This lecture

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

A naïve dictionary

- An array of struct:
  
<table>
<thead>
<tr>
<th>term</th>
<th>document frequency</th>
<th>postings list</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>656,265</td>
<td>1</td>
</tr>
<tr>
<td>aachen</td>
<td>65</td>
<td>2</td>
</tr>
<tr>
<td>zulu</td>
<td>221</td>
<td>4</td>
</tr>
</tbody>
</table>

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

Dictionary data structures

- Two main choices:
  - Hashtables
  - Trees

- Some IR systems use hash tables, some trees
Hashtables

- Each vocabulary term is hashed to an integer
  - (We assume you’ve seen hashtables 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 growing, need to occasionally do the expensive operation of rehashing everything

Trees

- Simplest: binary tree
- More usual: B-trees
- Trees require a standard ordering of characters and hence strings … but we typically have one
- 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

Wild-card queries: *

- mon*: find all docs containing any word beginning with “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 backwards. Can retrieve all words in range: $\textit{non} \leq w < \textit{nom}$

Exercise: from this, how can we enumerate all terms meeting the wild-card query 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: `se*ate AND fil*er`
  This may result in the execution of many Boolean AND queries.

Spell correction

- Two principal uses
  - Correcting document(s) being indexed
  - Correcting user queries to retrieve “right” answers
- 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 from Heathrow to Narita`.

Document correction

- Especially needed for OCR’ed documents
- Correcting document(s) being indexed
- 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).
- But also: web pages and even printed material have typos
- Goal: the dictionary contains fewer misspellings
- But often we don’t change the documents and instead fix the query-document mapping

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

- 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 (Levenshtein distance)
  - Weighted edit distance

Edit distance

- 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, (Transposition)
- E.g., the edit distance from $dof$ to $dog$ is 1
  - From $cat$ to $act$ is 2 (Just 1 with transpose.)
  - From $cat$ to $dog$ is 3.
- Generally found by dynamic programming.
  - See http://www.merriampark.com/id.htm for a nice example plus an applet.

Weighted edit distance

- As above, but the weight of an operation depends on the character(s) involved
  - Meant to capture OCR or keyboard errors
    - Example: $m$ more likely to be mis-typed as $e$ 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 matrix as input
- Modify dynamic programming to handle weights

Using edit distances

- Given query, first enumerate all character sequences within a preset (weighted) edit distance (e.g., 2)
- Intersect this set with list of "correct" words
- Show terms you found to user as suggestions
- Alternatively,
  - We can look up all possible corrections in our inverted index and return all docs ... slow
  - We can run with a single most likely correction
  - The alternatives disempower the user, but save a round of interaction with the user

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.

Context-sensitive 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 from heathrow
  - flea form heathrow
- Hit-based spelling correction: 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?

What queries can we process?

- We have
  - Positional inverted index with skip pointers
  - Wild-card index
  - Spell-correct

Resources

- IIR 3, MG 4.2
- Efficient spell retrieval:
    http://citeseer.ist.psu.edu/zobel95finding.html
    http://citeseer.ist.psu.edu/175505.html
- Nice, easy reading on spell correction:
  - Peter Norvig: How to write a spelling corrector
    http://norvig.com/spell-correct.html