Tight Bound for the Number of Distinct Palindromes in a Tree*

Paweł Gawrychowski
Institute of Computer Science
University of Wrocław
Poland
gawry@cs.uni.wroc.pl

Tomasz Kociumaka
Department of Computer Science
Bar-Ilan University
Ramat Gan, Israel
kociumaka@mimuw.edu.pl

Wojciech Rytter Tomasz Waleń
Institute of Informatics
University of Warsaw
Poland
{rytter,walen}@mimuw.edu.pl

November 30, 2020

Abstract

For an undirected tree with n edges labelled by single letters, we consider its substrings, which are labels of the simple paths between pairs of nodes. A palindrome is a word w such that $w = w^R$, where w^R denotes the reverse of w. We prove that $PAL(n) = \mathcal{O}(n^{1.5})$, where $PAL(n)$ denotes the number of distinct palindromic substrings in a tree of size n. This solves an open problem of Brlek, Lafrenière, and Provençal (DLT 2015 [4]), who showed that $PAL(n) = \Omega(n^{1.5})$. Hence, we settle the tight bound of $\Theta(n^{1.5})$ for the maximum palindromic complexity of trees. For standard strings, i.e., for itrees which are simple paths, the palindromic complexity is exactly $n + 1$.

We also propose $\mathcal{O}(n^{1.5}\log n)$-time algorithm for reporting all distinct palindromes and $O(n\,polylogn)$ time algorithm for palindrom testing and finding the longest palindrom in a tree.

1 Introduction

Regularities in words are extensively studied in combinatorics and text algorithms. One of the basic types of such structures are palindromes: symmetric words, the ones which are the same when read in both directions. The *palindromic complexity* of a word is the number of distinct palindromic substrings

*This is a full version of a paper presented at SPIRE 2015 [11].

in the word. An elegant argument shows that the palindromic complexity of a word of length n does not exceed $n+1$ [8], which is already attained by a unary word $\mathtt{a}^n$. Therefore the problem of palindromic complexity for words is completely settled, and a natural next step is to generalize it to trees.

In this paper, we consider the palindromic complexity of undirected trees with edges labelled by single letters. We define substrings of such a tree as the labels of simple paths between arbitrary two nodes. Each label is the concatenation of the labels of all edges on the path. Denote by $pals(T)$ the set of all palindromic substrings of a tree T and by $PAL(n)$ the maximum value of $pals(T)$ over all trees with n edges.

Fig. 1 illustrates palindromic substrings in a sample tree. Note that palindromes in a word of length n naturally correspond to palindromic substrings in a path of n edges.

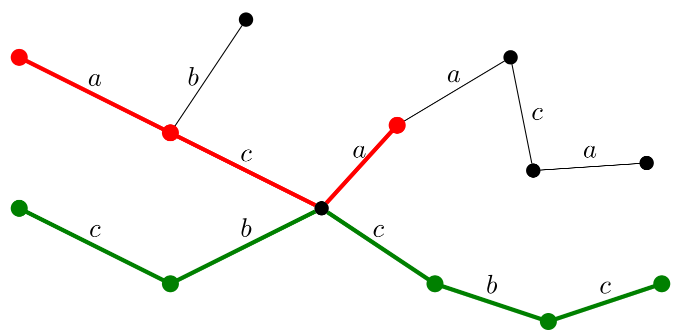

Figure 1: A sample tree T. We have $pals(T) = \{$ a, b, c, aa, aca, $acaaca$, bcb, $bccb$, $caac$, cbc, $cbcbc$, cc $\}$. An occurrence of a palindrome aca is marked red, and an occurrence of a palindrome $cbcbc$ is marked green.

The study of the palindromic complexity of trees was recently initiated by Brlek, Lafrenière, and Provençal [4], who constructed a family of trees with n edges containing $\Theta(n^{1.5})$ distinct palindromic substrings. They conjectured that there are no trees with asymptotically larger palindromic complexity and proved this claim for a special subclass of trees.

Our Result We show that $PAL(n) = \mathcal{O}(n^{1.5})$. This bound is tight by the construction given in [4]; hence, we completely settle the asymptotic maximum palindromic complexity for trees. We also provide $\mathcal{O}(n^{1.5}\log n)$ algorithm for reporting all distinct palindromes and $O(n\,polylogn)$ time algorithm for palindrom testing and finding the longest palindrom in a tree.

Related Work Palindromic complexity of words was studied in various aspects. This includes algorithms determining the complexity [14], bounds on the average complexity [1], and generalizations to circular words [18]. Finite and infinite palindrome-rich words received particularly high attention;

see e.g. [3, 8, 12]. This class contains, for example, all episturmian and thus all Sturmian words [8].

Recently, some almost exact bounds for the number of distinct palindromes in star-like trees have been shown by Glen et al. [13]. Also the palindromes in directed trees have been studied by Funakoshi et al. [10] who presented $\mathcal{O}(n \log h)$ time algorithm to compute all maximal palindromes and all distinct palindromes in a TRIE T of height h.

In the setting of labelled trees, other kinds of regularities were also studied. It has been shown that a tree with n edges contains $\mathcal{O}(n^{4/3})$ distinct squares [6] and $\mathcal{O}(n)$ distinct cubes [17]. Both bounds are known to be tight. Interestingly, the lower bound construction for squares resembles that for palindromes [4].

Outline of the Paper In Section 2, we introduce basic terminology and combinatorial toolbox. Next, in **??**, we quickly summarize previously known results for the lower bounds on the number of distinct palindromes.

In Section 3.2, we introduce the special family of the trees called *spine trees* and prove th upper bounds for those trees. In Section 3.3, we show how every tree can be decomposed into spine trees, and in the **??**, we combine those results to obtain the upper bound on the number of distinct palindromes.

In Section 4, we introduce algorithmic toolbox and provide an algorithm for reporting all distinct palindromes.

2 Preliminaries

A word w is a sequence of characters $w[1], w[2], \dots, w[|w|] \in \Sigma$, often denoted $w[1..|w|]$. A substring of w is any word of the form $w[i..j]$, and if $i = 1$ ($j = |w|$), then it is called a prefix (a suffix, respectively). A period of w is an integer p, $1 \leq p \leq |w|$, such that $w[i] = w[i+p]$ for $i = 1, 2, \dots, |w| - p$. The shortest period of w, denoted $\mathrm{per}(w)$, is the smallest such p.

2.1 Some combinatorics of words

The following well known periodicity lemma, characterizes the properties of periods.

Lemma 2.1 (Periodicity Lemma [9])**.** *If p, q are periods of a word w of length $|w| \geq p + q - gcd(p, q)$, then $gcd(p, q)$ is also a period of w.*

The following lemma is a straightforward consequence of the Periodicity Lemma (Lemma 2.1).

Lemma 2.2. *Suppose a word v is a substring of a longer word u which has a period $p \leq \frac{1}{2}|v|$. Then* $\mathrm{per}(u) = \mathrm{per}(v)$.

Proof. Let us assume that $p_u = \mathrm{per}(u)$, $p_v = \mathrm{per}(v)$ and $p_u \neq p_v$. Since v is a substring of u, and p is a period of both u, we have clearly that

$$p_v \leq p_u \leq p \leq \frac{1}{2}|v|.$$

The word v and periods p_v and p_u met the conditions of the Periodicity Lemma, so the $gcd(p_v, p_u)$ is also a period of v.

Since p_v is the minimal period, the $p_u = a \cdot p_v$ for some $a > 1$. But in such case the p_v is also a period of whole word u — contradiction. $\square$

We have the following connection between periods and palindromes.

Observation 2.3. *Suppose a palindrome v is a suffix of a longer palindrome u. Then v is a prefix of u and thus $|u| - |v|$ is a period of u and of v.*

2.2 Centroid decomposition

For a tree T and its node r denote by $pals(T, r)$ the set of palindromic substrings of T corresponding to simple paths containing the node r.

Lemma 2.4. *If $|pals(T, r)| = O(|T|^{1+\epsilon})$, for $\epsilon > 0$, then $PAL(n) = O(n^{1+\epsilon}$.*

Proof. We follow the approach from [6]. We use the folklore fact that every tree T on n edges contains a *centroid* node r such that every component of $T \setminus \{r\}$ is of size at most $\frac{n}{2}$. We separately count palindromic substrings corresponding to the paths going through the centroid r and paths fully contained in a single component of $T \setminus \{r\}$. Finally, we obtain the following recurrence for $\mathsf{pal}(n)$, the maximum number of palindromes in a tree with n edges:

$$PAL(n) = \mathcal{O}(n^{1+\epsilon}) + \max \left\{ \sum_i PAL(n_i) : \forall_i \, n_i \leq \tfrac{n}{2} \text{ and } \sum_i n_i < n \right\}.$$

It solves to $PAL(n) = \mathcal{O}(n^{1+\epsilon})$. $\square$

2.3 D-Trees

For a tree T and its node r denote by $pals(T, r)$ the set of palindromic substrings of T corresponding to simple paths containing the node r. We consider directed acyclic graphs, named D-trees, such that $pals(T, r)$ is a subset of palindromic strings corresponding to simple directed paths in such graphs containg the node r.

Define a *double tree* $\mathcal{D} = (T_\ell, T_r, r)$ as a labelled tree consisting of two trees T_ℓ and T_r sharing a common root r but otherwise disjoint. The edges of T_ℓ and T_r are directed to and from r, respectively. The size of $\mathcal{D}$ is defined as $|\mathcal{D}| = |T_\ell| + |T_r|$.

For any $u, v \in \mathcal{D}$, we denote by $path(u, v)$ the path drom u to v and by $\text{val}(u, v)$ denote the sequence of the labels of edges on this path. We say that a path is palindromic if it corresponds to a palindromic word. Denote by $pals(D)$ the set of palindromic substrings of a D-tree D.

A *substring* of $\mathcal{D}$ is any word $\text{val}(u, v)$ such that $u \in T_\ell$ and $v \in T_r$. Let

$$\text{dist}(u, v) = |\text{val}(u, v)| \text{ and } \text{per}(u, v) = \text{per}(\text{val}(u, v))$$

We consider only *deterministic* double trees (*D-trees*, in short), meaning that all the edges outgoing from a node have distinct labels, and similarly all the edges incoming into a node have distinct labels. An example of such a double tree is shown in Fig. 2.

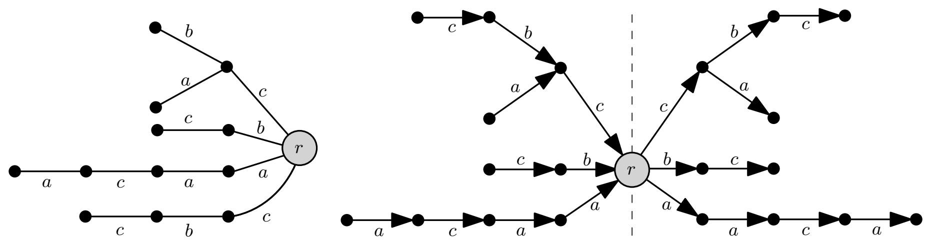

Figure 2: To the left: an example undirected tree with $pals(T, r)$ containing 9 palindromic substrings of length 2 or more *bcb*, *bccb*, *aca*, *cbc*, *caac*, *cc*, *cbcbc*, *aa*, *acaaca*. To the right: D-tree $\Psi(T, r)$ obtained after rooting the tree at r, merging both subtrees connected to r with edges labelled by c, and duplicating the resulting tree.

Trees ⇒ D-Trees. For a tree T and its node r we construct a D-tree $\Psi(T, r) = (T_l, T_r, r)$ in the following way.

We root T at r directing all the edges so that they point towards the root and then determinize the resulting tree by gluing together two children of the same node whenever their edges have the same label. Finally, we create a D-tree by duplicating the tree and changing the directions of the edges in the second copy; see Fig. 2 for a sample application of this process.

It is easy to see that for any simple path from u to v going through r in the original tree we can find $u' \in T_\ell$ and $v' \in T_r$ such that $\mathrm{val}(u, v) = \mathrm{val}(u', v')$. It implies the following fact.

Fact 2.5. $pals(T, r) \subseteq pals(\Psi(T, r))$.

3 Proof of $O(n^{1.5})$ Upper Bound

Due to Lemma 2.4 it is enough to consider only palindromic paths passing through a fixed node r of the tree. Then, due to Fact 2.5 this is reduced to the estimation of palindroms in a D-tree, which is easier.

We consider a D-tree (T_l, T_r, r). A directed palindromic subpath $path(u', v')$ of $path(u, v)$ is called its *central part* iff $dist(u, u') = dist(v', v)$ and $u' = r$ or $v' = r$. The end-nodes of the central part are called *paired* nodes.

The crucial role in our proof of the upper bound play D-trees called spine-trees. A *spine-tree* is a D-tree with a distinguished path, called *spine*, joining vertices $s_\ell \in T_\ell$ and $s_r \in T_r$. Additionally, we insist that this path cannot be extended preserving the period $p = \text{per}(s_\ell, s_r)$.

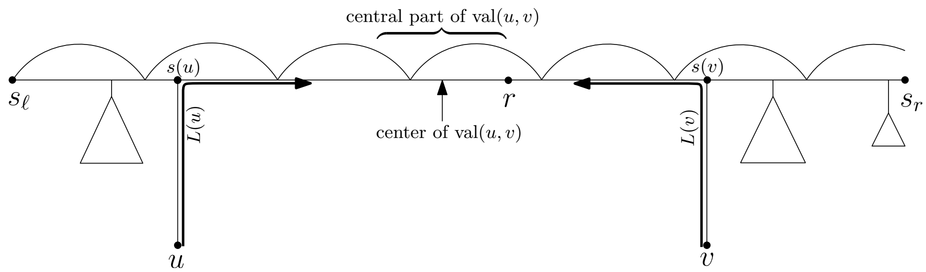

Figure 3: A spine-tree, whose spine is the path from s_ℓ to s_r, with an induced palindrome $\text{val}(u, v)$. Observe that $L(u) = L(v)$ is a prefix of the palindrome. Note that $d(s(u), r) \geq p$ but $d(r, s(v))$ might be smaller than p.

By symmetry of the counting problem (up to edge reversal in a double tree), we consider later only palindromic paths $path(u, v)$ such that $\text{dist}(u, r) \geq \text{dist}(r, v)$. The right end-point of the central part of each such path is the root.

3.1 Combinatorial outline

A palindromic substring is *induced* by a spine-tree if its central part is a fragment of the spine of length at least p, where p is the period of the spine; see Fig. 3 for an example.

The structure of the proof is described informally as follows.

- We show that the number of palindromes induced by a spine-tree is $O(n^{1.5})$.
- We partition set of palindromes into so called *middle* palindromes and others. The number of latter ones is easily estimated to be small.
- The D-tree is partitioned into smaller D-trees, each with a distinguished spine. The total size of all these D-subtrees is linear.
- Then we show that the set of all middle palindromes in a D-tree is a subset of the union of palindromes induced by smaller spine D-subtrees.
- Now the upper bound of all palindromes in a D-tree follows from the upper bound on induced palindromes.

3.2 Number of palindromes induced by a spine tree

For a node u of the spine-tree, let $s(u)$ denote the nearest node of the spine (if u is already on the spine, then $u = s(u)$). Since the spine-tree is deterministic, it satisfies the following property.

Fact 3.1. *For any induced palindrome* $\mathrm{val}(u, v)$*, the path* $\mathrm{val}(s(u), s(v))$ *is an inclusion–maximal fragment of* $\mathrm{val}(u, v)$ *admitting period* p*.*

Lemma 3.2. *There are up to* $n\sqrt{n}$ *distinct palindromic substrings induced by a given spine-tree of size* n*.*

Proof. Define the *label* $L(u)$ for a node $u \in T_\ell$ as the prefix of $\mathrm{val}(u, s_r)$ of length $\mathrm{dist}(u, s(u)) + p$. Similarly, the label $L(v)$ of a node $v \in T_r$ is the reversed suffix of $\mathrm{val}(s_\ell, v)$ of length $p + \mathrm{dist}(s(v), v)$. We leave the label undefined if $\mathrm{val}(u, s_r)$ or $\mathrm{val}(s_\ell, v)$ is not sufficiently long, i.e., if $d(s(u), s_r) < p$ or $d(s_\ell, s(v)) < p$.

Consider a palindrome $\mathrm{val}(u, v)$ induced by the spine-tree. Fact 3.1 implies that that the fragment $\mathrm{val}(s(u), s(v))$ is a maximal fragment of $\mathrm{val}(u, v)$ with period p. Since the central part of the palindrome is of length at least p and lies within this fragment, the fragment must be symmetric, i.e., we must have $\mathrm{dist}(u, s(u)) = \mathrm{dist}(s(v), v)$, and the labels of u and v are both defined.

Consequently, $|L(u)| = |L(v)|$ and actually the labels $L(u)$ and $L(v)$ are equal. Hence, to bound the number of distinct palindromes, we group together nodes with the same labels. Let V_L be the set of vertices of $T_\ell \cup T_r$ with label L. We have the following claim.

Claim 3.3. *For any label* L*, there are at most* $\min(|V_L|^2, n)$ *distinct induced palindromes with endpoints in* V_L*.*

Proof. (of the claim)
Consider all distinct induced palindromes $\mathrm{val}(u, v)$ such that $L(u) = L(v) = L$. A substring is uniquely determined by the endpoints of its occurrence, so $|V_L|^2$ is an upper bound on the number of these palindromes.

We claim that every such palindrome is also uniquely determined by its length, which immediately gives the upper bound of n. Indeed, $\mathrm{dist}(u, s(u)) = \mathrm{dist}(s(v), v) = |L| - p$ and $\mathrm{val}(s(u), s(v))$ has period p, so if the length is known, then $\mathrm{val}(s(u), s(v))$ can be recovered from its prefix of length p, i.e., the suffix of L of length p. □

The sets V_L are disjoint, so by the above claim and using the inequality $\min(x, y) \le \sqrt{xy}$, the number of distinct palindromes induced by the spine-tree is at most:

$$\sum_L \min(|V_L|^2, n) \le \sum_L \sqrt{|V_L|^2 \cdot n} \le \sqrt{n} \cdot \sum_L |V_L| \le n^{1.5}. \qquad \square$$

3.3 Number of all palindromes

Consider a node $u \in T_\ell$ and all distinct palindromes $P_1, \ldots, P_k$ with an occurrence starting at u. Observe that their central parts $C_1, \ldots, C_k$ have distinct lengths: indeed, $|P_i| = 2\mathrm{dist}(u, r) - |C_i|$ and $\mathrm{dist}(u, r) \ge \frac{1}{2}|P_i|$, so $\mathrm{val}(u, r)$ and $|C_i|$ determines the whole palindrome P_i. Hence, we can order these palindromes so that $|C_1| > \ldots > |C_k|$, (i.e., $|P_1| < \ldots < |P_k|$).

Denote $\alpha = 2\sqrt{n}$. Palindromes $P_{2\alpha+1}, \ldots, P_{k-\alpha}$ are called *middle palindromes*. There are $\mathcal{O}(\sqrt{n})$ remaining palindromes for fixed u and $\mathcal{O}(n^{1.5})$ in total, so we can focus on counting middle palindromes. We start with the following characterization.

Lemma 3.4. *Consider middle palindromes P_i starting at a given node u. Central parts of these palindromes satisfy $|C_i| \geq \alpha$ and $\mathrm{per}(C_i) \leq \frac{1}{2}\sqrt{n}$. Moreover, for each P_i extending the central part C_i by α characters in each direction preserves the shortest period.*

Proof. Since we excluded the α palindromes with the shortest central parts, the middle palindromes clearly have central parts of length at least α.

Let us now prove that $\mathrm{per}(C_\alpha) \leq \frac{1}{2}\sqrt{n}$.
By Observation 2.3, $|C_j| - |C_{j+1}|$ is a period of C_j for $1 \leq j \leq \alpha$.
Since

$$\sum_{j=1}^{\alpha}(|C_j| - |C_{j+1}|) < |C_1| \leq n,$$

for some j we have

$$\mathrm{per}(C_j) \leq |C_j| - |C_{j+1}| \leq \tfrac{1}{2}\sqrt{n}.$$

Moreover, C_α is a suffix of C_j, so the claim follows.

For $i > 2\alpha$ (in particular, if P_i is a middle palindrome), C_i is a suffix of C_α. Additionally, for $2\alpha < i \leq k - \alpha$ we can observe that $\mathrm{per}(C_\alpha) \leq \frac{1}{2}|C_i|$ since $\mathrm{per}(C_\alpha) \leq \frac{1}{2}\sqrt{n}$ and $|C_i| \geq \alpha$.

Hence, we can apply Lemma 2.2 that implies $\mathrm{per}(C_i) = \mathrm{per}(C_\alpha)$.
Moreover,

$$|C_i| \leq |C_\alpha| + \alpha - i < |C_\alpha| - \alpha,$$

so extending C_i by α characters to the left preserves the period. By symmetry of P_i, the extension to the right also preserves the period. $\square$

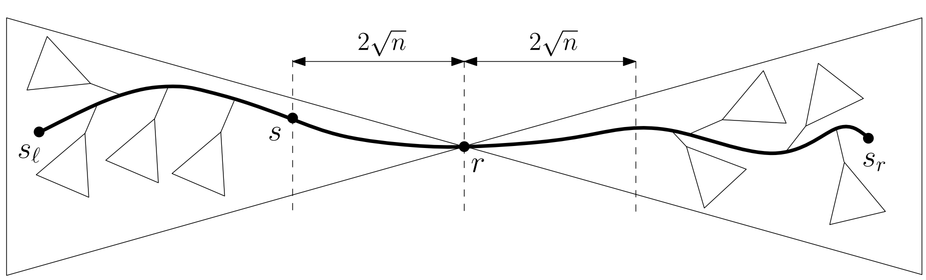

Figure 4: A spine-tree constructed for a vertex s in a D-tree. Note that we do not attach subtrees at distance less than α from the root.

Let us choose any $s \in T_\ell$ such that

$$\mathrm{dist}(s,r) = \alpha \text{ and } \mathrm{per}(s,r) \le \frac{1}{2}\sqrt{n}.$$

Then, extend the period of $\mathrm{val}(s,r)$ to the left and to the right as far as possible, arriving at nodes s_ℓ and s_r, respectively.

We create a spine-tree with spine corresponding to the path from s_ℓ to s_r as shown in Fig. 4. We attach to the spine all subtrees hanging off the original path at distance at least α from the root. In other words, a vertex $u \in T_\ell$ which does not belong the spine is added to the spine-tree if $\mathrm{dist}(s(u),r) \ge \alpha$ and a vertex $v \in T_r$ — if $\mathrm{dist}(r,s(v)) \ge \alpha$. If $\mathrm{dist}(r,s_r) < \alpha$, then this procedure leaves no subtrees hanging in T_r so we do not create any spine-tree for s.

Now, let us consider a middle palindrome. By Lemma 3.4, its central part satisfies $|C| \ge \alpha$ and $\mathrm{per}(C) \le \frac{1}{2}\sqrt{n}$. Moreover, by Lemma 2.2, we have $\mathrm{per}(C) = \mathrm{per}(s,r)$ for the unique node $s \in T_\ell$ located within C at distance α from the root.

Consequently, C lies on the spine of the spine-tree created for s, and u belongs to a subtree attached to the spine. Additionally, since C can be extended by α characters in each direction preserving the period, the other endpoint v must also belong to such a subtree in T_r (that is, we have $\mathrm{dist}(r,s(v)) \ge \alpha$). Hence, each middle palindromic substring is induced by some spine-tree.

The spine-trees are not disjoint, but, nevertheless, their total size is small.

Lemma 3.5. *The sizes $n_1,\ldots,n_k$ of the created spine-trees satisfy $\sum_i n_i \le 2n$.*

Proof. We claim that at least $n_i - \alpha$ nodes of the ith spine-tree are disjoint from all the other spine-trees. Let c_i be the node on the spine of the ith spine-tree such that $\mathrm{dist}(c_i,r) = \sqrt{n}$ and similarly let s_i satisfy $\mathrm{dist}(s_i,r) = \alpha$. Recall that $\mathrm{per}(s_i,r) \le \frac{1}{2}\sqrt{n}$. Thus, Lemma 2.2 yields $\mathrm{per}(s_i,r) = \mathrm{per}(c_i,r)$. Since the tree is deterministic, c_i uniquely determines s_i and hence the whole spine-tree. Thus, the nodes c_i are all distinct and so are their predecessors on the spines and all attached subtrees.

A similar argument shows that all nodes d_i on the spine of the ith spine-tree such that $\mathrm{dist}(r,d_i) = \sqrt{n}$ are also all distinct. Therefore, we proved $\sum_i n_i - \alpha \le n$.

Each spine-tree has at least 2α vertices on the spine, so this yields $n_i \ge 2\alpha$, and thus we obtain

$$\sum_i n_i \le 2\sum_i (n_i - \alpha) \le 2n. \qquad \square$$

Lemma 3.6. *Every D-tree of size n has $\mathcal{O}(n^{1.5})$ distinct palindromic substrings.*

Proof. By Lemma 3.2, the number of palindromes induced by the ith spine-tree is at most $n_i^{1.5}$. Accounting the $\mathcal{O}(n^{1.5})$ palindromes which do not occur as middle palindromes, we have

$$\mathcal{O}(n^{1.5}) + \sum_i n_i^{1.5} \leq \mathcal{O}(n^{1.5}) + \sum_i n_i \sqrt{n} = \mathcal{O}(n^{1.5})$$

palindromes in total. □

Due to Lemma 2.4 and Fact 2.5 we obtain.

Theorem 3.7. *A tree with n edges contains $\mathcal{O}(n^{1.5})$ distinct palindromic substrings.*

4 Algorithm Reporting All Distinct Palindromes

In this section, we consider following problem for palindromes in trees:

Problem 4.1 (ReportAll). Given tree a T with n edges, each labelled by single character from the alphabet Σ report all distinct palindromes in T.

There are various ways for reporting palindromes, the natural choice is to represent each palindrome as a pair for nodes (u, v) such that $val(u, v)$ is a palindrome. Unfortunately for efficiency reasons we would like to use slightly different format, each palindrome will be reported as triple (ℓ, u, v), such that ℓ is length of a palindrome, $val(u, v)$ is a first half of a palindrome ($\lceil \frac{\ell}{2} \rceil = |val(u, v)|$).
To simplify the description of the algorithm and introduce restricted version of the problem:

Problem 4.2 (ReportAllEven). Given tree T with n edges, each labelled by single character from the alphabet Σ report all distinct even palindromes in T.

The following lemma states that in fact the problem ReportAllEven is equivalent to the problem ReportAll.

Lemma 4.3. *Given an algorithm for problem* ReportAllEven *running in time $f(n)$, it is possible to solve problem* ReportAll *in $f(O(n))$ time.*

Proof. Given an instance T of the problem ReportAll, we can generate tree T' by replacing each edge (u, v) with label $c \in \Sigma$ from T by a path of length 4 with corresponding labels $\$, c, c, \$$ (where \$ is a character not in Σ). Each palindrome in T has a corresponding palindrome in T'. Also each palindrome of length $4k$ that starts (and ends) with character \$ in T' can be attributed to corresponding palindrome in T. In consequence we can solve the problem ReportAllEven for T' and report only those palindromes in T that corresponds to the palindromes of length $4k$ that start with character \$. □

4.1 Algorithmic tools

Before we describe the algorithm we introduce the algorithmic toolbox used in our algorithm. It is similar to the one described in the Section 3 from [16], but it is tailored to the palindromic case.

Lemma 4.4. *Given a family of D-trees $D_1, \ldots, D_k$ with total n nodes. It can be preprocessed in $\mathcal{O}(n)$ time, such that following operations can be done in $\mathcal{O}(1)$ time:*

- $\mathsf{dist}(u, v)$ – *distance between nodes u and v,*
- $\mathsf{up}(u, h)$ – *node v on a path from u towards the root, at distance h from u,*
- $\mathsf{center}(u, v)$ – *node at the center of path from u and v,*
- $\mathsf{isAncestor}(u, v)$ – *is v an ancestor of u,*
- $\mathsf{perLen}(u)$ – *length of the period of word on path from root to u*

Proof. Queries $\mathsf{dist}(u, v)$ can be implemented by precomputing depth of each node in a tree and using Lowest Common Ancestor Queries (LCA) [15]. Query $\mathsf{up}(u, h)$ is in fact Level Ancestor Query (LA) [2]. Operation $\mathsf{center}(u, v)$ can be realized by one dist and *up* query. Length of periods $\mathsf{perLen}(u)$ can be calculates from the border array P that can be computed in $\mathcal{O}(n)$ time ([16]). □

Lemma 4.5. *Given a family of D-trees $D_1, \ldots, D_k$ with total n nodes. It can be preprocessed in $\mathcal{O}(n \log n)$ time, such that following operations can be done in $\mathcal{O}(1)$ time:*

- $\mathsf{label}(u, v)$ – *pair of integers representing word $val(u, v)$ (this operation is defined only for u being ancestor of v or v being ancestor of u),*
- $\mathsf{isEqual}(u_1, v_1, u_2, v_2)$ – *is $val(u_1, v_1) = val(u_2, v_2)$,*
- $\mathsf{isPalindrome}(u, v)$ – *is word $val(u, v)$ a palindrome?,*
- $\mathsf{exists}(D_i, u, v)$ – *for $D_i = (L_i, R_i, r_i)$ it verifies if there exist a node $w \in R_i$ such that $val(w, r_i) = val(u, v)$ (this operation is defined only for $u, v \in L_i$ and u being ancestor of v or v being ancestor of u),*
- $\mathsf{child}(u, c)$ – *returns child node of u with label c (or null values if it does not exist)*

Proof. Operations label, $\mathsf{isEqual}$ and $\mathsf{isPalindrome}$ can be implemented using Dictionary of Basic Factors (DBF) [7], which clearly requires $\mathcal{O}(n \log n)$ preprocessing time. The only extension is that we need is that for each basic factor we also store code of its reversed version. For operation exists we store DBF codes of all possible values of $val(r, w)$ in a static dictionary with constant lookup time. □

4.2 Algorithm for spine trees

Lemma 4.6. *For a double tree $D = (T_l, T_r, r)$ with n nodes, the spine decomposition can be calculated in $\mathcal{O}(n)$ time, assuming that the D has been preprocessed with Lemma 4.4.*

Proof. The spine decomposition of double tree $D = (T_l, T_r, r)$ We start with calculating set C with nodes from T_l at distance $2\sqrt{n}$ from root with value $\mathsf{perLen}(u) < \frac{1}{2}\sqrt{n}$. Since D has been preprocessed with Lemma 4.4 the set C can be calculated in $\mathcal{O}(n)$ time. We can observe that for any nodes $u_1, u_2 \in C$ ($u_1 \neq u_2$) due to high periodicity of $val(u_1, r)$ and $val(u_2, r)$, the paths (u_1, r) and (u_2, r) have at least $\sqrt{n}$ distinct nodes, so $|C| < \sqrt{n}$.

Next, for each candidate node $u \in C$, we need to verify if it is a part of a spine. Let p a string period of $val(u, r)$, we locate lowest descendant S_l of u such that string period of $val(S_l, r)$ is p. Such node can be located by traversing subtree of u with $child(x, c)$ queries. Similarly we traverse T_r starting from root r to locate lowest node in S_r such that string period of $val(r, S_r)$ is p^R. If $\mathsf{dist}(r, S_r) \geq 2\sqrt{n}$ we add to the result spine tree with spine (S_l, S_r) and all subtrees with distance $\geq 2\sqrt{n}$ attached.

In this procedure only edges in T_r with distance $< \sqrt{n}$ can be visited multiple times, but since $|C| < \sqrt{n}$ the total processing time of such edges is still $\mathcal{O}(n)$. □

For efficient processing spine trees, we need one additional lemma:

Lemma 4.7. **[FFT Application]** *Given two set of integers $A, B \subset [0, \ldots, n]$ the set $A \ominus B = \{a - b : a \in A, b \in B\}$ can be computed in $\mathcal{O}(n \log n)$ time.*

Proof. We define two polynomials

$$f_A(x) = \sum_{i \in A} x^{n+i}, \quad f_B(x) = \sum_{i \in B} x^{n-i}$$

Using FFT we can multiply two polynomials with integer coefficients in time $\mathcal{O}(n \log n)$ ([5]). And clearly polynomial $f(x) = f_A(x) \cdot f_B(x)$ has non-zero i-th coefficient iff $i - 2n \in A \ominus B$. □

Lemma 4.8. *For a spine tree $S = (T_l, T_r, r)$ with n nodes it is possible to calculate in time $\mathcal{O}(n^{1.5} \log n)$, the set of even palindromes P in S such that:*

- $|P| = \mathcal{O}(n^{1.5})$,
- *P contains all even palindromes in S with left endpoint and middle point in T_l and right endpoint in T_r.*

Proof. First, let us remind that the complexity of the algorithm is very close to the actual limit on the number of distinct palindromes in spine tree, since there could be up to $\mathcal{O}(n^{1.5)}$ palindromes in S (see Lemma 3.2). Our approach is very similar to the one used in the proof of Lemma 3.2. We identify the labels

$L(u)$ for each $u \in S$ with a distance at least $2\sqrt{n}$ from the root. Since the tree is already preprocessed with Lemma 4.4 and Lemma 4.5 such labels can be retrieved and represented in constant time and space.

Next all labels $L(u)$ are sorted in $\mathcal{O}(n \log n)$ time and we group all nodes with the same label into groups $V_{L_1}, V_{L_2}, \ldots, V_{L_k}$.

For each group V_L with at most $\sqrt{n}$ nodes can be inspected in $\mathcal{O}(|V_L|^2)$ time, for each $x \in V_L \cap T_l$ and $y \in V_L \cap T_r$ we check in $\mathcal{O}(1)$ the condition $\mathsf{isPalindrome}(x, y)$ and report the palindrome if the condition is true.

For each group V_L with more than $\sqrt{n}$ nodes we will use discrete convolutions to speed up the calculations. First we need to verify if the spine part of the palindromes from V_L is in fact palindromic. This can be checked by locating any pair of nodes $x \in V_L \cap T_l, y \in V_L \cap T_r$ such that $\mathsf{dist}(x, y)$ is even. If the condition $\mathsf{isPalindrome}(x, y)$ is true, then for any pair of nodes $x \in V_L \cap T_l, y \in V_L$ with even distance we obtain palindrome, all we need to do is to identify all possible (even) values of $\mathsf{dist}(s(x), s(y))$.

Let s_ℓ be the left endpoint of the spine of S and

$$X_L = \{\mathsf{dist}(s_\ell, s(x)) \text{ for } x \in V_L \cap T_l\}$$

$$Y_L = \{\mathsf{dist}(s_\ell, r) + \mathsf{dist}(s(y), r) \text{ for } y \in V_L \cap T_r\}.$$

The set of all possible differences Δ_L can be obtained by computing $Y_L \ominus X_L$ and taking only even values. This step takes $\mathcal{O}(n \log n)$ due to Lemma 4.7.

Unfortunately using this step we don't have a witnesses for values $\delta \in \Delta$. Nevertheless we are able to reconstruct the palindromic substrings itself. Let x_0 is the node from $V_L \cap T_l$ that is the farthest from the root r. For each even value $\delta \in \Delta$ with $\frac{\delta}{2} \leq \mathsf{dist}(s(x_0), r)$ we report palindrome with value ww^R where $w = val(x_0, up(s(x), \frac{\delta}{2}))$. Please not that this palindrome might not occur in the node x_0, also we might over-report here and report also the palindromes that have a middle point in T_r. □

4.3 Algorithm for general D-trees

In our algorithm we will use similar approach to the one used in the proof of Lemma 3.6. There are a few technical issues that we need to overcome. First we need to strengthen a notion of D-trees, we need to make sure that all paths in D-tree correspond to simple paths in the original tree. Unfortunately due to repeated edges adjacent to the root this rule can be violated (see Figure 5). Second problem is efficient calculation of palindromes in spine trees.
We resolve those issues with 2D-trees using following lemma:

Lemma 4.9. *Given undirected tree T with n nodes, we can calculate decomposition of T into family of D-trees $\mathcal{D}$ such that:*

- *each simple path in $D \in \mathcal{D}$ corresponds to a simple path in T,*
- *for each even path $p \in T$, there exists $D \in \mathcal{D}$ such that there exists even path $p' \in D$ such that $val(p) = val(p')$ and middle point of p' is in the left subtree of D,*

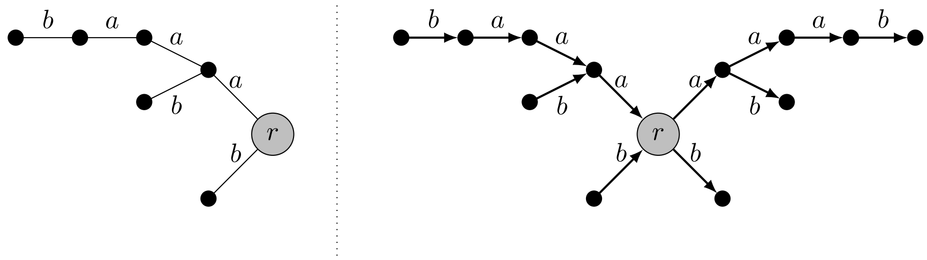

Figure 5: Undirected tree T and its D-tree $D = (T_l, r, T_r)$. Note that D contains path with even palindrome *baaaab* that is not present in the original tree T.

- *total number of edges in all double trees from $\mathcal{D}$ is $\mathcal{O}(n \log n)$.*

The decomposition $\mathcal{D}$ can be calculated in time $\mathcal{O}(n \log n)$.

Proof. The decomposition $\mathcal{D}$ can be created in recursive manner.
For a tree T, we use following procedure:

- identify centroid node r,
- divide subtrees adjacent to r into two trees T_1, T_2, such that $\max(|T_1|, |T_2|) \leq \frac{3}{4}|T|$,
- create deterministic version of trees T_1 and T_2 T_1' and T_2',
- to handle paths that have one endpoint in T_1 and other in T_2 we add to $\mathcal{D}$ D-trees (T_1', T_2', r) and (T_2', T_1', r),
- to handle paths that are contained in T_1 or T_2 recursively process decomposition of T_1 and T_2.

The total size of the created decomposition is $\mathcal{O}(n \log n)$. □

Now we are ready to outline the algorithm. For given tree T we decompose it into family or D-trees $\mathcal{D}$.
The family of trees $\mathcal{D}$ is preprocessed using Lemma 4.4 and Lemma 4.5. We need to process all trees from $\mathcal{D}$ altogether to obtain consistent DFS identifiers between different trees.
Each double tree $D \in \mathcal{D}$ is processed separately, we find the middle palindromes using spine decomposition and all other palindromes using exhaustive search.

Finally we remove possible duplicates in reported palindromes using sorting.

Algorithm 1: FindPalindromesInDoubleTree($D = (T_l, T_r, r)$)

---

preprocess the tree D for queries from lemmas 4.4 and 4.5
$P = \emptyset$
// handle middle palindromes
decompose D into spine trees $S_1, \ldots, S_k$
foreach $S_i \in$ SpineDecomposition(D) **do**
  add to P palindromes reported by Lemma 4.8 in S_i
end
// handle first (shorter) and last (longer) palindromes
foreach $u \in T_l$ **do**
  let L_u is a data structure that allows access to the list of ancestors u' of u such that $val(u', r)$ is a palindrome
  elements of L_u are sorted by their depth in a tree
  foreach $u' \in$ (*first* $4\sqrt{n}$ *nodes from* L_u) $\cup$ (*last* $2\sqrt{n}$ *nodes from* L_u) **do**
    if there exists node $v \in T_r$ such that $val(v, r) = val(u, u')$ and $val(u, v)$ is a palindrome add it to P
  end
end
return P

Lemma 4.10. *For a D-tree D with n nodes the problem* ReportAllEven *can be solved in in* $\mathcal{O}(n^{1.5} \log n)$ *time.*

Proof. The pseudocode of the solution is given in Algorithm 1. The correctness of the algorithm is proved by the Lemma 3.6. We need to prove that the algorithm can be implemented in $\mathcal{O}(n^{1.5} \log n)$ time. Calculating the spine decomposition requires $\mathcal{O}(n)$ time due to Lemma 4.6, and each spine S can be processed in $\mathcal{O}(|S|^{1.5} \log |S|)$ time due to Lemma 4.8. Since total size of the spine trees is $\mathcal{O}(n)$, this part takes $\mathcal{O}(n^{1.5} \log n)$.

For handling the first and last palindromes, we need a data structure to operate on lists L_u. We identify all nodes $X = \{x \in D : val(x, r) \text{ is a palindrome}\}$ using Lemma 4.5. Then we create a subtree D_X which contains only nodes from X and preprocess it for Level Ancestor queries ([2]). Additionally for each $u \in D$ we store its nearest ancestor in D_X and its depth in D_x (equal to L_u). With this approach any element of L_u can be retrieved in $\mathcal{O}(1)$ time using LA queries on D_x.
For each element $u' \in L_u$ we test the existence of node w in $\mathcal{O}(1)$ time using function exists. □

Theorem 4.11.
For a a tree T with n nodes the problem ReportAll *can be solved in* $\mathcal{O}(n^{1.5} \log n)$ *time.*

Proof. Due to Lemma 4.3 the problem reduces to counting even palindromes. Hence we later consider only even palindromes.

First we decompose the tree T into set of D-trees $\mathcal{D} = D_1, \ldots, D_k$ using Lemma 4.9. All trees are preprocessed using tools from Lemma 4.4 and Lemma 4.5.

Next we calculate palindromes in all D-trees $D_i \in \mathcal{D}$ using Lemma 1. Due to construction of $\mathcal{D}$ this requires time

$$T(n) = T(\alpha n) + T((1-\alpha)n) + \mathcal{O}(n^{1.5} \log n)$$

(for $\frac{1}{4} \leq \alpha \leq \frac{3}{4}$), which is $T(n) = \mathcal{O}(n^{1.5} \log n)$ and the total size of returned palindromes $\mathcal{P}$ is

$$P(n) = P(\alpha n) + P((1-\alpha)n) + \mathcal{O}(n^{1.5})$$

(for $\frac{1}{4} \leq \alpha \leq \frac{3}{4}$) which is $P(n) = \mathcal{O}(n^{1.5})$.

Finally we remove from $\mathcal{P}$ duplicates. Since all palindromes are identified by length, and pair of integers generated by label, we can sort $\mathcal{P}$ in $\mathcal{O}(|P|)$ time which is $\mathcal{O}(n^{1.5})$. □

5 Algorithm for finding longest palindrome in tree

In this section, we consider the following problems for palindromes in trees:

Problem 5.1 (PALINDROMETEST). Given tree a T with n edges, each labelled by single character from the alphabet Σ and integer $k > 0$, decide whatever T contains palindrome of length exactly k.

Problem 5.2 (FINDLONGEST). Given tree a T with n edges, each labelled by single character from the alphabet Σ find the length of the longest palindrome in T.

Theorem 5.3.
(a) *Problem* PALINDROMETEST *can be solved in $O(n \log^2 n)$ time.*
(b) *Problem* FINDLONGEST *can be solved in $O(n \log^3 n)$ time.*

Proof. The point **(a)** can be solved using the algorithm below. The point **(b)** can be solved using the testing function together with binary search. □

Algorithm 2: Test if there exists palindrome of length k in DD-tree D

```
foreach u ∈ nodes(T) do
    if depth(u) ≥ k then
        let v is a node up(u, k) ;
        if val(u, v) is a palindrome then
            return true
        end
    else
        let v′ is a node up(u, depth(u) − k) ;
        if isPalindrome(v′, r) and exists(D, u, v′) then
            return true
        end
    end
end
return false
```

6 Open problems

We conclude with following open question:

- is there an output sensitive version of the all palindromes reporting – can we report palindromes more efficiently for cases where we know that tree contains $o(n^{1.5})$ palindromes.

Acknowledgments

Paweł Gawrychowski's work was done while he held a post-doctoral position at Warsaw Center of Mathematics and Computer Science. Tomasz Kociumaka was supported by Polish budget funds for science in 2013–2017 as a research project under the 'Diamond Grant' program. Wojciech Rytter was supported by the Polish National Science Center, grant no NCN2014/13/B/ST6/00770. Tomasz Waleń was supported by the Polish Ministry of Science and Higher Education under the 'Iuventus Plus' program in 2015–2016 grant no 0392/IP3/2015/73.

References

[1] Mira-Cristiana Anisiu, Valeriu Anisiu, and Zoltán Kása. Total palindrome complexity of finite words. *Discrete Mathematics*, 310(1):109–114, 2010. doi:10.1016/j.disc.2009.08.002.

[2] Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004. URL: https://doi.org/10.1016/j.tcs.2003.05.002, doi:10.1016/j.tcs.2003.05.002.

[3] Srečko Brlek, Sylvie Hamel, Maurice Nivat, and Christophe Reutenauer. On the palindromic complexity of infinite words. *International Journal of Foundations of Computer Science*, 15(2):293–306, 2004. doi:10.1142/s012905410400242x.

[4] Srečko Brlek, Nadia Lafrenière, and Xavier Provençal. Palindromic complexity of trees. In Igor Potapov, editor, *Developments in Language Theory, DLT 2015*, volume 9168 of *LNCS*, pages 155–166. Springer, 2015. doi:10.1007/978-3-319-21500-6_12.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

[6] Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, Wojciech Tyczyński, and Tomasz Waleń. The maximum number of squares in a tree. In Juha Kärkkäinen and Jens Stoye, editors, *Combinatorial Pattern Matching*, volume 7354 of *LNCS*, pages 27–40. Springer, 2012. doi:10.1007/978-3-642-31265-6_3.

[7] Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology*. World Scientific, 2003.

[8] Xavier Droubay, Jacques Justin, and Giuseppe Pirillo. Episturmian words and some constructions of de Luca and Rauzy. *Theoretical Computer Science*, 255(1-2):539–553, 2001. doi:10.1016/s0304-3975(99)00320-5.

[9] Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. doi:10.2307/2034009.

[10] Mitsuru Funakoshi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing maximal palindromes and distinct palindromes in a trie. In Jan Holub and Jan Zdárek, editors, *Prague Stringology Conference 2019, Prague, Czech Republic, August 26-28, 2019*, pages 3–15. Czech Technical University in Prague, Faculty of Information Technology, Department of Theoretical Computer Science, 2019. URL: http://www.stringology.org/event/2019/p02.html.

[11] Pawel Gawrychowski, Tomasz Kociumaka, Wojciech Rytter, and Tomasz Walen. Tight bound for the number of distinct palindromes in a tree. In Costas S. Iliopoulos, Simon J. Puglisi, and Emine Yilmaz, editors, *String Processing and Information Retrieval - 22nd International Symposium, SPIRE 2015, London, UK, September 1-4, 2015, Proceedings*, volume 9309 of *Lecture Notes in Computer Science*, pages 270–276. Springer, 2015. URL: https://doi.org/10.1007/978-3-319-23826-5_26, doi:10.1007/978-3-319-23826-5_26.

[12] Amy Glen, Jacques Justin, Steve Widmer, and Luca Q. Zamboni. Palindromic richness. *European Journal of Combinatorics*, 30(2):510–531, 2009. doi:10.1016/j.ejc.2008.04.006.

[13] Amy Glen, Jamie Simpson, and William F. Smyth. Palindromes in starlike trees. *Australasian Journal of Combinatorics*, 73(1):242–246, 2019. URL: https://ajc.maths.uq.edu.au/pdf/73/ajc_v73_p242.pdf.

[14] Richard Groult, Élise Prieur, and Gwénaël Richomme. Counting distinct palindromes in a word in linear time. *Information Processing Letters*, 110(20):908–912, 2010. doi:10.1016/j.ipl.2010.07.018.

[15] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. URL: https://doi.org/10.1137/0213024, doi:10.1137/0213024.

[16] Tomasz Kociumaka, Jakub Pachocki, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Efficient counting of square substrings in a tree. *Theor. Comput. Sci.*, 544:60–73, 2014. URL: https://doi.org/10.1016/j.tcs.2014.04.015.

[17] Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. String powers in trees. *Algorithmica*, 79(3):814–834, 2017. doi:10.1007/s00453-016-0271-3.

[18] Jamie Simpson. Palindromes in circular words. *Theoretical Computer Science*, 550:66–78, 2014. doi:10.1016/j.tcs.2014.07.012.