Information Retrieval

WS 2015 / 2016

Lecture 5, Tuesday November 17th, 2015
(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
  - Experiences with ES4 — Compression, Codes, Entropy
- Contents
  - Fuzzy search — type uniwercity, find university
  - Edit Distance — a standard similarity measure
  - Q-gram Index — index for efficient fuzzy search

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

Experiences with ES4 1/4

- Summary / excerpts
  - Few liked it, for some it was OK, many didn't like it

    "Compared to sheets 1 – 3, this sheet was no fun"

    Only 20% theory in this course, but that's necessary
  - Confusion between "code" and "codeword"
  - Please explain Lagrange optimization in the lecture

    I will sketch the solutions to ES4 on the next slides

    As usual, master solutions in the SVN + on the Wiki
  - First steps with LaTeX cost some of you some time
  - Breakfast: Muesli, Currywurst, neighbor's cat, nothing, …

Experiences with ES4 2/4

- Proof sketch for Exercise 4.2
  - To show: no prefix-free code with length $\log_2 x + O(1)$

Assume $L_i \leq \log_2 i + A$ for some A.

$$\Rightarrow 2^{-L_i} \geq \underbrace{2^{-\log_2 i}}_{=1/i} \cdot 2^{-A} = 1/i \cdot 2^{-A}$$

$$\Rightarrow \sum_{i=1}^{n} 2^{-L_i} \geq \underbrace{\sum_{i=1}^{n} \frac{1}{i}}_{\approx \ln n \xrightarrow[n \to \infty]{} \infty} \cdot 2^{-A}$$

But $\sum_i 2^{-L_i}$ has to be ≤ 1 for PF code

Experiences with ES4 3/4

- Proof sketch for Exercise 4.3
  - To show: Golomb is optimal for $p_i = (1-p)^{i-1} \cdot p$
  - For that we have to show that: $L_i \leq -\log_2 p_i + O(1)$

the leading 0s

the 1

the remainder

Golomb

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

for this p_i

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

$c := \ln 2$

$M = c \cdot \frac{1}{p}$

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

$$\Rightarrow L_i = i/M + \log_2 M + O(1)$$
$$= \frac{1}{c} \cdot i \cdot p + \underbrace{\log_2 c \cdot \frac{1}{p}}_{= \log_2 c + \log_2 \frac{1}{p}} + O(1)$$
$$= \frac{1}{c}(i-1) \cdot p + \log_2 \frac{1}{p} + O(1)$$
$$= -(i-1) \cdot \log_2 (1-p) + \log_2 \frac{1}{p} + O(1)$$
$$= -\log_2 (1-p)^{i-1} \cdot p + O(1)$$

$\log_2 (1-p)$

$(1-p) \approx e^{-p}$
$= e^{-\ln 2 \cdot \frac{p}{\ln 2}}$
$= 2^{-\frac{p}{\ln 2}}$

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

Experiences with ES4 4/4

e.g. $\mathcal{L} = p_1 \cdot L_1 + p_2 \cdot L_2 + p_3 \cdot L_3 + \dots$ const. w.r.t. L_2

$\frac{\partial \mathcal{L}}{\partial L_2} = p_2$

$\sum_{i=1}^{m} 2^{-L_i} - 1 = 0$ that's what you plug into the $\mathcal{L}$

$2^{-L_i} = e^{-\ln 2 \cdot L_i}$

$[2^{-L_i}]' = -\ln 2 \cdot 2^{-L_i}$

- Proof sketch for Exercise 4.4
  - To show: $\Sigma_i\, p_i \cdot L_i \geq -\Sigma_i\, p_i \cdot \log_2 p_i$ when $\Sigma_i\, 2^{-L_i} \leq 1$
  - Let us assume $\Sigma_i\, 2^{-L_i} = 1$ … generally: $\Sigma_i\, 2^{-L_i} =: A \leq 1$

  1. Define $\mathcal{L}(L_1, \dots, L_n, \lambda) = \Sigma_i\, p_i \cdot L_i + \lambda \cdot (\Sigma_i\, 2^{-L_i} - 1)$
  2. Set partial derivatives = 0 to find all local optima
  3. Only one local optimum → also global optimum

$$\frac{\partial \mathcal{L}}{\partial L_i} = p_i - \lambda \cdot \ln 2 \cdot 2^{-L_i} \overset{!}{=} 0 \Rightarrow \underbrace{\lambda \cdot \ln 2}_{=1} \cdot 2^{-L_i} = p_i$$

$$\Rightarrow \lambda \cdot \ln 2 \underbrace{\sum_{i=1}^{m} 2^{-L_i}}_{=1} = \underbrace{\sum_{i=1}^{m} p_i}_{=1} \Rightarrow \lambda \cdot \ln 2 = 1$$

$$\Rightarrow 2^{-L_i} = p_i \Rightarrow L_i = \log_2 \frac{1}{p_i} \quad \blacksquare$$

Fuzzy Search 1/7

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

    Given a document collection and a query, find documents relevant for the query

  - Two main challenges:

    Challenge 1: good model of **relevance**

    Challenge 2: preprocess the document collection (= build a suitable index), so that queries can be answered **fast**

Fuzzy Search 2/7

- Motivation and problem setting
  - Problem setting in the lecture today:

    Given a dictionary and a query, or part of a query, suggest matching items from that dictionary … for example:

    Query: uni Match: university **prefix** search
    Query: uni*ty Match: university **wildcard** search
    Query: univerty Match: university **fuzzy** search

  - For fuzzy search, we have the same two challenges:

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

    Challenge 2: preprocess the dictionary (= build a suitable index), so that we find those matches **fast**

Fuzzy Search 3/7

- 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 (people, places, things, …)

    Your dictionary for ES5 will simply be the titles of the movies dataset we used for ES1 and ES2, with scores

Fuzzy Search 4/7

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

    1. Type: uni

    2. Match: universe, university, …

    3. Search: universe OR university OR …

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

    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 5/7

- 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 6/7

- 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:** not so easy

    The focus of the rest of today's lecture

Fuzzy Search 7/7

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

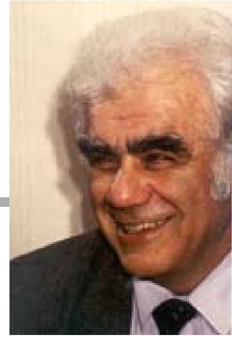
    n = #records, T = time for checking a single string
  - For fuzzy search, T ≈ 1µs ... find out yourself in ES5
  - In search, we always want interactive query times

    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 20 – 26

Edit distance 1/6

Vladimir Levenshtein
*1935, Russia



- 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 = D O O F
B O O F — REPLACE(1, B)
B L O F — REPLACE(2, L)
B L O E F — INSERT(4, E)
REPLACE(5, D)
y = B L O E D

$\Rightarrow$ ED(x, y) $\leq$ 4
and actually
ED(x, y) = 4
see next slides

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 $1 \le i \le j \le |x|$
- Some simple properties
  - $ED(x, y) = ED(y, x)$
  - $ED(x, \varepsilon) = |x|$
  - $ED(x, y) \ge abs(|x| - |y|)$ $\qquad abs(z) = z \ge 0 \ ? \ z : -z$
  - $\underbrace{ED(x, y)}_{=4} \le \underbrace{ED(\underbrace{x[1..n-1]}_{DOO}, \underbrace{y[1..m-1]}_{BLOE})}_{=3} + 1$ $\qquad n = |x|, m = |y|$

$x = DOOF$

$y = BLOED$

Edit distance 3/6

- 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 2015, Lecture 11a, slides 18 – 23

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	$\boxed{3}$	4
O	3	3	3	2	3	4
F	4	4	4	3	3	$\boxed{4}$

ED(ε, BLO) = 3

ED(DO, BLOE) = 3

ED(DOOF, BLOED) = 4

Edit distance 5/6

- **Prefix** edit distance
  - The prefix edit distance between x and y is defined as

    $\mathrm{PED}(x, y) = \min_{y'} \mathrm{ED}(x, y')$ 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

- 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 !**

y

x

	ε	U	N	I	V	E	R	S	I	T	Y
ε	0	1	2	3	4	5	6	7	8	9	10
U	1										
N	2										
I	3										
W	4	$\boxed{3}$	2	1	1	2	3	4	5	6	7

ED(UNIV, U) = 3

Say δ = 2

no need to compute those column

take the MIN of those

q-Gram Index 1/7

$|x| = 10$ $\quad |x| - (q-1) = |x| - q + 1$

- Definition of a q-gram

  $q = 3 \Rightarrow$ 3-gram

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

    university: uni, niv, ive, ver, ers, rsi, sit, ity

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

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

    university → $$university$$

    q-grams are then: $$u, $un, uni, …, sit, ity, ty$, y$$

    The number is then |x| + q – 1, where x is the original string

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

q-Gram Index 2/7

- 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

    $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

    Note: very similar to an inverted index, just with q-grams instead of words

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

q-Gram Index 3/7

- Space consumption
  - Each record x contributes |x| + O(1) ids to the inverted lists
  - The total number of ids in the lists is hence about the number of **characters** (not words) in the dictionary

    If we use 4 bytes per id, the index would hence be at least four times bigger than the original dictionary

    This can be reduced significantly using compression

    For ES5, it is fine to store the lists uncompressed

q-Gram Index 4/7

- Fuzzy search with a q-gram index, using ED
  - 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

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

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

  number of q-grams in common = **4**

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

q-Gram Index 5/7

$\max(|x|, |y|) + q - 1$

HIL~~L~~ARY

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

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

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

    Hence comm(x', y') ≥ 3 … and in the example comm = 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) changes at most q q-grams, and hence δ tra'fos affect at most δ · q q-grams

q-Gram Index 6/7

- Query algorithm, using ED
  - 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 keep track of which record contains how many q-grams … see TIP file on the Wiki
  - For each record y in the merge results, check whether the count is $\geq \max(|x|, |y|) - 1 - (\delta - 1) \cdot q$

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

    If yes: compute $ED(x, y)$ and check if $ED(x, y) \leq \delta$

q-Gram Index 7/7

- Fuzzy **prefix** search
  - Use the same algorithm, but with a different bound
  - Assume that **P**ED(x, y) ≤ δ
  - 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') ≥ **|x| – q · δ**

    Note that for δ = 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 δ · q q-grams

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