Information Retrieval

WS 2012 / 2013

Lecture 5, Wednesday November 21st, 2012

(Wildcard search, error-tolerant search)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

UNI FREIBURG

Overview of this lecture

- Organizational
  - Your experiences with ES#4 (compression and entropy)
- Wildcard search and error tolerant search
  - Type freib* or fr*rg or fr*b*rg, find freiburg
  - Type fraiburk, find freiburg
  - New techniques: Permuterm index, k-gram index, edit distance, Jaccard distance
  - Exercise Sheet 5: implement error-tolerant search using a k-gram index and edit distance

Experiences with ES#4 (compr. / entropy)

UNI FREIBURG

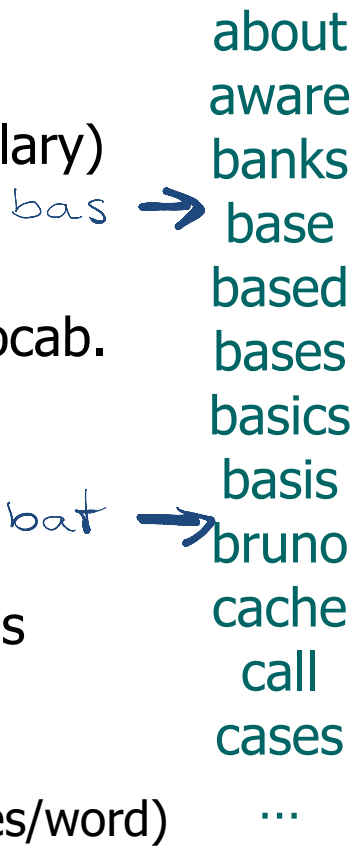
- Summary / excerpts last checked November 21, 15:13
  - Exercises 1, 2, and 3 were doable for most
  - Tricky to handle floor and ceil correctly though
  - Exercise 4 was the hardest for most ... there will be a master solution + maybe a proof sketch in the end today
  - Most of you don't like proofs it seems ... what a pity !
  - There was a mistake on the Elias-Delta slide ... fixed !
  - "Confusing slides" ... please be more explicit !
  - Why do we keep changing tutors ? ... one tutor more now

Wildcard search 1/2

- Let's start with prefix search
  - Example query: bas*
  - Locate bas using binary search (on sorted vocabulary)
  - Locate bat using another binary search
  - This takes time ~ $\log_2 n$, where n = #words in vocab.
  - For n = 100 million ≈ 2^{27} … $\log_2 n$ is 27
  - One string comparison takes ≈ 1 µsec
  - So a fraction of 1 msec even for large vocabularies

    … assuming that the vocabulary fits into memory

    Note: 100 million words take up ≈ 1GB (if 10 Bytes/word)



Wildcard search 2/2

- What if we allow the * in any place
  - Example query: ba*s
  - Should find banks, bases, basics, and basis
  - No longer a range of words!
  - Naïve approach: scan all words in the range ba* and check for each word whether it matches ba*s
  - If * is in the beginning, we have to scan the whole vocabulary, doing a string comparison for each word
  - For n = 100 million that would take 100 seconds

Permuterm Index 1/3

- For each word, add all "rotations" (not "permutations")
  - Before, append a $ to each word ... you will see why
  - Example: for base$, these rotations are

    base$, ase$b, seba, ebas, $base
  - Let each permutation point to the inverted list of the original word (the inverted lists are there only once)
  - Now for the query ba*s do a prefix search for s$ba*
  - Works for a single * in **any** position (because we can always "rotate" that * to the end)

Permuterm Index 2/3

- Efficiency in time and space
  - The vocabulary size increases by a factor of AVWL + 1

    where AVWL = average word length, typically ~ 8
  - A factor of 8 increases $\log_2 n$ by 3
  - So no problem for the locating binary searches
  - But a very large vocabulary might not fit into memory anymore
  - We would have to use a B-tree then … out of scope for this lecture

Permuterm Index 3/3

- How about more than one * ?
  - Example query: in*ma*tik
  - Simple trick: first collapse to one * as in in*tik
  - Solve this query → superset of matches … why?

    Will also find intervallarithmetik
  - Anyway, the number of matches will be relatively small
  - So just go over them, and filter out the false positives

k-Gram Index 1/4

word length = n − k + 3

- How to avoid the space blow-up of Permuterm
  - Definition: k-grams of a word = all substrings of length k
  - We now add a $ also at the beginning of each word
  - Example: the 3-grams of $informatik$ are

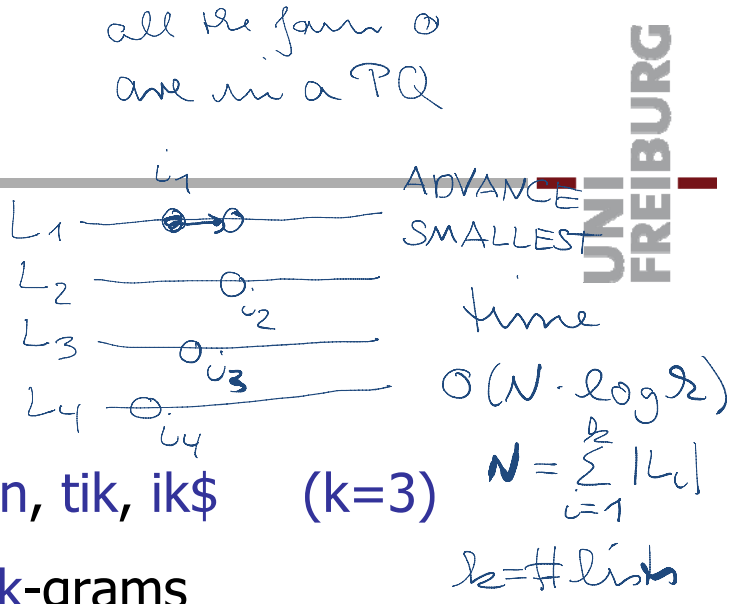
    $in, inf, nfo, for, orm, rma, mat, ati, tik, ik$
  - For each k-gram store an inverted list of the words (from our vocabulary) containing it

    $in : **in**accuracy, **in**exact, **in**formatik, **in**novate, ...

    mat : accla**mat**ion, ..., infor**mat**ik, infor**mat**ion, ...

in practice: store word ids, not string

k-Gram Index 2/4



- How to query a k-gram index
  - Example query: in*tik
  - Generate all k-grams from query: in, tik, ik (k=3)
  - Intersect the inverted lists for these k-grams

    Note for ES#5: typically more than two lists now !
  - All matching words will be included ... why?
  - But again, we can get a **super**set of results ... why?

    we would also find **in**dogermanis**tik**pickn**ik**
  - But again, result set will be small and we can just go over it and filter out the false positives

k-Gram Index 3/4

- Space efficiency
  - # of k-grams per word is AVWL – k + 3 ≈ AVWL on average
  - In the inverted lists, we store words ids, not strings
  - And have an Array<String> for mapping ids → words
  - Storing all words costs n · AVWL bytes (done anway)
  - Storing all inverted k-gram lists costs 4 · n · AVWL bytes

    provided we use 4 bytes per word id

k-Gram Index 4/4

- Time efficiency
  - Intersection of m inverted lists of total volume N

    takes time $\Theta(N \cdot \log m)$
  - Time for post-filtering depends on the specificity of the query; typically only few candidate (and final) matches
  - Compare: Time for producing candidates with Permuterm was $\Theta(\log n)$, where n = #words

Error-tolerant search 1/3

- Let's consider mistakes on the side of the query
  - Example query: innformaton retrievl
  - Should find matches for: information retrieval
  - We need an algorithm for approximate word matching:

    Given a query word (e.g. retrievl), find all similar words in a given vocabulary
  - We need a measure of similarity between words !

Edit distance 1/5

Vladimir Levenshtein
*1935, Russia

UNI FREIBURG

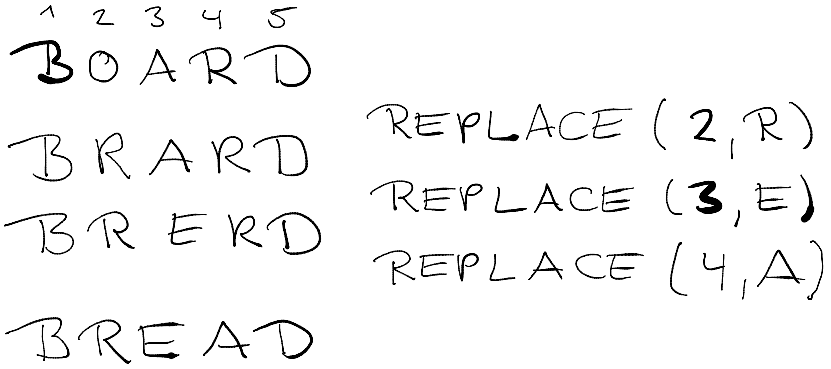
- Also known as Levenshtein distance (1965)
  - Definition: for two words / strings x and y

    ED(x, y) := minimal number of tra'fo's to get from x to y
  - Transformations allowed are:

    insert(i, c) : insert character c at position i

    delete(i) : delete character at position i

    replace(i, c) : replace character at position i by c



Edit distance 2/5

- Some notation
  - The empty word is denoted by ε
  - The length (#characters) of x is denoted by $|x|$
  - Substrings of x are denoted by $x[i..j]$, where $1 \le i \le j \le |x|$
- Some simple properties
  - $ED(x, y) = ED(y, x)$
  - $ED(x, \varepsilon) = |x|$
  - $ED(x, y) \ge abs(|x| - |y|)$ $\quad abs(z) = z \ge 0 \; ? \; z : -z$
  - $ED(x, y) \le ED(x[1..n-1], y[1..m-1]) + 1$ $\quad n = |x|, m = |y|$

Edit distance 3/5

- Recursive formula
  - For $|x| > 0$ and $|y| > 0$, $ED(x, y)$ is the minimum of

    (1a) $ED(x[1..n], y[1..m-1]) + 1$

    (1b) $ED(x[1..n-1], y[1..m]) + 1$

    (1c) $ED(x[1..n-1], y[1..m-1]) + 1$ if $x[n] \neq y[m]$

    (2) $ED(x[1..n-1], y[1..m-1])$ if $x[n] = y[m]$
  - For $|x| = 0$ we have $ED(x, y) = |y|$
  - For $|y| = 0$ we have $ED(x, y) = |x|$

Edit distance 4/5

- Proof sketch
  - Consider a sequence of $k = ED(x, y)$ tra'fo's from x to y
  - Turn it into a monotone sequence, that is:

    Positions of operations never decrease, and, except for successive deletions, strictly increase ... why possible?
  - Consider the last tra'fo $\sigma_k : z \rightarrow y$ in this sequence:

    If σ_k appends a char to z ... then $ED(x, y) = (1a)$

    If σ_k removes last char of z ... then $ED(x, y) = (1b)$

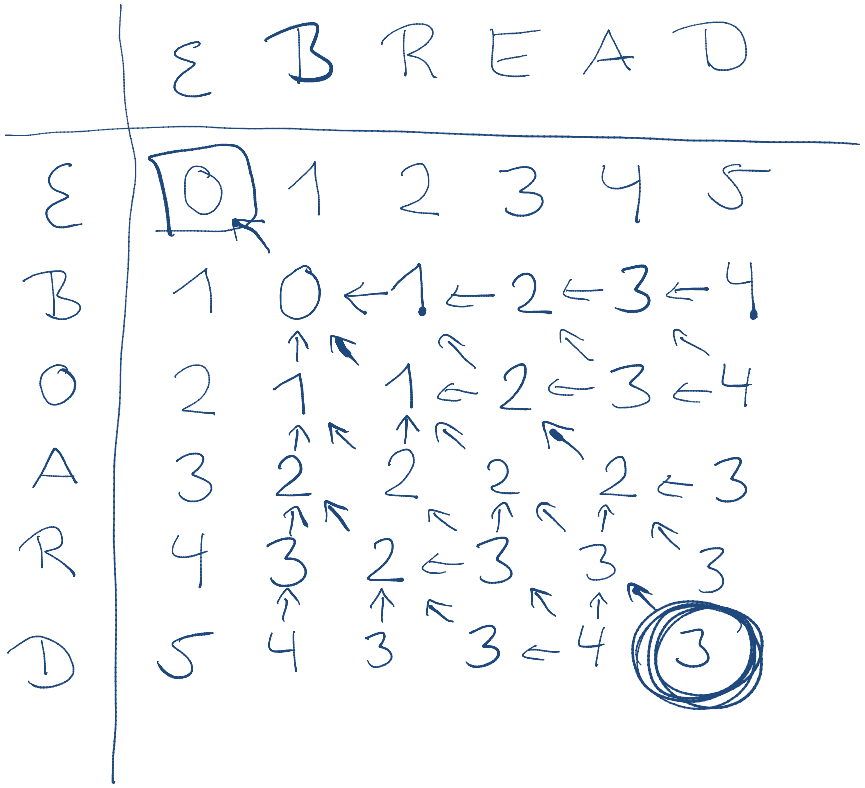
    If σ_k replaces last char of z ... then $ED(x, y) = (1c)$

    If σ_k leaves last char of z as is ... then $ED(x, y) = (2)$

Edit distance 5/5

Space can be improved to $O(\min(|x|, |y|))$ if sequence of trafos not needed

- Dynamic programming algorithm
  - Takes time and space $\Theta(|x| \cdot |y|)$



"dynamic Programming"

following the ← from ○ to □ gives you all possible MONOTONE sequences of trafo's

Error-tolerant search 2/3

- Approximate word matching
  - Definition: Given a query word w, a vocabulary V, and a treshold δ ... find all words v in V with ED(w, v) ≤ δ
  - Naive algorithm: compute ED(w, v) for each word in V
  - Need around 1µs / ED computation ... see Exercise 5.4 !
  - That's 1 second for each 1M words in the vocabulary
  - Note: from the Linux command line, you can do:

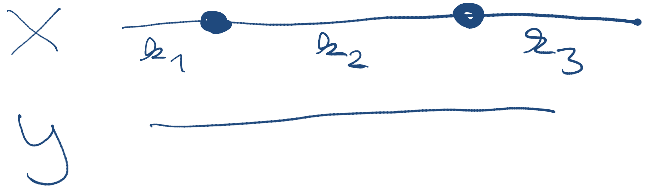
    agrep -2 –w retrievl wikipedia-sentences.vocabulary.txt

    The -2 means δ = 2, the –w means whole word match

Error-tolerant search 2/3

- Using a Permuterm index
  - Consider x and y with ED(x, y) ≤ δ
  - Intuitively: if x and y are not too short, they will have a substring of significant length in common
  - Lemma: there exist rotations x' of x and y' of y such that x' and y' have a common prefix of size

    ceil(max(|x|, |y|) / δ) - 1



$\delta = 2$

worst case

$l_1 + l_3 = l_2$

Error-tolerant search 2/3

UNI FREIBURG

- Using a k-gram index **slide corrected: 23Nov12 00:39**
  - Consider x and y with $ED(x, y) \leq \delta$
  - Intuitively: if x and y are not too short, they will have one or more k-grams in common
  - Lemma: let x' and y' be x and y with k-1 # padded **left and right**, then the number of common k-grams of x' and y' is $comm_k(x',y') \geq \mathbf{max(|x|,|y|) - 1 - (\delta - 1) \cdot k}$
  - Proof sketch: consider the longer string, which has $max(|x|,|y|) + k - 1$ k-grams (because of the padding); then one tra'fo (insert/delete/replace) "affects" at most k k-grams, and hence δ tra'fos affect at most δ·k k-grams
  - Example: $|x| = 5$, $|y| = 4$, $k = 3$, $\delta = 2$, $comm_k(x',y') \geq \mathbf{1}$

    x' = ##SILLY## k-grams: ##S #SI SIL **ILL** LLY LY# Y##

    y' = ##BILL## k-grams: ##B #BI BIL **ILL** LL# L##

Error-tolerant search 2/3

Paul Jaccard
*1868 Sainte-Croix
†1944 Zürich

- Jaccard distance
  - Actually, a k-gram index would more naturally give all words within a given **Jaccard distance**
  - Definition: the Jaccard co-efficient of two sets A and B is defined as $J(A, B) = |A \cap B| / |A \cup B|$
  - Definition: the (k-gram) Jaccard distance of two strings x and y is defined as $J_k(x, y) = J(A, B)$

    where A and B are the sets of k-grams of x and y (no # or $)
  - But does not capture intuitive word similarity well
  - Example 1: J_2("weigh", "weihg") = 2/6 = 1/3 too low
  - Example 2: J_2("aster", "terase") = 3/6 = 1/2 too high

Advanced stuff 1/2

- Generalized edit distance
  - Some changes in words happen more easily than others
  - Example 1: ED("weigh", "weihg") = 2
    ED("weigh", "eight") = 2
  - Example 2: ED("chebyshev", "tschebyscheff") = 5
    ED("chebyshev", "webster") = 5
  - Generalized edit distance: have individual costs for different substring transformations, for example:

    cost("ch" → "tsch") = 0.1
    cost("v" → "w") = 0.1
    cost("x" → "u") = 1

Advanced stuff 2/2

UNI FREIBURG

- Query suggestion
  - Example query: innformaton retrievl
  - Answer: Did you mean "information retrieval" ?
  - Simple solution: find the most frequent similar word for each query word ... would work for the example above
  - But what about: freiberger münster
  - Problem: freiberger is also a correct word, but most probably freiburger was meant here
  - Ideas: check which combination of words retrieves most hits ... or occurs most often in the query logs ... or both

References

- In the Raghavan/Manning/Schütze textbook

  Section 3: Tolerant Retrieval, in particular

  Section 3.2: Wildcard queries

  Section 3.3: Spelling correction

- Relevant Wikipedia articles

  http://en.wikipedia.org/wiki/N-gram

  http://en.wikipedia.org/wiki/Approximate_string_matching

  http://en.wikipedia.org/wiki/Levenshtein_distance

  http://en.wikipedia.org/wiki/Jaccard_index