Computing Maximal Palindromes and Distinct Palindromes in a Trie

Mitsuru Funakoshi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda

Department of Informatics, Kyushu University, Japan
{mitsuru.funakoshi, yuto.nakashima, inenaga, bannai, takeda}@inf.kyushu-u.ac.jp

Abstract. It is known that all maximal palindromes of a given string T of length n can be computed in $O(n)$ time by Manacher's algorithm [J. ACM '75]. Also, all distinct palindromes in T can be computed in $O(n)$ time [Groult et al., Inf. Process. Lett. 2010]. In this paper, we consider the problem of computing maximal palindromes and distinct palindromes of a given trie $\mathcal{T}$ (i.e. rooted edge-labeled tree). A trie is a natural generalization of a string which can be seen as a single path tree. We propose algorithms to compute all maximal palindromes and all distinct palindromes in $\mathcal{T}$ in $O(N \log h)$ time and $O(N)$ space, where N is the number of edges in $\mathcal{T}$ and h is the height of $\mathcal{T}$. To our knowledge these are the first sub-quadratic time solutions to these problems.

Keywords: palindromes, string and tree algorithms, periodicity, suffix arrays

1 Introduction

Palindromes are strings that read the same forward and backward. Finding palindromic structures in a given string is a fundamental task in string processing, and thus it has extensively been studied (e.g., see [2,27,17,25,33,23,32,14] and references therein).

Consider a set $C = \{1, 1.5, 2, \ldots, n\}$ of $2n-1$ half-integer and integer positions in a string T of length n. The maximal palindrome for a position $c \in C$ in T is a non-extensible palindrome whose center lies on c. It is easy to store all maximal palindromes with $O(n)$ total space; e.g., simply store their lengths in an array of length $2n-1$ together with the input string T. If $P = T[i..j]$ is a maximal palindrome with center $c = \frac{i+j}{2}$, then clearly any substrings $P' = T[i+d..j-d]$ with $0 \leq d \leq \frac{j-i}{2}$ are also palindromes. Hence, by computing and storing all maximal palindromes in T, we can obtain a compact representation of all palindromes in T. Manacher [26] gave an elegant $O(n)$-time algorithm to compute all maximal palindromes in T. This algorithm works for a general alphabet. For the case where the input string is drawn from a constant size alphabet or an integer alphabet of size polynomial in n, there is an alternative suffix tree [38] based algorithm which takes $O(n)$ time [18]. In this method, the suffix tree of $T\#T^R\$$ is constructed, where T^R is the reversed string of T, and $\#$ and $\$$ are special characters not occurring in T. By enhancing the suffix tree with a lowest common ancestor (LCA) data structure [10], *outward* longest common extension (LCE) queries from a given $c \in C$ can be answered in $O(1)$ time after an $O(n)$-time preprocessing.

Another central question regarding substring palindromes is distinct palindromes. Droubay et al. [9] showed that any string of length n can contain at most $n+1$ distinct palindromes (including the empty string). Strings of length n that contain exactly $n+1$ distinct palindromes are called *rich* strings in the literature [16,7]. Groult

Mitsuru Funakoshi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda: Computing Maximal Palindromes and Distinct Palindromes in a Trie, pp. 3–15.
Proceedings of PSC 2019, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-06618-8

et al. [17] proposed an $O(n)$-time algorithm for computing all distinct palindromes in a string of length n over a constant-size alphabet or an integer alphabet of size polynomial in n.

A *trie* is a rooted tree where each edge is labeled by a single character and the out-going edges of each node are labeled by mutually distinct characters. A trie is a natural extension to a string, and is a compact representation of a set of strings. There are a number of works for efficient algorithms on tries, such as indexing a (reversed) trie [5,24,35,21,29,11,31,20] for exact pattern matching, parameterized pattern matching on a trie [1,12], order preserving pattern matching on a trie [30], and finding all maximal repetitions (a.k.a. runs) in a trie [37].

In this paper, we tackle the problems of computing all maximal palindromes and all distinct palindromes in a given trie $\mathcal{T}$. Naïve methods for solving these problems would be to apply Manacher's algorithm [26] or Groult et al.'s algorithm [17] for each string in $\mathcal{T}$, but this requires $\Omega(N^2)$ time in the worst case since there exists a trie with N edges that can represent $\Theta(N)$ strings of length $\Theta(N)$ each. We also remark that a direct application of Manacher's algorithm to a trie does not seem to solve our problem efficiently, since the amortization argument in the case of a single string does not hold in our case of a trie. The aforementioned suffix tree approach [18] cannot be applied to our trie case either; while the number of suffixes in the reversed leaf-to-root direction of the trie $\mathcal{T}$ is N, the number of suffixes in the forward root-to-leaf direction can be $\Theta(N^2)$ in the worst case. Thus one cannot afford to construct the suffix tree that contains all suffixes of the forward paths of $\mathcal{T}$.

In this paper, we first show that the number of maximal palindromes in a trie $\mathcal{T}$ with N edges and L leaves is exactly $2N - L$ and that the number of distinct palindromes in $\mathcal{T}$ is at most $N + 1$. These generalize the known bounds for a single string. Then, we present two algorithms to compute all maximal palindromes both of which run in $O(N \log h)$ time and $O(N)$ space in the worst case, where h is the height of the trie $\mathcal{T}$. We then present how to compute all distinct palindromes in a given trie $\mathcal{T}$ in $O(N \log h)$ time with $O(N)$ space. The key tools we use are periodicities of suffix palindromes and string data structures that are built on the (reversed) trie. To the best of our knowledge, these are the first algorithms for finding maximal/distinct palindromes from a given trie in sub-quadratic time.

Related work

There are a few combinatorial results for palindromes in an *unrooted* edge-labeled tree. Brlek et al. [6] showed an $\Omega(M^{3/2})$ lower bound on the maximum number of distinct palindromes in an unrooted tree with M edges. Later Gawrychowski et al. [15] showed a matching upper bound $O(M^{3/2})$ on the maximum number of distinct palindromes in an unrooted tree with M edges. Note that these previous works consider an unrooted tree, and, to the best of our knowledge, palindromes of a trie (rooted edge-labeled tree) have previously not been studied. Concerning repetitive structures in tries, Sugahara et al. [37] proved that any trie with N edges can contain less than N maximal repetitions (or runs), and showed that all runs in a given trie can be found in $O(N(\log\log N)^2)$ time with $O(N)$ space. Our paper can be considered as computing palindromes, instead of runs, given the same input.

2 Preliminaries

2.1 String notation

Let Σ be the *alphabet*. An element of Σ^* is called a *string*. The length of a string T is denoted by $|T|$. The empty string ε is a string of length 0, namely, $|\varepsilon| = 0$. For a string $T = xyz$, x, y and z are called a *prefix*, *substring*, and *suffix* of T, respectively. For two strings X and Y, let $\mathsf{lcp}(X, Y)$ denote the length of the longest common prefix of X and Y.

For a string T and an integer $1 \leq i \leq |T|$, $T[i]$ denotes the ith character of T, and for two integers $1 \leq i \leq j \leq |T|$, $T[i..j]$ denotes the substring of T that begins at position i and ends at position j. For convenience, let $T[i..j] = \varepsilon$ when $i > j$. An integer $p \geq 1$ is said to be a *period* of a string T iff $T[i] = T[i+p]$ for all $1 \leq i \leq |T|-p$.

Let T^R denote the reversed string of T, i.e., $T^R = T[|T|] \cdots T[1]$. A string T is called a *palindrome* if $T = T^R$. We remark that the empty string ε is also considered to be a palindrome. For any non-empty substring palindrome $T[i..j]$ in T, $\frac{i+j}{2}$ is called its *center*. A non-empty substring palindrome $T[i..j]$ is said to be a *maximal palindrome* centered at $\frac{i+j}{2}$ in T if $T[i-1] \neq T[j+1]$, $i = 1$, or $j = |T|$. It is clear that for each center $c = 1, 1.5, \ldots, n-0.5, n$, we can identify the maximal palindrome $T[i..j]$ whose center is c (namely, $c = \frac{i+j}{2}$). Thus, there are exactly $2n-1$ maximal palindromes in a string of length n. In particular, maximal palindromes $T[1..i]$ and $T[i..|T|]$ for $1 \leq i \leq n$ are respectively called a *prefix palindrome* and a *suffix palindrome* of T.

2.2 Tries and algorithmic tools

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

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

Consider a trie with N edges such that the root has a single out-edge labeled with a special character \$ that does not appear elsewhere in the trie and is lexicographically the smallest. We consider the reversed trie of this trie. The *suffix array* of this reversed trie can be constructed in $O(N)$ time [36,11]. Also, the *longest common prefix array* (*LCP array*) for this suffix array can also be constructed in $O(N)$ time [22].

2.3 Computing palindromes in a string

Manacher [26] showed an elegant online algorithm which computes all maximal palindromes of a given string T of length n in $O(n)$ time. An alternative offline approach is

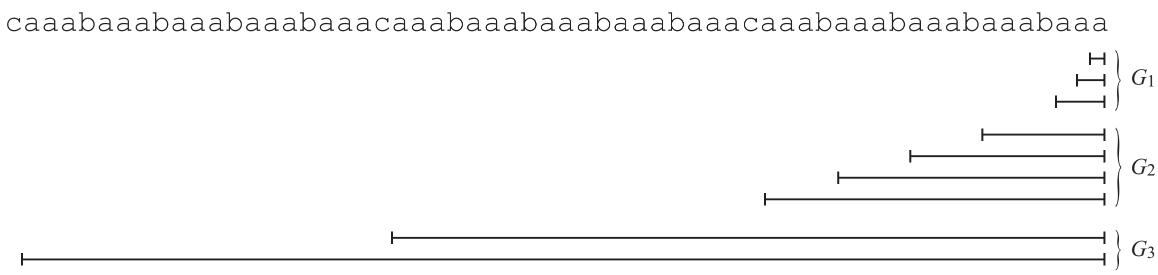


Figure 1. Examples of arithmetic progressions representing the suffix palindromes of a string. The first group G_1 is represented by $\langle 1, 1, 3\rangle$, the second group G_2 by $\langle 7, 4, 4\rangle$, and the third group G_3 by $\langle 39, 20, 2\rangle$.

to use outward LCE queries for $2n-1$ pairs of positions in T. Using the suffix tree [38] for string $T\$T^R\#$ enhanced with a lowest common ancestor data structure [19,34,3], where \$ and # are special characters which do not appear in T, each outward LCE query can be answered in $O(1)$ time. For any integer alphabet of size polynomial in n, preprocessing for this approach takes $O(n)$ time and space [10,18].

Let T be a string of length n. For each $1 \le i \le n$, let $\mathit{MaxPalEnd}_T(i)$ denote the set of maximal palindromes of T that end at position i. Let $\mathbf{S}_i = s_1, \ldots, s_g$ be the sequence of lengths of maximal palindromes in $\mathit{MaxPalEnd}_T(i)$ sorted in increasing order, where $g = |\mathit{MaxPalEnd}_T(i)|$. Let d_j be the progression difference for s_j, i.e., $d_j = s_j - s_{j-1}$ for $2 \le j \le g$. In particular, let $d_1 = s_1 - |\varepsilon| = s_1$. We use the following lemma which is based on periodic properties of maximal palindromes ending at the same position.

Lemma 1 (Lemma 2 of [13]).

(i) *For any* $1 \le j < g$, $d_{j+1} \ge d_j$.
(ii) *For any* $1 < j < g$, *if* $d_{j+1} \ne d_j$, *then* $d_{j+1} \ge d_j + d_{j-1}$.
(iii) $\mathbf{S}_i$ *can be represented by* $O(\log i)$ *arithmetic progressions, where each arithmetic progression is a tuple* $\langle s, d, t\rangle$ *representing the sequence* $s, s+d, \ldots, s+(t-1)d$ *with common difference* d.
(iv) *If* $t \ge 2$, *then the common difference* d *is a period of every maximal palindrome which ends at position* i *in* T *and whose length belongs to the arithmetic progression* $\langle s, d, t\rangle$.

Each arithmetic progression $\langle s, d, t\rangle$ is called a *group* of maximal palindromes. See also Figure 1 for a concrete example.

Since each arithmetic progression can be stored in $O(1)$ space, and since there are only $O(\log i)$ arithmetic progressions for each position i, we can represent all maximal palindromes ending at position i in $O(\log i)$ space.

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

Since suffix palindromes are also maximal palindromes, $\mathit{MaxPalEnd}_T(n)$ is the set of suffix palindromes of T, where $n = |T|$. Thus Lemma 1 holds for suffix palindromes in T. This particular case of Lemma 1 was shown in the literature [2,28].

Our algorithms will make a heavy use of periodicity of maximal/suffix palindromes of a string stated in Lemma 1.

3 Maximal/distinct palindromes in a trie

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

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

(1) $\mathsf{str}(\mathsf{parent}(u), v')$ is not a palindrome with *any* child v' of v,
(2) u is the root, or
(3) v is a leaf.

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

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

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

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

Lemma 3. *There are at most* $N+1$ *distinct palindromes in any trie* $\mathcal{T}$ *with* N *edges.*

Proof. We follow the proof from [9] which shows that the number of distinct palindromes in a string of length n is at most $n + 1$.

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

See Figure 2 for examples of maximal palindromes and distinct palindromes in a trie.

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

4 Computing maximal palindromes in a trie

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

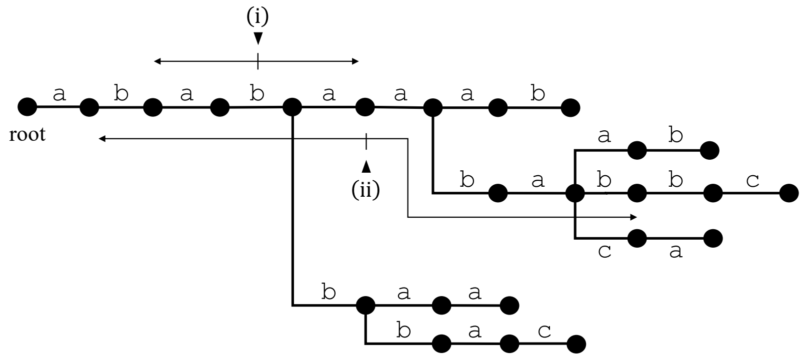


Figure 2. The maximal palindrome centered at (i) is aba and the maximal palindrome centered at (ii) is babaabab. The set of distinct palindromes in this trie is $\{\varepsilon$, a, b, c, aa, bb, aaa, aba, aca, bab, bbb, abba, baab, aabaa, ababa, abbba, baaab, abaaba, baabaab, babaabab$\}$.

4.1 $O(N \log h)$-time $O(h)$-space algorithm

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

The basic strategy of our algorithm is as follows. We perform a depth-first traversal on $\mathcal{T}$. Let r be the root of $\mathcal{T}$. We use Lemma 1 in our algorithm. When visiting a node u during the depth-first traversal on trie $\mathcal{T}$, we maintain the arithmetic progressions for the maximal palindromes in the path string $\mathsf{str}(r, u)$. In each node x in the path from r to u, the arithmetic progressions representing the maximal palindromes ending at x are sorted in the increasing order of the lengths of the corresponding maximal palindromes. Since $\mathsf{str}(r, u)$ is a single string, and since $|\mathsf{str}(r, u)|$ is bounded by the height h of $\mathcal{T}$, we can store all these arithmetic progressions in $O(h)$ total space during the traversal. Suppose that u has two or more children, and let v, v' be two distinct children of u. Notice that some of the maximal palindromes ending at u could be extended by the edge label from u to v. Furthermore, since the edge label between u and v differs from the edge label between u and v', those palindromes that are not extended with v could still be extended with v'. This in turn means that when we backtrack to u after visiting v, then we can use the maximal palindromes in the path string $\mathsf{str}(r, v)$ that ends at the parent u of v, for finding the palindromes ending at another child v'. In the sequel, we will describe how to efficiently maintain these maximal palindromes during the traversal.

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

1. $a_i = b$ and $a_i = c$ (namely $a_i = b = c$): In this case, all the suffix palindromes in the group extend with a_i and become suffix palindromes of $\mathsf{str}(r, v_i)$. We update $s \leftarrow s + 2$. The values of d and t stay unchanged.
2. $a_i = b$ and $a_i \neq c$. In this case, all the suffix palindromes but the longest one in the group extend with a_i and become suffix palindromes of $\mathsf{str}(r, v_i)$. We update $s \leftarrow s + 2$ and $t \leftarrow t - 1$. The value of d stays unchanged.
3. $a_i \neq b$ and $a_i = c$. In this case, only the longest suffix palindromes in the group extends with a_i and becomes a suffix palindrome of $\mathsf{str}(r, v_i)$. We first update $s \leftarrow s + (t-1)d + 2$ and then $t \leftarrow 1$. The new value of d is easily calculated from the length of the longest suffix palindrome in the previous group (recall the definition of d just above Lemma 1).
4. $a_i \neq b$ and $a_i \neq c$. In this case, none of the members in the group extends with a_i. Then, we do nothing.

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

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

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

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

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

We have shown the following:

Theorem 1. *We can compute all maximal palindromes in a given trie $\mathcal{T}$ in $O(N \log h)$ time and $O(h)$ working space, where N and h respectively denote the number of edges in $\mathcal{T}$ and the height of $\mathcal{T}$.*

Remark 1. Note that for a balanced trie with $h = \Theta(\log_\sigma N)$, our algorithm runs in $O(N \log\log_\sigma N)$ time with $O(\log_\sigma N)$ working space. In the worst case where $h = \Theta(N)$, our algorithm still runs in $O(N \log N)$ time with $O(N)$ space.

4.2 Alternative algorithm based on Manacher's algorithm

In this subsection, we present an alternative algorithm for computing all maximal palindromes in a given trie $\mathcal{T}$ that is based on Manacher's algorithm [26] that is originally designed for computing maximal palindromes in a single string.

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

The basic strategy of our alternative algorithm is as follows. We perform a depth first traversal on $\mathcal{T}'$, where only the root, branching internal nodes, and leaves are explicitly visited. Let u be any branching node visited in the traversal. As was done in the algorithm of Section 4.1, for each branching node v in the path from the root r to u, we maintain the arithmetic progressions representing the suffix palindromes ending at v, which will be used when the traversal traces back to these branching nodes.

Now we are processing node u to extend the suffix palindromes. For this sake, we use the idea of Manacher's algorithm [26]. Let Σ_u be the set of the first characters of the out-edges of u in $\mathcal{T}'$. For each $a \in \Sigma_u$, $e_a = (u, v_a)$ denote the out-edge of u in $\mathcal{T}'$ whose label begins with a. For each $a \in \Sigma_u$ (in any order), we search for the groups of the suffix palindromes of $\mathsf{str}(r, u)$ that are immediately preceded by a, since these will be the only groups that will extend with the edge e_a. Let $\mathbf{P}_a$ be the set of suffix palindromes extended with a (which are represented by $O(\log h)$ arithmetic progressions). For each $1 \le i \le |\mathbf{P}_a|$, let P_i denote the ith longest suffix palindrome in $\mathbf{P}_a$. While we move forward on the edge e_a, we keep two invariants ℓ and f such that P_ℓ denotes the longest suffix palindrome whose extension ends with the currently processed character on e_a, and P_f denotes the suffix palindrome whose extension is to be determined by symmetry of P_ℓ. We process the suffix palindromes in $\mathbf{P}_a$ in decreasing order of their lengths, by picking up their lengths from the arithmetic progressions. Namely, we initially set $\ell \leftarrow 1$ and $f \leftarrow 2$ and increase the values of ℓ and f accordingly while reading the characters on the edge e_a. In any following step $\ell \le f$ will hold.

When $\ell = 1$, as a initial step, we extend the left arm of P_ℓ on the reversed path and the right arm of P_ℓ on the path from u to v_a with naïve character comparisons. Now suppose we are processing P_ℓ. Let $s = |P_\ell|$, c be the center of P_ℓ in the path string from the root, and τ be the length of the extension of P_ℓ, namely, P_ℓ has been extended to a maximal palindrome of length $s + 2\tau$ for center c. This means that the maximal palindromes for any centers less than c in the path from the root to u have already been computed. Then we process P_f. Let $s' = |P_f|$ and c' be the center for P_f. There are three possible cases:

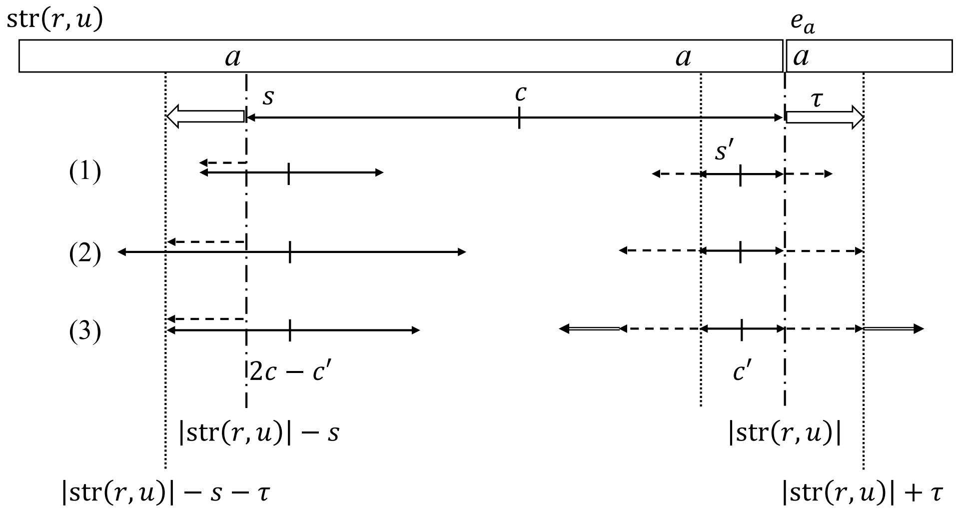


Figure 3. Illustration for our alternative algorithm that computes maximal palindromes in a given trie, that is based on Manacher's algorithm.

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

See Figure 3 for illustration of the above three cases.

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

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

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

Let us analyze the complexities of this method. Consider each branching node u in $\mathcal{T}'$. For each $a \in \Sigma_a$, we can find the arithmetic progressions representing $\mathbf{P}_a$ in

$O(\log h)$ time as in the previous subsection. Each character in edge e_a is involved in exactly one character comparison. To perform each character comparison on the trie in $O(1)$ time, we preprocess the original trie $\mathcal{T}$ with N edges in $O(N)$ time and space so that *level ancestor queries* on the trie can be answered in $O(1)$ time each [4]. Hence, if N' is the number of edges in the path-contracted trie $\mathcal{T}'$, then our algorithm of this section runs in $O(N' \log h + N)$ time and $O(N)$ space.

Theorem 2. *We can compute all maximal palindromes in a given trie $\mathcal{T}$ in $O(N' \log h + N)$ time and $O(N)$ working space, where N and h respectively denote the number of edges in $\mathcal{T}$ and the height of $\mathcal{T}$, and N' denotes the number of edges in the path-contracted trie $\mathcal{T}'$.*

Remark 2. Note that $N' \leq N$ always holds, and therefore the algorithm of Theorem 2 is at least as fast as the algorithm of Theorem 1. Moreover, in case where $N' = O(N/\log h)$ (which happens when the average length of the unary paths in $\mathcal{T}$ is $\Omega(\log h)$), then the algorithm of Theorem 2 runs in $O(N)$ time.

5 Computing distinct palindromes in a trie

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

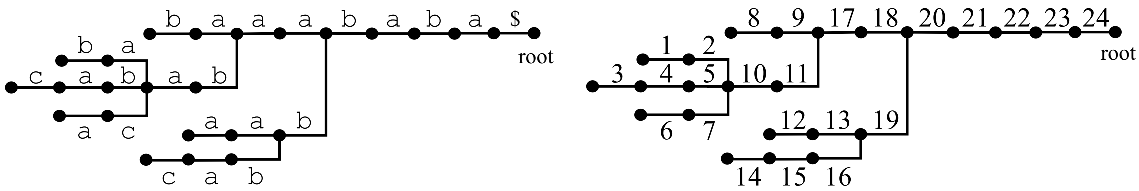
Our algorithm is based on Groult et al.'s [17] that finds distinct palindromes in a single string. Recall the proof of Lemma 3 in Section 3. There we showed that for each node u in a trie $\mathcal{T}$, only the longest suffix palindrome of $\mathsf{str}(r, u)$ can be accounted for as a distinct palindrome, where r is the root of $\mathcal{T}$. Let N and h be the number of edges in $\mathcal{T}$ and the height of $\mathcal{T}$. In this section, we assume that the root has a single out-edge labeled with a special character \$ that does not appear elsewhere in the trie and is lexicographically the smallest.

Lemma 4. *For each node u in a given trie $\mathcal{T}$, we can compute the longest suffix palindrome of $\mathsf{str}(r, u)$ in a total of $O(N' \log h + N)$ time with $O(N)$ working space, where N' denotes the number of edges in the path-contracted trie $\mathcal{T}'$.*

Proof. Clear from our algorithm to compute maximal palindromes in $\mathcal{T}$ which was presented in Section 4. □

Now, we consider the *reversed* trie $\mathcal{T}^R$. For any reversed path from u to u' in $\mathcal{T}^R$ in the leaf-to-root direction, let $(u, u') = \mathsf{str}(u', u)^R$. Observe that a suffix of $\mathsf{str}(r, u)$ is a prefix of $\mathsf{rev\_str}(u, r)$. Therefore, a suffix palindrome of $\mathsf{str}(r, u)$ that ends at node u in $\mathcal{T}$ is a prefix palindrome of $\mathsf{rev\_str}(u, r)$ that begins at node u in the reversed trie $\mathcal{T}^R$. For each $1 \leq j \leq N$, let e_j denote the $(N-j+1)$th visited edge in a breadth-first traversal on the original trie $\mathcal{T}$. The *id* of edge e_j is j. See Figure 4 for examples of a reversed trie and the associated integers to its edges.

For each edge id j, let $e_j = (v_j, u_j)$ be the corresponding reversed edge. Let *LPrePal* be an array of length N such that for each $1 \leq j \leq N$ $LPrePal[j]$ stores the length of the longest prefix palindrome in the reversed path string beginning with e_j (namely $\mathsf{rev\_str}(v_j, r)$). Also, let *LFF* be an array of length N called the *longest following factor array*, such that for each $1 \leq i \leq N$ $LFF[j]$ stores the length of the longest prefix of $\mathsf{rev\_str}(v_j, r)$ that occurs as a prefix of $\mathsf{rev\_str}(v_k, r)$ with $k > j$. See Figure 4 for examples of *LPrePal* and *LFF* arrays.



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

Figure 4. Upper left: An example of a reversed trie. Upper right: The edge id's based on a breadth-first traversal. Lower: *SA*, *LCP*, *LFF* and *LPrePal* arrays built on the reversed trie shown above.

We design an algorithm that reports a shallowest occurrence of each distinct palindrome in the (reversed) trie. If there are multiple occurrences of the same palindrome beginning at nodes on the same depth, then we report the occurrence that begins with the edge with the largest id. Now we can see that for each j, the occurrence of the longest prefix palindrome of $\mathsf{rev\_str}(v_j, r)$ should be reported iff $LFF[j] < LPrePal[j]$. Hence, we can report all distinct palindromes in the trie in $O(N)$ time by simply scanning the two arrays *LFF* and *LPrePal* from left to right. The *LFF* array can be computed in $O(N)$ time from the *LCP* array for the trie, by using the same technique for the longest previous factor array (LPF array) for a single string [8]. Together with Theorem 2, we obtain the following:

Theorem 3. *We can compute all distinct palindromes in a given trie* $\mathcal{T}$ *in* $O(N' \log h + N)$ *time and* $O(N)$ *working space, where* N *and* h *respectively denote the number of edges in* $\mathcal{T}$ *and the height of* $\mathcal{T}$*, and* N' *denotes the number of edges in the path-contracted trie* $\mathcal{T}'$*.*

Remark 3. The suffix array of the reversed trie with N edges can be constructed in $O(N)$ time and space if the edge labels are drawn from a constant-size alphabet or an integer alphabet of polynomial size in N [36]. In the case of a general ordered alphabet of size σ, the suffix array of the reversed trie can be constructed in $O(N \log \sigma)$ time and space [5]. The other arrays can be constructed in $O(N)$ time after the suffix array has been built. In summary, our algorithm runs in $O(N' \log h + N \log \sigma)$ time and $O(N \log \sigma)$ working space in the case of a general ordered alphabet.

Acknowledgements

This work was supported by JSPS Grant Numbers JP18K18002 (YN), JP17H01697 (SI), JP16H02783 (HB), and JP18H04098 (MT).

References

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

2. A. Apostolico, D. Breslauer, and Z. Galil: *Parallel detection of all palindromes in a string*. Theoretical Computer Science, 141 1995, pp. 163–173.
3. M. A. Bender and M. Farach-Colton: *The LCA problem revisited*, in LATIN 2000, 2000, pp. 88–94.
4. M. A. Bender and M. Farach-Colton: *The level ancestor problem simplified*. Theor. Comput. Sci., 321(1) 2004, pp. 5–12.
5. D. Breslauer: *The suffix tree of a tree and minimizing sequential transducers*. Theoretical Computer Science, 191(1–2) January 1998, pp. 131–144.
6. S. Brlek, N. Lafrenière, and X. Provençal: *Palindromic complexity of trees*, in Proc. DLT 2015, 2015, pp. 155–166.
7. M. Bucci, A. D. Luca, A. Glen, and L. Q. Zamboni: *A new characteristic property of rich words*. Theor. Comput. Sci., 410(30-32) 2009, pp. 2860–2863.
8. M. Crochemore and L. Ilie: *Computing longest previous factor in linear time and applications*. Inf. Process. Lett., 106(2) 2008, pp. 75–80.
9. X. Droubay, J. Justin, and G. Pirillo: *Episturmian words and some constructions of de Luca and Rauzy*. Theor. Comput. Sci., 255(1-2) 2001, pp. 539–553.
10. M. Farach-Colton, P. Ferragina, and S. Muthukrishnan: *On the sorting-complexity of suffix tree construction*. J. ACM, 47(6) 2000, pp. 987–1011.
11. P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan: *Compressing and indexing labeled trees, with applications*. J. ACM, 57(1) 2009.
12. N. Fujisato, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda: *The parameterized position heap of a trie*, in CIAC 2019, 2019, pp. 237–248.
13. M. Funakoshi, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda: *Longest substring palindrome after edit*, in CPM 2018, 2018, pp. 12:1–12:14.
14. P. Gawrychowski, T. I, S. Inenaga, D. Köppl, and F. Manea: *Tighter bounds and optimal algorithms for all maximal α-gapped repeats and palindromes - finding all maximal α-gapped repeats and palindromes in optimal worst case time on integer alphabets*. Theory Comput. Syst., 62(1) 2018, pp. 162–191.
15. P. Gawrychowski, T. Kociumaka, W. Rytter, and T. Walen: *Tight bound for the number of distinct palindromes in a tree*, in Proc. SPIRE 2015, 2015, pp. 270–276.
16. A. Glen, J. Justin, S. Widmer, and L. Q. Zamboni: *Palindromic richness*. Eur. J. Comb., 30(2) 2009, pp. 510–531.
17. R. Groult, É. Prieur, and G. Richomme: *Counting distinct palindromes in a word in linear time*. Inf. Process. Lett., 110(20) 2010, pp. 908–912.
18. D. Gusfield: *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, 1997.
19. D. Harel and R. E. Tarjan: *Fast algorithms for finding nearest common ancestors*. SIAM J. Comput., 13(2) 1984, pp. 338–355.
20. S. Inenaga: *Suffix trees, DAWGs and CDAWGs for forward and backward tries*. CoRR, abs/1904.04513 2019.
21. S. Inenaga, H. Hoshino, A. Shinohara, M. Takeda, and S. Arikawa: *Construction of the CDAWG for a trie*, in Proc. PSC 2001, 2001, pp. 37–48.
22. D. Kimura and H. Kashima: *A linear time subpath kernel for trees*, in IEICE Technical Report, vol. IBISML2011-85, 2011, pp. 291–298.
23. R. Kolpakov and G. Kucherov: *Searching for gapped palindromes*. Theor. Comput. Sci., 410(51) 2009, pp. 5365–5373.
24. S. R. Kosaraju: *Efficient tree pattern matching (preliminary version)*, in Proc. FOCS 1989, 1989, pp. 178–183.
25. D. Kosolobov, M. Rubinchik, and A. M. Shur: *Finding distinct subpalindromes online*, in PSC 2013, 2013, pp. 63–69.
26. G. Manacher: *A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string*. Journal of the ACM, 22 1975, pp. 346–351.
27. W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura, and K. Hashimoto: *Efficient algorithms to compute compressed longest common substrings and compressed palindromes*. Theor. Comput. Sci., 410(8–10) 2009, pp. 900–913.
28. W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura, and K. Hashimoto: *Efficient algorithms to compute compressed longest common substrings and compressed palindromes*. Theor. Comput. Sci., 410(8-10) 2009, pp. 900–913.

29. M. Mohri, P. J. Moreno, and E. Weinstein: *General suffix automaton construction algorithm and space bounds.* Theor. Comput. Sci., 410(37) 2009, pp. 3553–3562.
30. T. Nakamura, S. Inenaga, H. Bannai, and M. Takeda: *Order preserving pattern matching on trees and dags*, in Proc. SPIRE 2017, 2017, pp. 271–277.
31. Y. Nakashima, T. I, S. Inenaga, H. Bannai, and M. Takeda: *Constructing LZ78 tries and position heaps in linear time for large alphabets.* Inf. Process. Lett., 115(9) 2015, pp. 655–659.
32. S. Narisada, Diptarama, K. Narisawa, S. Inenaga, and A. Shinohara: *Computing longest single-arm-gapped palindromes in a string*, in SOFSEM 2017, 2017, pp. 375–386.
33. A. H. L. Porto and V. C. Barbosa: *Finding approximate palindromes in strings.* Pattern Recognition, 35 2002, pp. 2581–2591.
34. B. Schieber and U. Vishkin: *On finding lowest common ancestors: Simplification and parallelization.* SIAM J. Comput., 17(6) 1988, pp. 1253–1262.
35. T. Shibuya: *Constructing the suffix tree of a tree with a large alphabet.* IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E86-A(5) 2003, pp. 1061–1066.
36. T. Shibuya: *Constructing the suffix tree of a tree with a large alphabet.* IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E86-A(5) 2003, pp. 1061–1066.
37. R. Sugahara, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda: *Computing runs on a trie*, in Proc. CPM 2019, 2019, pp. 23:1–23:11.
38. P. Weiner: *Linear pattern matching algorithms*, in 14th Annual Symposium on Switching and Automata Theory, 1973, pp. 1–11.