# Forbidden Extension Queries

**Sudip Biswas**[1], **Arnab Ganguly**[1], **Rahul Shah**[1,2], **and Sharma V. Thankachan**[3]

1 **Louisiana State University, Baton Rouge, USA**
`{sbiswa7,agangu4}@lsu.edu,rahul@csc.lsu.edu`
2 **National Science Foundation, USA**
`rshah@nsf.gov`
3 **Georgia Institute of Technology, Atlanta, USA**
`sharma.thankachan@gatech.edu`

## Abstract

*Document retrieval* is one of the most fundamental problem in information retrieval. The objective is to retrieve all documents from a document collection that are relevant to an input pattern. Several variations of this problem such as ranked document retrieval, document listing with two patterns and forbidden patterns have been studied. We introduce the problem of document retrieval with forbidden extensions.

Let $\mathcal{D} = \{\mathsf{T}_1, \mathsf{T}_2, \ldots, \mathsf{T}_D\}$ be a collection of $D$ string documents of $n$ characters in total, and $P^+$ and $P^-$ be two query patterns, where $P^+$ is a proper prefix of $P^-$. We call $P^-$ as the forbidden extension of the included pattern $P^+$. A forbidden extension query $\langle P^+, P^- \rangle$ asks to report all *occ* documents in $\mathcal{D}$ that contains $P^+$ as a substring, but does not contain $P^-$ as one. A top-$k$ forbidden extension query $\langle P^+, P^-, k \rangle$ asks to report those $k$ documents among the *occ* documents that are most relevant to $P^+$. We present a linear index (in words) with an $O(|P^-| + occ)$ query time for the document listing problem. For the top-$k$ version of the problem, we achieve the following results, when the relevance of a document is based on PageRank:

- an $O(n)$ space (in words) index with $O(|P^-| \log \sigma + k)$ query time, where $\sigma$ is the size of the alphabet from which characters in $\mathcal{D}$ are chosen. For constant alphabets, this yields an optimal query time of $O(|P^-| + k)$.
- for any constant $\epsilon > 0$, a $|\mathsf{CSA}| + |\mathsf{CSA}^*| + D \log \frac{n}{D} + O(n)$ bits index with $O(\mathsf{search}(P) + k \cdot \mathsf{t_{SA}} \cdot \log^{2+\epsilon} n)$ query time, where $\mathsf{search}(P)$ is the time to find the suffix range of a pattern $P$, $\mathsf{t_{SA}}$ is the time to find suffix (or inverse suffix) array value, and $|\mathsf{CSA}^*|$ denotes the maximum of the space needed to store the *compressed suffix array* CSA of the concatenated text of all documents, or the total space needed to store the individual CSA of each document.

**1998 ACM Subject Classification** F.2.2 Pattern Matching

**Keywords and phrases** document retrieval, suffix trees, range queries, succinct data structure

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2015.320

## 1 Introduction and Related Work

Retrieving useful information from massive textual data is a core problem in information retrieval. *Document listing*, a natural formulation of this problem, has exciting applications in search engines, bioinformatics, data and Web mining. The task is to index a given collection of strings or documents, such that the relevant documents for an input query can be retrieved efficiently. More formally, let $\mathcal{D}$ be a given collection of $D$ string documents of total size $n$ characters. Given a query pattern $P$, document listing is to report all the documents that contain $P$ as a substring. The problem was introduced by Matias et al. [19]. Later, Muthukrishnan [22] proposed a linear space index with optimal query time of $O(|P| + occ)$, where $occ$ is the number of documents reported. Following this, several variations were introduced. Hon et al. [13] proposed the top-$k$ variation i.e., retrieve the $k$ documents that are most relevant to $P$ for some integer $k$ provided at query time They presented a linear index with $O(|P|+k \log k)$ time. Later this was improved to optimal $O(|P|+k)$ time [14, 23]. Compressed indexes have also been proposed for this variation [14, 17, 21, 25].

Most of the earlier document retrieval problems focus on the case where the query consists of a single pattern $P$. Often the queries are not so simplistic. Muthukrishnan [22] also considered the case of two patterns, say $P$ and $Q$, and showed that by maintaining an $O(n^{3/2} \log^{O(1)} n)$ space index, documents containing both $P$ and $Q$ can be reported in $O(|P| + |Q| + \sqrt{n} + occ)$ time. Cohen and Porat [4] presented an $O(n \log n)$ space (in words) index with query time $O(|P| + |Q| + \sqrt{n \cdot occ} \log^{5/2} n)$, which was improved by Hon et al. [15] to an $O(n)$ space index with query time $O(|P| + |Q| + \sqrt{n \cdot occ} \log^{3/2} n)$. Also see [14, 13] for a succinct solution and [18] for a recent result on the hardness of this problem.

Fischer et al. [5] introduced the *document listing with forbidden pattern problem* which consists of two patterns $P$ and $Q$, and all documents containing $P$ but not $Q$ are to be reported. They presented an $O(n^{3/2})$ bit solution with query time $O(|P| + |Q| + \sqrt{n} + occ)$. Hon et al. [16] presented an $O(n)$ word index with query time $O(|P|+|Q|+\sqrt{n \cdot occ} \log^{5/2} n)$. Later, Biswas et al. [1] offered linear space (in words) and $O(|P| + |Q| + \sqrt{nk})$ query time solution for the more general top-$k$ version of the problem, which yields a linear space and $O(|P|+|Q|+\sqrt{n \cdot occ})$ solution to the listing problem. They also showed that this is optimal via a reduction from the *set intersection/difference* problem.

In this paper, we introduce the *document listing with forbidden extension problem*, which is a stricter version of the forbidden pattern problem, and asks to report all documents containing an included pattern $P^+$, but not its forbidden extension $P^-$, where $P^+$ is a proper prefix of $P^-$. As shown by Biswas et al. [1], the forbidden pattern problem of Fischer et al. [5] suffers from the drawback that linear space (in words) solutions are unlikely to yield a solution better than $O(\sqrt{n/occ})$ per document reporting time. Thus, it is of theoretical interest to see whether this hardness can be alleviated by putting further restrictions on the forbidden pattern. We show that indeed in case when the forbidden pattern is an extension of the included pattern, by maintaining a linear space index, the document listing problem can be answered in optimal $O(|P^-| + occ)$ time. For further theoretical interest, we study the following more general top-$k$ variant.

▶ **Problem 1** (top-$k$ Document Listing with Forbidden Extension). *Let $\mathcal{D} = \mathsf{T}_1, \mathsf{T}_2, \ldots, \mathsf{T}_D$ be $D$ weighted strings (called documents) of $n$ characters in total, where each character is chosen from an alphabet of size $\sigma$. Our task is to index $\mathcal{D}$ such that when a pattern $P^+$, its extension $P^-$, and an integer $k$ come as a query, among all documents containing $P^+$, but not $P^-$, we can report the $k$ most weighted ones.*

**Results.** Our contributions to Problem 1 are summarized in the following theorems.

▶ **Theorem 1.** *The* top-$k$ *forbidden extension queries can be answered by maintaining an $O(n)$-words index in $O(|P^-| \log \sigma + k)$ time.*

▶ **Theorem 2.** *Let* CSA *be a* compressed suffix array *on $\mathcal{D}$ of size $|\mathsf{CSA}|$ bits using which we can find the suffix range of a pattern $P$ in $\mathsf{search}(P)$ time, and suffix (or inverse suffix) array value in $\mathsf{t_{SA}}$ time. Also, denote by $|\mathsf{CSA}^*|$ the maximum of the space needed to store the* compressed suffix array CSA *of the concatenated text of all documents, or the total space needed to store the individual* CSA *of each document. By maintaining* CSA *and additional*

*$|\mathsf{CSA}^*| + D \log \frac{n}{D} + O(n)$ bits structure, we can answer* $\mathsf{top}$-$k$ *forbidden extension queries in* $O(\mathsf{search}(P^-) + k \cdot \mathsf{t_{SA}} \cdot \log^{2+\epsilon} n)$ *time*

The rest of the paper is organized as follows. In Section 2, we briefly discuss standard data-structures, and terminologies. In Section 3, we present a linear index and arrive at Theorem 1. In Section 4, we present a succinct space index and arrive at Theorem 2. Finally, we conclude the paper in Section 5.

## 2 Preliminaries

We refer the reader to [11] for standard definitions and terminologies. We assume the Word-RAM model of computation, where the word size is $\omega = \Theta(\log n)$. Throughout this paper, $\mathcal{D} = \{\mathsf{T}_1, \mathsf{T}_2, \ldots, \mathsf{T}_D\}$ is a collection of $D$ documents of total size $n$ characters, where each character is chosen from an alphabet of size $\sigma$. Each document in $\mathcal{D}$ has a special terminating character that does not appear anywhere in the document. Furthermore, we assume that the PageRank of a document $\mathsf{T}_d$ is $d$, and $\mathsf{T}_d$ is more relevant than $\mathsf{T}_{d'}$ iff $d < d'$.

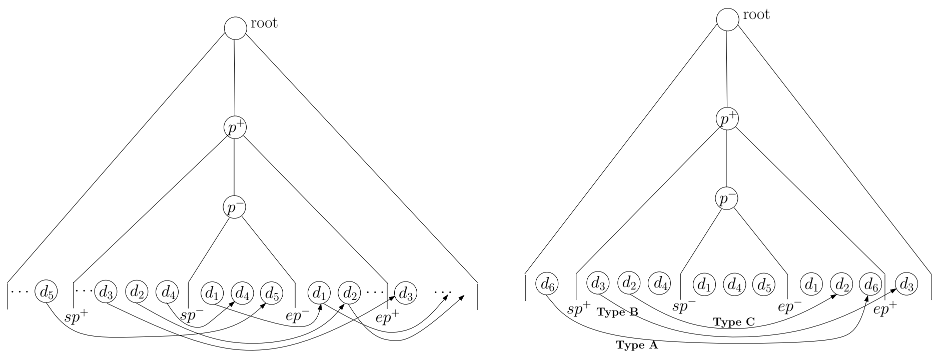
The *generalized suffix tree* $\mathsf{GST}$ is a compacted trie that stores all (non-empty) suffixes of every string in $\mathcal{D}$. The $\mathsf{GST}$ consists of $n$ leaves and at most $n-1$ internal nodes. We use $\ell_i$ to denote the $i$th leftmost leaf of $\mathsf{GST}$ i.e., the leaf corresponding to the $i$th lexicographically smallest suffix of the concatenated text $\mathsf{T}$ of every document. Further, $\mathsf{doc}(i)$ denotes the index of the document to which the suffix corresponding to $\ell_i$ belongs. Let $\mathsf{GST}(u)$ be the sub-tree of $\mathsf{GST}$ rooted at $u$, and $\mathsf{leaf}(u)$ be the set of leaves in $\mathsf{GST}(u)$. We use $\mathsf{leaf}(u, v)$ to denote the leaves in $\mathsf{GST}(u)$ but not in $\mathsf{GST}(v)$. The number of nodes (resp. concatenation of edge labels) on the path from root to a node $u$ is denoted by $\mathsf{depth}(u)$ (resp. $\mathsf{path}(u)$). The *locus* of $P$, denoted by $\mathsf{locus}(P)$, is the highest node $u$ such that $\mathsf{path}(u)$ is prefixed by $P$. Then, the *suffix range* of $P$ is $[L_u, R_u]$, where $L_u$ and $R_u$ are the leftmost and the rightmost leaves in $\mathsf{GST}(u)$. By maintaining $\mathsf{GST}$ of $\mathcal{D}$ in $O(n)$ words of space, the locus of any pattern $P$ can be computed in $O(|P|)$ time, where $|P|$ is the length of $P$. In general, suffix trees arrays require $O(n)$ words for storage. *Compressed Suffix Array* reduces this space close to the size of the text with a slowdown in query time.

Let $u$ and $v$ be any two nodes in $\mathsf{GST}$. Then $\mathsf{list}_k(u, v)$ is the set of $k$ most relevant document identifiers in $\mathsf{list}(u, v) = \{\mathsf{doc}(i) \mid \ell_i \in \mathsf{leaf}(u)\} \setminus \{\mathsf{doc}(i) \mid \ell_i \in \mathsf{leaf}(v)\}$. Any superset of $\mathsf{list}_k(u, v)$ is called a *$k$-candidate set* and is denoted by $\mathsf{cand}_k(u, v)$. Given $\mathsf{cand}_k(u, v)$, we can find $\mathsf{list}_k(u, v)$ in time $O(|\mathsf{cand}_k(u, v)|)$ using order statistics [1].

Moving forward, we use $\mathsf{CSA}$ to denote a compressed suffix array for $\mathcal{D}$ that occupies $|\mathsf{CSA}|$ bits. Using $\mathsf{CSA}$, we can find the suffix range of $P$ in $\mathsf{search}(P)$ time, and can compute a suffix array value (i.e., the text position of the suffix corresponding to a leaf) or inverse suffix array value (i.e., the lexicographic rank of a suffix) in $\mathsf{t_{SA}}$ time. Also, $p^+$ and $p^-$ (resp. $[sp^+, ep^+]$ and $[sp^-, ep^-]$) denotes the loci (resp. suffix ranges) of $P^+$ and $P^-$ respectively. Since $P^-$ is an extension of $P^+$, $p^- \in \mathsf{GST}(p^+)$, $\mathsf{leaf}(p^-) \subseteq \mathsf{leaf}(p^+)$, and $sp^+ \le sp^- \le ep^- \le ep^+$.

## 3 Linear Space Index

In this section, we present our linear space index. We use some well-known range reporting data-structures [2, 24, 26] and the chaining framework of Muthukrishnan [14, 22], which has been extensively used in problems related to document listing.Using these data structures, we first present a solution to the document listing problem. Then, we present a simple linear index for the $\mathsf{top}$-$k$ version of the problem, with a $O(|P^-| \log n + k \log n)$ query time. Using


root
$p^+$
$p^-$
$d_5$ $d_3$ $d_2$ $d_4$ $d_1$ $d_4$ $d_5$ $d_1$ $d_2$ $d_3$
$sp^+$
$sp^-$
$ep^-$
$ep^+$
root
$p^+$
$p^-$
$d_6$ $d_3$ $d_2$ $d_4$ $d_1$ $d_4$ $d_5$ $d_1$ $d_2$ $d_6$ $d_3$
$sp^+$
Type B
$sp^-$
Type C
$ep^-$
$ep^+$
Type A

**(a)** Chaining structure. **(b)** Types of chains.

**Figure 1** Chaining framework. Although $\mathsf{leaf}(p^+)$ has documents $d_1$, $d_2$, $d_3$, $d_4$, and $d_5$, only $d_2$ and $d_3$ qualify as output, since $d_1$, $d_4$, and $d_5$ are present in $\mathsf{leaf}(p^-)$.

more complicated techniques, based on the heavy path decomposition of a tree, we improve this to arrive at Theorem 1.

## Orthogonal Range Reporting Data Structure.

▶ **Fact 1** ([24]). *A set of $n$ weighted points on an $n \times n$ grid can be indexed in $O(n)$ words of space, such that for any $k \geq 1$, $h \leq n$ and $1 \leq a \leq b \leq n$, we can report $k$ most weighted points in the range $[a, b] \times [0, h]$ in decreasing order of their weights in $O(h + k)$ time.*

▶ **Fact 2** ([26]). *A set of $n$ 3-dimensional points $(x, y, z)$ can be stored in an $O(n)$-word data structure, such that we can answer a three-dimensional dominance query in $O(\log n + output)$ time, with outputs reported in the sorted order of $z$-coordinate.*

▶ **Fact 3** ([2]). *Let $A$ be an array of length $n$. By maintaining an $O(n)$-words index, given two integers $i, j$, where $j \geq i$, and a positive integer $k$, in $O(k)$ time, we can find the $k$ largest (or, smallest) elements in the subarray $A[i..j]$ in sorted order.*

**Chaining Framework.** For every leaf $\ell_i$ in $\mathsf{GST}$, we define $\mathsf{next}(i)$ as the minimum index $j > i$, such that $\mathsf{doc}(j) = \mathsf{doc}(i)$. We denote $i$ as the source of the chain and $\mathsf{next}(i)$ as the destination of the chain. We denote by $(-\infty, i)$ (resp. $(i, \infty)$) the chain that ends (resp. starts) at the first (resp. last) occurrence $\ell_i$ of a document. Figure 1(a) illustrates chaining.

The integral part of our solution involves categorizing the chains into the following 3 types, and then build separate data structure for each type.
**Type A:** $i < sp^+$ and $ep^- < \mathsf{next}(i) \leq ep^+$
**Type B:** $sp^+ \leq i < sp^-$ and $\mathsf{next}(i) > ep^+$
**Type C:** $sp^+ \leq i < sp^-$ and $ep^- < \mathsf{next}(i) \leq ep^+$
Figure 1(b) illustrates different types of chains. It is easy to see that any output of forbidden extension query falls in one of these 3 types. Also observe that the number of chains is $n$. For a type A chain $(i, \mathsf{next}(i))$, we refer to the leaves $\ell_i$ and $\ell_{\mathsf{next}(i)}$ as type A leaves; similar remarks hold for type B and type C chains. Also, LCA of a chain $(i, j)$ refers to the LCA of the leaves $\ell_i$ and $\ell_j$. Furthermore, with slight abuse of notation, for any two nodes $u, v \in \mathsf{GST}$, we denote by $\mathsf{depth}(u, v)$, the depth of the LCA of the nodes $u$ and $v$.

**Document Listing Index.** Linear space index for the forbidden extension document listing problem is achieved by using Fact 3. We store two arrays as defined below.
$\boldsymbol{A_{src}}$: $A_{src}[i]=\mathsf{next}(i)$, for each chain $(i, \mathsf{next}(i))$
$\boldsymbol{A_{dest}}$: $A_{dest}[\mathsf{next}(i)]=i$, for each chain $(i, \mathsf{next}(i))$
Querying in $A_{src}$ within the range $[sp^+, sp^- - 1]$ will give us the chains in descending order of their destination, we stop at $ep^-$ to obtain all the Type B and Type C chains. We query in $A_{dest}$ within the range $[ep^- + 1, ep^+]$ to obtain the chains in ascending order of their source and stop at $sp^+$ to obtain all the type A chains. Time, in addition to that required for finding the suffix ranges, can be bounded by $O(|P^-| + occ)$.

## 3.1 A Simple $O(|P^-|\log n + k\log n)$ time Index

We start with a simple indexing scheme for answering top-$k$ forbidden extension query. In this section, we design data structures by processing different types of chains separately and mapping them into range reporting problem.

**Processing Type A and Type B Chains.** For type A chains, we construct range reporting data structure, as described in Fact 1, with each chain $(i, j)$, $j = \mathsf{next}(i)$, mapped to a weighted two dimensional point $(j, \mathsf{depth}(i, j))$ with weight $\mathsf{doc}(i)$. Likewise, for type B chains, we map chain $(i, j)$ to the point $(i, \mathsf{depth}(i, j))$ with weight $\mathsf{doc}(i)$. Recall that $d$ is the PageRank of the document $\mathsf{T}_d$. For Type A chains, we issue a range reporting query for $[ep^- + 1, ep^+] \times [0, \mathsf{depth}(p^+)]$. For Type B chains, we issue a range reporting query for $[sp^+, sp^- - 1] \times [0, \mathsf{depth}(p^+)]$. In either case, we can obtain the top-$k$ leaves in sorted order of their weights in $O(|P^-| + k)$ time, which gives us the following lemma.

▶ **Lemma 3.** *There exists an $O(n)$ words data structure, such that for a* top-$k$ *forbidden extension query, we can report the* top-$k$ *Type A and Type B leaves in time $O(|P^-| + k)$.*

**Processing Type C Chains.** We maintain the 3-dimensional dominance structure of Fact 2 at each node of $\mathsf{GST}$. For a chain $(i, j)$, $j = \mathsf{next}(i)$, we store the point $(i, j, \mathsf{doc}(i))$ in the dominance structure maintained in the node $\mathsf{lca}(i, j)$. For query answering, we traverse the path from $p^+$ to $p^-$, and query the dominance structure of each node on this path with $x$-range $[-\infty, sp^- - 1]$ and $y$-range $[ep^- + 1, \infty]$. Any chain falling completely outside of $\mathsf{GST}(p^+)$ will not be captured by the query, since their LCA lies above $p^+$. There can be at most $\mathsf{depth}(p^-) - \mathsf{depth}(p^+) + 1 \le |P^-| = \Theta(n)$ sorted lists containing $k$ elements each. The $\log n$ factor in the query of Fact 2 is due to locating the first element to be extracted; each of the remaining $(k-1)$ elements can be extracted in constant time per element. Therefore, time required for dominance queries (without extracting the elements) is bounded by $O(|P^-|\log n)$. Using a max-heap of size $O(n)$, we obtain the top-$k$ points from all the lists as follows: insert the top element from each list into the heap, and extract the maximum element from the heap. Then, the next element from the list corresponding to the extracted element is inserted into the heap. Clearly, after extracting $k$ elements, the desired top-$k$ identifiers are obtained. Time required is $O(k\log n)$, which gives the following lemma.

▶ **Lemma 4.** *There exists a $O(n)$ words space data-structure for answering* top-$k$ *documents with forbidden extension queries in $O(|P^-|\log n + k\log n)$ time.*

## 3.2 $O(|P^-|\log\sigma + k)$ Index

In this section, we prove Theorem 1. Note that type A and type B chains can be processed in $O(|P^-| + k)$ time by maintaining separate range reporting data structures (refer to

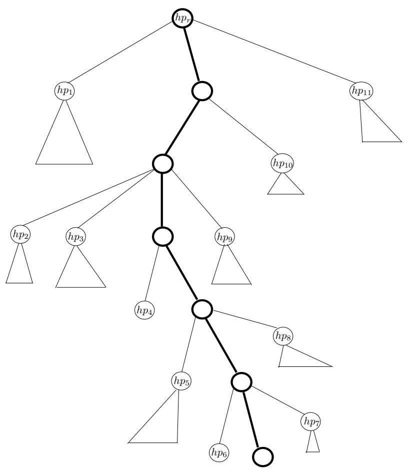
Section 3.1). Therefore, in what follows, the emphasis is to obtain type C outputs. Recall that for processing type C chains in Section 3.1, we traversed the path from $p^+$ to $p^-$, and query the individual data structure at each node. Our idea for more efficient solution is to group together the data structures of the nodes falling on the same heavy path.

**Heavy Path Decomposition.** We revisit the heavy path decomposition of a tree $T$, proposed by Harel et al. [12]. For any internal node $u$, the heaviest child of $u$ is the one having the maximum number of leaves in its subtree (ties broken arbitrarily). The first heavy path of $T$ is the path starting at $T$'s root, and traversing through every heavy node to a leaf. Each off-path subtree of the first heavy path is further decomposed recursively. Thus, a tree with $m$ leaves has $m$ heavy paths. With slight abuse of notation, let $\mathsf{leaf}(hp_i)$ be the leaf where heavy path $hp_i$ ends. Let $v$ be a node on a heavy path and $u$ be a child of $v$ not on that heavy path. We say that the subtree rooted at $u$ *hangs from node* $v$.

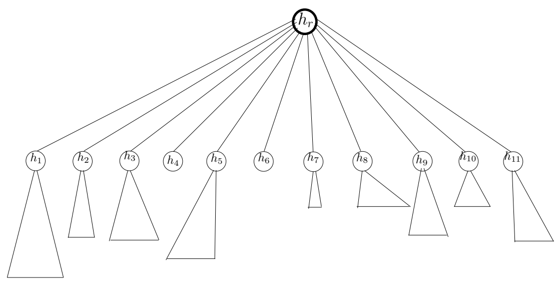
▶ **Property 1.** *For a tree having $m$ nodes, the path from the root to any node $v$ traverses at most $\log m$ heavy paths.*

**Heavy Path Tree.** We construct the heavy path tree $T_H$, in which each node corresponds to a distinct heavy path in $\mathsf{GST}$. The tree $T_H$ has $n$ nodes as there are so many heavy paths in $\mathsf{GST}$. For a heavy path $hp_i$ of $\mathsf{GST}$, the corresponding node in $T_H$ is denoted by $h_i$. All the heavy paths hanging from $hp_i$ in $\mathsf{GST}$ are the children of $h_i$ in $T_H$. Let the first heavy path in the heavy path decomposition of $\mathsf{GST}$ be $hp_r$, and $T_1, T_2, \ldots,$ be the subtrees hanging from $hp_r$. The heavy path tree $T_H$ is recursively defined as the tree whose root is $h_r$, representing $hp_r$, having children $h_1, h_2, \ldots$ with subtrees in $T_H$ resulting from the heavy path decomposition of $T_1, T_2, \ldots$ respectively. Figure 2 illustrates heavy path decomposition of $\mathsf{GST}$ and the heavy path tree $T_H$. Based on the position of a hanging heavy path w.r.t. $hp_i$ in $\mathsf{GST}$, we divide the children of $h_i$ into two groups: left children $h_i^l$ and right children $h_i^r$. A child heavy path $h_j$ of $h_i$ belongs to $h_i^l$ (resp. $h_i^r$) if $\mathsf{leaf}(hp_j)$ falls on the left (resp. right) of $\mathsf{leaf}(hp_i)$ in $\mathsf{GST}$. The nodes in $h_i^l$ and $h_i^r$ are stored contiguously in $T_H$. We traverse the left attached heavy paths of $hp_i$ in $\mathsf{GST}$ in top-to-bottom order, include them as the nodes of $h_i^l$, and place them in left-to-right order as children of $h_i$ in $T_H$. The $h_i^r$ nodes are obtained by traversing the right attached heavy paths of $hp_i$ in $\mathsf{GST}$ in bottom-to-top order, and place them after the $h_i^l$ nodes in $T_H$ in left-to-right order.

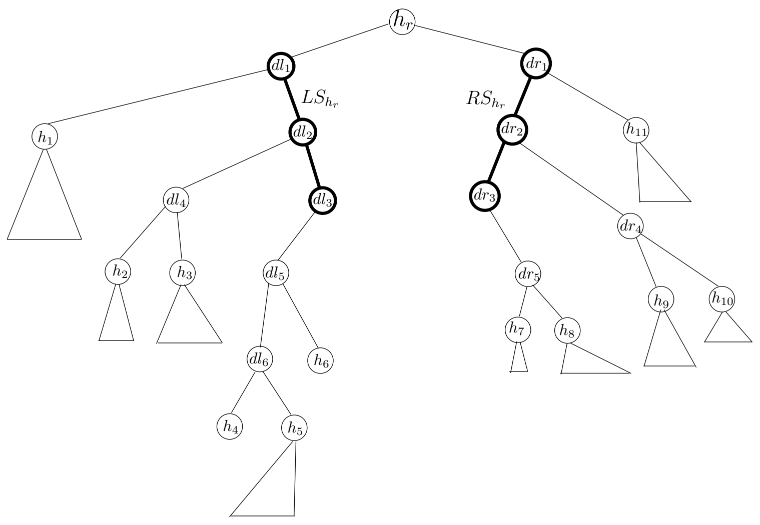
**Transformed Heavy Path Tree.** We transform the heavy path tree $T_H$ into a binary search tree $T_H^t$. For each node $h_i$ in $T_H$, we construct a left (resp. right) binary tree $BT_{h_i^l}$ (resp. $BT_{h_i^r}$) for the left children $h_i^l$ (resp. right children $h_i^r$). Leaves of $BT_{h_i^l}$ (resp. $BT_{h_i^r}$) are the nodes of $h_i^l$ (resp. $h_i^r$) preserving the ordering in $T_H$. The binary tree $BT_{h_i^l}$ (resp. $BT_{h_i^r}$) has a path, named left spine (resp. right spine), denoted by $LS_{h_i}$ (resp. $RS_{h_i}$) containing $\lfloor \log |h_i^l| \rfloor$ (resp. $\lfloor \log |h_i^r| \rfloor$) nodes, denoted by $dl_1, dl_2, \ldots$ (resp. $dr_1, dr_2, \ldots$) in the top-to-bottom order. The right child of $dl_i$ is $dl_{i+1}$. Left subtree of $dl_i$ is a height balanced binary search tree containing $h_{2^{i-1}}, \ldots, h_{\lfloor 2^i - 1 \rfloor}$ as the leaves and dummy nodes for binarization. Right spine is constructed in similar way, however left child of $dr_i$ is $dr_{i+1}$ and left subtree contains the leaves of $h_i^r$ in a height balanced binary tree. Clearly, the length of $LS_{h_i}$ (resp. $RS_{h_i}$) is bounded by $\lfloor \log |h_i^l| \rfloor$ (resp. $\lfloor \log |h_i^r| \rfloor$). Subtrees hanging from the nodes of $h_i^l$ and $h_i^r$ are decomposed recursively. See Figure 2(c) for illustration. We have the following important property of $T_H^t$.



**(a)** Heavy path decomposition of GST.



**(b)** Heavy path tree $T_H$.



**(c)** Transformed heavy path tree $T_H^t$.

**Figure 2** Heavy path decomposition, heavy path tree, and transformed heavy path tree.

▶ **Lemma 5.** *Let $u$ be an ancestor node of $v$ in* GST*. The path length from $u$ to $v$ is $d_{uv}$. The node $u$ (resp. $v$) falls on the heavy path $hp_1$ (resp. $hp_t$) and let $h_1$ (resp. $h_t$) be the corresponding node in $T_H^t$. Then, the $h_1$ to $h_t$ path in $T_H^t$ has $O(\min(d_{uv} \log \sigma, \log^2 n))$ nodes, where $\sigma$ is the size of the alphabet from which characters in the documents are chosen.*

**Proof.** We first recall from Property 1 that the height of $T_H$ is $O(\log n)$. Since each node in $T_H$ can have at most $n$ children, each level of $T_H$ can contribute to $O(\log n)$ height in $T_H^t$. Thus, the height of $T_H^t$ is bounded by $O(\log^2 n)$. Hence, the $\log^2 n$ bound in the lemma is immediate. Let $p_1, p_2, \ldots, p_t$ be the segments of the path from $u$ to $v$ traversing heavy paths $hp_1, hp_2, \ldots, hp_t$, where $p_i \in hp_i, 1 \leq i \leq t$. Let $h_1, h_2, \ldots, h_t$ be the corresponding nodes in $T_H^t$. We show that the number of nodes traversed to reach from $h_i$ to $h_{i+1}$ in $T_H^t$ is $O(|p_i| \log \sigma)$. Without loss of generality, assume $h_{i+1}$ is attached on the left of of $h_i$ and falls in the subtree attached with $dl_x$ on spine $LS_{h_i}$. We can skip all the subtrees attached to the nodes above $dl_x$ on $LS_{h_i}$. One node on a heavy path can have at most $\sigma$ heavy paths as children. Thus, the number of nodes traversed on the spine is $O(|p_i| \log \sigma)$. Within the subtree of the $dl_x$, we can search the tree to find the desired heavy path node. Since, each node in the GST can have at most $\sigma$ heavy paths as children, the height of this subtree is bounded by $O(\log \sigma)$. For each $p_i$, we may need to traverse the entire tree height to locate the desired heavy path, and hence the lemma follows. ◀

**Associating the chains.** Let $hp_i$ (resp. $hp_j$) be the heavy path having $i$ (resp. $j$) as the leaf node in GST and $h_i$ (resp. $h_j$) as the corresponding heavy path node in $T_H^t$. Then, we *associate* chain $(i, j)$ with $\mathsf{lca}(h_i, h_j)$ in $T_H^t$.

**Constructing the Index.** Our index consists of two components, maximum chain depth structure (MDS) and transformed heavy path structure (THS) defined as follows.

**MDS component:** Let $hp_t$ be the heavy path in the original heavy path decomposition (i.e., not a dummy heavy path), associated with chain $(i, j)$, $j = \mathsf{next}(i)$. Let, $d_i = \mathsf{depth}(i, \mathsf{leaf}(hp_t))$ and $d_j = \mathsf{depth}(j, \mathsf{leaf}(hp_t))$. Define $\mathsf{maxDepth}(i, j) = \max(d_i, d_j)$. Let $m_t$ be the number of chains associated with $hp_t$. Create two arrays $A_t$ and $A'_t$, each of length $m_t$. For each chain $(i, j)$ associated with $hp_t$, store $\mathsf{doc}(i)$ in the first empty cell of the array $A_t$, and $\mathsf{maxDepth}(i, j)$ in the corresponding cell of the array $A'_t$. Sort both the arrays w.r.t the values in $A'_t$. For each node $u$ lying on $hp_t$, maintain a pointer to the minimum index $x$ of $A$ such that $A'_t[x] = \mathsf{depth}(u)$. Discard the array $A'_t$. Finally, build the 1-dimensional sorted range-reporting structure (Fact 3) over $A_t$. Total space for all $t$ is bounded by $O(n)$ words.

**THS component:** We construct the transformed heavy path tree $T_H^t$ from GST. Recall that every chain in GST is associated with a node in $T_H^t$. For each node $h_i$ in $T_H^t$, we store two arrays, chain source array $CS_i$ and chain destination array $CD_i$. The arrays $CS_i$ (resp. $CD_i$) contains the weights (i.e., the document identifier) of all the chains associated with $h_i$ sorted by the start (resp. end) position of the chain in GST. Finally we build the RMQ data structure (Fact 4) $RMQ_{CS_i}$ and $RMQ_{CD_i}$ over $CS_i$ and $CD_i$ respectively. Total space can be bounded by $O(n)$ words.

▶ **Fact 4** ([6, 7]). *By maintaining a $2n+o(n)$ bits structure, range maximum query(RMQ) can be answered in $O(1)$ time (without accessing the array).*

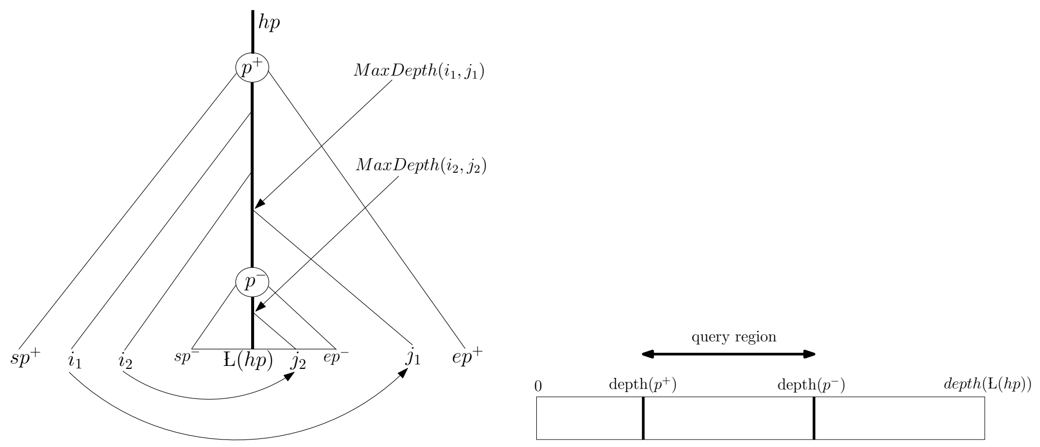
**Query Answering.** Query answering is done by traversing from $p^+$ to $p^-$ in GST. We start with the following observation.

▶ **Observation 1.** *For every type $C$ chain $(i, j)$, $\mathsf{lca}(i, j)$ falls on the $p^+$ to $p^-$ path in* GST.

This observation is crucial to ensure that we do not miss any type C chain in query answering. We consider the following two cases for query answering.

### 3.2.1 $p^+$ and $p^-$ falls on the same heavy path

In this case, we resort to component $MDS$ for query answering. Assume that $p^+$ and $p^-$ fall on heavy path $hp_t$. Note that a chain $(i, j)$ qualifies as an output, iff $\mathsf{maxDepth}(i, j)$ falls within the range $[\mathsf{depth}(p^+), \mathsf{depth}(p^-) - 1]$. See Figure 3(a) for illustration. For query answering, follow the pointers from $p^+$ and $p^-$ to the indexes $x$ and $y$ in the array $A_t$, and issue the query $\langle x, y-1, k\rangle$ in the corresponding Fact 3 data structure. Note that Type A and Type B outputs can arise. We obtain the following lemma.



**(a)** Chain $(i_1, j_1)$ qualifies since $\mathsf{maxDepth}(i, j) \in [\mathsf{depth}(p^+), \mathsf{depth}(p^-))$. $(i_2, j_2)$ does not qualify.

**(b)** Query range in the 1-dimensional sorted range reporting structure of $hp$.

**Figure 3** $p^+$ and $p^-$ falling on the same heavy path.

▶ **Lemma 6.** *There exists an $O(n)$ words data structure, such that for a* top-$k$ *forbidden extension query, we can report the* top-$k$ *Type C leaves in $O(|P^-| + k)$ time when $p^+$ and $p^-$ falls on the same heavy path.*

### 3.2.2 $p^+$ and $p^-$ falls on different heavy paths

Let $p_1, p_2, \ldots, p_t$ be the path segments of the path from $p^+$ to $p^-$ traversing heavy paths $hp_1, hp_2, \ldots, hp_t$, where $p_i \in hp_i$, $1 \leq i \leq t$. Let $h_1, h_2, \ldots, h_t$ be the corresponding nodes in $T_H^t$. In the following subsection, we show how to obtain answers for $h_1$ through $h_{t-1}$; we resolve $h_t$ separately. We use the THS component for processing the chains with LCA on $h_1, h_2, \ldots, h_{t-1}$. We start with the following lemma.

▶ **Lemma 7.** *Let $(i, j)$ be a chain associated with a node $h_k$ in $T_H^t$. If $p^-$ falls on the left (resp. right) subtree of $h_k$, and $sp^+ \leq i < sp^-$ (resp. $ep^- < j \leq ep^+$), then $(i, j)$ is qualified as an output of the forbidden extension query.*

**Proof.** Recall that chain $(i, j)$ is associated with $h_k = \mathsf{lca}(h_i, h_j)$ in $T_H^t$, where $h_i$ and $h_j$ are the heavy path nodes corresponding to $i$ and $j$ respectively. This implies $h_i$ (resp. $h_j$) falls on the left (resp. right) subtree of $h_k$. If $p^-$ falls on the left of $hp_k$ then $j > ep^-$. The added constraint $sp^+ \leq i < sp^-$ ensures that chain $(i, j)$ is either a Type B or a Type C chain, both of which are qualified as an output of the forbidden extension query. The case when $p^-$ falls on the right of $h_k$ is symmetric. ◀

Lemma 7 allows us to check only the source or destination of a chain based on the position of $p^-$, and collect the top weighted chains; this is facilitated using the RMQ data structure. We traverse the nodes in $T_H^t$ from $p^+$ to $p^-$. At each node $h_k$, if $p^-$ falls on the left of $h_k$, we issue a range maximum query within the range $[sp^+, sp^- - 1]$ on $RMQ_{CS_k}$ which gives us the top answer from each node in $O(1)$ time. Note that, $[sp^+, sp^- - 1]$ range needs to be transformed for different $RMQ_{CS}$ structures. We use fractional cascading for the range transformation to save predecessor searching time (refer to Appendix A for detailed discussion). Since the height of the tree is $O(\log^2 n)$ (refer to Lemma 5) at any instance, there are at most $O(\log^2 n)$ candidate points. We use the *atomic heap* of Fredman and Willard [9] which allows constant

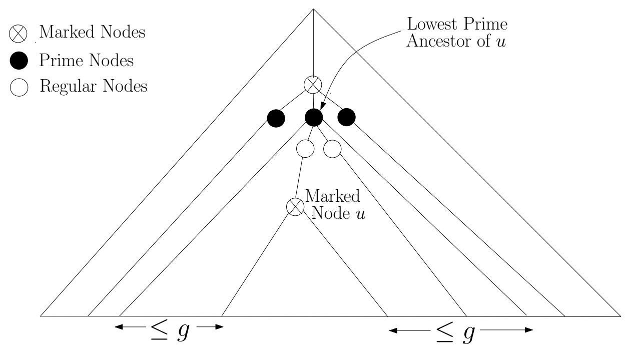


**Figure 4** Marked nodes and Prime nodes with respect to grouping factor $g$.

time insertion and delete-max operation when the heap size is $O(\log^2 m)$, where $m$ is the size of the universe. By maintaining each candidate point in the atomic heap, the highest weighted point (among all candidate points) can be obtained in constant time. Also, once the highest weighted point from a heavy path node is obtained, each subsequent candidate point can be obtained and inserted into the the atomic heap in $O(1)$ time. Hence the query time is bounded by the number of nodes traversed in $T_H^t$. From lemma 5, we obtain that the number of nodes traversed is bounded by $O(\min(|P^-|\log\sigma, \log^2 n))$.

For $hp_t$, we utilize component $MDS$. Let $r_t$ be the root of heavy path $hp_t$. A chain $(i, j)$ qualifies as an output, iff $\mathsf{maxDepth}(i, j)$ falls within the range $[\mathsf{depth}(r_t), \mathsf{depth}(p^-) - 1]$. For query answering, follow the pointers from $r_t$ and $p^-$ to the indexes $x$ and $y$ in the array $A_t$, and issue the query $\langle x, y - 1, k\rangle$ in the corresponding Fact 3 data structure. Note that Type A and Type B outputs can arise.

From the above discussion, we obtain the following lemma.

▶ **Lemma 8.** *There exists an $O(n)$ words data structure, such that for a* top-*$k$ forbidden extension query, we can report the* top-*$k$ Type C leaves in $O(|P^-|\log\sigma + k)$ time when $p^+$ and $p^-$ falls on different heavy paths.*

Combining Lemmas 3, 6, and 8, we obtain the result stated in Theorem 1.

## 4 Succinct Index

In this section, we prove Theorem 2. The key idea is to identify some special nodes in the GST, pre-compute the answers for a special node and its descendant special node, and maintain these answers in a data structure. By appropriately choosing the special nodes, the space can be bounded by $O(n)$ bits. Using other additional compressed data structures for document listing [14], we arrive at our claimed result.

We begin by identifying certain nodes in GST as *marked nodes* and *prime nodes* based on a parameter $g$ called *grouping factor* [13]. First, starting from the leftmost leaf in GST, we combine every $g$ leaves together to form a group. In particular, the leaves $\ell_1$ through $\ell_g$ forms the first group, $\ell_{g+1}$ through $\ell_{2g}$ forms the second, and so on. We mark the LCA of the first and last leaves of every group. Moreover, for any two marked nodes, we mark their LCA (and continue this recursively). Note that the root node is marked, and the number of marked nodes is at most $2\lceil n/g\rceil$. See Figure 4 for an illustration.

Corresponding to each marked node (except the root), we identify a unique node called the prime node. Specifically, the prime node $u'$ corresponding to a marked node $u^*$ is the node on the path from root to $u^*$, which is a child of the lowest marked ancestor of $u^*$; we refer to $u'$ as the lowest prime ancestor of $u^*$. Since the root node is marked, there is always such a node. If the parent of $u^*$ is marked, then $u^*$ is same as $u'$. Also, for every prime node, the corresponding closest marked descendant (and ancestor) is unique. Therefore number of prime nodes is one less than the number of marked nodes. The following lemma highlights some important properties of marked and prime nodes.

▶ **Fact 5** ([1, 14]). *(i) In constant time we can verify whether any node has a marked descendant or not. (ii) If a node $u$ has no marked descendant, then $|\mathsf{leaf}(u)| < 2g$. (iii) If $u^*$ is the highest marked descendant of $u$, and $u$ is not marked, then $|\mathsf{leaf}(u, u^*)| \le 2g$. (iv) If $u'$ is the lowest prime ancestor of $u^*$. Then $|\mathsf{leaf}(u', u^*)| \le 2g$.*

We now present a framework for proving the following lemma.

▶ **Lemma 9.** *Assume the following.*
**(a)** *The highest marked node $u^*$ and the sequence of prime nodes (if any) on the path from $p^+$ to $p^-$ can be found in $\mathsf{t}_{\mathsf{prime}}$ time.*
**(b)** *For any leaf $\ell_i$, we can find the corresponding document in $\mathsf{t}_{\mathsf{DA}}$ time.*
**(c)** *For any document identifier $d$ and a range of leaves $[sp, ep]$, we can check in $\mathsf{t}_{\in}$ time, whether $d$ belongs in $\{\mathsf{doc}(i) \mid sp \le i \le ep\}$, or not.*
*For any function $f(n)$, such that $f(n) = \Omega(1)$ and $f(n) = o(n)$, by maintaining* CSA *and additional $O((n/f(n)) \log^2 n)$ bits structures, we can answer* top-$k$ *forbidden extension queries in $O(\mathsf{search}(P^-) + \mathsf{t}_{\mathsf{prime}} + k \cdot f(n) \cdot (\mathsf{t}_{\mathsf{DA}} + \mathsf{t}_{\in}))$ time.*

**Creating the Index.** First we maintain a full-text index CSA on the document collection $\mathcal{D}$. Let $g_\kappa = \lceil \kappa \cdot f(n) \rceil$, where $\kappa$ is a parameter to be defined later. We begin by marking nodes in the GST as marked and prime nodes, as defined previously, based on $g_\kappa$. Consider any prime node $u$, and let $u^\uparrow$ and $u^\downarrow$ be its nearest marked ancestor and descendant (both of which are unique) respectively. We compute the arrays $\mathsf{list}_\kappa(u^\uparrow, u)$ and $\mathsf{list}_\kappa(u, u^\downarrow)$, each sorted by increasing importance (i.e., document identifier). The arrays are maintained in the node $u$ w.r.t grouping factor $g_\kappa$. Note that explicitly maintaining each array requires $O(\kappa \log n)$ bits. Space required in bits for all prime nodes w.r.t $g_\kappa$ can be bounded by $O((n/g_\kappa)\kappa \log n)$ i.e., by $O((n/f(n)) \log n)$ bits. We maintain this data-structure for $\kappa = 1, 2, 4 \ldots, D$. Total space is bounded by $O((n/f(n)) \log^2 n)$ bits.

**Querying Answering.** For a top-$k$ forbidden extension query $\langle P^+, P^-, k\rangle$, we begin by locating the suffix ranges $[sp^+, ep^+]$ and $[sp^-, ep^-]$ of the patterns $P^+$ and $P^-$ respectively; this can be achieved in time bounded by $\mathsf{search}(P^-)$ using the CSA. If the suffix ranges are the same, then clearly every document containing $P^+$ also contains $P^-$, and the top-$k$ list is empty. So, moving forward, we assume otherwise. Note that it suffices to obtain a $k$-candidate set of size $O(k \cdot f(n))$ in the time of Lemma 9.

Let $k' = \min\{D, 2^{\lceil \log k \rceil}\}$. Note that $k \le k' < 2k$. Moving forwards, we talk of prime and marked nodes w.r.t grouping factor $g' = \lceil k' f(n) \rceil$. We can detect the presence of marked nodes below $p^+$ and $p^-$ in constant time using Fact 5. Let the prime nodes on the path from $p^+$ to $p^-$ be $u_1, u_2, \ldots, u_t$ in order of depth. Possibly, $t = 0$. For each prime node $u_{t'}$, $1 \le t' \le t$, we denote by $u_{t'}^\uparrow$ and $u_{t'}^\downarrow$, the lowest marked ancestor (resp. highest marked descendant) of the $u_{t'}$. We have the following cases.

▶ **Case 1.** *We consider the following two scenarios: (i)* $\mathsf{GST}(p^+)$ *does not contain any marked node, and (ii)* $\mathsf{GST}(p^+)$ *contains a marked node, but the path from $p^+$ to $p^-$ does*

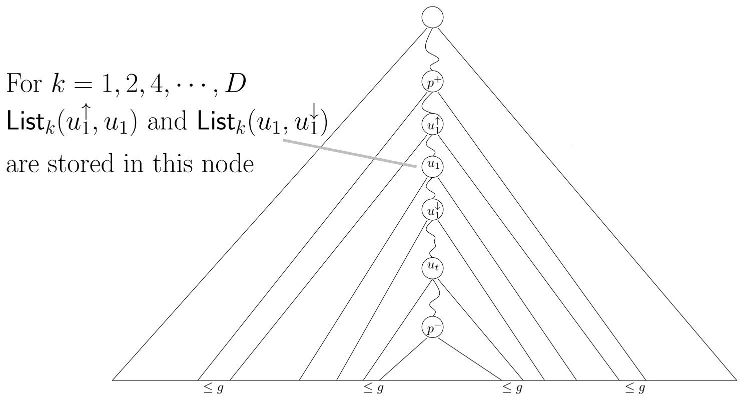

For $k = 1, 2, 4, \cdots, D$
$\mathsf{List}_k(u_1^{\uparrow}, u_1)$ and $\mathsf{List}_k(u_1, u_1^{\downarrow})$
are stored in this node
$p^+$
$u_1^{\uparrow}$
$u_1$
$u_1^{\downarrow}$
$u_t$
$p^-$
$\le g$
$\le g$
$\le g$
$\le g$

**Figure 5** Illustration of storage scheme and retrieval at every prime node w.r.t grouping factor $g$. Left and right fringes in $\mathsf{leaf}(p^+ \setminus u_1^{\uparrow})$ and $\mathsf{leaf}(u_t^{\downarrow} \setminus p^-)$ are bounded above by $g'$.

*not contain any prime node. In either case,* $|\mathsf{leaf}(p^+, p^-)| \le 2g'$ *(refer to Fact 5). The documents corresponding to these leaves constitute a $k$-candidate set, and can be found in* $O(g' \cdot \mathsf{t}_{\mathsf{DA}})$ *time i.e., in* $O(k \cdot f(n) \cdot \mathsf{t}_{\mathsf{DA}})$ *time. Now, for each document $d$, we check whether* $d \in \{\mathsf{doc}(i) \mid i \in [sp^-, ep^-]\}$*, which requires additional* $O(g' \cdot \mathsf{t}_{\in})$ *time. Total time can be bounded by* $O(g' \cdot (\mathsf{t}_{\mathsf{DA}} + \mathsf{t}_{\in}))$ *i.e., by* $O(k \cdot f(n) \cdot (\mathsf{t}_{\mathsf{DA}} + \mathsf{t}_{\in}))$.

▶ **Case 2.** *If the path from $p^+$ to $p^-$ contains a prime node, then let $u^*$ be the highest marked node. Possibly, $u^* = p^+$. Note that $u_1^{\uparrow}$ is same as $u^*$, and that $u_t^{\downarrow}$ is either $p^-$ or a node below it. For any $t'$, clearly* $\mathsf{list}_{k'}(u_{t'}, u_{t'}^{\downarrow})$ *and* $\mathsf{list}_{k'}(u_{t'}^{\uparrow}, u_{t'})$ *are mutually disjoint. Similar remarks hold for the lists stored at two different prime nodes $t'$ and $t''$, $1 \le t', t'' \le t$. Furthermore, let $d$ be an identifier in one of the lists corresponding to $u_{t'}$. Clearly there is no leaf $\ell_j \in$* $\mathsf{GST}(p^-)$*, such that* $\mathsf{doc}(j) = d$*. We select the* $\mathsf{top}$*-$k'$ document identifiers from the stored lists (arrays) in the prime nodes $u_1$ through $u_t$. Time, according to the following fact, can be bounded by $O(t + k)$.*

▶ **Fact 6** ([2, 8]). *Given $m$ sorted integer arrays, we can find the $k$ largest values from all these arrays in $O(m + k)$ time.*

*Now, we consider the* fringe leaves $\mathsf{leaf}(p^+, u^*)$ *and* $\mathsf{leaf}(u_t, p^-)$*, both of which are bounded above by $2g'$ (refer to Fact 5). The ranges of the these leaves are found in constant time using the following result of Sadakane and Navarro [27].*

▶ **Lemma 10** ([27]). *An $m$ node tree can be maintained in $O(m)$ bits such that given a node $u$, we can find $[sp(u), ep(u)]$ in constant time.*

*The relevant documents corresponding to these fringe leaves can be retrieved as in Case 1. Clearly, these fringe documents along with the $k$ documents obtained from the stored lists constitute our $k$-candidate set. Time required can be bounded by* $O(t + k + g' \cdot (\mathsf{t}_{\mathsf{DA}} + \mathsf{t}_{\in}))$ *i.e, by* $O(t + k \cdot f(n) \cdot (\mathsf{t}_{\mathsf{DA}} + \mathsf{t}_{\in}))$.

Note that $t \le \mathsf{depth}(p^-) \le |P^-| = O(\mathsf{search}(P^-))$, and Lemma 9 follows. ◀

We are now equipped to prove Theorem 2. First, the highest marked node and the $t$ prime nodes from $p^+$ to $p^-$ are obtained using Lemma 11 in $O(\log n + t)$ time. Maintain the data-structure of this lemma for with $\kappa = 1, 2, 4, \ldots, D$. Space can be bounded by $O(\frac{n}{f(n)} \log n)$

bits. Computing $\mathsf{doc}(i)$ is achieved in $\mathsf{t_{SA}}$ time, according to Lemma 12. Checking whether a document $d$ belongs in a contiguous range of leaves is achieved in $O(\mathsf{t_{SA}} \cdot \log\log n)$ using Lemma 13. Theorem 2 is now immediate by choosing $f(n) = \log^2 n$.

▶ **Lemma 11.** *By maintaining $O((n/g_\kappa)\log n)$ bits in total, we can retrieve the highest marked node, and all $t$ prime nodes, both w.r.t grouping factor $g_\kappa = \lceil \kappa \cdot f(n) \rceil$, that lie on the path from $p^+$ to $p^-$ in time bounded by $O(\log n + t)$.*

**Proof.** We use the following result of Patil et al. [26]: a set of $n$ three-dimensional points $(x, y, z)$ can be stored in an $O(n \log n)$ bits data structure, such that for a three-dimensional dominance query $\langle a, b, c\rangle$, in $O(\log n + t)$ time, we can report all $t$ points $(x, y, z)$ that satisfies $x \le a$, $y \ge b$, and $z \ge c$ with outputs reported in the sorted order of $z$ coordinate.

For each prime node $w$, we maintain the point $(L_w, R_w, |\mathsf{path}(w)|)$ in the data structure above, where $L_w$ and $R_w$ are the leftmost and the rightmost leaves in $\mathsf{GST}(w)$. Total space in bits can be bounded by $O((n/g_\kappa)\log n)$ bits. The $t$ prime nodes that lie on the path from $p^+$ to $p^-$ are retrieved by querying with $\langle sp^- - 1, ep^- + 1, |P^+|\rangle$. Time can be bounded by $O(\log n + t)$. Likewise, we maintain a structure for marked nodes. Using this, we can obtain the highest marked node in $O(\log n)$ time. ◀

▶ **Lemma 12.** *Given a* $\mathsf{CSA}$*, the document array can be maintained in additional $n + o(n)$ bits such that for any leaf $\ell_i$, we can find* $\mathsf{doc}(i)$ *in* $\mathsf{t_{SA}}$ *time i.e.,* $\mathsf{t_{DA}} = \mathsf{t_{SA}}$.

**Proof.** We use the following data-structure [10, 20]: a bit-array $\mathsf{B}[1 \ldots m]$ can be encoded in $m + o(m)$ bits, such that $\mathsf{rank_B}(q, i) = |\{j \in [1..i] \mid \mathsf{B}[j] = q\}|$ can be found in $O(1)$ time.

Consider the concatenated text $\mathsf{T}$ of all the documents which has length $n$. Let $\mathsf{B}$ be a bit array of length $n$ such that $\mathsf{B}[i] = 1$ if a document starts at the position $i$ in the text $\mathsf{T}$. We maintain a $\mathsf{rank}$ structure on this bit-array. Space required is $n + o(n)$ bits. We find the text position $j$ of $\ell_i$ in $\mathsf{t_{SA}}$ time. Then $\mathsf{doc}(i) = \mathsf{rank_B}(1, j)$, and is retrieved in constant time. Time required can be bounded by $\mathsf{t_{SA}}$. ◀

▶ **Lemma 13.** *Given the suffix range $[sp, ep]$ of a pattern $P$ and a document identifier $d$, by maintaining* $\mathsf{CSA}$ *and additional $|\mathsf{CSA}^*| + D \log \frac{n}{D} + O(D) + o(n)$ bits structures, in $O(\mathsf{t_{SA}} \log\log n)$ time we can verify whether $d \in \{\mathsf{doc}(i) \mid i \in [sp, ep]\}$, or not.*

**Proof.** Number of occurrences of $d$ in a suffix range $[sp, ep]$ is given by $\mathsf{rank_{DA}}(d, ep) - \mathsf{rank_{DA}}(d, sp - 1)$. Space and time complexity is due to the following result of Hon et al. [14]: the document array $\mathsf{DA}$ can be simulated using $\mathsf{CSA}$ and additional $|\mathsf{CSA}^*| + D \log \frac{n}{D} + O(D) + o(n)$ bits structures to support $\mathsf{rank_{DA}}$ operation in $O(\mathsf{t_{SA}} \log\log n)$ time. ◀

## 5 Concluding Remarks

In this paper, we introduce the problem of $\mathsf{top}$-$k$ forbidden extension query, and propose a linear space index for answering such queries. By maintaining a linear space index, the general forbidden pattern query for an included pattern $P$, and a forbidden pattern $Q$, can be answered in $O(|P| + |Q| + \sqrt{n \cdot occ})$ time, where $occ$ is the number of documents reported. We show that by maintaining a linear space index, we can answer forbidden extension queries in optimal $O(|P^-| + occ)$ time. We also address the more general $\mathsf{top}$-$k$ version of the problem, where the relevance measure is based on PageRank. We show that by maintaining linear space index, we obtain a query time of $O(|P^-| \log \sigma + k)$, which is optimal for constant alphabets. Furthermore, we obtain a succinct solution to this problem.

## References

1 Sudip Biswas, Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. Ranked document retrieval with forbidden pattern. In Ferdinando Cicalese, Ely Porat, and Ugo Vaccaro, editors, *Combinatorial Pattern Matching – 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 – July 1, 2015, Proceedings*, volume 9133 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2015.

2 Gerth Stølting Brodal, Rolf Fagerberg, Mark Greve, and Alejandro López-Ortiz. Online sorted range reporting. 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 173–182. Springer, 2009.

3 Bernard Chazelle and Leonidas J Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1(1-4):133–162, 1986.

4 Hagai Cohen and Ely Porat. Fast set intersection and two-patterns matching. *Theoretical Computer Science*, 411(40–42):3795–3800, 2010.

5 Johannes Fischer, Travis Gagie, Tsvi Kopelowitz, Moshe Lewenstein, Veli Mäkinen, Leena Salmela, and Niko Välimäki. Forbidden patterns. In *Proceedings of the 10th Latin American International Conference on Theoretical Informatics*, LATIN'12, pages 327–337, Berlin, Heidelberg, 2012. Springer-Verlag.

6 Johannes Fischer and Volker Heun. Theoretical and practical improvements on the rmq-problem, with applications to LCA and LCE. In Moshe Lewenstein and Gabriel Valiente, editors, *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5-7, 2006, Proceedings*, volume 4009 of *Lecture Notes in Computer Science*, pages 36–48. Springer, 2006.

7 Johannes Fischer and Volker Heun. A new succinct representation of rmq-information and improvements in the enhanced suffix array. In Bo Chen, Mike Paterson, and Guochuan Zhang, editors, *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, First International Symposium, ESCAPE 2007, Hangzhou, China, April 7-9, 2007, Revised Selected Papers*, volume 4614 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2007.

8 Greg N. Frederickson and Donald B. Johnson. The complexity of selection and ranking in X+Y and matrices with sorted columns. *J. Comput. Syst. Sci.*, 24(2):197–208, 1982.

9 Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.

10 Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 368–373. ACM Press, 2006.

11 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology.* Cambridge University Press, 1997.

12 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *siam Journal on Computing*, 13(2):338–355, 1984.

13 Wing-Kai Hon, R. Shah, and J.S. Vitter. Space-efficient framework for top-k string retrieval problems. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 713–722, Oct 2009.

14 Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Space-efficient frameworks for top-$k$ string retrieval. *J. ACM*, 61(2):9, 2014.

15 Wing-Kai Hon, Rahul Shah, SharmaV. Thankachan, and JeffreyScott Vitter. String retrieval for multi-pattern queries. In Edgar Chavez and Stefano Lonardi, editors, *String*

*Processing and Information Retrieval*, volume 6393 of *Lecture Notes in Computer Science*, pages 55–66. Springer Berlin Heidelberg, 2010.

**16** Wing-Kai Hon, Rahul Shah, SharmaV. Thankachan, and JeffreyScott Vitter. Document listing for queries with excluded pattern. In Juha Kärkkäinen and Jens Stoye, editors, *Combinatorial Pattern Matching*, volume 7354 of *Lecture Notes in Computer Science*, pages 185–195. Springer Berlin Heidelberg, 2012.

**17** Wing-Kai Hon, Sharma V. Thankachan, Rahul Shah, and Jeffrey Scott Vitter. Faster compressed top-k document retrieval. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2013 Data Compression Conference, DCC 2013, Snowbird, UT, USA, March 20-22, 2013*, pages 341–350. IEEE, 2013.

**18** KasperGreen Larsen, J.Ian Munro, JesperSindahl Nielsen, and SharmaV. Thankachan. On hardness of several string indexing problems. In AlexanderS. Kulikov, SergeiO. Kuznetsov, and Pavel Pevzner, editors, *Combinatorial Pattern Matching*, volume 8486 of *Lecture Notes in Computer Science*, pages 242–251. Springer International Publishing, 2014.

**19** Yossi Matias, S. Muthukrishnan, SüleymanCenk Sahinalp, and Jacob Ziv. Augmenting suffix trees, with applications. In Gianfranco Bilardi, GiuseppeF. Italiano, Andrea Pietracaprina, and Geppino Pucci, editors, *Algorithms — ESA' 98*, volume 1461 of *Lecture Notes in Computer Science*, pages 67–78. Springer Berlin Heidelberg, 1998.

**20** J. Ian Munro. Tables. In Vijay Chandru and V. Vinay, editors, *Foundations of Software Technology and Theoretical Computer Science, 16th Conference, Hyderabad, India, December 18-20, 1996, Proceedings*, volume 1180 of *Lecture Notes in Computer Science*, pages 37–42. Springer, 1996.

**21** J. Ian Munro, Gonzalo Navarro, Jesper Sindahl Nielsen, Rahul Shah, and Sharma V. Thankachan. Top- k term-proximity in succinct space. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation – 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*, volume 8889 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2014.

**22** S. Muthukrishnan. Efficient algorithms for document retrieval problems. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 657–666. ACM/SIAM, 2002.

**23** Gonzalo Navarro and Yakov Nekrich. Top-$k$ document retrieval in optimal time and linear space. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1066–1077. SIAM, 2012.

**24** Gonzalo Navarro and Yakov Nekrich. Top-k document retrieval in optimal time and linear space. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1066–1077. SIAM, 2012.

**25** Gonzalo Navarro and Sharma V. Thankachan. Faster top-k document retrieval in optimal space. In Oren Kurland, Moshe Lewenstein, and Ely Porat, editors, *String Processing and Information Retrieval – 20th International Symposium, SPIRE 2013, Jerusalem, Israel, October 7-9, 2013, Proceedings*, volume 8214 of *Lecture Notes in Computer Science*, pages 255–262. Springer, 2013.

**26** Manish Patil, Sharma V Thankachan, Rahul Shah, Yakov Nekrich, and Jeffrey Scott Vitter. Categorical range maxima queries. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 266–277. ACM, 2014.

**27** Kunihiko Sadakane and Gonzalo Navarro. Fully-functional succinct trees. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 134–149. SIAM, 2010.

## A Range Transformation using Fractional Cascading

We employ the fractional cascading idea of Chazelle et.al [3] for predecessor searching in $CS$ array. Successor searching and $CD$ array are handled in a similar way. The idea is to merge the $CS$ array for siblings and propagate the predecessor information from bottom-to-top. Two arrays are used for this purpose: merged siblings array $MS$ and merged children array $MC$. Let $h_i$ be an internal node in $T_H^t$ having sibling $h_j$ and two children leaf nodes $h_u$ and $h_v$. Array $MC_u$ (resp. $MC_v$) is same as $CS_u$ (resp. $CS_v$) and stored in $h_u$ (resp. $h_v$). The arrays $CS_u$ and $CS_v$ are merged to form a sorted list $MS_{uv}$. Note that, $CS_v$ values are strictly greater than $CS_u$; therefore, $CS_u$ and $CS_v$ form two disjoint partitions in $MS_{lr}$ after sorting. We denote the left partition as $MS_{uv}^l$ and the right partition as $MS_{uv}^r$. We also store a pointer from each value in $MS_{uv}^l$ ($MS_{uv}^r$) to its corresponding value in $MC_u$ (resp. $MC_v$). The list $MC_i$ is formed by merging $CS_i$ with every second item from $MS_{lr}$. With each item $x$ in $MC_i$, we store three numbers: the predecessor of $x$ in $CS_i$, the predecessor of $x$ in $MS_{uv}^l$ and the predecessor of $x$ in $MS_{uv}^r$. Total space required is linear in the number of chains, and is bounded by $O(n)$ words.

Using this data structure, we show how to find predecessor efficiently. Let $h_w$ be an ancestor node of $h_z$ in $T_H^t$. We want to traverse $h_w$ to $h_z$ path and search for the predecessor of $x$ in $CS_i$, where $h_i$ is a node on the $h_w$ to $h_z$ path. When we traverse from a parent node $h_i$ to a child node $h_j$, at first we obtain the predecessor value in parent node using $MC_i$. If $h_j$ is the left (resp. right) children of $h_i$, we obtain the predecessor value in $MS_{jk}^l$ (resp. $MS_{jk}^r$), where $h_k$ is the sibling of $h_j$. Following the pointer stored at $MS_{jk}^l$ or $MS_{jk}^r$, we can get the predecessor value at $MC_j$, and proceed the search to the next level. This way we can obtain the transformed range at each level in $O(1)$ time.