BITS Pilani
K K Birla Goa Campus

innovate achieve lead

M3. Dictionaries and Tolerant Retrieval

Information Retrieval

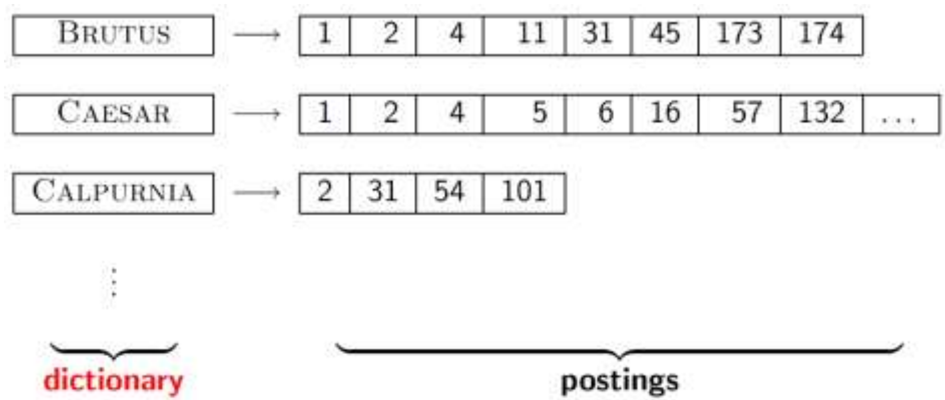
Outline

- Dictionary data structures
- “Tolerant” retrieval
  - Wild-card queries
  - Soundex
  - Spelling correction

1. Dictionary Data Structure

- **Vocabulary:** set of terms that are stored in the dictionary
- **Dictionary:** actual implementation of vocabulary
- The dictionary data structure stores:
    - term vocabulary
    - document frequency
    - pointers to each postings list

Key Question: in what data structure?



A naïve approach

- An array of struct:

An array of lexicographically sorted terms, integer document frequency, and pointer to posting list

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

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

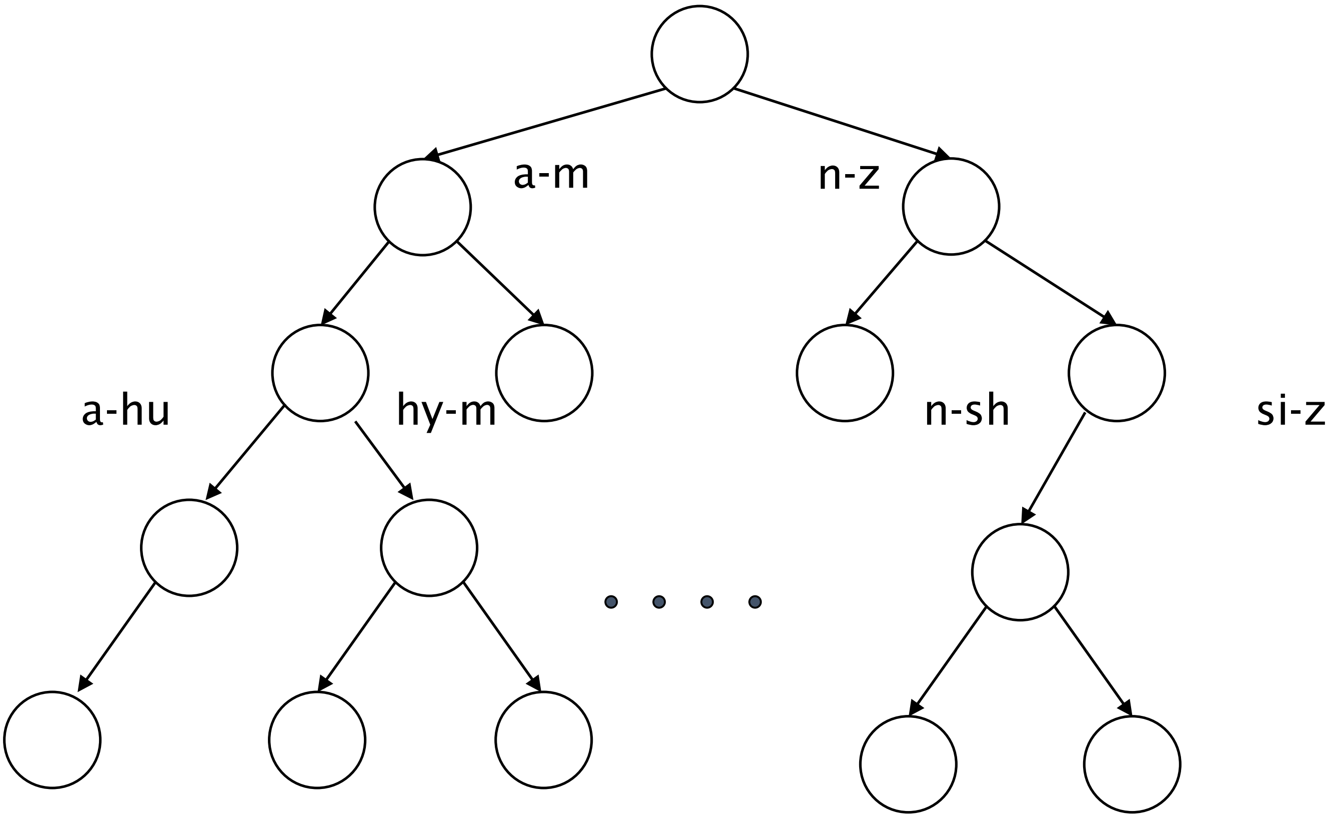
Dictionary Data Structure

- Two main choices:
  - Hashtables
  - Search Trees
- Both techniques are in use.

1.1 Hashtables

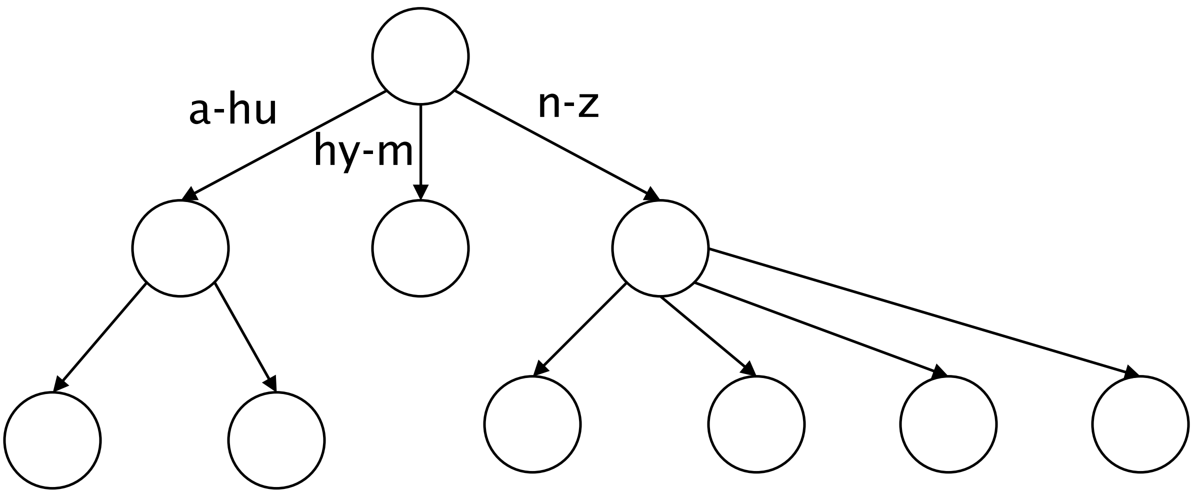
- Each vocabulary term is hashed to an integer
- Pros:
  - Lookup is faster than for a tree
  - O(1)
- Cons:
  - No easy way to find minor variants:
    - judgment/judgement
  - No prefix search [tolerant retrieval]
  - If vocabulary keeps growing, need to occasionally do the expensive operation of rehashing *everything*

1.2 Trees: BST

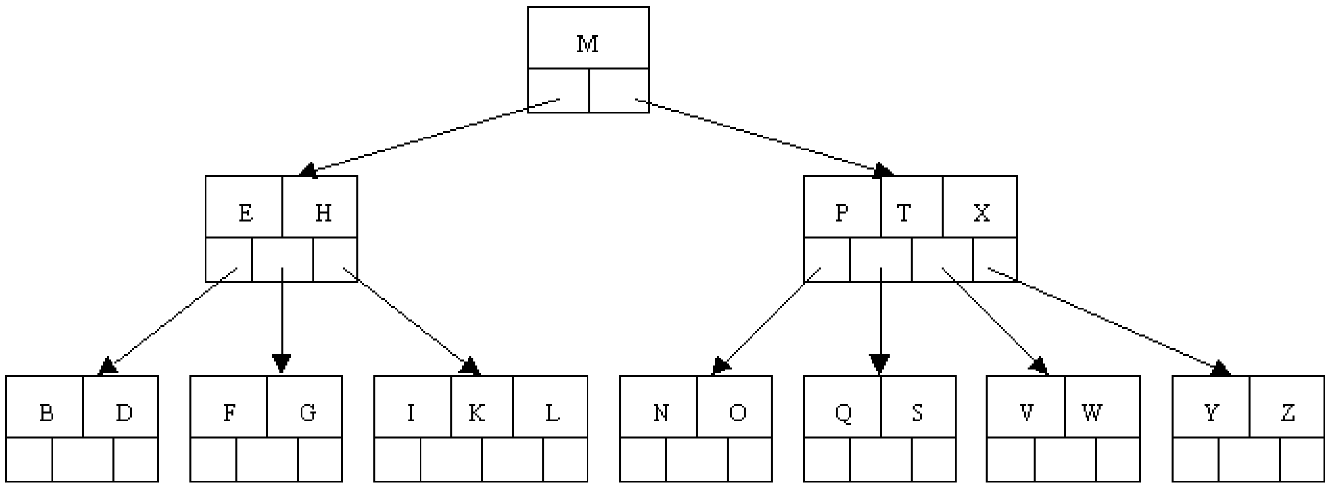


1.2 Trees: B-Tree

- 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].
- Need of balanced tree than just Binary tree



Example



Trees

- Simplest: binary tree
- More usual: B-trees
- Pros:
  - Solves the prefix problem (terms starting with *hyp*)
- Cons:
  - Slower: O(log M) [and this requires *balanced* tree]
  - Rebalancing binary trees is expensive
    - But B-trees mitigate the rebalancing problem

2. Wildcard Queries

Wildcard query

- Returns documents that contain terms matching a wildcard pattern.
- A wildcard operator is a placeholder that matches one or more characters.
- For example, the ***** wildcard operator matches zero or more characters.
- You can combine wildcard operators with other characters to create a wildcard pattern.
- Example: ***mon*:*** find all docs containing any word beginning with "mon".
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range: ***mon ≤ w < moo***
- ****mon:*** find words ending in "mon": harder
  - Maintain an additional B-tree for terms *backwards.*

  Can retrieve all words in range: ***nom ≤ w < non.***

Query Processing

- At this point, we have an enumeration of all terms in the dictionary that match the wild-card query.
- We still have to look up the postings for each enumerated term.
- E.g., consider the query:

  se*ate *AND* ***fil*er***

  This may result in the execution of many Boolean *AND* queries.

Wildcard operator in the middle of the query term

- How can we handle * in the middle of query term?
- co*tion
- We could look up co* AND *tion in a B-tree and intersect the two term sets
- Expensive
- The solution: transform wild-card queries so that the * occur at the end
- This gives rise to the Permuterm Index and k-gram index.

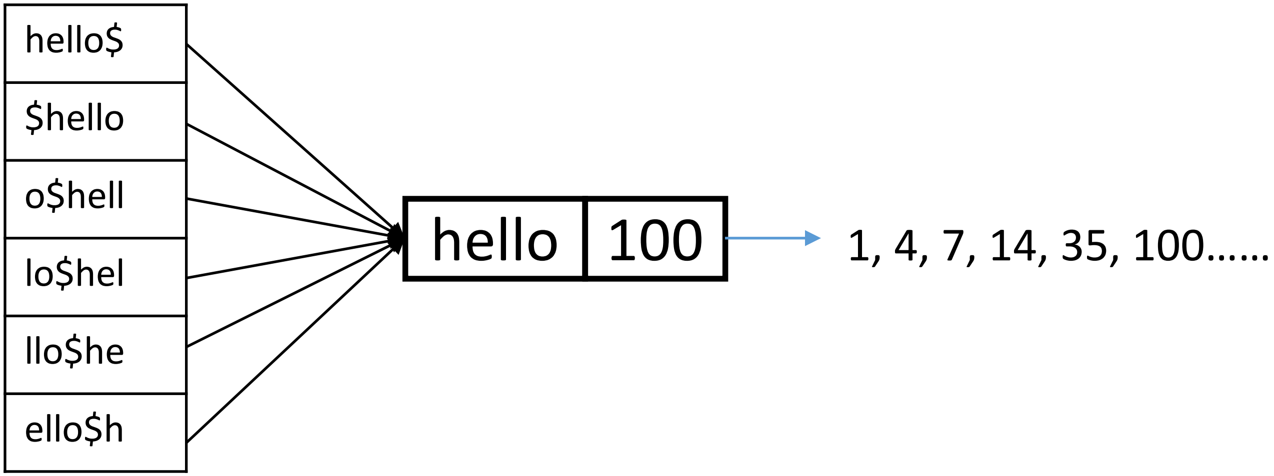
2.1 Permuterm Index

- Every term that goes into standard inverted index
- Create rotations (permutations) of that term
- For term ***hello***, index under:
  - ***hello$, ello$h, llohe, lohel, o$hell, $hello***

  where $ is a special symbol- represents the end of the term
- Queries:
  - **X** lookup on **X$**
  - ***X** lookup on **X$***
  - **X*Y** lookup on **Y$X***

X* lookup on $**X***
X lookup on **X***
X*Y*Z ??? Exercise!

Permuterm Index



Examples:
Simple: hel*
Complex: tr*di*on

Multiple * Symbols (X*Y*Z)

- In this case we first enumerate the terms in the dictionary that are in the permuterm index of Z$X*.
- Not all such dictionary terms will have the string Y in the middle
  - we filter these out by exhaustive enumeration, checking each candidate to see if it contains Y.
  - For a query: tr*di*on, the term "tradition" would survive this filtering but "transportation" or "transformation" would not.
  - We then run the surviving terms through the standard inverted index for document retrieval.
- One disadvantage of the permuterm index is that its dictionary becomes quite large, including as it does all rotations of each term.

2.2 K-gram index

- Enumerate all *k*-grams (sequence of *k* chars) occurring in any term
- *e.g.,* from text "*Information Retrieval*" we get the following 2-grams (*bigrams*):

$I, In, nf, fo, or, rm, ma, at, ti, io, on, n$, $R, Re, et, tr, ri, ie, ev, va, al, l$

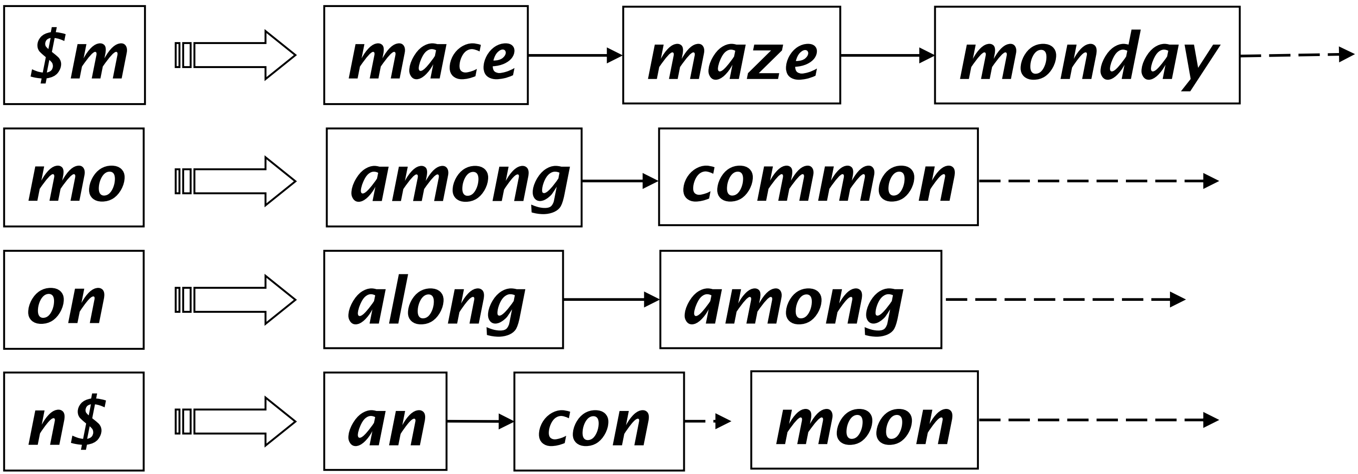
  - $ is a special word boundary symbol

K-gram index

- **Dictionary**: all possible k-grams generated from the terms (terms present in the corpus or the std. inverted index dictionary)
- **Posting lists**: all the terms containing that bigram
  - Always sorted in lexicographical order, similar to posting lists of inverted index: sorted in the ascending order of Doc IDs
- Example: for bi-gram **"tr"**, posting list will have the terms like
- Re**tr**ieval, **tr**ansform, cons**tr**uct, abs**tr**act, con**tr**ast.....

Example

- The *k*-gram index finds *terms* based on a query consisting of *k*-grams (here *k*=2).



Procedure

- Transform the query into k-grams (here k=2)
- Find the terms consisting of those k-grams from k-gram index
- Do an AND of all the terms found in k-gram index
- Surviving enumerated terms are then looked up in the term-document inverted index.
- Possibility of false alarms?
  - Reverse of the AND query to the original wild query might not be true.
- complex than standard inverted index but efficient than permuterm

Examples

- Query ***mon**** can now be run as
  - ***$m** AND **mo** AND **on***
    - *All the terms starting with **m** AND containing **mo** AND containing **on***
  - However, all the terms after boolean AND don't have the same wildcard query pattern **mon***

- Query **apr***
  - **$a** AND **ap** AND **pr**
  - Terms like **appreciate** doesn't have wildcard query pattern **apr***

Processing wildcard queries

- Wildcard queries are expensive
- Computational IR systems need to a lot more work than expected
- Most of the search engines hide wildcard query facility
  - Early search engines- “type wildcard queries only if needed”
  - Nowadays, hidden in form of advanced search
  - Also a constrained form of wildcard query.

3. Soundex

Soundex

- Soundex algorithm allows you to find all the documents that contain terms phonetically matching the query
- Class of heuristics to expand a query into phonetic equivalents
  - Language specific – mainly for proper nouns (names)
  - E.g., ***chebyshev*** $\rightarrow$ ***tchebycheff***
- Invented for the U.S. census … in 1918
  - Proposed as a way to keep a track of names of immigrants
    - Same name is spelled differently in Russian or English

Soundex

- Used in linguistic model phase of inverted index construction
- Maps all terms that sound similar to the same equivalence class
- Turn every token to be indexed into a 4-character reduced form
- Do the same with query terms
- Build and search an index on the reduced forms
  - (when the query calls for a soundex match)
- http://www.creativyst.com/Doc/Articles/SoundEx1/SoundEx1.htm#Top

Typical Algorithms

1. Retain the first letter of the word.
2. Change all occurrences of the following letters to '0' (zero):
   - A', E', 'I', 'O', 'U', 'H', 'W', 'Y'. → **Consonants that sound like vowel**
3. Change letters to digits as follows:
   - B, F, P, V → 1
   - C, G, J, K, Q, S, X, Z → 2
   - D,T → 3
   - L → 4
   - M, N → 5
   - R → 6

   Consonants that sound similar

Examples:
Robert and Rupert
Beijing and Peking
Smith and Smythe

4. Remove all pairs of consecutive digits.
5. Remove all 0s from the resulting string.
6. Pad the resulting string with trailing 0s and return the first four positions

Soundex

- Soundex is the classic algorithm, provided by not only IR systems but also by most databases (Oracle, Microsoft, …)
- How useful is soundex?
  - Not very – for information retrieval (poor precision)
  - Okay for “high recall” tasks (e.g., Interpol), though biased to names of certain nationalities
    - E.g., different variants of names
- Zobel and Dart (1996) show that other algorithms for phonetic matching perform much better in the context of IR

4. Spelling Correction

Spell Correction

- Two principal uses
  - Correcting user queries to retrieve “right” answers
    - Queries that are misspelled
    - You might not get correct results if the queries are spelled wrongly
  - Correcting document(s) being indexed
    - If the documents contain spelling mistakes, then queries on the right spelling might not retrieve those documents with error
- ^ Basic motivation of correcting spell errors in both query as well as the documents

Two main flavors

- Isolated word
  - Check each word on its own for misspelling
  - Individual check without keeping in mind what is the context of the query
  - Will not catch typos resulting in correctly spelled words
  - e.g., from → form
- Context-sensitive
  - Look at surrounding words
  - Spelling correction is not sufficient to check but the words should also match the correct grammatical context
  - e.g., I flew <u>form</u> Delhi to Goa.
  - ^able to detect in context-sensitive method while in isolated method "form" is a meaningful word

Query misspellings

- Our principal focus here
  - E.g., the query Chrstopher Maning instead of Christopher Manning
  - [click]()
  - Before you build your spell corrector, decide which type of interface you want to provide to the user
  - We can either
    - Retrieve documents indexed by the correct spelling, OR
    - Return several suggested alternative queries with the correct spelling
      - *Did you mean … ?*

4.1 Isolated word correction

1. Fundamental premise – there is a lexicon from which the correct spellings come

- Two basic choices for this
  - A standard lexicon
    - Oxford or Webster's English Dictionary
    - An "industry-specific" lexicon – hand-maintained (reference dictionary)
      - Healthcare, Chemistry, Law, Shopping
  - The lexicon of the indexed corpus
    - E.g., all words on the web
    - All names, acronyms etc. (Including the misspellings)
    - How many times a word appears in the corpus (typos will have less collection frequency **relatively rare**)

Isolated word correction

2. There is a way to calculate the distance between any two words

- Given a lexicon and a character sequence Q, return the words in the lexicon closest to Q
- What's "closest"? **(based on the distance)**
- We'll study several alternatives
  - Edit distance (Levenshtein distance)
  - Weighted edit distance
  - *n*-gram overlap

4.1.1 Edit Distance

- **Definition:** Given two strings S_1 and S_2, the minimum number of operations to convert one to the other.
- Operations are typically character-level
  - **Insert, Delete, Replace**.
- There can be multiple ways to convert S1 to S2
  - Delete all characters of S1 and insert all characters of S2 one by one.
  - Correct approach? (minimum #operations)
  - From ***cat*** to ***act*** is 2
  - from ***cat*** to ***dog*** is 3.

CAT and ACT have an edit distance of 1 with transpose

- Generally found by dynamic programming (Insert, Delete, Replace)
  - Some variations have **Transposition (swapping of two adjacent characters)**
  - QWERTY typos

Dynamic Programming Algorithm

LevenshteinDistance(s_1, s_2)

for $i \leftarrow 0$ **to** $|s_1|$
do $m[i, 0] = i$
for $j \leftarrow 0$ **to** $|s_2|$
do $m[0, j] = j$
for $i \leftarrow 1$ **to** $|s_1|$
do for $j \leftarrow 1$ **to** $|s_2|$
  do if $s_1[i] = s_2[j]$
    then $m[i, j] = \min\{m[i\text{-}1, j]+1, m[i, j\text{-}1]+1, m[i\text{-}1, j\text{-}1]\}$
    else $m[i, j] = \min\{m[i\text{-}1, j]+1, m[i, j\text{-}1]+1, m[i\text{-}1, j\text{-}1]+1\}$
return $m[|s_1|, |s_2|]$

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Example

- S1= Hello, S2= Yellow

	Φ	Y	E	L	L	O	W
Φ	0	1	2	3	4	5	6
H	1	1	2	3	4	5	6
E	2	2	1	2	3	4	5
L	3	3	2	1	2	3	4
L	4	4	3	2	1	2	3
O	5	5	4	3	2	1	2

4.1.2 Weighted Edit Distance

- As above, but the weight depends on
  - Which operation it is
  - the character(s) involved
    - Meant to capture OCR or keyboard errors
      - Example: ***m*** more likely to be mis-typed as ***n*** than as ***q***
    - Therefore, replacing ***m*** by ***n*** is a smaller edit distance than by ***q***
    - This may be formulated as a probability model
  - Requires weight (or cost) matrix as input
  - Modify dynamic programming to handle weights
    - What changes???

4.1.3 N-gram Overlap

- Enumerate all the *n*-grams in the query string as well as in the lexicon
- Use the *n*-gram index (recall wild-card search) to retrieve all lexicon terms matching any of the query *n*-grams
- Threshold by number of matching *n*-grams
  - Variants – weight by keyboard layout, etc.

Example with Trigram

- Suppose the text is ***november***
  - Trigrams are *nov, ove, vem, emb, mbe, ber*.
- The query is ***december***
  - Trigrams are *dec, ece, cem, emb, mbe, ber*.
- So 3 trigrams overlap (of 6 in each term)
- How can we turn this into a normalized measure of overlap?

One Option- Jaccard Coefficient

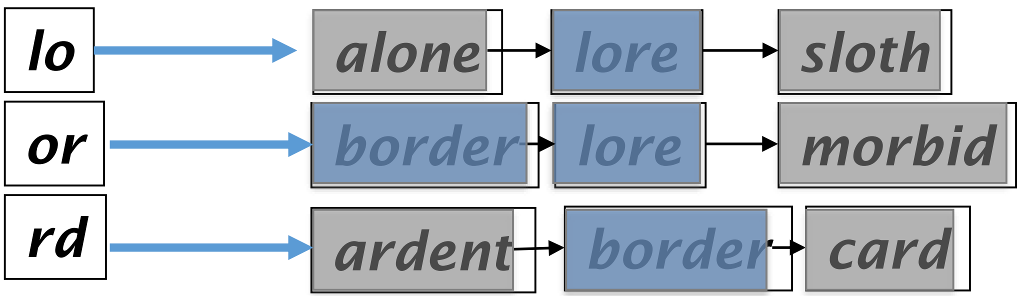
- A commonly-used measure of overlap
- Let *X* and *Y* be two sets; then the J.C. is

$$\frac{|X \cap Y|}{|X \cup Y|}$$

- Equals 1 when *X* and *Y* have the same elements and zero when they are disjoint
- *X* and *Y* don't have to be of the same size
- Always assigns a number between 0 and 1
  - Now threshold to decide if you have a match
  - E.g., if J.C. > 0.8, declare a match

Matching Trigrams

- Consider the query ***lord*** – we wish to identify words matching 2 of its 3 bigrams (***lo, or, rd***)



Standard postings "merge" will enumerate …

Adapt this to using Jaccard (or another) measure.

4.2 Context Sensitive Spell Correction

- Every word is correct in isolated word form but not in the context
- Google, MS Office (after 2007, one of their new features)

- Text: ***I flew from Heathrow to Narita.***
- Consider the phrase query ***"flew form Heathrow"***
- We'd like to respond

Did you mean "***flew from Heathrow***"?

because no (or very few) documents matched the query phrase.

Context Sensitive Spell Correction

- Need surrounding context to catch this.
- First idea: retrieve dictionary terms close (in weighted edit distance) to each query term
- Now try all possible resulting phrases with one word "fixed" at a time
  - ***flew from heathrow***
  - ***fled form heathrow***
  - ***flea form heathrow***

Context Sensitive Spell Correction

- **Hit-based spelling correction:** Suggest the alternative that has lots of hits.
  - Collection frequency in document corpus
  - Query log (more suitable approach)
  - Why?
- Variants:
  - Any word can have error
  - Assume only one word has to be corrected
- Empirical Analysis for spell correction
  - Depends on the accuracy of assumption

What queries can we process?

- We have
  - Positional inverted index
  - Wildcard index (n-gram, reverse b-tree, permuterm)
  - Spell correction
  - Soundex
- Queries such as
  - (SPELL(moriset) /3 toron*to) OR SOUNDEX (chaikofski)