Lecture 3: Index Representation and Tolerant Retrieval

Information Retrieval
Computer Science Tripos Part II

Ronan Cummins[1]

Natural Language and Information Processing (NLIP) Group

UNIVERSITY OF CAMBRIDGE

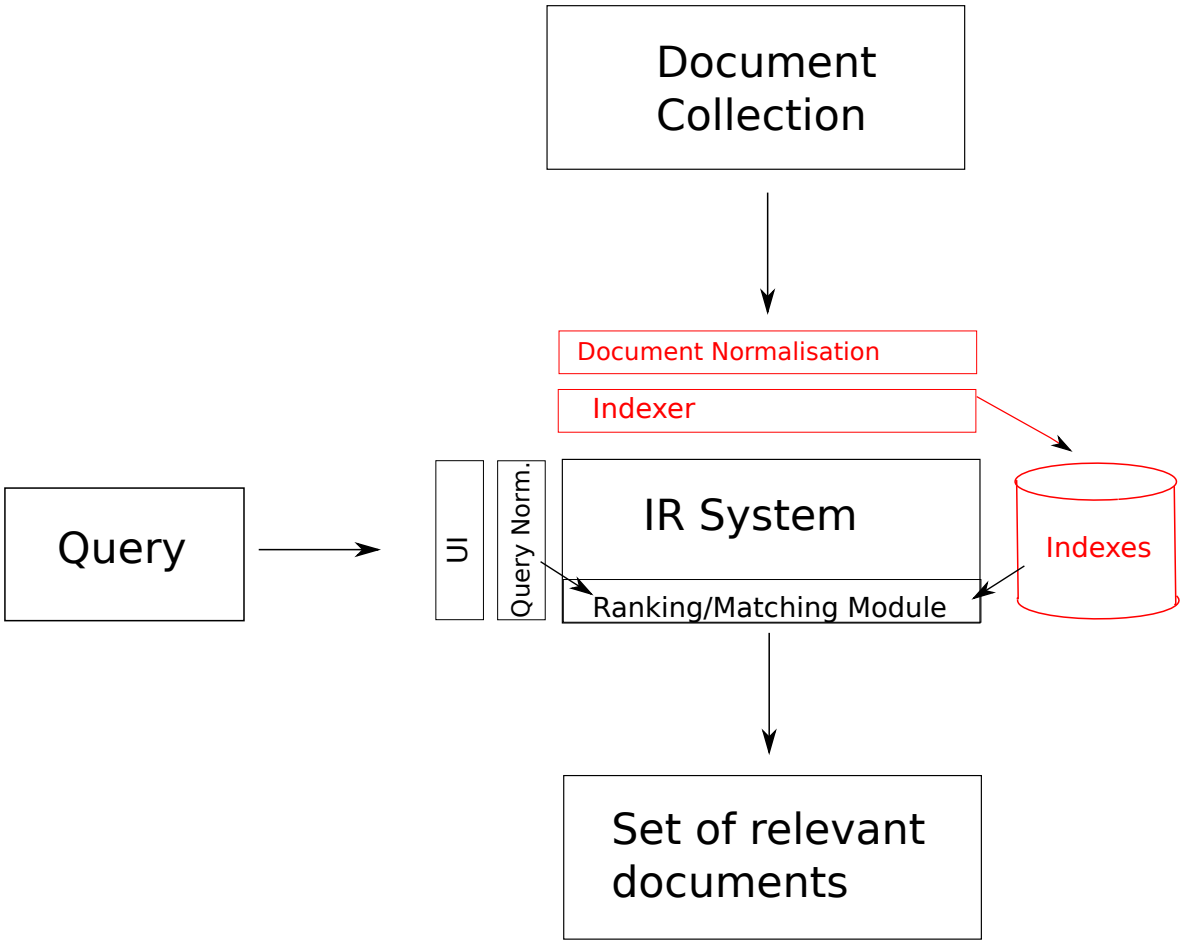
ronan.cummins@cl.cam.ac.uk

2017

[1] Adapted from Simone Teufel's original slides

Overview

IR System components



Last time: The indexer

Type/token distinction

- Token an instance of a word or term occurring in a document
- Type an equivalence class of tokens

> In June, the dog likes to chase the cat in the barn.

- 12 word tokens
- 9 word types

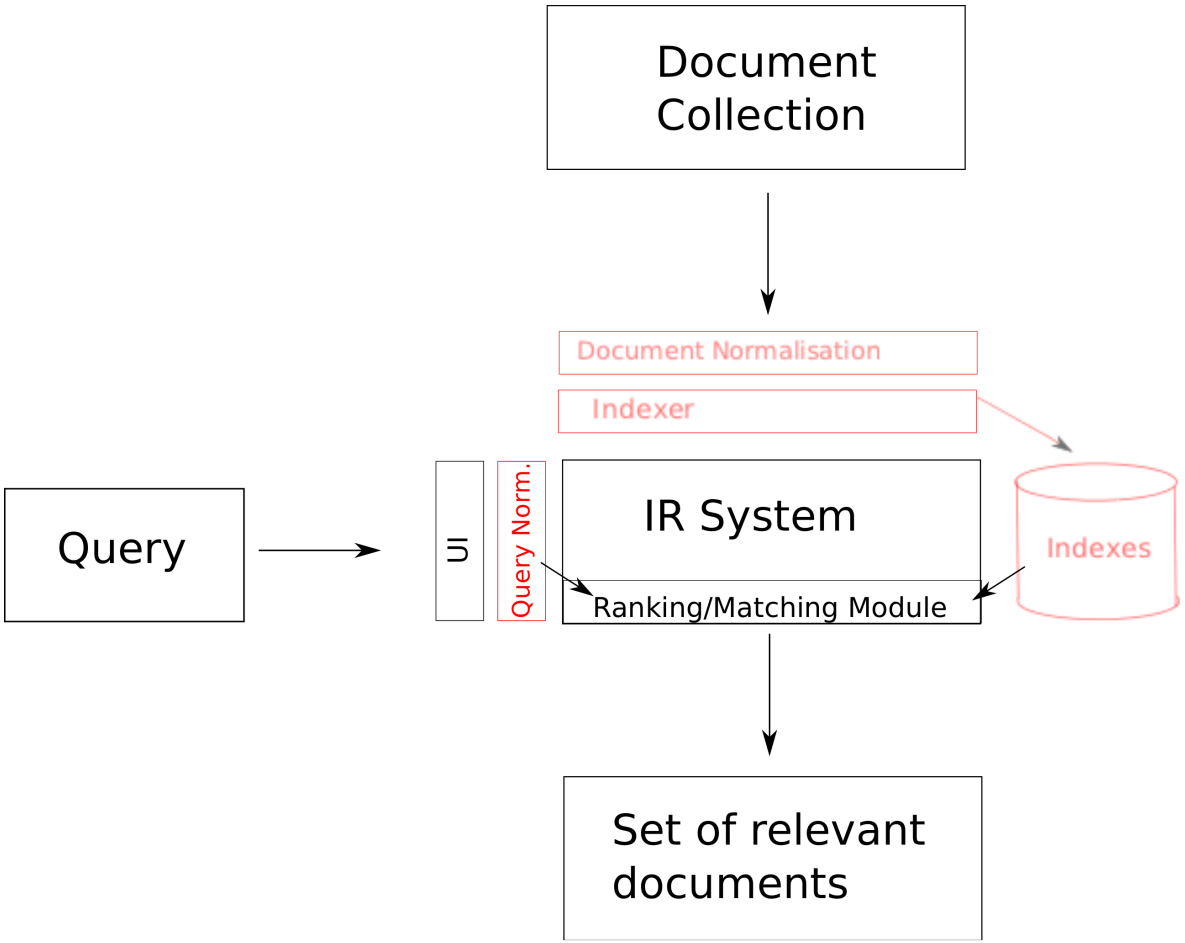
Problems with equivalence classing

- A term is an equivalence class of tokens.
- How do we define equivalence classes?
- Numbers (3/20/91 vs. 20/3/91)
- Case folding
- Stemming, Porter stemmer
- Morphological analysis: inflectional vs. derivational
- Equivalence classing problems in other languages

Positional indexes

- Postings lists in a nonpositional 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_1 be_2 or_3 not_4 to_5 be_6"
- With a positional index, we can answer
  - phrase queries
  - proximity queries

IR System components



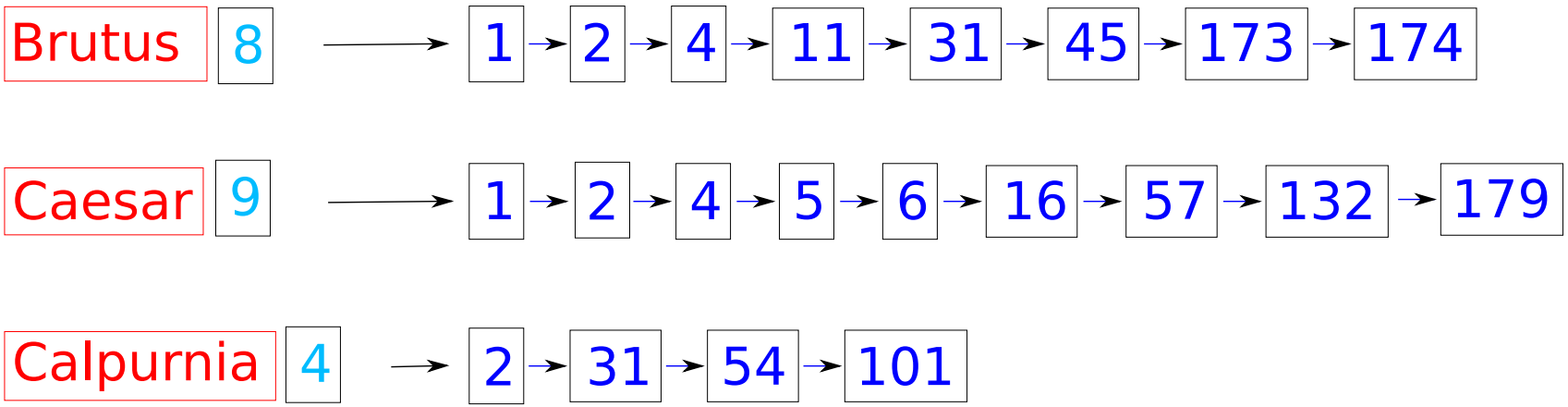
Today: more indexing, some query normalisation

Upcoming

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

Overview

Inverted Index



Dictionaries

- The dictionary is the data structure for storing the term vocabulary.
- Term vocabulary: the data
- Dictionary: the data structure for storing the term vocabulary

Dictionaries

- 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 main classes of data structures: hashes and trees
- Some IR systems use hashes, some use trees.
- Criteria for when to use hashes vs. trees:
  - Is there a fixed number of terms or will it keep growing?
  - What are the relative frequencies with which various keys will be accessed?
  - How many terms are we likely to have?

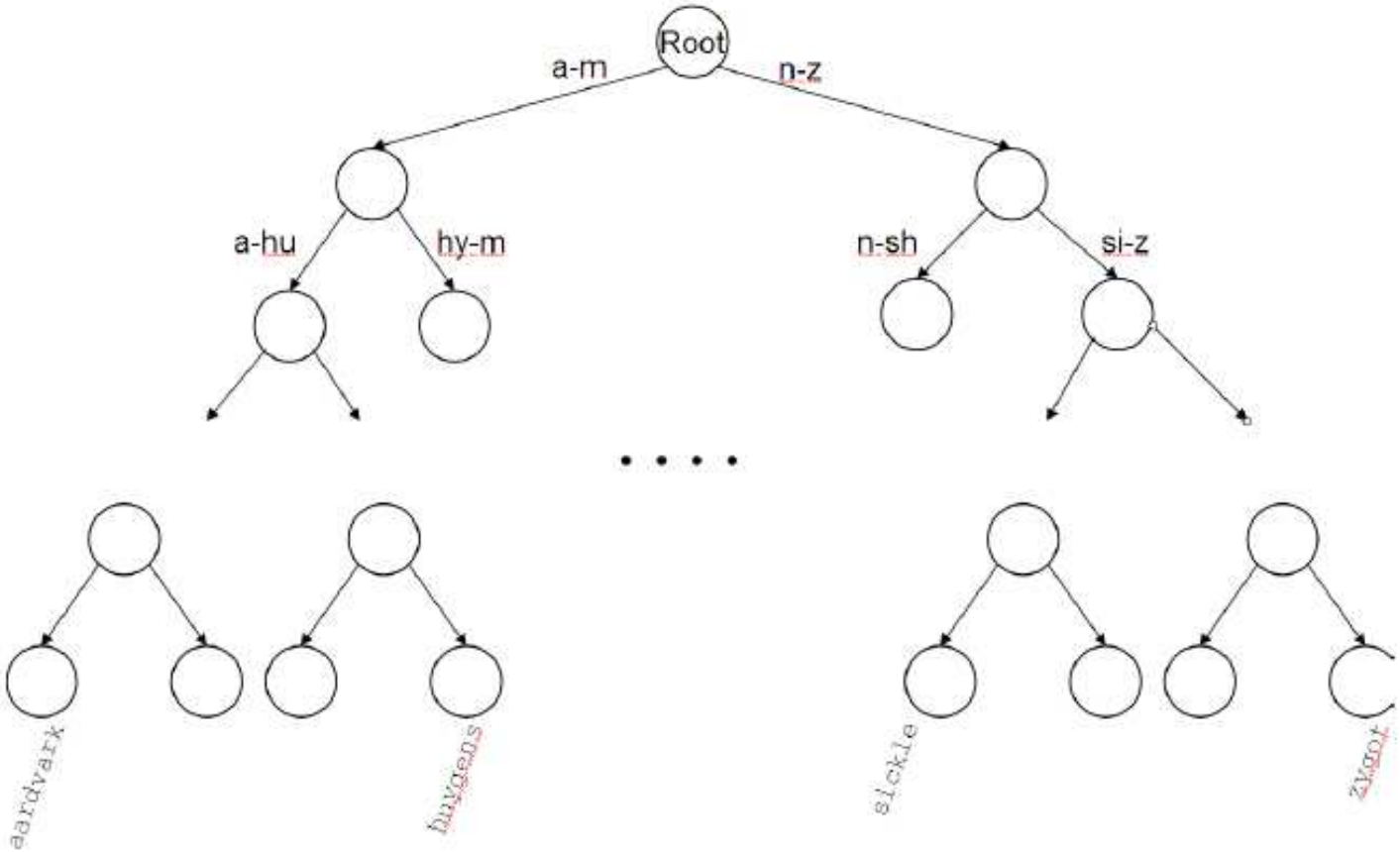
Hashes

- Each vocabulary term is hashed into an integer, its row number in the array
- At query time: hash query term, locate entry in fixed-width array
- Pros: Lookup in a hash is faster than lookup in a tree. (Lookup time is constant.)
- Cons
  - no 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

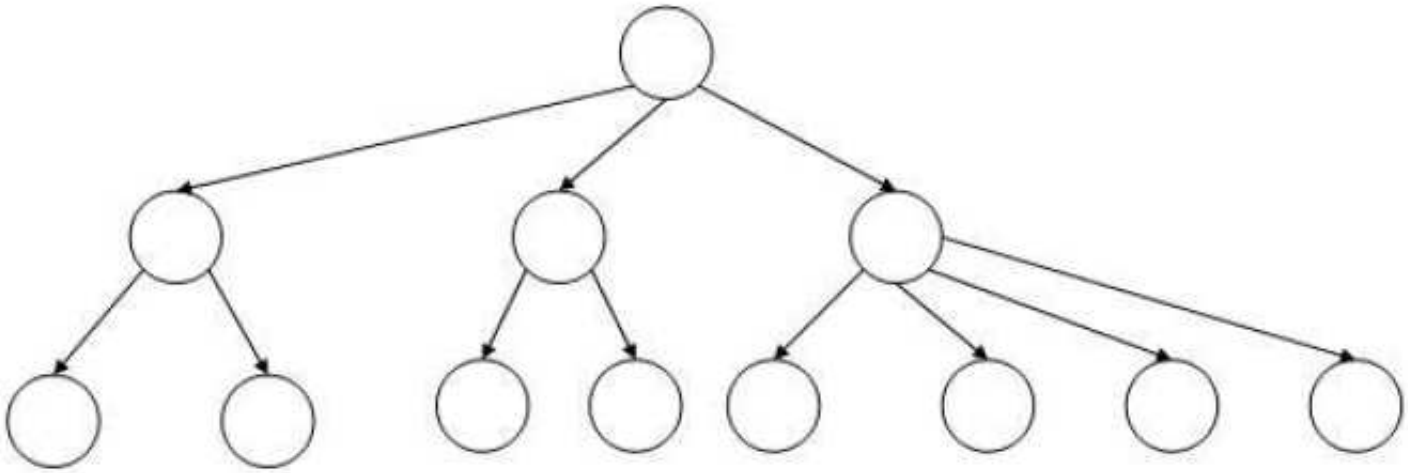
Trees

- Trees solve the prefix problem (find all terms starting with automat).
- Simplest tree: binary tree
- Search is slightly slower than in hashes: O(logM), where M is the size of the vocabulary.
- O(logM) only holds for balanced trees.
- Rebalancing binary trees is expensive.
- B-trees mitigate the rebalancing problem.
- 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].

Binary tree



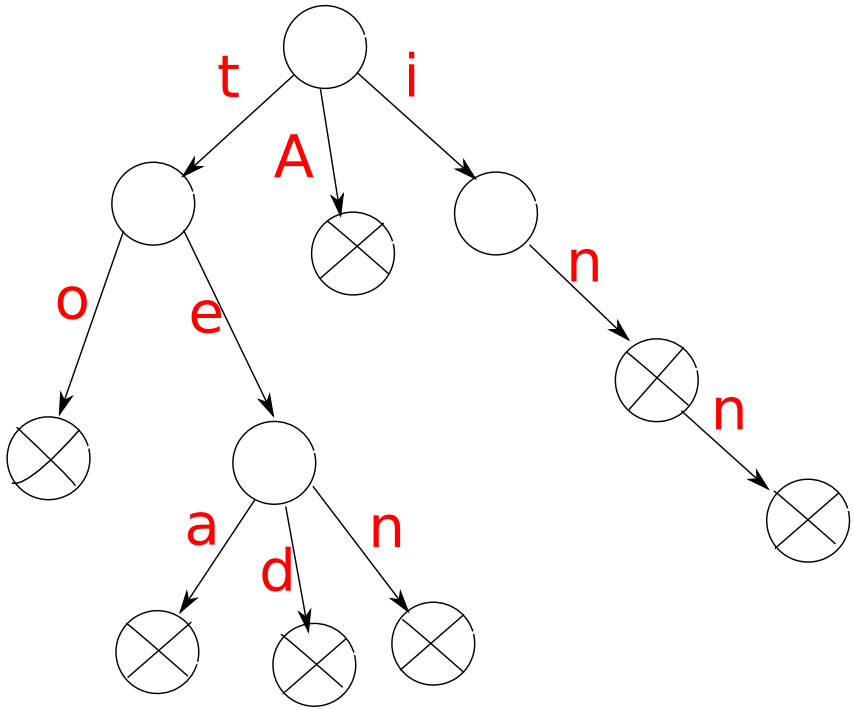
B-tree



Trie

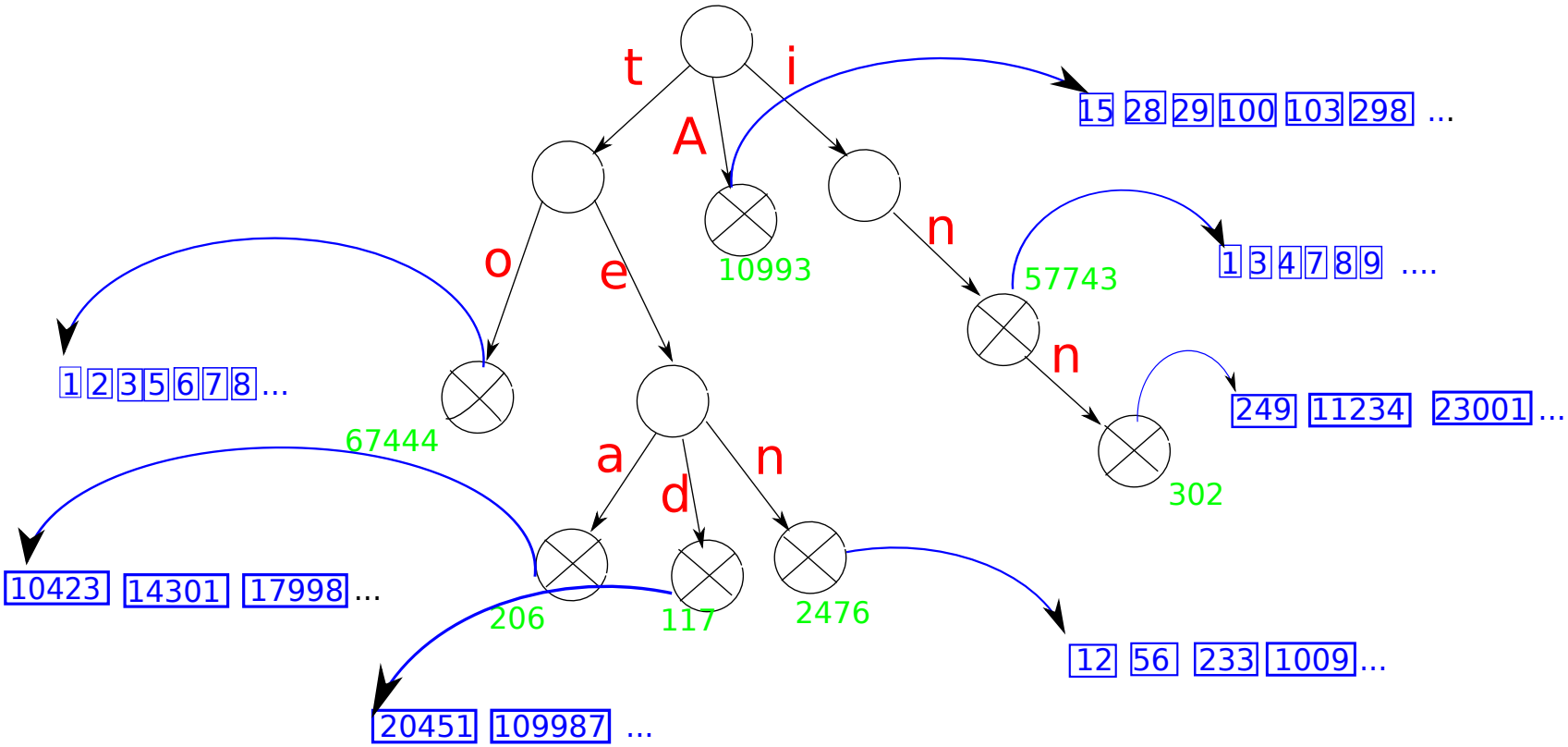
- An ordered tree data structure that is used to store an associative array
- The keys are strings
- The key associated with a node is inferred from the position of a node in the tree
  - Unlike in binary search trees, where keys are stored in nodes.
- Values are associated only with with leaves and some inner nodes that correspond to keys of interest (not all nodes).
- All descendants of a node have a common prefix of the string associated with that node $\rightarrow$ tries can be searched by prefixes
- The trie is sometimes called radix tree or prefix tree

Trie



A trie for keys "A", "to", "tea", "ted", "ten", "in", and "inn".

Trie with postings



Overview

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 t in subtree rooted at l-e-h

In both cases:

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

How to handle * in the middle of a term

hel*o

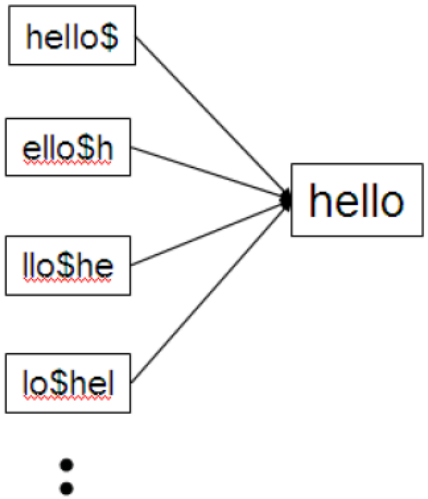
- We could look up "hel*" and "*o" in the tries as before and intersect the two term sets.
  - Expensive
- Alternative: permuterm index
- Basic idea: Rotate every wildcard query, so that the * occurs at the end.
- Store each of these rotations in the dictionary (trie)

Permuterm index

For term hello: add

hello$, ello$h, llohe, lohel, o$hell, $hello

to the trie where $ is a special symbol



for hel*o, look up o$hel*

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

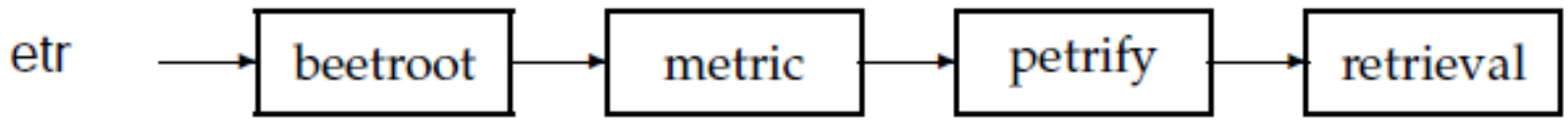
k-gram indexes

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

Bi-grams from April is the cruelest month

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

- Maintain an inverted index from k-grams to the term that contain the 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 terms in a bigram index

- Query hel* can now be run as:

$h AND he AND el

- ... but this will show up many false positives like heel.
- Postfilter, 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 postfiltering.

Overview

Spelling correction

an asterorid that fell form the sky

- 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

- **Edit distance** between two strings s_1 and s_2 is 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

Edit Distance: Four cells

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

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

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**	**1** 2	2 **1**	**2** **2**	3 **2**	**2** 3	4 **2**	4 **3**	5 **3**
s		**2** **2**	**1** 3	2 **1**	**2** **2**	3 **2**	**3** **3**	**3** **3**	**3** 4	4 **3**
l		**3** **3**	3 4	**2** **2**	**2** 3	3 **2**	**3** **3**	4 **3**	**4** **4**	**4** **4**
o		**4** **4**	4 5	**3** **3**	**3** 4	**3** **3**	**2** 4	4 **2**	4 **3**	5 **3**

Edit distance OSLO–SNOW is 3! How do I read out the editing operations that transform OSLO into SNOW?

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

Using edit distance for spelling correction

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

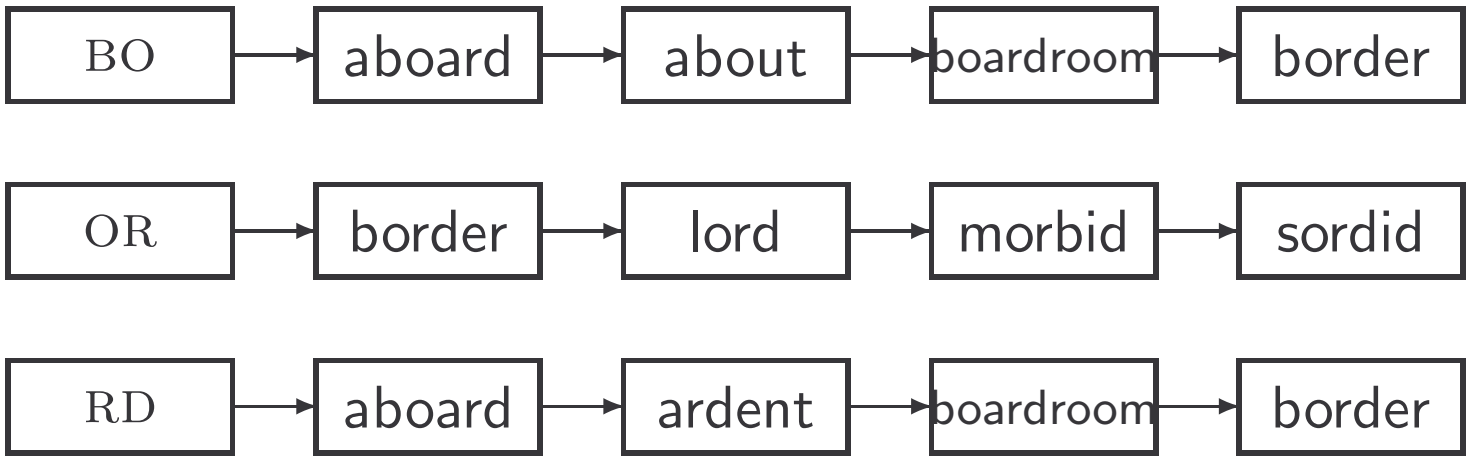
k-gram indexes for spelling correction

- 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



Context-sensitive Spelling correction

One idea: hit-based spelling correction

flew form munich

- Retrieve correct terms close to each query term

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?
  - Tradeoff: 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
- Datastructures 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