raytion

Did-you-mean

Bastian Mathes, Raytion GmbH

Berlin Buzzwords 2017

13th June 2017

Agenda

Introduction

Implementation Approaches (nothing fancy)

Summary

About me

Raytion GmbH, Düsseldorf

Enterprise search projects since
2001 (Raytion) / 2009 (me)

Solr, Elasticsearch, Exalead, FAST ESP, FS4SP,
GSA (me) + *sharepoint*, Attivio, … (Raytion)

Introduction

Introduction – Did-you-mean

The user misspelled a query

Therefore there a no or few results

DYM suggests the most likely correction

DYM – Usability

DYM „module“ delivers correction and estimated number of results (ideally)

When to show DYM at all ?

When to redirect to the correction immediately ?

When search for both original and correction ?

DYM – Requirements

Find a similar query with more results

→ Similar by edit distance limit (Levenshtein etc.)

→ If there is more than one possible correction rank them

- → By Similarity
- → By Frequency / Occurrence

→ Basis is a dictionary of valid query terms (one or more words) and their frequency, language, search area, ACLs…

→ Single word terms taken from the index, multi word terms / phrases are the difficult ones

→ Potentially split up long query (zero results or test query) – estimated number of results difficult / a lot of DYM lookups

Implementation Approaches

Naive – Automaton – BK Tree – Ngram Dictionary

Naive

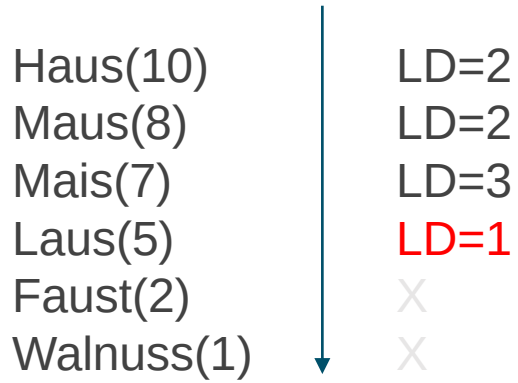
Run through list of terms and calculate distance

If list ordered by frequency, first=best correction

Often can be done in-place

Does not scale well (distance calculation is expensive, $O(n^2)$)

User query: Lauss
LD=1



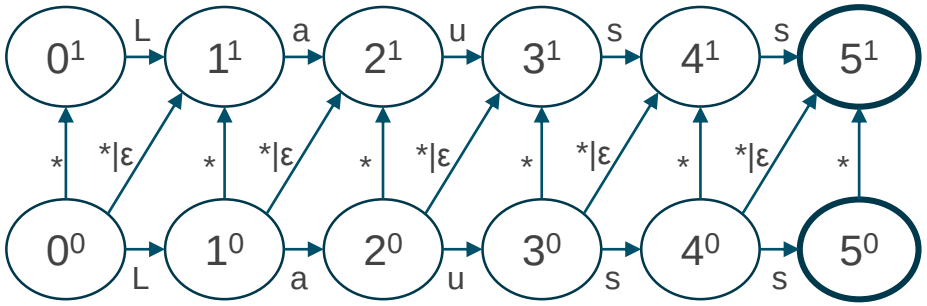
Levenshtein-Automaton

Reduce comparison runtime from O(n²) to O(n)

Initial effort (per Query) to construct Automaton (memory consumption !)

DirectSpellChecker in Solr or TermSuggester in Elasticsearch (in-place)

→ Stop after n visited terms



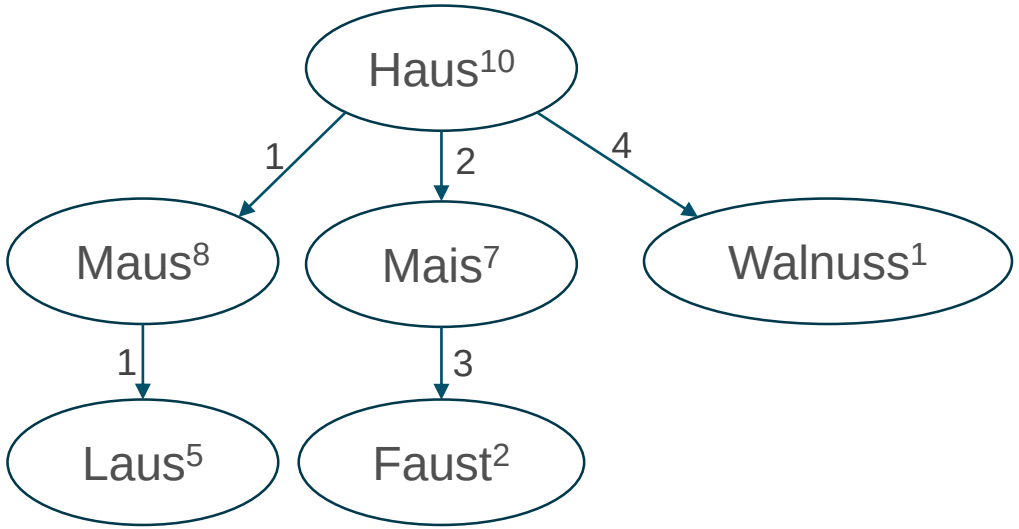
Haus(10)	not accepted
Maus(8)	not accepted
Mais(7)	not accepted
Laus(5)	accepted
Faust(2)	X
Walnuss(1)	X

BK-Tree

Additional data structure (memory, database)

Periodic effort to (re)create the tree

Distance on the edges, start with most common term



Reduce number of comparisons with triangle inequality

Ngram Dictionary

Terms are split to ngrams and indexed, query is split, too and used as query

Efficient infrastructure for search indexes available (in-memory, on-disk, with caching)

Additional filter criteria (security, search area) can be easily added as a filter

Ngram distance not and intuitive metric, post-check necessary

Solr IndexBasedSpellchecker

Ngram Dictionary

Word	Frequency	Length	Unigrams	Area
Haus	10	4	H, a, u, s	Search_A(7), Search_B(3)
Maus	8	4	M, a, u, s	Search_A(8)
Mais	7	4	M, a, i, s	Search_B(7)
Laus	5	4	L, a, u, s	Search_A(1), Search_B(4)
Faust	2	5	F, a, u, s, t	Search_A(2)
Walnuss	1	7	W, a, l, n, u, s, s	Search_B(1)

Lauss, LD=1, Search_A:
+Areas:Search_A +Length:[4 TO 6] +4of5(pos(L,0,2),pos(a,1,3),pos(u,2,4),pos(s,3,5),pos(s(4,6))
→ post Levenshtein check, potentially load more results

Summary

Summary

Algorithms are understood, implementations exists

Customization is important

Usability decisions depend on project context

Data is key (where to get valid terms from)

→ Search logs

→ co-occurance in the index (small indexes), PhraseSuggester in Elasticsearch

→ Part-of-speech tagging and extraction of patterns (POS is expensive)

→ Look around for project specific, “cheaper” sources

Thanks!

Bastian Mathes

bastian.mathes@raytion.com