IN3120/4120 - Search technology

Group 2, Second session
TA: Markus Sverdvik Heiervang

Today's plan

- In depth coverage of inverted indices
- Inverted index walkthrough in jupyter notebook
- Merging of postings
- Index compression

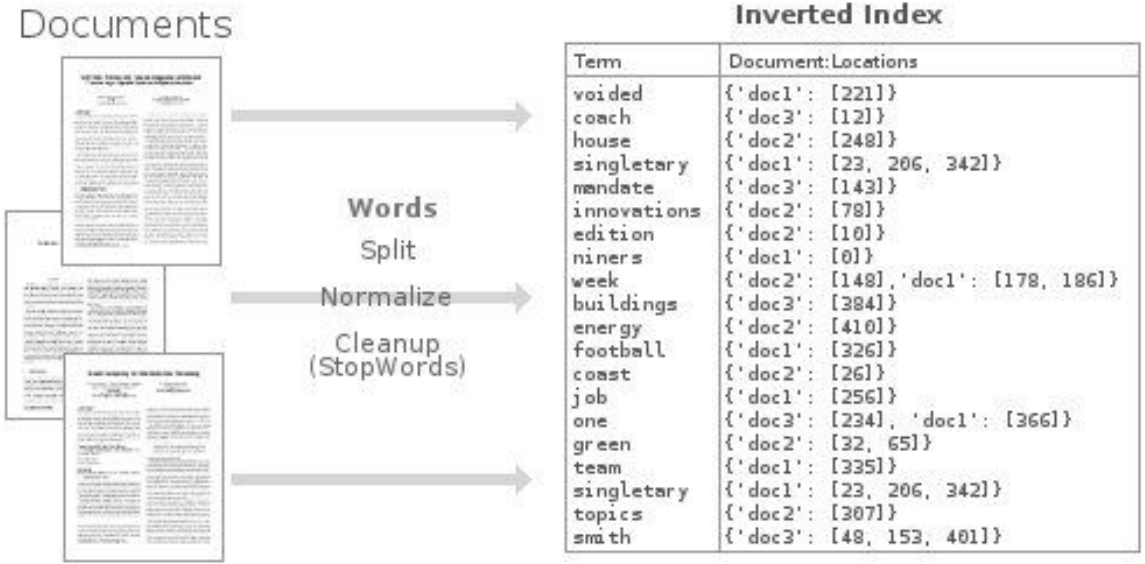
- Q&A, eventually topics you want me to cover

Inverted index: what is it?

- A data structure which lets you effectively find every document of a corpus that contains a given term

- Posting: typically a collection containing document id, and optionally additional information such as term frequency, term position in the document, etc.

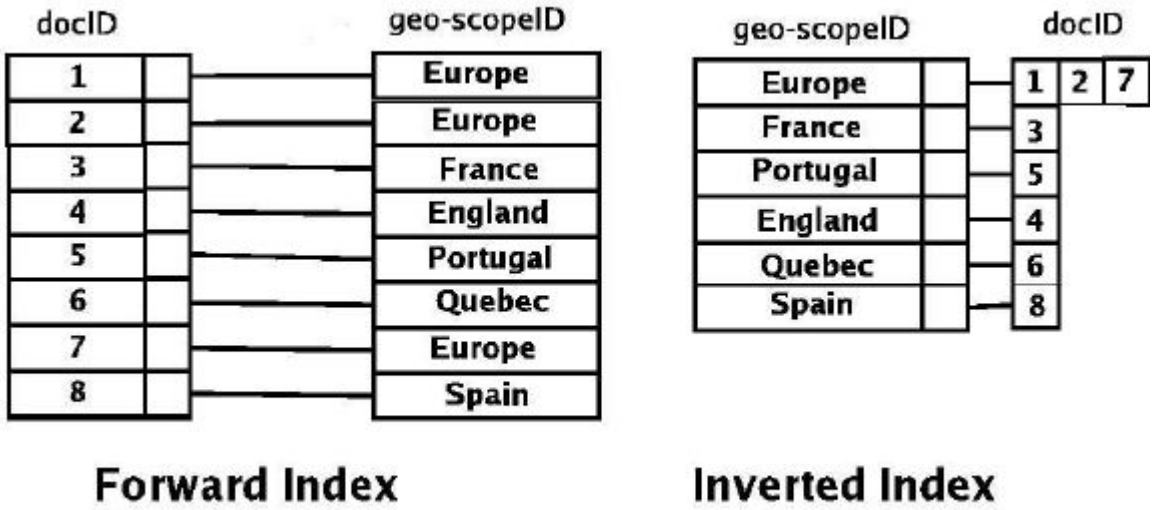
Inverted index: what is it?



Inverted Index

Term	Document:Locations
voided	{'doc1': [221]}
coach	{'doc3': [12]}
house	{'doc2': [248]}
singletary	{'doc1': [23, 206, 342]}
mandate	{'doc3': [143]}
innovations	{'doc2': [78]}
edition	{'doc2': [10]}
niners	{'doc1': [0]}
week	{'doc2': [148],'doc1': [178, 186]}
buildings	{'doc3': [384]}
energy	{'doc2': [410]}
football	{'doc1': [326]}
coast	{'doc2': [26]}
job	{'doc1': [256]}
one	{'doc3': [234], 'doc1': [366]}
green	{'doc2': [32, 65]}
team	{'doc1': [335]}
singletary	{'doc1': [23, 206, 342]}
topics	{'doc2': [307]}
smith	{'doc3': [48, 153, 401]}

Inverted index as opposed to forward index



Examples and how to build

Postings merging

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

Figure 2: pseudocode from the book

Necessary when searching for documents containing both term a and b, using the query:
a AND b

Example:
You've heard rumors that your favorite rapper Tupac is dead. To find out if it is true or not, you search using an intersectional query:
Tupac AND Dead

Postings merging

Problem: what if you want an intersection of n > 2 terms?
Solution:

```
INTERSECT(⟨t_1, ..., t_n⟩)
1  terms ← SORTBYINCREASINGFREQUENCY(⟨t_1, ..., t_n⟩)
2  result ← postings(first(terms))
3  terms ← rest(terms)
4  while terms ≠ NIL and result ≠ NIL
5  do result ← INTERSECT(result, postings(first(terms)))
6     terms ← rest(terms)
7  return result
```

Figure 3: pseudocode from the book

the query:
a and b and c and d
will be evaluated as such:
(((a and b) and c) and d)

Index compression

- When working with big data and limited memory, compression can be very useful
- Typical when serializing an inverted index (i.e. storing the inverted index on a disk)

Say we have a posting list for a given term t, containing only document ids, in an index of a relatively large corpus. There are many words that occur rarely, such as t

Postings(t) = [7, 40, 7000, 43021, 140236, 2773002]

Large numbers take up much space: 7 == 111, 2773002 == 10101001010000000001010.

The idea is to keep the first id, and then only keep the differences compressed

Differences(Postings(t)) = [7-0, 40-7, 7000-40, 43021-7000, …]

Then we apply a number-compression algorithm on these differences. You will learn more in the lecture about compression

What more do we need for a search engine?

- A ranker
- A filter for cheats
- A query processor
  - e.g. if the query is “Tupac dead”
  - search (Tupac AND dead), then (Tupac OR dead)
  - NLP/NLU
- etc ...

Tomorrows lecture

In tomorrows lecture, Aleksander will cover most of the string algorithms in this course

important algorithms for assignment B:

- Suffix arrays
- Tries and aho-corasick algorithm

I will go further in depth on some of these topics on the next group session