九州大学学術情報リポジトリ
Kyushu University Institutional Repository

Regularities and Algorithms on Dynamic Strings

舩越, 満

https://hdl.handle.net/2324/4784637

出版情報：Kyushu University, 2021, 博士（情報科学）, 課程博士
バージョン：
権利関係：

Regularities and Algorithms on Dynamic Strings

Mitsuru Funakoshi

January, 2022

Abstract

In recent years, with the spread of the Internet, the development of sensor technology, and the widespread use of mobile devices, an enormous amount of diverse data has been generated daily. It is easy to predict that this data will continue to increase dramatically in the future, but most of it is accumulated or discarded without being utilized. Therefore, there is an urgent need for technology to process and utilize vast amounts of data instantaneously. Since most data can be regarded as strings, it is necessary to develop efficient data processing technology that takes advantage of the properties of strings. For this purpose, knowledge of the combinatorial structures of strings is indispensable. For example, repetitive structures are closely related to string compression, and strings with many repetitive structures are known to be small in practice under several compressors. For another example, finding palindromic structures has important applications to analyze biological data. Besides, most of the Internet data, sensor data, and social data are updated and edited at any time, so it is necessary to develop dynamic string processing techniques. There are several types of dynamic data, fully dynamic data (namely, the data where arbitrary positions can be updated many times), semi-dynamic data where editing position is limited, and the data with a limited number of edits.

In this thesis, we focus on the analysis of combinatorial structures and the development of algorithms in dynamic strings. We tackle the following two problems regarding repetitive/palindromic structures among the combinatorial structures: (A) The problem of analyzing the changes of repetitiveness measures and the size of string compressors in a dynamic string. (B) The problem of analyzing the changes and developing efficient finding algorithms for palindromic structures in a dynamic string.

First, we analyze changes in the size of string compressors and repetitiveness measures in dynamic strings. The *sensitivity* of a string compression algorithm C asks how much the output size $C(T)$ for an input string T can increase when a single character edit operation is performed on T. This notion enables one to measure the robustness of compression algorithms in terms of errors and/or dynamic changes occurring in the input string. In this thesis, we analyze the worst-case multiplicative sensitivity of string compression algorithms, which is defined by $\max_{T \in \Sigma^n} \{C(T')/C(T) : \mathsf{ed}(T, T') = 1\}$, where $\mathsf{ed}(T, T')$ denotes the edit distance between T and T'. In particular, for the most common versions of the Lempel-Ziv 77 compressors, we prove that the worst-case multiplicative sensitivity is only a small constant (2 or 3, depending on the version of the Lempel-Ziv 77 and the edit operation type), i.e., the size z of the Lempel-Ziv 77 factorizations can be larger by only a small constant factor. We strengthen our upper bound results by presenting matching lower bounds on the worst-case sensitivity for all these major versions of the Lempel-Ziv 77 factorizations. We then generalize this result to the smallest bidirectional scheme b. These results contrast with the previously known related results such that the size z_{78} of the Lempel-Ziv 78 factorization can increase by a factor of $\Omega(n^{3/4})$ [Lagarde and Perifel, 2018], and the number r of runs in the Burrows-Wheeler transform can increase by a factor of $\Omega(\log n)$ [Giuliani et al., 2021] when a character is prepended to an input string of length n. Further, we extend the notion of the worst-case sensitivity to string repetitiveness measures such as the smallest string attractor size γ and the substring complexity δ, and present some non-trivial lower bound for γ and matching upper and lower bounds of the worst-case multiplicative sensitivity for δ. We also exhibit the worst-case additive sensitivity $\max_{T \in \Sigma^n} \{C(T') - C(T) : \mathsf{ed}(T, T') = 1\}$, which allows one to observe more details in the changes of the output sizes.

Second, we deal with the problems of computing the longest palindromic substring (LPS) after the string is edited. It is known that the length of the LPSs of a given string T of length n can be computed in $O(n)$ time [Manacher, 1975]. We present an algorithm that uses $O(n)$ time and space for preprocessing, and answers the length of the LPSs in $O(\log(\min\{\sigma, \log n\}))$ time after a single character substitution, insertion, or deletion, where σ denotes the number of distinct characters appearing in T. We also propose an algorithm that uses $O(n)$ time and space for preprocessing, and answers the length of the LPSs in $O(\ell + \log(\min\{\sigma, \log n\}))$ time, after an existing substring in T is replaced by a string of arbitrary length ℓ.

Third, we deal with the problem of updating the set of minimal unique palindromic substrings (MUPSs) of a string after a single-character substitution. MUPSs are utilized for answering the *shortest unique palindromic substring problem*, which is motivated by molecular biology [Inoue et al., 2018]. Given a string T of length n, all MUPSs of T can be computed in $O(n)$ time. We first analyze the number d of changes of MUPSs when a character is substituted, and show that d is in $O(\log n)$. Further, we present an algorithm that uses $O(n)$ time and space for preprocessing, and updates the set of MUPSs in $O(\log \sigma + (\log \log n)^2 + d)$ time where σ is the alphabet size. We also propose a variant of the algorithm, which runs in optimal $O(1 + d)$

time when the alphabet size is constant.

Fourth, we deal with the problem of computing maximal/distinct palindromes in a trie $\mathcal{T}$. A trie is a natural generalization of a string which can be seen as a single path tree. It is known that all maximal palindromes of a given string T of length n can be computed in $O(n)$ time [Manacher 1975]. Also, all distinct palindromes in T can be computed in $O(n)$ time [Groult et al., 2010]. We propose algorithms to compute all maximal palindromes and all distinct palindromes in $\mathcal{T}$ in $O(N \log h)$ time and $O(N)$ space, where N is the number of edges in $\mathcal{T}$ and h is the height of $\mathcal{T}$. We also show online algorithms to compute all maximal/distinct palindromes in a trie in $O(N \log N)$ time and $O(N)$ space.

Acknowledgments

First of all, I would like to show my most significant appreciation to Professor Masayuki Takeda, my supervisor, and my thesis committee member. He provided me with the environment to focus on my research activities and the opportunity to have many valuable experiences. I also express my deep gratitude to Professor Eiji Takimoto, my thesis committee chair, and Associate Professor Daisuke Ikeda, my thesis committee member. They gave me many fruitful comments for my thesis.

In addition, I sincerely appreciate Professor Hideo Bannai, Associate Professor Shunsuke Inenaga, and Assistant Professor Yuto Nakashima. Hideo Bannai, my supervisor when I was an undergraduate student, taught me how to do research and gave me a lot of various knowledge and ideas. Shunsuke Inenaga provided me with guidance and advice on multiple aspects of research and research activities. Yuto Nakashima took me to various research meetings, introduced me to the communities, and cared for me.

I am grateful to Professor Eiji Takimoto and Associate Professor Kohei Hatano for their advice in the weekly seminar. I also thank all of those in Department of Informatics, Kyushu University, for their support.

This research was partly supported by JSPS (Japan Society for the Promotion of Science). The results in this thesis were partially published in the Proc. of CPM'18 and '19, the Proc. of PSC'19, and the Proc. of SPIRE'21. Also, the journal version of CPM'18 and '19 was published in Theoretical Computer Science by Elsevier. I am thankful for all editors, committees, anonymous referees, and publishers.

I would like to express my appreciation to all co-authors: Professor Ayumi Shinohara, Associate Professor Tomohiro I, Assistant Professor Dominik Köppl, Dr. Takaaki Nishimoto, Mr. Julian Pape-Lange, Dr. Takuya Mieno, Mr. Tooru Akagi, and Mr. Takumi Ideue. I would also like to thank Dr. Keisuke Goto, Dr. Takuya Takagi, and Assistant Professor Diptarama Hendrian. The discussion with them was fascinating and instructive.

I am also grateful to past and present members and technical staff of our laboratory. In particular, I thank Dr. Shiho Sugimoto, Dr. Kazuya Tsuruta, Dr. Yuta Fujishige, Dr. Takuya Mieno, and Mr. Ryotaro Mitsuboshi; they gave me a lot of advice and support for my laboratory life.

Last but not least, I thank my family for their support.

Contents

Chapter 1

Introduction

1.1 Background and Motivations

In recent years, with the spread of the Internet, the development of sensor technology, and the widespread use of mobile devices, an enormous amount of diverse data has been generated daily. It is easy to predict that this data will continue to increase dramatically in the future, but most of it is accumulated or discarded without being utilized. Therefore, there is an urgent need for technology to process and utilize vast amounts of data instantaneously. Since most data can be regarded as strings, it is necessary to develop efficient data processing technology that takes advantage of the properties of strings. For this purpose, knowledge of the combinatorial structures of strings is indispensable. For example, repetitive structures are closely related to string compression, and strings with many repetitive structures are known to be small in practice under several compressors. For another example, finding palindromic structures has important applications to analyze biological data. Besides, most of the Internet data, sensor data, and social data are updated and edited at any time, so it is necessary to develop dynamic string processing techniques. There are several types of dynamic data, fully dynamic data (namely, the data where arbitrary positions can be updated many times), semi-dynamic data where editing position is limited, and the data with a limited number of edits.

In this thesis, we focus on the analysis of combinatorial structures and the development of algorithms in dynamic strings. We tackle the following two problems regarding repetitive/palindromic structures among the combinatorial structures: (A) The problem of analyzing the changes of repetitiveness measures and the size of string compressors in a dynamic string. (B) The problem of analyzing the changes and developing efficient finding algorithms for palindromic structures in a dynamic string.

1.2 Our Contribution

1.2.1 Repetitive Structures

Sensitivity of String Compressors and Repetitive Measures

In the first part of this thesis we introduce a new notion to quantify efficiency of (lossless) compression algorithms, which we call the *sensitivity* of compressors. Let C be a compression algorithm and let $C(T)$ denote the size of the output of C applied to an input text (string) T. Roughly speaking, the sensitivity of C measures how much the compressed size $C(T)$ can change when a single-character-wise edit operation is performed at an arbitrary position in T. Namely, the worst-case *multiplicative sensitivity* of C is defined by

$$\max_{T \in \Sigma^n} \{C(T')/C(T) : \mathsf{ed}(T, T') = 1\},$$

where $\mathsf{ed}(T, T')$ denotes the edit distance between T and T'. This new and natural notion enables one to measure the robustness of compression algorithms in terms of errors and/or dynamic changes occurring in the input string. Such errors and dynamic changes are commonly seen in real-world texts such as DNA sequences and versioned documents.

The so-called highly repetitive sequences, which are strings containing a lot of repeated fragments, are abundant today: Semi-automatically generated strings via M2M communications, and collections of individual genomes of the same/close species are typical examples. By intuition, such highly repetitive sequences should be highly compressible, however, statistical compressors are known to fail to capture repetitiveness in a string [79]. Therefore, other types of compressors, such as dictionary-based, grammar-based, and/or lex-based compressors are often used to compress highly repetitive sequences [61, 81, 84, 114].

Let us recall two examples of well-known compressors: The *run-length Burrows-Wheeler Transform* (*RLBWT*) is one kind of compressor that is based on the lexicographically sorted rotations of the input string. The number r of equal-character runs in the Burrows-Wheeler Transform (BWT) of a string is known to be very small in practice: Indeed, BWT is used in the bzip2 compression format, and several compressed data structures which support efficient queries have been proposed [11, 43, 101, 102]. The *Lempel-Ziv 78 compression* (*LZ78*) [124] is one of the most fundamental dictionary based compressors that is a core of in the gif and tiff compression formats. While LZ78 only allows $\Omega(\sqrt{n})$ compression for any string of length n, its simple structure has allowed to design efficient compressed pattern matching algorithms and compressed self-indices (c.f. [35, 46, 47, 70, 96] and references therein).

The recent work by Giuliani et al. [54], however, shows that the number r of runs in the BWT of a string of length n can grow by a multiplicative factor of $\Omega(\log n)$ when a single character is prepended to the input string. The other work by Lagarde and Perifel [83] shows that the size of the dictionary of LZ78, which is equal to the number of factors in the respective LZ78 factorization, can grow by a multiplicative factor of $\Omega(n^{1/4})$, again when a single character is prepended to the input string. Letting the LZ78 dictionary size be z_{78}, this multiplicative increase can also be described as $\Omega(z_{78}^{3/2})$. Lagarde and Perifel call this phenomenon "one-bit catastrophe". In our context, the sensitivity of r and that of z_{78} are both high, since a mere single character insertion at the beginning of the string can drastically change the output size of these compressors.

In this thesis, we show that such a catastrophe *never happens* with the other major dictionary compressors, the *Lempel-Ziv 77 compression family*. The *LZ77* compression [123], which is the greedy parsing of the input string T where each factor of length more than one refers to a previous occurrence to its left, is the most important dictionary-based compressor both in theory and in practice. The LZ77 compression without self-references (resp. with self-references) can achieve $O(\log n)$ compression (resp. $O(1)$ compression) in the best case as opposed to the $\Omega(\sqrt{n})$ compression by the LZ78 counterpart, and the LZ77 compression is a core of common lossless compression formats including gzip, zip, and png. In addition, its famous version called *LZSS* (Lempel-Ziv-Storer-Szymanski) [115], has numerous applications in string processing, including finding repetitions [12, 31, 59, 76], approximation of the smallest grammar-based compression [28, 110], and compressed self-indexing [13, 14, 18, 97], just to mentioned a few.

We show that the multiplicative sensitivity of LZ77 with/without self-references is at most 2, namely, the number of factors in the respective LZ77 factorization can increase by at most a factor of 2 for all types of edit operations (substitution, insertion, deletion of a character). Then, we prove that the multiplicative sensitivity of LZSS with/without self-references is at most 3 for substitutions and deletions, and that it is at most 2 for insertions. We also present matching lower bounds for the multiplicative sensitivity of LZ77/LZSS with/without self-references for all types of edit operations as well. These results show that, unlike RLBWT and LZ78, LZ77 and LZSS well capture the repetitiveness of the input string, since a mere single character edit operation should not much influence the repetitiveness of a sufficiently long string. We also consider the *smallest bidirectional scheme* [115] that is a generalization of the LZ family where each factor can refer to its other occurrence to its left or right. It is shown that for all types of edit operations, the multiplicative sensitivity of the size b of the smallest bidirectional scheme is at most 2, and that there exists a string for which the multiplicative sensitivity of b is 1.5.

Further, we extend the notion of the worst-case multiplicative sensitivity to string repetitiveness measures such as the size γ of the *smallest string attractor* [69] and the *substring complexity* δ [74], both receiving recent attention [29, 68, 82, 88, 106]. We prove that the value of δ can increase by at most a factor of 2 for substitutions and insertions, and by at most a factor of 1.5 for deletions. We show these upper bounds are also tight by presenting matching lower bounds for the sensitivity of δ. We also present non-trivial lower bounds for the sensitivity of γ.

As is mentioned above, the work by Lagarde and Perifel [83] considered only the case of prepending a character to the string for the multiplicative sensitivity of LZ78. We present the same lower bounds hold for the multiplicative sensitivity of LZ78 in the case of substitutions and deletions, and insertions inside the string, by using a completely different instance from the one used in [83]. Moreover, we consider the multiplicative sensitivity of other compressors and repetitiveness measures including Bisection [99], GCIS [103, 104], CDAWGs [22], α-balanced grammars [28], AVL-grammars [110], and Recompression [66] in our paper [2]. (We omit the details in this thesis.)

All the results reported in the article and in the related work are summarized in Table 1.1.

In addition to the afore-mentioned multiplicative sensitivity, we also present the worst-case *additive sensitivity* which is defined as

$$\max_{T \in \Sigma^n} \{C(T') - C(T) : \mathsf{ed}(T, T') = 1\}$$

for all the string compressors/repetitiveness measures C dealt in this thesis. We remark that the additive sensitivity allows one to observe and evaluate more details in the changes of the output sizes, as summarized in Table 1.2. For instance, we obtain *strictly tight upper and lower bounds* for the additive sensitivity of LZ77 with and without self-references.

1.2.2 Palindromic Structures

Palindromes are strings that read the same forward and backward. Finding palindromic structures in strings has important applications in analysis of DNA, RNA, and protein sequences, and thus a variety of efficient algorithms for finding palindromic structures occurring in a given string have been proposed (e.g., see [9, 48, 56, 75, 78, 90, 105] and references therein).

Consider a set $C = \{1, 1.5, 2, \ldots, n\}$ of $2n-1$ half-integer and integer positions in a string T of length n. The maximal palindrome for a position $c \in C$ in T is a non-extensible palindrome whose center lies on c. It is easy to store all maximal palindromes with $O(n)$ total space; e.g., simply store their lengths in an array of length $2n - 1$ together with the input string T. If

Table 1.1: Multiplicative sensitivity of the string compressors and string repetitiveness measures studied in this thesis and in the literature, where n is the input string length and Σ is the alphabet. In the table "sr" stands for "with self-references".

compressor/repetitiveness measure	edit type	upper bound	lower bound
Substring Complexity δ	ins./subst.	2	2
	deletion	1.5	1.5
Smallest String Attractor γ	all	-	2
RLBWT r	insertion	$O(\log n \log r)$	$\Omega(\log n)$ [54]
	del./subst.		-
Bidirectional Scheme b	all	2	1.5
LZ77 z_{77} LZ77sr $z_{77\mathrm{sr}}$	all	2	2
LZSS z_{SS} LZSSsr z_{SSsr}	del./subst.	3	3
	insertion	2	2
LZ78 z_{78}	insertion	-	$\Omega(n^{1/4})$ [83]
	del./subst.	-	$\Omega(n^{1/4})$
LZ-End z_{End}	all	-	2
α-balanced grammar g_{α}	all	$O(\log n)$	-
AVL grammar g_{avl}			
Recompression g_{rcmp}			
Bisection g_{bsc}	substitution	2	2
	ins./del.	$\lvert\Sigma\rvert + 1$	2
GCIS g_{is}	all	4	4
CDAWG e	all	-	2

Table 1.2: Additive sensitivity of the string compressors and string repetitiveness measures studied in this thesis, where n is the input string length and Σ is the alphabet. Some upper/lower bounds are described in terms of both the measure and n. In the table "sr" stands for "with self-references".

compressor/ repetitiveness measure	edit type	upper bound		lower bound	
Substring Complexity δ	all	1		1	
Smallest String Attractor γ	all	-		$\gamma - 3$	$\Omega(\sqrt{n})$
RLBWT r	insertion	$O(r \log n \log r)$		-	$\Omega(\log n)$ [54]
	del./subst.				-
Bidirectional Scheme b	all	$b + 2$		$b/2 - 3$	$\Omega(\sqrt{n})$
LZ77 z_{77}	subst./ins.	$z_{77} - 1$		$z_{77} - 1$	$\Omega(\sqrt{n})$
	deletion	$z_{77} - 2$		$z_{77} - 2$	
LZ77sr z_{77}	subst./ins.	$z_{77\mathrm{sr}}$		$z_{77\mathrm{sr}}$	$\Omega(\sqrt{n})$
	deletion			$z_{77\mathrm{sr}} - 2$	
LZSS z_{SS}	del./subst.	$2z_{\mathrm{SS}} - 2$		$2z_{\mathrm{SS}} - \Theta(\sqrt{z_{\mathrm{SS}}})$	$\Omega(\sqrt{n})$
	insertion	z_{SS}		$z_{\mathrm{SS}} - \Theta(\sqrt{z_{\mathrm{SS}}})$	
LZSSsr z_{SSsr}	del./subst.	$2z_{\mathrm{SSsr}}$		$2z_{\mathrm{SSsr}} - \Theta(\sqrt{z_{\mathrm{SSsr}}})$	$\Omega(\sqrt{n})$
	insertion	$z_{\mathrm{SSsr}} + 1$		$z_{\mathrm{SSsr}} - \Theta(\sqrt{z_{\mathrm{SSsr}}})$	
LZ78 z_{78}	insertion	-		$\Omega((z_{78})^{3/2})$ [83]	$\Omega(n/\log n)$ [83]
	del./subst.	-		$\Omega((z_{78})^{3/2})$	$\Omega(n^{3/4})$
LZ-End z_{End}	all	-		$z_{\mathrm{End}} - \Theta(\sqrt{z_{\mathrm{End}}})$	$\Omega(\sqrt{n})$
α-balanced grammar g_{α}	all	$O(z_{\mathrm{SS}} \log n)$		-	
AVL grammar g_{avl}					
Recompression g_{rcmp}					
Bisection g_{bsc}	substitution	g_{bsc}	$\lceil \log_2 n \rceil$	$g_{\mathrm{bsc}} - 4$	$2\log_2 n - 4$
	ins./del.	$\lvert\Sigma\rvert g_{\mathrm{bsc}}$		$g_{\mathrm{bsc}} - 3$	$2\log_2 n - 3$
GCIS g_{is}	all	$3g_{\mathrm{is}}$		$3g_{\mathrm{is}} - 29$	$(3/4)n + 1$
CDAWG e	all	-		e	n

$P = T[i..j]$ is a maximal palindrome with center $c = \frac{i+j}{2}$, then clearly any substrings $P' = T[i+d..j-d]$ with $0 \leq d \leq \frac{j-i}{2}$ are also palindromes. Hence, by computing and storing all maximal palindromes in T, we can obtain a compact representation of all palindromes in T.

Manacher [86] gave an elegant $O(n)$-time algorithm to compute all maximal palindromes in T. This algorithm utilizes symmetry of palindromes and character equality comparisons only, and therefore works in $O(n)$ time for any alphabet. For the case where the input string is drawn from a constant size alphabet or an integer alphabet of size polynomial in n, there is an alternative suffix tree [120] based algorithm which takes $O(n)$ time [58]. In this method, the suffix tree of $T\#T^R\$$ is constructed, where T^R is the reversed string of T, and $\#$ and $\$$ are special characters not occurring in T. By enhancing the suffix tree with a lowest common ancestor (LCA) data structure [34], *outward* longest common extension (LCE) queries from a given $c \in C$ can be answered in $O(1)$ time after an $O(n)$-time preprocessing. Therefore, we can compute all maximal palindromes in $O(n)$ time by using outward LCE queries for each center position.

Another central question regarding palindromic substrings is distinct palindromes. Droubay et al. [33] showed that any string of length n can contain at most $n+1$ distinct palindromes (including the empty string). Strings of length n that contain exactly $n+1$ distinct palindromes are called *rich* strings in the literature [25, 55]. Groult et al. [56] proposed an $O(n)$-time algorithm for computing all distinct palindromes in a string of length n over a constant-size alphabet or an integer alphabet of size polynomial in n.

Longest Palindromic Substring After Edit

In the second part of this thesis, we consider the fundamental problem of finding the *longest palindromic substring* (*LPS*) in a given string T. Observe that the longest palindromic substring is also a maximal palindrome in the string. Hence, in order to compute the LPS of a given string T, it suffices to compute all maximal palindromes in T. Therefore the LPS in a string can be computed in $O(n)$ time. Finding the longest palindromic substring in the streaming model has also been considered [17, 53]. Getting back to the problem of computing all maximal palindromes, there is a simple $O(n)$-space data structure representing all of the computed maximal palindromes. However, this data structure is apparently not flexible for string edits, since even a single character substitution, insertion, or deletion can significantly break palindromic structures of the string. Indeed, $\Omega(n^2)$ palindromic substrings and $\Omega(n)$ maximal palindromes can be affected by a single edit operation (E.g., consider to replace the middle character of string a^n with another character b). Hence, an intriguing question is whether there exists a space-efficient data structure for the input string T which can quickly answer the following query: What is the length of the longest palindromic substring(s), if single character substitution, insertion, or deletion is performed? We call this a *1-ELPS query*.

In this thesis, we present an algorithm which uses $O(n)$ time and space for preprocessing, and $O(\log(\min\{\sigma, \log n\}))$ time for 1-ELPS queries, where σ is the number of distinct characters appearing in T. Thus, our query algorithm runs in optimal $O(1)$ time for any constant-size alphabet, and runs in $O(\log \log n)$ time for larger alphabets of size $\sigma = \Omega(\log n)$. In addition, this algorithm can readily be extended to a randomized version with hashing, which answers queries in $O(1)$ time each and uses $O(n)$ expected time and $O(n)$ space for preprocessing. We also consider a more general variant of 1-ELPS queries which allows for a *block-wise* edit operation, where an existing substring in the input string T can be replaced with a string of arbitrary length ℓ, called an *ℓ-ELPS queries*. We present an algorithm which uses $O(n)$ time and space for preprocessing and $O(\ell + \log(\min\{\sigma, \log n\}))$ time for ℓ-ELPS queries. We emphasize that this query time is independent of the length of the original block (substring) to be edited.

Minimal Unique Palindromic Substring After Edit

In the third part of this thesis, we treat a notion of palindromic structures called *minimal unique palindromic substring* (*MUPS*) that is introduced in [65]. A palindromic substring $T[i..j]$ of a string T is called a MUPS of T if $T[i..j]$ occurs exactly once in T and $T[i + 1..j - 1]$ occurs at least twice in T. MUPSs are utilized for solving the *shortest unique palindromic substring* (*SUPS*) problem proposed by Inoue et al. [65], which is motivated by an application in molecular biology. They showed that there are no more than n MUPSs in any length-n string, and proposed an $O(n)$-time algorithm to compute all MUPSs of a given string of length n over an integer alphabet of size $n^{O(1)}$. After that, Watanabe et al. [119] considered the problem of computing MUPSs in an *run-length encoded* (*RLE*) string. They showed that there are no more than m MUPSs in a string whose RLE size is m. Also, they proposed an $O(m \log \sigma_R)$-time and $O(m)$-space algorithm to compute all MUPSs of a string given in RLE, where σ_R is the number of distinct single-character runs in the RLE string. Recently, Mieno et al. [91] considered the problems of computing palindromic structures in the *sliding window* model. They showed that the set of MUPSs in a sliding window can be maintained in a total of $O(n \log \sigma_W)$ time and $O(D)$ space while a window of size D shifts over a string of length n from the left-end to the right-end, where σ_W is the maximum number of distinct characters in the windows. This result can be rephrased as follows: The set of MUPSs in a string of length D can be updated in amortized $O(\log \sigma_W)$ time using $O(D)$ space after deleting the first character or inserting a character to the right-end.

To the best of our knowledge, there is no efficient algorithm for updating the set of MUPSs after editing a character at *any* position so far. Now, we consider the problem of updating the set of MUPSs in a string after substituting a character at any position. Formally, we tackle the following problem: Given a string T of length n over an integer alphabet of size $n^{O(1)}$ to preprocess, and then given a query of single-character substitution. Afterwards, we return the set of MUPSs of the edited string. In this thesis, we first show that the number d of changes of MUPSs after a single-character substitution is $O(\log n)$. In addition, we present an algorithm that uses $O(n)$ time and space for preprocessing, and updates the set of MUPSs in $O(\log \sigma + (\log \log n)^2 + d) \subset O(\log n)$ time. We also propose a variant of the algorithm, which runs in optimal $O(1 + d)$ time when the alphabet size is constant.

Palindromes in a Trie

In the fourth part of this thesis, we tackle the problems of computing palindromes in a given trie. A *trie* is a rooted tree where each edge is labeled by a single character and the out-going edges of each node are labeled by mutually distinct characters. A trie is a natural extension to a string, and is a compact representation of a set of strings. There are a number of works for efficient algorithms on tries, such as indexing a (reversed) trie [23, 36, 63, 64, 77, 93, 95, 113] for exact pattern matching, parameterized pattern matching on a trie [8, 39], order preserving pattern matching on a trie [94], finding all maximal repetitions (a.k.a. runs) in a trie [116], computing all minimal absent words in a trie [37], reporting all covers in a trie [108].

Now we consider the problems of computing all maximal palindromes and all distinct palindromes in a given trie $\mathcal{T}$. Naïve methods for solving the problems would be to apply Manacher's algorithm [86] or Groult et al.'s algorithm [56] for each string in $\mathcal{T}$, but this requires $\Omega(N^2)$ time in the worst case since there exists a trie with N edges that can represent $\Theta(N)$ strings of length $\Theta(N)$ each. We also remark that a direct application of Manacher's algorithm to a trie does not seem to solve our problem efficiently, since the amortization argument in the case of a single string does not hold in our case of a trie. The aforementioned suffix tree approach [58] cannot be applied to our trie case either; while the number of suffixes in the reversed leaf-to-root direction of the trie $\mathcal{T}$ is N, the number of suffixes in the forward root-to-leaf direction can be $\Theta(N^2)$ in the worst case. Thus one cannot afford to construct the suffix tree that contains all suffixes of the forward paths of $\mathcal{T}$.

In this thesis, we show that the number of maximal palindromes in a trie $\mathcal{T}$ with N edges and L leaves is exactly $2N - L$ and that the number of distinct palindromes in $\mathcal{T}$ is at most

$N+1$. These generalize the known bounds for a single string. Then, we present two algorithms to compute all maximal palindromes both of which run in $O(N \log h)$ time and $O(N)$ space in the worst case, where h is the height of the trie $\mathcal{T}$. We then present how to compute all distinct palindromes in a given trie $\mathcal{T}$ in $O(N \log h)$ time with $O(N)$ space. We also show online algorithms computing all maximal/distinct palindromes in a trie in $O(N \log N)$ time and $O(N)$ space. The key tools we use are periodicities of suffix palindromes and string data structures that are built on the (reversed) trie. To the best of our knowledge, these are the first algorithms for finding maximal/distinct palindromes from a given trie in sub-quadratic time.

1.3 Related Work

Related Work of Sensitivity

A string repetitiveness measure C is called *monotone* if, for any string T of length n, $C(T') \leq C(T)$ holds with any of its prefixes $T' = T[1..i]$ and suffixes $T' = [j..n]$ [74]. Kociumaka et al [74] pointed out that δ is monotone, and posed a question whether γ or the size b of the smallest bidirectional macro scheme [115] are monotone. This monotonicity for C can be seen as a special and extended case of our sensitivity for deletions, namely, if we restrict T' to be the string obtained by deleting either the first or the last character from T, then it is equivalent to asking whether $\max_{T \in \Sigma}\{C(T')/C(T) : T' \in \{T[1..n-1], T' = T'[2..n]\}\} \leq 1$. Mantaci et al. [88] proved that γ is not monotone, by showing a family of strings T such that $\gamma(T) = 2$ and $\gamma(T') = 3$ with $T' = T[1..n-1]$, which immediately leads to a lower bound $3/2 = 1.5$ for the multiplicative sensitivity of γ. Our matching upper and lower bounds for the multiplicative sensitivity of γ, which are both 2, improve this existing bound due to Mantaci et al.. Mitsuya et al. [92] considered the monotonicity of LZ77 without self-references z_{77} presented a family of strings T for which $z_{77}(T')/z_{77}(T) \approx 4/3$ with $T' = [2..n]$. Again, our matching upper and lower bounds for the multiplicative sensitivity of z_{77}, which are both 2, improve this $4/3$ bound.

The notion of the sensitivity of (general) algorithms was first introduced by Varma and Yoshida [118]. They studied the *average* sensitivity of well-known graph algorithms, and presented interesting lower and upper bounds on the expected number of changes in the output of an algorithm A, when a randomly chosen edge is deleted from the input graph G. The worst-case sensitivity of a graph algorithm for edge-deletions and vertex-deletions was considered by Yoshida and Zhou [122].

1.3.1 Related Work of After-Edit Problem

Amir et al. [5] proposed an algorithm to find the *longest common factor* (*LCF*) of two strings, after a single character edit operation is performed in one of the strings. Their data structure occupies $O(n \log^3 n)$ space and uses $O(\log^3 n)$ query time, where n is the length of the input strings. Their data structure can be constructed in $O(n \log^4 n)$ expected time. After that, Abedin et al. [1] showed that the above problem can be reduced to the heaviest induced ancestors problem over two trees and solved in $O(\log n \log \log n)$ (resp. $O(\log^2 n \log \log n)$) query time using an $O(n \log n)$ (resp. $O(n)$) space data structure. Urabe et al. [117] considered the problem of computing the longest *Lyndon word* in a string after an edit operation. They showed algorithms for $O(\log n)$-time queries for a single character edit operation and $O(\ell \log \sigma + \log n)$-time queries for a block-wise edit operation, both using $O(n)$ time and space for preprocessing. We note that in these results including ours, edit operations are given as *queries* and thus the input string(s) remain static even after each query. This is due to the fact that changing the data structure dynamically can be too costly in many cases.

It is noteworthy, however, that recently Amir et al. [7] solved dynamic versions for the LCF problem and some of its variants. In particular, when n is the maximum length of the string that can be edited, they showed a data structure of $O(n \log n)$ space that can be dynamically maintained and can answer 1-ELPS queries in $O(\sqrt{n} \log^{2.5} n)$ time, after $O(n \log^2 n)$ time preprocessing. Furthermore, Amir et al. [3] presented an algorithm for computing the longest palindrome in a dynamic string in $O(\text{polylog}\, n)$ time per single character substitution. In comparison to these recent results on dynamic strings, although our algorithm does not allow for changing the string, our algorithm answers 1-ELPS queries in $O(\log \log n)$ time or even faster for small alphabet size σ, which is exponentially faster than $O(\text{polylog}\, n)$.

In addition, Amir et al. [4] presented fully dynamic algorithm for maintaining a representation of the squares and Charalampopoulos et al. [26] showed polylogarithmic time algorithm for the LCF problem of two dynamic strings.

1.3.2 Related Work of Palindromes in a Trie

There are a few combinatorial results for palindromes in an *unrooted* edge-labeled tree. Brlek et al. [24] showed an $\Omega(M^{3/2})$ lower bound on the maximum number of distinct palindromes in an unrooted tree with M edges. Later Gawrychowski et al. [50] showed a matching upper bound $O(M^{3/2})$ on the maximum number of distinct palindromes in an unrooted tree with M edges. Moreover, they proposed $O(M^{1.5} \log M)$-time algorithm for reporting all distinct palindromes in an unrooted tree in [51]. Note that these works consider an unrooted tree, and, to the best of our knowledge, palindromes of a trie (rooted edge-labeled tree) have previously not been studied. Concerning repetitive structures in tries, Sugahara et al. [116] proved that any trie with N edges can contain less than N maximal repetitions (or runs), and showed that all runs in a given trie can be found in $O(N \log\log N)$ time with $O(N)$ space. In addition, Fici and Gawrychowski [37] showed that the size of the set of minimal absent words in a rooted (resp. unrooted) tree with N edges (resp. M edges) can be bounded in $O(N\sigma)$ (resp. $O(M^2\sigma)$) where σ is the alphabet size. Also they showed that these bounds are tight. Then, they presented algorithms to compute all minimal absent words in a rooted (resp. unrooted) tree in output-sensitive $O(N + \mathit{occ})$ time (resp. $O(M^2 + \mathit{occ})$ time). Recently, Radoszewski et al. [108] showed how to compute all covers of a rooted (resp. unrooted) tree in $O(N \log N / \log\log N)$ time (resp. $O(M^2)$ time and space or $O(M^2 \log M)$ time and $O(M)$ space).

1.4 Organization

This thesis is organized as follows. In Chapter 2, we introduce some notation and definitions. In addition, we explain some algorithmic tools. In Chapter 3, we present the worst-case sensitivity of several string compressors and repetitiveness measures. In Chapter 4, we show algorithms for computing the longest palindromic substring of a string after a single character/block-wise edit operation. In Chapter 5, we show upper bounds on the changes of the set of MUPSs after a single-character substitution. Furthermore, we present an algorithm for updating the set of MUPSs after a single-character substitution. In Chapter 6, we show tight bounds on the number of maximal palindromes and distinct palindromes in a trie. In addition, we present algorithms to compute all maximal/distinct palindromes in a trie. Finally, we conclude.

Chapter 2

Preliminaries

2.1 Notation

Let Σ be an *alphabet* of size σ. An element of Σ is called a *character*. An element of Σ^* is called a *string*. For any non-negative integer n, let Σ^n denote the set of strings of length n over Σ. The length of a string T is denoted by $|T|$. The *empty string* ε is the string of length 0. For a string $T = xyz$, then x, y, and z are called a *prefix*, *substring*, and *suffix* of T, respectively. They are called a *proper prefix*, *proper substring*, *proper suffix* of T if $x \neq T$, $y \neq T$, and $z \neq T$, respectively. For each $1 \leq i \leq |T|$, $T[i]$ denotes the i-th character of T. For each $1 \leq i \leq j \leq |T|$, $T[i..j]$ denotes the substring of T starting at position i and ending at position j. For convenience, let $T[i..j] = \varepsilon$ for any $i > j$.

For strings X and Y, let $lcp(X, Y)$ denotes the length of the *longest common prefix* (in short, lcp) of X and Y, i.e., $lcp(X, Y) = \max\{\ell \mid X[1..\ell] = Y[1..\ell]\}$. A *rightward longest common extension* (*rightward LCE*) query on a string T is to compute $lcp(T[i..|T|], T[j..|T|])$ for given two positions $1 \leq i \neq j \leq |T|$. Similarly, a *leftward LCE* query is to compute $lcp(T[1..i]^R, T[1..j]^R)$. We denote by $\mathsf{RightLCE}_T(i, j)$ and $\mathsf{LeftLCE}_T(i, j)$ rightward and leftward LCE queries for positions $1 \leq i \neq j \leq |T|$, respectively. An *outward LCE* query is, given two positions $1 \leq i < j \leq |T|$, to compute $lcp((T[1..i])^R, T[j..|T|])$. We denote by $\mathsf{OutLCE}_T(i, j)$ an outward LCE query for positions $i < j$ in the string T.

An integer $p \geq 1$ is said to be a *period* of a string T iff $T[i] = T[i+p]$ for all $1 \leq i \leq |T|-p$. If a string B is both a proper prefix and a proper suffix of another string T, then B is called a *border* of T. A factorization of a non-empty string T is a sequence $f_1, \ldots, f_x$ of non-empty substrings of T such that $T = f_1 \cdots f_x$. Each f_i is called a *factor*. The *size* of the factorization is the number x of factors in the factorization. The *run length* (*RL*) factorization of a string T is a sequence $f_1, \ldots, f_m$ of maximal runs of the same characters such that $T = f_1 \cdots f_m$ (namely, each RL factor f_j is a repetition of the same character a_j with $a_j \neq a_{j+1}$). For each position $1 \leq i \leq n$ in T, let $\mathit{RLFBeg}(i)$ and $\mathit{RLFEnd}(i)$ denote the beginning and ending positions of the RL factor that contains the position i, respectively. One can easily compute in $O(n)$ time the RL factorization of string T of length n together with $\mathit{RLFBeg}(i)$ and $\mathit{RLFEnd}(i)$ for all positions $1 \leq i \leq n$.

For non-empty strings T and w, $\mathit{beg}_T(w)$ denotes the set of beginning positions of occurrences of w in T. Also, for a text position i in T, $\mathit{inbeg}_{T,i}(w)$ denotes the set of beginning positions of occurrences of w in T where each occurrence covers position i. Namely, $\mathit{beg}_T(w) = \{b \mid T[b..e] = w\}$ and $\mathit{inbeg}_{T,i}(w) = \{b \mid T[b..e] = w \text{ and } i \in [b, e]\}$. Further, let $\mathit{xbeg}_{T,i}(w) = \mathit{beg}_T(w) \setminus \mathit{inbeg}_{T,i}(w)$. For convenience, $|\mathit{beg}_T(\varepsilon)| = |\mathit{inbeg}_{T,i}(\varepsilon)| = |\mathit{xbeg}_{T,i}(\varepsilon)| = |T| + 1$ for any T and i. We say that w is ***unique*** in T if $|\mathit{beg}_T(w)| = 1$, and that w is ***repeating*** in T if $|\mathit{beg}_T(w)| \geq 2$. Note that the empty string is repeating in any other string. Since every unique substring $u = T[i..j]$ of T occurs exactly once in T, we will sometimes identify u with its corresponding interval $[i, j]$.

Our model of computation is a standard word RAM model with machine word size $\Theta(\log n)$.

2.2 Worst-Case Sensitivity of Compressors and Repetitiveness Measures

For a string compression algorithm C and an input string T, let $C(T)$ denote the size of the compressed representation of T obtained by applying C to T. For convenience, we use the same notation when C is a string repetitiveness measure, namely, $C(T)$ is the value of the measure C for T.

Let us consider the following edit operations on strings: character substitution (sub), character insertion (ins), and character deletion (del). For two strings T and S, let $\mathsf{ed}(T, S)$ denote the *edit distance* between T and S, namely, $\mathsf{ed}(T, S)$ is the minimum number of edit operations that transform T into S.

Our interest in this thesis is: "How much can the compression size or the repetitiveness measure size change when a single-character-wise edit operation is performed on a string?" To answer this question, for a given string length n, we consider an arbitrarily fixed string T of length n and all strings T' that can be obtained by applying a single edit operation to T, that is, $\mathsf{ed}(T, T') = 1$. We define the worst-case *multiplicative sensitivity* of C w.r.t. a substitution, insertion, and deletion as follows:

$$\begin{aligned}\mathsf{MS}_{\mathrm{sub}}(C,n) &= \max_{T\in\Sigma^n}\{C(T')/C(T) : T' \in \Sigma^n, \mathsf{ed}(T,T')=1\},\\ \mathsf{MS}_{\mathrm{ins}}(C,n) &= \max_{T\in\Sigma^n}\{C(T')/C(T) : T' \in \Sigma^{n+1}, \mathsf{ed}(T,T')=1\},\\ \mathsf{MS}_{\mathrm{del}}(C,n) &= \max_{T\in\Sigma^n}\{C(T')/C(T) : T' \in \Sigma^{n-1}, \mathsf{ed}(T,T')=1\}.\end{aligned}$$

We also consider the worst-case *additive sensitivity* of C w.r.t. a substitution, insertion, and deletion, as follows:

$$\begin{aligned}\mathsf{AS}_{\mathrm{sub}}(C,n) &= \max_{T\in\Sigma^n}\{C(T')-C(T) : T' \in \Sigma^n, \mathsf{ed}(T,T')=1\},\\ \mathsf{AS}_{\mathrm{ins}}(C,n) &= \max_{T\in\Sigma^n}\{C(T')-C(T) : T' \in \Sigma^{n+1}, \mathsf{ed}(T,T')=1\},\\ \mathsf{AS}_{\mathrm{del}}(C,n) &= \max_{T\in\Sigma^n}\{C(T')-C(T) : T' \in \Sigma^{n-1}, \mathsf{ed}(T,T')=1\}.\end{aligned}$$

We remark that, in general, $C(T')$ can be larger than $C(T)$ even when T' is obtained by a character deletion from T (i.e. $|T'| = n-1$). Such strings T are already known for the Lempel-Ziv 77 factorization size z when $T' = T[2..n]$ [92], or for the smallest string attractor size γ when $T' = T[1..n-1]$ [88].

The above remark implies that in general the multiplicative/additive sensitivity for insertions and deletions may not be symmetric and therefore they need to be discussed separately for some C. Note, on the other hand, that the maximum difference between $C(T')$ and $C(T)$ when $|T'| = n-1$ (deletion) and $C(T') - C(T) < 0$ is equivalent to $\mathsf{AS}_{\mathrm{ins}}(C, n-1)$, and symmetrically the maximum difference of $C(T')$ and $C(T)$ when $|T'| = n+1$ (insertion) and $C(T')-C(T) < 0$ is equivalent to $\mathsf{AS}_{\mathrm{del}}(C, n+1)$, with the roles of T and T' exchanged. Similar arguments hold for the multiplicative sensitivity with insertions/deletions. Consequently, it suffices to consider $\mathsf{MS}_{\mathrm{ins}}(C,n)$, $\mathsf{MS}_{\mathrm{del}}(C,n)$, $\mathsf{AS}_{\mathrm{ins}}(C,n)$, $\mathsf{AS}_{\mathrm{del}}(C,n)$ for insertions/deletions.

2.3 Palindromes

Let T^R denote the reversed string of T, i.e., $T^R = T[|T|]\cdots T[1]$. A string T is called a *palindrome* if $T = T^R$. We remark that the empty string ε is also considered to be a palindrome. A palindrome w is called an *even-palindrome* (resp. *odd-palindrome*) if its length is even (resp. odd). For a palindrome w, its length-$\lfloor |w|/2 \rfloor$ prefix (resp. length-$\lfloor |w|/2 \rfloor$ suffix) is called the *left arm* (resp. *right arm*) of w, and is denoted by larm_w (resp. rarm_w). Also, we call $\mathsf{Larm}_w = \mathsf{larm}_w \cdot s_w$ (resp. $\mathsf{Rarm}_w = s_w \cdot \mathsf{rarm}_w$) the *extended left arm* (resp. *extended right arm*) of w where s_w is the character at the center of w if w is an odd-palindrome, and s_w is empty otherwise. Note that when w is an even-palindrome, $\mathsf{Rarm}_w = \mathsf{rarm}_w$ and $\mathsf{Larm}_w = \mathsf{larm}_w$. For a non-empty palindromic substring $w = T[i..j]$ of a string T, the center of w is $\frac{i+j}{2}$ and is denoted by $center(w)$. A non-empty palindromic substring $T[i..j]$ is said to be a *maximal palindrome* of T if $T[i-1] \neq T[j+1]$, $i = 1$, or $j = |T|$. It is clear that for each center $c = 1, 1.5, \ldots, n-0.5, n$, we can identify the maximal palindrome $T[i..j]$ whose center is c. Thus, there are exactly $2n-1$ maximal palindromes in a string of length n (including empty ones which occur at center $\frac{i+j}{2}$ when $T[i] \neq T[j]$). In particular, maximal palindromes $T[1..i]$ and $T[i..|T|]$ for $1 \leq i \leq n$ are respectively called a *prefix palindrome* and a *suffix palindrome* of T.

Manacher [86] showed an elegant online algorithm which computes all maximal palindromes of a given string T of length n in $O(n)$ time. An alternative offline approach is to use outward LCE queries for $2n-1$ pairs of positions in T. Using the suffix tree [120] for string $T\$T^R\#$ enhanced with a lowest common ancestor data structure [15, 60, 111], where $\$$ and $\#$ are special characters which do not appear in T, each outward LCE query can be answered in $O(1)$ time. For any integer alphabet of size polynomial in n, preprocessing for this approach takes $O(n)$ time and space [34, 58]. Let $\mathcal{M}$ be an array of length $2n-1$ storing the lengths of maximal palindromes in increasing order of centers. For convenience, we allow the index for $\mathcal{M}$ to be an integer or a half-integer from 1 to n, so that $\mathcal{M}[i]$ stores the length of the maximal palindrome of T centered at i.

A palindromic substring P of a string T is called a *longest palindromic substring* (*LPS*) if there are no palindromic substrings of T which are longer than P. Since any LPS of T is always a maximal palindrome of T, we can find all LPSs and their lengths in $O(n)$ time.

For a non-empty palindromic substring $w = T[i..j]$ of a string T and a non-negative integer ℓ, $v = T[i-\ell..j+\ell]$ is said to be an *expansion* of w if $1 \leq i-\ell \leq j+\ell \leq n$ and v is a palindrome. Also, $T[i+\ell..j-\ell]$ is said to be a *contraction* of w.

A non-empty string w is called a *1-mismatch palindrome* if there is exactly one mismatched position between $w[1..\lfloor |w|/2 \rfloor]$ and $w[\lceil |w|/2 \rceil + 1..|w|]^R$. Informally, a 1-mismatch palindrome is a pseudo palindrome with a mismatch position between their arms. As in the case of palindromes, a 1-mismatch palindromic substring $T[i..j]$ of a string T is said to be *maximal* if $i = 1$, $j = n$ or $T[i-1] \neq T[j+1]$.

A palindromic substring $T[i..j]$ of a string T is called a *minimal unique palindromic substring* (*MUPS*) of T if $T[i..j]$ is unique in T and $T[i+1..j-1]$ is repeating in T. We denote by $\mathsf{MUPS}(T)$ the set of intervals corresponding to MUPSs of a string T. A MUPS cannot be a substring of another palindrome with a different center. Also, it is known that the number of

MUPSs of T is at most n, and set $\mathsf{MUPS}(T)$ can be computed in $O(n)$ time for a given string T over an integer alphabet [65].

Properties of Palindromes

The following properties of palindromes are useful in our algorithms.

Lemma 2.1. *Any border B of a palindrome P is also a palindrome.*

Proof. Since P is a palindrome, for any $1 \leq m \leq |P|$, $P[1..m] = (P[|P| - m + 1..|P|])^R$. Since B is a border of P, we have that $B = P[1..|B|] = (P[|P| - |B| + 1..|P|])^R = B^R$. $\square$

Let T be a string of length n. For each $1 \leq i \leq n$, let $\mathit{MaxPalEnd}_T(i)$ denote the set of maximal palindromes of T that end at position i. Let $\mathbf{S}_i = s_1, \ldots, s_g$ be the sequence of lengths of maximal palindromes in $\mathit{MaxPalEnd}_T(i)$ sorted in increasing order, where $g = |\mathit{MaxPalEnd}_T(i)|$. Also, let $s_0 = \varepsilon$. Let d_j be the progression difference for s_j, i.e., $d_j = s_j - s_{j-1}$ for $1 \leq j \leq g$. We use the following lemma which is based on periodic properties of maximal palindromes ending at the same position.

Lemma 2.2.

(i) *For any $1 \leq j < g$, $d_{j+1} \geq d_j$.*

(ii) *For any $1 < j < g$, if $d_{j+1} \neq d_j$, then $d_{j+1} \geq d_j + d_{j-1}$.*

(iii) *$\mathbf{S}_i$ can be represented by $O(\log i)$ arithmetic progressions, where each arithmetic progression is a tuple $\langle s, d, t \rangle$ representing the sequence $s, s + d, \ldots, s + (t - 1)d$ with common difference d.*

(iv) *If $t \geq 2$, then the common difference d is a period of every maximal palindrome which ends at position i in T and whose length belongs to the arithmetic progression $\langle s, d, t \rangle$.*

Each arithmetic progression $\langle s, d, t \rangle$ is called a *group* of maximal palindromes. See also Figure 2.1 for a concrete example.

Similar arguments hold for the set $\mathit{MaxPalBeg}_T(i)$ of maximal palindromes of T that begin at position i. To prove Lemma 2.2, we use arguments from the literature [9, 45, 90]. Let us for now consider any string W of length m. In what follows we will focus on suffix palindromes in $\mathit{SufPals}(W)$ and discuss their useful properties. We remark that symmetric arguments hold for prefix palindromes in $\mathit{PrePals}(W)$ as well. Let $\mathbf{S}' = s'_1, \ldots, s'_{g'}$ be the sequence of lengths of

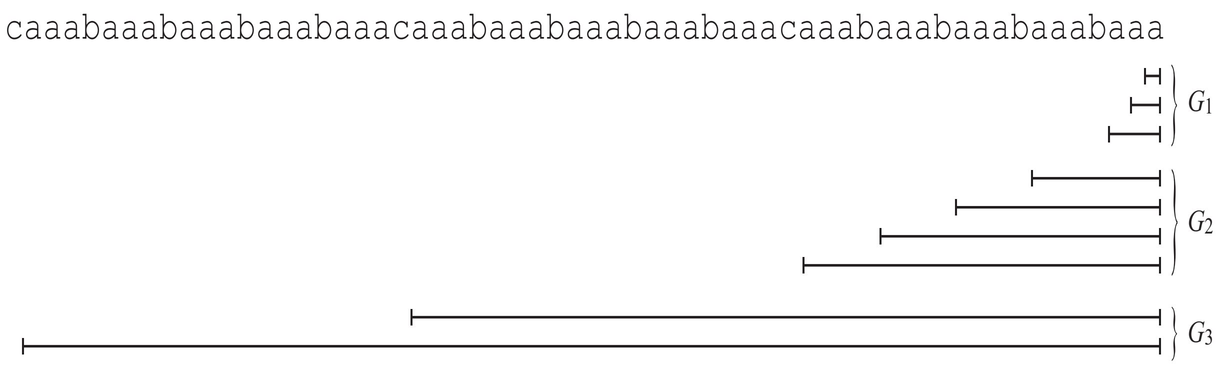

caaabaaabaaabaaabaaacaaabaaabaaabaaabaaacaaabaaabaaabaaabaaa
G_1
G_2
G_3

Figure 2.1: Examples of arithmetic progressions representing the maximal palindromes of a string. The first group G_1 is represented by $\langle 1, 1, 3\rangle$, the second group G_2 by $\langle 7, 4, 4\rangle$, and the third group G_3 by $\langle 39, 20, 2\rangle$.

suffix palindromes of $\mathbf{S}'$ sorted in increasing order, where $g' = |\mathit{SufPals}(W)|$. Also, let $s'_0 = \varepsilon$. Let d'_j be the progression difference for s'_j, i.e., $d'_j = s'_j - s'_{j-1}$ for $1 \leq j \leq g'$. Then, the following results are known:

Lemma 2.3 ([9, 45, 90])**.**

(A) For any $1 \leq j < g'$, $d'_{j+1} \geq d'_j$.

(B) For any $1 < j < g'$, *if* $d'_{j+1} \neq d'_j$, *then* $d'_{j+1} \geq d'_j + d'_{j-1}$.

(C) $\mathbf{S}'$ *can be represented by* $O(\log m)$ *arithmetic progressions, where each arithmetic progression is a tuple* $\langle s', d', t'\rangle$ *representing the sequence* $s', s' + d', \ldots, s' + (t' - 1)d'$ *of lengths of* t' *suffix palindromes with common difference* d'.

(D) If $t' \geq 2$, *then the common difference* d' *is a period of every suffix palindrome of* W *whose length belongs to the arithmetic progression* $\langle s', d', t'\rangle$.

The set of suffix palindromes of W whose lengths belong to the same arithmetic progression $\langle s', d', t'\rangle$ is also called a *group* of suffix palindromes. Clearly, every suffix palindrome in the same group has period d', and this periodicity will play a central role in our algorithms.

We are ready to prove Lemma 2.2.

Proof. It is clear that $\mathit{MaxPalEnd}_T(i) \subseteq \mathit{SufPals}(T[1..i])$, namely,

$$\mathit{MaxPalEnd}_T(i) = \{s' \in \mathit{SufPals}(T[1..i]) \mid T[i - s'] \neq T[i + 1], i - s' = 1, \text{ or } i = n\}.$$

The case where $i = n$ is trivial, and hence in what follows suppose that $i < n$. Let $c = T[i + 1]$, and for a group $\langle s', d', t'\rangle$ of suffix palindromes let $a = T[i - s']$ and $b = T[i -$

$s' - (t'-1)d']$, namely, a (resp. b) is the character that immediately precedes the shortest (resp. longest) palindrome in the group (notice that $a = b$ when $t' = 1$). Then, it follows from Lemma 2.3 (D) that $s', s'+d', \ldots, s'+(t'-2)d' \in \mathit{MaxPalEnd}_T(i)$ iff $a \neq c$. Also, $s'+(t'-1)d' \in \mathit{MaxPalEnd}_T(i)$ iff $b \neq c$. Therefore, for each group of suffix palindromes of $T[1..i]$, there are only four possible cases: (1) all members of the group are in $\mathit{MaxPalEnd}_T(i)$, (2) all members but the longest one are in $\mathit{MaxPalEnd}_T(i)$, (3) only the longest member is in $\mathit{MaxPalEnd}_T(i)$, or (4) none of the members is in $\mathit{MaxPalEnd}_T(i)$.

Now, it immediately follows from Lemma 2.3 that (i) $d_{j+1} \geq d_j$ for $1 \leq j < g$ and (ii) $d_{j+1} \geq d_j + d_{j-1}$ holds for $1 < j < g$. Properties (iii) and (iv) also follow from the above arguments and Lemma 2.3. □

For all $1 \leq i \leq n$ we can compute $\mathit{MaxPalEnd}_T(i)$ and $\mathit{MaxPalBeg}_T(i)$ in total $O(n)$ time: After computing all maximal palindromes of T in $O(n)$ time, we can bucket sort all the maximal palindromes with their ending positions and with their beginning positions in $O(n)$ time each.

Also, by using lemmas in [9, 45, 90], the following corollary can be proven immediately:

Corollary 2.1. *For a position i, divide the set of palindromic suffixes of $T[1..i]$ into $O(\log m)$ arithmetic progressions. Also, for each group $\mathcal{G}_k = \langle s'_k, d'_k, t'_k \rangle$, the following properties hold:*

1. *The difference between centers of any two palindromes in $\mathcal{G}_k$ is an integer power of $0.5d'_k$, where d'_k is their common difference.*

2. *For all maximal palindromes $e_1, \ldots, e_{t'}$ in T that are expansions of palindromes in $\mathcal{G}_k$, excluding at most one, their smallest period is also d'_k.*

3. *Among $e_1, \ldots, e_{t'}$, any palindrome is contained by the longest one or the second longest one.*

We remark that symmetric arguments hold for palindromic prefixes as well.

The following lemma states that the total sum of occurrences of strings which are extended arms of MUPSs is $O(n)$:

Lemma 2.4. *The total sum of occurrences of the extended right arms of all MUPSs in a string T is at most $2n$. Similarly, the total sum of occurrences of the extended left arms of all MUPSs in T is at most $2n$.*

Proof. It suffices to prove the former statement for the extended *right* arms since the latter can be proved symmetrically. Let w_1 and w_2 be distinct odd-length MUPSs of T with $|w_1| \leq |w_2|$. For the sake of contradiction, we assume that $T[j..j + |\mathsf{Rarm}_{w_1}| - 1] = \mathsf{Rarm}_{w_1}$ and $T[j..j + |\mathsf{Rarm}_{w_2}| - 1] = \mathsf{Rarm}_{w_2}$ for some position j in T. Namely, Rarm_{w_1} is a prefix of Rarm_{w_2}. Then, larm_{w_1} is a suffix of larm_{w_2} by palindromic symmetry. This means that w_1 is a substring of w_2. This contradicts that w_2 is a MUPS of T. Thus, all occurrences of the extended right arms of all odd-length MUPSs are different, i.e., the total number of the occurrences is at most n. Similarly, the total number of all occurrences of the right arms of all even-length MUPSs is also at most n. □

2.4 Algorithmic Tools

This section lists some data structures used in our algorithms.

Suffix Trees

The *suffix tree* of T is the compacted trie for all suffixes of T [120]. We denote by $\mathsf{STree}(T)$ the suffix tree of T. If a given string T is over an integer alphabet of size $n^{O(1)}$, $\mathsf{STree}(T)$ can be constructed in $O(n)$ time [34]. Not all substrings of T correspond to nodes in $\mathsf{STree}(T)$. However, the loci of such substrings can be made explicit in linear time:

Lemma 2.5 (Corollary 8.1 in [73])**.** *Given m substrings of T, represented by intervals in T, we can compute the locus of each substring in $\mathsf{STree}(T)$ in $O(n+m)$ total time. Moreover, the loci of all the substrings in $\mathsf{STree}(T)$ can be made explicit in $O(n + m)$ extra time.*

Also, this lemma implies the following corollary:

Corollary 2.2. *Given m substrings of T, represented by intervals in T, we can sort them in $O(n + m)$ time.*

LCE Queries

An *LCE query* on a string T is, given two indices i, j of T, to compute $\mathsf{RightLCE}_T(i, j)$. Using $\mathsf{STree}(T\$)$ enhanced with a lowest common ancestor data structure, we can answer any LCE query on T in constant time where \$ is a special character with $\$ \notin \Sigma$. In the same way, we can compute the lcp value between any two suffixes of T or T^R in constant time by using $\mathsf{STree}(T\$T^R\#)$ where # is another special character with $\# \notin \Sigma$.

NCA Queries

A *nearest colored ancestor query* (NCA query) on a tree $\mathcal{T}$ with *colored* nodes is, given a query node v and a color C, to compute the nearest ancestor u of v such that the color of u is C. Noticing that the notion of NCA is a generalization of well-known *nearest marked ancestor*. For NCA queries, we will use the following known results:

Lemma 2.6 ([52]). *Given a tree $\mathcal{T}$ with colored nodes, a data structure of size $O(N)$ can be constructed in deterministic $O(N \log\log N)$ time or expected $O(N)$ time to answer any NCA query in $O(\log\log N)$ time, where N is the number of nodes of $\mathcal{T}$.*

Lemma 2.7 ([21, 27]). *If the number of colors is $O(\log N)$, a data structure of size $O(N)$ can be constructed in $O(N)$ time to answer any NCA query in $O(1)$ time.*

Eertrees

The *eertree* (a.k.a. palindromic tree) of T is a pair of rooted edge-labeled trees $\mathcal{T}_{\mathsf{odd}}$ and $\mathcal{T}_{\mathsf{even}}$ representing all distinct palindromes in T [109]. The roots of $\mathcal{T}_{\mathsf{odd}}$ and $\mathcal{T}_{\mathsf{even}}$ represent ε. Each non-root node of $\mathcal{T}_{\mathsf{odd}}$ (resp. $\mathcal{T}_{\mathsf{even}}$) represents an odd-palindrome (resp. even-palindrome) which occurs in T. Let $pal(v)$ be the palindrome represented by a node v. For the root r_{odd} of $\mathcal{T}_{\mathsf{odd}}$, there is an edge (r_{odd}, u) labeled by $a \in \Sigma$ if there is a node u with $pal(u) = a$. For any node v in the eertree except for r_{odd}, there is an edge (v, w) labeled by $a \in \Sigma$ if there is a node w with $pal(w) = a \cdot pal(v) \cdot a$. We denote by $\mathsf{EERTREE}(T)$ the eertree of T. We will sometimes identify a node u in $\mathsf{EERTREE}(T)$ with its corresponding palindrome $pal(u)$. Also, the path from a node u to a node v in $\mathsf{EERTREE}(T)$ is denoted by $pal(u) \rightsquigarrow pal(v)$. If a given string T is over an integer alphabet of size $n^{O(1)}$, $\mathsf{EERTREE}(T)$ can be constructed in $O(n)$ time [109].

Path-Tree LCE Queries

A *path-tree LCE query* is a generalized LCE query on a rooted edge-labeled tree $\mathcal{T}$ [19]: Given three nodes u, v, and w in $\mathcal{T}$ where u is an ancestor of v, to compute the lcp between the path-string from u to v and any path-string from w to a descendant leaf. The following result is known:

Theorem 2.1 (Theorem 2 of [19]). *For a tree $\mathcal{T}$ with N nodes, a data structure of size $O(N)$ can be constructed in $O(N)$ time to answer any path–tree LCE query in $O((\log\log N)^2)$ time.*

We will use later path-tree LCE queries on the eertree of the input string.

Stabbing Queries

Let $\mathcal{I}$ be a set of n intervals, each of which is a subinterval of the universe $U = [1, O(n)]$. An *interval stabbing query* on $\mathcal{I}$ is, given a query point $q \in U$, to report all intervals $I \in \mathcal{I}$ such that I is *stabbed* by q, i.e., $q \in I$. We can answer such a query in $O(1+k)$ time after $O(n)$-time preprocessing, where k is the number of intervals to report [112].

2.5 Tries

A *trie* $\mathcal{T} = (V, E)$ is a rooted tree where each edge in E is labeled by a single character from Σ and the out-going edges of a node are labeled by pairwise distinct characters. For any non-root node u in $\mathcal{T}$, let $\mathsf{parent}(u)$ denote the parent of u. For any node v in $\mathcal{T}$, let $\mathsf{children}(v)$ denote the set of children of v. For any node u and its arbitrary descendant v, we denote by $\mathsf{str}(u, v)$ the substring of $\mathcal{T}$ that begins at u and ends at v.

A trie can be seen as a representation of a set of strings which are root-to-leaf path labels. Note that for a trie with N edges, the total length of such strings can be quadratic in N. An example can be given by the set of strings $X = \{xc_1, xc_2, \cdots xc_N\}$ where $x \in \Sigma^{N-1}$ is an arbitrary string and $c_1, \ldots, c_N \in \Sigma$ are pairwise distinct characters. Here, the size of the trie is $\Theta(N)$, while the total length of strings is $\Theta(N^2)$. Also notice that the total number of distinct suffixes of strings in X is also $\Theta(N^2)$. However if we consider the strings in the reverse direction, i.e., consider edges of the trie to be directed toward the root, the number of distinct suffixes is linear in the size N of the trie. We call it a reversed trie.

Consider a trie with N edges such that the root has a single out-edge labeled with a special character \$ that does not appear elsewhere in the trie and is lexicographically the smallest. We consider the reversed trie of this trie. The *suffix array* of this reversed trie can be constructed in $O(N)$ time [36, 113]. Also, the *longest common prefix array* (*LCP array*) for this suffix array can also be constructed in $O(N)$ time [72].

Chapter 3

Sensitivity of String Compressors and Repetitiveness Measures

In this chapter, we analyze changes of the size of string compressors and repetitiveness measures in dynamic strings. We present the worst-case sensitivity of string compressors and repetitiveness measures: Section 3.1 deals with the substring complexity δ; Section 3.2 deals with the smallest string attractor γ, Section 3.3 deals with the RLBWT r, Section 3.4 deals with the smallest bidirectional scheme b, Section 3.5 deals with the LZ77 with/without self-references z_{77} and $z_{77\mathrm{sr}}$; Section 3.6 deals with the LZSS with/without self-references z_{SS} and z_{SSsr}; Section 3.7 deals with the LZ78 z_{78}.

The results in this chapter primarily appeared in [2].

3.1 Substring Complexity

In this section, we consider the worst-case sensitivity of the string repetitiveness measure δ, which is the *substring complexity* of strings [74]. For any string T of length n, the substring complexity $\delta(T)$ is defined as $\delta(T) = \max_{1 \leq k \leq n} (\mathsf{Substr}(T, k)/k)$, where $\mathsf{Substr}(T, k)$ is the number of distinct substrings of length k in T. It is known that $\delta(T) \leq \gamma(T)$ holds for any T [74].

In what follows, we present tight upper and lower bounds for the multiplicative sensitivity of δ for all cases of substitutions, insertions, and deletions. We also present the additive sensitivity of δ.

3.1.1 Lower Bounds for Sensitivity of δ

Theorem 3.1. *The following lower bounds on the sensitivity of δ hold:*
substitutions: $\mathsf{MS}_{\mathrm{sub}}(\delta, n) \geq 2$. $\mathsf{AS}_{\mathrm{sub}}(\delta, n) \geq 1$.
insertions: $\mathsf{MS}_{\mathrm{ins}}(\delta, n) \geq 2$. $\mathsf{AS}_{\mathrm{ins}}(\delta, n) \geq 1$.
deletions: $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(\delta, n) \geq 1.5$. $\liminf_{n\to\infty} \mathsf{AS}_{\mathrm{del}}(\delta, n) \geq 1$.

Proof. **substitutions**: Consider strings $T = \mathtt{a}^n$ and $T' = \mathtt{a}^{n-1}\mathtt{b}$. Then $\delta(T) = 1$ and $\delta(T') = 2$ hold. Thus we get $\mathsf{MS}_{\mathrm{sub}}(\delta, n) \geq 2$ and $\mathsf{AS}_{\mathrm{sub}}(\delta, n) \geq 1$.

insertions: Consider strings $T = \mathtt{a}^n$ and $T' = \mathtt{a}^n\mathtt{b}$. Then $\delta(T) = 1$ and $\delta(T') = 2$ hold. Thus we get $\mathsf{MS}_{\mathrm{ins}}(\delta, n) \geq 2$ and $\mathsf{AS}_{\mathrm{ins}}(\delta, n) \geq 1$.

deletions: Consider string

$$T = (\mathtt{abb})^m\mathtt{a}(\mathtt{bba})^{m+1}\mathtt{a}^{3m}(\mathtt{bba})^m$$

with a positive integer m. Let $n = 12m+4 = |T|$. For the sake of exposition, let $w_1 = (\mathtt{abb})^m$, $w_2 = (\mathtt{bba})^{m+1}$, $w_3 = \mathtt{a}^{3m}$, and $w_4 = (\mathtt{bba})^m$ such that $T = w_1\mathtt{a}w_2w_3w_4$. To analyze $\delta(T)$, we consider $\mathsf{Substr}(T, k)$ for four different groups of k, as follows:

- For $1 \leq k \leq 2$: Since T is a binary string, $\max_{1\leq k\leq 2}(\mathsf{Substr}(T,k)/k) = 2$.
- For $3 \leq k \leq 3m$: The prefix $w_1\mathtt{a}w_2 = (\mathtt{abb})^m\mathtt{a}(\mathtt{bba})^{m+1}$ and the suffix $w_4 = (\mathtt{bba})^m$ contain three distinct substrings $(\mathtt{abb})^{k/3}$, $(\mathtt{bba})^{k/3}$, and $(\mathtt{bab})^{k/3}$ for each length k, and the substring $w_3 = \mathtt{a}^{3m}$ contains a unique substring $\mathtt{a}^k$ for each length k. The remaining distinct substrings must contain the range $[6m+4, 6m+5]$ or $[9m+4, 9m+5]$, which are the left and right boundaries of w_3, respectively. There are $k-1$ distinct substrings containing $[6m+4, 6m+5]$ of form:

$$\begin{array}{ll} (\mathtt{bba})^{l_1}\mathtt{a}^{k-3l_1} & \text{for } 1 \leq l_1 \leq \lfloor (k-1)/3 \rfloor; \\ \mathtt{a}(\mathtt{bba})^{l_2-1}\mathtt{a}^{k-3l_2+2} & \text{for } 1 \leq l_2 \leq \lfloor (k+1)/3 \rfloor; \\ \mathtt{ba}(\mathtt{bba})^{l_3-1}\mathtt{a}^{k-3l_3+1} & \text{for } 1 \leq l_3 \leq \lfloor k/3 \rfloor. \end{array}$$

  Also, there are $k-1$ distinct substrings containing $[9m+4, 9m+5]$ of form

$$\mathtt{a}^{k-l_4}(\mathtt{bba})^{l_4/3} \quad \text{for } 1 \leq l_4 \leq k-1.$$

  Notice however that the two substrings $\mathtt{a}(\mathtt{bba})^{l_2-1}\mathtt{a}^{k-3l_2+2} = \mathtt{a}^k$ with $l_2 = 1$ and $\mathtt{a}^{k-l_4}(\mathtt{bba})^{l_4/3} = (\mathtt{abb})^{k/3}$ with $l_4 = k-1$ have already been counted in the other positions in T, and thus these duplicates should be removed. Summing up all these, we obtain $\mathsf{Substr}(T,k) = 3+1+2(k-1)-2 = 2k$ for every $3 \leq k \leq 3m$, implying $\max_{3\leq k\leq 3m}(\mathsf{Substr}(T,k)/k) = 2$.

- For $3m < k \leq n$: The prefix $w_1 \mathtt{a} w_2$ contains at most three distinct substrings for every k and the substrings w_3 and w_4 contain no substrings of length $k > 3m$. The remaining distinct substrings must again contain the positions in $[6m+4, 6m+5]$ or $[9m+4, 9m+5]$. These substrings can also be described in a similar way to the previous case for $3 \leq k \leq 3m$, except for how we should remove duplicates. We have the two following sub-cases:
  - For $k = 3m+1$: Since $\mathtt{a}^k = \mathtt{a}^{3m+1}$ has no occurrences in T but $(\mathtt{abb})^{k/3}$ has other occurrences and it has already been counted, the number of such distinct substrings is at most $2(k-1)-1$.
  - For $k > 3m+1$: There exists at least one substring which contains both $[6m+4, 6m+5]$ and $[9m+4, 9m+5]$. Therefore, the number of such distinct substrings is at most $2(k-1)-1$.

  Hence, $\mathsf{Substr}(T, k) \leq 3 + 2(k-1) - 1 = 2k$ for every $3m < k \leq n$, implying $\max_{3m<k\leq n}(\mathsf{Substr}(T,k)/k) \leq 2$.

Consequently, we have that $\delta(T) = 2$.

Consider the string

$$T' = (\mathtt{abb})^m (\mathtt{bba})^{m+1} \mathtt{a}^{3m} (\mathtt{bba})^m = w_1 w_2 w_3 w_4$$

that can be obtained from T by removing $T[3m+1] = \mathtt{a}$ between w_1 and w_2. We consider the number of distinct substrings of length $3m+1$ in T': Because of the lengths of w_j with $j \in \{1,2,3,4\}$, each substring of length $3m+1$ is completely contained in w_2 or it contains some boundaries of w_j.

- The prefix $w_1(w_2[1..|w_2|-3]) = (\mathtt{abb})^m(\mathtt{bba})^m$ contains $3m$ distinct substrings of length $3m+1$.
- The substring w_2 contains 3 distinct substrings of length $3m+1$.
- The substring $w_2[4..|w_2|]w_3 = (\mathtt{bba})^m \mathtt{a}^{3m}$ contains $3m$ distinct substrings of length $3m+1$.
- The suffix $w_3 w_4 = \mathtt{a}^{3m}(\mathtt{bba})^m$ contains $3m-1$ distinct substrings of length $3m+1$ (note that $\mathtt{a}(\mathtt{bba})^m$ is a duplicate and is not counted here).

Hence,

$$\delta(T') \geq \mathsf{Substr}(T, 3m+1)/(3m+1) = \frac{9m+2}{3m+1} = 3 - \frac{1}{3m+1}.$$

Thus we obtain $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(\delta, n) \geq \liminf_{m\to\infty}((3 - 1/(3m+1))/2) \geq 1.5$ and $\liminf_{n\to\infty} \mathsf{AS}_{\mathrm{del}}(\delta, n) \geq \liminf_{m\to\infty}((3 - 1/(3m+1)) - 2) = 1$. □

3.1.2 Upper Bounds for Sensitivity of δ

Theorem 3.2. *The following upper bounds on the sensitivity of δ hold:*
substitutions: $\mathsf{MS}_{\mathrm{sub}}(\delta, n) \leq 2$. $\mathsf{AS}_{\mathrm{sub}}(\delta, n) \leq 1$.
insertions: $\mathsf{MS}_{\mathrm{ins}}(\delta, n) \leq 2$. $\mathsf{AS}_{\mathrm{ins}}(\delta, n) \leq 1$.
deletions: $\limsup_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(\delta, n) \leq 1.5$. $\limsup_{n\to\infty} \mathsf{AS}_{\mathrm{del}}(\delta, n) \leq 1$.

Proof. First we consider the additive sensitivity for δ. For each k, the number of substrings of length k that contains the edited position i is clearly at most k. Therefore, after a substitution or insertion, at most k new distinct substrings of length k can appear in the string T' after the modification. Also, after a deletion, at most $k-1$ new distinct substrings of length k can appear in T'. Hence, in the case of substitutions and insertions, $\delta(T') \leq \max_{1\leq k\leq n}((\mathsf{Substr}(T,k)+k)/k) \leq \max_{1\leq k\leq n}(\mathsf{Substr}(T,k))/k) + \max_{1\leq k\leq n}(k/k) = \delta(T)+1$ holds. Also, in the case of deletions, $\delta(T') \leq \max_{1\leq k\leq n}((\mathsf{Substr}(T,k)+k-1)/k) \leq \delta(T) + \max_{1\leq k\leq n}((k-1)/k)$ holds. Thus we obtain $\mathsf{AS}_{\mathrm{sub}}(\delta, n) \leq 1$, $\mathsf{AS}_{\mathrm{ins}}(\delta, n) \leq 1$, and $\limsup_{n\to\infty} \mathsf{AS}_{\mathrm{del}}(\delta, n) \leq \limsup_{k\to\infty}(k-1)/k = 1$.

Next we consider the multiplicative sensitivity for δ. Note that $\delta(T') \geq 1$ for any non-empty string T', since $\mathsf{Substr}(T', 1) \geq 1$. Combining this with the afore-mentioned additive sensitivity, we obtain $\mathsf{MS}_{\mathrm{sub}}(\delta, n) \leq 2$ and $\mathsf{MS}_{\mathrm{ins}}(\delta, n) \leq 2$. For the case of deletions, observe that $\delta(T) = 1$ only if T is a unary string. However $\delta(T')$ cannot increase after a deletion since T' is also a unary string. Thus we can restrict ourselves to the case where T contains at least two distinct characters. Then, we have $\limsup_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(\delta, n) \leq 1.5$, which is achieved when $\delta(T) = 2$ and $\delta(T') = 2 + \frac{k-1}{k}$ with $k \to \infty$. □

3.2 String Attractors

In this section, we consider the worst-case sensitivity of the string repetitiveness measure γ, which is the size of the smallest string attractor [69]. A string attractor $\Gamma(T)$ for a string T is a set of positions in T such that any substring T has an occurrence containing a position in $\Gamma(T)$. We denote the size of the smallest string attractor of T by $\gamma(T)$. It is known that $\gamma(T)$ is upper bounded by any of $z_{77}(T)$, $r(T)$, $e(T)$ for any string T [69].

In what follows, we present lower bounds for the multiplicative sensitivity of γ for all cases of substitutions, insertions, and deletions. We also present the additive sensitivity of γ.

3.2.1 Lower Bounds for Sensitivity of γ

Theorem 3.3. *The following lower bounds on the sensitivity of γ hold:*
substitutions: $\liminf_{n\to\infty} \mathsf{MS}_{\text{sub}}(\gamma, n) \geq 2$. $\mathsf{AS}_{\text{sub}}(\gamma, n) \geq \gamma - 2$ *and* $\mathsf{AS}_{\text{sub}}(\gamma, n) = \Omega(\sqrt{n})$.
insertions: $\liminf_{n\to\infty} \mathsf{MS}_{\text{ins}}(\gamma, n) \geq 2$. $\mathsf{AS}_{\text{ins}}(\gamma, n) \geq \gamma - 2$ *and* $\mathsf{AS}_{\text{ins}}(\gamma, n) = \Omega(\sqrt{n})$.
deletions: $\liminf_{n\to\infty} \mathsf{MS}_{\text{del}}(\gamma, n) \geq 2$. $\mathsf{AS}_{\text{del}}(\gamma, n) \geq \gamma - 3$ *and* $\mathsf{AS}_{\text{del}}(\gamma, n) = \Omega(\sqrt{n})$.

Proof. Consider string $T = \mathtt{a}^k\mathtt{x}\mathtt{a}^{k+1}\#_1\mathtt{a}^{k-1}\mathtt{x}\mathtt{a}\#_2\mathtt{a}^{k-2}\mathtt{x}\mathtt{a}^2\#_3\cdots\#_k\mathtt{x}\mathtt{a}^k$, where $\#_j$ for every $1 \leq j \leq k$ is a distinct character. The position where $\#_j$ for each $1 \leq j \leq k$ occurs has to be an element of any string attractor for T. Also, each of the intervals $[1, k+1]$ and $[k+2, 2k+2]$ has to contain at least one element of any string attractor for T, since each of the substrings $T[1..k+1] = \mathtt{a}^k\mathtt{x}$ and $T[k+2..2k+2] = \mathtt{a}^{k+1}$ occurs only once in T. Therefore, $\gamma(T) \geq k+2$ holds. Consider the set $S = \{k+1, k+2, 2k+3, 3k+5, \dots, k+1+k(k+2)\}$ of $k+2$ positions in T which contains all the positions required above. Since each substring $\mathtt{a}^{k-j}\mathtt{x}\mathtt{a}^j$ of length $k+1$ immediately preceded by $\#_j$ $(1 \leq j \leq k)$ occurs in the prefix $\mathtt{a}^k\mathtt{x}\mathtt{a}^{k+1}$ and contains the position $k+1$, S is indeed a string attractor for T, we get $\gamma(T) = k+2$. In the following, we use this string T for the analysis of lower bounds for the sensitivity of γ.

substitutions: Let T' be the string obtained by substituting the leftmost occurrence of $\mathtt{x}$ at position $k+1$ in T with character $\mathtt{b}$, yielding the new prefix $\mathtt{a}^k\mathtt{b}\mathtt{a}^{k+1}$ right before $\#_1$. The size of the smallest string attractor for T' is as follows: Each occurrence position of $\#_j$ for $1 \leq j \leq k$ still has to be an element of any string attractor for T'. Also, each of the intervals $[k+1]$ and $[k+2, 2k+2]$ has to contain at least one element of any string attractor for T'. In addition, each of the intervals $[2k+4, 3k+4], [3k+6, 4k+6], \dots, [k+2+k(k+2), 2k+2+k(k+2)]$ which are the occurrences of substrings $\mathtt{a}^{k-1}\mathtt{x}\mathtt{a}, \mathtt{a}^{k-2}\mathtt{x}\mathtt{a}^2, \dots, \mathtt{x}\mathtt{a}^k$ has to contain one string attractor, since we have lost the prefix $\mathtt{a}^k\mathtt{x}\mathtt{a}^{k+1}$. Therefore, $\gamma(T') \geq 2k+2$ holds and the set $\{k+1, k+2, 2k+3, 3k+5, \dots, k+1+k(k+2), 2k+4, 3k+6, \dots, k+2+k(k+2)\}$ of $2k+2$ positions in T' is a string attractor for T', implying $\gamma(T') = 2k+2$. Thus we get $\liminf_{n\to\infty} \mathsf{MS}_{\text{sub}}(\gamma, n) \geq \liminf_{k\to\infty}(2k+2)/(k+2) = 2$ and $\mathsf{AS}_{\text{sub}}(\gamma, n) \geq \gamma - 2$. Since $n = k^2 + 4k + 2$ and $\gamma(T) = k+2$, $\mathsf{AS}_{\text{sub}}(\gamma, n) = \Omega(\sqrt{n})$ holds.

insertions: Let T' be the string obtained by inserting $\mathtt{b}$ between $T[k+1] = \mathtt{x}$ and $T[k+2] = \mathtt{a}$, yielding the new prefix $\mathtt{a}^k\mathtt{x}\mathtt{b}\mathtt{a}^{k+1}$ right before $\#_1$. The size of the smallest string attractor for T' is as follows, using a similar argument to the case of substitutions: Each occurrence position of $\#_j$ for $1 \leq j \leq k$ still has to be an element of any string attractor for T'. Also, each of the intervals $[k+2]$ and $[k+3, 2k+3]$ have to contain at least one element of any string attractor for T'. In addition, each of the intervals $[2k+5, 3k+5], [3k+7, 4k+7], \ldots, [k+3+k(k+2), 2k+3+k(k+2)]$ which are the occurrences of substrings $(\mathtt{a}^{k-1}\mathtt{xa}), (\mathtt{a}^{k-2}\mathtt{xa}^2), \ldots, (\mathtt{xa}^k)$ have to contain one string attractor. Therefore, $\gamma(T') \geq 2k+2$ holds and the set $\{k+2, k+3, 2k+4, 3k+6, \ldots, k+2+k(k+2), 2k+5, 3k+7, \ldots, k+3+k(k+2)\}$ achieves $\gamma(T') = 2k+2$. Thus we get $\liminf_{n\to\infty} \mathsf{MS}_{\text{ins}}(\gamma, n) \geq \liminf_{k\to\infty}(2k+2)/(k+2) = 2$, $\mathsf{AS}_{\text{ins}}(\gamma, n) \geq \gamma - 3$, and $\mathsf{AS}_{\text{ins}}(\gamma, n) = \Omega(\sqrt{n})$.

deletions: Let T' be the string obtained by deleting $T[k+1] = \mathtt{x}$ from T, yielding the new prefix $\mathtt{a}^{2k+1}$ right before $\#_1$. The size of the smallest string attractor for T' is as follows, using a similar argument to the cases of insertions and substitutions: Each occurrence position of $\#_j$ for $1 \leq j \leq k$ still has to be an element of any string attractor for T'. Also, the interval $[1, 2k+1]$ has to contain one element of any string attractor for T'. In addition, each of the intervals $[2k+3, 3k+3], [3k+5, 4k+5], \ldots, [k+1+k(k+2), 2k+1+k(k+2)]$ has to contain one string attractor for T'. Therefore, $\gamma(T') \geq 2k+1$ holds and the set $\{1, 2k+2, 3k+4, \ldots, k+k(k+2), 2k+3, 3k+5, \ldots, k+1+k(k+2)\}$ achieves $\gamma(T') = 2k+1$. Thus we get $\liminf_{n\to\infty} \mathsf{MS}_{\text{del}}(\gamma, n) \geq \liminf_{k\to\infty}(2k+1)(k+2) = 2$, $\mathsf{AS}_{\text{del}}(\gamma, n) \geq \gamma - 3$, and $\mathsf{AS}_{\text{del}}(\gamma, n) = \Omega(\sqrt{n})$. □

3.3 Run-Length Burrows-Wheeler Transform (RLBWT)

The *Burrows-Wheeler transform* (*BWT*) of a string T, denoted $\mathsf{BWT}(T)$, is the string obtained by concatenating the last characters of the lexicographically sorted suffixes of T. The *run-length BWT* (*RLBWT*) of T is the run-length encoding of $\mathsf{BWT}(T)$ and $r(T)$ denotes its size, i.e., the number of maximal character runs in $\mathsf{BWT}(T)$.

For example, for string $T = \mathtt{abbaabababab}$, $r(T) = 4$ since $\mathsf{BWT}(T) = \mathtt{babbbbbaaaaa}$ consists in four maximal character runs $\mathtt{b}^1\mathtt{a}^1\mathtt{b}^5\mathtt{a}^5$.

Theorem 3.4 (Theorem 1 of [54]). *There exists a family of strings S such that $r(S) = 2$ and $r(S') = \Theta(\log n)$, where $n = |S|$ and S' is a string obtained by prepending a character to S. The string S is the reversed Fibonacci word.*

Theorem 3.4 immediately leads to the following lower bound for the sensitivity of r:

Corollary 3.1. *The following lower bound on the sensitivity of RLBWT with $|\Sigma| = 2$ hold:*
insertions: $\mathsf{MS}_{\text{ins}}(r, n) = \Omega(\log n)$. $\mathsf{AS}_{\text{ins}}(r, n) = \Omega(\log n)$.

To obtain a non-trivial upper bound for the sensitivity of r, we can use the following known result:

Theorem 3.5 (Theorem III.7 of [67])**.** *For any string T of length n,*

$$r(T) = O\left(\delta(T)\max\left(1, \log\frac{n}{\delta(T)\log\delta(T)}\right)\log\delta(T)\right).$$

Theorem 3.6. *The following upper bounds on the sensitivity of r hold:*
substitutions: $\mathsf{MS}_{\mathrm{sub}}(r,n) = O(\log n \log r)$. $\mathsf{AS}_{\mathrm{sub}}(r,n) = O(r\log n\log r)$.
insertions: $\mathsf{MS}_{\mathrm{ins}}(r,n) = O(\log n \log r)$. $\mathsf{AS}_{\mathrm{ins}}(r,n) = O(r\log n\log r)$.
deletions: $\mathsf{MS}_{\mathrm{del}}(r,n) = O(\log n \log r)$. $\mathsf{AS}_{\mathrm{del}}(r,n) = O(r\log n\log r)$.

Proof. For any string T, it is known that $\delta(T) \leq r(T)$ [74]. We also use a simplified and relaxed bound $r(T) = O(\delta(T)\log n\log\delta(T))$ from Theorem 3.5, which always holds and is sufficient for our purpose.

Let T' be any string with $\mathsf{ed}(T,T') = 1$. It follows from Theorem 3.2 that $\delta(T') \leq 2\delta(T)$. Therefore, we obtain $r(T') = O(\delta(T')\log n\log\delta(T')) = O(\delta(T)\log n\log\delta(T)) = O(r(T)\log n\log r(T))$. This leads to the claimed upper bounds for the sensitivity for r. □

We remark that the lower bounds $\mathsf{MS}_{\mathrm{ins}}(r,n) = \Omega(\log n)$ and $\mathsf{AS}_{\mathrm{ins}}(r,n) = \Omega(\log n)$ from Theorem 3.4 and Corollary 3.1 are asymptotically tight when $r = O(1)$, since $\mathsf{MS}_{\mathrm{ins}}(r,n) = O(\log n\log r) = O(\log n)$ and $\mathsf{AS}_{\mathrm{ins}}(r,n) = \Omega(\log n)$ in this case.

3.4 Bidirectional Scheme

In this section, we consider the worst-case sensitivity of the size of *bidirectional scheme* [115]. A factorization $T = f_1 \cdots f_b$ for a string T of length n is a bidirectional scheme of T if each phrase $f_j = T[p_j..p_j+\ell_j-1]$ is either a single character or corresponding to another substring $T[q_j..q_j+\ell_j-1]$ where $\ell_j = |f_j|$ such that $p_j \neq q_j$. We denote f_j either a single character or the pair (q_j,ℓ_j). If $|f_j| = 1$, then f_j is called a ground phrase. A bidirectional scheme B for T defines a function $F_B : [1..n]\cup\{0\} \to [1..n]\cup\{0\}$, where

$$\begin{cases} F_B(p_j) = 0, & \text{if } f_j \text{ is a ground phrase,}\\ F_B(p_j+k) = q_j+k, & \text{if } f_j = (q_j,\ell_j) \text{ and } 0 \leq k < \ell_j,\\ F_B(0) = 0. \end{cases}$$

Let $F_B^0(p_j) = p_j$ and $F_B^m(p_j) = F_B(F_B^{m-1}(p_j))$ for any $m \geq 1$. A bidirectional scheme B is called *valid* if F_B has no cycles; namely, there exists an $m \geq 1$ such that $F_B^m(x) = 0$ for every $x \in [1..n]$. The string T can be reconstructed from the bidirectional scheme if and only if it is valid. The *size* of a valid bidirectional scheme B is the number of phrases in B. We denote the size of the smallest valid bidirectional schemes of T by $b(T)$.

For example, for string $T = \mathtt{abaabababbbba}$, one of the smallest valid bidirectional schemes B is:

$$B = (4,3)(6,4)\mathtt{ab}(9,3)\mathtt{a},$$

and its corresponding factorization is:

$$B = \mathtt{aba}|\mathtt{abab}|\mathtt{a}|\mathtt{b}|\mathtt{bbb}|\mathtt{a}|.$$

Here we have $b(T) = 6$.

In what follows, we present upper and lower bounds for the multiplicative/additive sensitivity of b.

3.4.1 Lower bounds for the sensitivity of b

Theorem 3.7. *The following lower bounds on the sensitivity of* b *hold:*
substitutions: $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{sub}}(b,n) \geq 1.5$, $\mathsf{AS}_{\mathrm{sub}}(b,n) \geq b/2-1$, *and* $\mathsf{AS}_{\mathrm{sub}}(b,n) = \Omega(\sqrt{n})$.
insertions: $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{ins}}(b,n) \geq 1.5$, $\mathsf{AS}_{\mathrm{ins}}(b,n) \geq b/2-1$, *and* $\mathsf{AS}_{\mathrm{ins}}(b,n) = \Omega(\sqrt{n})$.
deletions: $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(b,n) \geq 1.5$, $\mathsf{AS}_{\mathrm{del}}(b,n) \geq b/2-3$, *and* $\mathsf{AS}_{\mathrm{del}}(b,n) = \Omega(\sqrt{n})$.

Proof. Consider string

$$T = \mathtt{a}^k\mathtt{x}\mathtt{a}^{k+1}\#_1\mathtt{a}^k\mathtt{xa}\#_2\mathtt{a}^{k-1}\mathtt{xa}^2\#_3\cdots\#_k\mathtt{axa}^k,$$

where $\#_j$ for every $1 \leq j \leq k$ is a distinct character. One of the valid bidirectional schemes B for T is

$$B = (k+2,k)\mathtt{xa}(k+2,k)\#_1(1,k+2)\#_2(2,k+2)\#_3\cdots\#_k(k,k+2).$$

The corresponding factorization of the above bidirectional scheme is as follows:

$$B = \mathtt{a}^k|\mathtt{x}|\mathtt{a}|\mathtt{a}^k|\#_1|\mathtt{a}^k\mathtt{xa}|\#_2|\mathtt{a}^{k-1}\mathtt{xa}^2|\#_3|\cdots|\#_k|\mathtt{axa}^k|.$$

The size of B is $2k+4$ and thus $b(T) \leq 2k+4$.

As for substitutions, let T' be the string obtained by substituting the leftmost occurrence of $\mathtt{x}$ at position $k+1$ in T with a character $\mathtt{y}$ such that $\mathtt{y} \neq \mathtt{x}$, that is,

$$T' = \mathtt{a}^k\mathtt{y}\mathtt{a}^{k+1}\#_1\mathtt{a}^k\mathtt{xa}\#_2\mathtt{a}^{k-1}\mathtt{xa}^2\#_3\cdots\#_k\mathtt{axa}^k.$$

Then, one of the valid bidirectional schemes B' of T' is:

$$B' = (k+2,k)\mathtt{ya}(k+2,k)\#_1(1,k)\mathtt{xa}\#_2(2k+5,k)(1,2)\#_3\cdots\#_k(3k+4,2)(1,k).$$

Also, the corresponding factorization for B is as follows:

$$B' = \mathtt{a}^k|\mathtt{y}|\mathtt{a}|\mathtt{a}^k|\#_1|\mathtt{a}^k|\mathtt{x}|\mathtt{a}|\#_2|\mathtt{a}^{k-1}\mathtt{x}|\mathtt{a}^2|\#_3|\cdots|\#_k|\mathtt{ax}|\mathtt{a}^k|.$$

The size of B' is $3k+5$. We show that B' is one of the smallest valid bidirectional schemes of T', namely, $b(T') = 3k+5$. Since $\mathtt{y}$ and $\#_j$ for every $1 \leq j \leq k$ are unique characters in T', they have to be ground phrases. Also, since each substring $\mathtt{a}^{k-j+1}\mathtt{xa}^j$ of length $k+2$ for all $1 \leq j \leq k$ and $\mathtt{a}^{k+1}$ are unique in T', each corresponding interval has to have at least one boundary of phrases. In addition, at least one occurrence of $\mathtt{x}$ has to be a ground phrase. Then, $b(T') = 3k+5$ holds. Since $|T| = n = k^2+5k+2$, we have $k = \Theta(\sqrt{n})$. Hence, we get $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{sub}}(b,n) \geq 1.5$ and $\mathsf{AS}_{\mathrm{sub}}(b,n) \geq k+1 = b/2-1 = \Omega(\sqrt{n})$.

Moreover, by considering the case where the character $T[k+1]$ is deleted and the case where the character $\mathtt{y}$ is inserted between positions $k+1$ and $k+2$, we obtain Theorem 3.7. □

3.4.2 Upper bounds for the sensitivity of b

Theorem 3.8. *The following upper bounds on the sensitivity of* b *hold:*
substitutions: $\limsup_{n\to\infty} \mathsf{MS}_{\mathrm{sub}}(b,n) \leq 2$. $\mathsf{AS}_{\mathrm{sub}}(b,n) \leq b+2$.
insertions: $\mathsf{MS}_{\mathrm{ins}}(b,n) \leq 2$. $\mathsf{AS}_{\mathrm{ins}}(b,n) \leq b$.
deletions: $\limsup_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(b,n) \leq 2$. $\mathsf{AS}_{\mathrm{del}}(b,n) \leq b+1$.

Proof. In the following, we consider the case that $T[i] = a$ is substituted by a character $\#$ that does not occur in T. The other cases of insertions, deletions, and substitutions with another character b ($\neq a$) occurring in T, can be proven similarly. We show how to construct a valid bidirectional scheme of T' of the size $b' \geq b(T')$ by dividing each phrase of B into some phrases, where B is one of the smallest valid bidirectional schemes of T. We categorize each phrase $f_j = T[p_j..p_j+\ell_j-1]$ of B into one of the three following cases:

(1) $i \in [p_j..p_j+\ell_j-1]$;

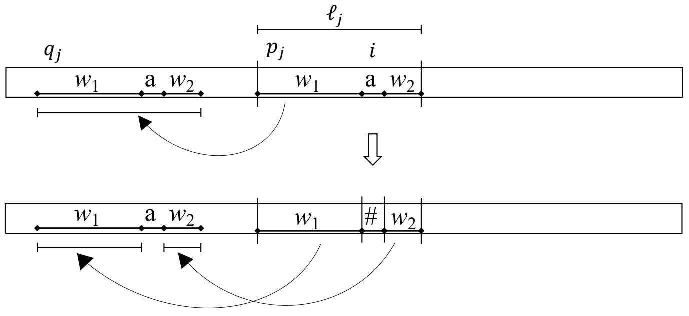
(2) $i \notin [p_j..p_j + \ell_j - 1]$ and $i \notin [q_j..q_j + \ell_j - 1]$;

(3) $i \notin [p_j..p_j + \ell_j - 1]$ and $i \in [q_j..q_j + \ell_j - 1]$.

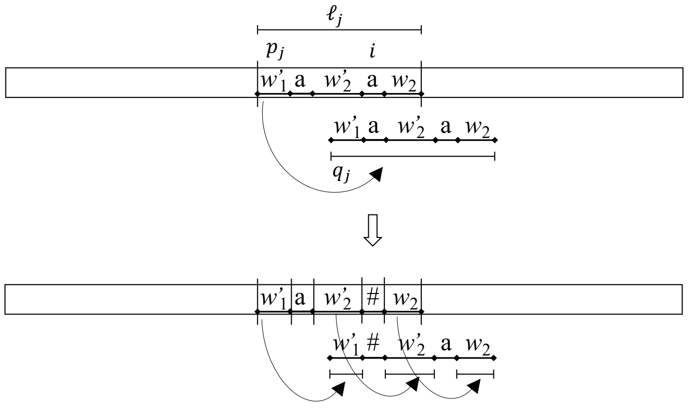
Case (1): Let $T[p_j..p_j + \ell_j - 1] = w_1 a w_2$ and $T'[p_j..p_j + \ell_j - 1] = w_1 \# w_2$, where $a \in \Sigma$ and $w_1, w_2 \in \Sigma^*$. If $i \notin [q_j..q_j + \ell_j - 1]$, then the phrase f_j is divided into three phrases $w_1 = (q_j, |w_1|), \#, w_2 = (q_j + |w_1| + 1, |w_2|)$ in T'. See also the top of Figure 3.1. Otherwise, i.e., if $i \in [q_j..q_j + \ell_j - 1]$, intervals $[p_j..p_j + \ell_j - 1]$ and $[q_j..q_j + \ell_j - 1]$ are overlapping. We consider the case $p_j < q_j$. (Another case can be treated similarly.) Then $[q_j..q_j + |w_1|]$ contains the edited position i. Let $T[p_j..p_j + \ell_j - 1] = w'_1 a w'_2 a w_2$, where $w'_1, w'_2 \in \Sigma^*$ and $q_j + |w'_1| = i$. We divide the phrase f_j into at most five phrases $w'_1 = (q_j, |w'_1|), a, w'_2 = (q_j + |w'_1| + 1, |w'_2|), \#, w_2 = (q_j + |w_1| + 1, |w_2|)$. See also the middle of Figure 3.1.
Case (2): No changes are made to the phrase f_j in this case, since f_j can continue to refer to the same reference.
Case (3): Among all phrases in Case (3), let f_k be the phrase whose ending position of the reference is the rightmost. Let $T[p_k..p_k + \ell_k - 1] = u_1 a u_2$, where $u_1, u_2 \in \Sigma^*$ and $q_k + |u_1| = i$. Then we divide the phrase f_k into at most three phrases $u_1 = (q_k, |u_1|), a, u_2 = (q_k + |u_1| + 1, |u_2|)$ in T'. For the other phrases of Case(3), we divide $f_j = v_1 a v_2$, where $v_1, v_2 \in \Sigma^*$ and $q_j + |v_1| = i$, into at most two phrases $v_1 = (q_j, |v_1|)$ and $a v_2 = (q_k + |u_1|, |v_2| + 1)$. From the above operations, the character that referred to position i in T becomes a ground phrase or refers to position $q_k + |u_1|$, which is a ground phrase, in T'. The other substrings refer to the original reference positions or to a subinterval of $[q_k + |u_1|..q_k + |f_k| - 1]$. The reference of the subinterval corresponds to the original reference of the substring. See also the bottom of Figure 3.1.

Then, the bidirectional scheme obtained from the above operations is ensured to be valid. The size of the bidirectional scheme b' is maximized if exactly one phrase of Case (1) is divided into five phrases, and the remaining $b(T) - 1$ phrases belong to Case (3). Since at most one of the $b(T) - 1$ phrases of Case (3) can be divided into three phrases, and all the others can be divided into two phrases, b' is at most $5 + 3 + 2(b(T) - 2) = 2b(T) + 4$. Furthermore, if T is a unary string, then $b(T) = 2$ and the valid bidirectional scheme of size $4 (= 2b(T))$ can be constructed easily. Otherwise, there are at least two ground phrases in T, and these phrases can not be divided into some phrases in T'. Then we get $b' \leq 2b(T) + 2$ and Theorem 3.8.

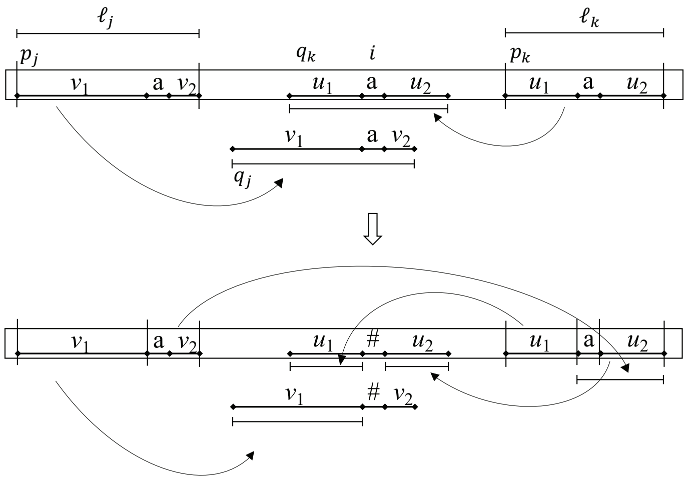
□



Subcase of Case (1): $i \in [p_j..p_j + \ell_j - 1]$ and $i \notin [q_j..q_j + \ell_j - 1]$.



Subcase of Case (1): $i \in [p_j..p_j + \ell_j - 1]$ and $i \in [q_j..q_j + \ell_j - 1]$.



Case (3): $i \notin [p_j..p_j + \ell_j - 1]$ and $i \in [q_j..q_j + \ell_j - 1]$.

Figure 3.1: Illustration for changes of references in Case (1) and Case (3).

3.5 Lempel-Ziv 77 Factorizations with/without Self-References

In this section, we consider the worst-case sensitivity of the *Lempel-Ziv 77 factorizations* (*LZ77*) [123] with/without self-references.

For convenience, let $f_0 = \varepsilon$. A factorization $f_1 \cdots f_z$ for a string T of length n is the non self-referencing LZ77 factorization $\mathsf{LZ77}(T)$ of T if for each $1 \leq i < z$ the factor f_i is the shortest prefix of $f_i \cdots f_z$ that does not occur in $f_0 f_1 \cdots f_{i-1}$ (or alternatively $f_k[1..|f_k| - 1]$ is the longest prefix of $f_i \cdots f_z$ that occurs in $f_0 f_1 \cdots f_{i-1}$). Since $f_k[1..|f_k| - 1]$ is never overlap with its previous occurrence, it is called non self-referencing. The last factor f_z is the suffix of T of length $n - |f_1 \cdots f_{z-1}|$ and it may have multiple occurrences in $f_1 \cdots f_z$.

A factorization $f_1 \cdots f_z$ for a string T of length n is the self-referencing LZ77 factorization $\mathsf{LZ77sr}(T)$ of T if for each $1 \leq i < z$ the factor f_i is the shortest prefix of $f_i \cdots f_z$ that occurs exactly once in $f_1 \cdots f_i$ as a suffix (or alternatively $f_k[1..|f_k| - 1]$ is the longest prefix of $f_i \cdots f_z$ which has a previous occurrence beginning at a position in range $[1..|f_1 \cdots f_{k-1}|]$). Since $f_k[1..|f_k| - 1]$ may overlap with its previous occurrence, it is called self-referencing. The last factor f_z is the suffix of T of length $n - |f_1 \cdots f_{z-1}|$ and it may have multiple occurrences in $f_1 \cdots f_z$.

If we use a common convention that the string T terminates with a unique character \$, then the last factor f_z satisfies the same properties as $f_1, \ldots, f_{z-1}$, in both cases of (non) self-referencing LZ77 factorizations.

To avoid confusions, we use different notations to denote the sizes of these factorizations. For a string T let $z_{77}(T)$ and $z_{77\mathrm{sr}}(T)$ denote the number z of factors in $\mathsf{LZ77}(T)$ and $\mathsf{LZ77sr}(T)$, respectively.

For example, for string $T = \mathtt{abaababababababab\$}$,

$$\begin{aligned} \mathsf{LZ77}(T) &= \mathtt{a|b|aa|bab|ababa|bab\$|}, \\ \mathsf{LZ77sr}(T) &= \mathtt{a|b|aa|bab|abababab\$|}, \end{aligned}$$

where $|$ denotes the right-end of each factor in the factorizations. Here we have $z_{77}(T) = 6$ and $z_{77\mathrm{sr}}(T) = 5$.

In what follows, we present tight upper and lower bounds for the multiplicative sensitivity of z_{77} and $z_{77\mathrm{sr}}$ for all cases of substitutions, insertions, and deletions. We also present the additive sensitivity of z_{77} and $z_{77\mathrm{sr}}$.

3.5.1 Lower Bounds for Sensitivity of z_{77}

Theorem 3.9. *The following lower bounds on the sensitivity of non self-referencing LZ77 factorization hold:*
substitutions: $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{sub}}(z_{77}, n) \geq 2$. $\mathsf{AS}_{\mathrm{sub}}(z_{77}, n) \geq z_{77} - 1$.
insertions: $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{ins}}(z_{77}, n) \geq 2$. $\mathsf{AS}_{\mathrm{ins}}(z_{77}, n) \geq z_{77} - 1$.
deletions: $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(z_{77}, n) \geq 2$. $\mathsf{AS}_{\mathrm{del}}(z_{77}, n) \geq z_{77} - 2$.

Proof. Let $p \geq 2$ and $\Sigma = \{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$. We use the following string T for our analysis in all cases of substitutions, insertions, and deletions.

Let $Q_1 = \mathtt{0}$ and $Q_k = Q_1 \cdots Q_{k-1}\mathtt{1}$ with $2 \leq k \leq p$. Let

$$\begin{aligned} T &= Q_1 Q_2 \cdots Q_p \\ &= \mathtt{0} \cdot \mathtt{01} \cdot \mathtt{0011} \cdot \mathtt{00100111} \cdot \mathtt{0010011001001111} \cdots Q_p \end{aligned}$$

with $|T| = n = \Theta(2^p)$. Since $Q_k[1..|Q_k| - 1] = T[1..|Q_k| - 1]$, $Q_k[|Q_k|] = \mathtt{1}$, and $T[|Q_k|] = \mathtt{0}$ for $2 \leq k \leq p$, each Q_k forms a single factor in the non self-referencing LZ77 factorization of T. Namely,

$$\begin{aligned} \mathsf{LZ77}(T) &= Q_1|Q_2|\cdots|Q_p| \\ &= \mathtt{0}|\mathtt{01}|\mathtt{0011}|\mathtt{00100111}|\mathtt{0010011001001111}|\cdots|Q_p| \end{aligned}$$

with $z_{77}(T) = p = \Theta(\log n)$.

substitutions: Consider the string

$$\begin{aligned} T' &= \mathtt{2} \cdot T[2..n] \\ &= \mathtt{2} \cdot Q_2 \cdots Q_p \\ &= \mathtt{2} \cdot \mathtt{01} \cdot \mathtt{0011} \cdot \mathtt{00100111} \cdot \mathtt{0010011001001111} \cdots Q_p \end{aligned}$$

which can be obtained from T by substituting the first $\mathtt{0}$ with $\mathtt{2}$. Let us analyze the structure of the non self-referencing LZ77 factorization $\mathsf{LZ77}(T')$ of T'. We prove by induction that Q_k is divided into exactly two factors for every $2 \leq k \leq p$ in $\mathsf{LZ77}(T')$. Q_2 is factorized as $\mathtt{0}|\mathtt{1}|$ in $\mathsf{LZ77}(T')$. Suppose that Q_{k-1} is divided into exactly two factors in $\mathsf{LZ77}(T')$, which means that the next factor is a prefix of $Q_k \cdots Q_p$. Since $T'[1] = \mathtt{2}$, each $Q_k[1..|Q_k| - 1]$ cannot occur as a prefix of T'. The longest prefix of $Q_k = Q_1 \cdots Q_{k-1}\mathtt{1}$ that occurs in $T'[1..|Q_1 \cdots Q_{k-1}|]$ is $Q_{k-1}[1..|Q_{k-1}| - 1] = Q_1 \cdots Q_{k-2}$. Thus, $Q_1 \cdots Q_{k-2}\mathtt{0}$ is the shortest prefix of $T'[|Q_1 \cdots Q_{k-1}| + 1..n] = Q_k \cdots Q_p$ that does not occur in $T'[1..|Q_1 \cdots Q_{k-1}|] =$

$Q_1 \cdots Q_{k-1}$. The remaining suffix of Q_k is $Q_{k-1}[2..|Q_{k-1}|]\mathtt{1} = Q_2 \cdots Q_{k-2}\mathtt{11}$. Since Q_k has $\mathtt{01}^{k-1}$ as a suffix and this is the leftmost occurrence of $\mathtt{1}^{k-1}$ in T', the next factor is this remaining suffix $Q_2 \cdots Q_{k-2}\mathtt{11}$ of Q_k. Thus, the non self-referencing LZ77 factorization of T' is

$$\mathsf{LZ77}(T') = \mathtt{2|0|1|00|11|0010|0111|00100110|01001111|}\cdots|Q_1 \cdots Q_{p-2}\mathtt{0}|Q_2 \cdots Q_{p-2}\mathtt{11}|$$

with $z_{77}(T') = 2p-1$, which leads to $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{sub}}(z_{77}, n) \geq \liminf_{p\to\infty}((2p-1)/p) = 2$, $\mathsf{AS}_{\mathrm{sub}}(z_{77}, n) \geq (2p-1) - p = p - 1 = z_{77} - 1 = \Omega(\log n)$.

insertions: Let T' be the string obtained by inserting $\mathtt{2}$ immediately after the first character $T[1] = \mathtt{0}$, namely,

$$\begin{aligned} T' &= Q_1 \cdot \mathtt{2} \cdot Q_2 \cdots Q_p \\ &= \mathtt{0} \cdot \mathtt{2} \cdot \mathtt{01} \cdot \mathtt{0011} \cdot \mathtt{00100111} \cdot \mathtt{0010011001001111} \cdots Q_p. \end{aligned}$$

Then, by similar arguments to the case of substitutions, we have

$$\mathsf{LZ77}(T') = \mathtt{0|2|01|00|11|0010|0111|00100110|01001111|}\cdots|Q_1 \cdots Q_{p-2}\mathtt{0}|Q_2 \cdots Q_{p-2}\mathtt{11}|$$

with $z_{77}(T') = 2p-1$, which leads to $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{ins}}(z_{77}, n) \geq \liminf_{p\to\infty}((2p-1)/p) = 2$, $\mathsf{AS}_{\mathrm{ins}}(z_{77}, n) \geq (2p-1) - p = p - 1 = z_{77} - 1 = \Omega(\log n)$.

deletions: Let T' be the string obtained by deleting the first character $T[1] = \mathtt{0}$, namely

$$\begin{aligned} T' &= Q_2 \cdots Q_p \\ &= \mathtt{01} \cdot \mathtt{0011} \cdot \mathtt{00100111} \cdot \mathtt{0010011001001111} \cdots Q_p. \end{aligned}$$

Then, by similar arguments to the case of substitutions, we have

$$\mathsf{LZ77}(T') = \mathtt{0|1|00|11|0010|0111|00100110|01001111|}\cdots|Q_1 \cdots Q_{p-2}\mathtt{0}|Q_2 \cdots Q_{p-2}\mathtt{11}|$$

with $z_{77}(T') = 2p-2$, which leads to $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(z_{77}, n) \geq \liminf_{p\to\infty}((2p-2)/p) = 2$, $\mathsf{AS}_{\mathrm{del}}(z_{77}, n) \geq (2p-2) - p = p - 2 = z_{77} - 2 = \Omega(\log n)$. □

The strings T and T' used in Theorem 3.9 gives us optimal additive lower bounds in terms z_{77}, are highly compressible ($z_{77}(T) = O(\log n)$) and only use two or three distinct characters. By using more characters, we can obtain larger lower bounds for the additive sensitivity for the size of the non self-referencing LZ77 factorizations $\mathsf{LZ77}$ in terms of the string length n, as follows:

Theorem 3.10. *The following lower bounds on the sensitivity of non self-referencing LZ77 factorization* $\mathsf{LZ77}$ *hold:*
substitutions: $\mathsf{AS}_{\mathrm{sub}}(z_{77}, n) = \Omega(\sqrt{n})$.
insertions: $\mathsf{AS}_{\mathrm{ins}}(z_{77}, n) = \Omega(\sqrt{n})$.
deletions: $\mathsf{AS}_{\mathrm{del}}(z_{77}, n) = \Omega(\sqrt{n})$.

Proof. Let $p = 2^h$ where $h \geq 1$.

substitutions: Consider the following string T of length $n = \Theta(p^2)$:

$$T = \mathtt{a}^{2p-2}\mathtt{b} \cdot \mathtt{a}^{p}\mathtt{b}\#_1 \cdot \mathtt{a}^{p+1}\mathtt{b}\#_2 \cdot \mathtt{a}^{p+2}\mathtt{b}\#_3 \cdots \mathtt{a}^{2p-2}\mathtt{b}\#_{p-1},$$

where $\#_j$ for every $1 \leq j \leq p-1$ is a distinct character. The non self-referencing LZ77 factorization of T is

$$\mathsf{LZ77}(T) = \mathtt{a}|\mathtt{a}^2|\mathtt{a}^4|\cdots|\mathtt{a}^{2^{h-1}}|\mathtt{a}^{p-1}\mathtt{b}|\mathtt{a}^{p}\mathtt{b}\#_1|\mathtt{a}^{p+1}\mathtt{b}\#_2|\mathtt{a}^{p+2}\mathtt{b}\#_3|\cdots|\mathtt{a}^{2p-2}\mathtt{b}\#_{p-1}|$$

with $z_{77}(T) = h + p$. Then, we consider the string

$$T' = \mathtt{a}^{p-1}\mathtt{c}\mathtt{a}^{p-2}\mathtt{b} \cdot \mathtt{a}^{p}\mathtt{b}\#_1 \cdot \mathtt{a}^{p+1}\mathtt{b}\#_2 \cdot \mathtt{a}^{p+2}\mathtt{b}\#_3 \cdots \mathtt{a}^{2p-2}\mathtt{b}\#_{p-1},$$

which can be obtained from T by substituting the p-th $\mathtt{a}$ with $\mathtt{c}$. Let us analyze the structure of the non self-referencing LZ77 factorization of T'. It is clear that h factors in the interval $[1..p-1]$ are unchanged. Since $\mathtt{c}$ is a fresh character, it becomes a factor of length 1. Also, $\mathtt{a}^{p-2}\mathtt{b}$ becomes a factor. The following each factor $\mathtt{a}^{p+k-2}\mathtt{b}\#_{k-1}$ with $2 \leq k \leq p$ is divided into two factors $\mathtt{a}^{p+k-2}$ and $\mathtt{b}\#_{k-1}$, since there are no previous occurrences of $\mathtt{a}^{p+k-2}$ and $\#_{k-1}$. Thus, the non self-referencing LZ77 factorization of T' is

$$\mathsf{LZ77}(T') = \mathtt{a}|\mathtt{a}^2|\mathtt{a}^4|\cdots|\mathtt{a}^{2^{h-1}}|\mathtt{c}|\mathtt{a}^{p-2}\mathtt{b}|\mathtt{a}^{p}|\mathtt{b}\#_1|\mathtt{a}^{p+1}|\mathtt{b}\#_2|\mathtt{a}^{p+2}|\mathtt{b}\#_3|\cdots|\mathtt{a}^{2p-2}|\mathtt{b}\#_{p-1}|$$

with $z_{77}(T') = h + 2p$, which leads to $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{sub}}(z_{77}, n) \geq \liminf_{p\to\infty}(h+2p)/(h+p) = 2$, $\mathsf{AS}_{\mathrm{sub}}(z_{77}, n) \geq (h+2p) - (h+p) = p = \Omega(\sqrt{n})$.

insertions: As for the same string T, we consider the string

$$T' = \mathtt{a}^{p-1}\mathtt{c}\mathtt{a}^{p-1}\mathtt{b} \cdot \mathtt{a}^{p}\mathtt{b}\#_1 \cdot \mathtt{a}^{p+1}\mathtt{b}\#_2 \cdot \mathtt{a}^{p+2}\mathtt{b}\#_3 \cdots \mathtt{a}^{2p-2}\mathtt{b}\#_{p-1},$$

which can be obtained from T by inserting $\mathtt{c}$ between position $p-1$ and position p in T. Then, by similar arguments to the case of substitutions, the non self-referencing LZ77 factorization of T' is

$$\mathsf{LZ77}(T') = \mathtt{a}|\mathtt{a}^2|\mathtt{a}^4|\cdots|\mathtt{a}^{2^{h-1}}|\mathtt{c}|\mathtt{a}^{p-1}\mathtt{b}|\mathtt{a}^{p}|\mathtt{b}\#_1|\mathtt{a}^{p+1}|\mathtt{b}\#_2|\mathtt{a}^{p+2}|\mathtt{b}\#_3|\cdots|\mathtt{a}^{2p-2}|\mathtt{b}\#_{p-1}|$$

with $z_{77}(T') = h+2p$, which leads to $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{ins}}(z_{77}, n) \geq \liminf_{p\to\infty}(h+2p)/(h+p) = 2$, $\mathsf{AS}_{\mathrm{ins}}(z_{77}, n) \geq p = \Omega(\sqrt{n})$.

deletions: Consider the following string T of length $n = \Theta(p^2)$:

$$T = \mathtt{a}^{p-1}\mathtt{cb} \cdot \mathtt{acb}\#_1 \cdot \mathtt{a}^2\mathtt{cb}\#_2 \cdot \mathtt{a}^3\mathtt{cb}\#_3 \cdots \mathtt{a}^{p-1}\mathtt{cb}\#_{p-1}.$$

The non self-referencing LZ77 factorization of T is

$$\mathsf{LZ77}(T) = \mathtt{a}|\mathtt{a}^2|\mathtt{a}^4|\cdots|\mathtt{a}^{2^{h-1}}|\mathtt{c}|\mathtt{b}|\mathtt{acb}\#_1|\mathtt{a}^2\mathtt{cb}\#_2|\mathtt{a}^3\mathtt{cb}\#_3|\cdots|\mathtt{a}^{p-1}\mathtt{cb}\#_{p-1}|$$

with $z_{77}(T) = h+p+1$. Then, we consider the string

$$T' = \mathtt{a}^{p-1}\mathtt{b} \cdot \mathtt{acb}\#_1 \cdot \mathtt{a}^2\mathtt{cb}\#_2 \cdot \mathtt{a}^3\mathtt{cb}\#_3 \cdots \mathtt{a}^{p-1}\mathtt{cb}\#_{p-1},$$

which can be obtained from T by deleting the first $\mathtt{c}$ in T. Let us analyze the structure of the non self-referencing LZ77 factorization of T'. It is clear that h factors in the interval $[1..p-1]$ are unchanged. The next factor is $\mathtt{b}$ of length 1. The following each factor $\mathtt{a}^k\mathtt{cb}\#_k$ with $1 \leq k \leq p-1$ is divided into two factors $\mathtt{a}^k\mathtt{c}$ and $\mathtt{b}\#_k$, since there are no previous occurrences of $\mathtt{a}^k\mathtt{c}$ and $\mathtt{b}\#_k$. Thus, the non self-referencing LZ77 factorization of T' is

$$\mathsf{LZ77}(T') = \mathtt{a}|\mathtt{a}^2|\mathtt{a}^4|\cdots|\mathtt{a}^{2^{h-1}}|\mathtt{b}|\mathtt{ac}|\mathtt{b}\#_1|\mathtt{a}^2\mathtt{c}|\mathtt{b}\#_2|\mathtt{a}^3\mathtt{c}|\mathtt{b}\#_3|\cdots|\mathtt{a}^{p-1}\mathtt{c}|\mathtt{b}\#_{p-1}|$$

with $z_{77}(T') = h+1+2(p-1) = h+2p-1$, which leads to $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(z_{77}, n) \geq \liminf_{p\to\infty}(h+2p-1)/(h+p+1) = 2$, $\mathsf{AS}_{\mathrm{del}}(z_{77}, n) \geq (h+2p-1)-(h+p+1) = p-2 = \Omega(\sqrt{n})$. ☐

3.5.2 Upper Bounds for Sensitivity of z_{77}

Theorem 3.11. *The following upper bounds on the sensitivity of non self-referencing LZ77 factorization* $\mathsf{LZ77}$ *hold:*
substitutions: $\limsup_{n\to\infty} \mathsf{MS}_{\mathrm{sub}}(z_{77}, n) \leq 2$. $\mathsf{AS}_{\mathrm{sub}}(z_{77}, n) \leq z_{77} - 1$.
insertions: $\limsup_{n\to\infty} \mathsf{MS}_{\mathrm{ins}}(z_{77}, n) \leq 2$. $\mathsf{AS}_{\mathrm{ins}}(z_{77}, n) \leq z_{77} - 1$.
deletions: $\limsup_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(z_{77}, n) \leq 2$. $\mathsf{AS}_{\mathrm{del}}(z_{77}, n) \leq z_{77} - 2$.

Proof. In the following, we consider the case that $T[i] = a$ is substituted by a character $\#$ that does not occur in T. The other cases of insertions, deletions, and substitutions with another character b ($\neq a$) occurring in T, can be proven similarly, which will be discussed at the end of the proof.

We denote the factorizations as $\mathsf{LZ77}(T) = f_1 \cdots f_z$ and $\mathsf{LZ77}(T') = f'_1 \cdots f'_{z'}$. We denote the interval of factor f_j (resp. f'_j) by $[p_j, q_j]$ (resp. $[p'_j, q'_j]$).

Now we prove the following claim:

Claim 3.1. *Each interval $[p_j, q_j]$ has at most two starting positions p'_k and p'_{k+1} of factors in $\mathsf{LZ77}(T')$ for some $1 \leq k < z'$.*

Proof. There are the three following cases:

(1) When the interval $[p_j, q_j]$ satisfies $q_j < i$: $f_j = f'_j$ holds for any such j. Therefore, in the interval $[p_j, q_j]$ there exists exactly one starting position $p'_j = p_j$ of a factor in $\mathsf{LZ77}(T')$.

(2) When the interval $[p_j, q_j]$ satisfies $p_j \leq i \leq q_j$: Let $T[p_j..q_j] = w_1 a w_2 c$ and $T'[p_j..q_j] = w_1 \# w_2 c$, where $a, c, \# \in \Sigma$ and $w_1, w_2 \in \Sigma^*$. By definition, $w_1 a w_2$ has at least one previous occurrence in $f_1 \cdots f_{j-1}$. After the substitution, $w_1\#$ becomes a factor f'_j of $\mathsf{LZ77}(T')$ since $\#$ is a fresh character, and $w_2 c$ becomes a prefix of the next factor f'_{j+1} in $\mathsf{LZ77}(T')$. This means that $p'_j = p_j$ and $q'_{j+1} \geq q_j$. Therefore, the interval $[p_j, q_j]$ has at most two starting positions p'_j and p'_{j+1} of factors in $\mathsf{LZ77}(T')$.

(3) When the interval $[p_j, q_j]$ satisfies $i < p_j$: There are the two following sub-cases:

(3-A) When $T[p_j..q_j - 1]$ has a previous occurrence which does not contain the edited position i in T: In this case, any suffix of $T[p_j..q_j - 1]$ has a previous occurrence in T'. Therefore, $[p'_k, q'_k]$ with $p_j \leq p'_k$ satisfies $q'_k \geq q_j$. Hence, the interval $[p_j..q_j]$ has at most one starting position p'_k of a factor in $\mathsf{LZ77}(T')$.

(3-B) When all previous occurrences of $T[p_j..q_j - 1]$ in T contain the edited position i: Let $u_1 a u_2 d = T[p_j, q_j]$ with $a, d \in \Sigma$ and $u_1, u_2 \in \Sigma^*$. u_1 and u_2 have previous occurrences in $T'[1..p_j - 1]$. Let p'_k be the starting position of the leftmost factor of $\mathsf{LZ77}(T')$ which begins in range $[p_j, q_j]$. If p'_k is in u_2, then $q'_k \geq q_k$ and thus there is only one starting position of a factor of $\mathsf{LZ77}(T')$ in the interval $[p_j..q_j]$. Suppose p'_k is in u_1. If a has no previous occurrences (which happens when $T[i]$ was the only previous occurrence of a), then $T'[p_k + |u_1|]$ is the first occurrence of a and thus $q'_k = p_k + |u_1|$. Otherwise, $q'_k \geq p_k + |u_1|$. In either case, since u_2 has a previous occurrence, $q'_{k+1} \geq q_{k+1}$. Thus, there can exist at most two starting positions of factors of $\mathsf{LZ77}(T')$ in the interval $[p_j..q_j]$.

This completes the proof for the claim. □

By the above claim, $z_{77}(T') \leq 2z_{77}(T)$ holds for any string T and any substitution operation. Since f_1 consists of a single character for any string and the interval $[1, 1]$ cannot have two starting positions of factors in $\mathsf{LZ77}(T')$, $z_{77}(T') \leq 2z_{77}(T) - 1$ holds. This completes the proof for the case of substitution with $\#$.

The above proof can be generalized to all the other cases, by replacing $\#$ in T' as follows:

- $\# \leftarrow b$ for substitutions with character b occurring in T, where we have $T'[p_j, q_j] = w_1 b w_2 c$ for Case (2);
- $\# \leftarrow T[i]\#$ for insertions with $\#$, where we have $T'[p_j, q_j] = w_1 T[i]\# w_2 c$ for Case (2);
- $\# \leftarrow T[i]b$ for insertions with character b occurring in T, where we have $T'[p_j, q_j] = w_1 T[i] b w_2 c$ for Case (2);
- $\# \leftarrow \varepsilon$ for deletions, where we have $T'[p_j, q_j] = w_1 w_2 c$ for Case (2).

The analysis for Case (2) and Case (3) is analogous for all these cases. Also, in the case of deletions, since $|f_2| \leq 2$ and the interval can have two starting positions of factors in $\mathsf{LZ77}(T')$ only when $f_1 = T[1]$ is deleted, $z_{77}(T') \leq 2z_{77}(T) - 2$ holds. □

3.5.3 Lower Bounds for Sensitivity of $z_{77\mathrm{sr}}$

Theorem 3.12. *The following lower bounds on the sensitivity of self-referencing LZ77 factorization* $\mathsf{LZ77sr}$ *with* $|\Sigma| = 3$ *hold:*
substitutions: $\mathsf{MS}_{\mathrm{sub}}(z_{77\mathrm{sr}}, n) \geq 2$. $\mathsf{AS}_{\mathrm{sub}}(z_{77\mathrm{sr}}, n) \geq z_{77\mathrm{sr}}$.
insertions: $\mathsf{MS}_{\mathrm{ins}}(z_{77\mathrm{sr}}, n) \geq 2$. $\mathsf{AS}_{\mathrm{ins}}(z_{77\mathrm{sr}}, n) \geq z_{77\mathrm{sr}}$.
deletions: $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(z_{77\mathrm{sr}}, n) \geq 2$. $\mathsf{AS}_{\mathrm{del}}(z_{77\mathrm{sr}}, n) \geq z_{77\mathrm{sr}} - 2$.

Proof. **substitutions:** Let $p \geq 2$ and $\Sigma = \{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$. We use the following string T for our analysis.

Let $R_1 = \mathtt{00}$ and $R_k = R_1 \cdots R_{k-1}\mathtt{1}$ with $2 \leq k \leq p$. Consider the following string T of length $n = \Theta(2^p)$:

$$\begin{aligned} T &= R_1 \cdots R_p \\ &= \mathtt{00} \cdot \mathtt{001} \cdot \mathtt{000011} \cdot \mathtt{000010000111} \cdots R_p \end{aligned}$$

with $|T| = n = \Theta(2^p)$. It immediately follows from the definition of T that the self-referencing LZ77 factorization of T is

$$\begin{aligned}\mathsf{LZ77sr}(T) &= \mathtt{0}|\mathtt{0001}|R_3|R_4|\cdots|R_p| \\ &= \mathtt{0}|\mathtt{0001}|\mathtt{000011}|\mathtt{000010000111}|\cdots|R_p|\end{aligned}$$

with $z_{77\mathrm{sr}}(T) = p = \Theta(\log n)$. Note that the second factor $\mathtt{0001}$ is self-referencing.

As for substitution, we consider the string

$$\begin{aligned}T' &= T[1] \cdot \mathtt{2} \cdot T[3..n] \\ &= \mathtt{02} \cdot \mathtt{001} \cdot \mathtt{000011} \cdot \mathtt{000010000111} \cdots R_p\end{aligned}$$

which can be obtained from T by substituting the second $\mathtt{0}$ with $\mathtt{2}$. Let us analyze the structure of the self-referencing LZ77 factorization of T'. The second factor $\mathtt{0001}$ in $\mathsf{LZ77sr}(T)$ becomes $\mathtt{2001}$ in the edited string T', and this is divided into exactly three factors as $\mathtt{2}|\mathtt{00}|\mathtt{1}|$ in $\mathsf{LZ77sr}(T')$ because $\mathtt{2}$ is a fresh character, $\mathtt{00}$ is the shortest prefix of $T[3..n] = \mathtt{001}R_3 \cdots R_p$ that does not occur in $T[1..2] = \mathtt{02}$, and $\mathtt{1}$ is a fresh character. Our claim is that each R_k with $3 \leq k \leq p$ is halved into two factors $R_k[1..|R_k|/2] = R_1 \cdots R_{k-2}\mathtt{0}$ and $R_k[|R_k|/2 + 1..|R_k|] = \mathtt{0}R_2 \cdots R_{k-2}\mathtt{11}$ of equal length in $\mathsf{LZ77sr}(T')$. Suppose that R_{k-1} is factorized as $R_{k-1}[1..|R_{k-1}|/2] \mid R_{k-1}[|R_{k-1}|/2 + 1..|R_{k-1}|] \mid$ in $\mathsf{LZ77sr}(T')$, which means that the next factor is a prefix of $R_k \cdots R_p$. Since $R_k[1..2] = \mathtt{00}$ and $T'[1..2] = \mathtt{02}$, $R_k[1..|R_k| - 1]$ does not have a previous occurrence as a prefix of T'. Since $R_k = R_1 \cdots R_{k-2}R_{k-1}\mathtt{1}$ and $R_1 \cdots R_{k-2} = R_{k-1}[1..|R_{k-1}| - 1]$, the longest prefix of $T'[|R_1 \cdots R_{k-1}| + 1..n] = R_k \cdots R_p$ that has a previous occurrence beginning in range $[1..|R_1 \cdots R_{k-1}|]$ is $R_1 \cdots R_{k-2}$, which implies $R_1 \cdots R_{k-2}R_{k-1}[1] = R_1 \cdots R_{k-2}\mathtt{0}$ is the next factor in $\mathsf{LZ77sr}(T')$. The remaining part of R_k is $R_{k-1}[2..|R_{k-1}|]\mathtt{1} = R_1[2]R_2 \cdots R_{k-2}\mathtt{11} = \mathtt{0}R_2 \cdots R_{k-2}\mathtt{11}$. Since its prefix $\mathtt{0}R_2 \cdots R_{k-2}\mathtt{1}$ has a previous occurrence and $\mathtt{0}R_2 \cdots R_{k-2}\mathtt{11}$ has a suffix $\mathtt{01}^{k-1}$ which is the leftmost occurrence of $\mathtt{1}^{k-1}$ in T', this remaining part $\mathtt{0}R_2 \cdots R_{k-2}\mathtt{11}$ becomes the next factor in $\mathsf{LZ77sr}(T')$. Thus, the self-referencing LZ77 factorization of T' is

$$\mathsf{LZ77sr}(T') = \mathtt{0}|\mathtt{2}|\mathtt{00}|\mathtt{1}|\mathtt{000}|\mathtt{011}|\mathtt{000010}|\mathtt{000111}|\cdots|R_1 \cdots R_{p-2}\mathtt{0}|\mathtt{0}R_2 \cdots R_{p-2}\mathtt{11}|$$

with $z_{77\mathrm{sr}}(T') = 2p$, which leads to $\mathsf{MS}_{\mathrm{sub}}(z_{77\mathrm{sr}}, n) \geq 2p/p = 2$ and $\mathsf{AS}_{\mathrm{sub}}(z_{77\mathrm{sr}}, n) \geq 2p - p = p = z_{77\mathrm{sr}} = \Omega(\log n)$.

insertions: We use the same string T in the case of substitutions. Let T' be the string obtained by inserting 2 immediately after $T[1] = \mathtt{0}$, namely,

$$\begin{aligned}T' &= \mathtt{0} \cdot \mathtt{2} \cdot \mathtt{0} \cdot R_2 \cdots R_p \\ &= \mathtt{0} \cdot \mathtt{2} \cdot \mathtt{0} \cdot \mathtt{001} \cdot \mathtt{000011} \cdot \mathtt{000010000111} \cdots R_p.\end{aligned}$$

Then, by similar arguments to the case of substitutions, we have

$$\mathsf{LZ77sr}(T') = \mathtt{0}|\mathtt{2}|\mathtt{00}|\mathtt{01}|\mathtt{0000}|\mathtt{11}|\mathtt{000010}|\mathtt{000111}|\cdots|R_1\cdots R_{p-2}\mathtt{0}|\mathtt{0}R_2\cdots R_{p-2}\mathtt{11}|$$

with $z_{77\mathrm{sr}}(T') = 2p$, which leads to $\mathsf{MS}_{\mathrm{ins}}(z_{77\mathrm{sr}}, n) \geq 2p/p = 2$ and $\mathsf{AS}_{\mathrm{ins}}(z_{77\mathrm{sr}}, n) \geq 2p - p = p = z_{77\mathrm{sr}} = \Omega(\log n)$.

deletions: As for deletions, we use the same strings T and T' from Theorem 3.9. This string and the deletion also achieve the same lower bound for the self-referencing LZ77 factorization in the case of deletions. Then, we obtain $z_{77\mathrm{sr}}(T) = p$, $z_{77\mathrm{sr}}(T') = 2p-2$, which leads to $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(z_{77\mathrm{sr}}, n) \geq 2$ and $\mathsf{AS}_{\mathrm{del}}(z_{77\mathrm{sr}}, n) \geq z_{77\mathrm{sr}} - 2 = \Omega(\log n)$.

□

The strings T and T' used in Theorem 3.12 gives us optimal additive lower bounds in terms $z_{77\mathrm{sr}}$, are highly compressible ($z_{77\mathrm{sr}}(T) = O(\log n)$) and only use two or three distinct characters. By using more characters, we can obtain larger lower bounds for the additive sensitivity for the size of the self-referencing LZ77 factorizations in terms of the string length n, as follows:

Theorem 3.13. *The following lower bounds on the sensitivity of self-referencing LZ77 factorization* $\mathsf{LZ77sr}$ *hold:*
substitutions: $\mathsf{AS}_{\mathrm{sub}}(z_{77\mathrm{sr}}, n) = \Omega(\sqrt{n})$.
insertions: $\mathsf{AS}_{\mathrm{ins}}(z_{77\mathrm{sr}}, n) = \Omega(\sqrt{n})$.
deletions: $\mathsf{AS}_{\mathrm{del}}(z_{77\mathrm{sr}}, n) = \Omega(\sqrt{n})$.

Proof. **substitutions:** Consider the following string T of length $n = \Theta(p^2)$:

$$T = \mathtt{a}^{p-1}\mathtt{a} \cdot \mathtt{a}^{p}\mathtt{b} \cdot \mathtt{a}^{p+1}\mathtt{b}\#_1 \cdot \mathtt{a}^{p+2}\mathtt{b}\#_2 \cdots \mathtt{a}^{2p-1}\mathtt{b}\#_{p-1}$$

which consists of $p+1$ components. The self-referencing LZ77 factorization of T is

$$\mathsf{LZ77sr}(T) = \mathtt{a}|\mathtt{a}^{2p-1}\mathtt{b}|\mathtt{a}^{p+1}\mathtt{b}\#_1|\mathtt{a}^{p+2}\mathtt{b}\#_2|\cdots|\mathtt{a}^{2p-1}\mathtt{b}\#_{p-1}|$$

with $z_{77\mathrm{sr}}(T) = p+1$. Notice that the second factor $\mathtt{a}^{2p-1}\mathtt{1}$ is self-referencing.

Consider the string T'

$$T' = \mathtt{a}^{p-1}\mathtt{c} \cdot \mathtt{a}^{p}\mathtt{b} \cdot \mathtt{a}^{p+1}\mathtt{b}\#_1 \cdot \mathtt{a}^{p+2}\mathtt{b}\#_2 \cdots \mathtt{a}^{2p-1}\mathtt{b}\#_{p-1}$$

that can be obtained from T by substituting the p-th $\mathtt{a}$ with $\mathtt{c}$. The self-referencing LZ77 factorization of T' is

$$\mathsf{LZ77sr}(T') = \mathtt{a}|\mathtt{a}^{p-2}\mathtt{c}|\mathtt{a}^{p}|\mathtt{b}|\mathtt{a}^{p+1}|\mathtt{b}\#_1|\mathtt{a}^{p+2}|\mathtt{b}\#_2|\cdots|\mathtt{a}^{2p-1}|\mathtt{b}\#_{p-1}|$$

with $z_{77\mathrm{sr}}(T') = 2p{+}2$, which leads to $\mathsf{MS}_{\mathrm{sub}}(z_{77\mathrm{sr}}, n) \geq (2p{+}2)/(p{+}1) = 2$, $\mathsf{AS}_{\mathrm{sub}}(z_{77\mathrm{sr}}, n) \geq (2p+2) - (p+1) = p+1 = z_{77\mathrm{sr}}$, and $\mathsf{AS}_{\mathrm{sub}}(z_{77\mathrm{sr}}, n) = \Omega(\sqrt{n})$.

insertions: Consider the following string T of length $n = \Theta(p^2)$:

$$T = \mathtt{a}^{p-1} \cdot \mathtt{a}^{p}\mathtt{b} \cdot \mathtt{a}^{p+1}\mathtt{b}\#_{1} \cdot \mathtt{a}^{p+2}\mathtt{b}\#_{2} \cdots \mathtt{a}^{2p-1}\mathtt{b}\#_{p-1}.$$

The self-referencing LZ77 factorization of T is

$$\mathsf{LZ77sr}(T) = \mathtt{a}|\mathtt{a}^{2p-2}\mathtt{b}|\mathtt{a}^{p+1}\mathtt{b}\#_{1}|\mathtt{a}^{p+2}\mathtt{b}\#_{2}|\cdots|\mathtt{a}^{2p-1}\mathtt{b}\#_{p-1}|$$

with $z_{77\mathrm{sr}}(T) = p+1$. Notice that the second factor $\mathtt{a}^{2p-1}\mathtt{1}$ is self-referencing.

Consider the string T'

$$T' = \mathtt{a}^{p-1}\mathtt{c} \cdot \mathtt{a}^{p}\mathtt{b} \cdot \mathtt{a}^{p+1}\mathtt{b}\#_{1} \cdot \mathtt{a}^{p+2}\mathtt{b}\#_{2} \cdots \mathtt{a}^{2p-1}\mathtt{b}\#_{p-1}$$

that can be obtained from T by inserting $\mathtt{c}$ between position $p-1$ and position p. The self-referencing LZ77 factorization of T' is

$$\mathsf{LZ77sr}(T') = \mathtt{a}|\mathtt{a}^{p-2}\mathtt{c}|\mathtt{a}^{p}|\mathtt{b}|\mathtt{a}^{p+1}|\mathtt{b}\#_{1}|\mathtt{a}^{p+2}|\mathtt{b}\#_{2}|\cdots|\mathtt{a}^{2p-1}|\mathtt{b}\#_{p-1}|$$

with $z_{77\mathrm{sr}}(T') = 2p+2$, which leads to $\mathsf{MS}_{\mathrm{ins}}(z_{77\mathrm{sr}}, n) \geq 2$, $\mathsf{AS}_{\mathrm{ins}}(z_{77\mathrm{sr}}, n) \geq p+1 = z_{77\mathrm{sr}}$, and $\mathsf{AS}_{\mathrm{ins}}(z_{77\mathrm{sr}}, n) = \Omega(\sqrt{n})$.

deletions: Consider the following string T of length $n = \Theta(p^2)$:

$$T = \mathtt{a}^{p}\mathtt{bc} \cdot \mathtt{abc}\#_{1} \cdot \mathtt{a}^{2}\mathtt{bc}\#_{2} \cdots \mathtt{a}^{p}\mathtt{bc}\#_{p}.$$

The self-referencing LZ77 factorization of T is

$$\mathsf{LZ77sr}(T) = \mathtt{a}|\mathtt{a}^{p-1}\mathtt{b}|\mathtt{c}|\mathtt{abc}\#_{1}|\mathtt{a}^{2}\mathtt{bc}\#_{2}|\cdots|\mathtt{a}^{p}\mathtt{bc}\#_{p}|$$

with $z_{77\mathrm{sr}}(T) = p+3$. Notice that the second factor $\mathtt{a}^{p-2}\mathtt{1}$ is self-referencing.

Consider the string T'

$$T' = \mathtt{a}^{p}\mathtt{b} \cdot \mathtt{abc}\#_{1} \cdot \mathtt{a}^{2}\mathtt{bc}\#_{2} \cdots \mathtt{a}^{p}\mathtt{bc}\#_{p}$$

that can be obtained from T by deleting the first $\mathtt{c}$ of position $p+2$. Let us analyze the structure of the self-referencing LZ77 factorization of T'. The first two factors are unchanged. The third factor $\mathtt{c}$ of $\mathsf{LZ77sr}(T)$ is removed, and each of the remaining factors of form $\mathtt{a}^{k}bc\#_{k}$ in $\mathsf{LZ77sr}(T)$ is divided into two factors as $\mathtt{a}^{k}bc|\#_{k}|$. Thus the self-referencing LZ77 factorization of T' is

$$\mathsf{LZ77sr}(T') = \mathtt{a}|\mathtt{a}^{p-1}\mathtt{b}|\mathtt{abc}|\#_{1}|\mathtt{a}^{2}\mathtt{bc}|\#_{2}|\cdots|\mathtt{a}^{p}\mathtt{bc}|\#_{p}|$$

with $z_{77\mathrm{sr}}(T') = 2p+2$, which leads to $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(z_{77\mathrm{sr}}, n) \geq \liminf_{p\to\infty}(2p+2)/(p+3) = 2$, $\mathsf{AS}_{\mathrm{del}}(z_{77\mathrm{sr}}, n) \geq 2p+2-(p+3) = p-1 = \Omega(\sqrt{n})$. $\square$

It is also possible to binarize the strings T and T' in the above proof for the cases of substitutions and insertions, while retaining the same lower bounds:

Corollary 3.2. *For the self-referencing LZ77 factorization, there are binary strings of length n that satisfy $\mathsf{MS}_{\mathrm{sub}}(z_{77\mathrm{sr}}, n) \geq 2$, $\mathsf{MS}_{\mathrm{ins}}(z_{77\mathrm{sr}}, n) \geq 2$, respectively.*

Proof. Let $p \geq 2$.

substitutions: Consider the following string T of length $n = \Theta(p^2)$:

$$T = \mathtt{0}^{p-1} \cdot \mathtt{0} \cdot \mathtt{0}^{2p}\mathtt{1} \cdot \mathtt{0}^{2p+1}\mathtt{101} \cdot \mathtt{0}^{2p+2}\mathtt{10}^{2}\mathtt{1} \cdots \mathtt{0}^{3p}\mathtt{10}^{p}\mathtt{1}.$$

The self-referencing LZ77 factorization of T is:

$$\mathsf{LZ77sr}(T) = \mathtt{0}|\mathtt{0}^{3p-1}\mathtt{1}|\mathtt{0}^{2p+1}\mathtt{101}|\mathtt{0}^{2p+2}\mathtt{10}^{2}\mathtt{1}|\cdots|\mathtt{0}^{3p}\mathtt{10}^{p}\mathtt{1}|$$

with $p+2$ factors. Then, we consider the string

$$T' = \mathtt{0}^{p-1} \cdot \mathtt{1} \cdot \mathtt{0}^{2p}\mathtt{1} \cdot \mathtt{0}^{2p+1}\mathtt{101} \cdot \mathtt{0}^{2p+2}\mathtt{10}^{2}\mathtt{1} \cdots \mathtt{0}^{3p}\mathtt{10}^{p}\mathtt{1}$$

is obtained by substituting p-th $\mathtt{0}$ with $\mathtt{1}$. The self-referencing LZ77 factorization of T' is:

$$\mathsf{LZ77sr}(T') = \mathtt{0}|\mathtt{0}^{p-2}\mathtt{1}|\mathtt{0}^{p}|\mathtt{0}^{p}\mathtt{1}|\mathtt{0}^{2p+1}|\mathtt{101}|\mathtt{0}^{2p+2}|\mathtt{10}^{2}\mathtt{1}|\cdots|\mathtt{0}^{3p}|\mathtt{10}^{p}\mathtt{1}|$$

with $2p+4$ factors. Then we obtain $\mathsf{MS}_{\mathrm{sub}}(z_{77\mathrm{sr}}, n) \geq (2p+4)/(p+2) = 2$.

insertions: Consider the following string T of length $n = \Theta(p^2)$:

$$T = \mathtt{0}^{p-1} \cdot \mathtt{0}^{2p}\mathtt{1} \cdot \mathtt{0}^{2p+1}\mathtt{101} \cdot \mathtt{0}^{2p+2}\mathtt{10}^{2}\mathtt{1} \cdots \mathtt{0}^{3p}\mathtt{10}^{p}\mathtt{1}.$$

The self-referencing LZ77 factorization of T is:

$$\mathsf{LZ77sr}(T) = \mathtt{0}|\mathtt{0}^{3p-2}\mathtt{1}|\mathtt{0}^{2p+1}\mathtt{101}|\mathtt{0}^{2p+2}\mathtt{10}^{2}\mathtt{1}|\cdots|\mathtt{0}^{3p}\mathtt{10}^{p}\mathtt{1}|$$

with $p+2$ factors. Consider the string

$$T' = \mathtt{0}^{p-1} \cdot \mathtt{1} \cdot \mathtt{0}^{2p}\mathtt{1} \cdot \mathtt{0}^{2p+1}\mathtt{101} \cdot \mathtt{0}^{2p+2}\mathtt{10}^{2}\mathtt{1} \cdots \mathtt{0}^{3p}\mathtt{10}^{p}\mathtt{1}$$

is obtained by inserting $\mathtt{1}$ between $p-1$ and p. The self-referencing LZ77 factorization of T' is:

$$\mathsf{LZ77sr}(T') = \mathtt{0}|\mathtt{0}^{p-2}\mathtt{1}|\mathtt{0}^{p}|\mathtt{0}^{p}\mathtt{1}|\mathtt{0}^{2p+1}|\mathtt{101}|\mathtt{0}^{2p+2}|\mathtt{10}^{2}\mathtt{1}|\cdots|\mathtt{0}^{3p}|\mathtt{10}^{p}\mathtt{1}|$$

with $2p+4$ factors. Then we get $\mathsf{MS}_{\mathrm{ins}}(z_{77\mathrm{sr}}, n) \geq (2p+4)/(p+2) = 2$.

$\square$

3.5.4 Upper Bounds for Sensitivity of $z_{77\mathrm{sr}}$

Theorem 3.14. *The following upper bounds on the sensitivity of self-referencing LZ77 factorization* $\mathsf{LZ77sr}$ *hold:*
substitutions: $\mathsf{MS}_{\mathrm{sub}}(z_{77\mathrm{sr}}, n) \leq 2$. $\mathsf{AS}_{\mathrm{sub}}(z_{77\mathrm{sr}}, n) \leq z_{77\mathrm{sr}}$.
insertions: $\mathsf{MS}_{\mathrm{ins}}(z_{77\mathrm{sr}}, n) \leq 2$. $\mathsf{AS}_{\mathrm{ins}}(z_{77\mathrm{sr}}, n) \leq z_{77\mathrm{sr}}$.
deletions: $\mathsf{MS}_{\mathrm{del}}(z_{77\mathrm{sr}}, n) \leq 2$. $\mathsf{AS}_{\mathrm{del}}(z_{77\mathrm{sr}}, n) \leq z_{77\mathrm{sr}}$.

Proof. We use the same notations as in Theorem 3.11 of Section 3.5.2. We consider the case where $T[i]$ is substituted by a fresh character $\#$, as in the proof for Theorem 3.11. We prove the following claim:

Claim 3.2. *Each interval* $[p_j, q_j]$ *has at most two starting positions* p'_k *and* p'_{k+1} *of factors in* $\mathsf{LZ77sr}(T')$ *for* $1 \leq k < z'$, *excluding the interval* $[p_I, q_I]$ *that contains the edited position* i. *The interval* $[p_I, q_I]$ *has at most three starting positions of factors in* $\mathsf{LZ77sr}(T')$.

Proof. Cases (1) and (3) which correspond to the positions before and after i can be shown by the same discussions in the case of non self-referencing LZ factorizations (Theorem 3.11 in Section 3.5.2). Now we consider case (2):

(2) The interval $[p_j, q_j]$ satisfies $p_j \leq i \leq q_j$ (namely, $f_j = f_I$): If f_I is not self-referencing, then by the same argument to the proof for Theorem 3.11 in Section 3.5.2, the interval has at most two starting positions of factors in $\mathsf{LZ77sr}(T')$. Now we consider the case that f_I is self-referencing. For the string $w_1 a w_2 c = T[p_I..q_I]$, only the substrings of w_2 can have a self-referencing previous occurrence that contains the edited position i in T. Therefore, w_1 has a previous occurrence in T' not containing i, which means that $q'_k = i$ where $T'[i] = \#$ is a fresh character. For the $w_2 c$ part, we can apply the same discussion of Case (3) in Theorem 3.11 of Section 3.5.2. Therefore, the $w_1\#$ part of $T'[p_j, q_j] = w_1 \# w_2 c$ can have at most one starting position, and the $w_2 c$ part can have at most two starting positions of a factor in $\mathsf{LZ77sr}(T')$.

This completes the proof for the claim. □

By the above claim, $z_{77\mathrm{sr}}(T') \leq 2z_{77\mathrm{sr}}(T) + 1$ holds for any string T and any substitution. Since again $|f_1| = 1$, we get $z_{77\mathrm{sr}}(T') \leq 2z_{77\mathrm{sr}}(T)$.

Using the same character(s) as in the proof for Theorem 3.11, we can generalize this proof to the other types of edit operations. □

3.6 Lempel-Ziv-Storer-Szymanski Factorizations with/without Self-References

In this section, we consider the worst-case sensitivity of the *Lempel-Ziv-Storer-Szymanski factorizations* (*LZSS*) [115] with/without self-references, a.k.a. C-factorizations [31].

Given a factorization $T = f_1 \cdots f_z$ for a string T of length n:

- it is the non self-referencing LZSS factorization $\mathsf{LZSS}(T)$ of T if for each $1 \leq i \leq z$ the factor f_i is either the first occurrence of a character in T, or the longest prefix of $f_i \cdots f_z$ occurs in $f_1 \cdots f_{i-1}$.
- it is the self-referencing LZSS factorization $\mathsf{LZSSsr}(T)$ of T if for each $1 \leq i \leq z$ the factor f_i is either the first occurrence of a character in T, or the longest prefix of $f_i \cdots f_z$ occurs at least twice in $f_1 \cdots f_i$.

To avoid confusions, we use different notations to denote the sizes of these factorizations. For a string T let $z_{\mathrm{SS}}(T)$ and $z_{\mathrm{SSsr}}(T)$ denote the number z of factors in the non self-referencing LZSS factorization and in the self-referencing LZSS factorization of T, respectively.

For example, for string $T = \mathtt{abaabababababab}$, we have

$$\begin{aligned} \mathsf{LZSS}(T) &= \mathtt{a|b|a|aba|ba|baba|bab|}, \\ \mathsf{LZSSsr}(T) &= \mathtt{a|b|a|aba|babababab|}, \end{aligned}$$

where $|$ denotes the right-end of each factor in the factorizations. Here we have $z_{\mathrm{SS}}(T) = 7$ and $z_{\mathrm{SSsr}}(T) = 5$.

3.6.1 Lower Bounds for Sensitivity of z_{SS}

Theorem 3.15. *The following lower bounds on the sensitivity of non self-referencing LZSS factorization* LZSS *hold:*

substitutions: $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{sub}}(z_{\mathrm{SS}}, n) \geq 3$. $\mathsf{AS}_{\mathrm{sub}}(z_{\mathrm{SS}}, n) \geq 2z_{\mathrm{SS}} - \Theta(\sqrt{z_{\mathrm{SS}}})$ *and* $\mathsf{AS}_{\mathrm{sub}}(z_{\mathrm{SS}}, n) = \Omega(\sqrt{n})$.

insertions: $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{ins}}(z_{\mathrm{SS}}, n) \geq 2$. $\mathsf{AS}_{\mathrm{ins}}(z_{\mathrm{SS}}, n) \geq z_{\mathrm{SS}} - \Theta(\sqrt{z_{\mathrm{SS}}})$ *and* $\mathsf{AS}_{\mathrm{ins}}(z_{\mathrm{SS}}, n) = \Omega(\sqrt{n})$.

deletions: $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(z_{\mathrm{SS}}, n) \geq 3$. $\mathsf{AS}_{\mathrm{del}}(z_{\mathrm{SS}}, n) \geq 2z_{\mathrm{SS}} - \Theta(\sqrt{z_{\mathrm{SS}}})$ *and* $\mathsf{AS}_{\mathrm{del}}(z_{\mathrm{SS}}, n) = \Omega(\sqrt{n})$.

Proof. Let $\Sigma = \{\mathtt{0}, \mathtt{1}, \mathtt{a}_1, \ldots, \mathtt{a}_p, \mathtt{b}_1, \ldots, \mathtt{b}_p\}$. Let

$$\begin{aligned} Q_1 &= (\mathtt{a}_1 \cdots \mathtt{a}_p)(\mathtt{a}_1 \cdots \mathtt{a}_{p-1}) \cdots (\mathtt{a}_1\mathtt{a}_2)(\mathtt{a}_1), \\ Q_2 &= (\mathtt{b}_1)(\mathtt{b}_1\mathtt{b}_2) \cdots (\mathtt{b}_1 \cdots \mathtt{b}_{p-1})(\mathtt{b}_1 \cdots \mathtt{b}_p), \end{aligned}$$

and $m = |Q_1| = |Q_2| = p(p+1)/2$. Consider the following string:

$$T = (Q_1\mathtt{a}_1\mathtt{1}Q_2)(\varepsilon\mathtt{a}_1\mathtt{1}\mathtt{b}_1)(\mathtt{a}_1\mathtt{a}_1\mathtt{1}\mathtt{b}_1\mathtt{b}_1) \cdots (Q_1[m-k+2..m]\mathtt{a}_1\mathtt{1}Q_2[1..k]) \cdots (Q_1[2..m]\mathtt{a}_1\mathtt{1}Q_2)$$

with $1 \le k \le m$.

Let us analyze the structure of the non self-referencing LZSS factorization of T. Q_1 consists of p characters $\mathtt{a}_1, \ldots, \mathtt{a}_p$, and the prefix $\mathtt{a}_1 \cdots \mathtt{a}_p$ of Q_1 forms p factors of length 1. The remaining part of Q_1 is divided into $p-1$ factors as $(\mathtt{a}_1 \cdots \mathtt{a}_k)$ with $p-1 \ge k \ge 1$ because $(\mathtt{a}_1 \cdots \mathtt{a}_k)\mathtt{a}_1$ does not occur before. Next, both $T[m+1] = \mathtt{a}_1$ and $T[m+2] = \mathtt{1}$ become a factor of length 1. As for the prefix of Q_2, $\mathtt{b}_1$ is a fresh character and becomes a factor of length 1. For each $(\mathtt{b}_1 \cdots \mathtt{b}_k)$ with $2 \le k \le p$, $\mathtt{b}_1 \cdots \mathtt{b}_{k-1}$ occurs previously, and $\mathtt{b}_1 \cdots \mathtt{b}_k$ does not occur before. Therefore, each interval of $(\mathtt{b}_1 \cdots \mathtt{b}_k)$ has two factors as $\mathtt{b}_1 \cdots \mathtt{b}_{k-1}|\mathtt{b}_k|$. Then, there are $4p$ factors in the interval $[1..|Q_1\mathtt{a}_1\mathtt{1}Q_2|]$. The substring $T[|Q_1\mathtt{a}_1\mathtt{1}Q_2|..|T|]$ is the sequence of m parts $(Q_1[m-k+2..m]\mathtt{a}_1\mathtt{1}Q_2[1..k])$ with $1 \le k \le m$. Each part becomes a factor because $(Q_1[m-k+2..m]\mathtt{a}_1\mathtt{1}Q_2[1..k])$ occurs at $T[m-k+2..m+k+2]$, and $(Q_1[m-k+2..m]\mathtt{a}_1\mathtt{1}Q_2[1..k])Q_1[m-k+1]$ does not occur before. Therefore, the factorization of T is:

$$\begin{aligned} \mathsf{LZSS}(T) = \quad & Q_1|\mathtt{a}_1|\mathtt{1}|Q_2|(\varepsilon\mathtt{a}_1\mathtt{1}\mathtt{b}_1)|(\mathtt{a}_1\mathtt{a}_1\mathtt{1}\mathtt{b}_1\mathtt{b}_1)| \cdots \\ & |(Q_1[m-k+2..m]\mathtt{a}_1\mathtt{1}Q_2[1..k])| \cdots |(Q_1[2..m]\mathtt{a}_1\mathtt{1}Q_2)|, \end{aligned}$$

where

$$\mathsf{LZSS}(Q_1) = \mathtt{a}_1| \cdots |\mathtt{a}_p|(\mathtt{a}_1 \cdots \mathtt{a}_{p-1})| \cdots |(\mathtt{a}_1\mathtt{a}_2)|(\mathtt{a}_1)|$$

and

$$\mathsf{LZSS}(Q_2) = \mathtt{b}_1|\mathtt{b}_1|\mathtt{b}_2| \cdots |\mathtt{b}_1 \cdots \mathtt{b}_{p-2}|\mathtt{b}_{p-1}|\mathtt{b}_1 \cdots \mathtt{b}_{p-1}|\mathtt{b}_p|.$$

Then $z_{\mathrm{SS}}(T) = 4p + (1/2)p(p+1)$ holds.

substitutions: Let

$$T' = (Q_1\mathtt{a}_1\mathtt{0}Q_2)(\varepsilon\mathtt{a}_1\mathtt{1}\mathtt{b}_1)(\mathtt{a}_1\mathtt{a}_1\mathtt{1}\mathtt{b}_1\mathtt{b}_1) \cdots (Q_1[m-k+2..m]\mathtt{a}_1\mathtt{1}Q_2[1..k]) \cdots (Q_1[2..m]\mathtt{a}_1\mathtt{1}Q_2)$$

be the string obtained from T by substituting the first $\mathtt{1}$ with $\mathtt{0}$. It is clear that the factorization of the interval $[1..|Q_1\mathtt{a}_1\mathtt{0}Q_2|]$ is unchanged, and there are $4p$ factors in. Next, m factors $(Q_1[m -$

$k{+}2..m]\mathtt{a}_1\mathtt{1}Q_2[1..k])$ with $1 \le k \le m$ lose the position they refer to. Then, each factor $Q_1[m-k+2..m]\mathtt{a}_1\mathtt{1}Q_2[1..k]$ is divided into three factors as $Q_1[m-k+2..m]\mathtt{a}_1|\mathtt{1}Q_2[1..k-1]|Q_2[k]|$ because of their previous occurrences. Therefore, the factorization of T' is:

$$\begin{aligned}\mathsf{LZSS}(T') =& Q_1|\mathtt{a}_1|\mathtt{0}|Q_2|\mathtt{a}_1|\mathtt{1}|\mathtt{b}_1|\mathtt{a}_1\mathtt{a}_1|\mathtt{1}\mathtt{b}_1|\mathtt{b}_1|\cdots|Q_1[m-k+2..m]\mathtt{a}_1|\mathtt{1}Q_2[1..k-1]|Q_2[k]|\cdots\\ &|Q_1[2..m]\mathtt{a}_1|\mathtt{1}Q_2[1..m-1]|Q_2[m]|,\end{aligned}$$

where

$$\mathsf{LZSS}(Q_1) = \mathtt{a}_1|\cdots|\mathtt{a}_p|(\mathtt{a}_1\cdots\mathtt{a}_{p-1})|\cdots|(\mathtt{a}_1\mathtt{a}_2)|(\mathtt{a}_1)|$$

and

$$\mathsf{LZSS}(Q_2) = \mathtt{b}_1|\mathtt{b}_1|\mathtt{b}_2|\cdots|\mathtt{b}_1\cdots\mathtt{b}_{p-2}|\mathtt{b}_{p-1}|\mathtt{b}_1\cdots\mathtt{b}_{p-1}|\mathtt{b}_p|.$$

Then, $z_{\mathrm{SS}}(T') = 4p + (3/2)p(p+1)$ holds. Also,

$$\begin{aligned}|T| &= p(p-1)+2+\sum_{k=1}^{(1/2)p(p+1)}(2k+1)\\ &= p(p-1)+2+2\sum_{k=1}^{\frac{p(p+1)}{2}}k+\frac{p(p+1)}{2}\\ &= p(p-1)+2+\frac{p^2(p+1)^2}{4}+p(p+1)=\Theta(p^4)\end{aligned}$$

holds. Hence, we obtain

$$\begin{aligned}\liminf_{n\to\infty}\mathsf{MS}_{\mathrm{sub}}(z_{\mathrm{SS}},n) &\ge \liminf_{p\to\infty}\left(\frac{\left(4p+\frac{3(p(p+1))}{2}\right)}{\left(4p+\frac{p(p+1)}{2}\right)}\right)=3,\\ \mathsf{AS}_{\mathrm{sub}}(z_{\mathrm{SS}},n) &\ge \left(4p+\frac{3(p(p+1))}{2}\right)-\left(4p+\frac{p(p+1)}{2}\right)=p(p+1)\\ &= 2z_{\mathrm{SS}}-\Theta(\sqrt{z_{\mathrm{SS}}})\in\Omega(\sqrt{n}).\end{aligned}$$

insertions: Let

$$T' = (Q_1\mathtt{a}_1\mathtt{0}\mathtt{1}Q_2)(\varepsilon\mathtt{a}_1\mathtt{1}\mathtt{b}_1)(\mathtt{a}_1\mathtt{a}_1\mathtt{1}\mathtt{b}_1\mathtt{b}_1)\cdots(Q_1[m{-}k{+}2..m]\mathtt{a}_1\mathtt{1}Q_2[1..k])\cdots(Q_1[2..m]\mathtt{a}_1\mathtt{1}Q_2)$$

be the string obtained from T by inserting 0 before the first 1. The non self-referencing LZSS factorization of T' is:

$$\begin{aligned}\mathsf{LZSS}(T') = &Q_1|\mathtt{a}_1|\mathtt{0}|\mathtt{1}|Q_2|\mathtt{a}_1|\mathtt{1}\mathtt{b}_1|\mathtt{a}_1\mathtt{a}_1|\mathtt{1}\mathtt{b}_1\mathtt{b}_1|\cdots\\ &|Q_1[m-k+2..m]\mathtt{a}_1|\mathtt{1}Q_2[1..k]|\cdots|Q_1[2..m]\mathtt{a}_1|\mathtt{1}Q_2|,\end{aligned}$$

where

$$\mathsf{LZSS}(Q_1) = \mathtt{a}_1|\cdots|\mathtt{a}_p|(\mathtt{a}_1\cdots\mathtt{a}_{p-1})|\cdots|(\mathtt{a}_1\mathtt{a}_2)|(\mathtt{a}_1)|$$

and

$$\mathsf{LZSS}(Q_2) = \mathtt{b}_1|\mathtt{b}_1|\mathtt{b}_2|\cdots|\mathtt{b}_1\cdots\mathtt{b}_{p-2}|\mathtt{b}_{p-1}|\mathtt{b}_1\cdots\mathtt{b}_{p-1}|\mathtt{b}_p|.$$

Then, $z_{\mathrm{SS}}(T') = 4p + p(p+1)$ holds. Hence, we obtain $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{ins}}(z_{\mathrm{SS}}, n) \geq 2$, $\mathsf{AS}_{\mathrm{ins}}(z_{\mathrm{SS}}, n) \geq z_{\mathrm{SS}} - \Theta(\sqrt{z_{\mathrm{SS}}})$, and $\mathsf{AS}_{\mathrm{ins}}(z_{\mathrm{SS}}, n) = \Omega(\sqrt{n})$.

deletions: As for deletions, by considering T' obtained from T by deleting the first $\mathtt{1}$, we get a similar decomposition to the case of substitutions. Thus, $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(z_{\mathrm{SS}}, n) \geq 3$, $\mathsf{AS}_{\mathrm{del}}(z_{\mathrm{SS}}, n) \geq 2z_{\mathrm{SS}} - \Theta(\sqrt{z_{\mathrm{SS}}})$, and $\mathsf{AS}_{\mathrm{del}}(z_{\mathrm{SS}}, n) = \Omega(\sqrt{n})$ also hold. □

3.6.2 Upper Bounds for Sensitivity of z_{SS}

Theorem 3.16. *The following upper bounds on the sensitivity of non self-referencing LZSS factorization* LZSS *hold:*
substitutions: $\limsup_{n\to\infty} \mathsf{MS}_{\mathrm{sub}}(z_{\mathrm{SS}}, n) \leq 3$. $\mathsf{AS}_{\mathrm{sub}}(z_{\mathrm{SS}}, n) \leq 2z_{\mathrm{SS}} - 2$.
insertions: $\mathsf{MS}_{\mathrm{ins}}(z_{\mathrm{SS}}, n) \leq 2$. $\mathsf{AS}_{\mathrm{ins}}(z_{\mathrm{SS}}, n) \leq z_{\mathrm{SS}}$.
deletions: $\limsup_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(z_{\mathrm{SS}}, n) \leq 3$. $\mathsf{AS}_{\mathrm{del}}(z_{\mathrm{SS}}, n) \leq 2z_{\mathrm{SS}} - 3$.

Proof. Let $\mathsf{LZSS}(T) = f_1 \cdots f_z$ and $\mathsf{LZSS}(T') = f'_1 \cdots f'_{z'}$. We denote the interval of the jth factor f_j (resp. f'_j) by $[p_j, q_j]$ (resp. $[p'_j, q'_j]$), namely $T[p_j..q_j] = f_j$ and $T'[p'_j..q'_j] = f'_j$. Also, let f_I be the factor of $\mathsf{LZSS}(T)$ whose interval $[p_I, q_I]$ contains the edited position i, namely $p_I \leq i \leq q_I$.

substitutions: In the following, we consider the case that the ith character $T[i] = a$ is substituted by a fresh character # which does not occur in T. The other cases can be proven similarly. Now we show the following claim:

Claim 3.3. *After the substitution, each interval* $[p_j, q_j]$ *has at most three starting positions* p'_k*,* p'_{k+1}*, and* p'_{k+2} *of factors in* $\mathsf{LZSS}(T')$ *for* $1 \leq k \leq z' - 2$.

Proof. There are the three following cases:

(i) When the interval $[p_j, q_j]$ satisfies $q_j < i$: By the same argument to Case (1) for LZ77, the interval $[p_j, q_j]$ contains exactly one starting position $p'_j = p_j$.

(ii) When the interval $[p_j, q_j]$ satisfies $p_j \leq i \leq q_j$ (namely, $f_j = f_I$): For the string $w_{j_1} a w_{j_2} = T[p_j..q_j]$, it is guaranteed that $w_1 a w_2$ has at least one occurrence in $f_1 \cdots f_{j-1}$. After the substitution which gives $T'[p_j..q_j] = w_1 \# w_2$, w_1 and # become factors as

f'_j and f'_{j+1}, and w_2 becomes the prefix of factor f'_{j+2}. This means that $p'_j = p_j$ and $q'_{j+2} \geq q_j$. Therefore, the interval $[p_j, q_j]$ contains at most three starting positions p'_j, p'_{j+1} and p'_{j+2} of factors in $\mathsf{LZSS}(T')$.

(iii) When the interval $[p_j, q_j]$ satisfies $i < p_j$: We consider the two following sub-cases:

(iii-A) When $T[p_j..q_j]$ has at least one occurrence which does not contain the edited position i in T: Any suffix of $T[p_j..q_j]$ still has a previous occurrence in T'. Therefore, $[p'_k, q'_k]$ with $p_j \leq p'_k$ satisfies $q'_k \geq q_j$, meaning the interval $[p_j, q_j]$ contains at most one starting position p'_k of a factor in $\mathsf{LZSS}(T')$.

(iii-B) All occurrences of $T[p_j..q_j]$ in T contain the edited position i: Let $u_1au_2 = T[p_j, q_j]$ with $a \in \Sigma$ and $u_1, u_2 \in \Sigma^*$. u_1 and u_2 have previous occurrences in $T'[1..p_j - 1]$. Let p'_k be the starting position of the leftmost factor of $\mathsf{LZ77}(T')$ which begins in range $[p_j, q_j]$. If p'_k is in u_2, then $q'_k \geq q_k$ and thus there is only one starting position of a factor of $\mathsf{LZ77}(T')$ in the interval $[p_j..q_j]$. Suppose p'_k is in u_1. If a has no previous occurrences (which happens when $T[i]$ was the only previous occurrence of a), then $T'[p_k + |u_1|]$ is the first occurrence of a in T' and thus $q'_k = p_k + |u_1| - 1$, $p'_{k+1} = q'_k + 1$ and $q'_{k+1} = p'_{k+1} + 1$. Otherwise, $q'_k \geq p_k + |u_1| - 1$, $p'_{k+1} \geq q'_k + 1$ and $q'_{k+1} \geq p'_{k+1} + 1$. In either case, since u_2 has a previous occurrence, $q'_{k+2} \geq q_{k+1}$. Thus, there can exist at most three starting positions of factors of $\mathsf{LZ77}(T')$ in the interval $[p_j..q_j]$.

This completes the proof for the claim. □

It follows from the above claim that $z_{\mathrm{SS}}(T') \leq 3z_{\mathrm{SS}}(T)$ for any string T and substitutions with #. Since $|f_1| = 1$, $z_{\mathrm{SS}}(T') \leq 3z_{\mathrm{SS}}(T) - 2$ holds. Thus we get $\limsup_{n\to\infty} \mathsf{MS}_{\mathrm{sub}}(z_{\mathrm{SS}}, n) \leq 3$ and $\mathsf{AS}_{\mathrm{sub}}(z_{\mathrm{SS}}, n) \leq 2z_{\mathrm{SS}} - 2$.

insertions: In the following, we consider the case that # is inserted to between positions $i - 1$ and i. The other cases can be proven similarly. Now we show the following claim:

Claim 3.4. *After the insertion, each interval $[p_j, q_j]$ contains at most two starting positions p'_k and p'_{k+1} of factors in $\mathsf{LZSS}(T')$ for $1 \leq k \leq z' - 1$, excluding the interval $[p_I, q_I]$. Also, the interval $[p_I, q_I]$ contains at most three starting positions of factors in $\mathsf{LZSS}(T')$.*

Proof. For Cases (i), (ii), and (iii-A), we can use the same discussions as in the case of substitutions. Now we consider Case (iii-B):

(iii-B) When all occurrences of $T[p_j..q_j]$ in T contain the edited position i: Let $u_1au_2 = T[p_j, q_j]$ with $a \in \Sigma$ and $w_1, w_2 \in \Sigma^*$. It is guaranteed that $w_{j_1}a$, and w_{j_2} still have previous occurrences in T'. Therefore, each range of $w_{j_1}a$ and w_{j_2} can contain at most one starting position of a factor in $\mathsf{LZSS}(T')$.

□

It follows from the above claim that $z_{\mathrm{SS}}(T') \leq 2z_{\mathrm{SS}}(T)+1$ holds any string T and insertions with #. By using the same discussion as for f_1, we obtain $z_{\mathrm{SS}}(T') \leq 2z_{\mathrm{SS}}(T)$ holds. Then we have $\mathsf{MS}_{\mathrm{ins}}(z_{\mathrm{SS}}, n) \leq 2$ and $\mathsf{AS}_{\mathrm{ins}}(z_{\mathrm{SS}}, n) \leq z_{\mathrm{SS}}$.

deletions: In the following, we consider the case that $T[i] = a$ is deleted. Now we show the following claim:

Claim 3.5. *After the deletion, each interval $[p_j, q_j]$ contains at most three starting positions p'_k, p'_{k+1}, and p'_{k+2} of factors in $\mathsf{LZSS}(T')$ for $1 \leq k \leq z' - 2$, excluding the interval $[p_I, q_I]$. The interval $[p_I, q_I]$ contains at most two starting positions of factors in $\mathsf{LZSS}(T')$.*

Proof. For Cases (i) and (iii), we can use the same discussions as in the case of substitutions. Now we consider case (ii):

(ii) When the interval $[p_j, q_j]$ satisfies $p_j \leq i \leq q_j$ (namely, $f_j = f_I$): Let $w_1aw_2 = T[p_j..q_j]$ with $a \in \Sigma$ and $w_1, w_2 \in \Sigma^*$. It is guaranteed that w_1aw_2 has at least one previous occurrence in $f_1 \cdots f_{j-1}$. Therefore, after the deletion of a, each range of w_1 and w_2 can contain at most one starting position of a factor in $\mathsf{LZSS}(T')$.

□

It follows from the above claim that $z_{\mathrm{SS}}(T') \leq 3z_{\mathrm{SS}}(T) - 1$ holds for any string T and deletions. By using the same discussion as for f_1, $z_{\mathrm{SS}}(T') \leq 3z_{\mathrm{SS}}(T) - 3$ holds. Then we get $\limsup_{n\to\infty} \mathsf{MS}_{\mathrm{del}}(z_{\mathrm{SS}}, n) \leq 3$ and $\mathsf{AS}_{\mathrm{del}}(z_{\mathrm{SS}}, n) \leq 2z_{\mathrm{SS}} - 3$. □

3.6.3 Lower bound for Sensitivity of z_{SSsr}

Theorem 3.17. *The following lower bounds on the sensitivity of self-referencing LZSS factorization LZSSsr hold:*

substitutions: $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{sub}}(z_{\mathrm{SSsr}}, n) \geq 3$. $\mathsf{AS}_{\mathrm{sub}}(z_{\mathrm{SSsr}}, n) \geq 2z_{\mathrm{SSsr}} - \Theta(\sqrt{z_{\mathrm{SSsr}}})$ *and* $\mathsf{AS}_{\mathrm{sub}}(z_{\mathrm{SSsr}}, n) = \Omega(\sqrt{n})$.

insertions: $\liminf_{n\to\infty} \mathsf{MS}_{\mathrm{ins}}(z_{\mathrm{SSsr}}, n) \geq 2$. $\mathsf{AS}_{\mathrm{ins}}(z_{\mathrm{SSsr}}, n) \geq z_{\mathrm{SSsr}} - \Theta(\sqrt{z_{\mathrm{SSsr}}})$ *and*

$\mathsf{AS}_{\text{ins}}(z_{\text{SSsr}}, n) = \Omega(\sqrt{n})$.
deletions: $\liminf_{n\to\infty} \mathsf{MS}_{\text{del}}(z_{\text{SSsr}}, n) \geq 3$. $\mathsf{AS}_{\text{del}}(z_{\text{SSsr}}, n) \geq 2z_{\text{SSsr}} - \Theta(\sqrt{z_{\text{SSsr}}})$ *and*
$\mathsf{AS}_{\text{del}}(z_{\text{SSsr}}, n) = \Omega(\sqrt{n})$.

Proof. We use the same strings T and T' as in the proof for Theorem 3.15 which shows the lower bounds of the sensitivity of the non self-referencing LZSS. For the string T and each edit operation, the self-referencing LZSS factorization is the same as the non self-referencing LZSS factorization. Hence, we obtain Theorem 3.17. □

3.6.4 Upper Bounds for Sensitivity of z_{SSsr}

Theorem 3.18. *The following upper bounds on the sensitivity of self-referencing LZSS factorization* LZSSsr *hold:*
substitutions: $\mathsf{MS}_{\text{sub}}(z_{\text{SSsr}}, n) \leq 3$. $\mathsf{AS}_{\text{sub}}(z_{\text{SSsr}}, n) \leq 2z_{\text{SSsr}}$.
insertions: $\limsup_{n\to\infty} \mathsf{MS}_{\text{ins}}(z_{\text{SSsr}}, n) \leq 2$. $\mathsf{AS}_{\text{ins}}(z_{\text{SSsr}}, n) \leq z_{\text{SSsr}} + 1$.
deletions: $\mathsf{MS}_{\text{del}}(z_{\text{SSsr}}, n) \leq 3$. $\mathsf{AS}_{\text{del}}(z_{\text{SSsr}}, n) \leq 2z_{\text{SSsr}} - 1$.

Proof. We use the same notations as in Theorem 3.16 of Section 3.6.2. As with the case of self-referencing LZ77 $z_{77\text{sr}}$, only the interval $[p_I, q_I]$ that contains the edited position i is effected in this case of self-referencing LZSS z_{SSsr}. For the string $w_1 a w_2 = T[p_I..q_I]$, only w_2 can have a self-referencing previous occurrence that contains the edited position i. For each edit operation, by applying the discussion of Case (iii) in Theorem 3.16 to the range of w_2 in $[p_I..q_I]$, we obtain Theorem 3.18. □

3.7 Lempel-Ziv 78 Factorizations

In this section, we consider the worst-case sensitivity of the *Lempel-Ziv 78 factorizations* (*LZ78*) [124].

For convenience, let $f_0 = \varepsilon$. A factorization $T = f_1 \cdots f_{z_{78}}$ for a string T of length n is the LZ78 factorization $\mathsf{LZ78}(T)$ of T if for each $1 \leq i < z_{78}$ the factor f_i is the longest prefix of $f_i \cdots f_{z_{78}}$ such that $f_i[1..|f_i| - 1] = f_j$ for some $0 \leq j < i$. The last factor $f_{z_{78}}$ is the suffix of T of length $n - |f_1 \cdots f_{z_{78}-1}|$ and it may be equal to some previous factor f_j ($1 \leq j < z_{78}$). Again, if we use a common convention that the string T terminates with a unique character \$, then the last factor $f_{z_{78}}$ can be defined analogously to the previous factors. Let $z_{78}(T)$ denote the number of factors in the LZ78 factorization of string T.

For example, for string $T = \texttt{abaababababababab\$}$,

$$\mathsf{LZ78}(T) = \texttt{a}|\texttt{b}|\texttt{aa}|\texttt{ba}|\texttt{bab}|\texttt{ab}|\texttt{aba}|\texttt{b\$}|,$$

where $|$ denotes the right-end of each factor in the factorization. Here we have $z_{78}(T) = 8$.

As for the sensitivity of LZ78, Lagarde and Perifel [83] showed that $\mathsf{MS}_{\mathrm{ins}}(z_{78}, n) = \Omega(n^{1/4})$, $\mathsf{AS}_{\mathrm{ins}}(z_{78}, n) = \Omega(z_{78}^{3/2})$, and $\mathsf{AS}_{\mathrm{ins}}(z_{78}, n) = \Omega(n/\log n)$ for insertions. [1] In this section, we present lower bounds for the multiplicative/additive sensitivity of LZ78 for the remaining cases, i.e., for substitutions and deletions, by using a completely different string from [83].

3.7.1 Lower Bounds for Sensitivity of z_{78}

Theorem 3.19. *The following lower bounds on the sensitivity of z_{78} hold:*
substitutions: $\mathsf{MS}_{\mathrm{sub}}(z_{78}, n) = \Omega(n^{1/4})$. $\mathsf{AS}_{\mathrm{sub}}(z_{78}, n) = \Omega(z_{78}^{3/2})$ *and* $\mathsf{AS}_{\mathrm{sub}}(z_{78}, n) = \Omega(n^{3/4})$.
deletions: $\mathsf{MS}_{\mathrm{del}}(z_{78}, n) = \Omega(n^{1/4})$. $\mathsf{AS}_{\mathrm{del}}(z_{78}, n) = \Omega(z_{78}^{3/2})$ *and* $\mathsf{AS}_{\mathrm{del}}(z_{78}, n) = \Omega(n^{3/4})$.

Proof. Consider the string

$$T = (\sigma_{k+1}) \cdots (\sigma_{2k}) \cdot (\sigma_1) \cdot (\sigma_1\sigma_2) \cdots (\sigma_1 \cdots \sigma_k) \cdot (\sigma_1 \cdots \sigma_{y_1} \cdot \sigma_{k+1}) \cdots (\sigma_1 \cdots \sigma_{y_k} \cdot \sigma_{2k}),$$

where σ_i for every $1 \le i \le 2k$ is a distinct character and y_j for every $1 \le j \le k$ satisfies the following property: y_j is the maximum integer at most k such that $2 + j + \ell_j - 1 \equiv y_j \pmod{\ell_j}$ where ℓ_j is an integer satisfying $(1/2)\ell_j(\ell_j - 1) + 1 \le j \le (1/2)\ell_j(\ell_j + 1)$. We remark that the parentheses (and) in T are shown only for the better visualization and exposition, and therefore they are not the characters in T.

Let n be the length of T. Since $k + (1/2)k(k+1) < n < k + (1/2)k(k+1) + k(k+1)$, $n \in \Theta(k^2)$ holds. In the LZ78 factorization of T, for each substring (w), its suffix $w[1..|w|-1]$ has a previous occurrence as $(w[1..|w|-1])$, and (w) is the leftmost occurrence of w in the string T. Therefore, the LZ78 factorization of T is

$$\mathsf{LZ78}(T) = \sigma_{k+1}| \cdots |\sigma_{2k}|\sigma_1|\sigma_1\sigma_2| \cdots |\sigma_1 \cdots \sigma_k|\sigma_1 \cdots \sigma_{y_1} \cdot \sigma_{k+1}| \cdots |\sigma_1 \cdots \sigma_{y_k} \cdot \sigma_{2k}|$$

with $z_{78}(T) = 3k$.

For our analysis of the sensitivity of z_{78} for substitutions, consider the string

$$T' = (\sigma_{k+1}) \cdots (\sigma_{2k}) \cdot (\sigma_1) \cdot (\sigma_1\sigma_2) \cdots (\sigma_1 \cdots \sigma_k) \cdot (\#\sigma_2 \cdots \sigma_{y_1} \cdot \sigma_{k+1}) \cdots (\sigma_1 \cdots \sigma_{y_k} \cdot \sigma_{2k}),$$

[1] In the restricted case of appending a character to the top of a string or deleting the first character of a string, they showed upper bounds that the ratio is $O(n^{1/4})$ and the increase is $O(z_{78}^{3/2})$.

which can be obtained from T by substituting the first character σ_1 of the string in the $2k+1$th paring parentheses with a fresh character $\#$, which does not occur in T. Let us analyze the structure of the LZ78 factorization of T'. Clearly, the first $2k$ factors are unchanged after the substitution. Next, we consider $(\#\sigma_2 \cdots \sigma_{y_1} \cdot \sigma_{k+1})$. First, the prefix $\#\sigma_2 \cdots \sigma_{y_1}$ is decomposed into y_1 factors of length 1. The next factor is $\sigma_{k+1}\sigma_1$ since σ_{k+1} has an occurrence as a previous factor and $\sigma_{k+1}\sigma_1$ has no occurrences as a previous factor. Now we show in each interval of the $2k+j$th paring parentheses for $2 \le j \le k$ (i.e., the interval of $\sigma_1 \cdots \sigma_{y_j}\sigma_{k+j}$) there appear the right-ends $|$ of factors in $\mathsf{LZ78}(T')$ as follows:

$$\sigma_1|\sigma_2 \cdots \sigma_{j+1}|\sigma_{j+2} \cdots \sigma_{j+\ell_j+1}| \cdots |\sigma_{y_j-\ell_j+1} \cdots \sigma_{y_j}|\sigma_{k+j}. \tag{3.1}$$

Namely, the interval is decomposed into $3 + ((y_j - j - \ell_j - 1)/\ell_j) + 1 = 3 + (y_j - j - 1)/\ell_j$ pieces $d_1, \ldots, d_{3+(y_j-j-1)/\ell_j}$, where $|d_1| = |d_{3+(y_j-j-1)/\ell_j}| = 1$, $|d_2| = j$, and each of the others is of length ℓ_j. At first, we show the partition the partition (3.1) is valid for $j = 2$. As mentioned above, there is a factor $\sigma_{k+1}\sigma_1$ constructed with the immediately preceded character and the first character of the interval of $\sigma_1 \cdots \sigma_{y_2}\sigma_{k+2}$. And then, $\sigma_2 \cdots \sigma_{y_2}$ is decomposed into $\sigma_2\sigma_3|\sigma_4\sigma_5| \cdots |\sigma_{y_2-1}\sigma_{y_2}|$ since y_2 is the maximum odd value less than or equal to k and each σ_{2i} for $1 \le i \le (y_2 - 1)/2$ is the longest prefix as some previous factor. Since $\ell_2 = 2$ holds, the partitions of the interval are $\sigma_1|\sigma_2\sigma_3|\sigma_4\sigma_5| \cdots |\sigma_{y_2-1}\sigma_{y_2}|\sigma_{k+2}$, and this satisfies the partition (3.1). Next, we assume the partition (3.1) is valid for $j \le h - 1$ for some integer h, and we consider whether the partition (3.1) is valid or not for $j = h$. From the assumption and the same discussion as the above, there is a factor $\sigma_{k+h-1}\sigma_1$ constructed with the immediately preceded character and the first character of the interval of $\sigma_1 \cdots \sigma_{y_h}\sigma_{k+h}$. Since the set of previous factors starting with the character σ_2 is $\{(\sigma_2), (\sigma_2\sigma_3), \ldots, (\sigma_2 \cdots \sigma_h)\}$, the next factor becomes $\sigma_2 \cdots \sigma_{h+1}$. In addition, it is guaranteed that the set of previous factors starting with the character σ_i for every $h + 2 \le i \le y_h - \ell_h + 1$ is equal to $\{(\sigma_i), (\sigma_i\sigma_{i+1}), \ldots, (\sigma_i \cdots \sigma_{i+\ell_j} - 2\}$. Since y_h can be described as $h + \ell_h + 1 + t\ell_h$ for some integer t, the decomposition of the interval becomes $\sigma_1|\sigma_2 \cdots \sigma_{h+1}|\sigma_{h+2} \cdots \sigma_{h+\ell_h+1}| \cdots |\sigma_{y_h-\ell_h+1} \cdots \sigma_{y_h}|\sigma_{k+h}$, and this satisfies the partition (3.1). From the above, the partition (3.1) is valid for $2 \le j \le k$ by induction.

Thus, the LZ78 factorization of T' is

$$\begin{aligned}\mathsf{LZ78}(T') = \sigma_{k+1}| \cdots |\sigma_{2k}|\sigma_1|\sigma_1\sigma_2| \cdots |\sigma_1 \cdots \sigma_k|\#|\sigma_2| \cdots |\sigma_{y_1}|\sigma_{k+1}\sigma_1|\sigma_2\sigma_3| \cdots |\sigma_{y_2-1}\sigma_{y_2}| \\ \sigma_{k+2}\sigma_1| \cdots |\sigma_2 \cdots \sigma_{j+1}|\sigma_{j+2} \cdots \sigma_{j+\ell_j+1}| \cdots |\sigma_{y_j-\ell_j+1} \cdots \sigma_{y_j}|\sigma_{k+j}\sigma_1| \cdots | \cdots \sigma_{y_k}|\sigma_{2k}|.\end{aligned}$$

See also Figure 3.2 for a concrete example.

The size of $\mathsf{LZ78}(T')$ is $z_{78}(T') = 2k + y_1 + \sum_{j=2}^{k}(2 + (y_j - j - 1)/\ell_j) + 1 = 5k - 1 + \sum_{j=2}^{k}((y_j - j - 1)/\ell_j)$. $\sum_{j=2}^{k}((y_j - j - 1)/\ell_j)$ is the total number of factors of length

ℓ_j for $1 \leq j \leq k$. Now we consider the number of factors of length in $L \in \{\ell_1, \ldots, \ell_k\}$. For all j such that $(1/2)L(L-1)+1 \leq j \leq (1/2)L(L+1)$, the total number of factors of length in L is $\sum_{j=j_{\min}^L}^{j_{\max}^L}((y_j - j - 1)/L)$, where $j_{\min}^L = (1/2)L(L-1)+1$ and $j_{\max}^L = (1/2)L(L+1)$. From the definition of y_j, $\sum_j y_j = k + (k-1) + \cdots + (k-L+1)$ holds. Therefore, $\sum_{j=j_{\min}^L}^{j_{\max}^L}((y_j - j - 1)/L) = (k + (k-1) + \cdots + (k-L+1) - j_{\min}^L - (j_{\min}^L + 1) - \cdots - j_{\max}^L - L)/L = (L(k - j_{\max}^L) - L)/L = k - (1/2)L(L+1) - 1$. Let $\ell_k = m$. Then the total number of factors of length ℓ_j for $1 \leq j \leq k$ is $\sum_{L=2}^{m}(k - (1/2)L(L+1) - 1) = (m-1)k - (1/12)m(m+1)(2m+1) - (1/4)m(m+1) - m + 2$. Consider the case of $m = \sqrt{k}$, then $z_{78}(T') \in \Omega(k\sqrt{k})$. Thus we obtain $\mathsf{MS}_{\mathrm{sub}}(z_{78}, n) = \Omega(n^{1/4})$, $\mathsf{AS}_{\mathrm{sub}}(z_{78}, n) = \Omega(z_{78}^{3/2})$, and $\mathsf{AS}_{\mathrm{sub}}(z_{78}, n) = \Omega(n^{3/4})$.

As for deletions, by considering T' obtained from T by deleting the first character of the $2k+1$th factor in $\mathsf{LZ78}(T)$, we obtain a similar decomposition as the above. Thus, $\mathsf{MS}_{\mathrm{del}}(z_{78}, n) = \Omega(n^{1/4})$, $\mathsf{AS}_{\mathrm{del}}(z_{78}, n) = \Omega(z_{78}^{3/2})$, and $\mathsf{AS}_{\mathrm{del}}(z_{78}, n) = \Omega(n^{3/4})$ also hold. $\square$

$\mathsf{LZ78}(T') = \sigma_{51} \mid \cdots \mid \sigma_{100} \mid \sigma_1 \mid \sigma_1\sigma_2 \mid \cdots \mid \sigma_1\cdots\sigma_{50} \mid$
$\# \mid \sigma_2 \mid \sigma_3 \mid \cdots \mid \sigma_{50} \mid \sigma_{51}\sigma_1 \mid$
$\sigma_2\sigma_3 \mid \sigma_4\sigma_5 \mid \cdots \mid \sigma_{49}\sigma_{50} \mid \sigma_{52}\sigma_1 \mid$
$\sigma_2\sigma_3\sigma_4 \mid \sigma_5\sigma_6 \mid \cdots \mid \sigma_{48}\sigma_{49} \mid \sigma_{53}\sigma_1 \mid$
$\sigma_2\sigma_3\sigma_4\sigma_5 \mid \sigma_6\sigma_7\sigma_8 \mid \cdots \mid \sigma_{48}\sigma_{49}\sigma_{50} \mid \sigma_{54}\sigma_1 \mid$
$\sigma_2\sigma_3\sigma_4\sigma_5\sigma_6 \mid \sigma_7\sigma_8\sigma_9 \mid \cdots \mid \sigma_{46}\sigma_{47}\sigma_{48} \mid \sigma_{55}\sigma_1 \mid$
$\sigma_2\sigma_3\sigma_4\sigma_5\sigma_6\sigma_7 \mid \sigma_8\sigma_9\sigma_{10} \mid \cdots \mid \sigma_{47}\sigma_{48}\sigma_{49} \mid \sigma_{56}\sigma_1 \mid$
...

Figure 3.2: Illustration for $\mathsf{LZ78}(T')$ for the string T' of Theorem 3.19 with $k = 50$.

We remark that our string also achieve $\mathsf{MS}_{\mathrm{ins}}(z_{78}, n) = \Omega(n^{1/4})$, $\mathsf{AS}_{\mathrm{ins}}(z_{78}, n) = \Omega(z_{78}^{3/2})$, and $\mathsf{AS}_{\mathrm{ins}}(z_{78}, n) = \Omega(n^{3/4})$ for insertions, if we consider the string T' obtained from T by inserting $\#$ between the first and second characters of the $2k+1$th factor of $\mathsf{LZ78}(T)$.

Chapter 4

Longest Palindromic Substring After Edit

Finding palindromic structures has important applications to analyze biological data such as DNA, RNA, and proteins [57]. In this chapter, we tackle the problems of computing the longest palindromic substring (LPS) after the string is edited. Such a setting can be utilized in several situations in the real-world, e.g., a substring of a text having errors, simulations on a text, and the analysis of some similar strings. In Section 4.1, we show how to compute the longest substring palindrome of a string after a single character edit operation. In Section 4.2, we show algorithm for computing the longest substring palindrome of a string after block-wise edit operation.

The results in this chapter primarily appeared in [42].

4.1 Algorithm for 1-ELPS

In this section, we will show the following result:

Theorem 4.1. *There is an algorithm for the 1-ELPS problem which uses $O(n)$ time and space for preprocessing, and answers each query in $O(\log(\min\{\sigma, \log n\}))$ time for single character substitution and insertion, and in $O(1)$ time for single character deletion.*

4.1.1 Algorithm for Substitutions

In what follows, we will present our algorithm to compute the length of the LPSs after a single character substitution. Our algorithm can also return an occurrence of an LPS.

Let i be any position in the string T of length n and let $c = T[i]$. Also, let $T' = T[1..i-1]c'T[i+1..n]$, i.e., T' is the string obtained by substituting character c' for the original character $c = T[i]$ at position i. In this subsection, we assume $c \neq c'$ without loss of generality. To compute the length of the LPSs of T', it suffices to consider maximal palindromes of T'. Those maximal palindromes of T' will be computed from the maximal palindromes of T.

The following observation shows that some maximal palindromes of T remain unchanged after a character substitution at position i.

Observation 4.1 (Unchanged maximal palindromes after a single character substitution)**.** *For any position $1 \leq j < i-1$, $\mathit{MaxPalEnd}_{T'}(j) = \mathit{MaxPalEnd}_T(j)$. For any position $i+1 < j \leq n$, $\mathit{MaxPalBeg}_{T'}(j) = \mathit{MaxPalBeg}_T(j)$.*

By Observation 4.1, for each position i ($1 \leq i \leq n$) of T, we precompute the largest element of $\bigcup_{1 \leq j < i-1} \mathit{MaxPalEnd}_T(j)$ and that of $\bigcup_{i+1 < j \leq n} \mathit{MaxPalBeg}_T(j)$, and store the larger one in the ith position of an array $\mathcal{U}$ of length n. $\mathcal{U}[i]$ is a candidate for the solution after the substitution at position i. For each position i, $\bigcup_{1 \leq j < i-1} \mathit{MaxPalEnd}_T(j)$ contains the lengths of all maximal palindromes which end to the left of i, and $\bigcup_{i+1 < j \leq n} \mathit{MaxPalBeg}_T(j)$ contains the lengths of all maximal palindromes which begin to the right of i. Thus, by simply scanning $\mathit{MaxPalEnd}_T(j)$ for increasing $j = 1, \ldots, n$ and $\mathit{MaxPalBeg}_T(j)$ for decreasing $j = n, \ldots, 1$, we can compute $\mathcal{U}[i]$ for every position $1 \leq i \leq n$. Since there are only $2n-1$ maximal palindromes in string T, it takes $O(n)$ time to compute the whole array $\mathcal{U}$.

Next, we consider maximal palindromes of the original string T whose lengths are extended in the edited string T'. As above, let i be the position where a new character c' is substituted for the original character $c = T[i]$. In what follows, let σ denote the number of distinct characters appearing in T.

Observation 4.2 (Extended maximal palindromes after a single character substitution)**.** *For any $s \in \mathit{MaxPalEnd}_T(i-1)$, the corresponding maximal palindrome $T[i-s..i-1]$ centered at $\frac{2i-s-1}{2}$ gets extended in T' iff $T[i-s-1] = c'$. Similarly, for any $p \in \mathit{MaxPalBeg}_T(i+1)$, the corresponding maximal palindrome $T[i+1..i+p]$ centered at $\frac{2i+p+1}{2}$ gets extended in T' iff $T[i+p+1] = c'$.*

Lemma 4.1. *Let T be a string of length n over an integer alphabet of size polynomial in n. It is possible to preprocess T in $O(n)$ time and space so that later we can compute in $O(\log(\min\{\sigma, \log n\}))$ time the length of the longest maximal palindromes in T' that are extended after a substitution of a character.*

Proof. In what follows, we consider maximal palindromes corresponding to $\mathit{MaxPalEnd}_T(i-1)$ by Observation 4.2. Those corresponding to $\mathit{MaxPalBeg}_T(i+1)$ can be treated similarly. Let $\langle s, d, t \rangle$ be an arithmetic progression representing a group of maximal palindromes in

$\mathit{MaxPalEnd}_T(i-1)$. Let us assume that the group contains more than 1 member (i.e., $t \geq 2$) and that $i - s \geq 2$, since the case where $t = 1$ or $i - s = 1$ is easier to deal with. Let P_j denote the jth shortest member of the group, i.e., $P_1 = T[i-s..i-1]$ and $P_t = T[i-s-(t-1)d..i-1]$. Then, it follows from Lemma 2.2 (iv) that if a is the character immediately preceding the occurrence of P_1 (i.e., $a = T[i-s-1]$), then a also immediately precedes the occurrences of $P_2, \ldots, P_{t-1}$. Hence, by Observation 4.2, P_j $(2 \leq j < t)$ gets extended in the edited text T' iff $c' = a$. Similarly, P_t gets extended iff $c' = b$, where b is the character immediately preceding the occurrence of P_t. For each $1 \leq j \leq t$ the final length of the extended maximal palindrome can be computed in $O(1)$ time by a single outward LCE query $\mathsf{OutLCE}(i-s-(j-1)d-2, i+1)$. Let P'_j denote the extended maximal palindrome for each $1 \leq j \leq t$. Since there are only $2n-1$ maximal palindromes in string T and all of them can be computed in $O(n)$ total time.

The above arguments suggest that for each group of maximal palindromes, there are at most two distinct characters that can extend those palindromes after a single character substitution. For each position i in T, let Σ_i denote the set of characters which can extend maximal palindromes w.r.t. $\mathit{MaxPalEnd}_T(i-1)$ after a character substitution at position i. It now follows from Lemma 2.2 and from the above arguments that $|\Sigma_i| = O(\min\{\sigma, \log i\})$. Also, when any character in $\Sigma \setminus \Sigma_i$ is given for character substitution at position i, then no maximal palindromes w.r.t. $\mathit{MaxPalEnd}_T(i-1)$ are extended.

For each maximal palindrome P of T, let (i', c, l) be a tuple such that i' is the ending position of P, and l is the length of the extended maximal palindrome P' after the immediately following character $T[i'+1]$ is substituted for the character $c = T[i'-|P|-1]$ which immediately precedes the occurrence of P in T. We then radix-sort the tuples (i', c, l) for all maximal palindromes in T as 3-digit numbers. This can be done in $O(n)$ time since T is over an integer alphabet of size polynomial in n. Then, for each position i', we compute the maximum value l_c for each character c. Since we have sorted the tuples (i', c, l), this can also be done in total $O(n)$ time for all positions and characters.

Let $\hat{c}$ be a special character which represents any character in $\Sigma \setminus \Sigma_i$ (if $\Sigma \setminus \Sigma_i \neq \emptyset$). Since no maximal palindromes w.r.t. $\mathit{MaxPalEnd}_T(i-1)$ are extended by $\hat{c}$, we associate $\hat{c}$ with the length $l_{\hat{c}}$ of the longest maximal palindrome w.r.t. $\mathit{MaxPalEnd}_T(i-1)$. For each position i and each character $c \in \Sigma_i$, if l_c is less than $l_{\hat{c}}$, then we rewrite $l_c = l_{\hat{c}}$. We assume that $\hat{c}$ is lexicographically larger than any characters in Σ_i. For each position i we store pairs (c, l_c) in an array $\mathcal{E}_i$ of size $|\Sigma_i| + 1 = O(\min\{\sigma, \log i\})$ in lexicographical order of c. See Figure 4.1 for a concrete example.

Then, given a character c' to substitute for the character at position i $(1 \leq i \leq n)$, we can

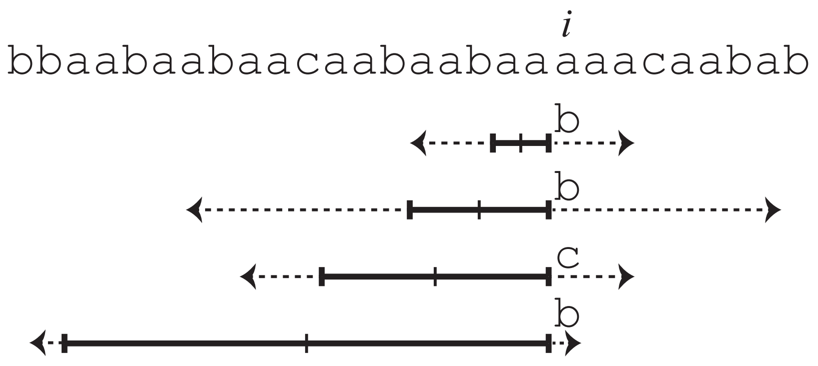


Figure 4.1: Example for Lemma 4.1, with string bbaabaabaacaabaabaaaaacaabab where the character a at position $i = 20$ is to be substituted. There are four maximal palindromes ending at position 19, whose lengths are represented by two groups $\langle 2, 3, 3\rangle$ and $\langle 17, 9, 1\rangle$. For the first group, c precedes the longest maximal palindrome and b precedes all the other maximal palindromes. The second group contains only one maximal palindrome and b precedes it. The largest extended lengths are 21 for b, and 14 for c. Thus we have $\mathcal{E}_i = [(\mathtt{b}, 21), (\mathtt{c}, 17), (\hat{c}, 17)]$, where 17 is the length of the longest maximal palindrome ending at position 19 in the original string.

binary search $\mathcal{E}_i$ for $(c', l_{c'})$ in $O(\log(\min\{\sigma, \log n\}))$ time. If c' is not found in the array, then we take the pair $(\hat{c}, l_{\hat{c}})$ from the last entry of $\mathcal{E}_i$. We remark that $\sum_{i=1}^{n} |\mathcal{E}_i| = O(n)$ since there are $2n - 1$ maximal palindromes in T and for each of them at most two distinct characters contribute to $\sum_{i=1}^{n} |\mathcal{E}_i|$. □

By using hashing instead of binary search, the query can be solved in $O(1)$ time after $O(n)$ expected time and $O(n)$ space for preprocessing.

Finally, we consider maximal palindromes of the original string T whose lengths are shortened in the edited string T' after substituting a character c' for the original character at position i.

Observation 4.3 (Shortened maximal palindromes after a single character substitution)**.** *A maximal palindrome $T[b..e]$ of T gets shortened in T' iff $b \leq i \leq e$ and $i \neq \frac{b+e}{2}$.*

Lemma 4.2. *It is possible to preprocess a string T of length n in $O(n)$ time and space so that later we can compute in $O(1)$ time the length of the longest maximal palindromes of T' that are shortened after a substitution of a character.*

Proof. Now we consider shortened maximal palindromes whose center $\frac{b+e+1}{2}$ is less than i. Shortened maximal palindromes whose center $\frac{b+e+1}{2}$ is more than i can be treated similarly. Let

$\mathcal{S}$ be an array of length n such that $\mathcal{S}[i]$ stores the length of the longest maximal palindrome after shortening by the character substitution at position i. To compute $\mathcal{S}$, we preprocess T by scanning it from left to right. Suppose that we have computed $\mathcal{S}[i]$. By Observation 4.3, we have that $\mathcal{S}[i] = 2(i - \frac{b+e+1}{2})$ where $T[b..e]$ is the longest maximal palindrome of T satisfying the conditions of Observation 4.3. In other words, $T[b..e]$ is the maximal palindrome of T of which the center $\frac{b+e}{2}$ is the smallest possible under the conditions.

For any position $i < i'' \leq e$, we have that $\mathcal{S}[i''] = \mathcal{S}[i] + 2(i'' - i)$. For the next position $e + 1$, we can compute $\mathcal{S}[e + 1]$ in amortized $O(1)$ time by simply scanning the array $\mathcal{M}$ from position $\frac{b+e+1}{2}$ to the right until finding the first (i.e., leftmost) entry of $\mathcal{M}$ which stores the length of a maximal palindrome whose ending position is at least $e+1$. Hence, we can compute $\mathcal{S}$ in $O(n)$ total time and space. □

Remark that maximal palindromes of T which do not satisfy the conditions of Observations 4.2 and 4.3 are also unchanged in T'. The following lemma summarizes this subsection:

Lemma 4.3. *Let T be a string of length n over an integer alphabet of size polynomial in n. It is possible to preprocess T of length n in $O(n)$ time and space so that later we can compute in $O(\log(\min\{\sigma, \log n\}))$ time the length of the LPSs of the edited string T' after a substitution of a character.*

4.1.2 Algorithm for Deletions

Suppose the character at position i is deleted from the string T, and let T'_i denote the resulting string, namely $T'_i = T[1..i-1]T[i+1..n]$. Now the RL factorization of T comes into play: Observe that for any $1 \leq i \leq n$, $T'_i = T'_{RLFBeg(i)} = T'_{RLFEnd(i)}$. Thus, it suffices for us to consider only the boundaries of the RL factors for T.

It is easy to see that an analogue of Observation 4.1 for unchanged maximal palindromes holds, as follows.

Observation 4.4 (Unchanged maximal palindromes after a single character deletion)**.** *For any position $1 \leq j < RLFEnd(i) - 1$, $MaxPalEnd_{T'_i}(j) = MaxPalEnd_T(j)$. For any position $RLFBeg(i) + 1 < j \leq n$, $MaxPalBeg_{T'_i}(j) = MaxPalBeg_T(j)$.*

See Figure 4.2 for a concrete example of Observation 4.4.

By the above observation, we can compute the lengths of the longest unchanged maximal palindromes for the boundaries of all RL factors in $O(n)$ time, in a similar way to the case of substitution.

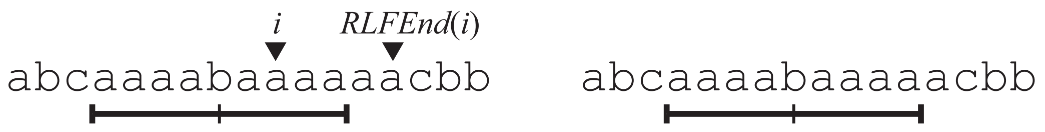


Figure 4.2: Example for Observation 4.4. The maximal palindrome aaaabaaaa does not change if the character a at position i is deleted. The result is the same if the character a at position $RLFEnd(i)$ is deleted.

Clearly the new character at position $RLFEnd(i)$ in the string T_i' after a deletion is always $T[RLFEnd(i)+1]$, and a similar argument holds for $RLFBeg(i)$. Thus, we have the following observation for extended maximal palindromes after a deletion, which is an analogue of Observation 4.2.

Observation 4.5 (Extended maximal palindromes after a single character deletion)**.** *For any $s \in MaxPalEnd_T(RLFEnd(i)-1)$, the corresponding maximal palindrome $T[RLFEnd(i)-s..RLFEnd(i)-1]$ centered at $\frac{2RLFEnd(i)-s-1}{2}$ gets extended in T_i' iff $T[RLFEnd(i)-s-1] = T[RLFEnd(i)+1]$. Similarly, for any $p \in MaxPalBeg_T(RLFBeg(i)+1)$, the corresponding maximal palindrome $T[RLFBeg(i)+1..RLFBeg(i)+p]$ centered at $\frac{2RLFBeg(i)+p+1}{2}$ gets extended in T_i' iff $T[RLFBeg(i)+p+1] = T[RLFBeg(i)-1]$.*

See Figure 4.3 for a concrete example for Observation 4.5.

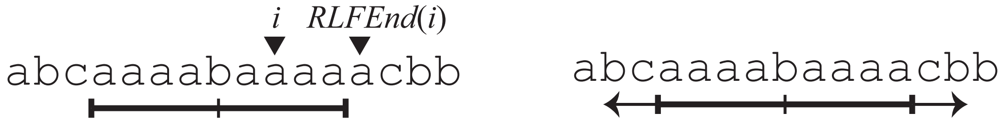


Figure 4.3: Example for Observation 4.5. The maximal palindrome aaaabaaaa gets extended to bcaaaabaaaacb if the character a at position i is deleted. The result is the same if the character a at position $RLFEnd(i)$ is deleted.

Since the new characters that come from the left and the right of each deleted position are always unique, for each $RLFEnd(i)$ and $RLFBeg(i)$, the longest maximal palindrome that gets extended after a deletion is also unique. Overall, we can precompute their lengths for all positions $1 \le i \le n$ in $O(n)$ total time by using $O(n)$ outward LCE queries in the original string T.

Next, we consider those maximal palindromes which get shortened after a single character deletion. We have the following observation which is analogue to Observation 4.3.

Observation 4.6 (Shortened maximal palindromes after a deletion)**.** *A maximal palindrome $T[b..e]$ of T gets shortened in T'_i iff $b \leq RLFBeg(i)$ and $RLFEnd(i) \leq e$.*

See Figure 4.4 for a concrete example for Observation 4.6.

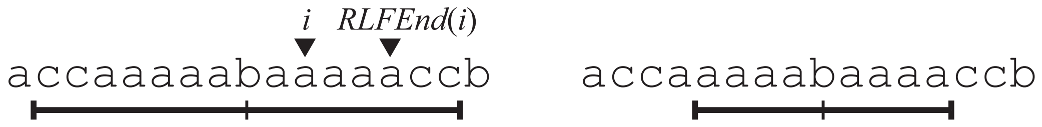


Figure 4.4: Example for Observation 4.6. The maximal palindrome `ccaaaaabaaaaacc` gets shortened to `aaaabaaaa` if the character `a` at position i is deleted. The result is the same if the character `a` at position $RLFEnd(i)$ is deleted.

By Observation 4.6, we can precompute the length of the longest maximal palindrome after deleting the characters at the beginning and ending positions of each RL factors in $O(n)$ total time, using an analogous way to Lemma 4.2.

Summing up all the above discussions, we obtain the following lemma:

Lemma 4.4. *It is possible to preprocess a string T of length n in $O(n)$ time and space so that later we can compute in $O(1)$ time the length of the LPSs of the edited string T'_i after a deletion of a character.*

4.1.3 Algorithm for Insertion

Consider the insertion of a new character c' between the ith and $(i+1)$th positions in T, and let $T' = T[1..i]c'T[i+1..n]$. If $c' \neq T[i]$ and $c' \neq T[i+1]$, we can find the length of the LPSs in T' in a similar way to substitution as follows: for the maximal palindromes in T' whose center is less than or equal to i, we regard as c' is substituted for $T[i+1]$. Then, we can compute shortened or unchanged maximal palindromes by using exactly the same algorithm for substitution. Extended maximal palindromes also can be computed in a similar way to substitution by taking care of the positions of outward LCE queries. The maximal palindromes in T' whose center is more than or equal to $i+2$ can be computed similarly by regarding as c' is substituted for $T[i]$. The remaining maximal palindromes in T' with the center $i+0.5$, $i+1$, or $i+1.5$ can be computed easily. The length of the maximal palindrome in T' with the center $i+1$ is equal to it of the maximal palindrome of T with the center $i+0.5$ plus one. Also, the maximal palindromes in T' with the center $i+0.5$ or $i+1.5$ are ε. Otherwise (if $c' = T[i]$ or $c' = T[i+1]$), we can find the length of the LPSs in T' in a similar way to deletion since c' is merged to an adjacent RL factor. Thus, we have the following.

Lemma 4.5. *Let T be a string of length n over an integer alphabet of size polynomial in n. It is possible to preprocess in $O(n)$ time and space string T so that later we can compute in $O(\log(\min\{\sigma, \log n\}))$ time the length of the LPSs of the edited string T' after a insertion of a character.*

4.1.4 Hashing

By using hashing instead of binary searches on arrays, the following corollary is immediately obtained from Theorem 4.1.

Corollary 4.1. *There is an algorithm for the 1-ELPS problem which uses $O(n)$* expected *time and $O(n)$ space for preprocessing, and answers each query in $O(1)$ time for single character substitution, insertion, and deletion.*

4.2 Algorithm for ℓ-ELPS

In this section, we consider the ℓ-ELPS problem where an existing block of length ℓ' in the string T is replaced with a new block of length ℓ. This generalizes substitution when $\ell' > 0$ and $\ell > 0$, insertion when $\ell' = 0$ and $\ell > 0$, and deletion when $\ell' > 0$ and $\ell = 0$.

This section presents the following result:

Theorem 4.2. *There is an $O(n)$-time and space preprocessing for the ℓ-ELPS problem such that each query can be answered in $O(\ell + \log\log n)$ time, where ℓ denotes the length of the block after an edit.*

Note that the time complexity for our algorithm is independent of the length of the original block to edit. Also, the length ℓ of a new block is arbitrary.

Consider the substitution of a string X of length ℓ for the substring $T[i_b..i_e]$ beginning at position i_b and ending at position i_e, where $i_e - i_b + 1 = \ell'$ and $X \neq T[i_b..i_e]$. Let $T'' = T[1..i_b - 1]XT[i_e + 1..n]$ be the string after the edit. In order to compute (the lengths of) maximal palindromes that are affected by the block-wise edit operation, we need to know the first (leftmost) mismatching position between T and T'', and that between T^R and T''^R. Let h and l be the smallest integers such that $T[h] \neq T''[h]$ and $T^R[l] \neq T''^R[l]$, respectively. If such h does not exist, then let $h = \min\{|T|, |T''|\} + 1$. Similarly, if such l does not exist, then let $l = \min\{|T|, |T''|\} + 1$. Let $j_1 = lcp(T[i_b..n], XT[i_e..n])$, $j_2 = lcp((T[1..i_e])^R, (T[1..i_b]X)^R)$, $p_b = i_b + j_1$, and $p_e = i_e - j_2$. There are two cases: (1) If $j_1 = j_2 = 0$, then the first and last characters of $T[i_b..i_e]$ differ from those of X. In this case, we have $i_b = h$ and $i_e = n - l + 1$. We use these positions i_b and i_e to compute maximal palindromes after the block-wise edit. (2) Otherwise, we have $p_b = i_b + j_1 = h$ and $p_e = i_e - j_2 = n - l + 1$. We use these positions p_b and p_e to compute maximal palindromes after the block-wise edit. See Figure 4.5 for illustration. (2-1) is a sub-case of Case (2) with $p_b(= i_b + j_1) < p_e(= i_e - j_2)$. In the example of this figure, the substring $T[i_b..i_e] =$ `abbccbabcb` is substituted by $X =$ `abbcb`. (2-2) is a sub-case of Case (2) with $p_e(= i_e - j_2) < p_b(= i_b + j_1)$. In the example of this figure, the substring $T[i_b..i_e] =$ `abbcc` is substituted by $X =$ `abbbcc`. (2-3) is the example $T[i_b..i_e] =$ `abbccbabcb` is substituted with $X =$ `abbccb` and this is the sub-case of Case (2) with $j_1 > \ell$. Note that p_b and p_e are only used to compute (the lengths of) maximal palindromes and the fact that $T[i_b..i_e]$ is substituted with X is never changed in any case.

In the following, we describe our algorithm for Case (1). Case (2) can be treated similarly, by replacing i_b and i_e with p_b and p_e, respectively. Our algorithm can handle the case where $p_e < p_b$. Remark that p_b and p_e can be computed in $O(\ell)$ time by naïve character comparisons and a single LCE query each.

4.2.1 Unchanged Maximal Palindromes

We have the following observation for those of maximal palindromes in T whose lengths do not change, which is a generalization of Observation 4.1.

Observation 4.7 (Unchanged maximal palindromes after a block-wise edit)**.** *For any position* $1 \leq j < i_b - 1$, $\mathit{MaxPalEnd}_{T''}(j) = \mathit{MaxPalEnd}_T(j)$. *For any position* $i_e + 1 < j \leq n$, $\mathit{MaxPalBeg}_{T''}(j) = \mathit{MaxPalBeg}_T(j)$.

Hence, we can use the same $O(n)$-time preprocessing and $O(1)$ queries as the 1-ELPS problem: When we consider substitution for an existing block $T[i_b..i_e]$, we take the length of the longest maximal palindrome ending before $i_b - 1$ and that of the longest maximal palindrome beginning after $i_e + 1$ as candidates for a solution to the ℓ-ELPS query. Hence, we obtain the following lemma.

Lemma 4.6. *We can preprocess a string T of length n in $O(n)$ time and space so that later we can compute in $O(1)$ time the length of the LPS of T'' that are unchanged after a block edit.*

4.2.2 Extended Maximal Palindromes

Next, we consider the maximal palindromes of T that get extended after a block edit.

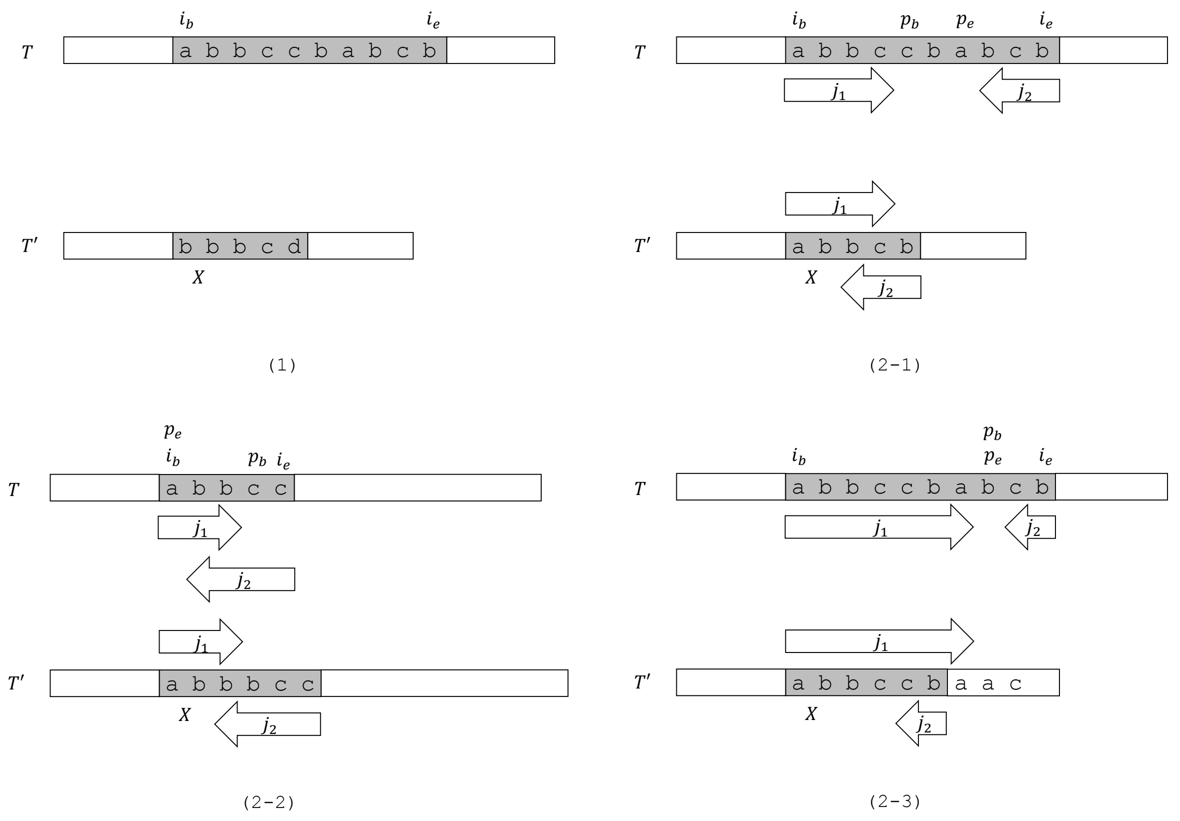


Figure 4.5: Illustration for the mismatching position between T and T'', and that between T^R and T''^R. In particular, (2-1) is the sub-case of Case (2) with $p_b(= i_b + 4) < p_e(= i_e - 3)$, (2-2) is the sub-case of Case (2) with $p_e(= i_e - 4) < p_b(= i_b + 3)$, and (2-3) is the sub-case of Case (2) with $j_1(= 7) > \ell(= 6)$.

Observation 4.8 (Extended maximal palindromes after a block-wise edit)**.** *For any $s \in MaxPalEnd_T(i_b - 1)$, the corresponding maximal palindrome $T[i_b - s..i_b - 1]$ centered at $\frac{2i_b - s - 1}{2}$ gets extended in T'' iff $\mathsf{OutLCE}_{T''}(i_b - s - 1, i_b) \geq 1$. Similarly, for any $p \in MaxPalBeg_T(i_e + 1)$, the corresponding maximal palindrome $T[i_e + 1..i_e + p]$ centered at $\frac{2i_e + p + 1}{2}$ gets extended in T'' iff $\mathsf{OutLCE}_{T''}(i_e, i_e + p + 1) \geq 1$.*

Computation of Extensions

It follows from Observation 4.8 that it suffices to compute outward LCE queries efficiently in the edited string T'' for all maximal palindromes corresponding to $MaxPalEnd_T(i_b - 1)$ or $MaxPalBeg_T(i_e + 1)$. The following lemma shows how to efficiently compute the extensions of any given maximal palindromes that end at position $i_b - 1$. Those that begin at position $i_e + 1$ can be treated similarly.

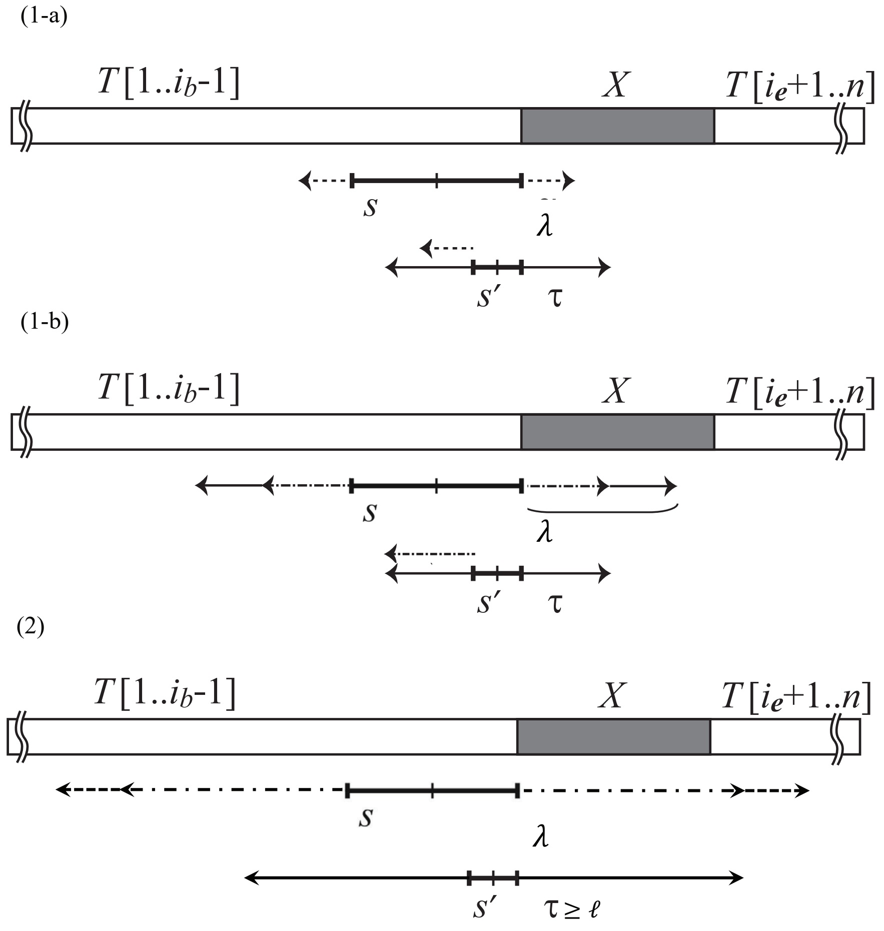


Figure 4.6: Illustration for Lemma 4.7, where solid arrows represent the matches obtained by naïve character comparisons, and broken arrows represent those obtained by LCE queries. This figure only shows the case where $s' < s$, but the other case where $s' > s$ can be treated similarly.

Lemma 4.7. *Let T be a string of length n over an integer alphabet of size polynomially bounded in n. We can preprocess T in $O(n)$ time and space so that later, given a list of any f maximal palindromes from $MaxPalEnd_T(i_b - 1)$, we can compute in $O(\ell + f)$ time the extensions of those f maximal palindromes in the edited string T'', where ℓ is the length of a new block.*

Proof. Let us remark that the maximal palindromes in the list can be given to our algorithm in any order. Firstly, we compute the extensions of given maximal palindromes from the list until finding the first maximal palindrome whose extension τ is at least one, and let s' be the length of this maximal palindrome. Namely, $s' + 2\tau$ is the length of the extended maximal palindrome for s', and the preceding maximal palindromes (if any) were not extended. Let s be the length of the next maximal palindrome from the list after s', and now we are to compute the extension λ for s. See also Figure 4.6. There are two cases: (1) If $0 < \tau < \ell$, then we first compute

$\delta = \mathsf{LeftLCE}_T(i_b - s - 1, i_b - s' - 1)$. We have two sub-cases: (1-a) If $\delta < \tau$, then $\lambda = \delta$. (1-b) Otherwise ($\delta \geq \tau$), then we know that λ is at least as large as τ. We then compute the remainder of λ by naïve character comparisons. If the character comparison reaches the end of X, then the remainder of λ can be computed by $\mathsf{OutLCE}_T(i_b - s - \ell - 1, i_e + 1)$. Then we update τ with λ. (2) If $\tau \geq \ell$, then we can compute λ by $\mathsf{LeftLCE}_T(i_b - s - 1, i_b - s' - 1)$, and if this value is at least ℓ, then by $\mathsf{OutLCE}_T(i_b - s - \ell - 1, i_e + 1)$. The extensions of the following palindromes can also be computed similarly.

The following maximal palindromes from the list after s can be processed similarly. After processing all the f maximal palindromes in the given list, the total number of matching character comparisons is at most ℓ since each position of X is involved in at most one matching character comparison. Also, the total number of mismatching character comparisons is $O(f)$ since for each given maximal palindrome there is at most one mismatching character comparison. The total number of LCE queries on the original text T is $O(f)$, each of which can be answered in $O(1)$ time. Thus, it takes $O(\ell + f)$ time to compute the length of the f maximal palindromes of T'' that are extended after the block edit. □

However, there can be $\Omega(n)$ maximal palindromes beginning or ending at each position of a string of length n. In what follows, we show how to reduce the number of maximal palindromes that need to be considered, by using periodic structure of maximal palindromes.

Longest Extended Palindromes from Each Group

Let $\langle s, d, t \rangle$ be an arithmetic progression representing a group of maximal palindromes ending at position $i_b - 1$. For each $1 \leq j \leq t$, we will use the convention that $s(j) = s + (j-1)d$, namely $s(j)$ denotes the jth shortest element for $\langle s, d, t \rangle$. For simplicity, let $Y = T[1..i_b - 1]$ and $Z = XT[i_e + 1..n]$. Let $Ext(s(j))$ denote the length of the maximal palindrome that is obtained by extending $s(j)$ in YZ.

Lemma 4.8. *For any $\langle s, d, t \rangle \subseteq MaxPalEnd_T(i_b - 1)$, there exist palindromes u, v and a non-negative integer p, such that $(uv)^{t+p-1}u$ (resp. $(uv)^p u$) is the longest (resp. shortest) maximal palindrome represented by $\langle s, d, t \rangle$ with $|uv| = d$. Let $\alpha = lcp((Y[1..|Y| - s(1)])^R, Z)$ and $\beta = lcp((Y[1..|Y| - s(t)])^R, Z)$. Then $Ext(s(j)) = s(j) + 2\min\{\alpha, \beta + (t-j)d\}$. Further, if there exists $s(h) \in \langle s, d, t \rangle$ such that $s(h) + \alpha = s(t) + \beta$, then $Ext(s(h)) = s(h) + 2lcp((Y[1..|Y| - s(h)])^R, Z) \geq Ext(s(j))$ for any $j \neq h$.*

Then let $\gamma = lcp((Y[1..|Y| - s(h)])^R, Z)$. Lemma 4.8 can be proven immediately from

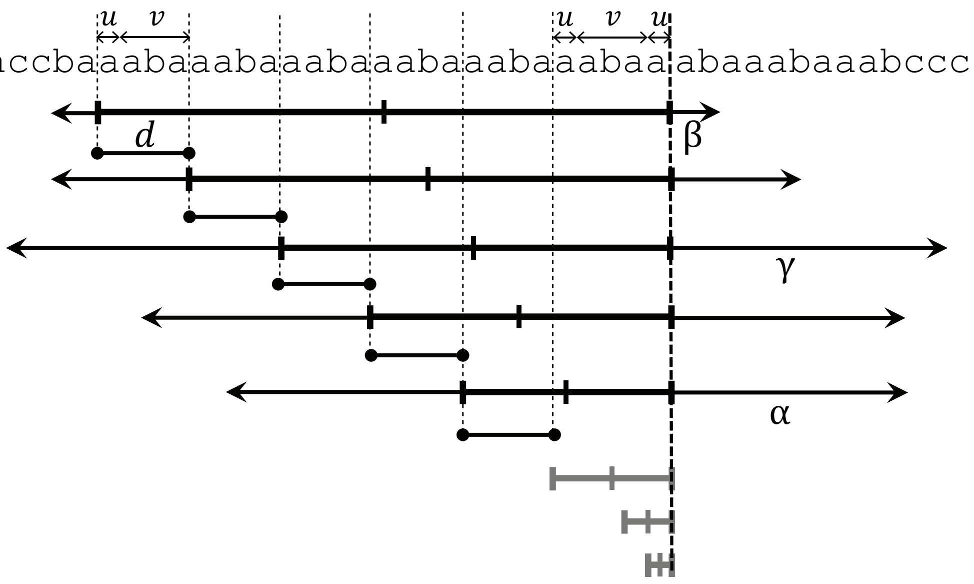


Figure 4.7: Example for Lemma 4.8, where Y = accbaaabaaabaaabaaabaaabaaabaa and Z = abaaabaaabccc. Here u = a and v = aba. The first five maximal palindromes $(uv)^p u$ = (aaba)^{p}a with $2 \le p \le 5$ belong to the same arithmetic progression (i.e. the same group) with common difference $|uv| = d = 4$. For this group of maximal palindromes, $\alpha = 10$, $\beta = 2$, and $\gamma = 12$. Notice that the sixth maximal palindrome uvu = aabaa belongs to another group since the length difference between it and the seventh one aa is 3.

Lemma 12 of [90]. However, for the sake of completeness we here provide a proof. We use the following known result:

Lemma 4.9 ([90]). *For any string Y and $\{s(j) \mid s(j) \in \langle s, d, t\rangle\} \subseteq SufPals(Y)$, there exist palindromes u, v and a non-negative integer p, such that $(uv)^{t+p-1}u$ is a suffix of Y, $|uv| = d$ and $|(uv)^p u| = s$.*

Now we are ready to prove Lemma 4.8 (see also Figure 4.7).

Proof. Let us consider $Ext(s(j))$, such that $s(j) \in \langle s, d, t\rangle$. By Lemma 4.9, $Y[|Y| - s(1) - (t-1)d + 1..|Y|] = (uv)^{t+p-1}u$, where $|uv| = d$ and $|(uv)^p u| = s$.

Let x be the largest integer such that $(Y[|Y| - x + 1..|Y|])^R$ has a period $|uv|$. Namely, $(Y[|Y| - x + 1..|Y|])^R$ is the longest prefix of Y^R that has a period $|uv|$. Then x is given as $x = lcp(Y^R, (Y[1..|Y| - d])^R) + d$. Let y be largest integer such that $(uv)^{y/d}$ is a prefix of Z. Then y is given as $y = \min\{lcp(Y^R, Z), x\}$.

Let $e_l = |Y| - x + 1$ and $e_r = |Y| + y$. Then, clearly string $T''[e_l..e_r]$ has a period d. We divide $\langle s, d, t\rangle$ into three disjoint subsets as

$$\langle s, d, t\rangle = \langle s, d, t_1\rangle \cup \langle s + t_1 d, d, t_2\rangle \cup \langle s + (t_1 + t_2)d, d, t_3\rangle,$$

such that
$|Y| - e_l - s(j) + 1 > e_r - |Y|$ for any $s(j) \in \langle s, d, t_1\rangle$,
$|Y| - e_l - s(j) + 1 = e_r - |Y|$ for any $s(j) \in \langle s + t_1 d, d, t_2\rangle$,
$|Y| - e_l - s(j) + 1 < e_r - |Y|$ for any $s(j) \in \langle s + (t_1 + t_2)d, d, t_3\rangle$,
$t_1 + t_2 + t_3 = t$, and $t_2 \in \{0, 1\}$.

Then, for any $s(j)$ in the first sub-group $\langle s, d, t_1\rangle$, $\mathit{Ext}(s(j)) = s(j) + 2(e_r - |Y|) = s(j) + 2y$. Also, for any $s(j)$ in the third sub-group $\langle s+(t_1+t_2)d, d, t_3\rangle$, $\mathit{Ext}(s(j)) = s(j)+2(|Y|-e_l-s(j)+1) = s(j)+2(x-s(j))$. Now let us consider $s(j) \in \langle a_2, d, t_2\rangle$, in which case $s(j) = s(h)$ (see the statement of Lemma 4.8). Note that $0 \le t_2 \le 1$, and here we consider the interesting case where $t_2 = 1$. Since the palindrome $s(h)$ can be extended beyond the periodicity with respect to uv, we have $\mathit{Ext}(s(h)) = s(h) + 2\gamma$, where $\gamma = \mathit{lcp}((Y[1..|Y| - s(h)])^R, Z)$.

Additionally, we have that $y = \mathit{lcp}(Y^R, Z) = \mathit{lcp}((Y[1..|Y| - s(1)])^R, Z) = \alpha$ where the second equality comes from the periodicity with respect to uv, and that $x - s(j) = \mathit{lcp}((Y[1..|Y| - s(t)])^R, Z) + (t - j)d = \beta + (t - j)d$. Therefore, for any $s(j) \in \langle s, d, t\rangle$, $\mathit{Ext}(s(j))$ can be represented as follows:

$$\mathit{Ext}(s(j)) = \begin{cases} s(j) + 2\alpha & (\alpha < \beta + (t - j)d) \\ s(j) + 2(\beta + (t - j)d) & (\alpha > \beta + (t - j)d) \\ s(j) + 2\gamma & (\alpha = \beta + (t - j)d) \end{cases}$$

This completes the proof. □

It follows from Lemma 4.8 that it suffices to consider only three maximal palindromes from each group (i.e. each arithmetic progression). Then using Lemma 4.7, one can compute the longest maximal palindrome that gets extended in $O(\ell + \log n)$ time.

Relationship of Groups

To further speed up computation, we take deeper insights into combinatorial properties of maximal palindromes in $\mathit{MaxPalEnd}_T(i_b - 1)$. Let $G_0, \ldots, G_m$ be the list of all groups for the maximal palindromes from $\mathit{MaxPalEnd}_T(i_b - 1)$, which are *sorted in increasing order of their common difference*. Namely, the jth shortest member of $\mathit{MaxPalEnd}_T(i_b - 1)$ belongs to $G_r = \langle s_r, d_r, t_r\rangle$ with $1 \le r \le m$, iff the difference between the jth shortest maximal palindrome and the $(j-1)$th one is equal to d_r. Then d_r with $1 \leq r \leq m$ is corresponding to the period of any maximal palindrome in G_r. Regardless of whether ε belongs to $\mathit{MaxPalEnd}_T(i_b - 1)$ or not, we define that G_0 is a singleton, $d_0 = 0$, and ε is the element of G_0. When $m = O(\log\log n)$, $O(\ell + \log\log n)$-time queries immediately follow from Lemmas 4.7 and 4.8. In what follows we consider the more difficult case where $m = \omega(\log\log n)$. Recall also that $m = O(\log n)$ always holds.

For each $G_r = \langle s_r, d_r, t_r \rangle$ with $1 \leq r \leq m$, let α_r, β_r, γ_r, u_r, and v_r be the corresponding variables used in Lemma 4.8. If there is only a single element in G_r, let β_r be the length of extension of the palindrome and $\alpha_r = \beta_{r-1}$. For convenience, let $\alpha_0 = -1$. For each G_r, let S_r (resp. L_r) denote the shortest (resp. longest) maximal palindrome in G_r, namely, $|S_r| = s_r(1)$ and $|L_r| = s_r(t_r)$. Each group G_r is said to be of *type-1* (resp. *type-2*) if $\alpha_r < d_r$ (resp. $\alpha_r \geq d_r$).

Let k $(1 \leq k \leq m)$ be the unique integer such that G_k is the type-2 group where d_k is the largest common difference among all the type-2 groups. Additionally, let $G'_k = G_k \cup \{u_k v_k u_k, u_k\}$. Note that u_k belongs to one of $G_1, \ldots, G_{k-1}$, and $u_k v_k u_k$ belongs to either G_k or one of $G_1, \ldots, G_{k-1}$, if $u_k v_k u_k$ and u_k exist. In the special case where $\alpha_k = \beta_k + t d_k$, the extensions of u_k and $u_k v_k u_k$ can be longer than the extension of the shortest maximal palindrome in G_k (see Figure 4.8 for a concrete example). Thus, it is convenient for us to treat $G'_k = G_k \cup \{u_k v_k u_k, u_k\}$ as if it is a single group. We also remark that this set G'_k is defined only for this specific type-2 group G_k.

Lemma 4.10. *There is a longest palindromic substring in the edited string T'' that is obtained by extending the maximal palindromes in G_m, G_{m-1}, or G'_k.*

Proof. The lemma holds if the two following claims are true:

Claim (1): The extensions of the maximal palindromes in $G_1, \ldots, G_{k-1}$, except for $u_k v_k u_k$ and u_k, cannot be longer than the extension of the shortest maximal palindrome in G_k.

Claim (2) Suppose both G_m and G_{m-1} are of type-1. Then, the extensions of the maximal palindromes from $G_{k+1}, \ldots, G_{m-2}$, which are also of type-1, cannot be longer than the extensions of the maximal palindromes from G_m or G_{m-1}.

Proof for Claim (1). Here we consider the case where the maximal palindrome $u_k v_k u_k$ does not belong to G_k, which implies that the shortest maximal palindrome S_k in G_k is $(u_k v_k)^2 u_k$ (The other case where $u_k v_k u_k$ belongs to G_k can be treated similarly). Now, $u_k v_k u_k$ belongs to one of $G_1, \ldots, G_{k-1}$. Consider the prefix $P = T[1..i_b - |u_k v_k u_k| - 1]$ of T that immediately

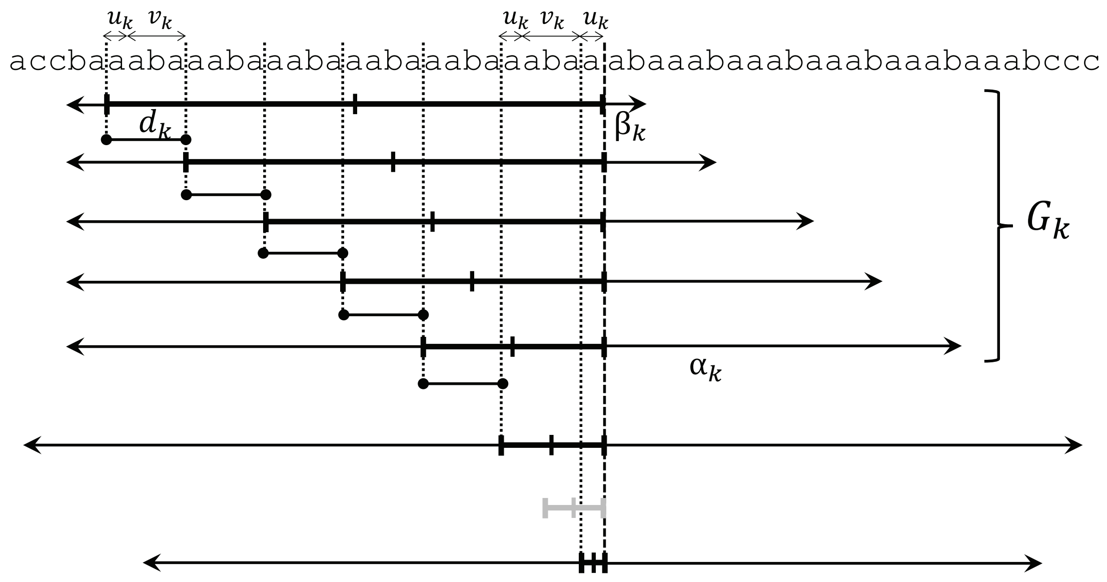


Figure 4.8: Example for $G'_k = G_k \cup \{u_k v_k u_k, u_k\}$, where the extensions of $u_k v_k u_k$ and u_k are longer than the extensions of any maximal palindromes in G_k.

precedes $u_k v_k u_k$. The extension of $u_k v_k u_k$ is obtained by $lcp(P^R, Z)$. Consider the prefix $P' = T[1..i_b - |(u_k v_k)^2 u_k| - 1]$ of T that immediately precedes $(u_k v_k)^2 u_k$. It is clear that P is a concatenation of P' and $u_k v_k$. Similarly, the prefix $T[1..i_b - |u_k| - 1]$ of T that immediately precedes u_k is a concatenation of P and $u_k v_k$. It suffices for us to consider only the three maximal palindromes from G'_k. For any other maximal palindrome Q from $G_0, \dots, G_{k-1}$, assume on the contrary that Q gets extended by at least d_k to the left and to the right. If $|u_k v_k| = d_k < |Q| < |u_k v_k u_k|$, then there is an internal occurrence of $u_k v_k$ inside the prefix $(u_k v_k)^2$ of $(u_k v_k)^2 u_k$. Otherwise ($|u_k| < |Q| < |u_k v_k| = d_k$ or $|Q| < |u_k|$), there is an internal occurrence of $u_k v_k$ inside $u_k v_k u_k$. Here we only consider the first case but other cases can be treated similarly. See also Figure 4.9. This internal occurrence of $u_k v_k$ is immediately followed by $u_k v_k w$, where w is a proper prefix of u_k with $1 \leq |w| < |u_k|$. Namely, $(u_k v_k)^2 w$ is a proper suffix of $(u_k v_k)^2 u_k$. On the other hand, $(u_k v_k)^2 w$ is also a proper prefix of $(u_k v_k)^2 u_k$. Since $(u_k v_k)^2 u_k$ is a palindrome, it now follows from Lemma 2.1 that $(u_k v_k)^2 w$ is also a palindrome. Since $1 \leq |w| < |u_k|$, we have $|(u_k v_k)^2 w| > |u_k v_k u_k|$ (note that this inequality holds also when v_k is the empty string). Then, $(u_k v_k)^2 w$ is also immediately preceded by $u_k v_k$ because of periodicity and is extended by at least d_k to the left and to the right because G_k is of type-2. Since $T''[i_b] = T[i_b - |(u_k v_k)^2 w| - 1]$ and $T''[i_b] \neq T[i_b]$, $(u_k v_k)^2 w$ is a maximal palindrome. However this contradicts that $(u_k v_k)^2 u_k$ belongs to G_k with common difference $d_k = |u_k v_k|$. Thus Q cannot be extended by d_k nor more to the left and to the right. Since G_k is of type-2,

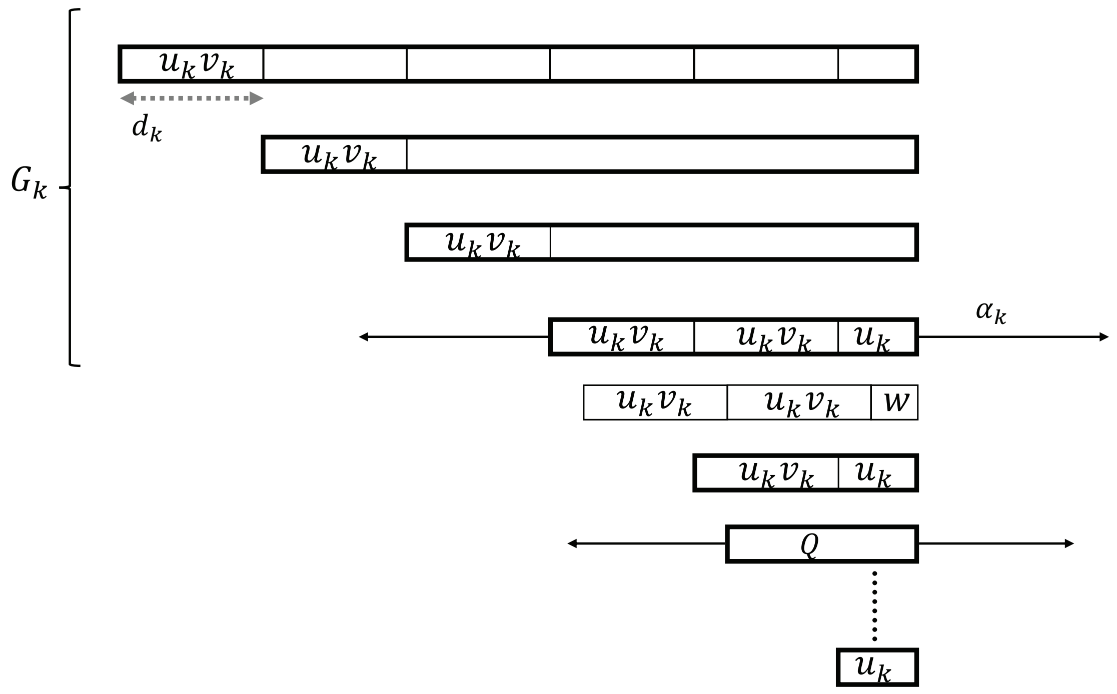


Figure 4.9: Illustration for the proof for Claim (1) of Lemma 4.10.

$\alpha_k \geq d_k$. Since $|Q| < |(u_k v_k)^2 u_k|$, the extension of Q cannot be longer than the extension for $(u_k v_k)^2 u_k$. This completes the proof for Claim (1).

Proof for Claim (2). Consider each group $G_r = \langle s_r, d_r, t_r \rangle$ with $k+1 \leq r \leq m-2$. By Lemma 4.8, $s_r(t_r) + 2\beta_r$ and $s_r(t_r - 1) + 2\alpha_r$ are the candidates for the longest extensions of the maximal palindromes from G_r. Recall that both G_{m-1} and G_m are of type-1, and that if G_r is of type-1 then G_{r+1} is also of type-1. Now the following sub-claim holds:

Lemma 4.11. *$\beta_r = \alpha_{r+1}$ for any $k+1 \leq r \leq m-2$.*

Proof. If G_{r+1} is a singleton, then by definition $\beta_r = \alpha_{r+1}$ holds. Now suppose $|G_{r+1}| \geq 2$. Since the shortest maximal palindrome S_{r+1} from G_{r+1} is either $(u_{r+1}v_{r+1})^2 u_{r+1}$ or $u_{r+1}v_{r+1}u_{r+1}$, the longest maximal palindrome L_r from G_r is either $u_{r+1}v_{r+1}u_{r+1}$ or u_{r+1}. The prefix $T[1..i_b - |L_r| - 1]$ of T that immediately precedes L_r contains $u_{r+1}v_{r+1}$ as a suffix, which alternatively means $(u_{r+1}v_{r+1})^R$ is a prefix of $(T[1..i_b - |L_r| - 1])^R$. Moreover, it is clear that the prefix $T[1..i_b - |S_{r+1}| - 1]$ of T that immediately precedes S_{r+1} contains $u_{r+1}v_{r+1}$ as a suffix since $|G_{r+1}| \geq 2$. In addition, $\alpha_{r+1} < d_{r+1} = |u_{r+1}v_{r+1}|$ since G_{r+1} is of type-1. From the above arguments, we get $\beta_r = \alpha_{r+1}$. $\square$

Since $\beta_r = \alpha_{r+1}$ and $\alpha_{r+1} < d_{r+1}$, we have $s_r(t_r) + 2\beta_r < s_r(t_r) + 2d_{r+1}$. In addition, $s_r(t_r - 1) + 2\alpha_r < s_r(t_r - 1) + 2d_r = s_r(t_r) + d_r$. It now follows from $d_r < d_{r+1}$ that

$s_r(t_r) + d_r < s_r(t_r) + 2d_{r+1}$. Since the lengths of the maximal palindromes and their common differences are arranged in increasing order in the groups $G_{k+1}, \ldots .G_{m-2}$, we have that the longest extension from $G_{k+1}, \ldots .G_{m-2}$ is shorter than $s_{m-2}(t_{m-2}) + 2d_{m-1}$. Since $d_{m-1} < d_m$, we have

$$s_{m-2}(t_{m-2}) + 2d_{m-1} < s_{m-2}(t_{m-2}) + d_{m-1} + d_m \leq s_{m-1}(t_{m-1}) + d_m \leq s_m = s_m(1).$$

This means that the longest extended maximal palindrome from the type-1 groups G_{k+1}, ..., G_{m-2} cannot be longer than the original length of the maximal palindrome from G_m before the extension. This completes the proof for Claim (2). □

It follows from Lemmas 4.7, 4.8 and 4.10 that given G_k, we can compute in $O(\ell)$ time the length of the LPS of T'' after the block edit. What remains is how to quickly find G_k, that has the largest common difference among all the type-2 groups. Note that a simple linear search from G_m or G_1 takes $O(\log n)$ time, which is prohibited when $\ell = o(\log n)$. In what follows, we show how to find G_k in $O(\ell + \log\log n)$ time.

How to Find G_k

Recall that $T[1..i_b - |L_{r-1}| - 1]$ which immediately precedes S_r contains $u_r v_r$ as a suffix. Thus, $(u_r v_r)^R$ is a prefix of $(T[1..i_b - |L_{r-1}| - 1])^R$. We have the following observation.

Observation 4.9. *Let $W_r = (T[1..i_b - |L_{r-1}| - 1])^R$ for $1 \leq r \leq m$. Let W be the string such that $lcp(W_r, Z)$ is the largest for all $1 \leq r \leq m$ (i.e. for groups $G_1, \ldots, G_m$), namely, $W = \arg\max_{1 \leq r \leq m} lcp(W_r, Z)$. Then $G_k = G_x$ such that*

(a) $(u_x v_x)^R$ is a prefix of W,

(b) $d_x \leq lcp(W, Z)$, and

(c) d_x is the largest among all groups that satisfy Conditions (a) and (b).

Due to Observation 4.9, the first task is to find W.

Lemma 4.12. *W can be found in $O(\ell + \log \min\{\sigma, \log n\})$ time after $O(n)$-time and space preprocessing.*

Proof. We preprocess T as follows. Let $\mathcal{A}_i$ be the *sparse suffix array* of size $m = O(\log i)$ such that $\mathcal{A}_i[j]$ stores the jth lexicographically smallest string in $\{W_1, \ldots, W_m\}$. We build $\mathcal{A}_i$ with the LCP array $\mathcal{L}_i$. Since there are only $2n-1$ maximal palindromes in T, $\mathcal{A}_i$ for all positions $1 \leq i \leq n$ can easily be constructed in a total of $O(n)$ time from the full suffix array of T. The LCP array $\mathcal{L}_i$ for all $1 \leq i \leq n$ can also be computed in $O(n)$ total time from the LCP array of T enhanced with a range minimum query (RMQ) data structure [15].

To find W, we binary search $\mathcal{A}_{i_b-1}$ for $Z[1..\ell] = X$ in a similar way to pattern matching on the suffix array with the LCP array [87]. This gives us the range of $\mathcal{A}_{i_b-1}$ such that the corresponding strings have the longest common prefix with X. Since $|\mathcal{A}_{i_b-1}| = O(\log n)$, this range can be found in $O(\ell + \log\log n)$ time. If the longest prefix found above is shorter than ℓ, then this prefix is W. Otherwise, we perform another binary search on this range for $Z[\ell+1..|Z|] = T[i_e+1..n]$, and this gives us W. Here each comparison can be done in $O(1)$ time by an outward LCE query on T. Hence, the longest match for $Z[\ell+1..|Z|]$ in this range can also be found in $O(\log\log n)$ time. Overall, W can be found in $O(\ell + \log\log n)$ time.

Also, W can be found similarly in $O(\ell + \log\sigma)$ time after $O(n)$-time and space preprocessing by using the sparse suffix tray [30] instead of the sparse suffix array. Then we can obtain Lemma 4.12. □

Lemma 4.13. *We can preprocess T in $O(n)$ time and space so that later, given W for a position in T, we can find G_k for that position in $O(1)$ time.*

Proof. Let $\mathcal{D}_i$ be an array of size $|\mathcal{A}_i|$ such that $\mathcal{D}_i[j]$ stores the value of $d_r = |u_r v_r|$, where W_r is the lexicographically jth smallest string in $\{W_1, \dots, W_m\}$. Let $\mathcal{R}_i$ be an array of size $|\mathcal{A}_i|$ where $\mathcal{R}_i[j]$ stores a sorted list of common differences $d_r = |u_r v_r|$ of groups G_r, such that G_r stores maximal palindromes ending at position i and $(u_r v_r)^R$ is a prefix of the string $\mathcal{A}_i[j]$. Clearly, for any j, $\mathcal{D}_i[j] \subseteq \mathcal{R}_i$.

Suppose that we have found W by Lemma 4.12, and let j' be the entry of $\mathcal{A}_{i_b-1}$ where the binary search for W terminated. We then find the largest d_x that satisfies Condition (b) of Observation 4.9, by binary search on the sorted list of common differences stored at $\mathcal{R}_{i_b-1}[j']$.

We remark however that the total number of elements in $\mathcal{R}_i$ is $O(\log^2 i)$ since each entry $\mathcal{R}_i[j]$ can contain $O(\log i)$ elements. Thus, computing and storing $\mathcal{R}_i$ explicitly for all text positions $1 \leq i \leq n$ can take superlinear time and space.

Instead of explicitly storing $\mathcal{R}_i$, we use a bit-vector representation of $\mathcal{R}_i$, defined as follows: Let $Bit\mathcal{R}_i[j]$ be a bit sequence of length m such that the r-th bit from the rightmost position is 1 if and only if $d_r \in \mathcal{R}_i[j]$. Also, let $MSB_d[1..m]$ be the array such that $MSB_d[r]$ stores the position of the leftmost 1 (namely, the most significant bit) in a bit-vector representation of d_r. Let $\mathcal{F}[1..\lfloor\log n\rfloor + 1]$ be a sequence where $\mathcal{F}[j]$ stores the number of d_r for all $1 \leq r \leq m$ such that $MSB_d[r] = j$. Since $d_0 = 0$, $d_1 = 1$, and $d_r \geq d_{r-1} + d_{r-2} \geq 2d_{r-2}$ for $r \geq 2$ holds [90], $\mathcal{F}[1..\lfloor \log n \rfloor + 1]$ is a ternary sequence. See also Figure 4.2.2 that illustrates a concrete example.

In the word RAM model with word size $\Theta(\log n)$, the size of $Bit\mathcal{R}_i[j]$ and $MSB_d[1..m]$ is $O(m)$, and these sequences can be constructed in $O(m)$ time by using the LCP array $\mathcal{L}_i$, bit-wise operations, and constructing the answer table for the most significant bit of every value from 1 to n. The ternary sequence $\mathcal{F}$ of length $O(\log n)$ can also be computed in $O(m)$ time. Therefore, the total size and construction time for all positions in T is $O(n)$.

We can find the largest d_x that satisfies Condition (b) of Observation 4.9 as follows: First, we compute the most significant bit b'' of the binary representation of $lcp(W^R, Z)$. Next, we compute $d_{r'}$ that is the predecessor of $lcp(W^R, Z)$ in the sorted list of common differences by using the following properties. The most significant bit b' of bit-vector representation of $d_{r'}$ is b'' or the largest value such that $b' < b''$ and $\mathcal{F}[b'] \geq 1$. Since the number of $d_{r''}$, which the most significant bit of $d_{r''}$ is b'', is at most two, the number of candidates of $d_{r'}$ is at most three. The indexes r_1, r_2, r_3 of these candidates can be found by using rank/select operations on $\mathcal{F}$. $d_{r'}$ can be obtained by comparing $d_{r_1}, d_{r_2}, d_{r_3}$ to $lcp(W^R, Z)$. Then, we construct a bit-vector $a[1..m]$ such that $a[m - d_{r'} + 1..m] = 1^{d_{r'}}$ and other positions store 0. By using the "AND" operation between $Bit\mathcal{R}_i[j]$ and $a[1..m]$ and computing the most significant bit of the answer sequence, we can obtain d_x. The above operations can be done in constant time in the word RAM model; we can obtain Lemma 4.13.

$\square$

By Lemmas 4.7, 4.8, 4.10, 4.12, and 4.13, we can compute in $O(\ell + \log \min\{\sigma, \log n\})$ time the length of the LPS of T'' that are extended after the block edit.

4.2.3 Shortened Maximal Palindromes

Next, we consider the maximal palindromes that get shortened after a block edit.

Observation 4.10 (Shortened maximal palindromes after a block-wise edit)**.** *A maximal palindrome $T[b..e]$ of T gets shortened in T'' iff $b \leq i_b \leq e$ or $b \leq i_e \leq e$.*

The difference between Observation 4.3 and this one is only in that here we need to consider two positions i_b and i_e. Hence, we obtain the next lemma using a similar method to Lemma 4.2:

Lemma 4.14. *We can preprocess a string T of length n in $O(n)$ time and space so that later we can compute in $O(1)$ time the length of the longest maximal palindromes of T'' that are shortened after a block edit.*

4.2.4 Maximal Palindromes whose Centers Exist in the New Block

Finally, we consider those maximal palindromes whose centers exist in the new block X of length ℓ. By symmetric arguments to Observation 4.8, we only need to consider the prefix palindromes and suffix palindromes of X. Using a similar technique to Lemma 4.7, we obtain:

Lemma 4.15. *We can compute the length of the longest maximal palindromes whose centers are inside X in $O(\ell)$ time and space.*

Proof. First, we compute all maximal palindromes in X in $O(\ell)$ time. Let $p_1, \ldots, p_u$ be a sequence of the lengths of the prefix palindromes of X sorted in increasing order. For each $1 \leq j \leq u$, let $\alpha_j = lcp(X[p_j + 1..\ell], (T[1..i_b - 1])^R)$, namely, $p_j + 2\alpha_j$ is the length of the extended maximal palindrome for each p_j. Suppose we have computed α_{j-1}, and we are to compute α_j. See also Figure 4.11. If $p_{j-1} + \alpha_{j-1} \leq p_j$, then we compute p_j by naïve character comparisons. Otherwise, then let $\alpha'_j = p_{j-1} + \alpha_{j-1} - p_j$. Then, we can compute $lcp(X[p_j + 1..p_j + \alpha'_j], (T[1..i_b - 1])^R)$ by a leftward LCE query in the original string T. If this value is less than α'_j, then it equals to α_j. Otherwise, then we compute $lcp(X[p_j + \alpha'_j + 1..\ell], (T[1..i_b - 1])^R)$ by naïve character comparisons. The total number of matching character comparisons is at most ℓ since each position in X can be involved in at most one matching character comparison. The total number of mismatching character comparisons is also ℓ, since there are at most ℓ prefix palindromes of X and for each of them there is at most one mismatching character comparison. Hence, it takes $O(\ell)$ time to compute the length of the longest maximal palindromes whose centers are inside X. □

j	$W_{\mathcal{A}_i[1]}, \ldots, W_{\mathcal{A}_i[m]}$	$\mathcal{D}_i$	$\mathcal{L}_i$	$\mathcal{R}_i$	$Bit\mathcal{R}_i$
1	`aaabaaabaaaaaabaaaba`$\cdots$	1	-	1,11,533	1000101
2	`aaabaaabaaaaaabaaaba`$\cdots$	533	1599	1,11,533	1000101
3	`aaabaaabaaaaaabaaaba`$\cdots$	11	33	1,11	0000101
4	`aaabaaabaaabaaaaaaba`$\cdots$	48	11	1,11,48	0010101
5	`baaabaaaaaabaaabaaaa`$\cdots$	4	0	4	0000010
6	`baaaaaabaaabaaaaaaba`$\cdots$	37	4	4,37	0001010
7	`caaabaaabaaaaaabaaab`$\cdots$	178	0	178	0100000

r	bit-vector representation of d_r	MSB_{d_r}
1	1	1
2	100	3
3	1011	4
4	100101	6
5	110000	6
6	10110010	8
7	1000010101	10

b	$\mathcal{F}$
1	1
2	0
3	1
4	1
5	0
6	2
$\vdots$	$\vdots$
$\lfloor \log n \rfloor + 1$	0

Figure 4.10: A concrete example for $T[1..i_b] = \mathtt{ddd} w_7^3 w_6^2 w_5^2 w_4 w_3^3 w_2^2 w_1^3 \mathtt{e}$ with $i_b = 2136$, where $w_1 = \mathtt{a}, w_2 = {w_1^3}^R \mathtt{b}, w_3 = {w_1^3}^R {w_2^2}^R, w_4 = {w_1^3}^R {w_2^2}^R {w_3^2}^R w_2, w_5 = {w_1^3}^R {w_2^2}^R {w_3^3}^R {w_2}^R, w_6 = {w_1^3}^R {w_2^2}^R {w_3^3}^R {w_4}^R {w_5^2}^R \mathtt{c}, w_7 = {w_1^3}^R {w_2^2}^R {w_3^3}^R {w_4}^R {w_5^2}^R {w_6^2}^R$. We remark that $w_7^3 w_6^2 w_5^2 w_4 w_3^3 w_2^2 w_1^3$, $w_7^2 w_6^2 w_5^2 w_4 w_3^3 w_2^2 w_1^3$, ..., w_1 are maximal palindromes of $T[1..i_b - 1]$. The remaining parts of the strings $W_{\mathcal{A}_i[1]}, \ldots, W_{\mathcal{A}_i[m]}$ are omitted due to lack of space. Now we describe the case that $j' = 4$ and $lcp(W^R, Z) = 40$. The most significant bit of $40_{(10)} = 101000_{(2)}$ is 6. Then by using rank/select operation on $\mathcal{F}$, we obtain three candidates d_3, d_4, and d_5 of the predecessor of 40 in the sorted list of common differences. Since $d_4 = 37 < 40 < d_5 = 48$, d_4 is the predecessor of 40 in the sorted list of common differences. By using the "AND" operation between $Bit\mathcal{R}_i[j'] = 0010101$ and 0001111, we obtain a bit sequence 0000101. Since the most significant bit of 0000101 is the third bit, $G_k = G_3$ holds.

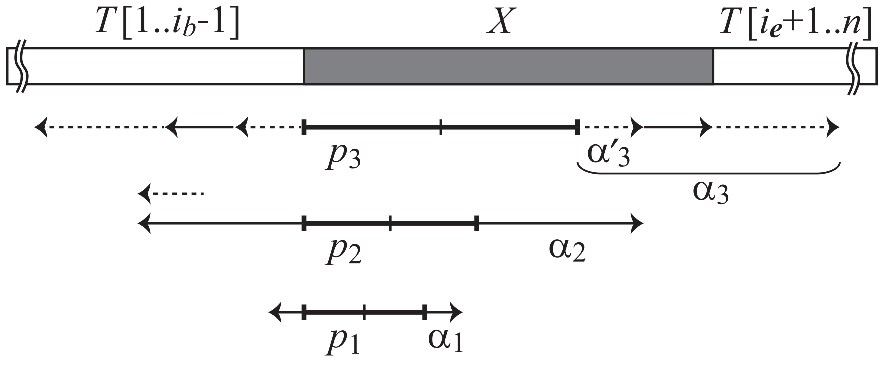


Figure 4.11: Illustration for Lemma 4.15, where solid arrows represent the matches obtained by naïve character comparisons, and broken arrows represent those obtained by LCE queries. Here are three prefix palindromes of X of length p_1, p_2, and p_3. We compute α_1 naïvely. Here, since $p_1 + \alpha_1 < p_2$, we compute p_2 naïvely. Since $p_2 + \alpha_2 > p_3$, we compute $\mathsf{LeftLCE}_T(i_b - 1, i_b - \alpha_2 + \alpha'_3 - 1)$. Here, since its value reached α'_3, we perform naïve character comparison for $X[p_3 + \alpha'_3 + 1..\ell]$ and $(T[1..i_b - \alpha'_3 - 1])^R$. Here, since there was no mismatch, we perform $\mathsf{OutLCE}_T(i_b - \ell + p_3 - 1, i_e + 1)$ and finally obtain α_3. Other cases can be treated similarly.

Chapter 5

Minimal Unique Palindromic Substring After Edit

In this chapter, as analogous to the previous chapter, we tackle the problems of computing the minimal unique palindromic substring (MUPS) of a string after a single-character substitution. In molecular biology, it is known that unique palindromic structures affect the immunostimulatory activities of oligonucleotides [80, 121]. In Section 5.1, we analyze the changes of the set of MUPSs after a single-character substitution. We then present an algorithm for updating the set of MUPSs after a single-character substitution in Section 5.2.

The results in this chapter primarily appeared in [40].

5.1 Changes of MUPSs after Single Character Substitution

In the following, we fix the original string T of length n, the text position i in T to be substituted, and the string T' after the substitution. Namely, $T[i] \neq T'[i]$ and $T[j] = T'[j]$ for each j with $1 \leq j \leq n$ and $j \neq i$. This section analyzes the changes of the set of MUPSs when $T[i]$ is substituted by $T'[i]$. For palindromes covering editing position i, Lemma 5.1 holds.

Lemma 5.1. *For a palindrome w, if $\mathit{inbeg}_{T,i}(w) \neq \emptyset$, then $\mathit{inbeg}_{T',i}(w) = \emptyset$.*

Proof. For the sake of contradiction, we assume that there is a palindrome w with $\mathit{inbeg}_{T,i}(w) \neq \emptyset$ and $\mathit{inbeg}_{T',i}(w) \neq \emptyset$. Let c (resp. c') be the center of an occurrence of w in T (resp. in T') covering position i. It is clear that $c \neq c'$ since $T[i]$ is substituted by another character $T'[i]$. Also, it suffices to consider when $c < c'$ from the symmetry of T and T'. Let d (resp. d') be the distance between c and i (resp. c' and i), i.e., $d = |i - c|$ and $d' = |i - c'|$.

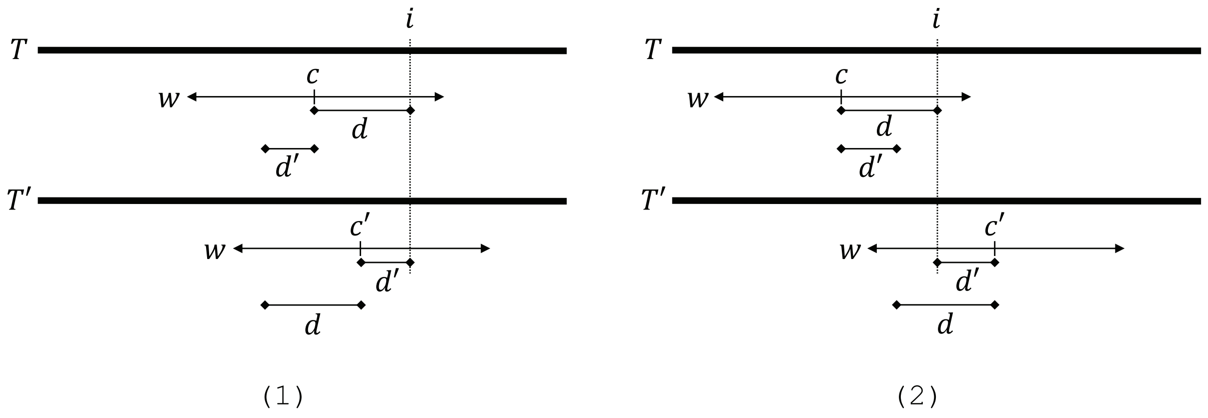


Figure 5.1: Illustration for the two cases of Lemma 5.1. Note that this illustration is for the sake of contradiction.

There are the following two cases: either (1) $i \notin [c, c']$ or (2) $i \in [c, c']$. See also Fig. 5.1 for illustration. (1) Now we consider the case when $c' < i$. Another case ($i < c$) can be treated similarly. On the one hand, since c and c' are the centers of w, $T'[c' + d] = T[c + d] = T[i]$. Further, $T'[c' - d] = T[c' + d]$ by palindromic symmetry, and hence, $T'[c' - d] = T[i]$. On the other hand, again, since c and c' are the centers of w, $T[c - d'] = T'[c' - d']$. Further, $T'[c' - d'] = T'[c' + d'] = T'[i]$ by palindromic symmetry, and hence, $T[c - d'] = T'[i]$. Also, $T'[c - d'] = T[c - d'] = T'[i]$ since $c - d' \neq i$. Since $c' - d = c - d'$ holds in this case, $T[i] = T'[c' - d] = T'[c - d'] = T'[i]$, a contradiction.

(2) Similar to the first case, it can be seen that $T'[c' - d] = T[c - d] = T[i]$. If $d = d'$, then $T[i] = T'[c' - d] = T'[c' - d'] = T'[i]$, a contradiction. Hence $d \neq d'$ holds, and thus, $T'[c + d'] = T[c + d']$. Also, $T[c + d'] = T'[c' + d'] = T'[c' - d'] = T'[i]$ holds. Finally, since $c' - d = c + d'$ holds in this case, $T[i] = T'[c' - d] = T'[c + d'] = T'[i]$, a contradiction. $\square$

For a position i, let $\mathcal{W}_i$ be the set of palindromes w such that $|inbeg_{T,i}(w)| \geq 1$, $|xbeg_{T,i}(w)| = 1$, and w is minimal, i.e., $|inbeg_{T,i}(v)| = 0$ or $|xbeg_{T,i}(v)| \geq 2$ where $v = w[2..|w| - 1]$. This set $\mathcal{W}_i$ is useful for analyzing the number of changes of MUPSs in the proof of Theorem 5.1.

Lemma 5.2. *For any position i in T, $|\mathcal{W}_i| \in O(\log n)$.*

Proof. First, by minimality of palindromes in $\mathcal{W}_i$, centers of palindromes in $\mathcal{W}_i$ are different from each other. Let $\mathcal{W}_i^L \subset \mathcal{W}_i$ (resp. $\mathcal{W}_i^R \subset \mathcal{W}_i$) be the set of palindromes in $\mathcal{W}_i$ whose center is at most i (resp. at least i).

Let us consider the size of $\mathcal{W}_i^L$. Every palindrome in $\mathcal{W}_i^L$ is an expansion of some palindromic suffix of $T[1..i]$. From Corollary 2.1, the set of palindromic suffixes of $T[1..i]$ can be

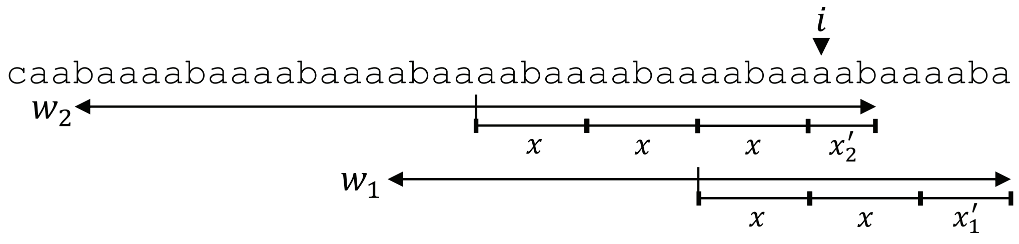


Figure 5.2: Example for Lemma 5.2, where w_1 and w_2 have the same smallest period 5 and the difference between centers of them is a power of 5. Here $x = \texttt{aabaa}$, $x_1' = \texttt{aaba}$, and $x_2' = \texttt{aab}$.

divided into $m_i \in O(\log i)$ groups w.r.t. their smallest period. Let $G_1, G_2, \ldots, G_{m_i}$ be such groups for palindromic suffixes of $T[1..i]$, and let p_k be the smallest period corresponding to G_k. Also, for each k with $1 \leq k \leq m_i$, let E_k be the union set of all expansions of every palindrome in G_k. Let $H_k = \mathcal{W}_i^L \cap E_k$. Since $|\mathcal{W}_i^L| = |\bigcup_{k=1}^{m_i} H_k| = \sum_{k=1}^{m_i} |H_k|$ and $m_i \in O(\log n)$, it suffices to show that $|H_k|$ is at most a constant.

For the sake of contradiction, we assume $|H_k| \geq 4$. From (2) of Corollary 2.1, at least three palindromes in H_k has the same smallest period p_k. Also, by (1) of Corollary 2.1, the difference between centers of any two palindromes in H_k is a power of $0.5p_k$. Therefore, at least two distinct palindromes in H_k have a difference of power of p_k in their center positions. Let w_1, w_2 be such palindromes, and assume that $|w_1| \leq |w_2|$ w.l.o.g.. Then, the string between the centers of w_1 and w_2 can be represented by x^j for positive integer j and string x of length p_k. Since the smallest period of w_1 is p_k, its extended right arm Rarm_{w_1} can be written by $\mathsf{Rarm}_{w_1} = x^{j_1}x_1'$ where j_1 is a non-negative integer and x_1' is a proper prefix of x. Similarly, the extended right arm Rarm_{w_2} of w_2 can be written by $\mathsf{Rarm}_{w_2} = x^{j_2}x_2'$ where j_2 is a non-negative integer and x_2' is a proper prefix of x. See also Fig. 5.2 for illustration. If $|w_1| = |w_2|$, then this leads $j_1 = j_2$ and $x_1' = x_2'$, i.e., $w_1 = w_2$, a contradiction. If $|w_1| < |w_2|$, then $j_1 < j_2$ or $j_1 = j_2$ and $|x_1'| < |x_2'|$. In both cases, Rarm_{w_1} is a proper prefix of Rarm_{w_2}, i.e, w_1 is a contraction of w_2. This contradicts the minimality of w_2. Thus $|H_k| \leq 3$ holds, and hence, we obtain $|\mathcal{W}_i^L| = \sum_{k=1}^{m_i} |H_k| \leq 3m_i \in O(\log n)$.

Similarly, the size of $\mathcal{W}_i^R$ is also $O(\log n)$. Therefore, $|\mathcal{W}_i| \in O(\log n)$. □

Lemma 5.3. *For each position i in T, the number of MUPSs covering i is $O(\log n)$.*

Proof. By symmetry, it suffices to show that the number of MUPSs covering i and centered *before* i is $O(\log n)$. Each of such MUPSs is an expansion of some palindromic suffix of $T[1..i]$. Thus, similar to the proof of Lemma 5.2, we consider dividing the set of palindromic suffixes of $T[1..i]$ into $m_i \in O(\log i)$ groups, $G_1, G_2, \ldots, G_{m_i}$ w.r.t. their smallest periods. In the following, we consider MUPSs that are expansions of palindromes in an arbitrary group G_k. We show that the number of such MUPSs is at most two by contradiction. We assume the contrary, i.e., there are three MUPSs that are expansions of palindromes in G_k. By (3) of Corollary 2.1, at least one of the three MUPSs is a substring of an expansion of a palindrome in G_k with a different center. This contradicts that a MUPS cannot be a substring of another palindrome with a different center. Thus, the number of MUPSs that are expansions of palindromes in G_k is at most two, and we finish the proof. □

By using Lemmas 5.1, 5.2, and 5.3, we show the following theorem:

Theorem 5.1. $|\mathsf{MUPS}(T) \ominus \mathsf{MUPS}(T')| \in O(\log n)$ *always holds.*

Proof. In the following, we consider the number of MUPSs to be removed.

First, at most one interval can be a MUPS of T centered at i. Also, any other interval in $\mathsf{MUPS}(T)$ covering position i cannot be an element of $\mathsf{MUPS}(T')$ since its corresponding string in T' is no longer a palindrome. By Lemma 5.3, the number of such MUPSs is $O(\log n)$.

Next, let us consider MUPSs not covering position i. When a MUPS w of T not covering i is no longer a MUPS of T', then either (A) w is repeating in T' or (B) w is unique in T' but is not minimal.

Let w_1 be a MUPS of the case (A). Since w_1 does not cover i, is unique in T, and is repeating in T', $|\mathit{inbeg}_{T',i}(w_1)| \geq 1$ and $|\mathit{xbeg}_{T',i}(w_1)| = 1$. Let v_1 be the minimal contraction of w_1 such that $|\mathit{inbeg}_{T',i}(v_1)| \geq 1$ and $|\mathit{xbeg}_{T',i}(v_1)| = 1$. Contrary, w_1 is the only MUPS of the case (A) which is an expansion of v_1 since $|\mathit{xbeg}_{T',i}(v_1)| = 1$. Namely, there is a one-to-one relation between w_1 and v_1. By Lemma 5.2, the number of palindromes that satisfy the above conditions of v_1 is $O(\log n)$. Thus, the number of MUPSs of the case (A) is also $O(\log n)$.

Let w_2 be a MUPS of the case (B). In T', w_2 covers some MUPS as a proper substring since it is not a MUPS and is unique in T'. Let v_2 be the MUPS of T', which is a proper substring of w_2. While v_2 is unique in T', it is repeating in T since w_2 is a MUPS of T. Namely, $|\mathit{inbeg}_{T,i}(v_2)| \geq 1$ and $|\mathit{xbeg}_{T,i}(v_2)| = 1$ hold. Also, v_2 is actually minimal: Let $u_2 = v_2[2..|v_2| - 1]$. If we assume that $|\mathit{inbeg}_{T,i}(u_2)| \geq 1$ and $|\mathit{xbeg}_{T,i}(u_2)| = 1$, then u_2 becomes unique in T', and this contradicts that v_2 is a MUPS of T'. Furthermore, similar to the above discussions, there is a one-to-one relation between w_2 and v_2. Again by Lemma 5.2, the number of palindromes that satisfy the above conditions of v_2 is $O(\log n)$. Thus, the number of MUPSs of the case (B) is also $O(\log n)$.

Therefore, $|\mathsf{MUPS}(T) \setminus \mathsf{MUPS}(T')| \in O(\log n)$ holds. Also, $|\mathsf{MUPS}(T') \setminus \mathsf{MUPS}(T)| \in O(\log n)$ holds by symmetry.

To summarize, $|\mathsf{MUPS}(T) \ominus \mathsf{MUPS}(T')| = |\mathsf{MUPS}(T) \setminus \mathsf{MUPS}(T') \cup \mathsf{MUPS}(T') \setminus \mathsf{MUPS}(T)|$ $= |\mathsf{MUPS}(T) \setminus \mathsf{MUPS}(T')| + |\mathsf{MUPS}(T') \setminus \mathsf{MUPS}(T)| \in O(\log n)$. □

5.2 Algorithms for Updating Set of MUPSs

In this section, we propose an algorithm for updating the set of MUPSs when a single-character in the original string is substituted by another character. We denote by $sub(i, s)$ the substitution query, that is, to substitute $T[i]$ by another character s. First, we define a sub-problem that will be used in our algorithm:

Problem 5.1. *Given a substitution query $sub(i, s)$ on T, compute the longest odd-palindromic substring v of T' such that $center(v) = i$ and v occurs in T if it exists. Also, if such v exists, determine whether v is unique in T or not. Furthermore, if v is unique in T, compute the contraction of v that is a MUPS of T.*

We show the following lemma:

Lemma 5.4. *After $O(n)$-time preprocessing, we can answer Problem 5.1 in $O((\log\log n)^2 + \delta(n, \sigma))$ time where $\delta(n, \sigma)$ denote the time to retrieve any child of the root of the odd-tree of $\mathsf{EERTREE}(T)$.*

Proof. In the preprocessing, we construct $\mathsf{EERTREE}(T)$ and apply the preprocessing for the path-tree LCE queries to the odd-tree $\mathcal{T}_{\mathsf{odd}}$ of $\mathsf{EERTREE}(T)$. Also, we mark the nodes in $\mathsf{EERTREE}(T)$ that correspond to MUPSs of T and apply the preprocessing for the nearest marked ancestor (NMA) queries. The preprocessing time is $O(n)$.

Given a query $sub(i, s)$, we query the path-tree LCE between path $T[i] \rightsquigarrow \mathsf{larm}_w T[i] \mathsf{rarm}_w$ and tree rooted at s on $\mathcal{T}_{\mathsf{odd}}$ where w is the maximal palindrome in T' centered at i. Let ℓ_w be the depth of the LCE nodes. Then the contraction v of w with $|\mathsf{Rarm}_v| = \ell_w$ occurs in T. Also, v is the longest since any other contraction u of w longer than v does not occur in T. Further, we can determine whether v is unique in T or not by checking the existence of a mark on path $s \rightsquigarrow v$. It can be done by querying NMA, and the MUPS of T contained in v can be computed simultaneously, if v is unique in T.

We can compute the value ℓ_w in $\delta(n, \sigma)$ time for searching for the node s in $\mathcal{T}_{\mathsf{odd}}$, plus $O((\log\log n)^2)$ time for the path-tree LCE query. □

When $\sigma \in O(n)$, we can easily achieve $\delta(n, \sigma) \in O(1)$ with linear space, by using an array of size σ. Otherwise, we achieve $\delta(n, \sigma) \in O(\log \sigma)$ for a general ordered alphabet by using a binary search tree.

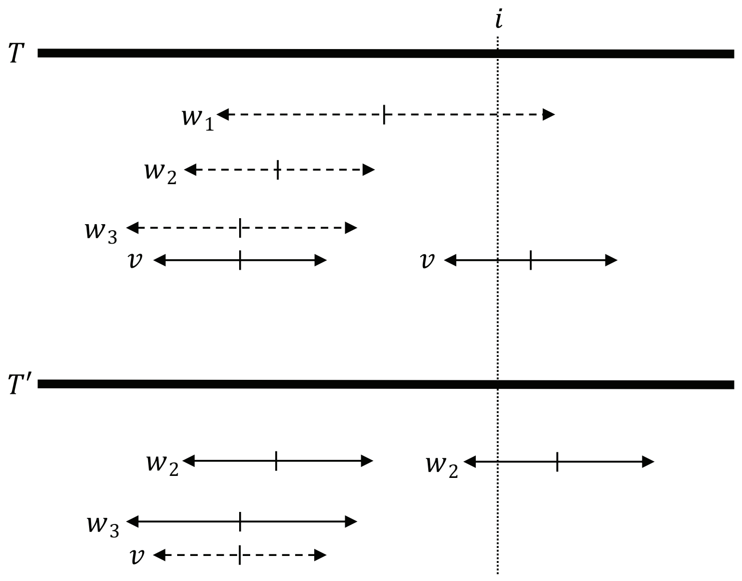


Figure 5.3: Illustration for three types of MUPSs to be removed. The broken arrows represent MUPSs. w_1, w_2, and w_3 are MUPSs of Type R1, Type R2, and Type R3 in T, respectively. Also, v is the MUPS of T' that is a contraction of w_3. It is not unique in T, but is unique in T'.

In the rest of this chapter, we propose an algorithm to compute the changes in MUPSs after a single-character substitution. We compute MUPSs to be removed and added separately. We show how to compute all MUPSs to be removed in Section 5.2.1. Also, we show how to compute all MUPSs to be added in Section 5.2.2. In Section 5.2.3, we introduce another solution for Problem 5.1. Our strategy is basically to pre-compute changes in MUPSs for some queries as much as possible within linear time. The other changes will be detected on the fly by using some data structures.

5.2.1 Computing MUPSs to be Removed

We categorize MUPSs to be removed into three types as follows:

R1) A MUPS of T that covers i.

R2) A MUPS of T that does not cover i and is repeating in T'.

R3) A MUPS of T that does not cover i and is unique but not minimal in T'.

See Fig. 5.3 for illustration. In the following, we describe how to compute all MUPSs for each type separately.

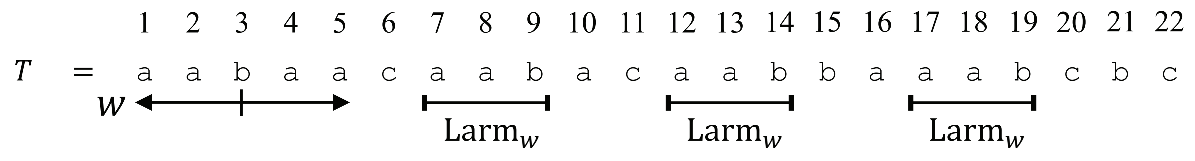


Figure 5.4: Example for observations about Type R2, where T = aabaacaabacaabbaaabcbc and w = aabaa. Larm_w = aab occurs at position 7, 12, and 17 excluding the occurrence of w. Since the Hamming distance between $T[10..11]$ and $w[4..5]$ equals 1, w occurs at position 7 when $T[11]$ is substituted by a. Also, w occurs at position 12 when $T[15]$ is substituted by a. Conversely, w cannot occur at 17 after any single-character substitution since the Hamming distance between $T[20..21]$ and $w[4..5]$ equals 2.

Type R1

All MUPSs covering editing position i are always removed. Thus, we can detect them in $O(1+\alpha_{rem})$ time after a simple linear time preprocessing (e.g., using stabbing queries), where α_{rem} is the number of MUPSs of Type R1.

Type R2

Before describing our algorithm, we give a few observations about MUPSs of Type R2. Let w be a MUPS of Type R2. Since w is unique in T and is repeating in T', $|inbeg_{T',i}(w)| \geq 1$. When w occurs in T' centered at editing position i, we retrieve such w by applying Problem 5.1. If it is not the case, we can utilize the following observations: Consider the starting position j of an occurrence of w in T' such that $T'[j..j+|w|-1] = w$ and $i \in [j, j+|w|-1]$. If position i is covered in the right arm of $T'[j..j+|w|-1]$, then Larm_w occurs at position j in both T and T'. Further, the Hamming distance between $T[j+|\mathsf{Larm}_w|..j+|w|-1]$ and $w[|\mathsf{Larm}_w|+1..|w|] = \mathsf{rarm}_w$ equals 1. Namely, for each occurrence at position k of string Larm_w in T, w can occur at k in T' only if the Hamming distance between $T[k+|\mathsf{Larm}_w|..k+|w|-1]$ and $w[|\mathsf{Larm}_w|+1..|w|]$ equals 1. In other words, if the Hamming distance is greater than 1, w cannot occur at k in T'. See also Fig. 5.4. The strategy of the algorithm for MUPSs of Type R2 is following: A MUPS of Type R2 having occurrence centered at editing position i in T' is found by applying Problem 5.1. For the remaining MUPSs of Type R2, we precompute them by using the above observations.

Preprocessing In the preprocessing phase, we first apply the $O(n)$-time preprocessing of Lemma 5.4 for Problem 5.1. Next, we initialize the set $\mathcal{A}_{R2} = \emptyset$. The set $\mathcal{A}_{R2}$ will become an *index* of MUPSs of Type R2 when the preprocessing is finished. For each MUPS

$w = T[b..e]$ of T, we process the followings: For the beginning position $j \neq b$ of each occurrence of Larm_w in T, we first compute the lcp value between $T[j + |\mathsf{Larm}_w|..|T|]\$$ and rarm_w with allowing one mismatch. Note that $T[j + |\mathsf{Larm}_w|..|T|]\$$ must have at least one mismatch with rarm_w, since $T[j..j + |\mathsf{Larm}_w| - 1] = \mathsf{Larm}_w$, $j \neq b$, and $T[b..e]$ is unique in T. If there are two mismatch positions between them, do nothing for this occurrence since w cannot occur at j after any substitution. We can check this by querying LCE at most twice. Otherwise, let $q = j + |\mathsf{Larm}_w| - 1 + d$ be the mismatched position in T. When the q-th character of T is substituted by the character $\mathsf{rarm}_w[d]$, $w = T[b..e]$ occurs at $j \neq b$, i.e., it is a MUPS of Type R2. So we add MUPS $w = T[b..e]$ into $\mathcal{A}_{R2}$ with the pair of index and character $(q, \mathsf{rarm}_w[d])$ as the key. In addition, symmetrically, we update $\mathcal{A}_{R2}$ for each occurrence of Rarm_w in T. After finishing the above processes for every MUPS of T, we sort the elements of $\mathcal{A}_{R2}$ by radix sort on the keys. If there are multiple identical elements with the same key, we unify them into a single element. Also, if there are multiple elements with the same key, we store them in a linear list. By Lemma 2.4, the total number of occurrences of arms of MUPSs is $O(n)$, and hence, the total preprocessing time is $O(n)$.

Query Given a query $sub(i, s)$, we query Problem 5.1 with the same pair (i, s) as the input. Then, we complete checking whether there exists a MUPS of Type R2 centered at i. Next, consider the existence of the remaining MUPSs of Type R2. First, an element in $\mathcal{A}_{R2}$ corresponding to the key (i, s) can be detected in $O(\log \sigma_i)$ time by using random access on indices and binary search on characters, where σ_i is the number of characters s_i such that the key (i, s_i) exists in $\mathcal{A}_{R2}$. After that, we can enumerate all the other elements with the key by scanning the corresponding linear list. Thus, the total query time is $O(\delta(n, \sigma) + (\log \log n)^2 + \log \sigma_i + \beta_{rem})$ where β_{rem} is the number of MUPSs of Type R2. Finally, we show $\sigma_i \in O(\min\{\sigma, \log n\})$. Let us consider palindromes in T' whose right arm covers position i. Those whose left arms cover i can be treated similarly. Any palindrome in T' whose right arm covers i is an expansion of some maximal palindrome in T ending at $i - 1$. It is known that the number of possible characters immediately preceding such maximal palindromes is $O(\log n)$ [42]. Therefore, $\sigma_i \in O(\log n)$ holds, and thus, the query time is $O(\delta(n, \sigma) + (\log \log n)^2 + \beta_{rem})$.

Type R3

Let $w = T[b..e]$ be a MUPS of T and let $v = T[b{+}1..e{-}1]$. Further let $T[b_{l1}..e_{l1}]$ and $T[b_{r1}..e_{r1}]$ be the leftmost and the rightmost occurrence of v in T except for $T[b + 1..e - 1]$. We define interval $\rho_w = \{k \mid k \notin [b + 1, e - 1] \text{ and } k \in [b_{r1}, e_{l1}]\}$. Note that ρ_w can be empty. See also

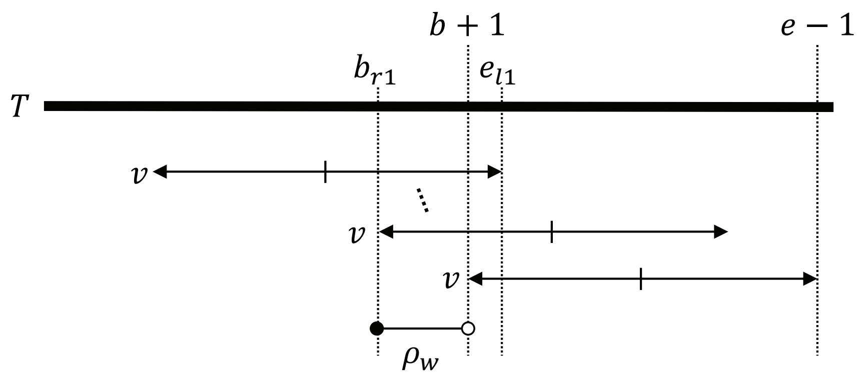


Figure 5.5: Illustration for ρ_w of Type R3. The top arrow (resp. the middle arrow) represents the leftmost (resp. rightmost) occurrence of v except for $T[b+1..e-1]$. Also, the bottom arrow represents $T[b+1..e-1]$. In this case, $\rho_w = [b_{r1}, b]$.

Fig. 5.5 for illustration. If the editing position i is in ρ_w, then the only occurrence of v in T' is $T'[b+1..e-1]$, i.e., v is unique in T'. Thus, w is a removed MUPS of Type R3. Contrary, if $i \notin [b, e]$ and $i \notin \rho_w$, there are at least two occurrences of v in T', i.e., w cannot be a MUPS of Type R3.

Preprocessing First, we compute the set of intervals $\mathcal{R} = \{\rho_w \mid w \text{ is a MUPS of } T\}$. $\mathcal{R}$ can be computed by traversing over the suffix tree of T enhanced with additional explicit nodes, each of which represents a substring $T[b+1..e-1]$ for each MUPS $T[b..e]$ of T. Also, we apply the preprocessing for stabbing queries to $\mathcal{R}$. The total time for preprocessing is $O(n)$.

Query Given a query $sub(i, s)$, compute all intervals in $\mathcal{R}$ stabbed by position i by answering a stabbing query. They correspond to MUPSs of Type R3. The query time is $O(1+\gamma_{rem})$, where γ_{rem} is the number of MUPSs of Type R3.

To summarize, we can compute all MUPSs to be removed after a single-character substitution in $O(\delta(n,\sigma) + (\log\log n)^2 + \alpha_{rem} + \beta_{rem} + \gamma_{rem})$ time.

5.2.2 Computing MUPSs to be Added

Next, we propose an algorithm to detect MUPSs to be added after a substitution. As in Section 5.2.1, we categorize MUPSs to be added into three types:

A1) A MUPS of T' that covers i.

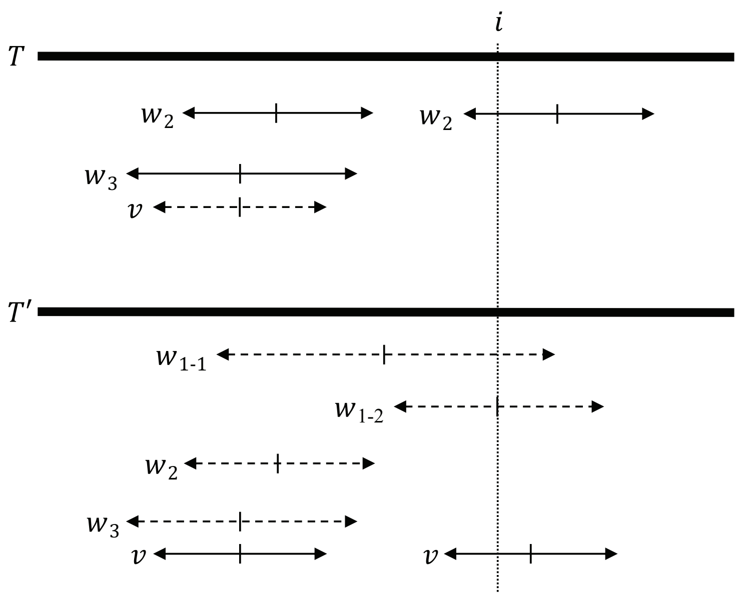


Figure 5.6: Illustration for four types of MUPSs to be added. The broken arrows represent MUPSs. $w_{1\text{-}1}$, $w_{1\text{-}2}$, w_2, and w_3 are MUPSs of Type A1-1, Type A1-2, Type A2, and Type A3 in T', respectively. Also, v is the MUPS of T that is a contraction of w_3. It is unique in T, but is not unique in T'.

A2) A MUPS of T' that does not cover i and is repeating in T.

A3) A MUPS of T' that does not cover i and is unique but not minimal in T.

Furthermore, we categorize MUPSs of Type A1 into two sub-types:

A1-1) A MUPS of T' that covers position i in its arm.

A1-2) A MUPS of T' centered at editing position i.

See Fig. 5.6 for illustration.

Type A1-1

A MUPS of Type A1-1 is a contraction of some maximal palindrome in T' covering editing position i in its arm. Further, such a maximal palindrome in T' corresponds to some 1-mismatch maximal palindrome in T, which covers i as a mismatch position. Thus, we preprocess for arms of each 1-mismatch maximal palindrome in T. For MUPSs of Type A1, we utilize the following observation:

Observation 5.1. *For any palindrome v covering position i in T', v is unique in T' if and only if $|inbeg_{T',i}(v)| = 1$ and $|xbeg_{T',i}(v)| = 0$.*

Preprocessing In the preprocessing phase, we first consider sorting extended arms of 1-mismatch maximal palindromes in T. Let EA be the multiset of strings that consists of the extended right arms and the reverse of the extended left arms of all 1-mismatch maximal palindromes in T. Note that each string in EA can be represented in constant space since it is a substring of T or T^R. Let MA' be a lexicographically sorted array of all elements in EA. Here, the order between the same strings can be arbitrary. Also, for each string in EA, we consider a quadruple of the form (par, pos, chr, rnk) where $par \in \{\mathsf{odd}, \mathsf{even}\}$ represents the parity of the length of the corresponded 1-mismatch maximal palindrome, pos is the mismatched position on the *opposite* arm, chr is the mismatched character on the extended arm, and rnk is the rank of the extended arm in MA'. Let MA be a radix sorted array of these quadruples. It can be seen that for each triple (p, i, s) of parity p, mismatched position i, and mismatched character s, all elements corresponding to the triple are stored continuously in MA. We denote by $\mathsf{MA}_{p,i,s}$ the subarray of MA consists of such elements. In other words, $\mathsf{MA}_{p,i,s}$ is a sorted array of extended arms of maximal palindromes of parity p covering position i in T' when the i-th character of T is substituted by s.

Now let us focus on odd-palindromes. Even-palindromes can be treated similarly. We construct the suffix tree of T and make the loci of strings in EA explicit. We also make the loci of the extended right arm of every odd-palindrome in T explicit. Simultaneously, we *mark* the nodes corresponding to the extended right arms and apply the preprocessing for the nearest marked ancestor (NMA) queries to the marked tree. We denote the tree by $\mathcal{ST}_{odd}$. Next, we initialize the set $\mathcal{A}_{A1,1} = \emptyset$. The set $\mathcal{A}_{A1,1}$ will become an index of MUPSs of Type A1-1 when the preprocessing is finished. For each non-empty $\mathsf{MA}_{\mathsf{odd},i,s}$ and for each string w in $\mathsf{MA}_{\mathsf{odd},i,s}$, we do the followings: Let x_w be the odd-palindrome whose extended right arm is w when $T[i]$ is substituted by s. Let u and v are the preceding and the succeeding string of w in $\mathsf{MA}_{\mathsf{odd},i,s}$ (if such palindromes do not exist, they are empty). Further let $\ell_w = \max\{lcp(u, w), lcp(w, v)\}$. When $T[i]$ is substituted by s, any contraction y of x_w such that y covers position i and the arm-length of y is at least ℓ_w has only one occurrence which covers position i in T', i.e.,$|inbeg_{T',i}(y)| = 1$. Next, we query the NMA for the node corresponding to w on $\mathcal{ST}_{odd}$. Let ℓ'_w be the length of the extended right arm obtained by the NMA query. When $T[i]$ is substituted by s, any contraction y' of x_w such that the arm-length of y' is at least ℓ'_w, has no occurrences which do not cover position i in T', i.e., $|xbeg_{T',i}(y')| = 0$. Thus, by Observation 5.1, the contraction $y^\star$ of x_w of arm-length $\max\{\ell_w, \ell'_w\}$ is a MUPS of Type A1-1 for the query $sub(i, s)$, if such $y^\star$ exists. In such a case, we store the information about $y^\star$ (i.e., its center and radius) into $\mathcal{A}_{A1,1}$ using (odd, i, s) as the key. After finishing the above preprocessing for all strings in MA, we sort all elements in $\mathcal{A}_{A1,1}$ by their keys.

Since each element in EA is a substring of $T\$T^R\#$, they can be sorted in $O(n + |EA|) = O(n)$ time by Corollary 2.2. Namely, MA' can be computed in linear time, and thus MA too. By Lemma 2.5, tree $\mathcal{ST}_{odd}$ can be constructed in $O(n)$ time. Also, we can answer each NMA query and LCP query in constant time after $O(n)$ time preprocessing. Hence, the total preprocessing time is $O(n)$.

Query Given a query $sub(i, s)$, we compute all MUPSs of Type A1-1 by searching for elements in $\mathcal{A}_{A1,1}$ with keys (odd, i, s) and (even, i, s). An element with each of the keys can be found in $O(\log \min\{\sigma, \log n\})$ time. Thus, all MUPSs of Type A1-1 can be computed in $O(\log \min\{\sigma, \log n\} + \alpha'_{add})$ time where α'_{add} is the number of MUPSs of Type A1-1.

Type A1-2

The MUPS of Type A1-2 is a contraction of the maximal palindrome in T' centered at i. By definition, there is at most one MUPS of Type A1-2.

Preprocessing In the preprocessing phase, we again construct MA and related data structures as in Type A1-1. Further, we apply the $O(n)$-time preprocessing of Lemma 5.4 for Problem 5.1. The total preprocessing time is $O(n)$.

Query Given substitution query $sub(i, s)$, we compute the MUPS centered at i in T' as follows (if it exists): It is clear that $T'[i..i] = s$ is the MUPS of Type A1-2 if s is a unique character in T'. In what follows, we consider the other case. Let w be the maximal palindrome centered at i in T'. First, we compute the maximum lcp value ℓ_w between Rarm_w and extended arms in $\mathsf{MA}_{\mathsf{odd},i,s}$. Then, any contraction y of w, such that the arm-length of y is at least ℓ_w, has no occurrences which cover position i in T', i.e., $|inbeg_{T',i}(y)| = 1$. We can compute ℓ_w in $O(\log \min\{\sigma, \log n\})$ time, combining LCE queries and binary search. Note that rarm_w occurs at $i+1$ in both T and T' while Rarm_w might be absent from T. Next we compute the arm-length ℓ'_w of the shortest palindrome v such that $center(v) = i$ and $|xbeg_{T',i}(v)| = 0$, i.e., v is absent from T. Since the contraction $\tilde{v}$ of w of arm-length $\ell'_w - 1$ is the longest palindrome such that $center(\tilde{v}) = i$ and $\tilde{v}$ occurs in T, we can reduce the problem of computing ℓ'_w to Problem 5.1. Thus, we can compute ℓ'_w in $O(\delta(n, \sigma) + (\log \log n)^2)$ time by Lemma 5.4. Similar to the case of Type A1-1, by Observation 5.1, the contraction $y^\star$ of x_w of arm-length $\max\{\ell_w, \ell'_w\}$ is a

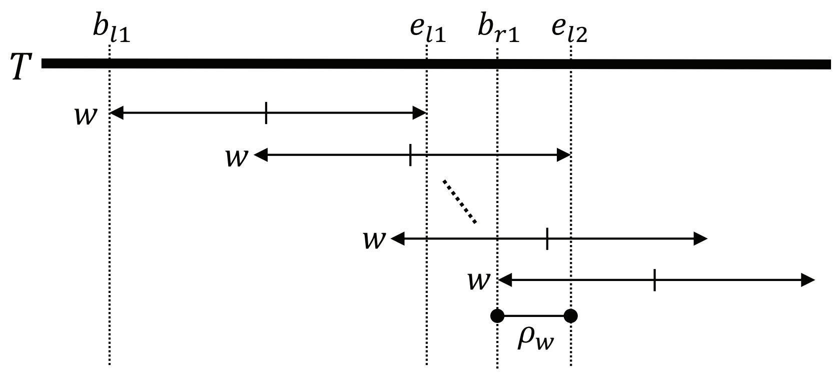


Figure 5.7: Illustration for ρ_w of Type A2. The top two arrows represent the leftmost and the second leftmost occurrence of w. Also, the bottom two arrows represent the second rightmost and the rightmost occurrence of w. In this case, $\rho_w = [b_{r1}, e_{l2}]$.

MUPS of Type A1-2, if such $y^\star$ exists. Therefore, the MUPS of Type A1-2 can be computed in $O(\delta(n, \sigma) + (\log\log n)^2)$ time.

Type A2

A MUPS of Type A2 occurs at least twice in T, and there is only one occurrence not covering editing position i. For a palindrome w repeating in T, let $T[b_{l1}..e_{l1}]$ and $T[b_{l2}..e_{l2}]$ be the leftmost and the second leftmost occurrence of w in T. Further, let $T[b_{r1}..e_{r1}]$ and $T[b_{r2}..e_{r2}]$ be the rightmost and the second rightmost occurrence of w in T. We define interval ρ_w as the intersection of all occurrences of w except for the leftmost one, i.e., $\rho_w = \{k \mid k \notin [b_{l1}, e_{l1}]$ and $k \in [b_{r1}, e_{l2}]\}$ (see also Fig. 5.7). Similarly, we define interval $\tilde{\rho}_w$ as the intersection of all occurrences of w except for the rightmost one. Note that ρ_w and $\tilde{\rho}_w$ can be empty. Then, w is unique after the i-th character is edited if and only if $i \in \rho_w \cup \tilde{\rho}_w$. Thus, any MUPS of Type A2 is a palindrome corresponding to some interval in $\rho_w \cup \tilde{\rho}_w$ stabbed by i. To avoid accessing intervals that do not correspond to the MUPSs to be added, we decompose each ρ_w. It is easy to see that for any contraction v of w, $\rho_v \subset \rho_w$ holds. Also, if $T[i]$, with $i \in \rho_v$, is edited, then both w and v become unique in T', i.e., w cannot be a MUPS of T'. For each unique palindrome w in T, we decompose ρ_w into at most three intervals $\rho_w = \rho_w^1 \rho_{w'} \rho_w^2$ where $w' = w[2..|w|-1]$. Similarly, we decompose $\tilde{\rho}_w$ into $\tilde{\rho}_w = \tilde{\rho}_w^1 \tilde{\rho}_{w'} \tilde{\rho}_w^2$. Then, w is a MUPS of Type A2 if and only if $i \in \rho_w^1 \cup \rho_w^2 \cup \tilde{\rho}_w^1 \cup \tilde{\rho}_w^2$.

Preprocessing In the preprocessing phase, we first construct the eertree of T and the suffix tree of T enhanced with additional explicit nodes for all distinct palindromes in T. Next, we compute at most four (leftmost, second leftmost, rightmost, second rightmost) occurrences of each palindrome in T by traversing the enhanced suffix tree. At the same time, we compute ρ_w and $\tilde{\rho}_w$ for each palindrome in w. Next, we sequentially access distinct palindromes by traversing $\mathsf{EERTREE}(T)$ in a pre-order manner. Then, for each palindrome w, we decompose ρ_w and $\tilde{\rho_w}$ based on the rules as mentioned above. Finally, we apply the preprocessing for stabbing queries to the $O(n)$ intervals obtained. The total preprocessing time is $O(n)$.

Query Given a query $sub(i, s)$, we compute all intervals stabbed by position i. The palindromes corresponding to the intervals are MUPSs of Type A2. Hence, the query time is $O(1 + \beta_{add})$, where β_{add} is the number of MUPSs of Type A2.

Type A3

A MUPS of Type A3 is unique but not minimal in T. Such a unique palindrome u in T contains a MUPS $w \neq u$ of T as a contraction. Since u is a MUPS of T', w is repeating in T', i.e., w is a removed MUPS of Type R2. Contrary, consider a MUPS w of Type R2, which is repeating in T'. Then, the shortest unique expansion of w in T' is an added MUPS of Type A3, if it exists. The preprocessing for Type A3 is almost the same as for Type R2. We store a bit more information for Type A3 in addition to the information in $\mathcal{A}_{R2}$.

Preprocessing In the preprocessing phase, we first apply the $O(n)$-time preprocessing of Lemma 5.4 for Problem 5.1. Next, we initialize the set $\mathcal{A}_{A3} = \emptyset$. This set $\mathcal{A}_{A3}$ will become an index of MUPSs of Type A3 when the preprocessing is finished. For each MUPS $w = T[b..e]$ of T, we process the followings: For the beginning position $j \neq b$ of each occurrence of Larm_w in T, we compute the lcp value ℓ_j between $T[j + |\mathsf{Larm}_w|..|T|]\$$ and $T[\lceil c \rceil..|T|]\$$ with allowing one mismatch where c is the center of w in T. If ℓ_j is smaller than rarm_w, then we do nothing for this occurrence since w cannot occur at j after any single-character substitution. Otherwise, let $q = j + |\mathsf{Larm}_w| - 1 + d$ be the first mismatched position in T. When the q-th character of T is substituted by the character $\mathsf{rarm}_w[d]$, $w = T[b..e]$ occurs at $j \neq b$, i.e., it is a MUPS of Type R2. Unlike for Type R2, we add the *pair* of MUPS and (1-mismatched) lcp value $(T[b..e], \ell_j)$ into $\mathcal{A}_{A3}$ with the pair of index and character $(q, \mathsf{rarm}_w[d])$ as the key. In addition, symmetrically, we update $\mathcal{A}_{A3}$ for each occurrence of Rarm_w in T. After finishing the above processes for every MUPS of T, we then sort the elements of $\mathcal{A}_{A3}$ by radix sort on the keys. If there are multiple identical elements with the same key, we unify them into a single element. Also, if there are multiple elements with the same key, we store them in a linear list. By Lemma 2.4, the total number of occurrences of arms of MUPSs is $O(n)$, and hence, the total preprocessing time is $O(n)$.

Query Given a query $sub(i, s)$, we query Problem 5.1 with the same pair (i, s) as the input. Then, we complete checking whether there exists a MUPS of Type A3 centered at i. For the remaining MUPSs of Type 3, we retrieve the MUPSs of Type A3 using the index $\mathcal{A}_{A3}$ as in the query algorithm for Type R2. This can be done in $O(\log \min\{\sigma, \log n\} + \gamma_{add})$ time where γ_{add} is the number of MUPSs of Type A3. Therefore, the total query time of Type A3 is $O(\delta(n, \sigma) + (\log \log n)^2 + \gamma_{add})$.

To summarize, we can compute all MUPSs to be added after a single-character substitution in $O(\delta(n, \sigma) + (\log \log n)^2 + \alpha'_{add} + \beta_{add} + \gamma_{add})$ time. Then, combining the results of Section 5.2.1 with the above results, we obtain the following theorem:

Theorem 5.2. *After $O(n)$-time preprocessing, we can compute the set of MUPSs after a single-character substitution in $O(\delta(n, \sigma) + (\log \log n)^2 + d) \subset O(\log n)$ time where d is the number of changes of MUPSs.*

5.2.3 Alternative Algorithm for Problem 5.1

The query time of Theorem 5.2 is dominated by the time to answer Problem 5.1. Here, we introduce another solution for Problem 5.1 utilizing *nearest colored ancestor queries* instead of path-tree LCE queries.

Preprocessing for Problem 5.1 We first construct the suffix tree of $T\$$. Also, for each odd-palindrome in T, we make the locus of the right arm explicit and label the node with the pair of the center character and the binary flag that indicates if the palindrome is a MUPS. We regard the pair as the *color* of the node. Furthermore, we apply a preprocessing for NCA queries to the colored tree[1]. The preprocessing time is $O(n + c_{\mathsf{nca}}(n, \sigma))$, where $c_{\mathsf{nca}}(n, \sigma)$ is the preprocessing time for NCA queries.

Query for Problem 5.1 Given a substitution query $sub(i, s)$, we start at the node corresponding to rarm_w where w is the maximal palindrome in T centered at i. We then compute the

[1]There can be a node with multiple colors in the tree. However, we can easily avoid such a situation by copying a node with k colors to k nodes. Also, in the case of Problem 5.1, the cumulative total number of colored nodes is $O(n)$.

	Time			
σ	Construction	Query	Space	Ref.
$n^{O(1)}$	$O(n)$	$O(\log\sigma + (\log\log n)^2 + d)$	$O(n)$	Theorem 5.2, 2.1
$n^{O(1)}$	$O(n \log\log n)$	$O(\log\log n + d)$	$O(n)$	Theorem 5.3, Lemma 2.6
$n^{O(1)}$	expected $O(n)$	$O(\log\log n + d)$	$O(n)$	Theorem 5.3, Lemma 2.6
$O(n)$	$O(n)$	$O((\log\log n)^2 + d)$	$O(n)$	Theorem 5.2, 2.1
$O(\log n)$	$O(n)$	$O(\log\log n + d)$	$O(n)$	Theorem 5.3, Lemma 2.7
$O(1)$	$O(n)$	$O(1 + d)$	$O(n)$	Theorem 5.3, Lemma 2.7

Table 5.1: Concrete complexities of our algorithms for the problem of computing MUPSs after a single-character substitution. All the above results require only linear space. Each query time is $O(\log n)$ since $\log\sigma \in O(\log n)$ and $d \in O(\log n)$.

nearest ancestor V colored with $(s, 0)$ by using NCA query. If such node V exists, palindrome $P = \mathsf{str}(V)^R \cdot s \cdot \mathsf{str}(V)$ is the answer of the former part of Problem 5.1 where $\mathsf{str}(V)$ denotes the string corresponding to V in the enhanced suffix tree of $T\$$. Also, we query NCA $(s, 1)$ from V. We can determine if P is unique, and if it is unique, we can find the MUPS contained in P. The query time is $O(q_{\mathsf{nca}}(n, \sigma))$ where $q_{\mathsf{nca}}(n, \sigma)$ is the query time for NCA.

Let $s_{\mathsf{nca}}(n, \sigma)$ denote the space for the NCA data structure. We obtain the following theorem:

Theorem 5.3. *After $O(n + c_{\mathsf{nca}}(n, \sigma))$-time and $O(n + s_{\mathsf{nca}}(n, \sigma))$-space preprocessing, we can compute the set of MUPSs after a single-character substitution in $O(\log\min\{\sigma, \log n\} + q_{\mathsf{nca}}(n, \sigma) + d)$ time.*

The results for NCA queries in Lemmas 2.6 and 2.7 can be plugged into the functions c_{nca}, q_{nca}, and s_{nca}. In addition, even when a general case, we can handle $\delta(n, \sigma)$ as a constant by utilizing a perfect hashing [38] after $O(n \log\log n)$-time or $O(n)$-expected time preprocessing. Table 5.1 lists different representations of the time/space complexities of Theorems 5.2 and 5.3. We emphasize that our algorithm runs in optimal $O(1 + d)$ time when σ is constant.

Corollary 5.1. *If $\sigma \in O(1)$, after $O(n)$-time and $O(n)$-preprocessing, we can compute the set of MUPSs after a single-character substitution in $O(1 + d)$ time.*

Chapter 6

Palindromes in a Trie

A trie can be seen as a representation of a set of strings which are root-to-leaf path labels. Therefore, the online algorithm of a trie can be regarded as the online algorithm for some strings. In this chapter, we tackle the problems of computing maximal palindromes and distinct palindromes in a trie in an online manner. In Section 6.1, we show tight bounds on the number of maximal palindromes and on the number of distinct palindromes in a trie. In Section 6.2, we present two algorithms to compute all maximal palindromes in a trie. In Section 6.3, we present how to compute all distinct palindromes in a trie.

The results in this chapter primarily appeared in [41].

6.1 Maximal/Distinct Palindromes in a Trie

In this section, we show tight bounds on the number of maximal palindromes and on the number of distinct palindromes in a trie.

Consider a trie $\mathcal{T}$ with N edges. A substring palindrome $P = \mathsf{str}(u, v)$ in $\mathcal{T}$ can be represented by the pair $(|P|, v)$ of its length and the ending point v. Since the reversed path from v to u is unique and since P is a palindrome, one can retrieved P from $\mathcal{T}$ in $O(|P|)$ time from this pair $(|P|, v)$.

A substring palindrome $\mathsf{str}(u, v)$ is called a *maximal palindrome* in $\mathcal{T}$ if

(1) $\mathsf{str}(\mathsf{parent}(u), v')$ is not a palindrome with *any* child v' of v,

(2) u is the root, or

(3) v is a leaf.

Lemma 6.1. *There are exactly $2N - L$ maximal palindromes in any trie $\mathcal{T}$ with N edges and L leaves.*

Proof. Let r be the root of $\mathcal{T}$ and u any internal node of $\mathcal{T}$. Because the reversed path from u to r is unique, and because the out-going edges of u are labeled by pairwise distinct characters, there is a unique longest palindrome of even length (or length zero) that is centered at u. Since there are $N + 1$ nodes in $\mathcal{T}$, there are exactly $(N + 1) - L - 1 = N - L$ maximal palindromes of even length in $\mathcal{T}$.

Let $e = (u, v)$ be any edge in $\mathcal{T}$. From the same argument as above, there is a unique longest palindrome of odd length that is centered at e. Thus there are exactly N maximal palindromes of odd length in $\mathcal{T}$. □

For any trie $\mathcal{T}$, let $\mathbf{P}_{\mathcal{T}} \subset \Sigma^*$ be the set of all strings such that each $P \in \mathbf{P}_{\mathcal{T}}$ is a substring palindrome in $\mathcal{T}$. We call the elements of $\mathbf{P}$ as *distinct palindromes* in $\mathcal{T}$.

Lemma 6.2. *There are at most $N + 1$ distinct palindromes in any trie $\mathcal{T}$ with N edges.*

Proof. We follow the proof from [33] which shows that the number of distinct palindromes in a string of length n is at most $n + 1$.

We consider a top-down traversal on $\mathcal{T}$. The proof works with any top-down traversal but for consistency with our algorithm to follow, let us consider a breadth first traversal. Let r be the root of $\mathcal{T}$ and let $\mathcal{T}_0$ be the trie consisting only of the root r. For each $1 \leq i \leq n$, let $e_i = (u_i, v_i)$ denote the ith visited edge in the traversal, and let $\mathcal{T}_i$ denote the subgraph of $\mathcal{T}_i$ consisting of the already visited edges when we have just arrived at e_i. Since we have just added e_i to $\mathcal{T}_{i-1}$, it suffices to consider only suffix palindromes of $\mathsf{str}(r, v_i)$ since any other palindromes in $\mathsf{str}(r, v_i)$ already appeared in $\mathcal{T}_{i-1}$. Moreover, only the longest suffix palindrome S_i of $\mathsf{str}(r, v_i)$ can be a new palindrome in $\mathcal{T}_i$ which does not exist in $\mathcal{T}_{i-1}$, since any shorter suffix palindrome S' is a suffix of S_i and hence is a prefix of S_i, which appears in $\mathcal{T}_{i-1}$. Thus there can be at most $N + 1$ distinct palindromes in $\mathcal{T}$ (including the empty string). □

See Figure 6.1 for examples of maximal palindromes and distinct palindromes in a trie.

In the next sections, we will present our algorithms to compute maximal/distinct palindromes from a given trie.

6.2 Computing Maximal Palindromes in a Trie

In this section, we present two algorithms that compute all maximal palindromes in a given trie.

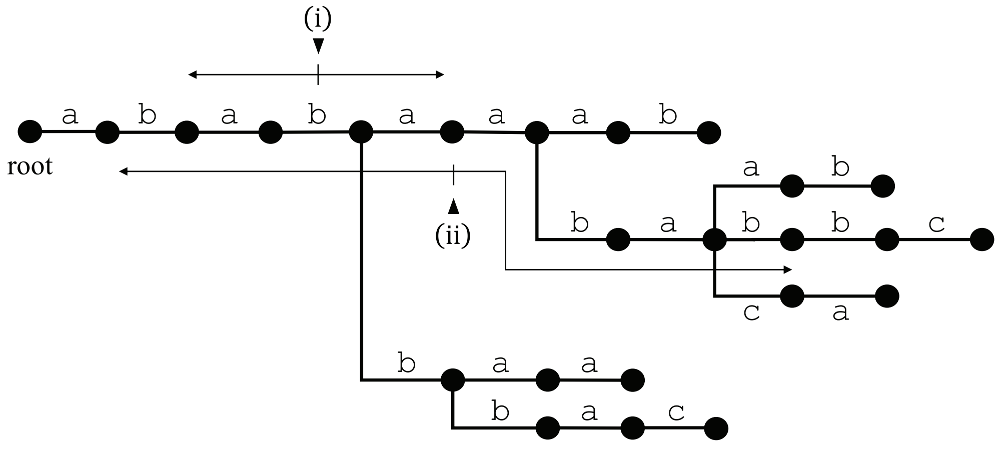


Figure 6.1: The maximal palindrome centered at (i) is aba and the maximal palindrome centered at (ii) is babaabab. The set of distinct palindromes in this trie is $\{\varepsilon$, a, b, c, aa, bb, aaa, aba, aca, bab, bbb, abba, baab, aabaa, ababa, abbba, baaab, abaaba, baabaab, babaabab$\}$.

6.2.1 $O(N \log h)$-Time $O(h)$-Space Algorithm

In this subsection, we present an algorithm that compute all maximal palindromes in a given trie $\mathcal{T}$ in $O(N \log h)$ time and $O(h)$ working space, where N is the number of edges in $\mathcal{T}$ and $h \leq N$ is the height of $\mathcal{T}$.

The basic strategy of our algorithm is as follows. We perform a depth-first traversal on $\mathcal{T}$. Let r be the root of $\mathcal{T}$. We use Lemma 2.2 in our algorithm. When visiting a node u during the depth-first traversal on trie $\mathcal{T}$, we maintain the arithmetic progressions for the maximal palindromes in the path string $\mathsf{str}(r, u)$. In each node x in the path from r to u, the arithmetic progressions representing the maximal palindromes ending at x are sorted in the increasing order of the lengths of the corresponding maximal palindromes. Since $\mathsf{str}(r, u)$ is a single string, and since $|\mathsf{str}(r, u)|$ is bounded by the height h of $\mathcal{T}$, we can store all these arithmetic progressions in $O(h)$ total space during the traversal. Suppose that u has two or more children, and let v, v' be two distinct children of u. Notice that some of the maximal palindromes ending at u could be extended by the edge label from u to v. Furthermore, since the edge label between u and v differs from the edge label between u and v', those palindromes that are not extended with v could still be extended with v'. This in turn means that when we backtrack to u after visiting v, then we can use the maximal palindromes in the path string $\mathsf{str}(r, v)$ that ends at the parent u of v, for finding the palindromes ending at another child v'. In the sequel, we will describe how to efficiently maintain these maximal palindromes during the traversal.

Suppose that now we are to process non-leaf node u in the traversal. For each $1 \leq i \leq |\mathsf{children}(u)|$, let v_i be the ith visited child of u in the tree traversal, and let a_i be the label of the edge (u, v_i). The task here is to check if the suffix palindromes ending at u extends with a_i. We will process the groups of suffix palindromes ending at u in increasing order of their lengths. Let $\langle s, d, t\rangle$ be the arithmetic progression representing a given group of suffix palindromes ending at u, where s is the length of the shortest suffix palindrome in the group, d is a common period of the suffix palindromes and t is the number of suffix palindromes in this group. The cases where $t = 1$ and $t = 2$ are trivial, so we consider the case where $t \geq 3$. Let P be any suffix palindrome in the group that is not the longest one (i.e, $s \leq |P| \leq s + (t-2)d$). Due to the periodicity (Claim (iv) of Lemma 2.2), every P is immediately preceded by a unique string $P[1..d]$ of length d. Let $b = P[d]$ and c be the character that immediately precedes the longest suffix palindrome in the group. There are four cases to consider:

1. $a_i = b$ and $a_i = c$ (namely $a_i = b = c$): In this case, all the suffix palindromes in the group extend with a_i and become suffix palindromes of $\mathsf{str}(r, v_i)$. We update $s \leftarrow s + 2$. The values of d and t stay unchanged.

2. $a_i = b$ and $a_i \neq c$. In this case, all the suffix palindromes but the longest one in the group extend with a_i and become suffix palindromes of $\mathsf{str}(r, v_i)$. We update $s \leftarrow s + 2$ and $t \leftarrow t - 1$. The value of d stays unchanged.

3. $a_i \neq b$ and $a_i = c$. In this case, only the longest suffix palindromes in the group extends with a_i and becomes a suffix palindrome of $\mathsf{str}(r, v_i)$. We first update $s \leftarrow s+(t-1)d+2$ and then $t \leftarrow 1$. The new value of d is easily calculated from the length of the longest suffix palindrome in the previous group (recall the definition of d just above Lemma 2.2).

4. $a_i \neq b$ and $a_i \neq c$. In this case, none of the members in the group extends with a_i. Then, we do nothing.

In each of the above cases, we store all these extended palindromes in v_i as the set of maximal palindromes ending at v_i in $\mathsf{str}(r, v_i)$, and exclude all these extended palindromes from the set of maximal palindromes ending at u.

See Figure 2.1 for concrete examples of the above cases. Let a_i be the next character that is appended to the string in Figure 2.1. Case 1 occurs to group G_3 when $a_i = \mathtt{c}$. Case 2 occurs to group G_1 when $a_i = \mathtt{a}$, and to group G_2 when $a_i = \mathtt{b}$. Case 3 occurs to group G_1 when $a_i = \mathtt{b}$, and to group G_2 when $a_i = \mathtt{c}$. Case 4 occurs to all the groups when $a_i = \mathtt{d}$.

Suppose that we have finished traversing the subtree rooted at u, namely, we have performed the above procedures for all characters a_i with $1 \leq i \leq |\mathsf{children}(k)|$. Then, we output, as the maximal palindromes ending at u, all suffix palindromes of u that did not extend with any a_i. Also, each time we reach a leaf in the traversal, we simply output all suffix palindromes ending at the leaf as the maximal palindromes ending at the leaf.

In each of the above four cases, we can check if the palindromes in a given group extends with a_i by at most two character comparisons. Since there are $O(\log h)$ arithmetic progressions representing the suffix palindromes ending at node u, for each child v_i of u, it takes $O(\log h)$ time to compute the suffix palindromes ending at v_i. The total cost to output the maximal palindromes is less than $2N$ (Lemma 6.1).

There is one more issue remaining. When only one or two members from a group extend with a_i, then we may need to merge these suffix palindromes into a single arithmetic progression with the suffix palindromes from the previous group. However, this can easily be done in a total of $O(\log h)$ time per node v_i, since the suffix palindromes ending at u was given as $O(\log h)$ arithmetic progressions (groups). See Figure 2.1 for a concrete example of this merging process. When $a_i = \texttt{c}$, `c` is a suffix palindrome and forms a single arithmetic progression $\langle 1, 0, 1 \rangle$. All the palindromes in G_1 are not extended. The longest suffix palindrome in group G_2 is extended to `caaabaaabaaabaaabaac` forming an arithmetic progression $\langle 21, 20, 1 \rangle$, where $20 = |\texttt{caaabaaabaaabaaabaac}| - |\texttt{c}|$, but all the other suffix palindromes in group G_2 are not extended. Finally all the suffix palindromes in group G_3 are extended and are represented by an arithmetic progression $\langle 41, 20, 2 \rangle$. Since the three suffix palindromes of lengths 21, 41, and 61 share the common difference 20, the two arithmetic progressions are merged into a single arithmetic progression $\langle 21, 20, 3 \rangle$.

We have shown the following:

Theorem 6.1. *We can compute all maximal palindromes in a given trie $\mathcal{T}$ in $O(N \log h)$ time and $O(h)$ working space, where N and h respectively denote the number of edges in $\mathcal{T}$ and the height of $\mathcal{T}$.*

Remark 6.1. *For a balanced trie with $h = \Theta(\log_\sigma N)$, our algorithm runs in $O(N \log \log_\sigma N)$ time with $O(\log_\sigma N)$ working space. In the worst case where $h = \Theta(N)$, our algorithm still runs in $O(N \log N)$ time with $O(N)$ space.*

6.2.2 Alternative Algorithm Based on Manacher's Algorithm

In this subsection, we present an alternative algorithm for computing all maximal palindromes in a given trie $\mathcal{T}$ that is based on Manacher's algorithm [86] that is originally designed for computing maximal palindromes in a single string.

For ease of explanation, we consider the path-contracted trie $\mathcal{T}'$ that can be obtained by contracting every unary path of the original trie $\mathcal{T}$ into a single edge that is labeled by a non-empty string. Let r denote the root of $\mathcal{T}'$. Throughout this subsection, for any node u in $\mathcal{T}'$, $\mathsf{parent}(u)$ and $\mathsf{children}(u)$ respectively denote the parent of u and the set of children of u in the path-contracted trie $\mathcal{T}'$.

The basic strategy of our alternative algorithm is as follows. We perform a depth first traversal on $\mathcal{T}'$, where only the root, branching internal nodes, and leaves are explicitly visited. Let u be any branching node visited in the traversal. As was done in the algorithm of subsection 6.2.1, for each branching node v in the path from the root r to u, we maintain the arithmetic progressions representing the suffix palindromes ending at v, which will be used when the traversal traces back to these branching nodes.

Now we are processing node u to extend the suffix palindromes. For this sake, we use the idea of Manacher's algorithm [86]. Let Σ_u be the set of the first characters of the out-edges of u in $\mathcal{T}'$. For each $a \in \Sigma_u$, $e_a = (u, v_a)$ denote the out-edge of u in $\mathcal{T}'$ whose label begins with a. For each $a \in \Sigma_u$ (in any order), we search for the groups of the suffix palindromes of $\mathsf{str}(r, u)$ that are immediately preceded by a, since these will be the only groups that will extend with the edge e_a. Let $\mathbf{P}_a$ be the set of suffix palindromes extended with a (which are represented by $O(\log h)$ arithmetic progressions). For each $1 \leq i \leq |\mathbf{P}_a|$, let P_i denote the ith longest suffix palindrome in $\mathbf{P}_a$. While we move forward on the edge e_a, we keep two invariants ℓ and f such that P_ℓ denotes the longest suffix palindrome whose extension ends with the currently processed character on e_a, and P_f denotes the suffix palindrome whose extension is to be determined by symmetry of P_ℓ. We process the suffix palindromes in $\mathbf{P}_a$ in decreasing order of their lengths, by picking up their lengths from the arithmetic progressions. Namely, we initially set $\ell \leftarrow 1$ and $f \leftarrow 2$ and increase the values of ℓ and f accordingly while reading the characters on the edge e_a. In any following step $\ell \leq f$ will hold.

When $\ell = 1$, as a initial step, we extend the left arm of P_ℓ on the reversed path and the right arm of P_ℓ on the path from u to v_a with naïve character comparisons. Now suppose we are processing P_ℓ. Let $s = |P_\ell|$, c be the center of P_ℓ in the path string from the root, and τ be the length of the extension of P_ℓ, namely, P_ℓ has been extended to a maximal palindrome of length

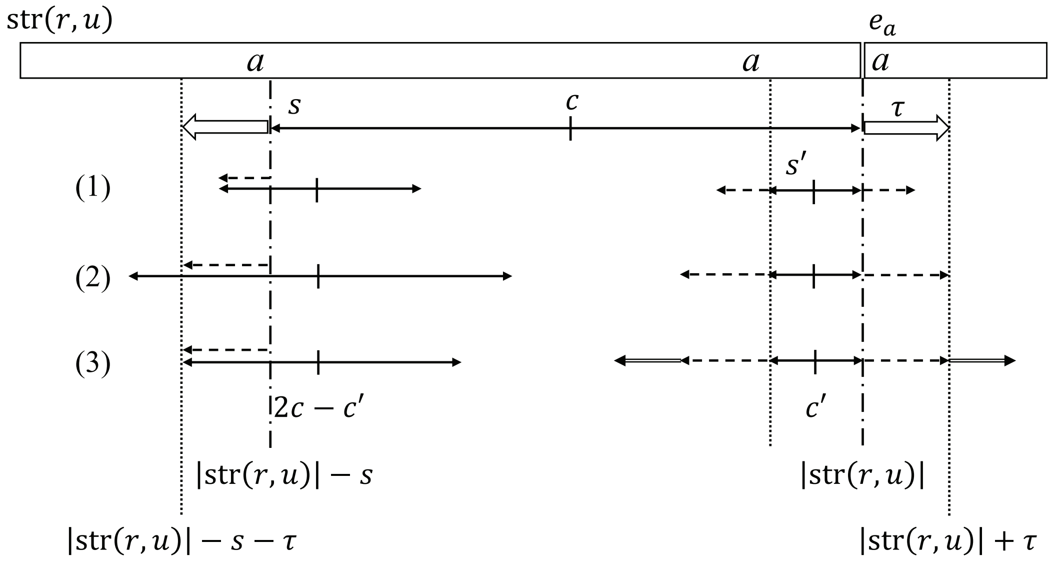


Figure 6.2: Illustration for our alternative algorithm that computes maximal palindromes in a given trie, that is based on Manacher's algorithm.

$s + 2\tau$ for center c. This means that the maximal palindromes for any centers less than c in the path from the root to u have already been computed. Then we process P_f. Let $s' = |P_f|$ and c' be the center for P_f. There are three possible cases:

(1) The depth of the left-end of the maximal palindrome for center $2c - c'$ in the path from the root is lager than $|\mathsf{str}(r, u)| - s - \tau$.

(2) The depth of the left-end of the maximal palindrome for center $2c - c'$ in the path from the root is less than $|\mathsf{str}(r, u)| - s - \tau$.

(3) The depth of the left-end of the maximal palindrome for center $2c - c'$ is equal to $|\mathsf{str}(r, u)| - s - \tau$.

See Figure 6.2 for illustration of the above three cases.

In Case(1), by symmetry P_f is extended exactly to the same length as the maximal palindrome for center $2c - c'$. We keep $\ell = 1$ and update $f \leftarrow f + 1$. In Case (2), P_f is extended exactly to length $s' + 2\tau$, because of the mismatching characters $\mathsf{str}(r, u)[|\mathsf{str}(r, u)| - s - \tau]$ and $\mathsf{str}(u, v_a)[\tau + 1]$. We keep $\ell = 1$ and update $f \leftarrow f + 1$. In Case (3), P_f is extended at least to length $s' + 2\tau$. Now we update $\ell \leftarrow f$ and then $f \leftarrow f + 1$. To check if this palindrome is further extended, we perform naïve character comparisons until we find the final value of the extension.

We perform the above procedure until we read all characters on the edge e_a, or we finish extending all palindromes from $\mathbf{P}_a$. This gives us the maximal palindromes whose centers are in the path spelling out $\mathsf{str}(r, u)$. Then we store all these extended maximal palindromes at v_a as $O(\log h)$ arithmetic progressions, and exclude all these maximal palindromes from the set of maximal palindromes ending at u. This ensures that, as in the previous subsection, the number of maximal palindromes stored at the nodes in the current path string is bounded by the height h of the original trie. Note that all maximal palindromes whose centers are on e_a need to be additionally computed. This can be done in linear time in the length of the label of e_a, by running Manacher's algorithm on this edge label.

Suppose that we have performed the above procedures for all out-edges of u in $\mathcal{T}'$. Then, we output, as the maximal palindromes ending at u, all suffix palindromes of u that did not extend with any out-edges. Also, each time we reach a leaf in the traversal, we simply output all suffix palindromes ending at the leaf as the maximal palindromes ending at the leaf.

Let us analyze the complexities of this method. Consider each branching node u in $\mathcal{T}'$. For each $a \in \Sigma_a$, we can find the arithmetic progressions representing $\mathbf{P}_a$ in $O(\log h)$ time as in the previous subsection. Each character in edge e_a is involved in exactly one character comparison. To perform each character comparison on the trie in $O(1)$ time, we preprocess the original trie $\mathcal{T}$ with N edges in $O(N)$ time and space so that *level ancestor queries* on the trie can be answered in $O(1)$ time each [16]. Hence, if N' is the number of edges in the path-contracted trie $\mathcal{T}'$, then our algorithm of this subsection runs in $O(N' \log h + N)$ time and $O(N)$ space.

Theorem 6.2. *We can compute all maximal palindromes in a given trie $\mathcal{T}$ in $O(N' \log h + N)$ time and $O(N)$ working space, where N and h respectively denote the number of edges in $\mathcal{T}$ and the height of $\mathcal{T}$, and N' denotes the number of edges in the path-contracted trie $\mathcal{T}'$.*

Remark 6.2. *Note that $N' \leq N$ always holds, and therefore the algorithm of Theorem 6.2 is at least as fast as the algorithm of Theorem 6.1. Moreover, in case where $N' = O(N/\log h)$ (which happens when the average length of the unary paths in $\mathcal{T}$ is $\Omega(\log h)$), then the algorithm of Theorem 6.2 runs in $O(N)$ time.*

Also, we obtain the following corollary:

Corollary 6.1. *We can compute all maximal palindromes in a trie $\mathcal{T}$ in an online manner in $O(N \log N)$ time and $O(N)$ working space, where N denotes the number of edges in $\mathcal{T}$.*

6.3 Computing Distinct Palindromes in a Trie

In this section we present our algorithm that computes all distinct palindromes in a given trie.

Our algorithm is based on Groult et al.'s [56] that finds distinct palindromes in a single string. Recall the proof of Lemma 6.2 in Section 6.1. There we showed that for each node u in a trie $\mathcal{T}$, only the longest suffix palindrome of $\mathsf{str}(r, u)$ can be accounted for as a distinct palindrome, where r is the root of $\mathcal{T}$. Let N and h be the number of edges in $\mathcal{T}$ and the height of $\mathcal{T}$. In this section, we assume that the root has a single out-edge labeled with a special character $\$$ that does not appear elsewhere in the trie and is lexicographically the smallest.

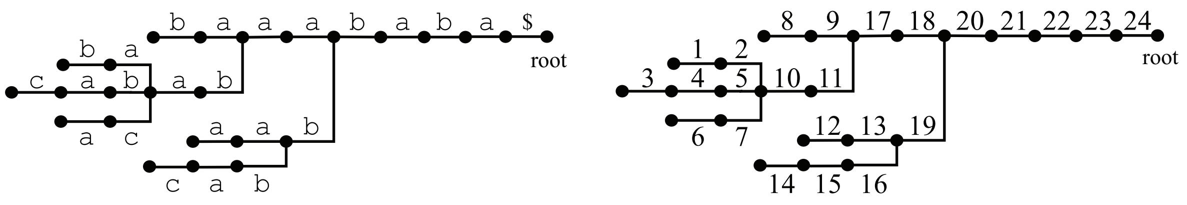
Lemma 6.3. *For each node u in a given trie $\mathcal{T}$, we can compute the longest suffix palindrome of* $\mathsf{str}(r, u)$ *in a total of $O(N' \log h + N)$ time with $O(N)$ working space, where N' denotes the number of edges in the path-contracted trie $\mathcal{T}'$.*

Proof. Clear from our algorithm to compute maximal palindromes in $\mathcal{T}$ which was presented in Section 6.2. □

Now, we consider the *reversed* trie $\mathcal{T}^R$. For any reversed path from u to u' in $\mathcal{T}^R$ in the leaf-to-root direction, let $(u, u') = \mathsf{str}(u', u)^R$. Observe that a suffix of $\mathsf{str}(r, u)$ is a prefix of $\mathsf{rev\_str}(u, r)$. Therefore, a suffix palindrome of $\mathsf{str}(r, u)$ that ends at node u in $\mathcal{T}$ is a prefix palindrome of $\mathsf{rev\_str}(u, r)$ that begins at node u in the reversed trie $\mathcal{T}^R$. For each $1 \leq j \leq N$, let e_j denote the $(N - j + 1)$th visited edge in a breadth-first traversal on the original trie $\mathcal{T}$. The *id* of edge e_j is j. See Figure 6.3 for examples of a reversed trie and the associated integers to its edges.

For each edge id j, let $e_j = (v_j, u_j)$ be the corresponding reversed edge. Let $\mathit{LPrePal}$ be an array of length N such that for each $1 \leq j \leq N$ $\mathit{LPrePal}[j]$ stores the length of the longest prefix palindrome in the reversed path string beginning with e_j (namely $\mathsf{rev\_str}(v_j, r)$). Also, let LFF be an array of length N called the *longest following factor array*, such that for each $1 \leq i \leq N$ $\mathit{LFF}[j]$ stores the length of the longest prefix of $\mathsf{rev\_str}(v_j, r)$ that occurs as a prefix of $\mathsf{rev\_str}(v_k, r)$ with $k > j$. See Figure 6.3 for examples of $\mathit{LPrePal}$ and LFF arrays.

We design an algorithm that reports a shallowest occurrence of each distinct palindrome in the (reversed) trie. If there are multiple occurrences of the same palindrome beginning at nodes on the same depth, then we report the occurrence that begins with the edge with the largest id. Now we can see that for each j, the occurrence of the longest prefix palindrome of $\mathsf{rev\_str}(v_j, r)$ should be reported iff $\mathit{LFF}[j] < \mathit{LPrePal}[j]$. Hence, we can report all distinct palindromes in the trie in $O(N)$ time by simply scanning the two arrays LFF and $\mathit{LPrePal}$ from left to



j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
$SA[j]$	24	23	9	2	17	12	21	10	18	4	13	15	6	22	8	1	11	20	5	19	16	7	3	14
$LCP[j]$	-	0	1	2	4	3	1	3	3	5	2	3	1	0	2	3	5	2	4	1	2	0	4	3
$LFF[j]$	0	0	2	4	1	3	1	3	3	5	3	2	1	0	3	5	2	2	4	1	2	3	4	0
$LPrePal[j]$	1	1	3	5	2	2	3	6	5	10	4	5	3	1	5	4	4	3	8	2	3	1	1	1

Figure 6.3: Upper left: An example of a reversed trie. Upper right: The edge id's based on a breadth-first traversal. Lower: SA, LCP, LFF and $LPrePal$ arrays built on the reversed trie shown above.

right. The LFF array can be computed in $O(N)$ time from the LCP array for the trie, by using the same technique for the longest previous factor array (LPF array) for a single string [32]. Together with Theorem 6.2, we obtain the following:

Theorem 6.3. *We can compute all distinct palindromes in a given trie $\mathcal{T}$ in $O(N' \log h + N)$ time and $O(N)$ working space, where N and h respectively denote the number of edges in $\mathcal{T}$ and the height of $\mathcal{T}$, and N' denotes the number of edges in the path-contracted trie $\mathcal{T}'$.*

Remark 6.3. *The suffix array of the reversed trie with N edges can be constructed in $O(N)$ time and space if the edge labels are drawn from a constant-size alphabet or an integer alphabet of polynomial size in N [113]. In the case of a general ordered alphabet of size σ, the suffix array of the reversed trie can be constructed in $O(N \log \sigma)$ time and space [23]. The other arrays can be constructed in $O(N)$ time after the suffix array has been built. In summary, our algorithm runs in $O(N' \log h + N \log \sigma)$ time and $O(N \log \sigma)$ working space in the case of a general ordered alphabet.*

With a little effort, we can obtain the following corollary:

Corollary 6.2. *We can compute all distinct palindromes in a trie $\mathcal{T}$ in an online manner in $O(N \log N)$ time and $O(N)$ working space, where N denotes the number of edges in $\mathcal{T}$.*

Chapter 7

Conclusions and Future Work

In this thesis, we studied regularities and algorithms on dynamic strings.

In Chapter 3, we analyzed sensitivities for string compressors and repetitiveness measures. In the seminal paper by Varma and Yoshida [118] which first introduced the notion of sensitivity for (general) algorithms and studied the sensitivity of graph algorithms, the authors wrote:

> "*Although we focus on graphs here, we note that our definition can also be extended to the study of combinatorial objects other than graphs such as strings and constraint satisfaction problems.*"

Our study was inspired by the afore-quoted suggestion, and our sensitivity for string compressors and repetitiveness measures enables one to evaluate the robustness and stability of compressors and repetitiveness measures.

The major technical contributions of this thesis are the *tight and constant upper and lower bounds* for the multiplicative sensitivity of the smallest bidirectional scheme, the LZ77 family, and the substring complexity δ. We also presented non-trivial upper and lower bounds for other string compressors and repetitive measures: the smallest string attractor, RLBWT, LZ78.

Apparent future work is to complete Tables 1.1 and 1.2 by filling the missing pieces and closing the gaps between the upper and lower bounds which are not tight there.

While we dealt with a number of string compressors and repetitiveness measures, it has to be noted that our list is far from being comprehensive: It is intriguing to analyze the sensitivity of other important and useful compressors and repetitiveness measures including the size ν of the smallest NU-systems [98], the sizes of the other locally-consistent compressed indices such as ESP-index [89] and SE-index [100], and the sizes of the global grammar compression algorithms such as Re-pair [84], Longest-Match [71], and Greedy [10].

Our notion of the sensitivity for string compressors/repetitiveness measures can naturally be extended to labeled tree compressors/repetitiveness measures. It would be interesting to analyze the sensitivity for the smallest tree attractor [107], the run-length XBWT [107], the tree LZ77 factorization [49], tree grammars [44, 85], and top-tree compression of trees [20].

In Chapter 4, we dealt with the problems of computing the LPS of a string after a single-character edit operation or a block-wise edit operation. We proposed an $O(\log(\min\{\sigma, \log n\}))$-time query algorithm that answers the LPS after a single-character edit operation, with $O(n)$-time and space preprocessing, where σ is the number of distinct characters appearing in the string. Furthermore, we presented an $O(\ell + \log(\min\{\sigma, \log n\}))$-time query algorithm that answers the LPS after a block-wise edit operation, with $O(n)$-time and space preprocessing, where ℓ denotes the length of the block after an edit. Our future work of this chapter includes the following:

(1) Can we efficiently compute the *longest gapped palindrome* in a string after an edit operation? We suspect that it might be possible with a fixed gap length, perhaps using combinatorial properties of gapped palindromes with a fixed gap length from [62].

(2) Can we extend our algorithm to biological palindromes with reverse complements such as those in DNA/RNA sequences? The key will be whether or not periodic properties hold for such palindromes.

(3) Amir et al. [4] proposed a fully-dynamic algorithm that can maintain a data structure of $\tilde{O}(n)$ space to report a longest square substring after a single character substitution in $n^{o(1)}$ time. The preprocessing cost for their data structure is $\tilde{O}(n)$ time. It is interesting if one can achieve a faster and/or more space-efficient algorithm for finding a longest square substring, if the edit operation is restricted to a *query* as in this thesis.

In Chapter 5, we dealt with the problem of updating the set of MUPSs of a string after a single-character substitution. We showed that the number d of changes of MUPSs after a single-character substitution is $O(\log n)$. Also, we showed tight lower bounds $d \in \Omega(\log n)$. Furthermore, we presented an algorithm that uses $O(n)$ time and space for preprocessing, and updates the set of MUPSs in $O(\log \sigma + (\log\log n)^2 + d)$ time where σ is the alphabet size. We also proposed a variant of the algorithm, which runs in optimal $O(1+d)$ time when the alphabet size is constant. Our future work of this chapter includes the following:

(1) Can our algorithm be adapted to the cases of insertions and deletions?

(2) Can we extend our algorithm to a fully dynamic setting? It is interesting whether the techniques in [3, 6] can be utilized in the dynamic version of the MUPS problem.

In Chapter 6, we dealt with the problem of computing maximal/distinct palindromes in a trie. In Section 6.1, we showed that the number of maximal palindromes in a trie $\mathcal{T}$ with N edges and L leaves is exactly $2N - L$ and that the number of distinct palindromes in $\mathcal{T}$ is at most $N + 1$. These generalize the known bounds for a single string [33, 86]. In Section 6.2, we presented two algorithms to compute all maximal palindromes both of which run in $O(N \log h)$ time and $O(N)$ space in the worst case, where h is the height of the trie $\mathcal{T}$. In Section 6.3, we presented the algorithm to compute all distinct palindromes in a given trie $\mathcal{T}$ in $O(N \log h)$ time with $O(N)$ space. Our future work of this chapter includes the following.

(1) Can we compute maximal/distinct palindromes in a trie in optimal $O(N)$ time?

(2) Can we efficiently compute maximal palindromes in an unrooted tree?

Bibliography

[1] P. Abedin, S. Hooshmand, A. Ganguly, and S. V. Thankachan. The heaviest induced ancestors problem revisited. In *Annual Symposium on Combinatorial Pattern Matching, CPM 2018*, volume 105 of *LIPIcs*, pages 20:1–20:13, 2018.

[2] T. Akagi, M. Funakoshi, and S. Inenaga. Sensitivity of string compressors and repetitiveness measures. *CoRR*, abs/2107.08615, 2021.

[3] A. Amir and I. Boneh. Dynamic palindrome detection. *CoRR*, abs/1906.09732, 2019.

[4] A. Amir, I. Boneh, P. Charalampopoulos, and E. Kondratovsky. Repetition detection in a dynamic string. In *27th Annual European Symposium on Algorithms, ESA 2019*, volume 144 of *LIPIcs*, pages 5:1–5:18, 2019.

[5] A. Amir, P. Charalampopoulos, C. S. Iliopoulos, S. P. Pissis, and J. Radoszewski. Longest common factor after one edit operation. In *24th International Symposium on String Processing and Information Retrieval, SPIRE 2017*, volume 10508 of *Lecture Notes in Computer Science*, pages 14–26, 2017.

[6] A. Amir, P. Charalampopoulos, S. P. Pissis, and J. Radoszewski. Longest common substring made fully dynamic. In *27th Annual European Symposium on Algorithms, ESA 2019*, volume 144 of *LIPIcs*, pages 6:1–6:17, 2019.

[7] A. Amir, P. Charalampopoulos, S. P. Pissis, and J. Radoszewski. Dynamic and internal longest common substring. *Algorithmica*, 82(12):3707–3743, 2020.

[8] A. Amir and G. Navarro. Parameterized matching on non-linear structures. *Inf. Process. Lett.*, 109(15):864–867, 2009.

[9] A. Apostolico, D. Breslauer, and Z. Galil. Parallel detection of all palindromes in a string. *Theor. Comput. Sci.*, 141(1&2):163–173, 1995.

[10] A. Apostolico and S. Lonardi. Off-line compression by greedy textual substitution. *Proceedings of the IEEE*, 88(11):1733–1744, 2000.

[11] H. Bannai, T. Gagie, and T. I. Refining the *r*-index. *Theor. Comput. Sci.*, 812:96–108, 2020.

[12] H. Bannai, S. Inenaga, and D. Köppl. Computing all distinct squares in linear time for integer alphabets. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, volume 78 of *LIPIcs*, pages 22:1–22:18, 2017.

[13] D. Belazzougui, M. Cáceres, T. Gagie, P. Gawrychowski, J. Kärkkäinen, G. Navarro, A. O. Pereira, S. J. Puglisi, and Y. Tabei. Block trees. *J. Comput. Syst. Sci.*, 117:1–22, 2021.

[14] D. Belazzougui, T. Gagie, P. Gawrychowski, J. Kärkkäinen, A. O. Pereira, S. J. Puglisi, and Y. Tabei. Queries on LZ-bounded encodings. In *2015 Data Compression Conference, DCC 2015*, pages 83–92, 2015.

[15] M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *4th Latin American Theoretical Informatics Symposium, LATIN 2000*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94, 2000.

[16] M. A. Bender and M. Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004.

[17] P. Berenbrink, F. Ergün, F. Mallmann-Trenn, and E. S. Azer. Palindrome recognition in the streaming model. In *31st International Symposium on Theoretical Aspects of Computer Science, STACS 2014*, volume 25 of *LIPIcs*, pages 149–161, 2014.

[18] P. Bille, M. B. Ettienne, I. L. Gørtz, and H. W. Vildhøj. Time-space trade-offs for Lempel-Ziv compressed indexing. *Theor. Comput. Sci.*, 713:66–77, 2018.

[19] P. Bille, P. Gawrychowski, I. L. Gørtz, G. M. Landau, and O. Weimann. Longest common extensions in trees. *Theor. Comput. Sci.*, 638:98–107, 2016.

[20] P. Bille, P. Gawrychowski, I. L. Gørtz, G. M. Landau, and O. Weimann. Top tree compression of tries. In *30th International Symposium on Algorithms and Computation, ISAAC 2019*, volume 149 of *LIPIcs*, pages 4:1–4:18, 2019.

[21] P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. R. Satti, and O. Weimann. Random access to grammar-compressed strings and trees. *SIAM J. Comput.*, 44(3):513–539, 2015.

[22] A. Blumer, J. Blumer, D. Haussler, R. M. McConnell, and A. Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *J. ACM*, 34(3):578–595, 1987.

[23] D. Breslauer. The suffix tree of a tree and minimizing sequential transducers. *Theor. Comput. Sci.*, 191(1-2):131–144, 1998.

[24] S. Brlek, N. Lafrenière, and X. Provençal. Palindromic complexity of trees. In *19th International Conference on Developments in Language Theory, DLT 2015*, volume 9168 of *Lecture Notes in Computer Science*, pages 155–166, 2015.

[25] M. Bucci, A. D. Luca, A. Glen, and L. Q. Zamboni. A new characteristic property of rich words. *Theor. Comput. Sci.*, 410(30-32):2860–2863, 2009.

[26] P. Charalampopoulos, P. Gawrychowski, and K. Pokorski. Dynamic longest common substring in polylogarithmic time. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020*, volume 168 of *LIPIcs*, pages 27:1–27:19, 2020.

[27] P. Charalampopoulos, T. Kociumaka, M. Mohamed, J. Radoszewski, W. Rytter, and T. Walen. Internal dictionary matching. *Algorithmica*, 83(7):2142–2169, 2021.

[28] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005.

[29] A. R. Christiansen, M. B. Ettienne, T. Kociumaka, G. Navarro, and N. Prezza. Optimal-time dictionary-compressed indexes. *ACM Trans. Algorithms*, 17(1):8:1–8:39, 2021.

[30] R. Cole, T. Kopelowitz, and M. Lewenstein. Suffix trays and suffix trists: Structures for faster text indexing. *Algorithmica*, 72(2):450–466, 2015.

[31] M. Crochemore. Linear searching for a square in a word. *Bull. EATCS*, 24:66–72, 1984.

[32] M. Crochemore and L. Ilie. Computing longest previous factor in linear time and applications. *Inf. Process. Lett.*, 106(2):75–80, 2008.

[33] X. Droubay, J. Justin, and G. Pirillo. Episturmian words and some constructions of de Luca and Rauzy. *Theor. Comput. Sci.*, 255(1-2):539–553, 2001.

[34] M. Farach-Colton, P. Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *J. ACM*, 47(6):987–1011, 2000.

[35] H. Ferrada and G. Navarro. Lempel-Ziv compressed structures for document retrieval. *Inf. Comput.*, 265:1–25, 2019.

[36] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1):4:1–4:33, 2009.

[37] G. Fici and P. Gawrychowski. Minimal absent words in rooted and unrooted trees. In *26th International Symposium on String Processing and Information Retrieval, SPIRE 2019*, volume 11811 of *Lecture Notes in Computer Science*, pages 152–161, 2019.

[38] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984.

[39] N. Fujisato, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda. The parameterized position heap of a trie. In *11th International Conference on Algorithms and Complexity, CIAC 2019*, volume 11485 of *Lecture Notes in Computer Science*, pages 237–248, 2019.

[40] M. Funakoshi and T. Mieno. Minimal unique palindromic substrings after single-character substitution. In *28th International Symposium on String Processing and Information Retrieval, SPIRE 2021*, volume 12944 of *Lecture Notes in Computer Science*, pages 33–46, 2021.

[41] M. Funakoshi, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda. Computing maximal palindromes and distinct palindromes in a trie. In *Proceedings of the Prague Stringology Conference, PSC 2019*, pages 3–15, 2019.

[42] M. Funakoshi, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda. Computing longest palindromic substring after single-character or block-wise edits. *Theor. Comput. Sci.*, 859:116–133, 2021.

[43] T. Gagie, G. Navarro, and N. Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM*, 67(1):2:1–2:54, 2020.

[44] M. Ganardi, D. Hucke, M. Lohrey, and E. Noeth. Tree compression using string grammars. *Algorithmica*, 80(3):885–917, 2018.

[45] L. Gasieniec, M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for lempel-zip encoding. In *5th Scandinavian Workshop on Algorithm Theory, SWAT 1996*, volume 1097 of *Lecture Notes in Computer Science*, pages 392–403, 1996.

[46] L. Gasieniec and W. Rytter. Almost optimal fully LZW-compressed pattern matching. In *Data Compression Conference, DCC 1999*, pages 316–325, 1999.

[47] P. Gawrychowski. Tying up the loose ends in fully LZW-compressed pattern matching. In *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012*, volume 14 of *LIPIcs*, pages 624–635, 2012.

[48] P. Gawrychowski, T. I, S. Inenaga, D. Köppl, and F. Manea. Tighter bounds and optimal algorithms for all maximal α-gapped repeats and palindromes - finding all maximal α-gapped repeats and palindromes in optimal worst case time on integer alphabets. *Theory Comput. Syst.*, 62(1):162–191, 2018.

[49] P. Gawrychowski and A. Jez. LZ77 factorisation of trees. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, volume 65 of *LIPIcs*, pages 35:1–35:15, 2016.

[50] P. Gawrychowski, T. Kociumaka, W. Rytter, and T. Walen. Tight bound for the number of distinct palindromes in a tree. In *22nd International Symposium on String Processing and Information Retrieval, SPIRE 2015*, volume 9309 of *Lecture Notes in Computer Science*, pages 270–276, 2015.

[51] P. Gawrychowski, T. Kociumaka, W. Rytter, and T. Walen. Tight bound for the number of distinct palindromes in a tree. *CoRR*, abs/2008.13209, 2020.

[52] P. Gawrychowski, G. M. Landau, S. Mozes, and O. Weimann. The nearest colored node in a tree. *Theor. Comput. Sci.*, 710:66–73, 2018.

[53] P. Gawrychowski, O. Merkurev, A. M. Shur, and P. Uznanski. Tight tradeoffs for real-time approximation of longest palindromes in streams. *Algorithmica*, 81(9):3630–3654, 2019.

[54] S. Giuliani, S. Inenaga, Z. 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*, volume 12607 of *Lecture Notes in Computer Science*, pages 249–262, 2021.

[55] A. Glen, J. Justin, S. Widmer, and L. Q. Zamboni. Palindromic richness. *Eur. J. Comb.*, 30(2):510–531, 2009.

[56] R. Groult, É. Prieur, and G. Richomme. Counting distinct palindromes in a word in linear time. *Inf. Process. Lett.*, 110(20):908–912, 2010.

[57] D. Gusfield. Algorithms on stings, trees, and sequences: Computer science and computational biology. *SIGACT News*, 28(4):41–60, 1997.

[58] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

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

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

[61] C. Hoobin, S. J. Puglisi, and J. Zobel. Relative Lempel-Ziv factorization for efficient storage and retrieval of web collections. *Proc. VLDB Endow.*, 5(3):265–273, 2011.

[62] T. I, W. Matsubara, K. Shimohira, S. Inenaga, H. Bannai, M. Takeda, K. Narisawa, and A. Shinohara. Detecting regularities on grammar-compressed strings. *Inf. Comput.*, 240:74–89, 2015.

[63] S. Inenaga. Suffix trees, DAWGs and CDAWGs for forward and backward tries. In *14th Latin American Theoretical Informatics Symposium, LATIN 2020*, volume 12118 of *Lecture Notes in Computer Science*, pages 194–206, 2020.

[64] S. Inenaga, H. Hoshino, A. Shinohara, M. Takeda, and S. Arikawa. Construction of the CDAWG for a trie. In *Proceedings of the Prague Stringology Conference, PSC 2001*, pages 37–48, 2001.

[65] H. Inoue, Y. Nakashima, T. Mieno, S. Inenaga, H. Bannai, and M. Takeda. Algorithms and combinatorial properties on shortest unique palindromic substrings. *J. Discrete Algorithms*, 52-53:122–132, 2018.

[66] A. Jez. A really simple approximation of smallest grammar. *Theor. Comput. Sci.*, 616:141–150, 2016.

[67] D. Kempa and T. Kociumaka. Resolution of the Burrows-Wheeler transform conjecture. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 1002–1013, 2020.

[68] D. Kempa, A. Policriti, N. Prezza, and E. Rotenberg. String attractors: Verification and optimization. In *26th Annual European Symposium on Algorithms, ESA 2018*, volume 112 of *LIPIcs*, pages 52:1–52:13, 2018.

[69] D. Kempa and N. Prezza. At the roots of dictionary compression: string attractors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 827–840, 2018.

[70] T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, and S. Arikawa. Multiple pattern matching in LZW compressed text. In *Data Compression Conference, DCC 1998*, pages 103–112, 1998.

[71] J. C. Kieffer and E. Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Trans. Inf. Theory*, 46(3):737–754, 2000.

[72] D. Kimura and H. Kashima. Fast computation of subpath kernel for trees. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, 2012.

[73] T. Kociumaka, M. Kubica, J. Radoszewski, W. Rytter, and T. Walen. A linear-time algorithm for seeds computation. *ACM Trans. Algorithms*, 16(2):27:1–27:23, 2020.

[74] T. Kociumaka, G. Navarro, and N. Prezza. Towards a definitive measure of repetitiveness. In *14th Latin American Theoretical Informatics Symposium, LATIN 2020*, volume 12118 of *Lecture Notes in Computer Science*, pages 207–219, 2020.

[75] R. Kolpakov and G. Kucherov. Searching for gapped palindromes. *Theor. Comput. Sci.*, 410(51):5365–5373, 2009.

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

[77] S. R. Kosaraju. Efficient tree pattern matching (preliminary version). In *30th Annual Symposium on Foundations of Computer Science, FOCS 1989*, pages 178–183, 1989.

[78] D. Kosolobov, M. Rubinchik, and A. M. Shur. Finding distinct subpalindromes online. In *Proceedings of the Prague Stringology Conference, PSC 2013*, pages 63–69, 2013.

[79] S. Kreft and G. Navarro. On compressing and indexing repetitive sequences. *Theor. Comput. Sci.*, 483:115–133, 2013.

[80] E. Kuramoto, O. Yano, Y. Kimura, M. Baba, T. Makino, S. Y, T. Yamamoto, T. Kataoka, and T. Tokunaga. Oligonucleotide sequences required for natural killer cell activation. *Japanese Journal of Cancer Research*, 83(11):1128–1131, 1992.

[81] S. Kuruppu, S. J. Puglisi, and J. Zobel. Relative Lempel-Ziv compression of genomes for large-scale storage and retrieval. In *17th International Symposium on String Processing and Information Retrieval, SPIRE 2010*, volume 6393 of *Lecture Notes in Computer Science*, pages 201–206, 2010.

[82] K. Kutsukake, T. Matsumoto, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda. On repetitiveness measures of Thue-Morse words. In *27th International Symposium on String Processing and Information Retrieval, SPIRE 2020*, volume 12303 of *Lecture Notes in Computer Science*, pages 213–220, 2020.

[83] G. Lagarde and S. Perifel. Lempel-Ziv: a "one-bit catastrophe" but not a tragedy. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1478–1495, 2018.

[84] N. J. Larsson and A. Moffat. Offline dictionary-based compression. In *Data Compression Conference, DCC 1999*, pages 296–305, 1999.

[85] M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using repair. *Inf. Syst.*, 38(8):1150–1167, 2013.

[86] G. Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *J. ACM*, 22:346–351, 1975.

[87] U. Manber and E. W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993.

[88] S. Mantaci, A. Restivo, G. Romana, G. Rosone, and M. Sciortino. A combinatorial view on string attractors. *Theor. Comput. Sci.*, 850:236–248, 2021.

[89] S. Maruyama, M. Nakahara, N. Kishiue, and H. Sakamoto. ESP-index: A compressed index based on edit-sensitive parsing. *J. Discrete Algorithms*, 18:100–112, 2013.

[90] W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura, and K. Hashimoto. Efficient algorithms to compute compressed longest common substrings and compressed palindromes. *Theor. Comput. Sci.*, 410(8-10):900–913, 2009.

[91] T. Mieno, K. Watanabe, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda. Palindromic trees for a sliding window and its applications. *Inf. Process. Lett.*, 173:106174, 2022.

[92] S. Mitsuya, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda. Compressed communication complexity of hamming distance. *Algorithms*, 14(4):116, 2021.

[93] M. Mohri, P. J. Moreno, and E. Weinstein. General suffix automaton construction algorithm and space bounds. *Theor. Comput. Sci.*, 410(37):3553–3562, 2009.

[94] T. Nakamura, S. Inenaga, H. Bannai, and M. Takeda. Order preserving pattern matching on trees and dags. In *24th International Symposium on String Processing and Information Retrieval, SPIRE 2017*, volume 10508 of *Lecture Notes in Computer Science*, pages 271–277, 2017.

[95] Y. Nakashima, T. I, S. Inenaga, H. Bannai, and M. Takeda. Constructing LZ78 tries and position heaps in linear time for large alphabets. *Inf. Process. Lett.*, 115(9):655–659, 2015.

[96] G. Navarro. Indexing text using the Ziv-Lempel trie. *J. Discrete Algorithms*, 2(1):87–114, 2004.

[97] G. Navarro. Document listing on repetitive collections with guaranteed performance. *Theor. Comput. Sci.*, 772:58–72, 2019.

[98] G. Navarro and C. Urbina. On stricter reachable repetitiveness measures. In *28th International Symposium on String Processing and Information Retrieval, SPIRE 2021*, volume 12944 of *Lecture Notes in Computer Science*, pages 193–206, 2021.

[99] G. Nelson, J. C. Kieffer, and P. C. Cosman. An interesting hierarchical lossless data compression algorithm, 1995. Invited Presentation.

[100] T. Nishimoto, T. I, S. Inenaga, H. Bannai, and M. Takeda. Dynamic index and LZ factorization in compressed space. *Discret. Appl. Math.*, 274:116–129, 2020.

[101] T. Nishimoto and Y. Tabei. Optimal-time queries on BWT-runs compressed indexes. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 101:1–101:15, 2021.

[102] T. Nishimoto and Y. Tabei. R-enum: Enumeration of characteristic substrings in BWT-runs bounded space. In *32nd Annual Symposium on Combinatorial Pattern Matching, CPM 2021*, volume 191 of *LIPIcs*, pages 21:1–21:21, 2021.

[103] D. S. N. Nunes, F. A. Louza, S. Gog, M. Ayala-Rincón, and G. Navarro. A grammar compression algorithm based on induced suffix sorting. In *Data Compression Conference, DCC 2018*, pages 42–51, 2018.

[104] D. S. N. Nunes, F. A. Louza, S. Gog, M. Ayala-Rincón, and G. Navarro. Grammar compression by induced suffix sorting. *CoRR*, abs/2011.12898, 2020.

[105] A. H. L. Porto and V. C. Barbosa. Finding approximate palindromes in strings. *Pattern Recognit.*, 35(11):2581–2591, 2002.

[106] N. Prezza. Optimal rank and select queries on dictionary-compressed text. In *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019*, volume 128 of *LIPIcs*, pages 4:1–4:12, 2019.

[107] N. Prezza. On locating paths in compressed tries. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 744–760, 2021.

[108] J. Radoszewski, W. Rytter, J. Straszynski, T. Walen, and W. Zuba. String covers of a tree. In *28th International Symposium on String Processing and Information Retrieval, SPIRE 2021*, volume 12944 of *Lecture Notes in Computer Science*, pages 68–82, 2021.

[109] M. Rubinchik and A. M. Shur. EERTREE: an efficient data structure for processing palindromes in strings. *Eur. J. Comb.*, 68:249–265, 2018.

[110] W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003.

[111] B. Schieber and U. Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. Comput.*, 17(6):1253–1262, 1988.

[112] J. M. Schmidt. Interval stabbing problems in small integer ranges. In *20th International Symposium on Algorithms and Computation, ISAAC 2009*, volume 5878 of *Lecture Notes in Computer Science*, pages 163–172, 2009.

[113] T. Shibuya. Constructing the suffix tree of a tree with a large alphabet. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 86-A(5):1061–1066, 2003.

[114] J. Sirén, N. Välimäki, V. Mäkinen, and G. Navarro. Run-length compressed indexes are superior for highly repetitive sequence collections. In *15th International Symposium on String Processing and Information Retrieval, SPIRE 2008*, volume 5280 of *Lecture Notes in Computer Science*, pages 164–175, 2008.

[115] J. A. Storer and T. G. Szymanski. Data compression via textual substitution. *J. ACM*, 29(4):928–951, 1982.

[116] R. Sugahara, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda. Efficiently computing runs on a trie. *Theor. Comput. Sci.*, 887:143–151, 2021.

[117] Y. Urabe, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda. Longest Lyndon substring after edit. In *Annual Symposium on Combinatorial Pattern Matching, CPM 2018*, volume 105 of *LIPIcs*, pages 19:1–19:10, 2018.

[118] N. Varma and Y. Yoshida. Average sensitivity of graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 684–703, 2021.

[119] K. Watanabe, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda. Fast algorithms for the shortest unique palindromic substring problem on run-length encoded strings. *Theory Comput. Syst.*, 64(7):1273–1291, 2020.

[120] P. Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory*, pages 1–11, 1973.

[121] S. Yamamoto, T. Yamamoto, T. Kataoka, E. Kuramoto, O. Yano, and T. Tokunaga. Unique palindromic sequences in synthetic oligonucleotides are required to induce IFN [correction of INF] and augment IFN-mediated [correction of INF] natural killer activity. *The Journal of Immunology*, 148(12):4072–4076, 1992.

[122] Y. Yoshida and S. Zhou. Sensitivity analysis of the maximum matching problem. In *12th Innovations in Theoretical Computer Science Conference, ITCS 2021*, volume 185 of *LIPIcs*, pages 58:1–58:20, 2021.

[123] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, 1977.

[124] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory*, 24(5):530–536, 1978.