Document Listing

- similar to document retrieval (next lecture)
- get all documents containing a phrase

**Definition: Document Listing**

Given a collection of $D$ documents $\mathcal{D} = \{d_1, d_2, \ldots, d_D\}$ containing symbols from an alphabet $\Sigma = [1, \sigma]$ and a pattern $P \in \Sigma^*$, return all $j \in [1, D]$, such that $d_j$ contains $P$. 
Document Listing

- similar to document retrieval (next lecture)
- get all documents containing a phrase

Definition: Document Listing

Given a collection of $D$ documents $\mathcal{D} = \{d_1, d_2, \ldots, d_D\}$ containing symbols from an alphabet $\Sigma = [1, \sigma]$ and a pattern $P \in \Sigma^*$, return all $j \in [1, D]$, such that $d_j$ contains $P$.

- $d_1 = ATA$
- $d_2 = TAAA$
- $d_3 = TATA$

And for queries:
- $P = TA$ is contained in $d_1$, $d_2$, and $d_3$
- $P = ATA$ is contained in $d_1$ and $d_3$
Basic Concepts

Definition: Document Concatenation

Given a collection of $D$ documents $\mathcal{D} = \{d_1, d_2, \ldots, d_D\}$ containing symbols from an alphabet $\Sigma = [1, \sigma]$ where each document ends with a special symbol $\#$ not in $\Sigma$, the string

$$C = d_1 d_2 \ldots d_D \#$$

is called the concatenation of the documents with $\#$ not in $\Sigma$ and $\$ < \# < \alpha$ for all $\alpha \in \Sigma$.

$\text{N} = |C| = \sum_{i=1}^{D} |d_i|$
Definition: Document Concatenation

Given a collection of $D$ documents $\mathcal{D} = \{d_1, d_2, \ldots, d_D\}$ containing symbols from an alphabet $\Sigma = [1, \sigma]$ where each document ends with a special symbol $\# \notin \Sigma$, the string

$$C = d_1 d_2 \ldots d_D$$

is called the concatenation of the documents with $\# \notin \Sigma$ and $\$ < \# < \alpha$ for all $\alpha \in \Sigma$.

$$N = |C| = \sum_{i=1}^{D} |d_i|$$

- $d_1 = ATA$
- $d_2 = TAAA$
- $d_3 = TATA$

Document Concatenation:

- $ATA#TAAA#TATA#$
Suffix Array for Document Concatenation

- given a document concatenation $C$, build the suffix array
- requires $O(n)$ time
- entries in suffix array correspond to documents
Suffix Array for Document Concatenation

given a document concatenation $C$, build the suffix array
requires $O(n)$ time
entries in suffix array correspond to documents

Definition: Document Array

Given a document concatenation $C$ and its suffix array $SA$, the document array $DA$ is defined as

$$DA[i] = j \iff \sum_{k=1}^{j-1} |d_k| < SA[i] \leq \sum_{k=1}^{j} |d_k|$$

for $i > 1$ and $DA[1] = 0$
Suffix Array for Document Concatenation

- given a document concatenation $C$, build the suffix array
- requires $O(n)$ time
- entries in suffix array correspond to documents

**Definition: Document Array**

Given a document concatenation $C$ and its suffix array $SA$, the document array $DA$ is defined as

$$DA[i] = j \iff \sum_{k=1}^{j-1} |d_k| < SA[i] \leq \sum_{k=1}^{j} |d_k|$$

for $i > 1$ and $DA[1] = 0$
Suffix Array for Document Concatenation

- given a document concatenation $C$, build the suffix array
- requires $O(n)$ time
- entries in suffix array correspond to documents

**Definition: Document Array**

Given a document concatenation $C$ and its suffix array $SA$, the document array $DA$ is defined as

$$DA[i] = j \iff \sum_{k=1}^{j-1} |d_k| < SA[i] \leq \sum_{k=1}^{j} |d_k|$$

for $i > 1$ and $DA[1] = 0$
Naive Document Listing

- given document concatenation $C$, its suffix array $SA$, and document array $DA$
- enhance suffix array to do pattern matching in $O(|P|)$ time only briefly discussed in lecture
- find interval in suffix array matching $P$
- report all documents in interval in $DA$
- problem: $O(|P| + N)$ query time very bad

<table>
<thead>
<tr>
<th>$T$</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>#</th>
<th>T</th>
<th>A</th>
<th>A</th>
<th>#</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>#</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SA$</td>
<td>15</td>
<td>14</td>
<td>4</td>
<td>9</td>
<td>13</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>11</td>
<td>1</td>
<td>12</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$DA$</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Naive Document Listing

given document concatenation $C$, its suffix array $SA$, and document array $DA$

enhance suffix array to do pattern matching in $O(|P|)$ time only briefly discussed in lecture

find interval in suffix array matching $P$

report all documents in interval in $DA$

problem: $O(|P| + N)$ query time very bad

$P = TA$
Naive Document Listing

- given document concatenation $C$, its suffix array $SA$, and document array $DA$
- enhance suffix array to do pattern matching in $O(|P|)$ time only briefly discussed in lecture
- find interval in suffix array matching $P$
- report all documents in interval in $DA$
- problem: $O(|P| + N)$ query time very bad

**$P = TA$**
Naive Document Listing

- given document concatenation $C$, its suffix array $SA$, and document array $DA$
- enhance suffix array to do pattern matching in $O(|P|)$ time only briefly discussed in lecture
- find interval in suffix array matching $P$
- report all documents in interval in $DA$
- problem: $O(|P| + N)$ query time very bad

- is there a better solution?
- better query time
- better (or at least equal) space requirements?

$P = TA$
Definition: Chain Array

Given document concatenation $C$, its suffix array $SA$, and document array $DA$, the chain array $CA$ is defined as

$$CA[i] = \max\{j < i : DA[j] = DA[i]\} \cup \{0\}$$

- chains same documents together
- find lexicographically smaller suffix of same document
- use it to report documents just once
- build RMQ data structure for $CA$

T | A | T | A | # | T | A | A | A | # | T | A | T | A | # | $
---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---
SA | 15 | 14 | 4 | 9 | 13 | 3 | 8 | 7 | 6 | 11 | 1 | 12 | 2 | 5 | 10
DA | 0 | 3 | 1 | 2 | 3 | 1 | 2 | 2 | 3 | 1 | 3 | 1 | 2 | 3
CA | 0 | 0 | 0 | 0 | 2 | 3 | 4 | 7 | 8 | 5 | 6 | 10 | 11 | 9 | 12
Definition: Chain Array

Given document concatenation $C$, its suffix array $SA$, and document array $DA$, the chain array $CA$ is defined as

$$CA[i] = \max\{j < i: DA[j] = DA[i]\} \cup \{0\}$$

- chains same documents together
- find lexicographically smaller suffix of same document
- use it to report documents just once
- build RMQ data structure for $CA$
Definition: Chain Array

Given document concatenation $C$, its suffix array $SA$, and document array $DA$, the chain array $CA$ is defined as

$$CA[i] = \max\{j < i : DA[j] = DA[i]\} \cup \{0\}$$

- chains same documents together
- find lexicographically smaller suffix of same document
- use it to report documents just once
- build RMQ data structure for $CA$

$P = TA$
Definition: Chain Array

Given document concatenation $C$, its suffix array $SA$, and document array $DA$, the chain array $CA$ is defined as

$$CA[i] = \max\{j < i : DA[j] = DA[i]\} \cup \{0\}$$

- chains same documents together
- find lexicographically smaller suffix of same document
- use it to report documents just once
- build RMQ data structure for $CA$
Optimal Time Document Listing (1/2) [Mut02]

Definition: Chain Array

Given document concatenation $C$, its suffix array $SA$, and document array $DA$, the **chain array** $CA$ is defined as

$$CA[i] = \max\{j < i : DA[j] = DA[i]\} \cup \{0\}$$

- chains same documents together
- find lexicographically smaller suffix of same document
- use it to report documents just once
- build RMQ data structure for CA

---

Given the text $\text{TATAATA#TAAA#TA#}$:

- $SA = [15, 14, 4, 9, 13, 3, 8, 7, 6, 11, 1, 12, 2, 5, 10]$
- $DA = [0, 3, 1, 2, 3, 1, 2, 2, 3, 1, 3, 1, 2, 3]$
- $CA = [0, 0, 0, 2, 3, 4, 7, 8, 5, 6, 10, 11, 9, 12]$
- $P = TA$

$P$ is the chain array with RMQs enough to list all documents in optimal time?
**Definition: Chain Array**

Given document concatenation $C$, its suffix array $SA$, and document array $DA$, the chain array $CA$ is defined as

$$CA[i] = \max\{j < i: DA[j] = DA[i]\} \cup \{0\}$$

- chains same documents together
- find lexicographically smaller suffix of same document
- use it to report documents just once
- build RMQ data structure for $CA$

---

**Example:**

<table>
<thead>
<tr>
<th>$T$</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>#</th>
<th>T</th>
<th>A</th>
<th>A</th>
<th>A</th>
<th>#</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>#</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SA$</td>
<td>15</td>
<td>14</td>
<td>4</td>
<td>9</td>
<td>13</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>11</td>
<td>1</td>
<td>12</td>
<td>2</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>$DA$</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$CA$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>11</td>
<td>9</td>
<td>12</td>
</tr>
</tbody>
</table>

$P = TA$

**PINGO** is the chain array with RMQs enough to list all documents in optimal time?
given document concatenation $C$, its suffix array $SA$, document array $DA$, and chain array $CA$ with RMQ data structure

find interval $SA[s, e]$ as before

report document $DA[m]$ only if $CA[m] < s$ for $m \in [s, e]$

$P = TA$
given document concatenation $C$, its suffix array $SA$, document array $DA$, and chain array $CA$ with RMQ data structure
find interval $SA[s, e]$ as before
report document $DA[m]$ only if $CA[m] < s$ for $m \in [s, e]$

find all positions where $CA[m] < s$ with RMQs
get arg min of $CA$ in interval and report $DA[m]$ if $CA[m] < s$
split interval in $[s, m - 1]$ and $[m + 1, e]$ and recurse
ignore intervals where nothing is reported

$P = TA$
Optimal Time Document Listing (2/2)

- Given document concatenation $C$, its suffix array $SA$, document array $DA$, and chain array $CA$ with RMQ data structure.
- Find interval $SA[s, e]$ as before.
- Report document $DA[m]$ only if $CA[m] < s$ for $m \in [s, e]$.

- Find all positions where $CA[m] < s$ with RMQs.
- Get arg min of $CA$ in interval and report $DA[m]$ if $CA[m] < s$.
- Split interval in $[s, m - 1]$ and $[m + 1, e]$ and recurse.
- Ignore intervals where nothing is reported.

$P = TA$

Lemma: Optimal Document Listing

Listing all documents containing a pattern $P$ can be done in $O(|P| + occ)$ time.
**Top-k Document Retrieval for Single-Term Frequencies**

**Definition: Top-k Document Retrieval**

Given a collection of $D$ documents $\mathcal{D} = \{d_1, d_2, \ldots, d_D\}$ containing symbols from an alphabet $\Sigma = [1, \sigma]$, a pattern $P \in \Sigma^*$, and a threshold $k$, return the top-$k$ documents $j \in [1, D]$, such that $d_j$ contains $P$ most often.

- retrieve $occ$ distinct documents where $P$ occurs
- determine frequency of $P$ in each document
- maintain min-heap of (frequency,document)-pairs of size $k$
- total time: $O(|P| + occ(\lg k + \lg N))$
Definition: Top-$k$ Document Retrieval

Given a collection of $D$ documents $\mathcal{D} = \{d_1, d_2, \ldots, d_D\}$ containing symbols from an alphabet $\Sigma = [1, \sigma]$, a pattern $P \in \Sigma^*$, and a threshold $k$, return the top-$k$ documents $j \in [1, D]$, such that $d_j$ contains $P$ most often.

- retrieve $occ$ distinct documents where $P$ occurs
- determine frequency of $P$ in each document
- maintain min-heap of $(frequency, document)$-pairs of size $k$
- total time: $O(|P| + occ(\lg k + \lg N))$

- $occ$ can be $N$
- can we do better
Definition: Top-$k$ Document Retrieval

Given a collection of $D$ documents $\mathcal{D} = \{d_1, d_2, \ldots, d_D\}$ containing symbols from an alphabet $\Sigma = [1, \sigma]$, a pattern $P \in \Sigma^*$, and a threshold $k$, return the top-$k$ documents $j \in [1, D]$, such that $d_j$ contains $P$ most often.

- retrieve $occ$ distinct documents where $P$ occurs
- determine frequency of $P$ in each document
- maintain min-heap of (frequency,document)-pairs of size $k$
- total time: $O(|P| + occ(\lg k + \lg N))$

- $occ$ can be $N$
- can we do better

- optimal solution: $O(|P| + k)$ query time in $O(N \lg N)$ bits [NN12]
- now: $O(|P| + k \lg N)$ [GKN17]
Recap: Suffix Tree

**Definition: Suffix Tree** [Wei73]

A suffix tree (ST) for a text $T$ of length $n$ is a

- compact trie
- over $S = \{ T[1..n], T[2..n], \ldots, T[n..n] \}$

suffixes are prefix-free due to sentinel
Recap: Suffix Tree

Definition: Suffix Tree [Wei73]

A suffix tree (ST) for a text $T$ of length $n$ is a

- compact trie
- over $S = \{T[1..n], T[2..n], \ldots, T[n..n]\}$
  - suffixes are prefix-free due to sentinel

Let $G = (V, E)$ be a compact trie with root $r$ and a node $v \in V$, then

- $\lambda(v)$ is the concatenation of labels from $r$ to $v$
- $d(v) = |\lambda(v)|$ is the string-depth of $v$
  - string depth $\neq$ depth
Recap: Suffix Tree

Definition: Suffix Tree [Wei73]

A suffix tree (ST) for a text $T$ of length $n$ is a

- compact trie
- over $S = \{ T[1..n], T[2..n], \ldots, T[n..n] \}$

 suffixes are prefix-free due to sentinel

Let $G = (V, E)$ be a compact trie with root $r$ and a node $v \in V$, then

- $\lambda(v)$ is the concatenation of labels from $r$ to $v$
- $d(v) = |\lambda(v)|$ is the string-depth of $v$

\[ \lambda(\cdot) = bba$ \]
\[ d(\cdot) = 4 \]
\[ \lambda(\cdot) = abba \]
\[ d(\cdot) = 4 \]
a generalized suffix tree is a suffix tree for a set of strings

document concatenation is a set of strings

Mark Document Numbers

mark all leaves with DA-entry $i$
a generalized suffix tree is a suffix tree for a set of strings

document concatenation is a set of strings

Mark Document Numbers

- mark all leaves with DA-entry \( i \)
- add \( i \) to nodes that are lowest common ancestor of two leaves marked with \( i \)
a generalized suffix tree is a suffix tree for a set of strings

document concatenation is a set of strings

Mark Document Numbers

- mark all leaves with DA-entry $i$
- add $i$ to nodes that are lowest common ancestor of two leaves marked with $i$
Inner Node Names

- leaf index is rank of suffix in $[1, N]$ in leaf
- each inner node gets $v$ gets $id(v)$, which is the leaf index of rightmost leaf in leftmost child
- $id(v) \neq id(w)$ for all inner nodes $v \neq w$
- $id(v) \in [1, N]$
- $id(v) - 1 \in [lb(v), rb(v)]$, with $[lb(v), rb(v)]$ being $v$’s suffix array interval
- example on the board
connect node with id $i$ to closest ancestor containing id $i$

- nodes marked with id $i$ correspond to suffix tree of $d_i$
- document id $i$ occurs at most $|d_i|$ times in leaves and $|d_i| - 1$ times in inner nodes
- there are at most $O(N)$ document ids in the generalized suffix tree
to retrieve documents containing pattern $P$

- select locus of $P$ as first node $v$ with $P$ is prefix of $\lambda(v)$

- per document at most one pointer leaves subtree of locus $v$
- associate each pointer with number of occurrences of documents in pointers source (weight)
- pointer of document $i$ leaving subtree has maximum weight of all document $i$ pointers in subtree
- document listing is listing all documents of pointers leaving subtree
now: report top-\(k\) documents
represent pointers in a grid
for simplicity only weights \(\geq 2\) starting at inner node

assign each pointer to \((x, y)\)-coordinate
- \(x\): \(id\)(source)
- \(y\): \(d\)(target)
each point is associated with pointers weight

given a locus \(v\), all pointers leaving the subtree have \(y\)-coordinate \(< d(v)\)
Representing Pointers on a Grid (2/2)

- grid can be represented using wavelet tree
- range \textit{maximum} query for each level

**Answering Queries**

- find string depth of locus in suffix tree
- answer range query in grid
- if represented as wavelet tree, use RMQs on each level to report top-\(k\) documents
- if \(\leq k\) documents, use document listing
- total time: \(O(m + k \log N)\)
Representing Pointers on a Grid (2/2)

- grid can be represented using wavelet tree
- range maximum query for each level

**Answering Queries**

- find string depth of locus in suffix tree
- answer range query in grid
- if represented as wavelet tree, use RMQs on each level to report top-$k$ documents
- if $\leq k$ documents, use document listing
- total time: $O(m + k \lg N)$
grid can be represented using wavelet tree
range maximum query for each level

Answering Queries
- find string depth of locus in suffix tree
- answer range query in grid
- if represented as wavelet tree, use RMQs on each level to report top-k documents
- if $\leq k$ documents, use document listing
- total time: $O(m + k \log N)$

PINGO how can we represent the pointers in a grid?

example range queries in wavelet trees on the board 🎨
Conclusion and Outlook

This Lecture
- document listing
- top-k document retrieval (single term frequency)

Linear Time Construction

- ST
- SA
- WT
- LZ
- LCP
- BWT
- FM-Index
- $r$-Index
Conclusion and Outlook

This Lecture
- document listing
- top-k document retrieval (single term frequency)

Next Lecture
- inverted indices (by Tim Niklas Uhl)

Linear Time Construction

- ST
- SA
- WT
- LZ
- LCP
- BWT
- FM-Index
- r-Index
Oral Exam

- registration is open
- oral exams 24.02. and 20.03.
- alternative dates possible
- register with our secretary
Bibliography I


