Palindromic Length in Linear Time

Kirill Borozdin[1], Dmitry Kosolobov[2], Mikhail Rubinchik[3], and Arseny M. Shur[4]

1 **Ural Federal University, Ekaterinburg, Russia**
borozdin.kirill,@gmail.com
2 **University of Helsinki, Helsinki, Finland**
dkosolobov@mail.ru
3 **Ural Federal University, Ekaterinburg, Russia**
mikhail.rubinchik@gmail.com
4 **Ural Federal University, Ekaterinburg, Russia**
arseny.shur@urfu.ru

Abstract

Palindromic length of a string is the minimum number of palindromes whose concatenation is equal to this string. The problem of finding the palindromic length drew some attention, and a few $O(n \log n)$ time online algorithms were recently designed for it. In this paper we present the first linear time online algorithm for this problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics

Keywords and phrases palindrome, palindromic length, palindromic factorization, online

Digital Object Identifier 10.4230/LIPIcs.CPM.2017.23

1 Introduction

Algorithmic and combinatorial problems involving palindromes attracted the attention of researchers since the first days of stringology. Recall that a string $w = a_0a_1 \cdots a_{n-1}$ is a *palindrome* if it is equal to the string $\overleftarrow{w} = a_{n-1} \cdots a_1a_0$. The early works [4, 6, 8, 11] considered palindromes as structures that might provide examples of (context-free) languages that are impossible to recognize in linear time, thus provably restricting the computational power of some models (RAM, in particular). Subsequently, it was shown that many of such languages are, in fact, linear recognizable. Recently it was proved [7] that the language $\mathcal{P}^k$, where $\mathcal{P}$ is the set of all palindromes on a given alphabet, is recognizable online in $O(kn)$ time, where n is the length of the input string. Roughly at the same time, a closely related notion of *palindromic length* of a string was introduced: this is the minimal number k such that the string belongs to $\mathcal{P}^k$. In 2014–2015 three different algorithms that compute the palindromic length of a string of length n in $O(n \log n)$ time were presented in [3, 5, 10] (however, they all are based on similar principles). In this paper we present the first linear algorithm computing the palindromic length. Moreover, our algorithm is *online*, i.e., it reads the input string sequentially from left to right and computes the palindromic length for each prefix after reading the rightmost letter of that prefix. Thus, we prove the following theorem.

▶ **Theorem 1.** *Palindromic length of a string is computable online in linear time.*

The implementation of our algorithm and tests for it can be found in [9]. Due to a large constant under the big-O, it is slower in practice (for 32/64 bit machine words) than the existing $O(n \log n)$ solutions; the fastest algorithm is the one of [10].

The paper is organized as follows. Section 2 contains a high-level description of the algorithm: it starts with a naive $O(n^2)$ algorithm, then improves the time to $O(n \log n)$, and, finally, describes on a high level a modified $O(n)$-time version of the $O(n \log n)$ algorithm. In Section 3 we discuss the main components of the linear algorithm in details.

1.1 Preliminaries

Let w be a string of length $n = |w|$. We write $w[i]$ for the ith letter of w ($i = 0, \ldots, n-1$) and $w[i..j]$ for $w[i]w[i+1] \cdots w[j]$. A string u is a *substring* of w if $u = w[i..j]$ for some i, j. Such pair (i, j) is not necessarily unique; i specifies an *occurrence* of u at *position* i. A substring $w[0..j]$ (resp., $w[i..n-1]$) is a *prefix* (resp. *suffix*) of w. The *empty string* is denoted by ε. For any i, j, $[i..j]$ denotes the set $\{k \in \mathbb{Z} \colon i \le k \le j\}$; let $(i..j] = [i..j] \setminus \{i\}$, $[i..j) = [i..j] \setminus \{j\}$, $(i..j) = [i..j) \cap (i..j]$. Our notation for arrays is the same as for strings.

A substring (resp. suffix, prefix) that is a palindrome is called a *subpalindrome* (resp. *suffix-palindrome*, *prefix-palindrome*). If $w[i..j]$ is a subpalindrome of w, then the number $(j + i)/2$ is the *center* of $w[i..j]$ and the number $\lfloor (j - i + 1)/2 \rfloor$ is the *radius* of $w[i..j]$. The following remarkable property of palindromic lengths is crucial for our algorithm.

▶ **Lemma 2** (see [10, Lemma 11]). *Denote by $\ell_0, \ell_1, \ldots, \ell_{n-1}$, resp., the palindromic lengths of the prefixes $w[0..0], w[0..1], \ldots, w[0..n-1]$ of a string w. Then, for any $i \in (0..n)$, $|\ell_i - \ell_{i-1}| \le 1$.*

An integer p is a *period* of w if $w[i] = w[i+p]$ for any $i \in [0..n-p)$. As the previous results [3, 5, 10], our approach relies on a number of periodic properties of palindromes.

▶ **Lemma 3** (see [7, Lemmas 2, 3]). *For any palindrome w and any $p \in (0..|w|]$, the following conditions are equivalent: (1) p is a period of w, (2) there are palindromes u, v such that $|uv| = p$ and $w = (uv)^k u$ for some $k \ge 1$, (3) $w[p..|w|-1]$ ($w[0..|w|-p-1]$) is a palindrome.*

▶ **Lemma 4** (see [7, Lemma 7]). *Suppose that $w = (uv)^k u$ for $k \ge 1$ and for palindromes u and v such that $|uv|$ is the minimal period of w; then, the center of any subpalindrome x of w such that $|x| \ge |uv| - 1$ coincides with the center of some u or v from the decomposition.*

Henceforth, let s denote the input string of length n. We assume that the algorithm works in the unit-cost word-RAM model with $\Theta(\log n)$-bit machine words (an assumption justified in, e.g., [2]) and standard operations like in the C programming language.

2 High-Level Description of the Algorithm

Our aim is to maintain an array $\mathsf{ans}[0..n-1]$ in which each element $\mathsf{ans}[i]$ is the palindromic length of $s[0..i]$. We always assume n to be the length of the string s processed so far (i.e., $s = s[0..n-1]$). Processing the next letter $s[n]$, we compute $\mathsf{ans}[n]$ and then increment n.

2.1 Naive approach

An easy quadratic-time approach is to maintain the list of all non-empty suffix-palindromes $u_1, \ldots, u_k$ of the string s and calculate $\mathsf{ans}[n] = 1 + \min_{i \in [1..k]} \mathsf{ans}[n - |u_i|]$. The list can be updated in linear time: the suffix-palindromes of wa have the form aua, where u is a suffix palindrome of w, plus the palindrome a and, optionally, aa. As a first speedup to this basic approach, we utilize the (palindromic) *iterator*, introduced in [7]; this data structure contains a string s and supports the following operations:

1. $\mathsf{add}(a)$ appends the letter a to the end of s;
2. maxPal returns the center of the longest suffix-palindrome of s;
3. $\mathsf{rad}(x)$ returns the radius of the longest subpalindrome of s with the center x;
4. $\mathsf{nextPal}(x)$ returns the center of the longest proper suffix-palindrome of the suffix-palindrome of s with the center x; undefined if x is not the center of a suffix-palindrome.

The iterator can be implemented so that all its operations work in $O(1)$ time (amortized, for add) [7, Prop. 1]. The same time bound applies to computing length of the longest subpalindrome centered at x: $\mathsf{len}(x) = 2 \cdot \mathsf{rad}(x) + \lfloor x \rfloor - \lfloor x - \frac{1}{2} \rfloor$. Still, the iterator alone cannot lower the asymptotic time of the naive algorithm; its improved version looks as follows:

1: $\mathsf{add}(s[n])$; $\mathsf{ans}[n] \leftarrow +\infty$
2: **for** $(x \leftarrow \mathsf{maxPal};\ x \neq n + \frac{1}{2};\ x \leftarrow \mathsf{nextPal}(x))$ **do** $\mathsf{ans}[n] \leftarrow \min\{\mathsf{ans}[n], 1 + \mathsf{ans}[n - \mathsf{len}(x)]\}$

2.2 Algorithm working in $O(n \log n)$ time

All subquadratic algorithms for palindromic length heavily use grouping of suffix-palindromes into *series*. Let $u_1, \ldots, u_k$ be all non-empty suffix-palindromes of a string s in the order of decreasing length. Since u_j is a suffix of u_i for any $i < j$, any period of u_i is a period of u_j; hence the sequence of minimal periods of $u_1, \ldots, u_k$ is non-increasing. The groups of suffix-palindromes with the same minimal period are *series of palindromes* (of s):

$$\underbrace{u_1, \ldots, u_{i_1}}_{p_1}, \underbrace{u_{i_1+1}, \ldots, u_{i_2}}_{p_2}, \ldots, \underbrace{u_{i_{t-1}+1}, \ldots, u_k}_{p_t}.$$

We refer to the longest and the shortest palindrome in a series as its *head* and *baby* respectively (they coincide in the case of a 1-element series); we enumerate the elements of a series from the head to the baby. Given an integer p, the *p-series* is the series with period p. A very useful observation [3, 5, 7] is that the length of a head is multiplicatively smaller than the length of the baby from the previous series, and thus every string of length n has $O(\log n)$ series. (As it was shown in [3], strings with $\Omega(\log n)$ series for $\Omega(n)$ prefixes do exist.)

The idea of the $O(n \log n)$ solution is to use the dynamic programming rule $\mathsf{ans}[n] = 1 + \min_U \min_{u \in U} \mathsf{ans}[n - |u|]$, where U runs through the series of s, and compute the internal minimum in $O(1)$ time using precalculations based on the structure of series. The structure of any series is described in the following lemma, which is easily implied by Lemmas 3, 4.

▶ **Lemma 5.** *For a string s and $p \geq 1$, let U be a p-series of palindromes. There exist $k \geq 1$ and unique palindromes u, v with $|uv| = p$, $v \neq \varepsilon$ such that one of three conditions hold:*

- *$U = \{(uv)^{k+1}u, (uv)^k u, \ldots, (uv)^2 u\}$ and the next series begins with uvu,*
- *$U = \{(uv)^k u, (uv)^{k-1}u, \ldots, uvu\}$ and the next series begins with u,*
- *$U = \{v^k, v^{k-1}, \ldots, v\}$, $p = 1$, $|v| = 1$, $u = \varepsilon$, and U is the last series for s.*

Let U be a p-series for $s[0..n]$ with $k > 1$ palindromes (w.l.o.g., $U = \{(uv)^k u, \ldots, uvu\}$). Updating $\mathsf{ans}[n]$ using this series, we compute $m = \min\{\mathsf{ans}[n-kp-|u|], \ldots, \mathsf{ans}[n-p-|u|]\}$. Now note that $s[0..n]$ ends with $(uv)^k u$ but not with $(uv)^{k+1}u$: otherwise, the latter string would belong to U. Then $s[0..n-p]$ ends with $(uv)^{k-1}u$ but not with $(uv)^k u$ and thus has the p-series $U' = \{(uv)^{k-1}u, \ldots, uvu\}$. Thus, at that iteration we computed $m' = \min\{\mathsf{ans}[n-kp-|u|], \ldots, \mathsf{ans}[n-2p-|u|]\}$ for updating $\mathsf{ans}[n-p]$. If we save m' into an auxiliary array, then $m = \min\{m', \mathsf{ans}[n - p - |u|]\}$ is computable in constant time, as required. Let us implement this construction using the iterator.

We start an iteration calling $\mathsf{add}(s[n])$. Let x be the center of a suffix-palindrome u. By Lemma 3, the minimal period p of u equals $\mathsf{len}(x) - \mathsf{len}(\mathsf{nextPal}(x))$. Let $\mathsf{cntr}(d)$ denote

the center of the length d suffix-palindrome of $s[0..n]$ (i.e., $\mathsf{cntr}(d) = n - (d-1)/2$). Let $x' = \mathsf{cntr}(p + (\mathsf{len}(x) \bmod p))$. By Lemma 5, x' is the center either of the baby of the p-series or of the head of the next series, depending on the period $\mathsf{len}(x') - \mathsf{len}(\mathsf{nextPal}(x'))$ of this suffix-palindrome. All these computations take $O(1)$ amortized time using the iterator.

Our algorithm maintains an array $\mathsf{left}[1..n]$: for $p \in [1..n]$, if there is a p-series, then $s[\mathsf{left}[p]{+}1..n]$ is the longest suffix (which is not necessarily a palindrome) of $s[0..n]$ with period p; otherwise, $\mathsf{left}[p]$ is undefined. E.g., if $s[0..n] = \cdots aaabaaba$ and $p = 3$, then the mentioned suffix is $s[n{-}6..n] = aabaaba$ and $\mathsf{left}[3] = n-7$. Computing $\mathsf{left}[p]$ in $O(1)$ time is done as follows. Let $w = (uv)^k u$ be the head of the p-series (see Lemma 5), x be the center of w, and $z(uv)^k u$ be the longest suffix of $s[0..n]$ with period p (in our example, $u = \varepsilon$, $v = aba$, $x = n - 5/2$, $z = a$). Then z is a proper suffix of uv. Hence $\mathsf{len}(x_1) = 2|z| + |u|$, where x_1 is the center of the prefix-palindrome u of w (in the example, $x_1 = n - 11/2$, $\mathsf{len}(x_1) = |aa|$). Note that $|u| = \mathsf{len}(x) \bmod p$ and $x_1 = 2x - x_2$, where $x_2 = \mathsf{cntr}(\mathsf{len}(x) \bmod p)$ is the center of the suffix u of w. Thus, $|z|$ and $\mathsf{left}[p] = n{-}\mathsf{len}(x){-}|z|$ are computed in $O(1)$ time.

All precalculated minimums are stored in an array $\mathsf{pre}[1..n]$, where each $\mathsf{pre}[p]$ is, in turn, an array $\mathsf{pre}[p][0..p{-}1]$ (we discuss in the next subsection why only $O(n)$ of possible $O(n^2)$ elements of pre are actually stored). For each j such that $n - j > \mathsf{left}[p]$, the string $s[0..n{-}j]$ usually has a suffix-palindrome with period p and thus can have a p-series; the array $\mathsf{pre}[p][0..p{-}1]$ contains the precalculations made for all these series. Formally,

$$\mathsf{pre}[p][i] = \min\{\mathsf{ans}[t]\colon (t - \mathsf{left}[p]) \bmod p = i \text{ and } s[t{+}1..n] \text{ has a prefix-palindrome of minimal period } p\};$$

$\mathsf{pre}[p][i]$ is undefined if there is no such t (i.e., no p-series for the corresponding string). So if $u_1, \ldots, u_k$ is a p-series for $s[0..n]$, then $\mathsf{pre}[p][n{-}|u_1|{-}\mathsf{left}[p]] = \min\{\mathsf{ans}[n{-}|u_i|]\colon i \in [1..k]\}$. Hence, given a new letter $s[n]$, we compute $\mathsf{ans}[n]$ as follows:

```
1: add(s[n]); ans[n] ← +∞;
2: for (x ← maxPal; x ≠ n + 1/2; x ← nextPal(cntr(d))) do          ▷ goes to next head each time
3:     p ← len(x) − len(nextPal(x));                    ▷ min. period of the suf.-pal. centered at x
4:     d ← p + (len(x) mod p);                                  ▷ length of the baby in the p-series
5:     if len(cntr(d)) − len(nextPal(cntr(d))) ≠ p then d ← d + p;          ▷ corrected length
6:     compute left[p];                                                ▷ in O(1) time, see above
7:     if len(x) = d then pre[p][n−len(x)−left[p]] ← ans[n−d];
8:     pre[p][n−len(x)−left[p]] ← min{pre[p][n−len(x)−left[p]], ans[n−d]};
9:     ans[n] ← min{ans[n], 1 + pre[p][n−len(x)−left[p]]};
```

Let $u_1, \ldots, u_k$ be a p-series, $i = n{-}|u_1|{-}\mathsf{left}[p]$. If $k = 1$, there was no p-series p iterations ago, so we set the undefined value $\mathsf{pre}[p][i]$ to $\mathsf{ans}[n{-}|u_k|]$ in line 7. Otherwise, by the definition of pre, we have $\mathsf{pre}[p][i] = \min\{\mathsf{ans}[n{-}|u_1|], \ldots, \mathsf{ans}[n{-}|u_{k-1}|]\}$. We update this value using $\mathsf{ans}[n{-}|u_k|]$ in line 8. So pre is correctly maintained, and the above algorithm computes the array ans in $O(n \log n)$ time due to logarithmic number of series.

2.3 Sketch of the linear algorithm

The idea of the linear solution is to perform the above log-time processing of all series of the current string not n times, but only $O(\frac{n}{\log n})$ times during the run of the algorithm. (However, we are able to make $\Theta(n)$ calls to the iterator.) To achieve this, during the processing of a series we replace each computation of the minimum $\mathsf{ans}[n] \leftarrow \min\{\mathsf{ans}[n], 1 + z\}$, for a precomputed value z from pre, with the simultaneous computation ("prediction") for the range of values $\mathsf{ans}[n..n{+}b]$, where $b = \lfloor \frac{\log n}{8} \rfloor$: we compute in advance $\mathsf{ans}[j] \leftarrow \min\{\mathsf{ans}[j], 1 + z_j\}$

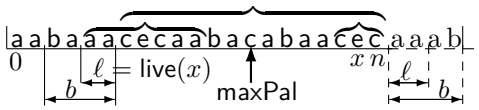

Figure 1 Predictable extensions.

for all $j \in [n..n+b]$ and corresponding precomputed z_j from pre. It is proved below that the arrays ans and pre can be organized so that, after a linear time preprocessing, such range operations on $O(b)$ elements of ans will take $O(1)$ time (this type of bit compression techniques is referred to as the four Russians' trick [1]).

Let us extend $s[0..n-1]$ with $s[n] = a$. We say that a suffix-palindrome u of $s[0..n-1]$, centered at x, *survives* if $s[0..n]$ has the suffix aua (i.e, x remains the center of a suffix-palindrome), and *dies* otherwise. We say that an extension of $s[0..n-1]$ by $s[n]$ is *predictable* if it retains maxPal, i.e., if the longest suffix-palindrome survives. From maxPal it can be calculated which of the other suffix-palindromes survive. If a suffix-palindrome of s centered at x survives $d \geq 0$ consecutive predictable extensions but dies after the $(d+1)$th such extension (or the $(d+1)$th predictable extension is not possible), we write $\mathsf{live}(x) = d$. We have $\mathsf{live}(\mathsf{maxPal}) = n - \mathsf{len}(\mathsf{maxPal})$ and $\mathsf{live}(x) = \mathsf{rad}(\mathsf{refl}(x)) - \mathsf{rad}(x)$ for $x \neq \mathsf{maxPal}$; here $\mathsf{refl}(x) = 2 \cdot \mathsf{maxPal} - x$ is the position symmetric to x w.r.t. maxPal. (See Fig. 1 for clarification; e.g., in Fig. 1 $\mathsf{live}(x) = 2$ and $\mathsf{live}(\mathsf{maxPal}) = 6$.)

Suppose that $\mathsf{ans}[n+j] = +\infty$ for $j \in [0..b]$. Having performed $\mathsf{add}(s[n])$, we get access to the suffix-palindromes of $s[0..n]$. If, for the center x of each such palindrome, we perform

$$\mathsf{ans}[n+j] \leftarrow \min\{\mathsf{ans}[n+j], 1 + \mathsf{ans}[n-\mathsf{len}(x)-j]\} \text{ for all } j \in [0..\min\{b, \mathsf{live}(x)\}], \tag{1}$$

then we accumulate all information we can obtain from these palindromes during the next b predictable extensions. Thus we get an approximation of $\mathsf{ans}[n..n+b]$, which later will be updated using suffix-palindromes with the centers $x \geq n+\frac{1}{2}$. One phase of our algorithm is roughly as follows:

- append $s[n]$ to the iterator, update precalculations, and "predict" $\mathsf{ans}[n..n+b]$ with the assignments (1), using operations on blocks of bits ($\mathsf{ans}[n]$ is computed exactly);
- append subsequent letters, each time updating the predictions with either one or two new palindromes (after processing $s[n+j]$, $\mathsf{ans}[n..n+j]$ contains correct values);
- stop after b iterations or at the moment when an unpredicted letter is encountered;
- discard unused predictions and start a new phase with the first unpredicted letter.

For arrays α, β and numbers $i, j, \ell \geq 0$, denote by $\alpha[i..i+\ell] \overset{\min}{\leftarrow} \beta[j..j+\ell]$ the sequence of assignments $\alpha[i+k] \leftarrow \min\{\alpha[i+k], \beta[j+k]\}$ for all $k \in [0..\ell]$. Let $\mathsf{increv}(i, j)$ be the function returning an array $a[0..j-i]$ such that $a[k] = 1 + \mathsf{ans}[j-k]$ for $k \in [0..j-i]$ ("increment & reverse"). The predictions are made by the function predict that uses precalculations stored in pre to perform in a fast way the assignments $\mathsf{ans}[n..n+c] \overset{\min}{\leftarrow} \mathsf{increv}(n-\mathsf{len}(x)-c, n-\mathsf{len}(x))$, where $c = \min\{b, \mathsf{live}(x)\}$, for all centers x of suffix-palindromes. (Hence predict computes the value $\mathsf{ans}[n]$ correctly even if $c = 0$ for some x.) Let precalc be a function that updates (possibly once in several iterations) the array pre to the actual state. The implementations of predict and precalc are discussed in Section 3. Our algorithm is as follows:

```
1: for (n ← 0, end ← 0; not(end_of_input); n ← n + 1) do
2:     if n = end or len(maxPal) = n or s[n] ≠ s[n−len(maxPal)−1] then        ▷ new phase
3:         add(s[n]); precalc; predict; end ← n + b
4:     else add(s[n])                          ▷ old phase continues, s[n] is predictable
5:     c ← min{b, live(n)}; ans[n..n+c] ⇐min increv(n−1−c, n−1)
6:     if s[n] = s[n−1] then c ← min{b, live(n−½)}; ans[n..n+c] ⇐min increv(n−2−c, n−2)
```

This algorithm computes the same values $\mathsf{ans}[n]$ as the $O(n \log n)$ algorithm above, because finally all suffix-palindromes of $s[0..n]$ are used. So, the algorithm is correct.

Let t be the number of series in the current string $s[0..n]$ and q is the time required to perform all the calls $\mathsf{add}(s[n]), \mathsf{add}(s[n{-}1]), \ldots, \mathsf{add}(s[n'{+}1])$, where $s[0..n']$ is the string for which $\mathsf{precalc}$ was called last time. Below we show that $\mathsf{predict}$ and $\mathsf{precalc}$ work in $O(t)$ and $O(t+q)$ time respectively, and the array ans can be organized so that the range operations in lines 5–6 can be performed in $O(1)$ time using the four Russians' trick. Let us estimate the running time of the algorithm under these assumptions.

During predictable extensions, line 3 is reached iff $n = end$, i.e., at most $O(\frac{n}{b})$ times. Since add works in $O(1)$ amortized time (see [7, Prop. 1]), the sum of all q's in the working time of $\mathsf{precalc}$ is $O(n)$. Since $O(t) = O(\log n)$, all predictable extensions take $O(n + \frac{n}{b} \log n) = O(n)$ overall time. To estimate the running time of unpredictable extensions, consider the value $\gamma_i = \mathsf{live}[\mathsf{maxPal}] = i - \mathsf{len}(\mathsf{maxPal})$ after processing $s[0..i]$. If $s[i{+}1]$ is predictable, one has $\gamma_{i+1} = (i+1) - (\mathsf{len}(\mathsf{maxPal}) + 2) = \gamma_i - 1$. If $s[i{+}1]$ is unpredictable, $\gamma_{i+1} \geq (i+1) - (\mathsf{len}(\mathsf{nextPal}(\mathsf{maxPal})) + 2)$; by Lemma 5, $\gamma_{i+1} - \gamma_i \geq p - 1$, where p is the minimal period of the longest suffix-palindrome of $s[0..i]$. By Lemmas 4 and 5, the length of the longest suffix-palindrome whose minimal period differs from p is less than $2p$. Therefore, $\mathsf{predict}$ and $\mathsf{precalc}$ take $O(p+q)$ time during this unpredictable extension (actually, $O(\log p + q)$). Since $\gamma_n - \gamma_1 < n$, the sum of the working times of all calls to $\mathsf{predict}$ and $\mathsf{precalc}$ is $O(n)$.

2.4 Organization of the arrays ans and pre

Informally, the four Russians' trick allows us to compute any operation on structures of size $\leq \varepsilon \log n$ bits in $O(1)$ time using a precomputed table of size $O(n^\varepsilon \log^{O(1)} n)$ bits. For example, let a $\lfloor \frac{\log n}{2} \rfloor$-bit integer x encode a sequence $x_1, \ldots, x_{\lfloor \log n/4 \rfloor}$ so that $x_j = 1 - (\lfloor x/2^{2j-2} \rfloor \bmod 4)$, i.e., $(2j{-}1)$th and $(2j{-}2)$th bits of x encode x_j. We can compute, for $j \in [1..\log n/4]$, the sum $x_1 + \cdots + x_j$ in $O(1)$ time using a table $T[0..\lfloor \sqrt{n} \rfloor][1..\lfloor \log n/4 \rfloor]$ such that $T[x][j] = x_1 + \cdots + x_j$ for any $x \in [0..2^{\log n/2}] = [0..\sqrt{n}]$ and $j \in [1..\log n/4]$. The table T can be precomputed in $O(\sqrt{n} \log^{O(1)} n)$ time.

In our case, we split ans into blocks of length b. By Lemma 2, adjacent elements of ans differ by at most one. This allows us to encode each block $\mathsf{ans}[ib{+}1..(i{+}1)b]$ as the number $\mathsf{ans}[ib{+}1]$ and the sequence $x_1, x_2, \ldots, x_b$ such that $x_j \in \{-1, 0, 1\}$ and $\mathsf{ans}[ib{+}j] = \mathsf{ans}[ib{+}1] + x_1 + \cdots + x_j$ for any $j \in [1..b]$. This sequence $x_1, x_2, \ldots, x_b$ is encoded in a $2b$-bit integer exactly as in the example above (note that $2b \leq \lfloor \frac{\log n}{4} \rfloor$). Using a precomputed table of size $O(\sqrt[4]{n} b)$, we can extract any element $\mathsf{ans}[j]$ in $O(1)$ time. It is shown in Sect. 3 that arrays in pre can be encoded in a similar way (with some additional complications).

Applying a similar trick, one can perform many other operations. Let $c[0..\ell]$ be an array of integers such that $\ell \in [0..b]$, $|c[i{-}1] - c[i]| \leq 1$, and c is encoded, like a block of ans, by $c[0]$ and a $2b$-bit integer. Let us show how to perform in $O(1)$ time the operation $\mathsf{ans}[n..n{+}\ell] \overset{\min}{\leftarrow} c[0..\ell]$ as in lines 5–6 of the algorithm (similar operations are also performed in $\mathsf{predict}$). We first check whether $c[0] > \mathsf{ans}[n] + 2\ell$: if so, then ans remains unchanged. It is guaranteed by the algorithm that $c[0] \geq \mathsf{ans}[n{-}1] - 1$. Then, we concatenate bit representations of all required components: the (at most) two blocks $\mathsf{ans}[ib{+}1..(i{+}1)b]$ and $\mathsf{ans}[(i{+}1)b{+}1..(i{+}2)b]$ encoding the subarray $\mathsf{ans}[n..n{+}\ell]$ are stored as two $2b$-bit sequences (encoding the differences $\mathsf{ans}[i] - \mathsf{ans}[i{-}1]$ for $i \in [ib{+}2..(i{+}2)b]$ as above), $c[0..\ell]$ is also stored as a $2b$-bit sequence, the offset $(n{-}ib)$ and the difference $c[0] - \mathsf{ans}[n{-}1]$ are stored as $O(\log b)$-bit integers; $6b + O(\log b)$ bits in total. We precompute a table T that, for a given

combined bit representation, stores two $2b$-bit sequences encoding two blocks that represent the resulting modified $\mathsf{ans}[n..n+\ell]$ array. It should be noted that the information provided in the given representation suffices to compute the result and, since the resulting array ans satisfies Lemma 2, we may put $\mathsf{ans}[n+\ell+j] = \min\{\mathsf{ans}[n+\ell+j], \mathsf{ans}[n+\ell]+j\}$ for $j \geq 1$ so that the structure of the last block is preserved. Since $6b \leq \frac{3}{4}\log n$, the size of the table T is $O(b \cdot 2^{6b+O(\log b)}) = O(n^{3/4}\log^k n)$ for some $k = O(1)$. Obviously, T can be precomputed in $O(n^{3/4}\log^{k+O(1)} n)$ time. Analogously, we precompute tables that allow us to calculate $\mathsf{increv}(i,j)$ in $O(1)$ time if $j - i \leq b$; the resulting array of increv is encoded, like the array c, by the first element and a $2b$-bit integer. Thus, all range operations in lines 5–6 of the algorithm can be performed in $O(1)$ time.

We use a number of different range operations on the arrays ans and pre in Section 3 but all of them are similar to the discussed ones, so we omit detailed descriptions.

3 Implementation of the Main Functions

Now it remains to describe the functions $\mathsf{predict}$ and $\mathsf{precalc}$ and prove their time complexity.

3.1 Function predict

At the beginning, the function $\mathsf{predict}$ sets $\mathsf{ans}[n+j] \leftarrow \mathsf{ans}[n-1]+j+1$ for $j \in [0..b]$. By Lemma 2, the assigned values are upper bounds for the elements of $\mathsf{ans}[n..n+b]$. The assignments are performed in $O(1)$ time using range operations. Then $\mathsf{predict}$ processes each of the t series; let us describe precisely how we process a p-series $u_1, \ldots, u_k$.

Let u, v be the palindromes described in Lemma 5, x_i be the center of u_i for $i \in [1..k]$. If $\mathsf{len}(x_i) < n - \mathsf{left}[p]$ (i.e., either $i > 1$ or $i = 1$ and u_1 is not the longest suffix of s with period p), then x_i will remain the center of a new suffix-palindrome after the appending of $s[n]$ iff $s[n] = s[n-p] = v[0]$. In this case, the period p "extends" and x_i remains the center of a suffix-palindrome with the minimal period p. In the remaining case $\mathsf{len}(x_1) = n - \mathsf{left}[p]$ (u_1 is the longest suffix with period p) x_1 will remain the center of a suffix-palindrome iff $s[n] = s[\mathsf{left}[p]]$; the period p breaks and the palindrome $s[n]u_1s[n]$ will belong to a different series.

Suppose that d upcoming predictable extensions extend the period p of the suffix $s[\mathsf{left}[p]+1..n-1]$ and the $(d+1)$st predictable extension breaks this period. It follows from the previous paragraph that the only suffix-palindrome u_i that can survive the $(d+1)$st extension (in other words, for which $\mathsf{live}(x_i) > d$) must have length $n - \mathsf{left}[p] - d$ (see Fig. 2). So if d is known, we check whether $x = \mathsf{cntr}(n-\mathsf{left}[p]-d)$ is the center of a suffix-palindrome (i.e., $\mathsf{cntr}(\mathsf{len}(x)) = x$) and, if so, we compute $\mathsf{ans}[n..n+c] \overset{\min}{\leftarrow} \mathsf{increv}(n-\mathsf{len}(x)-c, n-\mathsf{len}(x))$, where $c = \min\{b, \mathsf{live}(x)\}$, in $O(1)$ time using range operations.

Now it remains to find d and change $\mathsf{ans}[n..n+\min\{b,d\}]$ taking $u_1, \ldots, u_k$ into account. Since predictable extensions append the letters $s[n-\mathsf{len}(\mathsf{maxPal})], s[n-\mathsf{len}(\mathsf{maxPal})-1], \ldots$ to the right of s, we can approximately find d looking at the string $s[0..n-\mathsf{len}(\mathsf{maxPal})-1]$. Put $d' = \min\{\mathsf{live}(\mathsf{cntr}(|u|)), \mathsf{live}(\mathsf{cntr}(|uvu|))\}$ (see Fig. 2). Let us show that we can use d' instead of d. If $d' < n - \mathsf{left}[p] - |uvu|$, then the longest suffix-palindrome is preceded by the reversed prefix of $(vu)^\infty$ of length d'. In turn, this prefix either is preceded by a letter that breaks the period p of this prefix (the letter e in Fig. 2) or is a prefix of the whole string. In either case, $d' = d$. If $d' \geq n - \mathsf{left}[p] - |uvu|$, then the longest suffix-palindrome is also preceded by the reversed prefix of $(vu)^\infty$ of length d' but $d \geq d'$ in general. However, even in this case, we can use d' in the sequel since none of the suffix-palindromes from our series survives after $n - \mathsf{left}[p] - |uvu|$ predictable extensions; therefore, also, the possible processing of a suffix-palindrome of length $n - \mathsf{left}[p] - d'$ mentioned above is not required.

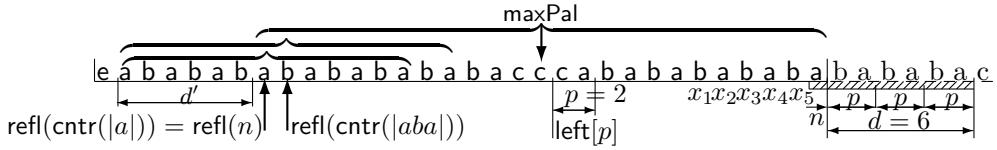

Figure 2 A series $x_1, \dots, x_5$: $\mathsf{live}(x_4) > d$, the shaded region corresponds to $\mathsf{ans}[n..n+d]$.

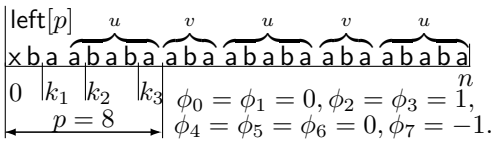

Figure 3 Partition of $[0..p)$ in Lemma 6.

Let us track the set $S = \{\ell_1 = n - \mathsf{len}(x_1)+1, \dots, \ell_k = n - \mathsf{len}(x_k)+1\}$ of the leftmost positions of the suffix-palindromes centered at $x_1, x_2, \dots, x_k$ in the d' predictable extensions: all these positions shift to the left by one after each extension; if a position reaches $\mathsf{left}[p]$, the corresponding palindrome dies and this position is excluded from S. By Lemma 3, for any $i \in [1..k]$, if ℓ_i is in the set after $f \in [0..d']$ predictable extensions, then the suffixes $s[\ell_i + jp..n]$ (here n is increased by f) are palindromes for all integers $j \geq 0$ such that $\ell_i + jp \leq n$; therefore, along with the assignments $\mathsf{ans}[n] \overset{\min}{\leftarrow} 1 + \mathsf{ans}[\ell_i - 1]$ (here n is increased by f) that we are intended to perform, we can occasionally perform $\mathsf{ans}[n] \overset{\min}{\leftarrow} 1 + \mathsf{ans}[\ell_i + jp - 1]$ for any such j.

Obviously, $|u_1| + p > n - \mathsf{left}[p]$ since otherwise uvu_1 would be a longer suffix-palindrome with the minimal period p. Based on the above observation, we perform the assignments $\mathsf{ans}[n+j] \overset{\min}{\leftarrow} 1 + \mathsf{pre}[p][r(j)]$ for all $j \in [0.. \min\{b, d'\}]$, where $r(j) = (n - |u_k| - \mathsf{left}[p] - j) \bmod p$ (see Fig. 3; $r(j)$ cyclically runs through the range $[0..p)$ from right to left when j increases). Recall that, immediately before the execution of $\mathsf{predict}$, the function $\mathsf{precalc}$ recalculates the array pre. After this recalculation $\mathsf{pre}[p]$ stores an array $\mathsf{pre}[p][0..p-1]$ for each $p \in [1..n]$ such that p is the minimal period of a suffix-palindrome of $s[0..n]$. For $i \in [0..p)$ we have $\mathsf{pre}[p][i] = \min\{\mathsf{ans}[\mathsf{left}[p] + i + jp] : j \in [0..\phi_i]\}$, where $\phi_i \geq 0$ is the maximal integer such that the string $s[\mathsf{left}[p] + i + \phi_i p + 1..n]$ has a prefix-palindrome with the minimal period p; if there is no such ϕ_i, we put $\mathsf{pre}[p][i] = +\infty$ and $\phi_i := -1$.

We perform $\mathsf{ans}[n+j] \overset{\min}{\leftarrow} 1 + \mathsf{pre}[p][r(j)]$, for all $j \in [0.. \min\{b, d'\}]$, in $O(1)$ time using range operations on the arrays pre and ans. (These operations are discussed below.) It follows from Lemma 5 that, after $f \in [0..d']$ predictable extensions, the strings $s[\ell_i..n']$ (here n' denotes the value of n before the extensions), for $i \in [1..k)$ such that ℓ_i is still in the set S, have prefix-palindromes with the minimal period p. Therefore, the above assignments will really process the palindromes $u_1, \dots, u_{k-1}$ for the upcoming d' predictable extensions (see Fig. 2) but will, probably, perform some additional unnecessary assignments for suffix-palindromes with period p that will appear only after a number of predictable extensions; but this does not harm since such assignments will be performed anyway in the future. For the baby u_k, we compute explicitly $\mathsf{ans}[n..n+c] \overset{\min}{\leftarrow} \mathsf{increv}(n - \mathsf{len}(x_k) - c, n - \mathsf{len}(x_k))$, where $c = \min\{b, \mathsf{live}(x_k)\}$, in $O(1)$ time using range operations. It remains to describe the structure of the array pre that allows us to perform constant time range operations on subarrays of length $\leq b$.

▶ **Lemma 6.** *For each $i \in [0..p)$, let ϕ_i be the minimal integer such that the string $s[\mathsf{left}[p] + i + (\phi_i + 1)p + 1..n]$ has no prefix-palindromes with the minimal period p. Then, the segment $[0..p)$ can be split into subsegments $[k_0..k_1), \dots, [k_6..k_7)$, for $0 = k_0 \leq \dots \leq k_7 = p$, such that, for $i \in (0..p)$, we have $\phi_i = \phi_{i-1}$ whenever i and $i-1$ belong to the same subsegment (see Fig. 3).*

Proof. For $i \in [0..p)$ and $j \geq 0$, denote $i(j) = \mathsf{left}[p]+i+jp$. Let k be an integer such that, for $i = p-1$, we have $i(k) < n$ and $i(k)+p \geq n$. So, for $i \in [0..p)$, we obtain $\phi_i = j'_i - 1$, where j'_i is the minimal integer such that $j'_i \in [0..k]$ and the string $s[i(j'_i)+1..n]$ has no prefix-palindromes with the minimal period p. While i descends from $p-1$ to 0 with step one, some of the suffixes $s[i(j)+1..n]$ may acquire prefix-palindromes with the minimal period p and some may lose such prefix-palindromes thus changing the value of ϕ_i (see Fig. 3).

Let us choose $i \in [0..p)$ that maximizes the value of ϕ_i. Denote $j' = \phi_i$ for this i. If $j' \geq 0$, then $s[i(j')+1..n]$ has a prefix-palindrome w with the minimal period p; by Lemma 3, there are palindromes u and v such that $|uv| = p$ and $w = (uv)^r u$ for $r \geq 1$. Thus, for any $j'' \in [0..j'-2]$, the suffix $s[i(j'')+1..n]$ has prefix-palindromes $(uv)^3u$ and $(uv)^2u$ both having the minimal period p. When i further decreases to 0, the prefix-palindrome $(uv)^2u$ "grows" together with $s[i(j'')+1..n]$ and, when i increases, $(uv)^3u$ "shrinks"; in both cases $s[i(j'')+1..n]$ retains a prefix-palindrome with the minimal period p while $i \in [0..p)$. Hence, only suffixes $s[i(j'-1)+1..n]$ and $s[i(j')+1..n]$ may lose or acquire a prefix-palindrome with the minimal period p while i changes from $p-1$ to 0, i.e., ϕ_i varies in the range $[j'-2..j']$.

Let us prove that any suffix $s[i(j)+1..n]$ can lose a prefix-palindrome with the minimal period p at most once during the descending of i from $p-1$ to 0. Then, the existence of the desired numbers $k_0, k_1, \ldots, k_7$ follows from a simple analysis of possible cases.

Suppose that $s[i(j)+1..n]$ has a prefix-palindrome centered at x with the minimal period p. When i decreases, $s[i(j)+1..n]$ grows and the prefix-palindrome "grows" simultaneously. Then, before $s[i(j)+1..n]$ loses the prefix-palindrome, we have $|s[i(j)+1..n]| = \mathsf{len}(x)$ for some $i \in [0..p)$. By Lemma 4, there are palindromes u' and v' such that $|u'v'| = p$ and $s[n-\mathsf{len}(x)+1..n] = (u'v')^{r'}u'$ for $r' \geq 1$. If, for some smaller $i \in [0..p)$, $s[i(j)+1..n]$ again acquires a prefix-palindrome with the minimal period p, then, by Lemma 4, the center x' of this prefix-palindrome must coincide with the center of u' or v' from the decomposition. Hence $x' \leq x - p/2$. Then, this prefix-palindrome can be lost only after p decrements of i once we have had $|s[i(j)+1..n]| = \mathsf{len}(x)$. This proves the claim. ◀

We partition $\mathsf{pre}[p][0..p-1]$ into subarrays $\mathsf{pre}[p][k_0..k_1-1], \ldots, \mathsf{pre}[p][k_6..k_7-1]$ according to Lemma 6. Consider a segment $[a..b] \subset [0..p)$ such that $\phi_{i_1} = \phi_{i_2}$ and $\phi_{i_1} \neq -1$ whenever $i_1, i_2 \in [a..b]$. Since $\mathsf{pre}[p][i] = \min\{\mathsf{ans}[\mathsf{left}[p]+i+jp] \colon j \in [0..\phi_i]\}$ and, by Lemma 2, $|\mathsf{ans}[j] - \mathsf{ans}[j-1]| \leq 1$ for any $j \in (0..n)$, we easily obtain $|\mathsf{pre}[p][i] - \mathsf{pre}[p][i-1]| \leq 1$ for any $i \in (a..b]$. Therefore, by Lemma 6, each of the subarrays of pre either contains only $+\infty$ or has a structure similar to the structure of ans described in Lemma 2. We do not store the subarrays that contain $+\infty$ and encode all other subarrays in a way described for ans in Sect. 2.4: we split them into blocks of length b and encode each block as its starting element and a $2b$-bit integer encoding the differences between adjacent elements (the last block may contain less than b elements). The linear size of pre measured in machine words (but not in the number of elements) follows from the overall linear running time of the function $\mathsf{precalc}$ maintaining pre; this analysis is given below.

To perform $\mathsf{ans}[n+j] \overset{\min}{\leftarrow} \mathsf{pre}[p][r(j)]$ for all $j \in [0..\min\{b, d'\}]$, we concatenate $2b$-bit integers from the blocks covering the subarray $\mathsf{ans}[n..n+\min\{b, d'\}]$, $2b$-bit integers from a constant number of blocks covering the subarrays of $\mathsf{pre}[p][0..p-1]$ containing positions $r(j)$ for $j \in [0..\min\{b, d'\}]$, and some other lightweight auxiliary data similar to the data used in the operation $\overset{\min}{\leftarrow}$ considered above; then we compute the resulting array $\mathsf{ans}[n..n+\min\{b, d'\}]$ using the obtained bit string and a precomputed table of size $o(n)$. This might require to duplicate the content of $\mathsf{pre}[p]$ if $p < \min\{b, d'\}$ (see the shaded region in Fig. 2); these duplications must be already precalculated in the tables. Note that thus defined changes of ans may affect the whole subarray $\mathsf{ans}[n..n+b]$ and not only $\mathsf{ans}[n..n+\min\{b, d'\}]$: e.g., if we

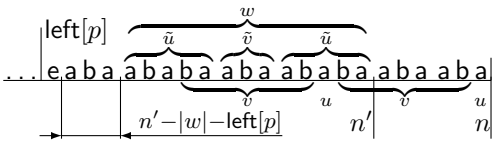

Figure 4 Palindrome w with the center x, the minimal period $p = 8$; for $i = 1, 2, 3$, $j = 0$.

perform $\mathsf{ans}[n] \overset{\min}{\leftarrow} x$, then, to maintain the property of ans described in Lemma 2, we must also perform $\mathsf{ans}[n{+}j] \overset{\min}{\leftarrow} x + j$ for $j \in [1..b]$ (it is guaranteed by the algorithm that the elements of $\mathsf{ans}[0..n{-}1]$ cannot be affected analogously since always $|\mathsf{ans}[n{-}1] - \mathsf{ans}[n]| \le 1$). Similar "normalizations" must be included in the precomputed assignments $\mathsf{ans}[n{+}j] \overset{\min}{\leftarrow} \mathsf{pre}[p][r(j)]$ for all $j \in [0..\min\{b, d'\}]$. Thus, the structure of ans is preserved.

The computations seem to be quite sophisticated but, nevertheless, since all involved structures occupy $\varepsilon \log n$ bits, for $\varepsilon < 1$, all required precalculations can be performed in $O(n^{\varepsilon} \log^{O(1)} n)$ time at the beginning of our algorithm. The tedious details are omitted here and can be retrieved from the implementation [9].

3.2 Function precalc

Denote by n' the value of n at the moment of the last call of precalc. (The first call of precalc for $n = 0$ is trivial.) Our goal is to compute the array $\mathsf{pre}[p][0..p{-}1]$ for each p for which there exists a p-series in $s[0..n]$. Note that since, as described above, $\mathsf{pre}[p][0..p{-}1]$ is stored as a constant number of pointers to subarrays containing non-infinite values, we can fill $\mathsf{pre}[p][0..p{-}1]$ with $+\infty$ in $O(1)$ time simply removing all these pointers.

The function precalc loops through all series in $s[0..n]$ and processes each p-series as follows: precalc computes the new value of $\mathsf{left}[p]$ in $O(1)$ time and, if $\mathsf{left}[p]$ has changed since $s[0..n']$ (this is where we really use the array left), then fills $\mathsf{pre}[p][0..p{-}1]$ with $+\infty$ in $O(1)$ time; otherwise, precalc uses the array $\mathsf{pre}[p][0..p{-}1]$ calculated for $s[0..n']$. In either case, for each $i \in [0..p)$, if there is an integer $j \ge 0$ such that $s[\mathsf{left}[p]{+}i{+}jp..n']$ does not have a prefix-palindrome with the minimal period p and $s[\mathsf{left}[p]{+}i{+}jp..n]$ has such a prefix-palindrome, then $\mathsf{pre}[p][i]$ is updated by performing $\mathsf{pre}[p][i] \overset{\min}{\leftarrow} \mathsf{ans}[\mathsf{left}[p]{+}i{+}jp{-}1]$. The methods by which we find such $i \in [0..p)$ and really perform the later assignments are described below. It follows from the definition of pre that thus defined precalc computes the arrays $\mathsf{pre}[p][0..p{-}1]$ for $s[0..n]$.

Let us process all centers x for which there are $i \in [0..p)$ and $j \ge 0$ such that x is the center of a prefix-palindrome of $s[\mathsf{left}[p]{+}i{+}jp..n]$ with the minimal period p but $s[\mathsf{left}[p]{+}i{+}jp..n']$ does not have a prefix-palindrome with the minimal period p. We consider two cases.

Case 1. Suppose that such x is less than $n'{+}1$ and the longest subpalindrome w in $s[0..n']$ centered at x has the minimal period p. Clearly, the leftmost position of w is greater than $\mathsf{left}[p] + i + jp$ and w must be a suffix-palindrome of $s[0..n']$. Let us describe all positions $h_m = n'{-}|w|{-}m$ such that x is the center of a prefix-palindrome of $s[h_m{+}1..n]$ and is not the center of a prefix-palindrome of $s[h_m{+}1..n']$. Obviously $m > 0$. After $n'{-}|w|{-}\mathsf{left}[p]{+}1$ extensions of $s[0..n']$, the suffix-palindrome centered at x dies because it reaches $\mathsf{left}[p]$ by its leftmost position (see Fig. 4). So, since w grows at most $n - n'$ times, we obtain $m \in [1..\min\{n{-}n', n'{-}|w|{-}\mathsf{left}[p]\}]$. For each such m, the prefix-palindrome of $s[h_m{+}1..n]$ centered at x has length $|w| + 2m$ and the minimal period p since the minimal period of w, centered at x, is p and the palindrome with the length $|w| + 2m$ and the center x, for the given m, is a substring of the suffix of s with period p. Hence, we can perform

$\mathsf{pre}[p][(r-m) \bmod p] \overset{\min}{\leftarrow} \mathsf{ans}[n'-|w|-m]$, where $r = n' - |w| - \mathsf{left}[p]$, for all such m. Among these assignments there is the required $\mathsf{pre}[p][i] \overset{\min}{\leftarrow} \mathsf{ans}[\mathsf{left}[p]+i+jp-1]$ (see Fig. 4). Since $n - n' \le b$, once x is known, we can perform all these assignments in $O(1)$ time using range operations and precomputed tables; the boundaries of subarrays of pre can be adjusted appropriately after these calculations. Now it remains to find all such centers x.

By Lemma 3, there are palindromes $\tilde{u}$ and $\tilde{v}$ such that $p = |\tilde{u}\tilde{v}|$ and $w = (\tilde{u}\tilde{v})^r\tilde{u}$ for $r \ge 1$ (see Fig. 4). If the minimal period of $(\tilde{u}\tilde{v})^{r-1}\tilde{u}$ is p, then all strings $s[h..n']$, for $h \in (\mathsf{left}[p]..n'-|w|]$, have prefix-palindromes of the form $\alpha(\tilde{u}\tilde{v})^{r-1}\tilde{u}\tilde{\alpha}$, where α is a suffix of $\tilde{u}\tilde{v}$, with the minimal period p. But, by our assumption, $s[\mathsf{left}[p]+i+jp..n']$ cannot have such a prefix-palindrome. Therefore, w is the baby in the p-series of the string $s[0..n']$, i.e., either $w = \tilde{u}\tilde{v}\tilde{u}$ or $w = \tilde{u}\tilde{v}\tilde{u}\tilde{v}\tilde{u}$. We find the baby in $O(1)$ time by the techniques described above using an instance of the iterator and the list of all series of suffix-palindromes for the string $s[0..n']$; these iterator and list are further discussed below.

Case 2. It remains to detect all x such that x is the center of a prefix-palindrome of $s[\mathsf{left}[p]+i+jp..n]$ with the minimal period p, for some $i \in [0..p)$ and $j \ge 0$, and either $x > n'$ or the minimal period of any subpalindrome in $s[0..n']$ centered at x is not p. Hence, a subpalindrome with the minimal period p and the center x appeared after several extensions of $s[0..n']$ and, thus, was a suffix-palindrome at that moment. To catch the moments when growing suffix-palindromes acquire new minimal periods, we need a device tracking changes of periods in all suffix-palindromes after extensions. The iterator can serve as such a device.

Let w be a suffix-palindrome of $s[0..n']$ with the minimal period p'. By Lemma 3, we have $p' = |w| - |u|$, where u is the longest proper suffix-palindrome of w. Suppose that $s[0..n']$ is extended by the letter $a = s[n'+1]$ and awa is a suffix-palindrome of the new string. By Lemma 3, awa has period p' iff aua is a suffix-palindrome of $s[0..n'+1]$. Thus, to detect new suffix-palindromes with a given period p, we can track, during the extensions of s, changes in the list of the centers of all suffix-palindromes. The iterator maintains such list. The following lemma is a straightforward corollary of the proof of [7, Prop. 1].

▶ **Lemma 7.** *The iterator maintains a linked list of the centers of all suffix-palindromes of $s[0..n]$. The function $\mathsf{add}(a)$ removes a number of centers from the list, adds the centers $n+\frac{1}{2}$ (if $a = s[n]$) and $n+1$ to the end of the list, and thus obtains a new list for the string $s[0..n]a$; all in $\Omega(1+c)$ time, where c is the number of removed centers.*

We maintain an instance of the iterator for the previously processed string $s[0..n']$ and store the list of the centers of all suffix-palindromes of $s[0..n']$ since the last call of $\mathsf{precalc}$. The function $\mathsf{precalc}$ performs $\mathsf{add}(s[n'+1]), \ldots, \mathsf{add}(s[n])$ and thus consecutively obtains the lists of the centers of all suffix-palindromes of $s[0..n'+1], \ldots, s[0..n]$.

Consider, for $n'' \in (n'..n]$, such list $x_1, \ldots, x_k$ for $s[0..n''-1]$ so that $x_1 < \cdots < x_k$. By Lemma 7, the call to $\mathsf{add}(s[n''])$ gives us a sublist $x_{i_1}, \ldots, x_{i_c}$ of the centers removed from $x_1, \ldots, x_k$. By Lemma 3, for $x_i \notin \{x_{i_1}, \ldots, x_{i_c}\}$ the minimal period of the suffix-palindrome with the center x_i has changed iff $x_{i+1} \in \{x_{i_1}, \ldots, x_{i_c}\}$. We easily find all such x_i parsing the list $x_{i_1}, \ldots, x_{i_c}$ from left to right and compute the new period as $p = \mathsf{len}(x_i) - \mathsf{len}(\mathsf{nextPal}(x_i))$. Denote by ℓ the number that is equal to $\mathsf{len}(x_i)$ for $s[0..n'']$. By the definition of pre, we must perform $\mathsf{pre}[p][r] \overset{\min}{\leftarrow} \mathsf{ans}[n''-\ell]$, where $r = (n'' - \ell - \mathsf{left}[p]) \bmod p$, if the string $s[0..n]$ has a p-series. In this case, we must also perform $\mathsf{pre}[p][(r-m) \bmod p] \overset{\min}{\leftarrow} \mathsf{ans}[n''-\ell-m]$ for all $m \in [0..\min\{n-n'', n''-\ell-\mathsf{left}[p]\}]$ because the strings $s[n''-\ell-m..n]$ have prefix-palindromes of length $\ell+2m$ centered at x_i with the minimal period p; after $n''-\ell-\mathsf{left}[p]+1$ extensions, such palindrome dies since it reaches $\mathsf{left}[p]$ by its leftmost position and thus its period breaks. Since $n - n'' \le b$, these assignments, for all such m, can be performed by

range operations on pre and ans in $O(1)$ time using precomputed tables like those described in Sect. 2.4 (subarrays of pre$[p]$ can be also adjusted appropriately).

Thus, precalc works in $O(t+q)$ time as required, where t is the number of series in $s[0..n]$ and q is the time required to perform the sequence of calls $\mathsf{add}(s[n'+1]), \ldots, \mathsf{add}(s[n])$. This finishes the proof of the linear time complexity of main algorithm.

References

1 Vladimir Arlazarov, Efim Dinic, Mikhail Kronrod, and Igor Faradzev. On economical construction of the transitive closure of a directed graph. *Dokl. Akad. Nauk SSSR*, 194(11):1209–1210, 1970.

2 Stefan Burkhardt and Juha Kärkkäinen. Fast lightweight suffix array construction and checking. In Ricardo A. Baeza-Yates, Edgar Chávez, and Maxime Crochemore, editors, *Proc. of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003)*, volume 2676 of *LNCS*, pages 55–69. Springer, 2003. doi:10.1007/3-540-44888-8_5.

3 Gabriele Fici, Travis Gagie, Juha Kärkkäinen, and Dominik Kempa. A subquadratic algorithm for minimum palindromic factorization. *J. Discrete Algorithms*, 28:41–48, 2014. doi:10.1016/j.jda.2014.08.001.

4 Zvi Galil and Joel I. Seiferas. A linear-time on-line recognition algorithm for "palstar". *J. ACM*, 25(1):102–111, 1978. doi:10.1145/322047.322056.

5 Tomohiro I, Shiho Sugimoto, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing palindromic factorizations and palindromic covers on-line. In Alexander S. Kulikov, Sergei O. Kuznetsov, and Pavel A. Pevzner, editors, *Proceedings of the 25th Annual Symposium on Combinatorial Pattern Matching (CPM 2014)*, volume 8486 of *LNCS*, pages 150–161. Springer, 2014. doi:10.1007/978-3-319-07566-2_16.

6 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024.

7 Dmitry Kosolobov, Mikhail Rubinchik, and Arseny M. Shur. Pal^k is linear recognizable online. In Giuseppe F. Italiano, Tiziana Margaria-Steffen, Jaroslav Pokorný, Jean-Jacques Quisquater, and Roger Wattenhofer, editors, *Proceedings of the 41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2015)*, volume 8939 of *LNCS*, pages 289–301. Springer, 2015. doi:10.1007/978-3-662-46078-8_24.

8 Glenn K. Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *J. ACM*, 22(3):346–351, 1975. doi:10.1145/321892.321896.

9 Palindromic Length. Sources and tests for linear palindromic length problem, 2017. URL: https://github.com/kborozdin/palindromic-length.

10 Mikhail Rubinchik and Arseny M. Shur. EERTREE: an efficient data structure for processing palindromes in strings. In Zsuzsanna Lipták and William F. Smyth, editors, *Proc. of the 26th International Workshop on Combinatorial Algorithms (IWOCA 2015)*, volume 9538 of *LNCS*, pages 321–333. Springer, 2015. doi:10.1007/978-3-319-29516-9_27.

11 Anatol O. Slisenko. A simplified proof of the real-time recognizability of palindromes on turing machines. *J. Soviet. Math.*, 15(1):68–77, 1977. doi:10.1007/BF01404109.