

Datenstrukturen

Text-Indizierung

3. April 2014

basierend auf:

J. Fischer: Text-Indexierung und Information Retrieval

S. Rahmann: Algorithmen auf Sequenzen

Fragestellungen

- gibt es Pattern P in Text T ?
- wie oft kommt P in T vor?
- wo kommt P in T vor?

- Wie schnell?
 - Abhängig von $m=|P|$, $n = |T|$, Alphabet $|\Sigma|$
- Wie viel zusätzlicher Speicherverbrauch?

Suffixarray

	1	2	3	4	5	6	7	8	9	10	11	12	13
T =	c	a	b	c	c	b	a	a	a	b	b	a	\$
A =	13	12	7	8	9	2	11	6	10	3	1	5	4
	\$	a \$	a a a b b a \$	a a b b a \$	a b b a \$	a b c c b a a a b b a \$	b a \$	b a a a b b a \$	b b a \$	b c c b a a a b b a \$	c a b c c b a a b b a \$	c b a a a b b a \$	c c b a a b b a \$

Suffixarray

- Konstruktion
 - Naiv mit Mergesort
 - $O(n \log n)$ Sortierzeit
 - $O(n)$ für Stringvergleiche
 - $\Rightarrow O(n^2 \log n)$
 - Es geht besser in $O(n)$

Suffixarray

- Größe

- Klar: $O(n)$
- Im Verhältnis zu Text?
 - Menschliches Genom $|\Sigma| = 4$; $|T| = 3 \text{ GB} \approx 2^{32}$
 - \Rightarrow 2 Bit pro Base. 1GB Textgröße.
 - \Rightarrow Suffixarray mit 4 Byte pro Eintrag
 - \Rightarrow Textgröße 1 GB, Index 16 GB

Suffixarray

- Anwendungen

- gibt es Pattern P in Text T ?
 - Binäre Suche $O(m \log n)$
 - m = Länge des Substrings, n = Länge des SA
- wie oft kommt P in T vor?
 - Binäre Suche von oben und unten, Abstand messen.
- wo kommt P in T vor?
 - Index hat man schon :)

Approximative Suche

- Levenshtein-Distanz
 - Einfügungen, Löschungen, Ersetzungen zählen.

b	a	n	a	n	a	
	a	n	a	n	a	s

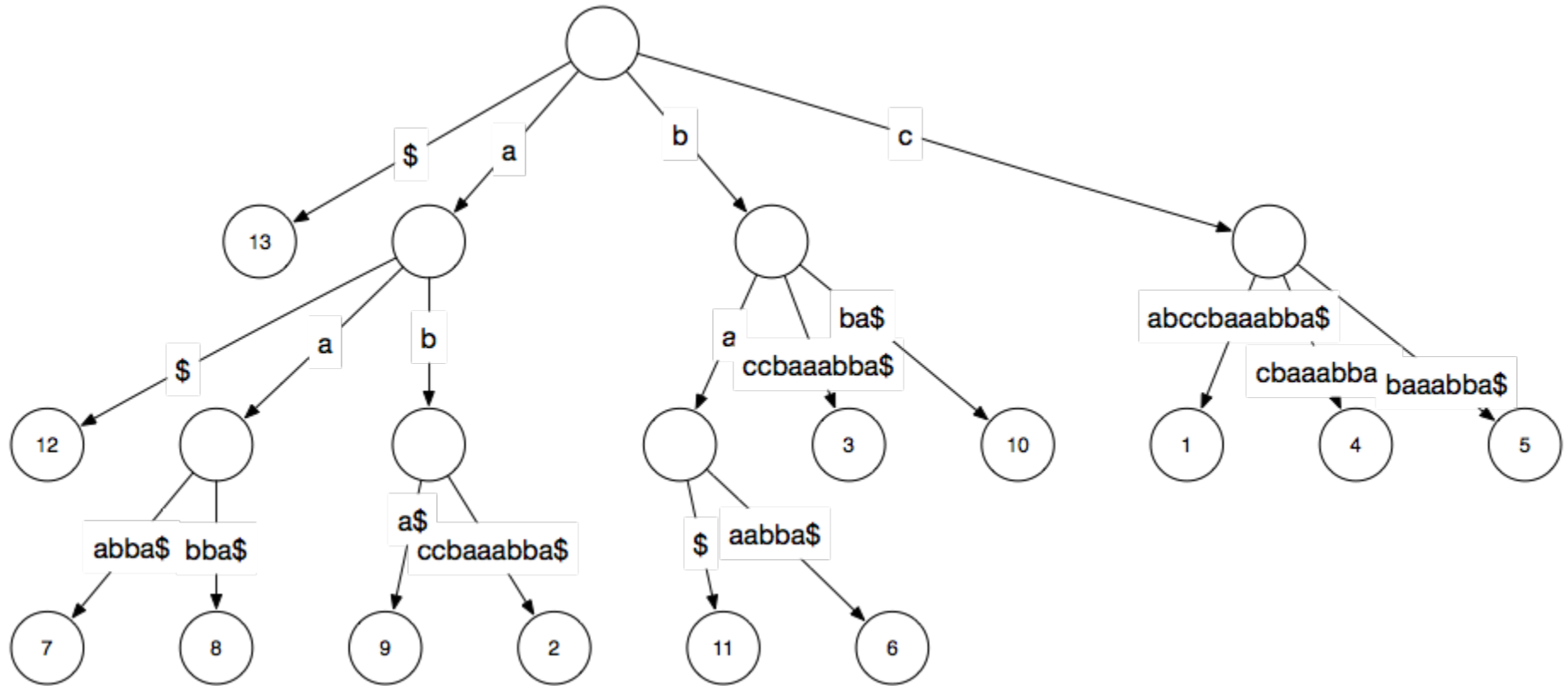
Distanz = 2

- Idee: Erzeuge für Suchstring P alle Strings mit Distanz = 1
- m Löschungen; $|\Sigma|(m+1)$ Einfügungen; $(|\Sigma|-1)m$ Ersetzungen
- Suchzeit $O(m^2 |\Sigma| \log n)$ (kann auf $O(m |\Sigma| \log n)$ verbessert werden)

LCP-Array

	1	2	3	4	5	6	7	8	9	10	11	12	13
T =	c	a	b	c	c	b	a	a	a	b	b	a	\$
A =	13	12	7	8	9	2	11	6	10	3	1	4	5
H =	⊥	0	1	2	1	2	0	2	1	1	0	1	1
	\$	a	a	a	a	a	b	b	b	b	c	c	c
		\$	a	a	b	b	a	a	a	c	a	c	b
			a	b	b	c	\$	a	b	c	b	b	a
			b	b	a	c		a	a	b	c	a	a
			a	a	\$	b		b	a	a	c	a	a
			\$	\$		a		\$		a	a	b	a
						a				b	b	a	\$
						a				a	a	\$	
						a				b	b		
						a				a	b		
						a				b	a		
						a				\$	b		
						a							

Suffixbaum



Suffixbaum

- Anwendungen
 - gibt es Pattern P in Text T ?
 - Baum traversieren.
 - wie oft kommt P in T vor?
 - Baum traversieren. Blätter zählen.
 - wo kommt P in T vor?
 - Index in den Blättern speichern.

Suffixbaum

- Größe?
 - $O(n)$ Knoten (maximal $2n$)
 - n Blätter, $n-1$ interne Knoten, 1 Wurzel
 - Aber: Hoher Aufwand um die Struktur zu speichern
 - 20-40x Textgröße
- Kann mit Suffixarray (und ein paar zusätzlichen Datenstrukturen) simuliert werden.

Burrows-Wheeler-Transformation

	1	2	3	4	5	6	7
T=	b	a	n	a	n	a	\$
A=	7	6	4	2	1	5	3
	b	a	n	a	n	a	\$
	a	n	a	n	a	\$	b
	n	a	n	a	\$	b	a
	a	n	a	\$	b	a	n
	n	a	\$	b	a	n	a
	a	\$	b	a	n	a	n
	\$	b	a	n	a	n	a

Burrows-Wheeler-Transformation

	1	2	3	4	5	6	7
T=	b	a	n	a	n	a	\$
A=	7	6	4	2	1	5	3
	\$	b	a	n	a	n	a
	a	\$	b	a	n	a	n
	a	n	a	\$	b	a	n
	a	n	a	n	a	\$	b
	b	a	n	a	n	a	\$
	n	a	\$	b	a	n	a
	n	a	n	a	\$	b	a

Burrows-Wheeler-Transformation

- Konstruktion in $O(n)$
 - $L[i] = T[A[i]-1]$ $T[0] = T[n]$
- Vorteile
 - Gleiche Buchstaben stehen jetzt häufiger nebeneinander.
 - Da in (natürlichsprachigen) Texten oft die gleichen Buchstabenpaare auftreten.
 - \Rightarrow Möglichkeiten zur Kompression

BWT + Run Length Encoding

$L = \text{annb\$aa}$

$\text{RLE}(L) = (a, 1) (n, 2) (b, 1) (\$, 1) (a, 2)$

In diesem Fall kein Gewinn

BWT + Move to Front

L = annb\$aa

\$	a	b	n		1
a	\$	b	n		3
n	a	\$	b		0
b	n	a	\$		3
\$	b	n	a		3
a	\$	b	n		3
					0

Ausgabe
1303330

inverse BWT

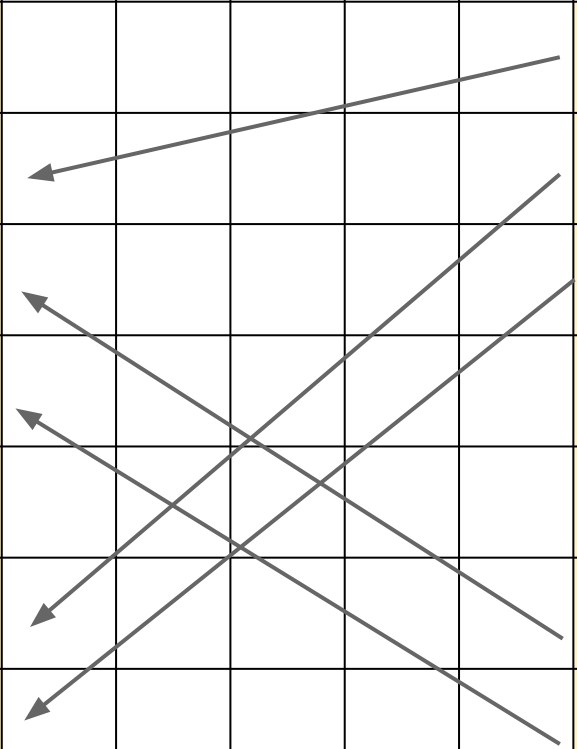
	F						L
1	\$						a
2	a						n
3	a						n
4	a						b
5	b						\$
6	n						a
7	n						a



inverse BWT

Gleiche Buchstaben kommen in L und F in gleicher Reihenfolge vor

	F						L	LF
1	\$						a	2
2	a						n	6
3	a						n	7
4	a						b	5
5	b						\$	1
6	n						a	3
7	n						a	4



inverse BWT

Gleiche Buchstaben kommen in L und F in gleicher Reihenfolge vor

$T[n] = \$$
 $T[n-1] = L[1]$
 $T[n-2] = L[LF(1)]$
 $T[n-i] = L[LF(LF(\dots(LF(1))\dots))]$
Wende LF $i-1$ mal an

	F						L	LF
1	\$						a	2
2	a						n	6
3	a						n	7
4	a						b	5
5	b						\$	1
6	n						a	3
7	n						a	4

FM-Index

- Definitionen

- $OCC(a, i)$: Anzahl der 'a's in $L[1, i]$
- $C[a]$: Anzahl der Buchstaben, die lexikographisch kleiner als 'a' sind.

Beispiel: banana\$

	\$	a	b	n
C=	0	1	4	5

- $LF(i)$: Last-To-Front-Mapping (BWT)
 $LF(i) = C[L[i]] + OCC(L[i], i)$

FM-Index

- Idee: Suche das Pattern rückwärts in L

	1	2	3	4	5	6	7
T=	b	a	n	a	n	a	\$
A=	7	6	4	2	1	5	3

P = ban

L=	a	n	n	b	\$	a	a
----	---	---	---	---	----	---	---

FM-Index

- Idee: Suche das Pattern rückwärts in L

	1	2	3	4	5	6	7
T=	b	a	n	a	n	a	\$
A=	7	6	4	2	1	5	3

n

P = ban

L=	a	n	n	b	\$	a	a
----	---	---	---	---	----	---	---

FM-Index

- Idee: Suche das Pattern rückwärts in L

	1	2	3	4	5	6	7
T=	b	a	n	a	n	a	\$
A=	7	6	4	2	1	5	3

n

P = ban

an

L=	a	n	n	b	\$	a	a
----	---	---	---	---	----	---	---

FM-Index

- Idee: Suche das Pattern rückwärts in L

	1	2	3	4	5	6	7
T=	b	a	n	a	n	a	\$
A=	7	6	4	2	1	5	3

n

P = ban

an

ban

L=	a	n	n	b	\$	a	a
----	---	---	---	---	----	---	---

FM-Index

- Idee: Suche das Pattern rückwärts in L

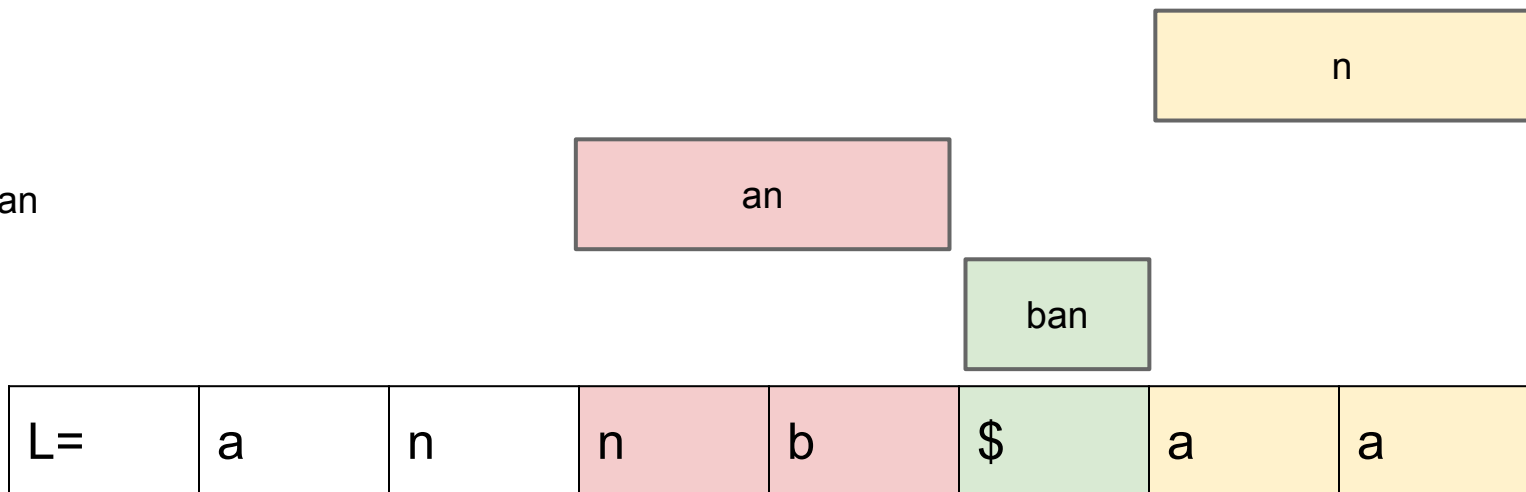
$s = 1$

$e = n$

$s = C[P[i]] + OCC(P[i], s - 1) + 1$

$e = C[P[i]] + OCC(P[i], e)$

$P = \text{ban}$

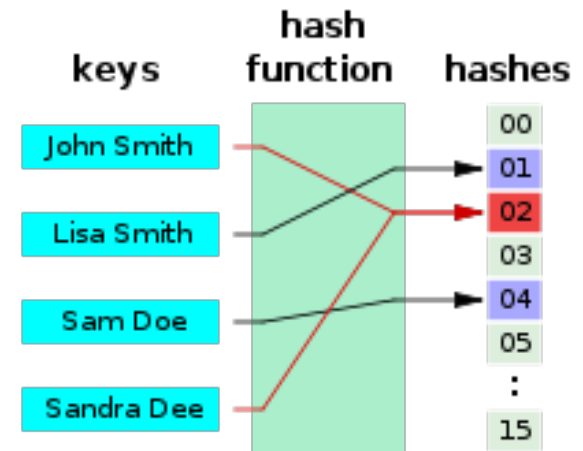


FM-Index

- gibt es Pattern P in Text T ?
 - Gerade gesehen. Laufzeit und Größe hängt direkt von OCC ab.
Geht in Textgröße und $O(n \log |\Sigma|) = O(n)$
- wie oft kommt P in T vor?
 - Bekommt man direkt dazu.
- wo kommt P in T vor?
 - Zusätzlicher Aufwand. Zum Beispiel mit gesampeltem Suffixarray. (+ Textgröße)

Hashing

Allgemein: Hashfunktion h ordnet jedem beliebig langem String s einen String $h(s)$ *fester Länge* zu.



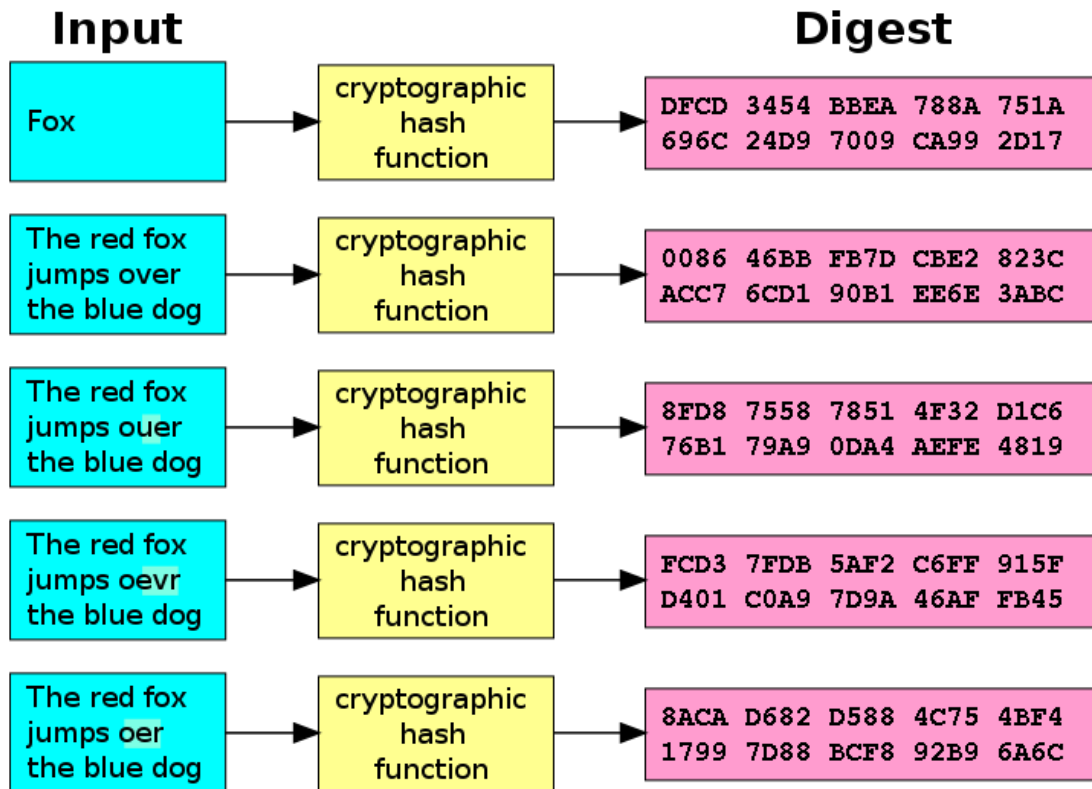
Hashfunktion: Anwendungen

- Assoziatives Array
 - Hashfunktion berechnen
 - Index als Hash modulo Arraygröße berechnen

Hashfunktion: Anwendungen

- Kryptographische Hashfunktion
- Wird zur Authentifizierung oder Signierung benutzt
- Bereits kleine Änderungen bei der Eingabe erzeugen sehr unterschiedliche Ausgabe

SHA-1



Hashfunktionen: Anwendungen

- Suche nach ähnlichen Datensätzen

Idee: eine Art Gegenteil der kryptographischen Hashfunktion

- “*Ähnliche*” Datensätze bekommen gleichen Hashwert

Szenenvervollständigung



Andere Anwendungen

- Ähnliche Internetseiten
- Allgemein: Ähnliche Texte

Idee

- Hochdimensionalen Vektor \mathbf{x} aus einem Dokument konstruieren
- Anschließend alle Dokumente suchen, die entsprechend einer Distanzfunktion $d(\mathbf{x}, \mathbf{y})$ nahe beieinander liegen
 - $d(\mathbf{x}, \mathbf{y}) \leq s$
- Hat eine Laufzeit von $O(n^2)$, aber kann durch Hashing auf $O(n)$ reduziert werden

Hochdimensionaler Vektor

- Bei einem Bild beispielsweise die Folge aller Pixelfarben
- Wie kann man möglichst effizient Dokumente repräsentieren?

q-gram-Index

AGGTAGATGATA, $q = 2$

AG	1, 5
GG	2
GT	3
TA	4, 11
GA	6, 9
AT	7, 10
TG	8

q-gram-Index

AGGTAGATGATA, $q = 2$

q-Gram	Index
AG	1, 5
GG	2
GT	3
TA	4, 11
GA	6, 9
AT	7, 10
TG	8

Beispiel: Suche nach AGA

$\text{Index(AG)} = 1, 5$

$\text{Index(GA)} = 6, 9$

=> AG an Stelle 5 und GA an Stelle 6
passen

q-gram

In diesem Fall war ein Token eine Nukleinbase

- Allgemein kann bei Dokumenten ein Token z.B. aber auch ein ganzes Wort sein
- Lange q-Gramme können durch Hash repräsentiert werden

Dokumente als Menge

- Ein Dokument kann als Menge von q -Grammen repräsentiert werden
- In Form eines binären Vektors
- Dokumente mit vielen gemeinsamen q -Grammen sind sich ähnlich, auch wenn der Text nicht in der gleichen Reihenfolge vorkommt
- Wahl von q muss geschickt getroffen werden

Ähnlichkeit von Dokumenten

Jaccard-Koeffizient

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Beispiel: A = 10111; B = 10011

=> $J(A, B) = 3/4$

q-Gram	A	B
00	0	1
01	1	1
10	1	1
11	1	1

Problem: Matrizen sind in der Praxis nur dünn belegt

Daher: Signaturen von Spalten berechnen

Ähnliche Signaturen \Leftrightarrow Ähnliche Spalten

q-Gram	A	B
00	0	1
01	1	1
10	1	1
11	1	1

Idee

- Hashfunktion h , für die
 - $h(C) = h(D)$, wenn $J(C, D)$ groß
 - $h(C) \neq h(D)$, wenn $J(C, D)$ klein
- Ähnliche Dokumente auf gleiche Hashwerte abbilden

=> Min-Hashing

Min-Hashing

- Berechne zufällige Permutation π der Zeilen
- Hashfunktion $h_{\pi}(C)$ = Index der ersten Zeile gemäß π , an der die Spalte C den Wert 1 hat
$$h_{\pi}(C) = \min_{\pi} \pi(C)$$
- Einige (z.B. 100) verschiedene Hashfunktionen ergeben die Signatur einer Spalte

2nd element of the permutation
is the first to map to a 1

Permutation π

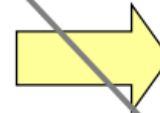
Input matrix (Shingles x Documents)

Signature matrix M

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

2	1	2	1
2	1	4	1
1	2	1	2



4th element of the permutation
is the first to map to a 1

Eigenschaft

- Man kann zeigen:

$$W(h_{\pi}(C) = h_{\pi}(D)) = J(C, D)$$

- Eine Zeilenpermutation einer praktischen Matrix lässt sich nicht effizient berechnen
- Daher: die Reihenfolge wird implizit auch wieder mit Hashfunktionen berechnet

Hashfunktion für Permutation

1. Initialisiere die Signaturmatrix überall mit ∞
2. Für eine Zeile r:
3. Wenn r in Spalte c eine 1 hat:
4. Für alle Hashfunktionen:
5. Wenn der berechnete Wert der Hashfunktion kleiner ist als der aktuelle Wert in der Signaturmatrix, dann schreibe ihn in die Signaturmatrix

Locality Sensitive Hashing

- Bisher: Signaturmatrix berechnet aber es ist immer noch zu aufwendig, alle möglichen Signaturpaare zu vergleichen
- Daher: Erneutes Hashing

Locality Sensitive Hashing

- Idee: Die Signaturmatrix wird zeilenweise in gleichgroße Bänder aufgeteilt
- Eine globale Hashfunktion wird auf jede Teilsignatur angewendet
 - Gleiche Teilsignatur wird auf gleichen Wert abgebildet
 - Bekommen zwei verschiedene Signaturen den gleichen Hashwert, dann sind sie Kandidaten für einen Gleichheitstest

band 1

...	1 0 0 2	...
	3 2 1 2 2	
	0 1 3 1 1	

band 2

band 3

band 4

Locality Sensitive Hashing

- Sind sich zwei Dokumente ähnlich, so muss nicht zwingendermaßen jedes Band einen Kandidaten erzeugen
- Aber: Wegen der Ähnlichkeit ist es wahrscheinlich, dass in einem anderen Band ein Kandidat erzeugt wird
- In der Praxis muss die Bandgröße noch geschickt ausgewählt werden

Quellen

Mining of Massive Dataasets, Kapitel 3

<http://infolab.stanford.edu/~ullman/mmds/ch3.pdf>

außerdem:

<http://matthewcasperson.blogspot.de/2013/11/minhash-for-dummies.html>

<http://www.stanford.edu/class/cs276b/handouts/minhash.ppt>

<http://nlp.stanford.edu/IR-book/html/htmledition/k-gram-indexes-for-wildcard-queries-1.html>

http://en.wikipedia.org/wiki/Locality_sensitive_hashing

http://www.informatik.hu-berlin.de/forschung/gebiete/wbi/teaching/archive/ws0910/ue_algbio/aufgabe3.pdf

www.stanford.edu/class/cs246/slides/03-lsh.pdf