Information Retrieval

WS 2017 / 2018

Lecture 5, Tuesday November 21st, 2017

(Fuzzy Search, Edit Distance, q-Gram Index)

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

UNI FREIBURG

Overview of this lecture

- Organizational
  - Experiences with ES4 — Compression, Codes, Entropy
  - Date of written exam — **Mon Feb 19** or **Tue Feb 20** ?
- Contents
  - Fuzzy search — type breifurg, find freiburg
  - Edit Distance — a standard similarity measure
  - Q-gram Index — index for efficient fuzzy search

  ES5: implement error-tolerant prefix search using a q-gram index and prefix edit distance

  We have compiled a beautiful new dataset for you:
  2.9M entities from Wikidata with scores and descriptions

Experiences with ES4 1/3

- Summary / excerpts
  - None of you found the exercises easy at first glance
  - But some of you sat down and eventually figured it out and realized that in retrospect it wasn't that hard

    "Looking back now the exercises were actually easy. But I think that's always the case when doing math exercises :)"
  - Some of you gave up or did only a part of the exercises

    "I am not a fan of this exercise sheet"
  - First time use of LaTeX for some ... don't worry, it's worth it!
  - In Exercise 3, confusing to use L_i for the length of the i-th inverted list, when we used it for the i-th code length before

Experiences with ES4 2/3

- Favorite movies watched in 2017 (selection)
  - One night in Paris ... "why? because of the romance"
  - Toy Story 2 ... "because it's very funny"
  - Captain Fantastic ... "not the type of movie I usually watch, but it made me reconsider"
  - Blade Runner 2049 ... "I totally expected Hollywood to ruin it but I was pleasantly surprised"
  - La La Land ... several mentions
  - The Hobbit ... "the scene with Gollum (not Golomb) was great"
  - Fack ju Göhte 3 ... "I really enjoy feeling my own IQ decrease constantly while watching movies like this one. Maybe that's the reason why this sheet took me so long to complete."

Experiences with ES4 3/3

$\lceil x \rceil \le x + 1$

$L_i = \lfloor i/M \rfloor + 1 + \lceil \log_2 M \rceil$

$M = \lceil \frac{\ln 2}{p} \rceil \le \frac{\ln 2}{p} + 1$

- Proof sketch for Exercise 4.2
  - Show that "Gollum" encoding with modulus $M = \lceil \ln 2/p \rceil$ is optimal if the prob. for number i is $p_i = (1-p)^{i-1} \cdot p$

we need to show: $L_i \le \log_2 \frac{1}{p_i} + O(1)$

(I) $\log_2 \frac{1}{p_i} = \log_2 \frac{1}{(1-p)^{i-1}} \cdot \frac{1}{p} = (i-1) \cdot \log_2 \frac{1}{1-p} + \log_2 \frac{1}{p}$

(II) $L_i \le \frac{i \cdot p}{\ln 2} + 1 + \log_2 \left(\underbrace{\underbrace{\frac{\ln 2}{p} + 1}_{\le 2 \cdot \frac{\ln 2}{p}}) + 1}_{\le 3 \cdot \frac{\ln 2}{p}}$

$= \frac{(i-1)p}{\ln 2} + \underbrace{\frac{p}{\ln 2}}_{\le 2}$

$\le \underbrace{\frac{(i-1) \cdot p}{\ln 2}}_{\le (i-1) \cdot \log_2 \frac{1}{1-p}} + \log_2 \frac{1}{p} + 4 + \underbrace{\log_2 (3 \ln 2)}_{\le 2}$

$\le \log_2 \frac{1}{p_i} + 6$

$\ln 2 = 0.69\ldots$

$2 \cdot \ln 2 \ge 1$

$\frac{2 \cdot \ln 2}{p} \ge \frac{1}{p} \ge 1$

HINT:

$1 + x \le e^x \quad \forall x$

$1 - p \le e^{-p}$

$\frac{1}{1-p} \ge e^p$

$\ln \frac{1}{1-p} \ge p$

$\log_2 \frac{1}{1-p} = \frac{\ln \frac{1}{1-p}}{\ln 2} \ge \frac{p}{\ln 2}$

Fuzzy Search 1/6

- Problem setting
  - Given a "dictionary" = a list of "names" of any kind

    For ES5: a list of **2,920,180** entities from Wikidata

  - For a given query, find matching names from that dict.

    Query: frei Match: freiburg **prefix** search
    Query: fr*rg Match: freiburg **wildcard** search
    Query: breifurg Match: freiburg **fuzzy** search

  - Similar challenges as for our search so far:

    Challenge 1: good model of what **matches**

    Challenge 2: preprocess the input (= build a suitable index), so that we find the matching names **fast**

Fuzzy Search 2/6

- Possible origins for the dictionary
  - Popular queries extracted from a query log

    Basis for Google's query-suggestion 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

    For example: person names, movie titles, places, street addresses, …

Fuzzy Search 3/6

- Combining matching and search
  - One could simply search for the top match, for example:

    Type: freib   Search: freiburg

  - Or one could search for several matches

    Type: freib   Search: freiburg OR freibach OR … OR …

  - In todays lecture, we will only look at the problem of finding matching names in a list of names

    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 4/6

- Simple solution
  - Iterate over all strings in the dictionary, 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" ... next slide

Fuzzy Search 5/6

- Simple solution, check match of single string
  - Given a query q and a string s
  - **Prefix search: easy-peasy**

    Just compare q and the first |q| characters of s … can be accelerated by finding the first match with a binary search
  - **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
  - **Fuzzy search: more complicated**

    Compute edit distance between q and s … slides 12 – 17

Fuzzy Search 6/6

- Simple solution, time complexity
  - The time complexity is obviously $n \cdot T$, where

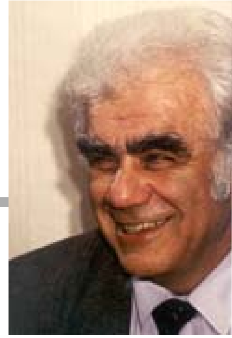
    n = #records, T = time for checking a single string
  - For fuzzy search, T ≈ 1µs ... find out yourself in ES5
  - In search, we always want interactive query times

    Respond times feel interactive until about **100ms**
  - So the simple solution is fine for up to ≈ 100K records
  - For larger input sets, we need to pre-compute something

    We will build a **q-gram index** ... slides 18 – 25

Edit distance 1/6

Vladimir
Levenshtein
1935 - 2017



- Definition ... aka Levenshtein distance, from 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

x = DOOF
BOOF REPLACE(1, B)
BLOOF INSERT(2, L)
BLOEF REPLACE(4, E)
y = BLOED REPLACE(5, D)

IMPORTANT:
This only proves
ED(x, y) $\leq$ 4

Edit distance 2/6

- Some simple 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 $1 \leq i \leq j \leq |x|$
- Some simple properties
  - $ED(x, y) = ED(y, x)$
  - $ED(x, \varepsilon) = |x|$
  - $ED(x, y) \geq abs(|x| - |y|)$ $\qquad abs(z) = z \geq 0 \; ? \; z : -z$
  - $ED(x, y) \leq ED(x[1..n-1], y[1..m-1]) + 1$ $\qquad n = |x|, m = |y|$

$\underbrace{\text{DOOF} \quad \text{BLOED}}_{=4} \quad \underbrace{\text{DOO} \quad \text{BLOE}}_{=3}$

Edit distance 3/6

NOTE:
Recursive implementation of this is NOT a good idea (takes $3^{|x|}$ time)

- 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 $ED(x, y) = |x|$

For a proof of that formula, see e.g. "Algorithmen und Datenstrukturen" SS 2017, Lecture 11a, slides 14 – 20

Edit distance 4/6

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

y →

x ↓

	ε	B	L	O	E	D
ε	0	1	2	$\boxed{3}$	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	$\boxed{4}$

ED(ε, BLO)

ED(DOOF, BLOED)

Edit distance 5/6

- **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 fuzzy search-as-you type suggestions

    By now, all the large web search engines have this feature, because it is so convenient for usability

Edit distance 6/6

PED (FIBU, FREIBURG) = 2

- 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 + 1 columns … **verify !**

δ = 2

	ε	F	R	E	I	B	U	R	G
ε	0	1	2	3	4	5	6	7	8
F	1								
I	2			. . . .					
B	3								
U	4	$\boxed{3}$	3	3	3	$\boxed{3}$	$\boxed{2}$	3	4

we can STOP here because for |y'| > |x| + δ ⇒ ED(x, y') > δ

ED(FIBU, F)

ED(FIBU, FREIB)

ED(FIBU, FREIBU)

q-Gram Index 1/8

- Definition of a q-gram
  - The q-grams of a string are simply all substrings of length q

    freiburg: fre, rei, eib, ibu, bur, urg

    q = 3
  - The number of q-grams of a string x is exactly |x| - q + 1

    8 - 3 + 1 = 6
  - For fuzzy search, we will **pad** the string with q – 1 special symbols (we use $) in the beginning and in the end

    freiburg → $$freiburg$$

    3-grams: $$f, $fr, fre, rei, eib, ibu, bur, urg, rg$, g$$
  - The number of q-grams is then |x| + q – 1, where x is the original string (the padding adds q – 1 q-grams per side)

    We will see in a minute, why that padding is useful

q-Gram Index 2/8

- Definition of a q-gram index
  - For each q-gram store an inverted list of the strings (from the input set) containing it, sorted lexicographically

    $fr : **fr**aberg, **fr**allach, **fr**eiburg, **fr**eiberg, **fr**ouville, …

    ibu : b**ibu**rg, fre**ibu**rg, garc**ibu**ey, se**ibu**ttendorf, …

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

  - A q-gram index for a collection of words is also an inverted index, just with different objects than a document index:

    document index : lists of **doc ids**, one for each **word**
    q-gram index : lists of **words ids**, one for each **q-gram**

  - Let's now adapt our code from Lecture 1 to q-grams …

q-Gram Index 3/8

- Space consumption
  - Each record x contributes |x| + Q ids to the inverted lists, where Q is the total number of padding characters

    For ES5, we have one-sided padding and q=3, hence Q=2
  - The **total number of ids** in the inverted lists is hence:

    #chars + Q · #words = #chars · (1 + Q / AVWL)
  - With 4 bytes per id, Q = 2 and AVWL = 8, the **total size** of the inverted lists in bytes would hence be 5 · #chars

    Note: for 1 byte per char, #chars is the size of the input
  - This can be reduced significantly using **compression**

    For ES5, it is fine to store the lists uncompressed

q-Gram Index 4/8

- Fuzzy search with a q-gram index, using ED ... part 1
  - Consider x and y with ED(x, y) = δ
  - 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
  - Example: x = HILLARY, y = HILARI, q = 3, δ = 2

    $$HILLARY$$ → $$H, $HI, HIL, ILL, LLA, LAR, ARY, RY$, Y$$

    $$HILARI$$ → $$H, $HI, HIL, ILA, LAR, ARI, RI$, I$$

    Number of q-grams in common: 4
  - Note: the padding in the beginning gives us two additional 3-grams in common here (because no mistake in first letter)

    The more q-grams in common, the more efficient the query algorithm on slide 22 will work

q-Gram Index 5/8

- Fuzzy search with a q-gram index, using ED … part 2
  - Formally: let x' and y' be the padded versions of x and y

    Then: comm(x', y') ≥ max(|x|, |y|) – 1 – (δ – 1) · q

    HILLARY HILARI

    Example from slide before: |x| = 7, |y| = 6, δ = 2, q = 3

    Hence: comm(x', y') ≥ max (7, 6) – 1 – 1 · 3 = 3

    In the example, actually: comm(x', y') = 4

    Verify: in the worst case, comm(x', y') = 3 can happen

  - **Proof:** 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) change at most q q-grams, and hence δ tra'fos change at most δ · q q-grams

q-Gram Index 6/8

- Fuzzy search with a q-gram index, using ED … part 3
  - 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 = HILARI, x' = $$HILARI$$

    Fetch lists for: $$H, $HI, HIL, ILA, LAR, ARI, RI$, I$$
  - Merge these lists and for each word in the union keep a **count** of the number of lists in which it is contained, for example:

HILLARY:	4	(contains $$H, $HI, HIL, LAR)
HAEMOPHILIA:	2	(contains $$H and HIL)
SOLAR:	1	(contains only LAR)

    This step considers each word that contains at least one of the q-grams, which is usually many more than actually match

q-Gram Index 7/8

- Fuzzy search with a q-gram index, using ED … part 4
  - For each record y in the merge results, check whether the count is $\geq \max(|x|, |y|) - 1 - (\delta - 1) \cdot q$

    NO: discard this y, we know that $ED(x, y) > \delta$

    YES: compute $ED(x, y)$ and check if $ED(x, y) \leq \delta$
  - Let's continue our example

    x = HILARY, $q = 3$, $\delta = 2$ … hence $-1 - (\delta - 1) \cdot q = -4$

    y_1 = HILLARY: 4 $\max(|x|, |y_1|) - 4 = 3$ → YES
    y_2 = HAEMOPHILIA: 2 $\max(|x|, |y_2|) - 4 = 6$ → NO
    y_3 = SOLAR: 1 $\max(|x|, |y_3|) - 4 = 2$ → NO
  - So for this example, we only have to compute ED(HILARY, HILLARY) … which is 2, hence HILLARY is output as a match

q-Gram Index 8/8

- Changes when using the **PED** … needed for ES5
  - We 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** (padding on the right makes no sense for prefix search)
  - Then we have: comm(x', y') $\geq$ $\mathbf{|x| - q \cdot \delta}$

    Note that for $\delta = 1$, this is ≥ 1 only for $|x| > q$
  - **Proof:** 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-gram

References

- Textbook
  - Section 3: Tolerant Retrieval, in particular:
  - Section 3.2: Wildcard queries
  - Section 3.3: Spelling correction
- Wikipedia
  - http://en.wikipedia.org/wiki/N-gram
  - http://en.wikipedia.org/wiki/Approximate_string_matching
  - http://en.wikipedia.org/wiki/Levenshtein_distance