Palindrome Recognition In The Streaming Model

by

Erfan Sadeqi Azer

B.Sc., Sharif University of Technology, 2011

Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Spring 2015

APPROVAL

Name:	Erfan Sadeqi Azer
Degree:	Master of Science
Title of Thesis:	Palindrome Recognition In The Streaming Model
Examining Committee:	Dr. Binay Bhattacharya, Professor, Chair
	Dr. Funda Ergun, Professor, Senior Supervisor
	Dr. Petra Berenbrink, Associate Professor, Supervisor
	Dr. Jian Pei, Professor, Internal Examiner
Date Approved:	August 28th, 2014

Partial Copyright Licence

Abstract

A palindrome is defined as a string which reads forwards the same as backwards, like, for example, the string "racecar." In the *Palindrome Problem*, one tries to find all *palindromes* in a given string. In contrast, in the case of the *Longest Palindromic Substring Problem*, the goal is to find an arbitrary one of the longest palindromes in the string.
In this paper we present three algorithms in the streaming model for the above problems, where at any point in time we are only allowed to use sublinear space. We first present a one-pass randomized algorithm that solves the *Palindrome Problem*. It has an *additive* error and uses $O(\sqrt{n})$ space. We also give two variants of the algorithm which solve related and practical problems. The second algorithm determines the exact locations of *all* the longest palindromes using two passes and $O(\sqrt{n})$ space. The third algorithm is a one-pass randomized algorithm, which solves the *Longest Palindromic Substring Problem*. It has a *multiplicative* error using only $O(\log(n))$ space. Moreover, we give a matching lower bound for any one-pass algorithm having additive error.

Keywords: Palindromes; Streaming Model; Sketching

Acknowledgments

I am using this opportunity to express my gratitude to everyone who supported me throughout the course of this M.Sc. program. First and foremost, I feel very lucky to finish my Master's under supervision of Dr. Funda Ergun. I am very thankful for her aspiring guidance, invaluably constructive criticism and friendly advice during the research leading to this thesis. I am specially grateful for her sympathetic patience through the highs and downs in recent three years of my education. I, also, express my warm thanks to Dr. Petra Berenbrink and Mr. Frederik Mallmann-Trenn for their collaboration on the research leading to this thesis. This work could have not been possible without their extensive help and insights. I would also like to thank my parent and family to be a real lifetime support and encouragement and I hope it will continue to be available up to end of my life. Finally, I recognize that this research would have not been possible without the financial funding in form of research assistantships, and scholarships provided by Simon Fraser University and especially by Dr. Funda Ergun and NSERC.

Contents

Chapter 1

Introduction

A palindrome is defined as a string which reads forwards the same as backwards, e.g., the string "racecar." Palindromes have been considered as interesting structures from very old times. They are used in poetry in different languages and cultures to make delightful phrases. Such structured words and phrases are found in ancient scriptures. So, naturally it motivated computer scientists to define and solve several problems around it. In the *Palindrome Problem* one tries to find all *palindromes* (palindromic substrings) in an input string. A related problem is the *Longest Palindromic Substring Problem* in which one tries to find any one of the longest palindromes in the input.

In this thesis, we regard the streaming version of both problems, where the input arrives over time (or, alternatively, is read as a stream) and the algorithms are allowed a space, sublinear in the size of the input. Our first contribution is a one-pass randomized algorithm that solves the *Palindrome Problem*. It has an *additive* error and uses $O(\sqrt{n})$ space. The second contribution is a two-pass algorithm which determines the exact locations of all longest palindromes. It uses the first algorithm as the first pass and uses $O(\sqrt{n})$ space. The third one is a one-pass randomized algorithm for the *Longest Palindromic Substring Problem*. It has a *multiplicative* error using $O(\log(n))$ space. We also give two variants of the first algorithm which solve other related practical problems.

Palindromes' interesting structure has applications in some applied areas, too. Recognition of palindromic substrings is important in computational biology. Palindromic structures can frequently be found in proteins and identifying them gives researchers hints about the structure of nucleic acids. For example, in *nucleic acid secondary structure prediction*, one is interested in complementary palindromes which is addressed in Chapter 2, as shown by

[45], [42], and [23]. Recognizing these structures, called "hairpin loops" in this area, helps researchers to predict the structure of nucleic acids. This problem is sometimes called *RNA folding problem*.

1.1 Streaming model

A common definition of the streaming model is as follows. Let f denote a function that we want to compute on a massively long input stream $\sigma =< a_1, a_2, ..., a_m >$, where each element a_i is drawn from alphabet set, Σ . Depending on the application, Σ can be a set of positive integers, possible Nucleobases, e.g., $\{A, C, G, T\}$ in DNAs, and $\{A, C, G, T\}$ in RNAs, or the set of all letters in a natural language. The length of the stream and alphabet size, i.e., $|\Sigma|$ are the most common parameters on which one can report space complexity of an algorithm. In this thesis, we will consider just n, because our algorithms are independent of $|\Sigma|$. Moreover, $|\Sigma|$ is not necessarily huge in some areas like bioinformatics. Then, our goal is to compute the function f on σ using a space of size s, i.e., s bits of random access working memory. Since n is going to be huge and we can not have all the input stream in our memory, s must be sublinear in terms of parameters of interest, only n in our case. Formally,

$$s = o(n).$$

Ideally,

$$s = O(\log n).$$

Beginning from late 70s, the concern of using sublinear space started and in late 90s got formulated, but usually concerned with statistical problems. Alon, Matias and Szegedy formulated the model for the problem of frequency moments [1]. Frequency moments are generalization of some statistical functions of data mostly motivated by database applications. In particular, they could answer inquires such as the number of distinct elements in a given sequence, length of it, or the repeat rate, i.e., F_2 [1]. Following this work a huge amount of research has been done for different problems through which the names "streaming algorithm" or "data stream model" are coined.

A brief list of important results in the category of statistical functions are as follows: Approximation of entropy [11],[6], L_p norm [28], [18], [29], [32], median and quantile estimations [25], [10] and histogram constructions [24], [41]. This field of research went further

than statistical properties of data and introduced some interesting tasks. Sampling from data streams are studied widely [4],[15],[7], [14], [2], in particular L_p samplers found a big interest and application in the community [38], [31].

For a long time, the function of interest, i.e., f, was unrelated to the order of the elements in input as in all the results mentioned so far. In the same category, variety of problems over geometric data such as clustering and classifications of points [26], [12], [21], estimating the diameter of a point set, bichromatic matching and minimum spanning tree on the plane [19], [20] have been studied. In terms of general graph theory, we have witnessed some important results. It is worth mentioning that there are several models one can consider for graph problems in streaming model. Most notable results in this area are: near-maximum matchings and approximate shortest path [36], [17]. In other hand, there are problems in streaming model with a function of interest depending on the order of the input elements. This thesis deals with problems in this category. In Section 1.2, we will mention the works that have been done for string problems in the streaming model. For more detailed history and techniques of streaming algorithms' literature see [9] and [40], and for graph problems see [37].

1.2 Related work

While palindromes are well-studied, to the best of our knowledge there are no results in the streaming model. Manacher [35] presents a linear time online algorithm that reports whether all symbols seen so far form a palindrome at any time . The authors of [3] show how to modify this algorithm in order to find all palindromic substrings in linear time (using a parallel algorithm). These works were considered to be optimal, at the time, in terms of time and space, because a careful amortized analysis implies that they don't take more cost than an amount linear to the input size. However, there is an intrinsic need to perform global random access to input elements, which makes it useless for the case of massive input stream. In other words, for the applications where the whole input do not fit any random access working memory, the need for the streaming model is felt.

Some of the techniques used in this paper have their origin in the streaming pattern matching literature. In the *Pattern Matching Problem*, one tries to find all occurrences of a given pattern P in a text T. The first algorithm for pattern matching in the streaming model is given in [43] and requires $O(\log(m))$ space. The authors of [16] give a simpler

pattern matching algorithm with no preprocessing, as well as a related streaming algorithm for estimating a stream's Hamming distance to p-periodicity. Breslauer and Galil [8] provide an algorithm which does not report false negatives and can also be run in real-time and is much simpler to express. All of the above algorithms in the streaming model take advantage of Karp-Rabin fingerprints [33]. We formally define this tool in Chapter 2.

1.3 Thesis Outline

In Chapter 3, a simple one-pass algorithm is presented that gives the overall idea and scenario of the algorithm in following chapter. In Chapter 4, a compression technique is presented. Using this technique, we present a streaming algorithm with additive approximation. In Chapter 5, we show how a second pass can help make a precise streaming algorithm for *Longest Palindromic Substring Problem*. In Chapter 6, we present an even better streaming algorithm for the case concerning multiplicative approximation. Finally, in Chapter 7, we give a matching lower bound for the additive case.

Chapter 2

Model and Definitions

Let $S \in \Sigma^n$ denote the input stream of length n over an alphabet Σ[1]. For simplicity we assume symbols to be positive integers, i.e., $\Sigma \subset \mathbb{N}$. We define $S[i]$ as the symbol at index i and $S[i,j] = S[i], S[i+1], \ldots S[j]$. In this paper we use the streaming model: In one *pass* the algorithm goes over the whole input stream S, reading $S[i]$ in *iteration* i of the pass. In this paper we assume that the algorithm has a memory of size $o(n)$, but the output space is unlimited. We use the so-called word model where the space equals the number of $O(\log(n))$ registers (See [8]).

S contains an *odd palindrome* of length ℓ with midpoint $m \in \{\ell, \ldots, n-\ell\}$ if $S[m-i] = S[m+i]$ for all $i \in \{1, \ldots, \ell\}$. Similarly, S contains an *even palindrome* of length ℓ if $S[m-i+1] = S[m+i]$ for all $i \in \{1, \ldots, \ell\}$. In other words, a palindrome is odd if and only if its length is odd. For simplicity, our algorithms assume palindromes to be even - it is easy to adjust our results for finding odd palindromes by apply the algorithm to $S[1]S[1]S[2]S[2]\cdots S[n]S[n]$ instead of $S[1,n]$.

The maximal palindrome (the palindrome of maximal length) in $S[1,i]$ with midpoint m is called $P[m,i]$ and the maximal palindrome in S with midpoint m is called $P[m]$ which equals $P[m,n]$. We define $\ell(m,i)$ as the maximum length of the palindrome with midpoint m in the substring $S[1,i]$. The maximal length of the palindrome in S with midpoint m is denoted by $\ell(m)$. Moreover, for $z \in \mathbb{Z} \setminus \{1, \ldots, n\}$ we define $\ell(z) = 0$. Furthermore, for $\ell^* \in \mathbb{N}$ we define $P[m]$ to be an ℓ^*-palindrome if $\ell(m) \geq \ell^*$. Throughout this paper, $\tilde{\ell}()$

[1] All soundness, space, and time complexity analyses assumes $|\Sigma|$ to be polynomial. One can use a proper random hash function for bigger alphabets.

refers to an estimate of $\ell()$.

We use the KR-Fingerprint, which was first defined by Karp and Rabin [33] to compress strings and was later used in the streaming pattern matching problem (see [43], [16], and [8]). For a string S' we define the forward fingerprint (similar to [8]) and its reverse as follows. $\phi^F_{r,p}(S') = \left(\sum_{i=1}^{|S'|} S'[i] \cdot r^i\right) \mod p$ $\phi^R_{r,p}(S') = \left(\sum_{i=1}^{|S'|} S'[i] \cdot r^{l-i+1}\right) \mod p,<$ where p is an arbitrary prime number in $[n^4, n^5]$ and r is randomly chosen from $\{1, \ldots, p\}$. We write $\phi^F(\phi^R$ respectively) as opposed to $\phi^F_{r,p}(\phi^R_{r,p}$ respectively) whenever r and p are fixed. We define for $1 \leq i \leq j \leq n$ the fingerprint $F^F(i,j)$ as the fingerprint of $S[i,j]$, i.e., $F^F(i,j) = \phi^F(S[i,j]) = r^{-(i-1)}(\phi^F(S[1,j]) - \phi^F(S[1,i-1]))$ mod p. Similarly, $F^R(i,j) = \phi^R(S[i,j]) = \phi^R(S[1,j]) - r^{j-i+1} \cdot \phi^R(S[1,i-1])$ mod p. For every $1 \leq i \leq n - \sqrt{n}$ the fingerprints $F^F(1, i-1-\sqrt{n})$ and $F^R(1, i-1-\sqrt{n})$ are called *Master Fingerprints*. Note that it is easy to obtain $F^F(i, j+1)$ by adding the term $S[j+1]r^{j+1}$ to $F^F(i,j)$. Similarly, we obtain $F^F(i+1, j)$ by subtracting $S[i]$ from $r^{-1} \cdot F^F(i,j)$. The authors of [8] observe useful properties which we state in Lemma 2.0.1 and Lemma 2.0.2.

Lemma 2.0.1 *(Similar to Lemma 1 and Corollary 1 of [8]) Consider two substrings $S[i,k]$ and $S[k+1,j]$ and their concatenated string $S[i,j]$ where $1 \leq i \leq k \leq j \leq n$.*

- $F^F(i,j) = \left(F^F(i,k) + r^{k-i+1} \cdot F^F(k+1,j)\right)$ *mod p.*
- $F^F(k+1,j) = r^{-(k-i+1)}\left(F^F(i,j) - F^F(i,k)\right)$ *mod p.*
- $F^F(i,k) = \left(F^F(i,j) - r^{k-i+1} \cdot F^F(k+1,j)\right)$ *mod p.*

The authors of [8] show that, for appropriate choices of p and r, it is very unlikely that two different strings share the same fingerprint.

Lemma 2.0.2 *(Theorem 1 of [8]) For two arbitrary strings s and s' with $s \neq s'$ the probability that $\phi^F(s) = \phi^F(s')$ is smaller than $1/n^4$.*

For many biological applications such as *nucleic acid secondary structure prediction*, one is interested in complementary palindromes which are defined in the following.

Definition 1 *Let $f : \Sigma \to \Sigma$ be a function indicating a complement for each symbol in Σ. A string $S \in \Sigma^n$ with length n contains a* complementary palindrome *of length ℓ with midpoint $m \in \{\ell, \ldots, n-\ell\}$ if $S[m-i+1] = f(S[m+i])$ for all $i \in \{1, \ldots, \ell\}$.*

The fingerprints can also be used for finding complementary palindromes: If one changes the forward *Master Fingerprints* to be $F_c^F(1,l) = \left(\sum_{i=1}^{l} f(S[i]) \cdot r^i\right) \mod p$ (as opposed to $F^F(1,l) = \left(\sum_{i=1}^{l} S[i] \cdot r^i\right) \mod p$) in all algorithms in this paper, then we obtain the following observation.

Observation 1 *All algorithms in this paper can be adjusted to recognize complementary palindromes with the same space and time complexity.*

Chapter 3

An Introductory Algorithm

In this chapter, we introduce a simple one-pass algorithm which reports all midpoints and length estimates of palindromes in S. Throughout this paper we use i to denote the current index which the algorithm reads. *Simple ApproxSqrt* keeps the last $2\sqrt{n}$ symbols of $S[1, i]$ in the memory.

3.1 Algorithm

It is easy to determine the exact length palindromes of length less than $\sqrt{n}$ since any such palindrome is fully contained in memory at some point. However, in order to achieve a better time bound the algorithm only *approximates* the length of short palindromes. It is more complicated to estimate the length of a palindrome with a length of at least $\sqrt{n}$. However, *Simple ApproxSqrt* detects that its length is at least $\sqrt{n}$ and stores it as an R_S-entry (introduced later) in a list L_i. The R_S-entry contains the midpoint as well as a length estimate of the palindrome, which is updated as i increases.

In order to estimate the lengths of the long palindromes the algorithm designates certain indices of S as *checkpoints*. For every checkpoint c the algorithm stores a fingerprint $F^R(1, c)$ enabling the algorithm to do the following. For every midpoint m of a long palindrome: Whenever the distance from a checkpoint c to m (c occurs before m) equals the distance from m to i, the algorithm compares the substring from c to m to the reverse of the substring from m to i by using fingerprints. We refer to this operation as *checking* $P[m]$ against checkpoint c. If $S[c + 1, m]^R = S[m + 1, i]$, then we say that $P[m]$ was *successfully checked*

with c and the algorithm updates the length estimate for $P[m]$, $\tilde{\ell}(m)$. The next time the algorithm possibly updates $\tilde{\ell}(m)$ is after d iterations where d equals the distance between checkpoints. This distance d gives the additive approximation. See Figure 3.1 for an illustration.

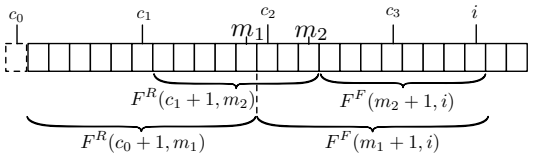

Figure 3.1: At iteration i two midpoints m_1 and m_2 are checked. Corresponding substrings are denoted by brackets. Note, the distance from c_0 to m_1 equals the distance from m_1 to i. Similarly, the distance from c_1 to m_2 equals the distance from m_2 to i.

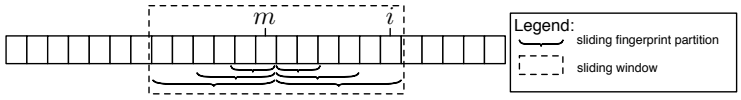

Figure 3.2: Illustration of the *Fingerprint Pairs* after iteration i of algorithm with $\sqrt{n} = 6$, $\varepsilon = 1/3$, and $m = i - \sqrt{n}$.

We need the following definitions before we state the algorithm: For $k \in \mathbb{N}$ with $0 \leq k \leq \lfloor \frac{\sqrt{n}}{\varepsilon} \rfloor$ checkpoint c_k is the index at position $k \cdot \lfloor \varepsilon\sqrt{n} \rfloor$ thus checkpoints are $\lfloor \varepsilon\sqrt{n} \rfloor$ indices apart. Whenever we say that an algorithm stores a checkpoint, this means storing the data belonging to this checkpoint. Additionally, the algorithm stores *Fingerprint Pairs*, fingerprints of size $\lfloor \varepsilon\sqrt{n} \rfloor, 2\lfloor \varepsilon\sqrt{n} \rfloor, \ldots$ starting or ending in the middle of the sliding window. In the following, we first describe the data that the algorithm has in its memory after reading $S[1, i-1]$, then we describe the algorithm itself. Let $R_S(m, i)$ denote the representation of $P[m]$ which is stored at time i. As opposed to storing $P[m]$ directly, the algorithm stores m, $\tilde{\ell}(m, i)$, $F^F(1, m)$, and $F^R(1, m)$.

Memory invariants. Just before algorithm *Simple ApproxSqrt* reads $S[i]$ it has stored the following information. Note that, for ease of referencing, during an iteration i data structures are indexed with the iteration number i.

That is, for instance, L_{i-1} is called L_i after $S[i]$ is read.

1. The contents of the sliding window $S[i - 2\sqrt{n} - 1, i - 1]$.

2. The two *Master Fingerprints* $F^F(1, i-1)$ and $F^R(1, i-1)$.

3. A list of *Fingerprint Pairs*: Let r be the maximum integer s.t. $r \cdot \lfloor \varepsilon\sqrt{n} \rfloor < \sqrt{n}$. For $j \in \{\lfloor \varepsilon\sqrt{n} \rfloor, 2 \cdot \lfloor \varepsilon\sqrt{n} \rfloor, \ldots, r \cdot \lfloor \varepsilon\sqrt{n} \rfloor, \sqrt{n}\}$ the algorithm stores the pair $F^R((i-\sqrt{n})-j, (i-\sqrt{n})-1)$, and $F^F(i-\sqrt{n}, (i-\sqrt{n})+j-1)$. See Figure 3.2 for an illustration.

4. A list CL_{i-1} which consists of all fingerprints of prefixes of S ending at already seen checkpoints, i.e., $\mathrm{CL}_{i-1} = \left[F^R(1, c_1), F^R(1, c_2), \ldots, F^R\left(1, c_{\rfloor (i-1)/\lfloor \varepsilon\sqrt{n} \rfloor \rfloor}\right)\right]$

5. A list L_{i-1} containing representation of all $\sqrt{n}$-palindromes with a midpoint located in
   $S[1, (i-1)-\sqrt{n}]$. The j^{th} entry of L_{i-1} has the form
   $R_S(m_j, i-1) = (m_j, \tilde{\ell}(m_j, i-1), F^F(1, m_j), F^R(1, m_j))$ where

   (a) m_j is the midpoint of the j^{th} palindrome in $S[1, (i-1)-\sqrt{n}]$ with a length of at least $\sqrt{n}$. Therefore, $m_j < m_{j+1}$ for $1 \le j \le |L_{i-1}| - 1$.

   (b) $\tilde{\ell}(m_j, i-1)$ is the current estimate of $\ell(m_j, i-1)$.

In the following, we explain how the algorithm maintains the above invariants.

Maintenance. At iteration i the algorithm performs the following steps. It is implicit that L_{i-1} and CL_{i-1} become L_i and CL_i respectively.

1. Read $S[i]$, set $m = i - \sqrt{n}$. Update the sliding window to $S[m-\sqrt{n}, i] = S[i-2\sqrt{n}, i]$
2. Update the *Master Fingerprints* to be $F^F(1, i)$ and $F^R(1, i)$.
3. If i is a checkpoint (i.e., a multiple of $\lfloor \varepsilon\sqrt{n} \rfloor$), then add $F^R(1, i)$ to CL_i.
4. Update all *Fingerprint Pairs*: For $j \in \{\lfloor \varepsilon\sqrt{n} \rfloor, 2 \cdot \lfloor \varepsilon\sqrt{n} \rfloor, , \ldots, r \cdot \lfloor \varepsilon\sqrt{n} \rfloor, \sqrt{n}\}$
   - Update $F^R(m-j, m-1)$ to $F^R(m-j+1, m)$ and $F^F(m, m+j-1)$ to $F^F(m+1, m+j)$.
   - If $F^R(m-j+1, m) = F^F(m+1, m+j)$, then set $\tilde{\ell}(m, i) = j$.
   - If $\tilde{\ell}(m, i) < \sqrt{n}$, output m and $\tilde{\ell}(m, i)$.
5. If $\tilde{\ell}(m, i) \ge \sqrt{n}$, *add* $R_S(m, i)$ to L_i:
   $L_i = L_i \circ (m, \tilde{\ell}(m, i), F^F(1, m), F^R(1, m))$.

6. For all c_k with $1 \leq k \leq \lfloor \frac{i}{\lfloor \varepsilon\sqrt{n} \rfloor} \rfloor$ and $R_S(m_j, i) \in L_i$ with $i - m_j = m_j - c_k$, check if $\tilde{\ell}(m_j, i)$ can be updated:

   - If the left side of m_j is the reverse of the right side of m_j (i.e., $F^R(c_k + 1, m_j) = F^F(m_j + 1, i)$) then update $R_S(m_j, i)$ by updating $\tilde{\ell}(m_j, i)$ to $i - m_j$.

7. If $i = n$, then report L_n.

3.2 Soundness

In all proofs in this paper which hold w.h.p. we assume that fingerprints do not fail as we take less than n^2 fingerprints and by using Lemma 2.0.2, the probability that a fingerprint fails is at most $1/n^{c+2}$.

Thus, by applying the union bound the probability that no fingerprint fails is at least $1-n^{-c}$. The following lemma shows that the *Simple ApproxSqrt* finds all palindromes along with the estimates as stated in Theorem 4.4.2. *Simple ApproxSqrt* does not fulfill the time and space bounds of Theorem 4.4.2; we will later show how to improve its efficiency.

Lemma 3.2.1 *For any ε in $[1/\sqrt{n}, 1]$* ApproxSqrt(S, ε) *reports for every palindrome $P[m]$ in S its midpoint m as well as an estimate $\tilde{\ell}(m)$ such that w.h.p. $\ell(m) - \varepsilon\sqrt{n} < \tilde{\ell}(m) \leq \ell(m)$.*

Proof Fix an arbitrary palindrome $P[m]$. First we assume $\ell(m) < \sqrt{n}$. Then *Simple ApproxSqrt* reports m and $\tilde{\ell}(m)$ in step 4 of iteration $m + \sqrt{n}$ which is the iteration where the entire palindrome is in the sliding window. Furthermore, in the step 4, the algorithm checks for all $j \in \{\lfloor \varepsilon\sqrt{n} \rfloor, 2 \cdot \lfloor \varepsilon\sqrt{n} \rfloor, , \ldots, r \cdot \lfloor \varepsilon\sqrt{n} \rfloor, \sqrt{n}\}$, where r is the maximum integer s.t. $r \cdot \lfloor \varepsilon\sqrt{n} \rfloor < \sqrt{n}$, if $F^R(m - j + 1, m) = F^F(m + 1, m + j)$, then sets $\tilde{\ell}(m, i) = j$. Let j_m be the maximum j such that $F^R(m - j + 1, m) = F^F(m + 1, m + j)$ is satisfied at iteration $m + \sqrt{n}$. Then $P[m]$ covers the *Fingerprint Pair* with length j_m but does not cover the *Fingerprint Pair* with length $j_m + \lfloor \varepsilon\sqrt{n} \rfloor$. Since *Simple ApproxSqrt* sets $\tilde{\ell}(m) = j_m$ we have $\ell(m) - \varepsilon\sqrt{n} < \tilde{\ell}(m) \leq \ell(m)$.
Now we assume $\ell(m) \geq \sqrt{n}$. Step 5 of iteration $m + \sqrt{n}$ adds $R_S(m, i)$ to L_{i-1}. We show for every $i \geq m + \sqrt{n}$ that the following holds: After *Simple ApproxSqrt* read $S[1, i]$ (iteration i) we have $\ell(m, i) - \varepsilon\sqrt{n} < \tilde{\ell}(m, i) \leq \ell(m, i)$. We first show the first inequality and afterwards the second. Define $i' \leq i$ to be the last iteration where the algorithm updated $\tilde{\ell}(m, i')$ in Step 6, i.e., it sets $\tilde{\ell}(m, i') = \ell(m, i') = i' - m$.

- $\ell(m,i) - \varepsilon\sqrt{n} < \tilde{\ell}(m,i)$: We first show $\ell(m,i) < i' + \varepsilon\sqrt{n} - m$ by distinguishing between two cases:

  1. $i' > i - \varepsilon\sqrt{n}$: By definition of $\ell(m,i)$, we have $\ell(m,i) \leq i - m$. And thus $\ell(m,i) < i' + \varepsilon\sqrt{n} - m$.
  2. $i' \leq i - \varepsilon\sqrt{n}$: Since the estimate of m was updated at iteration i' we know that there is a checkpoint at index $2m - i'$ and therefore we know that due to step 3 there is a checkpoint at index $2m-(i'+\lfloor\varepsilon\sqrt{n}\rfloor)$. Since i' is the last iteration where the estimate of $P[m]$ was updated we infer that $S[2m-(i'+\lfloor\varepsilon\sqrt{n}\rfloor), m]$ was not the reverse of $S[m+1, i'+\lfloor\varepsilon\sqrt{n}\rfloor]$. Hence, $\ell(m,i) < i'+\lfloor\varepsilon\sqrt{n}\rfloor - m \leq i'+\varepsilon\sqrt{n}-m$.

  With $\ell(m,i) < i' + \varepsilon\sqrt{n} - m$ we have

  $$\ell(m,i) < i' + \varepsilon\sqrt{n} - m = \ell(m,i') + \varepsilon\sqrt{n} = \tilde{\ell}(m,i') + \varepsilon\sqrt{n} = \tilde{\ell}(m,i) + \varepsilon\sqrt{n}.$$

  The last equality holds since i' was the last index where $\tilde{\ell}(m,i)$ was updated.

- $\tilde{\ell}(m,i) \leq \ell(m,i)$: Whenever $\tilde{\ell}(m,i)$ is updated to $\ell(m,i')$ by the algorithm, this means that $F^F(m+1,i') = F^R(2m-i',m)$ and since we assume that fingerprints do not fail we have that $S[m - \tilde{\ell}(m,i') + 1, m]$ is the reverse of $S[m+1, m+\tilde{\ell}(m,i')]$. It follows that $\tilde{\ell}(m,i) = \tilde{\ell}(m,i')$ and $\ell(m,i) \geq \ell(m,i')$.

Furthermore, step 7 reports L_n at iteration n which includes m and $\tilde{\ell}(m)$. ∎

Chapter 4

The Additive Approximation Algorithm

In this chapter, we show how to modify *Simple ApproxSqrt* so that it matches the time and space requirements of Theorem 4.4.2. The main idea of the space improvement is to store the lists L_i in a compressed form.

4.1 Compression

It is possible in the simple algorithm for L_i to have linear length. In such cases S contains many overlapping palindromes which show a certain *periodic* pattern as shown in Corollary 6.2.2, which our algorithm exploits to compress the entries of L_i. This idea was first introduced in [43], and is used in [16], and [8]. More specifically, our technique is a modification of the compression in [8]. In the following, we give some definitions in order to show how to compress the list. First we define a *run* which is a sequence of midpoints of overlapping palindromes.

Definition 2 (ℓ^*-Run) *Let ℓ^* be an arbitrary integer and $h \geq 3$. Let $m_1, m_2, m_3, \ldots, m_h$ be consecutive midpoints of ℓ^*-palindromes in S. $m_1, \ldots, m_h$ form an* ℓ^*-run *if $m_{j+1} - m_j \leq \ell^*/2$ for all $j \in \{1, \ldots, h-1\}$.*

In Corollary 6.2.2 we show that $m_2 - m_1 = m_3 - m_2 = \cdots = m_h - m_{h-1}$. We say that a run is maximal if the run cannot be extended by other palindromes. More formally:

Definition 3 (Maximal ℓ^*-Run) *An ℓ^*-run over $m_1, \ldots, m_h$ is* maximal *it satisfies both of the following: i) $\ell(m_1 - (m_2 - m_1)) < \ell^*$, ii) $\ell(m_h + (m_2 - m_1)) < \ell^*$.*

Simple ApproxSqrt stores palindromes explicitly in L_i, i.e., $L_i = [R_S(m_1, i); \ldots; R_S(m_{|L_i|}, i)]$ where $R_S(m_j, i) = (m_j, \tilde{\ell}(m_j, i), F^F(1, m_j), F^R(1, m_j))$, for all $j \in \{1, 2, \ldots, h\}$. The improved *Algorithm ApproxSqrt* stores these midpoints in a compressed way in list $\hat{L}_i$. *ApproxSqrt* distinguishes among three cases: Those palindromes which

1. are not part of a $\sqrt{n}$-run are stored explicitly as before. We call them R_S-entries. Let $P[m, i]$ be such a palindrome. After iteration i the algorithm stores $R_S(m, i)$.

2. form a maximal $\sqrt{n}$-run are stored in a data structure called R_F-entry. Let $m_1, \ldots, m_h$ be the midpoints of a maximal $\sqrt{n}$-run. The data structure stores the following information.

   - m_1, $m_2 - m_1$, h, $\tilde{\ell}(m_1, i)$, $\tilde{\ell}(m_{\lfloor \frac{1+h}{2} \rfloor}, i)$, $\tilde{\ell}(m_{\lceil \frac{1+h}{2} \rceil}, i)$, $\tilde{\ell}(m_h, i)$,
   - $F^F(1, m_1)$, $F^R(1, m_1)$, $F^F(m_1 + 1, m_2)$, $F^R(m_1 + 1, m_2)$

3. form a $\sqrt{n}$-run which is not maximal (i.e., it can possibly be extended) in a data structure called R_{NF}-entry. The information stored in an R_{NF}-entry is the same as in an R_F-entry, but it does not contain the entries: $\tilde{\ell}(m_{\lfloor \frac{1+h}{2} \rfloor}, i)$, $\tilde{\ell}(m_{\lceil \frac{1+h}{2} \rceil}, i)$, and $\tilde{\ell}(m_h, i)$.

The algorithm stores only the estimate (of the length) and the midpoint of the following palindromes explicitly.

- $P[m]$ for an R_S-entry (Therefore all palindromes which are not part of a $\sqrt{n}$-run)
- $P[m_1]$, $P[m_{\lfloor (h+1)/2 \rfloor}]$, $P[m_{\lceil (h+1)/2 \rceil}]$, and $P[m_h]$ for an R_F-entry
- $P[m_1]$ for an R_{NF}-entry.

In what follows we refer to the above listed palindromes as *explicitly stored* palindromes. We argue in Observation 2 that in any interval of length $\sqrt{n}$ the number of *explicitly stored* palindromes is bounded by a constant.

4.2 *Algorithm ApproxSqrt*

In this section, we describe some modifications of *Simple ApproxSqrt* in order to obtain a space complexity of $O(\frac{\sqrt{n}}{\varepsilon})$ and a total running time of $O(\frac{n}{\varepsilon})$. *ApproxSqrt* is the same as

Simple ApproxSqrt, but it compresses the stored palindromes. *ApproxSqrt* uses the same memory invariants as *Simple ApproxSqrt*, but it uses $\hat{L}_i$ as opposed to L_i.
ApproxSqrt uses the first four steps of *Simple ApproxSqrt*. Step 5, Step 6, and Step 7 are replaced. The modified Step 5 ensures that there are at most two R_S-entries per interval of length $\sqrt{n}$. Moreover, Step 6 is adjusted since *ApproxSqrt* stores only the length estimate of explicitly stored palindromes.

5. If $\tilde{\ell}(m,i) \geq \sqrt{n}$, obtain $\hat{L}_i$ by adding the palindrome with midpoint $m(= i - \sqrt{n})$ to $\hat{L}_{i-1}$ as follows:

   (a) The last element in $\hat{L}_i$ is the following R_{NF}-entry
   $\big(m_1, m_2{-}m_1, h, \tilde{\ell}(m_1,i), F^F(1,m_1), F^R(1,m_1), F^F(m_1{+}1,m_2), F^R(m_1{+}1,m_2)\big)$.

      i. If the palindrome can be added to this run, i.e., $m = m_1 + h(m_2 - m_1)$, then we increment the h in the R_{NF}-entry by 1.

      ii. If the palindrome cannot be added: Store $P[m,i]$ as an R_S-entry: $\hat{L}_i = \hat{L}_i \circ (m, \tilde{\ell}(m,i), F^F(1,m), F^R(1,m))$. Moreover, convert the R_{NF}-entry into the R_F-entry by adding $\tilde{\ell}(m_{\lfloor \frac{1+h}{2} \rfloor}, i)$, $\tilde{\ell}(m_{\lceil \frac{1+h}{2} \rceil}, i)$ and $\tilde{\ell}(m_h, i)$: First we calculate $m_{\lfloor \frac{1+h}{2} \rfloor} = m_1 + \big(\lfloor \frac{1+h}{2} - 1 \rfloor\big)(m_2 - m_1)$. One can calculate $m_{\lceil \frac{1+h}{2} \rceil}$ similarly. For $m' \in \{m_{\lfloor \frac{1+h}{2} \rfloor}, m_{\lceil \frac{1+h}{2} \rceil}, m_h\}$ calculate $\tilde{\ell}(m', i) =$
      $\max_{i-2\sqrt{n} \leq j \leq i} \{j - m' \mid \exists c_k \text{ with } j - m' = m' - c_k \text{ and } F^R(c_k{+}1, m') = F^F(m'{+}1, j)\}$.

   (b) The last two entries in $\hat{L}_i$ are stored as R_S-entries and together with $P[m,i]$ form a $\sqrt{n}$-run. Then remove the entries of the two palindromes out of $\hat{L}_{i-1}$ and add a new R_{NF}-entry with all three palindromes to $\hat{L}_{i-1}$:
   $m_1, \tilde{\ell}(m_1,i), F^F(1,m_1), F^F(1,m_2), F^R(1,m_1), F^R(1,m_2), m_2 - m_1, h = 3$. Retrieve $F^F(m_1+1, m_2)$ and $F^R(m_1+1, m_2)$.

   (c) Otherwise, store $P[m,i]$ as an R_S-entry: $\hat{L}_i = \hat{L}_i \circ (m, \tilde{\ell}(m,i), F^F(1,m), F^R(1,m))$

6. This step is similar to step 6 of *Simple ApproxSqrt* the only difference is that we check only for explicitly stored palindromes if they can be extended outwards. [1]

7. If $i = n$. If the last element in $\hat{L}_i$ is an R_{NF}-entry, then convert it into an R_F-entry as in 5(a)ii. Report L_n.

[1] This step is only important for the running time.

4.3 Structural Properties

In this section, we prove structural properties of palindromes. These properties allow us to compress (by using R_S-entries and R_F-entries) overlapping palindromes $P[m_1], \ldots, P[m_h]$ in such a way that at any iteration i all the information stored $R_S(m_1, i), \ldots, R_S(m_h, i)$ is available. The structural properties imply, informally speaking, that the palindromes are either far from each other, leading to a small number of them, or they are overlapping and it is possible to compress them. Lemma 4.3.1 shows this structure for short intervals containing at least three palindromes. Corollary 6.2.2 shows a similar structure for palindromes of a run which is used by *ApproxSqrt*. We first give the common definition of periodicity.

Definition 4 (*period*) *A string S' is said to have period p if it consists of repetitions of a block of p symbols. Formally, S' has period p if $S'[j] = S[j+p]$ for all $j = 1, \ldots, |S'| - p$.* [2]

Lemma 4.3.1 *Let $m_1 < m_2 < m_3 < \cdots < m_h$ be indices in S that are consecutive midpoints of ℓ^*-palindromes for an arbitrary natural number ℓ^*. If $m_h - m_1 \leq \ell^*$, then*

(a) $m_1, m_2, m_3, \ldots, m_h$ are equally spaced in S, i.e., $|m_2 - m_1| = |m_{k+1} - m_k|$ $\forall k \in \{1, \ldots, h-1\}$

(b) $S[m_1 + 1, m_h] = \begin{cases} (ww^R)^{\frac{h-1}{2}} & h \text{ is odd} \\ (ww^R)^{\frac{h-2}{2}} w & h \text{ is even} \end{cases}$, *where $w = S[m_1 + 1, m_2]$.*

Proof Given $m_1, m_2, \ldots, m_h$ and ℓ^* we prove the following stronger claim by induction over the midpoints $\{m_1, \ldots, m_j\}$. (a') $m_1, m_2, \ldots, m_j$ are equally spaced. (b') $S[m_1+1, m_j+\ell^*]$ is a prefix of ww^Rww^R... .

Base case $j = 2$: Since we assume m_1 is the midpoint of an ℓ^*-palindrome and $\ell^* \geq m_h - m_1 \geq m_2 - m_1 = |w|$, we have that $S[m_1 - |w| + 1, m_1] = w^R$. Recall that $\ell(m_2) \geq \ell^* \geq |w|$ and thus, $S[m_1 + 1, m_2 + |w|] = ww^R$.

We can continue this argument and derive that $S[m_1+1, m_2+\ell^*]$ is a prefix of $ww^R \ldots ww^R$. (a') for $j = 2$ holds trivially.

Inductive step $j - 1 \to j$: Assume (a') and (b') hold up to m_{j-1}. We first argue that $|m_j - m_1|$ is a multiple of $|m_2 - m_1| = |w|$. Suppose $m_j = m_1 + |w| \cdot q + r$ for some integers $q \geq 0$ and $r \in \{1, \ldots, |w| - 1\}$. Since $m_j \leq m_{j-1} + \ell^*$, the interval $[m_1 + 1, m_{j-1} + \ell^*]$

[2] *Here, p is called a period for S' even if $p > |S'|/2$*

contains m_j. Therefore, by inductive hypothesis, $m_j - r$ is an index where either w or w^R starts. This implies that the prefix of ww^R(or w^Rw) of size $2r$ is a palindrome and the string ww^R(or w^Rw) has period $2r$. On the other hand, by consecutiveness assumption, there is no midpoint of an ℓ^*-palindrome in the interval $[m_1 + 1, m_2 - 1]$. does not have a period of $2p$, a contradiction. We derive that $m_j - m_1$ is multiple of $|w|$.
Hence, we assume $m_j = m_{j-1} + q \cdot |w|$ for some q. The assumption that m_j is a midpoint of an ℓ^*-palindrome beside the inductive hypothesis implies (b') for j. The structure of $S[m_{j-1} + |w| - \ell^* + 1, m_{j-1} + |w| + \ell^*]$ shows that $m_{j-1} + |w|$ is a midpoint of an ℓ^*-palindrome. This means that $m_j = m_{j-1} + |w|$. This gives (a') and yields the induction step. ∎

Corollary 6.2.2 shows the structure of overlapping palindromes and is essential for the compression. The main difference between Corollary 6.2.2 and Lemma 4.3.1 is the required distance between the midpoints of a run. Lemma 4.3.1 assumes that every palindrome in the run overlaps with all other palindromes. In contrast, Corollary 6.2.2 assumes that every palindrome $P[m_j]$ overlaps with $P[m_{j-2}]$, $P[m_{j-1}]$, $P[m_{j+1}]$, and $P[m_{j+2}]$. It can be proven by an induction over the midpoints and using Lemma 4.3.1.

Corollary 4.3.2 *If $m_1, m_2, \ldots, m_h$ form an ℓ^*-run for an arbitrary natural number ℓ^* then*

(a) $m_1, m_2, m_3, \ldots, m_h$ *are equally spaced in S, i.e.,* $|m_2 - m_1| = |m_{k+1} - m_k|$ $\forall k \in \{1, \ldots, h-1\}$

(b) $S[m_1 + 1, m_h] = \begin{cases} (ww^R)^{\frac{h-1}{2}} & h \text{ is odd} \\ (ww^R)^{\frac{h-2}{2}} w & h \text{ is even} \end{cases}$, *where* $w = S[m_1 + 1, m_2]$.

Lemma 4.3.3 shows the pattern for the lengths of the palindromes in each half of the run. This allows us to only store a constant number of length estimates per run.

Lemma 4.3.3 *At iteration i, let $m_1, m_2, m_3, ..., m_h$ be midpoints of a maximal ℓ^*-run in $S[1, i]$ for an arbitrary natural number ℓ^*. For any midpoint m_j, we have:*

$$\ell(m_j, i) = \begin{cases} \ell(m_1, i) + (j-1) \cdot (m_2 - m_1) & j < \frac{h+1}{2} \\ \ell(m_h, i) + (h-j) \cdot (m_2 - m_1) & j > \frac{h+1}{2} \end{cases}$$

Proof We prove the first case where m_j is in the first half, i.e., $j < \frac{h+1}{2}$. The other case is similar. By Corollary 6.2.2, $S[m_1, m_h]$ is of the form $ww^Rww^R...$ and $m_j = m_1 + (j-1) \cdot |w|$,

where w is $S[m_1+1, m_2]$. Define m_0 to be the index $m_1 - |w|$. Since $\ell(m_1, i) \geq \ell^* \geq |w|$ we have $S[m_0+1, m_1] = w^R$.

By Corollary 6.2.2, we have that $S[m_0+1, m_j]^R = S[m_j+1, m_{2j}]$. This implies $\ell(m_j, i) \geq j \cdot |w|$. Define k to be $\ell(m_j, i) - j \cdot |w|$. We show that $k = \ell(m_0, i)$. By definition, k is the length of the longest suffix of $S[1, m_0]$ which is the reverse of the prefix of $S[m_{2j}+1, n]$. Corollary 6.2.2 shows that $S[m_{2j}+1, m_{2j}+\ell^*]$ is equal to $S[m_0+1, m_0+\ell^*]$ as both are prefixes of $w^R w w^R ...$. Therefore, k is also the same as the length of the longest suffix of $S[1, m_0]$ which is the reverse of the prefix of $S[m_0+1, m_0+\ell^*]$, i.e., $k = max\{k' | S[m_0 - k' + 1, m_0]^R = S[m_0+1, m_0+k']\} = \ell(m_0, i)$. Thus, $\ell(m_j, i) = \ell(m_0, i) + j \cdot |w|$. ∎

4.4 Analysis of the Algorithm

We show that one can convert R_S-entries into a run and vice versa and *ApproxSqrt*'s maintenance of R_F-entries and R_{NF}-entries does not impair the length estimates. The following lemma shows that one can retrieve the length estimate of a palindrome as well as its fingerprint from an R_F-entry.

Lemma 4.4.1 *At iteration i, the R_F-entry over $m_1, m_2, \ldots, m_h$ is a lossless compression of $[R_S(m_1, i); \ldots; R_S(m_h, i)]$*

Proof Fix an index j. We prove that we can retrieve $R_S(m_j, i)$ out of the R_F-entry representation. Corollary 6.2.2 gives a formula to retrieve m_j from the corresponding R_F-entry. Formally, $m_j = m_1 + (j-1) \cdot (m_1 - m_2)$.

Corollary 6.2.2 shows that $S[m_1+1, m_h]$ follows the structure $ww^R w^R \cdots$ where $w = S[m_1+1, m_2]$.

This structure allows us to retrieve $F^F(1, m_j), F^R(1, m_j)$, since we have $F^F(1, m_j) = \phi^F(S[1, m_1] w w^R w w^R \cdots)$.

We know argue that the length estimates have the same accuracy as R_S-entries. Note that the proof of Lemma 3.2.1 shows that after iteration i and any $R_S(m, i)$ we have $\ell(m, i) - \varepsilon\sqrt{n} < \tilde{\ell}(m, i) \leq \ell(m, i)$. We show that one can retrieve the length estimate for palindromes which are not stored explicitly by using the following equation. The equation from Lemma 4.3.3.

$$\tilde{\ell}(m_j, i) = \begin{cases} \tilde{\ell}(m_1, i) + (j-1) \cdot (m_2 - m_1) & j < \frac{h+1}{2} \\ \tilde{\ell}(m_h, i) + (h-j) \cdot (m_2 - m_1) & j > \frac{h+1}{2} \end{cases}.$$

Let i' be the index where R_F was finished. We distinguish among three cases:

1. $m_j = m_1$: Since the algorithm treats $P[m_j]$ as an R_S-entry in terms of comparisons, $\ell(m_j, i) - \varepsilon\sqrt{n} < \tilde{\ell}(m_j, i) \leq \ell(m_j, i)$ holds.

2. $m_j \in \{m_{\lfloor (h+1)/2 \rfloor}, m_{\lceil (h+1)/2 \rceil}, m_h\}$: At index i' *ApproxSqrt* executes step 5(a)ii and one can verify that $\ell(m_j, i') - \varepsilon\sqrt{n} < \tilde{\ell}(m_j, i') \leq \ell(m_j, i')$ holds. For all $i \geq i'$ palindrome $P[m_j]$ is treated as an R_S-entry in terms of comparisons. Thus, $\ell(m_j, i) - \varepsilon\sqrt{n} < \tilde{\ell}(m_j, i) \leq \ell(m_j, i)$ holds.

3. Otherwise, we assume WLOG. $1 < j < \lfloor \frac{h+1}{2} \rfloor$. Lemma 4.3.3 shows that $\ell(m_j, i) - \tilde{\ell}(m_j, i) = \ell(m_1, i) - \tilde{\ell}(m_1, i)$. We know $0 \leq \ell(m_1, i) - \tilde{\ell}(m_1, i) < \varepsilon\sqrt{n}$. Thus, $\ell(m_j, i) - \varepsilon\sqrt{n} < \tilde{\ell}(m_j, i) \leq \ell(m_j, i)$.

∎

Let *Compressed Run* be the general term for R_F-entry and R_{NF}-entry. We argue that in any interval of length $\sqrt{n}$ we only need to store at most two single palindromes and two *Compressed Runs*. Suppose there were three R_S-entries, then, by Corollary 6.2.2, they form a $\sqrt{n}$-run since they overlap each other. Therefore, the three R_S-entries would be stored in a *Compressed Run*. For a similar reason there cannot be more than two *Compressed Runs* in one interval of length $\sqrt{n}$. We derive the following observation.

Observation 2 *For any interval of length $\sqrt{n}$ there can be at most two R_S-entries and two* Compressed Runs *in L^*.*

We now have what we need in order to state and prove Theorem 4.4.2. It worth mentioning that the algorithm can easily be modified to report all palindromes $P[m]$ in S with $\ell(m) \geq t$ and no $P[m]$ with $\ell(m) < t - \varepsilon\sqrt{n}$ for some threshold $t \in \mathbb{N}$. For $t \leq \sqrt{n}$ one can modify the algorithm to report a palindrome $P[m]$ if and only if $\ell(m) \geq t$. Note, the algorithm is also $(1 + \varepsilon)$-approximation.

Theorem 4.4.2 (*ApproxSqrt*) *For any $\varepsilon \in [1/\sqrt{n}, 1]$* Algorithm ApproxSqrt$(S, \varepsilon)$ *reports for every palindrome $P[m]$ in S its midpoint m as well as an estimate $\tilde{\ell}(m)$ (of $\ell(m)$) such that w.h.p.*[3] *$\ell(m) - \varepsilon\sqrt{n} < \tilde{\ell}(m) \leq \ell(m)$. The algorithm makes one pass over S, uses $O(n/\varepsilon)$ time, and $O(\sqrt{n}/\varepsilon)$ space.*

[3] *We say an event happens* with high probability *(w.h.p.) if its probability is at least $1 - 1/n^c$ for $c \in \mathbb{N}$.*

Proof Similar to other proofs in this paper we assume that fingerprints do not fail as we take less than n^2 fingerprints and by Lemma 2.0.2, the probability that a fingerprint fails is at most $1/(n^4)$. Thus, by applying the union bound the probability that no fingerprint fails is at least $1 - n^{-2}$.

Correctness: Fix an arbitrary palindrome $P[m]$. For the case $\ell(m) < \sqrt{n}$ there is no difference between *Simple ApproxSqrt* and *ApproxSqrt*, so the correctness follows from Lemma 3.2.1. In the following, we assume $\ell(m) \geq \sqrt{n}$. Firstly, we argue that $R_S(m, n)$ is stored in $\hat{L}_n$. At index $i = m + \sqrt{n}$, *ApproxSqrt* adds $P[m]$ to $\hat{L}_i$. The algorithm does this by using the longest sliding fingerprint pair which guarantees that if $S[i - 2\sqrt{n} + 1, i - \sqrt{n}]$ is the reverse of $S[i - \sqrt{n} + 1, i]$, then the fingerprints of sliding window are equal. Moreover, a palindrome is never removed from $\hat{L}_i$ for $1 \leq i \leq n$. Additionally, Lemma 4.4.1 shows how to retrieve the midpoint. Hence, $R_S(m, n)$ is stored in $\hat{L}_n$.

We now argue $\ell(m) - \varepsilon\sqrt{n} < \tilde{\ell}(m) \leq \ell(m)$. Palindromes are stored in an R_S-entry, R_F-entry and R_{NF}-entry. Since we are only interested in the estimate $\tilde{\ell}(m)$ after the n^{th} iteration of *Simple ApproxSqrt* and since the algorithm finishes an R_{NF}-entry at iteration n, we know that there are no R_{NF}-entries at after iteration n.

1. $R_S(m, n)$ is stored as an R_S-entry. Since R_S-entries are treated in the same way as in *Simple ApproxSqrt*, $\ell(m) - \varepsilon\sqrt{n} < \tilde{\ell}(m) \leq \ell(m)$ holds by Lemma 3.2.1.

2. $R_S(m, n)$ is stored in the R_F-entry R_F. Then Lemma 4.4.1 shows the correctness.

Furthermore, the algorithm reports $\hat{L}_n$ step 7 of iteration n.

Space: The number of checkpoints equals $\lfloor n/\lfloor \varepsilon\sqrt{n} \rfloor \rfloor \leq 2n/\varepsilon\sqrt{n} = O(\sqrt{n}/\varepsilon)$, since $\varepsilon \geq 1/\sqrt{n}$. Moreover, there are $O(n/\varepsilon)$ *Fingerprint Pairs* which can be stored in $O(n/\varepsilon)$ space. The sliding window requires $2\sqrt{n}$ space. The space required to store the information of all $\sqrt{n}$-palindromes is bounded by $O(\sqrt{n})$: By Observation 2, the number of R_S-entries and *Compressed Runs* in an interval of length $\sqrt{n}$ is bounded by a constant. Each *Compressed Run* and each R_S-entry can be stored in constant space. Thus, in any interval of length $\sqrt{n}$ we only need constant space and thus altogether $O(\sqrt{n})$ space for storing the information of palindromes.

Running time: The running time of the algorithm is determined by the number of comparisons done at lines 6 and 4. First we bound the number of comparisons corresponding to line 6. For all $\sqrt{n}$-palindromes we bound the total number of comparisons by $O(\frac{n}{\varepsilon})$:

The *ApproxSqrt* checks only explicitly stored palindromes with checkpoints and therefore with R_S-entries and at most 4 midpoints per *Compressed Run*. As shown in Observation 2, there is at most a constant number of explicitly stored midpoints in every interval of length $\sqrt{n}$. In total, we have $O(\sqrt{n})$ explicitly stored midpoints and $O(\sqrt{n}/\varepsilon)$ fingerprints of checkpoints. We only check each palindrome at most once with each checkpoint [4]. Hence, the total number of comparisons is in order of $O(n/\varepsilon)$. Now, we bound the running time corresponding to Step 4. This step has two functions: There are $O(1/\varepsilon)$ *Fingerprint Pairs* which are updated every iteration. This takes $O(1/\varepsilon)$ time. Additionally, the middle of the sliding window is checked with at most $O(1/\varepsilon)$ *Fingerprint Pairs*. Thus, the time for Step 4 of the algorithm is bounded by $O(n/\varepsilon)$. ∎

[4] We can use an additional queue to store the index where the algorithm needs to *check* a checkpoint with a palindrome.

Chapter 5

The Exact Algorithm

This section describes *Algorithm Exact* which determines the exact length of the longest palindrome in S using $O(\sqrt{n})$ space and two passes over S.

5.1 Algorithm

For the first pass this algorithm runs *ApproxSqrt* $(S, \frac{1}{2})$ (meaning that $\varepsilon = 1/2$). The first pass returns ℓ_{max}, if $\ell_{max} < \sqrt{n}$ (Lemma 3.2.1). Otherwise, the first pass (Theorem 4.4.2) returns for every palindrome $P[m]$, with $\ell(m) \geq \sqrt{n}$, an estimate satisfying $\ell(m) - \sqrt{n}/2 < \tilde{\ell}(m) \leq \ell(m)$ w.h.p.

The algorithm for the second pass is determined by the outcome of the first pass. For the case $\ell_{max} < \sqrt{n}$, it uses the sliding window to find all $P[m]$ with $\ell(m) = \ell_{max}$. If $\ell_{max} \geq \sqrt{n}$, then the first pass only returns an additive $\sqrt{n}/2$-approximation of the palindrome lengths. We define the *uncertain intervals* of $P[m]$ to be: $I_1(m) = S[m - \tilde{\ell}(m) - \sqrt{n}/2 + 1, m - \tilde{\ell}(m)]$ and $I_2(m) = S[m + \tilde{\ell}(m) + 1, m + \tilde{\ell}(m) + \sqrt{n}/2]$. The algorithm uses the length estimate calculated in the first pass to delete all R_S-entries (Step 3) which cannot be the longest palindromes. Similarly, the algorithm (Step 2) only keeps the middle entries of R_F-entries since these are the longest palindromes of their run. In the second pass, *Algorithm Exact* stores $I_1(m)$ for a palindrome $P[m]$ if it was not deleted. *Algorithm Exact* compares the symbols of $I_1(m)$ symbol by symbol to $I_2(m)$ until the first mismatch is found. Then the algorithm knows the exact length $\ell(m)$ and discards $I_1(m)$. The analysis will show, at any time the number of stored uncertain intervals is bounded by a constant.

First Pass Run the following two algorithms simultaneously:

1. *ApproxSqrt* $(S, 1/2)$. Let L be the returned list.
2. The simple process in *ApproxSqrt* (See Lemma 4.4.2) which reports ℓ_{max} if $\ell_{max} < \sqrt{n}$.

Second Pass

- $\ell_{max} < \sqrt{n}$: Use a sliding window of size $2\sqrt{n}$ and maintain two fingerprints $F^R[i - \sqrt{n} - \ell_{max} + 1, i - \sqrt{n}]$, and $F^F[i - \sqrt{n} + 1, i - \sqrt{n} + \ell_{max}]$. Whenever these fingerprints match, report $P[i - \sqrt{n}]$.

- $\ell_{max} \geq \sqrt{n}$: In this case, the algorithm uses a preprocessing phase first.
  Preprocessing
  1. Set $\tilde{\ell}_{max} = \max\{\tilde{\ell}(m) \mid P[m] \text{ is stored in } L \text{ as an } R_F \text{ or an } R_S \text{ entry}\}$.
  2. For every R_F-entry R_F in L with midpoints $m_1, \ldots, m_h$ remove R_F from L and add
     $R_s(m, i) = (m, \tilde{\ell}(m), F^F(1, m), F^R(1, m))$ to L, for $m \in \{m_{\lfloor (h+1)/2 \rfloor}, m_{\lceil (h+1)/2 \rceil}\}$.
     To do this, calculate $m_{\lfloor \frac{1+h}{2} \rfloor} = m_1 + (\lfloor \frac{1+h}{2} \rfloor - 1)(m_2 - m_1)$ and $m_{\lceil \frac{1+h}{2} \rceil} = m_1 + (\lceil \frac{1+h}{2} \rceil - 1)(m_2 - m_1)$. Retrieve $F^F(1, m)$ and $F^R(1, m)$ for $m \in \{m_{\lfloor (h+1)/2 \rfloor}, m_{\lceil (h+1)/2 \rceil}\}$.
  3. Delete all R_S-entries $(m_k, \tilde{\ell}(m_k), F^F(1, m_k), F^R(1, m_k))$ with $\tilde{\ell}(m_k) \leq \tilde{\ell}_{max} - \sqrt{n}/2$ from L.
  4. For every palindrome $P[m] \in L$ set $I_1(m) := (m - \tilde{\ell}(m) - 1/2\sqrt{n}, m - \tilde{\ell}(m)]$ and set $finished(m) :=$ false.

  The resulting list is called L^*.
  String processing At iteration i the algorithm performs the following steps.
  1. Read $S[i]$. If there is a palindrome $P[m]$ such that $i \in I_1(m)$, then store $S[i]$.
  2. If there is a midpoint m such that $m + \tilde{\ell}(m) < i < m + \tilde{\ell}(m) + \frac{\sqrt{n}}{2}$, finished$(m)$ = false, and $S[m - (i - m) + 1] \neq S[i]$, then set finished$(m)$:= true and $\ell(m) = i - m - 1$.
  3. If there is a palindrome $P[m]$ such that $i \geq \tilde{\ell}(m) + m + \frac{\sqrt{n}}{2}$, then discard $I_1(m)$.
  4. If $i = n$, then output $\ell(m)$ and m of all $P[m]$ in L^* with $\ell(m) = \ell_{max}$.

5.2 Analysis

The analysis of *Algorithm Exact* is based on the observation that, after removing palindromes which are definitely shorter than the longest palindrome, at any time the number of stored uncertain intervals is bounded by a constant. The following Lemma shows that only the palindromes in the middle are strictly longer than the other palindromes of the run. This allows us to remove all palindromes which are not in the middle of the run. The techniques used in the lemma are very similar to the ideas used in Lemma 4.3.3. Let $\hat{L}_n$ be the list after the first pass.

Lemma 5.2.1 *Let $m_1, m_2, m_3, ..., m_h$ be midpoints of a maximal ℓ^*-run in S. For every $j \in \{1, \ldots, h\} \setminus \{\lfloor (h+1)/2 \rfloor, \lceil (h+1)/2 \rceil\}$,*

$$\ell(m_j) < max\{\ell(m_{\lfloor (h+1)/2 \rfloor}), \ell(m_{\lceil (h+1)/2 \rceil})\}.$$

Proof If h is even, the claim follows from Lemma 4.3.3. Therefore, we assume $h = 2d - 1$ which means that $m_{\lfloor (h+1)/2 \rfloor} = m_{\lceil (h+1)/2 \rceil} = m_d$. Hence, we have to show that $\ell(m_j) < \ell(m_d)$. Define w exactly as it is defined in Lemma 4.3.3 to be $S[m_1 + 1, m_2]$. Note that $S[m_{h-1} + 1, m_h] = w^R$. We need two claims:

1. $\ell(m_d) \geq (d-1)|w| + min\{\ell(m_1), \ell(m_h)\}$
   Proof: Suppose $\ell(m_d) < (d-1)|w| + min\{\ell(m_1), \ell(m_h)\}$. We know that $S[m_1 + 1, m_d]$ is the reverse of $S[m_d + 1, m_h]$. Therefore, $\ell(m_d) = (d-1)|w| + k$ where k is the length of the longest suffix of $S[1, m_1]$ which is the reverse of the prefix of $S[m_h + 1, n]$. Formally, $k = max\{k' | S[m_1 - k' + 1, m_1]^R = S[m_h + 1, m_h + k']\}$.
   Define $\ell' \triangleq min\{\ell(m_1), \ell(m_h)\}$. Thus, it suffices to show that $k \geq min\{\ell(m_1), \ell(m_h)\} = \ell'$. Observe, $\ell' < \ell^* + |w|$ since otherwise $m_1 - |w|$ or $m_h + |w|$ would be a part of the run. Since m_1 is the midpoint of a palindrome of length $\ell' < \ell^* + |w|$, by Corollary 6.2.2, the left side of m_1, i.e., $S[m_1 - \ell' + 1, m_1]$ is a suffix of length ℓ' of $ww^R \ldots ww^R$. Similarly, $S[m_h + 1, m_h + \ell']$ is a prefix of length ℓ' of $ww^R \ldots ww^R$. These two facts imply that $S[m_1 - \ell' + 1, m_1]$ is the reverse of $S[m_h + 1, m_h + \ell']$ and thus $k \geq \ell'$. □

2. $|\ell(m_h) - \ell(m_1)| < |w|$
   Proof: WLOG., let $\ell(m_h) \geq \ell(m_1)$, then suppose $\ell(m_h) - \ell(m_1) \geq |w|$. This implies that $\ell(m_h) \geq |w| + \ell(m_1) \geq |w| + \ell^*$. By Lemma 6.2.2, we derive that the run is not maximal, i.e., there is a midpoint of a palindrome with length of ℓ^* at index $m_h + |w|$. A contradiction. □

Using these properties we claim that $\ell(m_d) > \ell(m_j)$:

1. For $j < d$: $\ell(m_d) \underset{Property\ 1}{\geq} (d-1)|w| + min\{\ell(m_1), \ell(m_h)\} \underset{Property\ 2}{>} (d-1)|w| + \ell(m_1) - |w| = (d-2)|w| + \ell(m_1) - |w| \geq (j-1)|w| + \ell(m_1) \underset{Lemma\ 4.3.3}{=} \ell(m_j)$.
2. For $j > d$: $\ell(m_d) \underset{Property\ 1}{\geq} (d-1)|w| + min\{\ell(m_1), \ell(m_h)\} \underset{Property\ 2}{>} (d-1)|w| + \ell(m_h) - |w| = (d-2)|w| + \ell(m_h) - |w| \geq (j-1)|w| + \ell(m_h) \underset{Lemma\ 4.3.3.}{=} \ell(m_j)$.

This yields the claim. ∎

We conclude this section by stating and proving Theorem 5.2.2 covering the correctness of the algorithm as well as the claimed space and time bounds. We say a palindrome with midpoint m *covers* an index i if $|m - i| \leq \ell(m)$.

Theorem 5.2.2 (*Exact*) Algorithm Exact *reports w.h.p.* ℓ_{max} *and* m *for all palindromes* $P[m]$ *with a length of* ℓ_{max}*. The algorithm makes two passes over* S*, uses* $O(n)$ *time, and* $O(\sqrt{n})$ *space.*

Proof In this proof, similar to Theorem 4.4.2, we assume that the fingerprints do not fail w.h.p.
For the case $\ell_{max} < \sqrt{n}$ it is easy to see that the algorithm satisfies the theorem.
Therefore, we assume $\ell_{max} \geq \sqrt{n}$.
Correctness: After the first pass we know that due to Theorem 4.4.2 all $\sqrt{n}$-palindromes are in L. The algorithm removes some of those palindromes and we argue that a palindrome which is removed from L cannot be the longest palindrome. A palindrome removed in

- Step 2 is, by Lemma 5.2.1, strictly shorter than the palindromes of the middle of the run from which it was removed.
- Step 3 with midpoint m has a length which is bounded by $\tilde{\ell}_{max} - \tilde{\ell}(m) \geq \sqrt{n}/2$. We derive $\ell_{max} \geq \tilde{\ell}_{max} \geq \sqrt{n}/2 + \tilde{\ell}(m) > \ell(m)$ where the last inequality follows from $\ell(m) - \sqrt{n}/2 < \tilde{\ell}(m) \leq \ell(m)$.

Therefore, all longest palindromes are in L^*. Furthermore, the exact length of $P[m]$ is determined at iteration $m + \ell(m) + 1$ since this is the first iteration where $S[m - (\ell(m) + 1)+1] \neq S[m+\ell(m)+1]$. In Step 2, the algorithm sets the exact length $\ell(m)$. If $\ell(m) = \ell_{max}$, then the algorithm reports m and ℓ_{max} in step 4 of iteration n.

Space: For every palindrome we have to store at most one uncertain interval. At iteration i, the number of uncertain intervals we need to store equals the number of palindromes which cover index i. We prove in the following that this is bounded by 4. We assume ε to be $1/2$ and $\ell_{max} \geq \sqrt{n}$. Define $\tilde{\ell}_{min}$ to be the length of the palindrome in L^* for which the estimate is minimal. All palindromes in L^* have a length of at least $\sqrt{n}$, thus $\tilde{\ell}_{min} \geq \sqrt{n}$. We define the following intervals:

- $\mathcal{I}_1 = (i - \tilde{\ell}_{min} - \sqrt{n}, i - \tilde{\ell}_{min}]$
- $\mathcal{I}_2 = (i - \tilde{\ell}_{min}, i]$

Recall that the algorithm removes all palindromes which have a length of at most $\tilde{\ell}_{max} - \sqrt{n}/2$ and thus $\tilde{\ell}_{min} \geq \tilde{\ell}_{max} - \sqrt{n}/2$. Additionally, we know by Theorem 4.4.2 that $\ell_{max} - \tilde{\ell}_{max} < \sqrt{n}/2$. We derive: $\ell_{max} - \tilde{\ell}_{min} \leq \ell_{max} - \tilde{\ell}_{max} + \sqrt{n}/2 < \sqrt{n}/2 + \sqrt{n}/2 = \sqrt{n}$ and therefore $\ell_{max} < \sqrt{n} + \tilde{\ell}_{min}$. Hence, there is no palindrome which covers i and has a midpoint outside of the intervals $\mathcal{I}_1$ and $\mathcal{I}_2$. It remains to argue that the number palindromes which are centered in $\mathcal{I}_1$ and $\mathcal{I}_2$ and stored in L^* is bounded by four: Suppose there were at least four palindromes in $\mathcal{I}_1$ and $\mathcal{I}_2$ which cover i. Lemma 4.3.1 shows that in any interval of length $\tilde{\ell}_{min}$ either the number of palindromes is bounded by two or they form an $\tilde{\ell}_{min}$-run. Thus there has to be at least one $\tilde{\ell}_{min}$-run. Recall that step 2 keeps for all R_F-entries only the midpoints in the middle of the run. The first pass does not create R_F-entries for all runs where the difference between two consecutive midpoints is more than $\sqrt{n}/2$ (See Definition 3 and step 5(a)i of *ApproxSqrt*). Thus, there is a run where the distance between two consecutive midpoints is greater than $\sqrt{n}/2$. By Lemma 4.3.3, the difference between $\ell(m)$ for distinct midpoints m of one side of this run is greater than $\sqrt{n}/2$. Since the checkpoints are equally spaced with consecutive distance of $\sqrt{n}/2$, the difference between $\tilde{\ell}(m)$ for distinct midpoints m of one side of this run is greater than $\sqrt{n}/2$ as well. This means that just the palindrome(s) in the middle of the run, say $P[m]$, satisfy the constraint $\tilde{\ell}(m) \geq \tilde{\ell}_{max} - \sqrt{n}/2$ and therefore the rest of the palindromes of this run would have been deleted. A contradiction. Thus, there are at most two palindromes in both intervals and thus in total at most four palindromes and four uncertain intervals. This yields the space bound of $O(\sqrt{n})$.

Running time: As shown in Theorem 4.4.2 the running time is in $O(n)$. The preprocessing and the last step of the second pass can be done in $O(n)$. If the first pass returned an $\ell_{max} < \sqrt{n}$ then the running time of the second pass is trivially in $O(n)$. Suppose $\ell_{max} \geq \sqrt{n}$.

The preprocessing and the last step of *Algorithm Exact* can be done in $O(n)$ time and they are only executed once. The remaining operations can be done in constant time per symbol. This yields the time bound of $O(n)$. ∎

Chapter 6

The Logarithmic Space Algorithm

In this chapter, we present an algorithm which reports one of the longest palindromes and uses only logarithmic space. We call it *ApproxLog* and we express it in the following section.

6.1 Algorithm

ApproxLog has a multiplicative error instead of an additive error term. Similar to *ApproxSqrt* we have special indices of S designated as checkpoints that we keep along with some constant size data in memory. The checkpoints are used to estimate the length of palindromes. However, this time checkpoints (and their data) are only stored for a limited time. Since we move from additive to multiplicative error we do not need checkpoints to be spread evenly in S. At iteration i, the number of checkpoints in any interval of fixed length decreases exponentially with distance to i. The algorithm stores a palindrome $P[m]$ (as an R_S-entry or R_{NF}-entry) until there is a checkpoint c such that $P[m]$ was checked unsuccessfully against c. A palindrome is stored in the lists belonging to the last checkpoint with which is was checked successfully. In what follows we set $\delta \triangleq \sqrt{1+\varepsilon} - 1$ for the ease of notation. Every checkpoint c has an attribute called $level(c)$. It is used to determine the number of iterations the checkpoint data remains in the memory.

Memory invariants. After algorithm *ApproxLog* has processed $S[1, i-1]$ and before reading $S[i]$ it contains the following information:

1. Two *Master Fingerprints* up to index $i-1$, i.e., $F^F(1, i-1)$ and $F^R(1, i-1)$.
2. A list of checkpoints CL_{i-1}. For every $c \in \mathrm{CL}_{i-1}$ we have

- $level(c)$ such that c is in CL_{i-1} if and only if $c \geq (i-1) - 2(1+\delta)^{level(c)}$.
- $fingerprint(c) = F^R(1, c)$
- a list L_c. It contains all palindromes which were successfully checked with c, but with no other checkpoint $c' < c$. The palindromes in L_c are either R_S-entries or R_{NF}-entries (See the *Algorithm ApproxSqrt*).

3. The midpoint m^*_{i-1} and the length estimate $\tilde{\ell}(m^*_{i-1}, i-1)$ of the longest palindrome found so far.

The algorithm maintains the following property. If $P[m, i]$ was successfully checked with checkpoint c but with no other checkpoint $c' < c$, then the palindrome is stored in L_c. The elements in L_c are ordered in increasing order of their midpoint. The algorithm stores palindromes as R_S-entries and R_{NF}-entries. This time however, the length estimates are not maintained. Adding a palindrome to a current run works exactly (the length estimate is not calculated) as described in *Algorithm ApproxSqrt*.

Maintenance. At iteration i the algorithm performs the following steps.

1. Read S[i]. Update the *Master Fingerprints* to be $F^F(1, i)$ and $F^R(1, i)$.
2. For all $k \geq k_0 = {}^{\log(1/\delta)}/_{\log(1+\delta)}$(The algorithm does not maintain intervals of size 0.)
   (a) If i is a multiple of $\lfloor \delta(1+\delta)^{k-2} \rfloor$, then add the checkpoint $c = i$ (along with the checkpoint data) to CL_i. Set $level(c) = k$, $fingerprint(c) = F^R(1, i)$ and $L_c = \emptyset$.
   (b) If there exists a checkpoint c with $level(c) = k$ and $c < i - 2(1+\delta)^k$, then prepend L_c to $L_{c'}$ where $c' = max\{c'' \mid c'' \in \mathrm{CL}_i \text{ and } c'' > c\}$. Merge and create runs in L_c if necessary (Similar to step 5 of *ApproxSqrt*). Delete c and its data from CL_i.
3. For every checkpoint $c \in \mathrm{CL}_i$
   (a) Let m_c be the midpoint of the first entry in L_c and $c' = max\{c'' \mid c'' \in \mathrm{CL}_i \text{ and } c'' < c\}$.
   If $i - m_c = m_c - c'$, then we *check* $P[m]$ against c' by doing the following:
      i. If the left side of m_c is the reverse of the right side of m_c (i.e., $F^R(c', m_c) = F^F(m_c, i)$) then move $P[m_c]$ from L_c to $L_{c'}$ by adding $P[m_c]$ to $L_{c'}$:
         A. If $|L_{c'}| \leq 1$, store $P[m_c]$ as a R_S-entry.

B. If $|L_{c'}| = 2$, create a run out of the R_S-entries stored in $L_{c'}$ and $P[m_c]$.

C. Otherwise, add $P[m_c]$ to the R_{NF}-entry in $L_{c'}$.

ii. If the left side of m_c is *not* the reverse of the right side of m_c, then remove m_c from L_c.

iii. If $i - m_c > \tilde{\ell}(m_i^*)$, then set $m_i^* = m_c$ and set $\tilde{\ell}(m_i^*) = i - m_c$.

4. If $i = n$, then report m_i^* and $\tilde{\ell}(m_i^*)$.

6.2 Analysis

ApproxLog relies heavily on the interaction of the following two ideas. The pattern of the checkpointing and the compression which is possible due to the properties of overlapping palindromes (Lemma 4.3.1). On the one hand the checkpoints are close enough so that the length estimates are accurate (Lemma 6.2.4). The closeness of the checkpoints ensures that palindromes which are stored at a checkpoint form a run (Lemma 6.2.3) and therefore can be stored in constant space. On the other hand the checkpoints are far enough apart so that the number of checkpoints and therefore the required space is logarithmic in n.
We start off with an observation to characterise the checkpointing. Step 2 of the algorithm creates a checkpoint pattern: Recall that the level of a checkpoint is determined when the checkpoint and its data are added to the memory. The checkpoints of every level have the same distance. A checkpoint (along with its data) is removed if its distance to i exceeds a threshold which depends on the level of the checkpoint. Note that one index of S can belong to different levels and might therefore be stored several times. The following observation follows from Step 2 of the algorithm.

Observation 3 *At iteration i, $\forall k \geq k_0 = \lceil \frac{log(\frac{(1+\delta)^2}{\delta})}{log(1+\delta)} \rceil$. Let $C_{i,k} = \{c \in CL_i \mid level(c) = k\}$.*

1. *$C_{i,k} \subseteq [i - 2(1+\delta)^k, i]$.*
2. *The distance between two consecutive checkpoints of $C_{i,k}$ is $\lfloor \delta(1+\delta)^{k-2} \rfloor$.*
3. *$|C_{i,k}| = \lceil \frac{2(1+\delta)^k}{\lfloor \delta(1+\delta)^{k-2} \rfloor} \rceil$.*

This observation can be used to calculate the size of the checkpoint data which the algorithm stores at any time.

Lemma 6.2.1 *At Iteration i of the algorithm the number of checkpoints is in $O\left(\frac{\log(n)}{\varepsilon \log(1+\varepsilon)}\right)$.*

Proof Given $m_1, m_2, \ldots, m_h$ and ℓ^* we prove the following stronger claim by induction over the midpoints $\{m_1, \ldots, m_j\}$. (a') $m_1, m_2, \ldots, m_j$ are equally spaced. (b') $S[m_1+1, m_j+\ell^*]$ is a prefix of ww^Rww^R... .

Base case $j = 2$: Since we assume m_1 is the midpoint of an ℓ^*-palindrome and $\ell^* \geq m_h - m_1 \geq m_2 - m_1 = |w|$, we have that $S[m_1 - |w| + 1, m_1] = w^R$. Recall that $\ell(m_2) \geq \ell^* \geq |w|$ and thus, $S[m_1 + 1, m_2 + |w|] = ww^R$.

We can continue this argument and derive that $S[m_1+1, m_2+\ell^*]$ is a prefix of $ww^R \ldots ww^R$. (a') for $j = 2$ holds trivially.

Inductive step $j - 1 \to j$: Assume (a') and (b') hold up to m_{j-1}. We first argue that $|m_j - m_1|$ is a multiple of $|m_2 - m_1| = |w|$. Suppose $m_j = m_1 + |w| \cdot q + r$ for some integers $q \geq 0$ and $r \in \{1, \ldots, |w| - 1\}$. Since $m_j \leq m_{j-1} + \ell^*$, the interval $[m_1 + 1, m_{j-1} + \ell^*]$ contains m_j. Therefore, by inductive hypothesis, $m_j - r$ is an index where either w or w^R starts. This implies that the prefix of ww^R(or w^Rw) of size $2r$ is a palindrome and the string ww^R(or w^Rw) has period $2r$. On the other hand, by consecutiveness assumption, there is no midpoint of an ℓ^*-palindrome in the interval $[m_1 + 1, m_2 - 1]$ without a period of $2p$, a contradiction. We derive that $m_j - m_1$ is multiple of $|w|$.

Hence, we assume $m_j = m_{j-1} + q \cdot |w|$ for some q. The assumption that m_j is a midpoint of an ℓ^*-palindrome beside the inductive hypothesis implies (b') for j. The structure of $S[m_{j-1} + |w| - \ell^* + 1, m_{j-1} + |w| + \ell^*]$ shows that $m_{j-1} + |w|$ is a midpoint of an ℓ^*-palindrome. This means that $m_j = m_{j-1} + |w|$. This gives (a') and yields the induction step. ∎

Corollary 6.2.2 shows the structure of overlapping palindromes and is essential for the compression. The main difference between Corollary 6.2.2 and Lemma 4.3.1 is the required distance between the midpoints of a run. Lemma 4.3.1 assumes that every palindrome in the run overlaps with all other palindromes. In contrast, Corollary 6.2.2 assumes that every palindrome $P[m_j]$ overlaps with $P[m_{j-2}]$, $P[m_{j-1}]$, $P[m_{j+1}]$, and $P[m_{j+2}]$. It can be proven by an induction over the midpoints and using Lemma 4.3.1.

Corollary 6.2.2 *If $m_1, m_2, \ldots, m_h$ form an ℓ^*-run for an arbitrary natural number ℓ^* then*

(a) $m_1, m_2, m_3, \ldots, m_h$ are equally spaced in S, i.e., $|m_2 - m_1| = |m_{k+1} - m_k|$ $\forall k \in \{1, \ldots, h-1\}$

(b) $S[m_1+1, m_h] = \begin{cases} (ww^R)^{\frac{h-1}{2}} & h \text{ is odd} \\ (ww^R)^{\frac{h-2}{2}}w & h \text{ is even} \end{cases}$, *where* $w = S[m_1+1, m_2]$.

Proof We prove this by induction. Suppose j_0 is the highest index where $m_{j_0} < m_1 + \ell^*$. By Definition 2, we have $j_0 \geq 3$. We start by proving the induction basis. By Lemma 4.3.1, the claim holds for $m_1, m_2, \ldots, m_{j_0}$, i.e., they are equally spaced and $S[m_1+1, m_{j_0}]$ is a prefix of $ww^R \ldots ww^R$. For the inductive step we assume that the claim holds for m_{j-1}. Consider the midpoints m_{j-1}, m_j, m_{j+1}. Since $m_{j+1} - m_{j-1} \leq \ell^*$, Lemma 4.3.1 shows that those midpoints fulfill the claimed structure. ∎

The space bounds of Theorem 6.2.5 hold due to the following property of the checkpointing: If there are more than three palindromes stored in a list L_c for checkpoint c, then the palindromes form a run and can be stored in constant space as the following lemma shows.

Lemma 6.2.3 *At iteration i, let $c \in CL_i$ be an arbitrary checkpoint. The list L_c can be stored in constant space.*

Proof We fix an arbitrary $c \in \mathrm{CL}_i$. For the case that there are less than three palindromes belonging to L_c, they can be stored as R_S-entries in constant space. Therefore, we assume the case where there are at least three palindromes belonging to L_c and we show that they form a run. Let c' be the highest (index) checkpoint less than c, i.e., $c' = max\{c'' \mid c'' \in \mathrm{CL}_i \text{ and } c'' < c\}$. We disregard the case that the index of c is 1. Let k be the minimum value such that $(1+\delta)^{k-1} < i - c \leq (1+\delta)^k$. Recall that L_c is the list of palindromes which the algorithm has successfully checked against c and not against c' yet. Let $P[m]$ be a palindrome in L_c. Since it was successfully checked against c we know that $i-m \geq m-c$. Similarly, since $P[m]$ was not checked against c' we have $i-m < m-c'$. Thus, for every $P[m]$ in L_c we have $\frac{i+c'}{2} < m \leq \frac{i+c}{2}$. Therefore, all palindromes stored in L_c are in an interval of length less than $\frac{i+c}{2} - \frac{i+c'}{2} = \frac{c-c'}{2}$. If we show that $\ell(m) \geq \frac{c-c'}{2}$ for all $P[m]$ in L_c, then applying Lemma 4.3.1 with $\ell^* = \frac{c-c'}{2}$ on the palindromes in L_c implies that they are forming a run. The run can be stored in constant space in an R_{NF}-entry. Therefore, it remains to show that $\ell(m) \geq \frac{c-c'}{2}$. We first argue the following: $c - c' \underset{Obs.\ 3}{\leq} \delta(1+\delta)^{k-2} \underset{\delta \leq 1}{\leq} \frac{(1+\delta)^{k-1}}{2} \underset{Def.\ of\ k}{<} \frac{i-c}{2}$. Since $P[m]$ was successfully checked against c and since $m > \frac{i+c'}{2}$ we derive that $\ell(m) > \frac{i+c'}{2} - c$. Therefore, $\ell(m) > \frac{i+c'}{2} - c = \frac{i-c}{2} + \frac{c'-c}{2} \underset{(6.2)}{>} c - c' + \frac{c'-c}{2} = \frac{c-c'}{2}$. ∎

The following lemma shows that the checkpoints are sufficiently close in order to satisfy the multiplicative approximation.

Lemma 6.2.4 ApproxLog *reports a midpoint* m^* *such that w.h.p.* $\frac{\ell_{max}}{(1+\varepsilon)} \leq \tilde{\ell}(m^*) \leq \ell_{max}$.

Proof In step 4 of iteration i, *ApproxLog* reports the midpoint and length estimate of $P[m^*]$. We first argue that $\tilde{\ell}(m^*) \leq \ell(m^*) \leq \ell_{max}$. Let i' be the last time $\tilde{\ell}(m^*)$ was updated by step 3(a)iii of the algorithm. By the condition of step 3(a)iii, $S[c'+1, m^*]$ is the reverse of $S[m^*+1, i']$, where $c' = 2 \cdot m^* - i'$. Hence, we derive $\ell(m^*) \geq i' - m = \tilde{\ell}(m^*)$. Moreover, by the definition of ℓ_{max} we have $\ell_{max} \geq \ell(m^*)$.
We now argue $\frac{\ell_{max}}{(1+\varepsilon)} \leq \tilde{\ell}(m^*)$. Let $P[m_{max}]$ be a palindrome of maximum length, i.e., $\ell(m_{max}) = \ell_{max}$. Let k be an integer such that $(1+\delta)^{k-1} < \ell_{max} \leq (1+\delta)^k$. Consider $\tilde{\ell}(m^*)$ after the algorithm processed $S[1, i']$, where $i' = m_{max} + (1+\delta)^{k-1}$. By Observation 3, there is a checkpoint in interval $[i' - 2 \cdot (1+\delta)^{k-3}, i' - 2 \cdot (1+\delta)^{k-1} + \delta(1+\delta)^{k-3}]$. Let c denote this checkpoint. *ApproxLog* successfully checked $P[m_{max}]$ against this checkpoint and therefore the value $\tilde{\ell}(m^*)$ is set to at least $m_{max} - c$. We have $m_{max} - c \geq (1+\delta)^{k-1} - \delta(1+\delta)^{k-3}$. Thus, we have $\tilde{\ell}(m^*) \geq m_{max} - c \geq (1+\delta)^{k-1} - \delta(1+\delta)^{k-3}$. Therefore, $\frac{\ell_{max}}{\tilde{\ell}(m^*)} \leq \frac{(1+\delta)^k}{(1+\delta)^{k-1} - \delta(1+\delta)^{k-3}} = \frac{(1+\delta)^3}{(1+\delta)^2 - \delta} \leq (1+\delta)^2 = 1 + \varepsilon$.where the last equation follows from $\delta = \sqrt{1+\varepsilon} - 1$.
∎

We conclude this chapter by stating and proving Theorem 6.2.5. Arguably the most significant contribution of this paper is an algorithm which requires only logarithmic space. In contrast to *ApproxSqrt* (Theorem 4.4.2) this algorithm has a multiplicative error and it reports only one of the longest palindromes (see *Longest Palindromic Substring Problem*) instead of all of them due to the limited space.

Theorem 6.2.5 (*ApproxLog*) *For any* ε *in* $(0, 1]$, Algorithm ApproxLog *reports w.h.p. an arbitrary palindrome* $P[m]$ *of length at least* $\ell_{max}/(1+\varepsilon)$. *The algorithm makes one pass over* S, *uses* $O(\frac{n \log(n)}{\varepsilon \log(1+\varepsilon)})$ *time, and* $O(\frac{\log(n)}{\varepsilon \log(1+\varepsilon)})$ *space.*

Proof In this proof, similar to Theorems 4.4.2 and 5.2.2, we assume that the fingerprints do not fail w.h.p. as *ApproxLog*, similar to *ApproxSqrt*, does not take more than n^2 fingerprints during the processing of any input of length n.
Correctness: The correctness of the algorithm follows from Lemma 6.2.4. It remains to argue that the space and time are not exceeded.

Space: The space required by *ApproxLog* is dominated by space needed to store the palindromes (corresponding midpoints and fingerprints) in L_c for all $c \in \mathrm{CL}_i$. Lemma 6.2.3 shows that for any $c \in \mathrm{CL}_i$ the list L_c can be stored in constant space. Furthermore, Lemma 6.2.1 shows that there are $O\left(\frac{\log(n)}{\varepsilon \log(1+\varepsilon)}\right)$ elements in CL_i.
Running time: The running time is determined by step 2b and 3. The algorithm goes in every iteration through all checkpoints in CL_i which has $O\left(\frac{\log(n)}{\varepsilon \log(1+\varepsilon)}\right)$ elements as Lemma 6.2.1 shows. For each checkpoint the steps (2b and 3) take only constant time. Thus, the required time to process the whole input is $O\left(\frac{n \log(n)}{\varepsilon \log(1+\varepsilon)}\right)$. ∎

We are ready to prove Theorem 6.2.5. The correctness follows from Lemma 6.2.4. Lemma 6.2.1 and Lemma 6.2.3 yield the claimed space. In every iteration the algorithm processes every checkpoint in CL_i in constant time. The number of checkpoints is bounded by Lemma 6.2.1.

Chapter 7

Lower Bound

In this chapter, we prove that lower bounds on the space of any randomized algorithm. We give in Lemma 7.1.1 a probability distribution over input streams. This distribution gives a lower bound on the required space of an optimal deterministic algorithm which approximates the length of the longest palindrome within an additive error.

7.1 Proof

Theorem 7.1.2 applies Yao's principle (for further reading see [39]) and shows a lower bound on the space of any randomized algorithm which approximates the length of the longest palindrome. Recall that Σ denotes the set of all input symbols. A *memory cell* is a memory unit to store a symbol from Σ. Let $C(S, A)$ denote the required space of algorithm A on the input stream S.

Lemma 7.1.1 *Let A be an arbitrary deterministic algorithm which approximates the length of the longest palindrome up to an additive error of e_r elements. For any positive integers m and e_r, there is a probability distribution p over $\Sigma^{2m(2e_r+1)+4e_r}$ such that $E[C(S, A)] \geq m$, where $S \in \Sigma^{2m(2e_r+1)+4e_r}$ is a random variable and follows the distribution p.*

Proof We construct a set of input streams $\mathcal{S}$ and we discuss afterwards the expected space that A needs to process a random input from $S \in \mathcal{S}$. We define the string $[2e_r] = 1, 2, 3, ..., 2 \cdot e_r$ and similarly $[2e_r]^R = 2 \cdot e_r, 2 \cdot e_r - 1, \ldots, 2, 1$. Define $\mathcal{S}$ such that each element of $\mathcal{S}$ consists of $2m$ arbitrary symbols from Σ separated by the string $[2e_r]$ in the first half and

by $[2e_r]^R$ in the second half. Formally,

$$S = \{[2e_r]a_1[2e_r]a_2[2e_r]...a_m[2e_r][2e_r]^R a_{m+1}[2e_r]^R a_{m+2}\dots[2e_r]^R a_{2m}[2e_r]^R \mid \forall j, a_j \in \Sigma\}.$$

We define the input distribution p to be the uniform distribution on $\mathcal{S}$. Let ℓ denote the length of a stream $S \in \mathcal{S}$, i.e., $\ell = 2m(2e_r + 1) + 4e_r$. In the following, we show that A needs to store $a_1, \dots, a_m$ in order to have an approximation of at most e_r.
If A does not store all of them, then A behaves the same for two streams $S_1, S_2 \in \mathcal{S}$ with

1. $S_1[1, \ell/2] \neq S_2[1, \ell/2]$
2. $S_1[1, \ell/2] = S_2[\ell/2, \ell]^R = S_1[\ell/2, \ell]^R$

Let $x = j(2e_r+1)$ be the highest index of $S_1[1, \ell/2]$ such that $S_1[x] \neq S_2[x]$ for $j \in \{1, \dots, m\}$. If A has the same memory content at index $\ell/2$ for S_1 and S_2, then A returns the same approximation for both streams, but their actual longest palindromes are of lengths $(m + 1)2e_r$ and $(m - j + 1) \cdot 2e_r$ where $j \geq 1$. Hence, no matter which approximation A returns, it differs by more than e_r of either S_1 or S_2. Thus, A must have distinct memory contents after reading index $\ell/2$.
In what follows we argue that the expected required space to store $a_1, \dots, a_m$ is m.
We use Shannon's entropy theorem (for further reading on information theory see [44]) to derive a lower bound on the expected size of the memory. By Shannon's entropy theorem, the expected length of this encoding cannot be smaller than the distribution's entropy. Fix an arbitrary assignment for the variables $a_1, \dots, a_m$. The probability for a string to S to have the same assignment is $\frac{1}{|\Sigma|^m}$. The entropy of an uniform distribution is logarithmic in the size of the domain. Hence, $log(|\Sigma|^m) = m \cdot log(|\Sigma|)$ (or m memory cells) is the lower bound on the expected space of A. ∎

Theorem 7.1.2 uses Yao's technique to prove the lower bound for randomized algorithms' space on the worst-case input, using deterministic algorithms' space on random inputs.

Theorem 7.1.2 *Any randomized algorithm for approximating the length of the longest palindrome in a stream of length ℓ has the following property: In order to approximate the length up to an* additive error *of e_r elements it must use $\Omega(\ell/e_r)$ space.*

Proof Let $\mathcal{S}$ be the set of all possible input streams. Let $\mathcal{D}$ be the set of all deterministic algorithms. By Yao's principle ([39]), we derive the following. For any random variable

S_p over input streams which follows a probability distribution p and for any randomized algorithm D_q which is a probability distribution q on deterministic algorithms we have:

$$\underset{A\in\mathcal{D}}{Min}E[C(s_p, A)] \leq \underset{I\in\mathcal{S}}{Max}E[C(I, D_q)]$$

Lemma 7.1.1 gives a lower bound for the left hand side of the above inequality: It shows that for a stream of length $2m(2e_r+1)+4e_r$ at least m memory cells are required in order to achieve an additive error of at most e_r. Therefore, the required space for $\ell = 2m(2e_r+1)+2e_r$ is $\Omega(\ell/e_r)$. One can generalize this for any ℓ by using padding. Thus,

$$\Omega(l/e) \leq \underset{I\in\mathcal{S}}{Max}E[C(I, D_q)].$$

∎

Theorem 7.1.3 can be derived by setting $\ell = n$ and $e_r = \varepsilon\sqrt{n}$.

Theorem 7.1.3 *Any randomized one-pass algorithm that approximates the length of the longest palindrome up to an additive error of $\varepsilon\sqrt{n}$ must use $\Omega(\sqrt{n}/\varepsilon)$ space.*

Corollary 7.1.4 is another direct implication that can be obtained by setting $e_r = 1$.

Corollary 7.1.4 *There is no any randomized algorithm that computes the length of the longest palindrome in a given string precisely and uses a sublinear number of memory cells.*

Chapter 8

Conclusion

The work presented in this thesis studies complexity of the *Longest Palindromic Substring Problem* in the streaming model. Palindromes are motivated to be studied in computer science through variety of applications. It has been used in several old literature in different languages. However we consider a new application in bioinformatcs area, where a complementary palindrome in an RNA corresponds to certain shape, called hairpin loop. The results discussed in this thesis can be considered as a starting point to process long RNA sequences and estimate overall shape and how it could fold.

For the case, where one is interested in finding all palindromic substrings with an additive approximation to their length, A sublinear space algorithm is presented. The tools used for this end, includes Karp-Rabin fingerprint, which shows the strength of this tool despite the time that has passed since it was first introduced. Also, the properties of overlapping palindromes are studied closely further than what has been known in the past. To prove optimality, matching lower bound is given. Moreover, an interesting, carefully designed checkpointing scheme is introduced, which allows us to present a logarithmic space algorithm considering multiplicative approximation.

There are still plenty of classical string problems needed to be studied in streaming model, which finds its applications in bioinformatics areas. Namely, different generalizations of palindrome concept can be good candidates to be discussed in the area which allows human being to be able to process long streams of data.

Bibliography

[1] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[2] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 363–372, 2011.

[3] Alberto Apostolico, Dany Breslauer, and Zvi Galil. Parallel detection of all palindromes in a string. *Theoretical Computer Science*, 141(12):163 – 173, 1995.

[4] Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling from a moving window over streaming data. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 633–634, 2002.

[5] Valérie Berthé and Laurent Vuillon. Palindromes and two-dimensional sturmian sequences. 6(2):121–138, 2001.

[6] Lakshminath Bhuvanagiri and Sumit Ganguly. Estimating entropy over data streams. In *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, pages 148–159, 2006.

[7] Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Optimal sampling from sliding windows. *J. Comput. Syst. Sci.*, 78(1):260–272, 2012.

[8] Dany Breslauer and Zvi Galil. Real-time streaming string-matching. In Raffaele Giancarlo and Giovanni Manzini, editors, *Combinatorial Pattern Matching*, volume 6661 of *Lecture Notes in Computer Science*, pages 162–172. Springer Berlin Heidelberg, 2011.

[9] Amit Chakrabarti. CS85: Data Stream Algorithms (Lecture Notes), December 2009. Lecture Notes, Dartmouth College.

[10] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:62, 2011.

[11] Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 107–117, 2003.

[12] Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 30–39, 2003.

[13] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[14] Graham Cormode, S. Muthukrishnan, Ke Yi, and Qin Zhang. Continuous sampling from distributed streams. *J. ACM*, 59(2):10, 2012.

[15] Nick G. Duffield, Carsten Lund, and Mikkel Thorup. Priority sampling for estimation of arbitrary subset sums. *J. ACM*, 54(6), 2007.

[16] Funda Ergun, Hossein Jowhari, and Mert Sağlam. Periodicity in streams. In *Proceedings of the 13th international conference on Approximation, and 14 the International conference on Randomization, and combinatorial optimization: algorithms and techniques*, APPROX/RANDOM'10, pages 545–559, Berlin, Heidelberg, 2010. Springer-Verlag.

[17] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the streaming model: the value of space. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 745–754, 2005.

[18] Joan Feigenbaum, Sampath Kannan, Martin Strauss, and Mahesh Viswanathan. An approximate l1-difference algorithm for massive data streams. *SIAM J. Comput.*, 32(1):131–151, 2002.

[19] Joan Feigenbaum, Sampath Kannan, and Jian Zhang. Computing diameter in the streaming and sliding-window models. *Algorithmica*, 41(1):25–41, 2005.

[20] Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *Int. J. Comput. Geometry Appl.*, 18(1/2):3–28, 2008.

[21] Gereon Frahling and Christian Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 209–217, 2005.

[22] Zvi Galil. String matching in real time. *J. ACM*, 28(1):134–149, January 1981.

[23] Malgorzata Giel-Pietraszuk, Marcin Hoffmann, Sylwia Dolecka, Jacek Rychlewski, and Jan Barciszewski. Palindromes in proteins. *Journal of Protein Chemistry*, 22(2):109–113, 2003.

[24] Anna C. Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 389–398, 2002.

[25] Sudipto Guha and Andrew McGregor. Approximate quantiles and the order of the stream. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 273–279, 2006.

[26] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Trans. Knowl. Data Eng.*, 15(3):515–528, 2003.

[27] Dan. Gusfield. Cambridge University Press, 1997.

[28] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

[29] Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 202–208, 2005.

[30] Johan Jeuring. The derivation of on-line algorithms, with an application to finding palindromes. *Algorithmica*, 11(2):146–184, 1994.

[31] Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 49–58, 2011.

[32] Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 745–754, 2011.

[33] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, March 1987.

[34] D. Knuth, J. Morris, Jr., and V. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[35] Glenn Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *J. ACM*, 22(3):346–351, July 1975.

[36] Andrew McGregor. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, pages 170–181, 2005.

[37] Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014.

[38] Morteza Monemizadeh and David P. Woodruff. 1-pass relative-error l_p-sampling with applications. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1143–1160, 2010.

[39] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. Cambridge University Press, New York, NY, USA, 1995.

[40] S. Muthukrishnan. Data Streams: Algorithms and Applications.

[41] S. Muthukrishnan and Martin Strauss. Rangesum histograms. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 233–242, 2003.

[42] S. Ohno. Intrinsic evolution of proteins. the role of peptidic palindromes. *Riv. Biol.*, 83:297–291, 1990.

[43] Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '09, pages 315–323, Washington, DC, USA, 2009. IEEE Computer Society.

[44] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.

[45] Michael Zuker and David Sankoff. Rna secondary structures and their prediction. *Bulletin of Mathematical Biology*, 46(4):591–621, 1984.