

CSE 4/535

Information Retrieval

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



Department of CSE

Before we start

1. Project 1 released, check website or Brightspace
2. Project 1 due 27th September
3. Remind: 5:50 PM
4. Last 20 mins of the class: Project discussion



Recap - Previous Class

1. Tokenization, Stemming, Lemmatization
2. Tokens vs Terms
3. When skip pointers help?
4. Data structure used for storing terms: 3 types
5. Wild card queries





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

- How can we handle *'s in the middle of query term?





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

- How can we handle *'s in the middle of query term?
 - Maintain forward and backward B-Tree
 - Use Intersection
 - Expensive!?



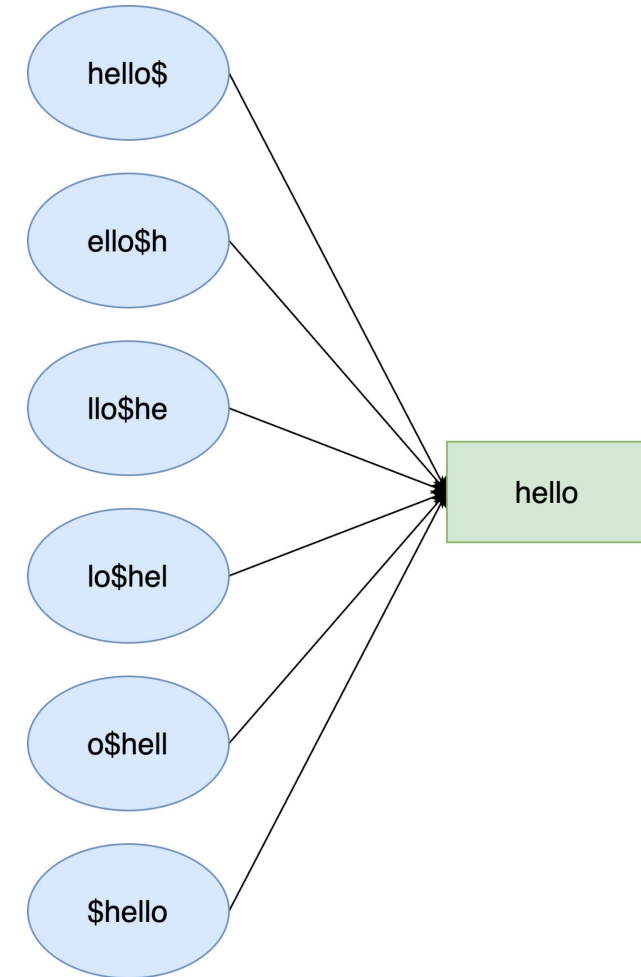
Issues: B-trees handle *'s at the end of a query term

- How can we handle *'s in the middle of query term?
 - Maintain forward and backward B-Tree
 - Use Intersection
 - Expensive!?
- The solution: transform every wild-card query so that the *'s occur at the end
- This gives rise to the Permuterm Index.



Permuterm index

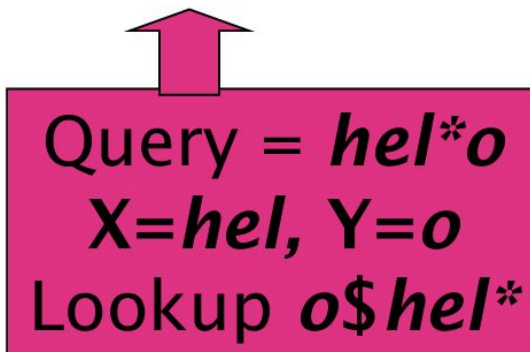
- Create rotations of the term
- For term “hello” index under:
 - hello\$
 - ello\$h
 - llo\$he
 - lo\$hel
 - o\$hell
 - \$hello
- where \$ is a special symbol.





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.



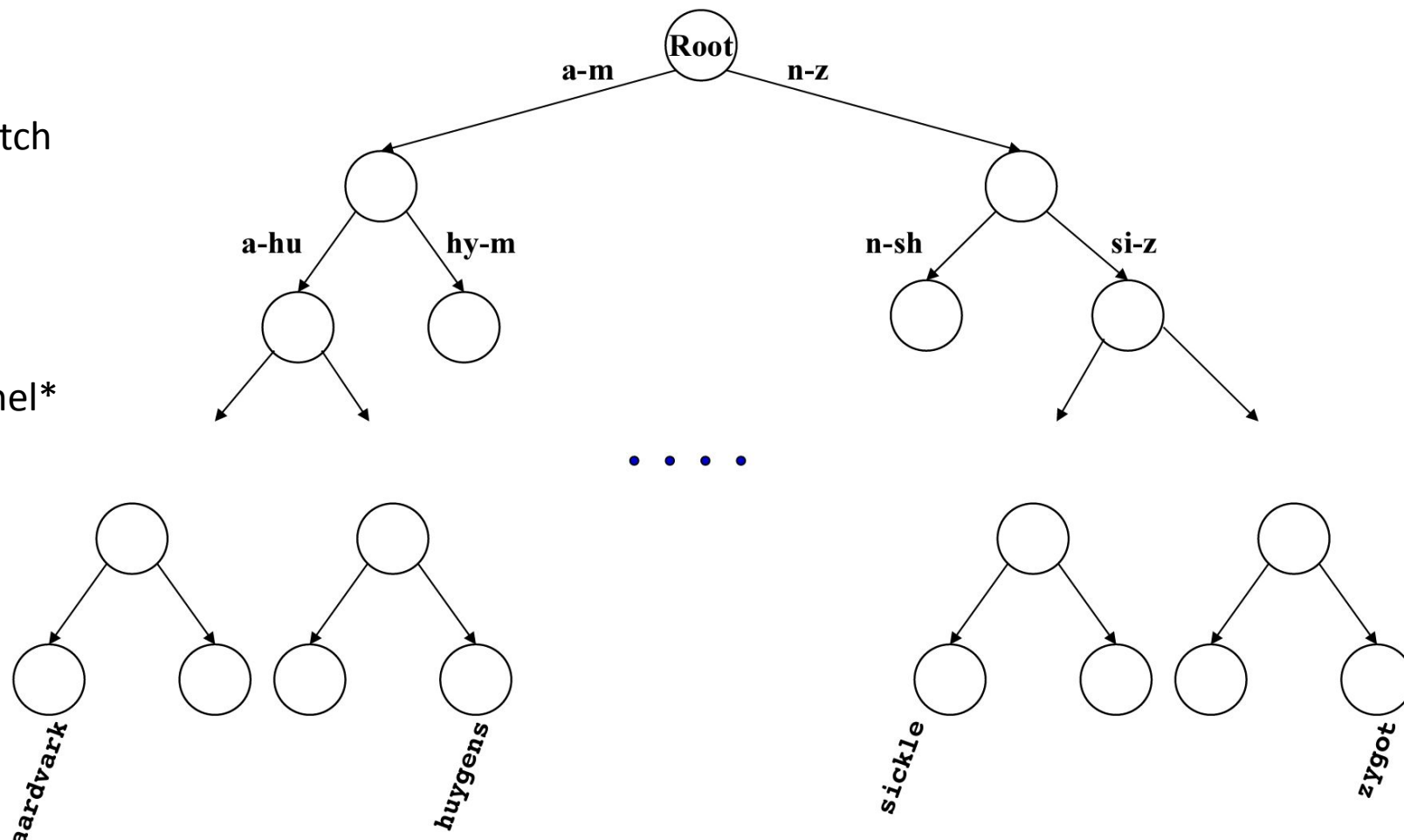
Confused?.... Let us break it down

- Think about searching: `hel*o`
- There are 3 terms which should match
 - `hello`
 - `helloo`
 - `hellloo`
- Use permuterm and search for: `o$hel*`



Confused?.... Let us break it down

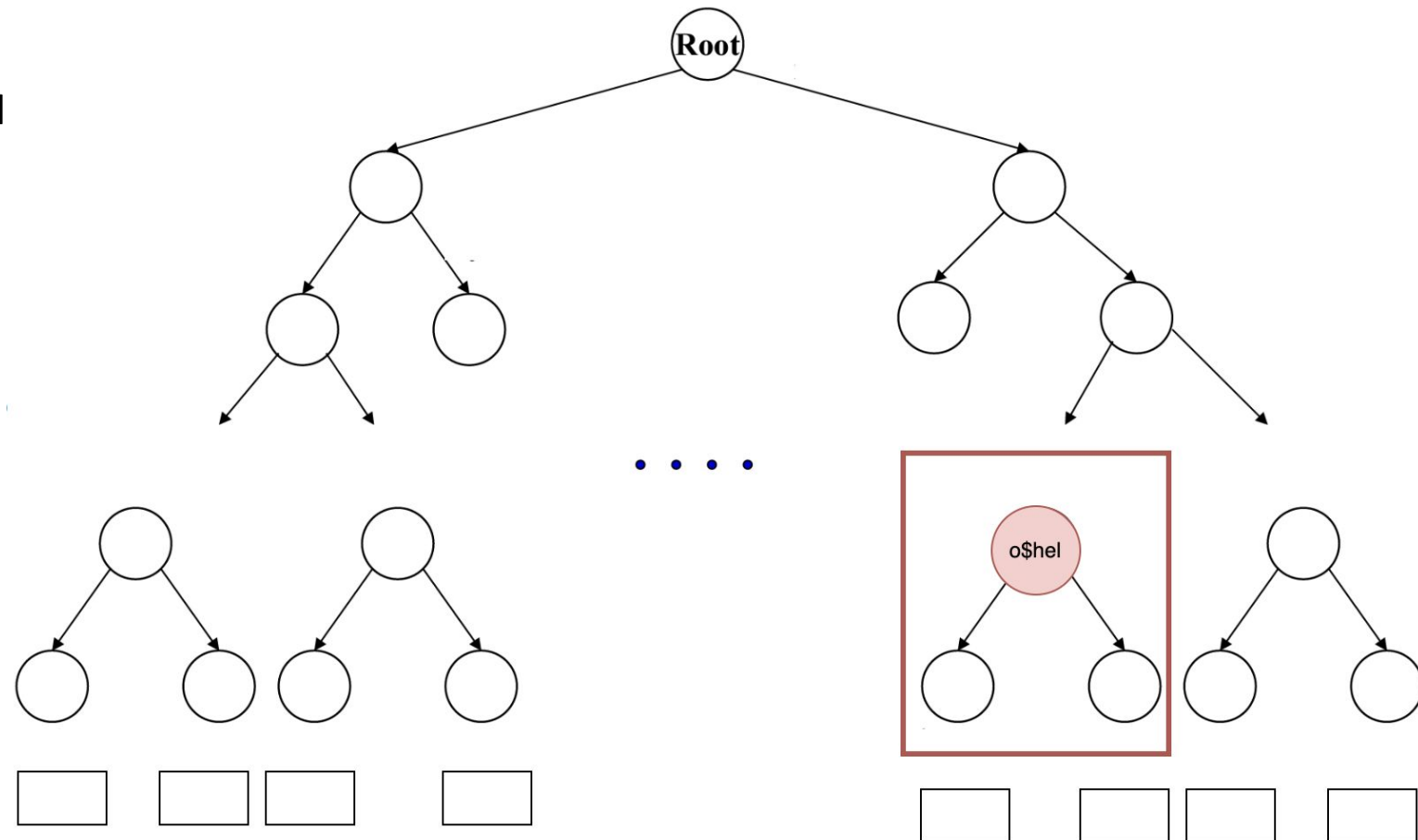
- Think about searching: hel*o
- There are 3 terms which should match
 - hello
 - hello
 - hellllo
- Use permuterm and search for: o\$hel*
- Without permuterm index
- Leaves were the terms





Confused?.... Let us break it down

- Think about searching: hel*o
- Assume, there are 3 terms which should
 - hello
 - hello
 - hellllo
- Use permuterm and search for: o\$hel*
- With permuterm index
- Leaves are the terms
 - includes all rotations





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)

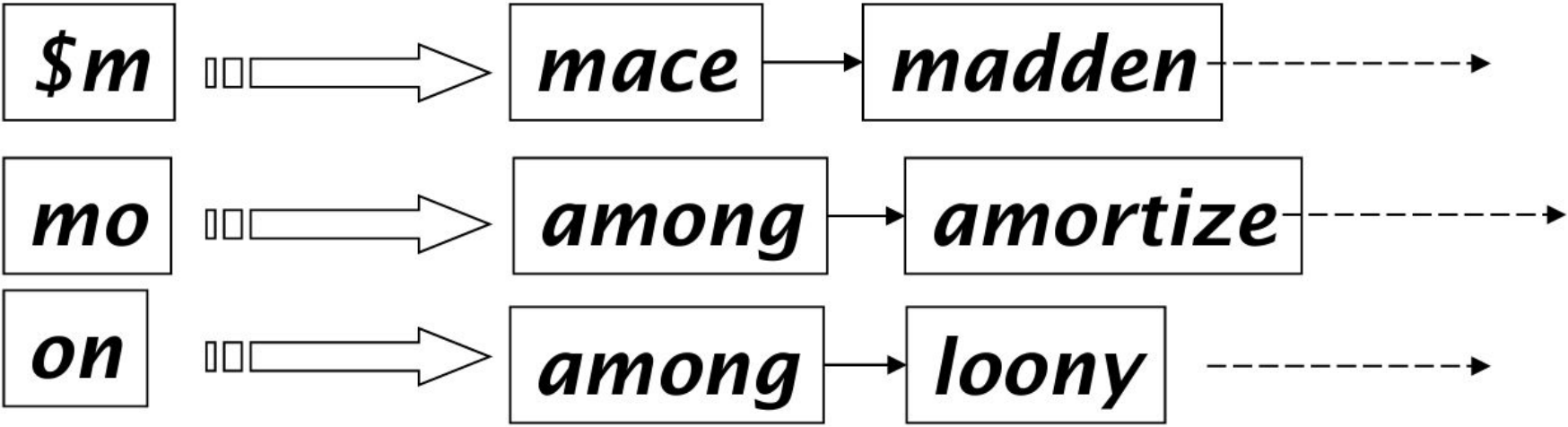
\$a,ap,pr,ri,il,l\$, \$i,is,s\$, \$t,th,he,e\$, \$c,cr,ru,
ue,el,le,es,st,t\$, \$m,mo,on,nt,h\$

- \$ 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. (Ex: check using **in** operator `mon in moon`)
- Surviving enumerated terms are then looked up in the term-document inverted index.



Processing wild-card queries

- It is expensive, thus hidden

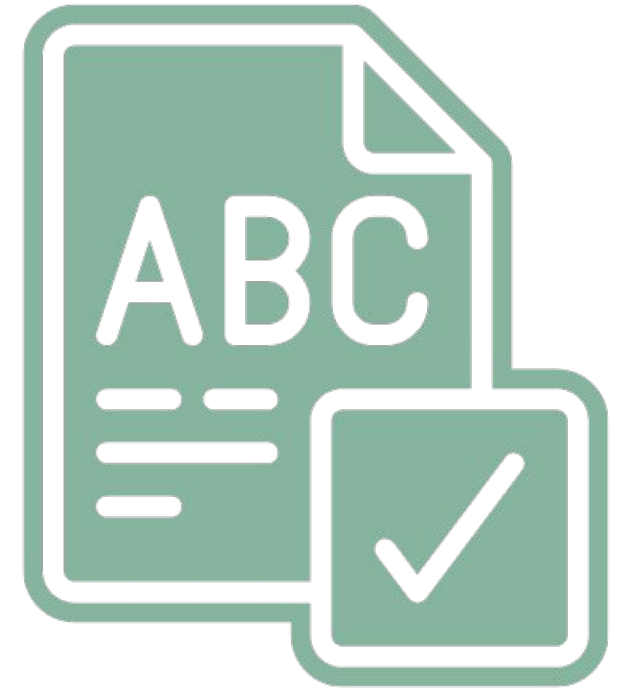
Search

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



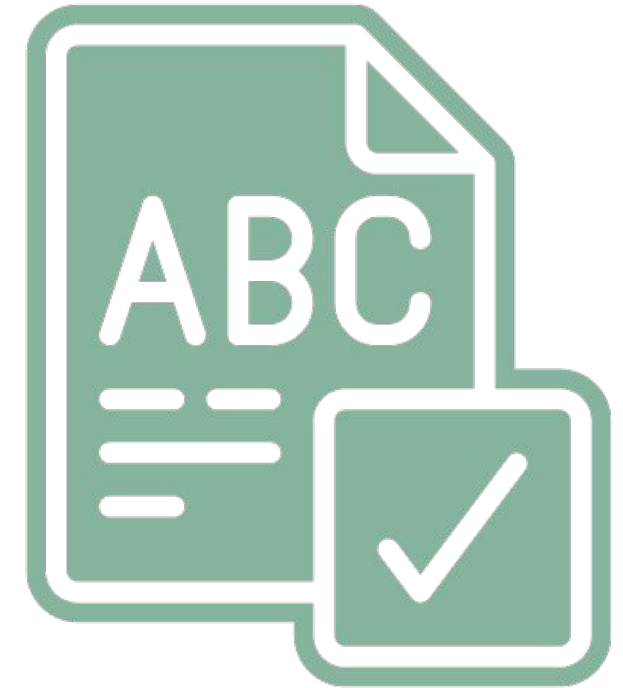
Spell correction

- Two principal uses
 - Correcting document(s) being indexed
 - Retrieve matching documents when query contains a spelling error



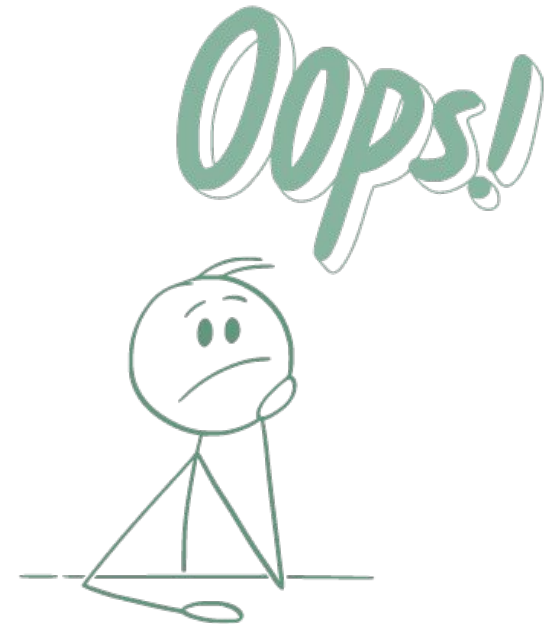
Spell correction

- 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.



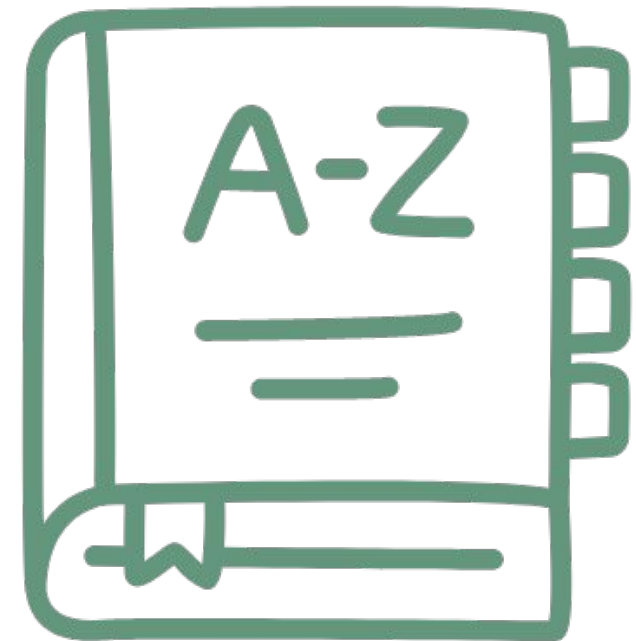
Query mis-spellings

- cigarett..... did you mean cigarette?
- 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



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)

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
- We see this is an application of N-gram index
 - Variants –weight by keyboard layout, etc.



Example with trigrams

- 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

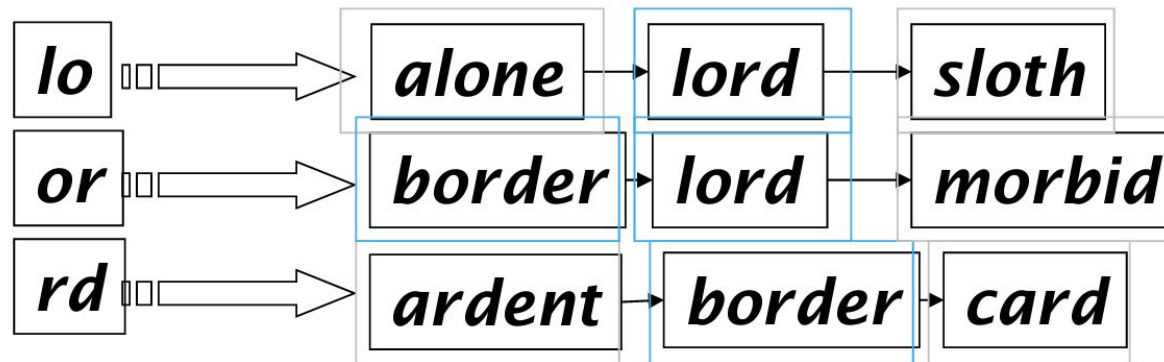
- A commonly-used measure of overlap
- Let X and Y be two sets; then the J.C. is

$$|X \cap Y| / |X \cup Y|$$

- 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 bigrams

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



Standard postings “merge” will enumerate ...



Context-sensitive spell correction

- 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 docs matched the query phrase.



Context-sensitive spell correction

- Need surrounding context to catch this.
 - NLP too heavyweight for 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
 - etc.
- Suggest the alternative that has lots of hits?



Exercise

- Suppose that for “flew form Heathrow” we have 7 alternatives for flew, 19 for form and 3 for heathrow. How many “corrected” phrases will we enumerate in this scheme?
- Once, you find all possible, return the query that has most number of docs



Another approach

- Break phrase query into a conjunction of bi-words (Lecture 2).
- Look for biwords that need only one term corrected.
- Enumerate phrase matches and ... rank them!



General issue in spell correction

- Will enumerate multiple alternatives for “Did you mean”
- Need to figure out which one (or small number) to present to the user
- Use heuristics
 - The alternative hitting most docs (document/collection frequency)
 - Query log analysis + tweaking -> For especially popular, topical queries



Computational cost

- Spell-correction is computationally expensive
- Avoid running routinely on every query?
- Run only on queries that matched few docs



Soundex

- Class of heuristics to expand a query into phonetic equivalents
 - Language specific –mainly for names
 - E.g., chebyshev -> tchebycheff



Soundex –typical algorithm

- 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#To>
[p](#)



Soundex –typical algorithm

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'.
3. Change letters to digits as follows:
 - B, F, P, V \rightarrow 1
 - C, G, J, K, Q, S, X, Z \rightarrow 2
 - D, T \rightarrow 3
 - L \rightarrow 4
 - M, N \rightarrow 5
 - R \rightarrow 6



Soundex –typical algorithm

4. Remove all pairs of consecutive digits.
5. Remove all zeros from the resulting string.
6. Pad the resulting string with trailing zeros and return the first four positions, which will be of the form <uppercase letter> <digit> <digit> <digit>.

E.g., *Herman* becomes H655.

Will *hermann* generate the same code?



Soundex –typical algorithm

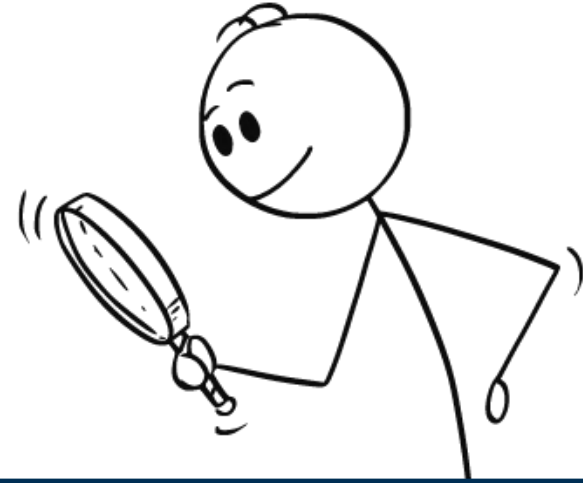
4. Remove all pairs of consecutive digits.
5. Remove all zeros from the resulting string.
6. Pad the resulting string with trailing zeros and return the first four positions, which will be of the form <uppercase letter> <digit> <digit> <digit>.

E.g., *Herman* becomes H655.

Will *hermann* generate the same code?

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>



Project 1 Introduction

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

SOLR features



Schema when you want, schemaless when you don't

Use Solr's data-driven schemaless mode when getting started and then lock it down when it's time for production.



Powerful Extensions

Solr ships with optional plugins for indexing rich content (e.g. PDFs, Word), language detection, search results clustering and more



Faceted Search and Filtering

Slice and dice your data as you see fit using a large array of faceting algorithms



Geospatial Search

Enabling location-based search is simple with Solr's built-in support for spatial search



Advanced Configurable Text Analysis

Solr ships with support for most of the widely spoken languages in the world (English, Chinese, Japanese, German, French and many more) and many other analysis tools designed to make indexing and querying your content as flexible as possible



Highly Configurable and User Extensible Caching

Fine-grained controls on Solr's built-in caches make it easy to optimize performance



Performance Optimizations

Solr has been tuned to handle the world's largest sites



Security built right in

Secure Solr with SSL, Authentication and Role based Authorization. Pluggable, of course!



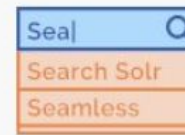
Advanced Storage Options

Building on Lucene's advanced storage capabilities (codecs, directories and more), Solr makes it easy to tune your data storage needs to fit your application



Monitorable Logging

Easily access Solr's log files from the admin interface



Query Suggestions, Spelling and More

Solr ships with advanced capabilities for auto-complete (typeahead search), spell checking and more



Your Data, Your Way!

JSON, CSV, XML and more are supported out of the box. Don't waste precious time converting all your data to a common representation, just send it to Solr!



Rich Document Parsing

Solr ships with Apache Tika built-in, making it easy to index rich content such as Adobe PDF, Microsoft Word and more.



Apache UIMA

Ready to enhance your content with advanced annotation engines? Solr integrates into Apache UIMA, making it easy to leverage NLP and other tools as part of your application.



Multiple search indices

Solr supports multi-tenant architectures, making it easy to isolate users and content.

Where does SOLR fit in

