مبانی بازیابی اطلاعات و جستجوی وب

۴- Dictionaries and tolerant retrieval

Original slides by: Hinrich Schütze and Christina Lioma

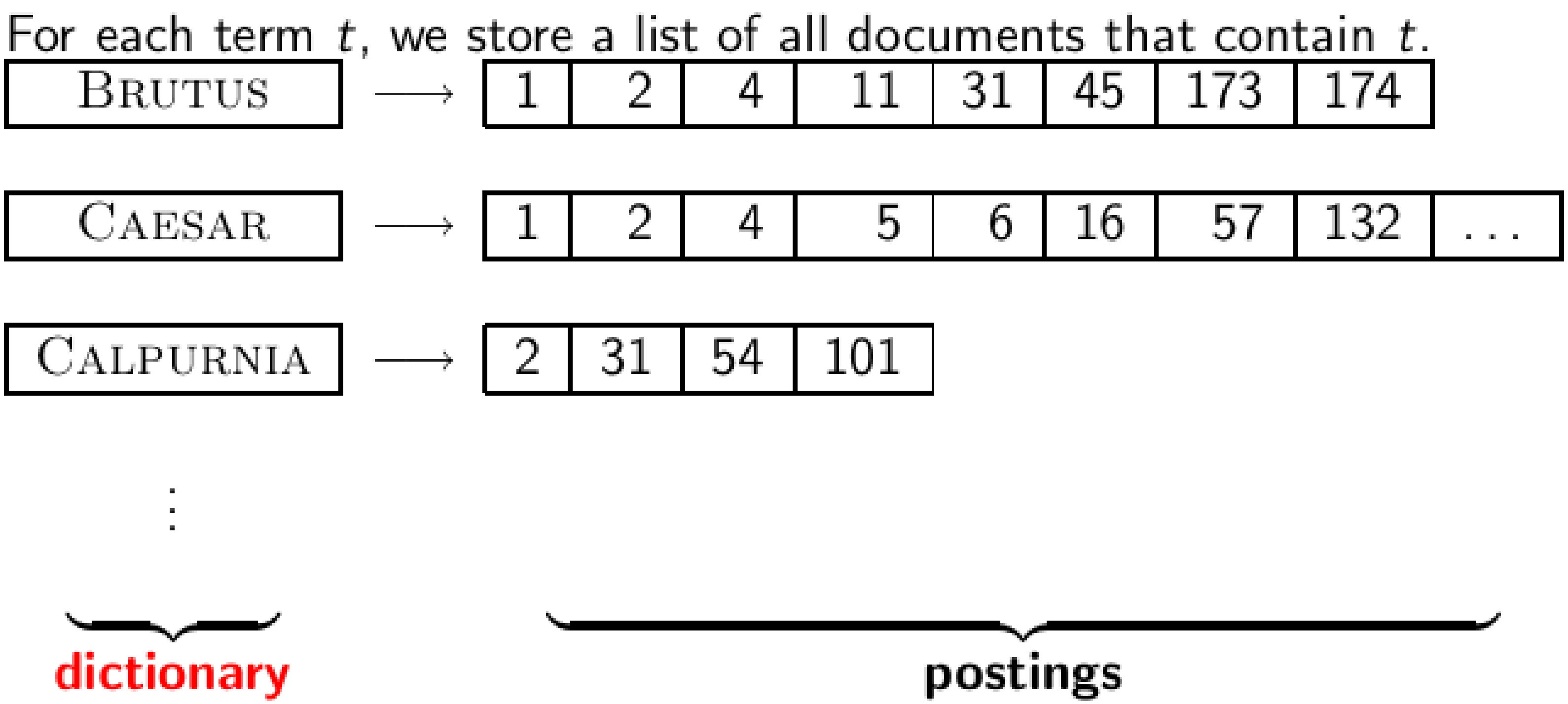
Overview

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

Outline

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

Inverted index



4

Dictionary as array of fixed-width entries

- For each term, we need to store a couple of items:
  - document frequency
  - pointer to postings list
  - . . .
- Assume for the time being that we can store this information in a fixed-length entry.
- Assume that we store these entries in an array.

5

Dictionary as array of fixed-width entries

term	document frequency	pointer to postings list
a	656,265	$\longrightarrow$
aachen	65	$\longrightarrow$
...	...	...
zulu	221	$\longrightarrow$

space needed: 20 bytes 4 bytes 4 bytes

How do we look up a query term q_i in this array at query time? That is: which data structure do we use to locate the entry (row) in the array where q_i is stored?

6

Data structures for looking up term

- 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?

7

Hashes

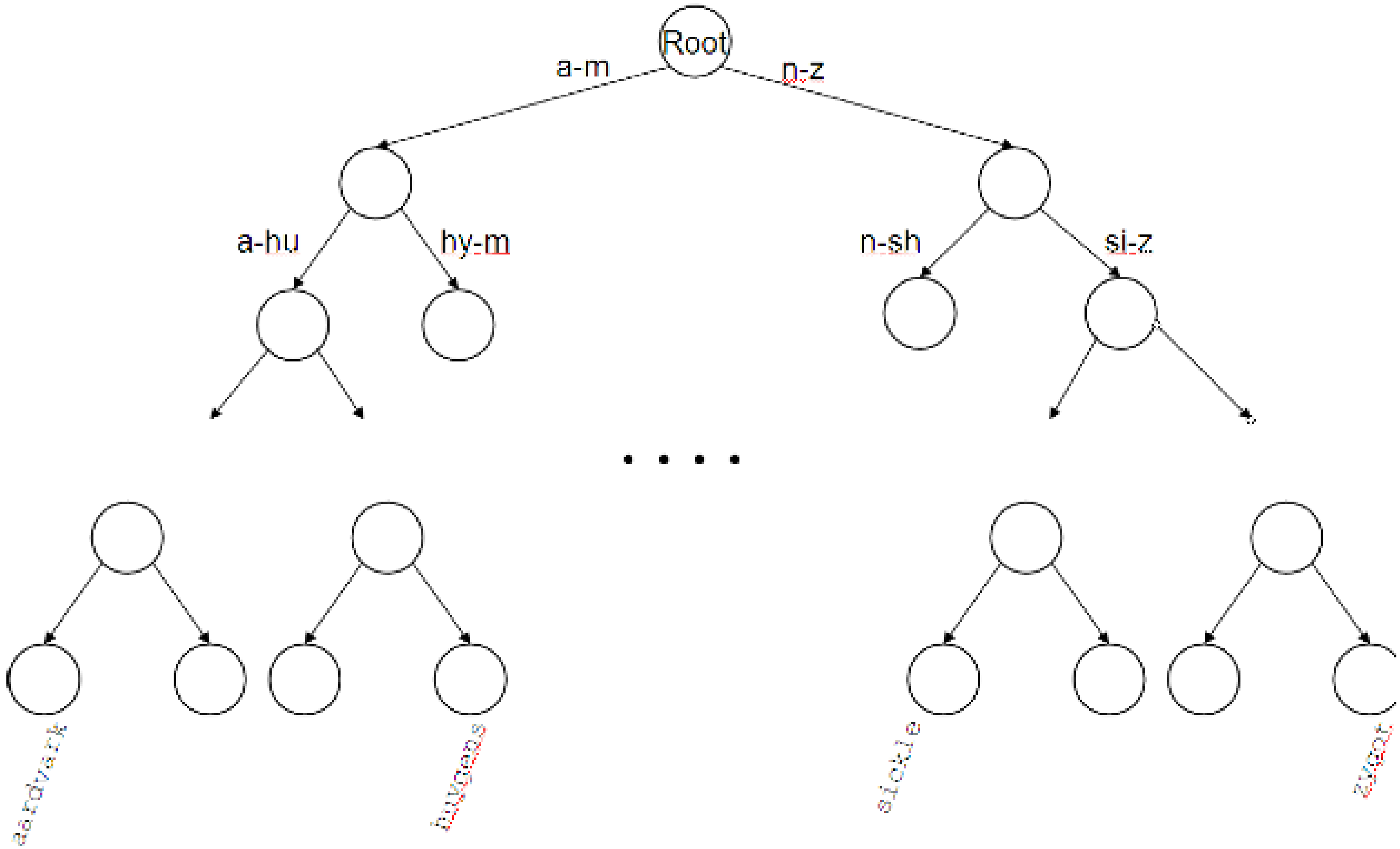
- Each vocabulary term is hashed into an integer.
- Try to avoid collisions
- At query time, do the following: hash query term, resolve collisions, locate entry in fixed-width array
- Pros: Lookup in a hash is faster than lookup in a tree.
  - Lookup time is constant.
- Cons
  - no way to find minor variants (*resume* vs. *résumé*)
  - no prefix search (all terms starting with *automat*)
  - need to rehash everything periodically if vocabulary keeps growing

8

Trees

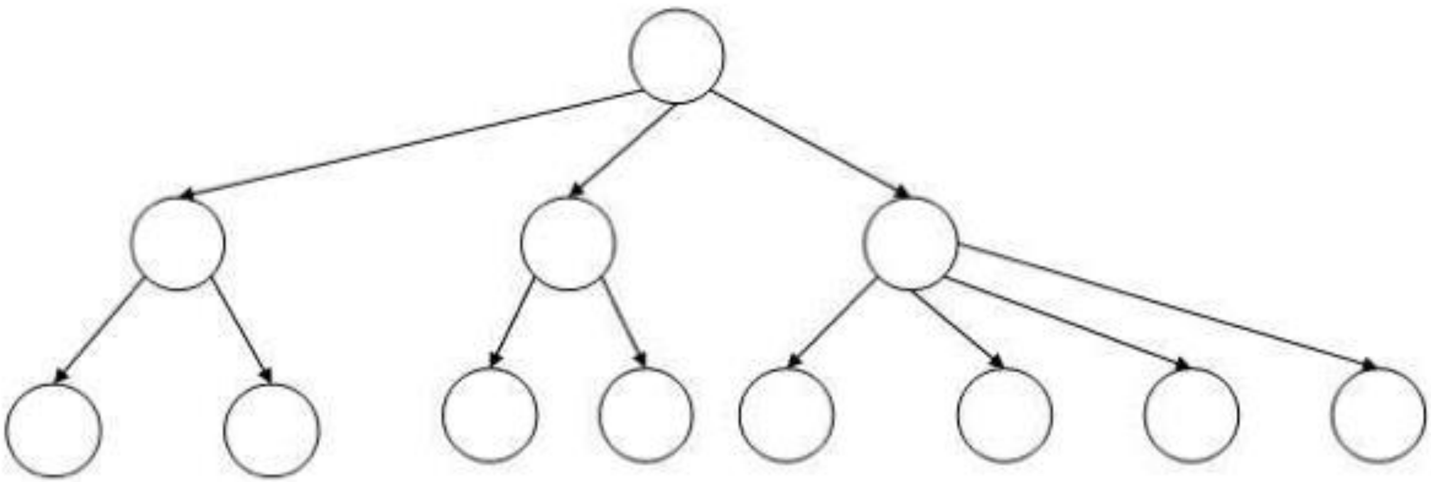
- Trees solve the prefix problem (find all terms starting with *automat*).
- Simplest tree: binary tree
- Search is slightly slower than in hashes: $O(\log M)$, where M is the size of the vocabulary.
- $O(\log M)$ only holds for balanced trees.
- Rebalancing binary trees is expensive.
- B-trees mitigate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



10

B-tree



11

Outline

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

Wildcard queries

- mon*: find all docs containing any term beginning with *mon*
- Easy with B-tree dictionary: retrieve all terms t in the range: mon ≤ t < moo
- *mon: find all docs containing any term ending with *mon*
  - Maintain an additional tree for terms *backwards*
  - Then retrieve all terms t in the range: nom ≤ t < non
- Result: A set of terms that are matches for wildcard query
- Then retrieve documents that contain any of these terms

13

How to handle * in the middle of a term

- Example: m*nchen
- We could look up m* and *nchen in the B-tree 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, say, in a B-tree

14

Permuterm index

- For term HELLO: add *hello$, ello$h, llohe, lohel, and o$hell* to the B-tree where $ is a special symbol

15

Permuterm index

- For HELLO, we've stored: *hello$, ello$h, llohe, lohel, and o$hell*
- Queries
  - For X, look up X$
  - For X*, look up X*$
  - For *X, look up X$*
  - For *X*, look up X*
  - For X*Y, look up Y$X*
  - Example: For hel*o, look up o$hel*
- Permuterm index would better be called a permuterm tree.
- But permuterm index is the more common name.

16

k-gram indexes

- More space-efficient than permuterm index
- Enumerate all character *k*-grams (sequence of *k* characters) occurring in a term
- 2-grams are called bigrams.
- Example: from *April is the cruelest month* we get the 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, as before.
- Maintain an inverted index from bigrams to the terms that contain the bigram

Postings list in a 3-gram inverted index

18

Processing wildcarded terms in a bigram index

- Query mon* can now be run as: $m AND mo AND on
- Gets us all terms with the prefix *mon* . . .
- . . . but also many "false positives" like MOON.
- We must postfilter these terms against query.
- Surviving terms are then looked up in the term-document inverted index.
- *k*-gram index vs. permuterm index
  - *k*-gram index is more space efficient.
  - Permuterm index doesn't require postfiltering.

Exercise

- Google has very limited support for wildcard queries.
- For example, this query doesn’t work very well on Google: [gen* universit*]
  - Intention: you are looking for the University of Geneva, but don’t know which accents to use for the French words for university and Geneva.
- According to Google search basics, 2010-04-29: “Note that the * operator works only on whole words, not parts of words.”
- But this is not entirely true. Try [pythag*] and [m*nchen]
- Exercise: Why doesn’t Google fully support wildcard queries?

20

Outline

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

Spelling correction

- Two principal uses
  - Correcting documents being indexed: The general philosophy in IR is: don't change the documents
  - Correcting user queries
- Two different methods for spelling correction
- Isolated word spelling correction
  - Check each word on its own for misspelling
  - Will not catch typos resulting in correctly spelled words, e.g., *an asteroid that fell form the sky*
- Context-sensitive spelling correction
  - Look at surrounding words
  - Can correct *form/from* error above

22

Edit distance

- The edit distance between string s_1 and string s_2 is the minimum number of basic operations that convert s_1 to s_2.
- Levenshtein distance: The admissible basic operations are insert, delete, and replace
- Levenshtein distance *dog-do*: 1
- Levenshtein distance *cat-cart*: 1
- Levenshtein distance *cat-cut*: 1
- Levenshtein distance *cat-act*: 2

23

Optional : Levenshtein distance: Algorithm

$\text{LevenshteinDistance}(s_1, s_2)$

for $i \leftarrow 0$ **to** $|s_1|$
do $m[i, 0] = i$
for $j \leftarrow 0$ **to** $|s_2|$
do $m[0, j] = j$
for $i \leftarrow 1$ **to** $|s_1|$
do for $j \leftarrow 1$ **to** $|s_2|$
    do if $s_1[i] = s_2[j]$
        then $m[i, j] = \min\{m[i\text{-}1, j]+1, m[i, j\text{-}1]+1, m[i\text{-}1, j\text{-}1]\}$
        else $m[i, j] = \min\{m[i\text{-}1, j]+1, m[i, j\text{-}1]+1, m[i\text{-}1, j\text{-}1]+1\}$
return $m[|s_1|, |s_2|]$

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

24

Spelling correction

- Now that we can compute edit distance: how to use it for isolated word spelling correction

Next:

- *k*-gram indexes for isolated word spelling correction.
- Context-sensitive spelling correction
- General issues

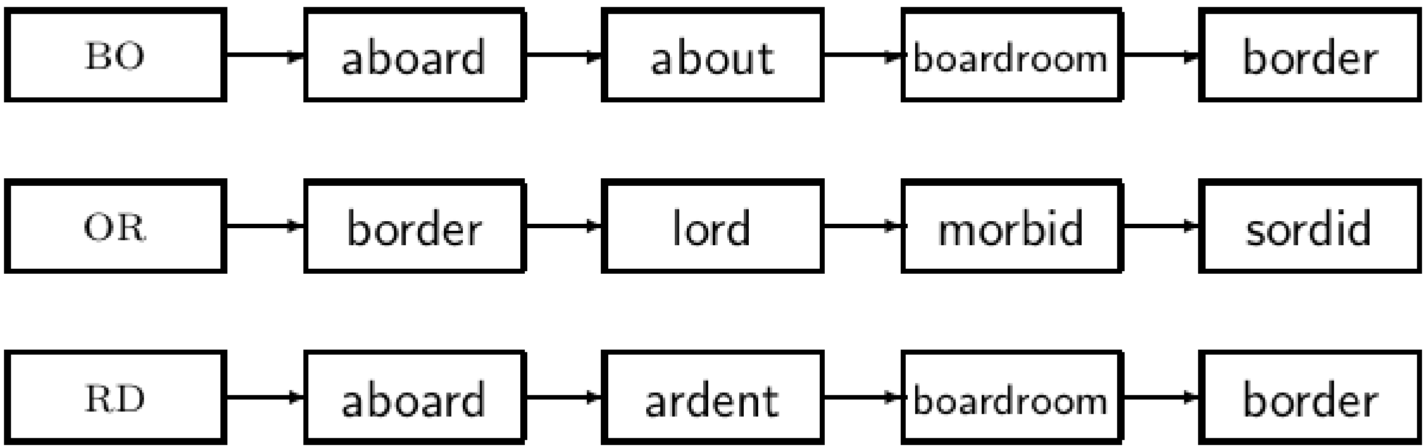
25

k-gram indexes for spelling correction

- Enumerate all *k*-grams in the query term
- Example: bigram index, misspelled word bordroom
- Bigrams: *bo, or, rd, dr, ro, oo, om*
- Use the *k*-gram index to retrieve "correct" words that match query term *k*-grams
- Threshold by number of matching *k*-grams
- E.g., only vocabulary terms that differ by at most 3 *k*-grams

26

k-gram indexes for spelling correction: *bordroom*



27

Optional: Context-sensitive spelling correction

- Our example was: *an asteroid that fell form the sky*
- How can we correct *form* here?
- One idea: hit-based spelling correction
  - Retrieve "correct" terms close to each query term
  - *for flew form munich: flea for flew, from for form, munch for*
  - *munich*
  - Now try all possible resulting phrases as queries with one word "fixed" at a time
  - Try query *"flea form munich"*
  - Try query *"flew from munich"*
  - Try query *"flew form munch"*
  - The correct query *"flew from munich"* has the most hits.
- Suppose we have 7 alternatives for *flew*, 20 for form and 3 for *munich*, how many "corrected" phrases will we enumerate?

28

Context-sensitive spelling correction

- The “hit-based” algorithm we just outlined is not very efficient.
- More efficient alternative: look at “collection” 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 vs. powerful UI
- Cost
  - Spelling correction is potentially expensive.
  - Avoid running on every query?
  - Maybe just on queries that match few documents.
  - Guess: Spelling correction of major search engines is efficient enough to be run on every query.

30

Outline

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

Optional: Soundex

- Soundex is the basis for finding phonetic (as opposed to orthographic) alternatives.
- Example: *chebyshev / tchebyscheff*
- 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

32

Soundex algorithm

1. Retain the first letter of the term.
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 to 1
   - C, G, J, K, Q, S, X, Z to 2
   - D,T to 3
   - L to 4
   - M, N to 5
   - R to 6
4. Repeatedly remove one out of each pair of consecutive identical digits
5. Remove all zeros from the resulting string; pad the resulting string with trailing zeros and return the first four positions, which will consist of a letter followed by three digits

33

Example: Soundex of *HERMAN*

- Retain H
- *ERMAN* → *0RM0N*
- *0RM0N* → *06505*
- *06505* → *06505*
- *06505* → *655*
- Return *H655*
- Note: *HERMANN* will generate the same code

How useful is Soundex?

- Not very – for information retrieval
- Ok for "high recall" tasks in other applications (e.g., Interpol)
- Zobel and Dart (1996) suggest better alternatives for phonetic matching in IR.

35

Take-away

- Tolerant retrieval: What to do if there is no exact match between query term and document term
- Wildcard queries
- Spelling correction

منابع

- فصل سوم کتاب An introduction to information retrieval