Abstract
The longest common extension problem is the problem of preprocessing a given string of length \( n \) into a data structure that uses \( S(n) \) bits on top of the input and answers in \( T(n) \) time the queries \( \text{LCE}(i,j) \) computing the length of the longest string that occurs at both positions \( i \) and \( j \) in the input. We prove that the trade-off \( S(n)T(n) = \Omega(n \log n) \) holds for any data structure that solves the LCE problem using \( S(n) \) bits of space and \( T(n) \) time for the LCE queries in assumption \( S(n) = \Omega(n) \). This trade-off is tight.

**Keywords:** longest common extension, LCE, LCP, trade-off, lower bounds, cell-probe model

1. Introduction

Data structures for solving the so-called longest common extension (LCE) problem (sometimes referred to as the longest common prefix problem) play the central role in the wide range of string algorithms. In this problem we must preprocess an input string of length \( n \) so that one can answer the queries \( \text{LCE}(i,j) \) computing the length of the longest string that occurs at both positions \( i \) and \( j \) in the input. Since the existing solutions to this problem often, in practice particularly, constitute a bottleneck either in space or in time of the algorithms relying in their core on the LCE queries, much efforts have been made in the past decades to the development of better LCE data structures.

In this paper we prove that the trade-off\(^1\)
\[
S(n)T(n) = \Omega(n \log n)
\]
holds for any data structure that solves the LCE problem using \( S(n) \) bits of space (called additional space) on top of the input and \( T(n) \) time for the LCE queries in assumption that the input string is read-only, each letter occupies a separate memory cell, and \( S(n) = \Omega(n) \) (such space is used in most applications of the LCE problem). For \( S(n) = \Omega(n) \), this new trade-off improves by \( \log n \) factor the trade-off \( S(n)T(n) = \Omega(n) \) established by Bille et al.\(^2\) who used a simple reduction to a lower bound obtained by Brodal et al.\(^3\) for the so-called range minimum queries problem.

Our result is proved in the cell-probe model\(^4\), in which the computation is free and time is counted as the number of cells accessed (probed) by the query algorithm. The algorithm is also allowed to be non-uniform, i.e., we can have different algorithms for different sizes \( n \) of the input. We assume that each letter of the input string is an integer located in a separate memory cell and each cell can store any integer from the set \( \{0, 1, \ldots, n-1\} \).

So, the maximal size of the input alphabet is \( n \); this is a common assumption justified in, e.g., \( \cite{2} \). However, our main theorem poses a weaker restriction: the size of the input alphabet must be at least \( 2^{\lceil S(n)/n \rceil} \). For instance, our trade-off is applicable for constant alphabets if \( S(n) = \Theta(n) \), but to apply the trade-off in the case \( S(n) = \Theta(n \sqrt{\log n}) \), we have to have an alphabet of at least \( 2^{\Theta(n \sqrt{\log n})} \) size.

**Overview of LCE data structures.** The classical solutions for the LCE problem use \( \Theta(n \log n) \) bits of space and \( O(1) \) time for queries (e.g., see \( \cite{3} \)). In \( \cite{1} \) Bille et al. presented a RAM data structure that solves the LCE problem using \( O(\tau) \) time for queries and \( O(\frac{n \log n}{\tau}) \) bits of additional space, where \( \tau \) is a parameter such that \( 1 \leq \tau \leq n \). This result shows that our trade-off is tight and cannot be improved. The construction time of this data structure (in \( O(\frac{n \log n}{\tau}) \) bits of space) is \( O(n^2 + \tau) \), which is unacceptably slow. In \( \cite{14} \) Tanimura et al. proposed a data structure with significantly better \( O(n \tau) \) construction time within the same \( O(\frac{n \log n}{\tau}) \) bits of space.
bits of additional space but with slightly slower query time \(O(\tau \min\{\log \tau, \log \frac{n}{\tau}\})\).

Denote by \(\sigma\) the size of the input alphabet. We believe that our lower bound does not hold if the algorithm can read \(\Omega(\log_\sigma n)\) consecutive letters of the input string in \(O(1)\) time packing them in one \(\Omega(\log n)\)-bit machine word; this model reflects the situation that one can often observe in practice.

The data structure of \([9]\) using this model occupies \(O(\frac{n \log n}{\tau})\) bits of additional space and answers LCE queries in \(O(\log^* n (\log \frac{n}{\tau} + \tau \log^3 / \log_\sigma n))\) time, where \(\tau\) is a parameter such that \(1 \leq \tau \leq n\) (however, this result still does not break our trade-off). The construction time for this structure (in \(O(\frac{n \log n}{\tau})\) bits) is \(O(n (\log^* n + \log \frac{n}{\tau} + \log_\sigma \tau))\).

All mentioned results consider applications in which the input string is treated as read-only. In practice, however, we usually need a data structure that provides fast access to the string and allows us to answer the LCE queries but the space occupied by the string itself can be reorganized. In \([13]\) Prezza described an “in-place” data structure that replaces \(n[\log \sigma]\) bits occupied by the input with a data structure that allows to retrieve any sub-string of length \(m\) of the input in optimal \(O(\frac{m \log n}{\log_\sigma n})\) time and answers the LCE queries in \(O(\log \ell)\) time, where \(\ell\) is the result of the query. For his data structure, Prezza presents only a randomized construction algorithm working in \(O(n \log n)\) expected time and \(O(n \log n)\) bits of space.

In certain applications the exact accuracy of the LCE queries is less important than construction time, query time, and space. For such applications, several Monte Carlo data structures, which answer the LCE queries correctly with high probability (w.h.p.), were developed. Prezza described such in-place data structure that answers the LCE queries w.h.p. in \(O(\log \ell)\) time and has \(O(n)\) expected construction time (also in-place). Bille et al. \([1]\) presented a Monte Carlo version of their data structure for read-only inputs that answers the LCE queries w.h.p. in \(O(\tau)\) time using \(O(\frac{n \log n}{\tau})\) bits of additional space and has \(O(n \log \frac{n}{\tau})\) construction time (within the same space), where \(1 \leq \tau \leq n\).

Recently, several LCE data structures for compressed strings were developed. For a more detailed discussion on this topic, we refer the reader to \([9]\) and references therein.

2. Main Result

Preliminaries. A string \(s\) of length \(n\) over an alphabet \(\Sigma\) is a map \(\{0, 1, \ldots, n-1\} \mapsto \Sigma\), where \(n\) is referred to as the length of \(s\), denoted by \(|s|\). We write \(s[i]\) for the \(i\)th letter of \(s\). A string \(s[0]s[1] \cdots s[j]\) is a prefix of \(s\). For any \(i, j\), the set \(\{k \in \mathbb{Z} : i \leq k \leq j\}\) (possibly empty) is denoted by \([i..j]\).

Theorem. In the non-uniform cell-probe model the trade-off \(S(n)T(n) = \Omega(n \log n)\) holds for any data structure that solves the LCE problem for a read-only string of length \(n\) using \(S(n)\) bits of space and \(T(n)\) time for queries in assumption that each input letter occupies a separate cell, the size of the input alphabet is at least \(2^{\mathcal{O}(S(n)/n)}\), and \(S(n) = \Omega(n)\).

Proof. Without loss of generality, assume that \(T(n) \geq 1\). Suppose, for the sake of contradiciton, that \(S(n)T(n) \neq \Omega(n \log n)\). Then, there is an infinite set \(N\) of positive integers such that \(\lim_{n \in N} \frac{S(n)T(n)}{n \log n} = 0\). Hence, we obtain \(\lim_{n \in N} \frac{S(n)}{n \log n} = 0\). Therefore, there is a positive function \(\varepsilon(n)\) such that \(S(n) = \varepsilon(n)n \log n\) for \(n \in N\) and \(\varepsilon(n)\) tends to 0 as \(n \to +\infty\).

Let us first construct a family \(\mathcal{F}\) of inputs for the subsequent analysis. Define \(\sigma = 2^{\varepsilon(n) \log n}\). The input alphabet is \([1, \sigma]\). Note that \(\sigma \geq 2^n\) for \(n > 1\) and \(\sigma = 2^n(S(n)/n)\) for \(n \in N\). Since \(\log \sigma = o(\log n)\) and, hence, \(\sigma < n\) for sufficiently large \(n\), each letter of the alphabet fits in one memory cell. Observe that, since \(S(n) = \varepsilon(n)n \log n \leq \frac{1}{\varepsilon(n)} \log \sigma\) for \(n \in N\), we are not able to encode the whole string in \(S(n)\) bits and answer the LCE queries without any access to the string.

Denote \(k = \lceil \frac{\log_\sigma n}{\ell} \rceil\). Since \(\log \sigma = o(\log n)\), we have \(k = \Theta(\frac{\log n}{\log \sigma}) = \omega(1)\). So, \(k \geq 1\) for sufficiently large \(n\). Let \(s_1, s_2, \ldots, s_k\) denote all strings of length \(k\) over the alphabet \([1, \sigma]\). The family \(\mathcal{F}\) consists of all strings of the form \(s_1s_2 \cdots s_kt\), where \(t\) is a string of length \(n - ka^k\) over the alphabet \([1, \sigma]\). Since \(\sigma^k k \leq \sqrt{n} \log n\), it is easy to verify that \(\frac{1}{\varepsilon(n)} \leq n - ka^k\) for \(n \geq 2^9\). Hence, we obtain \(|\mathcal{F}| \geq \sigma^k = 2^{\frac{k}{2} n \log \sigma} \geq 2^{8^n}\). For convenience, we assume hereafter that \(\min N > 2^8\).

By the pigeonhole principle, there is a subfamily \(\tilde{\mathcal{F}} \subseteq \mathcal{F}\) such that \(|\tilde{\mathcal{F}}| \geq |\mathcal{F}|/2^S(n)\) and, for
any strings \(s, s' \in \mathcal{I}\), the encodings of \(s\) and \(s'\) in the \(S(n)\) bits of the considered LCE data structure are equal. Since \(S(n) \leq \frac{2}{\epsilon} n \log \sigma\) for \(n \in N\), we obtain \(|\mathcal{I}| \geq |\mathcal{F}|/2^{S(n)} \geq 2^{\frac{1}{2} n \log \sigma - S(n)} \geq 2^{\frac{1}{2} n \log \sigma - \frac{n}{2} \log \sigma}\) for \(n \in N\).

Let us prove that, for any \(s \in \mathcal{I}\), there is a set of \(T_s \subseteq [0..n-1]\) such that \(|T_s| \leq \frac{T(n)n}{n\log n}\) and, for any string \(s' \in \mathcal{I}\), we have \(s = s'\) iff \(s[i] = s'[i]\) for every \(i \in T_s\). Choose \(s \in \mathcal{I}\); \(s = s_1 s_2 \cdots s_r t\) for a string \(t\). Without loss of generality, assume that \(|t|\) is a multiple of \(k\) (the case \(|t|\) mod \(k \neq 0\) is similar). Since \(\{s_1, s_2, \ldots, s_r\}\) is the set of all strings of length \(k\) over the alphabet \([1..\sigma]\), there must exist exactly one sequence \(i_1, i_2, \ldots, i_r\), where \(r = |t|/k\), such that \(t = s_i s_{i+1} \cdots s_{i+r}\) and \(i_j \in [1..\sigma^k]\) for \(j \in [1..r]\). Let \(j \in [1..r]\). By the definition of LCE, the query \(LCE([s_1 s_2 \cdots s_{j-1}], s_1 s_2 \cdots s_{j-1})\) reads at most \(T(n)\) letters of the string \(s\) and returns as an answer a number that is not less than \(|s_1|\). Denote by \(T_j\) the set of all positions \(i \in [0..n-1]\) such that the letter \(s[i]\) was accessed by the query. Define \(T_s = T_1 \cup T_2 \cup \ldots \cup T_r\). It is easy to see that \(|T_s| \leq \frac{T(n)n}{n\log n} \leq \frac{T(n)n}{n\log n}\). If a string \(s' \in \mathcal{I}\) coincides with the string \(s\) on the positions \(T_s\), then, by the definition of \(T_s\), any query \(LCE([s_1 s_2 \cdots s_{j-1}], s_1 s_2 \cdots s_{j-1})\) on the string \(s'\) must return the same result as the corresponding query on the string \(s\), i.e., the algorithm cannot distinguish \(s\) and \(s'\) on such queries. Thus, the numbers computed by these queries are not less than \(|s_1|\). Hence, since the algorithm is assumed to be correct and the string \(s_1 s_2 \cdots s_{j-1}\) is a common prefix of \(s\) and \(s'\), the string \(s'\) must be equal to \(s\).

Clearly, there are at most \(2^n\) subsets \(T_s\) (as there are at most \(2^n\) subsets of \([0..n-1]\)). Thus, by the pigeonhole principle, there exists a subfamily \(\mathcal{T} \subseteq \mathcal{I}\) such that \(|\mathcal{T}| \geq |\mathcal{I}|/2^n\) and \(T_s = T_s'\) whenever \(s, s' \in \mathcal{T}\). Since \(|\mathcal{T}| \geq \frac{2^{k} n \log \sigma}{n \log \sigma}\) for \(n \in N\) and \(\log \sigma \geq 8\), we obtain \(|\mathcal{T}| \geq |\mathcal{I}|/2^n \geq 2^{\frac{1}{2} n \log \sigma - \frac{8}{2} n \log \sigma} \geq 2^{\frac{1}{2} n \log \sigma - \frac{n}{2} \log \sigma}\) for \(n \in N\).

By the choice of \(\mathcal{T}\), the set \(T_s\) is the same for every \(s \in \mathcal{T}\). Denote this set by \(T\). By the definition of \(T\), the size of the family \(\mathcal{T}\) is upper bounded by the number \(\sigma^{|T|} = 2^{|T| n \log \sigma} \leq 2^{T(n)n \log \sigma / k}\). Thus, if \(T(n) < \frac{1}{4} k\) for arbitrarily large numbers \(n \in N\) (recall that \(k = \omega(1)\)), we have \(\sigma^{|T|} < 2^{\frac{1}{2} n \log \sigma} \leq |\mathcal{I}|\), which is a contradiction. So, we obtain \(S(n) T(n) \geq \varepsilon(n) n \log n \cdot \frac{1}{4} k\) for all sufficiently large \(n \in N\). Since \(S(n) = \varepsilon(n) n \log n\) for \(n \in N\) and \(S(n) = \Omega(n)\), the function \(\varepsilon(n)\) can be chosen so that \(\varepsilon(n) n \log n = \Omega(1)\). Hence \(k \varepsilon(n) n \log n = \frac{1}{2} n \log n \cdot \varepsilon(n) n \log n = \Theta(n \log n)\). Therefore, we obtain \(\lim_{n \to N} \inf \frac{S(n) T(n)}{n \log n} \geq \lim_{n \to N} \inf \frac{\varepsilon(n) n \log n \cdot \frac{1}{4} k}{n \log n} = \lim_{n \to N} \inf \frac{\varepsilon(n) n \log n \cdot \frac{1}{4} k}{n \log n} = 0\), which contradicts to the assumption \(\lim_{n \to N} \frac{S(n) T(n)}{n \log n} = 0\).

Remark. The assumption \(S(n) = \Omega(n)\), which plays its role in the last lines of the proof, is crucial for our construction. The simple information theoretic argument by which we obtained the subfamily \(\mathcal{T}\) eliminates any dependency on the \(S(n)\) bits of space so that the query algorithm can “distinguish” the strings of \(\mathcal{I}\) from each other only by probing the input cells. Since each string from the family \(\mathcal{I}\) must contain the common “dictionary” prefix of length \(k \sigma^k\), \(k\) cannot exceed \(n\). The idea of the construction of the sets \(T_s\), the set \(T_s\) is the same for any \(s \in \mathcal{I}\) that is any algorithm using significantly less than \(k\) cell probes (as in the case \(T(n) = o(k)\)) cannot obtain enough information to distinguish all strings of \(\mathcal{I}\). Thus, since \(k = O(\log n)\), the best bound for \(T(n)\) that one can obtain in this way is \(T(n) = \Omega(\log n)\). This is the main reason why it is not immediately clear how to adapt our proof for the case \(S(n) = o(n)\).

\[ \blacksquare \]

3. Open Problems

As it follows from Theorem, the data structure of Bille et al. \[1\] is optimal when additional space is restricted to \(S(n) = \Omega(n)\) bits. Still there is a \(n \log n\) gap between the upper and lower bounds for the problem when \(S(n) = o(n)\). We believe that the approach of \[1\] can be modified to achieve the better \(S(n) T(n) = O(n \log S(n))\) trade-off. We conjecture that this trade-off is optimal.

Conjecture. In the non-uniform cell-probe model the trade-off \(S(n) T(n) = \Omega(n \log S(n))\) holds for any data structure that solves the LCE problem for a read-only string of length \(n\) using \(S(n)\) bits of space and \(T(n)\) time for the LCE queries.

There are several other promising directions for further investigations of the LCE data structures. First, it would be interesting to obtain a version of our result for randomized LCE data structures that answer the LCE queries with high probability.

It is an open problem to provide a tight time and space lower bound on the LCE data structures that use the knowledge of the alphabet size \(\sigma\). Further,
it is not clear how to generalize our result to the practically important case when \( \sigma \) is very small and the algorithm can read \( \Omega(\log_2 n) \) consecutive letters of the input string in \( O(1) \) time packing them in one \( \Omega(\log n) \)-bit machine word.

It seems that still there are many possibilities for improvements of upper bounds for the algorithms that solve the LCE problem. The best currently known algorithm constructing the data structure [1] is unacceptably slow. In [14], there was presented a data structure with significantly faster construction time but with slightly suboptimal query time. So, the development of a new optimal LCE data structure with optimal construction time remains an open problem.

For applications that do not consider the input string as read-only, it is an open problem to develop a data structure that provides access to the string in optimal time as in the data structure of Prezza [13] discussed above and supports the LCE queries in \( o(\log \ell) \) time, where \( \ell \) is the result of the query. It is also interesting to obtain, if possible, any non-trivial lower bounds for this model.

In practice, randomized construction algorithms for LCE data structures usually behave better than deterministic ones. There is a room for improvements in this direction in the algorithms presented in [13], for read-only inputs, and [13], for editable inputs. It is also interesting to consider in both these setting the development of more time or space efficient Monte Carlo LCE data structures, which compute the LCE queries correctly with high probability.

Finally, on the purely theoretical side there are some related open problems in the line of research on general ordered and unordered alphabets. In this classical setting even an LCE data structure with optimal construction time and \( O(1) \) query time still was not described. However, there is a strong evidence [7] that, surprisingly, such data structure exists; see [7, 10].

Acknowledgement. The author wishes to acknowledge anonymous referees for detailed comments that helped to improve the paper.

References

[1] P. Bille, I. L. Gørtz, M. B. T. Knudsen, M. Lewenstein, H. W. Vildhøj, Longest common extensions in sublinear space, in: CPM 2015, vol. 9133 of LNCS, Springer, 2015.
[2] P. Bille, I. L. Gørtz, B. Sach, H. W. Vildhøj, Time-space trade-offs for longest common extensions, Journal of Discrete Algorithms 25 (2014) 42–50.
[3] G. S. Brodal, P. Davoodi, S. S. Rao, On space efficient two dimensional range minimum data structures, in: ESA 2010, vol. 6347 of LNCS, Springer, 2010.
[4] S. Burkhardt, J. Kärkkäinen, Fast lightweight suffix array construction and checking, in: CPM 2003, vol. 2676 of LNCS, Springer, 2003.
[5] J. Fischer, V. Heun, Theoretical and practical improvements on the rmq-problem, with applications to lca and lce, in: CPM 2006, vol. 4009 of LNCS, Springer, 2006.
[6] J. Fischer, T. I, D. Köppl, Deterministic sparse suffix sorting on rewratable texts, in: LATIN, vol. 9644 of LNCS, Springer, 2016.
[7] P. Gawrychowski, T. Kociumaka, W. Rytter, T. Waleń, Faster longest common extension queries in strings over general alphabets, in: CPM 2016, vol. 54 of LIPIcs, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
[8] D. Harel, R. E. Tarjan, Fast algorithms for finding nearest common ancestors, SIAM Journal on Computing 13 (2) (1984) 338–355.
[9] T. I, Longest common extension with recompression, arXiv preprint arXiv:1611.05359.
[10] D. Kosolobov, Computing runs on a general alphabet, Information Processing Letters 116 (3) (2016) 241–244.
[11] P. B. Miltersen, Cell probe complexity — a survey, in: FSTTCS 1999, 1999.
[12] A. Policriti, N. Prezza, Fast longest common extensions in small space, arXiv preprint arXiv:1607.06660.
[13] N. Prezza, In-place longest common extensions, arXiv preprint arXiv:1608.05100.
[14] Y. Tanimura, T. I, H. Bannai, S. Inenaga, S. J. Puglisi, M. Takeda, Deterministic sub-linear space LCE data structures with efficient construction, in: CPM 2016, vol. 54 of LIPIcs, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.