Consecutive Occurrences with Distance Constraints

Waseem Akram & Sanjeev Saxena

Indian Institute of Technology, Kanpur (INDIA)

Feb 15, 2024

Definitions and Notations

- A string is a sequence of characters.
- Let $P[1:m]$ and $T[1:n]$ be two strings with $m \leq n$.
- An index i is an *occurrence* of P if $P[1:m] = T[i:i+m-1]$.

Definitions and Notations

- A string is a sequence of characters.
- Let $P[1:m]$ and $T[1:n]$ be two strings with $m \le n$.
- An index i is an *occurrence* of P if $P[1:m] = T[i:i+m-1]$.

Example:

P	:	a	b	c															
T	:	e	f	a	b	c	g	z	t	a	b	c	x	y	a	b	c		

- A string is a sequence of characters.
- Let $P[1:m]$ and $T[1:n]$ be two strings with $m \leq n$.
- An index i is an *occurrence* of P if $P[1:m] = T[i:i+m-1]$.

Example:

P : a b c

T : e f $\boxed{\text{a b c}}$ g z t $\boxed{\text{a b c}}$ x y $\boxed{\text{a b c}}$

3 9 14

Definitions and Notations...

An ordered pair (i, j) is a *consecutive occurrence* of P if

1. P occurs at i and j
2. P has no occurrence between them

An ordered pair (i, j) is a *consecutive occurrence* of P if

1. P occurs at i and j
2. P has no occurrence between them

P : a b c

T : e f $\boxed{\text{a b c}}$ g z t $\boxed{\text{a b c}}$ x y $\boxed{\text{a b c}}$

3 9 14

Consecutive occurrences: $(3, 9)$ and $(9, 14)$.

The distance of a consecutive occurrence (i, j) is defined as $j - i$.

Preprocess a given text $T[1:n]$ to support queries

1. given P and $[\alpha, \beta]$, report consecutive occurrences (i, j) with $j - i \in [\alpha, \beta]$. **(bounded-gap query)**
2. given P and $k > 0$, report k consecutive occurrences (i, j) with minimal distance. **(top-k query)**

Problem Statement

Preprocess a given text $T[1:n]$ to support queries

1. given P and $[\alpha, \beta]$, report consecutive occurrences (i, j) with $j - i \in [\alpha, \beta]$. **(bounded-gap query)**
2. given P and $k > 0$, report k consecutive occurrences (i, j) with minimal distance. **(top-k query)**

Space	**Query Time**	**References**
$O(n \log n)$	$O(m + \#output)$	CPM'15 and FSTTCS'20

Existing solutions employ complex data structures (*persistent van Emde Boas*, *perfect hashing*, *persistent linked lists*).

Our Result

- We present present a solution using simpler data structures.

Space	top-k	bounded gap
$O(n \log n)$	$O(m+k)$	$O(m+\log\alpha+k)$

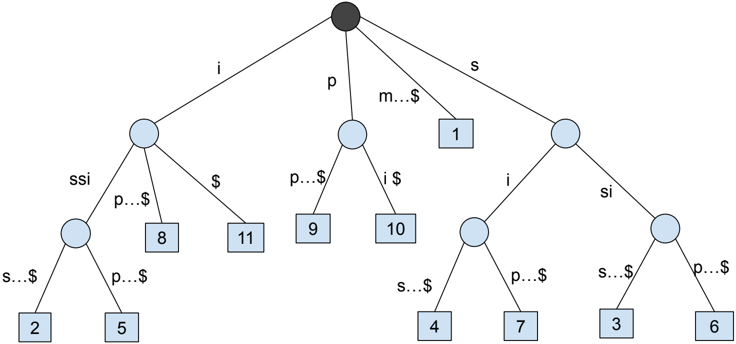
- If α is known, query time can be improved to $O(m+k)$.
- The preprocessing takes $O(n^2)$-time.

Suffix Tree

- A substring of the form $T[i : n]$ is called a *suffix* of $T[1 : n]$.
- It is a rooted tree with n leaves numbered from 1 to n.
- Every non-leaf node has at least two children.
- Each edge is labelled with a non-empty substring s.t.

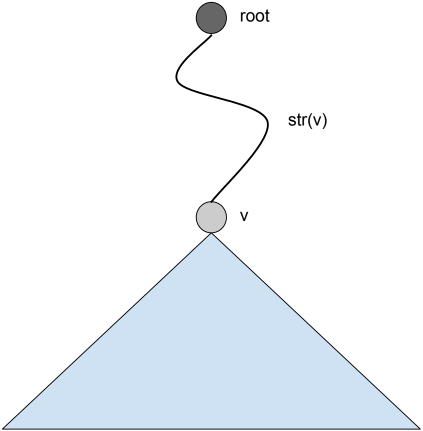
  concatenation of edge-labels from root to leaf i gives $T[i : n]$.

Suffix Tree...

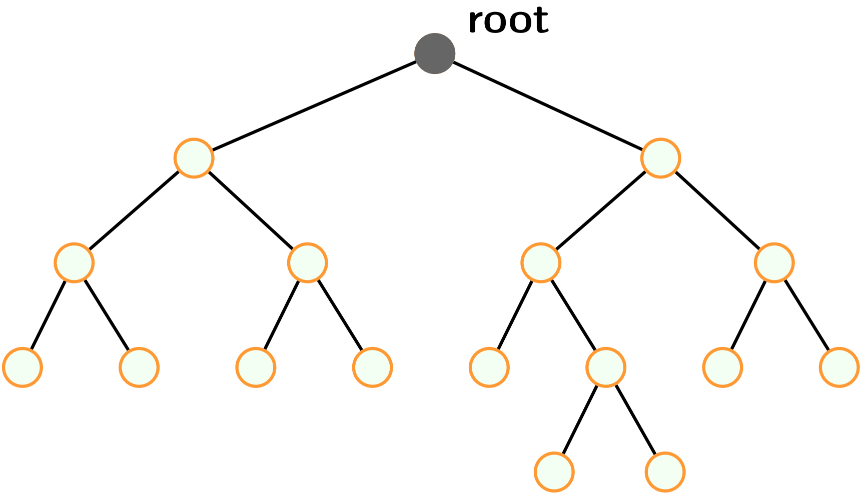


m	**i**	**s**	**s**	**i**	**s**	**s**	**i**	**p**	**p**	**i**	**$**
1	2	3	4	5	6	7	8	9	10	11	12

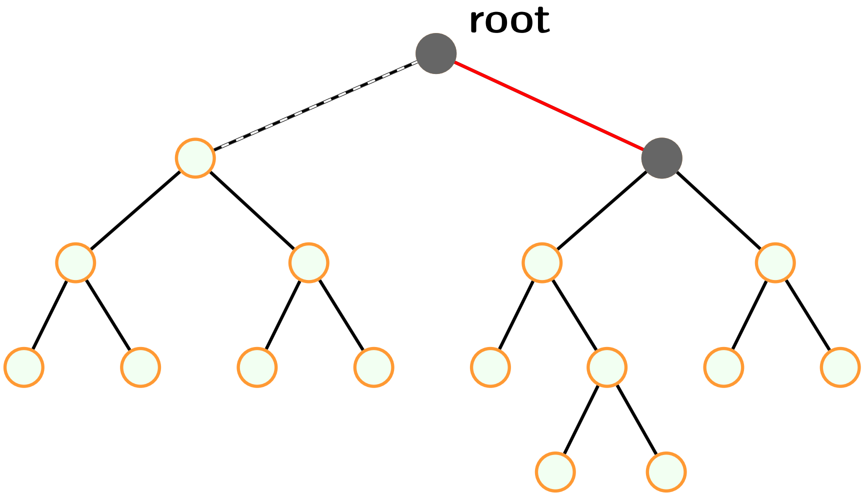
Suffix Tree...



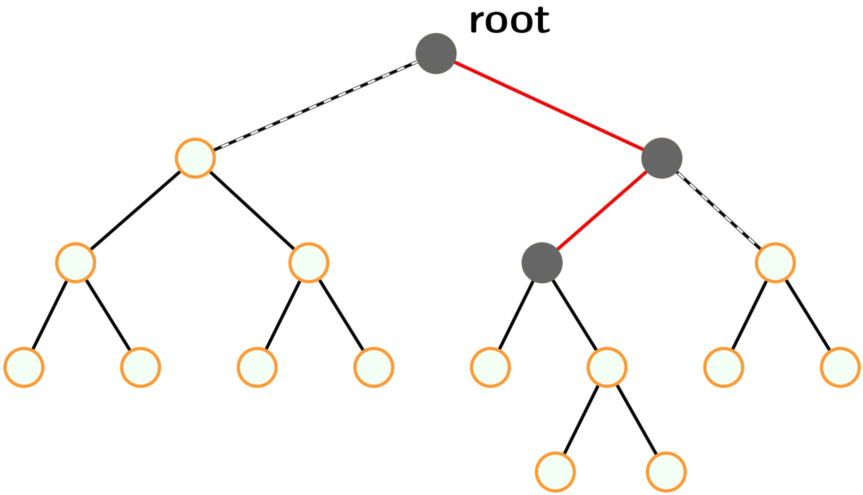
Heavy Path



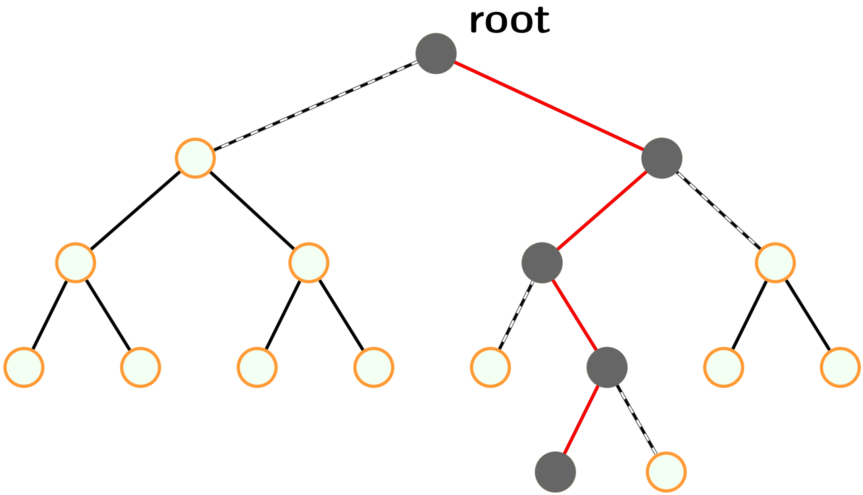
Heavy Path



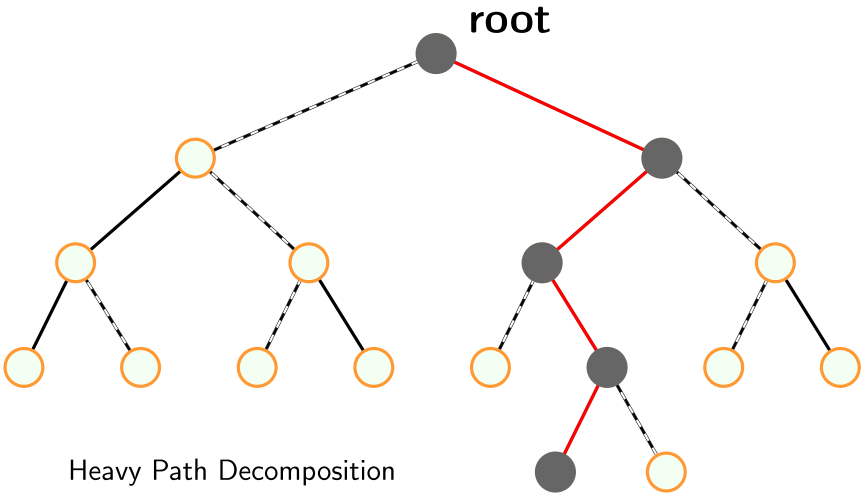
Heavy Path



Heavy Path



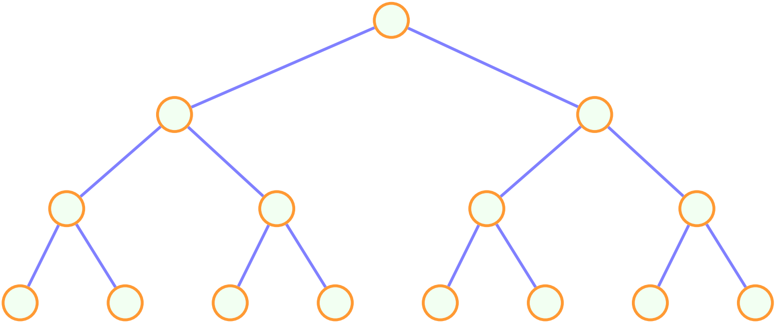
Heavy Path



Heavy Path Decomposition

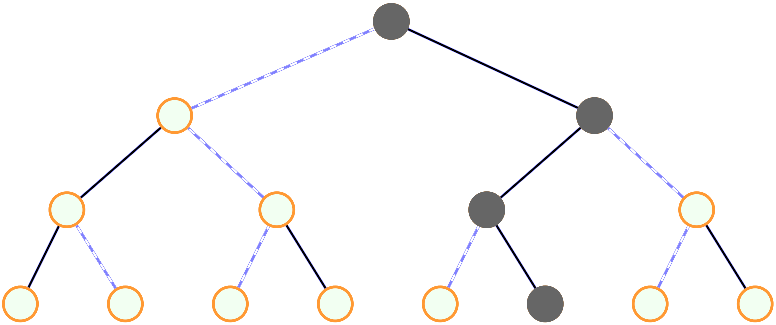
Preprocessing

- Let $\mathcal{T}$ be a suffix tree for the text $T[1:n]$



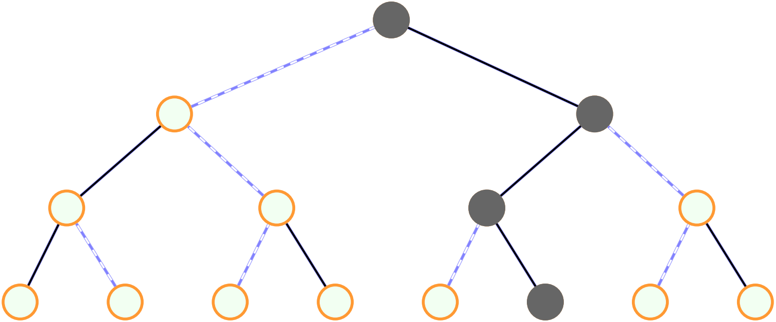
$\mathcal{T}$

- Decompose $\mathcal{T}$ using *heavy path decomposition*



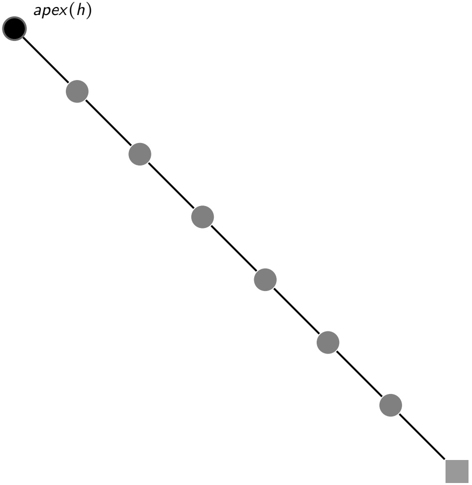
$\mathcal{T}$

- Create a data structure for each h.

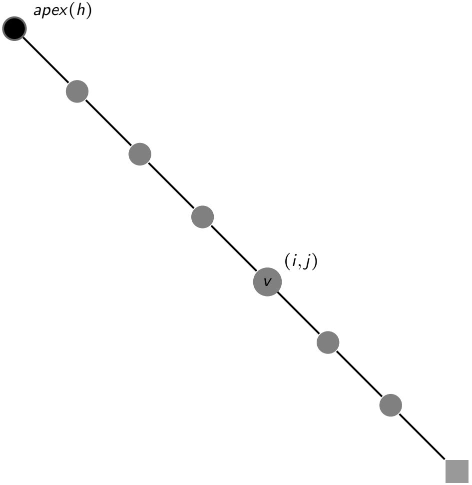


$\mathcal{T}$

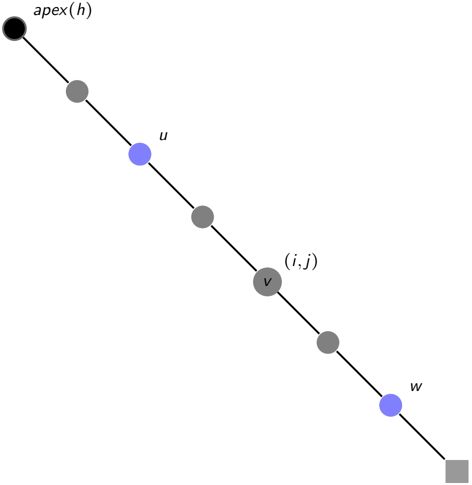
Data structure for heavy path h



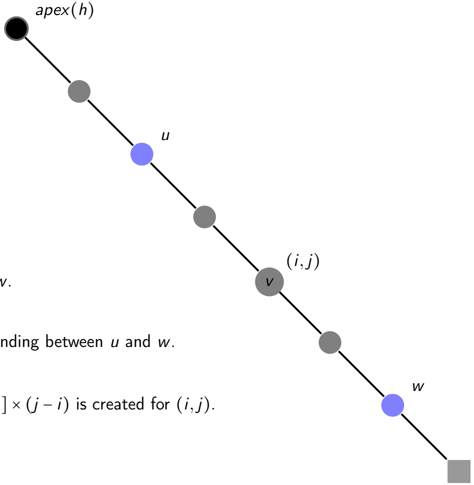
Data structure for heavy path h



Data structure for heavy path h



Data structure for heavy path h



(i,j) is alive from node u to node w.

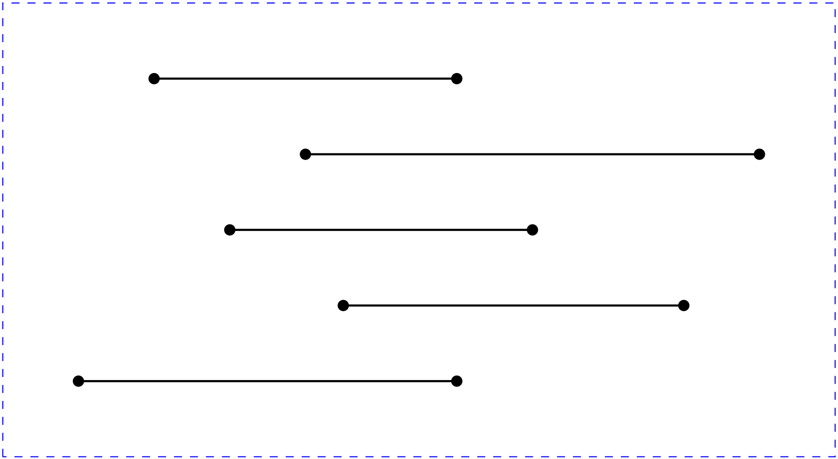
(i,j) is a cons. occ. of a pattern ending between u and w.

A horizontal segment $[d(u), d(w)] \times (j - i)$ is created for (i,j).

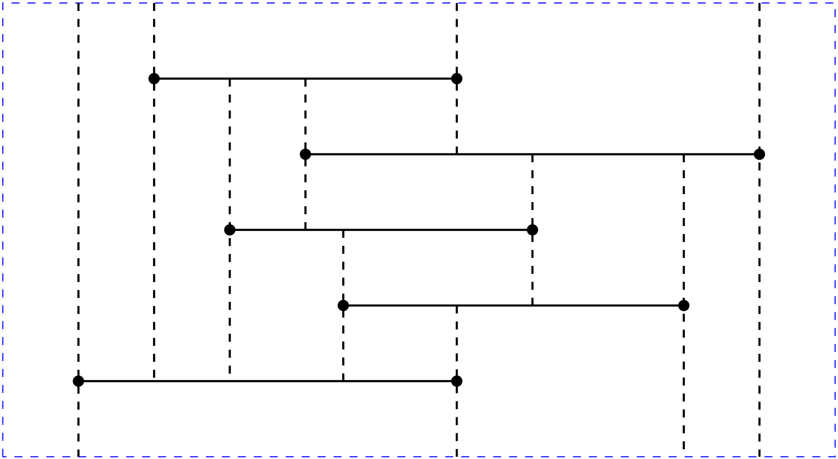
- Create a set of horizontal segments for h
- Preprocess the set for *orthogonal segment intersection queries*
- We employ the *hive-graph* data structure given by Chazelle[1].

[1] Chazelle, B.: Filtering Search: A New Approach to Query-Answering, 1985

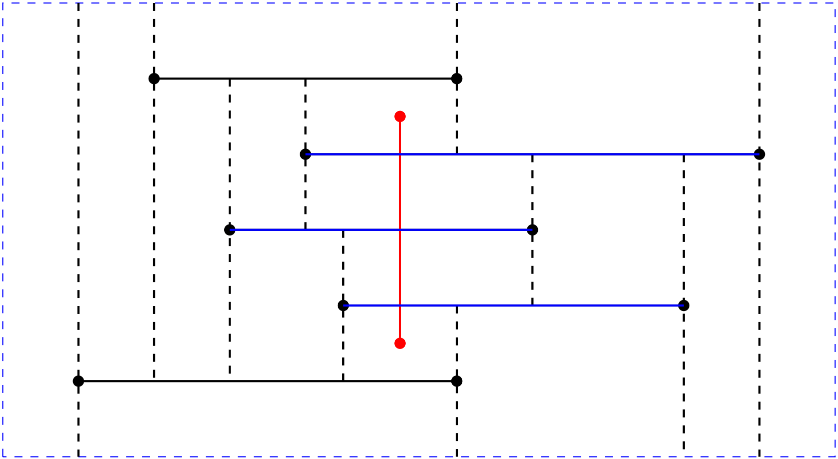
Hive Graph



Hive Graph



Hive Graph



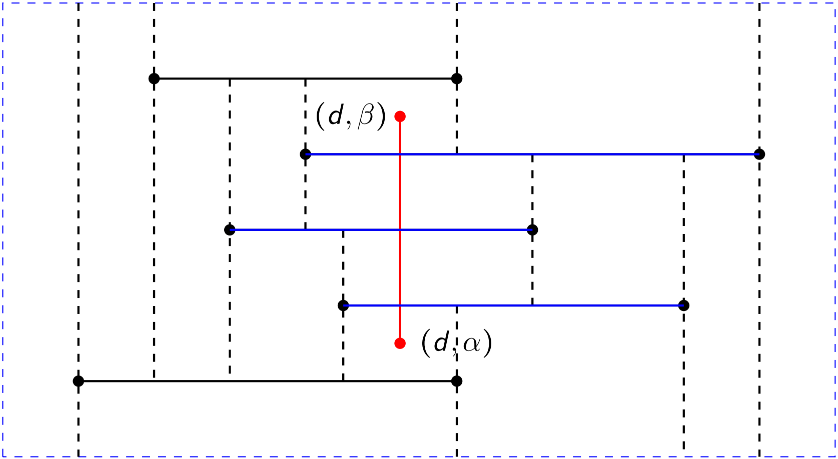
Preprocessing

1. build a suffix tree $\mathcal{T}$ for the text $T[1:n]$.
2. decompose the tree $\mathcal{T}$ using heavy path decomposition.
3. for each heavy path, create a set of horizontal segments, and

   preprocess the set of segments for orthogonal segment intersection queries.

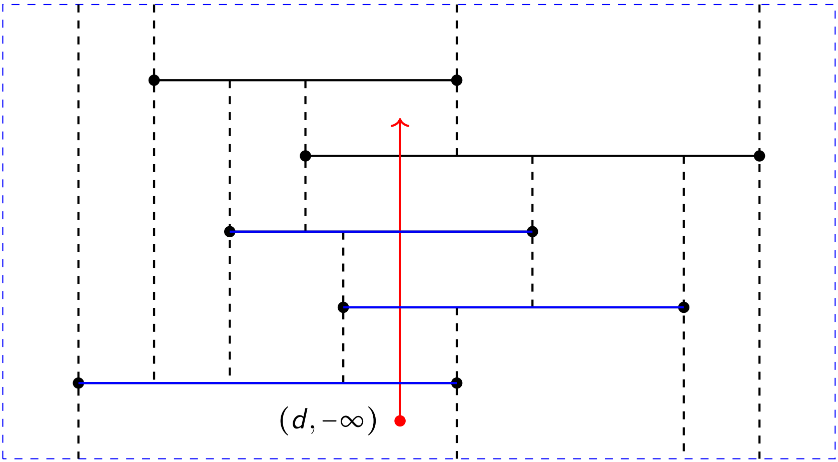
1. Let $P[1:m]$ and $[\alpha, \beta]$ be the query parameters
2. find the node $v \in \mathcal{T}$ at which the search for P terminates
3. Let h be the heavy path containing node v
4. query the associated structure with vertical segment $d \times [\alpha, \beta]$

Hive Graph



1. Let $P[1:m]$ and integer $k > 0$ be the query parameters.
2. find the node $v \in \mathcal{T}$ at which the search for P terminates
3. Let h be the heavy path on which v lies.
4. query the structure with vertical ray emanating from $(d, -\infty)$, and
   report the first k segments intersected by the ray

Hive Graph



Improving Query Time

- query time in each case is $O(m + \log n + \#output)$
- optimal for the case when m is at least $\log n$.
- improve the query time for the case $m = o(\log n)$
  - store the list of consecutive occurrences at each node v with $str(v) = o(\log n)$, sorted by distance.

bounded-gap query	**top-k query**
$O(m + \log \alpha + \#output)$	$O(m + k)$

Open Questions

1. Improving the space bound?
2. Answering the queries in a substring $T[i:j]$?

Thank you!