abbb`	$(0,1,2,3)$	2	
`b$bba`	`abbb$`	$(0,2,3,4,1)$		1
`bbb$a`	`babb$`	$(0,2,4,1,3)$		3
`bbbb`	`bbbb`	$(0)(1)(2)(3)$	1	
`bbbb$`	`bbbb$`	$(0,1,2,3,4)$		4

Table 5.3: All strings w of length 4 over a binary alphabet. See text for details.

w	v	π	$h(w)$	i
`aaaaa`	`aaaaa`	$(0)(1)(2)(3)(4)$	1	
`aaaaa$`	`aaaaa$`	$(0,1,2,3,4,5)$		5
`aaaab`	-	$(0)(1)(2)(3)(4)$	1	
`aaaab$`	`baaaa$`	$(0,1,2,3,4,5)$		5
`aaaba`	-	$(0)(1)(2)(3,4)$	1	
`aaab$a`	`abaaa$`	$(0,1,2,3,5,4)$		4
`aaabb`	-	$(0)(1)(2)(3)(4)$	1	
`aaabb$`	`bbaaa$`	$(0,1,2,3,4,5)$		5
`aabaa`	-	$(0)(1)(2,4,3)$	1	
`aab$aa`	`aabaa$`	$(0,1,2,5,4,3)$		3
`aabab`	-	$(0)(1)(2,3)(4)$	0	
`aabba`	-	$(0)(1)(2,3,4)$	2	
`aab$ba`	`abbaa$`	$(0,1,2,4,5,3)$		3
`aabba$`	`babaa$`	$(0,1,2,4,3,5)$		5
`aabbb`	-	$(0)(1)(2)(3)(4)$	1	
`aabbb$`	`bbbaa$`	$(0,1,2,3,4,5)$		5
`abaaa`	-	$(0)(1,4,3,2)$	1	
`ab$aaa`	`aaaba$`	$(0,1,5,4,3,2)$		2
`abaab`	-	$(0)(1,3,2)(4)$	0	
`ababa`	-	$(0)(1,3,4,2)$	1	
`ab$aba`	`aabba$`	$(0,1,4,5,3,2)$		2
`ababb`	-	$(0)(1,2)(3)(4)$	0	
`abbaa`	-	$(0)(1,3)(2,4)$	2	
`abb$aa`	`ababa$`	$(0,1,4,2,5,3)$		3
`abbaa$`	`baaba$`	$(0,1,4,3,2,5)$		5
`abbab`	-	$(0)(1,2,3)(4)$	1	
`abbab$`	`bbaba$`	$(0,1,3,2,4,5)$		5
`abbba`	-	$(0)(1,2,3,4)$	2	
`ab$bba`	`abbba$`	$(0,1,3,4,5,2)$		2
`abbb$a`	`babba$`	$(0,1,3,5,2,4)$		4
`abbbb`	-	$(0)(1)(2)(3)(4)$	1	
`abbbb$`	`bbbba$`	$(0,1,2,3,4,5)$		5
`baaaa`	`aaaab`	$(0,4,3,2,1)$	1	
`b$aaaa`	`aaaab$`	$(0,5,4,3,2,1)$		1
`baaab`	-	$(0,3,2,1)(4)$	0	
`baaba`	`aaabb`	$(0,3,4,2,1)$	1	
`b$aaba`	`aaabb$`	$(0,4,5,3,2,1)$		1
`baabb`	-	$(0,2,1)(3)(4)$	0	
`babaa`	-	$(0,3,1)(2,4)$	0	
`babab`	-	$(0,2,3,1)(4)$	0	
`babba`	`aabbb`	$(0,2,3,4,1)$	1	
`b$abba`	`aabbb$`	$(0,3,4,5,2,1)$		1
`babbb`	-	$(0,1)(2)(3)(4)$	0	
`bbaaa`	`aabab`	$(0,3,1,4,2)$	3	
`b$baaa`	`aabab$`	$(0,4,2,5,3,1)$		1
`bba$aa`	`abaab$`	$(0,4,2,1,5,3)$		3
`bbaaa$`	`baaab$`	$(0,4,3,2,1,5)$		5
`bbaab`	-	$(0,2)(1,3)(4)$	1	
`bbaab$`	`bbaab$`	$(0,3,2,1,4,5)$		5
`bbaba`	-	$(0,2)(1,3,4)$	2	
`bb$aba`	`abbab$`	$(0,3,1,4,5,2)$		2
`bbab$a`	`baabb$`	$(0,3,5,2,1,4)$		4
`bbabb`	-	$(0,1,2)(3)(4)$	1	
`bbabb$`	`bbbab$`	$(0,2,1,3,4,5)$		5
`bbbaa`	`ababb`	$(0,2,4,1,3)$	2	
`b$bbaa`	`ababb$`	$(0,3,5,2,4,1)$		1
`bbbaa$`	`babab$`	$(0,3,1,4,2,5)$		5
`bbbab`	-	$(0,1,2,3)(4)$	0	
`bbbba`	`abbbb`	$(0,1,2,3,4)$	3	
`b$bbba`	`abbbb$`	$(0,2,3,4,5,1)$		1
`bbb$ba`	`babbb$`	$(0,2,4,5,1,3)$		3
`bbbba$`	`bbabb$`	$(0,2,4,1,3,5)$		5
`bbbbb`	`bbbbb`	$(0)(1)(2)(3)(4)$	1	
`bbbbb$`	`bbbbb$`	$(0,1,2,3,4,5)$		5

Table 5.4: All strings w of length 5 over a binary alphabet. See text for details.

	w	v	π	c	nice positions	$h(w)$
1	bcacbaaacb	aabccacabb	$(0,4,5,1,7,3,8,9,6,2$	1	$1,3,7,9$	4
2	ccaaaaaabb	(aaabc)^2	$(0,8,6,4,2),(1,9,7,5,3)$	2	$2,4,6,8,10$	5
3	cbcaaacaba	-	$(0,7,3),(1,5,2,8,6,9,4)$	2	$2,6,10$	3
4	accbcbaaaa	-	$(0),(1,7,2,8,3,5,6),(4,9)$	3	$5,7,9$	3
5	ccaaabcaac	-	$(0,6,8,4,2),(1,7,3),(5),(9)$	4	10	1
6	bacbacacab	-	$(0,4,1),(2,7,9,6),(3,5,8)$	3	-	0
7	bbaabbbbbbbba	aabbbbbbbbabb	$(0,3,1,4,5,6,7,8,9,10,11,12,2)$	1	$1,3,5,7,9,11,13$	7
8	babbbbabbbbba	aabbabbbbbbbb	$(0,3,5,7,8,9,10,11,12,2,4,6,1)$	1	1	1
9	abbabbbbbabba	-	$(0),(1,4,6,8,10,11,12,3),(2,5,7,9)$	3	$3,7,9$	3
10	abbaaaaaaaaaa	-	$(0),(1,11,9,7,5,3),(2,12,10,8,6,4)$	3	$3,5,7,9,11,13$	6
11	babbaaabaaaba	-	$(0,8,4,1),(2,9,5),(3,10,6),(7,11,12)$	4	-	0
12	bbaaaabbbbbbbba	aaabbbbbbbbbaab	$(0,5,3,1,6,7,8,9,10,11,12,13,14,4,2)$	1	$1,3,5,7,9,11,13,15$	8
13	babababababbbba	aaabaabbbbbbabb	$(0,6,9,4,8,10,11,12,13,14,5,2,7,3,1)$	1	1	1
14	bbbaaaaaaaaaaaa	(aaaab)^3	$(0,12,9,6,3),(1,13,10,7,4),(2,14,11,8,5)$	3	$3,5,9,11,15$	5
15	bbbbbaaaaaaaaaa	(aab)^5	$(0,10,5),(1,11,6),(2,12,7),(3,13,8),(4,14,9)$	5	$5,7,9,13,15$	5
16	bbaababaaabbbab	-	$(0,7,3,1,8,4,9,5,2),(6,10,11,12,13),(14)$	3	15	1
17	bbaababaaabbbaa	-	$(0,8,4,10,12,14,7,3,1,9,5,2),(6,11,13)$	2	$8,10,14$	3
18	bbabaabaaabbaaa	-	$(0,9,5,2),(1,10,13,7,3,11,14,8,4),(6,12)$	3	$9,11,13,15$	4
19	babbabbababaaab	-	$(0,7,2,8,12,5,10,13,6,11,4,1),(3,9),(14)$	3	-	0
20	bbaababbbabbabaaaa	aaababbabbababbaab	$(0,9,3,1,10,15,6,12,4,11,16,7,13,17,8,14,5,2)$	1	$1,3,5,7,9,11,13,15,17$	9
21	bbaaaaaaabbabbbbba	aaaaabbbbbbbaaaabb	$(0,9,11,7,5,3,1,10,12,13,14,15,16,17,8,6,4,2)$	1	$1,3,5,7,9,11$	6
22	bbbaaabaaabaaababa	aaabaaabaaababbbab	$(0,11,6,14,16,17,10,15,9,5,2,13,8,4,1,12,7,3)$	1	$1,5,17$	3
23	bbaaaabbbbaabbbbaa	(aaababbbb)^2	$(0,8,12,14,16,6,10,4,2),(1,9,13,15,17,7,11,5,3)$	2	$2,4,6,8,10$	5
24	bbbaaabbbbbbaaaaaa	(aababb)^3	$(0,9,15,6,12,3),(1,10,16,7,13,4),(2,11,17,8,14,5)$	3	$3,5,9,11$	4
25	bbbbbbbbbbbbaaaaaa	(abb)^6	$(0,6,12),(1,7,13),(2,8,14),(3,9,15),(4,10,16),(5,11,17)$	6	$6,8,10,12,16,18$	6

Table 5.5: Some examples of words over alphabets of size 2 and 3; of length 10, 13, 15 18. (See text for more details.)

5.5.2 Statistics

We present here some statistics to know how words of the same length are distributed with respect to the number of nice positions they have, i.e. for a given length n, how many words have 0 nice positions? How many have 1, 2, 3, ...? How many of these words are already BWT of some other words? Are they BWT of a primitive word or of a power of a primitive word?

In Tables 5.6 and 5.7, we present statistics on the number of nice positions $h(w)$. Table 5.6 contains the statistics for a binary alphabet and $n = 15, 16, 17, 18$. We give the statistics for all $n = 3, \ldots, 18$ in the Appendix (Table B.2). Table 5.7 contains the same information for a ternary alphabet and $n = 3, \ldots, 10$.

For fixed n, we give the absolute number of strings of length n with k nice positions (column 3), as well as the corresponding percentage (column 4). Percentages have been rounded to the next integer, where we write '0.5' for percentages x such that $0 < x < 0.5$. In columns 5 and 6, we give strings with k nice positions that are not BWT images, in absolute and percentage numbers; in columns 7 and 8, the same for BWT images. The last two columns contain a subdivision of column 7: the number of BWT images with k nice positions which are the BWT of a primitive word (column 9) and powers of primitive words (column 10).

Looking at the statistics, there are 0 words that are the BWT of some other word and have 0 nice positions. As we mention in section 5.2, BWT images have at least one nice position. An interesting fact that emerged from these statistics is that the number of words with 0 nice positions seems to slowly increase in percentage with the increase of their length: e.g. 40% for binary words of length 7, 50% for length 10, 60% for length 15, 63% for length 18, suggesting that there may exist some words which do not allow a $ insertion even when extended.

$\|\boldsymbol{\Sigma}\| = \mathbf{2}$								BWTs of	
	$h(w)$	all	%	noBWTs	%	BWTs	%	prim	pow
$n = 15$	0	19664	60	19664	64	0	0	0	0
	1	5874	18	4695	15	1179	54	1177	2
	2	1464	4	1381	5	83	4	83	0
	3	1940	6	1775	6	165	8	165	0
	4	1880	6	1637	5	243	11	242	1
	5	1218	4	991	3	227	10	222	5
	6	574	2	380	1	194	9	192	2
	7	140	< 0.5	53	< 0.5	87	4	87	0
	8	14	< 0.5	0	0	14	1	14	0
	total	32768	100	30576	100	2192	100	2182	10
$n = 16$	0	40094	61	40094	65	0	0	0	0
	1	11062	17	8843	14	2219	54	2217	2

$\mathbf{\vert\Sigma\vert = 2}$								BWTs of	
	$h(w)$	all	%	noBWTs	%	BWTs	%	prim	pow
	2	2882	4	2709	4	173	4	173	0
	3	3702	6	3346	5	356	9	353	3
	4	3622	6	3157	5	465	11	459	6
	5	2426	4	1988	3	438	11	427	11
	6	1298	2	980	2	318	8	309	9
	7	402	1	273	< 0.5	129	3	125	4
	8	48	< 0.5	30	< 0.5	18	< 0.5	17	1
	total	65536	100	61420	100	4116	100	4080	36
$n = 17$	0	81602	62	81602	66	0	0	0	0
	1	20898	16	16758	14	4140	54	4138	2
	2	5711	4	5423	4	288	4	288	0
	3	7146	5	6589	5	557	7	557	0
	4	6863	5	6139	5	724	9	724	0
	5	4820	4	4020	3	800	10	800	0
	6	2736	2	2112	2	624	8	624	0
	7	1042	1	639	1	403	5	403	0
	8	234	< 0.5	78	< 0.5	156	2	156	0
	9	20	< 0.5	0	0	20	< 0.5	20	0
	total	131072	100	123360	100	7712	100	7710	2
$n = 18$	0	165632	63	165632	67	0	0	0	0
	1	39704	15	31871	13	7833	54	7831	2
	2	11281	4	10696	4	585	4	585	0
	3	13798	5	12641	5	1157	8	1153	4
	4	13167	5	11672	5	1495	10	1485	10
	5	9620	4	8113	3	1507	10	1492	15
	6	5722	2	4579	2	1143	8	1119	24
	7	2450	1	1818	1	632	4	622	10
	8	696	< 0.5	474	< 0.5	222	2	218	4
	9	74	< 0.5	46	< 0.5	28	< 0.5	27	1
	total	262144	100	247542	100	14602	100	14532	70

Table 5.6: Statistics of words over a binary alphabet of lengths from 15 to 18. Percentages rounded to nearest integer. See text for further details.

$\mathbf{\vert\Sigma\vert = 3}$								BWTs of	
	$h(w)$	all	%	noBWTs	%	BWTs	%	prim	pow
$n = 3$	0	4	15	4	25	0	0	0	0
	1	19	70	12	75	7	64	4	3
	2	4	15	0	0	4	36	4	0
	total	27	100	16	100	11	100	8	3
$n = 4$	0	18	22	18	32	0	0	0	0

$\mathbf{\|\Sigma\| = 3}$	$h(w)$	all	%	noBWTs	%	BWTs	%	BWTs of prim	BWTs of pow
	1	45	56	31	54	14	58	11	3
	2	18	22	8	14	10	42	7	3
	total	81	100	57	100	24	100	18	6
$n = 5$	0	74	30	74	39	0	0	0	0
	1	109	45	83	43	26	51	23	3
	2	46	19	35	18	11	22	11	0
	3	14	6	0	0	14	27	14	0
	total	243	100	192	100	51	100	48	3
$n = 6$	0	258	35	258	43	0	0	0	0
	1	277	38	212	35	65	50	62	3
	2	130	18	94	16	36	28	29	7
	3	64	9	35	6	29	22	25	4
	total	729	100	599	100	130	100	116	14
$n = 7$	0	884	40	884	47	0	0	0	0
	1	709	32	564	30	145	46	142	3
	2	348	16	290	15	58	18	58	0
	3	202	9	134	7	68	22	68	0
	4	44	2	0	0	44	14	44	0
	total	2187	100	1872	100	315	100	312	3
$n = 8$	0	2870	44	2870	50	0	0	0	0
	1	1897	29	1509	26	388	46	385	3
	2	932	14	766	13	166	20	164	2
	3	648	10	456	8	192	23	176	16
	4	214	3	124	2	90	11	85	5
	total	6561	100	5725	100	836	100	810	26
$n = 9$	0	9208	47	9208	53	0	0	0	0
	1	5135	26	4164	24	971	44	968	3
	2	2558	13	2193	13	365	17	365	0
	3	1834	9	1452	8	382	17	378	4
	4	810	4	471	3	339	15	335	4
	5	138	1	0	0	138	6	138	0
	total	19683	100	17488	100	2195	100	2184	11
$n = 10$	0	29024	49	29024	55	0	0	0	0
	1	14059	24	11438	22	2621	44	2618	3
	2	7148	12	6085	11	1063	18	1059	4
	3	5290	9	4184	8	1106	19	1088	18
	4	2816	5	1965	4	851	14	828	23
	5	712	1	419	1	293	5	287	6
	total	59049	100	53115	100	5934	100	5880	54

Table 5.7: Statistics of words of length 3 to 10 over an alphabet of size 3. Percentages rounded to nearest integer. See text for further details.

5.6 Inserting $ in fully clustered words

In this section, we present properties of nice positions in fully clustered words. Recall that a fully clustered word is a word that has one run per character, such as `cccabb`. We will study these questions for different alphabets, where we restrict our attention to words w in which every character in the alphabet appears at least once. We are interested in the number $h(w)$ of nice positions of a word w, and in the number of words $H(k) = H_n(k)$ which have k nice positions, for given length n.

As an example, for $\sigma = 2$, there are 10 fully clustered words of length 6, 6 of which have 1 nice position, 2 have 2 nice positions, and 2 have 3, see Table 5.8 below. Of the 60 fully clustered words for $\sigma = 3$, 5 have 0 nice positions, 26 have 1, 16 have 2, and 13 have 3 (see Table B.1 in the Appendix).

word	$h(w)$	nice positions
`abbbbb`	1	6
`aabbbb`	1	6
`aaabbb`	1	6
`aaaabb`	1	6
`aaaaab`	1	6

word	$h(w)$	nice positions
`baaaaa`	1	1
`bbaaaa`	3	2, 4, 6
`bbbaaa`	2	3, 5
`bbbbaa`	2	2, 4
`bbbbba`	3	1, 3, 5

Table 5.8: Nice positions of fully clustered binary words of length 6. We report the number of nice positions and the nice positions for each word.

First, we gather some useful properties using pseudo-cycles:

Lemma 16 *Let w be any word and π_w its standard permutation. The following hold:*

i) If $n-1$ is a fixpoint, then no $i < n$ is nice (by Theorem 6).

ii) If $\pi(1) = 0$, then no $i > 1$ is nice (by Theorem 6).

iii) n is nice if and only if π_w has no left-only pseudo-cycle.

iv) 0 is not nice.

Proof.

i) If $(n-1)$ is a fixpoint, then $\{n-1\}$ is a right-only pseudo-cycle, so it blocks all elements from 0 to $n-1$.

ii) If $\pi(1) = 0$, then $\{1\}$ is a left-only pseudo-cycle, blocking any $i > 1$.

iii) If n is nice, then it cannot be in the critical interval of any pseudo-cycle. But the pseudo-cycles whose critical interval contains n are exactly the left-only pseudo-cycles, since if there is a non-empty right part, then its minimum must be smaller than n.

iv) The entire set $\{0, \ldots, n-1\}$ is a right-only pseudo-cycle, so its minimum 0 blocks all $i \leq 0$.

□

Lemma 17 *Let w be any word, $|w| = n$. If π_w is the identity permutation, then $h(w) = 1$. In particular, n is the only nice position.*

Proof. $\pi_w = (0)(1) \cdots (n-1)$ (in cycle representation), consisting of n fixpoints. Since $n-1$ is a fixpoint, by Lemma 16, no $i < n$ can be nice. Clearly, π_w has no left-only pseudo-cycles, so again by Lemma 16, n is nice. □

Since the standard permutation of any word over a unary alphabet is the identity permutation, these words have exactly one nice position:

Corollary 9 *If w is a word over an alphabet of size $\sigma = 1$, then $h(w) = 1$.*

5.6.1 Number of nice positions for alphabets of size 2

Our first result says that every fully clustered word over a binary alphabet has at least one nice position.

Proposition 31 *If w begins with an* a*, then $h(w) = 1$. If w begins with a* b*, then $h(w) \geq 1$.*

Proof. First, let $w = \mathtt{a}^i\mathtt{b}^{n-i}$, for some $1 \leq i \leq n-1$. Then $\pi_w = id$, thus, by Lemma 17, there is exactly one nice position, namely n (See Figure 5.8).

Now let $w = \mathtt{b}^i\mathtt{a}^{n-i}$, for some $1 \leq i \leq n-1$. It is known that a word of this form is the BWT of a standard word, or of a power of a standard word [64]. Thus, by Theorem 4 (the result of Likhomanov and Shur [52]), the number of cycles c of π_w equals the greatest common divisor of the runlengths, namely $c = \gcd(i, n-i)$. By Theorem 5, c is nice. □

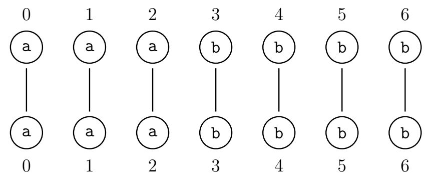
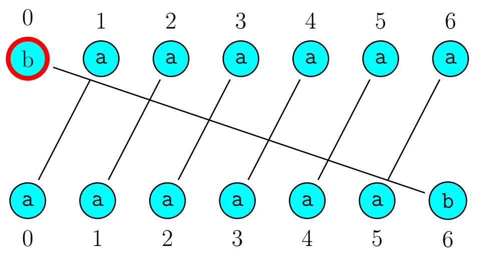


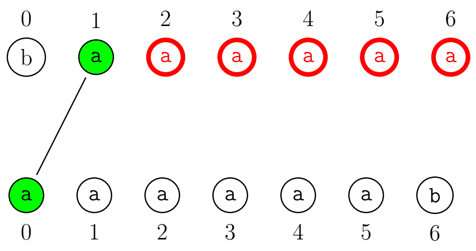
Figure 5.8: The right-only pseudo-cycles of the word aaabbbb. All of them are fixpoints, therefore only position 7 is eligible to be nice.

Proposition 32 *Let $w = \mathtt{b}\mathtt{a}^{n-1}$. Then $h(w) = 1$.*

Proof. The standard permutation of w is $\pi_w = (n-1, 0, 1, \ldots, n-2)$. By Theorem 5, 1 is nice, π_w has one cycle. 0 is not nice by Lemma 16. Every $i > 0$ is a singleton left-only pseudo-cycle, since $\pi(i) = i - 1$, while all other pseudo-cycles are subsets of $\{1, \ldots, n-1\}$, and thus left-only. Every left-only pseudo-cycle $\{i\}$ is i-essential, blocking $[i+1, n]$, i.e. the only position which is nice is 1 (Fig. 5.9).



(a) $S_1 = \{0,1,2,3,4,5,6\}$, $\pi(S_1) = \{0,1,2,3,4,5,6\}$, $R_1 = [0,0]$.



(b) $S_2 = \{1\}$, $\pi(S_2) = \{0\}$, $R_2 = [2,7]$.

Figure 5.9: Two pseudo-cycles S_1, S_2 in the word `baaaaaa` and their critical intervals R_1, R_2, which block all positions of the word except for 1.

Proposition 33 *Let $w = \mathtt{b}^i\mathtt{a}^{n-i}$ where $c = \gcd(i, n-i) > 1$. Then w has at least two nice positions, namely c and $c+2$.*

Proof. Let $d = \frac{n}{c}$. By Corollary 4 the standard permutation of w is

$$\pi = (0, e_1, \ldots, e_{d-1})(1, e_1+1, \ldots, e_{d-1}+1)\ldots(c-1, e_1+c-1, \ldots, e_{d-1}+c-1).$$

By Theorem 6 position c is nice. In particular, the standard permutation of the word with \$ in position c has the form

$$\pi_c = (0, e_1+1, \ldots, e_{d-1}+1, 1, e_1+2, \ldots, e_{d-1}+2, 2, \ldots, e_1+c, \ldots, e_{d-1}+c, c).$$

Note that each element of π is increased by one except for the first element of the cycles, which are the positions smaller than c. The order of the elements in the cycle of π_c is as follows: first there are all the elements of the cycle of π containing 0 (increased by one accordingly), then all the elements of the cycle of π containing 1, and so on. Positions c and $c+1$ were respectively in the first and the second cycle of π, therefore they occur (increased by 1) in the same order in π_c. When \$ is moved by one position to the right in the word (namely, from position c to position $c+1$), then c and $c+1$ are swapped in π_c, leading to a split of the cycle. The split breaks the cycle right after $c+1$, generating two cycles: one containing $c+1$, and the other one containing $c+2$ and c. Moving \$ by one further position in the word causes the swap of $c+1$ and $c+2$ in the permutation. Since they are in two distinct cycles in π_{c+1}, then the cycles are merged into one in π_{c+2}. Thus, π_{c+2} consists of one single cycle, $dol(w, c + 2)$ is a BWT, and therefore $c + 2$ is nice. $\square$

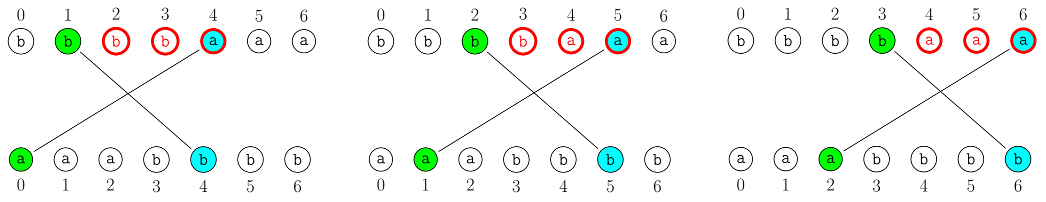
Example 9 *Consider the word* `bbbaaaaaa` *and its standard permutation* $\pi = (0, 6, 3)(1, 7, 4)(2, 8, 5)$. *The first nice position is 3, and the word* `bbb`$\underline{\texttt{\$}}$`aaaaaa` *has the following standard permutation* $\pi_3 = (0, 7, 4, 1, 8, 5, 2, 9, 6, \underline{3})$. *Moving the dollar in the next position, we have* `bbba`$\underline{\texttt{\$}}$`aaaaa` *with standard permutation* $\pi_4 = (0, 7, \underline{4})(1, 8, 5, 2, 9, 6, 3)$. *Finally, we arrive at* `bbbaa`$\underline{\texttt{\$}}$`aaaa` *with standard permutation* $\pi_5 = (0, 7, 4, 2, 9, 6, 3, 1, 8, \underline{5})$.

Proposition 34 *Let* $w = \mathtt{b}^{j+1}\mathtt{a}^j$. *Then* $h(w) = 2$, *in particular,* 1 *and* n *are nice.*

Proof. First, since w is a BWT, the number of cycles c of π_w is nice, by Theorem 5. Since the runlengths j and $j + 1$ are relatively prime, $c = 1$, so 1 is nice.

Let $0 < i \leq j$. The i-essential pseudo-cycle is $S_i = \{i, i + j\}$, since $\pi(i) = i + j$ and $\pi(j) = i + j - (j + 1) = i - 1$. The critical interval of S_i is $R_i = [i + 1, i + j]$. The union of these critical intervals is $[2, n - 1] \subseteq R_w$, since $n = |w| = 2j + 1$.

Finally, no i-essential pseudo-cycles exist for $i > j$. Therefore, there are no left-only pseudo-cycles, and thus, by Lemma 16, n is nice. (See Figure 5.10.) $\square$



(a) $S_1 = \{1, 4\}$, $\pi(S_1) = \{0, 4\}$, $R_1 = [2, 4]$, boundary $i = 1$.

(b) $S_2 = \{2, 5\}$, $\pi(S_2) = \{1, 5\}$, $R_2 = [3, 5]$, boundary $i = 2$.

(c) $S_3 = \{3, 6\}$, $\pi(S_3) = \{2, 6\}$, $R_3 = [4, 6]$, boundary $i = 3$.

Figure 5.10: Three pseudo-cycles S_1, S_2, S_3 of the same form in the word `bbbbaaa`, and their critical intervals R_1, R_2, R_3. (S_{left} in green, S_{right} in blue, critical intervals in red.)

5.6.2 Number of words with k nice positions over alphabet size 2

We now turn to the number of words with k nice positions.

Definition 19 *For $n > 0, k \geq 0$, let $H_n^{\pi}(k)$ denote the number of fully clustered words of length n with exactly k nice positions over an alphabet of size σ. When it is clear from the context, we drop the superscript π.*

From Proposition 31 it follows that $H_n^2(0) = 0$, and from Proposition 31 and 32 it follows that $H_n^2(1) \geq n$, since the $n-1$ words beginning with $\mathtt{a}$ and the word $\mathtt{ba}^{n-1}$ have 1 nice position.

For a better understanding of the words with k nice positions, we ran experiments on fully clustered words up to length 100, and studied their pseudo-cycles. These led to the following conjectures:

Conjecture 1 *For all n, $H_n^2(1) = n$.*

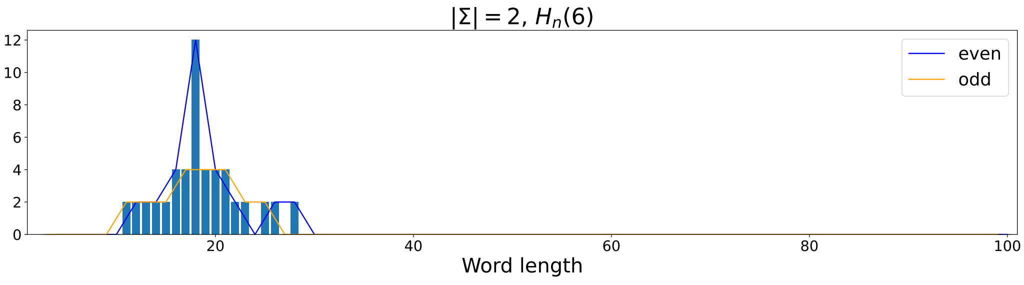
Conjecture 2 *For n even, $n \geq 8$, $H_n^2(2) = 0$. For n odd, $H_n^2(2) = 1$.*

Conjecture 3 *For every n, $H_n^2(\lceil \frac{n}{2} \rceil) = 2$.*

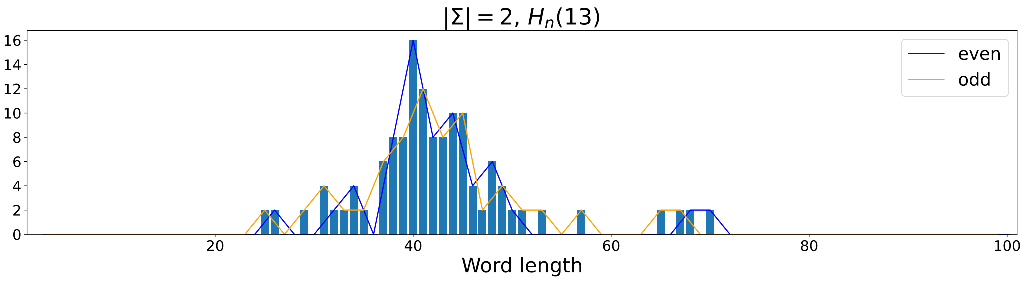
The conjectured two words with $\lceil n/2 \rceil$ nice positions are $\mathtt{bba}^{n-2}$ and $\mathtt{b}^{n-1}\mathtt{a}$. This is because both have $\lfloor n/2 \rfloor$ pseudo-cycles whose critical intervals contain exactly one position each. From Theorem 9, we know that for words that are BWT images, the parity of nice positions is the same as the parity of the number of cycles. In these two cases, the pseudo-cycles block every position that has the opposite parity, leaving all the remaining positions available.

All our conjectures are supported by the histograms we produced for words up to length 100 and k nice positions, k up to 19 (see Figure 5.11). Furthermore, the words show regular behavior with distinct k's. In particular, we can see that the 8 longest words (4 of even length, 4 of odd length) are disconnected from all the others. Moreover, the greater k, the farther this group of 8 words is from the others. The first k where this group is visible is $k = 6$, and we identify the 4 words of even length as follows: two primitive words ($w_1 = \mathtt{b}^7\mathtt{a}^{13}$, $w_2 = \mathtt{b}^{15}\mathtt{a}^7$), two power words ($w_3 = \mathtt{b}^{14}\mathtt{a}^6$, $w_4 = \mathtt{b}^8\mathtt{a}^{14}$); and the 4 primitive words of odd length ($w_5 = \mathtt{b}^8\mathtt{a}^{15}$, $w_6 = \mathtt{b}^{16}\mathtt{a}^7$, $w_7 = \mathtt{b}^9\mathtt{a}^{16}$, $w_8 = \mathtt{b}^{17}\mathtt{a}^8$).

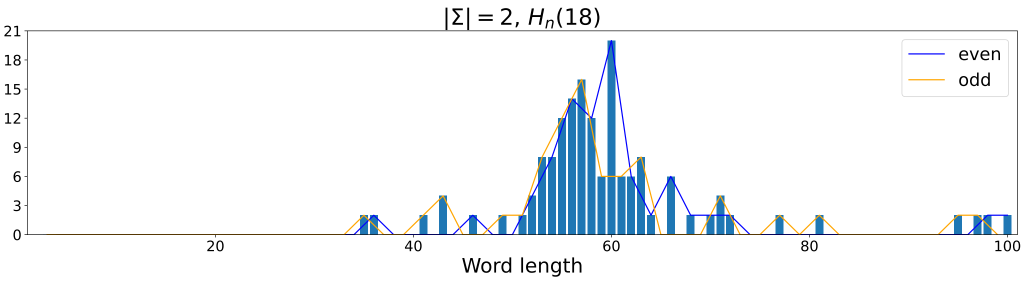
These words give an idea of the regularity of $H_n^2(k)$ across different k. Consider $w_1 = \mathtt{b}^7\mathtt{a}^{13}$, and the same word with 4 more $\mathtt{a}$'s and 2 more $\mathtt{b}$'s, namely $w_1' = \mathtt{b}^9\mathtt{a}^{17}$. In both cases, there are $|w_1|_{\mathtt{b}} - 1$ number of pseudo-cycles that have the same form, and starting from the one with the smallest boundary, they are shifted by one position to the right. Moreover, the critical interval of each of these pseudo-cycles has length $|w_1|_{\mathtt{b}} - 1$ (in total, they cover $2 \cdot |w_1|_b - 1$ positions). On the other hand, there are $\frac{|w_1| - 2 \cdot |w_1|_{\mathtt{b}} - 1}{2}$ additional pseudo-cycles blocking just one position each, e.g. 5 and 7 in w resp. w'. For this reason, $h(w_1) = |w_1| - \frac{|w_1| - 2 \cdot |w_1|_{\mathtt{b}} - 1}{2} - 2 \cdot |w_1|_{\mathtt{b}} - 1$ and $h(w_1') = |w_1'| - \frac{|w_1'| - 2 \cdot |w_1'|_{\mathtt{b}} - 1}{2} - 2 \cdot |w_1'|_{\mathtt{b}} - 1$, and therefore $h(w_1') = h(w_1) + 1$. We observed this phenomenon holds adding iteratively the same number of $\mathtt{a}$'s and $\mathtt{b}$'s (up to length 100). Further, the same happens adding 2 $\mathtt{b}$'s and 4 $\mathtt{a}$'s



(a) Number of words of lengths from 0 to 100 with $k = 6$ nice positions.



(b) Number of words of lengths from 0 to 100 with $k = 13$ nice positions.



(c) Number of words of lengths from 0 to 100 with $k = 18$ nice positions.

Figure 5.11: In the figure, the number of words $H_2(k)$ having $k = 6, 13, 18$ nice positions are plotted for each length from 0 to 100. The blue line highlights even lengths, while the yellow line the odd ones.

k	3	4	5	6	7	8	9	10	11	12	13	14	15	16
n_k	10	16	22	28	34	40	46	52	58	64	70	76	82	88

Table 5.9: Conjecture 4: There are no binary words of length greater than n_k with k nice positions.

to w_4, w_5, w_7, and adding 4 b's and 2 a's to w_2, w_3, w_6, w_8. Our experiments suggest that for fixed k, there are no words with greater length than those above, see Table 5.9.

On the basis of these observations, we conjecture that the set $\mathcal{F}^2(k)$ of binary words with k nice positions is finite. Formally:

Conjecture 4 *For every $k \geq 3$, there exists a length n_k such that no word of length greater than n_k has exactly k nice positions.*

Finally, we noticed from our experiments that the smallest nice position is always a divisor of n. Let d divide n. As we have seen before, $w = \mathtt{b}^i\mathtt{a}^j$ is the BWT of a word u^d with u a standard word and $d = \gcd(j, i)$. Thus, d is nice by Theorem 5. The standard permutation of w has d cycles, and the largest minimum of a cycle is $d - 1$, blocking all positions $i \leq d - 1$. Thus, given n, d where d is a divisor of n, e.g. the word $\mathtt{b}^d\mathtt{a}^{(\frac{n}{d}-1)d}$ has smallest nice position d. We conjecture that the converse is true also:

Conjecture 5 *An integer d can occur as a smallest nice position for some fully clustered word of length n if and only if d divides n.*

5.6.3 Larger alphabets

We partially defined a mapping from fully clustered words over a ternary alphabet having k nice positions to fully clustered words over a binary alphabet having at least k nice positions. We could not define a mapping for ternary words of the form $\mathtt{c}^i\mathtt{b}^j\mathtt{a}^\ell$, $i, j, \ell > 1$.

For a word $w = c_0^{i_0}c_1^{i_1}\cdots c_{r-1}^{i_{r-1}}$ the *pattern* $pat(w) = c_0c_1\cdots c_{r-1}$ is the concatenation of a single occurrence of a character for each run $c_0, c_1, \ldots, c_{r-1}$. E.g. the word bbaaaabbaac has 5 runs and $pat(\mathtt{bbaaaabbaac}) = \mathtt{babac}$.

In Table B.1 in the Appendix, we report all ternary fully clustered words with the number of nice positions for each word and the position where \$ can be inserted.

Definition 20 *We define the following mapping $g : \mathcal{F}^3 \to \mathcal{F}^2$*

i) $pat(w) = \mathtt{abc}; w = \mathtt{a}^i\mathtt{b}^j\mathtt{c}^\ell, g(w) = \mathtt{a}^i\mathtt{b}^j$

ii) $pat(w) = \mathtt{acb}; w = \mathtt{a}^i\mathtt{c}^j\mathtt{b}^\ell, g(w) = \mathtt{b}^j\mathtt{a}^\ell$

iii) $pat(w) = \mathtt{bac}; w = \mathtt{b}^i\mathtt{a}^j\mathtt{c}^\ell, g(w) = \mathtt{b}^i\mathtt{a}^j$

iv) $pat(w) = \texttt{bca}; w = \texttt{b}^i\texttt{c}^j\texttt{a}^\ell, g(w) = \texttt{b}^{i+j}\texttt{a}^\ell$

v) $pat(w) = \texttt{cab}; w = \texttt{c}^i\texttt{a}^j\texttt{b}^\ell, g(w) = \texttt{b}^i\texttt{a}^{j+\ell}$

A preliminary study on inferring nice positions for ternary words from their corresponding binary words is summarized in the following Proposition. We show that given $w \in \mathcal{F}^3$ we can go to $\mathcal{F}^2$ mapping the length of the word and the number of the nice positions. For example, consider the word `aaacccccbb` having length 10 and three nice positions 4, 8, 10. If we map it to the corresponding binary word `bbbbbaa`, the length is now 7, and positions 1,5,7 are nice.

Proposition 35 *Referring to the numbering in Definition 20, we state that:*

i) $pat(w) = \texttt{abc}; h(w) = h(g(w)) = 1$ *and* $|w| = |g(w)| + |w|_\texttt{c}$

ii) $pat(w) = \texttt{acb}; h(w) = h(g(w))$ *and* $|w| = |g(w)| + |w|_\texttt{a}$

iii) $pat(w) = \texttt{bac}; h(w) \leq \min\{h(g(w)), 1\}$ *and* $|w| = |g(w)| + |w|_\texttt{c}$

iv) $pat(w) = \texttt{bca}; h(w) = h(g(w))$ *and* $|w| = |g(w)|$

v) $pat(w) = \texttt{cab}; h(w) = h(g(w))$ *and* $|w| = |g(w)|$

Proof. i) The standard permutation of both $g(w)$ and w is $\pi_{g(w)} = \pi_w = id$, then from Lemma 17 the only nice position of $g(w)$ is $|g(w)|$, that of w is $|w|$ and $h(w) = h(g(w))$.

ii) In the standard permutation of w the first $|w|_a$ elements are fixpoints. By removing these characters in w, the resulting permutation is the same except for the initial fixpoints. The form of π_w is preserved but shifted in $\pi_{g(w)}$, and the value of the indices is decreased by $|w|_a$. Therefore, every nice position of w decreased by $|w|_a$ is a nice position in $g(w)$.

iii) In this case, w has $|w|_c$ fixpoints at the end, blocking every position before. Therefore, the only candidate position to be nice is $|w|$. By removing the last $|w|_c$ characters, we remove the fixpoints that block the possible nice positions in the first part of the standard permutation, therefore we can have more nice positions in the binary word.

iv), v) Clearly, $\pi_w = \pi_{g(w)}$, and therefore $h(w) = h(g(w))$.

Example 10 *Referring to i): Given the word* $w =$ `aabbbbccc`*, its standard permutation is* $\pi_w = id$*. Therefore, the only nice position is* 9*. If we apply the mapping we get* $g(w) =$ `aabbbb`*, then* $\pi_{g(w)} = id$*, and the cycles block every position up to 5, leaving out 6, which is the only nice position.*

Referring to ii): $w =$ `aaccccbb` $\pi_w = (0)(1)(2, 4, 6)(3, 5, 7)$*, 4 and 6 begin nice positions. By removing the two* `a`*'s we get* $g(w) =$ `bbbbaa` *with* $\pi_{g(w)} = (0, 2, 4)(1, 3, 5)$ *resulting in 2 and 4 being nice.*

Referring to iii): $w =$ `bbbbbbaaac`*,* $\pi_w = (0, 3, 6)(1, 4, 7)(2, 5, 8)(9)$*, the only nice position is 10. By removing the only* `c` *at the end we get* $g(w) =$

`bbbbbbaaa` *with* $\pi_{g(w)} = (0,3,6)(1,4,7)(2,5,8)$ *having the following nice positions: 3, 5, 7, 9.*

Referring to iv), v)): $w =$ `cccaaaabbbbbb`, $\pi_w = (0,10,7,4,1,11,8,5,2,12,9,6,3)$, $g(w) =$ `bbbaaaaaaaaaa`, $\pi_{g(w)} = (0,10,7,4,1,11,8,5,2,12,9,6,3)$, *therefore the nice positions are the same in the two words, namely 1, 3, 7, 9, 13.*

Based on Proposition 35 we believe that there is a constant number of words longer than a certain length with k nice positions. (For binary alphabet this phenomenon is described in Conjecture 4.) This is suggested by the fact that among the words involved in the mapping, only those of the pattern `acb` maintain the same number of nice positions while increasing in length.

This result suggests that some regularities on the number of words $H(k)$ having a fixed number of nice positions are preserved in ternary, and maybe even larger, alphabets.

Chapter 6

Two Fundamental Text Compressors in Bioinformatics

In this chapter, we present two applications in bioinformatics, one for BWT and one for LZ77.

6.1 MUM-PHINDER: BWT application in MUM search

Maximum Unique Matches (MUMs) are substrings of a pattern that occur only once in a reference text and in the pattern, and that cannot be extended in either direction. Maximal Exact Matches (MEMs) are a superset of MUMs that are not necessarily unique.

Formally, given a text $T[0..n-1]$ and a pattern $P[0..m-1]$, we refer to any factor in P that also occurs in T as a *match*. A match u in P is defined as a pair (i, ℓ) such that $w = P[i..i+\ell-1]$. The match u is *maximal* if it cannot be extended neither on the left nor on the right, i.e. either $i = 0$ or $P[i-1..i+\ell-1]$ does not occur in T and either $i = m-\ell$ or $P[i..i+\ell]$ does not occur in T.

Definition 21 (Maximal Unique Matches) *Given a text T and a pattern P, a Maximal Unique Match (MUM) is a maximal match that occurs exactly once both in T and in P.*

Example 11 *Let $T = ACACTCTTAC\mathbf{ACC}ATATCATCAA\$$ be the text and $P = AACCTAA$ the pattern. The factor* **AA** *is maximal in P and occurs only once in T, while it is repeated in P at positions* 0 *and* 5*. The factor* **CT** *of P starting in position 3 is a maximal exact match that occurs only once in P, but it is not unique in T. The factor* **CC** *of P starting in position 2 is unique in both T and P, but both can be extended on the left with an* **A**. *Finally, the factor $P[1..3] = T[10..12] =$* **ACC** *is a MUM.*

MEMs and MUMs have proven themselves to be useful in multiple bioinformatics problems, such as short read alignment and multiple genome alignment. Standard techniques to compute them on genomes use suffix trees and the FM-index, which is the base of widely used bioinformatics tools such as Bowtie [49] and BWA [51]. Recently, pangenomes received increasing attention from the scientific community for their ability to incorporate population variation information and alleviate reference genome bias. With the advent of third-generation sequencing, the quality of assembled genomes drastically increased. Very recently, the Telomere-to-Telomere project released the first complete haploid human genome [71] and the Human Pangenome Reference Consortium (HPRC) plans to release hundreds of high-quality assembled genomes to be used as a pangenome reference. One important step to enable the use of these high-quality assembled genomes is to build a multiple sequence alignment of the genomes. However, the FM-index does not scale well to a pangenomic level. Although the BWT can be stored and queried in compressed space [60], the size of the FM-index grows with the length of the uncompressed text. Tools like MUMmer [46, 65] and Mauve [18] proposed a solution to the original problem of multiple sequence alignment by using Maximal Unique Matches (MUMs) between two input sequences as prospective anchors for an alignment. Recently, Gagie et al. [30] introduced the r-index, which is a BWT-based index able to handle hundreds of human genomes, and whose size grows with the number of runs of the BWT. The r-index is a text index composed of the run-length encoded BWT and the Suffix Array (SA) sampled at run boundaries, i.e. in correspondence of the first and last character of a run of the BWT, and it can retrieve the missing values of the SA by using a predecessor data structure on the samples of the SA. However, the r-index itself cannot find MEMs nor MUMs. Bannai et al. [5] introduced a variant of the r-index that supports finding MEMs. In [8, 75], the authors proposed a tool to index hundreds of human genomes and to query such an index to find MEMs using the r-index variant described in [5]. The tool is called PHONI [8], and is built on top of an r-index [30] and a straight-line program (SLP) [28]. Their main objective is to compute the so-called matching statistics (MS) of the pattern with respect to the text, which can be used to compute the MEMs with a linear scan. The SLP is used to compute efficient longest common extension (LCE) queries, which allows computing the matching statistics and the MEMs with only one scan of the query. Overall, PHONI computes the MS array of a pattern of length m in $\mathcal{O}(m \cdot (t_{\mathrm{LF}} + t_{\mathrm{LCE}} + t_{pred}))$ time, where t_{LF}, t_{LCE}, and t_{pred} represent the time to perform respectively one LF-mapping step, one LCE, and one predecessor query.

In this section, we present MUM-PHINDER, an algorithm to compute MUMs that builds on the approach of Boucher et al. [8] for the computation of the MS array. Our algorithm extends Boucher et al.'s method by storing additional $\mathcal{O}(r)$ samples of the LCP array. Given a text $T[0..n-1]$ and a pattern $P[0..m-1]$, in the following, we show how we extended the algorithm for finding MEMs in order to find unique occurrences of the matches, i.e. MUMs.

We evaluated MUM-PHINDER on real-world datasets. We tested our algorithm against MUMmer [65], and we measured time and memory required by both tools for sets of increasing size of haplotypes of human chromosome 19 and SARS-CoV2 genomes. We queried them using one haplotype of chromosome 19 and one SARS-CoV2 genome not present in the dataset. We report that MUM-PHINDER requires consistently less memory than MUMer for all experiments, being up to 25 times smaller. Although MUMer is generally faster than MUM-PHINDER (18 times faster for 1 haplotype of chromosome 19, and 6.5 times faster for 12,500 SARS-CoV2 genomes), it cannot process longer sequences due to memory limitations. Additionally, we observe that when increasing the number of sequences in the dataset, the construction time of MUM-PHINDER increases, while the query time decreases. This phenomenon is due to the increase in the number of matches in the search process, which prevents the use of more computational-demanding operations. Note that, due to the use of the r-index, the efficiency of our method increases when the dataset is highly repetitive as in the case of pangenomes.

The full results on MUM-PHINDER are published in [37].

6.1.1 Additional preliminaries

We recall that, given a text T, the suffix array SA_T is the array consisting of the indices of its suffixes ordered lexicographically. The *Inverse Suffix array* (ISA_T) is the inverse of SA_T, i.e. $\mathrm{ISA}_T[i] = j$ if and only if $\mathrm{SA}_T[j] = i$. The LF-*mapping* is the function that maps every character in the BWT with its preceding text character in the BWT. In other words, it gives a correspondence between characters from the *L*ast and the *F*irst column of the BWT matrix. This mapping is exactly the standard permutation π of $L = \mathrm{BWT}(v)$. Given the i^{th} character in the BWT, we can compute the character preceding it in text order as follows: $\mathrm{LF}(i) = \mathrm{ISA}_T[\mathrm{SA}_T[i] - 1 \bmod n]$.

A *context-free grammar* $\mathcal{G} = \{V, \Sigma, R, S\}$ consists in a set of *variables* V, a set of *terminal symbols* Σ, a set of *rules* R of the type $A \mapsto \alpha$, where $A \in V$ and $\alpha \in \{V \cup \Sigma\}^*$, and the *start variable* $S \in V$. The *language of the grammar* $\mathcal{L}(\mathcal{G}) \subseteq \Sigma^*$ is the set of all words over the alphabet of terminal symbols generated after applying some rules in R starting from S. When $\mathcal{L}(\mathcal{G})$ contains only one string T, that is $\mathcal{G}$ only generates T, then the grammar $\mathcal{G}$ is called *straight-line program* (SLP).

Given a text $T[0..n-1]$, the *longest common extension* (LCE) query between two positions $0 \leq i, j < n$ in T is the length of the longest common prefix of $T[i..n-1]$ and $T[j..n-1]$. Thus, if $\ell = \mathrm{LCE}(i, j)$, then $T[i..i+\ell-1] = T[j..j+\ell-1]$ and either $T[i+\ell] \neq T[j+\ell]$ or either $i+\ell = n$ or $j+\ell = n$.

Given a character c and an integer i, we define $T.\mathrm{rank}_c(i)$ as the number of occurrences of the character c in the prefix $T[0..i-1]$, while we define

$T.\text{select}_c(i)$ as the position $p \in [0..n-1]$ of the ith occurrence of c in T if it exists, and $p = n$ otherwise.

For our purpose, we extend the standard definition of the matching statistics (MS) array as follows.

Definition 22 *Given a text $T = [0..n-1]$ and a pattern $P = [0..m-1]$, we define the extended matching statistics* $\texttt{eMS}$ *as an array of* $(\texttt{pos}, \texttt{len}, \texttt{slen})$*-tuples* $\texttt{eMS}[0..m-1]$ *such that*

i) $P[i..i + \texttt{eMS}[i].\texttt{len} - 1] = T[\texttt{eMS}[i].\texttt{pos}..\texttt{eMS}[i].\texttt{pos} + \texttt{eMS}[i].\texttt{len} - 1]$*;*

ii) either $i = m - \texttt{eMS}[i].\texttt{len}$ *or* $P[i..i + \texttt{eMS}[i].\texttt{len}]$ *does not occur in T.*

iii) $\texttt{eMS}[i].\texttt{slen}$ *is the largest value ℓ for which there exists* $p \neq \texttt{eMS}[i].\texttt{pos}$ *such that* $P[i..i + \ell - 1] = T[p..p + \ell - 1]$*.*

In other words, $\texttt{eMS}[i].\texttt{slen}$ *is the length of the second longest match of a prefix $P[i..n-1]$ of P in T.*

Note that $\texttt{eMS}[i].\texttt{slen} \leq \texttt{eMS}[i].\texttt{len}$, for any $i \in [0..m-1]$. In the following, $\text{SA}_T, \text{ISA}_T$ and LCP_T are referred to the reference text T, thus we omit the subscript and we write only SA, ISA and LCP. We state next some results from [37], for the proofs we refer the reader to the paper.

Proposition 36 (Results from [37]) *Given a text T, a pattern P, and the* $\texttt{eMS}$ *array computed for P with respect to T,*

a let $w = P[i..i+\texttt{eMS}[i].\texttt{len}-1] = T[\texttt{eMS}[i].\texttt{pos}..\texttt{eMS}[i].\texttt{pos}+\texttt{eMS}[i].\texttt{len}-1]$ *be a maximal match between a pattern $P[0..m-1]$ and a text $T[0..n-1]\$$. Then w occurs exactly once in T if and only if* $\texttt{eMS}[i].\texttt{slen} < \texttt{eMS}[i].\texttt{len}$ *(*uniqueness in T*).*

b let $\mathcal{L}$ be the subset of positions in P such that $w_i = P[i..i+\texttt{eMS}[i].\texttt{len}-1]$ *is maximal and occurs only once in T, for all $i \in \mathcal{L}$. Then, w_i occurs only once in P if and only if, for all $i' \in \mathcal{L} \setminus \{i\}$, either* $\texttt{eMS}[i].\texttt{pos} < \texttt{eMS}[i'].\texttt{pos}$ *or* $\texttt{eMS}[i].\texttt{len}+\texttt{eMS}[i].\texttt{pos} > \texttt{eMS}[i'].\texttt{len}+\texttt{eMS}[i'].\texttt{pos}$ *(*uniqueness in P*).*

c let $w = P[i..i + \texttt{eMS}[i].\texttt{len} - 1]$ *be a match with a text T. Then w is a maximal match if and only if either $i = 0$ or* $\texttt{eMS}[i-1].\texttt{len} \leq \texttt{eMS}[i].\texttt{len}$ *(*maximality*).*

d for all $0 \leq i < m$, $w_i = P[i..i + \texttt{eMS}[i].\texttt{len} - 1]$ *is a MUM if and only if b holds and $i \in \mathcal{L}$.*

e let $P[i..i+\texttt{eMS}[i].\texttt{len}-1] = T[\texttt{eMS}[i].\texttt{pos}..\texttt{eMS}[i].\texttt{pos}+\texttt{eMS}[i].\texttt{len}-1]$ *and* $q = \mathit{ISA}[\texttt{eMS}[i].\texttt{pos}]$*. If $q < n$, then* $\texttt{eMS}[i].\texttt{slen} = \max\{\mathit{LCP}[q], \mathit{LCP}[q+1]\}$*, where $\mathit{LCP}[n] = 0$. Otherwise, i.e. $q = n$,* $\texttt{eMS}[i].\texttt{slen} = \mathit{LCP}[n]$ *(*second longest match via LCP*).*

f let LCP and SA be the longest common prefix array, suffix array and inverse suffix array of T, respectively. Then, for all $0 < q \leq n$, let i, j be two integers such that $q-1 = LF[i]$ and $q = LF[j]$, then if $BWT[i] \neq BWT[j]$ then $LCP[q] = 0$, otherwise $LCP[q] = LCE(SA[i], SA[j]) + 1$.

Example 12 *Let T = ACACTCTTAC***ACC***ATATCATCAA\$ be the text and P = AACCTAA the pattern. In the table below, we report the values of the* `eMS` *of P with respect to T.*

i	0	1	2	3	4	5	6
$P[i]$	A	A	C	C	T	A	A
`eMS`$[i]$`.pos`	21	10	11	5	6	21	8
`eMS`$[i]$`.len`	2	3	2	2	2	2	1
`eMS`$[i]$`.slen`	1	2	1	2	2	1	1

It is easy to check that $\mathcal{L} = \{0, 1, 5\}$, where $\mathcal{L}$ contains those indices i which verify both `eMS`$[i]$`.slen` $<$ `eMS`$[i]$`.len` *(Proposition 36b), and either $i = 0$ or* `eMS`$[i-1]$`.len` $\leq$ `eMS`$[i]$`.len` *(Proposition 36c). Note that* `eMS`$[0]$`.pos` $=$ `eMS`$[5]$`.pos` *and* `eMS`$[0]$`.len` $=$ `eMS`$[5]$`.len`*, and thus $P[0..1] = P[5..6]$ is repeated in P (Proposition 36b). Since* `eMS`$[1]$`.pos` $<$ `eMS`$[0]$`.pos` $=$ `eMS`$[5]$`.pos`*, the match $P[1..3] = T[10..12] = ACC$ is a MUM (Proposition 36d).*

6.1.2 Algorithm description

In this section, we present the algorithm for computing MUMs that builds on the approach of Boucher et al. [8] for the computation of the MS array. The authors showed how to use the r-index and the SLP of [28, 29] to compute the MS array of a pattern $P[0..m-1]$ in $\mathcal{O}(m \cdot (t_{\text{LF}} + t_{\text{LCE}} + t_{pred}))$ time, where t_{LF}, t_{LCE}, and t_{pred} represent the time to perform respectively one LF, one LCE, and one predecessor query. Our algorithm extends Boucher et al.'s method by storing additional $\mathcal{O}(r)$ samples of the LCP array. Given a text $T[0..n-1]$ and a pattern $P[0..m-1]$, in the following, we first show how to compute the `eMS` array of P with respect to T using the r-index, the SLP, and the additional LCP array samples. Then we show how to apply Proposition 36d to compute the MUMs from the `eMS` array.

Computing the `eMS` array The key point of the algorithm is to extend the last computed match backward when possible, otherwise, we search for the new longest match that can be extended on the left by using the BWT. Let q be the index such that $P[i..i+$`eMS`$[i]$`.len`$-1] = T[\text{SA}[q]..\text{SA}[q]+$`eMS`$[i]$`.len`$-1]$ is the longest match found at step i:

- if $\text{BWT}[q] = P[i-1]$, then it can be extended on the left, i.e. $P[i-1..i+$`eMS`$[i]$`.len`$-1] = T[\text{SA}[q]-1..\text{SA}[q]+$`eMS`$[i]$`.len`$-1]$;

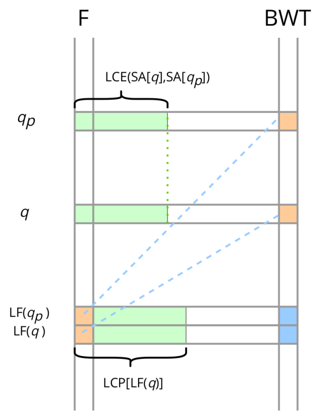


Figure 6.1: Application of Proposition 36f to compute LCP[LF(q)] by extending the result of the last LCE query.

- otherwise, we want to find the longest prefix of $P[i..i + \texttt{eMS}[i].\texttt{len} - 1]$ that is preceded by $P[i-1]$ in the text T. As observed in Bannai et al. [5] it can be either the suffix corresponding to the occurrence of $P[i-1]$ in the BWT immediately preceding or immediately following q, that we refer to as q_p and q_s respectively. Formally, $q_p = \max\{j < q \mid \text{BWT}[j] = P[i-1]\}$ and $q_s = \min\{j > q \mid \text{BWT}[j] = P[i-1]\}$.

The algorithm to compute the `pos` and `len` entry of the `eMS` array is analogous to the procedure detailed in [8]. We use the same data structures as the one defined in [8], that are the run-length encoded BWT and the samples of the SA in correspondence of positions q such that BWT[q] is either the first or the last symbol of a run of the BWT. Note that both q_p and q_s are respectively the last and the first index of their corresponding run.

Analogous reasoning can be formulated to compute the second longest match. Note that the second longest match can be retrieved from the LCP values (Proposition 36e). Once we have the maximal match in position q in the BWT, we can compute LCP[q] and LCP[$q+1$] from the LCE queries on $T[\text{SA}[q]..n-1]$ with $T[\text{SA}[q_p]..n-1]$ and $T[\text{SA}[q_s]..n-1]$ (Proposition 36f).

Moreover, assuming the index q_p is the greatest index smaller than q such that $\text{BWT}[q_p] = \text{BWT}[q]$, then $\text{LF}(q_p) = \text{LF}(q) - 1$. It follows that if $\text{BWT}[\text{LF}(q_p)] = \text{BWT}[\text{LF}(q) - 1] = \text{BWT}[\text{LF}(q)]$, then LCP[LF($q$)]) is an extension of the LCE query computed between SA[q_p] and SA[q] (see Figure 6.1). Symmetrically, if q_s is the smallest index greater than q such that $\text{BWT}[q_s] = \text{BWT}[q]$, then $\text{LF}(q_s) = \text{LF}(q) + 1$. Thus, at each iteration, we keep track of both LCP values computed to find the second longest match.

With respect to the implementation in [8], we add $\mathcal{O}(r)$ sampled values from the LCP array. Precisely, we store the LCP values between the first and the last two suffixes in correspondence of each run (if only one suffix corresponds to a run we simply store 0). As shown later, this allows overcoming the problem of computing the LCE queries in case a position p in T is not stored in the sampled SA, i.e. when ISA[p] is neither the first nor the last index of its equal-letter run.

For simplicity of exposition, we ignore the cases when a select query of a symbol c in the BWT fails. However, whenever it happens, either c does not occur in T or we are attempting to find an occurrence out of the allowed range, that is between 0 and the number of occurrences of the character c minus 1. For the first case, we can simply reset the algorithm starting from the next character of P to process, while the second occurs when we are attempting to compute an LCE query, whose result can be safely set to 0.

Algorithm 2 computes the extended matching statistics `eMS` of the pattern $P = [0..m-1]$ with respect to the text $T = [0..n-1]$ starting from the last element of the pattern (line 2). Moreover, we keep track of the first LCP values with respect to the maximal match of length 1 (line 3).

At each iteration of the loop (line 5), the algorithm tries to extend the match backward position by position. If the match can be extended (line 7), then we use Algorithm 3 to compute the entry of the `eMS`. Otherwise, we use Algorithm 4 to compute the next entry of `eMS` (line 9).

Algorithm 2: COMPUTEEMS($P[0..m-1]$)

```
1  $q \leftarrow \text{BWT.select}_{P[m-1]}(1)$
2  $\texttt{eMS}[m-1] \leftarrow (\texttt{pos} : \text{SA}[q]-1, \texttt{len} : 1, \texttt{slen} : 1)$
3  $lcp_p \leftarrow 0$, $lcp_s \leftarrow 1$
4  $q \leftarrow \text{LF}(q)$
5  for $i \leftarrow m-2$ down to $0$ do
6      if $BWT[q] = P[i]$ then
7          $\texttt{eMS}[i], lcp_p, lcp_s \leftarrow$
             $\text{MSMatch}(P[i], q, \texttt{eMS}[i+1].\texttt{pos}, \texttt{eMS}[i+1].\texttt{pos}, lcp_p, lcp_s)$
8      else
9          $\texttt{eMS}[i], lcp_p, lcp_s \leftarrow$
             $\text{MSMisMatch}(P[i], q, \texttt{eMS}[i+1].\texttt{pos}, \texttt{eMS}[i+1].\texttt{pos}, lcp_p, lcp_s)$
10     $q \leftarrow \text{LF}(q)$
11 return eMS
```

Match case Suppose `eMS`$[i+1..m-1]$ has already been processed and that $P[i] = T[$`eMS`$[i+1]$.`pos`$-1]$, namely we can further extend the longest match at the previous step by one position to the left. Algorithm 3 handles such a scenario.

Let q be such that SA[q] = `eMS`$[i+1]$.`pos` $-$ 1. Hence, we have that `eMS`$[i]$.`pos` = `eMS`$[i+1]$.`pos` $-$ 1 and `eMS`$[i]$.`len` = `eMS`$[i+1]$.`len` + 1 (line 1). At this point, we search for the greatest index q_p among those smaller than q such that BWT[q_p] = $P[i]$. As discussed before, when $q_p = q-1$, then LCP[LF(q)] = LCP[q] + 1 = $lcp_p + 1$ (line 3). Otherwise, we can compute the LCE query between SA[q] and SA[q_p], to which we add 1 for the match with $P[i]$ in correspondence of BWT$[q]$ and BWT$[q_p]$ (line 6). Note that SA$[q]$ = `eMS`$[i+1]$`.pos`, while q_p is the last index of its run (and therefore SA$[q_p]$ is stored).

Analogously, we compute lcp_s (lines 7-10) and, by Proposition 36e and f, we assign to `eMS`$[i]$`.slen` the maximum between lcp_p and lcp_s.

Algorithm 3: MSMatch($P[i], q,$ `eMS`$[i+1]$`.pos`, `eMS`$[i+1]$`.len`$, lcp_p, lcp_s$)

```
1  pos ← eMS[$i+1$].pos − 1, len ← eMS[$i+1$].len + 1
2  $c \leftarrow \text{BWT.rank}_{P[i]}(q)$
3  if $BWT[q-1] = P[i]$ then
       $lcp_p \leftarrow lcp_p + 1$
4  else
5      $q_p \leftarrow \text{BWT.select}_{P[i]}(c)$
6      $lcp_p \leftarrow min(lcp_p, \text{LCE}($eMS[$i+1$].pos$, \text{SA}[q_p])) + 1$
7  if $BWT[q+1] = P[i]$ then
       $lcp_s \leftarrow lcp_s + 1$
8  else
9      $q_s \leftarrow \text{BWT.select}_{P[i]}(c+2)$
10     $lcp_s \leftarrow min(lcp_s, \text{LCE}($eMS[$i+1$].pos$, \text{SA}[q_s])) + 1$
11 slen ← $max(lcp_p, lcp_s)$
12 return (pos, len, slen), $lcp_p, lcp_s$
```

Mismatch case We use Algorithm 4 when q is such that BWT$[q] \neq P[i]$. We search for the index q' in SA such that, among the suffixes of T preceded by $P[i]$, at position SA$[q']$ in T starts the longest match with a prefix of $P[i+1..m-1]$. Note that $T[\text{SA}[q']-1] = P[i]$, and that q' is either q_p or q_s.

Hence, if $q_p = q-1$, then by Proposition 36f the longest common prefix of $T[\text{SA}[q']..n-1]$ and $P[i+1..m-1]$ has length $lcp'_p = lcp_p$ computed at the previous step (line 5), otherwise we compute and store the LCE between $T[q..n-1]$ and $T[q_p..n-1]$ (line 7). A symmetric procedure is used to compute lcp'_s (lines 8-11).

Without loss of generality, we assume that $lcp'_s \geq lcp'_p$, hence `eMS`$[i]$`.pos` = SA$[q_s]-1$. Then `eMS`$[i]$`.len` $= lcp'_s + 1$ and $lcp_p = lcp'_p + 1$ (line 13). We add 1 to both lcp'_s and lcp'_p because both matches can be extended by one position on the left since $P[i]$ = BWT$[q_p]$ = BWT$[q_s]$. To compute `eMS`$[i]$`.slen` we need to compute the value of lcp_s with respect to q_s. To do so, we look for the smallest index q'_s greater than q_s such that BWT$[q'_s] = P[i]$, and then apply a similar procedure to Algorithm 2 (lines 14-18). In this case, if BWT$[q_s+1] = P[i]$, then we can retrieve lcp_s from LCP$[q_s+1]$ since q_s is in correspondence of a run boundary. Symmetrically, we handle the case $lcp'_p > lcp'_s$ (lines 20-26). Finally, we compute `eMS`$[i]$`.slen` by picking the maximum between lcp_p and lcp_s.

Theorem 11 *Given a text $T[0..n-1]$, we can build a data structure in $\mathcal{O}(r+g)$ space that allows to compute the set of MUMs between any pattern $P[0..m-1]$ and T in $\mathcal{O}(m \cdot (t_{LF} + t_{LCE} + t_{pred}))$ time.*

Proof. Algorithm 2, Algorithm 3 and Algorithm 4 show how to compute the `eMS` array in m steps by using the data structure used in [8] of size $\mathcal{O}(r+g)$, to which we add $\mathcal{O}(r)$ words from the LCP array, preserving the space bound. Since at each step the dominant cost depends on the LF, LCE, and rank/select queries, `eMS` is computed in $\mathcal{O}(m(t_{\mathrm{LF}}+t_{\mathrm{LCE}}+t_{pred}))$ time. By Proposition 36a and c, we can build the set $\mathcal{L}$ in $\mathcal{O}(m)$ steps from the `eMS` array. Recall that $\mathcal{L}$ contains those indices $i \in [0..m-1]$ such that $P[i..i+\texttt{eMS}[i].\texttt{len}-1]$ is a maximal match that occurs only once in T.

Now we have to look for those indices in $\mathcal{L}$ that are also unique in P. A simple algorithm is to build both the LCP and ISA array of P, and then check for each $i \in \mathcal{L}$ if both LCP[ISA[i]] and LCP[ISA[i] + 1] (or only LCP[ISA[i]] if ISA[i] $= m$) are smaller than $\texttt{eMS}[i].\texttt{len}$, i.e. the same property that we use to check the uniqueness in T. Both structures can be built in $\mathcal{O}(m)$ time. The overall time is $\mathcal{O}(m(t_{\mathrm{LF}} + t_{\mathrm{LCE}} + t_{pred}) + m + m)$, which is $\mathcal{O}(m(t_{\mathrm{LF}} + t_{\mathrm{LCE}} + t_{pred}))$.

□

6.1.3 Experimental results

We implemented our algorithm for computing MUMs and measured its performances on real biological datasets. We performed the experiments on a desktop computer equipped with 3.4 GHz Intel Core i7-6700 CPU, 8 MiB L3 cache, and 16 GiB of DDR4 main memory. The machine had no other significant CPU tasks running, and only a single thread of execution was used. The OS was Linux (Ubuntu 16.04, 64bit) running kernel 4.4.0. All programs were compiled using `gcc` version 8.1.0 with `-O3 -DNDEBUG -funroll-loops -msse4.2` options. We recorded the runtime and memory usage using the wall clock time, CPU time, and maximum resident set size from `/usr/bin/time`.

Setup We compare our method (MUM-PHINDER) with MUMmer [65] (`mummer`). We tested two versions of `mummer`, v3.27 [46] (`mummer3`) and v4.0 [65] (`mummer4`). We executed `mummer` with the `-mum` flag to compute MUMs that are unique in both the text and the pattern, `-l 1` to report all MUMs of length at least 1, and `-n` to match only A,C,G,and T characters. We setup MUM-PHINDER to produce the same output as `mummer`. We did not test against Mauve [19] because the tool does not directly report MUMs. We also did not consider algorithms that do not produce an index for the text that can be queried with different patterns without reconstructing the index, e.g. the algorithm described in Mäkinen et al. [59, Section 11.1.2]. The experiments that exceeded 16 GB of memory were omitted from further consideration.

Algorithm 4: MSMismatch($P[i], q,$ `eMS`$[i+1]$.`pos`, `eMS`$[i+1]$.`len`, lcp_p, lcp_s)

```
1  $c \leftarrow \mathrm{BWT.rank}_{P[i]}(q)$
2  $q_p \leftarrow \mathrm{BWT.select}_{P[i]}(c)$
3  $q_s \leftarrow \mathrm{BWT.select}_{P[i]}(c+1)$
4  if $q_p = q - 1$ then
5      $lcp'_p \leftarrow lcp_p$
6  else
7      $lcp'_p \leftarrow min(\texttt{eMS}[i+1].\texttt{len}, \mathrm{LCE}(\texttt{eMS}[i+1].\texttt{pos}, \mathrm{SA}[q_p]))$
8  if $q_s = q + 1$ then
9      $lcp'_s \leftarrow lcp_s$
10 else
11     $lcp'_s \leftarrow min(\texttt{eMS}[i+1].\texttt{len}, \mathrm{LCE}(\texttt{eMS}[i+1].\texttt{pos}, \mathrm{SA}[q_s]))$
12 if $lcp'_p \leq lcp'_s$ then
13     $\texttt{pos} \leftarrow \mathrm{SA}[q_s] - 1, \texttt{len} \leftarrow lcp'_s + 1,\ lcp_p \leftarrow lcp'_p + 1$
14     $q'_s \leftarrow \mathrm{BWT.select}_{P[i]}(c+2)$
15     if $q'_s = q_s + 1$ then
16         $lcp_s \leftarrow min(\texttt{len}, \mathrm{LCP}[q_s + 1] + 1)$
17     else
18         $lcp_s \leftarrow min(\texttt{len}, \mathrm{LCE}(\mathrm{SA}[q_s], \mathrm{SA}[q'_s]) + 1)$
19     $q \leftarrow q_s$
20 else
21     $\texttt{pos} \leftarrow \mathrm{SA}[q_p] - 1, \texttt{len} \leftarrow lcp_p,\ lcp_s \leftarrow lcp'_s + 1$
22     $q'_p \leftarrow \mathrm{BWT.select}_{P[i]}(c-1)$
23     if $q'_p = q_p - 1$ then
24         $lcp_p \leftarrow min(\texttt{len}, \mathrm{LCP}[q_p] + 1)$
25     else
26         $lcp_p \leftarrow min(\texttt{len}, \mathrm{LCE}(\mathrm{SA}[q_p], \mathrm{SA}[q'_p]) + 1)$
27     $q \leftarrow q_p$
28 $\texttt{slen} \leftarrow max(lcp_p, lcp_s)$
29 return $(\texttt{pos}, \texttt{len}, \texttt{slen}), lcp_p, lcp_s$
```

No. seqs	n (MB)	n/r
1	59	1.92
2	118	3.79
4	236	7.47
8	473	14.78
16	946	29.19
32	1892	57.63
64	3784	113.49
128	7568	222.23
256	15 136	424.93
512	30 272	771.53

(a) Collections of chromosomes 19.

No. seqs	n (MB)	n/r
1562	46	459.57
3125	93	515.42
6250	186	576.47
12 500	372	622.92
25 000	744	704.73
50 000	1490	848.29
100 000	2983	1060.07
200 000	5965	1146.24
300 000	8947	1218.82

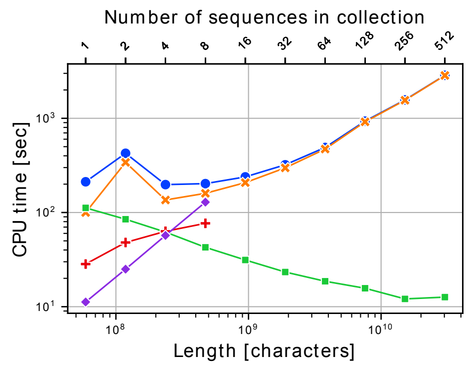
(b) Collections of SARS-CoV2 genomes.

Table 6.1: Dataset used in the experiments. For each collection of datasets of the human chromosome 19 (`chr19`) dataset in Table 6.1a and for the SARSCoV2 (`sars-cov2`) dataset in Table 6.1b, we report the number of sequences (No. seqs), the length n in Megabytes (MB), and the ratio n/r, where r is the number of runs of the BWT for each number of sequences in a collection.

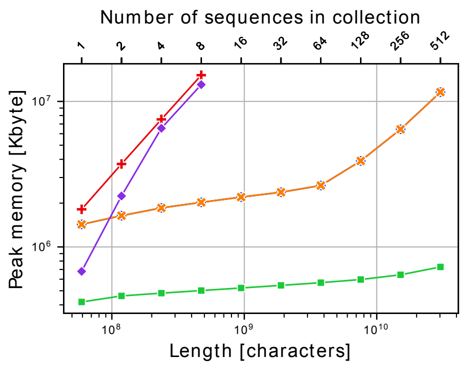
Datasets We evaluated our method using real-world datasets. We build our index for up to 512 haplotypes of human chromosome 19 from the 1000 Genomes Project [84] and up to 300,000 SARS-CoV2 genomes from EBI's COVID data portal [38]. We provide a complete list of accession numbers in the repository. We divide the sequences into 11 collections of 1, 2, 3, 4, 8, 16, 32, 64, 128, 256, 512 chromosomes 19 (`chr19`) and 9 collections of 1,562, 3,125, 6,250, 1250,00, 25,000, 50,000, 100,000, 200,000, 300,000 genomes of SARS-CoV2 (`sars-cov2`). In both datasets, each collection is a superset of the previous one. In Table 6.1 we report the length n of each collection and the ratio n/r, where r is the number of runs of the BWT.

Furthermore, for querying the datasets, we used the first haplotype of chromosome 19 of the sample NA21144 from the 1000 Genomes Project, and the genome with accession number MZ477765 from EBI's COVID data portal [38].

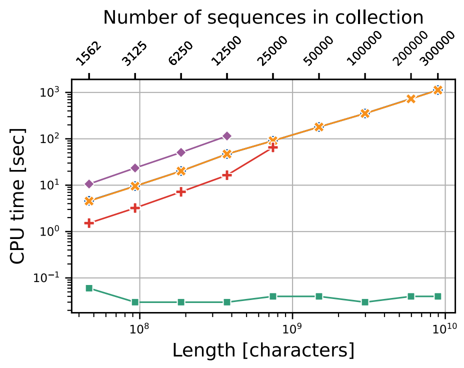
Results In Figure 6.2 we show the construction and query time and space for MUM-PHINDER and `mummer`. Since `mummer` is not able to decouple the construction of the suffix tree from the query, for our method we report the sum of the running times for construction and query, and the maximum resident set size of the two steps. We observe that on `chr19` `mummer3` is up to 9 times faster than MUM-PHINDER, while using up to 8 times more memory, while `mummer4` is up to 19 times faster than MUM-PHINDER, while using up to 7 times more memory. However, both `mummer3` and `mummer4` cannot process more than 8 haplotypes of `chr19` due to memory limitations. MUM-PHINDER was able to build the index and query in 48 minutes for 512 haplotypes of `chr19` while using less than 11.5 GB of RAM. On `sars-cov2`, `mummer3` is up to 6.5 times faster than MUM-PHINDER, while using up to 24 times more memory, while `mummer4` is up to 1.2 times slower than MUM-PHINDER, while using up to 25 times more memory. `mummer3` was not able to process more than 25,000 genomes while `mummer4` were not able to query mote than 12,500 genomes of `sars-cov2` due to memory limitations.



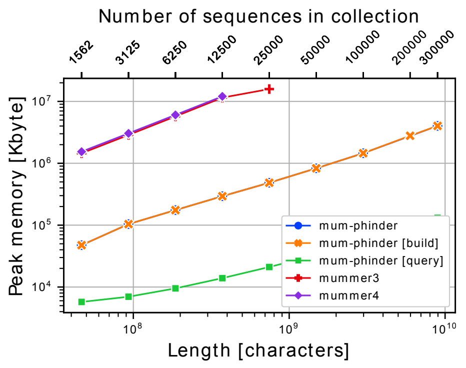
(a) Construction time `chr19`.



(b) Peak memory `chr19`.



(c) Construction time `sars-cov2`.



(d) Peak memory `sars-cov2`.

Figure 6.2: Human chromosome 19 and SARS-CoV2 genomes dataset construction CPU time and peak memory usage. We compare MUM-PHINDER with `mummer3` and `mummer4`. For MUM-PHINDER we report a breakdown of the construction (build) and query time and space. Note that for MUM-PHINDER we consider as time the sum of construction and query time, while for memory we consider the maximum between construction and query memory.

In Figure 6.2 we also show the construction time and space for MUM-PHINDER. We observe that the construction time grows with the number of sequences in the dataset, however, the query time decreases while increasing the number of sequences in the index with a 9x speedup when moving from 1 to 512 haplotypes of `chr19`. A similar phenomenon is observed in [8] and it is attributed to the increased number of match cases (Algorithm 3) while increasing the number of sequences in the index. From our profiling (data not shown) the more time-demanding part of the queries are LCE queries, which are not performed in case of matches. This observation also motivates the increase in the control logic of Algorithm 4 to limit the number of LCE queries to the essential ones.

6.2 Compression of collections of substring samples with LZ77

The extraordinarily wide adoption of high-throughput sequencing in medical and evolutionary biology over the last decade has made short read data sets abundant. These data sets are also very large. For example, a typical human sequencing experiment might run at 20x coverage on an underlying genome of size $n = 3 \cdot 10^9$ nucleotides. The resulting read set in FASTQ format (a standard file format, which stores one ASCII-encoded short read sequence per line) is then 60 gigabytes in size. The de facto standard in most labs and large institutions is to compress such files with the gzip all-purpose file compressor, which usually leads to a factor four reduction in size[1], or 15GB in our human sequencing example. A gzip'd large read set is thus, alas, still relatively large.

With the rapid growth in, and the need to store, short read data sets, specialized compressors that exploit properties inherent to such data sets will become paramount. Several read set specific compressors have now been developed (see, e.g., [2, 14, 22, 39]). None are yet in wide use. However, to our knowledge, no careful analysis of the compressibility of short read data sets—even in an idealized setting such as that described below—has been undertaken. This section addresses that need. In particular, we consider the problem of compressing a set of substrings sampled from a string. (Note that only in this section, strings and permutations are indexed from 1.) Given a string X of length n and two integer parameters d and m, $m \geq 2d$, we call a *sample* of X a substring of X of length m starting at any position $1+i\cdot d$ in X, where $i \geq 0$. We refer to the samples as $S_i = X[(i-1)\cdot d+1..(i-1)\cdot d+m]$, for $i \geq 1$, and to the concatenation as $S = S_1 S_2 \cdots S_r$, where $r = \lfloor (n-m)/d \rfloor + 1$. Note that if $\frac{n-m}{d}$ is not an integer, then the final few characters of X will not be part of any sample.

If X is viewed as a genome sequence, the above substring sampling process corresponds to an idealized model of short read DNA sequencing. In the language of genome sequencing, m is the *read length* and m/d is the so-called *coverage* (the number of samples that cover a given position in X, on average). Our assumption is that $m \geq 2d$ corresponds to a coverage of at least 2, which is the relevant case for DNA sequencing. S represents a file of short read sequences — the typical output of a sequencing experiment. Our sampling

[1]This can be loosely interpreted as gzip, a sliding-window dictionary compressor with window size too small to capture any large dispersed repeated substrings present in the file, essentially reducing the space used by each DNA letter from the 8 bits used in the plain ASCII encoding to the 2 bits that a flat minimal binary code for four letters would use.

process idealizes short read sequencing in at least two ways. Firstly, short read sequencing may produce strings of slightly different lengths and may introduce errors — insertions, deletions, and substitutions of letters — to the sampled strings, albeit with fairly low probability for present-day short read technology ($p < 0.01$ for Illumina short reads, for example). Secondly, in short read sequencing, coverage is not completely uniform, fluctuating across the genome for a variety of reasons (see, e.g. [24]).

We present a non-trivial upper bound on the size of the LZ77 parsing of S, the concatenation of sampled substrings of X, in terms of n, m, d and the size of the LZ parsing of X. We also show a different upper bound that holds regardless of the order in which the samples are concatenated to form S. The contents of this section have been published in [4].

Recall the definition of the Lempel-Ziv 77 factorization.

Definition 23 (LZ Factorization) *The LZ factorization of X is a factorization $X = f_1 f_2 \dots f_{z_X}$ of X into z_X* phrases *such that each phrase f_i (a substring of X) is either*

1. *a letter that does not occur in $f_1 \cdots f_{i-1}$, or*
2. *the longest substring that occurs at least twice in $f_1 \cdots f_i$.*

Example 13 *Consider the example string $X =$ zzzzzipzip. It produces the following $z_X = 5$ factors, $f_1 =$ z, $f_2 =$ zzzz, $f_3 =$ i, $f_4 =$ p, $f_5 =$ zip.*

We denote with $\ell_i = |f_i|$ the length of phrase f_i, and with s_i the starting position of phrase f_i in X, i.e., $s_i = 1 + \sum_{j<i} \ell_j$. Finally, we denote with p_i the position of the first occurrence of substring f_i in X and we call substring $X[p_i..p_i + \ell_i - 1]$ the *source* of f_i. Note that phrases and sources are allowed to overlap, as is the case with f_2 in our example.

The following is a known upper bound on the number of phrases in the parsing.

Theorem 12 ([41], Theorem 5.20) *Let X be a string of length n over Σ, with $|\Sigma| = \sigma$. Then $z_X \leq Z$, where*

$$Z = \frac{n - (\sigma/(\sigma - 1))}{\log_\sigma n - \log_\sigma \log_\sigma n - (1/(\sigma - 1))}.$$

Fact 1 *Let T be a string, and $1 < i \leq j \leq |T|$. If $t = T[i..j]$ has an occurrence before i, i.e., if there exists an $i' < i$ s.t. $T[i'..i' + |t| - 1] = t$, then the interval $[i..j]$ can contain at most one starting position of a phrase.*

6.2.1 Compression of overlapping substring samples

In the following, we formally define the problem of compressing the string S consisting of the concatenation of samples from string X. We first show a bound on the compression size of S when the samples are concatenated in the same order as they appear in X.

Given a string X of length n and two integer parameters m, d, such that $m \geq 2d$, let $S = S_1 S_2 \cdots S_r$, where $r = \lfloor (n-m)/d \rfloor + 1$, and, for $i \in [1..r]$, $S_i = X[(i-1) \cdot d + 1..(i-1) \cdot d + m]$.

Let us write, for $i \geq 1$, $S_i = v_i x_i$, where $|v_i| = m - d$ and $|x_i| = d$. Then X and S can be written as follows, where $|u| < d$:

$$X = v_1 x_1 x_2 \cdots x_r u, \tag{6.1}$$
$$S = v_1 x_1 v_2 x_2 v_3 x_3 \cdots v_r x_r. \tag{6.2}$$

Lemma 18 *Let b_i and e_i denote the beginning resp. ending position of the v_i's in the factorization of S given in* (6.2)*. Then, for $i > 1$, the interval $[b_i..e_i]$ can contain at most one starting position of a phrase.*

Proof. Since two consecutive samples S_i and S_{i+1} overlap by $m - d$ characters, it follows that, for all $i > 1$, $S_i = y_i v_{i+1}$, where y_i is the d-length prefix of S_i. Therefore, S contains the square $v_i v_i$ for every $i > 1$, and b_i is the start of the second occurrence of v_i in this square. By Fact 1, therefore, $[b_i..e_i]$ can contain at most one starting position. □

Definition 24 *Let $1 \leq b \leq e \leq n$. We define the* projection of substring *$X[b..e]$ (which covers the substrings of X from x_i to x_j completely) on S as a collection of substrings in S as follows: For x_ι, let b_ι and e_ι denote the starting respectively ending positions of x_ι in S with respect to the factorization given in* (6.1)*. Then*

$$Proj_S([b..e]) = \begin{cases} (\bigcup_{\iota=i}^{j} [\iota m - d..\iota m]) \cup [(i-2) \cdot (m-d) + b..(i-1)m] \cup \\ [mj + m - d + 1..j(m-d) + e] \qquad \text{if } m - d < b \\ [b..e] \qquad \text{if } b, e \leq m - d \\ [b..m-d] \cup Proj_S[m-d+1..e] \quad \text{if } b \leq m - d < e. \end{cases}$$

Definition 25 *Let f be a phrase of the LZ parsing of X, with starting position s and length ℓ. Define $g(f)$ as the number of phrase starting positions in the projection $Proj_S([s..s+\ell-1])$ of $X[s..s+\ell-1]$ on S.*

Lemma 19 *Let $m \geq 2d$. Let f be a phrase of the LZ factorization of X, with starting position $s > m - d$. Then $g(f) \leq \frac{|f|}{d} + 2$.*

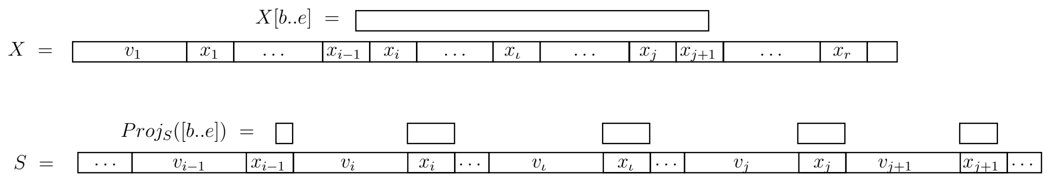


Figure 6.3: The projection of the substring $X[b..e]$ on the string S is shown. The number of substrings of X contained in the collection of substrings produced by the projection is the number of x_i's intersected by $X[b..e]$ in X, and they are all contained in the corresponding x_i's in S.

Proof. Let $\ell = |f|$, $k = \lceil \frac{\ell}{d} \rceil$ and $f = X[s..s+\ell-1] = x' x_i x_{i+1} \cdots x_j x''$, where x' and x'' are a proper suffix of x_{i-1} respectively a proper prefix of x_{j+1}, both possibly empty. We will show that each of these substrings is charged with at most one phrase starting position. From Fact 1, the claim follows.

By construction of S, each of the substrings $x', x_i, x_{i+1}, \ldots, x_j, x''$ will appear contiguously in S. The number of these substrings is either k or $k+1$. Additionally, a v substring separates the projections in S of each pair of contiguous aforementioned substrings of X.

Consider now the source f' of f occurring in X in some position $s' < s$. Then $X[s..s+\ell-1] = X[s'..s'+\ell-1] = u' x_{i'} x_{i'+1} \cdots x_{j'} u''$, where u' and u'' are a proper suffix of $x_{i'-1}$ respectively a proper prefix of $x_{j'+1}$, both possibly empty. As for f, the projection of f' is also split in S in such a way that the projection of each pair of mentioned substrings of f' is separated by a v substring in S.

Notice that, since $m \geq 2d$, each x_b in X has an occurrence in S also as a suffix of $v_{b'+1}$, occurring immediately after x_{b+1} in S. This means that, even if the projections of f and f' in S are split asynchronously, each of the substrings $x', x_i, x_{i+1}, \ldots, x_j, x''$ in the projection of f has already occurred as the concatenation of a suffix of some $v_{b'}$ intersecting the projection of f' and the contiguous $x_{b'+1}$.

There are at most $k+1$ factors $x', x_i, x_{i+1}, \ldots, x_j, x''$, and each of them has a previous occurrence in S. Therefore, by Fact 1, there will be at most $k+1$ phrase starting positions for f. See Fig. 6.4 for an illustration. □

Lemma 20 *With respect to the factorization of S given in* (6.2)*, the number of phrase starting positions in x_i's is at most $\frac{n}{d} + 2z_X$.*

Proof. The sum of the lengths of all phrases $f_1, \ldots, f_{z_X}$ in the LZ factorization of X is the length n of the string X. Therefore, we can bound the total contribution to z_S of X as follows:

$$\sum_{j=1}^{z_X} g(f_j) \leq \sum_{j=1}^{z_X} (\frac{|f_j|}{d} + 2) = \frac{n}{d} + 2z_X.$$

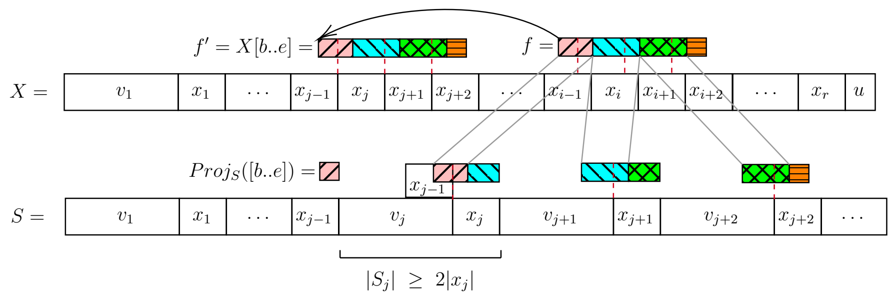


Figure 6.4: The original string X on top, and the concatenation S of the r samples of X below are shown. In particular, a phrase f and its corresponding source f' in X are represented with distinct colors for each of the x_i segments of X intersected by the phrase. Finally, the projection $Proj_S(f')$ of the source is shown. It is clear from the figure that, in S', the samples intersecting some string in the projection of f' fully contain at least one substring of the projection of f.

$\square$

Theorem 13 *Let z_X be the number of phrases of the LZ parsing of X, and z_S the number of phrases of the LZ parsing of S. Then $z_S \leq \frac{2n-m}{d} + 2z_X + m - d$.*

Proof. We show the maximum number of phrase starting positions in S summing up the contribution of the $(m-d)$-length prefix of S, and of the remaining x_i's (Lemma 20) and v_i's (Lemma 18) substrings.

$$\begin{aligned} z_S &= \text{number of starting positions in } v_1 + \text{number of starting positions in } x_i\text{'s} \\ &\quad + \text{number of starting positions in } v_i\text{'s, for } i \geq 2 \\ &\leq (m-d) + \frac{n}{d} + 2z_X + \frac{n-m}{d} = \frac{2n-m}{d} + 2z_X + m - d. \end{aligned}$$

$\square$

Theorem 13 essentially says that the number of LZ phrases of S is at most twice the number of samples plus twice the number of LZ phrases of X. Note that the important parameter that determines the number of samples is d, while m influences only the number of samples in which a given position occurs (i.e. m/d is the so-called coverage).

6.2.2 Arbitrary order of the samples in the concatenation

The question we posed at this point was: does the order of the samples in concatenation matter? We now examine the effect that the ordering of the samples in the concatenation has on the number of phrases.

As before, we are given a string X of length n and two integer parameters m and d, such that $m \geq 2d$. Let $r = \lfloor (n-m)/d \rfloor + 1$, and, for $i \in [1..r]$, $S_i = X[(i-1) \cdot d + 1..(i-1) \cdot d + m]$.

We will show that, regardless of the order in which the samples S_i of X are concatenated, the number of phrases in the LZ factorization of that concatenation is at most $n + \frac{2(n-m)}{d}$. We make this argument precise below.

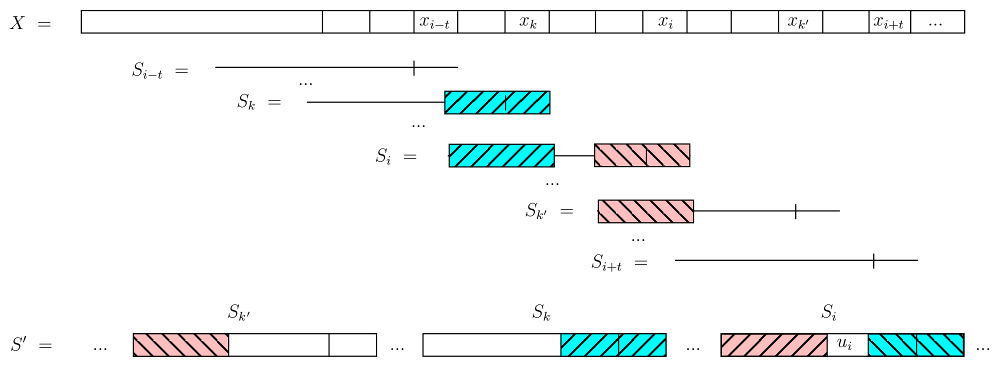


Figure 6.5: The contribution of the samples of X to $z_{S'}$, where S' is the concatenation of the samples in an arbitrary order. The colored substrings are the prefixes and suffixes with multiple occurrences in S' of some sample. Each sample consists of a prefix and a suffix (possibly empty) that has already occurred in S', and a substring in between them (i.e., labeled u_i, in white) that is not assumed to necessarily occur previously in S'.

Theorem 14 *Let S' be the concatenation of the samples of X in any order, then the number of phrases is $z_{S'} \leq n + \frac{2(n-m)}{d}$.*

Proof. Fix i. We will show that the number of phrase starting positions in sample S_i where it appears in S' is at most $2 + |u_i|$, where u_i is a specific substring of S_i. Let $k = \max\{i' < i \mid S_{i'} \text{ appears before } S_i \text{ in } S'\}$, and let w_i be the longest suffix of S_k which is also a prefix of S_i (i.e., w_i is the maximum overlap). Note that w_i may be empty. Similarly, let $k' = \min\{i' > i \mid S_{i'} \text{ appears before } S_i \text{ in } S'\}$, and let w'_i be the longest prefix of $S_{k'}$ which is also a suffix of S_i, again possibly empty. If $|w_i| + |w'_i| \geq m = |S_i|$, then set $u_i = \varepsilon$. Otherwise, S_i can be written as $S_i = w_i u_i w'_i$, for some u_i.

This u_i is a substring of X that has not so far been covered by any sample in S', and this is because of the definition of k and k'. We call u_i the *new part of* S_i; in particular, if $u_i = \varepsilon$, then the new part of S_i is empty.

By Fact 1, if w_i is non-empty, then at most one phrase starting position is contained within w_i. Similarly, if w'_i is non-empty, at most one starting position is contained within w'_i. The number of starting positions within the new part u_i can be trivially upper bounded by $|u_i|$.

Summing over all samples S_i, we thus get

$$z_{S'} \leq \sum_{i=1}^{r}(2 + |u_i|) = 2r + \sum_{i=1}^{r}|u_i| = 2r + n,$$

with the last equality using the fact that every position of X occurs exactly once in some u_i. See Figure 6.5.

$\square$

Theorem 14 essentially says that the number of LZ phrases of S', i.e. of the concatenation of the samples in an arbitrary order, is at most the length of X plus twice the number of samples.

6.2.3 Experimental results

To gauge the tightness of our bounds, we computed the number of phrases in the conventional LZ factorization of the concatenation of the samples taken from each of the texts in Table 6.2. We did this for a range of d and m parameters, and concatenated the samples both in string order and after a random shuffling.

Data Our test data, which contains files of varying repetitiveness is shown in Table 6.2.

Data	Description	n	z	n/z
SARS-CoV2	Taken from the COVID-19 Data Portal [38]	29 835	4373	6.8
50 SARS-CoV2	Concatenation of 50 virus genomes taken from the COVID-19 Data Portal [38]	1 490 134	5421	275
Fibonacci word	Fibonacci word of order 22	28 657	22	1302.6
Random	Word over an alphabet of size 4 built with python function `random.choices()`	29 835	4575	6.5

Table 6.2: Data used in the experiments. The table shows the length (n) and the number of phrases (z) of each text used for the experiments.

By nature, viruses contain very little recurrent genetic heritage, and so a viral genome represents a real-world non-repetitive string. We used a genome of SARS-CoV2 taken from the COVID-19 Data Portal [38] of length 29 836, which is factorized in 4 373 LZ phrases. We also performed experiments on extreme cases between which real-life genomes lie: Fibonacci words and random words.

Let $s_0 = \texttt{b}$, $s_1 = \texttt{a}$, the Fibonacci word of order $i + 1$ is the binary word $s_{i+1} = s_i s_{i-1}$. In other words, the Fibonacci word of order $i + 1$ is the concatenation of the Fibonacci words of the two previous orders.

By construction, Fibonacci words have a very high degree of repetition, resulting in very few phrases with respect to their length. Random words, on the other hand, could be considered, instead, not repetitive at all. To perform a comparison with the virus genome, we chose a random word of the same length as the genome, and the Fibonacci word of length 28 657, the closest to the length of the genome. The strings are factorized in 4 575 respectively 22 phrases.

We also performed experiments with larger data. In particular, we counted the number of phrases of the concatenation of 50 SARS-CoV2 genomes, which, due to the high similarity of the individual genomes, constitutes a highly repetitive dataset.

Experiments We are interested in simulating the coverage provided by the most used technologies in genome sequencing. The coverage is given by the number of samples that cover a given position in the text. In the context of genome sequencing, a reasonable coverage is given by large m and small d, in order to have the coverage as large as possible with reasonable sample lengths.

We perform the experiments on each of the aforementioned texts with $d = 1, 2, 4, 8, 16, 32, 64$, and m from 50 to 1 000, in increments of 10. We counted the number of phrases for $m > d$.

Figure 6.6, on top, displays the number of phrases for concatenations of samples in string order of a Fibonacci word (top-left corner), and of a random string of a similar length (top-right corner). Below, we show the counts for an arbitrarily ordered concatenation of the samples of the same data. In all figures, the number of phrases for some selected m's is shown in distinct colors, in gray for all other m's. The corresponding colored line shows our bounds for each m and d pair. Finally, the colored rounded points show the existing bound given by Kärkkäinen [41], see Theorem 12.

The analogous comparison is shown in Figure 6.7 between a single SARS-CoV2 genome and the concatenation of 50 genomes.

The overlapping of the bound lines in all plots reflects the small impact of the sampling length m in the bound for fixed d.

When the samples are concatenated in string order (on the top of both Figure 6.6 and 6.7), the number of phrases in the original string has a big influence on the bound. We can see that the bound lines in the Fibonacci word's plot are closer to the actual counts than in the other plots, where the number of phrases of the original strings is higher. On the other hand, because of the different nature of the bound for the samples in string order versus random order, we can see that the latter bound has a similar trend for the three strings of similar length. In this case, the length and the number of samples of the original string determine the bound rather than its number of phrases.

Comparing the bound given in this paper (lines) to the one reported in Theorem [41] by Kärkkäinen (rounded points), when repetitive strings are considered, our bound is not worse than the existing one for the string order concatenation, while it gets looser with the concatenation in arbitrary order.

See the plots for Fibonacci words and the concatenation of 50 genomes of SARS-CoV-2. On the other hand, for non repetitive strings, namely random strings and the single viral genome in the plots, the existing bound is often better than ours.

Data	$d = 1$	$d = 2$	$d = 4$	$d = 8$	$d = 16$	$d = 32$	$d = 64$
Fibonacci							
max m	1000	980	990	970	960	960	940
phrases	1336	1321	1308	1021	1007	910	603
bound	57 357	29 189	15 111	8049	4510.1	2733.1	1800.8
Fibonacci shuffled							
max m	1000	990	990	990	630	970	980
phrases	25 164	13 179	6802	3505	1802	942	521
bound	83 971	56 324	42 490.5	35 573.75	32 160.4	30 387.4	29 521.906 25
SARS-CoV2							
max m	80	150	50	70	50	90	210
phrases	44 438	27 823	14 880	8945	6759	5575	4973
bound	68 417	38 655	23 697.5	16 258.25	12 506.38	10 665.93	9821.09
SARS-CoV2 shuffled							
max m	200	260	350	540	780	680	760
phrases	45 725	27 898	17 538	11 620	7969	6142	5238
bound	89 108	59 412	44 574	37 185	33 468	31 658.25	30 744.63

Table 6.3: A summary of relevant results for the Fibonacci word and the single SARS-CoV2 genome. For each sampling frequency d, we report the value of m for which the maximum number of phrases in the concatenation of the samples is produced in both string order and after the random shuffling. We further show the counts and the value of our bound for the mentioned m.

In Table 6.3 we show, for fixed d, the maximum number of phrases in S, varying m. For strings X that are already very repetitive (Fibonacci word), the longer the string, the higher the number of phrases. On the other hand, for strings that are not that repetitive (single SARS-CoV2 genome), introducing long repetitions may decrease the number of phrase starting positions in contrast to having a shorter string with very short repetitions. The concatenation of 50 SARS-CoV2 genomes lies in the middle.

(a) Fibonacci word of length 28 657 **(b)** Random word of length 29 835

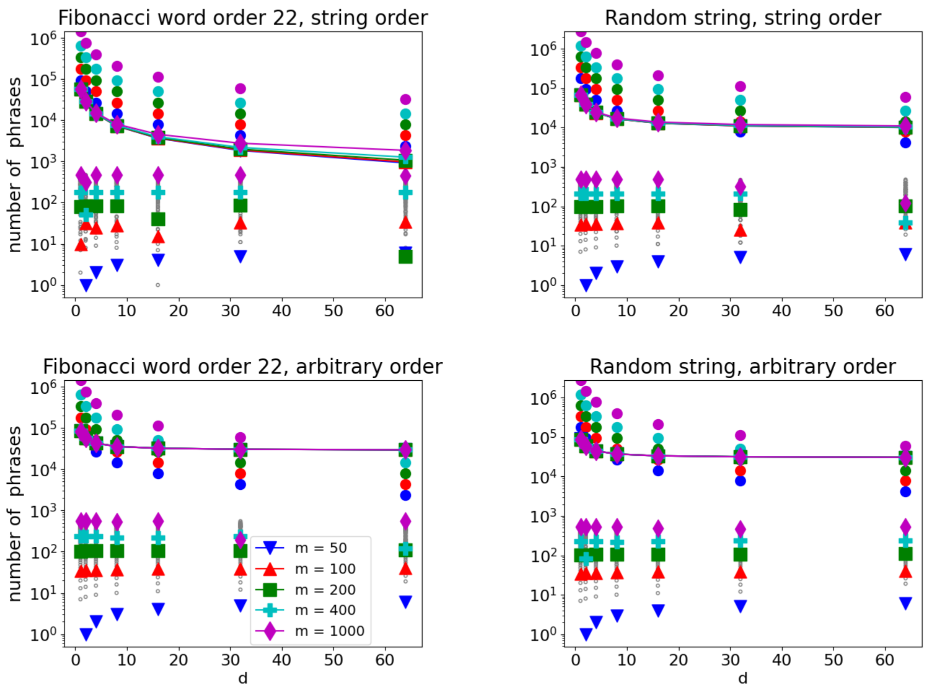


Figure 6.6: The number of phrases in log scale for concatenation of samples of a Fibonacci word (left column) and a random word (right column), both in string order (on top) and in an arbitrarily chosen order (on bottom). The counts are shown for each concatenation of m-length samples at every d position. Colored markers indicate the number of phrases for $m = 50, 100, 200, 400, 1000$, while we use grey points to mark all others m's. The bounds shown in Theorem 13 and 14 for the concatenation in string order respectively random order are shown with colored and shaped lines accordingly to the corresponding m, d pair, while the colored rounded points indicate the bound given by Kärkkäinen [41], see Theorem 12.

(a) SARS-CoV2 genome of length 29 835

(b) Concatenation of 50 SARS-CoV2 genome, of total length 1 490 134

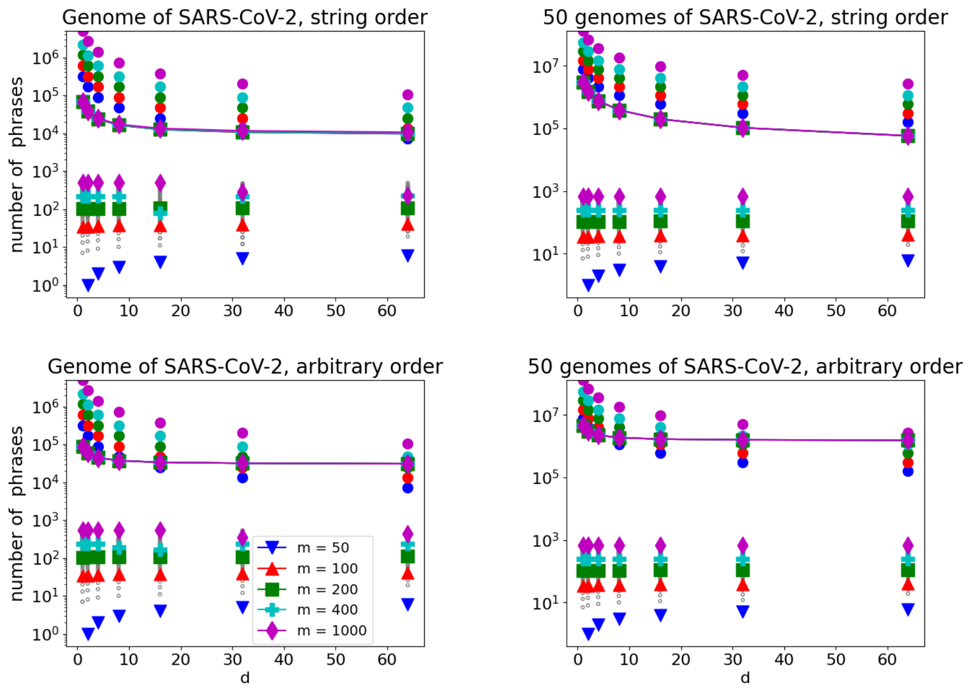


Figure 6.7: The number of phrases in log scale for concatenation of samples of a single SARS-CoV2 genome (left column) and of a concatenation of 50 SARS-CoV2 genomes (right column), both in string order (on top) and in any arbitrarily chosen order (on bottom). The counts are shown for each concatenation of m-length samples at every d position. Coloured markers indicate the number of phrases for $m = 50, 100, 200, 400, 1000$, while we use grey points to mark all others m's. The bounds shown in Theorem 13 and 14 for the concatenation in string order respectively random order are shown with colored and shaped lines accordingly to the corresponding m, d pair, while the colored rounded points indicate the bound given by Kärkkäinen [41], see Theorem 12.

Chapter 7

Conclusion

This thesis focused mainly on the study of combinatorial properties of words concerning their Burrows-Wheeler Transform (BWT), and of words that are the BWT of some other word. It further treated applications of two of the most commonly used compression schemes in bioinformatics, BWT and Lempel-Ziv (LZ77).

Both BWT and LZ77 are widely used in bioinformatics, a field of computer science that tackles problems of understanding and interpreting questions on biological data. In particular, the extremely repetitive and redundant textual data produced by genome sequencing requires algorithms and data structures able to compress and/or index this huge amount of data, to store and use it efficiently.

i) The first research direction we took regards $r(v)$ the number of runs of the BWT of a word v as a measure of its repetitiveness. The repetitiveness of a word is preserved when the word is read back-to-front, i.e. in its reverse. We were interested in the number of runs r of the BWT of a word v and $r(v^{rev})$, the number of runs of the BWT of its reverse. We showed that there exists a family of words for which the ratio $\rho = \frac{r(v)}{r(v^{rev})}$ grows logarithmically with the length of the word. This result suggests that r is not an ideal parameter of the repetitiveness of words because it gives information about the repetitiveness of a word only up to a logarithmic factor.

ii) Another interesting aspect of the BWT is how much it can be affected by small character changes. The so-called *bit-catastrophe* was introduced and studied on the LZ78 factorization of a word, where it was shown that prepending a single character to a word may increase the size of its factorization by a logarithmic factor in the length of the word. It was also shown that well compressible words still show good compressibility after the additional character, and that words which became particularly badly compressible were already poorly compressible before the new character was added. We followed a similar reasoning, treating the bit-catastrophe on the BWT with a looser meaning: we studied the changes in the number of runs in the BWT of a word after a one-character edit operation is performed. We showed that adding, substituting or deleting a character to a Fibonacci word may increase r by $\Theta(\log n)$. Additionally, Fibonacci words are special cases for BWT due to their property of having r constant and being minimal for binary words: it is known that the BWT of all rotations of Fibonacci words has 2 runs. This makes our result even more interesting, showing that, for words characterized by the smallest number of runs possible in their BWT, r may increase by a non-constant factor when even a one-character change occurs. Additionally, we showed that appending a character at the end of a word which is smaller than any other character in the word may cause the bit-catastrophe. Since the BWT is widely used in bioinformatics, where single character errors easily occur and, in particular, where an end-of-string character (smaller than the alphabet characters) is always appended, these results may be relevant for general applications of the BWT in this field.

iii) The relevance of the \$ character in BWT applications directed our attention to characterizing positions where this character can be inserted in a word to make it a BWT of some \$-terminated word over the same alphabet. This is the third direction we explored. We showed that whether and where the \$ can be inserted in a word to turn it into a BWT depends entirely on a specific permutation of the indices of the word, called *standard permutation*. We defined subsets of the elements of the standard permutation of a word called *pseudo-cycles* that determine positions in the word which are "forbidden" for the dollar. Pseudo-cycles allowed us to characterize positions where the dollar can be inserted to make a word the BWT of a \$-terminated word, and we called these positions *nice*. Finally, we studied nice positions and the number of nice positions of a word for *fully clustered words*, which are words consisting of exactly one run for each distinct character of the word, i.e. the best possible instances of BWT words.

Finally, we presented two applications of string compressors in bioinformatics, one for BWT and one for LZ77.

iv) Regarding the BWT, we presented an extension of an already existing algorithm to find Maximal Exact Matches (MEMs), which we modified to compute Maximal Unique Matches (MUMs). MEMs and MUMs are substrings of a pattern P that occur also in a reference text T and that cannot be extended in either direction. MUMs are also unique both in P and in T. These matches between patterns and a reference are used in bioinformatics as prospective anchors for building multiple sequence alignments of genomes. We tested our algorithm on real-world datasets against MUMmer, the first tool that used MUMs to compute multiple sequence alignments. We report that MUM-PHINDER requires consistently less memory than MUMmer in all experiments, being up to 25 times smaller. Although MUMmer is generally faster than MUM-PHINDER (18 times faster for 1 haplotype of chromosome 19, and 6.5 times faster for 12,500 SARS-CoV2 genomes), it cannot process longer sequences due to memory limitations.

v) The application we described for LZ77 is that of compressing DNA short reads. DNA short reads data sets are abundant, very large and repetitive, properties that make these data sets suitable for compression with LZ77. We studied a model of short read data sets where the reads fully cover the genome, from the beginning to the end, and start at a regular distance from one another. In other words, given a text X we extract samples of length m at distance d, and we concatenate the samples in the same order as they appear in X, resulting in a new string S. We evaluated the size of the LZ77 factorization of S, giving an upper bound in terms of the length n and the size of the factorization $z(X)$ of the original text X, and the two parameters m and d. We showed that the number of phrases of S is at most twice the number of samples plus twice the number of phrases of X, and that the important parameter that determines the number of samples is d, while m influences only the number of samples in which a given position occurs. We gave an additional and distinct upper bound on the factorization of the string S' where the samples are concatenated in an arbitrary order. In this case, we showed that the number of phrases of S' is at most the length of X plus twice the number of samples.

7.1 Future work

i) Several open questions remain on the study of the runs-ratio ρ between the number of runs of the BWT of a word and that of its reverse. We saw that Fibonacci-plus words are maximal among the class of standard-plus words with respect to ρ. However, we experimentally saw that they stay strictly below $\rho(n)$, the maximum among all words of length n: it is possible to construct binary words of arbitrary length with greater runs-ratio ρ than any standard-plus word of the same length. However, we currently do not know the asymptotic growth of the ρ value for such words. Therefore, the question of closing the gap for $\rho(n)$ between our lower bound $\Omega(\log n)$ and the upper bound $\mathcal{O}(\log^2 n)$ remains open. It would be interesting also to explore the question for larger alphabets.

ii) We saw that even modifications of just one character on Fibonacci words may affect r by a logarithmic factor, and this holds for all one-character edit operations: insertion, deletion, substitution. This also proves that the upper bound $\mathcal{O}(\log n \log r)$ on the multiplicative sensitivity of r shown in [1] is tight for each edit operation when $r = \mathcal{O}(1)$. It would be interesting to consider other prefixes of infinite words, and study whether words whose r is not constant also reach the same increment in r when a single character is modified, and how the BWT matrix structure changes after this modification.

On the other hand, in [33] our lower bound for the additive sensitivity of r was improved, by showing an infinite family of words in which insertion, deletion and substitution of a character increase r by a $\Theta(\sqrt{n})$ additive factor, where n is the length of the word. However, the tightness of the upper bound $\mathcal{O}(r \log r \log n)$ for the additive sensitivity of r proved in [1] is still an open question.

iii) We characterized nice positions via specific subsets of indices of the word, that is that essential pseudo-cycles in the standard permutation of a word describe which positions allow a \$ to be inserted and turn the word in the BWT of a \$-terminated word. It would be interesting to investigate further the standard permutation, for example, whether and how the number of cycles and pseudo-cycles of the standard permutation is related to the number of nice positions.

Additionally, we extensively studied nice positions in fully clustered words over a binary alphabet. We defined a mapping from fully clustered ternary words with k nice positions to binary words with at least k nice positions. Our preliminary results show that, assuming the phenomenon produced by n_k for binary words is true (Conjecture 4, Chapter 5, page 94), for fixed k after a certain length there is a constant number of ternary words with k nice positions. This is suggested by the fact that among the words involved in the mapping, only those of one specific form maintain the same number of nice positions while increasing in length. We have not yet fully defined the mapping, and a generalization from an alphabet of arbitrary size would be interesting to know whether the aforementioned regularities in binary and ternary fully clustered words are preserved for larger alphabets.

Bibliography

[1] T. Akagi, M. Funakoshi, and S. Inenaga. “Sensitivity of string compressors and repetitiveness measures”. In: *Information and Computation* 291 (2023), p. 104999.

[2] S. Al Yami and C.-H. Huang. “LFastqC: A lossless non-reference-based FASTQ compressor”. In: *PLoS One* 14.11 (2019), e0224806.

[3] S. Aršon. “Proof of the existence on n-valued infinite asymmetric sequences”. In: *Mat. Sb* 2.44 (1937), pp. 769–779.

[4] G. Badkobeh, S. Giuliani, Zs. Lipták, and S. J. Puglisi. “On Compressing Collections of Substring Samples”. In: *Italian Conference on Theoretical Computer Science (ICTCS 2022)*. Vol. 3284. CEUR Workshop Proceedings. CEUR-WS.org, pp. 136–147.

[5] H. Bannai, T. Gagie, and T. I. “Refining the r-index”. In: *Theoretical Computer Science* 812 (2020), pp. 96–108.

[6] J. Berstel and A. de Luca. “Sturmian Words, Lyndon Words and Trees”. In: *Theoretical Computer Science* 178.1-2 (1997), pp. 171–203.

[7] J. Borel and C. Reutenauer. “On Christoffel classes”. In: *RAIRO Theoretical Informatics Application* 40.1 (2006), pp. 15–27.

[8] C. Boucher, T. Gagie, T. I, D. Köppl, B. Langmead, G. Manzini, G. Navarro, A. Pacheco, and M. Rossi. “PHONI: Streamed Matching Statistics with Multi-Genome References”. In: *Data Compression Conference (DCC 2021)*. IEEE, 2021, pp. 193–202.

[9] M. Burrows and D. J. Wheeler. *A block-sorting lossless data compression algorithm.* Tech. rep. DIGITAL System Research Center, 1994.

[10] G. Castiglione, A. Restivo, and M. Sciortino. “Circular Sturmian words and Hopcroft’s algorithm”. In: *Theoretical Computer Science* 410.43 (2009), pp. 4372–4381.

[11] G. Castiglione, A. Restivo, and M. Sciortino. “On extremal cases of Hopcroft’s algorithm”. In: *Theoretical Computer Science* 411.38-39 (2010), pp. 3414–3422.

[12] G. Castiglione, A. Restivo, and M. Sciortino. “Hopcroft’s algorithm and cyclic automata”. In: *International Conference on Language and Automata Theory and Applications (LATA2008).* Vol. 5196. 2008, pp. 172–183.

[13] B. Cazaux and E. Rivals. “Reverse engineering of compact suffix trees and links: A novel algorithm”. In: *J. Discrete Algorithms* 28 (2014), pp. 9–22.

[14] S. Chandak, K. Tatwawadi, I. Ochoa, M. Hernaez, and T. Weissman. “SPRING: a next-generation compressor for FASTQ data”. In: *Bioinformatics* 35.15 (2019), pp. 2674–2676.

[15] J. Clément, M. Crochemore, and G. Rindone. “Reverse Engineering Prefix Tables”. In: *International Symposium on Theoretical Aspects of Computer Science (STACS 2009).* 2009, pp. 289–300.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT press, 2022.

[17] T. M. Cover and J. A. Thomas. *Elements of Information Theory (2. ed.)* Wiley, 2006.

[18] A. C. E. Darling, B. Mau, F. R. Blattner, and N. T. Perna. “Mauve: multiple alignment of conserved genomic sequence with rearrangements”. In: *Genome Res.* 14.7 (2004), pp. 1394–1403.

[19] A. E. Darling, B. Mau, and N. T. Perna. “progressiveMauve: multiple genome alignment with gene gain, loss and rearrangement”. In: *PLoS One* 5.6 (2010), e11147.

[20] J. W. Daykin, F. Franek, J. Holub, A. S. M. S. Islam, and W. F. Smyth. “Reconstructing a string from its Lyndon arrays”. In: *Theoretical Computer Science* 710 (2018), pp. 44–51.

[21] M. Deloger, M. El Karoui, and M.-A. Petit. “A genomic distance based on MUM indicates discontinuity between most bacterial species and genera”. In: *J. Bacteriol.* 191.1 (2009), pp. 91–99.

[22] S. Deorowicz. “FQSqueezer: k-mer-based compression of sequencing data”. In: *Scientific Reports* 10.1 (2020), pp. 1–9.

[23] X. Droubay, J. Justin, and G. Pirillo. “Episturmian words and some constructions of de Luca and Rauzy”. In: *Theoretical Computer Science* 255.1-2 (2001), pp. 539–553.

[24] R. Ekblom, L. Smeds, and H. Ellegren. “Patterns of sequencing coverage bias revealed by ultra-deep sequencing of vertebrate mitochondria”. In: *BMC Genomics* 15.1 (2014), p. 467.

[25] S. Ferenczi and L. Q. Zamboni. “Clustering words and interval exchanges”. In: *Journal of Integer Sequences* 16.2 (2013), p. 3.

[26] P. Ferragina and G. Manzini. "Opportunistic Data Structures with Applications". In: *Annual Symposium on Foundations of Computer Science (FOCS 2000)*. IEEE Computer Society, 2000, pp. 390–398.

[27] A. Frosini, I. Mancini, S. Rinaldi, G. Romana, and M. Sciortino. "Logarithmic Equal-Letter Runs for BWT of Purely Morphic Words". In: *Developments in Language Theory (DLT 2022)*. Vol. 13257. Lecture Notes in Computer Science. Springer, 2022, pp. 139–151.

[28] T. Gagie, T. I, G. Manzini, G. Navarro, H. Sakamoto, L. Seelbach Benkner, and Y. Takabatake. "Practical Random Access to SLP-Compressed Texts". In: *International Symposium on String Processing and Information Retrieval (SPIRE 2020)*. Vol. 12303. Lecture Notes in Computer Science. Springer, 2020, pp. 221–231.

[29] T. Gagie, T. I, G. Manzini, G. Navarro, H. Sakamoto, and Y. Takabatake. "Rpair: Rescaling RePair with Rsync". In: *String Processing and Information Retrieval (SPIRE 2019)*. Vol. 11811. Lecture Notes in Computer Science. Springer, 2019, pp. 35–44.

[30] T. Gagie, G. Navarro, and N. Prezza. "Fully functional suffix trees and optimal text searching in BWT-runs bounded space". In: *Journal of the ACM (JACM)* 67.1 (2020), pp. 1–54.

[31] T. Gagie, G. Navarro, and N. Prezza. "Optimal-Time Text Indexing in BWT-runs Bounded Space". In: *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*. Ed. by A. Czumaj. SIAM, 2018, pp. 1459–1477.

[32] S. Giuliani, S. Inenaga, Zs. Lipták, N. Prezza, M. Sciortino, and A. Toffanello. "Novel Results on the Number of Runs of the Burrows-Wheeler-Transform". In: *47th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2021)*. Vol. 12607. Lecture Notes in Computer Science. Springer, 2021, pp. 249–262.

[33] S. Giuliani, S. Inenaga, Zs. Lipták, G. Romana, M. Sciortino, and C. Urbina. "Bit Catastrophes for the Burrows-Wheeler Transform". In: *Developments in Language Theory (DLT 2023)*. Vol. 13911. Lecture Notes in Computer Science. Springer, 2023, pp. 86–99.

[34] S. Giuliani, Zs. Lipták, and F. Masillo. "When a Dollar in a Fully Clustered Word Makes a BWT". In: *Italian Conference on Theoretical Computer Science (ICTCS 2022)*. Vol. 3284. CEUR Workshop Proceedings. CEUR-WS.org, 2022, pp. 122–135.

[35] S. Giuliani, Zs. Lipták, F. Masillo, and R. Rizzi. "When a dollar makes a BWT". In: *Theoretical Computer Science* 857 (2021), pp. 123–146.

[36] S. Giuliani, Zs. Lipták, and R. Rizzi. "When a Dollar Makes a BWT". In: *Italian Conference on Theoretical Computer Science (ICTCS 2019)*. Vol. 2504. CEUR Workshop Proceedings. 2019, pp. 20–33.

[37] S. Giuliani, G. Romana, and M. Rossi. “Computing Maximal Unique Matches with the r-Index”. In: *20th International Symposium on Experimental Algorithms (SEA 2022)*. Vol. 233. LIPIcs. 2022, 22:1–22:16.

[38] P. W. Harrison, R. Lopez, N. Rahman, S. G. Allen, R. Aslam, N. Buso, C. Cummins, Y. Fathy, E. Felix, et al. “The COVID-19 Data Portal: accelerating SARS-CoV-2 and COVID-19 research through rapid open access data sharing”. In: *Nucleic Acids Research* 49.W1 (2021), W619–W623.

[39] C. Hoobin, T. Kind, C. Boucher, and S. J. Puglisi. “Fast and efficient compression of high-throughput sequencing reads”. In: *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*. 2015, pp. 325–334.

[40] T. I, S. Inenaga, H. Bannai, and M. Takeda. “Inferring strings from suffix trees and links on a binary alphabet”. In: *Discrete Applied Mathematics* 163 (2014), pp. 316–325.

[41] J. Kärkkäinen. “Repetition-Based Text Indexes”. PhD thesis. University of Helsinki, Faculty of Science, Department of Computer Science, Dec. 1999.

[42] J. Kärkkäinen, M. Piatkowski, and S. J. Puglisi. “String Inference from Longest-Common-Prefix Array”. In: *International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. 2017, 62:1–62:14.

[43] D. Kempa and T. Kociumaka. “Resolution of the Burrows-Wheeler transform conjecture”. In: *Commun. ACM* 65.6 (2022), pp. 91–98.

[44] D. Knuth, J. Morris, and V. Pratt. “Fast Pattern Matching in Strings”. In: *SIAM Journal on Computing* 6.2 (1977), pp. 323–350.

[45] S. Kreft and G. Navarro. “On compressing and indexing repetitive sequences”. In: *Theoretical Computer Science* 483 (2013), pp. 115–133.

[46] S. Kurtz, A. Phillippy, A. L. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S. L. Salzberg. “Versatile and open software for comparing large genomes”. en. In: *Genome Biol.* 5.2 (2004), R12.

[47] G. Lagarde and S. Perifel. “Lempel-Ziv: a “one-bit catastrophe” but not a tragedy”. In: *Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1478–1495.

[48] T. W. Lam, R. Li, A. Tam, S. Wong, E. Wu, and S. M. Yiu. “High Throughput Short Read Alignment via Bi-directional BWT”. In: *2009 IEEE International Conference on Bioinformatics and Biomedicine*. 2009, pp. 31–36.

[49] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg. “Ultrafast and memory-efficient alignment of short DNA sequences to the human genome”. In: *Genome Biology* 10.3 (2009), R25.

[50] J. I. Lathrop and M. Strauss. “A universal upper bound on the performance of the Lempel-Ziv algorithm on maliciously-constructed data”. In: *Compression and Complexity of SEQUENCES 1997*. IEEE, 1997, pp. 123–135.

[51] H. Li and R. Durbin. “Fast and accurate long-read alignment with Burrows–Wheeler transform”. In: *Bioinformatics* 26.5 (2010), pp. 589–595.

[52] K. M. Likhomanov and A. M. Shur. “Two Combinatorial Criteria for BWT Images”. In: *International Computer Science Symposium in Russia (CSR 2011)*. 2011, pp. 385–396.

[53] M. López-Valdés. “Lempel-Ziv Dimension for Lempel-Ziv Compression”. In: *International Symposium on Mathematical Foundations of Computer Science 2006 (MFCS 2006)*. Vol. 4162. Lecture Notes in Computer Science. Springer, 2006, pp. 693–703.

[54] M. Lothaire. *Algebraic Combinatorics on Words*. Cambridge University Press, 2002.

[55] M. Lothaire. *Combinatorics on words*. Vol. 17. Cambridge University Press, 1997.

[56] A. de Luca. “Combinatorics of Standard Sturmian Words”. In: *Structures in Logic and Computer Science, A Selection of Essays in Honor of Andrzej Ehrenfeucht*. Vol. 1261. 1997, pp. 249–267.

[57] A. de Luca. “Sturmian Words: Structure, Combinatorics, and Their Arithmetics”. In: *Theoretical Computer Science* 183.1 (1997), pp. 45–82.

[58] A. de Luca and F. Mignosi. “Some combinatorial properties of Sturmian words”. In: *Theoretical Computer Science* 136.2 (1994), pp. 361–385.

[59] V. Mäkinen, D. Belazzougui, F. Cunial, and A. I. Tomescu. *Genome-scale algorithm design*. Cambridge University Press, 2015.

[60] V. Mäkinen and G. Navarro. “Succinct Suffix Arrays based on Run-Length Encoding”. In: *Nordic Journal of Computing* 12.1 (2005), pp. 40–66.

[61] V. Mäkinen and G. Navarro. “Succinct Suffix Arrays Based on Run-Length Encoding”. In: *Annual Symposium on Combinatorial Pattern Matching (CPM 2005)*. Vol. 3537. Lecture Notes in Computer Science. Springer, 2005, pp. 45–56.

[62] S. Mantaci, A. Restivo, G. Rosone, M. Sciortino, and L. Versari. “Measuring the clustering effect of BWT via RLE”. In: *Theoretical Computer Science* 698 (2017), pp. 79 –87.

[63] S. Mantaci, A. Restivo, G. Rosone, F. Russo, and M. Sciortino. “On Fixed Points of the Burrows-Wheeler Transform”. In: *Fundamenta Informaticae* 154.1-4 (2017), pp. 277–288.

[64] S. Mantaci, A. Restivo, and M. Sciortino. “Burrows–Wheeler transform and Sturmian words”. In: *Information Processing Letters* 86.5 (2003), pp. 241–246.

[65] G. Marçais, A. L. Delcher, A. M. Phillippy, R. Coston, S. L. Salzberg, and A. Zimin. “MUMmer4: A fast and versatile genome alignment system”. In: *PLoS Comput. Biol.* 14.1 (2018), e1005944.

[66] H. M. Morse. “Recurrent geodesics on a surface of negative curvature”. In: *Transactions of the American Mathematical Society* 22.1 (1921), pp. 84–100.

[67] M. Morse. “A solution of the problem of infinite play in chess”. In: *Bull. Amer. Math. Soc* 44 (1938), p. 632.

[68] M. Morse and G. A. Hedlund. “Symbolic dynamics”. In: *American Journal of Mathematics* 60.4 (1938), pp. 815–866.

[69] G. Navarro. “Indexing highly repetitive string collections, part I: repetitiveness measures”. In: *ACM Computing Surveys (CSUR)* 54.2 (2021), pp. 1–31.

[70] T. Nishimoto and Y. Tabei. “LZRR: LZ77 parsing with right reference”. In: *Information and Computation* 285 (2022), p. 104859.

[71] S. Nurk, S. Koren, A. Rhie, M. Rautiainen, A. V. Bzikadze, A. Mikheenko, M. R. Vollger, N. Altemose, L. Uralsky, et al. “The complete sequence of a human genome”. In: *bioRxiv* (2021).

[72] I. Pavlov. “7z format”. URL: `https://7-zip.org/7z.html`.

[73] A. Restivo and G. Rosone. “Balancing and clustering of words in the Burrows–Wheeler transform”. In: *Theoretical Computer Science* 412.27 (2011), pp. 3019–3032.

[74] A. Restivo and G. Rosone. “Burrows-Wheeler transform and palindromic richness”. In: *Theoretical Computer Sciece* 410.30-32 (2009), pp. 3018–3026.

[75] M. Rossi, M. Oliva, B. Langmead, T. Gagie, and C. Boucher. “MONI: A Pangenomic Index for Finding Maximal Exact Matches”. In: *J. Comput. Biol.* (Jan. 2022).

[76] L. M. Russo, A. D. Correia, G. Navarro, and A. P. Francisco. “Approximating optimal bidirectional macro schemes”. In: *2020 Data Compression Conference (DCC)*. IEEE. 2020, pp. 153–162.

[77] M. Sciortino and L. Q. Zamboni. “Suffix Automata and Standard Sturmian Words”. In: *Developments in Language Theory (DLT 2007)*. Vol. 4588. Lecture Notes in Computer Science. Springer, 2007, pp. 382–398.

[78] J. Seward. https://sourceware.org/bzip2/manual/manual.html. 1996. URL: `"https://sourceware.org/bzip2/manual/manual.html"`.

[79] C. E. Shannon. “A mathematical theory of communication”. In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423.

[80] J. Simpson and S. J. Puglisi. “Words with Simple Burrows-Wheeler Transforms”. In: *Electronic Journal of Combinatorics* 15.1 (2008).

[81] D. D. Sleator and R. E. Tarjan. “Self-adjusting binary search trees”. In: *Journal of the ACM (JACM)* 32.3 (1985), pp. 652–686.

[82] T. A. Starikovskaya and H. W. Vildhøj. “A suffix tree or not a suffix tree?” In: *J. Discrete Algorithms* 32 (2015), pp. 14–23.

[83] J. A. Storer and T. G. Szymanski. “Data compression via textual substitution”. In: *Journal of the ACM (JACM)* 29.4 (1982), pp. 928–951.

[84] The 1000 Genomes Project Consortium. “A global reference for human genetic variation”. In: *Nature* (2015), pp. 68–74.

[85] T. A. Welch. “A technique for high-performance data compression”. In: *Computer* 17.06 (1984), pp. 8–19.

[86] K. Zhu, W. Robinson, A. A. Schäffer, J. Xu, E. Ruppin, A. Funda Ergun, Y. Ye, and S. Cenk Sahinalp. “Strain Level Microbial Detection and Quantification with Applications to Single Cell Metagenomics”. In: *bioRxiv* (2020), p. 149245.

[87] J. Ziv and A. Lempel. “A universal algorithm for sequential data compression”. In: *IEEE Transactions on Information Theory* 23.3 (1977), pp. 337–343.

[88] J. Ziv and A. Lempel. “Compression of individual sequences via variable-rate coding”. In: *IEEE Transactions on Information Theory* 24.5 (1978), pp. 530–536.

Appendix A

Appendix chapter 3

n	runs	r(reverse)	ρ	highest no. of palindromic factors
45	2, 5, 8, 11, 13, 6	22	3.67	29
45	2, 5, 10, 15, 8, 5	22	3.67	26
45	5, 8, 15, 10, 5, 2	22	3.67	26
45	6, 13, 11, 8, 5, 2	22	3.67	29
45	7, 3, 4, 13, 11, 7	22	3.67	33
45	7, 11, 13, 4, 3, 7	22	3.67	33
47	2, 3, 13, 22, 5, 2	22	3.67	28
47	2, 3, 22, 13, 5, 2	22	3.67	28
47	2, 5, 13, 22, 3, 2	22	3.67	28
47	2, 5, 22, 13, 3, 2	22	3.67	28
47	2, 7, 8, 11, 13, 6	22	3.67	30
47	2, 7, 10, 15, 8, 5	22	3.67	27
47	3, 6, 8, 13, 12, 5	22	3.67	23
47	4, 3, 8, 15, 12, 5	22	3.67	24
47	5, 8, 15, 10, 7, 2	22	3.67	27
47	5, 12, 13, 8, 6, 3	22	3.67	23
47	5, 12, 15, 8, 3, 4	22	3.67	24
47	6, 7, 22, 12	16	4	22
47	6, 9, 12, 3, 10, 7	22	3.67	26
47	6, 13, 11, 8, 7, 2	22	3.67	30
47	6, 19, 10, 12	16	4	20
47	7, 10, 3, 12, 9, 6	22	3.67	26
47	7, 16, 13, 11	16	4	36
47	11, 13, 16, 7	16	4	36
47	12, 10, 19, 6	16	4	20
47	12, 22, 7, 6	16	4	22
53	1, 4, 6, 16, 12, 14	22	3.67	23
53	1, 4, 23, 11, 10, 4	22	3.67	36
53	2, 3, 10, 7, 16, 15	22	3.67	35
53	2, 3, 13, 22, 11, 2	22	3.67	31
53	2, 3, 14, 23, 8, 3	22	3.67	27
53	2, 3, 22, 13, 11, 2	22	3.67	31
53	2, 5, 13, 22, 9, 2	24	4	31
53	2, 5, 22, 13, 9, 2	24	4	31
53	2, 5, 24, 5, 11, 6	22	3.67	34
53	2, 7, 8, 7, 11, 18	22	3.67	29

n	runs	r(reverse)	ρ	highest no. of palindromic factors
53	2, 7, 8, 11, 19, 6	22	3.67	33
53	2, 7, 12, 17, 9, 6	22	3.67	27
53	2, 7, 12, 19, 8, 5	22	3.67	28
53	2, 7, 13, 22, 7, 2	24	4	31
53	2, 7, 22, 13, 7, 2	24	4	31
53	2, 9, 5, 12, 15, 10	22	3.67	42
53	2, 9, 7, 4, 17, 14	22	3.67	38
53	2, 9, 10, 13, 11, 8	22	3.67	33
53	2, 9, 12, 15, 5, 10	22	3.67	36
53	2, 9, 13, 16, 5, 8	22	3.67	31
53	2, 9, 13, 22, 5, 2	24	4	31
53	2, 9, 22, 13, 5, 2	24	4	31
53	2, 11, 13, 22, 3, 2	22	3.67	31
53	2, 11, 22, 13, 3, 2	22	3.67	31
53	2, 13, 8, 11, 13, 6	22	3.67	33
53	2, 13, 10, 15, 8, 5	22	3.67	30
53	3, 4, 22, 13, 8, 3	22	3.67	36
53	3, 6, 9, 14, 15, 6	22	3.67	29
53	3, 7, 12, 18, 6, 7	22	3.67	26
53	3, 8, 13, 22, 4, 3	22	3.67	36
53	3, 8, 23, 14, 3, 2	22	3.67	27
53	3, 12, 8, 13, 12, 5	22	3.67	25
53	4, 3, 9, 16, 15, 6	22	3.67	30
53	4, 7, 17, 8, 13, 4	22	3.67	34
53	4, 8, 9, 15, 12, 5	22	3.67	24
53	4, 9, 8, 17, 11, 4	22	3.67	34
53	4, 10, 11, 23, 4, 1	22	3.67	36
53	4, 11, 17, 8, 9, 4	22	3.67	34
53	4, 13, 8, 17, 7, 4	22	3.67	34
53	5, 8, 15, 10, 13, 2	22	3.67	30
53	5, 8, 17, 9, 8, 6	22	3.67	27
53	5, 8, 19, 12, 7, 2	22	3.67	28
53	5, 11, 22, 15	16	4	30
53	5, 12, 13, 8, 12, 3	22	3.67	25
53	5, 12, 15, 9, 8, 4	22	3.67	24
53	5, 12, 19, 8, 3, 6	22	3.67	30
53	5, 13, 8, 15, 3, 9	22	3.67	37
53	6, 3, 8, 19, 12, 5	22	3.67	30
53	6, 8, 3, 20, 10, 6	22	3.67	37
53	6, 8, 9, 17, 8, 5	22	3.67	27
53	6, 9, 17, 12, 7, 2	22	3.67	27
53	6, 10, 17, 3, 6, 11	22	3.67	25
53	6, 10, 20, 3, 8, 6	22	3.67	37
53	6, 11, 5, 24, 5, 2	22	3.67	34
53	6, 13, 11, 8, 13, 2	22	3.67	33
53	6, 13, 22, 12	18	4.5	24
53	6, 15, 12, 3, 10, 7	22	3.67	29
53	6, 15, 14, 9, 6, 3	22	3.67	29
53	6, 15, 16, 9, 3, 4	22	3.67	30
53	6, 19, 11, 8, 7, 2	22	3.67	33
53	6, 25, 10, 12	16	4	22
53	7, 3, 1, 14, 15, 13	22	3.67	37

n	runs	r(reverse)	ρ	highest no. of palindromic factors
53	7, 6, 18, 12, 7, 3	22	3.67	26
53	7, 9, 4, 14, 12, 7	22	3.67	40
53	7, 9, 14, 4, 12, 7	22	3.67	26
53	7, 10, 3, 12, 15, 6	22	3.67	29
53	7, 12, 4, 14, 9, 7	22	3.67	26
53	7, 12, 14, 4, 9, 7	22	3.67	40
53	7, 13, 18, 15	16	4	19
53	7, 16, 17, 13	16	4	39
53	8, 3, 5, 16, 13, 8	22	3.67	41
53	8, 5, 16, 13, 9, 2	22	3.67	31
53	8, 9, 6, 2, 16, 12	22	3.67	26
53	8, 9, 8, 2, 14, 12	22	3.67	29
53	8, 11, 9, 10, 5, 10	22	3.67	33
53	8, 11, 13, 10, 9, 2	22	3.67	33
53	8, 13, 16, 5, 3, 8	22	3.67	41
53	8, 14, 16, 15	16	4	19
53	9, 3, 15, 8, 13, 5	22	3.67	37
53	9, 19, 10, 15	16	4	22
53	10, 5, 10, 9, 11, 8	22	3.67	33
53	10, 5, 15, 12, 9, 2	22	3.67	36
53	10, 15, 12, 5, 9, 2	22	3.67	42
53	11, 6, 3, 17, 10, 6	22	3.67	25
53	12, 10, 25, 6	16	4	22
53	12, 14, 2, 8, 9, 8	22	3.67	29
53	12, 16, 2, 6, 9, 8	22	3.67	26
53	12, 22, 13, 6	18	4.5	24
53	13, 15, 14, 1, 3, 7	22	3.67	37
53	13, 17, 16, 7	16	4	39
53	14, 12, 16, 6, 4, 1	22	3.67	23
53	14, 17, 4, 7, 9, 2	22	3.67	38
53	15, 10, 19, 9	16	4	22
53	15, 16, 7, 10, 3, 2	22	3.67	35
53	15, 16, 14, 8	16	4	19
53	15, 18, 13, 7	16	4	19
53	15, 22, 11, 5	16	4	30
53	18, 11, 7, 8, 7, 2	22	3.67	29
55	6, 11, 26, 12	18	4.5	25
55	12, 26, 11, 6	18	4.5	25
68	2, 5, 14, 21, 19, 7	26	4.33	35
68	2, 7, 13, 19, 19, 8	26	4.33	40
68	7, 13, 33, 15	18	4.5	25
68	7, 19, 21, 14, 5, 2	26	4.33	35
68	8, 14, 31, 15	18	4.5	25
68	8, 19, 19, 13, 7, 2	26	4.33	40
68	9, 19, 25, 15	18	4.5	29
68	12, 16, 5, 3, 20, 12	26	4.33	36
68	12, 20, 3, 5, 16, 12	26	4.33	36
68	15, 25, 19, 9	18	4.5	29
68	15, 31, 14, 8	18	4.5	25
68	15, 33, 13, 7	18	4.5	25
78	2, 5, 10, 7, 37, 17	26	4.33	50
78	2, 5, 15, 21, 25, 10	26	4.33	45

n	runs	r(reverse)	ρ	highest no. of palindromic factors
78	2, 5, 17, 25, 21, 8	26	4.33	35
78	2, 5, 31, 13, 19, 8	26	4.33	46
78	2, 7, 7, 31, 17, 14	26	4.33	52
78	2, 9, 13, 19, 27, 8	26	4.33	45
78	2, 9, 21, 27, 13, 6	26	4.33	33
78	2, 15, 14, 21, 19, 7	26	4.33	40
78	2, 17, 13, 19, 19, 8	26	4.33	45
78	3, 7, 13, 10, 24, 21	26	4.33	39
78	4, 7, 16, 27, 17, 7	26	4.33	28
78	4, 7, 32, 17, 13, 5	26	4.33	24
78	4, 8, 29, 15, 17, 5	26	4.33	24
78	5, 9, 19, 33, 4, 8	26	4.33	36
78	5, 13, 17, 32, 7, 4	26	4.33	24
78	5, 17, 15, 29, 8, 4	26	4.33	24
78	6, 8, 37, 11, 10, 6	26	4.33	36
78	6, 10, 11, 37, 8, 6	26	4.33	36
78	6, 13, 27, 21, 9, 2	26	4.33	33
78	7, 5, 15, 26, 15, 10	26	4.33	22
78	7, 11, 27, 13, 8, 12	26	4.33	26
78	7, 16, 36, 19	18	4.5	27
78	7, 17, 27, 16, 7, 4	26	4.33	28
78	7, 19, 21, 14, 15, 2	26	4.33	40
78	8, 4, 33, 19, 9, 5	26	4.33	36
78	8, 12, 9, 18, 20, 11	26	4.33	36
78	8, 19, 13, 31, 5, 2	26	4.33	46
78	8, 19, 19, 13, 17, 2	26	4.33	45
78	8, 21, 25, 17, 5, 2	26	4.33	35
78	8, 27, 19, 13, 9, 2	26	4.33	45
78	9, 11, 9, 14, 20, 15	26	4.33	34
78	9, 14, 20, 5, 19, 11	26	4.33	36
78	9, 14, 22, 5, 16, 12	26	4.33	43
78	9, 16, 36, 17	18	4.5	25
78	10, 15, 26, 15, 5, 7	26	4.33	22
78	10, 16, 5, 17, 19, 11	26	4.33	39
78	10, 22, 29, 17	18	4.5	29
78	10, 25, 21, 15, 5, 2	26	4.33	45
78	11, 17, 17, 3, 18, 12	26	4.33	34
78	11, 19, 5, 20, 14, 9	26	4.33	36
78	11, 19, 17, 5, 16, 10	26	4.33	39
78	11, 20, 18, 9, 12, 8	26	4.33	36
78	12, 8, 13, 27, 11, 7	26	4.33	26
78	12, 16, 5, 22, 14, 9	26	4.33	43
78	12, 18, 3, 17, 17, 11	26	4.33	34
78	12, 22, 13, 31	18	4.5	35
78	12, 26, 11, 29	18	4.5	35
78	14, 17, 31, 7, 7, 2	26	4.33	52
78	15, 10, 5, 10, 21, 17	26	4.33	53
78	15, 20, 14, 9, 11, 9	26	4.33	34
78	17, 21, 10, 5, 10, 15	26	4.33	53
78	17, 29, 22, 10	18	4.5	29
78	17, 36, 16, 9	18	4.5	25
78	17, 37, 7, 10, 5, 2	26	4.33	50

n	runs	r(reverse)	ρ	highest no. of palindromic factors
78	19, 36, 16, 7	18	4.5	27
78	21, 24, 10, 13, 7, 3	26	4.33	39
78	29, 11, 26, 12	18	4.5	35
78	31, 13, 22, 12	18	4.5	35
89	2, 5, 20, 27, 25, 10	28	4.67	34
89	2, 9, 22, 19, 20, 17	28	4.67	40
89	2, 11, 19, 28, 21, 8	28	4.67	32
89	4, 7, 41, 24, 9, 4	28	4.67	29
89	4, 8, 18, 30, 21, 8	28	4.67	26
89	4, 9, 24, 41, 7, 4	28	4.67	29
89	6, 4, 14, 24, 32, 9	28	4.67	26
89	8, 21, 28, 19, 11, 2	28	4.67	32
89	8, 21, 30, 18, 8, 4	28	4.67	26
89	9, 4, 14, 30, 23, 9	28	4.67	27
89	9, 23, 30, 14, 4, 9	28	4.67	27
89	9, 32, 24, 14, 4, 6	28	4.67	26
89	10, 25, 27, 20, 5, 2	28	4.67	34
89	12, 15, 6, 23, 21, 12	28	4.67	44
89	12, 21, 23, 6, 15, 12	28	4.67	44
89	12, 26, 28, 23	20	5	25
89	15, 17, 4, 10, 28, 15	28	4.67	47
89	15, 28, 10, 4, 17, 15	28	4.67	47
89	16, 17, 6, 10, 24, 16	28	4.67	27
89	16, 24, 10, 6, 17, 16	28	4.67	27
89	17, 20, 19, 22, 9, 2	28	4.67	40
89	23, 28, 26, 12	20	5	25
90	2, 5, 20, 17, 31, 15	28	4.67	39
90	2, 5, 22, 31, 19, 11	28	4.67	32
90	2, 5, 25, 31, 19, 8	28	4.67	31
90	4, 7, 18, 29, 23, 9	28	4.67	34
90	6, 10, 11, 37, 20, 6	28	4.67	38
90	6, 14, 37, 11, 16, 6	28	4.67	40
90	6, 16, 11, 37, 14, 6	28	4.67	40
90	6, 20, 37, 11, 10, 6	28	4.67	38
90	7, 10, 13, 42, 11, 7	28	4.67	35
90	7, 11, 42, 13, 10, 7	28	4.67	35
90	8, 19, 31, 25, 5, 2	28	4.67	31
90	9, 23, 29, 18, 7, 4	28	4.67	34
90	11, 18, 24, 6, 18, 13	28	4.67	40
90	11, 19, 31, 22, 5, 2	28	4.67	32
90	13, 18, 6, 24, 18, 11	28	4.67	40
90	13, 28, 10, 9, 11, 19	28	4.67	38
90	15, 31, 17, 20, 5, 2	28	4.67	39
90	18, 13, 5, 13, 23, 18	28	4.67	53
90	18, 23, 13, 5, 13, 18	28	4.67	53
90	19, 11, 9, 10, 28, 13	28	4.67	38
93	2, 5, 40, 19, 19, 8	28	4.67	51
93	2, 7, 19, 28, 29, 8	28	4.67	32
93	2, 9, 20, 27, 25, 10	28	4.67	36
93	2, 13, 22, 19, 20, 17	28	4.67	42
93	2, 15, 19, 28, 21, 8	28	4.67	34
93	4, 7, 41, 24, 13, 4	28	4.67	30

n	runs	r(reverse)	ρ	highest no. of palindromic factors
93	4, 9, 24, 41, 11, 4	28	4.67	30
93	4, 11, 41, 24, 9, 4	28	4.67	30
93	4, 12, 18, 30, 21, 8	28	4.67	27
93	4, 13, 24, 41, 7, 4	28	4.67	30
93	6, 10, 13, 41, 16, 7	28	4.67	37
93	6, 10, 16, 26, 25, 10	28	4.67	25
93	6, 10, 37, 20, 14, 6	28	4.67	24
93	6, 14, 20, 37, 10, 6	28	4.67	24
93	6, 14, 42, 12, 12, 7	28	4.67	46
93	7, 12, 12, 42, 14, 6	28	4.67	46
93	7, 16, 41, 13, 10, 6	28	4.67	37
93	8, 15, 37, 4, 17, 12	28	4.67	44
93	8, 19, 19, 40, 5, 2	28	4.67	51
93	8, 21, 28, 19, 15, 2	28	4.67	34
93	8, 21, 30, 18, 12, 4	28	4.67	27
93	8, 29, 28, 19, 7, 2	28	4.67	32
93	9, 11, 12, 17, 26, 18	28	4.67	33
93	9, 13, 20, 35, 4, 12	28	4.67	37
93	9, 14, 12, 16, 23, 19	28	4.67	33
93	10, 16, 27, 5, 20, 15	28	4.67	39
93	10, 25, 26, 16, 10, 6	28	4.67	25
93	10, 25, 27, 20, 9, 2	28	4.67	36
93	12, 4, 35, 20, 13, 9	28	4.67	37
93	12, 8, 18, 27, 16, 12	28	4.67	20
93	12, 16, 27, 18, 8, 12	28	4.67	20
93	12, 17, 4, 37, 15, 8	28	4.67	44
93	12, 20, 24, 5, 18, 14	30	5	39
93	14, 18, 5, 24, 20, 12	30	5	39
93	14, 30, 22, 27	20	5	18
93	15, 20, 5, 27, 16, 10	28	4.67	39
93	17, 20, 19, 22, 13, 2	28	4.67	42
93	18, 26, 17, 12, 11, 9	28	4.67	33
93	19, 23, 16, 12, 14, 9	28	4.67	33
93	27, 22, 30, 14	20	5	18
95	2, 5, 24, 21, 24, 19	28	4.67	43
95	2, 7, 40, 19, 19, 8	28	4.67	52
95	2, 9, 19, 28, 29, 8	28	4.67	33
95	2, 11, 20, 27, 25, 10	28	4.67	37
95	2, 15, 22, 19, 20, 17	28	4.67	43
95	2, 17, 19, 28, 21, 8	28	4.67	35
95	3, 7, 20, 31, 25, 9	28	4.67	29
95	5, 8, 19, 34, 21, 8	28	4.67	28
95	5, 13, 32, 15, 21, 9	28	4.67	49
95	6, 10, 14, 24, 32, 9	28	4.67	27
95	6, 10, 18, 20, 28, 13	28	4.67	22
95	6, 10, 23, 40, 6, 10	28	4.67	36
95	6, 10, 38, 21, 14, 6	28	4.67	24
95	6, 13, 20, 37, 11, 8	28	4.67	27
95	6, 14, 21, 38, 10, 6	28	4.67	24
95	7, 5, 17, 28, 27, 11	28	4.67	24
95	7, 12, 15, 26, 25, 10	28	4.67	25
95	8, 11, 37, 20, 13, 6	28	4.67	27

n	runs	r(reverse)	ρ	highest no. of palindromic factors
95	8, 12, 9, 35, 20, 11	28	4.67	44
95	8, 19, 19, 40, 7, 2	28	4.67	52
95	8, 21, 28, 19, 17, 2	28	4.67	35
95	8, 21, 34, 19, 8, 5	28	4.67	28
95	8, 29, 28, 19, 9, 2	28	4.67	33
95	9, 13, 18, 29, 16, 10	28	4.67	27
95	9, 14, 7, 30, 15, 20	28	4.67	31
95	9, 21, 15, 32, 13, 5	28	4.67	49
95	9, 25, 31, 20, 7, 3	28	4.67	29
95	9, 32, 24, 14, 10, 6	28	4.67	27
95	10, 6, 40, 23, 10, 6	28	4.67	36
95	10, 16, 29, 18, 13, 9	28	4.67	27
95	10, 25, 26, 15, 12, 7	28	4.67	25
95	10, 25, 27, 20, 11, 2	28	4.67	37
95	10, 26, 23, 5, 18, 13	28	4.67	39
95	11, 20, 35, 9, 12, 8	28	4.67	44
95	11, 27, 28, 17, 5, 7	28	4.67	24
95	12, 16, 7, 22, 24, 14	28	4.67	40
95	12, 16, 8, 23, 22, 14	28	4.67	35
95	12, 20, 25, 7, 18, 13	28	4.67	45
95	13, 18, 5, 23, 26, 10	28	4.67	39
95	13, 18, 7, 25, 20, 12	28	4.67	45
95	13, 28, 20, 18, 10, 6	28	4.67	22
95	14, 19, 12, 4, 26, 20	28	4.67	27
95	14, 20, 13, 4, 24, 20	28	4.67	35
95	14, 22, 23, 8, 16, 12	28	4.67	35
95	14, 24, 22, 7, 16, 12	28	4.67	40
95	14, 30, 23, 28	20	5	18
95	16, 8, 9, 22, 24, 16	28	4.67	25
95	16, 20, 10, 7, 24, 18	28	4.67	30
95	16, 24, 22, 9, 8, 16	28	4.67	25
95	17, 20, 19, 22, 15, 2	28	4.67	43
95	17, 20, 24, 10, 5, 19	28	4.67	42
95	18, 19, 12, 7, 20, 19	28	4.67	36
95	18, 24, 7, 10, 20, 16	28	4.67	30
95	18, 34, 19, 24	20	5	34
95	19, 5, 10, 24, 20, 17	28	4.67	42
95	19, 20, 7, 12, 19, 18	28	4.67	36
95	19, 24, 21, 24, 5, 2	28	4.67	43
95	20, 15, 30, 7, 14, 9	28	4.67	31
95	20, 24, 4, 13, 20, 14	28	4.67	35
95	20, 26, 4, 12, 19, 14	28	4.67	27
95	24, 19, 34, 18	20	5	34
95	28, 23, 30, 14	20	5	18
106	2, 5, 15, 21, 29, 34	28	4.67	21
106	2, 5, 21, 41, 25, 12	28	4.67	29
106	2, 5, 25, 31, 35, 8	28	4.67	33
106	2, 5, 25, 33, 29, 12	28	4.67	34
106	2, 5, 29, 35, 25, 10	28	4.67	31
106	2, 5, 39, 13, 35, 12	28	4.67	59
106	2, 7, 19, 49, 21, 8	28	4.67	38
106	2, 7, 45, 13, 17, 22	28	4.67	29

n	runs	r(reverse)	ρ	highest no. of palindromic factors
106	2, 9, 22, 19, 37, 17	30	5	47
106	2, 9, 25, 31, 25, 14	28	4.67	28
106	2, 9, 43, 13, 25, 14	28	4.67	59
106	2, 11, 24, 21, 17, 31	28	4.67	40
106	2, 13, 25, 31, 27, 8	28	4.67	36
106	2, 13, 38, 13, 25, 15	28	4.67	46
106	2, 15, 42, 21, 19, 7	28	4.67	50
106	2, 17, 39, 13, 23, 12	28	4.67	59
106	2, 21, 20, 17, 31, 15	28	4.67	47
106	2, 21, 22, 31, 19, 11	28	4.67	40
106	2, 21, 25, 31, 19, 8	28	4.67	39
106	3, 14, 22, 34, 24, 9	28	4.67	31
106	4, 6, 45, 25, 20, 6	28	4.67	38
106	4, 8, 29, 48, 12, 5	28	4.67	27
106	4, 9, 44, 17, 21, 11	28	4.67	34
106	4, 11, 20, 31, 29, 11	28	4.67	41
106	4, 23, 18, 29, 23, 9	28	4.67	38
106	5, 9, 42, 16, 23, 11	28	4.67	29
106	5, 11, 19, 34, 29, 8	28	4.67	31
106	5, 11, 22, 36, 21, 11	28	4.67	27
106	5, 12, 37, 16, 25, 11	28	4.67	49
106	5, 12, 48, 29, 8, 4	28	4.67	27
106	5, 13, 17, 14, 30, 27	28	4.67	41
106	5, 13, 21, 36, 22, 9	28	4.67	29
106	5, 13, 45, 17, 14, 12	28	4.67	28
106	5, 14, 22, 36, 18, 11	28	4.67	32
106	5, 15, 20, 34, 23, 9	28	4.67	27
106	5, 23, 18, 32, 20, 8	28	4.67	31
106	6, 4, 27, 37, 23, 9	28	4.67	24
106	6, 8, 21, 37, 24, 10	28	4.67	40
106	6, 9, 22, 39, 21, 9	28	4.67	28
106	6, 10, 14, 44, 23, 9	30	5	32
106	6, 10, 18, 44, 15, 13	28	4.67	26
106	6, 10, 21, 22, 32, 15	28	4.67	22
106	6, 10, 37, 19, 18, 16	28	4.67	24
106	6, 14, 8, 26, 33, 19	28	4.67	31
106	6, 14, 20, 36, 21, 9	28	4.67	35
106	6, 16, 13, 41, 23, 7	28	4.67	41
106	6, 16, 18, 28, 27, 11	28	4.67	26
106	6, 19, 19, 35, 19, 8	28	4.67	30
106	6, 20, 19, 35, 18, 8	28	4.67	30
106	6, 20, 25, 45, 6, 4	28	4.67	38
106	6, 20, 42, 12, 19, 7	28	4.67	51
106	7, 5, 17, 28, 38, 11	28	4.67	26
106	7, 5, 18, 29, 34, 13	28	4.67	30
106	7, 5, 20, 32, 30, 12	28	4.67	24
106	7, 5, 24, 44, 16, 10	28	4.67	25
106	7, 9, 23, 42, 15, 10	28	4.67	26
106	7, 10, 18, 31, 29, 11	28	4.67	25
106	7, 11, 10, 30, 18, 30	28	4.67	35
106	7, 11, 10, 30, 28, 20	28	4.67	41
106	7, 11, 13, 44, 24, 7	28	4.67	38

n	runs	r(reverse)	ρ	highest no. of palindromic factors
106	7, 11, 17, 28, 31, 12	28	4.67	31
106	7, 11, 22, 42, 17, 7	28	4.67	24
106	7, 11, 41, 21, 18, 8	28	4.67	24
106	7, 11, 47, 14, 19, 8	28	4.67	47
106	7, 11, 52, 15, 12, 9	28	4.67	51
106	7, 12, 18, 30, 28, 11	28	4.67	25
106	7, 12, 20, 23, 30, 14	28	4.67	22
106	7, 17, 15, 46, 13, 8	28	4.67	38
106	7, 17, 42, 22, 11, 7	28	4.67	24
106	7, 17, 44, 13, 18, 7	28	4.67	40
106	7, 18, 13, 44, 17, 7	28	4.67	40
106	7, 19, 12, 42, 20, 6	28	4.67	51
106	7, 19, 16, 28, 26, 10	28	4.67	26
106	7, 19, 21, 42, 15, 2	28	4.67	50
106	7, 23, 41, 13, 16, 6	28	4.67	41
106	7, 24, 44, 13, 11, 7	28	4.67	38
106	8, 10, 50, 14, 15, 9	28	4.67	36
106	8, 12, 9, 35, 31, 11	28	4.67	49
106	8, 13, 46, 15, 17, 7	28	4.67	38
106	8, 14, 38, 14, 17, 15	28	4.67	25
106	8, 14, 45, 5, 20, 14	28	4.67	43
106	8, 18, 21, 41, 11, 7	28	4.67	24
106	8, 18, 35, 19, 20, 6	28	4.67	30
106	8, 19, 14, 47, 11, 7	28	4.67	47
106	8, 19, 31, 25, 21, 2	28	4.67	39
106	8, 19, 35, 19, 19, 6	28	4.67	30
106	8, 20, 6, 26, 29, 17	28	4.67	40
106	8, 20, 32, 18, 23, 5	28	4.67	31
106	8, 21, 49, 19, 7, 2	28	4.67	38
106	8, 27, 31, 25, 13, 2	28	4.67	36
106	8, 29, 34, 19, 11, 5	28	4.67	31
106	8, 35, 31, 25, 5, 2	28	4.67	33
106	9, 12, 15, 52, 11, 7	28	4.67	51
106	9, 13, 16, 27, 32, 9	28	4.67	29
106	9, 13, 22, 27, 20, 15	28	4.67	20
106	9, 13, 50, 14, 8, 12	28	4.67	31
106	9, 14, 27, 16, 31, 9	28	4.67	33
106	9, 14, 32, 5, 29, 17	28	4.67	40
106	9, 14, 36, 19, 16, 12	28	4.67	45
106	9, 15, 14, 50, 10, 8	28	4.67	36
106	9, 15, 16, 29, 26, 11	28	4.67	25
106	9, 15, 25, 38, 5, 14	28	4.67	50
106	9, 20, 12, 17, 19, 29	28	4.67	20
106	9, 21, 36, 20, 14, 6	28	4.67	35
106	9, 21, 39, 22, 9, 6	28	4.67	28
106	9, 22, 16, 17, 10, 32	28	4.67	37
106	9, 22, 16, 27, 23, 9	28	4.67	31
106	9, 22, 36, 21, 13, 5	28	4.67	29
106	9, 23, 27, 16, 22, 9	28	4.67	31
106	9, 23, 29, 18, 23, 4	28	4.67	38
106	9, 23, 34, 20, 15, 5	28	4.67	27
106	9, 23, 37, 27, 4, 6	28	4.67	24

n	runs	r(reverse)	ρ	highest no. of palindromic factors
106	9, 23, 44, 14, 10, 6	30	5	32
106	9, 24, 34, 22, 14, 3	28	4.67	31
106	9, 31, 16, 27, 14, 9	28	4.67	33
106	9, 32, 27, 16, 13, 9	28	4.67	29
106	10, 15, 42, 23, 9, 7	28	4.67	26
106	10, 16, 44, 24, 5, 7	28	4.67	25
106	10, 24, 37, 21, 8, 6	28	4.67	40
106	10, 25, 35, 29, 5, 2	28	4.67	31
106	10, 25, 38, 15, 5, 13	28	4.67	30
106	10, 26, 28, 16, 19, 7	28	4.67	26
106	11, 15, 11, 24, 28, 17	28	4.67	36
106	11, 17, 12, 24, 27, 15	28	4.67	36
106	11, 18, 24, 5, 34, 14	28	4.67	35
106	11, 18, 32, 6, 22, 17	28	4.67	48
106	11, 18, 36, 22, 14, 5	28	4.67	32
106	11, 19, 31, 22, 21, 2	28	4.67	40
106	11, 21, 17, 44, 9, 4	28	4.67	34
106	11, 21, 36, 22, 11, 5	28	4.67	27
106	11, 23, 16, 42, 9, 5	28	4.67	29
106	11, 25, 16, 37, 12, 5	28	4.67	49
106	11, 26, 29, 16, 15, 9	28	4.67	25
106	11, 27, 28, 18, 16, 6	28	4.67	26
106	11, 28, 30, 18, 12, 7	28	4.67	25
106	11, 29, 31, 18, 10, 7	28	4.67	25
106	11, 29, 31, 20, 11, 4	28	4.67	41
106	11, 31, 35, 9, 12, 8	28	4.67	49
106	11, 38, 28, 17, 5, 7	28	4.67	26
106	12, 8, 14, 50, 13, 9	28	4.67	31
106	12, 14, 17, 45, 13, 5	28	4.67	28
106	12, 16, 10, 24, 27, 17	28	4.67	31
106	12, 16, 12, 20, 27, 19	30	5	33
106	12, 16, 12, 26, 25, 15	28	4.67	41
106	12, 16, 19, 36, 14, 9	28	4.67	45
106	12, 20, 21, 5, 18, 30	28	4.67	41
106	12, 20, 35, 5, 11, 23	28	4.67	32
106	12, 23, 13, 39, 17, 2	28	4.67	59
106	12, 25, 41, 21, 5, 2	28	4.67	29
106	12, 26, 33, 8, 10, 17	28	4.67	27
106	12, 29, 33, 25, 5, 2	28	4.67	34
106	12, 30, 32, 20, 5, 7	28	4.67	24
106	12, 31, 28, 17, 11, 7	28	4.67	31
106	12, 35, 13, 39, 5, 2	28	4.67	59
106	13, 5, 15, 38, 25, 10	28	4.67	30
106	13, 15, 10, 20, 29, 19	28	4.67	39
106	13, 15, 44, 18, 10, 6	28	4.67	26
106	13, 17, 8, 26, 27, 15	28	4.67	45
106	13, 17, 34, 10, 10, 22	28	4.67	28
106	13, 19, 9, 18, 28, 19	28	4.67	48
106	13, 20, 28, 8, 22, 15	28	4.67	40
106	13, 34, 29, 18, 5, 7	28	4.67	30
106	14, 5, 38, 25, 15, 9	28	4.67	50
106	14, 16, 49, 27	20	5	22

n	runs	r(reverse)	ρ	highest no. of palindromic factors
106	14, 20, 5, 45, 14, 8	28	4.67	43
106	14, 22, 35, 8, 8, 19	28	4.67	41
106	14, 25, 13, 43, 9, 2	28	4.67	59
106	14, 25, 31, 25, 9, 2	28	4.67	28
106	14, 30, 23, 20, 12, 7	28	4.67	22
106	14, 34, 5, 24, 18, 11	28	4.67	35
106	15, 8, 7, 30, 17, 29	28	4.67	45
106	15, 10, 19, 10, 21, 31	28	4.67	55
106	15, 12, 10, 21, 33, 15	28	4.67	26
106	15, 17, 14, 38, 14, 8	28	4.67	25
106	15, 17, 26, 8, 25, 15	28	4.67	28
106	15, 20, 14, 4, 30, 23	28	4.67	27
106	15, 20, 23, 12, 19, 17	28	4.67	26
106	15, 20, 27, 22, 13, 9	28	4.67	20
106	15, 21, 15, 4, 28, 23	28	4.67	31
106	15, 22, 6, 23, 24, 16	28	4.67	34
106	15, 22, 8, 28, 20, 13	28	4.67	40
106	15, 23, 34, 8, 7, 19	30	5	40
106	15, 25, 8, 26, 17, 15	28	4.67	28
106	15, 25, 13, 38, 13, 2	28	4.67	46
106	15, 25, 26, 12, 16, 12	28	4.67	41
106	15, 25, 34, 8, 6, 18	28	4.67	42
106	15, 27, 24, 12, 17, 11	28	4.67	36
106	15, 27, 26, 8, 17, 13	28	4.67	45
106	15, 31, 17, 20, 21, 2	28	4.67	47
106	15, 31, 24, 10, 7, 19	28	4.67	46
106	15, 32, 22, 21, 10, 6	28	4.67	22
106	15, 33, 21, 10, 12, 15	28	4.67	26
106	15, 35, 31, 25	20	5	39
106	16, 18, 19, 37, 10, 6	28	4.67	24
106	16, 24, 23, 6, 22, 15	28	4.67	34
106	17, 3, 10, 22, 37, 17	28	4.67	46
106	17, 10, 3, 30, 19, 27	28	4.67	45
106	17, 10, 8, 33, 26, 12	28	4.67	27
106	17, 19, 7, 8, 22, 33	28	4.67	26
106	17, 19, 8, 9, 32, 21	28	4.67	27
106	17, 19, 12, 23, 20, 15	28	4.67	26
106	17, 21, 12, 7, 28, 21	28	4.67	30
106	17, 22, 6, 32, 18, 11	28	4.67	48
106	17, 26, 14, 7, 22, 20	28	4.67	27
106	17, 26, 16, 5, 21, 21	28	4.67	34
106	17, 27, 24, 10, 16, 12	28	4.67	31
106	17, 28, 24, 11, 15, 11	28	4.67	36
106	17, 29, 5, 32, 14, 9	28	4.67	40
106	17, 29, 26, 6, 20, 8	28	4.67	40
106	17, 36, 10, 9, 11, 23	28	4.67	23
106	17, 37, 19, 22, 9, 2	30	5	47
106	17, 37, 22, 10, 3, 17	28	4.67	46
106	17, 39, 26, 24	20	5	36
106	18, 6, 8, 34, 25, 15	28	4.67	42
106	18, 15, 9, 15, 27, 22	28	4.67	56
106	18, 20, 13, 13, 16, 26	28	4.67	43

n	runs	r(reverse)	ρ	highest no. of palindromic factors
106	18, 22, 9, 5, 32, 20	28	4.67	36
106	19, 7, 8, 34, 23, 15	30	5	40
106	19, 7, 10, 24, 31, 15	28	4.67	46
106	19, 8, 8, 35, 22, 14	28	4.67	41
106	19, 13, 9, 14, 28, 23	28	4.67	63
106	19, 27, 20, 12, 16, 12	30	5	33
106	19, 28, 18, 9, 19, 13	28	4.67	48
106	19, 29, 20, 10, 15, 13	28	4.67	39
106	19, 33, 26, 8, 14, 6	28	4.67	31
106	20, 22, 7, 14, 26, 17	28	4.67	27
106	20, 28, 30, 10, 11, 7	28	4.67	41
106	20, 32, 5, 9, 22, 18	28	4.67	36
106	21, 21, 5, 16, 26, 17	28	4.67	34
106	21, 28, 7, 12, 21, 17	28	4.67	30
106	21, 32, 9, 8, 19, 17	28	4.67	27
106	22, 10, 10, 34, 17, 13	28	4.67	28
106	22, 17, 5, 13, 27, 22	28	4.67	61
106	22, 17, 13, 45, 7, 2	28	4.67	29
106	22, 26, 15, 7, 8, 28	28	4.67	24
106	22, 27, 13, 5, 17, 22	28	4.67	61
106	22, 27, 15, 9, 15, 18	28	4.67	56
106	23, 11, 5, 35, 20, 12	28	4.67	32
106	23, 11, 9, 10, 36, 17	28	4.67	23
106	23, 28, 4, 15, 21, 15	28	4.67	31
106	23, 28, 14, 9, 13, 19	28	4.67	63
106	23, 30, 4, 14, 20, 15	28	4.67	27
106	24, 26, 39, 17	20	5	36
106	25, 31, 35, 15	20	5	39
106	26, 16, 13, 13, 20, 18	28	4.67	43
106	27, 19, 30, 3, 10, 17	28	4.67	45
106	27, 30, 14, 17, 13, 5	28	4.67	41
106	27, 49, 16, 14	20	5	22
106	28, 8, 7, 15, 26, 22	28	4.67	24
106	29, 17, 30, 7, 8, 15	28	4.67	45
106	29, 19, 17, 12, 20, 9	28	4.67	20
106	30, 18, 5, 21, 20, 12	28	4.67	41
106	30, 18, 30, 10, 11, 7	28	4.67	35
106	31, 17, 21, 24, 11, 2	28	4.67	40
106	31, 21, 10, 19, 10, 15	28	4.67	55
106	32, 10, 17, 16, 22, 9	28	4.67	37
106	33, 22, 8, 7, 19, 17	28	4.67	26
106	34, 29, 21, 15, 5, 2	28	4.67	21

Table A.2: Examples of words with higher ρ than standard-plus words.

n	p	q	t	ρ
4	1	1	1	1.0
9	2	4	1	2.0
13	3	6	1	2.0
17	4	7	2	2.5
23	5	10	3	3.0
33	6	13	8	3.5
39	7	16	9	3.5
39	9	16	5	3.5
45	8	19	10	3.5
47	10	23	4	3.5
53	11	28	3	3.5
55	12	24	7	4.0
65	15	28	7	4.5
68	13	21	21	4.0
77	14	30	19	4.5
77	18	31	10	4.5
78	17	39	5	4.0
89	16	36	21	4.5
90	19	45	7	4.5
93	21	46	5	4.5
93	22	39	10	4.5
95	20	48	7	4.5
106	23	53	7	4.5
107	28	39	12	5.0
118	25	59	9	5.0
119	32	40	15	5.0
123	24	43	32	5.0
124	29	51	15	5.0
131	30	55	16	5.0
139	26	46	41	5.0
140	27	51	35	5.0
143	33	64	13	5.0
159	34	73	18	5.5
160	35	69	21	5.5
170	31	67	41	5.5
179	38	90	13	5.0
182	39	83	21	5.5
185	43	78	21	5.5
185	44	76	21	5.5
191	40	96	15	5.5
197	36	77	48	5.5
197	42	91	22	6.0
199	46	85	22	5.5
206	37	85	47	5.5
207	48	88	23	5.5
210	45	103	17	5.0
221	50	105	16	5.5
226	41	89	55	6.0
227	54	102	17	5.5
232	53	99	27	6.0
233	49	118	17	5.5
236	63	81	29	6.0
239	51	110	27	6.0
245	52	120	21	5.5
255	68	88	31	6.0
256	59	105	33	6.0
259	60	111	28	5.5
260	61	117	21	5.5
262	47	107	61	6.0
262	55	131	21	5.5
263	56	122	29	6.0
266	57	121	31	6.0
269	58	118	35	6.0
271	62	115	32	6.0
288	75	115	23	6.0
299	67	150	15	5.5
303	64	151	24	6.0
304	65	139	35	6.0
305	69	136	31	6.0
308	81	111	35	6.0
310	71	131	37	6.0
311	66	145	34	6.0
329	70	153	36	6.0
329	86	136	21	6.0
335	72	151	40	6.0
338	89	121	39	6.0
340	77	147	39	6.0
344	73	171	27	6.0
349	74	174	27	6.0
363	76	171	40	6.0
363	82	174	25	6.5
363	95	134	39	6.0
365	78	169	40	6.5
369	79	168	43	6.5
376	97	159	23	6.0
377	87	162	41	6.5
380	101	133	45	6.5
383	80	180	43	6.5
387	84	193	26	6.0
387	90	163	44	6.5
391	104	136	47	7.0
397	83	186	45	6.5
398	103	147	45	6.5
399	92	172	43	6.5
401	107	136	51	6.5
403	93	174	43	6.5
404	85	189	45	6.5
413	88	191	46	6.5
413	106	156	45	6.5
417	99	172	47	6.5
425	98	184	45	6.5
428	91	195	51	6.5
429	94	184	57	6.5
431	96	176	63	6.5
437	102	181	52	6.5
446	105	175	61	6.5
465	108	197	52	6.5
465	111	188	55	6.5
469	109	198	53	6.5
471	110	190	61	6.5
476	113	197	53	6.5
479	112	193	62	6.5
493	114	202	63	6.5

Table A.1

Appendix B

Appendix chapter 5

$\|\boldsymbol{\Sigma}\| = \mathbf{2}$	$h(w)$	all	%	noBWTs	%	BWTs	%	BWTs of prim	BWTs of pow
$n = 3$	0	1	13	1	25	0	0	0	0
	1	6	75	3	75	3	75	1	2
	2	1	13	0	0	1	25	1	0
	total	8	100	4	100	4	100	2	2
$n = 4$	0	3	19	3	30	0	0	0	0
	1	10	63	6	60	4	67	2	2
	2	3	19	1	10	2	33	1	1
	total	16	100	10	100	6	100	3	3
$n = 5$	0	9	28	9	38	0	0	0	0
	1	16	50	11	46	5	63	3	2
	2	5	16	4	17	1	13	1	0
	3	2	6	0	0	2	25	2	0
	total	32	100	24	100	8	100	6	2
$n = 6$	0	21	33	21	42	0	0	0	0
	1	28	44	20	40	8	57	6	2
	2	9	14	6	12	3	21	1	2
	3	6	9	3	6	3	21	2	1
	total	64	100	50	100	14	100	9	5
$n = 7$	0	51	40	51	47	0	0	0	0
	1	46	36	35	32	11	55	9	2
	2	14	11	13	12	1	5	1	0
	3	14	11	9	8	5	25	5	0
	4	3	2	0	0	3	15	3	0
	total	128	100	108	100	20	100	18	2
$n = 8$	0	111	43	111	50	0	0	0	0
	1	84	33	64	29	20	56	18	2
	2	20	8	19	9	1	3	1	0
	3	32	13	21	10	11	31	8	3
	4	9	4	5	2	4	11	3	1

word	$h(w)$	nice positions
abcccc	1	6
abbccc	1	6
abbbcc	1	6
abbbbc	1	6
aabccc	1	6
aabbcc	1	6
aabbbc	1	6
aaabcc	1	6
aaabbc	1	6
aaaabc	1	6
acbbbb	1	2
accbbb	3	2, 4, 6
acccbb	2	2, 6
accccb	3	2, 4, 6
aacbbb	1	3
aaccbb	2	4, 6
aacccb	2	3, 5
aaacbb	1	4
aaaccb	2	4, 6
aaaacb	1	5
bacccc	0	
baaccc	0	
baaacc	0	
baaaac	0	
bbaccc	1	6
bbaacc	1	6
bbaaac	1	6
bbbacc	0	
bbbaac	1	6
bbbbac	1	6

word	$h(w)$	nice positions
bcaaaa	3	2, 4, 6
bccaaa	2	3, 5
bcccaa	2	2, 4
bcccca	3	1, 3, 5
bbcaaa	2	3, 5
bbccaa	2	2, 4
bbccca	3	1, 3, 5
bbbcaa	2	2, 4
bbbcca	3	1, 3, 5
bbbbca	3	1, 3, 5
cabbbb	1	1
caabbb	1	1
caaabb	1	1
caaaab	1	1
ccabbb	3	2, 4, 6
ccaabb	3	2, 4, 6
ccaaab	3	2, 4, 6
cccabb	2	3, 5
cccaab	2	3, 5
ccccab	2	2, 4
cbaaaa	3	1, 3, 5
cbbaaa	2	1, 5
cbbbaa	3	1, 3, 5
cbbbba	1	5
ccbaaa	2	1, 3
ccbbaa	2	4, 6
ccbbba	1	1
cccbaa	1	1
cccbba	3	1, 3, 5
ccccba	2	1, 5

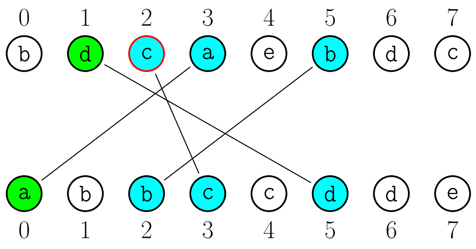
Table B.1: Fully clustered ternary words of length 6.

$\lvert\boldsymbol{\Sigma}\rvert = \mathbf{2}$								BWTs of	
	$h(w)$	all	%	noBWTs	%	BWTs	%	prim	pow
	total	256	100	220	100	36	100	30	6
$n = 9$	0	243	47	243	54	0	0	0	0
	1	150	29	118	26	32	53	30	2
	2	31	6	29	6	2	3	2	0
	3	56	11	48	11	8	13	7	1
	4	28	5	14	3	14	23	13	1
	5	4	1	0	0	4	7	4	0
	total	512	100	452	100	60	100	56	4
$n = 10$	0	515	50	515	56	0	0	0	0
	1	274	27	216	24	58	54	56	2
	2	53	5	48	5	5	5	5	0
	3	98	10	81	9	17	16	14	3
	4	70	7	48	5	22	20	19	3
	5	14	1	8	1	6	6	5	1
	total	1024	100	916	100	108	100	99	9
$n = 11$	0	1088	53	1088	58	0	0	0	0
	1	494	24	393	21	101	54	99	2
	2	104	5	96	5	8	4	8	0
	3	164	8	147	8	17	9	17	0
	4	142	7	114	6	28	15	28	0
	5	50	2	22	1	28	15	28	0
	6	6	< 0.5	0	0	6	3	6	0
	total	2048	100	1860	100	188	100	186	2
$n = 12$	0	2258	55	2258	60	0	0	0	0
	1	914	22	724	19	190	54	188	2
	2	200	5	183	5	17	5	17	0
	3	288	7	252	7	36	10	34	2
	4	284	7	224	6	60	17	51	9
	5	130	3	90	2	40	11	37	3
	6	22	1	13	< 0.5	9	3	8	1
	total	4096	100	3744	100	352	100	335	17
$n = 13$	0	4679	57	4679	62	0	0	0	0
	1	1680	21	1339	18	341	54	339	2
	2	388	5	365	5	23	4	23	0
	3	536	7	478	6	58	9	58	0
	4	522	6	455	6	67	11	67	0
	5	292	4	209	3	83	13	83	0
	6	85	1	35	< 0.5	50	8	50	0
	7	10	< 0.5	0	0	10	2	10	0
	total	8192	100	7560	100	632	100	630	2
$n = 14$	0	9601	59	9601	63	0	0	0	0

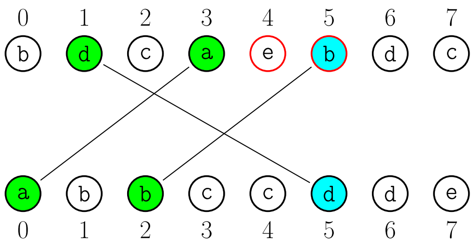
$\|\mathbf{\Sigma}\| = \mathbf{2}$								BWTs of	
	$h(w)$	all	%	noBWTs	%	BWTs	%	prim	pow
	1	3140	19	2498	16	642	54	640	2
	2	748	5	696	5	52	4	52	0
	3	1024	6	907	6	117	10	114	3
	4	982	6	843	6	139	12	134	5
	5	620	4	475	3	145	12	139	6
	6	235	1	161	1	74	6	70	4
	7	34	< 0.5	21	< 0.5	13	1	12	1
	total	16384	100	15202	100	1182	100	1161	21
$n = 15$	0	19664	60	19664	64	0	0	0	0
	1	5874	18	4695	15	1179	54	1177	2
	2	1464	4	1381	5	83	4	83	0
	3	1940	6	1775	6	165	8	165	0
	4	1880	6	1637	5	243	11	242	1
	5	1218	4	991	3	227	10	222	5
	6	574	2	380	1	194	9	192	2
	7	140	< 0.5	53	< 0.5	87	4	87	0
	8	14	< 0.5	0	0	14	1	14	0
	total	32768	100	30576	100	2192	100	2182	10
$n = 16$	0	40094	61	40094	65	0	0	0	0
	1	11062	17	8843	14	2219	54	2217	2
	2	2882	4	2709	4	173	4	173	0
	3	3702	6	3346	5	356	9	353	3
	4	3622	6	3157	5	465	11	459	6
	5	2426	4	1988	3	438	11	427	11
	6	1298	2	980	2	318	8	309	9
	7	402	1	273	< 0.5	129	3	125	4
	8	48	< 0.5	30	< 0.5	18	< 0.5	17	1
	total	65536	100	61420	100	4116	100	4080	36
$n = 17$	0	81602	62	81602	66	0	0	0	0
	1	20898	16	16758	14	4140	54	4138	2
	2	5711	4	5423	4	288	4	288	0
	3	7146	5	6589	5	557	7	557	0
	4	6863	5	6139	5	724	9	724	0
	5	4820	4	4020	3	800	10	800	0
	6	2736	2	2112	2	624	8	624	0
	7	1042	1	639	1	403	5	403	0
	8	234	< 0.5	78	< 0.5	156	2	156	0
	9	20	< 0.5	0	0	20	< 0.5	20	0
	total	131072	100	123360	100	7712	100	7710	2
$n = 18$	0	165632	63	165632	67	0	0	0	0
	1	39704	15	31871	13	7833	54	7831	2

$\mathbf{\lvert\Sigma\rvert = 2}$								BWTs of	
	$h(w)$	all	%	noBWTs	%	BWTs	%	prim	pow
	2	11281	4	10696	4	585	4	585	0
	3	13798	5	12641	5	1157	8	1153	4
	4	13167	5	11672	5	1495	10	1485	10
	5	9620	4	8113	3	1507	10	1492	15
	6	5722	2	4579	2	1143	8	1119	24
	7	2450	1	1818	1	632	4	622	10
	8	696	< 0.5	474	< 0.5	222	2	218	4
	9	74	< 0.5	46	< 0.5	28	< 0.5	27	1
	total	262144	100	247542	100	14602	100	14532	70

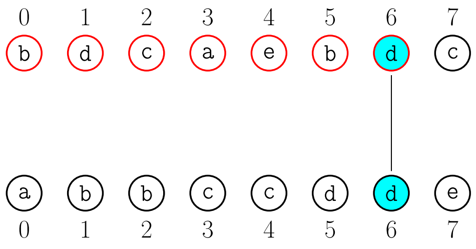
Table B.2: Statistics of words over a binary alphabet of lengths from 3 to 18. Percentages rounded to nearest integer. See text for further details.



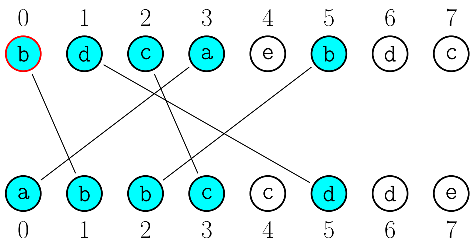
$S_1 = \{1,2,3,5\}$,
$\pi(S_1) = \{0,2,3,5\}$, $R_1 = [2,2]$



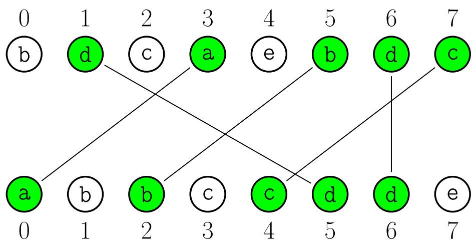
$S_2 = \{1,3,5\}$,
$\pi(S_2) = \{0,2,5\}$, $R_2 = [4,5]$.



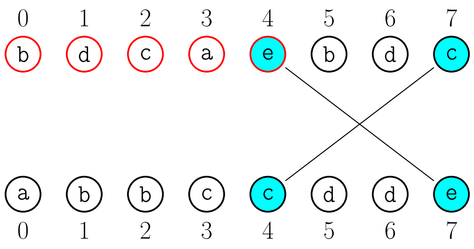
$S_3 = \{6\}$, $\pi(S_3) = \{6\}$, $R_3 = [0,6]$.



$S_2 = \{0,1,2,3,5\}$,
$\pi(S_2) = \{0,1,2,3,5\}$, $R_2 = [0,0]$.



$S_3 = \{1,3,5,6,7\}$,
$\pi(S_3) = \{0,2,4,5,6\}$, $R_3 = [8]$.



$S_2 = \{4,7\}$,
$\pi(S_2) = \{4,7\}$, $R_2 = [0,4]$.

Figure B.1: Essential and minimal right-only pseudo-cycles of the word w = `bdcaebdc`. For each pseudo-cycle S, elements contained in S_{left} are colored in green, and elements contained in S_{right} are colored in blue. Elements in the critical interval are highlighted in red.