Indexstrukturen in Datenbanken

für Zeichendaten und Texte sowie mehrdimensionale Dateiorganisation und Zugriffspfade

Mario Wenzel

3. Juli 2014

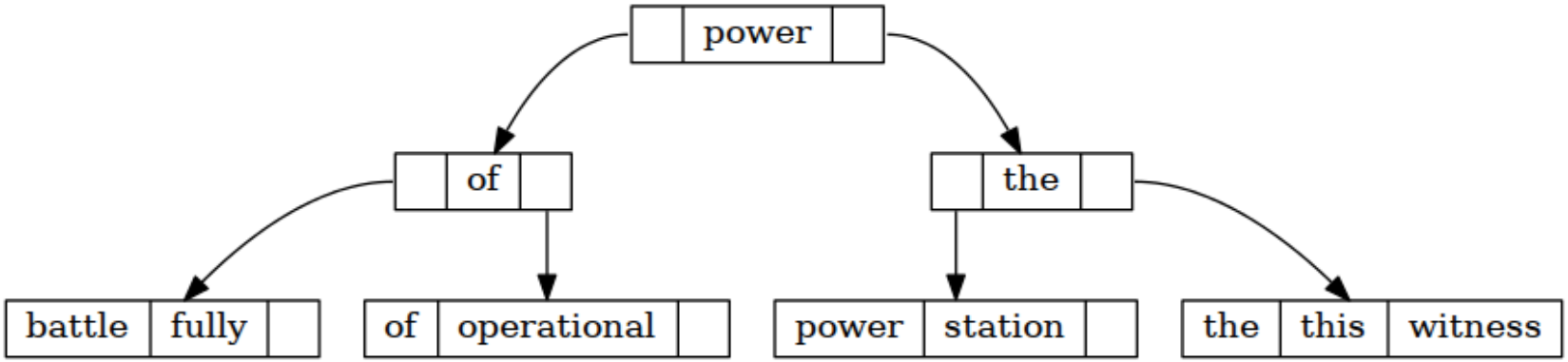
Inhaltsverzeichnis

Was wollen wir mit Indexstrukturen erreichen und welche Eingenschaften sollten sie haben?

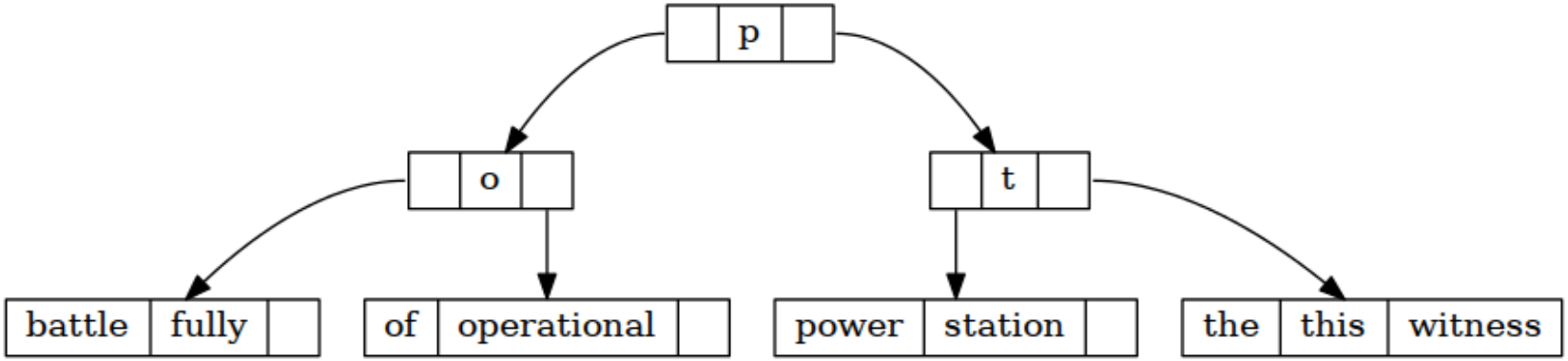
- Effizientes Suchen
- Wenig Reorganisation/wenig Wartungsaufwand haben
- Sortierung erhalten für Range-Abfragen
- Nicht zu viel Speicher nutzen
- Schnell und fehlerfrei implementierbar sein

B+-Bäume sind gut, aber:

- Schlüsselwerte stehen in den Knoten. Diese können besonders lang sein, besonders für Zeichenketten.



Besser: Nur Präfixe in den Knoten. Prefix ist nur genau so lang, wie er sein muss.



Besonders bei vielen Seperatoren pro Knoten macht sich kurze Schlüssellänge besonders bemerkbar.
String-Vergleich ist eh ein Prefix-Vergleich. Das Finden das passenden Kind-Knotens kostet also nicht extra.

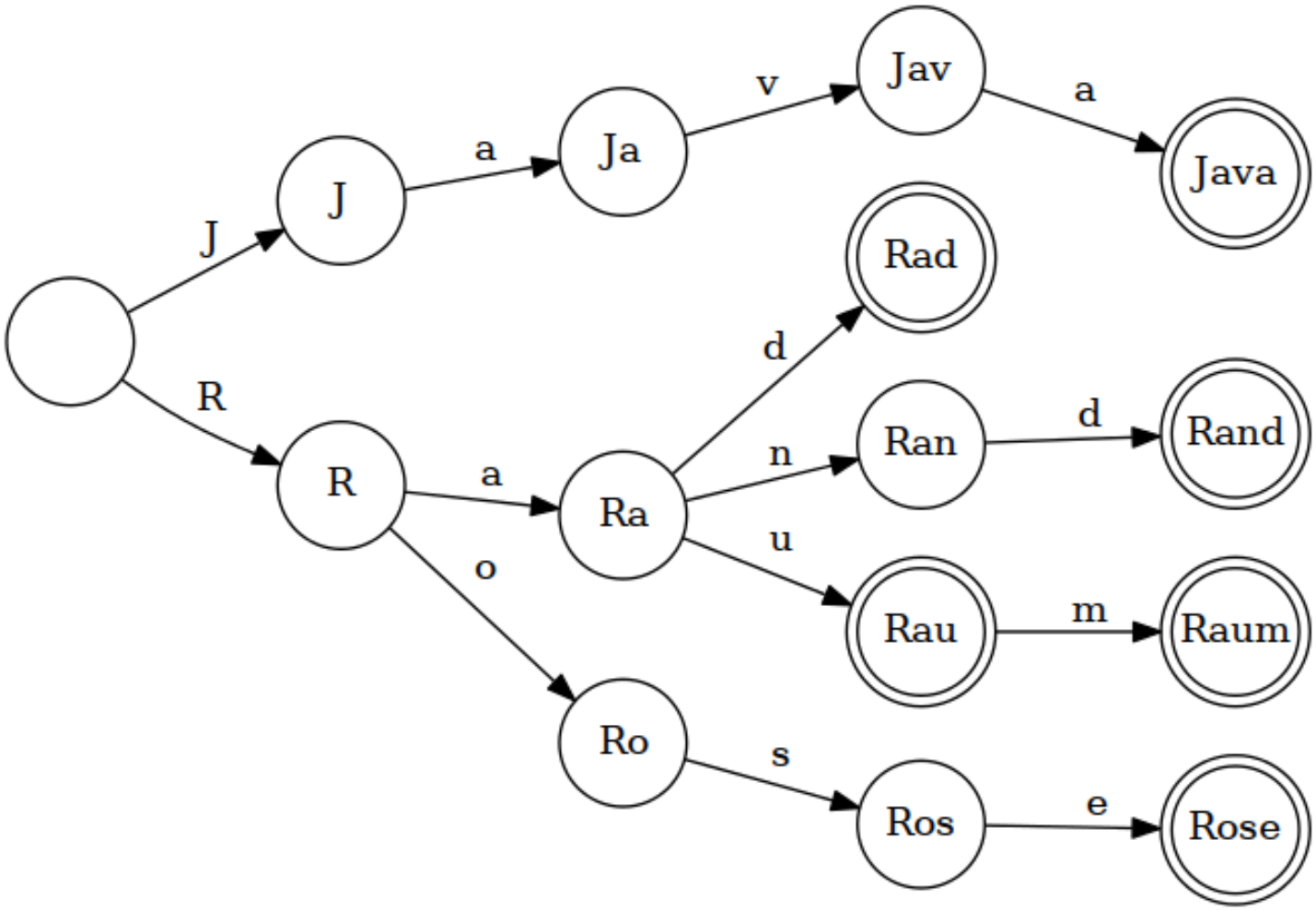
Trie/Präfixbaum

Datenstruktur zum Abspeichern vieler Zeichenketten.

- Wurzel ist das leere Wort bzw. längster Prefix aller Wörter
- Knoten haben Prefix oder sind leer (Prefix ergibt sich dann durch Verkettung)
- Kanten haben nächstes Zeichen
- Blätter sind ganze Wörter oder Daten

Trie von reTrieval

Trie/Präfixbaum



Trie/Präfixbaum

Nachteile:

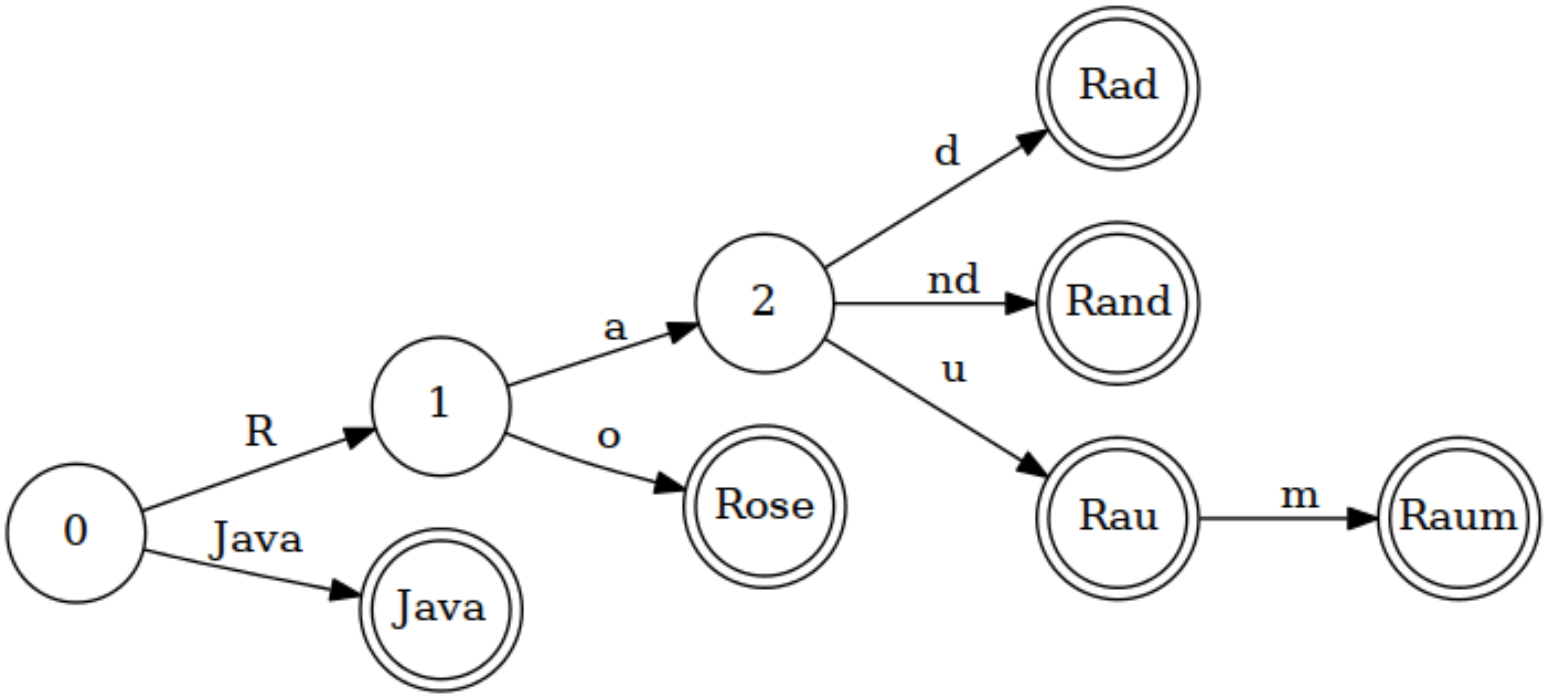
- Datenstruktur kann sehr tief werden denn
  Tiefe = länge(längstes Wort)
- Viele Knoten/Kanten überflüssig bei Wegen ohne viele Verzweigungen
- zwei beliebig lange Wörter (Längend n, m) $\rightarrow$ zwischen max(n, m) und n + m Knoten
- jedes weitere Wort $\rightarrow$ maximal l Knoten

Idee: Patricia-Trie

Patricia-Trie

- Knoten mit nur einem Kind werden entfernt
- An Kanten steht nicht nur ein Buchstabe sondern alle bis zur nächsten Verzweigung

Patricia-Trie



Patricia-Trie

- Geringe Tiefe selbst bei langen Wörtern
- zwei beliebig lange Wörter (Längend n, m) $\rightarrow$ 3 Knoten
- jedes weitere Wort $\rightarrow$ maximal +2 Knoten

Homogener k-d-Baum

- k-dimensionaler binärer Suchbaum
- Gesamtschlüssel im Index ist Zusammensetzung aus k Teilschlüsseln
- Aufbau wie Binärbaum nur ein weiteres Feld ("Diskriminator Teilschlüssel zum Unterscheiden)
- Diskriminatoren in Reihenfolge nach Tiefe

Homogener k-d-Baum

Folgende Knoten an der Tafel:

	X	Y
A	60	40
B	75	70
C	30	60
D	45	30
E	90	10
F	45	85
G	40	70

Homogener k-d-Baum

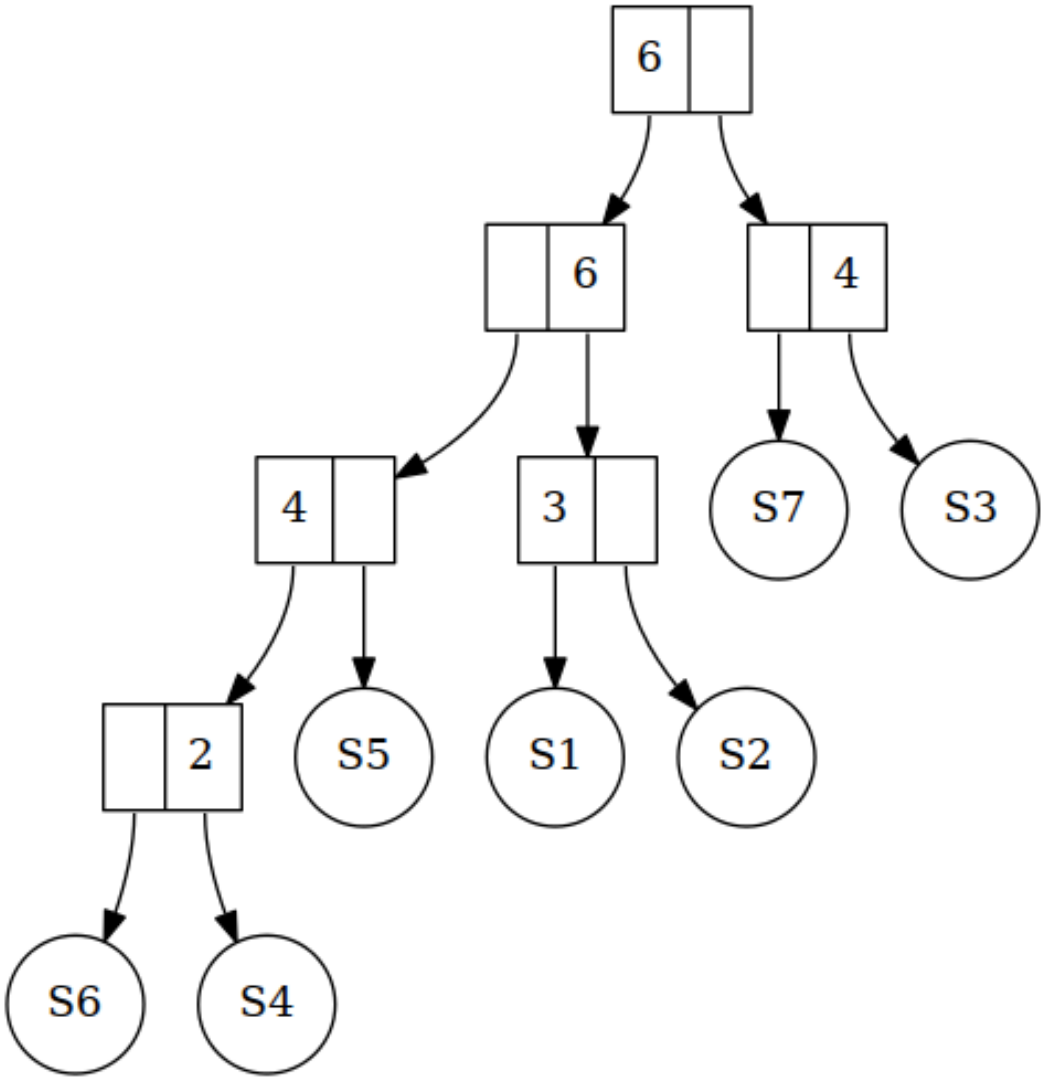
Nachteile:

- Keine Balancierung
- Keine Bereichsanfragen oder Anfrage nach Teilschlüsseln (nur mit rekursivem Aufruf)

Heterogener k-d-Baum

- k-dimensionaler binärer Suchbaum
- Keine Datensätze in Ästen
- weiterhin mit "Diskriminator"
- Datenraum wird geteilt (Divide and Conquer)

Heterogener k-d-Baum



Heterogener k-d-Baum

Nachteile:

- Keine Balancierung
- Keine Bereichsanfragen oder Anfrage nach Teilschlüsseln (nur mit rekursivem Aufruf)

k-d-B*Baum

- Mischung aus heterogenem k-d-Baum und B*Baum
- Aufwändige Balancierung unter Beibehaltung des k-d Baum-Prinzips
- aber: durch Balancierung gleichschneller Zugriff auf alle Datensätze

Grid-File

- Zerlegen des Datenraums in k-dimensionale Quader (Zellen)
- Jede Zelle ein Bucket (benachbarte Zellen können sich Buckets teilen)
- Bei überlauf werden alle Zellen einer Dimensionsachse geteilt
- Exact-Match Anfragen brauchen nur zwei Zugriffe

Grid-Directory

- Grid besteht aus k eindimensionalen Feldern (Skalen). Dies sind die Achsen des Datenraums. Jede Skala ist ein Attribut (Teil des Schlüssels, Schlüsseldimension.
- Skala ist die Partition des zugehörigen Wertebereichs. Bsp.: $[0, 500, 750, 1000]$
- Grid-Directory ist k-dimensionales Array (zugriff über Skala). Eine Zelle zeigt auf eine Seite
- Grid-Region: mehrere Zellen können auf die selbe Seite zeigen (sind dann eine Region)

Grid-File

Beispiel zum Gridfile an der Tafel

A_1	A_2
45	D
02	A
87	S
75	M
55	K
03	Z
15	D
25	K
48	F

Grid-File

Nachteile:

- Grid-Directory wächst nichtlinear
- Zellen können leer sein, da neue Zellen durch die gemeinsame Nähe in nur einer der k Dimensionen entstehen

mehrdimensionales Hashing

Durch Bit-interleaving werden mehrere Dimensionen angebunden
Beispiel:

	*0*0	*0*1	*1*0	*1*1
0*0*	0000	0001	0100	0101
0*1*	0010	0011	0110	0111
1*0*	1000	1001	1100	1101
1*1*	1010	1011	1110	1111

Quellen

- Datenbanken: Implementierungstechniken (Saake, Heuer)
- Wikipedia (Beispiel Trie)