Longest Palindromic Substring in Sublinear Time

Panagiotis Charalampopoulos
Reichman University, Herzliya, Israel

Solon P. Pissis
CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands

Jakub Radoszewski
Institute of Informatics, University of Warsaw, Poland

Abstract
We revisit the classic algorithmic problem of computing a longest palindromic substring. This problem is solvable by a celebrated $O(n)$-time algorithm [Manacher, J. ACM 1975], where $n$ is the length of the input string. For small alphabets, $O(n)$ is not necessarily optimal in the word RAM model of computation: a string of length $n$ over alphabet $[0, \sigma)$ can be stored in $O(n \log \sigma/\log n)$ space and read in $O(n \log \sigma/\log n)$ time. We devise a simple $O(n \log \sigma/\log n)$-time algorithm for computing a longest palindromic substring. In particular, our algorithm works in sublinear time if $\sigma = 2^{o(\log n)}$. Our technique relies on periodicity and on the Kempa and Kociumaka [STOC 2019] that answers longest common extension queries in $O(1)$ time.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases string algorithms, longest palindromic substring, longest common extension

1 Introduction

We start with some basic definitions and notation. Let $S = S[0] \cdots S[n-1]$ be a string of length $n = |S|$ over an alphabet $\Sigma$ of $\sigma$ letters. We consider throughout an integer alphabet $\Sigma = [0, \sigma) \subseteq [0, n)$. The empty string is the unique string of length 0. For any two positions $i$ and $j \geq i$ of $S$, $S[i..j]$ is the fragment of $S$ starting at position $i$ and ending at position $j$; it is represented in $O(1)$ space by $i$ and $j$. The fragment $S[i..j]$ is an occurrence of the underlying substring $P = S[i] \cdots S[j]$; we say that $P$ occurs at position $i$ in $S$. A fragment $S[i..j]$ can be equivalently written as $S[i..j+1]$, $S[i-1..j]$, or $S[i-1..j+1]$. A prefix of $S$ is a fragment of the form $S[0..j]$ and a suffix of $S$ is a fragment of the form $S[i..n)$. A substring of $S$ is proper when it does not equal $S$. By $ST$ we denote the concatenation of two strings $S$ and $T$. We denote the reverse string of $S$ by $S^R$, i.e., $S^R = S[n-1] \cdots S[0]$. A palindrome is a symmetric word that reads the same backward and forward. Formally, a string $S$ is said to be a palindrome if and only if $S = S^R$.

In this work, we consider the classic algorithmic problem of computing a longest palindromic substring.
Longest Palindromic Substring in Sublinear Time

**Longest Palindromic Substring**

**Input:** A string $S$ of length $n$ over an integer alphabet $[0, \sigma]$ with $\sigma \leq n$.

**Output:** Positions $i, j \in [0, n)$ such that $S[i..j]$ is a longest palindromic substring of $S$.

**Longest Palindromic Substring** can be solved in $O(n)$ time by Manacher’s celebrated algorithm [34, 3], by Jeuring’s algorithm [31] or by Gusfield’s simple algorithm, which uses longest common extension queries [28]. Other settings in which the problem has been studied include the compressed setting, where the input string is given as a straight-line program [36], the streaming setting [26], the dynamic setting, where the string undergoes updates [1, 2], and a semi-dynamic setting [23]. Le Gall and Seddighin [24] have recently presented a strongly sublinear-time quantum algorithm for the problem and a quantum lower bound.

The detection of palindromes is a well-studied problem with a lot of variants [29, 30, 19, 25, 5, 22, 12, 41, 40, 39] arising out of different practical scenarios. For instance, in computational biology, palindromes are found in both prokaryotic and eukaryotic genomes and they have been linked with countless possible functions. They play an important role in the regulation of gene activity and other cell processes because these are often observed near promoters, introns, and specific untranslated regions; for more details see [38, 13, 18, 42, 43, 35].

**Our Model and Result**

The main contribution of our work is to improve on the existing linear-time solutions to **Longest Palindromic Substring** in the word RAM model of computation when the input string is given in a packed representation. Let us now describe this model in more detail.

We assume the unit-cost word RAM model with word size $w = \Theta(\log n)$ and a standard instruction set including arithmetic operations, bitwise Boolean operations, and shifts. We count the space complexity of our algorithms in machine words used by the algorithm. The packed representation of a string $S$ over an integer alphabet $[0, \sigma)$ is a list obtained by storing $\Theta(\log_\sigma n)$ letters per machine word thus representing $S$ in $O(|S|/\log_\sigma n)$ machine words. If $S$ is given in the packed representation we simply say that $S$ is a packed string.

We prove the following result.

**Theorem 1.** **Longest Palindromic Substring** can be solved in $O(n/\log_\sigma n)$ time, if the input is given in a packed representation.

In Section 2 we provide the necessary background. In Section 3 we recall the linear-time algorithm for solving **Longest Palindromic Substring** by Gusfield [28]. We provide our sublinear-time algorithm in Section 4 and conclude in Section 5.

**Other Related Work**

A large body of work exploits bit-level parallelism in the word RAM model to speed-up string matching algorithms; see [4, 37, 21, 9, 6, 10, 7, 14, 27, 8, 11, 15] and references therein.

**2 Preliminaries**

**Palindromes.** Let $S$ be a string of length $n$. If $S[i..j]$, $0 \leq i \leq j < n$, is a palindrome, the number $\frac{i+j}{2}$ is called the center of $S[i..j]$ and the number $\frac{i+j+1}{2}$ is called the radius of $S[i..j]$. A palindromic fragment $S[i..j]$ of $S$ is said to be a maximal palindrome if there is no longer palindrome in $S$ with center $\frac{i+j}{2}$. Note that a maximal palindrome of $S$ can be a fragment of another palindrome of $S$ and that the longest palindrome in $S$ must be maximal.
Periodicity. A positive integer $p$ is called a period of a string $S$ if $S[i] = S[i + p]$ for all $i \in [0, |S| - p)$. We refer to the smallest period as the period of the string, and denote it by $\text{per}(S)$. A string $S$ is called periodic if $2 \cdot \text{per}(S) \leq |S|$. A border of a nonempty string $S$ is a proper substring of $S$ that occurs both as a prefix and as a suffix of $S$. A string $S$ has a period $p$ if and only if it has a border of length $|S| - p$.

Lemma 2 (Periodicity Lemma (weak version) [20]). If a string $S$ has periods $p$ and $q$ such that $p + q \leq |S|$, then $\gcd(p, q)$ is also a period of $S$.

Let $B(S)$ denote the set of lengths of borders of $S$. The following characterization of long borders of a string is generally known; cf. [17]. We give a proof of the lemma for completeness.

Lemma 3. Assume that a string $S$ of length $n$ is periodic with smallest period $p$. Then $B(S) \cap [p, n] = \{n - kp : k \in \mathbb{Z}^+ \} \cap [p, n]$.

Proof. ($\subseteq$) If $b \in B(S)$, then $q = n - b$ is a period of $S$. As $p$ is a period of $S$ as well, if $b \geq p$, then by the Periodicity Lemma $\gcd(p, q)$ is also a period of $S$. This means that $p$ divides $q$, as otherwise $\gcd(p, q)$ would have been a period of $S$ smaller than $p$, which is impossible.

($\supseteq$) For each integer $k \in [0, n/p)$, the string $S$ has a period $kp$ and hence a border of length $n - kp$.

Longest Common Extension. An important building block of our technique is a so-called longest common extension data structure, first used by Landau and Vishkin in their textbook solution for approximate pattern matching with at most $k$ mismatches [33]. Let us denote the lengths of the longest common prefix and the longest common suffix of two strings $U$ and $V$ by $\text{LCP}(U, V)$ and $\text{LCS}(U, V) = \text{LCP}(U^R, V^R)$ respectively. Given a string $S$, it is often useful to have a data structure that can efficiently return $\text{LCP}(S[0..i], S[0..j])$ or $\text{LCS}(S[0..i], S[0..j])$; we collectively call such queries longest common extension (LCE) queries. Kempa and Kociumaka presented an optimal LCE data structure for packed strings.

Theorem 4 ([32, Theorem 5.4]). Given a packed representation of a string $S \in [0, \sigma]^n$, LCE queries on $S$ can be answered in $O(1)$ time after $O(n/\log n)$-time preprocessing.

3 LCE-based Linear-Time Algorithm

We describe the linear-time algorithm given by Gusfield for LONGEST PALINDROMIC SUBSTRING [28]. Gusfield’s algorithm is based on the following simple fact — its proof follows by the definition of palindromes and by the definition of $\text{LCP}(U, V)$ for two strings $U, V$.

Fact 5. Let $S$ be a string of length $n$. $S[i..j]$ is a palindrome of odd length with center $c = \frac{i + j}{2}$ if and only if $\text{LCP}(S[c + 1..n], S[0..c - 1])^R \geq \frac{i + j}{2}$. $S[i..j]$ is a palindrome of even length with center $c = \frac{i + j}{2}$ if and only if $\text{LCP}(S[c..n], S[0..c])^R \geq \frac{i + j}{2}$.

Thus, after constructing an LCE data structure for string $T = SS^R$, it suffices to perform $O(n)$ LCP queries: one for each integer or half-integer possible center in $[0, n)$. By using any LCE data structure, which is constructible in $O(n)$ time and answers LCP queries in $O(1)$ time, such as the one by Landau and Vishkin [33], we obtain a linear-time solution to LONGEST PALINDROMIC SUBSTRING; in fact this algorithm computes all maximal palindromes.
4 Computing a Longest Palindromic Substring in Sublinear Time

The main goal of this section is to prove Theorem 1; namely, to design an algorithm for LONGEST PALINDROMIC SUBSTRING that works in $O(n/\log_\sigma n)$ time. Recall that our input is a string $S$ of length $n$ over alphabet $[0, \sigma)$. Let us set $\ell' = \max(1, \lceil \frac{1}{2}\log_\sigma n \rceil)$ and $\ell = 4\ell'$. Intuitively, $\ell'$ and $\ell$ correspond to lengths of chunks and extended chunks of $S$, respectively. Our algorithm proceeds with processing each chunk separately. We assume that $n \geq 8$.

Preprocessing. We compute the radii of maximal palindromes with each possible center for every distinct length-$\ell$ string over $[0, \sigma)$. The number of such length-$\ell$ strings is $\sigma^\ell = \sigma^{4\ell'} = O(\sqrt{n})$ and each of them can be stored in one machine word. All the radii can be computed using Manacher’s algorithm [34] in $O(\ell)$ time per string, which takes $O(\ell\sqrt{n}) = o(n/\log_\sigma n)$ time overall. In the end of the preprocessing step, we store, in an $O(\sqrt{n})$-sized array, for each length-$\ell$ string $X$, a constant amount of data:

(a) a longest palindrome in $X$;
(b) a longest palindrome in $X$ that has its center in $[\ell/2 - \ell', \ell/2 - 1/2]$; and
(c) the two longest prefix palindromes of $X$, if they exist.

Algorithm. The precomputed data allows us to compute the longest palindrome in the length-$\ell$ prefix and in the length-$\ell$ suffix of $S$. This will account for the longest palindrome with the center in the first and last $\ell/2$ positions of $S$. Let us partition $S[\ell/2..n - \ell/2)$ into chunks of length $\ell'$; if the final chunk has length smaller than $\ell'$, we complete it to a length-$\ell'$ string by taking letters of $S$ preceding it. Our goal is to compute, for each chunk $C$, the longest palindrome in the whole string $S$ with a center in $C$; let us note that this palindrome may be much longer than chunk $C$, as its length may even be $\Theta(n)$. We assume that a chunk $S[i..i + \ell')$ includes all centers in $[i, i + \ell' - 1/2]$, consistently with Item b above.

For each chunk $C$, we consider the length-$\ell$ fragment $X$ (extended chunk) of $S$ such that $C$ is the second quarter of $X$, i.e., $C$ is a suffix of $X[0..\ell/2)$. Let $\mathcal{P}_X$ denote the set of maximal palindromes in $S$ with centers in $C$ that either exceed $X$ or are prefixes of $X$ (inspect Figure 1 for an illustration). We will show that the longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $O(1)$ LCP queries on substrings of $SS^R$.

![Figure 1](image)

**Figure 1** Two of the possible palindromes from the set $\mathcal{P}_X$.

Using the packed representation of $S$, we can recover the string $X$ packed into one machine word in constant time with word RAM operations. Using the precomputed data for $X$, we know the at most two longest prefix palindromes $P_1, P_2$ of $X$; we assume that $|P_1| > |P_2|$. If any $P_i, i \in \{1, 2\}$, satisfies $|P_i| \leq \ell - 2\ell'$, we discard it, as the center of the occurrence of this palindrome as a prefix of $X$ does not lie in $C$. Let $\mathcal{Q}_X$ be the set of palindromes which are prefixes of $X$ of length greater than $\ell - 2\ell'$. Let us note that each of $P_1$ and $P_2$ that was not discarded belongs to $\mathcal{Q}_X$. Each palindrome $P \in \mathcal{P}_X$ has a subpalindrome (palindromic substring) $P' \in \mathcal{Q}_X$ with the same center. If $P_1$ does not exist, then $\mathcal{P}_X = \emptyset$. If $P_1$ exists but $P_2$ does not, then $|\mathcal{P}_X| = 1$. In this case, we can apply Fact 5 to compute the only palindrome in $\mathcal{P}_X$ from $P_1$ using one LCP query on suffixes of $SS^R$. 

Finally, we consider the case where both $P_1$ and $P_2$ exist. Here we use the following well-known property of palindromes.

**Lemma 6** (cf. [19, Lemma 3]). Let $U$ be a proper prefix of a palindrome $V$. Then $|V| - |U|$ is a period of $V$ if and only if $U$ is a palindrome. In particular, $\text{per}(V) = |V| - |U|$ if and only if $U$ is the longest palindromic proper prefix of $V$.

Let $p := |P_1| - |P_2|$. By Lemma 6, $p = \text{per}(P_1)$. We can check how far this periodicity extends on both sides by using two LCE queries. Namely, if $X = S[i..i + \ell]$, the maximal fragment with period $p$ that contains $P_1$ is $S[i-a..i+b]$, where $a := \text{LCS}(S[0..i], S[0..i+p])$ and $b := p + \text{LCP}(S[i..n], S[i+p..n])$. We next provide a characterization of the lengths of the palindromes in $Q_X$.

**Lemma 7.** Let $P_1$ be the longest prefix palindrome in $Q_X$. Further, let $p = \text{per}(P_1)$. The set of lengths of prefix palindromes in $Q_X$ is $L := \{|P_1| - kp : k \geq 0\} \cap (\ell - 2\ell', \ell]$.

**Proof.** ($\subseteq$) Let $Q \in Q_X$. We have $p < \ell - (\ell - 2\ell') = 2\ell'$ and $|Q| > \ell - 2\ell'$, so $|Q| - p > \ell - 4\ell' = 0$. By Lemma 6, $Q$ is a border of $P_1$, so by Lemma 3, $|Q| \in L$.

($\supseteq$) For each $k$ such that $|P_1| - kp \in L$, the string $P_1$ has a period $kp$, hence a border of length $|P_1| - kp$. This border is a palindrome by Lemma 6.

![Figure 2](image_url) Configuration in Lemmas 7 and 8 for $\ell' = 8$ and $\ell = 4\ell' = |X|$. The prefix palindromes in $Q_X$ are denoted by red arrows; the maximal palindromes in $P_X$ are denoted by black arrows. The fragment $S[i-a..i+b]$ is shaded in blue.

The periodicity of the elements of $Q_X$ enables an efficient computation of the longest palindrome in $P_X$. For intuition, consider answering $\text{LCP}(S[c..n], (S[0..c])^R)$ (see Fact 5), for some half-integer $c \in [i-a..i+b] \setminus \mathbb{Z}$, by comparing pairs of letters: either we reach $i - a$ and $i + b - 1$ at the same time, which can happen for at most a single value of $c$, namely for $(i-a+i+b-1)/2 = (2i-a+b-1)/2$, or we reach one of the two endpoints first, in which case we get a mismatch (inspect Figure 2).

**Lemma 8.** Let $P_1$ be the longest prefix palindrome in $Q_X$. Further let $P_1 = S[i..i + |P_1|]$, $p = \text{per}(P_1)$, $a = \text{LCS}(S[0..i], S[0..i + p])$ and $b := p + \text{LCP}(S[i..n], S[i + p..n])$. For palindromes $P \in P_X$ and $Q \in Q_X$ with the same center $c$, either $|Q| = b - a$ or $|P| = \min(|Q| + 2a, 2b - |Q|)$.

**Proof.** Let us first consider the case where $Q$ and $P$ are even-length palindromes. Note that $[c] = i + |Q|/2$ and recall that $|Q| \geq 2p$. Let $F := S[c..[c] + p]$. We have

$$\lambda := \text{LCP}(S[0..c])^R, F^n) = |S[i-a..c]| = |c| - i + a = |Q|/2 + a,$$

and

$$\rho := \text{LCP}(S[c..n], F^n) = |S[c..i + b]| = i + b - |c| = b - |Q|/2.$$
Then, if \( \lambda \neq \rho \), we have
\[
|P| = 2 \cdot \text{LCP}((S[0..|c|])^R, S[c..n]) = 2 \cdot \min\{\lambda, \rho\} = \min\{|Q| + 2a, 2b - |Q|\}.
\]

Else, we have \( \lambda = \rho \Leftrightarrow |Q|/2 + a = b - |Q|/2 \Leftrightarrow |Q| = b - a \).

The proof for the case where \( Q \) and \( P \) are odd-length palindromes is similar, but we include it for completeness. In this case, \( c = i + (|Q| - 1)/2 \). Let \( F := S[c..c + p] \). We have
\[
\lambda := \text{LCP}((S[0..c])^R, F^R) = |S[i - a..c]| = c - i + a + 1 = |Q|/2 + 1/2 + a,
\]
and
\[
\rho := \text{LCP}(S[c..n], F^R) = |S[i + b]| = i + b - c = b - |Q|/2 + 1/2.
\]

Then, if \( \lambda \neq \rho \), we have
\[
|P| = 2 \cdot \text{LCP}((S[0..c])^R, S[c..n]) - 1 = 2 \cdot \min\{\lambda - 1/2, \rho - 1/2\} = \min\{|Q| + 2a, 2b - |Q|\}.
\]

Else, we have \( \lambda = \rho \Leftrightarrow |Q| = b - a \) as before.

We use Lemma 8 to compute the longest palindrome in \( P_X \) as follows. For two palindromes \( Q \in Q_X \) and \( P \in P_X \) with the same center such that \( |Q| \neq b - a \), either \( |Q| < b - a \Leftrightarrow |Q| + 2a < 2b - |Q| \) and hence \( |P| = |Q| + 2a \) due to Lemma 8 or \( |Q| > b - a \Leftrightarrow |Q| + 2a > 2b - |Q| \) and hence \( |P| = 2b - |Q| \) due to Lemma 8. Thus, it suffices to consider only three palindromes in \( Q_X \). Specifically, with the characterization of Lemma 7 we compute in \( O(1) \) time: the longest palindrome \( Q_1 \) in \( Q_X \) of length smaller than \( b - a \); the shortest palindrome \( Q_2 \) in \( Q_X \) of length greater than \( b - a \); and check if there is a palindrome \( Q_3 \) in \( Q_X \) of length \( b - a \). Finally we pick the longest of the following palindromes from \( P_X \):

- The palindrome \( P_I \) corresponding to \( Q_1 \) if \( Q_1 \) exists; the length of \( P_I \) is \(|Q_1| + 2a |\) due to the formula from Lemma 8.
- The palindrome \( P_{II} \) corresponding to \( Q_2 \) if \( Q_2 \) exists; the length of \( P_{II} \) is \( 2b - |Q_2| \) due to the formula from Lemma 8.
- The palindrome \( P_{III} \) corresponding to \( Q_3 \) if \( Q_3 \) exists; the center of \( P_{III} \) is \( i + (|Q_3| - 1)/2 \) and hence the length of \( P_{III} \) can be computed using one LCP query on suffixes of \( SS^R \) due to Fact 5.

Thus we have proved the following lemma.

\textbf{Lemma 9.} The longest palindrome in \( P_X \) can be computed in the time required to answer \( O(1) \) LCP queries on suffixes of \( SS^R \).

For each chunk \( C \) (we have \( O(n/\ell) \) of them), we take the longer palindrome of the one computed by an application of Lemma 9 and the longest palindrome stored in Item b for the corresponding substring \( X \). Over all chunks, using Theorem 4 to answer LCE queries in \( O(1) \) time, the algorithm requires time \( O(n/\log_\sigma n) \), and we thus obtain Theorem 1.

\section{Final Remarks}

We have shown an \( O(n/\log_\sigma n) \)-time algorithm for computing a longest palindromic substring of a string of length \( n \) over alphabet \([0, \sigma]\). Our algorithm can be easily modified to compute the number of all palindromic fragments of the string within the same time complexity.

We anticipate that our technique will be applicable in many other problems on strings, which currently admit only linear-time solutions. For instance, our approach applied to the prefix array of a string [16] can be used to compute the longest repeating prefix of a string of length \( n \) over alphabet \([0, \sigma]\), still in \( O(n/\log_\sigma n) \) time.
References


