Fehlertolerante Suche und Index Design

Dr. Barbara Löhle und Thomas Kirchhoff

Barbara.Loehle@uni-konstanz.de,

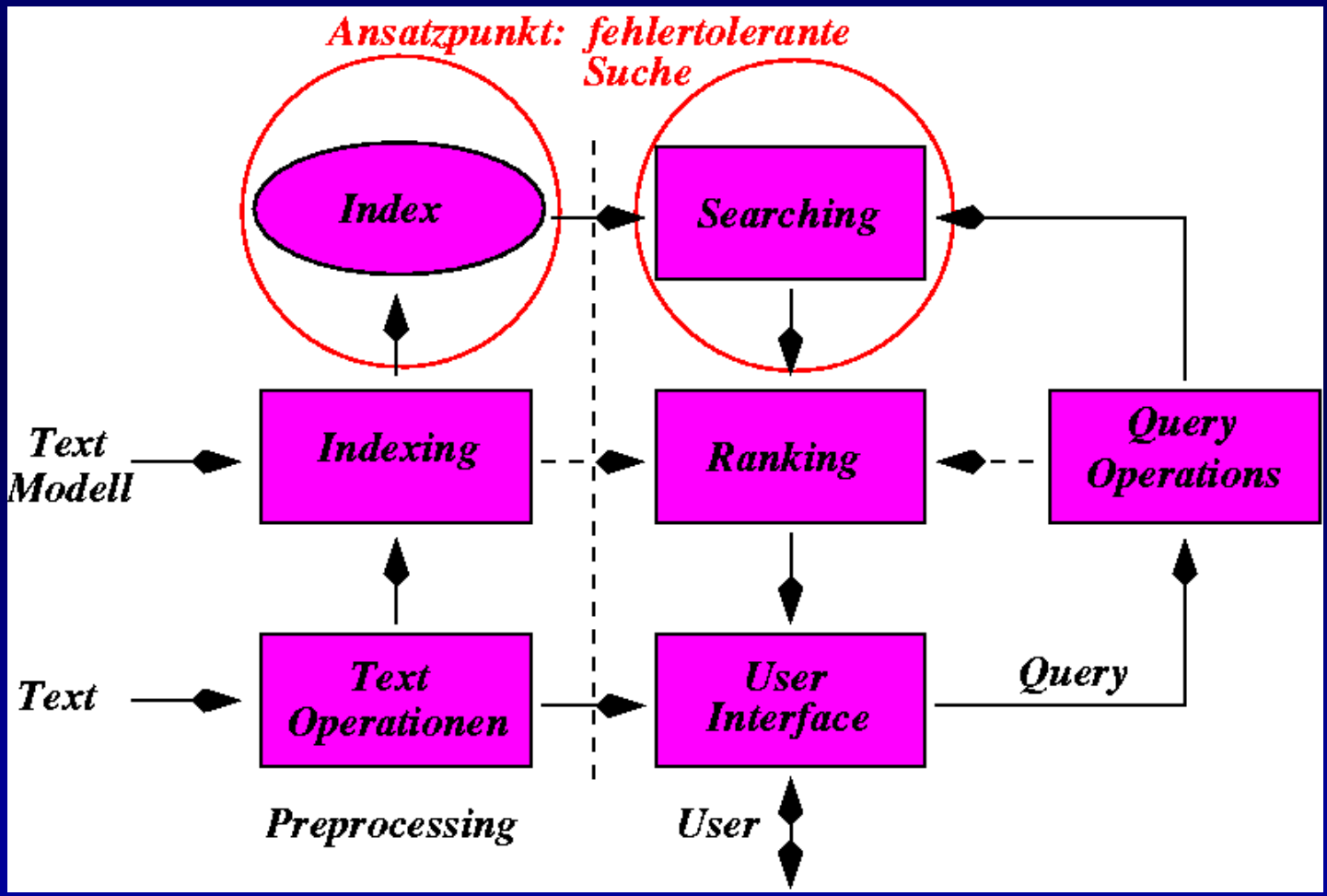
Thomas.Kirchhoff@bsz-bw.de

Inhalt

- Einführung
- Word-based Index
- Fehlertolerante Suche
  - Unit Cost Edit Distance (Levenshtein)
  - q-gram Index
- Fehlertolerante Suche mit Lucene
  - Unit Cost Edit Distance
  - "Spellchecker" (q-gram Index)
- Fehlertolerante Suche mit Approximus

Einführung

- Information Retrieval Library, z.B. Lucene, Approximus
- Information Retrieval Prozess:



Word-based Index

Realisierung mittels eines Inverted Index, bestehend aus:

- Vokabular (Lexikon) , in Memory
- Inverted File (Posting File), on Disk
- Dokumente, on Disk

Document	Text
1	Pease porridge hot, pease porridge cold
2	Pease porridge in the pot,
3	Nine days old,
4	Some like it hot, some like it cold,
5	some like it in the pot,
6	Nine days old,

Vokabular Postings

Number	Term	(Document; Words)
1	cold	(1;6), (4;8)
2	days	(3;2), (6;2)
3	hot	(1;3), (4;4)
4	in	(2;3), (5;4)
5	it	(4;3,7), (5;3)
6	like	(4;2,6), (5;2)
7	nine	(3;1), (6;1)
8	old	(3;3), (6;3)
9	pease	(1;1,4), (2;1)
10	porridge	(1;2,5), (2;2)
11	pot	(2;5), (5;6)
12	some	(4;1,5), (5;1)
13	the	(2;4), (5;5)

Eigenschaften des Vokabulars

- Festlegung von "word"
  - Abhängig von der Applikation
  - Bei Einsatz von Text-Analysern, z.B. Stop words, stemming, wird der Begriff "term" als Verallgemeinerung von "word" benutzt.
- Größe des Vokabulars
  - Heaps' Law: $V = O(N^{\beta})$ mit N : Zahl der "words" der Dokumente; $\beta = 0.4 - 0.6$
- Implementierung
  - z.B. Array: Einfach und geringer Speicherbedarf Bearbeitung von Queries ist langsam.
  - Datenstruktur des Vokabulars: wesentlich für die Suchgeschwindigkeit !!!
  - Datenstrukturen benötigen zusätzliches Memory
  - Datenstruktur des Index hängt von der Speicherart ab, z.B. in Memory.

Fehlertolerante Suche

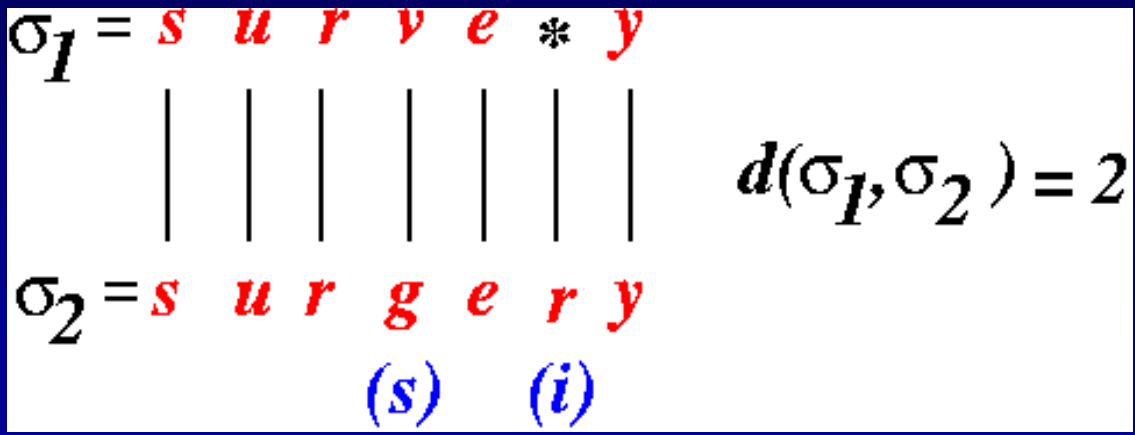
Fehlertolerante Suche und Index Design

Barbara Löhle und Thomas Kirchhoff

Unit Cost Edit Distance (Levenshtein)

Unit Cost Edit Distance (Levenshtein Distance)

- Minimum Edit Distance zwischen zwei Strings σ_1 bzw. σ_2 mit der Stringlänge $|\sigma_1| = m$ bzw. $|\sigma_2| = n$
- Minimale Zahl der Editing Operationen: Insertion, Deletion, Substitution = $d(\sigma_1, \sigma_2) = k$, um σ_1 in σ_2 zu überführen.
- Falls der Beitrag von Insertion, Deletion und Substitution zu $d(\sigma_1, \sigma_2)$ den Wert 1 beträgt.
- "string matching with k differences"



Berechnung der Levenshtein Distance

Dynamic Programming (DP): Worst Case

- anwendbar auf Terms eines Vokabulars
- Worst Case: Zeitaufwand: O(nm)
- Worst Case: Memorybedarf: O(m)
- Ziel: Seltene Ausführung von DP !!!

Prinzip:

		s	u	r	g	e	r	y
	0	1	2	3	4	5	6	7
s	1	0	1	2	3	4	5	6
u	2	1	0	1	2	3	4	5
r	3	2	1	0	1	2	3	4
y	4	3	2	1	1	2	3	4
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	2	2	2

$$C_{i,0} = i$$

$$C_{0,j} = j$$

$$C_{i,j} = \begin{cases} C_{i-1,j-1} & (\text{falls } \sigma 1_i = \sigma 2_j) \\ 1 + \min(C_{i-1,j}, C_{i,j-1}, C_{i-1,j-1}) & (\text{sonst}) \end{cases}$$

String Similarity

Ähnlichkeit zweier Terme σ_1 und σ_2 mit $d(\sigma_1, \sigma_2)$ mittels Similarity Measures z.B.:

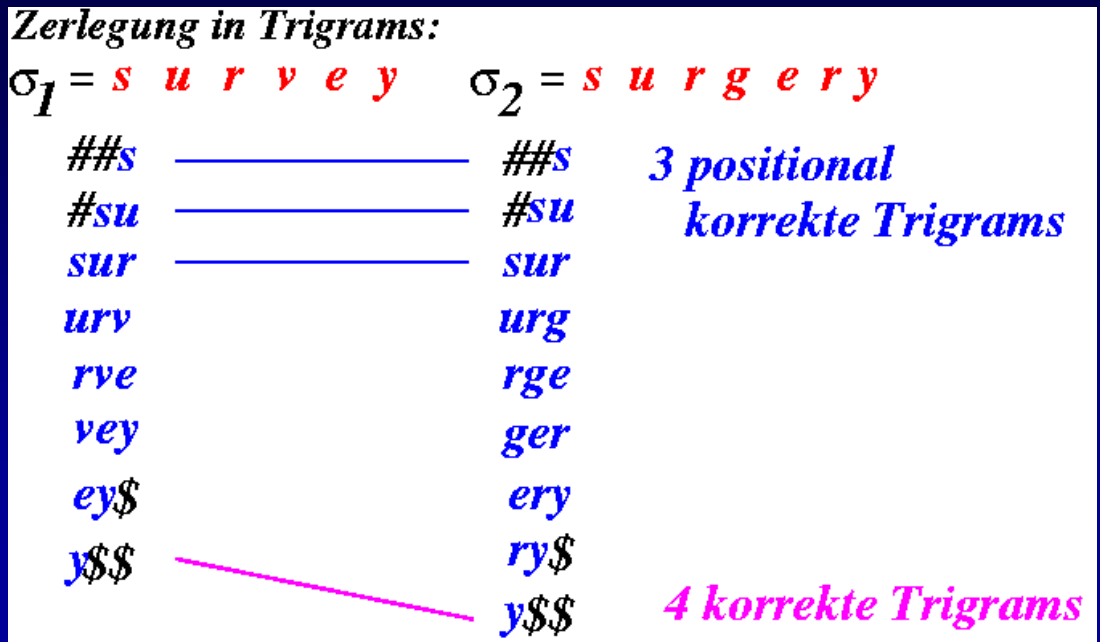
- Minimum String Similarity:
  $minSim = 1 - d(\sigma_1, \sigma_2) / \min(|\sigma_1|, |\sigma_2|)$
- es gilt:
  - » $minSim = 1$, falls $\sigma_1 = \sigma_2$
  - » $minSim = 0$, falls $d(\sigma_1, \sigma_2) = \min(|\sigma_1|, |\sigma_2|)$
  - » falls die Forderung gilt: $minSim >= minSim_{min}$

    $d_{max}(\sigma_1, \sigma_2) = \min(|\sigma_1|, |\sigma_2|) (1 - minSim_{min})$

q–gram

Positional q-gram mit q ε [1, .., N]:

- eines Strings σ besteht aus einem Paar (i, σ(i, ..., i + q – 1))
- σ besitzt |σ| + q – 1 q-grams
- q = 1 : unigram; q = 2 : bigram; q = 3: trigram)
- Ähnliche Strings besitzen eine grosse Anzahl gleicher q-grams
- Beispiel für Trigrams : mit Prefixfüller # und Suffixfüller $



q-grams und Edit Distance

- Substitution: $(|\sigma| + q - 1) - q$ gemeinsame q-grams
  - survey: { ##s, #su, sur, urv, rve, vey, ey$, y$$}
  - surley: { ##s, #su, sur, url, rle, ley, ey$, y$$}
- Insertion: $(\max(|\sigma_1|, |\sigma_2|) + q - 1) - q$ gemeinsame q-grams
  - survey: { ##s, #su, sur, urv, rve, vey, ey$, y$$}
  - survley: { ##s, #su, sur, urv, rvl, vle, ley, ey$, y$$}
- Deletion: $(\max(|\sigma_1|, |\sigma_2|) + q - 1) - q$ gemeinsame q-grams
  - survey: { ##s, #su, sur, urv, rve, vey, ey$, y$$}
  - surey: { ##s, #su, sur, ure, rey, ey$, y$$}
- Similarity : mittels eines Counting-Filters (c = Zahl der gemeinsamen q-grams) unter Verlust der positional Information gilt: similarity = $2\,c\,/(|\sigma_1| + |\sigma_2|)$

q-gram Index

q-gram Index:

- Erstellung eines Inverted Index analog zum word-based Index
- Festlegung der zu verwendenden q-gram
  - » hohe Fehlertoleranz: Nutzung kurzer q-grams
  - » Problem: grosse Anzahl z.T. schlechter Ergebnisse
  - » Praxis: Kombination aus 3- und 4-grams bzw. für kurzWorte Unigrams und Bigrams.

Fehlertolerante Suche mit Lucene

Fehlertolerante Suche und Index Design

Barbara Löhle und Thomas Kirchhoff

Versuchsbedingungen

- Ausgangspunkt: realer Datensatz, z.B. GBV Daten
  - indiziert mittels Lucene innerhalb des BAM-Projekts
  - Datensatz: 15 GB
  - Bestimmung der Gesamtgröße der Vokabulare: 63.7 MB
- Benutzung von Dictionary Files (dict) von puzzlers.org
- Indizierung von factorDict ● dict (factorDict = 1 - 20)
- exacte Suche auf factorDict ● dict: nach factor Versuchen (factor = 1 - 30):
  - query-Response Zeit: 0 - 1 ms
- postings File liefert keinen Beitrag zur query-Response Zeit
- fehlertolarante Suche unter der Randbedingung:
  - query-Response Zeit bezieht sich auf eine Single-"field"-Suche
  - reale Multi-"field"-Suche: Summe der query-Response Zeit der einzelnen "fields" bzw. Dictionary Files.

Testdaten

- GBV Daten

Dictionary File	#Strings	Filegröße (MB)
GBV Author	831358	6.73
GBV Location	135067	1.1
GBV Publisher	372554	3.3
GBV Subtitle	1562400	16.7
GBV Title	3226340	35.9
	6127719	63.71

- Dictionary Files von puzzlers.org

Dictionary File	#Strings	Filegröße (MB)
Enable Word	173000	1.7
Moby Single Word	355000	3.5

Unit Cost Edit Distance

- Inverser Index
  - Erzeugt mittels: Class DocumentationWriter
  - method: addDocument aus Field Values
  - Vokabular: lexikographisch sortiert; ansonsten keine Datenstruktur des Vokabulars
- Unit Cost Edit Distance
  - Implementiert mittels des DP Algorithmus
  - Similarity: Parameter minSim (default Wert: 0.5)
- query-Response Zeit wird bestimmt durch
  - Maximum der Wortlängenverteilung bei 7 – 8 Characters
  - DP ~ O(mn)
  - lineare Abhängigkeit von der Dictionary Größe

((#res) in ms)	"pier"	"abnormal"	"quantitativeness"
Enable Word	(17) 235	(20) 510	(79) 470
Moby Single Word	(27) 475	(47) 1080	(244) 1050

“Spellchecker” (q-gram Index)

- zusätzlicher Inverser Index
  - unter trunk/contrib/spellchecker
  - Vokabulare: bestehend aus 3- und 4-grams (kurze Worte indiziert mittels Unigrams und Bigrams) (4 Fields)
  - Vergrößerung des Index um einen Faktor 5 – 7
- Exakte Suche nach gemeinsamen q-grams: hits
- Anwendung der Unit Cost Edit Distance (minSim = 0.5)
  - Zahl der #res: min(hits, 10 suggests);
  - Problem: Zahl der suggests für minSim ist nicht bekannt

((#res = 1) in ms)	“pier”	“abnormal”	“quantitativeness”
Enable Word	15	15	40
Moby Single Word	25	25	115

((#res) in ms)	“pier”	“abnormal”	“quantitativeness”
Enable Word	(17) 30	(20) 30	(79) 230
Moby Single Word	(27) 75	(47) 90	(244) 610

Fehlertolerante Suche mit Approximus

- Inverser Index
  - "time vs. space trade-off" optimierte Datenstrukturen des Vokabulars
- Unit Edit Distance Distance (Levenshtein Distance)
  - Implementiert mittels des optimierter DP Algorithmus: O([m/w] n) mit w = computer word length.
  - Diverse Similarity Measures, z.B. Minimum String Similarity
  - range query mit range r : alle Results mit $d(\sigma_1, \sigma_2) <= r$

r = 1 (in ms)	"pier"	"abnormal"	"quantitativeness"
Enable Word	0.83 – 5.14	0.04 – 7.2	0.06 – 1.01
Moby Single Word	1.48 – 10.95	0.05 – 13.2	0.14 - 2.82
r = 2			
Enable Word	2.25	1.3 – 10.8	0.17 – 2.05
Moby Single Word	4.0	3.18 - 20.44	0.47 – 5.18

Gesamtvergleich

Minimum String Similarity: minSim = 0.7

- Forderung: #res Spellcheckers = #res Lucene: unit cost edit distance search
- benutzt wird das Moby Single Word Vokabular

((#res) in ms)	"pier"	"abnormal"	"quantitativeness"
Lucene: UCED	(24) 440	(11) 680	(5) 640
Lucene: Spellchecker	(24) 90 (sugg = 55)	(11) 75 (sugg = 22)	(5) 100 (sugg = 8)
Approximus: range query	r = 1 (27) 1.45	r = 2 (14) 3.2	r = 4 (6) 13.0

Zusammenfassung und Ausblick

- Die Suche auf den Vokabularn bedingt im wesentlichen die Query-Response Zeit. (exakte als auch fehlertolerante Suche)
- Datenstrukturen des Vokabulars verkürzen essentiell die Query-Response Zeit. (Datenstrukturen benötigen zusätzliches Memory: time vs. space trade-off)
- festgesetzte Randbedingungen erfordern Investitionen in Algorithmik
  - maximale, tolerierbare Query-Response Zeit
  - maximales, verfügbares Memory
- Lucene bietet eine sehr gute Text-Retrieval Infrastruktur, aber ohne Datenstrukturen der Vokabulare.
- diverse Error-Modelle der fehlertoleranten Suche: z.B. Edit Distance Varianten
- Der Anwender muss das passende Error-Modell und die Similarity-Measures festlegen, d.h. die Qualität der Ergebnisse.