Advanced topics in Computer Science

Jiaheng Lu
Department of Computer Science
Renmin University of China
www.jiahenglu.net
This lecture

- “Tolerant” retrieval
  - Wild-card queries
  - Spelling correction
  - Soundex
Wild-card queries
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: \textit{mon} \leq w < \textit{moo}
- ***mon**: find words ending in “mon”: harder
  - Maintain an additional B-tree for terms \textit{backwards}.
  - Can retrieve all words in range: \textit{nom} \leq w < \textit{non}.

Exercise: from this, how can we enumerate all terms meeting the wild-card query \textit{pro*cent}?
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:
  \texttt{se*ate AND fil*er}
  
This may result in the execution of many Boolean \texttt{AND} queries.
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.

- Queries:
  - **X** lookup on **X$**
  - **X** lookup on **X$**
  - **X** lookup on **X$**
  - **X** lookup on **X$**

Exercise!

```
Query = *hel*o
X=hel, Y=o
Lookup o$hel*
```
Permuterm query processing

- Rotate query wild-card to the right
- Now use B-tree lookup as before.
- *Permuterm problem:* \( \approx \) quadruples lexicon size

\[\text{Empirical observation for English.}\]
Bigram indexes

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

\[ s, a, a p, p r, r i, i l, l s, i s, s s, s t, t h, h e, e s, c, c r, r u, u e, e l, l e, e s, s t, t s, m, m o, o n, n t, h s \]

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

$m$ -> mace -> madden

mo -> among -> amortize

on -> among -> loony
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

- As before, we must execute a Boolean query for each enumerated, filtered term.
- Wild-cards can result in expensive query execution
  - Avoid encouraging “laziness” in the UI:

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

- Avoiding UI clutter is one reason to hide advanced features behind an “Advanced Search” button.
- It also deters most users from unnecessarily hitting the engine with fancy queries.
Spelling correction
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.*
Document correction

- Primarily for OCR’ed documents
  - Correction algorithms tuned for this
- Goal: the index (dictionary) contains fewer OCR-induced misspellings
- Can use domain-specific knowledge
  - E.g., OCR can confuse O and D more often than it would confuse O and I (adjacent on the QWERTY keyboard, so more likely interchanged in typing).
Query mis-spellings

- Our principal focus here
  - E.g., the query *Alanis Morisett*
- We can either
  - 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)
Isolated word correction

- Given a lexicon and a character sequence Q, return the words in the lexicon closest to Q
- What’s “closest”?
- 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 covert 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.
Edit distance

- Also called “Levenshtein distance”
- See http://www.merriampark.com/ld.htm for a nice example plus an applet to try on your own
Weighted edit distance

- As above, but the weight of an operation depends on the character(s) involved
  - Meant to capture keyboard errors, e.g. $m$ more likely to be mis-typed as $n$ than as $q$
  - Therefore, replacing $m$ by $n$ is a smaller edit distance than by $q$
  - (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
- (Some literature formulates weighted edit distance as a probability of the error)
- Then look up enumerated dictionary terms in the term-document inverted index
  - Slow but no real fix
  - Tries help
- Better implementations – see Kukich, Zobel/Dart references.
Edit distance to all dictionary terms?

- Given a (mis-spelled) query – do we compute its edit distance to every dictionary term?
  - Expensive and slow
- How do we cut the set of candidate dictionary terms?
- Here we use $n$-gram overlap for this
**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 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
  \[ \frac{|X \cap Y|}{|X \cup Y|} \]
  - 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 bigrams

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

Adapt this to using Jaccard (or another) measure.
Caveat

- Even for isolated-word correction, the notion of an index token is critical – what’s the unit we’re trying to correct?
- In Chinese/Japanese, the notions of spell-correction and wildcards are poorly formulated/understood
Context-sensitive spell correction

- Text: *I flew from Heathrow to Narita.*
- Consider the phrase query “*flew from Heathrow*”
- We’d like to respond

Did you mean “*flew from Heathrow*”? because no docs matched the query phrase.