# Computing Runs on a Trie

**Ryo Sugahara**
Department of Informatics, Kyushu University, Japan
sugahara.ryo.408@s.kyushu-u.ac.jp

**Yuto Nakashima**
Department of Informatics, Kyushu University, Japan
yuto.nakashima@inf.kyushu-u.ac.jp

**Shunsuke Inenaga**
Department of Informatics, Kyushu University, Japan
inenaga@inf.kyushu-u.ac.jp

**Hideo Bannai**
Department of Informatics, Kyushu University, Japan
bannai@inf.kyushu-u.ac.jp

**Masayuki Takeda**
Department of Informatics, Kyushu University, Japan
takeda@inf.kyushu-u.ac.jp

## Abstract

A maximal repetition, or run, in a string, is a maximal periodic substring whose smallest period is at most half the length of the substring. In this paper, we consider runs that correspond to a path on a trie, or in other words, on a rooted edge-labeled tree where the endpoints of the path must be a descendant/ancestor of the other. For a trie with $n$ edges, we show that the number of runs is less than $n$. We also show an $O(n\sqrt{\log n}\log\log n)$ time and $O(n)$ space algorithm for counting and finding the shallower endpoint of all runs. We further show an $O(n \log n)$ time and $O(n)$ space algorithm for finding both endpoints of all runs. We also discuss how to improve the running time even more.

**2012 ACM Subject Classification** Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Combinatorics on words

**Keywords and phrases** runs, Lyndon words

**Digital Object Identifier** 10.4230/LIPIcs.CPM.2019.23

**Related Version** https://arxiv.org/abs/1901.10633

**Funding** *Yuto Nakashima*: Supported by JSPS KAKENHI Grant Number JP18K18002.
*Shunsuke Inenaga*: Supported by JSPS KAKENHI Grant Number JP17H01697.
*Hideo Bannai*: Supported by JSPS KAKENHI Grant Number JP16H02783.
*Masayuki Takeda*: Supported by JSPS KAKENHI Grant Number JP18H04098.

## 1 Introduction

Repetitions are fundamental characteristics of strings, and their combinatorial properties as well as their efficient computation has been a subject of extensive studies. Maximal periodic substrings, or *runs*, is one of the most important types of repetitions, since they essentially capture all occurrences of consecutively repeating substrings in a given string. One of the reasons which makes runs important and interesting is that the number of runs contained in a given string of length $n$ is $O(n)$ [18], in fact, less than $n$ [1], and can be computed in $O(n)$ time assuming a constant or integer alphabet [18, 1], or in $O(n\alpha(n))$ time for general ordered alphabets [9], where $\alpha$ is the inverse Ackermann function.

In this paper, we consider runs that correspond to a path on a trie, or in other words, on a rooted edge-labeled tree where the endpoints of the path must be a descendant/ancestor of the other. The contributions of this paper are as follows. For a trie with $n$ edges, we show:

- that the number of such periodically maximal paths is linear, and in fact less than $n$.
- an $O(n\sqrt{\log n}\log\log n)$ time and $O(n)$ space algorithm for counting and finding the shallower endpoint of all such paths.
- an $O(n\sqrt{\log n}\log^2\log n)$ time and $O(n)$ space algorithm for finding both endpoints of such paths.

Furthermore, we also discuss how to improve the running time even more.

### 1.1 Related Work

A similar problem was considered in [16, 8, 17], but differs in three aspects: they consider *distinct* repetitions with *integer powers* on an *unrooted* (or undirected) tree. In this work, we consider *occurrences* of repetitions with maximal (possibly fractional) powers on a *rooted* (directed) tree.

## 2 Preliminaries

### 2.1 Strings, Periods, Maximal Repetitions, Lyndon Words

Let $\Sigma = \{1, \ldots, \sigma\}$ denote the alphabet. We consider an integer alphabet, i.e., $\sigma = n^c$ for some constant $c$. $\Sigma^*$ is the set of strings over $\Sigma$. For any string $w \in \Sigma^*$, let $w[i]$ denote the $i$th symbol of $w$, and $|w|$ the length of $w$. For any $1 \leq i \leq j \leq |w|$, let $w[i..j] = w[i] \cdots w[j]$. For technical reasons, we assume that $w$ is followed by a distinct character (i.e. $w[|w|+1]$) in $\Sigma$ that does not occur in $w[1..|w|]$.

A string is *primitive*, if it is not a concatenation of 2 or more complete copies of the same string. A string $w = u^2$, for some string $u$, is called a *square*, and in particular, if $u$ is primitive, then $w$ is called a *primitively rooted square*. An integer $1 \leq p \leq |w|$ is called a period of $w$, if $w[i] = w[i+p]$ for all $1 \leq i \leq |w| - p$. The smallest period of $w$ will be denoted by $per(w)$. For any period $p$ of $w$, there exists a string $x$, called a *border* of $w$, such that $|x| = |w| - p$ and $w = xy = zx$ for some $y, z$. A string is a *repetition*, if its smallest period is at most half the length of the string. A *maximal repetition*, or *run*, is a maximal periodic substring that is a repetition, i.e., a maximal repetition of a string $w$ is an interval $[i..j]$ of positions where $per(w[i..j]) \leq (j-i+1)/2$, and $per(w[i..j]) \neq per(w[i'..j'])$ for any $1 \leq i' \leq i$ and $j \leq j' \leq n$ such that $i' \neq i$ or $j' \neq j$. In other words, a run contains at least two consecutive occurrences of a substring of length $p$, and the periodicity does not extend to the left or right of the run. The smallest period of the run will be called the period of the run. The fraction $(j-i+1)/p \geq 2$ is called the *exponent* of the run.

Let $\prec_0$ denote an arbitrary total ordering on $\Sigma$, as well as the lexicographic ordering on $\Sigma^*$ induced by this ordering. We also consider the reverse ordering $\prec_1$ on $\Sigma$ (i.e., $\forall a, b \in \Sigma, a \prec_0 b \iff b \prec_1 a$), and the induced lexicographic ordering on $\Sigma^*$. For $\ell \in \{0, 1\}$, let $\bar{\ell} = 1 - \ell$. A string $w$ is a *Lyndon word* w.r.t. to a given lexicographic ordering, if $w$ is lexicographically smaller than any of its proper suffixes. A well known fact is that a Lyndon word cannot have a non-empty border.

Crochemore et al. observed that in any run $[i..j]$ with period $p$, and any lexicographic ordering, there exists a substring of length $p$ in the run, that is a Lyndon word [10, 11]. Such Lyndon words are called L-roots. Below, we briefly review the main result of [1] which essentially tied longest Lyndon words starting at specific positions within the run, to L-roots of runs. This will be the basis for our new results for tries.

▶ **Lemma 1** (Lemma 3.2 of [1]). *For any position $1 \leq i \leq |w|$ of string $w$, let $\ell \in \{0,1\}$ be such that $w[k] \prec_\ell w[i]$ for $k = \min\{k' \mid w[k'] \neq w[i], k' > i\}$. Then, the longest Lyndon word that starts at position $i$ is $w[i..i]$ w.r.t $\prec_\ell$, and $w[i..j]$ for some $j \geq k$ w.r.t. $\prec_{\bar{\ell}}$.*[1]

▶ **Lemma 2** (Lemma 3.3 of [1]). *Let $r = [i..j]$ be a run in $w$ with period $p$, and let $\ell \in \{0,1\}$ be such that $w[j+1] \prec_\ell w[j+1-p]$. Then, any L-root $w[i'..j']$ of $r$ with respect to $\prec_\ell$ is the longest Lyndon word with respect to $\prec_\ell$ that is a prefix of $w[i'..|w|]$.*

Since an L-root cannot be shared by two different runs, it follows from Lemma 2 that the number of runs is at most $2n$, since each position can be the starting point of at most two L-roots that correspond to distinct runs. In [1], a stronger bound of $n$ was shown from the observation that each run contains at least one L-root that does not begin at the first position of the run, and that the two longest Lyndon words starting at a given position for the two lexicographic orders cannot simultaneously be such L-roots of runs. This is because if $w[i'..i']$ and $w[i'..j']$ were L-roots and the runs start before position $i'$, then, from the periods of the two runs, it must be that $w[i'-1] = w[i'] = w[j']$ contradicting that $w[i'..j']$ is a Lyndon word and cannot have a non-empty border. In Section 3.1, we will see that the last argument does not completely carry over to the case of tries, but show that we can still improve the bound again to $n$.

The above lemmas also lead to a new linear time algorithm for computing all runs, that consists of the following steps:

1. compute the longest Lyndon word that starts at each position for both lexicographic orders $\prec_0$ and $\prec_1$,
2. check whether there is a run for which the longest Lyndon word corresponds to an L-root.

There are several ways to compute the first step in amortized constant time for each position, but it essentially involves computing the next smaller values (NSV) in the inverse suffix array of the string. We describe the algorithm in more detail in Section 3.2, and will see that the amortization of the standard algorithm does not carry over to the trie case. We give a new linear time algorithm using the static tree set-union data structure (more specifically, decremental nearest marked ancestor queries) [14], which *does* carry over to the trie case.

The second step can be computed in constant time per candidate L-root with linear time-preprocessing, by using longest common extension queries (e.g. [13]) in the forward and reverse directions of the string. Unfortunately again, this does not directly carry over to the trie case because, as far as we know, longest common extension queries on trees can be computed in constant time only in the direction toward the root of the trie, when space is restricted to linear in the size of the trie. In Section 3.3, we will show how to apply the range predecessor/successor data structure of [2] to this problem.

## 2.2 Common Suffix Trie

A *trie* is a rooted tree with labeled edges, such that each edge to the children of a node is labeled with distinct symbols. A trie can be considered as representing a set of strings obtained by concatenating the labels on a root to leaf path. 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 $c_1, \ldots, c_n \in \Sigma$

[1] Note that $j$ becomes $|w|+1$ when $i = |w|$.

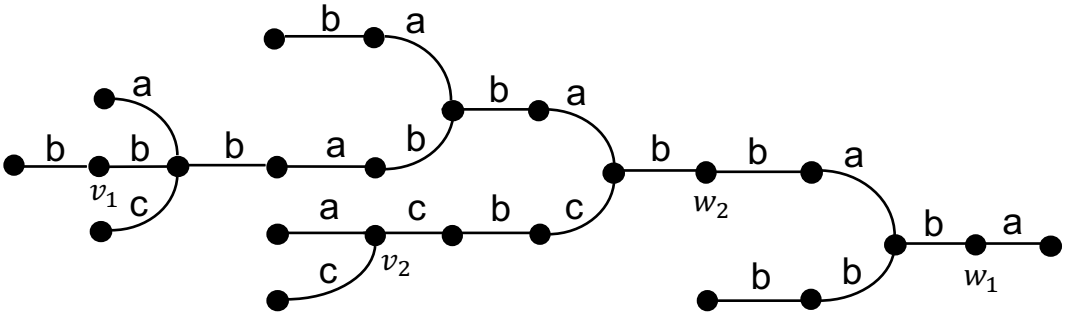


**Figure 1** Example of runs in a trie. $(v_1, w_1)$ is a run with period 3, and $(v_2, w_2)$ is a run with period 2.

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 of the tree. Such tries are called *common suffix tries* [7]. We will use the terms parent/child/ancestor/descendant with the standard meaning based on the undirected trie, e.g., the root is an ancestor of all nodes in the trie. For any node $u$ of the trie, $par(u)$ will denote its parent node. Also, we consider a node to be an ancestor/descendant of itself.

For any nodes $u, v$ of the trie where $v$ is an ancestor of $u$, let $str(u, v)$ denote the string obtained by concatenating the labels on the path from $u$ to $v$. For technical reasons, we assume that the root node has an auxiliary parent node $\perp$, where the edge is labeled by a distinct character in $\Sigma$ that is not used elsewhere in the trie. We denote by $suf(v)$ the string obtained by concatenating the labels on the path from $v$ to $\perp$, i.e., $suf(v) = str(v, \perp)$. Such strings will be called a *suffix* of the trie.

Note that a trie can be pre-processed in linear time so that for any node $v$, the ancestor of node $v$ at an arbitrary specified depth $d$ can be obtained in constant time (e.g. [4]).

# 3 Runs in a Trie

## 3.1 The Number of Runs in a Trie

We first define runs on a trie. A run $(v_i, v_j)$ on a trie $T$ is a maximal periodic path with endpoints $v_i$ and $v_j$, where $v_j$ is an ancestor of $v_i$ and $str(v_i, v_j)$ is a repetition. More precisely, $per(str(v_i, v_j)) \leq |str(v_i, v_j)|/2$, and for any descendant $v_{i'}$ of $v_i$ and ancestor $v_{j'}$ of $v_j$, $per(str(v_i, v_j)) \neq per(str(v_{i'}, v_{j'}))$ if $v_{i'} \neq v_i$ or $v_{j'} \neq v_j$.

Noticing that for any node in the trie its parent is unique, it is easy to see that analogies of Lemmas 1 and 2 hold for tries. Thus, we have the following.

▶ **Corollary 3.** *For any node $v$ except the root or $\perp$, let $w_v = suf(v)$, and $\ell \in \{0, 1\}$ be such that $w_v[k] \prec_\ell w_v[1]$ for $k = \min\{k' \mid w_v[k'] \neq w_v[1], k' > 1\}$. Then, the longest Lyndon word that is a prefix of $w_v$ is $w_v[1..1]$ w.r.t. $\prec_\ell$ and $w_v[1..j]$ w.r.t. $\prec_{\bar{\ell}}$ for some $j \geq k$.*

▶ **Corollary 4.** *Let $r = (v_i, v_j)$ be a run with period $p$ in the trie, $w_v = suf(v_i)$, and $\ell \in \{0, 1\}$ be such that $w_v[x + 1] \prec_\ell w_v[x - p + 1]$, where $x = |str(v_i, v_j)|$. Then, any $L$-root $str(v_{i'}, v_{j'})$ of the run with respect to $\prec_\ell$ is the longest Lyndon word that is a prefix of $suf(v_{i'})$.*

Since we assumed that the edge labels of the children of a given node in a trie are distinct, a given L-root can only correspond to one distinct run; i.e., the extension of the period in both directions from an L-root is uniquely determined. Therefore, the argument for standard strings carries over to the trie case, and it follows that the number of runs must be less than $2n$. We further observe the following

▶ **Theorem 5.** *The maximum number of runs in a trie with $n$ edges is less than $n$.*

**Proof.** Suppose $str(v_{i'}, par(v_{i'}))$ and $str(v_{i'}, v_{j'})$ are simultaneously L-roots of runs respectively w.r.t. $\prec_\ell$ and $\prec_{\bar{\ell}}$, and that they do not start at the beginning of the runs. Let $p = |str(v_{i'}, v_{j'})|$ and $w_{v_{i'}} = suf(v_{i'})$. If $v_{i'}$ has only one child, this leads to a contradiction using the same argument as the case for strings; i.e., if $u$ is the child of $v_{i'}$, then, from the periods of the two runs, $str(u, v_{i'}) = w_{v_{i'}}[1] = w_{v_{i'}}[p]$ contradicting that $w_{v_{i'}}[1..p]$ is a Lyndon word and cannot have a non-empty border. Thus, $v_{i'}$ must have at least two children, $u, u'$, where $str(u, v_{i'}) = w_{v_{i'}}[1]$ and $str(u', v_{i'}) = w_{v_{i'}}[p]$. Let $k$ be the number of branching nodes in the trie. Then, the number of leaves is at least $k + 1$. Since a run cannot start before a leaf node, this means that a longest Lyndon word starting at a leaf cannot be an L-root that does not start at the beginning of the run. Therefore, although there can be at most $k$ nodes such that both longest Lyndon words are such L-roots, there exist at least $k + 1$ nodes where both are not. Thus, the theorem holds. ◀

In a similar way to Theorem 3.6 of [1], we can bound the sum of exponents of all runs in a trie.

▶ **Corollary 6.** *The sum of exponents of all runs in a trie with $n$ edges is less than $3n$.*

**Proof.** A given run $r$ with exponent $e_r$ contains at least $\lfloor e_r - 1 \rfloor \geq 1$ occurrences of its L-roots that do not start at the beginning of the run, each corresponding to a longest Lyndon word starting at that position. From the proof of Theorem 5, the total number of these L-roots is less than $n$. Let *Runs* denote the set of all runs in the trie. Then, $\sum_{r \in Runs}(e_r - 2) \leq \sum_{r \in Runs} \lfloor e_r - 1 \rfloor < n$, and the Corollary follows. ◀

## 3.2 Computing Longest Lyndon Words

Next, we consider the problem of computing, for any node $v$ of the trie, the longest Lyndon word that is a prefix of $suf(v)$. We first describe the algorithm for strings, which is based on the following lemma.

▶ **Lemma 7.** *For any string $w$ and position $1 \leq i < |w|$, the longest Lyndon word starting at $i$ w.r.t. $\prec$ is $w[i..j-1]$, where $j$ is such that $j = \min\{k > i \mid w[k..|w|] \prec w[i..|w|]\}$.*

**Proof.** Let $j = \min\{k > i \mid w[k..|w|] \prec w[i..|w|]\}$. By definition, we have $w[k..|w|] \succ w[i..|w|]$ for any $i < k < j$. For any such $k$, $w[k..|w|] = w[k..j-1]w[j..|w|] \succ w[i..i+(j-1-k)]w[i+(j-k)..j-1]w[j..|w|] = w[i..|w|]$. If the longest common prefix of $w[k..|w|]$ and $w[i..|w|]$ is longer than or equal to $w[k..j-1]$, this implies $w[j..|w|] \succ w[i+(j-k)..j-1]w[j..|w|] \succ w[i..|w|]$, a contradiction. Therefore, the longest common prefix of $w[k..|w|]$ and $w[i..|w|]$ must be shorter than $w[k..j-1]$, implying that $w[i..j-1] \prec w[k..j-1]$. Thus, $w[i..j-1]$ is a Lyndon word. Suppose $w[i..k]$ is a Lyndon word for some $k \geq j$. Then, $w[i..k] \prec w[j..k]$. Since $|w[i..k]| > |w[j..k]|$, $w[i..k]$ cannot be a prefix of $w[j..k]$ which implies $w[i..|w|] \prec w[j..|w|]$, contradicting the definition of $j$. Thus, $w[i..j-1]$ is the longest Lyndon word starting at $i$. ◀

From Lemma 7, the longest Lyndon word starting at each position of a string $w$ can be computed in linear time, given the inverse suffix array of $w$. The inverse suffix array $ISA[1..|w|]$ of $w$ is an array of integers such that $ISA[i] = j$ when $w[i..|w|]$ is the lexicographically $j$th

smallest suffix of $w$. That is, the $j$ in Lemma 7 can be restated as $j = \min\{k > i \mid ISA[k] < ISA[i]\}$. This can be restated as the problem of finding the next smaller value (NSV) for each position of the $ISA$, for which there exists a simple linear time algorithm (e.g. [24]) as show in Algorithm 1. The linear running time comes from a simple amortized analysis;

**Algorithm 1:** Computing NSV on array $A$ of integers.

```
   // assumes A[n+1] is smaller than all values in A.
1  NSV[n] = n + 1;
2  for i = n − 1 to 1 do
3  |  x = i + 1;
4  |  while A[i] ≤ A[x] do
5  |  |  x = NSV[x];
6  |  NSV[i] = x;
```

in the while loop, $NSV[x]$ is only accessed once for any position $x$ since $NSV[i]$ is set to a larger value and thus will subsequently be skipped.

Since, as before, the parent of a node is unique, Lemma 7 carries over to the trie case. We can assign the lexicographic rank $ISA[v]$ of $suf(v)$ to each node $v$ in linear time from the suffix tree of the trie, which is a compacted trie containing all and only suffixes of the trie.

▶ **Theorem 8** (suffix tree of a trie [7, 22]). *The suffix tree of a trie on a constant or integer alphabet can be represented and constructed in $O(n)$ time.*

The problem now is to compute, for each node $v_i$, the closest ancestor $v_j$ of $v_i$ such that the lexicographic rank of $suf(v_j)$ is smaller than that of $suf(v_i)$. Algorithm 1 can be modified to correctly compute the $NSV$ values on the trie; the for loop is modified to enumerate nodes in some order such that the parent of a considered node is already processed, and line 3 can be changed to $x = par(x)$. However, the amortization will not work; the existence of branching paths means there can be more than one child of a given node, and the same position (node) $x$ could be accessed in the while loop for multiple paths, leading to a super-linear running time.

To overcome this problem, we introduce a new, (conceptually) simple linear time algorithm based on nearest marked ancestor queries.

▶ **Theorem 9** (decremental nearest marked ancestor [14]). *A given tree can be processed in linear time such that all nodes are initially marked, and the following operations can be done in amortized constant time:*

- *$nma(v)$: return the nearest ancestor node of $v$ that is marked.*
- *$unmark(v)$: unmark the node $v$.*

The pseudo-code of our algorithm is shown in Algorithm 2.

**Algorithm 2:** Computing NSV on trie with values $ISA$.

```
1  Preprocess trie for decremental nearest marked ancestor;
2  foreach node v in decreasing order of ISA[v] do
3  |  unmark(v);
4  |  NSV[v] = nma(v);
```

▶ **Theorem 10.** *Given a trie of size $n$, the longest Lyndon word that is a prefix of $suf(v)$ for each $v$ can be computed in total $O(n)$ time and space.*

**Proof.** It is easy to see the linear running time of Algorithm 2. The correctness is also easy to see, because the nodes are processed in decreasing order of lexicographic rank, and thus, all and only nodes with larger lexicographic rank are unmarked. ◀

## 3.3 Computing Runs

To compute all runs in a trie, we extend the algorithm for strings to the trie case. After computing the longest Lyndon word that is a prefix of $suf(v)$ for each node $v$ for the two lexicographic orderings $\prec_0$ and $\prec_1$, we must next see if they are $\mathsf{L}$-roots of runs by checking how long the periodicity extends. Given a longest Lyndon word $y = str(v_i, v_j)$ w.r.t. $\prec_\ell$ that starts at $v_i$, we can compute the longest common extension from nodes $v_i$ and $v_j$ towards the root, i.e., the longest common prefix $z = str(v_j, v_k)$ between $suf(v_i)$ and $suf(v_j)$. To avoid outputting duplicate runs, $y$ will be a candidate $\mathsf{L}$-root only if $|z| < |y|$ and $str(v_i)[|y| + |z| + 1] \prec_\ell str(v_i)[|z| + 1]$. Using the suffix tree of the trie, this longest common extension query can be computed in constant time after linear time preprocessing, since it amounts to lowest common ancestor queries (e.g. [3]). The central difficulty of our problem is in computing the longest common extension in the opposite direction, i.e. towards the leaves, because the paths can be branching. We cannot solve this problem by simply considering longest common extensions on the common suffix trie for the reverse strings, since, as observed in Section 2, this can lead to a quadratic blow-up in the size of the trie.

We overcome this problem by reducing the longest common extension query in the leaf direction to several queries of finding the lexicographically closest suffix that is in a specific subtree and at a specific depth, combined with some naive traversal. We use the following result for range predecessor queries multiple times to achieve Lemma 12. A range predecessor problem is to pre-process $n$ points on a $[1, n]^2$ grid where all points differ in both coordinates, so as to answer the query: given three integers $x_1, x_2, y$ find the point $rpred(x_1, x_2, y) = (u, v)$ such that $x_1 \leq u \leq x_2$, and $v$ is the largest such that $v \leq y$.

▶ **Theorem 11** (Range Predecessor Queries (Theorem 5.1 in [2])). *Given $n$ points from the grid $[1, n]^2$, we can in $O(n\sqrt{\log n})$ time build a data structure that occupies $O(n \log n)$ bits of space and that answers range predecessor queries in $O(\sqrt{\log n} \log \log n)$ time. The construction uses $O(n \log n)$ bits of working space.*

By converting each of the $y$ coordinates to $n - y + 1$, range successor queries $rsucc(x_1, x_2, y)$ can also be achieved in the same time/space bounds.

▶ **Lemma 12.** *A trie of size $n$ can be pre-processed in $O(n\sqrt{\log n})$ time and $O(n)$ space so that given a node $v$ and a positive integer $d > |suf(v)|$, we can answer in $O(\sqrt{\log n} \log \log n)$ time, the node $v'$ such that $v'$ is a descendant of $v$ where $|suf(v')| = d$ and of all such nodes, has the longest common prefix with $suf(v)$.*

**Proof.** We construct several range predecessor/successor data structures of Theorem 11, where each point in the grid corresponds to a node in the trie. The first coordinate (which we will consider as the identifier of the node) is given by the order of nodes that would appear in a breadth-first traversal of the trie, where nodes of the same depth are ordered as they would appear in a depth-first traversal. Thus, for any depth $d$, all nodes at depth $d$ can be represented in an interval $[i_d..j_d]$. The second coordinate is given by each of the following three types of values in the range $[1, n]$, and can be assigned to each node $v$ by a simple depth-first traversal on the trie.

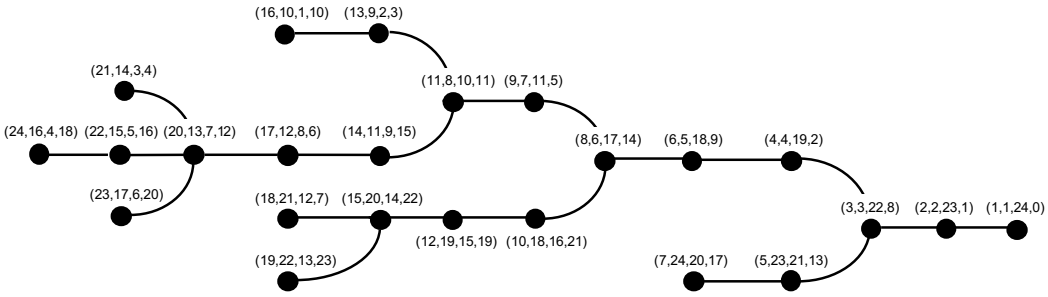


**Figure 2** An example of values assigned for each node $v$ of the trie for Figure 1. Each 4-tuple shows $(v, b_v, e_v, l_v)$, where $v$ is the identifier of the node, $b_v$ is the pre-order rank, $e_v$ is the post-order rank, and $l_v$ is the lexicographic rank of $suf(v)$.

- $b_v$: pre-order rank of node $v$ in a depth-first traversal.
- $e_v$: post-order rank of node $v$ in a depth-first traversal.
- $l_v$: lexicographic rank of $suf(v)$ (i.e. $ISA[v]$).

The construction costs $O(n\sqrt{\log n})$ time and $O(n)$ (words of) space due to Theorem 11.

Now, the desired node $u$ that is the answer to our query lies in the range $[i_d..j_d]$. The nodes corresponding to descendants of $v$ further lie in a sub-range, namely, $[i_{d,v}..j_{d,v}]$, which can be computed by $(i_{d,v}, b') = rsucc(i_d, j_d, b_v)$ and $(j_{d,v}, e') = rpred(i_d, j_d, e_v)$, respectively using the successor and predecessor data structures for pre-order rank and post-order rank.

Finally, the node $u$ which gives the longest common prefix must be one of the lexicographically closest ones in this range, i.e., it is one of $(u_1, l'_1) = rpred(i_{d,v}, j_{d,v}, l_v)$ or $(u_2, l'_2) = rsucc(i_{d,v}, j_{d,v}, l_v)$, using the predecessor/successor data structure for lexicographic rank. The longest common prefix between $suf(u_1)$ and $suf(v)$ as well as that between $suf(u_2)$ and $suf(v)$ can be computed in constant time with linear time pre-processing, as mentioned before. Thus, the total time is $O(\sqrt{\log n} \log\log n)$ for the four range queries. ◀

We first use Lemma 12, once for each candidate L-root, to determine whether the periodicity of the candidate L-root can extend long enough in the leaf direction to form a run.

▶ **Theorem 13.** *Given a trie of size $n$, we can in $O(n\sqrt{\log n}\log\log n)$ time and $O(n)$ space, count the total number of runs in the trie, as well as identify the shallower endpoint of all runs.*

**Proof.** Let $y = str(v_i, v_j)$ be a candidate L-root and let $z = str(v_j, v_k)$ be its extension in the root direction as described in the beginning of Section 3.3. Let $p = |y|$. Also, let $v'$ be the node on the path from $v_i$ and $v_j$ at depth $|suf(v_k)| + p$. We use Lemma 12 in order to check if there exists a node $v_l$ that is a descendant of $v'$ and at depth $|suf(v_k)| + 2p$ such that the longest common prefix between $str(v_l)$ and $str(v')$ is at least $p$. If $v_l$ does not exist, then the L-root cannot be extended to a run. If $v_l$ does exist, this implies that $str(v_l, v') = str(v', v_k)$, and since $z = str(v_j, v_k) = str(v_i, v')$ is a suffix of $str(v', v_k)$ and thus of $str(v_l, v')$, $v_l$ must be a descendant of $v_i$ because labels on the edges to child nodes are distinct. Thus, the prefix of length $2p$ of $suf(v_l)$ has period $p$, contains $str(v_i, v_j)$, and the periodicity ends at $v_k$, i.e., it is an endpoint of a run with $str(v_i, v_j)$ as an L-root.

Such queries are conducted at most once for each candidate L-root, so the total time is $O(n\sqrt{\log n}\log\log n)$. ◀

Finally, we describe how to compute the other endpoint. We make use of Theorem 13 and the following static dictionary.

▶ **Theorem 14** (Static Dictionary (Theorem 1 in [21])). *Suppose that a set of $n$ integers from the universe $\{0, 1, \ldots, n^{O(1)}\}$ is given. A static linear space dictionary on that set can be deterministically constructed on a word RAM in time $O(n \log \log n)$, so that lookups to the dictionary take constant time.*

▶ **Theorem 15.** *Given a trie of size $n$, we can compute in $O(n \log n)$ time and $O(n)$ space, all runs $(v_i, v_j)$ in the trie.*

**Proof.** After confirming an occurrence of the run as described in Theorem 13, we further repeatedly use Lemma 12 to check if the periodicity can be further extended in the leaf direction by length $p$. The total number of times that this is repeated is bounded by the sum of exponents of all runs in the trie, which is linear (Corollary 6). Therefore, the total time for this is $O(n\sqrt{\log n} \log \log n)$.

It remains to find the remaining extension shorter than the period $p$ of the run. We compute this extension naively edge by edge. We know which character we need to extend by, since we know $str(v_i, v_j)$ is what is repeating. To avoid the $O(\log \sigma)$ factor for finding the child node of a branching node, we use the static dictionary by Ruzic [21], so that each child node can be found in constant time. This can be done with $O(n \log \log n)$ time and $O(n)$ space pre-processing using Theorem 14; each edge can be considered as a pair of integers from 1 to $n$ and 1 to $n^c$, representing a unique id of the node, and the label on the edge. The pair can be encoded as an integer from 1 to $n^{O(1)}$, where the encoding and decoding can be done in constant time. Given a node and edge label, the child can be obtained in constant time by looking up the dictionary.

Finally, since each naive extension implies an occurrence of a primitively rooted square, the total number of naive extensions is bounded by the total number of primitively rooted squares that occur in the trie. Due to the three squares lemma (Lemma 10 of [12]), it follows that for each node $v$, there can only be $O(\log |suf(v)|)$ primitively rooted squares that are prefixes of $suf(v)$. Thus, it follows that their total number is $O(n \log n)$.

From the above arguments, computing both endpoints of all runs can be done in $O(n \log n)$ time using $O(n)$ space. ◀

## 4 Discussion

Shortly after the submission of the paper, we realized that running time could be improved by using a doubling + binary search when extending the run in the leaf direction, rather than a naive traversal. If the remaining length of a run is $t$, the query of Lemma 12 is conducted $O(\log t)$ times. Let $Runs$ be the set of all runs in the trie, and let $t_r$ denote the length of the remaining extension for run $r \in Runs$. The total number of queries is thus $\sum_{r \in Runs} \log t_r$, where $\sum_{r \in Runs} t_r = O(n \log n)$ as mentioned in the proof of Theorem 15. Since $\sum_{r \in Runs} \log t_r = \log(\prod_{r \in Runs} t_r)$, this is maximized when each $t_r$ has the same length, i.e., $t_r = \Theta(\frac{n \log n}{|Runs|})$. Thus, $\sum_{r \in Runs} \log t_r = O(|Runs| \log \frac{n \log n}{|Runs|})$. Noticing that $|Runs| \log \frac{n}{|Runs|} = O(n)$, the total number of times Lemma 12 is used can be bounded by $O(n \log \log n)$. Therefore, the total running time would be $O(n\sqrt{\log n}(\log \log n)^2)$.

Furthermore, after posting our paper with the above improvement to arXiv [23], Tomohiro I pointed us to the paper by Bille et al. [5], where they consider LCE queries on tries in the leaf direction. The path-tree query in their paper can be used in place of Lemma 12 and is more powerful. A path-tree query, given nodes $v_1, v_2$ and $w$, where $v_2$ is a descendant of $v_1$, returns the longest common prefix of the path from $v_1$ to $v_2$, and any path from $w$ to a descendant leaf.

▶ **Theorem 16** (Theorem 2 of [5]). *For a tree $T$ with $n$ nodes, a data structure of size $O(n)$ can be constructed in $O(n)$ time to answer path–tree LCE queries in $O((\log \log n)^2)$ time.*

The number of times the path-tree query is used can be bounded by the total sum of exponents of runs, and thus is linear. Therefore, the total computation time can be improved to $O(n(\log\log n)^2)$.

## 5 Conclusion

We generalized the notion of runs in strings to runs in tries, and showed that the analysis of the maximum number of runs, as well as algorithms for computing runs can be extended and adapted to the trie case, but with an increase in running time.

Our algorithm can output all primitively rooted squares in $O(n \log n)$ time, which is tight, since there can be $\Theta(n \log n)$ primitively rooted squares in a string, e.g., Fibonacci words [20], and thus in a trie.

An obvious open problem is whether there exists a linear time algorithm for computing all runs in a trie. For strings, there exists another linear time algorithm for computing all runs that is based on the Lempel-Ziv parsing [18]. It is not clear how this algorithm could be extended to the case of tries. The case for general ordered alphabets, instead of integer alphabets, is another open problem [6, 19, 15, 9].

## References


1 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "Runs" Theorem. *SIAM J. Comput.*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.

2 Djamal Belazzougui and Simon J. Puglisi. Range Predecessor and Lempel-Ziv Parsing. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 2053–2071. SIAM, 2016. doi:10.1137/1.9781611974331.ch143.

3 Michael A. Bender and Martin Farach-Colton. The LCA Problem Revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

4 Michael A. Bender and Martin Farach-Colton. The Level Ancestor Problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004. doi:10.1016/j.tcs.2003.05.002.

5 Philip Bille, Pawel Gawrychowski, Inge Li Gørtz, Gad M. Landau, and Oren Weimann. Longest common extensions in trees. *Theor. Comput. Sci.*, 638:98–107, 2016. doi:10.1016/j.tcs.2015.08.009.

6 Dany Breslauer. *Efficient String Algorithmics*. PhD thesis, Columbia University, 1992.

7 Dany Breslauer. The Suffix Tree of a Tree and Minimizing Sequential Transducers. *Theor. Comput. Sci.*, 191(1-2):131–144, 1998. doi:10.1016/S0304-3975(96)00319-2.

8 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, Wojciech Tyczynski, and Tomasz Walen. The Maximum Number of Squares in a Tree. In Juha Kärkkäinen and Jens Stoye, editors, *Combinatorial Pattern Matching - 23rd Annual Symposium, CPM 2012, Helsinki, Finland, July 3-5, 2012. Proceedings*, volume 7354 of *Lecture Notes in Computer Science*, pages 27–40. Springer, 2012. doi:10.1007/978-3-642-31265-6_3.

9 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Ritu Kundu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Near-Optimal Computation of Runs over General Alphabet via Non-Crossing LCE Queries. In Shunsuke Inenaga, Kunihiko Sadakane, and Tetsuya Sakai, editors, *String Processing and Information Retrieval - 23rd International Symposium, SPIRE 2016, Beppu, Japan, October 18-20,*

*2016, Proceedings*, volume 9954 of *Lecture Notes in Computer Science*, pages 22–34, 2016. doi:10.1007/978-3-319-46049-9_3.

10 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. The maximal number of cubic runs in a word. *J. Comput. Syst. Sci.*, 78(6):1828–1836, 2012. doi:10.1016/j.jcss.2011.12.005.

11 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Extracting powers and periods in a word from its runs structure. *Theor. Comput. Sci.*, 521:29–41, 2014. doi:10.1016/j.tcs.2013.11.018.

12 Maxime Crochemore and Wojciech Rytter. Sqares, Cubes, and Time-Space Efficient String Searching. *Algorithmica*, 13(5):405–425, 1995. doi:10.1007/BF01190846.

13 Johannes Fischer and Volker Heun. Theoretical and Practical Improvements on the RMQ-Problem, with Applications to LCA and LCE. In Moshe Lewenstein and Gabriel Valiente, editors, *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5-7, 2006, Proceedings*, volume 4009 of *Lecture Notes in Computer Science*, pages 36–48. Springer, 2006. doi:10.1007/11780441_5.

14 Harold N Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–221, 1985.

15 Pawel Gawrychowski, Tomasz Kociumaka, Wojciech Rytter, and Tomasz Walen. Faster Longest Common Extension Queries in Strings over General Alphabets. In Roberto Grossi and Moshe Lewenstein, editors, *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27-29, 2016, Tel Aviv, Israel*, volume 54 of *LIPIcs*, pages 5:1–5:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CPM.2016.5.

16 Tomasz Kociumaka, Jakub Pachocki, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Efficient counting of square substrings in a tree. *Theoretical Computer Science*, 544:60–73, 2014.

17 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. String Powers in Trees. *Algorithmica*, 79(3):814–834, 2017. doi:10.1007/s00453-016-0271-3.

18 Roman M. Kolpakov and Gregory Kucherov. Finding Maximal Repetitions in a Word in Linear Time. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.

19 Dmitry Kosolobov. Computing runs on a general alphabet. *Inf. Process. Lett.*, 116(3):241–244, 2016. doi:10.1016/j.ipl.2015.11.016.

20 M. Lothaire. *Periodic Structures in Words*, page 430–477. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2005. doi:10.1017/CBO9781107341005.009.

21 Milan Ruzic. Constructing Efficient Dictionaries in Close to Sorting Time. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2008. doi:10.1007/978-3-540-70575-8_8.

22 Tetsuo Shibuya. Constructing the Suffix Tree of a Tree with a Large Alphabet. In Alok Aggarwal and C. Pandu Rangan, editors, *Algorithms and Computation, 10th International Symposium, ISAAC '99, Chennai, India, December 16-18, 1999, Proceedings*, volume 1741 of *Lecture Notes in Computer Science*, pages 225–236. Springer, 1999. doi:10.1007/3-540-46632-0_24.

23 Ryo Sugahara, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing runs on a trie. *CoRR*, abs/1901.10633, 2019. arXiv:1901.10633.

24 Wikipedia. All nearest smaller values — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=All%20nearest%20smaller%20values&oldid=882430084, 2019. [Online; accessed 29-March-2019].