Information Retrieval

WS 2013 / 2014

Lecture 5, Tuesday November 19th, 2013
(Fuzzy Search, Edit Distance, q-Gram Index)

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

Overview of this lecture

- Organizational
  - Your experiences with ES#4 (compression and entropy)
- Fuzzy search
  - So far, exact matches: type university, find university
  - Fuzzy search: type uni or uniwercity, find university
  - Similarity measure: (prefix) edit distance
  - New data structure: q-gram index

  Exercise Sheet 5: implement error-tolerant prefix search using a q-gram index and prefix edit distance

Experiences with ES#4 (compr. / entropy)

- Summary / excerpts   last checked November 19, 02:00
  - Harder and more time-consuming than previous sheets

    We tried hard to keep the effort reasonable for you though

    Seems the bit fiddling cost many of you some time
  - Wrong hint in Exercise 2 cost some of you time

    Needed: $1 - p \leq e^{p}$; Provided: $1 - p \geq e^{p/2}$

    SORRY! Was corrected on the forum though
  - Golomb ↔ Gollum … exactly !

Results for ES#4 (compression / entropy)

- Summary
  - Compression ratio (G = Gollum, VB = Variable-Byte)

    On the **long dense list** ("american", 165K elements)

    G compresses almost twice as good as VB (ratio 7.9 vs 4.0)

    On the **short sparse list** ("freiburg", 310 elements)

    G compresses only slightly better than VB (ratio 2.3 vs. 2.1)
  - Time for compression / decompression

    G compression ~ 10 times slower than VB

    G decompression ~ 3 times slower than VB

    Use Gollum only when you don't get the ring otherwise

Fuzzy Search 1/7

- Motivation and problem setting
  - Problem setting in the lectures so far:

    Given a query, find relevant documents for that query
  - Problem setting in the lecture **today**:

    Given a query, or part of a query, suggest a "matching" string or strings from a given (typically large) input set

    Given: uni match: university (**prefix** search)

    Given: uni*ty match: university (**wildcard** search)

    Given: univerty match: university (**error-tolerant** search)

    Of course, there could be more than one match, for example, uni*ty also matches unidimensionality

Fuzzy Search 2/7

- Some possible origins for the input set
  - Popular queries extracted from a query log

    This is the basis of Google's auto-completion feature
  - Words + common phrases from a text collection

    Extracting common phrases from a given text collection is an interesting problem by itself, however, not one we will deal with in this course
  - A list of names of entities (people, places, things, …)

    Your input set for ES5 will be a selection of ~ 8 million entity names from Freebase (www.freebase.com)

Fuzzy Search 3/7

- Matching vs. Search
  - Once we have found a "matching" string or strings, we can do an exact search like before, for example:

    1. Type: uni

    2. Match: universe, university

    3. Search: universe OR university

    In todays lecture, we will only look at parts 1 + 2 = finding matching strings in the input set

    The search part is also interesting, when the number of matching strings is very large; then a simple OR of a lot of strings will be too slow and we need better solutions

Fuzzy Search 4/7

- Simple solution
  - Go over all strings in the input set, and for each check whether it matches
  - This is what the Linux commands **grep** and **agrep** do

    grep –x uni.* <file>

    grep –x un.*ity <file>

    agrep –x –2 univerty <file>

    All matching lines in <file> will be output

    The option –x means match whole line (not just a part)

    The option –2 means allow up to two errors

Fuzzy Search 5/7

- How to check whether a single string matches
  - Given a query q and a string s
  - **Prefix search:** easy-peasy

    Just compare q and the first $|q|$ characters of s
  - **Wildcard search**: also easy if only one *

    If $q = q_1 * q_2$, check that $|s| > |q_1| + |q_2|$ and then compare the first $|q_1|$ characters of s with q_1 and the last $|q_2|$ characters of s with q_2
  - **Error-tolerant search:** not so easy

    We need to define a similarity measure between strings, and then compute it; we will take edit distance … slides 11 - 17

Fuzzy Search 6/7

- Complexity
  - The time complexity is obviously $n \cdot T$, where
    n = #words, T = time for checking a single string
  - For the searches from the previous slide T ranges from:
    0.1µs for wildcard search to 1µs for error-tolerant search
  - In search, we always want interactive query times
    respond times feel interactive until about **100ms**
  - So the simple solution is fine for up to 100K - 1M words
  - For larger input sets, we need to pre-compute something
    We will build a so-called **q-gram index** … slides 18 – 24

Fuzzy Search 7/7

bas*

- For prefix search, there is a faster solution
  - Assume the input strings are in sorted order:

    Then we can find the first match for a prefix with $\lceil \log_2 n \rceil$ string comparisons using a binary search
  - This is fast enough also for very large values of n

    Example: $n = 1 \text{ Tera} = 10^{12} \approx 2^{36}$

    Then: $\log_2 n = 36$

about
aware
banks
base
based
bases
basics
basis
bruno
cache
call
cases
...

Edit distance 1/7

Vladimir
Levenshtein
*1935, Russia

- Also known as Levenshtein distance (1965)
  - Definition: for two 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

D O O F
↓ REPLACE(1, B)
B O O F
↓ REPLACE(2, L)
B L O F
↓ INSERT(4, E)
B L O E F
↓ REPLACE(5, D)
B L O E D

Edit distance 2/7

- 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
  - $ED(x, y) = ED(y, x)$
  - $ED(x, \varepsilon) = |x|$
  - $ED(x, y) \ge abs(|x| - |y|)$ $\quad$ $abs(z) = z \ge 0 \ ? \ z : -z$
  - $ED(x, y) \le ED(x[1..n-1], y[1..m-1]) + 1$ $\quad$ $n = |x|, m = |y|$

DOOF BLOED DOO BLOE

DOOF BLOEF DOO BLOE

Edit distance 3/7

- 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 4/7

- Proof sketch
  - Consider a sequence of $k = ED(x, y)$ tra'fo's from x to y
  - There is always a **monotone** such sequence … **verify**

    Monotone = positions of operations never decrease, and, except for successive deletions, strictly increase
  - 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 5/7

- Algorithm for computing ED(x, y)
  - The recursive formula from Slide 11 naturally leads to the following dynamic programming algorithm
  - Takes time and space $\Theta(|x| \cdot |y|)$

PREFIXES →

	ε	B	L	O	E	D
ε	0	1	2	3	4	5
D	1	1	2	3	4	4
O	2	2	2	2	3	4
O	3	3	3	2	3	4
F	4	4	4	3	3	4

PREFIXES ↓

ED(ε, BLO) = 3

ED(DOOF, BLOED) = 4

Edit distance 6/7

- An interesting variation: **prefix** edit distance
  - The prefix edit distance between x and y is defined as

    $PED(x, y) = \min_{y'} ED(x, y')$ where y' is a prefix of y
  - For example

    PED(uni, university) = 0 ... but ED = 7

    PED(uniwer, university) = 1 ... but ED = 5

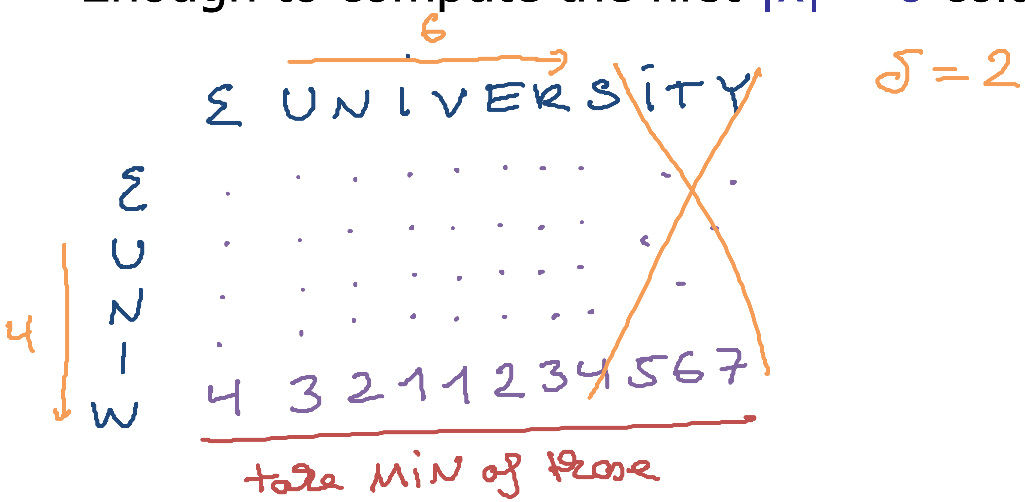
    Important for error-tolerant query suggestions as you know them from Google

    There you get error-tolerant **completions** as you type, that is, already for prefixes of your query

Edit distance 7/7

- Computation of the PED
  - Compute the entries of the |x| · |y| table, just as for ED
  - The PED is just the minimum of the entries in the last row
  - Important optimization when |x| << |y| and you only want to know if PED(x, y) ≤ δ for some given δ:

    Enough to compute the first |x| + δ columns … **verify!**



q-Gram Index 1/7

- Index construction
  - **Definition:** q-grams of a string = all substrings of length q
  - For wildcard search, add a $ before and after each string

    For error-tolerant search, we will add the $s a little differently
  - **Example:** the 3-grams of $university$ are

    $un, uni, niv, ive, ver, ers, rsi, sit, ity, ty$
  - For each q-gram store an inverted list of the strings (from the input set) containing it, sorted lexicographically

    $un : **un**animous, **un**expected, **un**iversity, **un**nötig, …

    ers : aargau**ers**traße, …, univ**ers**ity, unv**ers**tändlich, …

    As usual, store **ids** of the strings, not the strings themselves

q-Gram Index 2/7

- Space consumption
  - For $q = 3$, the number of q-grams for x is exactly $|x|$

    Each x thus contributes $|x|$ ids to the inverted lists
  - The total number of ids in the lists is hence $4 \cdot N$, where N is the size of the input file
  - We also need to store the input strings in an array, so that we can map ids back to strings again
  - Hence: **total size = five times the input file**

    Note that we could reduce this using compression

    For ES5, it is fine if you store the lists uncompressed

q-Gram Index 3/7

- Wildcard search (single *)
  - Example query: un*ity
  - Generate all q-grams from query: un, ity, ty (q=3)
  - Take **intersection** of inverted lists for these q-grams
  - Each matching string from the input set will be included

    If it matches, it also contains the q-grams from the query
  - However, not all strings in the intersection are matches

    For example: **un**ivers**ity**facul**ty**
  - Go over each string in intersection and check if it matches

    In the simple algorithm, we do this for **every** input string

q-Gram Index 4/7

UNI FREIBURG

- Error-tolerant search, Preliminaries
  - Consider x and y with $ED(x, y) \leq \delta$
  - Intuitively: if x and y are not too short, and δ is not too large, they will have one or more q-grams in common
  - Let x' and y' be x and y with $q - 1$ times \$ left and right

    Otherwise, fewer q-grams containing the first / last letters !
  - Example: $|x| = 5$, $|y| = 4$, $q = 3$, $\delta = 2$

    x' = \$\$KILL\$\$ 3-grams: \$\$K \$KI KIL ILL LL\$ L\$\$

    y' = \$\$BILLY\$\$ 3-grams: \$\$B \$BI BIL ILL LLY LY\$ Y\$\$

    Number of q-grams in common is: comm(x', y') = **1**

    Lemma: $\text{comm}(x', y') \geq \mathbf{max(|x|, |y|) - 1 - (\delta - 1) \cdot q}$

    x=KILL y=BILLY 4 5 5 1 3

q-Gram Index 5/7

- Error-tolerant search, Proof sketch of Lemma
  - **Lemma:** comm(x', y') ≥ max(|x|, |y|) – 1 – (δ – 1) · q

    Repetition of example: |x| = 5, |y| = 4, q = 3, δ = 2

    x' = $$KILL$$ 3-grams: $$K $KI KIL ILL LL$ L$$

    y' = $$BILLY$$ 3-grams: $$B $BI BIL ILL LLY LY$ Y$$

  - **Proof sketch:**

    Consider the longer string, which has max(|x|, |y|) + q – 1 q-grams (because of the left and right $ padding)

    Then one tra'fo (insert / delete / replace) changes at most q q-grams, and hence δ tra'fos affect at most δ · q q-grams

q-Gram Index 6/7

- Error-tolerant search, Query Algorithm
  - Given a query x and a q-gram index for the input strings
  - Compute q-grams of x' and fetch their inverted lists

    For example: x = BILL, x' = $$BILL$$

    Fetch lists for: $$B , BI , BIL , ILL , LL , L$$
  - Compute the **union** of these inverted lists + beware this:

    Keep duplicates in the union or maintain a count for each id
  - For each elem y with count $\geq \max(|x|, |y|) - 1 - (\delta - 1) \cdot q$:

    Compute the actual ED using the dynamic programming algo

    For the ids with lower counts, we know that ED > δ

q-Gram Index 7/7

- Error-tolerant **prefix** search
  - Use the same algorithm, but with a different bound
  - Assume that **P**ED(x, y) $\leq \delta$
  - Let x' and y' be x and y with $q - 1$ times \$ to the **left only**
  - **Lemma:** then comm(x', y') $\geq$ **$|x| - q \cdot \delta$**

    Note that for $\delta = 1$, this is ≥ 1 only for $|x| > q$
  - **Proof sketch:** consider x, which has exactly $|x|$ q-grams, then one tra'fo (insert / delete / replace) changes at most q q-grams, and hence δ tra'fos change at most $\delta \cdot q$ q-grams

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