Journal of Discrete Algorithms 25 (2014) 42–50

Contents lists available at SciVerse ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Time–space trade-offs for longest common extensions ☆

Philip Bille [a], Inge Li Gørtz [a], Benjamin Sach [b], Hjalte Wedel Vildhøj [a,*]

[a] Technical University of Denmark, DTU Compute, Denmark
[b] University of Warwick, Department of Computer Science, United Kingdom

ARTICLE INFO

Article history:
Available online 6 June 2013

Keywords:
Longest common extension
Approximate string matching
Palindrome
Tandem repeat
Time–space trade-off

ABSTRACT

We revisit the longest common extension (LCE) problem, that is, preprocess a string T into a compact data structure that supports fast LCE queries. An LCE query takes a pair (i, j) of indices in T and returns the length of the longest common prefix of the suffixes of T starting at positions i and j. We study the time–space trade-offs for the problem, that is, the space used for the data structure vs. the worst-case time for answering an LCE query. Let n be the length of T. Given a parameter τ, $1 \leqslant \tau \leqslant n$, we show how to achieve either $O(n/\sqrt{\tau})$ space and $O(\tau)$ query time, or $O(n/\tau)$ space and $O(\tau \log(|\text{LCE}(i, j)|/\tau))$ query time, where $|\text{LCE}(i, j)|$ denotes the length of the LCE returned by the query. These bounds provide the first smooth trade-offs for the LCE problem and almost match the previously known bounds at the extremes when $\tau = 1$ or $\tau = n$. We apply the result to obtain improved bounds for several applications where the LCE problem is the computational bottleneck, including approximate string matching and computing palindromes. We also present an efficient technique to reduce LCE queries on two strings to one string. Finally, we give a lower bound on the time–space product for LCE data structures in the non-uniform cell probe model showing that our second trade-off is nearly optimal.

1. Introduction

Given a string T, the *longest common extension* of suffix i and j, denoted $\text{LCE}(i, j)$, is the length of the longest common prefix of the suffixes of T starting at position i and j. The *longest common extension problem* (LCE problem) is to preprocess T into a compact data structure supporting fast longest common extension queries.

The LCE problem is a basic primitive that appears as a subproblem in a wide range of string matching problems such as approximate string matching and its variations [3,8,20,22,28], computing exact or approximate tandem repeats [13,21,24], and computing palindromes. In many of the applications, the LCE problem is the computational bottleneck.

In this paper we study the time–space trade-offs for the LCE problem, that is, the space used by the preprocessed data structure vs. the worst-case time used by LCE queries. We assume that the input string is given in read-only memory and is not counted in the space complexity. There are essentially only two time–space trade-offs known: At one extreme we can store a suffix tree combined with an efficient nearest common ancestor (NCA) data structure [14] (other combinations of $O(n)$ space data structures for the string can also be used to achieve this bound, e.g. [10]). This solution uses $O(n)$ space and supports LCE queries in $O(1)$ time. At the other extreme we do not store any data structure and instead answer queries simply by comparing characters from left-to-right in T. This solution uses $O(1)$ space and answers an $\text{LCE}(i, j)$ query in $O(|\text{LCE}(i, j)|) = O(n)$ time. This approach was recently shown to be very practical [15].

☆ This work was partly supported by EPSRC. An extended abstract of this paper appeared in the proceedings of the 23rd Annual Symposium on Combinatorial Pattern Matching.

* Corresponding author.
E-mail addresses: phbi@dtu.dk (P. Bille), inge@dtu.dk (I.L. Gørtz), sach@dcs.warwick.ac.uk (B. Sach), hwv@hwv.dk (H.W. Vildhøj).

1570-8667/$ – see front matter
http://dx.doi.org/10.1016/j.jda.2013.06.003

1.1. Our results

We show the following main result for the longest common extension problem.

Theorem 1. *For a string T of length n and any parameter τ, $1 \leqslant \tau \leqslant n$, T can be preprocessed into a data structure supporting* $\mathrm{LCE}(i, j)$ *queries on T. This can be done such that the data structure*

(i) *uses $O(\frac{n}{\sqrt{\tau}})$ space and supports queries in $O(\tau)$ time. The preprocessing of T can be done in $O(\frac{n^2}{\sqrt{\tau}})$ time and $O(\frac{n}{\sqrt{\tau}})$ space;*

(ii) *uses $O(\frac{n}{\tau})$ space and supports queries in $O(\tau \log(\frac{|\mathrm{LCE}(i,j)|}{\tau}))$ time. The preprocessing of T can be done in $O(n)$ time and $O(\frac{n}{\tau})$ space. The solution is randomised (Monte Carlo); with high probability, all queries are answered correctly;*

(iii) *uses $O(\frac{n}{\tau})$ space and supports queries in $O(\tau \log(\frac{|\mathrm{LCE}(i,j)|}{\tau}))$ time. The preprocessing of T can be done in $O(n \log n)$ time and $O(n)$ space. The solution is randomised (Las Vegas); the preprocessing time bound is achieved with high probability.*

Unless otherwise stated, the bounds in the theorem are worst-case, and with high probability means with probability at least $1 - 1/n^c$ for any constant c.

Our results provide a smooth time–space trade-off that allows several new and non-trivial bounds. For instance, with $\tau = \sqrt{n}$ Theorem 1(i) gives a solution using $O(n^{3/4})$ space and $O(\sqrt{n})$ time. If we allow randomisation, we can use Theorem 1(iii) to further reduce the space to $O(\sqrt{n})$ while using query time $O(\sqrt{n}\log(|\mathrm{LCE}(i, j)|/\sqrt{n})) = O(\sqrt{n}\log n)$. Note that at both extremes of the trade-off ($\tau = 1$ or $\tau = n$) we almost match the previously known bounds. In the conference version of this paper [4], we mistakenly claimed the preprocessing space of Theorem 1(iii) to be $O(n/\tau)$ but it is in fact $O(n)$. It is possible to obtain $O(n/\tau)$ preprocessing space by using $O(n \log n + n\tau)$ preprocessing time. For most applications, including those mentioned in this paper, this issue has no implications, since the time to perform the LCE queries typically dominates the preprocessing time.

Furthermore, we also consider LCE queries between two strings, i.e. the pair of indices to an LCE query is from different strings. We present a general result that reduces the query on two strings to a single one of them. When one of the strings is significantly smaller than the other, we can combine this reduction with Theorem 1 to obtain even better time–space trade-offs.

Finally, we give a reduction from *range minimum queries* that shows that any data structure using $O(n/\tau)$ bits space in addition to the string T must use at least $\Omega(\tau)$ time to answer an LCE query. Hence, the time–space trade-offs of Theorem 1(ii) and Theorem 1(iii) are almost optimal.

1.2. Techniques

The high-level idea in Theorem 1 is to combine and balance out the two extreme solutions for the LCE problem. For Theorem 1(i) we use *difference covers* to sample a set of suffixes of T of size $O(n/\sqrt{\tau})$. We store a compact trie combined with an NCA data structure for this sample using $O(n/\sqrt{\tau})$ space. To answer an LCE query we compare characters from T until we get a mismatch or reach a pair of sampled suffixes, which we then immediately compute the answer for. By the properties of difference covers we compare at most $O(\tau)$ characters before reaching a pair of sampled suffixes. Similar ideas have previously been used to achieve trade-offs for suffix array and LCP array construction [17,29].

For Theorem 1(ii) and Theorem 1(iii) we show how to use Rabin–Karp fingerprinting [18] instead of difference covers to reduce the space further. We show how to store a sample of $O(n/\tau)$ fingerprints, and how to use it to answer LCE queries using doubling search combined with directly comparing characters. This leads to the output-sensitive $O(\tau \log(|\mathrm{LCE}(i, j)|/\tau))$ query time. We reduce space compared to Theorem 1(i) by computing fingerprints on-the-fly as we need them. Initially, we give a Monte Carlo style randomised data structure (Theorem 1(ii)) that may answer queries incorrectly. However, this solution uses only $O(n)$ preprocessing time and is therefore of independent interest in applications that can tolerate errors. To get the error-free Las Vegas style bound of Theorem 1(iii) we need to verify the fingerprints we compute are collision free; i.e. two fingerprints are equal if and only if the corresponding substrings of T are equal. The main challenge is to do this in only $O(n \log n)$ time. We achieve this by showing how to efficiently verify fingerprints of composed samples which we have already verified, and by developing a search strategy that reduces the fingerprints we need to consider.

Finally, the reduction for LCE on two strings to a single string is based on a simple and compact encoding of the larger string using the smaller string. The encoding could be of independent interest in related problems, where we want to take advantage of different length input strings.

1.3. Applications

With Theorem 1 we immediately obtain new results for problems based on LCE queries. We review some the most important below.