Theoretical Computer Science 638 (2016) 98–107

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Longest common extensions in trees[☆]

Philip Bille[1], Paweł Gawrychowski[2], Inge Li Gørtz, Gad M. Landau[3], Oren Weimann[*,4]

A R T I C L E   I N F O

Article history:
Received 31 March 2015
Received in revised form 30 June 2015
Accepted 9 August 2015
Available online 12 August 2015

Keywords:
Longest common prefix
Suffix tree of a tree
Pattern matching in trees

A B S T R A C T

The longest common extension (LCE) of two indices in a string is the length of the longest identical substrings starting at these two indices. The LCE problem asks to preprocess a string into a compact data structure that supports fast LCE queries.

In this paper we generalize the LCE problem to trees and suggest a few applications of LCE in trees to tries and XML databases. Given a labeled and rooted tree T of size n, the goal is to preprocess T into a compact data structure that support the following LCE queries between subpaths and subtrees in T. Let v_1, v_2, w_1, and w_2 be nodes of T such that w_1 and w_2 are descendants of v_1 and v_2 respectively.

- $\mathrm{LCE}_{PP}(v_1, w_1, v_2, w_2)$: (path–path LCE) return the longest common prefix of the paths $v_1 \leadsto w_1$ and $v_2 \leadsto w_2$.
- $\mathrm{LCE}_{PT}(v_1, w_1, v_2)$: (path–tree LCE) return maximal path–path LCE of the path $v_1 \leadsto w_1$ and any path from v_2 to a descendant leaf.
- $\mathrm{LCE}_{TT}(v_1, v_2)$: (tree–tree LCE) return a maximal path–path LCE of any pair of paths from v_1 and v_2 to descendant leaves.

We present the first non-trivial bounds for supporting these queries. For LCE_{PP} queries, we present a linear-space solution with $O(\log^* n)$ query time. For LCE_{PT} queries, we present a linear-space solution with $O((\log\log n)^2)$ query time, and complement this with a lower bound showing that any path–tree LCE structure of size $O(n \operatorname{polylog}(n))$ must necessarily use $\Omega(\log\log n)$ time to answer queries. For LCE_{TT} queries, we present a time-space trade-off, that given any parameter τ, $1 \le \tau \le n$, leads to an $O(n\tau)$ space and $O(n/\tau)$ query-time solution (all of these bounds hold on a standard unit-cost RAM model with logarithmic word size). This is complemented with a reduction from the set intersection problem implying that a fast linear space solution is not likely to exist.

☆ A preliminary version of this paper appeared in the Proceedings of the 26th Annual symposium on Combinatorial Pattern Matching, 2015.
* Corresponding author.
E-mail addresses: phbi@dtu.dk (P. Bille), gawry@mimuw.edu.pl (P. Gawrychowski), inge@dtu.dk (I.L. Gørtz), oren@cs.haifa.ac.il (O. Weimann).
1 Partially supported by the Danish Research Council (DFF – 4005-00267, DFF – 1323-00178) and the Danish National Advanced Technology Foundation.
2 Work done while the author held a post-doctoral position at the Warsaw Center of Mathematics and Computer Science.
3 Partially supported by the Israel Science Foundation grant 571/14, the United States–Israel Binational Science Foundation grant 2014028, and DFG.
4 Partially supported by the Israel Science Foundation grant 794/13.

http://dx.doi.org/10.1016/j.tcs.2015.08.009
0304-3975/© 2015 Elsevier B.V. All rights reserved.

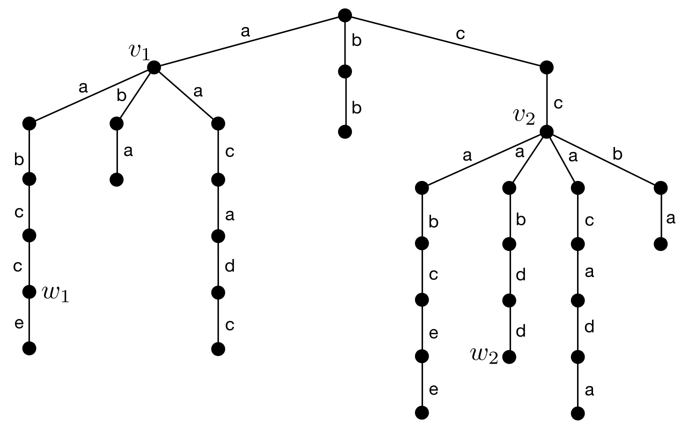

Fig. 1. LCE in trees. LCE$_{PP}(v_1, w_1, v_2, w_2)$ is the path "ab", LCE$_{PT}(v_1, w_1, v_2)$ is the path "abc", and LCE$_{TT}(v_1, v_2)$ is the path "acad".

1. Introduction

Given a string S, the *longest common extension* (LCE) of two indices is the length of the longest identical substring starting at these indices. The *longest common extension problem* (LCE problem) is to preprocess S into a compact data structure supporting fast LCE queries. The LCE problem is a well-studied basic primitive [12,26,13,11,18] with a wide range of applications in problems such as approximate string matching, finding exact and approximate tandem repeats, and finding palindromes [5,16,30,32,24,29,28,31]. The classic textbook solution to the LCE problem on strings combines a suffix tree with a nearest common ancestor (NCA) data structure leading to a linear space and constant query-time solution [23].

In this paper we study generalizations of the LCE problem to trees. The goal is to preprocess an edge-labeled, rooted tree T to support the various LCE queries between paths in T. Here a path starts at a node v and ends at a descendant of v, and the LCEs are on the strings obtained by concatenating the characters on the edges of the path from top to bottom (each edge contains a single character). We consider path–path LCE queries between two specified paths in T, path–tree LCE queries defined as the maximal path–path LCE of a given path and *any* path starting at a given node, and tree–tree LCE queries defined as the maximal path–path LCE between *any* pair of paths starting from two given nodes. We next define these problems formally.

Tree LCE problems Let T be an edge-labeled, rooted tree with n nodes. We denote the subtree rooted at a node v by $T(v)$, and given nodes v and w such that w is in $T(v)$ the path going down from v to w is denoted $v \rightsquigarrow w$. A *path prefix* of $v \rightsquigarrow w$ is any subpath $v \rightsquigarrow u$ such that u is on the path $v \rightsquigarrow w$. Two paths $v_1 \rightsquigarrow w_1$ and $v_2 \rightsquigarrow w_2$ *match* if concatenating the labels of all edges in the paths gives the same string. Given nodes v_1, w_1 such that $w_1 \in T(v_1)$ and nodes v_2, w_2 such that $w_2 \in T(v_2)$ define the following queries:

- LCE$_{PP}(v_1, w_1, v_2, w_2)$: (path–path LCE) return the longest common matching prefix of the paths $v_1 \rightsquigarrow w_1$ and $v_2 \rightsquigarrow w_2$.
- LCE$_{PT}(v_1, w_1, v_2)$: (path–tree LCE) return the maximal path–path LCE of the path $v_1 \rightsquigarrow w_1$ and any path from v_2 to a descendant leaf.
- LCE$_{TT}(v_1, v_2)$: (tree–tree LCE) return a maximal path–path LCE of any pair of paths from v_1 and v_2 to descendant leaves.

The queries are illustrated in Fig. 1. We assume that the output of the queries is reported compactly as the endpoint(s) of the LCE. This allows us to report the shared path in constant time. Furthermore, we will assume w.l.o.g. that for each node v in T, all the edge-labels to children of v are distinct. If this is not the case, then we can merge all identical edges of a node to its children in linear time, without affecting the result of all the above LCE queries.

We note that the direction of the paths in T is important for the LCE queries. In the above LCE queries, the paths start from a node and go downwards. If we instead consider paths from a node going upwards towards the root of T, the problem is easier and can be solved in linear space and constant query-time by combining Breslauer's suffix tree of a tree [14] with a nearest common ancestor (NCA) data structure [25].

Our results First consider the LCE$_{PP}$ and LCE$_{PT}$ problems. To answer an LCE$_{PP}(v_1, w_1, v_2, w_2)$ query, a straightforward solution is to traverse both paths in parallel-top down. Similarly, to answer an LCE$_{PT}(v_1, w_1, v_2)$ query we can traverse $v_1 \rightsquigarrow w_1$ top-down while traversing the matching path from v_2 (recall that all edges to a child are distinct and hence the longest matching path is unique). This approach leads to a linear-space solution with $O(h)$ query-time to both problems, where h is the height of T. Note that for worst-case trees we have that $h = \Omega(n)$.