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
Max Planck Institute for Informatics
Saarland Informatics Campus, Saarbrücken
Germany
tomasz.kociumaka@mpi-inf.mpg.de

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

Submitted: Oct 31, 2021; Accepted: Mar 21, 2023; Published: Apr 21, 2023

Abstract

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

We also propose an $\mathcal{O}(n^{1.5} \log^{0.5} n)$-time algorithm reporting all distinct palindromes and an $\mathcal{O}(n \log^2 n)$-time algorithm finding the longest palindrome in a tree.

Mathematics Subject Classifications: 05C05, 68W32

1 Introduction

Regularities in words are extensively studied in combinatorics and text algorithms. Among the basic types of such structures are palindromes: symmetric words that are the same

[*]A preliminary version of this paper, without algorithmic results, was presented at SPIRE 2015 [11].

when read in both directions. The *palindromic complexity* of a word is the number of distinct palindromic substrings 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 maximum 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 labeled by single letters. We define the substrings of such a tree as the labels of simple paths between arbitrary two nodes. Each path label is the concatenation of the labels of all edges on the path. Denote by $\mathsf{pals}(T)$ the set of all palindromic substrings of a tree T and by $\mathrm{PAL}(n)$ the maximum size of $\mathsf{pals}(T)$ over all trees T 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.

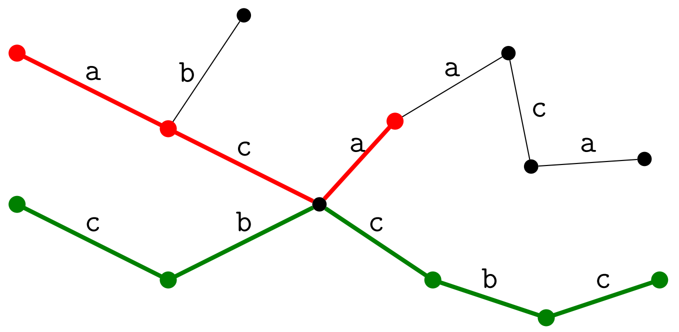

Figure 1: A sample tree T with $\mathsf{pals}(T) = \{\ \varepsilon$ (empty word), `a`, `b`, `c`, `aa`, `aca`, `acaaca`, `bcb`, `bccb`, `caac`, `cbc`, `cbcbc`, `cc` $\}$. An occurrence of a palindrome `aca` is marked red, and an occurrence of a palindrome `cbcbc` is marked green.

The study of the palindromic complexity of trees was 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 Results. We show that $\mathrm{PAL}(n) = \mathcal{O}(n^{1.5})$. This bound is tight by the construction given in [4]; hence, we completely settle the asymptotic maximum palindromic complexity for trees. We also provide an $\mathcal{O}(n^{1.5}\log^{0.5} n)$-time algorithm reporting all distinct palindromes and an $\mathcal{O}(n\log^2 n)$-time algorithm finding the longest palindrome 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 [20]. 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, Simpson, and Smyth [13]. The palindromes in directed trees have been also studied. Mieno, Funakoshi, and Inenaga [19] presented $\mathcal{O}(n)$-time algorithms computing all maximal palindromes and all distinct palindromes in a trie, improving upon earlier work of Funakoshi, Nakashima, Inenaga, Bannai, and Takeda [10].

In the setting of labeled 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 [18]. 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 a combinatorial toolbox. Next, in Section 3.1, we briefly describe the outline of the proof of the upper bound on the number of distinct palindromes. In Section 3.2, we introduce the special family of trees called *spine trees* and prove the upper bounds for those trees. Finally, in Section 3.3, we show how every tree can be decomposed into spine trees, and we combine those results to obtain the upper bound on the number of distinct palindromes. In Section 4, we introduce an algorithmic toolbox and provide an $\mathcal{O}(n^{1.5}\log^{0.5} n)$-time algorithm reporting all distinct palindromes. Next, in Section 5, we show an $\mathcal{O}(n\log^2 n)$-time algorithm finding the longest palindrome in a tree.

2 Preliminaries

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

2.1 Some Combinatorics of Words

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

Lemma 1 (Periodicity Lemma [9])**.** *If p, q are periods of a word w of length $|w| \geqslant p + q - \gcd(p,q)$, then $\gcd(p,q)$ is also a period of w.*

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

Lemma 2. *If v is a substring of a word u with period $p \leqslant \frac{1}{2}|v|$, then $\operatorname{per}(u) = \operatorname{per}(v)$.*

Proof. Suppose that $p_u = \operatorname{per}(u)$ and $p_v = \operatorname{per}(v)$ yet $p_u \neq p_v$. Since v is a substring of u and p is a period of u, we clearly have

$$p_v \leqslant p_u \leqslant p \leqslant \tfrac{1}{2}|v|.$$

The word v and its periods p_v and p_u meet the conditions of the Periodicity Lemma, so $\gcd(p_v, p_u)$ is also a period of v. By definition of p_v as the shortest period of v, this implies $p_v = \gcd(p_v, p_u)$, i.e., that $p_u = a \cdot p_v$ for some integer $a > 1$. But then p_v is also a period of the whole word u — a contradiction. □

We have the following connection between periods and palindromes.

Observation 3. *If 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 both u and v.*

2.2 Centroid Decomposition

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

Lemma 4. *Let $\epsilon > 0$ be a constant. If $|\mathsf{pals}(T, r)| = \mathcal{O}(|T|^{1+\epsilon})$ holds for every tree T and its node r, then $\mathrm{PAL}(n) = \mathcal{O}(n^{1+\epsilon})$ holds for every positive integer n.*

Proof. We follow the approach from [6] using the folklore fact that every tree T on n edges contains a *centroid* node r such that every component of $T \setminus \{r\}$ has at most $\frac{n}{2}$ edges. 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 $\mathrm{PAL}(n)$, the maximum number of palindromes in a tree with n edges:

$$\mathrm{PAL}(n) \leqslant \mathcal{O}(n^{1+\epsilon}) + \max\left\{\sum_i \mathrm{PAL}(n_i) : \max_i n_i \leqslant \tfrac{n}{2} \text{ and } \sum_i n_i \leqslant n\right\}.$$

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

2.3 D-Trees

For a tree T and its node r, we consider directed acyclic graphs, named *D-trees*, such that $\mathsf{pals}(T, r)$ is a subset of palindromic strings corresponding to simple directed paths in such graphs containing the node r.

Define a *double tree* $\mathcal{D} = (T_\ell, T_r, r)$ as an edge-labeled 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 (number of edges) of $\mathcal{D}$ is defined as $|\mathcal{D}| = |T_\ell| + |T_r|$.

For any nodes $u, v \in \mathcal{D}$, we denote by $\mathrm{path}(u, v)$ the path from u to v and by $\mathrm{val}(u, v)$ the sequence of the labels of edges on this path. We further denote

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

A *substring* of $\mathcal{D}$ is any word $\mathrm{val}(u, v)$ such that $u \in T_\ell$ and $v \in T_r$. We say that a path $\mathrm{path}(u, v)$ is palindromic if $\mathrm{val}(u, v)$ is a palindrome. Denote by $\mathsf{pals}(\mathcal{D})$ the set of palindromic substrings of a double tree $\mathcal{D}$.

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 D-tree is shown in Fig. 2.

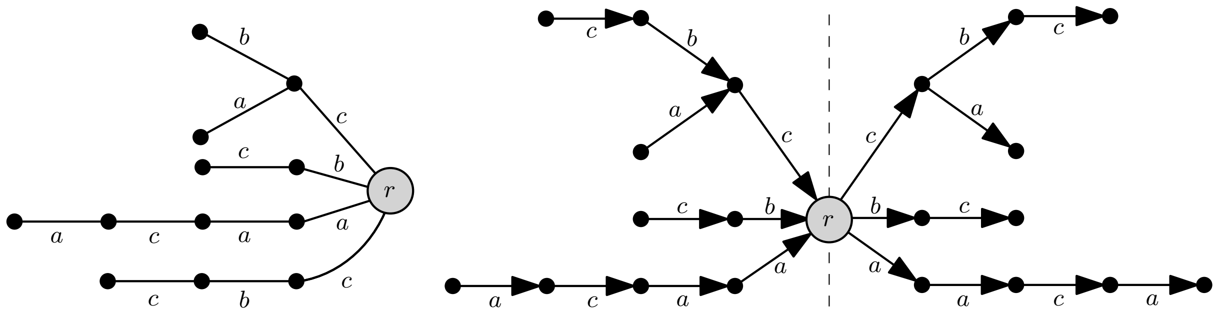

Figure 2: To the left: an example undirected tree with $\mathsf{pals}(T, r)$ containing 9 palindromic substrings of length two or more `bcb`, `bccb`, `aca`, `cbc`, `caac`, `cc`, `cbcbc`, `aa`, `acaaca`. To the right: a D-tree $\Psi(T, r)$ obtained after rooting the tree at r, merging both subtrees connected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

Trees $\Rightarrow$ 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 repeatedly gluing together two children of the same node whenever their outgoing 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 $\mathrm{val}(u, v) = \mathrm{val}(u', v')$. This implies the following fact.

Fact 5. $\mathsf{pals}(T, r) \subseteq \mathsf{pals}(\Psi(T, r))$.

Note that $\mathsf{pals}(T, r)$ might be a proper subset of $\mathsf{pals}(\Psi(T, r))$. For example, in the tree T depicted in Fig. 2, a palindrome `bb` appears in $\mathsf{pals}(\Psi(T, r))$ but not in $\mathsf{pals}(T, r)$.

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

Due to Lemma 4, it is enough to consider palindromic paths passing through a fixed node r of the tree. Fact 5 reduces our task to bounding the number of distinct palindromes in a D-tree, which is easier.

Consider a D-tree (T_l, T_r, r) and a directed path $\mathrm{path}(u, v)$ from $u \in T_l$ to $v \in T_r$. We define the *central part* of $\mathrm{path}(u, v)$ as the subpath $\mathrm{path}(u', v')$ such that $\mathrm{dist}(u, u') = \mathrm{dist}(v', v)$ and $r \in \{u', v'\}$. By symmetry of the counting problem (up to edge reversal in a double tree), we henceforth consider only palindromic paths $\mathrm{path}(u, v)$ such that $\mathrm{dist}(u, r) \geqslant \mathrm{dist}(r, v)$. The central part of such a path is of the form $\mathrm{path}(u', r)$ for some $u' \in T_l$ (intuitively, this means that $\mathrm{path}(u, v)$ is centered within T_l).

Our proof relies on a particular family of D-trees that we call *spine trees*. A spine tree is a D-tree with a distinguished path, called the *spine*, joining nodes $s_\ell \in T_\ell$ and $s_r \in T_r$. Additionally, we insist that this path cannot be extended preserving the period

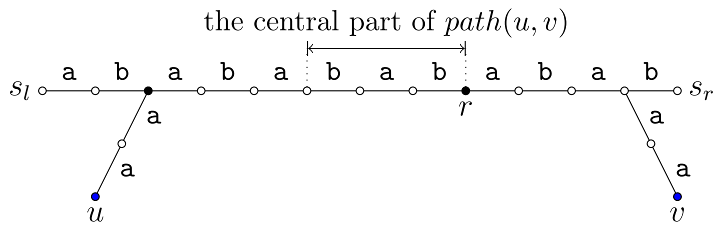

Figure 3: A spine tree whose spine, $\text{path}(s_l, s_r)$, has period 2 and is depicted as a horizontal path. The palindrome $\text{val}(u, v) = x_1 \cdot x_2 \cdot x_1^R =$ `aaaba bab abaaa` is induced by this spine tree, and the value of its central part is $x_2 =$ `bab`.

$p = \text{per}(s_\ell, s_r)$. We say that a palindromic substring $\text{val}(u, v)$ is *induced* by a spine tree if the central part of $\text{path}(u, v)$ is a fragment of the spine of length at least p, where p is the period of the spine; see Figures 3 and 4.

Remark 6. A spine tree with n edges may induce $\Omega(n^{1.5})$ distinct palindromes; see the construction of the tree used in lower bound proof presented by Brlek et al. [4].

3.1 Combinatorial Outline

The structure of the proof is described informally as follows.

- We show that the number of palindromes induced by a spine tree is $\mathcal{O}(n^{1.5})$.
- We identify so-called *middle palindromes* among the palindromes in a D-tree. The number of the remaining palindromes is easily bounded by $\mathcal{O}(n^{1.5})$.
- We cover the D-tree with smaller spine trees of linear total size.
- We show that each middle palindrome is induced by at least one of the spine trees.
- Now, the upper bound on all palindromes in the D-tree follows from the upper bounds on palindromes induced by spine trees.

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 $u = s(u)$). Define the *label* $L(u)$ of a node $u \in T_\ell$ as the prefix of $\text{val}(u, s_r)$ of length $\text{dist}(u, s(u)) + p$, where $p = \text{per}(s_l, s_r)$. Similarly, the label $L(v)$ of a node $v \in T_r$ is the reversed suffix of $\text{val}(s_\ell, v)$ of length $p + \text{dist}(s(v), v)$ (see Figure 4). We leave the label undefined if $\text{val}(u, s_r)$ or $\text{val}(s_\ell, v)$ is not sufficiently long, i.e., if $\text{dist}(s(u), s_r) < p$ or $\text{dist}(s_\ell, s(v)) < p$, respectively.

Since the spine tree is deterministic, it satisfies the following property.

Fact 7. *If $u \in T_\ell$, $v \in T_r$, and $\text{val}(u, v)$ is an induced palindrome, then $L(u) = L(v)$ and $\text{val}(s(u), s(v))$ is an inclusion-wise maximal fragment of $\text{val}(u, v)$ admitting period p.*

Denote by V_L the set of nodes of $T_\ell \cup T_r$ with label L and by P_L the set of induced palindromes corresponding to paths with endpoints in V_L. The following bound holds for every label L.

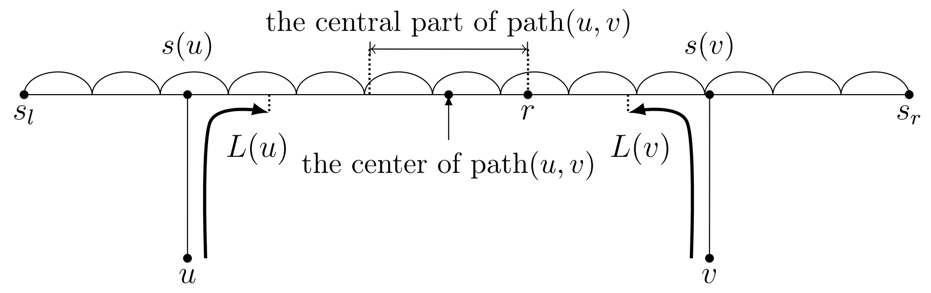

Figure 4: A spine tree, whose spine is the path from s_ℓ to s_r, with an induced palindrome $\mathrm{val}(u,v)$. Observe that $L(u) = L(v)$ is a prefix of the palindrome. Note that $\mathrm{dist}(s(u),r) \geqslant p$ (because the central part is of length at least p) but $\mathrm{dist}(r, s(v))$ might be smaller than p.

Proposition 8. *$|P_L| \leqslant n$.*

Proof. We use the following simple property.

Claim 9. *Assume z is a factor of a given word with period p. Then, z is uniquely determined by its length $|z|$ and its length-p prefix $z[1..p]$.*

Every palindrome $\mathrm{val}(u,v)$ in P_L starts with L and ends with L^R. Consider the internal part $\mathrm{val}(s(u),s(v))$ of this palindrome obtained by removing the prefix and the suffix of length $|L|-p$; this part has period p. Due to the claim above, such an internal part is uniquely determined by its length and length-p prefix, both of which are determined by the palindrome length and the label L. Thus, P_L contains at most one palindrome of any given length, which may range from $|L|$ to n. Consequently, $|P_L| \leqslant n - |L| + 1 \leqslant n$. □

Lemma 10. *There are at most $(n+1)\sqrt{n}$ distinct palindromic substrings induced by a given spine tree of size n.*

Proof. Due to Fact 7, the set of palindromes induced by a given spine tree equals $\bigcup_L P_L$. Obviously, $|P_L| \leqslant |V_L|^2$ since there are at most $|V_L|^2$ paths with endpoints in V_L. Hence, by Proposition 8 and the disjointness of sets V_L, we know that

$$|P_L| \leqslant \min(n, |V_L|^2) \quad \text{and} \quad \sum_L |V_L| \leqslant n+1.$$

Now, using the inequality $\min(x,y) \leqslant \sqrt{xy}$ (valid for all $x,y \geqslant 0$), the number of distinct palindromes induced by the spine tree can be bounded as follows:

$$\sum_L |P_L| \leqslant \sum_L \min(|V_L|^2, n) \leqslant \sum_L |V_L|\sqrt{n} \leqslant (n+1)\sqrt{n}.$$ □

3.3 Number of All Palindromes

We start with the definition of *middle palindromes.*

Definition 11. A palindrome $P = \mathrm{val}(u, v)$ is a *middle palindrome* if the central part C of this palindrome satisfies $\mathrm{per}(C) \leqslant \frac{1}{2}\sqrt{n}$ and $\alpha \leqslant |C| \leqslant |P| - 2\alpha$, where $\alpha = \lceil 2\sqrt{n}\rceil$, and extending C by α characters in each direction preserves the shortest period $\mathrm{per}(C)$.

Next, we show that there are $\mathcal{O}(\sqrt{n})$ non-middle palindromes $\mathrm{val}(u, v)$ for fixed node $u \in T_l$ (and hence $\mathcal{O}(n^{1.5})$ in total), so we can focus on counting middle palindromes.

Lemma 12. *Consider a node $u \in T_l$ and palindromic suffixes $C_1, \ldots, C_k$ of $\mathrm{val}(u, r)$ such that $|C_1| > \cdots > |C_k|$. For each $i \in [1..k]$, there is at most one palindrome $P = \mathrm{val}(u, v)$ with central part C_i. If $i \in [2\alpha..k - \alpha]$, then such a palindrome is a middle palindrome.*

Proof. Consider a palindrome $P = \mathrm{val}(u, v)$ with central part C_i. Observe that $|P_i| = 2\,\mathrm{dist}(u, r) - |C_i|$ and $\mathrm{dist}(u, r) \geqslant \frac{1}{2}|P_i|$, so $\mathrm{val}(u, r)$ and $|C_i|$ determine the palindrome P.

If $i \in [\alpha..k - \alpha]$, then $|C_i| \geqslant |C_{k-\alpha}| \geqslant \alpha$ because we excluded the α shortest suffixes $C_{k-\alpha+1}, \ldots, C_k$. Let us now prove that $\mathrm{per}(C_i) \leqslant \frac{1}{2}\sqrt{n}$ holds for $i \in [\alpha..k - \alpha]$. By Observation 3, $|C_j| - |C_{j+1}|$ is a period of C_j for $1 \leqslant j \leqslant i$. Since

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

for some $j \in [1..i]$ we have

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

Moreover, C_i is a suffix of C_j, so we indeed have $\mathrm{per}(C_i) \leqslant \mathrm{per}(C_j) \leqslant \frac{1}{2}\sqrt{n}$.

Since C_i is a substring of C_α of length $|C_i| \geqslant \alpha > \sqrt{n} \geqslant 2 \cdot \mathrm{per}(C_\alpha)$, Lemma 2 further implies $\mathrm{per}(C_i) = \mathrm{per}(C_\alpha)$. Moreover, for $i \in [2\alpha..k - \alpha]$, we have

$$|C_\alpha| \geqslant |C_i| + i - \alpha \geqslant |C_i| + \alpha,$$

so extending C_i by α characters to the left preserves the period $\mathrm{per}(C_i) = \mathrm{per}(C_\alpha)$. By symmetry of P, the extension to the right also preserves the period. In particular, $|P| \geqslant |C_i| + 2\alpha$. This concludes the proof that P is a middle palindrome if $i \in [2\alpha..k - \alpha]$. □

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

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

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 the spine corresponding to the path from s_ℓ to s_r as shown in Figure 5. We attach to the spine all subtrees hanging off the original path at distance at least α from the root. In other words, a node $u \in T_\ell$ which does not belong to the spine is added to the spine tree if $\mathrm{dist}(s(u), r) \geqslant \alpha$ and

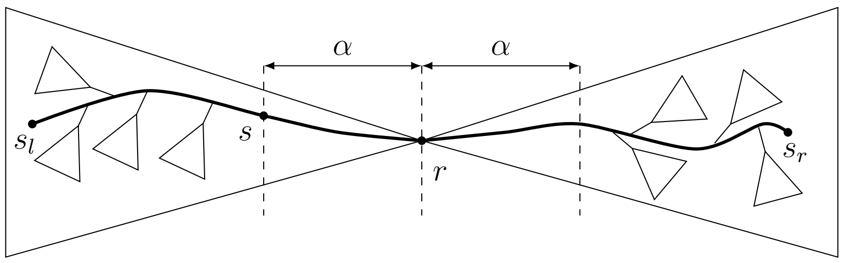

Figure 5: A spine tree constructed for a node $s \in T_l$ in a D-tree. Note that we do not attach subtrees at distance less than α from the root.

a node $v \in T_r$ — if $\operatorname{dist}(r, s(v)) \geqslant \alpha$. If $\operatorname{dist}(r, s_r) < \alpha$, 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 $P = \operatorname{val}(u, v)$. By definition, its central part C satisfies $|C| \geqslant \alpha$ and $\operatorname{per}(C) \leqslant \frac{1}{2}\sqrt{n}$. Moreover, Lemma 2 implies $\operatorname{per}(C) = \operatorname{per}(s, r)$ for the unique node $s \in T_\ell$ within C located 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 to the right preserving the period, the other endpoint v must also belong to such a subtree in T_r (that is, $\operatorname{dist}(r, s(v)) \geqslant \alpha$). Hence, every middle palindrome is induced by some spine tree.

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

Lemma 13. *The sizes $n_1, \ldots, n_k$ of the created spine trees satisfy $\sum_i n_i \leqslant 2n$.*

Proof. We claim that at least $n_i - \alpha$ edges 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 $\operatorname{dist}(c_i, r) = \lfloor\sqrt{n}\rfloor$ and similarly let s_i satisfy $\operatorname{dist}(s_i, r) = \alpha$. Recall that $\operatorname{per}(s_i, r) \leqslant \frac{1}{2}\sqrt{n}$. Thus, Lemma 2 yields $\operatorname{per}(s_i, r) = \operatorname{per}(c_i, r)$. 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 all the edges in the subtree of c_i.

A symmetric argument shows that all nodes d_i on the spine of the ith spine tree such that $\operatorname{dist}(r, d_i) = \lfloor\sqrt{n}\rfloor$ are also all distinct, and so are all the edges in the subtree of d_i. Therefore, we proved $\sum_i (n_i - 2\lfloor\sqrt{n}\rfloor) \leqslant n$.

Each spine tree has at least 2α edges on the spine, so this yields $n_i \geqslant 2\alpha \geqslant 4\lfloor\sqrt{n}\rfloor$ and $n_i \leqslant 2(n_i - 2\lfloor\sqrt{n}\rfloor)$. Hence, we obtain

$$\sum_i n_i \leqslant 2\sum_i (n_i - 2\lfloor\sqrt{n}\rfloor) \leqslant 2n. \qquad \square$$

Lemma 14. *A D-tree with n edges contains $\mathcal{O}(n^{1.5})$ distinct palindromic substrings.*

Proof. By Lemma 10, the number of palindromes induced by the ith spine tree is at most $(n_i + 1)\sqrt{n_i}$. Accounting for the $\mathcal{O}(n^{1.5})$ palindromes which do not occur as middle palindromes, the total number of palindromes becomes

$$\mathcal{O}(n^{1.5}) + \sum_i (n_i + 1)\sqrt{n_i} \leqslant \mathcal{O}(n^{1.5}) + \sum_i (n_i + 1)\sqrt{n} = \mathcal{O}(n^{1.5}). \qquad \square$$

Due to Lemma 4 and Fact 5, we obtain the following result:

Theorem 15. *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 the following problem for palindromes in trees:

Problem 16 (ReportAll)**.** Given a tree T with n edges, each labeled by a single character from the alphabet Σ, report all distinct palindromic substrings of T.

Among various ways of representing a palindromic substring P, perhaps the most natural choice is to use a pair for nodes (u, v) such that $\mathrm{val}(u, v) = P$. For efficiency reasons, we use a slightly different format: each palindrome P is reported as triple (ℓ, u, v) such that $\ell = |P|$ is the length of P and $\mathrm{val}(u, v) = P[1..\lceil \frac{\ell}{2} \rceil]$ is the first half of P.

4.1 Algorithmic Tools

We use the tools similar to the ones described in [17, Section 3] but tailored to the palindromic case.

We say that a node v is an ancestor of a node u in a given D-tree if there is a directed path from u to v. Additionally, we define *depth*(u) in D-tree as a distance between u and the root r (ignoring the edge orientation).

Lemma 17. *A family of D-trees $D_1, \ldots, D_k$ with n edges in total can be preprocessed in $\mathcal{O}(n)$ time so that following operations can be performed in $\mathcal{O}(1)$ time:*
- $\mathsf{dist}(u, v)$ *– the distance between nodes u and v;*
- $\mathsf{up}(u, h)$ *– the node v on a path from u towards the root at distance h from u;*
- $\mathsf{isAncestor}(u, v)$ *– decides if v is an ancestor of u;*
- $\mathsf{perLen}(u)$ *– the shortest period of the label of the path between u and the root.*

Proof. The dist queries can be implemented by precomputing the depth of each node and using the Lowest Common Ancestor (LCA) queries; see [16]. The up queries are, in fact, the Level Ancestor (LA) queries [2]. The period lengths $\mathsf{perLen}(u)$ can be calculated from the border array, which can be constructed in $\mathcal{O}(n)$ time [17]. □

Lemma 18. *A family of D-trees $D_1, \ldots, D_k$ with n edges in total can be preprocessed in $\mathcal{O}(n \log n)$ time so that following operations can be performed in $\mathcal{O}(1)$ time:*
- $\mathsf{label}(u, v)$ *– returns an integer identifier (from range $[1..n^2]$) that represents the word $\mathrm{val}(u, v)$ (this operation is defined if u is an ancestor of v or v is an ancestor of u);*
- $\mathsf{isEqual}(u_1, v_1, u_2, v_2)$ *– decides if $\mathrm{val}(u_1, v_1) = \mathrm{val}(u_2, v_2)$;*
- $\mathsf{isPalindrome}(u, v)$ *– decides if a word $\mathrm{val}(u, v)$ a palindrome;*
- $\mathsf{exists}(D_i, u, v)$ *– for $D_i = (L_i, R_i, r_i)$, decides if there exists a node $w \in R_i$ such that $\mathrm{val}(w, r_i) = \mathrm{val}(u, v)$ (this operation is defined only if $u, v \in L_i$ and u is an ancestor of v or v is an ancestor of u);*

- child(u, c) – *returns the child of u with label c (a null value if no such child exists).*

Proof. Operations label, isEqual, and isPalindrome can be implemented using a Dictionary of Basic Factors (DBF) [7, 17], which requires $\mathcal{O}(n \log n)$ preprocessing time. The only extension needed is that, for each basic factor, we also store the identifier of its reverse. For operation exists, we store the DBF codes of all possible values of $\mathrm{val}(w, r)$ in a static dictionary with constant lookup time. This can be achieved using deterministic dictionaries of [15]. The same approach as for exists can be used to implement child: for each tree edge (u, v) with label c, we store a dictionary entry with key (u, c) and value v. $\square$

Recall that we denoted $\alpha = \lceil 2\sqrt{n} \rceil$.

Lemma 19. *For a double tree $D = (T_l, T_r, r)$ with n edges, the spine decomposition can be calculated in $\mathcal{O}(n)$ time, assuming that D has been preprocessed with Lemma 17.*

Proof. We start with calculating the set S consisting of all nodes $u \in T_l$ with $\mathrm{dist}(u, r) = \alpha$ and $\mathrm{per}(u, r) \leqslant \frac{1}{2}\sqrt{n}$. Since D has been preprocessed with Lemma 17, the set S can be calculated in $\mathcal{O}(n)$ time. Observe that, for any node $u \in S$, due to periodicity of $\mathrm{val}(u, r)$, the path (u, r) has at least $\sqrt{n}$ distinct edges from the all other paths (u', r) for $u' \in S \backslash \{u\}$; hence, $|S| \leqslant \sqrt{n}$.

Next, for each candidate node $u \in S$, we extend $\mathrm{path}(u, r)$ to a spine $\mathrm{path}(s_l, s_r)$. For this, we locate the lowest descendant s_l of u such that $\mathrm{per}(s_l, r) = \mathrm{per}(u, r)$. Such a node can be located by traversing the subtree of u with child queries. Similarly, we traverse T_r starting from the root r to locate the lowest node in s_r such that $\mathrm{per}(u, s_r) = \mathrm{per}(u, r)$. If $\mathrm{dist}(r, s_r) \geqslant \alpha$, then we create a spine tree with the spine (s_l, s_r), attaching all subtrees to every spine node at distance at least α from the root.

In this procedure, only the first $\lfloor \sqrt{n} \rfloor$ edges on the path from r to s_r can be visited multiple times; since $|S| < \sqrt{n}$, the total processing time of such edges is still $\mathcal{O}(n)$. $\square$

4.2 Algorithm for Spine Trees

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

Lemma 20. [FFT Application] *Given two sets of integers $A, B \subseteq [0, \ldots, n]$ the set $A \oplus B = \{a + b : a \in A, b \in B\}$ can be computed in $\mathcal{O}(n \log n)$ time.*

Proof. We define two polynomials

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

Using FFT, we can multiply two polynomials with integer coefficients in $\mathcal{O}(n \log n)$ time [5]. Clearly, $j \in A \oplus B$ if and only if $f(x) = f_A(x) \cdot f_B(x)$ has a non-zero coefficient at x^j. $\square$

Note that the complexity of the following algorithm is very close to the tight bound $\mathcal{O}(n^{1.5})$ on the number of distinct palindromes induced by a spine tree (see Lemma 10).

Lemma 21. *Given a spine tree $S = (T_l, T_r, r)$ with n edges, it is possible to construct in time $\mathcal{O}(n^{1.5}\log^{0.5} n)$ the set of palindromes induced by this spine tree.*

Proof. Our approach is very similar to the one used in the proof of Lemma 10. Let us assume that the spine of S has period p. We identify the labels $L(u)$ for each node $u \in T_l$ with $\mathrm{dist}(s(v), r) \geqslant p$ and $L(v)$ for each $v \in T_r$. If the tree is preprocessed with Lemmas 17 and 18, such labels can be retrieved and represented in constant time and space.

Next, we sort the labels in $\mathcal{O}(n \log n)$ time and partition the nodes into classes of nodes sharing the same label.

Each class V_L with at most $\sqrt{n \log n}$ 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 $\mathcal{O}(1)$ time the condition $\mathsf{isPalindrome}(x, y)$ and report the palindrome if the test succeeds.

For each class V_L with more than $\sqrt{n \log n}$ nodes, we use Lemma 20 to speed up the calculations. First, we need to verify if the spine part of the palindromes from V_L is palindromic. This can be checked by locating any pair of nodes $x \in V_L \cap T_l$ and $y \in V_L \cap T_r$. The condition $\mathsf{isPalindrome}(x, y)$ is true if and only if any two nodes $x' \in V_L \cap T_l$ and $y' \in V_L \cap T_r$ yield a palindrome $\mathrm{val}(x', y')$. The lengths of these palindromes are given by the set

$$\Delta_L = \{\mathsf{dist}(x, y) \ : \ x \in V_L \cap T_l, y \in V_L \cap T_r\}.$$

Claim 22. *The set Δ_L can be computed in time $\mathcal{O}(n \log n)$.*

Proof. Let $X_L = \{\mathsf{dist}(x, r) \ : \ x \in V_L \cap T_l\}$ and $Y_L = \{\mathsf{dist}(r, y) \ : \ y \in V_L \cap T_r\}$. The set Δ_L can be expressed as $X_L \oplus Y_L$, so it can be computed in $\mathcal{O}(n \log n)$ time using Lemmas 17 and 20. □

Recall (from the proof of Proposition 8) that every palindrome in P_L has unique length, so the set Δ_L provides a complete description of P_L. However, this representation does not allow removing duplicates across multiple labels and spine trees. Moreover, Claim 22 does not provide witness occurrences of palindromes in P_L.

Nevertheless, we can represent these palindromes following way. Let x_0 be a node from V_L of maximum depth. For each length $d \in \Delta_L$, we report the palindrome of length d whose left half is $\mathrm{val}(x_0, \mathsf{up}(x_0, \lceil \frac{d}{2} \rceil))$; by construction of x_0 and Δ_L, we have $\mathit{depth}(x_0) \geqslant \lceil \frac{d}{2} \rceil$. Observe that the underlying palindrome might not occur starting at the node x_0, and we may report palindromes whose occurrences are centered in T_r. □

4.3 Algorithm for General D-trees

In our algorithm, we will use a similar approach to the one used in the proof of Lemma 14. However, we cannot use the construction $\Psi(T)$, which was useful in obtaining the combinatorial bound, since simple paths in $\Psi(T)$ can correspond to non-simple walks in T (see Figure 6).

First, we strengthen our notion of D-trees, because we need to make sure that all paths in D-tree correspond to simple paths in the original tree. The second problem is the efficient calculation of the palindromes in spine trees.

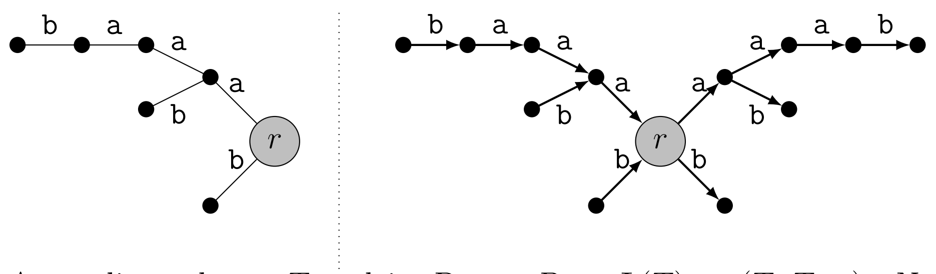

Figure 6: An undirected tree T and its D-tree $D = \Psi(T) = (T_l, T_r, r)$. Note that D contains a palindrome `baaaab`, which is not present in the original tree T.

For both short and long palindromes, it is possible to manually check if the reported palindrome occurs in the original tree; unfortunately, for the middle palindromes, this is not possible. This is caused by a lack of witnesses (endpoint nodes) for middle palindromes reported in Lemma 21.

We resolve this issue with the following lemma:

Lemma 23. *There exists an algorithm that, given an undirected tree T with n edges, in $\mathcal{O}(n \log n)$ time builds a decomposition of T into a family of D-trees $\mathcal{D}$ such that:*

- *each simple path in $D \in \mathcal{D}$ corresponds to a simple path in T with the same value;*
- *for each path in T, there exists a D-tree $D = (T_l, T_r, r) \in \mathcal{D}$ and a path $\mathrm{path}(u, v)$ in D with the same value such that $u \in T_l$, $v \in T_r$, and $\mathrm{dist}(u, r) \geqslant \mathrm{dist}(r, v)$;*
- *the total number of edges in all double trees $D \in \mathcal{D}$ is $\mathcal{O}(n \log n)$.*

Proof. The decomposition $\mathcal{D}$ for a tree T can be created recursively as follows:

1. identify the centroid node r of T;
2. partition the subtrees adjacent to r into two groups and form two edge-disjoint trees T_1, T_2 sharing the node r and satisfying $\max(|T_1|, |T_2|) \leqslant \frac{3}{4}|T|$;
3. create deterministic versions T'_1 and T'_2 of trees T_1 and T_2, respectively;
4. add D-trees (T'_1, T'_2, r) and (T'_2, T'_1, r) to $\mathcal{D}$;
5. recursively process T_1 and T_2.

Since (T_1, T_2, r) and (T_2, T_1, r) are orientations of T (with no edges doubled), every simple path in their deterministic versions (T'_1, T'_2, r) and (T'_2, T'_1, r) corresponds to a simple path in T. The total size of the created decomposition is $\mathcal{O}(n \log n)$. □

We are now ready to outline the algorithm. Given a tree T, we decompose it into a family of D-trees $\mathcal{D}$. The family of trees $\mathcal{D}$ is preprocessed using Lemmas 17 and 18. We process all trees from $\mathcal{D}$ simultaneously to obtain consistent DBF identifiers between different trees. Then, each double tree $D \in \mathcal{D}$ is processed separately, and 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.

Lemma 24. *For a D-tree D with n edges, the* ReportAll *problem can be solved in $\mathcal{O}(n^{1.5} \log^{0.5} n)$ time.*

Proof. The pseudocode of the solution is given in Algorithm 1. The correctness of the algorithm is proved by Lemmas 12 and 14. It remains to provide an $\mathcal{O}(n^{1.5}\log^{0.5} n)$-time implementation. Constructing the spine decomposition requires $\mathcal{O}(n)$ time due to Lemma 19, and each spine S can be processed in $\mathcal{O}(|S|^{1.5}\log^{0.5}|S|)$ time due to Lemma 21. Since the total size of the spine trees is $\mathcal{O}(n)$, this part takes $\mathcal{O}(n^{1.5}\log^{0.5} n)$ time in total.

For handling the non-middle palindromes, we need a data structure to operate on the lists L_u representing the palindromic suffixes of $\mathrm{val}(u, r)$. We identify all nodes $X = \{x \in D : \mathrm{val}(x, r) \text{ is a palindrome}\}$ using Lemma 18. 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 $|L_u|$). This way, any element of L_u can be retrieved in $\mathcal{O}(1)$ time using an LA query on D_X.

For each element $v' \in L_u$, we test the existence of an appropriate node v in $\mathcal{O}(1)$ time using the function exists. □

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

```
Output: a representation of palindromes in $D$ with start and center in $T_l$ and
        end in $T_r$
preprocess the tree $D$ for queries from Lemmas 17 and 18
$\mathcal{P} := \emptyset$
// handle middle palindromes
decompose $D$ into spine trees $S_1, \dots, S_k$
foreach spine tree $S_i$ do
    add to $\mathcal{P}$ palindromes induced by $S_i$        // reported using Lemma 21
// handle first (shorter) and last (longer) palindromes
foreach $u \in T_l$ do
    let $L_u$ be the list of ancestors $v'$ of $u$ such that $\mathrm{val}(v', r)$ is a palindrome, with
      elements of $L_u$ ordered by decreasing depths
    $First$ := the first $2\alpha - 1$ nodes from $L_u$
    $Last$ := the last $\alpha$ nodes from $L_u$
    foreach $v' \in First \cup Last$ do
        if exists($D, u, v'$) then      // there is $v \in T_r$ with $\mathrm{val}(v, r) = \mathrm{val}(u, v')$
            add to $\mathcal{P}$ the palindrome $\mathrm{val}(u, v)$
return $\mathcal{P}$
```

Theorem 25. *For a tree T with n edges, the* ReportAll *problem can be solved in $\mathcal{O}(n^{1.5}\log^{0.5} n)$ time.*

Proof. First, we decompose the tree T into a set $\mathcal{D} = \{D_1, \dots, D_k\}$ of D-trees using Lemma 23. All trees are preprocessed using tools from Lemmas 17 and 18. This step takes $\mathcal{O}(n \log^2 n)$ time.

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

$$T(n) = T(\beta n) + T((1-\beta)n) + \mathcal{O}(n^{1.5}\log^{0.5} n)$$

(for $\frac{1}{4} \leqslant \beta \leqslant \frac{3}{4}$), which is $T(n) = \mathcal{O}(n^{1.5} \log^{0.5} n)$. The total number of the returned palindromes (including duplicates) is

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

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

Finally, we remove duplicates from $\mathcal{P}$. Since all palindromes are represented by the length and the integer identifier of the first half (generated by label), we can sort $\mathcal{P}$ in $\mathcal{O}(|P|) = \mathcal{O}(n^{1.5})$ time. □

5 Algorithm Finding the Longest Palindrome in Tree

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

Problem 26 (PalindromeTest)**.** Given an integer $k > 0$ and a tree T with n edges, each labeled by a single character from the alphabet Σ, decide whether T contains a palindrome of length exactly k.

Problem 27 (FindLongest)**.** Given a tree T with n edges, each labeled by a single character from the alphabet Σ, find the length of the longest palindrome in T.

Theorem 28. *Given a tree T with n edges, each labeled by a single character from the alphabet Σ, the* PalindromeTest *and* FindLongest *problems can be solved in $\mathcal{O}(n \log^2 n)$ time.*

Proof. First, we use Lemma 23 to decompose the tree T into a set of D-trees with $\mathcal{O}(n \log n)$ edges in total and Lemmas 17 and 18 to preprocess these D-trees; this takes $\mathcal{O}(n \log^2 n)$ time in total.

For each D-tree, the PalindromeTest problem can be solved in linear time using Algorithm 2 provided below. We consider each node $u \in T_l$ and the palindromes starting at u with the center in T_l. If $depth(u) > k$, then there is no path(u, v) of length k with $v \in T_r$. Otherwise, we find a node $v' = \mathsf{up}(u, k - depth(u))$. We know that a potential palindrome starting at u of size k ends in some node $v \in T_r$ with distance $k - depth(u)$ from r. We check, using the function $\mathsf{exists}(D, u, v')$, if there is such a node v satisfying $\mathrm{val}(v, r) = \mathrm{val}(u, v')$.

Algorithm 2: Test if there exists palindrome of length k in D-tree $D = (T_l, T_r, r)$

```
foreach $u \in T_l$ do
    if $depth(u) \leqslant k$ then                                    // see Fig. 7
        $v' := \mathsf{up}(u, k - depth(u))$;
        if $\mathsf{isPalindrome}(v', r)$ and $\mathsf{exists}(D, u, v')$ then
            return $true$;  // there is a node $v \in T_r$ with $\mathrm{val}(v, r) = \mathrm{val}(u, v')$
return $false$;
```

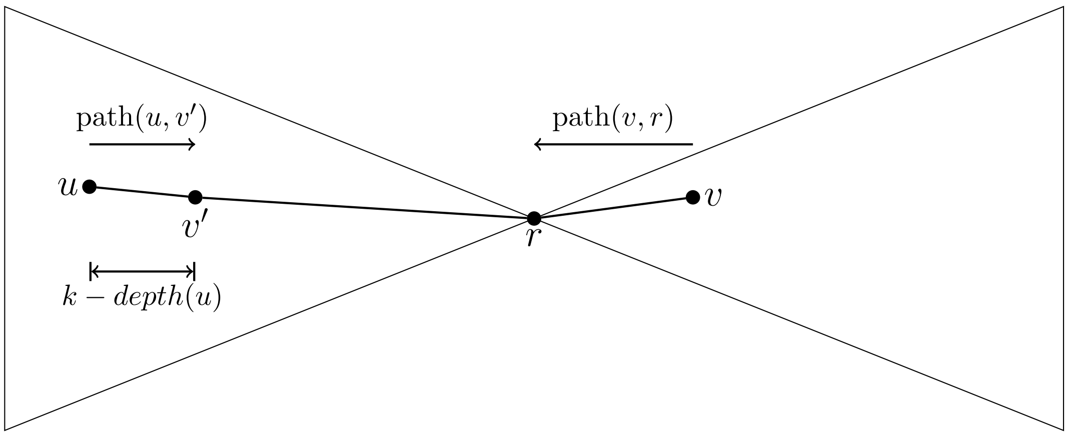

Figure 7: Illustration of Algorithm 2 for a D-tree with root r.

The FindLongest problem can be solved using binary search on top of our solution to the PalindromeTest problem. Both the preprocessing and the application of PalindromeTest (for $\mathcal{O}(\log n)$ lengths and D-trees of $\mathcal{O}(n \log n)$ edges in total) take $\mathcal{O}(n \log^2 n)$ time. $\square$

6 Open problem

We conclude with the following open problem:

- Given labelled tree T, report $\mathsf{pals}(T)$ in $\mathcal{O}((|\mathsf{pals}(T)| + |T|) \log^{\mathcal{O}(1)} |T|)$ time.

Acknowledgments

We thank Timothy Chan for his suggestions regarding the test algorithm. Paweł Gawrychowski's work was done during 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. 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. https://doi.org/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. https://doi.org/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. https://doi.org/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. https://doi.org/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. https://doi.org/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. https://doi.org/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. https://doi.org/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. https://doi.org/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. https://doi.org/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. https://doi.org/10.1016/j.ipl.2010.07.018.

[15] Torben Hagerup, Peter Bro Miltersen, and Rasmus Pagh. Deterministic dictionaries. *J. Algorithms*, 41(1):69–85, 2001. https://doi.org/10.1006/jagm.2001.1171.

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

[17] 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.

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

[19] Takuya Mieno, Mitsuru Funakoshi, and Shunsuke Inenaga. Computing Palindromes on a Trie in Linear Time. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, volume 248 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:15, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.ISAAC.2022.15.

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