Non-Overlapping Indexing – Cache Obliviously

Sahar Hooshmand
Dept. of Computer Science, University of Central Florida - Orlando, USA
sahar@cs.ucf.edu

Paniz Abedin
Dept. of Computer Science, University of Central Florida - Orlando, USA
paniz@cs.ucf.edu

M. Oğuzhan Külekci
Informatics Institute, Istanbul Technical University - Turkey
kulekci@itu.edu.tr

Sharma V. Thankachan
Dept. of Computer Science, University of Central Florida - Orlando, USA
sharma.thankachan@ucf.edu

Abstract

The *non-overlapping indexing* problem is defined as follows: pre-process a given text $\mathsf{T}[1,n]$ of length n into a data structure such that whenever a pattern $P[1,p]$ comes as an input, we can efficiently report the largest set of non-overlapping occurrences of P in T. The best known solution is by Cohen and Porat [ISAAC, 2009]. Their index size is $O(n)$ words and query time is optimal $O(p+\mathsf{nocc})$, where nocc is the output size. We study this problem in the cache-oblivious model and present a new data structure of size $O(n \log n)$ words. It can answer queries in optimal $O(\frac{p}{B}+\log_B n+\frac{\mathsf{nocc}}{B})$ I/Os, where B is the block size.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Suffix Trees, Cache Oblivious, Data Structure, String Algorithms

Digital Object Identifier 10.4230/LIPIcs.CPM.2018.8

Funding Part of this work was done while the last author was visiting the third author with the TÜBİTAK-BİDEB 2221 program grant number 1059B211700766. This work has also received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 69094 in the form of student travel grant to present a preliminary version of this work in the 12th Workshop on Compression, Text and Algorithms (WCTA), 2017.

1 Introduction and Related Work

Text indexing is fundamental to many areas in Computer Science such as Information Retrieval, Bioinformatics, etc. The primary goal here is to pre-process a long text $\mathsf{T}[1,n]$ (given in advance), such that whenever a shorter pattern $P[1,p]$ comes as query, all occ occurrences (or simply, starting positions) of P in T can be reported efficiently. Such queries can be answered in optimal $O(p+\mathsf{occ})$ time using the classic *Suffix tree* data structure [14, 15]. It takes $O(n)$ words of space. In this paper, we focus on a variation of the text indexing problem, known as the *non-overlapping indexing*. Here we are interested in finding the largest set of occurrences of P in T (denote its size by nocc), such that any two (distinct) text positions in the output are separated by at least p characters. This primitive is central to data compression [2, 5]. The above task can be easily reduced to a set of geometric range

29th Annual Symposium on Combinatorial Pattern Matching (CPM 2018).
Editors: Gonzalo Navarro, David Sankoff, and Binhai Zhu; Article No. 8; pp. 8:1–8:9
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

queries, specifically $(1 + \mathsf{nocc})$ number of *orthogonal range next value* queries on the suffix array [12] of T. Although efficient, the solutions based on this approach are not optimal in terms of query time [11, 13]. The first space-efficient (linear) and optimal $O(p + \mathsf{nocc})$ time solution is due to Cohen and Porat [5]. They took an alternative strategy in which the periodicity of both text and the pattern are exploited. Subsequently, Ganguly *et al.* [9] showed that the problem can also be solved in succinct space.

Unfortunately, all the aforementioned indexes heavily relay on random access over the data structure, therefore efficient only when resides in the internal memory (usually RAM, the random access memory). To this end, we revisit this problem in the secondary memory model in the context of very large input data. Here we assume that the data (and the data structure) is too big to fit within the main memory, therefore deployed in a (much larger, but slower) secondary memory. Popular models of computation are (i) the cache-aware model and (ii) the cache-oblivious model. We now preset a brief description of both models.

In the **cache-aware model** (a.k.a. external memory model, I/O model and disk access model), introduced by Aggarwal and Vitter [1] the CPU is connected directly to an internal memory (of size M words), which is then connected to a very large external memory (disk). The disk is partitioned into blocks/pages and the size of each block is B words. The CPU can only work on data inside the internal memory. Therefore, to work on some data in the external memory, the corresponding blocks have to be transferred to internal memory. The transfer of a block from external memory to internal memory (or vice versa) is referred as an I/O operation. The operations inside the internal memory are orders of magnitude faster than the time for an I/O operation. Therefore, they are considered free, and the efficiency of an algorithm is measured in terms of the number of I/O operations. The **cache-oblivious model** is essentially the same as above, except the following key twist: M and B are unknown at the time of the design of algorithms and data structures [8, 7]. This means, if a cache-oblivious algorithm performs optimally between two levels of the memory hierarchy, then it is optimal at any level of the memory hierarchy. Lastly, cache-oblivious algorithms are usually more intricate than cache-aware algorithms.

Contribution. We present the first I/O-optimal solution for the non-overlapping indexing problem. To the best of our knowledge, this is the first of its kind over both cache-oblivious and cache-aware models of computation. The main result is summarized below.

▶ **Theorem 1.** *There exists an $O(n \log n)$ space data structure for the non-overlapping indexing problem in the cache oblivious model, where n is the length of the input text T. It can report the largest set of non-overlapping occurrences of an input pattern $P[1, p]$ in their* ***sorted order*** *in optimal $O(\frac{p}{B} + \log_B n + \frac{\mathsf{nocc}}{B})$ I/Os, where nocc is the output size.*

The main component of our index is the **suffix tree** data structure and its cache-oblivious counter part [4]. The suffix tree of T (denoted by ST) is a compact trie of all n suffixes of T. It has n leaves and at most $(n - 1)$ internal nodes (each having at least two children). Corresponding to each leaf in ST, there is a unique suffix in T. Specifically, the ith leftmost leaf ℓ_i corresponds to the ith lexicographically smallest suffix of T, denoted by $\mathsf{T}[\mathsf{SA}[i], n]$. Edges are labeled and the concatenation of edge labels on the path from root to a node u is called its path, denoted by $\mathsf{path}(u)$. The locus of a pattern P, denoted by $\mathsf{locus}(P)$ is the node closest to root, such that P is a prefix of its path. The array SA is called the **suffix array** of T. The suffix range of a pattern P, denoted by $[\mathsf{sp}(P), \mathsf{ep}(P)]$ is the range of (contiguous) leaves in the subtree of $\mathsf{locus}(P)$. Therefore, the set of occurrences of P is $\{\mathsf{SA}[i] \mid \mathsf{sp}(P) \leq i \leq \mathsf{ep}(P)\}$ and $\mathsf{ep}(P) - \mathsf{sp}(P) + 1 = \mathsf{occ}$. The suffix range can be computed in $O(p)$ time. The space is $O(n)$ words for both suffix array and suffix tree. For our problem, we maintain both the suffix tree and its cache-oblivious equivalent by Brodal and Fagerberg [4], which occupies $O(n)$ space and can compute $\mathsf{locus}(P)$ in optimal $O(p/B + \log_B n)$ I/Os. Moreover, we design a data structure for reporting occurrences in the sorted order (see Theorem 2), which may be of independent interest. We remark that all results in this paper assumes $M > B^{2+\Theta(1)}$ as in [4].

▶ **Theorem 2.** *A given text* $\mathsf{T}[1, n]$ *can be indexed in* $O(n \log n)$ *words in the cache oblivious model, such that we can report all* occ *occurrences of an input pattern* $P[1, p]$ *in their* ***sorted order*** *in optimal* $O(\frac{p}{B} + \log_B n + \frac{\mathsf{occ}}{B})$ *I/Os.*

We arrive at Theorem 1 by exploring the periodicity of query pattern. Let Q be the shortest prefix of P such that P can be written as the concatenation of $\alpha \geq 1$ copies of Q and a (possibly empty) prefix R of Q. i.e., $P = Q^\alpha R$. Then, the period of P, denoted by $\mathsf{period}(P)$ is $|Q|$. For example, $Q = cat, R = ca$ and $\alpha = 3$ when $P = catcatcatca$. The period can be computed in $O(p)$ time [6]. Note that $\mathsf{nocc} \leq \mathsf{occ} \leq (\alpha + 1)\mathsf{nocc}$.

2 An Overview of Our Non-Overlapping Indexing Framework

In this section, we present a high level description of our query algorithm with some key steps summarized as lemmas (long proofs are deferred to later sections). We maintain a suffix tree ST of T, however all pattern matching tasks are performed using its cache-oblivious counter part [4]. The structure in Theorem 2 is also maintained.

We say that the input pattern P is periodic if $\mathsf{period}(P) \leq |P|/2$ (equivalently $\alpha \geq 2$), else we say P is **aperiodic** (i.e., $\alpha = 1$). The first step of our algorithm is to verify if P is periodic or not, and we rely on the result in Lemma 3. We handle both cases separately.

▶ **Lemma 3.** *Given a pattern* $P[1, p]$ *which appears at least once in* T*, we can find if* P *is periodic or not in in* $O(p/B + \log_B n)$ *I/Os using an* $O(n \log n)$ *space structure. Also, it returns* $\mathsf{period}(P)$ *if* P *is periodic.*

2.1 Handling aperiodic case

When P is aperiodic, $\mathsf{occ} = \Theta(\mathsf{nocc})$ and we answer queries using the structure in Theorem 2 as follows. First obtain all occurrences of P in their sorted order. Then, scan them in the ascending order and do the following: report the first occurrence and report any other occurrence iff it is not overlapping with the last reported occurrence. This step can be implemented in $\mathsf{occ}/B = \Theta(\mathsf{nocc}/B)$ I/Os. Thus $O(p/B + \log_B n + \mathsf{nocc}/B)$ I/Os overall.

2.2 Handling periodic case

For periodic case, we start with the following simple observation.

▶ **Observation 4.** *If we list all the occurrences of* $P = Q^\alpha R$ *in* T *in the* ***ascending order****, we can see* ***clusters*** *of occurrences holding the following property: two consecutive occurrences*

1. *within a cluster, are exactly* $\mathsf{period}(P)$ *distance apart*
2. *not within a cluster cannot have an overlap of length* $\mathsf{period}(P)$ *or more.*

▶ **Lemma 5.** *The number of clusters, denoted by* π *is* $O(\mathsf{nocc})$*.*

Proof. Two occurrences i, j not within the same cluster overlap only if i is the last occurrence in a cluster and j is the first occurrence within the next cluster (follows from Observation 4(2)). Clearly, only one of them can be a part of the final output. Therefore, $\mathsf{nocc} \geq \pi/2$. ◀

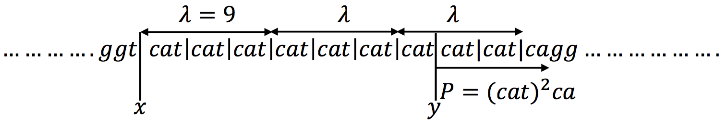
λ = 9
λ
λ
… … … … . ggt cat|cat|cat|cat|cat|cat|cat|cat|cat|cagg … … … … … … .
P = (cat)²ca
x
y

Figure 1 Here $P = catcatca$, x is the cluster-head and $y = x + 21$ is the cluster-tail. Then, the largest set of non-overlapping occurrences with the first occurrence included, and the first occurrence excluded are $\{x, x + 9, x + 18\}$ and $\{x + 3, x + 12, x + 21\}$, respectively.

Algorithm 1 Reports the largest set of non-overlapping occurrences of P in T.

```
report $S_1$
for ($i = 2$ to $\pi$) do
    if (the last reported occurrence and $L'[i]$ are non-overlapping) then report $S_i$
    else report $S_i^*$
end for
```

▶ **Definition 6.** An occurrence is a cluster-head (resp., cluster-tail) iff it is the first (resp., last) occurrence within a cluster. Also, let L' (resp., L'') be the list of all cluster heads (resp., tails) in their *ascending order*.

Observe that the distance between two consecutive non-overlapping occurrences within the same cluster, denoted by λ is $\mathsf{period}(P) \times \lceil p/\mathsf{period}(P) \rceil$. Let C_i be the ith leftmost cluster and S_i (resp., S_i^*) be the largest set of non-overlapping occurrences in C_i including (resp., excluding) the first occurrence $L'[i]$ in C_i. Specifically (see Figure 1 for an illustration),

$$S_i = \{L'[i] + k\lambda \mid \textbf{ for } k = 0, 1, 2, 3, \dots \textbf{ until } L'[i] + k\lambda \leq L''[i]\}$$

$$S_i^* = \{\mathsf{period}(P) + L'[i] + k\lambda \mid \textbf{ for } k = 0, 1, 2, 3, \dots \textbf{ until } (\mathsf{period}(P) + L'[i] + k\lambda \leq L''[i])\}$$

Then, the final output can be generated by just examining L' and L'' using the procedure in Algorithm 1. Correctness follows from Observation 4. In short, the periodic case can be handled in $O(\mathsf{nocc}/B)$ I/Os, given L' and L''. What remains to show is, how to obtain the arrays L' and L'' in optimal I/Os and we rely on the following lemmas for this crucial step.

▶ **Lemma 7.** *By maintaining an $O(n \log n)$ space structure, the array L'' corresponding to any query pattern $P[1, p]$ can be obtained in $O(p/B + \log_B n + \pi/B)$ I/Os.*

▶ **Lemma 8.** *By maintaining an $O(n \log n)$ space structure, the array L' corresponding to any query pattern $P[1, p]$ can be obtained in $O(p/B + \log_B n + \pi/B)$ I/Os.*

By combining all pieces together, we obtain $O(n \log n)$ total space and $p/B + \log_B n + \pi/B + \mathsf{nocc}/B = O(p/B + \log_B n + \mathsf{nocc}/B)$ query I/Os. This completes the proof of Theorem 1. The rest of this paper is dedicated to missing proofs.

3 Preliminaries for Missing Proofs

3.1 Heavy Path Decomposition

We first categorize the nodes in ST into light and heavy. The root is light. For each internal node u, its heaviest child is the one with the maximum number of leaves, denoted by $\mathsf{size}(\cdot)$, in its subtree, breaking ties arbitrarily. Therefore, the size of a light node is at most half of the size of its parent. Thus we have the following result.

▶ **Lemma 9** (Harel and Tarjan [10]). *The number of light nodes on any root to leaf path is at most* $(\log n)$*.*

▶ **Corollary 10.** *The sum of sub-tree sizes of all light nodes in* ST *is* $\leq n \log n$*.*

A heavy path is a downward path in the tree, starting from a light node with all other nodes on the path are heavy. Each heavy path ends at a unique leaf node. Also, each node intersect with exactly one heavy path. For brevity, we shall use the following terminologies: for any node u in ST, let

- $\mathsf{hp\_root}(u)$ be the first light node on the path from u to root. Equivalently, $\mathsf{hp\_root}(u)$ is the root of the heavy path that intersects with u.
- $\mathsf{hp\_leaf}(u) = \ell_j$, where u and ℓ_j are on the same heavy path. Note that ℓ_j is unique for u.

3.2 Right-Maximally-Periodic Prefixes

▶ **Definition 11.** We call a substring $\mathsf{T}[i, i+l-1]$ *right-maximally-periodic* iff $\mathsf{T}[i, i+l-1]$ is periodic and $\mathsf{T}[i, i+l]$ is aperiodic.

▶ **Lemma 12.** *For a fixed suffix* $\mathsf{T}[i, n]$*, let* $l_1, l_2, ..., l_k$ *be the length of all* right-maximally-periodic *prefixes in their ascending order and* $q_1, q_2, ..., q_k$ *be their respective periods. A* p*-long prefix of* $\mathsf{T}[i, n]$ *is periodic (with period* q_j*) iff* $2 \times q_j \leq p \leq l_j$ *for some* j*.*

▶ **Lemma 13.** *The number of* right-maximally-periodic *prefixes of* $\mathsf{T}[i, n]$ *is* $O(\log n)$*.*

Proof. From the definition of periodic and aperiodic, $q_j \geq l_{j-1}$ and $l_j \geq 2 \times q_j$. Therefore, $l_j \geq 2 \times l_{j-1}$ and $l_k \geq 2^{k-1} l_1$. Hence $k \leq 1 + \log(n - i + 1)$. ◀

3.3 1-Sided Sorted Range Reporting

We now prove two useful results.

▶ **Lemma 14.** *Let* $S = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$ *be a set of* m *points in 2D. A 1-Sided range sorted range reporting query "*r*" asks to return the points in* $S(r, -) = \{(x_i, y_j) \in S \mid x_i \leq r\}$ *in the sorted order of their* y*-coordinates. The query can be answered in optimal* $O(1 + k/B)$ *I/Os using an* $O(m)$ *space data structure, where* k *is the size of* $S(r, -)$*.*

Proof. Without loss of generality, assume $x_1 \leq x_2 \leq x_3 \leq ... \leq x_m$. Let $A[1, m]$ be an array of length m, such that $A[i] = x_i$. We maintain a Cache-Oblivious B-tree [3] over A, so that for any query r, we can find $k = \max\{i \mid A[i] \leq r\}$ in $O(\log_B m)$ I/Os. Let D be an array of points in the *ascending order of* x*-coordinates* (i.e., $D[i] = (x_i, y_i)$) and D^j be an array of first j-points in D in the *ascending order of* y*-coordinates.* We explicitly maintain D^j for $j = 1, 2, 4, 8, ..., m$. Total space is $O(m)$.

Now to answer a query r, first find k using a predecessor search query. Then, simply scan through the array $D^{k'}$ (in the left to right order) and report only those points (x_i, y_j) with $x_i \leq r$, where $k' = 2^{\lceil \log k \rceil}$. I/Os required is $k'/B = O(k/B)$. ◀

▶ **Lemma 15.** *Let* $S = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$ *be a set of* m *points in 2D. A 1-Sided range sorted range reporting query "*r*" asks to return the points in* $S(r, +) = \{(x_i, y_j) \in S \mid x_i \geq r\}$ *in the sorted order of their* y*-coordinates. The query can be answered in optimal* $O(1 + k/B)$ *I/Os using an* $O(m)$ *space data structure, where* k *is the size of* $S(r, +)$*.*

Proof. Maintain the data structure in Lemma 14 over the set $S^* = \{(-x_i, y_j) \mid (x_i, y_j) \in S\}$. When r comes as an input, obtain $S^*(-r, -)$ in y-sorted order and report them in the same order after replacing each point (a, b) by $(-a, b)$. ◀

4 Proof of Theorem 2

Our data structure is simple. For each light node w in the suffix tree, define a set H_w of two-dimensional points, where

$$H_w = \{(\delta(i), \mathsf{SA}[i]) \mid \ell_i \text{ is under } w\}$$

Here $\delta(i)$ is the string depth of the lowest common ancestor (lca) of ℓ_i and $\mathsf{hp\_leaf}(w)$. Note that $|H_w| = \mathsf{size}(w)$. For each light node w, we maintain a 1-sided sorted range reporting structure (in Lemma 15) over the set H_w. The total space is $O(n \log n)$ words (from Corollary 10).

To answer a query P, we first find the locus of P via searching in the cache-oblivious suffix tree in $O(p/B + \log_B n)$ I/Os [4]. The remaining task is to report the set $\{\mathsf{SA}[i] \mid \ell_i \text{ is under } \mathsf{locus}(P)\}$ with its elements sorted. Let w be the first light node on the path from $\mathsf{locus}(P)$ to the root of ST. Specifically, $w = \mathsf{hp\_root}(\mathsf{locus}(P))$. Then, the following holds.

$$\{\mathsf{SA}[i] \mid \ell_i \text{ is under } \mathsf{locus}(P)\} = \{\mathsf{SA}[i] \mid \text{ string depth of } \mathsf{lca}(\ell_i, \mathsf{hp\_leaf}(w)) \text{ is } \geq p\}$$

The remaining part of the query can be completed via a single 1-sided sorted range reporting query "p" over the set of points in H_w. The total number of I/Os required is $O(p/B + \log_B n + \mathsf{occ}/B)$.

5 Proof of Lemma 3

The design of our data structure is straightforward from the discussion in Section 3.2. For each $\mathsf{T}[i, n]$, we maintain the lengths and periods of all its right-maximally-periodic prefixes. The space required is $O(n \log n)$ words (refer to Lemma 13).

When a pattern P comes, we first find an occurrence i of P in T. Then examine lengths of all right-maximally-periodic prefixes (and their periods) of $\mathsf{T}[i, n]$ and decide if P is periodic or not in $(\log n)/B = O(\log_B n)$ I/Os. Also, retrieve its period if it is periodic.

6 Proof of Lemma 7

We use the following observation [9]: a text position y is the rightmost occurrence of P within a cluster (i.e., cluster-tail) iff $\mathsf{T}[y, n]$ is prefixed by $P = Q^\alpha R$, but not by $QP = Q^{1+\alpha}R$. This means, L'' is the sorted list of all elements in the following set of size π (see Figure 2).

$$\{\mathsf{SA}[i] \mid i \in [\mathsf{sp}(P), \mathsf{ep}(P)] \wedge i \notin [\mathsf{sp}(QP), \mathsf{ep}(QP)]\}$$

We consider the following two cases separately.

6.1 Case 1: locus(P) and locus(QP) are on different heavy paths

Here $\mathsf{hp\_root}(\mathsf{locus}(P)) \neq \mathsf{hp\_root}(\mathsf{locus}(QP))$ (we perform this check in $O(1)$ I/Os). The following lemma is the key.

▶ **Lemma 16.** *When* $\mathsf{locus}(P)$ *and* $\mathsf{locus}(QP)$ *are on different heavy paths,* $\mathsf{occ} = \Theta(\mathsf{nocc})$.

Proof. Let u be the first node on the path from $\mathsf{locus}(QP)$ to root, such that $\mathsf{locus}(P)$ and the parent of u are on the same heavy path and u' be the heavy sibling of u. Then, clearly, $\mathsf{size}(\mathsf{locus}(QP)) \leq \mathsf{size}(u) \leq \mathsf{size}(u') \leq \pi$ and $\pi + \mathsf{size}(\mathsf{locus}(QP)) = \mathsf{occ}$. Therefore, $\pi \geq \mathsf{occ}/2$ and $\pi \leq \mathsf{nocc}$, hence $\mathsf{occ} = \Theta(\mathsf{nocc})$. ◀

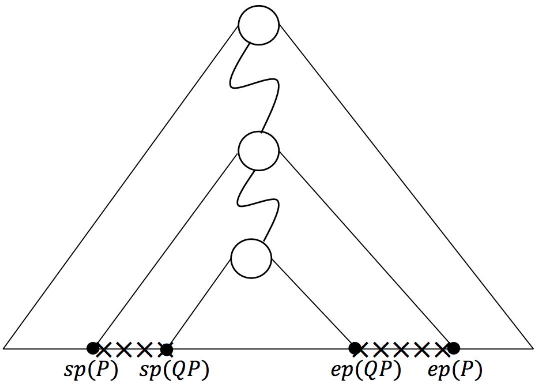

Figure 2 Suffix tree with the region corresponding to L' highlighted.

In the light of the above lemma, when the query P falls in this case, we can in fact generate the final output directly instead of generating L'' first and using it. First obtain all occurrences of P in the sorted order (using the structure in Theorem 2) and extract the largest set of non-overlapping occurrences from it by following the exact same procedure as in aperiodic case. I/Os required is $p/B + \log_B n + \mathsf{occ}/B = O(p/B + \log_B n + \mathsf{nocc}/B)$.

6.2 Case 2: locus(P) and locus(QP) are on the same heavy path

We start with two definitions and a crucial observation based on them.

▶ **Definition 17.** A pattern P is a power (or perfectly periodic) iff $P = Q^\alpha$ for an integer $\alpha \geq 2$.

For example, *aabaabaab* is a power, whereas *aabaaba* is not.

▶ **Definition 18.** We call a node in the suffix tree *special* if it is the locus of at least one power. Note that a special node can be the locus of a pattern which is not a power.

▶ **Observation 19.** *Let P be a periodic pattern with Q being its* $\mathsf{period}(P)$*-long prefix. Then, among all nodes on the path from* $\mathsf{locus}(QP)$ *to* $\mathsf{locus}(P)$*, excluding* $\mathsf{locus}(QP)$*, exactly one node is special.*

6.2.1 The Data Structure

For each light node w in the suffix tree, we maintain the following structure.

Let $v_0 = w, v_1, v_2, v_3, .., v_h$ be the special nodes on the heavy path corresponding to w (in the ascending order of pre-order rank) and let $v_{h+1} = \mathsf{hp\_leaf}(w)$ if it is not special. Define sets $G_w(v_j)$ for $j = 0, 1, 2, ..., h$, such that $G_w(v_h) = \{(\delta(i), \mathsf{SA}[i]) \mid \ell_i \text{ is under } v_h\}$ and for $j < h$,

$$G_w(v_j) = \{(\delta(i), \mathsf{SA}[i]) \mid \ell_i \text{ is under } v_j, \text{ but not under } v_{j+1}\}$$

We then maintain the 1-sided sorted range reporting structures (in Lemma 14 and Lemma 15) over the set of points in each $G_w(v_j)$. The space required for a fixed w is $O(\mathsf{size}(w))$ words. Hence, the space over all lights nodes is $O(n \log n)$.

6.2.2 The Algorithm

When P is a power, L'' can be obtained via a single 1-sided sorted range reporting query "$(p+|Q|-1)$" on the structure in Lemma 14 over the set $G_w(v_j)$, where $w = \mathsf{hp\_root}(\mathsf{locus}(P))$ and $v_j = \mathsf{locus}(P)$.

For the other case, choose $w = \mathsf{hp\_root}(\mathsf{locus}(P))$ and $v_j = \mathsf{locus}(Q^{\alpha+1})$. Then obtain elements in the following two sets, in the sorted order of $\mathsf{SA}[\cdot]$ via 1-sided sorted range reported queries.

1. $\{(\delta(i), \mathsf{SA}[i]) \mid \ell_i$ is under v_j, but not under $\mathsf{locus}(QP)\}$ via a query "$(p+|Q|-1)$" on the structure in Lemma 14 over $G_w(v_j)$.
2. $\{(\delta(i), \mathsf{SA}[i]) \mid \ell_i$ is under $\mathsf{locus}(P)$, but not under $v_j\}$ via a 1-sided sorted range reporting query "p" on the structure in Lemma 15 maintained over $G_w(v_{j-1})$.

By scanning both arrays in linear I/Os, specifically $O(\pi/B)$ I/Os, we obtain L''.

7 Proof of Lemma 8

For L', we use the following observation: for each cluster of P in T, there is a corresponding cluster of $\overleftarrow{P}$ in $\overleftarrow{\mathsf{T}}$. Here $\overleftarrow{\mathsf{T}}$ (resp., $\overleftarrow{P}$) is the reverse of T (resp.., P). Then, we have the following simple observation.

▶ **Observation 20.** *Suppose z is the last occurrence of $\overleftarrow{P}$ within a cluster of $\overleftarrow{P}$ in $\overleftarrow{T}$, then $(n+2-p-z)$ is the first occurrence of P within the corresponding cluster of P in T.*

Therefore, we simply construct and maintain our previous data structure for computing L'', but on $\overleftarrow{T}$. When P comes as input to the original problem, we find L'' corresponding to $\overleftarrow{P}$ in $\overleftarrow{T}$. Then, simply report $(n+2-p-L''[i])$'s in the descending order of i. Note that we need to maintain the suffix tree (and its cache-oblivious version) of $\overleftarrow{T}$ as well. However, the total space is still $O(n \log n)$.

8 Concluding Remarks

We present the first I/O optimal data structure for the non-overlapping indexing problem in both cache-award and cache-oblivious models of computation. We remark that by combining our framework with standard techniques, we can design an I/O optimal, $O(n \log^2 n)$ space structure for the *range* non-overlapping problem in the cache-aware model. However, it is not clear if the same is possible in cache-oblivious model. An interesting question is: *Can we improve the space (yet, keeping the query I/Os optimal), at least in the cache-award model ?*

References

1 Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988. `doi:10.1145/48529.48535`.

2 Alberto Apostolico and Franco P Preparata. Data structures and algorithms for the string statistics problem. *Algorithmica*, 15(5):481–494, 1996.

3 Michael A. Bender, Erik D. Demaine, and Martin Farach-Colton. Cache-oblivious b-trees. *SIAM J. Comput.*, 35(2):341–358, 2005. `doi:10.1137/S0097539701389956`.

4 Gerth Stølting Brodal and Rolf Fagerberg. Cache-oblivious string dictionaries. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 581–590. ACM Press, 2006. URL: `http://dl.acm.org/citation.cfm?id=1109557.1109621`.

5 Hagai Cohen and Ely Porat. Range non-overlapping indexing. In Yingfei Dong, Ding-Zhu Du, and Oscar H. Ibarra, editors, *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*, volume 5878 of *Lecture Notes in Computer Science*, pages 1044–1053. Springer, 2009. doi:10.1007/978-3-642-10631-6_105.

6 Maxime Crochemore. String-matching on ordered alphabets. *Theoretical Computer Science*, 92(1):33–47, 1992.

7 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 285–298. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814600.

8 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. *ACM Trans. Algorithms*, 8(1):4:1–4:22, 2012. doi:10.1145/2071379.2071383.

9 Arnab Ganguly, Rahul Shah, and Sharma V Thankachan. Succinct non-overlapping indexing. In *Annual Symposium on Combinatorial Pattern Matching*, pages 185–195. Springer, 2015.

10 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. doi:10.1137/0213024.

11 Orgad Keller, Tsvi Kopelowitz, and Moshe Lewenstein. Range non-overlapping indexing and successive list indexing. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Norbert Zeh, editors, *Algorithms and Data Structures, 10th International Workshop, WADS 2007, Halifax, Canada, August 15-17, 2007, Proceedings*, volume 4619 of *Lecture Notes in Computer Science*, pages 625–636. Springer, 2007. doi:10.1007/978-3-540-73951-7_54.

12 Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993. doi:10.1137/0222058.

13 Yakov Nekrich and Gonzalo Navarro. Sorted range reporting. In Fedor V. Fomin and Petteri Kaski, editors, *Algorithm Theory - SWAT 2012 - 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4-6, 2012. Proceedings*, volume 7357 of *Lecture Notes in Computer Science*, pages 271–282. Springer, 2012. doi:10.1007/978-3-642-31155-0_24.

14 Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995. doi:10.1007/BF01206331.

15 Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11. IEEE Computer Society, 1973. doi:10.1109/SWAT.1973.13.