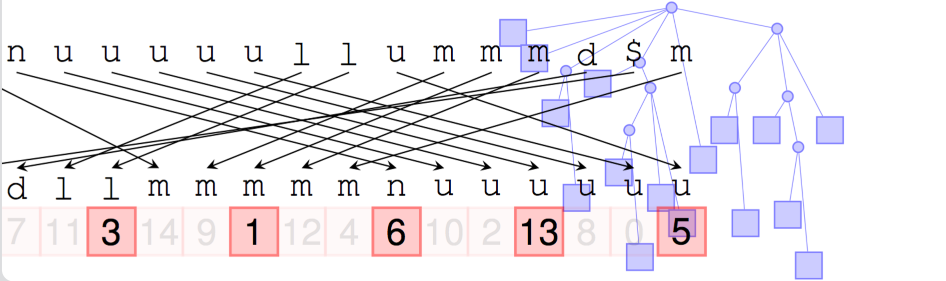
Karlsruhe Institute of Technology

Text Indexing: Lecture 3

Simon Gog – *gog@kit.edu*

Institute of Theoretical Informatics - Algorithmics



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

Example

Fix a concatenation $\mathcal{C}$ of $\mathcal{D}$.

$$
\begin{array}{rcccccccccccccc}
i = & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
\mathcal{C}^{word} = & LA & O & LA & \# & O & LA & LA & LA & \# & O & O & LA & \# & \$ \\
\mathcal{C} = & \underbrace{2 \quad 3 \quad 2 \quad 1}_{d_1} & & & & \underbrace{3 \quad 2 \quad 2 \quad 2 \quad 1}_{d_3} & & & & & \underbrace{3 \quad 3 \quad 2 \quad 1}_{d_2} & & & & \underbrace{0}_{d_0}
\end{array}
$$

- $\mathcal{S}^{sfreq}(d, q) := f_{d,q}$ (i.e. single term frequency ranking)
- $\mathcal{S}^{sfreq}(d_0, LA) = 0,$
  $\mathcal{S}^{sfreq}(d_1, LA) = 2,$
  $\mathcal{S}^{sfreq}(d_2, LA) = 1,$
  $\mathcal{S}^{sfreq}(d_3, LA) = 3.$
- Top-2: $\mathrm{T} = \{3, 1\}$

Okapi BM25 similarity measure

Successful IR similarity measure:

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

depends on 3 document-dependent factors:

- $f_{d,q}$ term frequncy
- $F_{\mathcal{D},q}$ 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 σ-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 : $\{(1,2),(2,1),(3,3)\}$ $\quad F_{\mathcal{D},\mathsf{LA}} = 3$

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

The Inverted Index (IVI)

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	:	$\{(1,2),(2,1),(3,2)\}$	$F_{\mathcal{D},\text{big}} = 3$
data	:	$\{(1,1),(3,1)\}$	$F_{\mathcal{D},\text{data}} = 2$
in	:	$\{(2,1)\}$	$F_{\mathcal{D},\text{in}} = 1$
is	:	$\{(1,1),(2,1),(3,1)\}$	$F_{\mathcal{D},\text{is}} = 3$
really	:	$\{(1,1)\}$	$F_{\mathcal{D},\text{really}} = 1$
science	:	$\{(2,1)\}$	$F_{\mathcal{D},\text{science}} = 1$

Inverted Index (IVI)

Karlsruhe Institute of Technology

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)

LA : $L_{\mathrm{LA}} = 1, 2, 3$ `011001`

O : $L_{\mathrm{O}} = 1, 2, 3$ `1011`

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:

$$
\begin{aligned}
&= \underbrace{\sum_{q \in \Sigma} 2F_{\mathcal{D},q} + F_{\mathcal{D},q} \log \frac{N}{F_{\mathcal{D},q}} + o(F_{\mathcal{D},q})}_{\text{document ids}} + \underbrace{f_{\mathcal{D},q}}_{\text{frequencies}} + \underbrace{O(\log n)}_{\text{pointer}} \\
&\leq 3n + o(n) + O(\sigma \cdot \log n) + \sum_{q \in \Sigma} F_{\mathcal{D},q} \log \frac{n}{F_{\mathcal{D},q}} \\
&\overset{*}{\leq} 3n + o(n) + O(\sigma \cdot \log n) + n \sum_{q \in \Sigma} \frac{f_{\mathcal{D},q}}{n} \log \frac{n}{f_{\mathcal{D},q}} \\
&= n\mathcal{H}_0(\mathcal{D}) + 3n + o(n) + O(\sigma \cdot \log n)
\end{aligned}
$$

* 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

$$\begin{aligned}
& (m+1) \cdot \log \frac{n}{m+1} - m \cdot \log \frac{n}{m} \\
= \quad & -m \cdot \log \frac{m+1}{m} + \log \frac{n}{m+1} \\
\overset{\log x \leq \ln 2 (x-1)}{\geq} \quad & -m \cdot \ln 2 \left(\frac{m+1}{m} - 1 \right) + \log \frac{n}{m+1} \\
= \quad & -\ln 2 + \log \frac{n}{m+1} \overset{m+1 \leq \frac{n}{2}}{\geq} 0
\end{aligned}$$

Suffix-Index Based Retrieval Systems

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 $\mathcal{D}$
- Wavelet Tree of the Document Array of $\mathcal{D}$

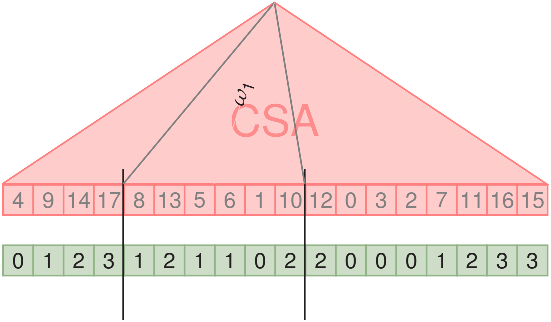
Document Array $\mathcal{D}$

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

We denote a suffix array/suffix tree as *generalized* suffix array/suffix tree when this information was added.

The GREEDY framework

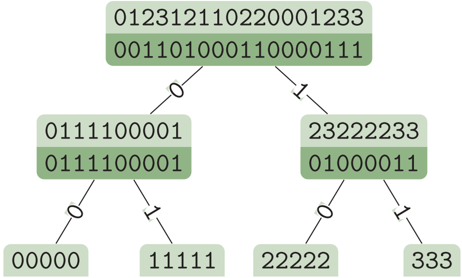
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\mathcal{T} =$	ω_2	ω_1	ω_3	ω_3	#	ω_1	ω_1	ω_4	ω_1	#	ω_1	ω_4	ω_3	ω_1	#	ω_5	ω_5	#
$b =$	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	1



Interval of $q = \omega_1$ in $\mathcal{D}$ corresponds to the (multi)set of documents which contain q.

The GREEDY framework

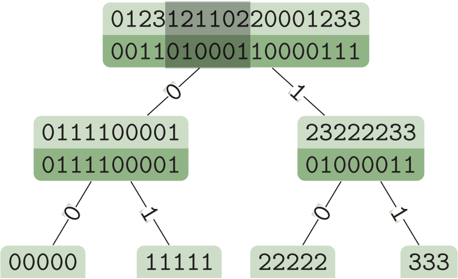
- Represent document array $\mathcal{D}$ as wavelet tree *WTD*
- Example: Search top-2 documents (frequency based ranking)



Top documents containing ω_1:

The GREEDY framework

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

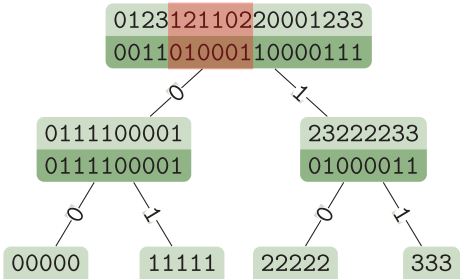


push state

Top documents containing ω_1:

The GREEDY framework

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

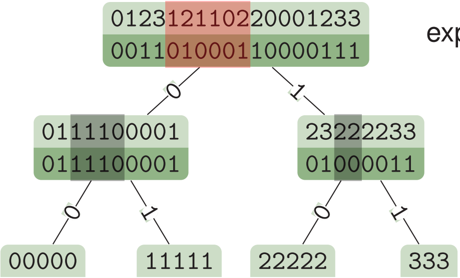


pop state

Top documents containing ω_1:

The GREEDY framework

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

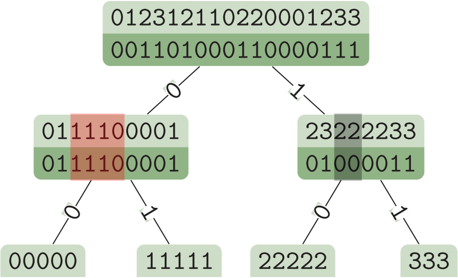


expand ($\mathcal{O}(1)$ time) and push

Top documents containing ω_1:

The GREEDY framework

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

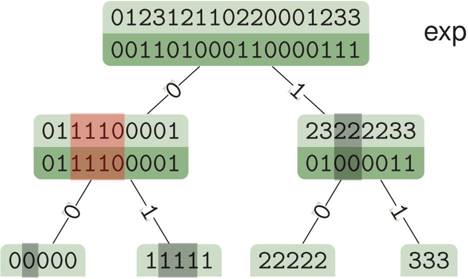


pop state

Top documents containing ω_1:

The GREEDY framework

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

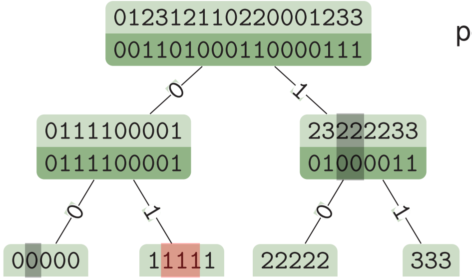


expand and push

Top documents containing ω_1:

The GREEDY framework

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

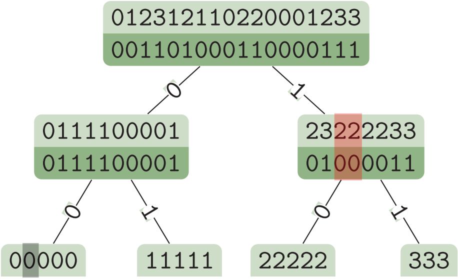


pop and report

Top documents containing ω_1: d_1 (3 times)

The GREEDY framework

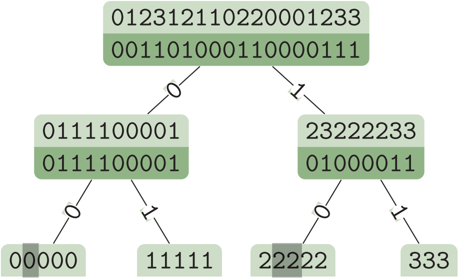
- Represent document array $\mathcal{D}$ as wavelet tree *WTD*
- Example: Search top-2 documents (frequency based ranking)



Top documents containing ω_1: d_1 (3 times)

The GREEDY framework

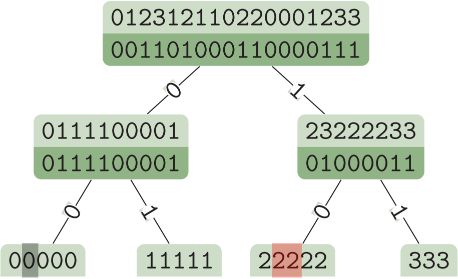
- Represent document array $\mathcal{D}$ as wavelet tree *WTD*
- Example: Search top-2 documents (frequency based ranking)



Top documents containing ω_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 ω_1: d_1 (3 times), d_2 (2 times)

Pseudo Code

```
ranked_search(CSA, WTD, q, k)
    [l, r] ← backward_search(CSA, q)
    pq.push(⟨r−l+1, [l, r], WTD.root()⟩)
    h ← 0
    while h < k and not pq.empty() do
        ⟨s, [l, r], v⟩ ← pq.pop()
        if WTD.is_leaf(v) then
            output ⟨WTD.symbol(v), s⟩
            h ← h + 1
        else
            ⟨⟨[l_l, r_l], v_l⟩, ⟨[l_r, r_r], v_r⟩⟩ ← 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.

The GREEDY framework

Karlsruhe Institute of Technology

To show:

(a) GREEDY return the correct result
(b) WT method expand runs in constant time

See blackboard.

Improving the algorithm

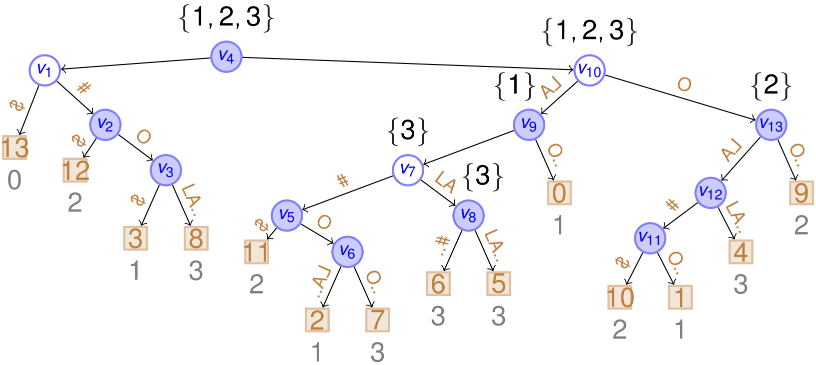
Let v_ω be a WT node which represents the sub-collection $\mathcal{D}_\omega$ (i.e. all documents whose id are prefixed by ω). The interval size for query q at node v_ω is an upper bound for $\max_{d \in \mathcal{D}_\omega}\{f_{d,q}\}$.
Better upper bound by subtracting the document frequency $F_{\mathcal{D}_\omega,q}$ of q in sub-collection $\mathcal{D}_\omega$ and adding one.

Document Frequency $F_{\mathcal{D},q}$

Karlsruhe Institute of Technology

- 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)
- $F_{\mathcal{D},q}$ 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 $F_{\mathcal{D},q}$



$$H = \underline{\ }^{v_1}\, 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 1$$

Document Frequency $F_{\mathcal{D},q}$

Solution of [4]:

- H is at most $2n - N$ bits
- add $o(n)$-bit select structure
- Use CSA to get SA-interval

For $[l, r] \leftarrow \mathit{backward\_search}(\mathit{CSA}, q)$:

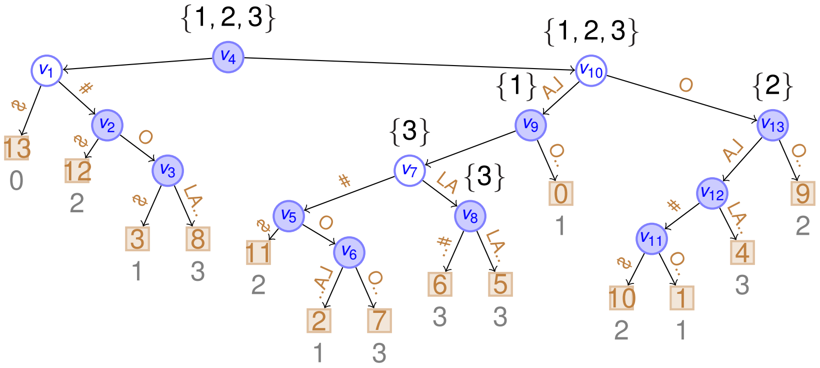
document_frequency$(H, [l, r])$
 $s \leftarrow r - l + 1$
 $y \leftarrow \mathit{select}(H, r, 1)$
 if $l = 0$ **then**
  return $s - (y - r + 1)$
 else
  $x \leftarrow \mathit{select}(H, l, 1)$
  return $s - (y - r + 1 - (x - l + 1))$

Calculating $F_{\mathcal{D}_v,q}$

- Let $\mathcal{D}_v$ be the subset of documents which are represented by a node v of the wavelet tree over the document array
- How can we calculate $F_{\mathcal{D}_v,q}$ 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 $F_{\mathcal{D}_v,q}$ 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



		v_1	v_2	v_3				v_4	v_5	v_6		v_7		v_8		v_9				v_{10}	v_{11}	v_{12}		v_{13}
$H =$	1	1	1	1	0	0	0	1	1	1	0	1	0	1	0	1	0	0	0	1	1	1	0	1
$R =$					1	2	3				3		3		1		1	2	3				2	

Bibliography

[1] J. Shane Culpepper, Gonzalo Navarro, Simon J. Puglisi, and Andrew Turpin. Top-k ranked document search in general text databases. In *Proc. ESA*, pages 194–205, 2010.

[2] Simon Gog and Gonzalo Navarro. Improved single-term top-k document retrieval. In *Proc. ALENEX*, pages 24–32, 2015.

[3] Gonzalo Navarro and Yakov Nekrich. Top-k document retrieval in optimal time and linear space. In *Proc. SODA*, pages 1066–1077, 2012.

[4] Kunihiko Sadakane. Succinct data structures for flexible text retrieval systems. *J. Discrete Alg.*, 5(1):12–22, 2007.

[5] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comp. Surv.*, 38(2), 2006.