A Linear-Space Data Structure for Range-LCP Queries in Poly-Logarithmic Time

Paniz Abedin[1], Arnab Ganguly[2], Wing-Kai Hon[3], Yakov Nekrich[4], Kunihiko Sadakane[5], Rahul Shah[6,7], and Sharma V. Thankachan[1(✉)]

[1] Department of Computer Science, University of Central Florida, Orlando, USA
paniz@cs.ucf.edu, sharma.thankachan@ucf.edu
[2] Department of Computer Science, University of Wisconsin - Whitewater, Whitewater, USA
gangulya@uww.edu
[3] Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
wkhon@cs.nthu.edu.tw
[4] Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
yakov.nekrich@googlemail.com
[5] Department of Computer Science, The University of Tokyo, Tokyo, Japan
sada@mist.i.u-tokyo.ac.jp
[6] Department of Computer Science, Louisiana State University, Baton Rouge, USA
rahul@csc.lsu.edu
[7] National Science Foundation (NSF), Alexandria, USA

Abstract. Let $\mathsf{T}[1,n]$ be a text of length n and $\mathsf{T}[i,n]$ be the suffix starting at position i. Also, for any two strings X and Y, let $\mathsf{LCP}(X,Y)$ denote their longest common prefix. The range-LCP of T w.r.t. a range $[\alpha,\beta]$, where $1 \leq \alpha < \beta \leq n$ is

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

Amir et al. [ISAAC 2011] introduced the indexing version of this problem, where the task is to build a data structure over T, so that $\mathsf{rlcp}(\alpha,\beta)$ for any query range $[\alpha,\beta]$ can be reported efficiently. They proposed an $O(n\log^{1+\epsilon} n)$ space structure with query time $O(\log\log n)$, and a linear space (i.e., $O(n)$ words) structure with query time $O(\delta\log\log n)$, where $\delta = \beta - \alpha + 1$ is the length of the input range and $\epsilon > 0$ is an arbitrarily small constant. Later, Patil et al. [SPIRE 2013] proposed another linear space structure with an improved query time of $O(\sqrt{\delta}\log^{\epsilon}\delta)$. This poses an interesting question, whether it is possible to answer $\mathsf{rlcp}(\cdot,\cdot)$ queries in poly-logarithmic time using a linear space data structure. In this paper, we settle this question by presenting an $O(n)$ space data structure with query time $O(\log^{1+\epsilon} n)$ and construction time $O(n\log n)$.

A part of this work was done at NII Shonan Meeting No. 126: Computation over Compressed Structured Data.

L. Wang and D. Zhu (Eds.): COCOON 2018, LNCS 10976, pp. 615–625, 2018.
https://doi.org/10.1007/978-3-319-94776-1_51

1 Introduction and Related Work

The longest common prefix (LCP) is an important primitive employed in various string matching algorithms. By preprocessing a text $\mathsf{T}[1,n]$ (over an alphabet set Σ) into a suffix tree data structure, we can compute the longest common prefix of any two suffixes of T, say $\mathsf{T}[i,n]$ and $\mathsf{T}[j,n]$, denoted by $\mathsf{LCP}(\mathsf{T}[i,n],\mathsf{T}[j,n])$, in constant time. From now onwards, we use the shorthand notation $\mathsf{lcp}(i,j)$ for the length of $\mathsf{LCP}(\mathsf{T}[i,n],\mathsf{T}[j,n])$. Given its wide range of applicability, various generalizations of LCP has also been studied [1–3,9,15].

In this paper, we focus on the "range" versions of this problem. The line of research was initiated by Cormode and Muthukrishnan. They studied the *Interval Longest Common Prefix* (Interval-LCP) problem in the context of data compression [6,12,13].

Definition 1 (Interval-LCP). *The Interval-LCP of a text* $\mathsf{T}[1,n]$ *w.r.t a query* (p,α,β)*, where* $p,\alpha,\beta \in [1,n]$ *and* $\alpha < \beta$ *is*

$$\mathsf{ilcp}(p,\alpha,\beta) = \max\{\mathsf{lcp}(p,i) \mid i \in [\alpha,\beta]\}$$

As observed by Keller et al. [12], any Interval-LCP query on T can be reduced to two orthogonal range successor/predecessor queries over n points in two dimensions (2D). Therefore, using the best known data structures for orthogonal range successor/predecessor queries [14], we can answer any Interval-LCP query on T in $O(\log^{\epsilon} n)$ time using an $O(n)$ space data structure, where $\epsilon > 0$ is an arbitrarily small positive constant.[1] Moreover, queries with $p \in [\alpha,\beta]$ can be answered in faster $O(\log^{\epsilon} \delta)$ time, where $\delta = \beta - \alpha + 1$ is the length of the input range [15].

Another variation of LCP, studied by Amir et al. [1,2], is the following.

Definition 2 (Range-LCP). *The Range-LCP of a text* $\mathsf{T}[1,n]$ *w.r.t a range* $[\alpha,\beta]$*, where* $1 \le \alpha < \beta \le n$ *is*

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

In order to efficiently solve the data structure version of this problem, Amir et al. [1,2] introduced the concept of "**bridges**" and "**optimal bridges**" and showed that any Range-LCP query on $\mathsf{T}[1,n]$ can be reduced to an equivalent 2D range maximum query over a set of $O(n \log n)$ weighted points in 2D. Therefore, an $O(n \log^{1+\epsilon} n)$ space data structure with $O(\log \log n)$ query time is immediate from the best known result for 2D range maximum problem [4]. The construction time is $O(n \log^2 n)$. By choosing an alternative structure for 2D (2-sided) range maximum query[2], the space can be improved to $O(n \log n)$ with a slowdown in

[1] All results throughout this paper assume the standard unit-cost word RAM model, in which any standard arithmetic or boolean bitwise operation on word-sized operands takes constant time. The space is measured in words of $\log n$ bits unless specified otherwise.

[2] See Theorem 9 in [16] on sorted dominance reporting in 3D.

query time to $O(\log n)$. This sets an interesting question, whether it is possible to reduce the space further without sacrificing the poly-logarithmic query time. Unfortunately, the query times of the existing linear space solutions (listed below) are dependent on the parameter δ, which is $\Theta(n)$ in the worst case.

- $O(n)$ space and $O(\delta \log\log n)$ query time [1,2].
- $O(n)$ space and $O(\sqrt{\delta}\log^{\epsilon}\delta)$ query time [15]

To this end, we present our main contribution below. Our model of computation is the word RAM with word size $\Omega(\log n)$.

Theorem 1. *A text* $\mathsf{T}[1,n]$ *can be preprocessed into an* $O(n)$ *space data structure in* $O(n\log n)$ *time, such that any Range-LCP query on* T *can be answered in* $O(\log^{1+\epsilon} n)$ *time.*

Map. We start with some preliminaries (Sect. 2). We the briefly sketch the framework by Amir et al. [1] in Sect. 3. Sections 4 and 5 are dedicated for the details of our solution. The details of the construction of our data structure is deferred to Appendix.

2 Preliminaries

2.1 Predecessor/Successor Queries

Let $\mathcal{S}$ be a subset of $\{1,2,\ldots,n\}$. Then, $\mathcal{S}$ can be preprocessed into an $O(|\mathcal{S}|)$ space data structure, such that for any query p, we can return $\mathsf{pred}(p,\mathcal{S})$ and $\mathsf{succ}(p,\mathcal{S})$ in $O(\log\log n)$ time [19], where

$$\mathsf{pred}(p,\mathcal{S}) = \max\{i \mid i \le p \quad \textbf{and} \quad i \in \mathcal{S}\cup\{-\infty\}\}$$
$$\mathsf{succ}(p,\mathcal{S}) = \min\{i \mid i \ge p \quad \textbf{and} \quad i \in \mathcal{S}\cup\{\infty\}\}$$

2.2 Range Minimum Query

Let $A[1,n]$ be an array of length n. A range minimum query (RMQ) with an input range $[i,j]$ asks to report $\mathsf{rmq}(i,j) = \arg\min_k\{A[k] \mid k \in [i,j]\}$. By maintaining a data structure of size $2n + o(n)$ bits, any RMQ on A can be answered in $O(1)$ time [8] (even without accessing A).

2.3 2D Range Maximum Query

Let $\mathcal{S}$ be a set of m weighted points in a $[1,n]\times[1,n]$ grid. A 2D-RMQ with input (a,b,a',b') asks to return the highest weighted point in $\mathcal{S}$ within the orthogonal region corresponding to $[a,b]\times[a',b']$. Data structures with the following space-time trade-offs are known for this problem.

- $O(m)$ space, $O(m\log m)$ preprocessing time and $O(\log^{1+\epsilon} m)$ query time [5].
- $O(m\log^{\epsilon} n)$ space, $O(m\log m)$ preprocessing time and $O(\log\log n)$ query time [4].

2.4 Orthogonal Range Predecessor/Successor Queries in 2D

A set $\mathcal{P}$ of n points in an $[1, n] \times [1, n]$ grid can be preprocessed into a linear-space data structure, such that the following queries can be answered in $O(\log^{\epsilon} n)$ time [14].

- $\mathsf{ORQ}([x', x''], [-\infty, y'']) = \arg\max_j\{(i, j) \in \mathcal{P} \cap [x', x''] \times [-\infty, y'']\}$
- $\mathsf{ORQ}([-\infty, x''], [y', y'']) = \arg\max_i\{(i, j) \in \mathcal{P} \cap [-\infty, x''] \times [y', y'']\}$
- $\mathsf{ORQ}([x', x''], [y', +\infty]) = \arg\min_j\{(i, j) \in \mathcal{P} \cap [x', x''] \times [y', +\infty]\}$
- $\mathsf{ORQ}([x', +\infty], [y', y'']) = \arg\min_i\{(i, j) \in \mathcal{P} \cap [x', +\infty] \times [y', y'']\}$

2.5 Suffix Trees and Suffix Arrays

For a string $\mathsf{T}[1, n]$, the suffix array $\mathsf{SA}[1, n]$ is an array of length n, such that $\mathsf{SA}[i]$ denotes the starting position of the lexicographically ith smallest suffix among all suffixes of T. The suffix tree ST is a compact trie of all its suffixes [18]. The suffix tree consists of n leaves and at most $n - 1$ internal nodes. The edges are labeled with substrings of T. For any node u, $\mathsf{path}(u)$ is defined as the concatenation of edge labels on the path from the root of the suffix tree to u. Therefore, $\mathsf{path}(\ell_x) = \mathsf{T}[\mathsf{SA}[x], n]$, where ℓ_x is the xth leftmost leaf node. Moreover, $\mathsf{path}(\mathsf{lca}(\ell_x, \ell_y)) = \mathsf{LCP}(\mathsf{T}[\mathsf{SA}[x], n], \mathsf{T}[\mathsf{SA}[y], n])$, where $\mathsf{lca}(\cdot, \cdot)$ denotes the lowest common ancestor. The suffix tree of T occupies $O(n)$ space, can be constructed in $O(n)$ time and space, and for any two text positions i, j, we can compute $\mathsf{lcp}(i, j)$ in constant time. Also, define inverse suffix array $\mathsf{ISA}[1, n]$, such that $\mathsf{ISA}[i] = j$, where $\mathsf{SA}[j] = i$.

2.6 Heavy Path Decomposition

We define the heavy path decomposition [11,17] of a suffix tree ST as follows. First, we categorize the nodes in ST into light and heavy. The root node is *light* and for any internal node, exactly one child is heavy. Specifically, the child having the largest number of leaves in its subtree (ties are broken arbitrarily). When all incident edges to the light nodes are removed, the remaining edges of ST are decomposed into maximal downward paths, each starting from an internal light node and following a sequence of heavy nodes. We call each path a heavy path.

Lemma 1. *The number of heavy paths intersected by any root to leaf path is at most* $\log_2 n$. *Equivalently, the number of light nodes on any root to leaf path is at most* $\log_2 n$.

3 Amir et al.'s Framework

We start with some definitions.

Definition 3 (Bridges). *Let i and j and two distinct positions in the text T and let $h = \mathsf{lcp}(i, j)$ and $h > 0$. Then, we call the tuple (i, j, h) a bridge. Moreover, we call h its height, i its left leg and j its right leg, and $\mathsf{LCP}(\mathsf{T}[i, n], \mathsf{T}[j, n])$ its label.*

Let $\mathcal{B}_{all}$ be the set of all such bridges. Then clearly,

$$\mathsf{rlcp}(\alpha, \beta) = \max\{h \mid (i, j, h) \in \mathcal{B}_{all} \quad \textbf{and} \quad i, j \in [\alpha, \beta]\}$$

Therefore, by mapping each bridge $(i, j, h) \in \mathcal{B}_{all}$ to a 2D point (i, j) with weight h, the problem can be reduced to a 2D-RMQ problem (refer to Sect. 2.3). This yields an $O(|\mathcal{B}_{all}| \log^{\epsilon} n)$ space data structure with query time $O(\log \log n)$. Unfortunately, this is not a space efficient approach as the size of $\mathcal{B}_{all}$ is $\Theta(n^2)$ in the worst case. To circumvent this, Amir et al. [1] introduced the concept of optimal bridges.

Definition 4 (Optimal Bridges). *A bridge $(i, j, h) \in \mathcal{B}_{all}$ is optimal if there exists no other bridge (i', j', h'), such that $i', j' \in [i, j]$ and $h' \geq h$.*

Let $\mathcal{B}_{opt}$ be the set of all optimal bridges. Then, it is easy to observe that

$$\mathsf{rlcp}(\alpha, \beta) = \max\{h \mid (i, j, h) \in \mathcal{B}_{opt} \quad \textbf{and} \quad i, j \in [\alpha, \beta]\}.$$

Thus, to answer an rlcp query, it is sufficient to examine the bridges in $\mathcal{B}_{opt}$, instead of all the bridges in $\mathcal{B}_{all}$. The crux of Amir et al.'s [1] data structure is the following lemma.

Lemma 2 ([1]). *The size of $\mathcal{B}_{opt}$ is $O(n \log n)$.*

Therefore, by applying the above reduction (from Range-LCP to 2D-RMQ) on the bridges in $\mathcal{B}_{opt}$, they got an $O(|\mathcal{B}_{opt}| \log^{\epsilon} n) = O(n \log^{1+\epsilon} n)$ space data structure with query time $O(\log \log n)$. Additionally, they showed that there exist cases where the the size of $\mathcal{B}_{opt}$ is $\Omega(n \log n)$. For example, when T is a Fibonacci word (see Sect. 4 in [1] for its definition). This means that the bound on the number of optimal bridges is tight.

4 Our Framework

Firstly, we present a replacement for optimal bridges, called *special bridges*.

Definition 5 (Special Bridges). *A bridge $(i, j, h) \in \mathcal{B}_{all}$ is special if there exists no other bridge $(i', j', h') \in \mathcal{B}_{all}$, such that $i', j' \in [i, j]$ and*

$$\mathsf{LCP}(\mathsf{T}[i, n], \mathsf{T}[j, n]) = \mathsf{LCP}(\mathsf{T}[i', n], \mathsf{T}[j', n])$$

Let $\mathcal{B}_{spe}$ be the set of all special bridges. Clearly $\mathcal{B}_{opt} \subseteq \mathcal{B}_{spe}$, therefore

$$\mathsf{rlcp}(\alpha, \beta) = \max\{h \mid (i, j, h) \in \mathcal{B}_{spe} \quad \textbf{and} \quad i, j \in [\alpha, \beta]\}$$

From Lemma 3, $|\mathcal{B}_{spe}| = \Theta(|\mathcal{B}_{opt}|)$, the same space-time trade-off as in Amir et al. [1] can be obtained by employing special bridges instead of optimal bridges. However, the main advantage over optimal bridges is that special bridges can be encoded efficiently, in $O(1)$-bits per bridge.

Lemma 3. *The size of $\mathcal{B}_{spe}$ is $O(n \log n)$.*

Proof. Firstly, we show how to bound the number of special bridges with a fixed label P. Let u be the node in ST such that $\mathsf{path}(u) = P$. For any such bridge (i, j, h), the leaves (say $\ell_{\mathsf{ISA}[i]}$ and $\ell_{\mathsf{ISA}[j]}$) corresponding to the suffixes $\mathsf{T}[i, n]$ and $\mathsf{T}[j, n]$ must be under the subtree of u, but not under the subtree of the same child of u. This means, either $\ell_{\mathsf{ISA}[i]}$ or $\ell_{\mathsf{ISA}[j]}$ must be under a light child of u. Moreover, the starting position of the suffix corresponding to a leaf can be the left-leg (resp., right-leg) of at most one special bridge with label P. Therefore the number of special bridges with a fixed label P is at most twice the sum of subtree sizes of all light children of u. Hence, the total number of special bridges is at most twice the sum of subtree sizes of all light nodes in the suffix tree, which is bounded by $O(n \log n)$, since each leaf is under at most $O(\log n)$ light ancestors. □

We now present an overview of our solution.

4.1 An Overview of Our Data Structure

We start by defining two queries, which are weaker than Range-LCP.

Definition 6. *For a parameter $\Delta = \Theta(\log n)$, a query $\mathcal{E}_\Delta(\alpha, \beta)$ asks to return an estimate τ of $\mathsf{rlcp}(\alpha, \beta)$, such that*

$$\tau \leq \mathsf{rlcp}(\alpha, \beta) < \tau + \Delta$$

Definition 7. *A query $\mathcal{Q}(\alpha, \beta, h)$ asks to return* **YES** *if there exists special bridge (i, j, h), such that $i, j \in [\alpha, \beta]$. Otherwise, $\mathcal{Q}(\alpha, \beta, h)$ returns* **NO**.

The following two are the main components of our data structure.

1. A linear space structure for $\mathcal{E}_\Delta(\cdot, \cdot)$ queries in $O(\log^{1+\epsilon} n)$ time.
2. A linear space structure for $Q(\cdot, \cdot, \cdot)$ queries in $O(\log^{\epsilon} n)$ time.

Our algorithm for computing $\mathsf{rlcp}(\alpha, \beta)$ is straightforward. First obtain $\tau = \mathcal{E}_\Delta(\alpha, \beta)$. Then, for $h = \tau, \tau + 1, \tau + 2, ..., \tau + \Delta - 1$, compute $\mathcal{Q}(\alpha, \beta, h)$. Then report

$$\mathsf{rlcp}(\alpha, \beta) = \max\{h \mid h \in [\tau, \tau + \Delta - 1] \quad \textbf{and} \quad \mathcal{Q}(\alpha, \beta, h) = \textbf{YES}\ \}$$

The time complexity is $\log^{1+\epsilon} n + \Delta \cdot \log^{\epsilon} n = O(\log^{1+\epsilon} n)$ and the space complexity is $O(n)$, as claimed. In what follows, we present the details of these two components of our data structure.

5 Details of the Components

We maintain the suffix tree ST of T and the linear space data structure for various 2D range successor/predecessor queries (in $O(\log^{\epsilon} n)$ time [14]) over the following set of n points.

$$\mathcal{P} = \{(i, \mathsf{SA}[i]) \mid i \in [1, n]\}$$

We rely on this structure for computing interval-LCP and left-leg/right-leg queries (to be defined next).

Lemma 4. *We can answer an interval-LCP query* $\mathsf{ilcp}(p, \alpha, \beta)$ *in time* $O(\log^{\epsilon} n)$.

Proof. Find the leaf $\ell_{\mathsf{ISA}[p]}$ first. Then find the rightmost leaf ℓ_x before $\ell_{\mathsf{ISA}[p]}$ and the leftmost leaf ℓ_y after $\ell_{\mathsf{ISA}[p]}$, such that $\mathsf{SA}[x], \mathsf{SA}[y] \in [\alpha, \beta]$. We can rely on the following queries for this:

$$x = \mathsf{ORQ}([-\infty, p-1], [\alpha, \beta]) \quad \textbf{and} \quad y = \mathsf{ORQ}([p+1, +\infty], [\alpha, \beta])$$

Clearly, $\mathsf{ilcp}(p, \alpha, \beta)$ is given by $\max\{\mathsf{lcp}(p, x), \mathsf{lcp}(p, y)\}$. This completes the proof. □

Definition 8. *Let* $(i, j, h) \in \mathcal{B}_{spe}$, *then define*

$$\mathsf{rightLeg}(i, h) = j \quad \textbf{\textit{and}} \quad \mathsf{leftLeg}(j, h) = i$$

If there exists no j, *such that* $(i, j, h) \in \mathcal{B}_{spe}$, *then* $\mathsf{rightLeg}(i, h) = \infty$. *Similarly, if there exists no* i, *such that* $(i, j, h) \in \mathcal{B}_{spe}$, *then* $\mathsf{leftLeg}(j, h) = -\infty$. *If exists, then* $\mathsf{rightLeg}(i, h)$ *(resp.* $\mathsf{leftLeg}(j, h)$*) is unique.*

Lemma 5. *By maintaining a linear space data structure, we can answer* $\mathsf{rightLeg}(k, h)$ *and* $\mathsf{leftLeg}(k, h)$ *queries in* $O(\log^{\epsilon} n)$ *time.*

Proof. Find the ancestor u (if it exists) of $\ell_{\mathsf{ISA}[k]}$, such that $|\mathsf{path}(u)| = h$ via a weighted level ancestor query on ST[3]. If u does not exist, then $\mathsf{rightLeg}(k, h) = +\infty$ and $\mathsf{leftLeg}(k, h) = -\infty$. Otherwise, let u' be the child of u, such that $\ell_{\mathsf{ISA}[k]}$ is under u'. Also, let $[x, y]$ and $[x', y']$ be the range of leaves under u and u', respectively. Then,

$$\mathsf{rightLeg}(k, h) = \min(\mathsf{ORQ}([x, y] \backslash [x', y'], [k+1, +\infty]), +\infty)$$

$$\mathsf{leftLeg}(k, h) = \max(\mathsf{ORQ}([x, y] \backslash [x', y'], [-\infty, k-1]), -\infty)$$

This completes the proof. □

The structures described in Lemma 4 and Lemma 5 are the building blocks of our main components, to be described next. The following observation is exploited in both.

Lemma 6. *Suppose that* $(i, j, h) \in \mathcal{B}_{spe}$. *Then,* $\forall k \in [1, h-1]$, *there exists* $(i+k, \cdot, h-k) \in \mathcal{B}_{spe}$ *such that* $\mathsf{rightLeg}(i+k, h-k) \in (i+k, j+k]$.

Proof. Given $\mathsf{lcp}(i, j) = h$, we have $\mathsf{lcp}(i+k, j+k) = (h-k)$. Clearly, $(i+k, j+k, h-k) \in \mathcal{B}_{all}$. This means, there exists a special bridge $(i+k, l_k, h-k)$, where l_k is the smallest integer after $i+k$, such that $\mathsf{lcp}(i+k, l_k) = h-k$. Equivalently, $l_k = \mathsf{rightLeg}(i+k, h-k)$. Clearly, $l_k \leq j+k$, since $\mathsf{lcp}(i+k, j+k) = h-k$. This completes the proof. □

[3] Weighted level ancestor queries on suffix trees can be answered in $O(1)$ time using a linear space data structure [10] (also see [7]).

5.1 The Structure for Estimating Range-LCP

Let $\mathcal{B}_t$ denotes the set of all special bridges with height t. Also, for $f = 0, 1, 2, \ldots, (\Delta - 1)$, where $\Delta = \Theta(\log n)$, define $\mathcal{C}_f$: the set of all special bridges with its height divided by Δ leaving remainder f. Specifically,

$$\mathcal{C}_f = \bigcup_{k=0}^{\lfloor \frac{n-f}{\Delta} \rfloor} \mathcal{B}_{(f+k\Delta)}$$

Let $\mathcal{C}_\pi : \pi \in [0, \Delta - 1]$ be the smallest set among all $\mathcal{C}_f$'s. Its size can be bounded by $O((n \log n)/\Delta)$ (by pigeonhole principle), which is $O(n)$. We map each special bridge $(i, j, h) \in \mathcal{C}_\pi$ into a 2D point (i, j) with weight h and maintain the linear-space data structure over them for answering 2D-RMQ. We use the linear-space structure by Chazelle [5]. The space is $|\mathcal{C}_\pi| = O(n)$ words and the query time is $O(\log^{1+\epsilon} n)$.

Our Algorithm. Let (α^*, β^*, h^*) be the tallest special bridge, such that both $\alpha^*, \beta^* \in [\alpha, \beta]$. For computing $\mathcal{E}_\Delta(\alpha, \beta)$, we query on the 2D-RMQ structure over $\mathcal{C}_\pi$ and find the tallest bridge $(i', j', h') \in \mathcal{C}_\pi$, such that $i', j' \in [\alpha, \beta]$. Two possible scenarios are

1. $\beta^* \in (\alpha, \beta - \Delta]$: We claim that $h^* \in [h', h' + \Delta)$. Proof follows from Lemma 6.
2. $\beta^* \in (\beta - \Delta, \beta]$: We can rely on Interval-LCP queries. Specifically, $h^* = \max\{\mathsf{ilcp}(p, \alpha, \beta) \mid p \in (\beta - \Delta, \beta]\}$.

By combining both cases, we have

$$\mathcal{E}_\Delta(\alpha, \beta) = \max\Big(\{\mathsf{ilcp}(p, \alpha, \beta) \mid p \in (\beta - \Delta, \beta]\} \cup \{h'\}\Big)$$

The time complexity is proportional to that of one 2D-RMQ and at most Δ number of Interval-LCP queries. That is, $\log^{1+\epsilon} n + \Delta \cdot \log^{\epsilon} n = O(\log^{1+\epsilon} n)$.

5.2 The Structure for Handling $\mathcal{Q}(\alpha, \beta, h)$ Queries

Recall that $\mathcal{B}_t$ is the set of all special bridges with height t. Let L_t represent the sorted list of left-legs of all bridges in $\mathcal{B}_t$ in the form of a y-fast trie for fast predecessor search. Also, let R_t be another array, such that $R_t[k] = \mathsf{rightLeg}(L_t[k], t)$. In other words, for $k = 1, 2, ..., |\mathcal{B}_t|$, $L_t[k]$ (resp., $R_t[k]$) denotes the left-leg (resp., right-leg) of kth bridge among all bridges in $\mathcal{B}_t$ in the ascending order of left-leg. Also, let

$$\mathcal{S}_\pi = \{\pi, \pi + \Delta, \pi + 2\Delta, \pi + 3\Delta, ..., (\pi + \lfloor (n - \pi)/\Delta \rfloor \Delta)\}$$

For each $t \in [1, n]$, we maintain a separate structure that can answer queries of the type $\mathcal{Q}(\cdot, \cdot, t)$. Based on whether h in the query $\mathcal{Q}(\alpha, \beta, h)$ is in $\mathcal{S}_\pi$ or not, we have two cases.

Case 1: $h \in \mathcal{S}_\pi$ To handle this case, we maintain L_t and the succinct data structure for range minimum query (RMQ) on R_t for all $t \in \mathcal{S}_\pi$. The total space is $|\mathcal{C}_\pi| = O(n)$ words. Therefore, any query $\mathcal{Q}(\alpha, \beta, h)$ with $h \in \mathcal{S}_\pi$ can be answered using the following steps.

1. Find the smallest k, such that $L_h[k] \geq \alpha$ via a successor query.
2. Then, find the index k' corresponding to the smallest element in $R_h[k, |R_h|]$ using a range minimum query. Note that R_h is not stored.
3. Then, find $\mathsf{rightLeg}(L_h[k'], h)$ and report "YES" if it is $\leq \beta$, and report "NO" otherwise.

The time complexity is $(\log\log n + \log^{\epsilon} n) = O(\log^{\epsilon} n)$. The correctness can be easily verified.

Case 2: $h \notin \mathcal{S}_\pi$ We first show how to design a structure for a predefined h. Let $q = \mathsf{pred}(h, S_\pi) = \pi + \Delta \cdot \lfloor (h-\pi)/\Delta \rfloor$ and $z = (h-q)$. Note that for each special bridge (i, j, h), there exists a special bridge $(i+z, \cdot, h-z) = (i+z, \cdot, q)$ (refer to Lemma 6). This implies the following.

$$\{L_h[k] \mid k \in [1, |\mathcal{B}_h|]\} \subseteq \{(L_q[k] - z) \mid k \in [1, |\mathcal{B}_q|]\} \tag{1}$$

Now, define an array R'_h of length $|\mathcal{B}_q|$, such that for any $k \in [1, |\mathcal{B}_q|]$, $R'_h[k] = \mathsf{rightLeg}((L_q[k] - z), h)$. Note that $R'_h[k] = \infty$ if there exists no special bridge with left-leg $(L_q[k] - z)$ and height h. Our data structure is a succinct range minimum query (RMQ) structure over R'_h. We now show how to answer an $Q(\alpha, \beta, h)$ query using R'_h and L_q (in Case 1). The steps are as follows.

1. Find the smallest k, such that $(L_q[k] - z) \geq \alpha$. We perform a successor query on L_q for this.
2. Then, find the index k' corresponding to the smallest element in $R'_h[k, |R_q|]$ using a range minimum query.
3. Then, find $\mathsf{rightLeg}(L_h[k'] - z, h)$ and report "YES" if it is $\leq \beta$, and report "NO" otherwise.

The time complexity is $(\log\log n + \log^{\epsilon} n) = O(\log^{\epsilon} n)$. The correctness follows from the definition of R'_h and Eq. 1. The space complexity for a fixed h is $|\mathcal{B}_q|(2 + o(1))$ bits. Therefore, by maintaining the above structure for all values of h, we can answer $\mathcal{Q}(\alpha, \beta, h)$ for any α, β and h in $O(\log^{\epsilon} n)$ time. Total space (in bits) is:

$$(2 + o(1)) \sum_{h=1}^{n} |\mathcal{B}_{\pi + \Delta \cdot \lfloor (h-\pi)/\Delta \rfloor}| = (2 + o(1))\Delta \sum_{q \in \mathcal{S}_\pi} |\mathcal{B}_q| = O(n \log n).$$

In summary, any Range-LCP query on the text $\mathsf{T}[1, n]$ can be answered in $O(\log^{1+\epsilon} n)$ time using a linear space data structure. We remark that our data structure can be constructed in $O(n \log n)$ time.

Acknowledgments. This research is supported in part by the U.S. NSF under the grants CCF-1703489 and CCF-1527435, and the Taiwan Ministry of Science and Technology under the grant 105-2221-E-007-040-MY3.

References

1. Amir, A., Apostolico, A., Landau, G.M., Levy, A., Lewenstein, M., Porat, E.: Range LCP. In: Asano, T., Nakano, S., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 683–692. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25591-5_70
2. Amir, A., Apostolico, A., Landau, G.M., Levy, A., Lewenstein, M., Porat, E.: Range LCP. J. Comput. Syst. Sci. **80**(7), 1245–1253 (2014)
3. Amir, A., Lewenstein, M., Thankachan, S.V.: Range LCP queries revisited. In: Iliopoulos, C., Puglisi, S., Yilmaz, E. (eds.) SPIRE 2015. LNCS, vol. 9309, pp. 350–361. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23826-5_33
4. Chan, T.M., Larsen, K.G., Patrascu, M.: Orthogonal range searching on the RAM, revisited. In: Symposium on Computational Geometry, pp. 1–10 (2011)
5. Chazelle, B.: A functional approach to data structures and its use in multidimensional searching. SIAM J. Comput. **17**(3), 427–462 (1988)
6. Cormode, G., Muthukrishnan, S.: Substring compression problems. In: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 321–330. Society for Industrial and Applied Mathematics (2005)
7. Farach, M., Muthukrishnan, S.: Perfect hashing for strings: formalization and algorithms. In: Hirschberg, D., Myers, G. (eds.) CPM 1996. LNCS, vol. 1075, pp. 130–140. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61258-0_11
8. Fischer, J., Heun, V.: Space-efficient preprocessing schemes for range minimum queries on static arrays. SIAM J. Comput. **40**(2), 465–492 (2011)
9. Gagie, T., Karhu, K., Navarro, G., Puglisi, S.J., Sirén, J.: Document listing on repetitive collections. In: Fischer, J., Sanders, P. (eds.) CPM 2013. LNCS, vol. 7922, pp. 107–119. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38905-4_12
10. Gawrychowski, P., Lewenstein, M., Nicholson, P.K.: Weighted ancestors in suffix trees. In: Schulz, A.S., Wagner, D. (eds.) ESA 2014. LNCS, vol. 8737, pp. 455–466. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44777-2_38
11. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**(2), 338–355 (1984)
12. Keller, O., Kopelowitz, T., Feibish, S.L., Lewenstein, M.: Generalized substring compression. Theor. Comput. Sci. **525**, 42–54 (2014)
13. Lewenstein, M.: Orthogonal range searching for text indexing. In: Brodnik, A., López-Ortiz, A., Raman, V., Viola, A. (eds.) Space-Efficient Data Structures, Streams, and Algorithms. LNCS, vol. 8066, pp. 267–302. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40273-9_18
14. Nekrich, Y., Navarro, G.: Sorted range reporting. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 271–282. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31155-0_24
15. Patil, M., Shah, R., Thankachan, S.V.: Faster range LCP queries. In: Kurland, O., Lewenstein, M., Porat, E. (eds.) SPIRE 2013. LNCS, vol. 8214, pp. 263–270. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-02432-5_29
16. Patil, M., Thankachan, S.V., Shah, R., Nekrich, Y., Vitter, J.S.: Categorical range maxima queries. In: Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2014, 22–27 June 2014, Snowbird, UT, USA, pp. 266–277 (2014)

17. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. In: Proceedings of the 13th Annual ACM Symposium on Theory of Computing, 11–13 May 1981, Milwaukee, Wisconsin, USA, pp. 114–122 (1981)
18. Weiner, P.: Linear pattern matching algorithms. In: SWAT, pp. 1–11 (1973)
19. Willard, D.E.: Log-logarithmic worst-case range queries are possible in space theta(n). Inf. Process. Lett. **17**(2), 81–84 (1983)