

Databases and Information Systems WS 25/26

Lecture 8: Fuzzy Search, Prefix Edit Dist, Q-Gram Index
December 9, 2025

Prof. Dr. Hannah Bast
Department of Computer Science
University of Freiburg

Overview of this lecture

■ Organizational

- Experiences with ES7 Knowledge Graphs and SPARQL

■ Contents

- Fuzzy search type **breif**, find **freiburg**
- Edit distance similarity measure for strings
- Fuzzy + q-grams a fascinating relationship
- Q-gram Index tying it all together

ES8: Implement error-tolerant ("fuzzy") prefix search using a q-gram index and prefix edit distance, on a nice dataset we have compiled for you, with 1.7 million Wikidata entities

Experiences with ES7 1/2

■ Excerpts from your feedback [somewhat representative]

"Again, I really liked the lecture and the sheet"

"Really interesting to see the difference between SPARQL and SQL"

"This exercise sheet took 6 hours and was straightforward"

"Took me a while to get the idea of putting all triples in one table"

"It took me way too long to write the sparql to sql function"

"The sheet was very hard and took me a long time to understand"

"Donald Trump has the predicate 'personal pronouns' (P6553)"

"The title of the British monarch has changed three times: Monarch of [Great Britain | the UK of Great Britain and Ireland | the UK]"

Experiences with ES7 2/2

■ Which do you like better, SQL or SPARQL?

"I definitely prefer SPARQL, it feels way more intuitive to me ... the examples shown in the lecture were really impressive / inspiring"

"I like SPARQL more. Kind of sad that it's not the standard"

"I theoretically prefer SPARQL because it is more intuitive, but since barely anything worked for me for ES7, I can't really say"

"I think it depends on the use case ... maybe SPARQL is not the best suited for a bank app, but useful for things like Wikidata"

"The statement notes are kind of annoying, that is better on SQL, but SPARQL is easier to write"

"I enjoy SQL more because of the frustrating slow autocompletion on qlever.dev ... you should really get a student to improve this"

Fuzzy Search 1/3

■ Problem setting

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

For ES8: a list of 1,736,613 entities from Wikidata

- For a given query, find matching names from that dictionary

Query: frei	Match: freiburg	prefix search
Query: breifurg	Match: freiburg	fuzzy search
Query: brei	Match: freiburg	fuzzy prefix search

- We will solve the last two (normal prefix search is a special case); there are **two** challenges, similar as in Lectures 1+2

Challenge 1: find a good model of "relevance" (what matches)

Challenge 2: build an index (so that we can find matches fast)

■ Possible origins for the dictionary

- Popular queries extracted from a query log

That is the 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 entities from some knowledge graph

For ES8, we have prepared for you a selection of entities from Wikidata, together with their popularity and synonyms

■ Simple solution

- Iterate over all strings in the dictionary, and for each string check whether it matches
- This is what the Linux command-line tools `grep` and `agrep` do
 - `grep -x frei.* <file>` `-x` means: match whole line
 - `agrep -x -2 breifurg <file>` `-2` means: allow up to 2 "errors"
 - `agrep -x -2 brei.* <file>`
- This obviously takes time **linear** in the size of the dictionary

Good enough for medium-sized, but not for large dictionaries
- In the rest of this lecture, we show how to do this faster, with a suitable **index** precomputed for the dictionary



■ Definition of the edit distance

aka Levenshtein distance, from a paper from 1965

- Definition: for two strings x and y , define **ED**(x , y) as the minimal number of transformations to get from x to y , where each transformation is one of the following:

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

- Example: transform **DOOF** into **BLOED**

$x = \text{DOOF}$	
B O O F	↙
B L O F	↙
B L O E	↙
y = B L O E D	↙

	↘	REPLACE(1, B)
	↘	REPLACE(2, L)
	↘	REPLACE(4, E)
	↘	INSERT(5, D)

*This proves that
 $ED(x, y) \leq 4$
(could be less)*

Edit distance 2/7

■ 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

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

Proving these is a great exercise for the exam preparation

Edit distance 3/7

■ Recursive formula

- For $|x| > 0$ and $|y| > 0$, $ED(x, y)$ is the minimum of
 - (1) $ED(x[1..n], y[1..m-1]) + 1$
 - (2) $ED(x[1..n-1], y[1..m]) + 1$
 - (3a) $ED(x[1..n-1], y[1..m-1]) + 1$ if $x[n] \neq y[m]$
 - (3b) $ED(x[1..n-1], y[1..m-1])$ if $x[n] = y[m]$
- For $|x| = 0$ we have $ED(x, y) = |y|$
- For $|y| = 0$ we have $ED(x, y) = |x|$

The proof of this formula is not trivial, see Lecture 11 from "Algorithmen und Datenstrukturen", SS 2021, slides 13 – 19

You should know the formula, but the proof is not klausur 

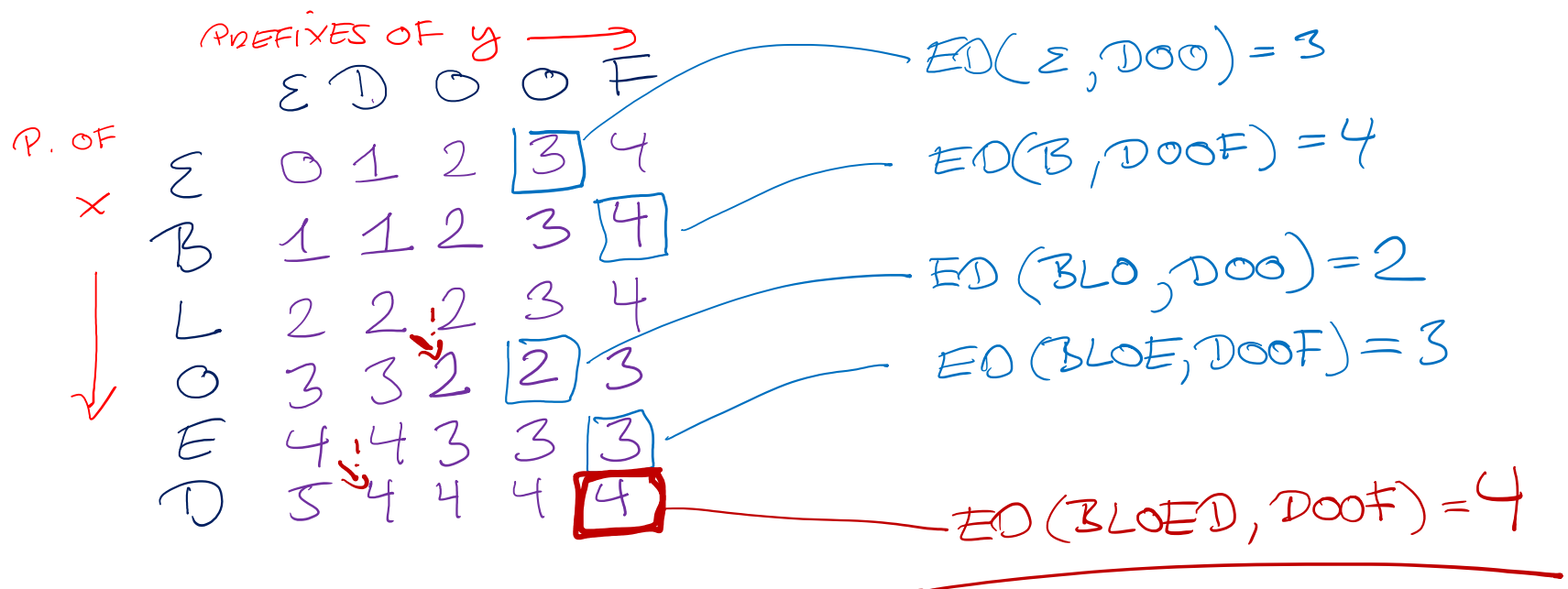
Edit distance 4/7

■ 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: $\Theta(|x| \cdot |y|)$

$\Theta(\min\{|x|, |y|\})$
space works too



Edit distance 5/7

■ Prefix edit distance

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

PED(x, y) := $\min_{y'} \text{ED}(x, y')$ where y' is a prefix of y

- For example $\text{ED}(\text{uni}, \text{uni})$

$\text{PED}(\overset{x}{\text{uni}}, \overset{y}{\text{university}}) = 0$... but $\text{ED} = 7$

$\text{PED}(\overset{x}{\text{uniwer}}, \overset{y}{\text{university}}) = 1$... but $\text{ED} = 5$

- PED is important for fuzzy search-as-you type suggestions

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

The entity search on <https://wikidata.org/> does **not** have it

See also <https://www.bild-video-ton.ch/ansicht/gross/274590.jpg>

Edit distance 6/7

■ Computation of the PED

- Compute the entries of the $|x| \cdot |y|$ table, just as for ED
- The PED is simply the **minimum** of the entries in the last row
- Important optimization: when $|x| \ll |y|$ and you only want to know if $\text{PED}(x, y) \leq \delta$ for some given δ , then it is enough to compute only the first $|x| + \delta + 1$ columns ... **understand why !**
- For example: is $\text{PED}(\text{fibu}, \text{freiburg}) \leq 2$?

if you wonder if $\text{PED}(x, y) \leq 2$, no need to look to the right of this

		Σ	F	R	E	I	B	U	R	G	
			0	1	2	3	4	5	6	7	8
$\times$	Σ	0	1	2	3	4	5	6	7	8	
	F	1	-	-	-	-	-	-	-	-	
	I	2	-	-	-	-	-	-	-	-	
$\downarrow$	B	3	-	-	-	-	-	-	-	-	
	U	4	3	3	3	3	2	3	4		

← PED is the min. of these

will all be ≥ 3

$\text{ED}(\text{FIBU}, \text{FREIBU})$

$\text{ED}(\text{FIBU}, \text{FR}) = 3$

■ Implementation of the PED for Exercise Sheet 8

- Implementing the **PED** is fun, but we figured that ES8 is already enough work (and enough to think) without this

We therefore provide the implementation for you

- Since the PED computation is critical for the performance, we provide an implementation in **Rust**

`pip install ad-freiburg-qgram-utils`

- As a fallback (in case anyone has problems with this package), we also provide a **Python** implementation

The code template provided on the Wiki uses the Rust version if it is installed, and the Python version otherwise

- Both versions also provide an implementation of `merge_lists`

Fuzzy search with q-grams 1/8

■ Motivation

- Find fuzzy matches for a given word x in a dictionary D
faster than by computing $ED(x, y)$ or $PED(x, y)$ for all $y \in D$
- Intuitive idea: when two strings have a small edit distance, they overlap in many of their parts

Example 1: is $ED(\text{"freiburg"}, \text{"stuttgart"})$ small ?

No, the strings are very different "everywhere"

Example 2: is $ED(\text{"freiburg"}, \text{"breifurg"})$ small ?

Yes, the strings are similar "at many places"

- On the following slides, we will make this idea more precise and turn it into an efficient algorithm

■ Definition of a q-gram

- For a string x , and an integer $q \in \mathbf{N}$, the multiset of q-grams, denoted by $Q_q(x)$, consists of all substrings of length q

$$Q_3(\text{"freiburg"}) = \{ \text{"fre"}, \text{"rei"}, \text{"eib"}, \text{"ibu"}, \text{"bur"}, \text{"urg"} \}$$

- We define it as a **multiset** because the same q-gram may occur multiple times and we want to know when it does

$$Q_3(\text{"ababa"}) = \{ \text{"aba"}, \text{"bab"}, \text{"aba"} \}$$

- The number of q-grams of a string x is:

$$|Q_q(x)| = |x| - q + 1$$

Fuzzy search with q-grams 3/8

Intuition: every op. "affects" at most q q-grams

■ Similar words have many q-grams in common

- **Lemma:** for strings x and y : $|Q_q(x) \setminus Q_q(y)| \leq q \cdot ED(x, y)$

Understand: $A \setminus B$ denotes the set difference, that is, the elements of A without the elements from B

If B is very similar to A , then $A \setminus B$ is small

- Example:

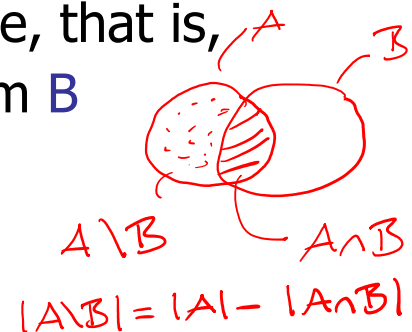
$x = \underline{f}r\underline{ee}b\underline{u}rger, y = \underline{b}r\underline{ee}u\underline{r}g\underline{e}r, ED(x, y) = 2$

$Q_2(x) = \{ \underline{fr}, re, ee, \underline{eb}, \underline{bu}, ur, rg, ge, er \}$

$Q_2(y) = \{ \underline{br}, re, ee, \underline{eu}, ur, rg, ge, er \}$

$Q_2(x) \setminus Q_2(y) = \{ fr, eb, bu \}$

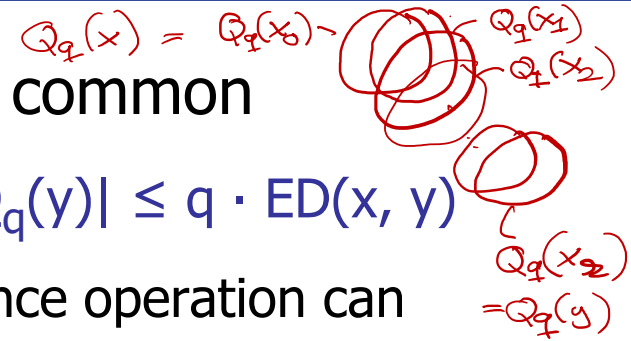
$|Q_2(x) \setminus Q_2(y)| = 3$



Fuzzy search with q-grams 4/8

■ Similar words have many q-grams in common

- **Lemma:** for strings x and y : $|Q_q(x) \setminus Q_q(y)| \leq q \cdot \text{ED}(x, y)$
- **Proof:** For $\text{ED}(x, y) = 1$, one edit distance operation can change at most q q-grams, hence $|Q_q(x) \setminus Q_q(y)| \leq q$



For the general case $\text{ED}(x, y) = k$, let $x = x_0 \rightarrow \dots \rightarrow x_k = y$ be a sequence of k ED operations that turn x into y

That is, $\text{ED}(x_{i-1}, x_i) = 1$ and therefore $|Q_q(x_{i-1}) \setminus Q_q(x_i)| \leq q$

It remains to prove that the following holds for any sequence

$A_0, \dots, A_k$ of sets: $A_0 \setminus A_k \subseteq \bigcup_{i=1..k} (A_{i-1} \setminus A_i)$ $\Rightarrow |A_0 \setminus A_k| \leq \sum_{i=1}^k |A_{i-1} \setminus A_i|$

$x \in A_0 \setminus A_k \Rightarrow x \in A_0$ but $x \notin A_k$

✓ ✓ ✓ ✗
 $A_0 \ A_1 \ A_2 \ A_3 \dots A_k$

$\Rightarrow \exists i \ x \in A_{i-1}$ but not $x \in A_i$

$\Rightarrow x \in A_{i-1} \setminus A_i \Rightarrow x \in \bigcup_{i=1}^k (A_{i-1} \setminus A_i)$ $\square$

Fuzzy search with q-grams 5/8

- Similar words have many q-grams in common



$$|A \setminus B| = |A| - |A \cap B|$$

- **Corollary 1:** for arbitrary strings x and y

$$|Q_q(x) \cap Q_q(y)| \geq \max\{|Q_q(x)|, |Q_q(y)|\} - q \cdot \text{ED}(x, y)$$

- Example:

$$= |\{re, ee, uv, vg, ge, er\}| = 6$$

$$|Q_2(\text{freeburger}) \cap Q_2(\text{breeurger})| \geq \max\{|Q_2(x)|, |Q_2(y)|\} - 2 \cdot 2$$

- **Proof:** (using the Lemma from slide 18)

For arbitrary sets A and B , $|A \setminus B| = |A| - |A \cap B|$, hence

$$|Q_q(\mathbf{x})| - \underline{|Q_q(x) \cap Q_q(y)|} = |Q_q(x) \setminus Q_q(y)| \stackrel{\text{LEMMA}}{\leq} q \cdot \text{ED}(x, y)$$

Since $\text{ED}(x, y) = \text{ED}(y, x)$, the same lemma also gives us

$$|Q_q(\mathbf{y})| - \underline{|Q_q(x) \cap Q_q(y)|} = |Q_q(y) \setminus Q_q(x)| \stackrel{\text{LEMMA}}{\leq} q \cdot \text{ED}(y, x) \stackrel{\text{ED}(x,y)}{=}$$

The two inequalities together give us the corollary

Fuzzy search with q-grams 6/8

- Similar words have many q-grams in common

- **Corollary 2:** for arbitrary strings x and y

$$|Q_q(x) \cap Q_q(y)| \geq |Q_q(x)| - q \cdot \text{PED}(x, y)$$

- **Example:**

$$|Q_2(\text{bree}) \cap Q_2(\text{freeburger})| \geq 3 - 2 \cdot 1$$

$\begin{array}{ccc} |Q_2(x)| & q & \text{PED}(x, y) \\ \downarrow & \downarrow & \downarrow \\ 3 & 2 & 1 \end{array}$

$\begin{array}{l} = |\{re, ee\}| = 2 \\ \text{bree} \quad \text{freeburger} \end{array}$

- **Proof sketch:**

$\text{PED}(x, y) = \text{ED}(x, y')$ for some prefix y' of y

Like on the previous slide:

$$|Q_q(x)| - |Q_q(x) \cap Q_q(y')| = |Q_q(x) \setminus Q_q(y')| \leq q \cdot \text{ED}(x, y')$$

$\swarrow$ **LEMMA**
 $= \text{PED}(x, y)$

The corollary follows because $Q_q(y) \supseteq Q_q(y')$

$$\Rightarrow |Q_q(x) \cap Q_q(y)| \geq |Q_q(x) \cap Q_q(y')|$$

Note that in general, $\text{PED}(x, y)$ is **not** equal to $\text{PED}(y, x)$

Fuzzy search with q-grams 7/8

■ Algorithm for ED ... for some fixed $q \in \mathbb{N}$

- Problem: Given x and δ , find all $y \in D$ with $\mathbf{ED}(x, y) \leq \delta$
- For each word $y \in D$, do the following:
 1. Is $|Q_q(x) \cap Q_q(y)| \geq \max\{|Q_q(x)|, |Q_q(y)|\} - q \cdot \delta$?
 2. NO $\rightarrow$ We know that $\mathbf{ED}(x, y) > \delta$ and y is not a match
 3. YES $\rightarrow$ Compute $\mathbf{ED}(x, y)$ to determine if y is a match

*relatively
CHEAP*

*relatively
EXPENS.*

■ Algorithm for PED ... for some fixed $q \in \mathbb{N}$

- Problem: Given x and δ , find all $y \in D$ with $\mathbf{PED}(x, y) \leq \delta$
- For each word $y \in D$, do the following:
 1. Is $|Q_q(x) \cap Q_q(y)| \geq |Q_q(x)| - q \cdot \delta$?
 2. and 3. exactly like above

mit PED instead of ED

Fuzzy search with q-grams 8/8

■ Improvement for ED

- **Pad** all strings in D with $q - 1$ times \$ on **both sides**

For example ($q = 3$): freiburg $\rightarrow$ \$\$freiburg\$\$

The algorithm then gives the same result because:

x' and y' padded version of x and $y \rightarrow ED(x, y) = ED(x', y')$

■ Improvement for PED

- **Pad** all strings in D with $q - 1$ times \$ on the **left side only**

For example ($q = 3$): freiburg $\rightarrow$ \$\$freiburg

x'' and y'' padded versions of x and $y \rightarrow PED(x'', y'') = PED(x, y)$

It's an improvement because with the padding we get more q-grams and hence Step 2 from Slide 21 applies more often

Q-gram index 1/4

■ Motivation

- For the algorithm described on slide 21, we need to compute, for a given x , $|Q_q(x) \cap Q_q(y)|$ for **all** $y \in D$

For ES7, the D contains **1.7** million strings, so a lot

- How do we do that efficiently?

The answer is: with a **q-gram index** ... see next three slides

Q-gram index 2/4

■ Definition

- For each q-gram of a string from **D**, store an inverted list of all strings from **D** containing it, sorted lexicographically

\$fr : **fr**ankfurt, **fr**eiburg, **fr**etown, **fr**esno, ...

ibu : ch**ic**ib**u**, fr**ei**bu**rg**, **i**b**u**suki, mal**i**b**u**, ...

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 : for each **word**, a list of **doc ids**

q-gram index : for each **q-gram**, a lists of **words ids**

- Let's adapt our code from Lecture 1 to q-grams, to see how similar the code is ...

Q-gram index 3/4

■ Using the q-gram index for our algorithm

– Proceed as follows for the (appropriately padded) input x :

1. Compute $Q_q(x)$, the set of q-grams of x

2. For each q-gram from $Q_q(x)$, fetch its inverted list

3. Compute the union of these $|Q_q(x)|$ inverted lists, and count how many times each word occurs

– Understand how that gives us $|Q_q(x) \cap Q_q(y)|$ for all $y \in D$:

If a word y occurs k times in the union, it contains exactly k q-grams from $Q_q(x)$, and hence $|Q_q(x) \cap Q_q(y)| = k$

For all words y that are not encountered in the union at all, we know that $|Q_q(x) \cap Q_q(y)| = 0$... these are only relevant if the right-hand side from Step 1 of Slide 22 is ≤ 0

Q-gram index 4/4

- Example: $x = \text{brei}$, $\delta = 1$... using a 2-gram index and PED
 - Fetch the following inverted lists ($\$brei \rightarrow \b, br, re, ei)
 - $\$b$: bangalore, beijing, berlin, ...
 - br : brisbane, brussels, gibraltar, ...
 - re : bangalore, freiburg, singapore, ...
 - ei : beijing, freiburg, marseille, ...
 - According to Corollary 2, for a y with $PED(x, y) \leq \delta = 1$, $|Q_2(x) \cap Q_2(y)|$ must be at least $|Q_2(x)| - q \cdot \delta = 2$
 - If we merge the inverted lists above, these three words occur (at least) twice: bangalore, beijing, freiburg, so we have to check these (for the others, we know that $PED(x, y) > 1$)
 $PED(\text{brei}, \text{bangalore}) = 3$, $PED(\text{brei}, \text{beijing}) = 1$, $PED(\text{brei}, \text{freiburg}) = 1$

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