Text Indexing: Lecture 3
Simon Gog – gog@kit.edu
Intermezzo: Top-$k$ document retrieval

Given
- Collection $\mathcal{D}' = \{d_1, \ldots, d_{N-1}\}$
- Each $d_i$ is a string over alphabet $\Sigma' = [2, \sigma]$ sentinel symbol terminated by 1 (also #)
- $\mathcal{D} = \mathcal{D}' \cup d_0$, with $d_0 = 0$
- „Bag of words” query $Q = \{q_0, q_1, \ldots, q_{m-1}\}$ (unordered set of size $m$)

Problem
Given a collection $\mathcal{D}$, a query $Q$ of length $m$, and a similarity measure $S : \mathcal{D} \times \mathcal{P}_m(\Sigma') \rightarrow \mathbb{R}$. Calculate the top-$k$ documents of $\mathcal{D}$ with regard to $Q$ and $S$. That is a sorted list of document identifiers $T = \{\tau_0, \ldots, \tau_{k-1}\}$, with $S(d_{\tau_i}, Q) \geq S(d_{\tau_{i+1}}, Q)$ for $0 \leq i < k$ and $S(d_{\tau_{k-1}}, Q) \geq S(d_j, Q)$ for $j \notin T$. 
Example

Fix a concatenation $C$ of $D$.

$$i = \begin{array}{ccccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
\end{array}
$$

$$C_{\text{word}} = \begin{array}{ccccccccccccccc}
\text{LA} & \text{O} & \text{LA} & \# & \text{O} & \text{LA} & \text{LA} & \text{LA} & \# & \text{O} & \text{O} & \text{LA} & \# & \$ \\
\end{array}
$$

$$C = \begin{array}{ccccccccccccccc}
2 & 3 & 2 & 1 & 3 & 2 & 2 & 2 & 1 & 3 & 3 & 2 & 1 & 0 \\
\end{array}
$$

- $S_{\text{sfreq}}(d, q) := f_{d,q}$ (i.e. single term frequency ranking)
- $S_{\text{sfreq}}(d_0, \text{LA}) = 0,$
  $S_{\text{sfreq}}(d_1, \text{LA}) = 2,$
  $S_{\text{sfreq}}(d_2, \text{LA}) = 1,$
  $S_{\text{sfreq}}(d_3, \text{LA}) = 3.$
- Top-2: $T = \{3, 1\}$
Okapi BM25 similarity measure

Successful IR similarity measure:

\[
S_{Q,d}^{BM25} = \sum_{q \in Q} \frac{(k_1 + 1)f_{d,q}}{k_1 (1 - b + b \frac{n_d}{n_{avg}}) + f_{d,q}} \cdot f_{Q,q} \cdot \ln \left( \frac{N - F_{D,q} + 0.5}{F_{D,q} + 0.5} \right)
\]

\[
= w_{d,q}
\]

\[
= w_{Q,q}
\]

depends on 3 document-dependent factors:

- \( f_{d,q} \) term frequency
- \( F_{D,q} \) document frequency (# of distinct \( d \)s which contain \( q \))
- \( n_d \) length of document \( d \)
Other similarity measures

- Static weighting (e.g. Page-Rank)
- Language Model (Compute probability to generate the query using the text statistics of each document)
- Vector space model (compute the cosine of the angle in $\sigma$-dimensional space between a query vector and document vector)
- Zone ranking (e.g. words which appear in the title of a web page weight more than words in the body)

More details in survey of Zobel & Moffat [5].
The Inverted Index (IVI)

The classical index in Information Retrieval

For each term $q$ (excluding sentinel symbols)

- a list of pairs of document id and document frequency is stored
- pairs are ordered according to document ids
- the document frequency (=list length) is stored

Sequential processing is used to calculate the ranking function, i.e. query complexity dependent on document frequency.

Example (for collection of last lecture)

$$LA : \{(1, 2), (2, 1), (3, 3)\} \quad F_{D,LA} = 3$$

$$O : \{(1, 1), (2, 2), (3, 1)\} \quad F_{D,O} = 3$$
Another example

\(d_1\) : is big data really big
\(d_2\) : is it big in science
\(d_3\) : big data is big

Inverted Lists

- **big** : \{(1,2), (2,1), (3,2)\}
- **data** : \{(1,1), (3,1)\}
- **in** : \{(2,1)\}
- **is** : \{(1,1), (2,1), (3,1)\}
- **really** : \{(1,1)\}
- **science** : \{(2,1)\}

\[ F_{D, \text{big}} = 3 \]
\[ F_{D, \text{data}} = 2 \]
\[ F_{D, \text{in}} = 1 \]
\[ F_{D, \text{is}} = 3 \]
\[ F_{D, \text{really}} = 1 \]
\[ F_{D, \text{science}} = 1 \]
Inverted Index (IVI)

Possible IVI representation

- use Elias-Fano coding to store the increasing list of document ids $ID_q$
- unary code the list of frequencies decreased by one (i.e. each frequency $x$ is represented by $x$ bits)

Example (for collection of last lecture)

<table>
<thead>
<tr>
<th>LA</th>
<th>$L_{LA} = 1, 2, 3$</th>
<th>011001</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>$L_{O} = 1, 2, 3$</td>
<td>1011</td>
</tr>
</tbody>
</table>

Note: It is not possible to answer phrase queries (e.g. "LA O") with this variant of IVI. Direct support of arbitrary phrase queries would require $O(n^2)$ lists.
Inverted Index (IVI)

Let \(n = \sum_{d \in \mathcal{D}} n_d\) and \(f_{\mathcal{D},q} = \sum_{d \in \mathcal{D}} f_{d,q}\). Space consumption of this representation:

\[
\sum_{q \in \Sigma} 2F_{\mathcal{D},q} + F_{\mathcal{D},q} \log \frac{N}{F_{\mathcal{D},q}} + o(F_{\mathcal{D},q}) + f_{\mathcal{D},q} \leq 3n + o(n) + O(\sigma \cdot \log n) + \sum_{q \in \Sigma} F_{\mathcal{D},q} \log \frac{n}{F_{\mathcal{D},q}}
\]

\[
\sum_{q \in \Sigma} F_{\mathcal{D},q} \log \frac{n}{F_{\mathcal{D},q}} \leq 3n + o(n) + O(\sigma \cdot \log n) + n \sum_{q \in \Sigma} \frac{f_{\mathcal{D},q}}{n} \log \frac{n}{f_{\mathcal{D},q}}
\]

\[
= nH_0(\mathcal{D}) + 3n + o(n) + O(\sigma \cdot \log n)
\]

* Assuming \(f_{\mathcal{D},q} < n/2\) for all \(q\).
Let $m \geq 0$ and $m + 1 \leq \frac{n}{2}$. Then it holds

$$m \cdot \log \frac{n}{m} \leq (m + 1) \cdot \log \frac{n}{m + 1}$$

since

$$(m + 1) \cdot \log \frac{n}{m + 1} - m \cdot \log \frac{n}{m}$$

$$= -m \cdot \log \frac{m + 1}{m} + \log \frac{n}{m + 1}$$

$$\leq -m \cdot \log \frac{m + 1}{m} - 1 + \log \frac{n}{m + 1}$$

$$= -\ln 2 + \log \frac{n}{m + 1}$$

for $m + 1 \leq \frac{n}{2}$.
Outline:

- Greedy top-$k$ framework for single-term (single-phrase) queries in $O(n \log n)$ bits of space
- Optimal query time top-$k$ framework for single-term (single-phrase) in $O(n \log n)$ bits of space [3, 2]
- Greedy top-$k$ framework for multi-term queries in $O(n \log n)$ bits of space
The GREEDY framework

Self-Index Based System
The GREEDY framework for single term $f_{d,q}$-ranking of Culpepper et al. [1] consists of
- a Compressed Suffix Array (CSA) of concatenation $D$
- Wavelet Tree of the Document Array of $D$

Document Array $D$
Array of length $n$. For each suffix $SA[i]$ the document array entry $D[i]$ contains the identifier of the document, in which suffix $SA[i]$ starts.

We denote a suffix array/suffix tree as generalized suffix array/suffix tree when this information was added.
The GREEDY framework

\[ T = \omega_2 \omega_1 \omega_3 \omega_3 \# \omega_1 \omega_1 \omega_4 \omega_1 \# \omega_1 \omega_4 \omega_3 \omega_1 \# \omega_5 \omega_5 \# \]
\[ b = 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 \]

Interval of \( q = \omega_1 \) in \( D \) corresponds to the (multi)set of documents which contain \( q \).
The GREEDY framework

- Represent document array $\mathcal{D}$ as wavelet tree $WTD$
- Example: Search top-2 documents (frequency based ranking)

Top documents containing $\omega_1$:
The GREEDY framework

- Represent document array $D$ as wavelet tree $WTD$
- Example: Search top-2 documents (frequency based ranking)

Top documents containing $\omega_1$:
The GREEDY framework

- Represent document array $D$ as wavelet tree $WTD$
- Example: Search top-2 documents (frequency based ranking)

Top documents containing $\omega_1$:
The GREEDY framework

- Represent document array $\mathcal{D}$ as wavelet tree $WTD$
- Example: Search top-2 documents (frequency based ranking)

Top documents containing $\omega_1$:
The GREEDY framework

- Represent document array $\mathcal{D}$ as wavelet tree $WTD$
- Example: Search top-2 documents (frequency based ranking)

Top documents containing $\omega_1$:
The GREEDY framework

- Represent document array $\mathcal{D}$ as wavelet tree $WTD$
- Example: Search top-2 documents (frequency based ranking)

Top documents containing $\omega_1$: expand and push
The GREEDY framework

- Represent document array $\mathcal{D}$ as wavelet tree $WTD$
- Example: Search top-2 documents (frequency based ranking)

Top documents containing $\omega_1: d_1$ (3 times)
The GREEDY framework

- Represent document array $\mathcal{D}$ as wavelet tree $WTD$
- Example: Search top-2 documents (frequency based ranking)

Top documents containing $\omega_1: d_1$ (3 times)
The GREEDY framework

- Represent document array $D$ as wavelet tree $WTD$
- Example: Search top-2 documents (frequency based ranking)

Top documents containing $\omega_1$: $d_1$ (3 times)
The GREEDY framework

- Represent document array $D$ as wavelet tree $WTD$
- Example: Search top-2 documents (frequency based ranking)

```
012312110220001233
001101000110000111
```

Top documents containing $\omega_1$: $d_1$ (3 times), $d_2$ (2 times)
ranked_search(CSA, WTD, q, k)

\[ [l, r] \leftarrow \text{backward_search}(CSA, q) \]

\[ \text{pq.push}(\langle r - l + 1, [l, r], \text{WTD.root}() \rangle) \]

\[ h \leftarrow 0 \]

\[ \text{while } h < k \text{ and not } \text{pq.empty()} \text{ do} \]

\[ \langle s, [l, r], v \rangle \leftarrow \text{pq.pop()} \]

\[ \text{if } \text{WTD.is_leaf}(v) \text{ then} \]

\[ \text{output } \langle \text{WTD.symbol}(v), s \rangle \]

\[ h \leftarrow h + 1 \]

\[ \text{else} \]

\[ \langle \langle [l_i, r_i], v_i \rangle, \langle [l_r, r_r], v_r \rangle \rangle \leftarrow \text{WTD.expand}(v, [l, r]) \]

\[ \text{pq.push}(\langle r_l - l_l + 1, [l_l, r_l], v_l \rangle) \]

\[ \text{pq.push}(\langle r_r - l_r + 1, [l_r, r_r], v_r \rangle) \]

Max-Priority-Queue pq sorted according to interval size.
The GREEDY framework

To show:
(a) GREEDY returns the correct result
(b) WT method expand runs in constant time

Exercise.

Improving the algorithm
Let \(v_\omega\) be a WT node which represents the sub-collection \(D_\omega\) (i.e. all documents whose id are prefixed by \(\omega\)). The interval size for query \(q\) at node \(v_\omega\) is an upper bound for \(\max_{d \in D_\omega} \{f_d, q\}\).
Better upper bound by subtracting the document frequency \(F_{D_\omega, q}\) of \(q\) in sub-collection \(D_\omega\) and adding one.
Build binary generalized suffix tree $BGST$. 
For each inner node $v$ in $BGST$ keep a list $L_v$ of repeated documents. 
A document $d$ is added to $L_v$ if $d$ occurs in a leaf of the left and right subtree.
For a pattern $q$ let $v_q$ be the locus (i.e. the lowest node which path is prefixed by $q$)
$F_{D,q}$ equals the number of leaves in the subtree of $v_q$ minus the number of repeated documents ($\sum_{v \in T_{v_q}} |L'_v|$) in $v_q$’s subtree $T_{v_q}$.
Nodes are numbered in-order.
Traverse node in-order and append $|L_v|$ in unary coding to bitvector $H$, which was initialized with a single 1.
As all nodes in subtrees are contiguous the number of repeated documents can be calculated by two select queries.
Document Frequency $F_{D,q}$

$$H = \begin{bmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1
\end{bmatrix}$$
Document Frequency $F_{D,q}$

Solution of [4]:
- $H$ is at most $2n - N$ bits
- add $o(n)$-bit select structure
- Use CSA to get SA-interval

For $[l, r] \leftarrow \text{backward_search}(CSA, q)$:

```
00  def document_frequency(H, [l, r])
01      s \leftarrow r - l + 1
02      y \leftarrow \text{select}(H, r, 1)
03      if l = 0 then
04          return s - (y - r + 1)
05      else
06          x \leftarrow \text{select}(H, l, 1)
07          return s - (y - r + 1 - (x - l + 1))
```
Calculating $F_{D_v,q}$

- Let $D_v$ be the subset of documents which are represented by a node $v$ of the wavelet tree over the document array.
- How can we calculate $F_{D_v,q}$ efficiently?
- Introduce repetition array $R$.
- The repetition array contains for each 0 in $H$ the corresponding repeated element of the associated node.
- Map SA-interval of $q$ to $R$ using select operation on $H$.
- To get $F_{D_v,q}$ use the expand method (constant time per WT level).

More details and practical results on the GREEDY framework are available here: http://arxiv.org/abs/1406.3170
Document Frequency for Subsets: Repetition Array

$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$

$R = \begin{bmatrix} 1 & 2 & 3 & 3 & 3 & 1 & 1 & 2 & 3 & 2 \end{bmatrix}$


