A Faster Longest Common Extension Algorithm on Compressed Strings and its Applications (*Invited talk*)

Shunsuke Inenaga

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

Abstract. In this talk, we introduce our recent data structure for *longest common extension (LCE)* queries on grammar-compressed strings. Our preprocessing input is a *straight-line program (SLP)* of size n describing a string w of length N, which is essentially a CFG in the Chomsky normal form generating only w. We can preprocess the input SLP in $O(n \log \log n \log N \log^* N)$ time so that later, given two variables and two positions in the strings derived by the variables, we can answer the corresponding LCE query in $O(\log N \log^* N)$ time. Our LCE data structure requires $O(z \log N \log^* N)$ words of space, where z is the size of the Lempel-Ziv 77 factorization of w. We also show several applications of our LCE data structure on SLPs.

1 Longest common extension (LCE) problem

The *longest common extension (LCE) problem* is to compute the length of the longest common prefix of two query suffixes of a string. More formally, the problem is defined as follows: Preprocess an input string w so that later, given a query pair (i, j) of positions on w, we can quickly answer the length ℓ of the longest common prefix of $w[i..|w|]$ and $w[j..|w|]$. The LCE problem often appears as important subproblems of various kinds of string processing problems, e.g., computing gapped palindromes [15] and gapped repeats [16], approximate pattern matching [4], computing runs [5], etc.

Let N denote the length of an input string w. It is well known that after preprocessing the string w in $O(N)$ time and with $O(N)$ words of space (or $O(N\omega)$ bits of space, if ω is the machine word size), the LCE of any two query suffixes can be computed in $O(1)$ time, by applying a lowest common ancestor data structure [11,23,6] to the suffix tree of w [24,10]. The $O(N)$-word space usage, however, can be problematic for massively long strings, and hence, a great deal of effort has been put towards developing more space-efficient LCE data structures.

2 LCE problem on grammar-compressed strings

In this research, we consider the LCE problem on *grammar compressed strings* which are represented by *straight-line programs (SLPs)*. An SLP for a string w is a context-free grammar in the Chomsky normal form which derives only w. Let $V = X_1, \ldots, X_n$ be the sequence of n variables of an SLP $\mathcal{S}$ which represents a string w of length N, where X_n is the last variable deriving w. The number n of variables is called the *size* of the SLP $\mathcal{S}$. We assume that V has no redundant variables, i.e., each X_u in V appears at least once in the derivation tree of X_n. On this assumption, $n \leq N$ always holds, and hence, any SLP is asymptotically never larger than the original string. Also, since every internal node of the derivation tree of any SLP has exactly

Shunsuke Inenaga: A Faster Longest Common Extension Algorithm on Compressed Strings and its Applications, pp. 1–4.
Proceedings of PSC 2015, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-05787-2

two children, $\log_2 N \leq n$ holds. Indeed, SLPs are capable of *exponential compression* for some instances, i.e., the sizes of SLPs for highly repetitive strings can be as small as $\Theta(\log N)$.

We consider the LCE problem on SLPs in the context of *compressed string processing (CSP)* [3]: We assume that the string w is stored as an SLP $\mathcal{S}$, and $\mathcal{S}$ is given to us as an input for preprocessing. The task is to build a data structure which: (1) supports efficient LCE queries on any pair of variables of $\mathcal{S}$. Namely, given a query quartet (u, v, i, j), we are to compute the longest common prefix of $val(X_u)[i..|val(X_u)|]$ and $val(X_v)[j..|val(X_v)|]$, where $val(\cdot)$ denotes the string derived by the variable; (2) requires $n^{O(1)}$ space; and (3) can be constructed in $n^{O(1)}$ time. LCE data structures with properties (1)-(3) are of great significance, when the original string w is highly compressible. In particular, when N is as large as $\Theta(2^n)$, such LCE data structures on SLPs achieve exponential space-saving w.r.t. the uncompressed counterparts. Note that *no* algorithms which explicitly decompress the input SLP can achieve (3), since the length N of the original uncompressed string w can be as large as $\Theta(2^n)$.

A folklore LCE algorithm on SLPs is the following: Precompute the length of the decompressed string $val(X_u)$ for every variable X_u in V. This can be done in $O(n)$ total time in a simple bottom-up manner, and all the lengths can be stored with $O(n)$ words of total space (assuming the machine word size ω is at least $\log_2 N$). Then, we can simulate a traversal from the root to each leaf of the derivation tree of each variable X_u in $O(h)$ time, where h is the height of the last variable X_n. Thus, LCE query (u, v, i, j) on an input SLP can be answered in $O(h\ell)$ time, where ℓ is the answer (LCE length) to the query. Note that $\log_2 N \leq h \leq n$ always holds, and that the answer ℓ can be as large as $O(N)$.

Karpinski et al. [14] showed the first non-trivial LCE data structure on SLPs which requires $O(n^3)$ words of space and answers LCE queries of limited form $(u, v, i, 1)$ in $O(n \log n)$ time. Their data structure can be constructed in $O(n^4 \log n)$ time. Miyazaki et al. [19] proposed a data structure which requires $O(n^2)$ words of space and can answer LCE queries of limited form $(u, v, i, 1)$ in $O(n^2)$ time. Their algorithm can be extended to support LCE queries of general form (u, v, i, j) in $O(n^2 h)$ time with the same space bound [12]. Lifshits [17] showed how to construct Miyazaki et al.'s data structure in $O(n^2 h)$ time. I et al. [12] developed an LCE data structure on SLPs which requires $O(n^2)$ words of space, supports LCE queries of general form in $O(h \log N)$ time, and can be constructed in $O(n^2 h)$ time. The common basic idea to all these data structures is to virtually align the leaves of the derivation trees of two variables X_u and X_v with appropriate offsets, and compute maximal subtrees whose leaves correspond to the LCE. Bille et al. [7] proposed a randomized LCE data structure. We omit its details, since here we concentrate on deterministic LCE data structures.

2.1 A new faster LCE data structure on SLPs

In this talk, we introduce our new LCE data structure on SLPs which requires $O(z \log N \log^* N)$ words of space, and supports LCE queries of general form (u, v, i, j) on SLPs in $O(\log N \log^* N)$ time, where z is the size of the Lempel-Ziv 77 factorization [25] of the original string w. Rytter [21] showed that z is a lower bound of the size of *any* SLP representing the string w, i.e., $z \leq n$ always holds. Hence this new LCE data structure is rather small. Also, since $\log^* N$ is smaller than h, our LCE query time is always better than that of the state-of-the-art data structure by

I et al. [12]. We also show that our new LCE data structure can be constructed in $O(n \log \log n \log N \log^* N)$ time from a given SLP of size n.

The mechanism of our new LCE data structure is significantly different from the previous LCE data structures on SLPs. The new algorithm works on the trees induced by the *signature encodings* [2,1] of the strings derived by the variables, rather than on the derivation trees of the variables.

Using our faster LCE data structure, we improve the best known solutions to several important problems on SLPs, e.g. computing all palindromic substrings [18] and computing the Lyndon factorization of the original string [13].

These results are an outcome of a joint work with Takaaki Nishimoto, Tomohiro I, Hideo Bannai, and Masayuki Takeda. A full version of this work is available at [20].

3 Related work

Another line of research for space-efficient LCE data structures is to develop *succinct data structures* which use space close to the information theoretic lower bound. The longest common prefix (LCP) array for string w of length N is an array of length N which stores the lengths of the longest common prefixes of consecutive suffixes of w that are lexicographically sorted. Then, LCE queries on string w reduce to *range minimum queries (RMQs)*. Sadakane [22] proposed an RMQ data structure for an array of length N which occupies $4N + o(N)$ bits of space and answers each query in $O(1)$ time. His data structure can be constructed in $O(N)$ time with $O(N \log N)$ bis of working space. Later, Fischer and Heun showed a smaller RMQ data structure which uses only $2N + o(N)$ bits of space, answers each query in $O(1)$ time, and can be built in $O(N)$ time with $O(N)$ bits of working space. Each of these data structures is an *encoding* of the LCP array of w, namely, the LCP array is not needed for answering queries.

Yet another line of research is to find trade-offs between the space complexity and the LCE query time with a parameter τ with $1 \leq \tau \leq N$. Bille et al. [9] proposed an LCE data structure which requires $O(N/\sqrt{\tau})$ words of space, answers each LCE query in $O(\tau)$ time, and can be built in $O(N^2/\sqrt{\tau})$ time with $O(N/\sqrt{\tau})$ words of working space. Recently, Bille et al. [8] discovered a better trade-off with $O(N/\tau)$ words of space and $O(\tau)$ LCE query time. This data structure can be built in $O(N^{2+\varepsilon})$ time using $O(N/\tau)$ words of working space, where $\varepsilon > 0$ is any constant. Some randomized LCE algorithms were also proposed by these authors.

Note that all the above LCE data structures use space which is proportional to the length N of the uncompressed string w.

References

1. S. Alstrup, G. S. Brodal, and T. Rauhe: *Dynamic pattern matching*, tech. rep., Department of Computer Science, University of Copenhagen, 1998.
2. S. Alstrup, G. S. Brodal, and T. Rauhe: *Pattern matching in dynamic texts*, in SODA 2000, 2000, pp. 819–828.
3. A. Amir, G. Benson, and M. Farach: *Let sleeping files lie: Pattern matching in z-compressed files*. J. Comput. Syst. Sci., 52(2) 1996, pp. 299–307.
4. A. Amir, M. Lewenstein, and E. Porat: *Faster algorithms for string matching with k mismatches*. J. Algorithms, 50(2) 2004, pp. 257–275.
5. H. Bannai, T. I, S. Inenaga, Y. Nakashima, M. Takeda, and K. Tsuruta: *A new characterization of maximal repetitions by Lyndon trees*, in SODA 2015, 2015, pp. 562–571.

6. M. A. Bender and M. Farach-Colton: *The LCA problem revisited*, in LATIN 2000, 2000, pp. 88–94.
7. P. Bille, P. H. Cording, I. L. Gørtz, B. Sach, H. W. Vildhøj, and S. Vind: *Fingerprints in compressed strings*, in WADS 2013, 2013, pp. 146–157.
8. P. Bille, I. L. Gørtz, M. B. T. Knudsen, M. Lewenstein, and H. W. Vildhøj: *Longest common extensions in sublinear space*, in CPM 2015, 2015, pp. 65–76.
9. P. Bille, I. L. Gørtz, B. Sach, and H. W. Vildhøj: *Time-space trade-offs for longest common extensions*. J. Discrete Algorithms, 25 2014, pp. 42–50.
10. M. Farach-Colton, P. Ferragina, and S. Muthukrishnan: *On the sorting-complexity of suffix tree construction*. J. ACM, 47(6) 2000, pp. 987–1011.
11. D. Harel and R. E. Tarjan: *Fast algorithms for finding nearest common ancestors*. SIAM J. Comput., 13(2) 1984, pp. 338–355.
12. T. I, W. Matsubara, K. Shimohira, S. Inenaga, H. Bannai, M. Takeda, K. Narisawa, and A. Shinohara: *Detecting regularities on grammar-compressed strings*. Inf. Comput., 240 2015, pp. 74–89.
13. T. I, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda: *Faster Lyndon factorization algorithms for SLP and LZ78 compressed text*, in SPIRE 2013, 2013, pp. 174–185.
14. M. Karpinski, W. Rytter, and A. Shinohara: *An efficient pattern-matching algorithm for strings with short descriptions*. Nordic Journal of Computing, 4 1997, pp. 172–186.
15. R. Kolpakov and G. Kucherov: *Searching for gapped palindromes*. Theor. Comput. Sci., 410(51) 2009, pp. 5365–5373.
16. R. Kolpakov, M. Podolskiy, M. Posypkin, and N. Khrapov: *Searching of gapped repeats and subrepetitions in a word*, in CPM 2014, 2014, pp. 212–221.
17. Y. Lifshits: *Processing compressed texts: A tractability border*, in CPM 2007, vol. 4580 of LNCS, 2007, pp. 228–240.
18. W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura, and K. Hashimoto: *Efficient algorithms to compute compressed longest common substrings and compressed palindromes*. Theor. Comput. Sci., 410(8–10) 2009, pp. 900–913.
19. M. Miyazaki, A. Shinohara, and M. Takeda: *An improved pattern matching algorithm for strings in terms of straight-line programs*, in CPM 1997, 1997, pp. 1–11.
20. T. Nishimoto, T. I, S. Inenaga, H. Bannai, and M. Takeda: *Dynamic index, LZ factorization, and LCE queries in compressed space*. CoRR, abs/1504.06954 2015.
21. W. Rytter: *Application of Lempel-Ziv factorization to the approximation of grammar-based compression*. Theor. Comput. Sci., 302(1-3) 2003, pp. 211–222.
22. K. Sadakane: *Succinct data structures for flexible text retrieval systems*. J. Discrete Algorithms, 5(1) 2007, pp. 12–22.
23. B. Schieber and U. Vishkin: *On finding lowest common ancestors: Simplification and parallelization*. SIAM J. Comput., 17(6) 1988, pp. 1253–1262.
24. P. Weiner: *Linear pattern-matching algorithms*, in Proc. of 14th IEEE Ann. Symp. on Switching and Automata Theory, 1973, pp. 1–11.
25. J. Ziv and A. Lempel: *A universal algorithm for sequential data compression*. IEEE Transactions on Information Theory, IT-23(3) 1977, pp. 337–349.