Text Indexing: Lecture 3

Simon Gog – gog@ira.uka.de
Exercise: select(i, c, X)

```
00  select(i, c, X)
01       d ← 0
02       v ← WT.root()
03   while not WT.is_leaf(v) do
04       v ← WT.select_child(v, code(c)[d])
05       d ← d + 1
06       s ← i
07   while not WT.root() = v do
08       s ← WT.select(B_v, code(v)[d], s)
09       d ← d − 1
10       v ← WT.parent(v)
11  return s
```

code(c) is the prefix code of symbol c; B_v is bitvector of node v.
Query time: $O(\log \sigma)$.
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 $D$.

<table>
<thead>
<tr>
<th>$i$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{\text{word}}$</td>
<td>LA</td>
<td>O</td>
<td>LA</td>
<td>#</td>
<td>O</td>
<td>LA</td>
<td>LA</td>
<td>LA</td>
<td>#</td>
<td>O</td>
<td>O</td>
<td>LA</td>
<td>#</td>
<td>$$</td>
</tr>
<tr>
<td>$C$</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>-</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>-</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

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

Successful IR similarity measure:

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

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 ranking functions

- 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 and Moffat [2006].
Exhaustive search
Consider every $d_i$, calculate $S(d_i, Q)$ and keep top-$k$ results in a min-heap of size $k$.

- Exhaustive search is rank-safe but
- very slow
- indexes help to make rank-safe approach feasible
  - $d_i$s has to contain at least one query term
  - precomputed scores used to skip the evaluation of documents

Heuristics
Many very efficient systems are not rank-safe, but authors argue that effectiveness of their results is close to optimal result.
Inverted Index (II)
For each term $t_i$ store a postings list. Each posting consists of a (document,frequency)-pair.
- Very small (compressed, $\approx 6$ GiB for GOV2)
- Limited functionality (choice of $S$, phrases, snipped extraction)

Positional Inverted Index (PII)
Inverted index + positions of occurrences inside documents.
- Larger (compressed, $\approx 30$ GiB for GOV2)
- Drawbacks of II, but proximity based re-ranking, phrases a bit faster
Documents (already normalized)

\( d_0 \) : is big data really big
\( d_1 \) : is it big in science
\( d_2 \) : big data is big

Inverted Lists

- big : \{(0,1),(0,4),(1,2),(2,0),(2,3)\}
- data : \{(0,2),(2,1)\}
- in : \{(1,3)\}
- is : \{(0,0),(1,0),(2,2)\}
- really : \{(0,3)\}
- science : \{(1,4)\}
Limitations

- Vocabulary has to be known at indexing time
- Query time depends on the number of occurrences of the query terms

Adapt input to index

- Input is parsed into words
- Words are normalized (lowercasing, stemming)
- „Stopwords” are removed
Self-Index Based System

The GREEDY framework for single term $f_{d,q}$-ranking of Culpepper et al. [2010] consists of

- a Compressed Suffix Array (CSA) of concatenation $\mathcal{D}$
- Wavelet Tree of the Document Array of $\mathcal{D}$

Document Array $\mathcal{D}$

Array of length $n$. For each suffix $SA[i]$ the document array entry $\mathcal{D}[i]$ 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.
Explaining GREEDY

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

\[ D = \begin{array}{cccccccccccccccc}
0 & 1 & 2 & 3 & 1 & 2 & 1 & 1 & 0 & 2 & 2 & 0 & 0 & 0 & 1 & 2 & 3 & 3
\end{array} \]

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

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

Top documents containing \( \omega_1 \):
Represent document array $D$ as wavelet tree $WTD$
Example: Search top-2 documents (frequency based ranking)

Top documents containing $\omega_1$:
Explaining GREEDY

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

Top documents containing $\omega_1$: 

[Diagram showing wavelet tree with nodes and numbers]
Represent document array $D$ as wavelet tree $WTD$

Example: Search top-2 documents (frequency based ranking)

Top documents containing $\omega_1$: expand ($O(1)$ time) and push
Represent document array $D$ as wavelet tree $WTD$

Example: Search top-2 documents (frequency based ranking)

Top documents containing $\omega_1$: 

```
012312110220001233
001101000110000111
```

```
0111100001
0111100001
00000
11111
22222
333
23222233
01000011
```
Represent document array $\mathcal{D}$ as wavelet tree $WTD$

Example: Search top-2 documents (frequency based ranking)

Top documents containing $\omega_1$:
Explaining GREEDY

- 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)
Explaining GREEDY

- 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)
Explaining GREEDY

- 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)
Explaining GREEDY

- 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), $d_2$ (2 times)
Pseudo Code

```plaintext
ranked_search(CSA, WTD, q, k)

\[ [l, r] \leftarrow CSA.search(q) \]

\[ pq.push(⟨r - l + 1, [l, r], WTD.root()⟩) \]

\[ h \leftarrow 0 \]

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

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

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

\[ \text{output } ⟨WTD.symbol(v), s⟩ \]

\[ h \leftarrow h + 1 \]

\[ \text{else} \]

\[ \langle [l, r], v_l \rangle, \langle [l, r], v_r \rangle \leftarrow WTD.expand(v, [l, r]) \]

\[ pq.push(⟨r_l - l_l + 1, [l_l, r_l], v_l⟩) \]

\[ pq.push(⟨r_r - l_r + 1, [l_r, r_r], v_r⟩) \]

Max-Priority-Queue \( pq \) sorted according to interval size.
```
Generalized GREEDY

- multi-term (state consists of multiple intervals)
- ranked-and (conjunctive) or ranked-or (disjunctive) version
  - ranked-and: all intervals have to be present in a child state
  - ranked-or: at least on interval has to be present in a child state
- more complex similarity measures: $\text{TF} \times \text{IDF}$, BM25, LMDS

Better estimation for $\max_{d \in D_v} \{f_{d,q}\}$?

see blackboard
Document Frequency

\[ H = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix} \]
Document Frequency

Solution of Sadakane [2007]:
- $H$ is at most $2n$ bits
- add $o(n)$-bit select structure

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

00  \textbf{document_frequency}(H, [l, r])
01      s \leftarrow r - l + 1
02      y \leftarrow \text{select}(H, r, 1)
03  \textbf{if} \ l = 0 \ \textbf{then}
04      \textbf{return} \ s - (y - r + 1)
05  \textbf{else}
06      x \leftarrow \text{select}(H, l, 1)
07      \textbf{return} \ s - (y - r + 1 - (x - l + 1))
More WT operations: Query point grids

Operations:

\[
\text{count}(x_0, x_1, y_0, y_1) \quad \# \text{ points in rectangle } [x_0, x_1][y_0, y_1].
\]

\[
\text{report}(x_0, x_1, y_0, y_1) \quad \text{Report points in rectangle } [x_0, x_1][y_0, y_1].
\]
