

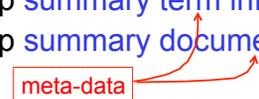
Using and storing the index

1

Review: Inverted Index

- For each term, keep list of document entries, one for each document in which it appears: a **postings list**
 - Document entry is list of positions at which term occurs and attributes for each occurrence: a **posting**
- Keep **summary term information**
- Keep **summary document information**

meta-data



2

Consider “advanced search” queries

Content Coordination

- Phrases
- Numeric range
- NOT
- OR

Document Meta-data

- Language
- Geographic region
- File format
- Date published
- From specific domain
- Specific licensing rights
- Filtered by “safe search”

Issue of **efficient retrieval**

3

Basic retrieval algorithms

- One term
- AND of several terms
- OR of several terms
- NOT term
- proximity

4

Basic postings list processing: Merging posting lists

- Have two lists must **coordinate**
 - Find shared entries and do “something”
 - “something” changes for different operations
 - Set operations UNION? INTERSECTION? DIFFERENCE? ...
 - Filter with document meta-data as process

5

Basic retrieval algorithms

- One term:
 - look up posting list in (inverted) index
- AND of several terms:
 - **Intersect** posting lists of the terms: **a list merge**
- OR of several terms:
 - **Union** posting lists of the terms
 - eliminate **duplicates**: **a list merge**
- NOT term
 - If *terms* AND NOT(*other terms*), take a difference
 - **a list merge** (similar to AND)
- Proximity
 - **a list merge** (similar to AND)

6

Merging two unsorted lists

- **X** Read 2nd list **over and over** - once for each entry on 1st list
 - computationally expensive
 - time $O(|L_1|*|L_2|)$ where $|L|$ length list L
- **Build hash table** on entry values; insert entries of one list, then other; look for collisions
 - must have good hash table
 - unwanted collisions expensive
 - often can't fit in memory: disk version
- **Sort lists**; use algorithm for sorted lists
 - often lists on disk: external sort
 - can sort in $O(|L| \log |L|)$ operations

7

Sorted lists

- Lists sorted by some identifier
 - same identifier both lists; not nec. unique
- Read both lists in “parallel”
 - **Classic list merge**:
(sorted list₁, sorted list₂) ⇒ sorted set union
 - General merge: if no duplicates, get time $|L_1|+|L_2|$
- Build lists so sorted
 - pay cost at most once
 - maybe get sorted order “naturally”
- If only one list sorted, can do binary search of sorted list for entries of other list
 - Must be able to binary search! - **rare!**
 - can't binary search disk

8

Duplicates in sorted lists

- Sorted on a value v_i that is **not** unique identifier.
- docID# identifies doc. uniquely

postings list "cat"	postings list "dog"
v1: docIDx	v1: docIDx
v2: docIDk	v3: docIDz
v4: docIDd	v4: docIDu
v4: docIDv	v4: docIDd
v4: docIDf	v4: docIDv
v5: docIDq	v4: docIDp
v6: docIDw	v7: docIDr

9

Keys for documents

For postings lists, entries are documents
What value is used to sort?

- Unique **document IDs**
 - can still be duplicate documents
 - consider for Web when consider crawling
- document **scoring function** that is **independent of query**
 - PageRank, HITS authority
 - sort on document IDs as secondary key
 - allows for **approximate "highest k" retrieval**
 - approx. k highest ranking doc.s for a query

10

Keys within document list

Processing within document posting

- Proximity of terms
 - merge lists of terms occurrences within same doc.
- Sort on term position

11

Computing document score

1. "On fly" - as find each satisfying document
 2. Separate phase after build list of satisfying documents
- For either, must sort doc.s by score

12

Web query processing: limiting size

- For Web-scale collections, may **not process complete posting list** for each term in query
 - at least not initially
- Need docs **sorted first on global (static) quantity**
 - why not by term frequency for doc?
- Only **take first k doc.s** on each term list
 - k depends on query - how?
 - k depends on how many want to be able to return
 - Google: 1000 max returns
 - Flaws w/ partial retrieval from each list?
- Other limits? **query size**
 - Google: 32 words max query size

13

Limiting size with term-based sorting

- Can sort doc.s on postings list by score of term
 - term frequency + ...
- Lose linear merge - salvage any?
- Tiered index:
 - tier 1: docs with highest term-based scores, sorted by ID or global quantity
 - tier 2: docs in next bracket of score quality, sorted
 - etc.
 - need to decide size or range of brackets
- If give up AND of query terms, can use idf too
 - only consider terms with high idf = rarer terms

14