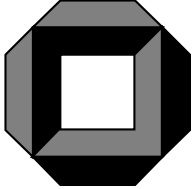


Exkurs

Eine Algorithm-Engineering-Fallstudie

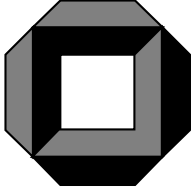
Volltextsuche mit invertiertem Index

Motivation:

Volltextsuchmaschinen wie Google haben die Welt verändert.

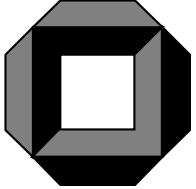
und machen das WWW erst nützlich.

Dazu sind Algorithmen unerlässlich



Algorithmen die Google Ausmachen

- Invertierter Index. Zum Beispiel Informatik Karlsruhe
- Phrasensuche, z.B. “to be or not to be”
- Ranking. PageRank braucht im wesentlichen wiederholte Multiplikation dünn besetzter Matrizen mit Vektoren (iterative Eigenwertberechnung).
- Versteigerungsmechanismen für Werbung.



Konjunktive Volltextsuche

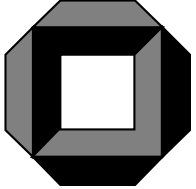
Gegeben:

Dokumente $d_1, \ldots, d_U$

Schlüsselwörter $t_1, \ldots, t_k$ aus Wörterbuch mit W Einträgen.

Gesucht:

IDs ($\in 1..U$) der Dokumente, die $t_1, \ldots, t_k$

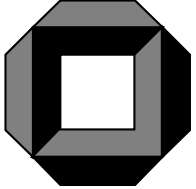


Invertierter Index

Für jedes Wort t berechne eine sortierte Liste $L(t)$ mit den docIDs der Dokumente, die t enthalten. (Entwurfstrategie Vorberechnung)

Ergebnis ist $L(t_1) \cap \cdots \cap L(t_k)$

$\rightsquigarrow$ Reduktion auf Mengenschnitt sortierter Folgen.

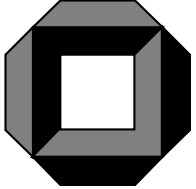


Indexkonstruktion

foreach Document d_i **do**

  foreach term $t \in d$ **do**

    $L(t)$.pushBack(i)



Mengenschnitt sortierter Folgen

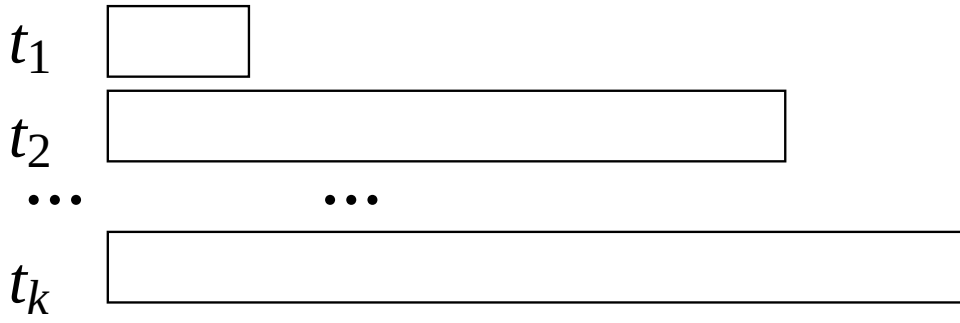
Gut, wenn Folgenlängen sehr unterschiedlich:

Sortiere Suchterme so um, dass

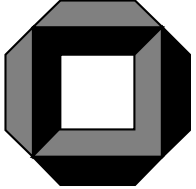
$|L(t_1)| \leq \cdots \leq |L(t_k)|$

berechne

$(\cdots((L(t_1) \cap L(t_2)) \cap L(t_3)) \cdots) \cap L(t_k)$



$\rightsquigarrow$ Reduktion auf Mengenschnitt zweier sortierter Folgen.



Mengenschnitt zweier sortierter Folgen

Function zipper$(\langle \alpha_1, \beta_1, \ldots, \omega_1 \rangle, \langle \alpha_2, \beta_2, \ldots, \omega_2 \rangle)$

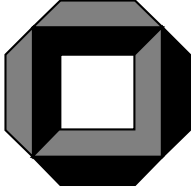
if $\alpha_1 < \alpha_2$ **then return** zipper$(\langle \beta_1, \ldots, \omega_1 \rangle, \langle \alpha_2, \beta_2, \ldots, \omega_2 \rangle)$

if $\alpha_1 > \alpha_2$ **then return** zipper$(\langle \alpha_1, \beta_1, \ldots, \omega_1 \rangle, \langle \beta_2, \ldots, \omega_2 \rangle)$

else return $\langle \alpha_1 \rangle \circ$ zipper$(\langle \beta_1, \ldots, \omega_1 \rangle, \langle \beta_2, \ldots, \omega_2 \rangle)$

Hochoptimierte, nichtrekursive, feldbasierte Variante von zipper

funkioniert sehr gut, solange Längenverhältnis < 10



Mengenschnitt zweier ungleich langer Folgen

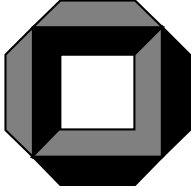
O.B.d.A. sei $m = |L_1| \leq |L_2| = n$ Zipper braucht $O(m+n)$ Zeit.

Vergleichsbasierte untere Schranke (Skizze):

Es gibt $\binom{n}{m}$ verschiedene Möglichkeiten, m Elemente in n Elementen zu positionieren.

$\Rightarrow$ wir brauchen $\log \binom{n}{m}$ Bits Information (durch Vergleiche).

$$\log \binom{n}{m} \geq \log \left(\frac{n}{m}\right)^m = m \log \frac{n}{m}$$



Mengenschnitt zweier ungleich langer Folgen

Obere Schranke

Function exponentialSearchIntersect$(\langle \alpha_1, \beta_1, \ldots, \omega_1 \rangle, L_2)$

suche α_1 in L_2, d.h., A_2 B_2

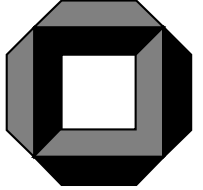
spalte $L_2 = A_2 \circ B_2$ mit

A_2	B_2
$\leq \alpha_1$	$> \alpha_1$

$C =$ exponentialSearchIntersect$(\langle \beta_1, \ldots, \omega_1 \rangle, B_2)$

if $\alpha_1 =$ last(A_2) **then** $C := \langle \alpha_1 \rangle \circ C$

return C



Exponentielle Suche

Function exponentialSearch$(e, a$: **Array** $)$

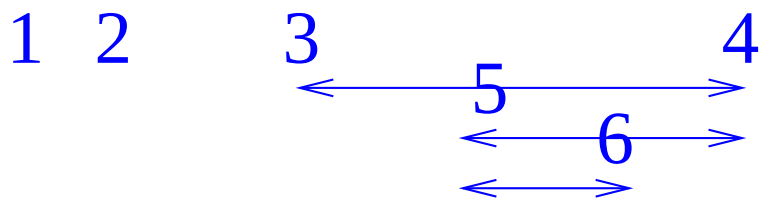
i:= 1

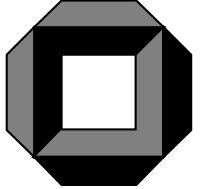
while $e \leq a[i]$ **do** i:= $2i$

now binary search in $a[i/2..i]$ $\rightsquigarrow$ **return** found pos j ...

Zeitaufwand $O(1+\log k)$

```
1  2  3  4  5  6   7   8   9 10 11 12 13 14 15 16 17
1  3  4  5  7  9 11 13 17 19 20 21 22 25 30 34 35
```





Mengenschnitt zweier ungleich langer Folgen

Analyse

exponentialSearchIntersect teilt L_2 in Teillisten der Längen $\Delta_1, \ldots, \Delta_m$.

Aufwand $\sum_{i=1}^{m} \mathrm{O}(1+\log \Delta_i)) = \mathrm{O}\left(m(1+\log \frac{n}{m})\right)$
(Konvexität der Logarithmusfunktion)...

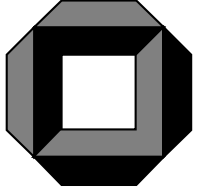
...

Leider sind diese Verfahren in der Praxis ziemlich langsam.

Idee: weitergehende Vorverarbeitung der Listen.

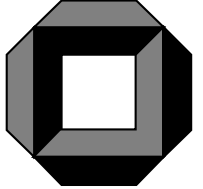
Idee: breche untere Schranke. Nutze aus, dass die Elemente ganze Zahlen sind

$\rightsquigarrow$ mittlere Ausführungszeit $\mathrm{O}(m)$.



Verfeinerungen und Verallgemeinerungen

- Parallele Konstruktion. Zum Beispiel mittels MCSTL? `http://algo2.iti.uni-karlsruhe.de/singler/mcstl/`
- Phrasensuche mittels Positional Index
- Nur Paare bestimmter Suchwörter ermöglichen vernünftige Sucheingrenzung? Vorberechnung von Listen auch für solche Wortpaare.
- Datenkompression



Zusammenfassung

Algorithmenentwurfstechnik Vorverarbeitung: Hier Reduktion auf Manipulaiton von Begriffs- und Dokument-Nummern.

Algorithmenentwurfstechnik Vorberechnung: Hier Inverted Index

Algorithmenentwurfstechnik Basic Toolbox: Hier unbounded array, mischen, Mengenschnitt, exponentielle Suche, binäre Suche, Wörterbücher, sortieren, Datenkompression,. . .