Information Retrieval

WS 2016 / 2017

Lecture 5, Tuesday November 22nd, 2016

(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 breifurg, find freiburg
  - 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   1/3

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

    "Very elegant explanations … no problems with exercises"

    "Some natural frustration … but an enjoyable challenge"

    "Did not enjoy … don't like mathematical proofs a lot"
  - Very helpful to understand the concepts from the lecture
  - Help in the forum was much appreciated
  - Looking forward to the master solution (it's there!)
  - Looking forward to coding exercises again
  - Entropy of human DNA is 7.13 on average according to

    https://www.hindawi.com/journals/mpe/2012/132625/tab1

Experiences with ES4 2/3

$$\log_2 e^{-px} = \frac{\ln e^{+px}}{\ln 2} = \frac{px}{\ln 2}$$

Assume: $p \le \frac{1}{2}$

$\Rightarrow 1-p \ge \frac{1}{2}$

- Proof sketch for Exercise 4.2
  - Show that Gollum is optimal for $p_x = (1-p)^{x-1} \cdot p$

I $\quad p_x = (1-p)^{x-1} \cdot p = (1-p)^x \cdot \frac{p}{1-p} \le e^{-px} \cdot \frac{p}{1-p}$

$\quad \log_2 \frac{1}{p_x} \ge \underbrace{\log_2 e^{+px}}_{px/\ln 2} + \log_2 \frac{1}{p} + \underbrace{\log_2 (1-p)}_{\ge -1}$

$$M = \left\lceil \frac{\ln 2}{p} \right\rceil \ge \frac{\ln 2}{p}$$

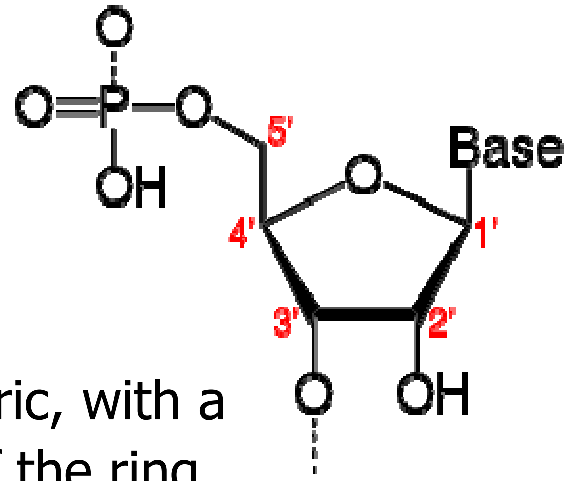
II $\quad L_x = \lfloor x/M \rfloor + \lceil \log_2 M \rceil + 1 \le x/M + \log_2 M + 2$

$\quad \le \frac{px}{\ln 2} + \log_2 \frac{1}{p} + \underbrace{\log_2 \ln 2}_{\le 0} + 3$

III $\quad$ I + II $\Rightarrow L_x \le \log_2 \frac{1}{p_x} + 4$

IV $\quad E L_x = \sum_x p_x \cdot L_x \le \underbrace{\sum_x p_x \cdot \log_2 \frac{1}{p_x}}_{H(X)} + \underbrace{\sum_x p_x}_{=1} \cdot 4$

Experiences with ES4 3/3



- Your DNA
  - The nucleotides of your DNA are asymmetric, with a phosphate group attached to the 5' side of the ring
  - Synthesizing only works in the 5'-to-3' direction, because making bonds in that direction is more energy efficient
  - However, if one strand of DNA goes in the 5'-to-3' direction, the other must go in the 3'-to-5' direction
  - So how does the cell manage to copy both strands?

    [The answer is quite amazing]()
  - You are quite a machine … on the biomolecular level
  - More about that on future sheets

Fuzzy Search 1/6

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

    For ES5, a list of 181,296 cities in Western Europe
  - 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 11 – 16

Fuzzy Search 6/6

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

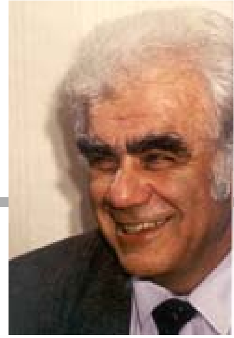
    n = #records, T = time for checking a single string
  - For fuzzy search, T ≈ 1µs ... find out yourself in ES5
  - In search, we always want interactive query times

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

This just proves that $ED(x,y) \leq 4$!

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

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

Edit distance 3/6

DON'T implement this recursively (will take time $\Omega(3^{|x|})$) but as explained in the following.

- 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 $\Theta(|x| \cdot |y|)$

$y \rightarrow$

$x \downarrow$

	ε	B	L	O	E	D
ε	0	1	$\boxed{2}$	3	4	5
D	1	1	2	3	4	$\boxed{4}$
O	2	2	2	2	3	4
O	3	3	3	2	3	4
F	4	4	4	3	3	$\boxed{4}$

ED(ε, BL)

ED(D, BLOED)

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

- 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 !**

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

let's say δ = 2

Then you don't have to look beyond this point.

minimum of THOSE ⇒ PED(FIBU, FREIBURG) = 2

ED(FIBU, F)

ED(FIBU, FREIBU)

ED(FIBU, FREI)

q-Gram Index 1/7

q=3

- 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

    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

    freiburg → $$freiburg$$

    3-grams: $$f, $fr, fre, rei, eib, ibu, bur, urg, rg$, g$$

    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

    $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

    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

$x = \text{HILLARY}$, $x' = \text{\$\$HILLARY\$\$}$

$y = \text{HILARI}$, $y' = \text{\$\$HILARI\$\$}$

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

    Then: $\text{comm}(x', y') \geq \max(|\overset{7}{x}|, |\overset{6}{y}|) - 1 - (\delta \overset{1}{-} 1) \cdot \overset{3}{q}$ $= 3$

    Example from slide before: $|x| = 7$, $|y| = 6$, $\delta = 2$, $q = 3$

    Hence $\text{comm}(x', y') \geq 3$ … and in the example comm = 4

    Verify: in the worst case, $\text{comm}(x', y') = 3$ 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 $\delta \cdot q$ q-grams

q-Gram Index 6/7

- Query algorithm, using ED (for PED: analogous)
  - 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