Faster Range LCP Queries in Linear Space

Yakov Nekirch
Michigan Technological University, Houghton, MI, USA

Sharma V. Thankachan
North Carolina State University, Raleigh, NC, USA

Abstract

A range LCP query $\mathsf{rlcp}(\alpha, \beta)$ on a text $T[1..n]$ asks to return the length of the longest common prefix of any two suffixes of T with starting positions in a range $[\alpha, \beta]$. In this paper we describe a data structure that uses $O(n)$ space and supports range LCP queries in time $O(\log^{\varepsilon} n)$ for any constant $\varepsilon > 0$. Our result is the fastest currently known linear-space solution for this problem.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Data Structures, String Algorithms, Longest Common Prefix

Digital Object Identifier 10.4230/OASIcs.Grossi.2025.16

Category Research

Funding *Yakov Nekirch*: Supported by the National Science Foundation under NSF grant 2203278.
Sharma V. Thankachan: Supported by the National Science Foundation under NSF grant 2315822.

Problem Definition and Previous Work

In this note we consider a variant of the longest common prefix (LCP) problem, called the *range LCP problem.* In this problem we store a text $T[1..n]$ in a data structure so that range LCP queries can be answered efficiently. A range LCP query $[\alpha, \beta]$ asks to return the length of the longest common prefix of any two suffixes with starting positions in a range $[\alpha, \beta]$,

$$\mathsf{rlcp}(\alpha, \beta) = \max\{\mathsf{LCP}(i, j) \mid i \neq j \text{ and } i, j \in [\alpha, \beta]\},$$

where $\mathsf{LCP}(i, j)$ denotes the length of the longest common prefix of $T[i..n]$ and $T[j..n]$.

This problem and its variants were considered in several papers, see e.g., [5, 9, 2, 3, 1, 10, 7]. The currently fastest data structure by Amir et al. [2] uses $O(n \log n)$ words of space and answers range LCP queries in $O(\log \log n)$ time. Henceforth we assume that a word of space consists of $\log n$ bits. The data structure with $O(n)$ space usage by Abedin et al. [1] supports queries in $O(\log^{1+\varepsilon} n)$ time for any constant $\varepsilon > 0$. The data structure of Matsuda et al. [10] uses $O(nH_0)$ bits of space where H_0 is the 0-order entropy of the text T; however this space usage is achieved at a cost of significantly higher query time as their data structure supports queries in time $O(n^{\varepsilon})$.

In this note we describe a new trade-off between the space usage and the query time: Our data structure uses linear space and supports queries in time $O(\log^{\varepsilon} n)$ for any constant $\varepsilon > 0$. Thus we achieve the same space usage as in [1] and query time that is close to [2]. Our solution combines the techniques from some previous papers with some new ideas. The compact data structure for predecessor queries by Grossi et al. [8] is also used in our construction.

From Strings to Graphs, and Back Again: A Festschrift for Roberto Grossi's 60th Birthday.
Editors: Alessio Conte, Andrea Marino, Giovanna Rosone, and Jeffrey Scott Vitter; Article No. 16; pp. 16:1–16:6
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Notation

We will say that a triple (i, j, h) is a *bridge* if $1 \le i < j \le n$ and $\mathsf{LCP}(i, j) = h$. The total number of bridges is $O(n^2)$. However in order to answer a range LCP query it is sufficient to consider a subset of all bridges of size $O(n \log n)$ [2, 1]. Following the method of Abedin et al. [1], we consider *special bridges* that are defined below.

We consider the suffix tree of the text T and divide its nodes into heavy and light nodes [11]. Let size(u) denote the number of leaves in the subtree rooted at u. The root is a light node. Exactly one child u' of every internal node u is designated as a heavy node, specifically one with the largest size(u') (ties are broken arbitrarily). All other nodes are light. Let ℓ_i and ℓ_j denote the leaves in the suffix tree that hold suffixes $T[i..n]$ and $T[j..n]$ respectively. Let u denote the lowest common ancestor of ℓ_i and ℓ_j and let u_i and u_j denote the children of u that are ancestors of ℓ_i and ℓ_j respectively. A bridge (i, j, h) is a special bridge if one of the following conditions is satisfied:

1. u_i is a light node **and** $j = \min\{x \mid (i, x, h) \text{ is a bridge}\}$
2. u_j is a light node **and** $i = \max\{x \mid (x, j, h) \text{ is a bridge}\}$

▶ **Lemma 1** ([1]). *There are $O(n \log n)$ special bridges. For any α and β such that $1 \le \alpha \le \beta \le n$, we have $\mathsf{rlcp}(\alpha, \beta) = \max\{\, h \mid (i, j, h)$ is a special bridge and $\alpha \le i, j \le \beta \,\}$*

In the rest of this paper a bridge will denote a special bridge.

Bridge Classification

Let $\Delta = \log n$. For a bridge (i, j, h) we will say that i is its *left leg*, j is its *right leg*, and h is its *height*. We will say that a bridge (i, j, h) is in the interval $[a, b]$ if $a \le i \le j \le b$. Let $\mathcal{B}_t$ denote the set of all bridges of height t.

By pigeonhole principle, there exists some value π, $1 \le \pi \le \Delta$, such that the total number of bridges in $\cup_{k \ge 0} \mathcal{B}_{\pi + k\Delta}$ is bounded by $O(\frac{n \log n}{\Delta}) = O(n)$; see also [1, Section 5.1] We will say that all bridges in $\mathcal{B}_1 \cup (\cup_{k \ge 0} \mathcal{B}_{\pi + k\Delta})$ are *base bridges*. All other bridges are *implicit* bridges. Data structures for different categories of bridges are described below.

Base bridges

The total number of base bridges is $O(n)$. Furthermore we can find the maximum height base bridge in a query range by answering a variant of an orthogonal range searching query. Our data structure for base bridges is summarized in the following lemma.

▶ **Lemma 2.** *There exists an $O(n)$-word data structure that finds, for any interval $[\alpha, \beta]$, the base bridge with maximal height in $[\alpha, \beta]$. The query time is $O(\log \log n)$.*

Proof. There is at most one bridge $(i, j, 1)$ for every value of i, $1 \le i \le n$ and the total number of bridges in $\mathcal{B}_1$ is $O(n)$. Hence the total number of base bridges is $O(n)$. In order to answer a query, we must find the largest h such that $\alpha \le i \le \beta$, $\alpha \le j \le \beta$, and there is a base bridge (i, j, h). Since $i \le j$, this is equivalent to finding the triple (i, j, h) such that $i \ge \alpha$, $j \le \beta$, and h is maximized. The latter query is equivalent to a two-dimensional dominance maxima query. Using a data structure for top-k dominance queries with $k = 1$, such a query can be answered in $O(\log \log n)$ time using $O(n)$ space, see [4, Theorem 7]. ◀

Implicit Bridges

Using Lemma 2, we can find the largest h_0 such that there is a base bridge of height h_0 in the query range $[\alpha, \beta]$. Now we explain how to find the largest h, $h_0 < h < h_0 + \Delta$, such that there is (an implicit) bridge of height h in $[\alpha, \beta - \Delta]$.

The following properties of special bridges will be used.

▶ **Lemma 3** ([1], Lemma 6)**.** *If there is a special bridge (i, j, h), then for any $d < h$ there is a special bridge $(i + d, j' + d, h - d)$ for some $j' \leq j$ and a special bridge $(i' + d, j + d, h - d)$ for some $i' \geq i$.*

▶ **Lemma 4** ([1], Lemma 5)**.** *There exists a data structure that uses $O(n)$ space and supports the following queries in $O(\log^{\varepsilon} n)$ time:*

- *find the right leg j of a special bridge (i, j, h) if its left leg i and its height h are known*
- *find the left leg i of a special bridge (i, j, h) if its right leg j and its height h are known*

Let H_0 denote the set of heights of base bridges. For every $h_0 \in H_0$ we consider all bridges of height h_0 and construct the list of their left legs sorted in increasing order $L(h_0) = \{ s_1, s_2, \ldots, s_{n_0} \}$, where n_0 is the number of special bridges with height h_0. For each k, $1 \leq k \leq \Delta$, we also construct an array $R_{h_0,k}[1..n_0]$ so that $R_{h_0,k}[i] = \min\{ j \,|\, \mathsf{LCP}(s_i - k, j) \geq h_0 + k \}$. Finally let

$$\mathrm{Min}(h_0, k, a) = \min\{R_{h_0,k}[i] \,|\, i_a \leq i \},$$

where s_{i_a} is the successor of $(a + k)$ in $L(h_0)$ and i_a is the position of s_{i_a} in $L(h_0)$.

▶ **Lemma 5.** *If $\mathrm{Min}(h_0, k, a) \leq b$, then there is a bridge of height $h_0 + k$ in $[a, b]$.*
If $\mathrm{Min}(h_0, k, a) > b$, then there is no bridge (i, j, h') in $[a, b - \Delta]$ such that $h' \geq h_0 + k$

Proof. The first statement directly follows from the definition of Min: if $\mathrm{Min}(h_0, k, a) \leq b$, then there is an index i_m, such that $s_{i_m} \geq a + k$ and a position j, such that $s_{i_m} - k < j \leq b$ and $\mathsf{LCP}(s_{i_m} - k, j) \geq h_0 + k$. Hence there is also a special bridge $(s_{i_m} - k, j', h_0 + k)$ where $j' \leq j \leq b$ and $a \leq s_{i_m} - k \leq b$.
To prove the second part of the lemma, suppose that there is a bridge (i, j, h') where $a \leq i \leq b - \Delta$, $a \leq j \leq b - \Delta$ and $h' = h_0 + k + f$ for some f, $0 \leq f \leq \Delta - k$. Then $\mathsf{LCP}(i, j) = h_0 + k + f$ and $\mathsf{LCP}(i + k + f, j + k + f) = h_0$. Hence there is a base bridge $(i + k + f, j_0', h_0)$ for some $j_0' \leq j + k + f$. Let t denote the position of $(i + k + f, j_0', h_0)$ in $L(h_0)$. Furthermore $\mathsf{LCP}(i + f, j + f) = h_0 + k$ and there is an implicit bridge $(i + f, j_1', h_0 + k)$ for some $j_1' \leq j + f$. Since $j + f \leq b$, $R_{h_0,k}[t] \leq b$ and $\mathrm{Min}(h_0, k, a) \leq b$. ◀

▶ **Lemma 6.** *For any $h_0 \in H_0$, there exists a data structure that uses $O(n_0 \log n)$ bits and determines whether $\mathrm{Min}(h_0, k, a) \leq b$ in $O(\log^{\varepsilon} n)$ time for any $1 \leq a \leq b \leq n$ and for any $1 \leq k \leq \Delta$.*

Proof. For every k, $1 \leq k \leq \Delta$, we store a compact data structure that supports range minimum queries on $R_{h_0,k}$ in $O(1)$ time and uses $O(n_0)$ bits of space. We can use the data structure from [6] for this purpose. All compact range minima data structures use $O(n_0 \Delta) = O(n_0 \log n)$ bits. Arrays $R_{h_0,k}$ are not stored. Additionally we store $L(h_0)$ in a data structure that uses $O(n_0 \log n)$ bits and supports successor queries in $O(\log \log n)$ time [12].

To compute $\text{Min}(h_0, k, a)$, we find the smallest $s \in L(h_0)$ such that $s \geq a + k$. Then we use the range minimum data structure and find the index i_m such that $i_m \geq s$ and $R_{h_0,k}[i_m] \leq R_{h_0,k}[i]$ for any $i \geq s$. Finally we compute the right leg j_m of the bridge $(i_m, j_m, h_0 + k)$ using Lemma 4. If $j_m > b$, then $\text{Min}(h_0, k, a) > b$. If $j_m \leq b$, then $\text{Min}(h_0, k, a) \leq b$. ◀

We keep a data structure from Lemma 6 for every $h_0 \in H_0$. The total space usage of these data structures is $O(n \log n)$ bits. Using Lemma 6 and binary search, we find the largest k such that $\text{Min}(h_0, k, a) \leq b$. By Lemma 5, there is a bridge of height $h_0 + k$ in $[a, b]$; hence $\mathsf{LCP}(a, b) \geq h$. Also, by Lemma 5 there is no bridge (i, j, h) in $[a, b - \Delta]$, such that $h \geq h_0 + k + 1$. The total time is $O(\log^\varepsilon n \log\log n)$. We thus proved the following lemma.

▶ **Lemma 7.** *There exists an $O(n)$-word data structure that finds, for any interval $[\alpha, \beta]$, the implicit bridge with height h_i in $[\alpha, \beta]$, so that there is no bridge of height $h > h_i$ in $[\alpha, \beta - \Delta]$. The query time is $O(\log^\varepsilon n \log\log n)$.*

Block Bridges

It remains to consider the case of bridges with right leg in the interval $[\beta - \Delta, \beta]$ and of height h, $h_0 < h < h_0 + \Delta$. We apply the pigeonhole principle again. Let $\Delta_1 = \log\log n$. There exists some value π_1, $1 \leq \pi_1 \leq \Delta_1$, such that the total number of bridges in $\cup_{k\geq 0}\mathcal{B}_{\pi_1 + k\Delta_1}$ is bounded by $O(\frac{n \log n}{\Delta_1}) = O(n\frac{\log n}{\log\log n})$. Let $H_1 = \{\, \pi_1 + k\Delta_1 \mid 1 \leq k \leq (n - \pi_1)/\Delta_1 \,\}$. We will say that all bridges $\mathcal{B}_t$ where $t \in H_0 \cup H_1$ are *good bridges*. All other bridges are *bad* bridges.

We will denote by Block_{t,h_0} the set of all bridges (i, r, h) such that (a) $h \in H_0 \cup H_1$ and $h_0 \leq h < h_0 + \Delta$ for some $h_0 \in H$ and (b) $t\Delta + 1 \leq r \leq (t + 1)\Delta$ for some k, $0 \leq k < \frac{n}{\Delta}$.

▶ **Lemma 8.** *Let m denote the number of bridges in Block_{t,h_0}. There exists a data structure that finds, for any interval $[\alpha, \beta]$, the bridge from Block_{t,h_0} with maximal height such that its right and left legs are in $[\alpha, \beta]$. This data structure uses $O(\log n + m \log\log n)$ bits and supports queries in $O(\log^\varepsilon n)$ time.*

Proof. Let I_{t,h_0} denote the set that contains left legs of all bridges in Block_{t,h_0}. Every bridge (i, j, h) from Block_{t,h_0} is represented as follows: We replace the right leg j with $d(j) = j - t\Delta$ and the height h with $d(h) = h - h_0$. We replace the left leg i with $r(i)$, where $r(i) = |\{\, x \leq i \,|\, x \in I_{t,h_0} \}|$ is the rank of i in I_{t,h_0}. Thus a bridge $(i, j, h) \in \text{Block}_{t,h_0}$ is represented by a triple $(r(i), d(j), d(h))$. Since $r(i)$, $d(j)$, and $d(h)$ are bounded by Δ we can store each triple $(r(i), d(j), d(h))$ using $O(\log\log n)$ bits. For each $(r(i), d(j), d(h))$, we can retrieve the corresponding values of j and h with one addition. If j and h of some bridge (i, j, h) are known, we can obtain the value of its left leg i in $O(\log^\varepsilon n)$ time using the data structure from Lemma 4.

In addition, we store all elements of I_{t,h_0} in a compact data structure that is described by Grossi et al. in [8, Lemma 3.3]. This data structure supports successor queries on a set of integers S; provided that we can access an arbitrary element of S in time t_{acc}, a successor query can be answered in time $O(\log m / \log\log n + t_{\text{acc}})$ where m is the number of elements in S. The data structure uses $O(\log\log u)$ bits per element, where u is the size of the universe (in addition to the space required to store S). In our case, I_{t,h_0} has $O(\Delta^2)$ elements and the size of the universe is n. Hence for every left leg $i \in I_{t,h_0}$, the data structure uses $O(\log\log n)$ bits. We can obtain the value of any left leg in time $O(\log^\varepsilon n)$. Hence successor queries are answered in $O((\log \Delta^2)/\log\log n + \log^\varepsilon n) = O(\log^\varepsilon n)$ time. That is, we can find for any α the smallest $i_\alpha \in I_{t,h_0}$ such that $i_\alpha \geq \alpha$.

Finally all triples $(r(i), d(j), d(h))$ are stored in the data structure described in [4, Theorem 7] [1]. This data structure uses $O(m \log m) = O(m \log \log n)$ bits and supports the following range maxima queries in $O(\log \log n)$ time: for any r_α and d_β, find the highest $d(h)$, among all tuples $(r(i), d(j), d(h))$ satisfying $r(i) \geq r_\alpha$ and $d(j) \geq d_\beta$.

In order to find the maximum-height bridge in $[\alpha, \beta]$ from Block_{t,h_0}, we find the successor of α and its rank $r(\alpha)$, using the compact successor data structure. We also compute $d(\beta) = \beta - t\Delta$. Then we find the highest value $d(h_{\max})$ among all $(r(i), d(j), d(h))$ satisfying $r(i) \geq r_\alpha$ and $d(j) \leq d_\beta$. The maximum height of a bridge in $[\alpha, \beta]$ is $h_{\max} = d(h_{\max}) + h_0$. The total time required to answer a query is $O(\log^\varepsilon n + \log \log n) = O(\log^\varepsilon n)$. ◀

▶ **Lemma 9.** *There are $O(n)$ non-empty blocks.*

Proof. Suppose that there is at least one implicit bridge (i, j, h) in Block_{t,h_0}. Let $k = h - h_0$. Then by Lemma 3 there is a special bridge $(i', j + k, h_0)$ such that $i + k \leq i' \leq j + k$. Since $1 \leq k < \Delta$ and $t\Delta < j \leq (t+1)\Delta$, we have $t\Delta < j + k \leq t + 2\Delta$. Thus for every base bridge (i', j', h_0) where $h_0 \in H_0$ there are at most two non-empty blocks. Since there are $O(n)$ base bridges, the number of non-empty blocks is $O(n)$. ◀

▶ **Lemma 10.** *There exists a data structure that finds, for any $\alpha \leq \beta$, and any $h_0 \in H_0$, the highest good bridge (i, j, h) in $[\alpha, \beta]$ such that its right leg j is in $[\beta - \Delta, \beta]$ and its height h is in $[h_0, h_0 + \Delta]$. The data structure uses $O(n)$ words of space and supports queries in $O(\log^\varepsilon n)$ time.*

Proof. We store a block data structure from Lemma 8 for each non-empty block Block_{t,h_0}. The total number of bridges in all blocks is equal to the total number of good bridges. By Lemma 9, the total number of non-empty blocks is $O(n)$. Hence the total space usage of all block data structures is $O(n \log n + (n \frac{\log n}{\log \log n}) \log \log n) = O(n \log n)$ bits. The range $[\beta - \Delta, \beta]$ intersects with at most two blocks. Hence we can find the highest bridge satisfying the conditions of this lemma in time $O(\log^\varepsilon n)$ by answering two queries to block data structures. ◀

Putting All Parts Together

In order to answer a range LCP query $[\alpha, \beta]$ we need to identify the largest h such that there is a bridge (i, j, h) in $[\alpha, \beta]$. Our algorithm works in four stages:

1. First, we find the largest h_0 such that there is a base bridge of height h_0 in $[\alpha, \beta]$. This step takes $O(\log \log n)$ time by Lemma 2.
2. Then we find the largest h_i, where $h_0 < h_i < h_0 + \Delta$, such that there is an implicit bridge of height h_i in $[\alpha, \beta - \Delta]$. This can be done in $O(\log^\varepsilon n \log \log n)$ time by Lemma 7
3. We find the largest h_n such that there is a good bridge of height $h_n > h_0$ with right leg in $[\beta - \Delta, \beta]$. This step takes $O(\log^\varepsilon n)$ time by Lemma 10.
4. Let $h_1 = \max(h_0, h_i, h_n)$. We check if there is a bridge of height h in $[\alpha, \beta]$ for each h, $h_1 < h < h_1 + \Delta_1$. By Lemma 5 we can check each candidate value of h in $O(\log^\varepsilon n)$ time. Hence this step takes $O(\log^\varepsilon \Delta_1) = O(\log^\varepsilon n \log \log n)$ time.

The total query time is $O(\log^\varepsilon n \log \log n)$. By replacing ε with a constant $\varepsilon' < \varepsilon$ in the above construction, we obtain our final result.

▶ **Theorem 11.** *There exists a data structure that uses $O(n)$ words of space and answers range LCP queries in time $O(\log^\varepsilon n)$ time.*

---

[1] The same data structure was also used in Lemma 2.

References

1 Paniz Abedin, Arnab Ganguly, Wing-Kai Hon, Kotaro Matsuda, Yakov Nekrich, Kunihiko Sadakane, Rahul Shah, and Sharma V. Thankachan. A linear-space data structure for range-lcp queries in poly-logarithmic time. *Theor. Comput. Sci.*, 822:15–22, 2020. doi:10.1016/J.TCS.2020.04.009.

2 Amihood Amir, Alberto Apostolico, Gad M. Landau, Avivit Levy, Moshe Lewenstein, and Ely Porat. Range LCP. *J. Comput. Syst. Sci.*, 80(7):1245–1253, 2014. doi:10.1016/J.JCSS.2014.02.010.

3 Amihood Amir, Moshe Lewenstein, and Sharma V. Thankachan. Range LCP queries revisited. In *22nd International Symposium String Processing and Information Retrieval (SPIRE)*, pages 350–361, 2015. doi:10.1007/978-3-319-23826-5_33.

4 Timothy M. Chan, Yakov Nekrich, Saladi Rahul, and Konstantinos Tsakalidis. Orthogonal point location and rectangle stabbing queries in 3-d. *J. Comput. Geom.*, 13(1):399–428, 2022. doi:10.20382/JOCG.V13I1A15.

5 Graham Cormode and S. Muthukrishnan. Substring compression problems. In *16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 321–330, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070478.

6 Johannes Fischer and Volker Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM J. Comput.*, 40(2):465–492, 2011. doi:10.1137/090779759.

7 Arnab Ganguly, Manish Patil, Rahul Shah, and Sharma V. Thankachan. A linear space data structure for range LCP queries. *Fundam. Informaticae*, 163(3):245–251, 2018. doi:10.3233/FI-2018-1741.

8 Roberto Grossi, Alessio Orlandi, Rajeev Raman, and S. Srinivasa Rao. More haste, less waste: Lowering the redundancy in fully indexable dictionaries. In *26th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 517–528, 2009. doi:10.4230/LIPICS.STACS.2009.1847.

9 Orgad Keller, Tsvi Kopelowitz, Shir Landau Feibish, and Moshe Lewenstein. Generalized substring compression. *Theor. Comput. Sci.*, 525:42–54, 2014. doi:10.1016/j.tcs.2013.10.010.

10 Kotaro Matsuda, Kunihiko Sadakane, Tatiana Starikovskaya, and Masakazu Tateshita. Compressed orthogonal search on suffix arrays with applications to range LCP. In *31st Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 23:1–23:13, 2020. doi:10.4230/LIPICS.CPM.2020.23.

11 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

12 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inf. Process. Lett.*, 6(3):80–82, 1977. doi:10.1016/0020-0190(77)90031-X.