TÉCNICO LISBOA

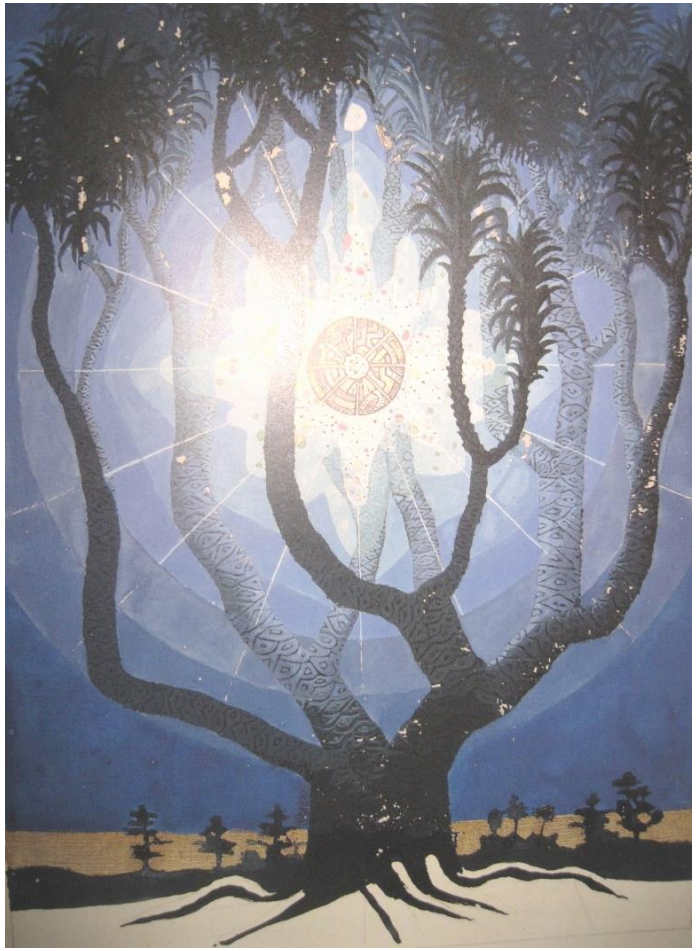
Basics of Indexing and Boolean Querying

Acknowledgments: Standford NLP group

PRI 2023/24

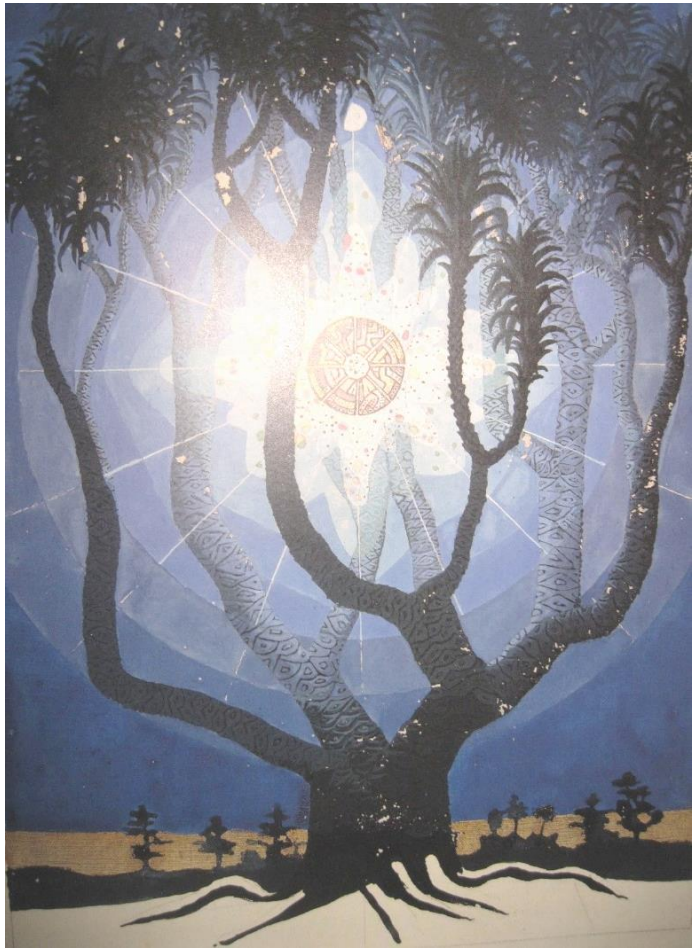
Rui Henriques
rmch@tecnico.ulisboa.pt

Outline



- **Introduction**
- **Inverted indexing**
  - essentials
  - construction
  - dictionary structures
- **Text processing**
  - tokenization
  - normalization
- **Boolean querying**
  - AND queries
  - large queries
  - flexible queries
- **Indexing dynamic collections**
- **Final remarks**

Outline



- **Introduction**
- Inverted indexing
  - essentials
  - construction
  - dictionary structures
- Text processing
  - tokenization
  - normalization
- Boolean querying
  - AND queries
  - large queries
  - flexible queries
- Indexing dynamic collections
- Final remarks

Boolean retrieval

Which plays of *Shakespeare* contain the words Brutus AND Caesar but NOT Calpurnia ?

- one could grep all of plays for Brutus and Caesar, then strip out documents containing Calpurnia
- why is that not the answer?
  - **slow** (for large corpora)
  - best plays? (ranked retrieval)
  - Brutus nearby Caesar occurrences

Antony and Cleopatra, Act III, Scene ii

Agrippa [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,
When Antony found Julius ***Caesar*** dead,
He cried almost to roaring; and he wept
When at Philippi he found ***Brutus*** slain.

Hamlet, Act III, Scene ii

Lord Polonius: I did enact Julius ***Caesar*** I was killed i' the
Capitol; ***Brutus*** killed me.

Term-document incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth …
ANTHONY	1	1	0	0	0	1
BRUTUS	1	1	0	1	0	0
CAESAR	1	1	0	1	1	1
CALPURNIA	0	1	0	0	0	0
CLEOPATRA	1	0	0	0	0	0
MERCY	1	0	1	1	1	1
WORSER	1	0	1	1	1	0
. . .						

1 if play contains word, **0** otherwise

Brutus AND Caesar but NOT Calpurnia

Incidence vectors

- so we have a 0/1 vector for each term
- to answer the query Brutus AND Caesar but NOT Calpurnia:
  1. take the vectors for Brutus, Caesar and NOT Calpurnia
  2. complement the vector of CALPURNIA
  3. do a (bitwise) and on the three vectors

110100 AND **110111** AND **101111** = **100100**

	Anthony & Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth …
ANTHONY	1	1	0	0	0	1
BRUTUS	**1**	**1**	**0**	**1**	**0**	**0**
CAESAR	**1**	**1**	**0**	**1**	**1**	**1**
CALPURNIA	**0**	**1**	**0**	**0**	**0**	**0**
CLEOPATRA	1	0	0	0	0	0
MERCY	1	0	1	1	1	1
WORSER	1	0	1	1	1	0

Boolean retrieval

- Boolean model is arguably the simplest model for IR
  - queries are Boolean expressions
  - search engine returns all documents that satisfy the Boolean expression
  - does Google use the Boolean model? Yes

- **still** our incidence matrices…
  - consider $N = 10^6$ documents, each with about 1000 tokens ⇒ total of 10^9 tokens
  - on average 6 bytes per token ⇒ size of document collection is about $6 \cdot 10^9 = 6$ GB
  - assume there are $M = 500{,}000$ distinct terms in the collection (notice the term-token distinction)

⇒ $M = 500{,}000 \times 10^6$ = half a trillion 0s and 1s

cannot build the incidence matrix!!

Sparse incidence matrices

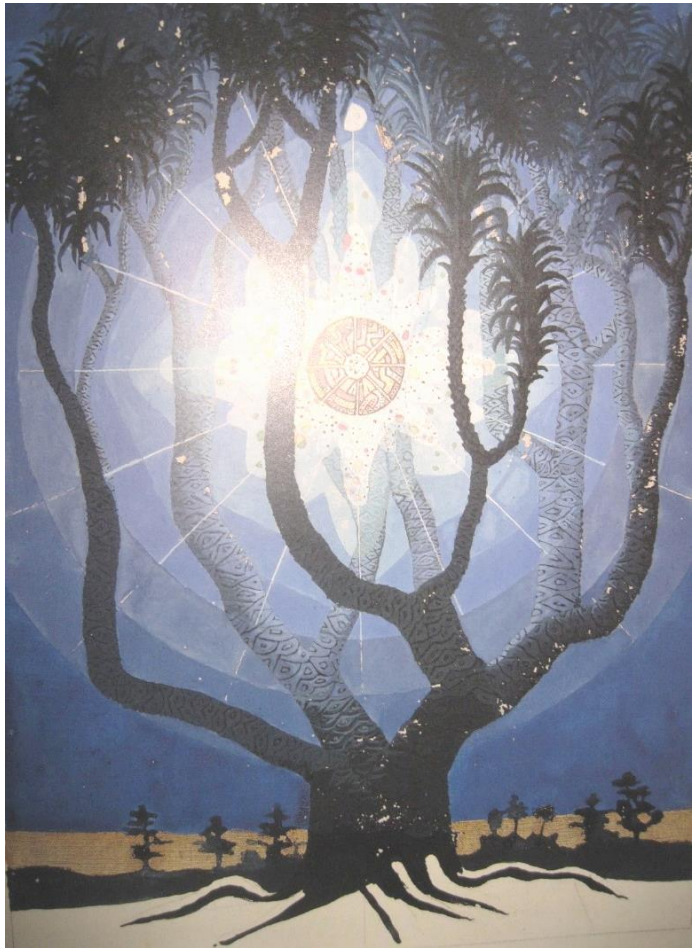
Yet, our incidence matrix has no more than one billion 1's! *Why*?

- matrix is extremely ***sparse***
- what's a better representation?
  - only record the 1 positions!

How?

- ***simple index***
  - index terms and link for each document a list of term IDs
    - how can we query Brutus AND Caesar but NOT Calpurnia
    - other problems?
- ***inverted index***
  - index documents and link for each term a list of document IDs

Outline



- Introduction
- **Inverted indexing**
  - **essentials**
  - **construction**
  - **dictionary structures**
- Text processing
  - tokenization
  - normalization
- Boolean querying
  - AND queries
  - large queries
  - flexible queries
- Indexing dynamic collections
- Final remarks

Inverted index

- For each term *t*, list of all documents that contain *t*

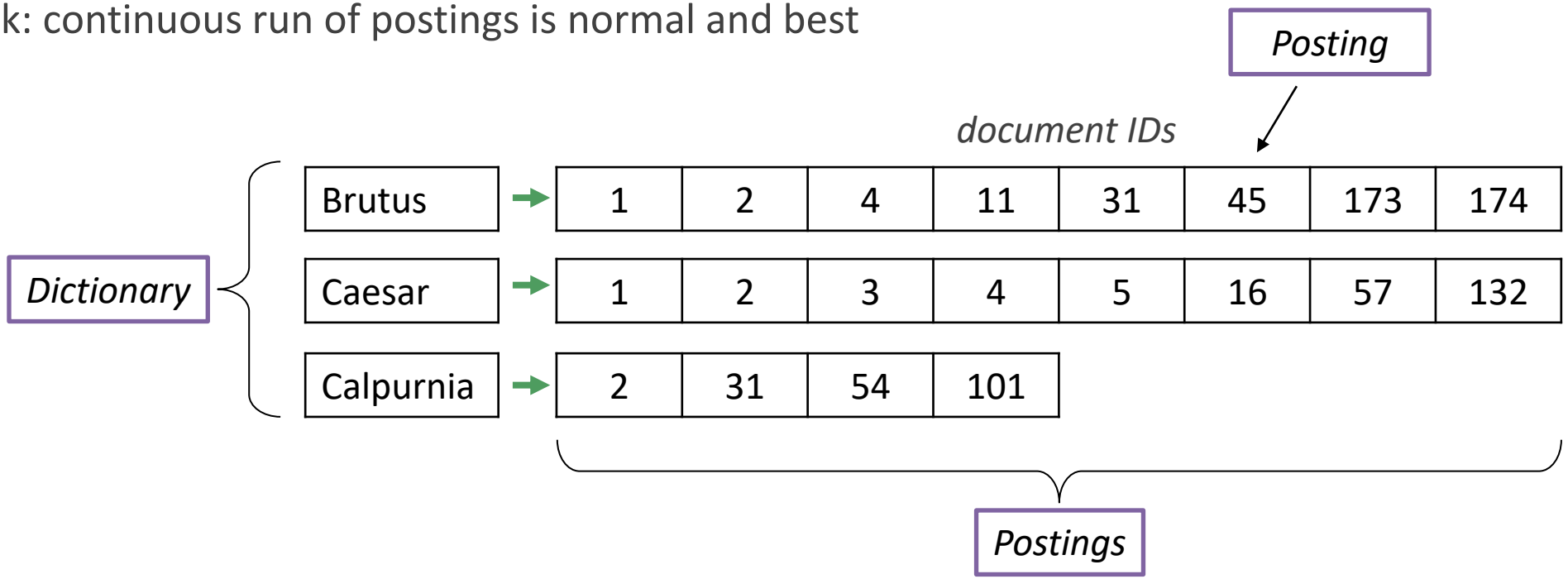
document IDs

Brutus	→	1	2	4	11	31	45	173	174
Caesar	→	1	2	3	4	5	16	57	132
Calpurnia	→	2	31	54	101				

- Can we use fixed-size arrays for this?
  - what happens in construction time if a new document has Caesar term?
- Time complexity?

Inverted index

- Solution: variable-size **postings lists**
  - memory: ***linked lists*** or ***variable length arrays***
    - tradeoffs: size? Insertion time?
  - disk: continuous run of postings is normal and best



sorted by *docID* (more later on why)

Inverted index construction

Documents to be indexed

Friends, Romans, countrymen.

Tokenizer

Token stream

Friends | Romans | Countrymen

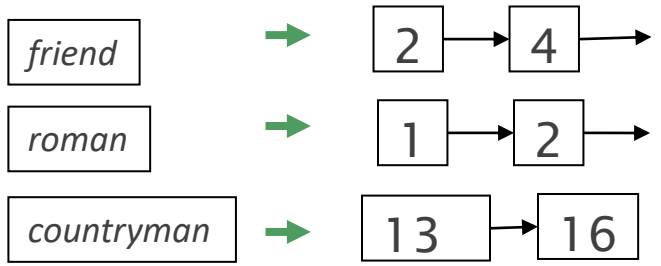
Linguistic modules

Modified tokens

friend | roman | countryman

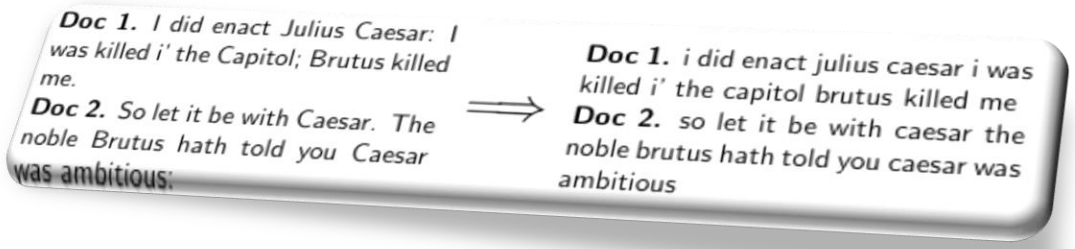
Indexer

Inverted index



Initial stages of text processing

- **parsing**
- **tokenization**: cut character sequence into word tokens
  - ***John's***, ***state-of-the-art***?
  - punctuation? capitalization?



- **normalization**: reduce text and query terms to single form
  - match ***U.S.A.*** and ***USA***
- **stemming**: we may wish different forms of a root to match
  - ***authorize***, ***authorization***
- **stop words**: omit very common words (or not?)
  - ***the, a, to, of***

Initial stages of text processing

Yet to **parse** a document…

- What format? Pdf, word, excel, html, xml?
- What language?
- Character set? CP1252, UTF-8?
- What is the document unit for indexing? File? Email?

Challenges:

- sometimes a document or its components contain multiple formats
  - email with 5 attachments (ppt or latex in HTML)
  - webpage with embed files and media
- a single index can contain terms of several languages
  - French email with Spanish pdf attachment

Answering these questions

- classification problem (e.g. language detector) *versus* simple heuristics

Indexer steps: tokenize and sort

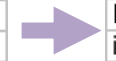
doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2



Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

1. **traverse** documents to build sequence of (token, doc ID) pairs
2. **sort** by terms and then docID
   - computational complexity?

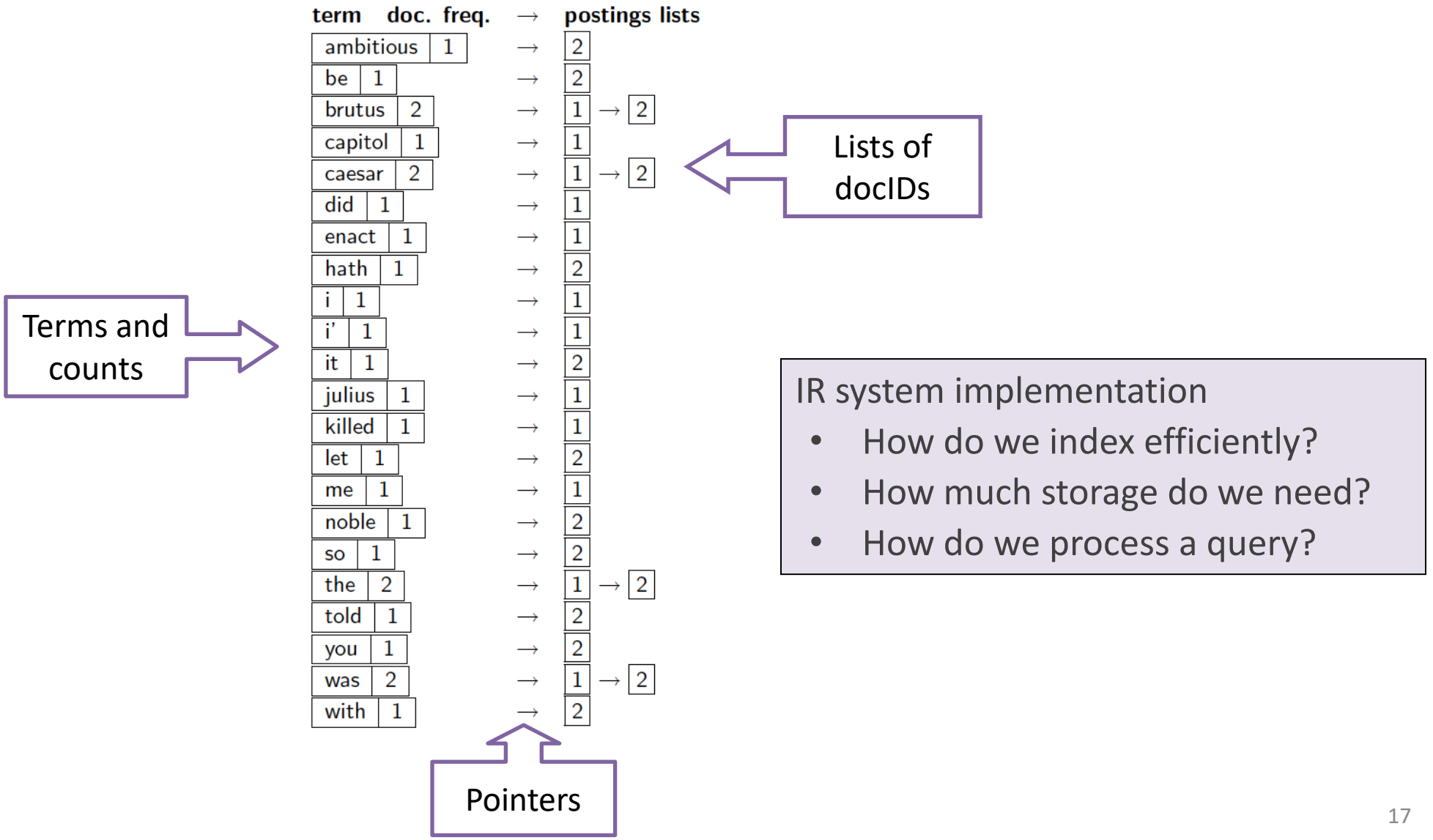
Indexer steps: dictionary and postings

1. multiple term entries in a single document are merged
2. structure is built: dictionary and postings
   - computational complexity?
3. document frequency is added
   - why?

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Space and time complexity



IR system implementation

- How do we index efficiently?
- How much storage do we need?
- How do we process a query?

Naïve dictionary

Naïve: fixed-size array structure

term	document frequency	pointer to postings list
a	656,265	⟶
aachen	65	⟶
. . .	. . .	. . .
zulu	221	⟶
char[20]	int	postings *
20 bytes	4/8 bytes	4/8 bytes

- How do we store a dictionary in memory efficiently?
- How do we quickly look up elements at query time?

- Two main classes of data structures: **hashes** and **trees**
  - *some IR systems use hashes, others use trees*

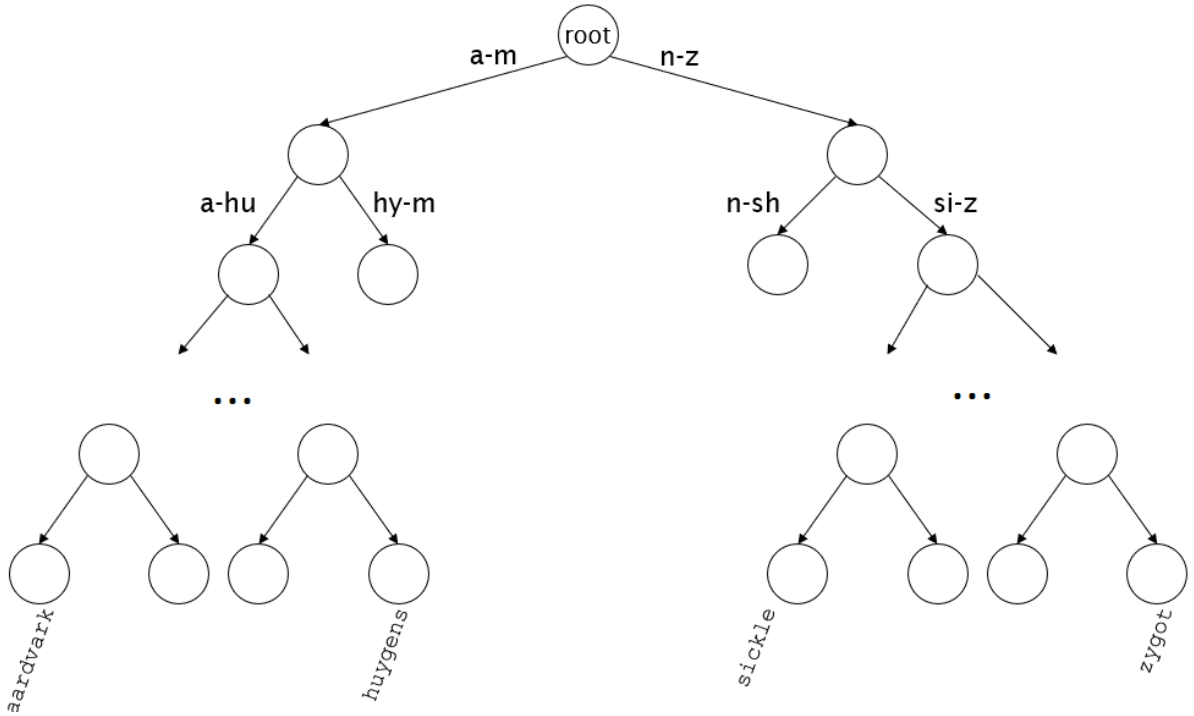
Dictionaries as trees

- Criteria for when to use ***hashes versus trees***:
  - Is there a fixed number of terms or will it keep growing?
  - What are the relative frequencies with which various keys will be accessed?
  - How many terms are we likely to have?

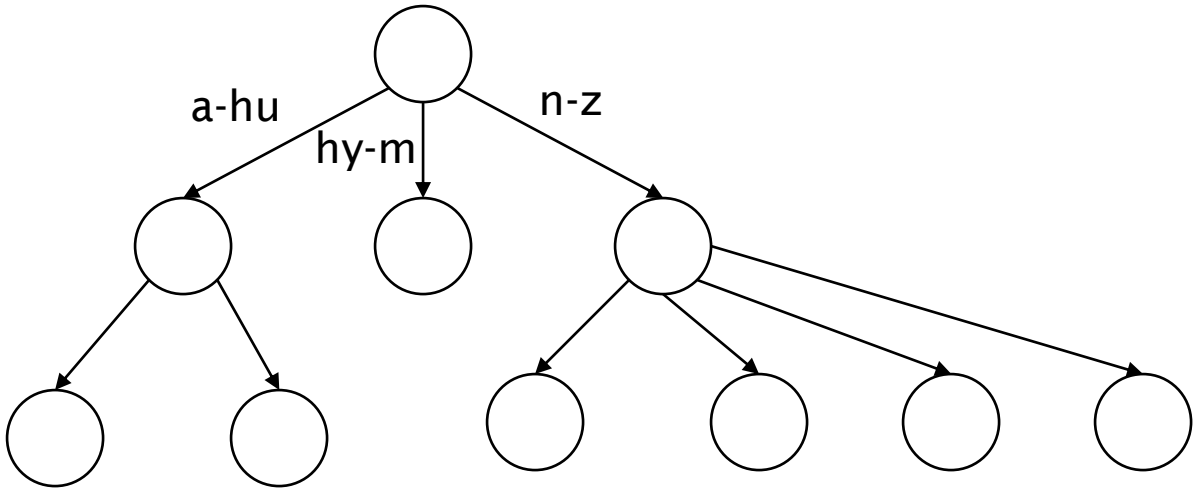
- **Trees**
  - require a standard ordering of characters… and we generally have one
  - simplest: binary tree
  - more usual: B-trees
  - **Pros**: solves the prefix problem when querying (terms starting with hyp)
  - **Cons**
    - slower: **O(log M)** lookup [and this requires a balanced tree]
    - Rebalancing binary trees is expensive
      - B-trees mitigate the rebalancing problem

Dictionaries as trees

Binary tree



B-Tree



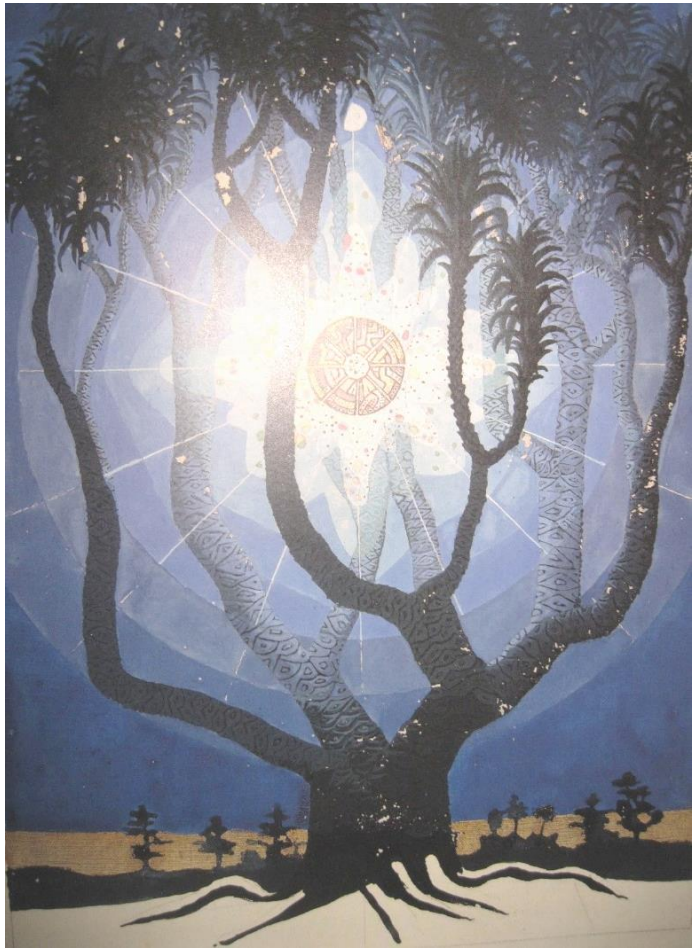
Definition: every internal node has a number of children in the interval [*a*,*b*] (e.g. [2,4])

Dictionaries as hashtables

- Each vocabulary term is hashed into an integer
  - integer indexes position in array
  - collisions to be resolved (same integer for two terms) are rare

- **Pros**: lookup is faster than for a tree: **O(1)**
- **Cons**:
  - no easy way to find minor variants (e.g. *judgment* and *judgement*)
  - no prefix search [*tolerant retrieval*]
  - if vocabulary keeps growing: occasionally do the expensive operation of rehashing *everything*

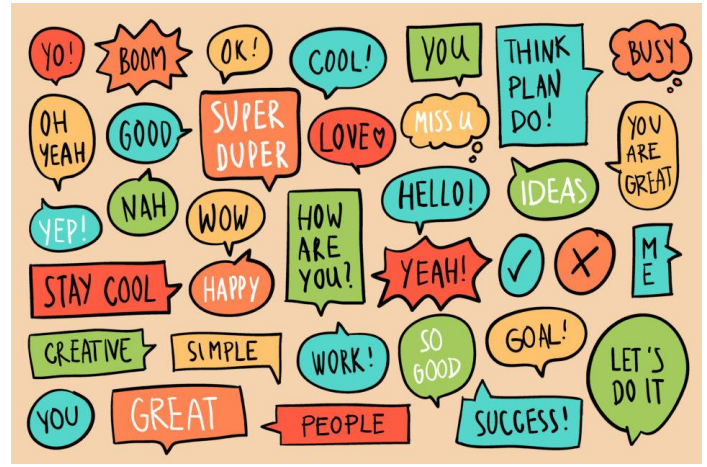
Outline



- Introduction
- Inverted indexing
  - essentials
  - construction
  - dictionary structures
- **Text processing**
  - **tokenization**
  - **normalization**
- Boolean querying
  - AND queries
  - large queries
  - flexible queries
- Indexing dynamic collections
- Final remarks

Text processing: upfront definitions

- As we saw, text processing is an essential part of the indexing and querying process
  — sound identification of tokens and phrases, handling case and accents, morphological and spelling variations, amongst many other ends
- At this stage, a precise universe discourse is essential:
  - **Word:** a delimited string of characters as it appears in the text
  - **Term:** an equivalence class of words
    (i.e. a *normalized* word)
  - **Token:** an instance of a word or term occurring in a document
  - **Type:** an equivalence class of tokens
    (the same as a *term* for our course)
  - **Topic:** an abstraction for a set of terms



Tokenization challenges

- Valid tokens? Issues in tokenization:
  - *Finland's capital* → *Finland* AND *s*? *Finlands*? *Finland's*?
  - *San Francisco*
  - *Los Angeles-based company*
  - *data base*
  - *Hewlett-Packard*
  - *state-of-the-art*
  - *co-education*
  - *lowercase, lower-case, lower case*?
  - *co-education*
  - *the hold-him-back-and-drag-him-away maneuver*
  - *cheap San Francisco-Los Angeles fares*
  - *York University vs. New York University*

Tokenization: language issues

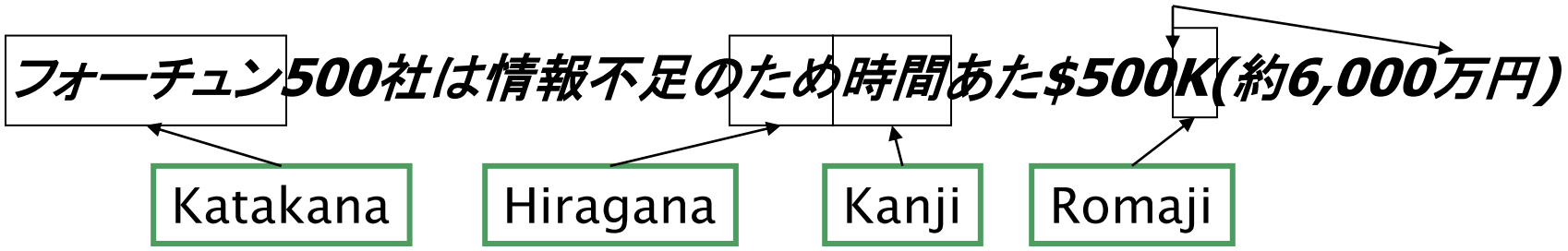
- **French**
  - ***L'ensemble*** → one token or two?
    - ***L*** ? ***L'*** ? ***Le*** ?
    - ***l'ensemble*** matches ***un ensemble***?
      - until at least 2003, it didn't on Google

- **German** noun compounds are not segmented
  - ***Lebensversicherungsgesellschaftsangestellter***
    = '*life insurance company employee*'
  - German retrieval systems benefit greatly from a **compound splitter** module
    - 15% performance boost for German

- Dutch and Swedish languages have similar problems
- Many other languages with segmentation difficulties: Finnish, Urdu…

Tokenization: language issues

- **Chinese** and **Japanese** have no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - not always guaranteed a unique tokenization
- further complicated in Japanese, with multiple alphabets intermingled
  - dates/amounts in multiple formats



End-user can express query entirely in hiragana!

Tokenization: language issues

- **Chinese** is particularly prone to ambiguous segmentation
  - beyond the absence of white spaces…

莎拉波娃现在居住在美国东南部的佛罗里达。今年４月９日，莎拉波娃在美国第一大城市纽约度过了１８岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

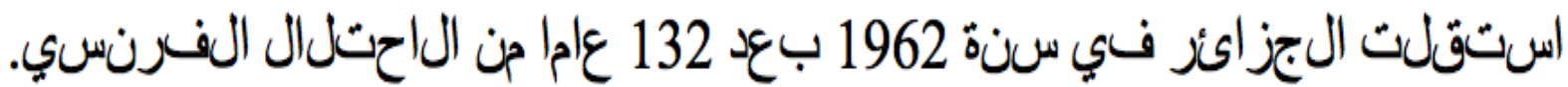
  - characters have multiple meanings
    - the following two characters can be treated as one word meaning '***monk***' or as a sequence of two words meaning '***and***' and '***still***'

和尚

Tokenization: language issues

- **Arabic** (or **Hebrew**)
  - generally written right to left, with certain items like numbers from left to right
  - words are separated, but letter forms within a word form complex ligatures

← → ← → ← start



"Algeria achieved its independence in 1962 after 132 years of French occupation"

(with unicode, the surface presentation is complex, but the stored form is straightforward)

Numbers

- Challenges
  - *3/20/91* and *20/3/91*
  - *Mar. 12, 1991*
  - *55 B.C.*
  - *B-52*
  - *my PGP key is 324a3df234cb23e*
  - *(800) 234-2333*

- Older IR systems do not index numbers
  - yet often very useful: e.g. error codes or stack-traces on the web
- Numbers... often have embedded spaces
  - **solution**: hold metadata structures for specific number formats (e.g. dates)

Stop words

- **stop words** = extremely common words which would appear to be of little value to IR
  - a, an, and, are, as, at, be, by, for, from, has, in, is, it, its, of, on, that, the, to, was, were, will, with
  - they have little semantic content
  - there are a lot of them: ~30% of postings for top 30 words

- Stop word elimination used to be standard in older IR systems
- Yet we need stop words
  - **phrase queries**: “King of Denmark”
  - various song **titles**, etc.: “Let it be”, “To be or not to be”
  - **relational queries**: “flights to London”

- In fact, most web search engines index stop words!
  - good compression techniques (IIR chapter 5): the space of stop words in an index is very small
  - good query optimization techniques (IIR chapter 7): little expense at query time for including stop words

Normalization to terms

- We may need to *normalize* words in document and query text
  - *U.S.A.* and *USA*
  - *anti-discriminatory* and *antidiscriminatory*
- Result: **terms**
  - a term is a (normalized) word type, which is an entry in our IR system dictionary
- How? define equivalence classes
  - collapse
    - e.g. deleting periods and hyphens to form a term
  - asymmetric expansion (more powerful, but less efficient)
    - Enter: *window* Search: *window, windows*
    - Enter: *windows* Search: *Windows, windows, window*
    - Enter: *Windows* Search: *Windows*
  - why don't you want to put *window, Window, windows,* and *Windows* in the same equivalence class?

Accents and diacritics

- Accents: résumé vs. resume (simple omission of accent)
- Umlauts: Universität vs. Universitaet (substitution with special letter sequence “ae”)

- **Important criterion**: How are users likely to write their queries for these words?
  - even in languages that standardly have accents, users often do not type them (Polish?)
  - often best to normalize to a de-accented term: *Tuebingen, Tübingen, Tubingen*

- **Exercise**
  - “*In June, the dog likes to chase the cat in the barn.*”
    - How many word tokens? How many word types?
  - Tokenize: “*Mr. O’Neill thinks that the boys’ stories about Chile’s capital aren’t amusing.*”

Case folding

"*PETER WILL NICHT MIT*" ⇒ MIT = mit
"*He got his PhD from MIT*" ⇒ MIT ≠ mit

- Reduce all letters to lower case
  - exception: upper case in mid-sentence?
    - General Motors
    - Fed vs. fed
    - SAIL vs. sail
  - often best to lower case everything, since users will use lowercase regardless of 'correct' capitalization…

- Longstanding Google example [fixed in 2011]:
  - query C.A.T.
  - #1 result is for "cats" (well, Lolcats) not Caterpillar Inc.

Thesauri and soundex

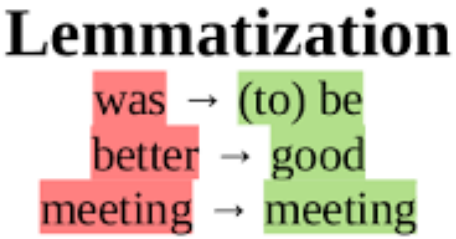
- More equivalence classing
  - Soundex: IIR chapter 3 (phonetic equivalence, Muller = Mueller)
  - Thesauri: IIR chapter 9 (semantic equivalence, car = automobile)
- Do we handle synonyms and homonyms?
  - E.g. by hand-constructed equivalence classes
    - *car* = *automobile*
    - *color* = *colour*
  - we can rewrite to form equivalence-class terms
    - When the document contains ***automobile***, index it under ***car-automobile*** (and vice-versa)
  - or we can expand a query
    - When the query contains ***automobile***, look under ***car*** as well
- What about spelling mistakes?
  - one approach is Soundex, which forms equivalence classes of words based on phonetic heuristics

Lemmatization

- Reduce inflectional and variant forms to base form
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*

- Implies doing "*proper*" reduction to dictionary headword form
  - **inflectional morphology** (*cutting* → *cut)*
  - **derivational morphology** (*destruction* → *destroy)*

- Exercise
  - *"the boy's cars are different colors"*
  - solution: "*the boy car be different color*"



Stemming

- Stemming suggests crude affix chopping
  - e.g. *automate(s), automatic, automation* all reduced to *automat*
- *Definition*: heuristic process that chops off the ends of words in the hope of achieving what "principled" lemmatization attempts to do with a lot of linguistic knowledge
- *Characteristics*:
  - language dependent
  - often inflectional and derivational
  - reduce terms to their "roots" before indexing
  - language dependent

for example compressed and compression are both accepted as equivalent to compress.

for exampl compress and compress ar both accept as equival to compress

Porter's algorithm

- Most common algorithm for stemming English
  - results suggest it is at least as good as other stemming options
- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands
  - illustrative convention
    - *Of the rules in a compound command, select the one that applies to the longest suffix*
  - Illustrative commands
    - sses → ss
    - ies → I
    - ss → ss
    - s →
    - ational → ate
    - tional → tion

caresses → caress

ponies → poni

caress → caress

cats → cat

Three stemmers: a comparison

- Sample text

  Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

- **Porter** stemmer

  such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to pictur of express that is more biolog transpar and access to interpret

- **Lovins** stemmer

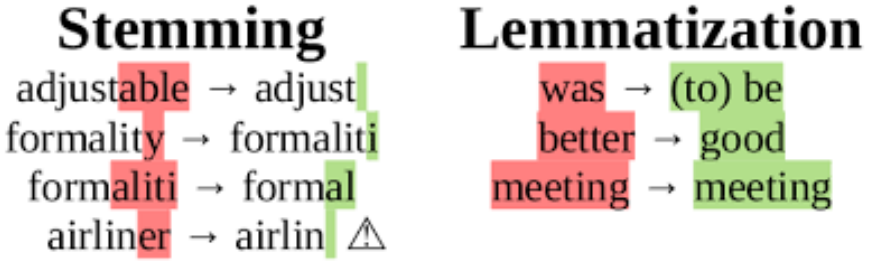
  such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

- **Paice** stemmer

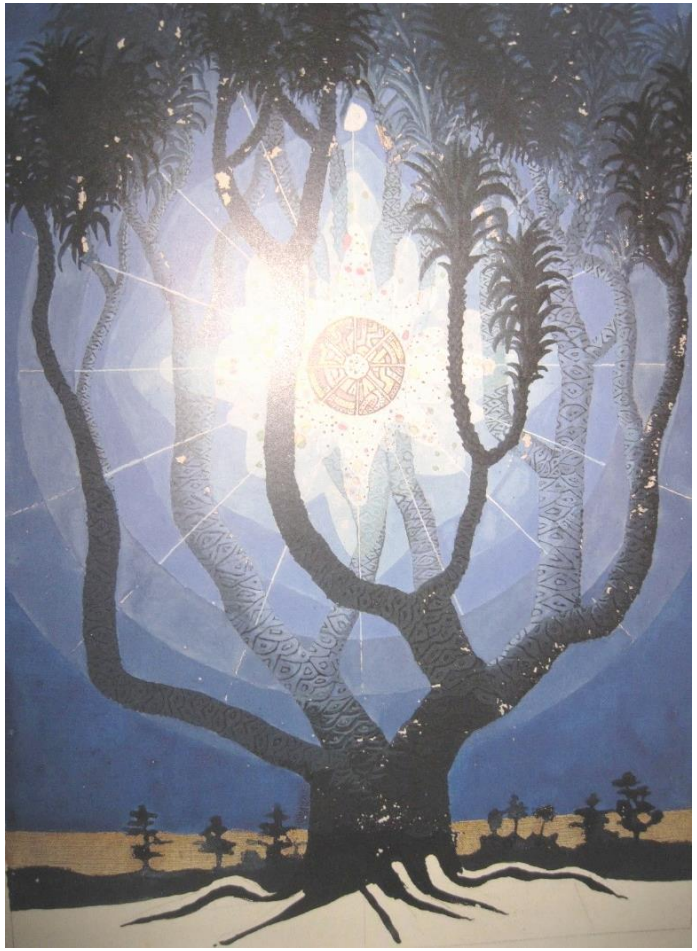
  such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

Does stemming help?

- English: very mixed results
  - helps *recall* for some queries
    but harms *precision* on others
- Definitely useful for Spanish, German, Finnish, …
  - >30% performance gains for Finnish!
- queries where stemming is likely to **help**
  - [tartan sweaters]
  - [sightseeing tour san francisco]
  - equivalence classes: {sweater,sweaters}, {tour,tours}
- queries where stemming **hurts**
  - [operational AND research]
  - [operating AND system]
  - [operative AND dentistry]



Outline

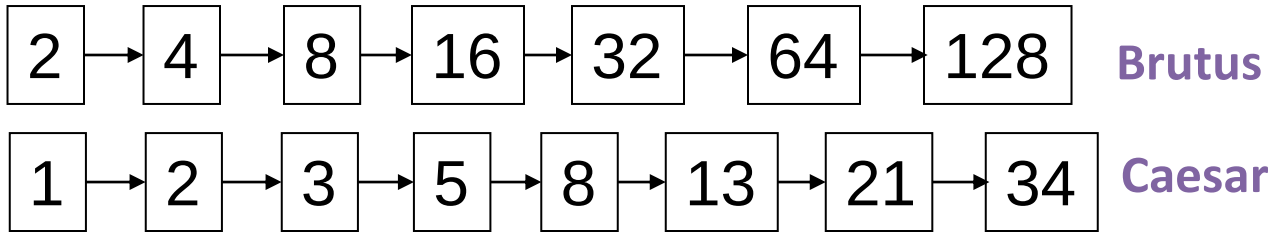


- Introduction
- Inverted indexing
  - essentials
  - construction
  - dictionary structures
- Text processing
  - tokenization
  - normalization
- **Boolean querying**
  - **AND queries**
  - **large queries**
  - **flexible queries**
- Indexing dynamic collections
- Final remarks

Query processing: AND

- How do we process a query?
- What kinds of queries can we process?

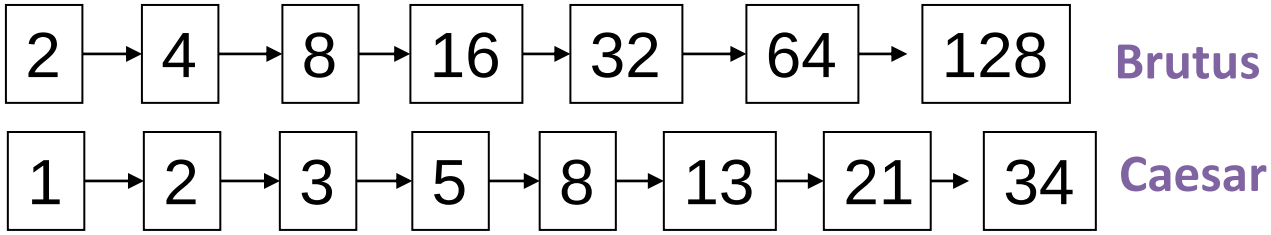
- Consider processing the query: **Brutus** AND **Caesar**
  - locate **Brutus** in the *dictionary*: retrieve its *postings*
  - locate **Caesar** in the *dictionary*: retrieve its *postings*
  - **merge** the two postings (*intersect* the document sets)
    - computational complexity? no longer a bit-wise intersection ☹



Merging postings

Walk through the two postings simultaneously

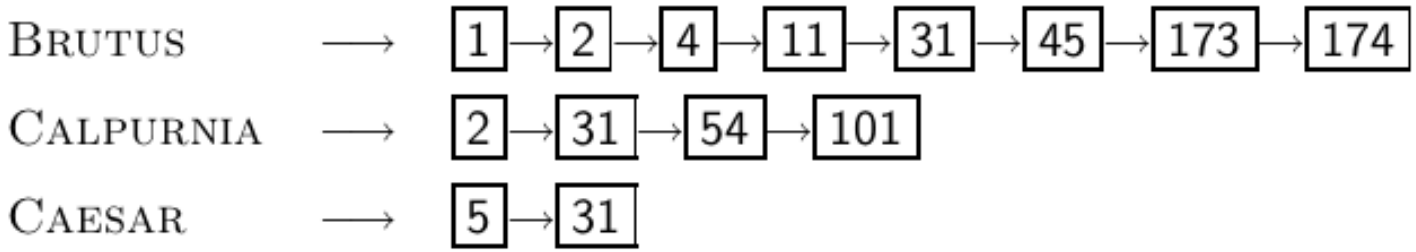
- time linear in the total number of posting entries
  - If the list lengths are x and y, the merge takes O(x+y)
  - *crucial*: postings *sorted* by docID




```
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
```

Large Boolean queries

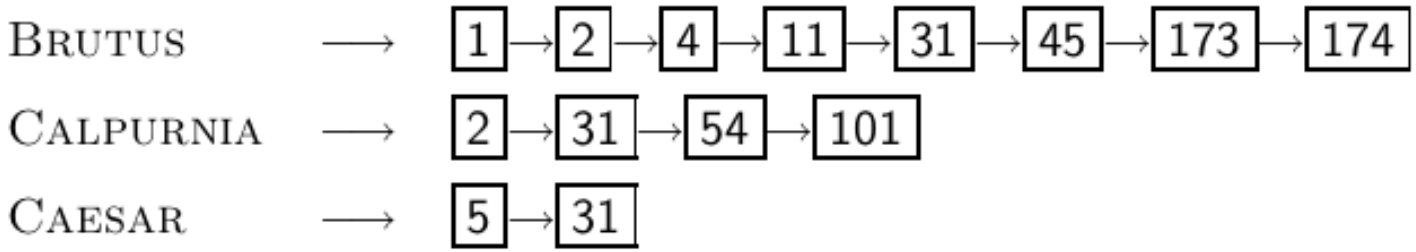
- Consider a query that is an *AND* of *n*>2 terms
  - **BRUTUS** AND **CALPURNIA** AND **CAESAR**
- How to solve?
  - for each of the *n* terms, get its postings, then *AND* them together
  - complexity? There is an optimal order for processing this query?



Query optimization

- How? process in order of increasing frequency
  - *start with smallest set, then keep cutting further*
- The why we keep doc frequency in dictionary!

```
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
```



Query: **BRUTUS** AND **CALPURNIA** AND **CAESAR**

More general queries

- *Exercise*: adapt the merge for the queries:

  Brutus AND NOT **Caesar**

  Brutus OR NOT **Caesar**

- Can we still run through the merge in linear time?
- Extend the merge to an arbitrary Boolean query
  - *hint*: begin with cases where terms appear only once in the query

- *Example*: (**madding** OR **crowd**) AND (**ignoble** OR **strife**)
  - get document frequency for all terms
  - estimate the size of each **OR** by the sum of its document frequencies (conservative)
  - process in increasing order of **OR** sizes

Additional exercises

- For the query **friends** AND **romans** AND (NOT **countrymen**) how can we use the frequency of countrymen to guide retrieval?
- Recommend a query processing order for

  (tangerine OR **trees)** AND
  (marmalade OR **skies)** AND
  (kaleidoscope OR **eyes)**

  Which two terms should we process first?

Term	Freq
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

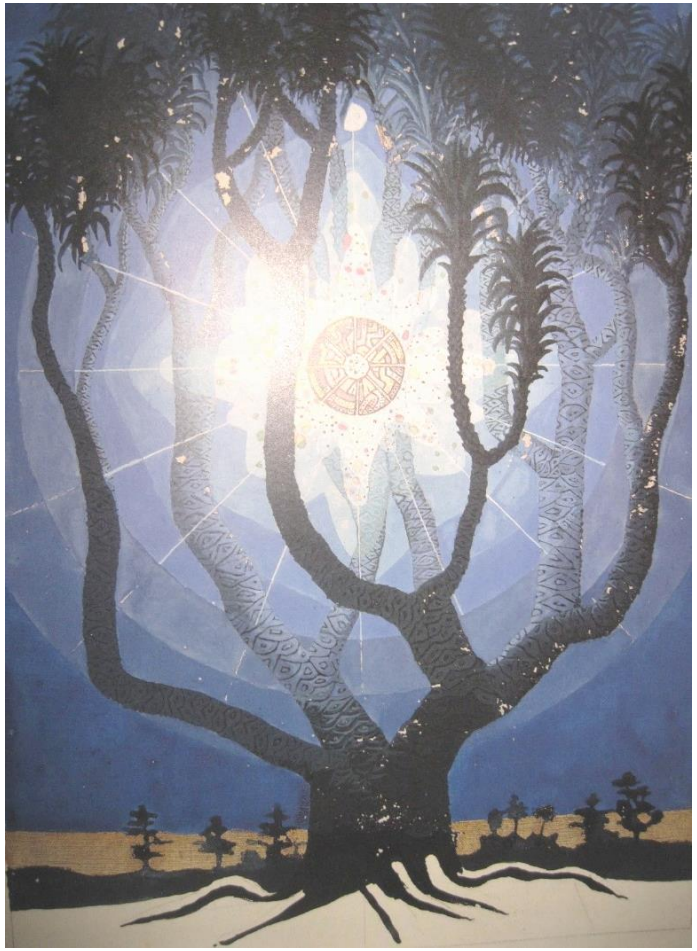
- Try the search feature at http://www.rhymezone.com/shakespeare/
  - write down *five* features to improve the Boolean search

Boolean retrieval in practice: Google

On Google

- default interpretation of a query [w_1 w_2 ... w_n] is ***w_1*** AND ***w_2*** AND ***…*** AND ***w_n***
- cases where you get hits not containing one of the w_i
  - **anchor text**
  - page contains **variant** of w_i (morphology, spelling correction, synonym)
  - **long queries** (high *n*): at most *k* terms within the inputted *n*
  - Boolean expression generates very **few hits**: complementary retrieval model
- simple Boolean vs. ranking retrieval
  - Google further ranks matched documents (according to some estimator of relevance)

Outline



- Introduction
- Inverted indexing
  - essentials
  - construction
  - dictionary structures
- Text processing
  - tokenization
  - normalization
- Boolean querying
  - AND queries
  - large queries
  - flexible queries
- **Indexing dynamic collections**
- Final remarks

Dynamic collections

- Up to now: collections are **static**
- They rarely are:
  - documents come in over time and need to be inserted
  - documents are deleted and modified

- This means that the *dictionary* and *postings* lists must be **dynamically** modified
  - implications for **indexing**?
    - postings updates for terms already in dictionary
    - new terms added to dictionary
  - implications for **querying**?

Dynamic indexing

How?

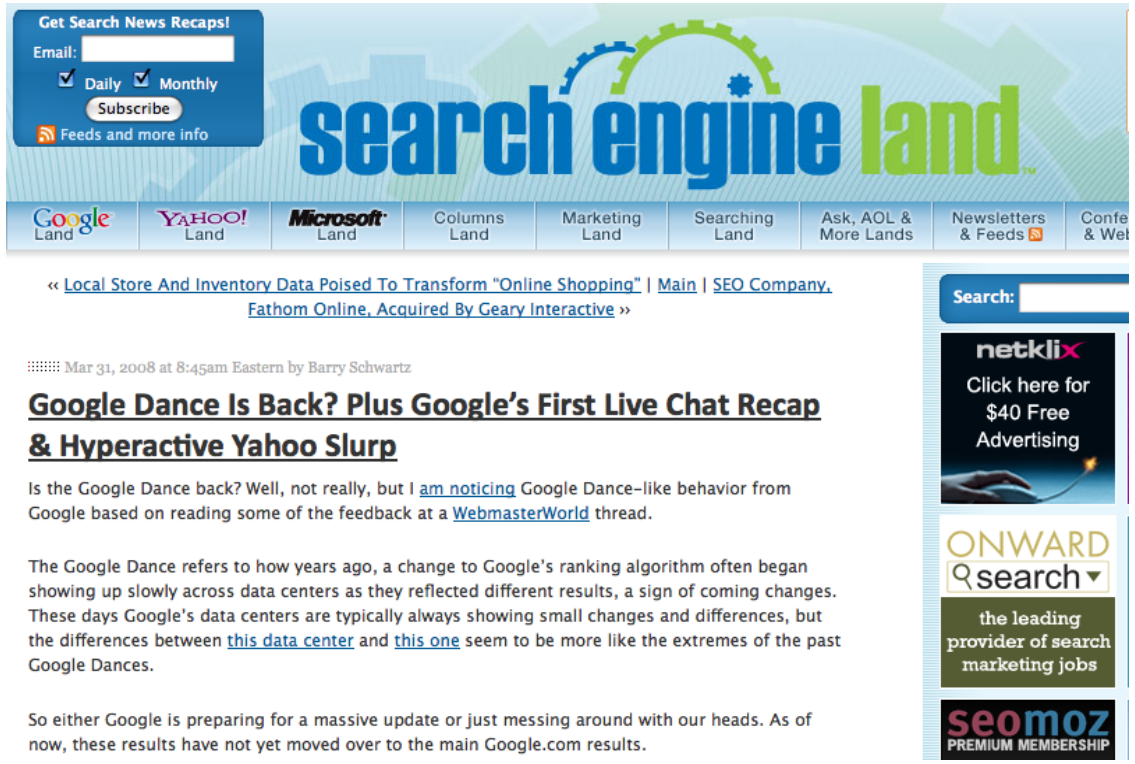
- maintain *big* main index
- new docs go into *small* auxiliary index
- search across both, merge results
- periodically re-index into one main index
  - rehash dictionary
  - efficient (ordered) concatenation of postings
- how to handle changes and deletions?

Problems?

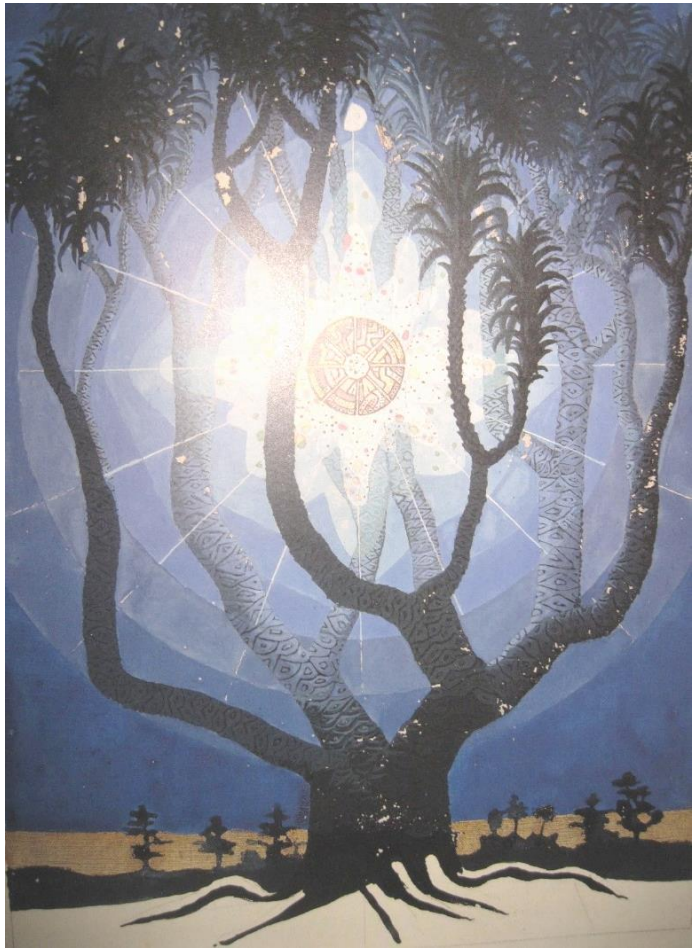
- degraded performance by the need to merge results per query and slowing down during re-indexing
- two poles
  - large auxiliary index to prevent frequent reindexing (yet heavy reindexing) versus…
  - all changes in the main index (yet impractical for ongoing queries)
- solution? somewhere in between! Optimal auxiliary index size and merging principles?
  - logarithmic merge principles (to be recovered in the web chapter)

Dynamic indexing and querying at search engines

- All the large search engines now do dynamic indexing
- Their indices have frequent incremental changes
  - new items, blogs, topical web pages
- But they also periodically reconstruct the index from scratch
  - query processing is then switched to the new index, and the old index is deleted
- How, in the meantime, we query two indexes?
  - adapt INTERSECT



Outline



- Introduction
- Inverted indexing
  - essentials
  - construction
  - dictionary structures
- Text processing
  - tokenization
  - normalization
- Boolean querying
  - AND queries
  - large queries
  - flexible queries
- Indexing dynamic collections
- **Final remarks**

Boolean model: exact match

Boolean retrieval model

- ask a query that is a Boolean expression: **AND**, **OR** and **NOT** to join query terms
- *precise*: document matches condition or not
  - *You know exactly what you are getting!* But that doesn't mean it actually works better….
- *simple:* perhaps the simplest IR model

- primary commercial retrieval tool for 3 decades
- many search systems you still use are Boolean:
  - e-mail
  - library catalog
  - Mac OS X Spotlight

Example: WestLaw

https://content.next.westlaw.com/Search/AdvancedSearchPage.html

- largest commercial (paying subscribers) legal search service
  - +700,000 users, tens of terabytes of data
  - started 1975; ranking added 1992; federated search added 2010
- majority of users still use Boolean queries
- precise queries with *proximity operators* (next lectures)

- example queries:
  1. *"what is the statute of limitations in cases involving the federal tort claims act?"*
     - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
     - /3 = within 3 words, /S = in same sentence
     - SPACE is disjunction, not conjunction!
  2. *"requirements for disabled people to be able to access a workplace?"*
     - disabl! /p access! /s work-site work-place (employment /3 place

Thank You

rmch@tecnico.ulisboa.pt