QUERY PROCESSING

Phrase queries

A phrase query is a query where multiple words are considered as an atomic unit, therefore we can't solve the query with the posting lists as they are now

Example

The query "standford university" should not match with "I went at standford my university".

With the postings we have, this doc would match

- Solution 1: 2-word (Biword) indexes

Consider every pair of consecutive terms in a document as a phrase and add them to the dictionary of the inverted list. The query processing is immediate, and it is basically the same process as before (we just have a longer dictionary).

$$d = \text{``} \ldots t_1 \, t_2 \ldots \text{''}$$

$$t_1 \longrightarrow \ldots\ldots d \ldots\ldots$$

$$t_2 \longrightarrow \ldots\ldots d \ldots\ldots$$

$$t_1 t_2 \longrightarrow \ldots\ldots d \ldots\ldots$$

This solution also works with longer query phrases: just divide them in overlapping pairs and process the concatenation (AND operator of all pairs)

EXAMPLE

Q = "STANDFORD UNIVERSITY PALO ALTO"

$\Big\downarrow$

Q = "STANFORD UNIVERSITY" AND "UNIVERSITY PALO" AND "PALO ALTO"

However the result of this boolean query may returns false positives, therefore it should be verified by checking the documents and using other solutions, such as PoS Tagging (it constructs an extended biwords index)

• **SOLUTION 2: positional indexes**

For every document, store in the posting lists the positions where the term occurs

EXAMPLE

DICTIONARY POSTING LIST

< TERM, #docIDs > ⟶ < doc 1: pos 1, pos 2, ...
doc 2: ... -
, - - - - - >

NOTE: The same method can be used for proximity searches, used to solve free text queries (just a set of terms typed in the query box), where the users prefer retrieving documents in which the query terms are in close proximity of each other.

NOTE: If you are worried about size, you can compress the positions just like the docIDs (see posting list compression). Nevertheless, they are quite useful thanks of the power of phrase and proximity queries in modern information retrieval (web searches)

Solution 1 + Solution 2 is a profitable combination scheme

Another useful process, applied by search engines on phrase query, is the Soft-AND: an algorithms that tries to maximize the number of results:

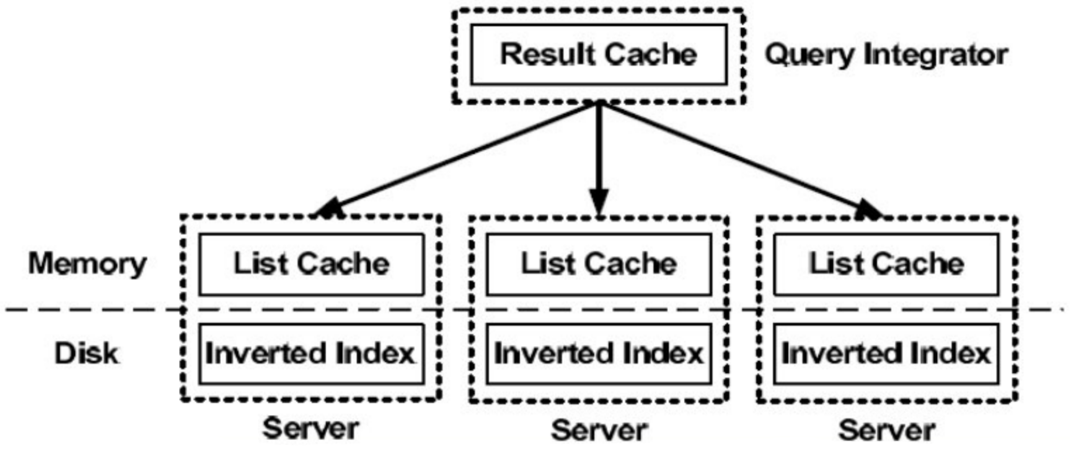
Soft-AND (q, K)

- Run the phrase query as it is
- If the results contains too few documents (< K), divide the query in overlapping biwords and run the ANDed query.
- if we still have less than K docs, run the vector space query and rank the results (see related slide)

Run queries faster

Two (actually three) approaches

- Caching
    - Chache the query results to exploit query locality
    - Chache the pages of posting lists to exploit term locality

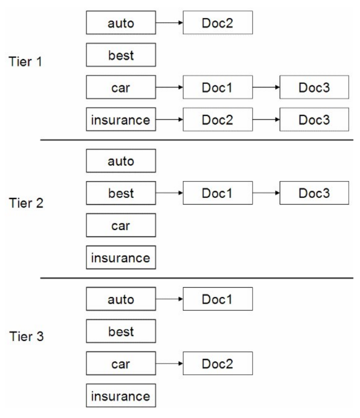


- Tiered indexes

Break the posting lists into a hierarchy of list sorted by importance: this way the inverted index is divided in tiers of decreasing importance.

At query time, use only the top tier (unless you fail to retrieve at least K documents)

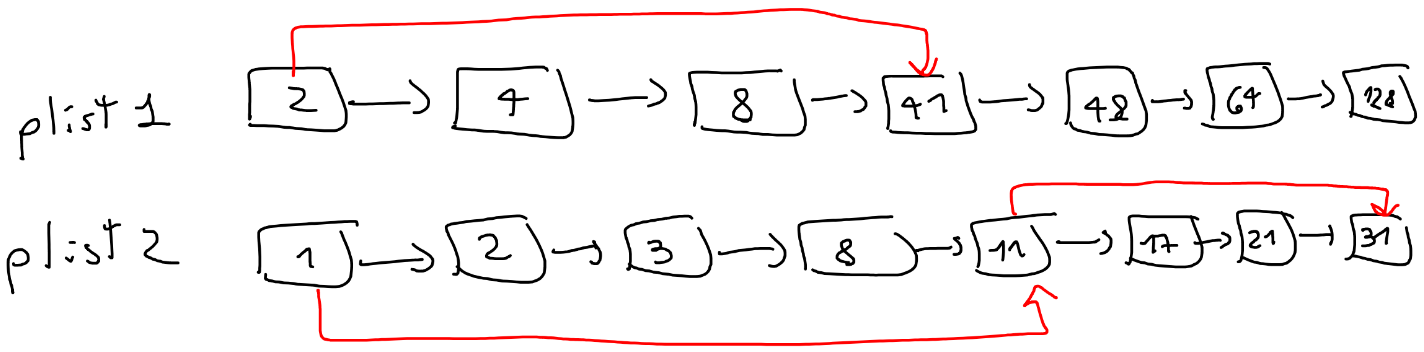
EXAMPLE



SKIP POINTERS

Query processing optimization techique. Add to the posting lists some skip pointers at indexing time: this way you can make the list intersection process faster.

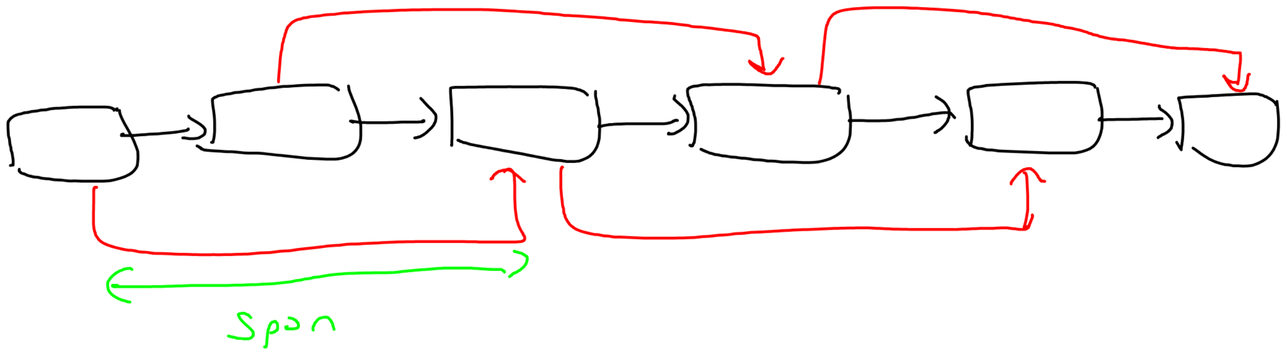
EXAMPLE



Let's imagine we are at docId = 8 in both lists. After adding it to the final results, we advance plist 1 to 41 and plist 2 to 11: since 11 is smaller than 41, we can safely use the skip pointer and go to 31.

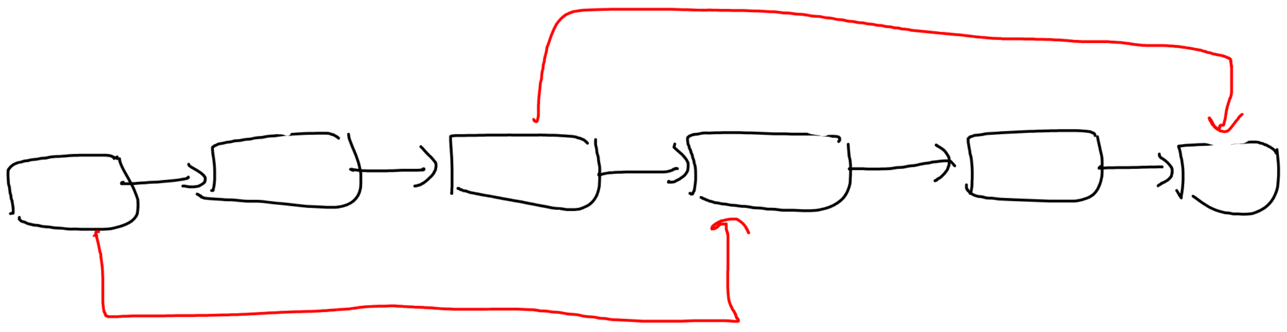
Obviously, the number of skip pointers is the key parameter here

- More skip pointers => shorter spans => more skips possible



But it requires a lot of comparison at each step (which is better: skip pointer or normal pointer?)

- Few skip pointers => longer spans => less skips



Less Comparison

OSS Having bigger posting lists may be inefficient for in-memory indexes, due to the cost in I/O of the loading (which could outweight the gains)

So how many skip pointers should I use?

Simple heuristic : For a skip-list of length L, use $\sqrt{L}$ evenly-spaced skip pointers. However, the distribution of query terms is ignored