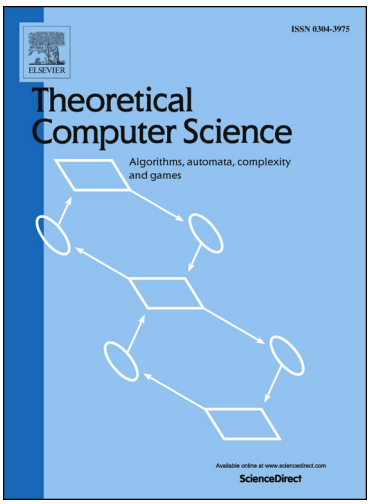
Journal Pre-proof



I/O-Efficient Data Structures for Non-Overlapping Indexing

Sahar Hooshmand, Paniz Abedin, M. Oğuzhan Külekci and Sharma V. Thankachan

PII: S0304-3975(20)30713-1
DOI: https://doi.org/10.1016/j.tcs.2020.12.006
Reference: TCS 12749

To appear in: *Theoretical Computer Science*

Received date: 26 September 2019
Revised date: 16 November 2020
Accepted date: 3 December 2020

Please cite this article as: S. Hooshmand, P. Abedin, M. Oğuzhan Külekci et al., I/O-Efficient Data Structures for Non-Overlapping Indexing, *Theoretical Computer Science*, doi: https://doi.org/10.1016/j.tcs.2020.12.006.

I/O-Efficient Data Structures for Non-Overlapping Indexing

Sahar Hooshmand[a], Paniz Abedin[a], M. Oğuzhan Külekci[b], Sharma V. Thankachan[a,*]

[a] *Dept. of Computer Science, University of Central Florida, Orlando, USA*
[b] *Informatics Institute, Istanbul Technical University, Turkey*

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 $P[1,m]$ 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]. The size of their structure is $O(n)$ words and the query time is optimal $O(m+\mathsf{nocc})$, where nocc is the output size. Later, Ganguly *et al.* [CPM 2015 and Algorithmica 2020] proposed a compressed space solution. 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 $O(\frac{m}{B} + \log_B n + \frac{\mathsf{nocc}}{B})$ I/O operations, where B is the block size. The space can be improved to $O(n \log_{M/B} n)$ in the cache-aware model, where M is the size of main memory. Additionally, we study a generalization of this problem with an additional range $[s,e]$ constraint. Here the task is to report the largest set of non-overlapping occurrences of P in T, that are within the range $[s,e]$. We present an $O(n \log^2 n)$ space data structure in the cache-aware model that can answer queries in optimal $O(\frac{m}{B} + \log_B n + \frac{\mathsf{nocc}_{[s,e]}}{B})$ I/O operations, where $\mathsf{nocc}_{[s,e]}$ is the output size.

Keywords: `Suffix Trees, Data Structure, String Algorithms.`
2010 MSC: 00-01, 99-00

*Corresponding author
Email addresses: `Sahar@cs.ucf.edu` (Sahar Hooshmand), `Paniz@cs.ucf.edu` (Paniz Abedin), `kulekci@itu.edu.tr` (M. Oğuzhan Külekci), `Sharma.thankachan@ucf.edu` (Sharma V. Thankachan)

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 preprocess a long text $\mathsf{T}[1, n]$ (given in advance), such that whenever a shorter pattern $P[1, m]$ 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 $O(m + \mathsf{occ})$ time using the classic *Suffix tree* data structure [1, 2]. 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*, which is central to data compression [3, 4].

Problem 1 (Non-overlapping Indexing)**.** *Preprocess a text* $\mathsf{T}[1, n]$ *into a data structure that supports the following query: given a pattern* $P[1, m]$*, report a 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* m *characters.*

The *range non-overlapping indexing* is a generalization of the above problem.

Problem 2 (Range Non-overlapping Indexing)**.** *Preprocess a text* $\mathsf{T}[1, n]$ *into a data structure that supports the following query: given a pattern* $P[1, m]$ *and a range* $[s, e]$*, report a largest set of occurrences of* P *in* T*, that are within the range* $[s, e]$ (*denote its size by* $\mathsf{nocc}_{[s,e]}$)*, such that any two (distinct) text positions in the output are separated by at least* m *characters.*

Both these problems are well studied in the internal memory (RAM) model of computation. The initial solutions were obtained via a reduction to the *orthogonal range next value* problem. Although efficient, they were not optimal in terms of query time [5, 6, 7]. The first non-trivial solutions were by Cohen and Porat [4]: they presented an $O(n)$ space and optimal $O(m + \mathsf{nocc})$ query time solution for Problem 1 and an $O(n \log^{\epsilon} n)$ space and near-optimal $O(m + \log \log n + \mathsf{nocc}_{[s,e]})$ query time solution for Problem 2, where $\epsilon > 0$ is an arbitrarily small constant.

To achieve these results, they exploited the periodicity of both pattern and (substrings of) text. Subsequently, Ganguly *et al.* [8, 9] presented the first succinct space solution for Problem 1 and an optimal query time solution for Problem 2. We revisit these problems 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 present a brief description of both models.

Models. In the *cache-aware model* (a.k.a. external memory model, I/O model, and disk access model), introduced by Aggarwal and Vitter [10] 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 to 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 [11, 12]. 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. Generally, we assume $M > B^{2+\Theta(1)}$, known as the tall cache assumption.

Our Results. For the non-overlapping indexing problem, we present an $O(n \log n)$ (resp., $O(n \log_{M/B} n)$) space solution in the cache-oblivious (resp., cache-aware) model. For the range non-overlapping indexing problem, we present an $O(n \log^2 n)$

space solution in the cache-aware model. In both cases, the occurrences are reported in their sorted order and the number of I/O operations is optimal.

2. Preliminaries and Basic Structures

2.1. Suffix Trees

The suffix tree data structure of $\mathsf{T}[1,n]$ (denoted by ST) is a compact trie of all n suffixes of T [2]. 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 $\mathsf{T}[\mathsf{SA}[i],n]$. The array SA is called the *suffix array* of T. Edges in ST are labeled and the concatenation of edge labels on the path from the root to a node u is called its path, denoted by $\mathsf{path}(u)$. Also, $\mathsf{size}(u)$ denote the number of leaves under u. The locus of a string S in ST, denoted by $\mathsf{locus}(S)$ is the node closest to root, such that S is a prefix of the node's path. The suffix range of a string S, denoted by $[\mathsf{sp}(S), \mathsf{ep}(S)]$ is the range of (contiguous) leaves in the subtree of $\mathsf{locus}(S)$. Therefore, the set of occurrences of S is $\{\mathsf{SA}[i] \mid \mathsf{sp}(S) \leq i \leq \mathsf{ep}(S)\}$. The suffix range of S can be computed in $O(|S|)$ time. Space is $O(n)$ words for both suffix array and suffix tree.

There also exist linear-space suffix tree representations in both secondary memory models. The solution in the cache-aware model is by Ferragina [13] and is called the String-B tree. The cache-oblivious solution is by Brodal and Fagerberg [14], which is based on an intricate concept of decomposing the tree into what they call Giraffe trees. In both cases, the locus, as well as the suffix range of any string S can be computed in optimal $O(|S|/B + \log_B n)$ I/O operations.

2.2. Sorted Range Reporting on Arrays

Definition 3. *Let $A[1,n]$ be an array of n integers. A sorted range reporting query $[s,e]$ on A asks to report the elements in the subarray $A[s,e]$ in their ascending order.*

We now present two useful results.

Lemma 4. *There exists an $O(n \log n)$ space cache-oblivious data structure that can answer sorted range reporting queries on arrays in optimal $O((e-s+1)/B)$ I/O operations.*

PROOF. Let $L(t,l)$ be the list of all $(i, A[i])$ pairs with $i \in [t, t+l-1]$ in the ascending order of $A[i]$. Maintain $L(1,l), L(1+l, 2l), L(1+2l, 3l), \ldots, L(1+l\lfloor n/l \rfloor, n)$ for $l = 2, 4, 8, \cdots, 2^{\lfloor \log n \rfloor}$. The total space is $O(n \log n)$.

To answer a query $[s,e]$, compute $l = 2^{\lfloor \log(e-s+1) \rfloor}$ and $k = l\lfloor e/l \rfloor$. Then, simply merge the two sorted lists $L(k-l+1, k)$ and $L(k+1, k+l)$ and discard those elements that are not in $A[s,e]$. The correctness follows from the fact that $k-l+1 \leq s \leq e \leq k+l$. The number of I/O operations required is $2l/B = O((e-s+1)/B)$. □

Lemma 5. *There exists an $O(n \log_{M/B} n)$ space cache-aware data structure that can answer sorted range reporting queries on arrays in optimal $O((e-s+1)/B)$ I/O operations.*

PROOF. Consider the definition of $L(t,l)$ in the proof of Lemma 4. Since M and B are known in advance, maintain $L(1,l), L(1+l, 2l), L(1+2l, 3l), \ldots, L(1+l\lfloor n/l \rfloor, n)$ with $l = (M/B)^{j/2}$ for $j = 0, 1, 2, \ldots, \lfloor \log_{M/B} n \rfloor$. The total space is $O(n \log_{M/B} n)$.

Given a query $[s,e]$, first compute l and j, such that $t = (M/B)^{j/2} \leq e-s+1 \leq (M/B)^{(j+1)/2}$. If t is small enough, say below $(M/B)^{1/2}$, then the entire subarray $A[s,e]$ can be taken to the internal memory and sorted. Otherwise, the query can be answered by merging $O((M/B)^{1/2} \log_{M/B} n)$ sorted lists of total length $\Theta(e-s+1)$. This can be implemented in optimal $O((e-s+1)/B)$ I/O operations. □

3. Non-Overlapping Indexing - Cache Obliviously

Our data structure consists of a (cache-oblivious) suffix tree ST, and a sorted range reporting structure as in Lemma 4 over the suffix array SA. Additional

structures will be introduced along the way. We start with a definition.

Definition 6 (Shortest Period)**.** *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, we denote the length of Q by* $\mathsf{period}(P)$.

We say that an input pattern P is periodic if $\mathsf{period}(P) \leq |P|/2$ (equivalently $\alpha \geq 2$), else (i.e., $\alpha = 1$), we say P is *aperiodic*. The first step of our query algorithm is to check if P is periodic or not, and we rely on the result in Lemma 7.

Lemma 7. *Given a pattern $P[1, m]$ which appears at least once in T, there exists an algorithm that finds if P is periodic or not in $O(m/B+\log_B n)$ I/O operations using an $O(n \log n)$ space structure. Also, the algorithm returns* $\mathsf{period}(P)$ *if P is periodic.*

Proof. We start with some terminologies. A substring $\mathsf{T}[i, i + l - 1]$ is *right-maximally-periodic* iff $\mathsf{T}[i, i + l - 1]$ is periodic and $\mathsf{T}[i, i + l]$ is aperiodic. Let $l_1, l_2, ..., l_k$ be the lengths of all substrings starting at position i that are *right-maximally-periodic* in their ascending order and let $q_1, q_2, ..., q_k$ be their respective $\mathsf{period}(\cdot)$'s. From the definition of periodic and aperiodic, $l_{j-1} \leq q_j$ and $2 \cdot q_j \leq l_j$, therefore $2 \cdot l_{j-1} \leq l_j$ for all $j \in [2, k]$. Also, $2^{k-1} l_1 \leq l_k \leq n - i + 1$ and the number k of *right-maximally-periodic* prefixes of $\mathsf{T}[i, n]$ can be bounded by $1 + \log(l_k/l_1) = O(\log n)$. Moreover, any substring $\mathsf{T}[i, i + m - 1]$ of length m is periodic (with period q_j) iff $2 \cdot q_j \leq m \leq l_j$ for some j.

The proof of Lemma 7 is straightforward from the discussion above. 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. When a pattern $P[1, m]$ comes, the algorithm first finds an occurrence i of P in T in $O(m/B + \log_B n)$ I/O operations. Then from the lengths of all right-maximally-periodic prefixes (and their periods) of $\mathsf{T}[i, n]$, it decides if P is periodic or not in $(\log n)/B = O(\log_B n)$ I/O operations. The algorithm also retrieves $\mathsf{period}(P)$, if P is periodic. □

3.1. Handling aperiodic case

When P is aperiodic, $\mathsf{occ} = \Theta(\mathsf{nocc})$. Therefore, obtain all occurrences of P in their 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/O operations. Thus, the total number of I/O operations in this case is $O(m/B + \log_B n + \mathsf{nocc}/B)$.

3.2. Handling periodic case

We start with the following simple observation by Ganguly *et al.* [9].

Observation 8. *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 (i) within a cluster, are exactly $\mathsf{period}(P)$ distance apart and (ii) not within a cluster cannot have an overlap of length $\mathsf{period}(P)$ or more.*

Lemma 9. *Let π be the number of clusters. Then $\pi = O(\mathsf{nocc})$.*

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 8(ii)). However, only one of them can be a part of the final output. Therefore, $\mathsf{nocc} \geq \pi/2$. □

Definition 10. *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 $\mathsf{period}(P) \cdot \lceil m/\mathsf{period}(P) \rceil$. See Fig 1.

Let C_i be the ith leftmost cluster and S_i (resp., S_i^*) be a largest set of non-overlapping occurrences in C_i including (resp., excluding) the first occurrence $L'[i]$ in C_i. Specifically,

$$S_i = \{L'[i] + k\lambda \mid \text{ for } k = 0, 1, 2, 3, .. \text{ as long as } L'[i] + k\lambda \leq L''[i]\}$$

$$S_i^* = \{\mathsf{period}(P)+L'[i]+k\lambda \mid \text{ for } k = 0, 1, 2, 3, .. \textit{ as long as } \mathsf{period}(P)+L'[i]+k\lambda \leq L''[i]\}$$

Therefore, the final output can be generated by just examining L' and L'' using the procedure in Algorithm 1. This step takes only $O(\mathsf{nocc}/B)$ I/O operations. The correctness follows from Observation 8.

What remains to show is, how to compute L' and L'' efficiently.

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

1: report S_1
2: **for** ($i = 2$ to π) **do**
3: **if** (the last reported occurrence and $L'[i]$ are non-overlapping) **then** report S_i
4: **else** report S_i^*
5: **end for**

3.3. Computing L'': The List of Cluster Tails

We use the following observation by Ganguly *et al.* [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 set $\{\mathsf{SA}[i] \mid i \in [\mathsf{sp}(P), \mathsf{ep}(P)] \wedge i \notin [\mathsf{sp}(QP), \mathsf{ep}(QP)]\}$ of size π (see Fig 2).

The first step is to compute $\mathsf{locus}(P)$ and $\mathsf{locus}(QP)$, which takes $|P|/B+|QP|/B+\log_B n = O(m/B + \log_B n)$ I/O operations. Then L'' can be obtained via two sorted range reporting queries on SA, in $O(\pi/B)$ I/O operations.

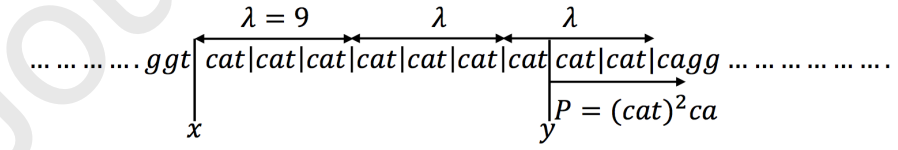


Figure 1: Here $P = catcatca$, x is the cluster-head and $y = x + 21$ is the cluster-tail. Then, the largest set of non-overlapping occurrences with the first occurrence included, and the first occurrence excluded are $\{x, x + 9, x + 18\}$ and $\{x + 3, x + 12, x + 21\}$, respectively.

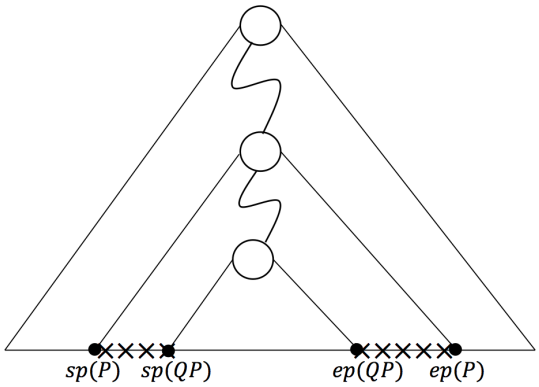


Figure 2: Highlighted are the regions corresponding to cluster tails.

3.4. Computing L': The List of Cluster Heads

Clearly, 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 observation.

Observation 11. *Let z be the last occurrence of $\overleftarrow{P}$ within a cluster of $\overleftarrow{P}$ in $\overleftarrow{T}$, then $(n+2-m-z)$ 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 $(n+2-m-L''[i])$'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. Additional space required is $O(n \log n)$.

In summary, both L' and L'' can be computed in $O(m/B + \log_B n + \pi/B)$ I/O operations using an $O(n \log n)$ space structure. The final output can be generated in additional $O(\mathsf{nocc}/B)$ I/O operations. Also, $\pi = O(\mathsf{nocc})$ from Lemma 9. Therefore, by combining everything, we have the following result.

Theorem 12. *There exists a data structure that is initialized with a text $\mathsf{T}[1,n]$, takes $O(n \log n)$ words of space and supports the following query in the cache-oblivious model. Given a string $P[1,m]$, the data structure reports the largest set of non-overlapping occurrences of $P[1,m]$ in T in their sorted order in optimal $O(\frac{m}{B} + \log_B n + \frac{\mathsf{nocc}}{B})$ I/O operations, where nocc is the output size.*

The space complexity can be improved in the cache-aware model.

Theorem 13. *There exists a data structure that is initialized with a text* $\mathsf{T}[1,n]$*, takes* $O(n \log_{M/B} n)$ *words of space and supports the following query in the cache-aware model. Given a string* $P[1,m]$*, the data structure reports the largest set of non-overlapping occurrences of* $P[1,m]$ *in* T *in their sorted order in optimal* $O(\frac{m}{B} + \log_B n + \frac{\mathsf{nocc}}{B})$ *I/O operations, where* nocc *is the output size.*

Proof. We modify our data structure for Theorem 12 as follows. Replace the sorted range reporting structure of Lemma 4 by Lemma 5. Also, discard the structure of Lemma 7, because in the cache-aware model, $\mathsf{period}(P)$ can be computed in $O(m/B)$ I/O operations [15]. □

4. Range Non-Overlapping Indexing in Cache-Aware Model

The range non-overlapping problem is to preprocess a text $\mathsf{T}[1,n]$ into a data structure that supports the following query: given a pattern $P[1,m]$ and a range $[s,e]$, report the largest set of occurrences of P in T, that are within the range $[s,e]$ (denote its size by $\mathsf{nocc}_{[s,e]}$), such that any two (distinct) text positions in the output are separated by at least m characters. The problem is a combination of the non-overlapping indexing problem and the position restricted pattern matching problem [16, 17, 18, 19, 20, 21].

When $s = 1$ or $e = n$, we call this *the one-sided range non-overlapping indexing* problem. We now proceed to present our solutions in the cache-aware model.

4.1. The Data Structure

In [22], Afshani *et al.* showed that an array $A[1,n]$ can be preprocessed into an $O(n \log n)$ space structure, such that given a query $([i,j],k)$, we can report the top-k elements in $A[i,j]$ in their sorted order in optimal $O(\log_B n + k/B)$ I/O operations. Using standard techniques, this result can be extended as follows: an array $A[1,n]$ can be preprocessed into an $O(n \log^2 n)$ space structure, such that given a query $([i,j],[s,e],k)$, we can report the top-k elements in $\{A[t] \mid t \in [i,j], A[t] \in [s,e]\}$ in sorted order in optimal number of I/O operations. The main component of data structure for range non-overlapping indexing is essentially this structure with A being the suffix array of T. As before, we handle the aperiodic and periodic cases separately.

4.2. Handling aperiodic case

Find the suffix range of P and then report all those occurrences of P that are within $[s, e]$ in their sorted order. This step can be performed in optimal I/O operations using the structure described above. 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. Total number of I/O operations required is $O(m/B + \log_B n + \mathsf{nocc}_{[s,e]}/B)$.

4.3. Handling periodic case

First, we find all cluster-heads that are within the range $[s, e]$ in their sorted order. Additionally, we find the first occurrence of P in $\mathsf{T}[s, e]$ and add to the beginning of this list and call it L'. Similarly, find all cluster-tails that are within the range $[s, e]$ in their sorted order. Additionally, we find the last occurrence of P in $\mathsf{T}[s, e]$ and add it to the end of this list and call it L''. In order to perform these steps in optimal I/O operations, we rely on our $O(n \log^2 n)$ space structure. To obtain our final answer, we simply run Algorithm 1 described in Section 3.2. The correctness and the I/O complexity can be easily verified.

Theorem 14. *There exists an $O(n \log^2 n)$ space data structure for the range non-overlapping indexing problem in the cache-aware model, where n is the length of the input text T. The data structure supports reporting the largest set of non-overlapping occurrences of an input pattern $P[1, m]$ within a given range $[s, e]$ in their sorted order in optimal $O(\frac{m}{B} + \log_B n + \frac{\mathsf{nocc}_{[s,e]}}{B})$ I/O operations, where $\mathsf{nocc}_{[s,e]}$ is the output size.*

Remark. In the case of one-sided range non-overlapping indexing, the space complexity of the result in Theorem 14 can be improved to $O(n \log n)$.

Acknowledgement

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. An early partial version of this paper appeared in *Proc. CPM 2018* [23].

References

[1] E. Ukkonen, On-line construction of suffix trees, Algorithmica 14 (3) (1995) 249–260. doi:10.1007/BF01206331.
URL http://dx.doi.org/10.1007/BF01206331

[2] P. Weiner, Linear pattern matching algorithms, in: 14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973, 1973, pp. 1–11. doi:10.1109/SWAT.1973.13.
URL http://dx.doi.org/10.1109/SWAT.1973.13

[3] A. Apostolico, F. P. Preparata, Data structures and algorithms for the string statistics problem, Algorithmica 15 (5) (1996) 481–494.

[4] H. Cohen, E. Porat, Range non-overlapping indexing, in: Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings, 2009, pp. 1044–1053. doi:10.1007/978-3-642-10631-6_105.
URL http://dx.doi.org/10.1007/978-3-642-10631-6_105

[5] M. Crochemore, C. S. Iliopoulos, M. Kubica, M. S. Rahman, G. Tischler, T. Walen, Improved algorithms for the range next value problem and applications, Theor. Comput. Sci. 434 (2012) 23–34. doi:10.1016/j.tcs.2012.02.015.
URL https://doi.org/10.1016/j.tcs.2012.02.015

[6] O. Keller, T. Kopelowitz, M. Lewenstein, Range non-overlapping indexing and successive list indexing, in: Algorithms and Data Structures, 10th International Workshop, WADS 2007, Halifax, Canada, August 15-17, 2007, Proceedings, 2007, pp. 625–636. doi:10.1007/978-3-540-73951-7_54.
URL http://dx.doi.org/10.1007/978-3-540-73951-7_54

[7] Y. Nekrich, G. Navarro, Sorted range reporting, in: Algorithm Theory - SWAT 2012 - 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4-6, 2012. Proceedings, 2012, pp. 271–282. doi:10.1007/978-3-642-31155-0_24.
URL http://dx.doi.org/10.1007/978-3-642-31155-0_24

[8] A. Ganguly, R. Shah, S. V. Thankachan, Succinct non-overlapping indexing, in: Annual Symposium on Combinatorial Pattern Matching, Springer, 2015, pp. 185–195.

[9] A. Ganguly, R. Shah, S. V. Thankachan, Succinct non-overlapping indexing, Algorithmica 82 (1) (2020) 107–117. doi:10.1007/s00453-019-00605-5.
URL https://doi.org/10.1007/s00453-019-00605-5

[10] A. Aggarwal, J. S. Vitter, The input/output complexity of sorting and related problems, Commun. ACM 31 (9) (1988) 1116–1127. doi:10.1145/48529.48535.
URL http://doi.acm.org/10.1145/48529.48535

[11] M. Frigo, C. E. Leiserson, H. Prokop, S. Ramachandran, Cache-oblivious algorithms, ACM Trans. Algorithms 8 (1) (2012) 4:1–4:22. doi:10.1145/2071379.2071383.
URL http://doi.acm.org/10.1145/2071379.2071383

[12] M. Frigo, C. E. Leiserson, H. Prokop, S. Ramachandran, Cache-oblivious algorithms, in: 40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA, 1999, pp. 285–298. doi:10.1109/SFFCS.1999.814600.
URL https://doi.org/10.1109/SFFCS.1999.814600

[13] P. Ferragina, R. Grossi, The string b-tree: A new data structure for string search in external memory and its applications, J. ACM 46 (2) (1999) 236–280. doi:10.1145/301970.301973.
URL https://doi.org/10.1145/301970.301973

[14] G. S. Brodal, R. 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, 2006, pp. 581–590.
URL http://dl.acm.org/citation.cfm?id=1109557.1109621

[15] K. Roh, M. Crochemore, C. S. Iliopoulos, K. Park, External memory algorithms for string problems, Fundam. Inform. 84 (1) (2008) 17–32.
URL http://content.iospress.com/articles/fundamenta-informaticae/fi84-1-03

[16] P. Bille, I. L. Gørtz, Substring range reporting, Algorithmica 69 (2) (2014) 384–396. doi:10.1007/s00453-012-9733-4.
URL https://doi.org/10.1007/s00453-012-9733-4

[17] S. Biswas, T. Ku, R. Shah, S. V. Thankachan, Position-restricted substring searching over small alphabets, J. Discrete Algorithms 46-47 (2017) 36–39. doi:10.1016/j.jda.2017.10.001.
URL https://doi.org/10.1016/j.jda.2017.10.001

[18] M. Crochemore, C. S. Iliopoulos, M. Kubica, M. S. Rahman, T. Walen, Improved algorithms for the range next value problem and applications, in: STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings, 2008, pp. 205–216. doi:10.4230/LIPIcs.STACS.2008.1359.
URL http://dx.doi.org/10.4230/LIPIcs.STACS.2008.1359

[19] W. Hon, R. Shah, S. V. Thankachan, J. S. Vitter, On position restricted substring searching in succinct space, J. Discrete Algorithms 17 (2012) 109–114. doi:10.1016/j.jda.2012.09.002.
URL https://doi.org/10.1016/j.jda.2012.09.002

[20] T. Kopelowitz, M. Lewenstein, E. Porat, Persistency in suffix trees with applications to string interval problems, in: R. Grossi, F. Sebastiani, F. Silvestri (Eds.), String Processing and Information Retrieval, 18th International Symposium, SPIRE 2011, Pisa, Italy, October 17-21, 2011. Proceedings, Vol. 7024 of Lecture Notes in Computer Science, Springer, 2011, pp. 67–80. doi:10.1007/978-3-642-24583-1_8.
URL https://doi.org/10.1007/978-3-642-24583-1_8

[21] V. Mäkinen, G. Navarro, Position-restricted substring searching, in: J. R. Correa, A. Hevia, M. A. Kiwi (Eds.), LATIN 2006: Theoretical Informatics, 7th Latin American Symposium, Valdivia, Chile, March 20-24, 2006, Proceedings, Vol. 3887 of Lecture Notes in Computer Science, Springer, 2006, pp. 703–714. doi:10.1007/11682462_64.
URL https://doi.org/10.1007/11682462_64

[22] P. Afshani, G. S. Brodal, N. Zeh, Ordered and unordered top-k range reporting in large data sets, in: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011, 2011, pp. 390–400. doi:10.1137/1.9781611973082.31.
URL https://doi.org/10.1137/1.9781611973082.31

[23] S. Hooshmand, P. Abedin, M. O. Külekci, S. V. Thankachan, Non-overlapping indexing - cache obliviously, in: Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China, 2018, pp. 8:1–8:9. doi:10.4230/LIPIcs.CPM.2018.8.
URL https://doi.org/10.4230/LIPIcs.CPM.2018.8

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: