Tight Upper and Lower Bounds on Suffix Tree Breadth

Badkobeh, Golnaz

2021-01-16

Badkobeh , G , Gawrychowski , P , Kärkkäinen , J , Puglisi , S & Zhukova , B 2021 , ' Tight Upper and Lower Bounds on Suffix Tree Breadth ' , Theoretical Computer Science , vol. 854 , pp. 63-67 . https://doi.org/10.1016/j.tcs.2020.11.037

http://hdl.handle.net/10138/341373
https://doi.org/10.1016/j.tcs.2020.11.037

Tight Upper and Lower Bounds on Suffix Tree Breadth

Golnaz Badkobeh[a], Paweł Gawrychowski[b], Juha Kärkkäinen[c], Simon J. Puglisi[c,*], Bella Zhukova[c]

[a] *Department of Informatics, Goldsmiths University of London, London, United Kingdom*
[b] *University of Wrocław, Wrocław, Poland*
[c] *Helsinki Institute for Information Technology, Department of Computer Science, University of Helsinki, Helsinki, Finland*

Abstract

The suffix tree — the compacted trie of all the suffixes of a string — is the most important and widely-used data structure in string processing. We consider a natural combinatorial question about suffix trees: for a string S of length n, how many nodes $\nu_S(d)$ can there be at (string) depth d in its suffix tree? We prove $\nu(n,d) = \max_{S\in\Sigma^n} \nu_S(d)$ is $O((n/d)\log(n/d))$, and show that this bound is asymptotically tight, describing strings for which $\nu_S(d)$ is $\Omega((n/d)\log(n/d))$.

1. Introduction

The *suffix tree*, T_S, of a string S of n symbols is a compacted trie containing all the suffixes of S. Since its discovery by Weiner 44 years ago [8] — as an optimal solution to the longest common substring problem — the suffix tree has emerged as perhaps the most important abstraction in string processing [1], and now has dozens of applications, most notably in bioinformatics [7].

Consequently, combinatorial properties of suffix trees are of great interest, and have been exploited in various ways to obtain faster construction algorithms, succinct representations, and efficient pattern matching and discovery algorithms.

Our focus in this article is on a natural combinatorial question about suffix trees: how many nodes $\nu_S(d)$ can there be at (string) depth d in the suffix tree of a string S? We prove that $\nu(n,d) = \max_{S\in\Sigma^n} \nu_S(d)$ is $O((n/d)\log(n/d))$, and show that this bound is asymptotically tight, describing strings for which $\nu_S(d)$ is $\Omega((n/d)\log(n/d))$.

This article is an extension of an earlier paper where a weaker upper bound of $O((n/d)\log n)$ was shown. The stronger upper bound relies on a new result on

*Corresponding author
Email addresses: g.badkobeh@warwick.ac.uk (Golnaz Badkobeh), gawry@cs.uni.wroc.pl (Paweł Gawrychowski), juha.karkkainen@cs.helsinki.fi (Juha Kärkkäinen), puglisi@cs.helsinki.fi (Simon J. Puglisi), bella.zhukova@helsinki.fi (Bella Zhukova)

another interesting combinatorial quantity on strings: the sum of irreducible lcp values in the LCP array [5, 4]. Specifically, we show that the sum of irreducible values greater than or equal to d is $O(n \log(n/d))$.

In the following section we lay down notation and formally define basic concepts. Section 3 and Section 4 deal with the upper bound and lower bound in turn, and we close with a discussion of the results.

2. Preliminaries

Throughout we consider a string $S = S[1..n] = S[1]S[2] \dots S[n]$ of n symbols drawn from an ordered alphabet Σ of size σ. For $i = 1, \dots, n$ we write $S[i..n]$ to denote the *suffix* of S of length $n-i+1$, that is $S[i..n] = S[i]S[i+1] \cdots S[n]$. For convenience we will frequently refer to suffix $S[i..n]$ simply as "suffix i".

The suffix tree of S is a compact trie representing all the suffixes of S. Every suffix tree node either represents a suffix or is a branching node. Each branching node represents a string that occurs at least twice in S and has at least two distinct symbols following those occurrences. The string depth — or simply depth — of a node is the length of the string it represents. Figs. 1 and 2 show examples of suffix trees.

The *suffix array* of S, denoted SA, is an array $\mathrm{SA}[1..n]$ which contains a permutation of the integers $1..n$ such that $S[\mathrm{SA}[1]..n] < S[\mathrm{SA}[2]..n] < \cdots < S[\mathrm{SA}[n]..n]$. In other words, $\mathrm{SA}[j] = i$ iff $S[i..n]$ is the j^{th} suffix of S in ascending lexicographical order. We use SA^{-1} to denote the inverse permutation. For convenience, we also define $\mathrm{SA}[0] = n+1$ to represent the empty suffix.

The *lcp array* $\mathrm{LCP} = \mathrm{LCP}[1..n]$ is an array defined by S and SA. Let $\mathrm{lcp}(i, j)$ denote the length of the longest common prefix of suffixes i and j. For every $j \in \{1..n\}$,

$$\mathrm{LCP}[j] = \mathrm{lcp}(\mathrm{SA}[j-1], \mathrm{SA}[j]),$$

that is, LCP contains the length of the longest common prefix for each pair of lexicographically adjacent suffixes.

The *permuted lcp array* — $\mathrm{PLCP}[1..n]$ — has the same contents as LCP but in a different order. Specifically, for every $j \in \{1..n\}$,

$$\mathrm{PLCP}[\mathrm{SA}[j]] = \mathrm{LCP}[j]. \tag{1}$$

Then $\mathrm{PLCP}[i] = \mathrm{lcp}(i, \phi(i))$ when we define $\phi(i) = \mathrm{SA}[\mathrm{SA}^{-1}[i] - 1]$.

A binary de Bruijn sequence of order k, denoted by β_k, is a binary word of length $2^k + k - 1$ where each of the 2^k words of length k over the binary alphabet appears as a factor exactly once. As an example, $\beta_4 = aaaabaabbababbbbaaa$ is a de Bruijn sequence of order 4, see Fig. 1.

A positive integer p is a period of a string w, $|w| \geq p$, if there exists a string x of length p such that w is a prefix of x^ω (an infinite repetition of x).

Lemma 1 (Weak periodicity [3]). *If p and q are periods of a string w, $|w| \geq p + q$, then $\gcd(p, q)$ (greatest common divisor) is a period of w too.*

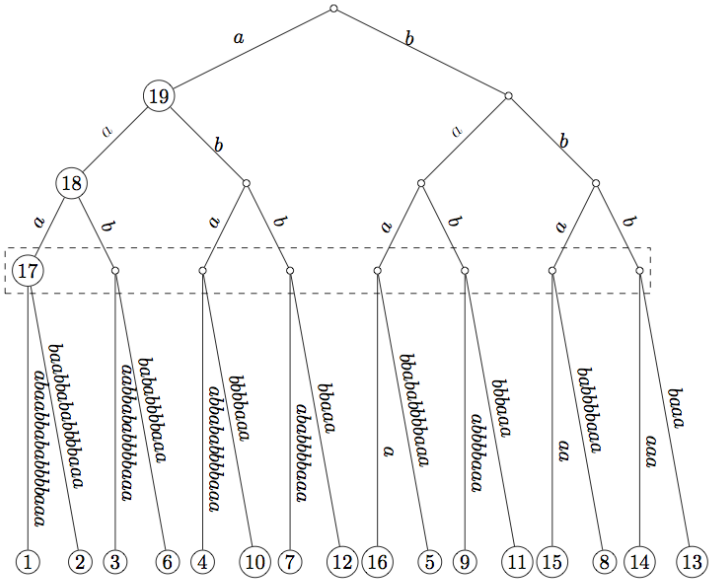

Figure 1: The suffix tree of string $\beta_4 = aaaabaabbbabablbbbaaa$, the binary de Bruijn sequence of order 4. The dashed rectangle contains internal nodes at depth 3.

3. Upper Bound

We are interested in the quantity $\nu(n, d)$, which is the maximum number of branching nodes at depth d over any string of length n. By depth we mean the string depth, the length of the string represented by the node.

A trivial upper bound on $\nu(n, d)$ — relevant for shallow depths — is $\nu(n, d) \leq \sigma^d$ for strings over an alphabet of size σ. Another easy upper bound is $\nu(n, d) \leq (n - d)/2$, since there are $n - d$ suffixes longer than d and each branching node at depth d must represent a prefix of at least two such suffixes. In particular, $\nu(2^k + k - 1, k - 1) = 2^{k-1}$ since the upper bound is matched by a binary de Bruijn sequence of order k, as shown in Fig. 1.

Based on the above, $\nu(n, d)$ increases with d up to depth $d \approx \log_\sigma n$ and then starts to go down. The main result of this section is a much tighter upper bound for larger d showing a quick decrease after depth $\log n$.

Our upper bound proof makes use of the concept of *irreducible* lcp values, first defined in [5]. We say that $\mathrm{PLCP}[i] = \mathrm{lcp}(i, \phi(i))$ is *reducible* if $S[i-1] = S[\phi(i)-1]$ and *irreducible* otherwise. In particular, it is irreducible if $i = 1$ or $\phi(i) = 1$. Reducible values are easy to compute via the next lemma.

Lemma 2 ([5]). *If* $\mathrm{PLCP}[i]$ *is reducible, then* $\mathrm{PLCP}[i] = \mathrm{PLCP}[i - 1] - 1$.

Our proof relies on an upper bound on the sum of irreducible lcp values. The following result was used in an earlier version of this paper [2] for deriving a weaker upper bound $O((n/d) \log n)$.

Lemma 3 ([5, 4]). *The sum of all irreducible lcp values is* $\leq n \log n$.

For a tighter upper bound on the suffix tree breadth, we need a tighter upper bound on *large* irreducible lcp values. First we need the following result.

Lemma 4. *Let* u *and* w *be nonempty strings and* $a \neq b$ *two distinct characters. Then at least one of* wa *and* wb *occurs fewer than* $2|u|/|wa|$ *times in* u.

Proof. Assume to the contrary that both wa and wb occur at least $2|u|/|wa|$ times in u. Then there must be two occurrences of wa starting $p < |wa|/2$ positions apart, and thus p is a period of wa. Similarly, wb must have a period $q < |wb|/2$. Then $p, q \leq |w|/2$, and since p and q are both periods of w, so is $r = \gcd(p, q)$. Let x, y and z be prefixes of w of length p, q and r, respectively. Since r divides both p and q, we must have $x^\omega = z^\omega = y^\omega$. Thus both wa and wb are prefixes of z^ω, but this is not possible, resulting a contradiction. □

Now we are ready for an improved bound on large irreducible lcp values.

Lemma 5. *The sum of irreducible lcp values greater than or equal to* d *is less than* $12n + 4n \log(n/d)$.

Proof. We will follow the proof of the $O(n \log n)$ bound in [5] with a few modifications.

Let $\ell = \text{PLCP}[i] = lcp(i, j) \geq d$ (i.e., $j = \phi(i)$) be an irreducible lcp value, i.e., $S[i-1] \neq S[j-1]$, $S[i..i+\ell-1] = S[j..j+\ell-1]$ and $S[i+\ell] \neq S[j+\ell]$. In [5], the cost ℓ was distributed over the matching pairs of characters $S[i+k] = S[j+k]$, $k \in \{0..\ell-1\}$. Here we distribute the cost over the pairs $S[i+k] = S[j+k]$, $k \in \{\lfloor d/2 \rfloor..\ell-1\}$, assigning a cost of at most two to each pair.

Consider the suffix tree of the reverse of S, and let v_{i+k} and v_{j+k} be the leaves corresponding to the prefixes $S[1..i+k]$ and $S[1..j+k]$. The nearest common ancestor u of v_{i+k} and v_{j+k} represents the reverse of $S[i..i+k]$ (because $S[i-1] \neq S[j-1]$). If v_{i+k} is in a smaller subtree of u than v_{j+k}, the cost of the pair $S[i+k] = S[j+k]$ is assigned to v_{i+k}, otherwise to v_{j+k}.

Now we show that each leaf v carries a cost of less than $12 + 4\log(n/d)$. Whenever v is assigned a cost, this is associated with an ancestor u of v and another leaf w under u. We call u a costly ancestor of v and w a costly cousin of v. We will show that (a) each leaf v has less than $3 + \log(n/d)$ costly ancestors, and that (b) for each costly ancestor, there are at most two costly cousins.

To show (a), we use the "smaller half trick". Consider the path from v to the root. At each costly ancestor u, the size of the subtree at least doubles with the addition of the subtree containing w. Since the highest costly ancestor is at depth $\geq \lfloor d/2 \rfloor$, its subtree containing v must be smaller than $2n/(d/2) = 4n/d$ by Lemma 4. Thus there are less than $1 + \log(4n/d) = 3 + \log(n/d)$ costly ancestors.

Finally, to show (b), let v be leaf, u a costly ancestor of v and w a corresponding costly cousin representing the reverse of the strings $S[1..i+k]$, $S[i..i+k]$ and $S[1..j+k]$, respectively. Then i and j are adjacent in the suffix array. Since i can be adjacent to only two values, there can be at most two such costly cousins for each costly ancestor. □

Now we are ready for the main result.

Theorem 6. *The number of branching nodes at depth d in the suffix tree for a string of length n is at most $R_{\geq d}/d$, where $R_{\geq d}$ is the sum of irreducible lcp values greater than or equal to d, and more specifically, it is at most*

$$\min\{(n/d)\log n, 12n/d + 4(n/d)\log(n/d)\} \ .$$

PROOF. Let S be a string with $\nu(n,d)$ branching nodes at depth d in the suffix tree of S. Every such branching node corresponds to one or more values d in the lcp array, each of which in turn corresponds to a position in the PLCP array with value d. In other words, the number of d's in the PLCP array of S is an upper bound on $\nu_S(d)$. Let $i_1, \ldots, i_r$ be the positions of irreducible values in the PCLP array in ascending order, and let $i_{r+1} = n+1$. Since $i_1 = 1$, the intervals PLCP$[i_j..i_{j+1}-1]$, $j = 1..n$, form a partitioning of the PLCP array. Due to Lemma 2, for every $j = 1..n$, PLCP$[i_j..i_{j+1}-1]$ contains at most one d and only if PLCP$[i_j] \geq d$. Therefore, each occurrence of d can be mapped to a unique irreducible lcp value $\geq d$. Thus $d\nu(n,d) \leq R_{\geq d}$.

The more specific bound follows by inserting Lemmas 3 and 5. □

4. Lower Bound

This section is devoted to proving the following result.

Theorem 7. *For any positive integers $j \geq 1$ and $k \geq 3$, there exists a string of length $n = j(2^k + k - 1)$ such that its suffix tree has $\geq \frac{1}{2}\left(\frac{n}{d} - 1\right)\log\left(\frac{n}{d} - 1\right)$ branching nodes at depth $d = j(k-1)$.*

PROOF. Our proof is based on a construction of the following string, $W_{j,k}$. Let β_k be a binary de Bruijn sequence of order k. Clearly, the suffix tree of β_k is full up to depth $k-1$, and has 2^{k-1} nodes at depth $k-1$. Now, let $W_{j,k} = w_j(\beta_k)$ where the morphism w_j is the following

$$\begin{cases} w_j(a) = 0^j \\ w_j(b) = 10^{j-1} \end{cases}$$

It is clear that $|W_{j,k}| = n = j(2^k + k - 1)$. Let $m = \nu_{W_{j,k}}(j(k-1))$ denote the number of branching nodes of the suffix tree of string $W_{j,k}$ at depth $d = j(k-1)$. We claim that $m \geq 2^{k-1}$. If both ya and yb occur in β_k for some string y, then both $x0$ and $x1$ occur in $W_{i,j}$ for $x = w_j(y)$. Thus every branching node representing y in the suffix tree of β_k is uniquely mapped to a branching node representing $x = w_j(y)$ in the suffix tree of $W_{i,j}$. Since the suffix tree of β_k has 2^{k-1} branching nodes at depth $k-1$, the claim $m \geq 2^{k-1}$ follows.

What remains is to show the steps for the calculation of the lower bound. Since $m \geq 2^{k-1}$ and $d = j(k-1) = \frac{n(k-1)}{(2^k+k-1)}$, we have $\frac{n}{d} = \frac{2^k}{k-1} + 1 \leq \frac{2m}{k-1} + 1$, which implies

$$m \geq \frac{1}{2}\left(\frac{n}{d} - 1\right)(k-1) \geq \frac{1}{2}\left(\frac{n}{d} - 1\right)\log\left(\frac{2^k}{k-1}\right) = \frac{1}{2}\left(\frac{n}{d} - 1\right)\log\left(\frac{n}{d} - 1\right) \ .$$

□

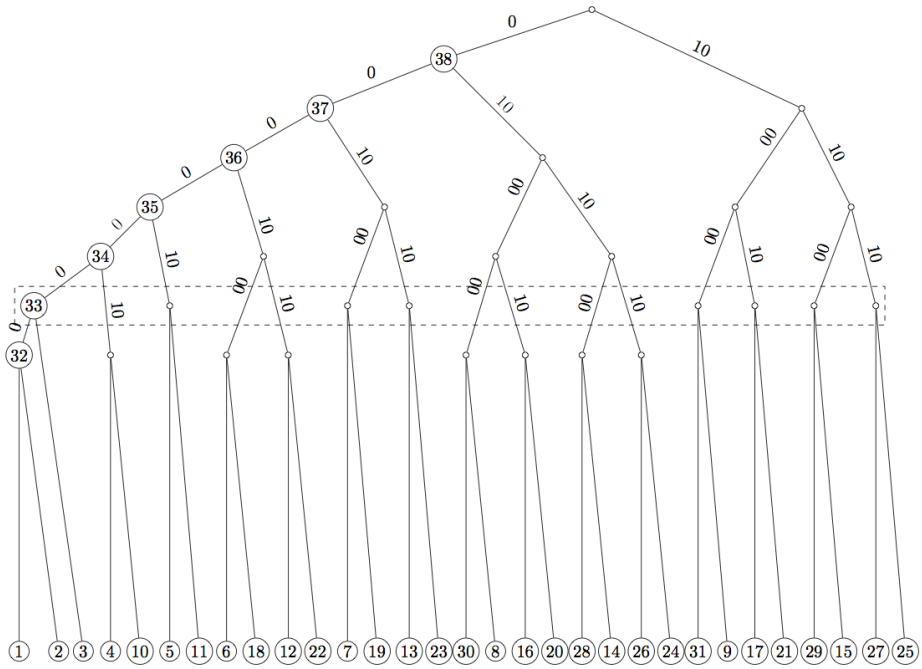

Figure 2: The suffix tree of string $W_{2,4} = 00000000100000101000100010101010000000$. The dashed rectangle contains internal nodes at depth 6.

5. Discussion

Notice that the lower bound construction implies $d = \Omega(\log n)$; thus it does not contradict the upper bounds for small d discussed in Section 3.

Essentially the same bounds hold for all variants and generalizations. We have counted only branching nodes but including leaves (and unary nodes representing suffixes) too would not change much as there can be only one leaf (or unary node) at each depth. Similarly, adding a unique terminator symbol to the end of the string adds at most one node per depth. Considering a suffix tree of multiple strings (containing all suffixes of all strings) could add more leaves to a depth but no more than n/d leaves at a depth d; thus the asymptotic results do not change. Another variant considers the string to be cyclic — replacing suffixes with rotations — and even suffix trees for collections of cyclic strings have been considered [4, 6]. We believe that all the results hold in this case too: a key result for the upper bound, Lemma 3, was explicitly proved for collections of cyclic strings [4], and for the lower bound, de Bruijn sequences are naturally defined as cyclic strings. Finally, notice that Theorems 6 and 7 hold for any alphabet size.

References

[1] Apostolico, A., Crochemore, M., Farach-Colton, M., Galil, Z., Muthukrishnan, S.: 40 years of suffix trees. Commun. ACM 59(4), 66–73 (2016)

[2] Badkobeh, G., Kärkkäinen, J., Puglisi, S.J., Zhukova, B.: On suffix tree breadth. In: String Processing and Information Retrieval - 24th International Symposium, SPIRE 2017, Palermo, Italy, September 26-29, 2017, Proceedings. pp. 68–73 (2017), https://doi.org/10.1007/978-3-319-67428-5_6

[3] Fine, N.J., Wilf, H.S.: Uniqueness theorems for periodic functions. Proc. Amer. Math. Soc. 16(1), 109–114 (1965)

[4] Kärkkäinen, J., Kempa, D., Piatkowski, M.: Tighter bounds for the sum of irreducible LCP values. Theor. Comput. Sci. 656, 265–278 (2016), https://doi.org/10.1016/j.tcs.2015.12.009

[5] Kärkkäinen, J., Manzini, G., Puglisi, S.J.: Permuted longest-common-prefix array. In: Combinatorial Pattern Matching, 20th Annual Symposium, CPM 2009. Lecture Notes in Computer Science, vol. 5577, pp. 181–192. Springer (2009), https://doi.org/10.1007/978-3-642-02441-2_17

[6] Kärkkäinen, J., Piatkowski, M., Puglisi, S.J.: String inference from longest-common-prefix array. In: 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017. pp. 62:1–62:14 (2017), https://doi.org/10.4230/LIPIcs.ICALP.2017.62

[7] Mäkinen, V., Belazzougui, D., Cunial, F., Tomescu, A.I.: Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing. Cambridge University Press (2015), http://www.genome-scale.info/

[8] Weiner, P.: Linear pattern matching algorithms. In: 14th Annual Symposium on Switching and Automata Theory. pp. 1–11. IEEE Computer Society (1973)