

CSE 4/535

Information Retrieval

Sayantan Pal
PhD Student, Department of CSE
338Z Davis Hall



Department of CSE

Before we start

1. AI Quiz released due this Friday.
2. Project 1 will be released this Friday (Discussion: next week)
3. New students: Go through the AI Policy, slides.
4. Remind me when last 10 mins will be left (If I had not finished already)
5. If you are writing me an email, please DO NOT use ChatGPT. Keep it simple.



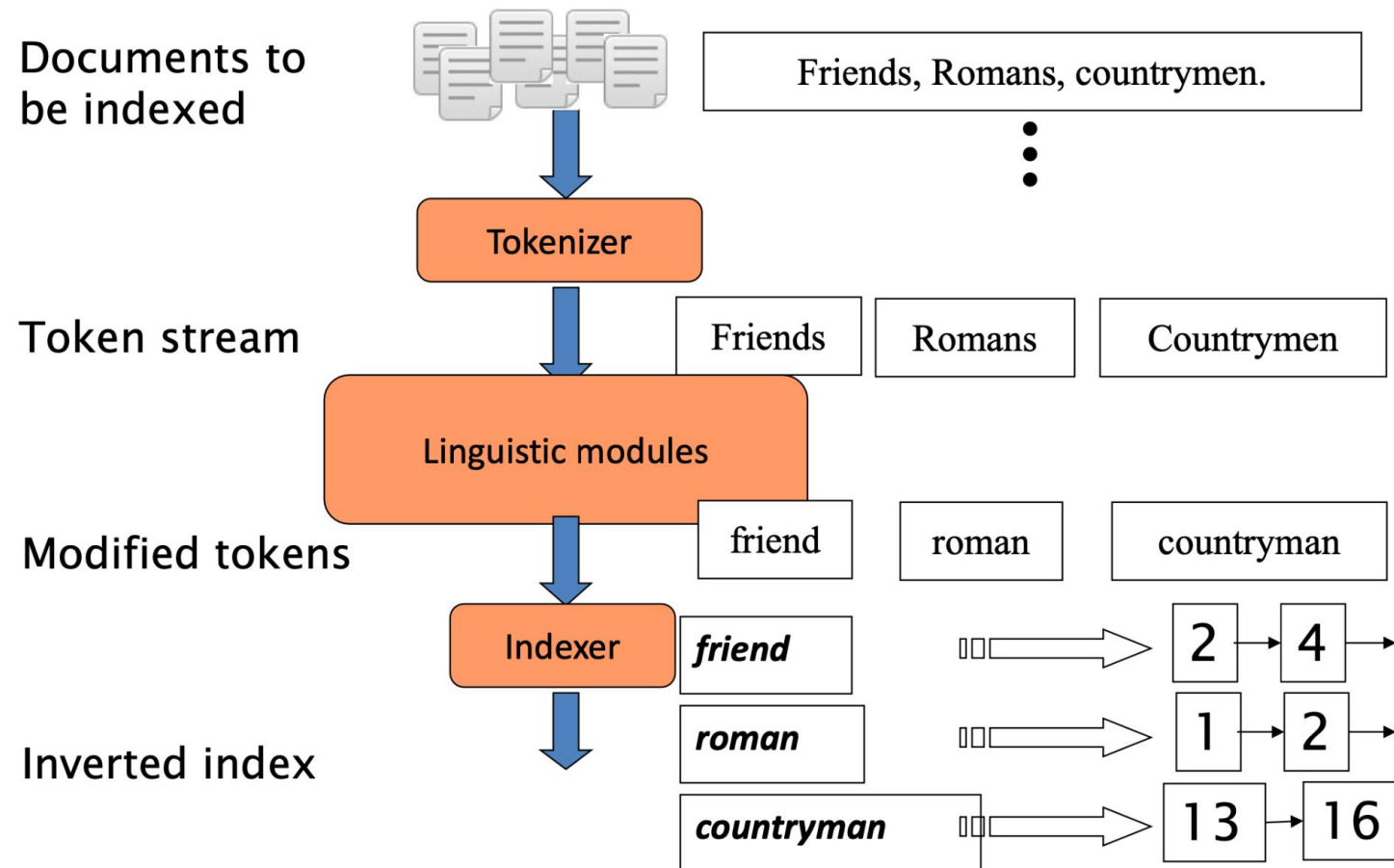
Recap - Previous Class

1. What is IR
2. How good are the retrieved docs?
3. Inverted Index construction
4. Boolean Model

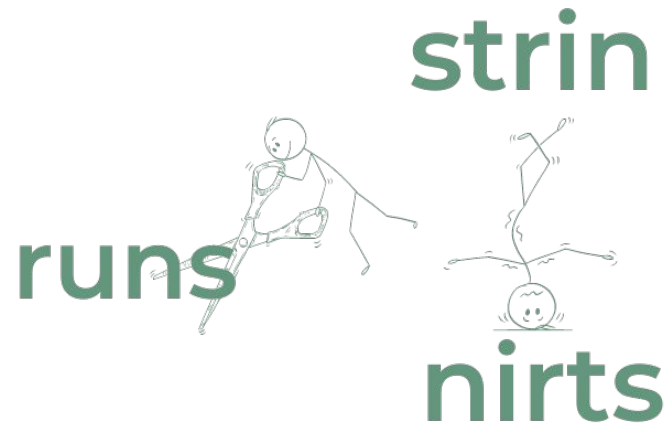




Recall the basic indexing pipeline

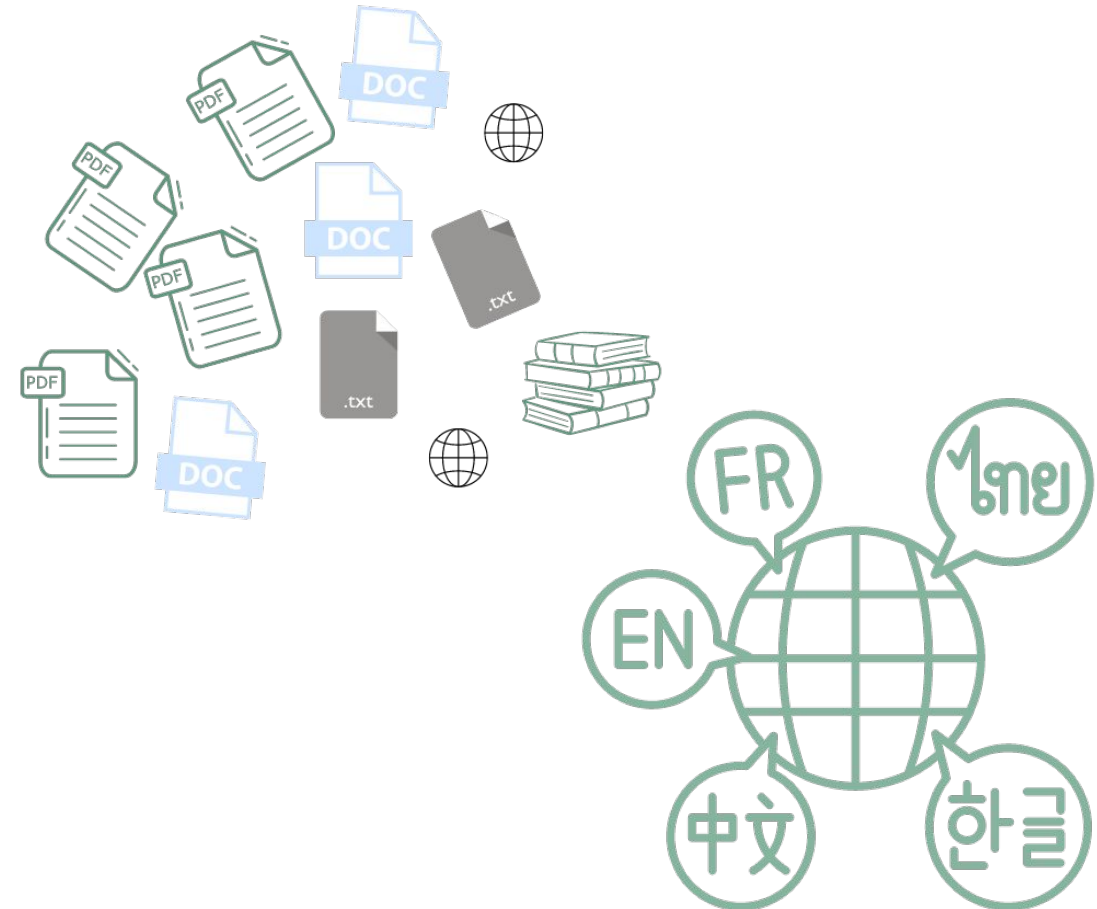


Tokenization, Stemming, Lemmatization



Parsing a document

- What format is it in?
 - pdf/word/excel/html?
- What language is it in?
- What character set is in use?
 - (CP1252, UTF-8, ...)

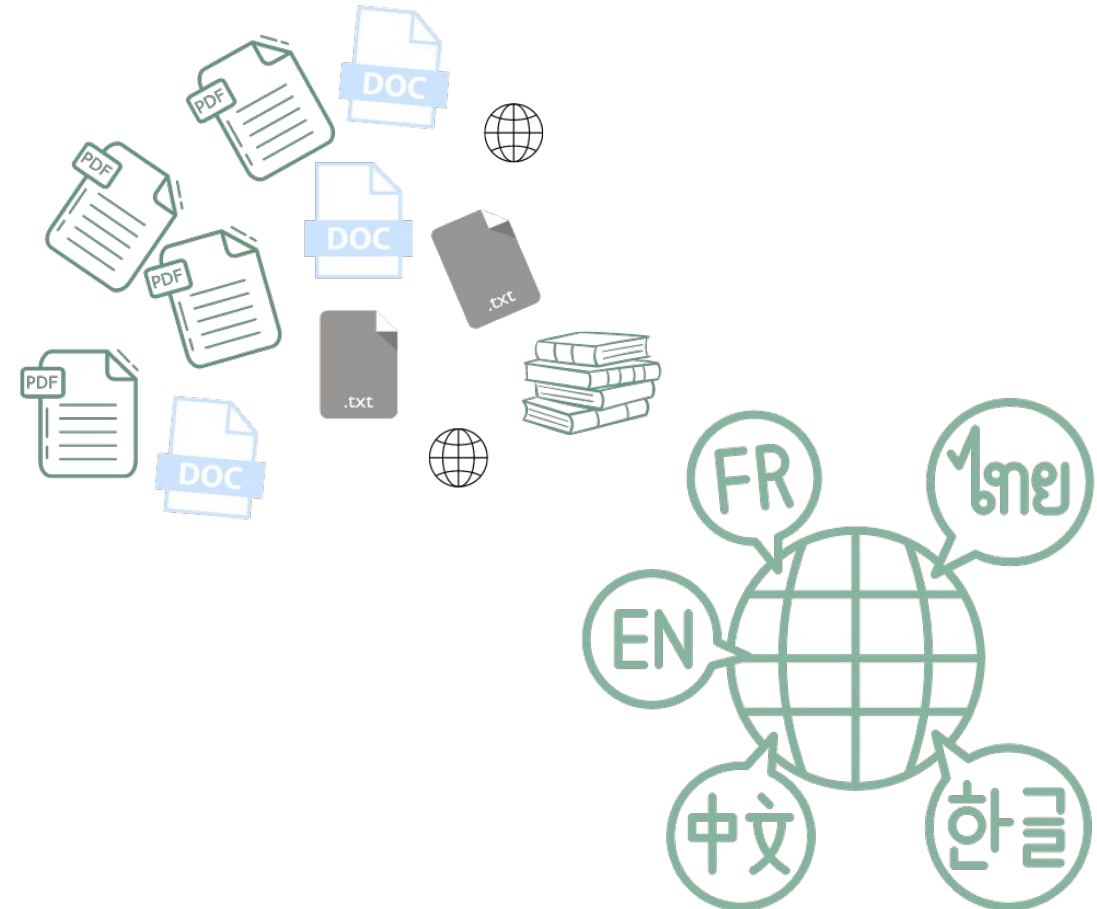


Parsing a document

- What format is it in?
 - pdf/word/excel/html?
- What language is it in?
- What character set is in use?
 - (CP1252, UTF-8, ...)

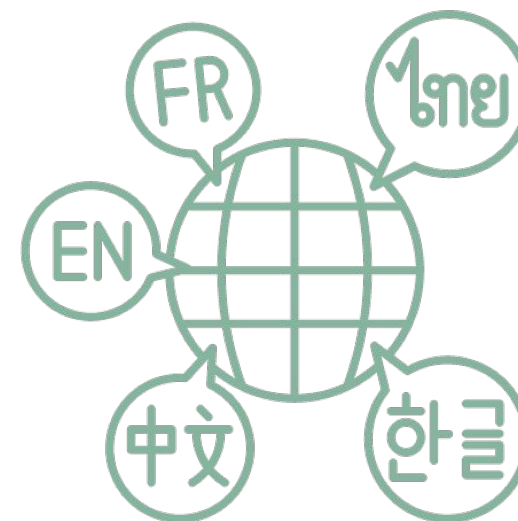
Each of these is a classification problem, which we will study later in the course.

But these tasks are often done heuristically ...



Complications: Format/language

1. Documents being indexed can include docs from many different languages
 - a. A single index may contain terms from many languages.
2. Sometimes a document or its components can contain multiple languages/formats
 - a. French email with a German pdf attachment.
 - b. French email quote clauses from an English-language contract
3. There are commercial and open source libraries that can handle a lot of this stuff





Complications: What is a document?

1. We return from our query “documents” but there are often interesting questions of grain size:
2. What is a unit document?
 - a. A file?
 - b. An email? (Perhaps one of many in a single mbox file)
 - i. What about an email with 5 attachments?
 - c. A group of files (e.g., PPT or LaTeX split over HTML pages)
 - d. A social media post (include replies?)





Tokens





Tokenization

- Input: “Friends, Romans and Countrymen”
- Output: Tokens
 - Friends
 - Romans
 - Countrymen
- A token is an [instance of a sequence of characters](#)
- Each such token is now a candidate for an index entry, after further processing





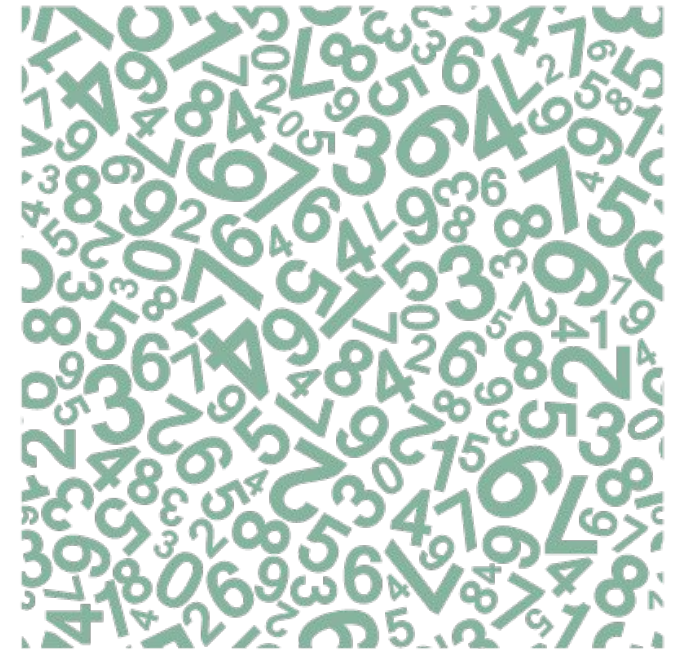
Issues in tokenization

- Finland's capital -> Finland AND s? Finlands? Finland's?
- Hewlett-Packard -> Hewlett and Packard as two tokens?
 - state-of-the-art: break up hyphenated sequence.
 - co-education
 - lowercase, lower-case, lower case?
 - It can be effective to get the user to put in possible hyphens
- San Francisco: one token or two?
 - How do you decide it is one token?



Numbers

- 3/20/91 Mar. 12, 1991 20/3/91 , 55 B.C., B-52
- My PGP key is 324a3df234cb23e
- (800) 234-2333
 - Often have embedded spaces
 - Older IR systems may not index numbers
 - But often very useful: think about things like looking up error codes/stack traces on the web
 - (One answer is using n-grams)
 - Will often index “meta-data” separately
 - Creation date, format, etc.





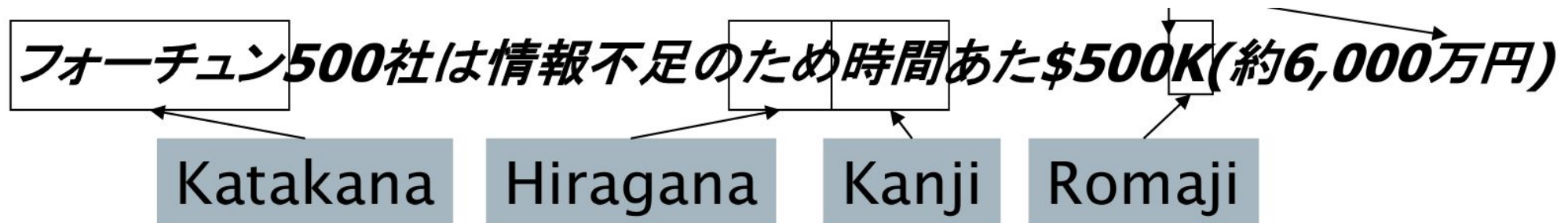
Tokenization: language issues

- French
- L'ensemble -> one token or two?
 - L ? L' ? Le ?
 - Want l'ensemble to match with un ensemble
 - Until at least 2003, it didn't on Google
 - Internationalization!
- German noun compounds are not segmented
 - Lebensversicherungsgesellschaftsangestellter
 - 'life insurance company employee'
 - German retrieval systems benefit greatly from a compound splitter module
 - Can give a 15% performance boost for German



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





Tokenization: language issues

- Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right
- Words are separated, but letter forms within a word form complex ligatures
- '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

استقلت الجزائر في سنة 1962 بعد 132 عام من الاحتلال الفرنسي.

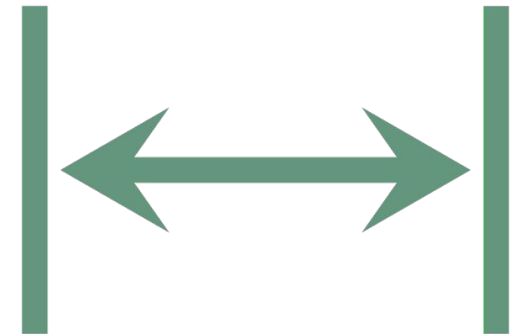
← → ← →

← start



Equivalence Classes

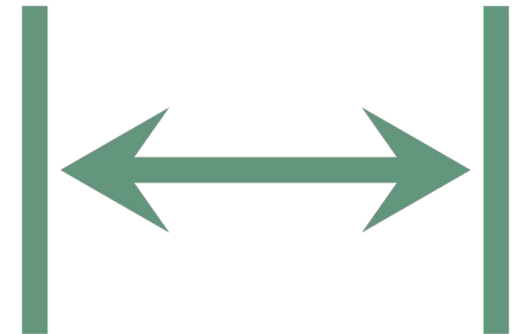
- Handle synonyms and homonyms
 - Equivalence Classes: Mapping rules that remove characters like hyphens: terms that happen to become identical are the equivalence classes
 - Hand-constructed equivalence classes (implicit)
 - e.g., car = automobile
 - your and you're
 - Soundex: Chebyshev, Tchebycheff (will be discussed later)





Equivalence Classes

- Handle synonyms and homonyms
 - Equivalence Classes: Mapping rules that remove characters like hyphens: terms that happen to become identical are the equivalence classes
 - Hand-constructed equivalence classes (implicit)
 - e.g., car = automobile
 - your and you're
 - Soundex: Chebyshev, Tchebycheff (will be discussed later)
- Index such equivalences
 - When the document contains automobile, index it under car as well (usually, also vice-versa)
- Or expand query?
 - When the query contains automobile, look under car as well



Terms



Stop words

- With a stop list, you exclude from the dictionary entirely the commonest words.
Intuition:
 - They have little semantic content: the, a, and, to, be
 - There are a lot of them: ~30% of postings for top 30 words
- But the trend is away from doing this:
 - Good compression techniques means the space for including stop words in a system is very small
 - Good query optimization techniques mean you pay little at query time for including stop words.
 - You need them for:
 - Phrase queries: “King of Denmark”
 - Various song titles, etc.: “Let it be”, “To be or not to be”
 - “Relational” queries: “flights to London”



OAC Stopword list (~ 275 words)*

The Complete OAC-Search Stopword List

a	did	has	nobody	somewhere	usually
about	do	have	noone	soon	very
all	does	having	nor	still	was
almost	doing	here	not	such	way
along	done	how	nothing	than	ways
already	during	however	now	that	we
also	each	i	of	the	well
although	either	if	once	their	went
always	enough	ii	one	theirs	were
am	etc	in	one's	then	what
among	even	including	only	there	when
an	ever	indeed	or	therefore	whenever
and	every	instead	other	these	where
any	everyone	into	others	they	wherever
anyone	everything	is	our	thing	whether
anything	for	it	ours	things	which

Stopword list contd.*

anywhere	from	its	out	this	while
are	further	itself	over	those	who
as	gave	just	own	though	whoever
at	get	like	perhaps	through	why
away	getting	likely	rather	thus	will
because	give	may	really	to	with
between	given	maybe	same	together	within
beyond	giving	me	shall	too	without
both	go	might	should	toward	would
but	going	mine	simply	until	yes
by	gone	much	since	upon	yet
can	good	must	so	us	you
cannot	got	neither	some	use	your
come	great	never	someone	used	yours
could	had	no	something sometimes	uses using	yourself



Normalization to terms

- We may need to “normalize” words in indexed text as well as query words into the same form
 - We want to match U.S.A. and USA
- Result is terms: a term is a (normalized) word type, which is an entry in our IR system dictionary
- We most commonly implicitly define equivalence classes of terms by, e.g.,
 - deleting periods to form a term
 - U.S.A., USA
 - deleting hyphens to form a term
 - anti-discriminatory, antidiscriminatory

Normalization: other languages

- Accents: e.g., French résumé vs. resume.
- Umlauts: e.g., German: Tuebingen vs. Tübingen
 - Should be equivalent
- Most important criterion:
 - How are your users like to write their queries for these words?
- Even in languages that standardly have accents, users often may not type them
 - Often best to normalize to a de-accented term
 - Tuebingen, Tübingen -> Tübingen

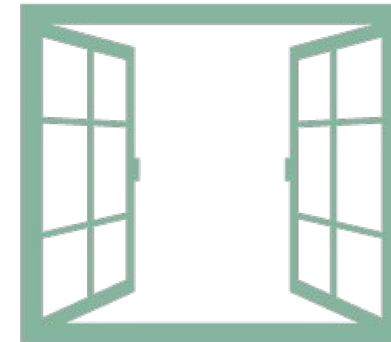
Case folding

- Reduce all letters to lowercase
 - exception: upper case in mid-sentence?
 - e.g., General Motors
 - Fed vs. fed
 - SAIL vs. sail
 - Often best to lowercase 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.



Normalization to terms

- An alternative to equivalence classing is to do [asymmetric expansion](#)
- An example of where this may be useful
 - Enter: window, Search: window, windows
 - Enter: windows, Search: Windows, windows, window
 - Enter: Windows, Search: Windows
- Potentially more powerful, but less efficient





Thesauri and soundex

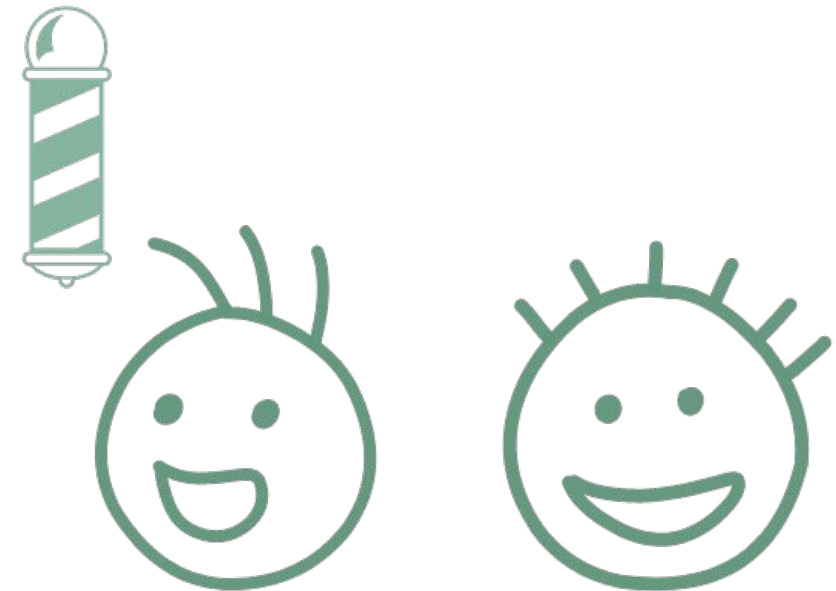
- Do we handle synonyms and homonyms?
 - E.g., by hand-constructed equivalence classes
 - car = automobile, color = colour
 - We can rewrite to form equivalence-class terms
 - When the document contains automobile, index it under car automobile(and vice-versa)
 - Or we can expand a query
 - When the 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 (will be discussed later)

Stemming and Lemmatization



Lemmatization vs. Stemming: The Haircut You Never Knew Words Needed!

- Lemmatization: The VIP Haircut — Keeps the Essence!
- Stemming: The Lawnmower Treatment — Quick and Dirty!
- Words, like hair, need occasional grooming. But not all cuts are equal!
- Lemmatization gives your words a salon-quality trim. Stemming? Well, let's just say it's the DIY buzz cut you gave yourself at 2 AM.
- Examples:
 - Original Word: "Flies"
 - Lemmatized Version: "Fly"
 - Stemmed Version: "Fli"





Lemmatization

- Reduce inflectional/variant forms to Base form
 - E.g.,
 - am, are, is -> be
 - car, cars, car's, cars' -> car
- the boy's cars are different colors -> the boy car be different color
- Lemmatization implies doing “proper” reduction to dictionary headword form
- WordNet: https://www.nltk.org/_modules/nltk/stem/wordnet.html



Stemming

- Reduce terms to their “roots” before indexing
- “Stemming” suggests crude affix chopping
 - Language dependent
 - e.g., automate(s), automatic, automationall reduced to automat.

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



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



Porter's algorithm

- Most common algorithm for stemming English
 - Results suggest it's at least as good as other stemming options
- Conventions: 5 phases of reductions
 - phases applied sequentially
 - each phase consists of a set of commands
 - sample convention: Of the rules in a compound command, select the one that applies to the longest suffix.

Typical rules in Porter

- *sses* → *ss*
- *ies* → *i*
- *ational* → *ate*
- *tional* → *tion*

- Weight of word sensitive rules
- $(m > 1)$ *EMENT* →
 - *replacement* → *replac*
 - *cement* → *cement*

Other stemmers

- Other stemmers exist:
 - Lovins stemmer:
 - <http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm>
 - Single-pass, longest suffix removal (about 250 rules)
 - Paice/Husk stemmer
 - Snowball
- Full morphological analysis (lemmatization)
 - At most modest benefits for retrieval

Stemming variants

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

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 interpre

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

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
 - E.g., operative (dentistry) \Rightarrow oper
- Definitely useful for Spanish, German, Finnish, ...
 - 30% performance gains for Finnish!



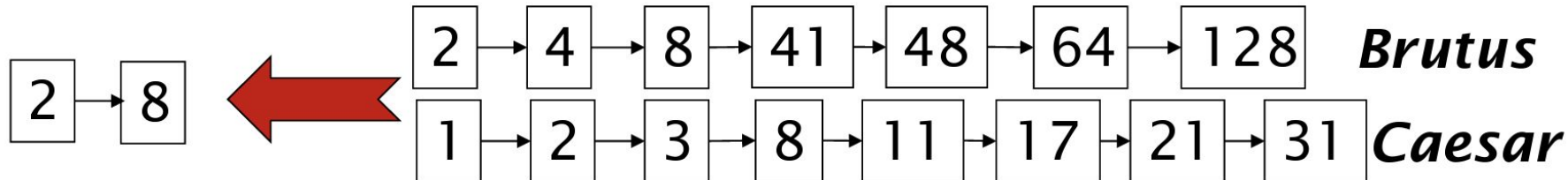
Faster postings merges: Skip pointers/Skip lists





Recall basic merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries

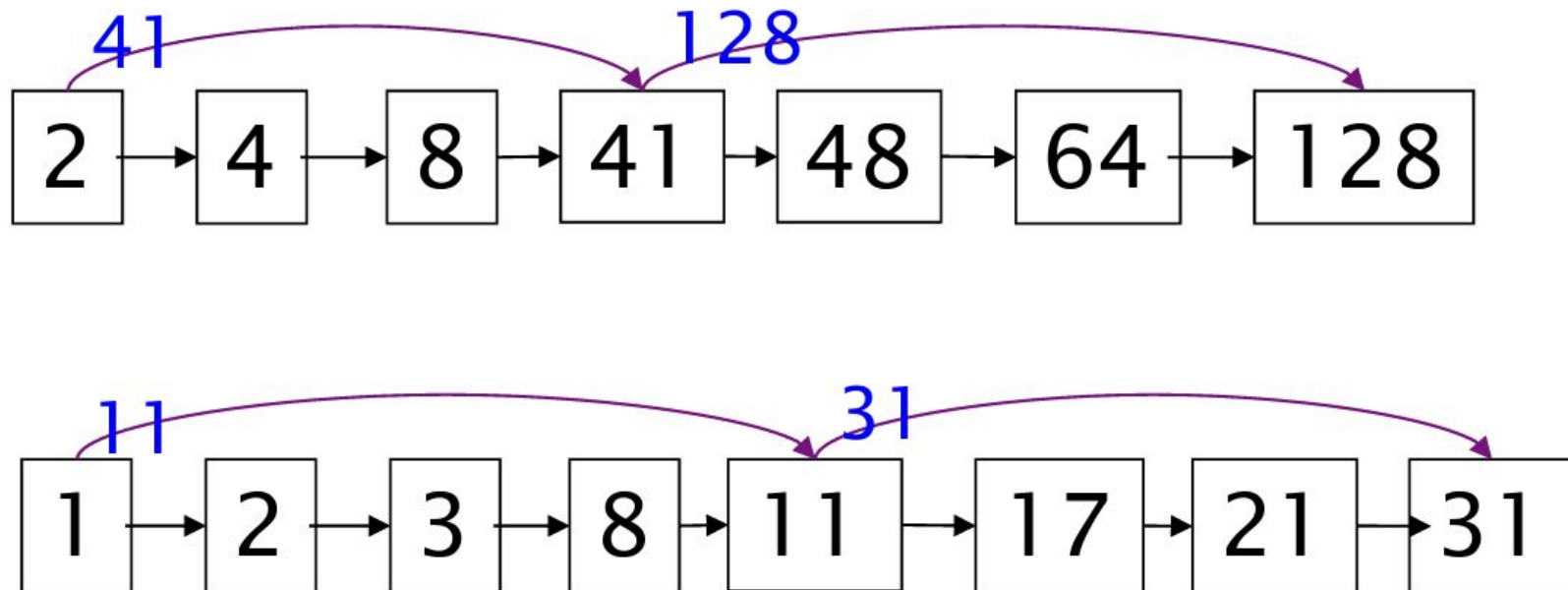


If the list lengths are m and n , the merge takes $O(m+n)$ operations.

Can we do better?

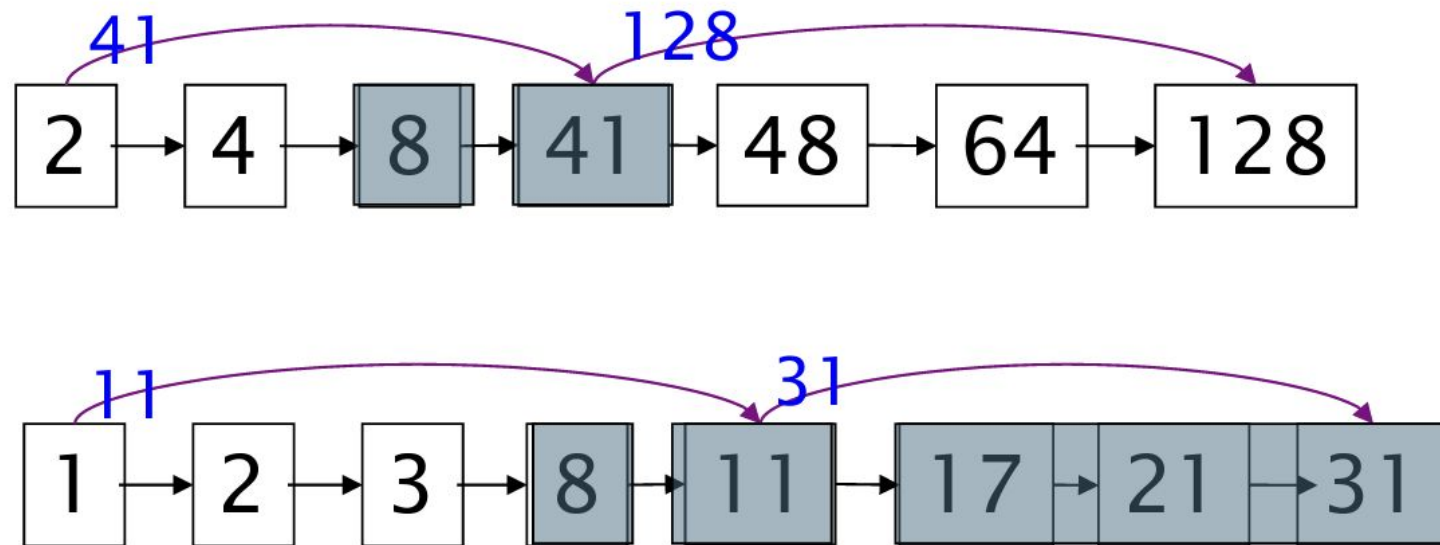
Yes (if the index isn't changing too fast).

Augment postings with skip pointers (at indexing time)





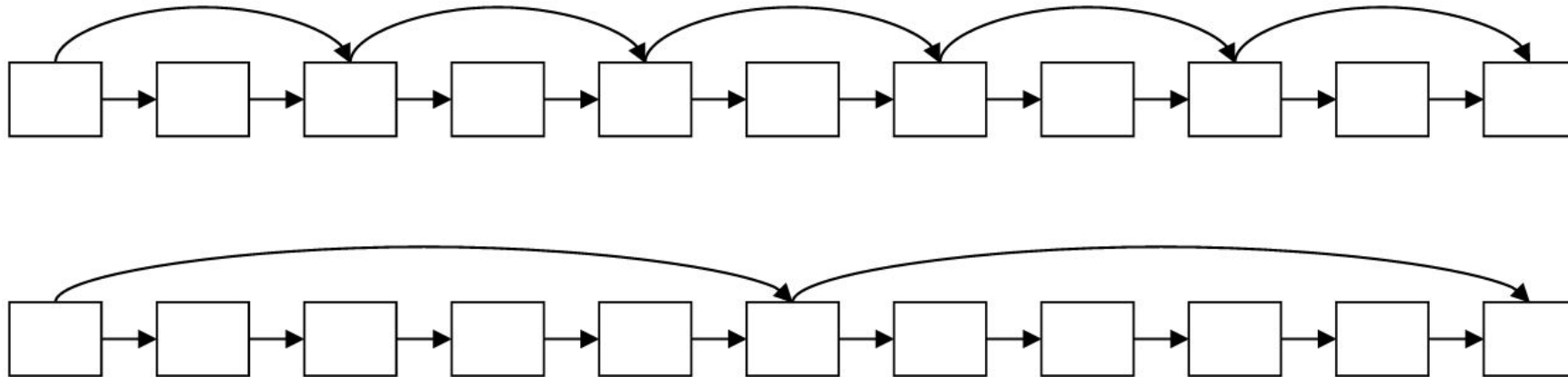
Query processing with skip pointers



- Suppose we've stepped through the lists until we process 8 on each list. We match it and advance. We then have 41 and 11 on the lower. 11 is smaller. But the skip successor of 11 on the lower list is 31, so we can skip ahead past the intervening postings

Where do we place skips?

- Tradeoff:
- More skips \rightarrow shorter skip spans \Rightarrow more likely to skip. But lots of comparisons to skip pointers.
- Fewer skips \rightarrow few pointer comparison, but then long skip spans \Rightarrow few successful skips.





Placing skips

- Simple heuristic: for postings of length L , use \sqrt{L} evenly-spaced skip pointers [Moffat and Zobel 1996]
- This ignores the distribution of query terms.
- Easy if the index is relatively static; harder if L keeps changing because of updates.
- This definitely used to help; with modern hardware it may not unless you're memory-based [Bahle et al. 2002]
- The I/O cost of loading a bigger postings list can outweigh the gains from quicker in memory merging!

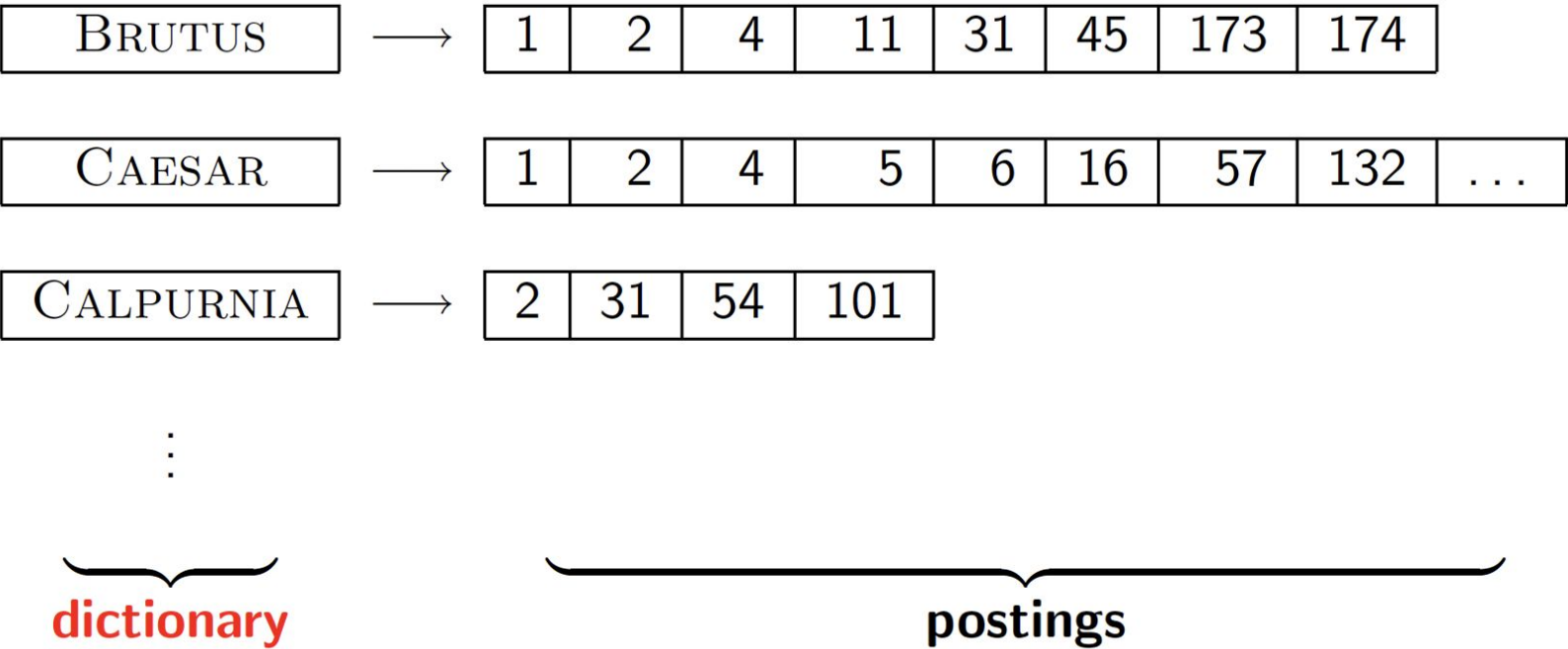
Tolerant Retrieval



Dictionary data structures for inverted indexes

- The dictionary data structure stores the term vocabulary, document frequency, pointers to each postings list

 ...in what data structure?



A naïve dictionary

- An array of struct:

term	document frequency	pointer to postings list
a	656,265	→
aachen	65	→
...
zulu	221	→

char[20]

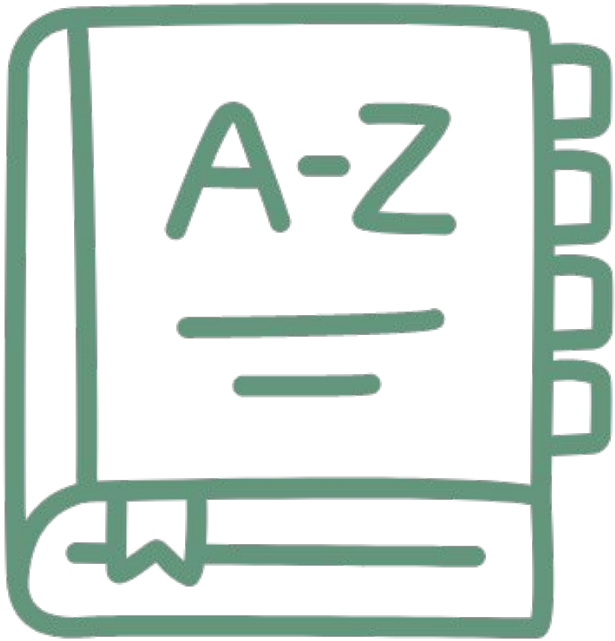
int

Postings *

20 bytes

4/8 bytes

4/8 bytes

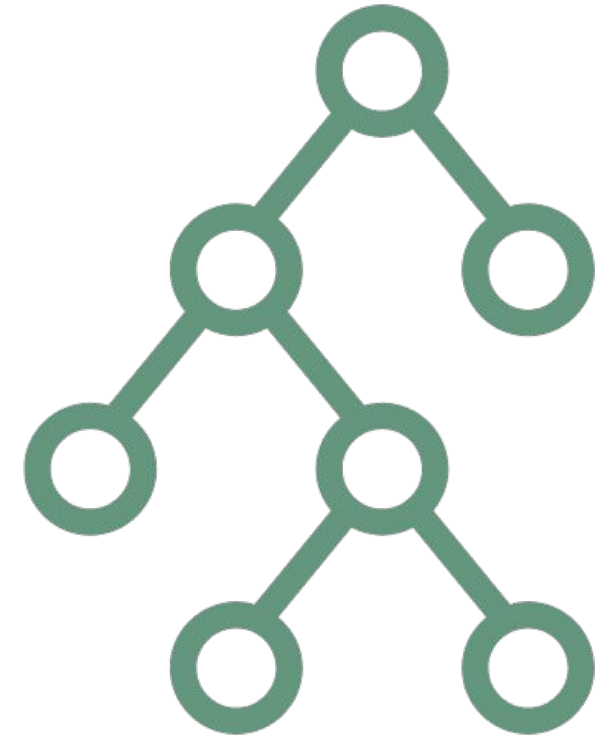


- How do we store a dictionary in memory efficiently?
- How do we quickly look up elements at query time?



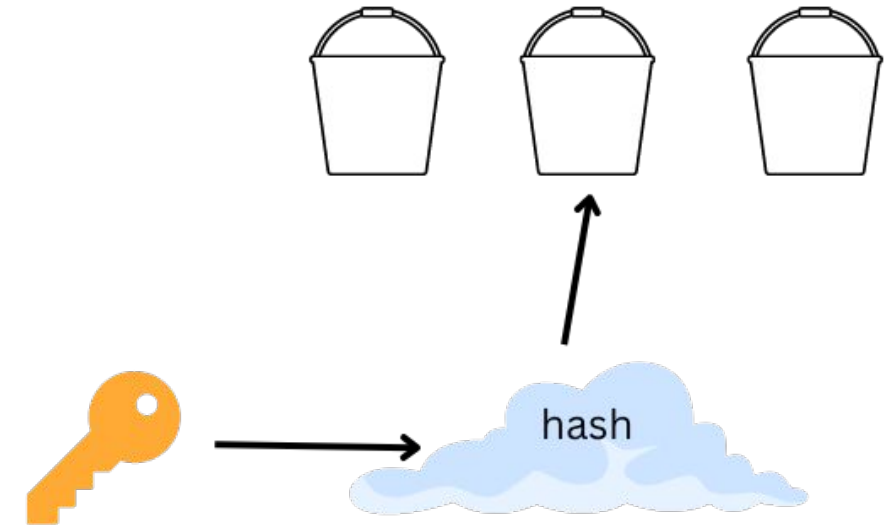
Dictionary

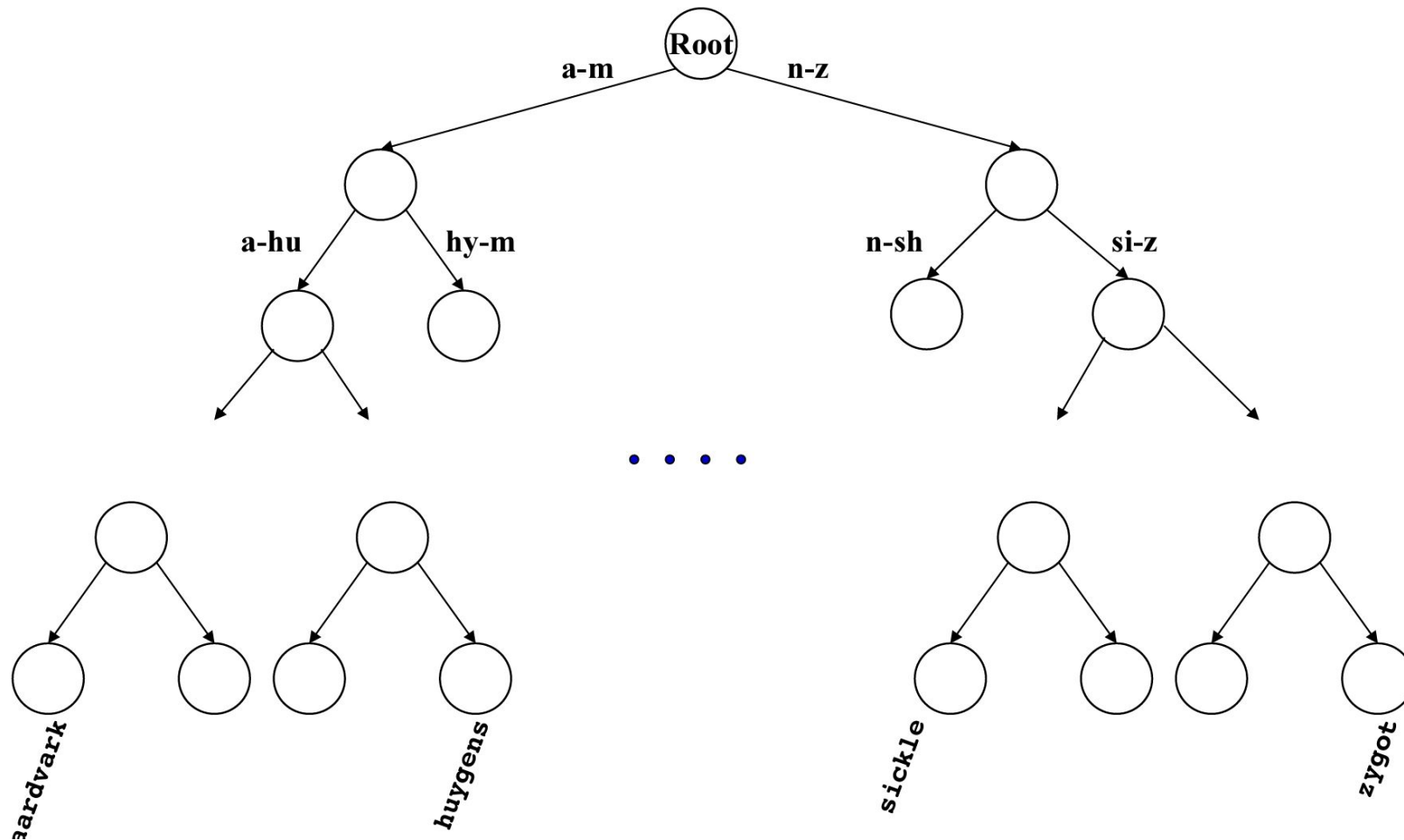
- Hash Tables
 - Each vocabulary entry hashed into an integer over a large enough space to avoid collisions
- Binary Tree
 - $O(\log n)$ time to search
 - Needs to be balanced
- B-tree
 - Allows tree to be balanced
 - Every internal node has between a and b children



Hashes

- Each vocabulary term is hashed to an integer
 - (We assume you've seen hash tables 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 going, need to occasionally do the expensive operation of rehashing everything

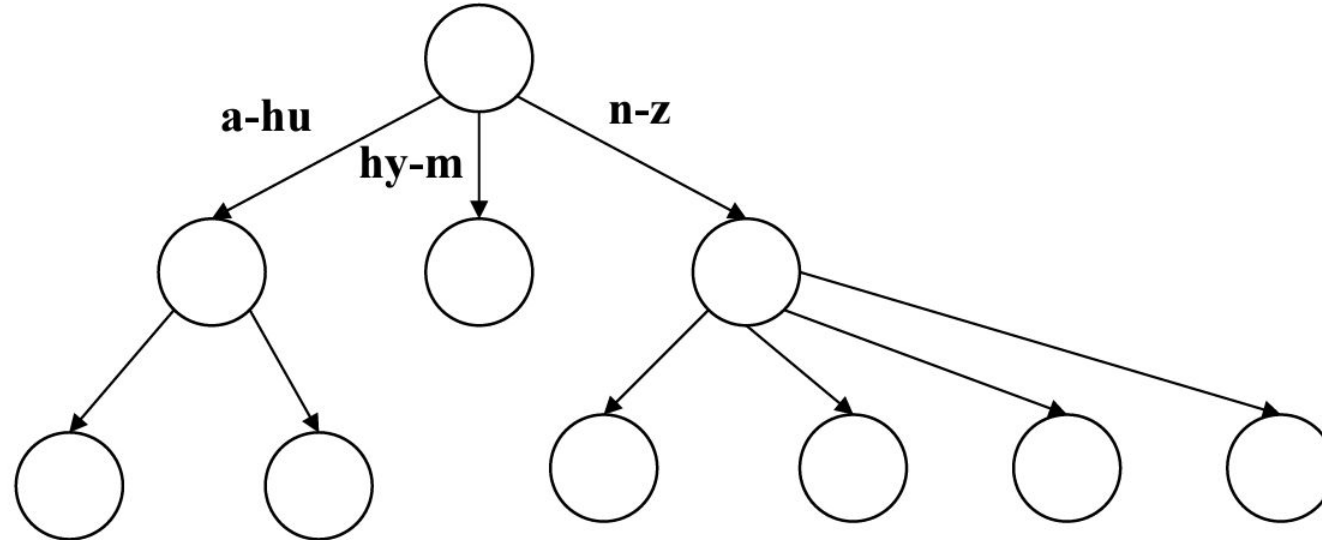






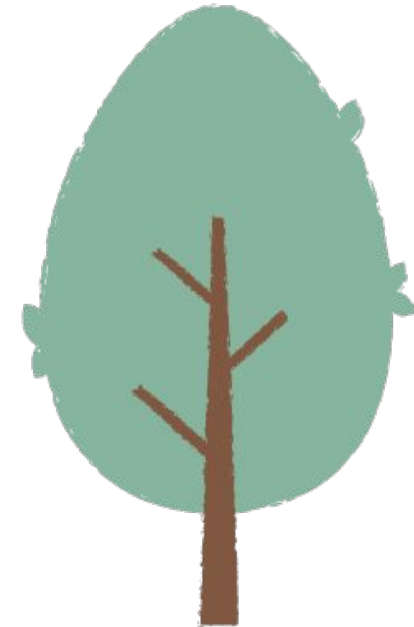
Tree: B-tree

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



Trees

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



Wild-card queries:

- `mon*`: find all docs containing any word beginning “mon”.
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range:
 $\text{mon} \leq w < \text{moo}$
- `*mon`: find words ending in “mon”: harder



Wild-card queries:

- `mon*`: find all docs containing any word beginning “mon”.
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range:
 $\text{mon} \leq w < \text{moo}$
- `*mon`: find words ending in “mon”: harder
 - Maintain an additional B-tree for terms backwards. Can retrieve all words in range: $\text{nom} \leq w < \text{non}$.



Wild-card queries:

- `mon*`: find all docs containing any word beginning “mon”.
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range:
 $\text{mon} \leq w < \text{moo}$
- `*mon`: find words ending in “mon”: harder
 - Maintain an additional B-tree for terms backwards. Can retrieve all words in range: $\text{nom} \leq w < \text{non}$.
- Exercise: from this, how can we enumerate all terms meeting the wild-card query `pro*cent`?





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

- How can we handle *'s in the middle of query term?
 - (Especially multiple *'s)



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

- How can we handle *'s in the middle of query term?
 - (Especially multiple *'s)
- The solution: transform every wild-card query so that the *'s occur at the end
- This gives rise to the Permuterm Index.



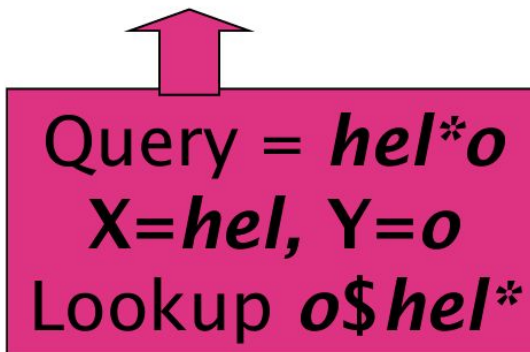
Permuterm index

- For term hello index under:
- hello\$, ello\$h, llo\$he, lo\$hel, o\$hell
- where \$ is a special symbol.



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*
 - X*Y lookup on Y\$X*



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

The idea behind Permuterm Index is to rotate wildcard query such that * goes to the end.



Bigram indexes

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

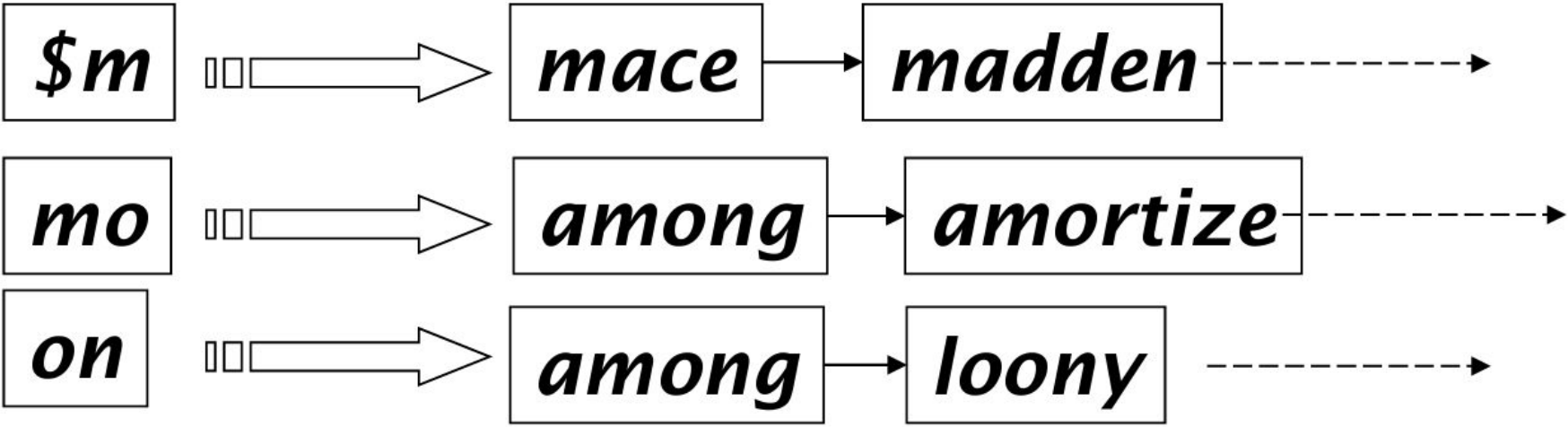
\$a,ap,pr,ri,il,l\$, \$i,is,s\$, \$t,th,he,e\$, \$c,cr,ru,
 ue,el,le,es,st,t\$, \$m,mo,on,nt,h\$

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





Bigram index example





Processing n-gram wild-cards

- Query mon* can now be run as
 - \$m AND mo AND on
- Fast, space efficient.
- Gets terms that match AND version of our wildcard query. Any issues?



Processing n-gram wild-cards

- Query `mon*` can now be run as
 - `$m AND mo AND on`
- Fast, space efficient.
- 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.



Processing wild-card queries

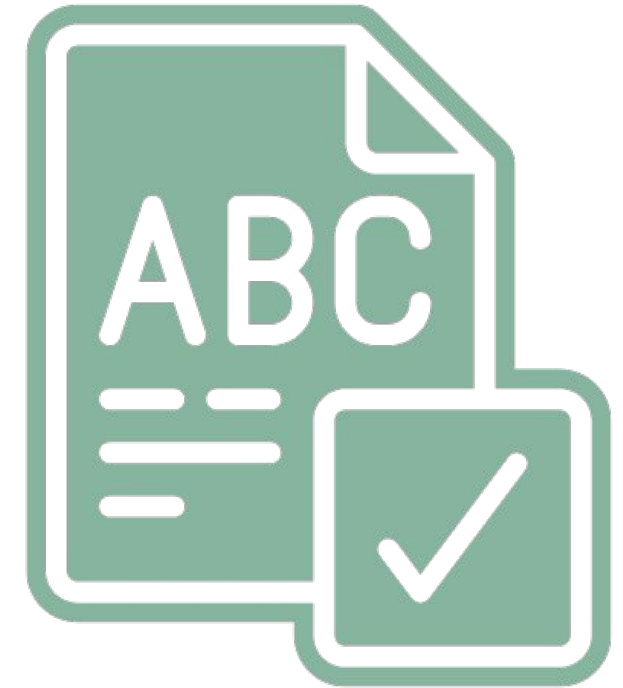
- It is expensive, thus hidden

Search

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

Spell correction

- Two principal uses
 - Correcting document(s) being indexed
 - Retrieve matching documents when query contains a spelling error
- Two main flavors:
 - Isolated word
 - Check each word on its own for misspelling
 - Will not catch typos resulting in correctly spelled words
e.g., from -> form
 - Context-sensitive
 - Look at surrounding words, e.g.,
 - I flew form Heathrow to Narita.





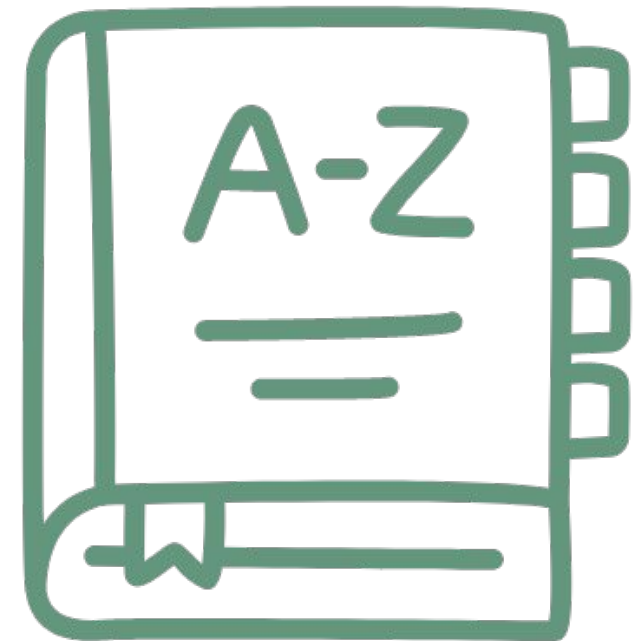
Query mis-spellings

- Retrieve documents indexed by the correct spelling, OR
- Return several suggested alternative queries with the correct spelling
 - Did you mean ... ?



Isolated word correction

- Fundamental premise – there is a lexicon from which the correct spellings come
- Two basic choices for this
 - A standard lexicon such as
 - Webster's English Dictionary
 - An "industry-specific" lexicon – hand-maintained
 - The lexicon of the indexed corpus
 - E.g., all words on the web
 - All names, acronyms etc.
 - (Including the mis-spellings)
 - For phrase correction, there may be a third choice –what is it?
- Query Logs!!!





Isolated word correction

- Given a lexicon and a character sequence Q , return the words in the lexicon closest to Q
- What's "closest"?
 - E.g. query is "grnt"
 - What should this match? How do you define "closest"?
 - For two choices that are about the same in closeness, which one do you choose?
- We'll study several alternatives
 - Edit distance
 - Weighted edit distance
 - n-gram overlap



Edit distance

- Given two strings S_1 and S_2 , the minimum number of basic operations to convert one to the other
- Basic operations are typically character-level
 - Insert
 - Delete
 - Replace
- E.g., the edit distance from cat to dog is 3.
- Generally found by dynamic programming.
- Also known as: Levenshtein distance: <http://www.levenshtein.net/>



Weighted edit distance

- As above, but the weight of an operation depends on the character(s) involved
 - Meant to capture keyboard errors, e.g. **mmore** likely to be mis-typed as **nthan**
 - Therefore, replacing m by n is a smaller edit distance than by q
 - (Same ideas usable for OCR, but with different weights)
- Require weight matrix as input
- Modify dynamic programming to handle weights



Using edit distances

- Given query, first enumerate all dictionary terms within a preset (weighted) edit distance
- Then look up enumerated dictionary terms in the term-document inverted index
 - Slow but no real fix
 - Tries help
- Given a (misspelled) query – do we compute its edit distance to every dictionary term?
 - Expensive and slow
- How do we cut the set of candidate dictionary terms?
 - E.g. Restrict to those that start with the same letter
- Here we use n-gram overlap for this.... (will be covered next week along with soundex)

References

1. Slides provided by Sougata Saha (Instructor, Fall 2022 - CSE 4/535)
2. Materials provided by Dr. Rohini K Srihari
3. <https://nlp.stanford.edu/IR-book/information-retrieval-book.html>