

CS4051

Information Retrieval

Week 03

Muhammad Rafi

February 07, 2023

Dictionaries & Tolerant
Retrieval

Review Chapter No. 2

- We developed idea of inverted indexes for handling Boolean and proximity queries.
 - We discussed positional indexes for supporting general phrase queries.
 - We modify intersection of posting list to speedup in generating result-set.
 - What about the dictionary? Large Dictionary are still a challenge?
-

Chapter No. 3

- In this chapter we will develop techniques that are robust to typographical errors in the query, as well as alternative spellings.
 - We also develop data structures that help search for terms in the vocabulary in an inverted index.
 - We explore the idea of a wildcard query: a query such as `*a*e*i*o*u*`, which seeks documents containing any term that includes all the five vowels in sequence.
-

Data Structures for Dictionary

- There are two choices
 - Trees
 - Hashtable
 - IR systems can use either of the approach.
-

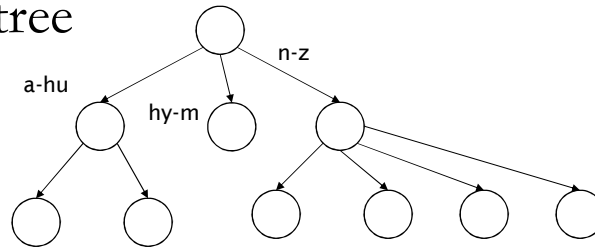
Sec. 3.1

Hashtables

- Each vocabulary term is hashed to an integer
 - (We assume you've seen hashtables before)
 - Pros:
 - Lookup is faster than for a tree: $O(1)$
 - Cons:
 - No easy way to find minor variants:
 - judgment/judgement
 - No prefix search [tolerant retrieval]
 - If vocabulary keeps growing, need to occasionally do the expensive operation of rehashing *everything*
-

Sec. 3.1

Tree: B-tree



- Definition: Every internal node has a number of children in the interval $[a,b]$ where a, b are appropriate natural numbers, e.g., $[2,4]$.
-

Sec. 3.1

Trees

- Simplest: binary tree
 - More usual: B-trees
 - Trees require a standard ordering of characters and hence strings ... but we typically have one
 - Pros:
 - Solves the prefix problem (terms starting with *hyp*)
 - Cons:
 - Slower: $O(\log M)$ [and this requires *balanced* tree]
 - Rebalancing binary trees is expensive
 - But B-trees mitigate the rebalancing problem
-

Wild Card Queries (*)

- Wildcard queries are used in any of the following situations:
 - the user is uncertain of the spelling of a query term (e.g., Sydney vs. Sidney, which leads to the wildcard query S*dney);
 - the user is aware of multiple variants of spelling a term and (consciously) seeks documents containing any of the variants (e.g., color vs. colour);
-

Wild Card Queries (*)

- Wildcard queries are used in any of the following situations:
 - the user seeks documents containing variants of a term that would be caught by stemming, but is unsure whether the search engine performs stemming (e.g., judicial vs. judiciary, leading to the wildcard query judicia*);
 - the user is uncertain of the correct rendition of a foreign word or phrase (e.g., the query Universit* Stuttgart).
-

Sec. 3.2

Wild-card queries: *

- ***mon****: find all docs containing any word beginning with “mon”.
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range: ***mon ≤ w < moo***
- ****mon***: find words ending in “mon”: harder
 - Maintain an additional B-tree for terms *backwards*.

Can retrieve all words in range: ***nom ≤ w < non***.

Exercise: from this, how can we enumerate all terms meeting the wild-card query ***pro*cent*** ?

Sec. 3.2

Query processing

- At this point, we have an enumeration of all terms in the dictionary that match the wild-card query.
- We still have to look up the postings for each enumerated term.
- E.g., consider the query:
se*ate AND fil*er

This may result in the execution of many Boolean ***AND*** queries.

Sec. 3.2

B-trees handle *'s at the end of a query term

- How can we handle *'s in the middle of query term?
 - *co*tion*
 - We could look up **co*** AND ***tion** in a B-tree and intersect the two term sets
 - Expensive
 - The solution: transform wild-card queries so that the *'s occur at the end
 - This gives rise to the **Permuterm** Index.
-

General Wild Card Query

- We now study two techniques for handling general wildcard queries.
 - Both techniques share a common strategy: express the given wildcard query q_w as a Boolean query Q on a specially constructed index, such that the answer to Q is a superset of the set of vocabulary terms matching q_w .
 - Then, we check each term in the answer to Q against q_w , discarding those vocabulary terms that do not match q_w . At this point we have the vocabulary terms matching q_w and can resort to the standard inverted index.
-

Sec. 3.2.1

Permuterm index

- For term **hello**, index under:
 - **hello\$, ello\$h, llo\$he, lo\$hel, o\$hell**
 where \$ is a special symbol.
- Queries:
 - **X** lookup on **X\$** **X*** lookup on **\$X***
 - ***X** lookup on **X\$*** ***X*** lookup on **X***
 - **X*Y** lookup on **Y\$X*** **X*Y*Z** ??? Exercise!

Query = **hel*o**
 X=**hel**, Y=**o**
 Lookup **o\$hel***

Sec. 3.2.1

Permuterm query processing

- Rotate query wild-card to the right
- Now use B-tree lookup as before.
- *Permuterm problem: \approx quadruples lexicon size*

Empirical observation for English.

Sec. 3.2.2

Bigram (k -gram) indexes

- Enumerate all k -grams (sequence of k chars) occurring in any term
- e.g., from text “***April is the cruellest month***” we get the 2-grams (*bigrams*)

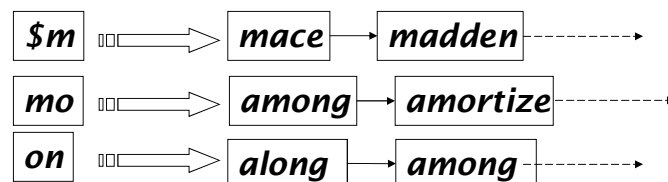
\$a,ap,pr,ri,il,l\$, \$i,is,s\$, \$t,th,he,e\$, \$c,cr,ru,ue,el,le,es,st,t\$, \$m,mo,on,nt,h\$

- \$ is a special word boundary symbol
 - Maintain a second inverted index from bigrams to dictionary terms that match each bigram.
-

Sec. 3.2.2

Bigram index example

- The k -gram index finds *terms* based on a query consisting of k -grams (here $k=2$).



Sec. 3.2.2

Processing wild-cards

- Query ***mon**** can now be run as
 - ***\$m AND mo AND on***
 - Gets terms that match AND version of our wildcard query.
 - But we'd enumerate ***moon***.
 - Must post-filter these terms against query.
 - Surviving enumerated terms are then looked up in the term-document inverted index.
 - Fast, space efficient (compared to permuterm).
-

Sec. 3.2.2

Processing wild-card queries

- As before, we must execute a Boolean query for each enumerated, filtered term.
- Wild-cards can result in expensive query execution (very large disjunctions...)
 - ***pyth* AND prog****
- If you encourage “laziness” people will respond!

Search

Type your search terms, use '*' if you need to.
E.g., Alex* will match Alexander.
