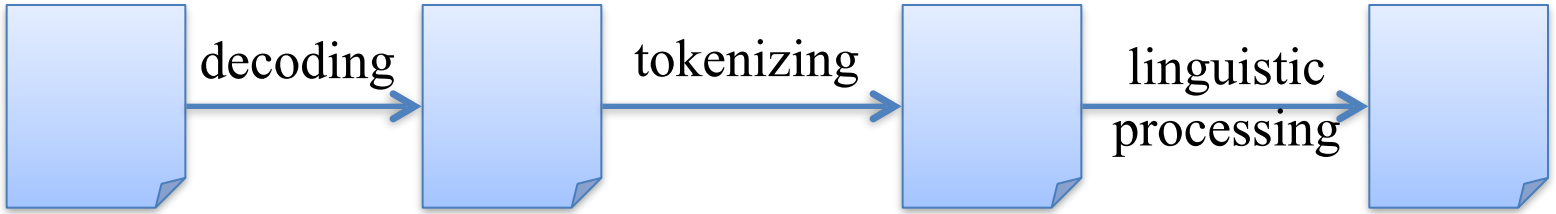
Dictionaries and Tolerant Retrieval

Debapriyo Majumdar
Information Retrieval
Indian Statistical Institute Kolkata

Pre-processing of a document



document
text, word, XML, ...

simple text sequence of characters
ASCII, UTF-8

sequence of tokens
ASCII, UTF-8

sequence of processed tokens
ASCII, UTF-8

The dictionary

Tokenization

- Simplest: split sentences by whitespace
  - But, that's too naive, not good enough
- A punctuation may not mean end of sentence, or even a word
  - Acronyms: M.Sc., U.S.A., …
  - Dates: 08.02.2021, 08/02/21, …
  - Decimal numbers: interest rate 4.90%
  - URLs: https://www.isical.ac.in, email IDs, hashtags
  - Some phrases (multiple words) need to be treated as "one word"
  - New York, rock 'n' roll
- Some languages do not use spaces to separate words!
- Reasonable approach: tokenize text using *regular expressions*

Stop words

- Exclude the common words from the dictionary entirely
- 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: 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.
  - We 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”

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 the 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, Tubingen → Tubingen***

Normalization: other languages

- Normalization of different date formats
  - 02/08/2021, 08/02/2021, 08 Feb 2021, …
  - Note: more intelligence required here

- Tokenization and normalization may depend on the language and so is intertwined with language detection

- Normalization may depend on the language

Morgen will ich in MIT …

Is this MIT the institution, or the German word “mit” (means “with”)

Case folding

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

- CAT
  - Does it mean the animal cat?
  - Or the common admission test?

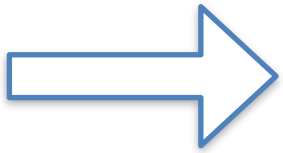
Lemmatization

- 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

Stemming

- Reduce terms to their "roots" before indexing
- "Stemming" suggests crude affix chopping
  - language dependent
  - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

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

- Commonest 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

Language-specificity

- The above methods embody transformations that are
  - Language-specific, and often
  - Application-specific
- These are “plug-in” addenda to the indexing process
- Both open source and commercial plug-ins are available for handling these

Does stemming help?

- Search
  - English: very mixed results. Helps recall for some queries but harms precision on others
    - E.g., operative (dentistry) ⇒ oper
  - Definitely useful for Spanish, German, Finnish, …
    - 30% performance gains for Finnish!
- NLP Tasks
  - Generic “classification tasks” aiming to predict the target by identifying “topic”: yes
  - Machine translation: no!
  - Sentiment analysis: no!

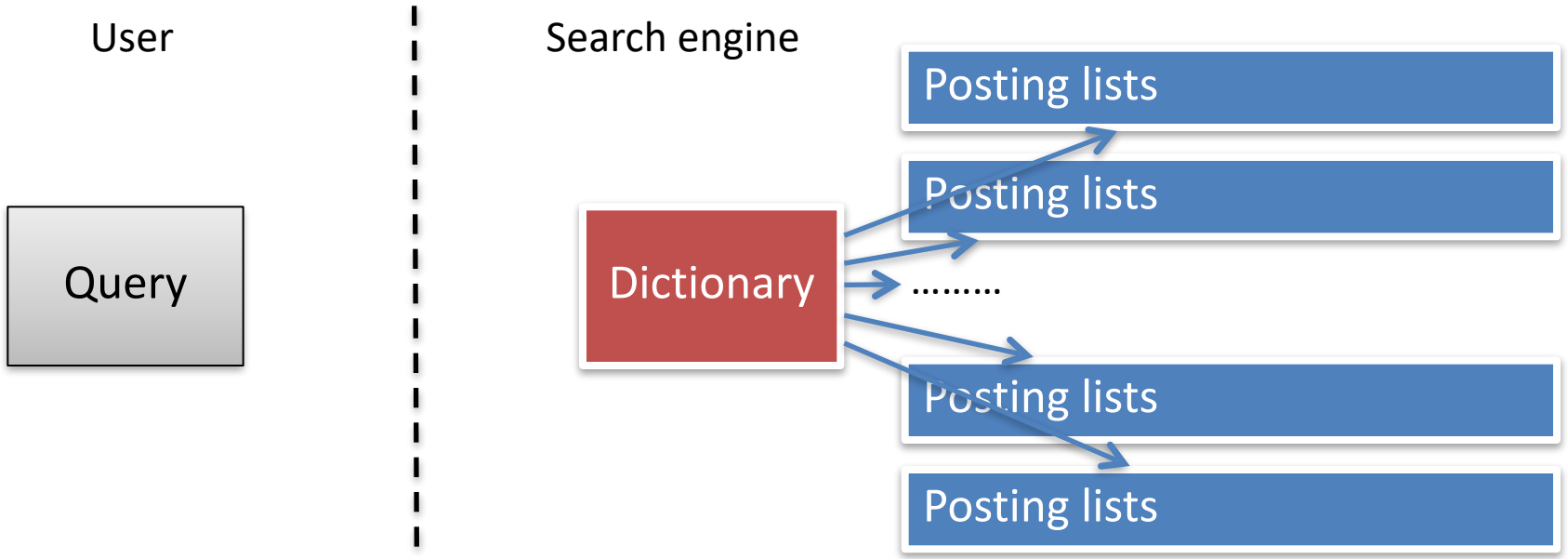
Stemming for sentiment analysis

Porter stemmer (suffix stripping)

Positive sentiment	Negative sentiment	Stem
defense	defensive	defens
affection	affect	affect
objective	objection	object
tolerant	tolerable	toler
extravagance	extravagant	extravag

- Different forms of the same stem may carry different sentiments
- WordNet stemmer does not have this problem, but still it removes comparative morphology, e.g. **happiest, happy → happy**

The dictionary



- User sends the query
- The engine
  - Determine the query terms
  - Determine if each query term is present in the dictionary
  - Dictionary: Lookup
  - Search trees or hashing

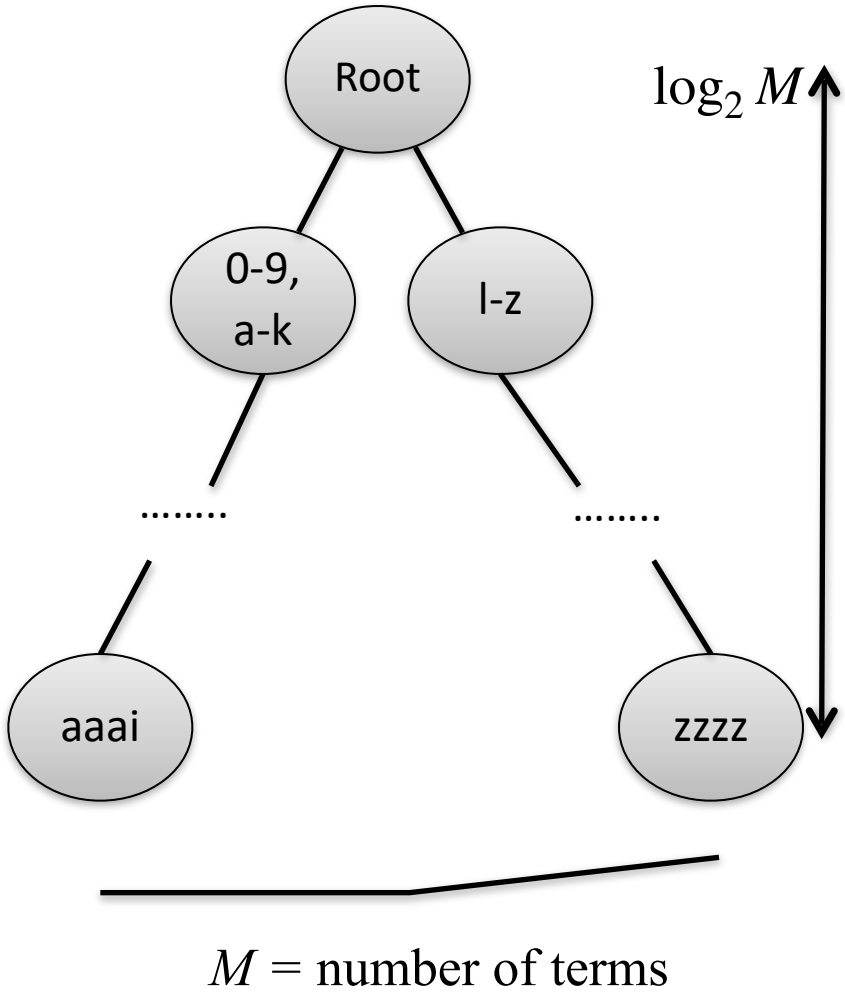
Binary search trees

Binary search tree

- Each node has two children
- O(log M) comparisons if the tree is balanced

Problem

- Balancing the tree when terms are inserted and deleted



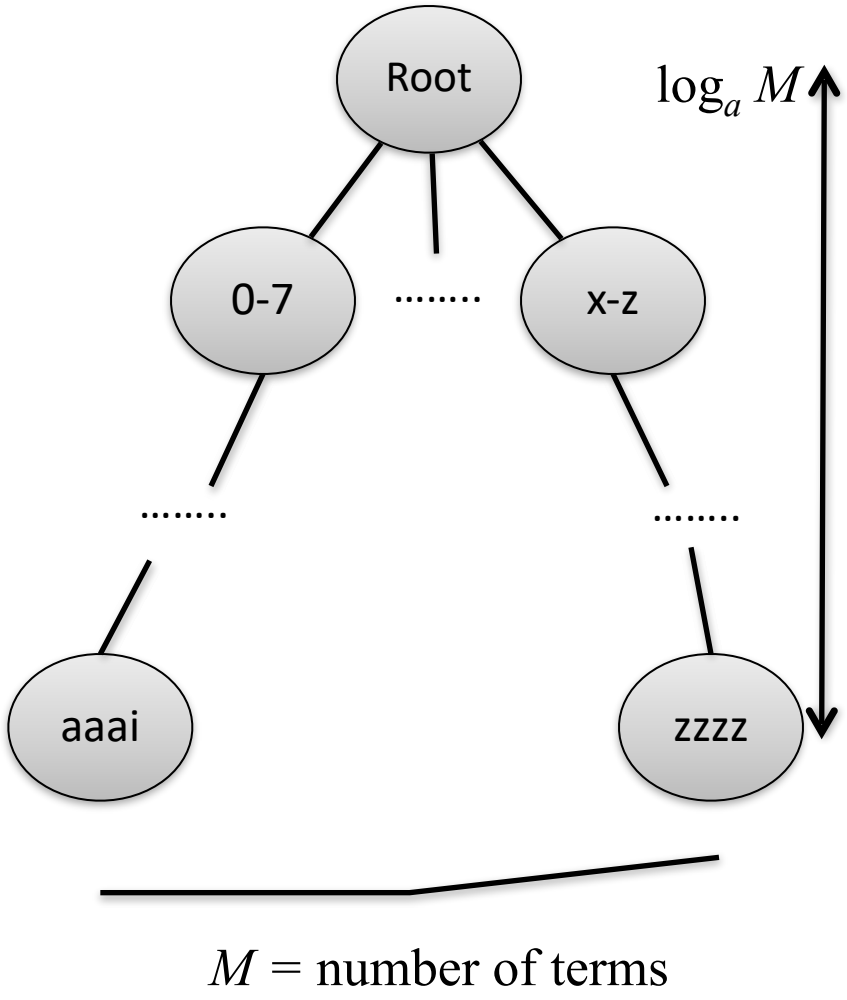
B-tree

B-tree

- Number of children for each node is between a and b for some predetermined a and b
- O($\log_a M$) comparisons
- Very few rebalancing required

B+ tree

- Similar to B-tree
- All data (pointers to posting lists) are in leaf nodes
- Linear scan of data easier



WILDCARD QUERIES

Wildcard queries

- Wildcard: is a character that may be substituted for any of a defined subset of all possible characters
- Wildcard queries: queries with wildcards
  - sydney/sidney: `s*dney`
  - debapriyo/debopriya/debopriyo: deb`*priy*`
  - judicial/judiciary: `judicia*`
- Trailing wildcard queries
  - Simplest: search trees work well
  - Determine the node(s) which correspond to the range of terms specified by the query
  - Retrieve the posting lists for the set *W* of all the terms in the entire sub-tree under those nodes

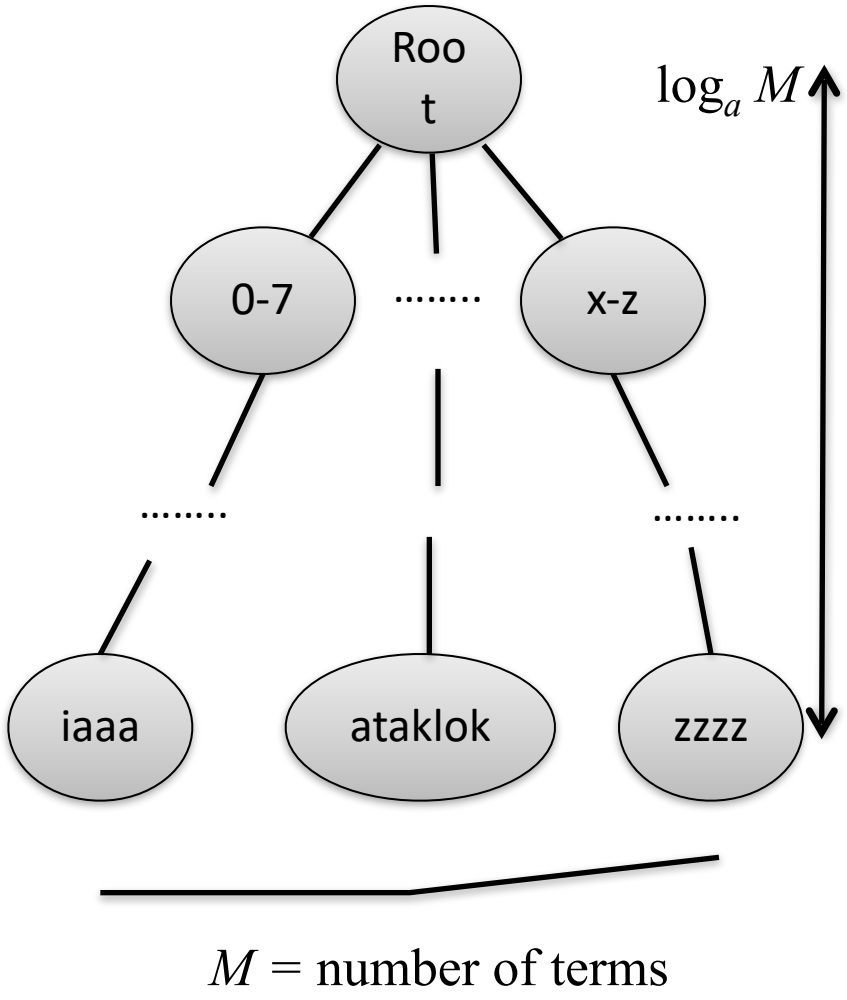
Trailing wildcard query

Queries with a single *

- Leading wildcard queries: `*ata`
  - Matching kolkata, bata, …
- Use a reverse B-tree
  - B-tree obtained by considering terms backwards
  - Consider the leading wildcard query backwards, it becomes a trailing wildcard query
  - Lookup as before for a trailing wildcard query on a normal B-tree

Queries with a single *

- Queries of the form: `s*dney`
  - Matching sydney, sidney, …
- Use a B-tree and a reverse B-tree
  - Use the B-tree to get the set *W* of all terms matching s*
  - Use the reverse B-tree to get the set *R* of all terms matching *dney
  - Intersect *W* and *R*



General wildcard queries

The permuterm index

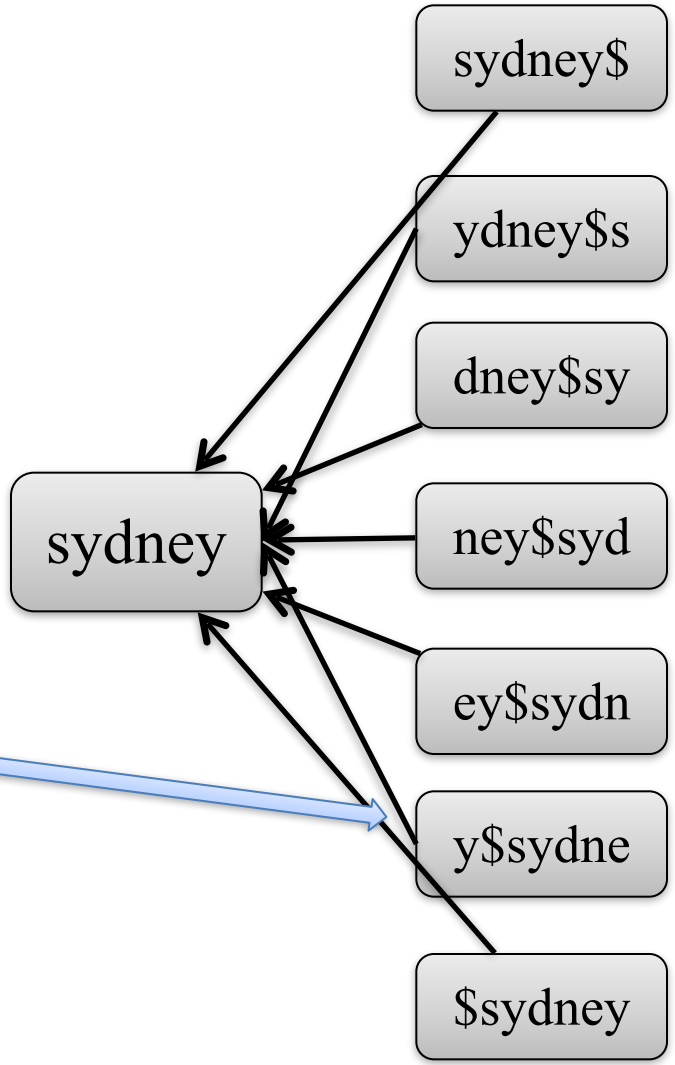
- Special character $ as end of term
- The term sydney → sydney$
- Enumerate all rotations of sydney$ and have all of them in the B-tree, finally pointing to sydney

Wildcard queries

- A single *: sy*ey
  - Query ey$sy* to the B-tree
  - One rotation of sydney$ will be a match
- General: s*dn*y
  - Query y$s* to the B-tree
  - Works equivalent to s*y, not all of the matches would have "dn" in the middle
  - Filter the others by exhaustive checks

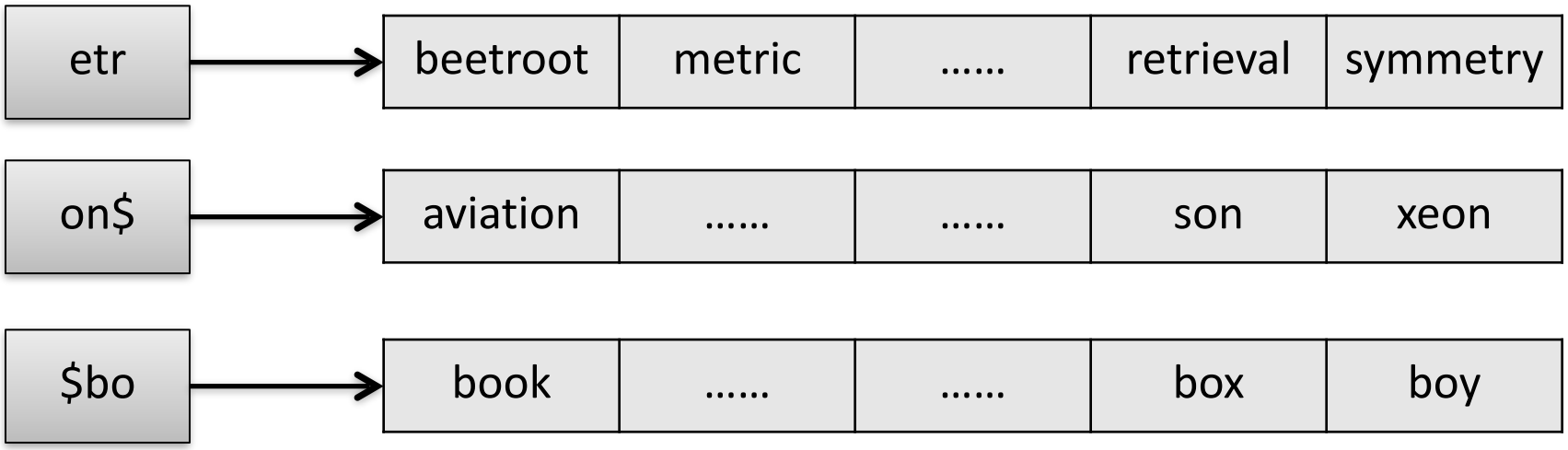
Problems

- Blows up the dictionary
- Empirically seen to be 10 times for English



k-gram index for wildcard queries

- *k*-gram: sequence of *k* characters
- *k*-gram index: <*k*-gram> → words in which the *k*-gram appears, sorted lexicographically
  - Consider all words with the beginning and ending marker $



k is predetermined

Wildcard queries with *k*-gram index

- User query: `re*ve`
  - Send the Boolean query $re AND ve$ to the *k*-gram index
  - Will return terms such as revive, remove, …
  - Proceed with those terms and retrieve from inverted index
- User query: `red*`
  - Send the query $re AND red to the 3-gram index
  - Returns all results starting with "re" and containing "red"
  - Post-processing to keep only the ones matching red*
- Exercise: more general wildcard query `s*dn*y`
  - Can we do this using the *k*-gram index (assume 3-gram)?

Discussion on wildcard queries

- Semantics
  - What does `re*d AND fe*ri` mean?
  - (Any term matching re*d) AND (Any term matching fe*ri)
  - Once the terms are identified, the operation on posting lists
  - (… Union …) Intersection (… Union …)
  - Expensive operations, particularly if there are many matching terms
- Expensive even without Boolean combinations
- Hidden functionality in search engines
  - Otherwise users would "play around" even when not necessary
  - For example, a query "s*" produces huge number of terms for which the union of posting lists need to be computed

Why search trees are better than hashing?

- Possible hash collision
- Prefix queries cannot be performed
  - red and re may be hashed to entirely different range of values
- Almost similar terms do not hash to almost similar integers
- A hash function designed now may not be suitable if the data grows to a much larger size

Did you mean?

SPELLING CORRECTIONS

Misspelled queries

- People type a lot of misspelled queries
  - britian spears, britney's spears, brandy spears, prittany spears → britney spears
- What to do?
  1. Among the possible corrections, choose the "nearest" one
  2. Among the possible "near" corrections, choose the most frequent one (probability of that being the user's intention is the highest)
  3. Context sensitive correction
  4. The query may not be actually incorrect. Retrieve results for the original as well as possible correction of the query
     - debapriyo majumder → returns results for debapriyo majumdar and majumder both
- Approaches for spelling correction
  - Edit distance
  - *k*-gram overlap

Edit distance

- Edit distance E(A,B) = minimum number of operations required to obtain B from A
  - Operations allowed: insertion, deletion, substitution
- Example: E(food, money) = 4
  - food → mood → mond → moned → money
- Computing edit distance in O(|A| . |B|) time
- Spelling correction
  - Given a (possibly misspelled) query term, need to find other terms (in the dictionary) with very small edit distance
  - Precomputing edit distance for all pairs of terms → absurd
  - Use several heuristics to limit possible pairs
    - Only consider pairs of terms starting with same letter

Computing edit distance

Observation:

- E(food, money) = 4
  - One sequence: food → mood → mond → moned → money
- E(food, moned) = 3
- Why?
  - If E(food, moned) < 3, then E(food, money) < 4

Prefix property: If we remove the last step of an optimal edit sequence then the remaining steps represent an optimal edit sequence for the remaining substrings

Computing edit distance

- Fix the strings A and B. Let $|A| = m$, $|B| = n$.
- Define: $E(i, j) = E(A[1, \ldots, i], B[1, \ldots, j])$
  - That is, edit distance between the length i prefix of A and length j prefix of B
- Note: $E(m, n) = E(A,B)$
- Recursive formulation

  (a) $E(i, 0) = i$

  (b) $E(0, j) = j$
- The last step: 4 possibilities
  - Insertion: $E(i, j) = E(i, j - 1) + 1$
  - Deletion: $E(i, j) = E(i - 1, j) + 1$
  - Substitution: $E(i, j) = E(i - 1, j - 1) + 1$
  - No action: $E(i, j) = E(i - 1, j - 1)$

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$

$$E(0,j) = j$$

$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable

P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0						
1	M					
2	O					
3	N					
4	E					
5	Y					

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$

$$E(0,j) = j$$

$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable

P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1				
2	O	2				
3	N	3				
4	E	4				
5	Y	5				

Backtrace:

Compute E(*i*, *j*) and also keep track of where E(*i*, *j*) came from

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$

$$E(0,j) = j$$

$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable

P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1	1			
2	O	2				
3	N	3				
4	E	4				
5	Y	5				

Backtrace:

Compute E(*i*, *j*) and also keep track of where E(*i*, *j*) came from

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$

$$E(0,j) = j$$

$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable

P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1	1			
2	O	2	2			
3	N	3				
4	E	4				
5	Y	5				

Backtrace:

Compute E(*i*, *j*) and also keep track of where E(*i*, *j*) came from

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$

$$E(0,j) = j$$

$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable

P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1	1	2	3	4
2	O	2	2			
3	N	3	3			
4	E	4	4			
5	Y	5	5			

Backtrace:

Compute E(i, j) and also keep track of where E(i, j) came from

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$

$$E(0,j) = j$$

$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable

P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1	1	2	3	4
2	O	2	2	1		
3	N	3	3			
4	E	4	4			
5	Y	5	5			

Backtrace:

Compute E(*i*, *j*) and also keep track of where E(*i*, *j*) came from

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$

$$E(0,j) = j$$

$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable

P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1	1	2	3	4
2	O	2	2	1	2	3
3	N	3	3	2		
4	E	4	4	3		
5	Y	5	5	4		

Backtrace:

Compute E(i, j) and also keep track of where E(i, j) came from

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$

$$E(0,j) = j$$

$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable

P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1	1	2	3	4
2	O	2	2	1	2	3
3	N	3	3	2	2	3
4	E	4	4	3	3	3
5	Y	5	5	4	4	4

Backtrace:

Compute E(*i*, *j*) and also keep track of where E(*i*, *j*) came from

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$

$$E(0,j) = j$$

$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable

P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1	1	2	3	4
2	O	2	2	1	2	3
3	N	3	3	2	2	3
4	E	4	4	3	3	3
5	Y	5	5	4	4	4

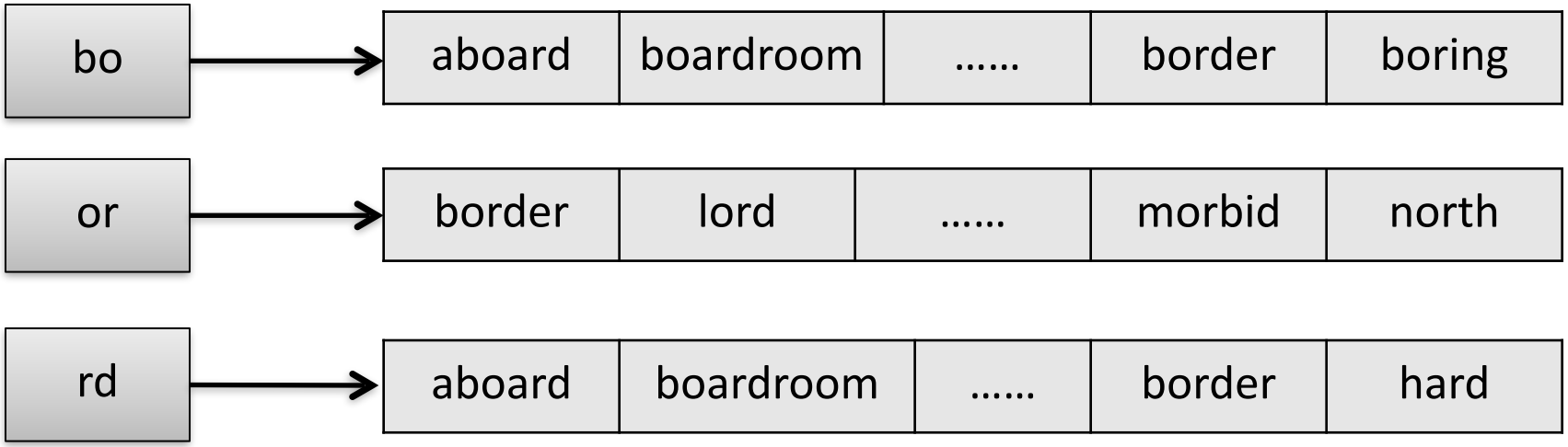
Backtrace to find an optimal edit path

Backtrace:

Compute E(*i*, *j*) and also keep track of where E(*i*, *j*) came from

Spelling correction using *k*-gram index

- The *k*-grams are *small* portions of words
- Misspelled word would still have some *k*-grams intact
- Misspelled query: `bord`



- Intersect the list of words for *k*-grams
- Problem: long words which contain the *k*-grams but are not good corrections

Phonetic correction

- Some users misspell because they don't know the spelling
- Types as it "sounds"
- Approach for correction: use a phonetic hash function
  - Hash similarly sounding terms to the same hash value
- Soundex algorithm
  - Several variants

Soundex algorithm

1. Retain the first letter of the term
2. Change all

   A, E, I, O, U, H, W, Y → 0

   B, F, P, V → 1.

   C, G, J, K, Q, S, X, Z → 2.

   D,T to 3.

   L to 4.

   M, N to 5.

   R to 6.
3. Repeat: remove one of each pair of same digit
4. Remove all 0s. Pad the result with trailing 0s. Return the first 4 positions: one letter, 3 digits

Example: Hermann → H065055 → H06505 → H655

References and acknowledgements

- Primarily: IR Book by Manning, Raghavan and Schuetze: http://nlp.stanford.edu/IR-book/
- The part on edit distance: lectures notes by John Reif, Duke University:https://www.cs.duke.edu/courses/fall08/cps230/Lectures/L-04.pdf