Lecture 3: Index Representation and Tolerant Retrieval

Information Retrieval
Computer Science Tripos Part II

Helen Yannakoudakis[1]

Natural Language and Information Processing (NLIP) Group
UNIVERSITY OF CAMBRIDGE

helen.yannakoudakis@cl.cam.ac.uk

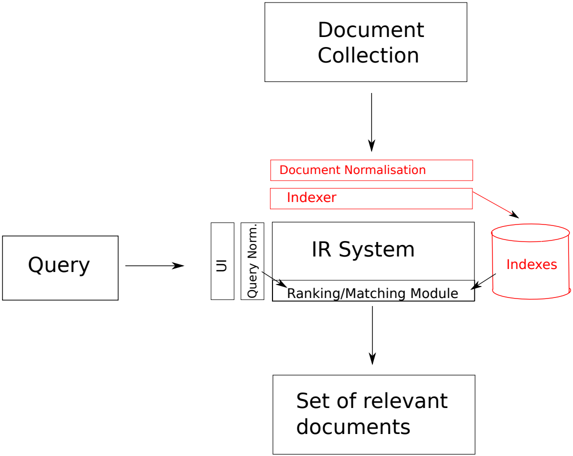
2018

[1] Based on slides from Simone Teufel and Ronan Cummins

Overview

1. Recap
2. Dictionaries
3. Wildcard queries
4. Spelling correction

IR System components



Last time: The indexer

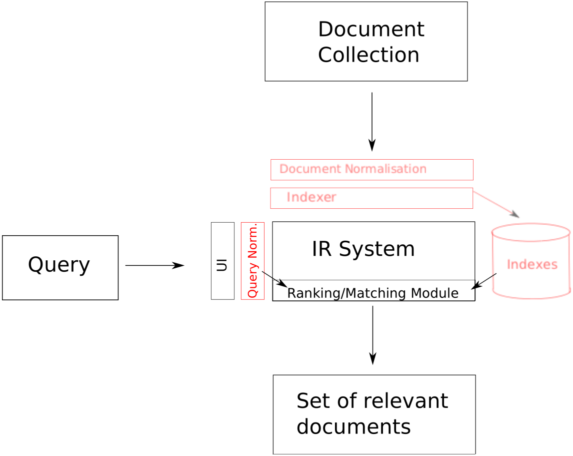
Challenges with equivalence classing

- A term is an equivalence class of tokens.
- How do we define equivalence classes?
- Example: we want to match U.S.A. to USA – can this fail?
- Numbers (3/20/91 vs. 20/3/91)
- Case folding
- Stemming (Porter stemmer)
- Lemmatisation
- Equivalence classing challenges in other languages

Positional indexes

- Postings lists in a non-positional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example query: "to be or not to be"
- With a positional index, we can answer
  - phrase queries
  - proximity queries

IR System components



Today: more indexing, some query normalisation

Upcoming

- Data structures for dictionaries
  - Hashes
  - Trees
  - k-term index
  - Permuterm index
- Tolerant retrieval: What to do if there is no exact match between query term and document term
- Spelling correction

Overview

1. Recap
2. Dictionaries
3. Wildcard queries
4. Spelling correction

Inverted Index

Brutus 8 → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Caesar 9 → 1 → 2 → 4 → 5 → 6 → 16 → 57 → 132 → 179

Calpurnia 4 → 2 → 31 → 54 → 101

Dictionaries

- Dictionary: the data structure for storing the term vocabulary
- Term vocabulary: the data
- For each term, we need to store a couple of items:
  - document frequency
  - pointer to postings list
- How do we look up a query term q_i in the dictionary at query time?

Data structures for looking up terms

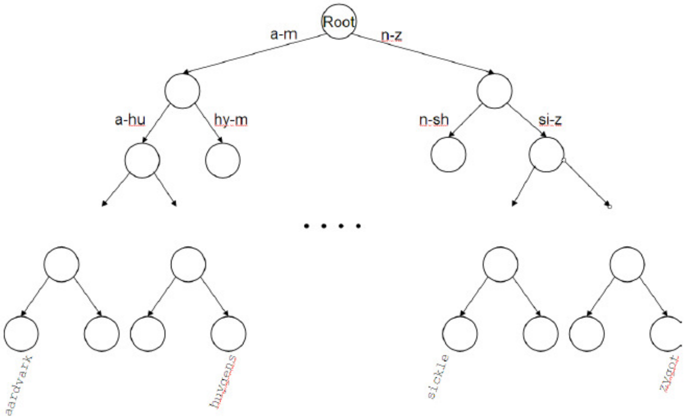
- Two different types of implementations: hashes and search trees.
- Some IR systems use hashes, some use search trees.
- Criteria for when to use hashes vs. search trees:
  - How many terms are we likely to have?
  - Is the number likely to remain fixed, or will it keep growing?
  - What are the relative frequencies with which various terms will be accessed?

Hashes

- Hash table: an array with a hash function
  - Input key; output integer: index in array.
  - Hash function: determine where to store / search key.
  - Hash function that minimises chance of collisions
    - Use all info provided by key (among others).
- Each vocabulary term (key) is hashed into an integer.
- At query time: hash each query term, locate entry in array.
- Pros: Lookup in a hash is faster than lookup in a tree. (Lookup time is constant.)
- Cons:
  - No easy way to find minor variants (resume vs. résumé)
  - No prefix search (all terms starting with automat)
  - Need to rehash everything periodically if vocabulary keeps growing
  - Hash function designed for current needs may not suffice in a few years' time

Search trees overcome many of these issues

- Simplest tree: binary search tree



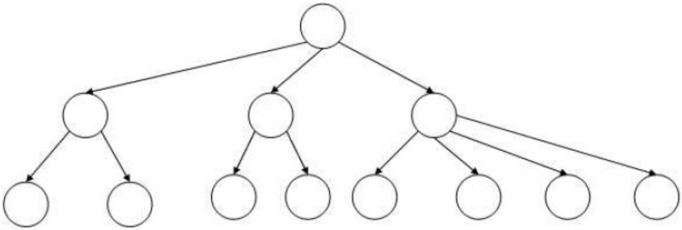
- Figure: partition vocabulary terms into two subtrees, those whose first letter is between a and m, and the rest (actual terms stored in the leafs).
- Anything that is on the left subtree is smaller than what's on the right.
- Trees solve the prefix problem (find all terms starting with automat).

Binary search tree

- Cost of operations depends on height of tree.
- Keep height minimum / keep binary tree balanced: for each node, heights of subtrees differ by no more than 1.
- $O(\log M)$ search for balanced trees, where M is the size of the vocabulary.
- Search is slightly slower than in hashes
- But: re-balancing binary trees is expensive (insertion and deletion of terms).

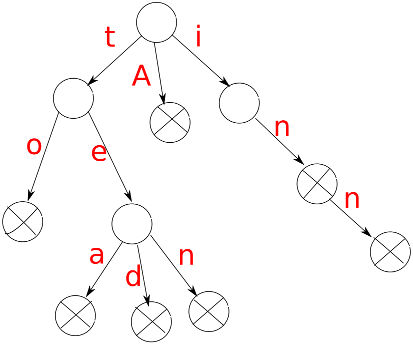
B-tree

- Need to mitigate re-balancing problem – allow the number of sub-trees under an internal node to vary in a fixed interval.
- B-tree definition: every internal node has a number of children in the interval [a, b] where a, b are appropriate positive integers, e.g., [2, 4].



- Figure: every internal node has between 2 and 4 children.

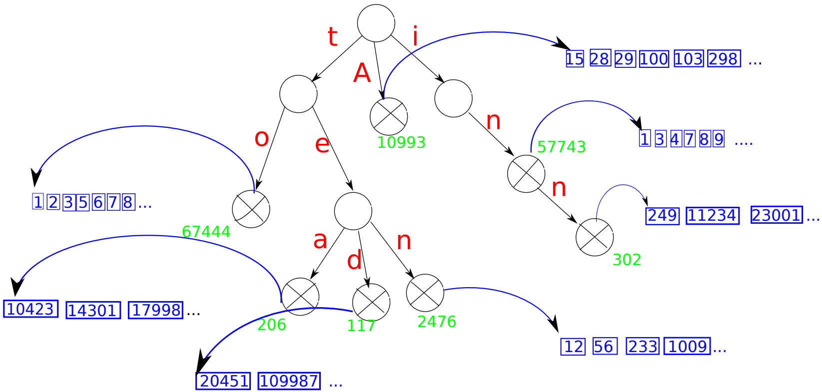
Trie (from *trie* in ret*rie*val)



- An ordered tree data structure for strings
  - A tree where the keys are strings (keys "tea", "ted")
  - Each node is associated with a string inferred from the position of the node in the tree (node stores bit indicating whether string is in collection)
- Tries can be searched by prefixes: all descendants of a node have a common prefix of the string associated with that node
- Search time linear on length of term / key [2]
- The trie is sometimes called radix tree or prefix tree

[2] See https://thenextcode.wordpress.com/2015/04/12/trie-vs-bst-vs-hashtable/

Trie with postings



Overview

1. Recap
2. Dictionaries
3. Wildcard queries
4. Spelling correction

Wildcard queries

hel*

- Find all docs containing any term beginning with "hel"
- Easy with trie: follow letters h-e-l and then lookup every term you find there

*hel

- Find all docs containing any term ending with "hel"
- Maintain an additional trie for terms backwards
- Then retrieve all terms in subtree rooted at l-e-h

In both cases:

- This procedure gives us a set of terms that are matches for the wildcard queries
- Then retrieve documents that contain any of these terms

How to handle * in the middle of a term

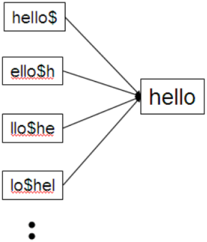
hel*o

- We could look up “hel*” and “*o” in the tries as before and intersect the two term sets (expensive!).
- Solution: permuterm index – special index for general wildcard queries

Permuterm index

For term hello$ (given $ to match the end of a term), store each of these rotations in the dictionary (trie):

hello$, ello$h, llohe, lohel, o$hell, $hello : permuterm vocabulary



Rotate every wildcard query, so that the * occurs at the end:
for hel*o$, look up o$hel*

Problem: Permuterm more than quadrupels the size of the dictionary compared to normal trie (empirical number).

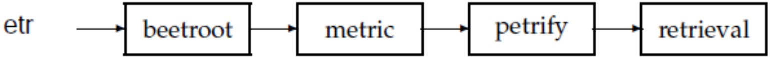
k-gram indexes

- More space-efficient than permuterm index
- Enumerate all character k-grams (sequence of k characters) occurring in a term and store in a dictionary

Character bi-grams from April is the cruelest month

\$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 th h\$

- \$ special word boundary symbol
- A postings list that points to all vocabulary terms containing a k-gram



k-gram indexes

Note that we have two different kinds of inverted indexes:

- The term–document inverted index for finding documents based on a query consisting of terms
- The k-gram index for finding terms based on a query consisting of k-grams

Processing wildcard queries in a (char) bigram index

- Query hel* can now be run as:

$h AND he AND el

- ... but this will show up many false positives like heel.
- Post-filter, then look up surviving terms in term–document inverted index.
- k-gram vs. permuterm index
  - k-gram index is more space-efficient
  - permuterm index does not require post-filtering.

Overview

1. Recap
2. Dictionaries
3. Wildcard queries
4. Spelling correction

Spelling correction

an asterorid that fell form the sky

information need: britney spears

queries: britian spears, britney's spears, brandy spears, prittany spears

- In an IR system, spelling correction is only ever run on queries.
- The general philosophy in IR is: don't change the documents (exception: OCR'ed documents)

Spelling correction

- In an IR system, spelling correction is only ever run on queries.
- The general philosophy in IR is: don't change the documents (exception: OCR'ed documents)
- Two different methods for spelling correction:
  - Isolated word spelling correction
    - Check each word on its own for misspelling
    - Will only attempt to catch first typo above
  - Context-sensitive spelling correction
    - Look at surrounding words
    - Should correct both typos above

Isolated word spelling correction

- There is a list of “correct” words – for instance a standard dictionary (Webster’s, OED. . .)
- Then we need a way of computing the distance between a misspelled word and a correct word
  - for instance Edit/Levenshtein distance
  - k-gram overlap
- Return the “correct” word that has the smallest distance to the misspelled word.

informaton $\rightarrow$ information

- **Edit distance** between two strings s_1 and s_2 is defined as the minimum number of basic operations that transform s_1 into s_2.
- **Levenshtein distance:** Admissible operations are insert, delete and replace

Levenshtein distance

dog	–	do	1 (delete)
cat	–	cart	1 (insert)
cat	–	cut	1 (replace)
cat	–	act	2 (delete+insert)

Levenshtein distance: Distance matrix

		s	n	o	w
	0	1	2	3	4
o	1	1	2	3	4
s	2	1	3	3	3
l	3	3	2	3	4
o	4	3	3	2	3

Cormen et al:

- Optimal substructure: The optimal solution contains within it subsolutions, i.e, optimal solutions to subproblems
- Overlapping subsolutions: The subsolutions overlap and would be computed over and over again by a brute-force algorithm.

For edit distance:

- Subproblem: edit distance of two prefixes
- Overlap: most distances of prefixes are needed 3 times (when moving right, diagonally, down in the matrix)

Example: Edit Distance OSLO – SNOW

			s		n		o		w	
		0	1	1	2	2	3	3	4	4
o		1	1	2	2	3	2	4	4	5
		1	2	1	2	2	3	2	3	3
s		2	1	2	2	3	3	3	3	4
		2	3	1	2	2	3	3	4	3
l		3	3	2	2	3	3	4	4	4
		3	4	2	3	2	3	3	4	4
o		4	4	3	3	3	2	4	4	5
		4	5	3	4	3	4	2	3	3

Edit distance OSLO–SNOW is 3! (minimum number of basic operations that transform OSLO to SNOW)

How do I read out the editing operations that transform OSLO into SNOW?

cost	operation	input	output
1	delete	o	*
0	(copy)	s	s

Each cell of Levenshtein matrix

Cost of getting here from my upper left neighbour (by copy or replace)	Cost of getting here from my upper neighbour (by delete)
Cost of getting here from my left neighbour (by insert)	Minimum cost out of these

Levenshtein Distance: Four cells

			s		n		o		w	
		0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o		**1**	**1**	2	**2**	3	**2**	4	4	5
		1	2	**1**	**2**	**2**	3	**2**	**3**	**3**
s		**2**	**1**	2	**2**	3	**3**	**3**	**3**	4
		2	3	**1**	**2**	**2**	**3**	**3**	4	**3**
l		**3**	3	**2**	**2**	3	**3**	4	**4**	**4**
		3	4	**2**	3	**2**	**3**	**3**	**4**	**4**
o		**4**	4	**3**	**3**	**3**	**2**	4	4	5
		4	5	**3**	4	**3**	4	**2**	**3**	**3**

Example: (2, 2):

- Upper left: cost to replace "o" to "s" (cost: 0+1)
- Upper right: come from above where I have already inserted "s": all I need to do is delete "o" (cost: 1+1)
- Bottom left: come from left neighbour where I have deleted "o": all I need to do is insert "s" (cost: 1+1)
- Then choose the minimum of the three (bottom right).

Using edit distance for spelling correction

- Given a query, enumerate all character sequences within a pre-set edit distance.
- Intersect this list with our list of “correct” words.
- Suggest terms in the intersection to user.

k-gram indexes for spelling correction

- Enumerate all k-grams in the query term

Misspelled word bordroom

bo – or – rd – dr – ro – oo – om

- Use k-gram index to retrieve "correct" words that match query term k-grams
- Threshold by number of matching k-grams
- Eg. only vocabularly terms that differ by at most 3 k-grams

BO → aboard → about → boardroom → border

OR → border → lord → morbid → sordid

RD → aboard → ardent → boardroom → border

Context-sensitive Spelling correction

One idea: hit-based spelling correction

flew form munich

- Enumerate corrections of each of the query terms

flew → flea
form → from
munich → munch

- Holding all other terms fixed, try all possible phrase queries for each replacement candidate

flea form munich – 62 results
flew from munich –78900 results
flew form munch – 66 results

Not efficient. Better source of information: large corpus of queries, not documents

General issues in spelling correction

- User interface
  - automatic vs. suggested correction
  - "Did you mean" only works for one suggestion; what about multiple possible corrections?
  - Trade-off: Simple UI vs. powerful UI
- Cost
  - Potentially very expensive
  - Avoid running on every query
  - Maybe just those that match few documents

Takeaway

- What to do if there is no exact match between query term and document term
- Data structures for tolerant retrieval:
  - Dictionary as hash, B-tree or trie
  - k-gram index and permuterm for wildcards
  - k-gram index and edit-distance for spelling correction

Reading

- Wikipedia article "trie"
- MRS chapter 3.1, 3.2, 3.3