Algorithmen und Datenstrukturen 1

12. Vorlesung

Peter F. Stadler

Universität Leipzig
Institut für Informatik
studla@bioinf.uni-leipzig.de

Schlüsselkomprimierung I

Zeichenkomprimierung ermöglicht weit höhere Anzahl von Einträgen pro Seite (v.a. bei B*-Baum)

- Verbesserung der Baumbreite (höherer Fan-Out)

  Fan-out (Elektronik) = Zahl der anschließbaren Gatter an ein gegebenes Gatter

- wirkungsvoll v.a. für lange, alphanumerische Schlüssel (z.B. Namen)

Präfix-Komprimierung

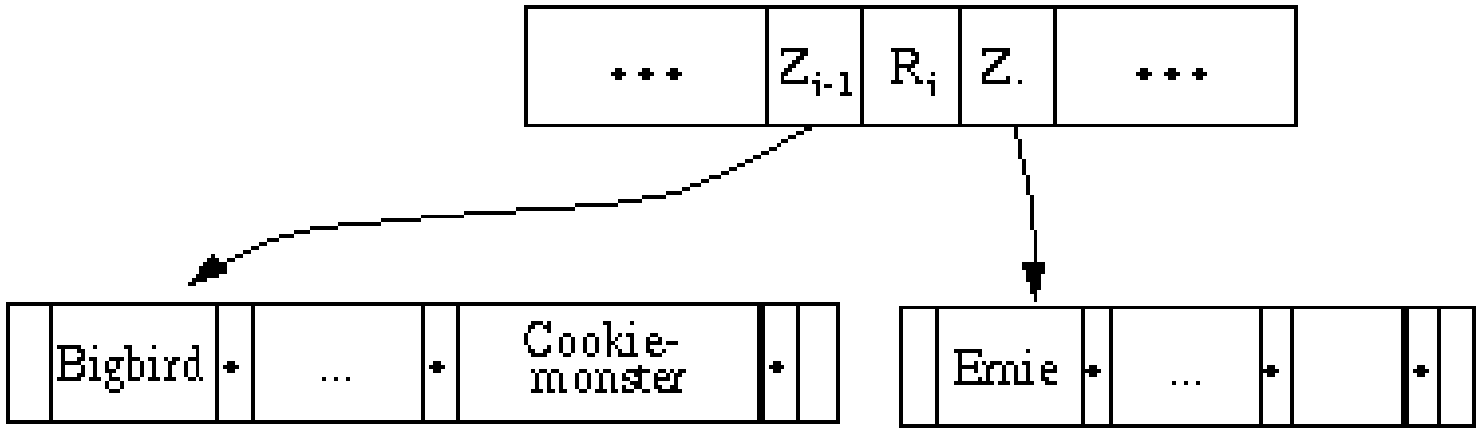
- mit Vorgängerschlüssel übereinstimmender Schlüsselanfang (Präfix) wird nicht wiederholt
- v.a. wirkungsvoll für Blattseiten
- höherer Aufwand zur Schlüsselrekonstruktion

Schlüssel	F	kompr. Schlüssel
HARALD	*0*	*HARALD*
HARTMUT	*3*	*TMUT*
HEIN	*1*	*EIN*
HEINRICH	*4*	*RICH*
HEINZ	*4*	*Z*
HELMUT	*2*	*LMUT*
HOLGER	*1*	*OLGER*

Schlüsselkomprimierung II

Suffix-Komprimierung

- für innere Knoten ist vollständige Wiederholung von Schlüsselwerten meist nicht erforderlich, um Wegweiserfunktion zu erhalten
- Weglassen des zur eindeutigen Lokalisierung nicht benötigten Schlüsselendes (Suffix)
- Präfix-B-Bäume: Verwendung minimale Separatoren (Präfixe) in inneren Knoten



Schlüsselkomprimierung III

Für Zwischenknoten kann Präfix- und Suffix-Komprimierung kombiniert werden:

Präfix-Suffix-Komprimierung (Front and Rear Compression)

- gespeichert werden nur solche Zeichen eines Schlüssels, die sich vom Vorgänger und Nachfolger unterscheiden
- u.a. in VSAM eingesetzt

Schlüssel	V	N	F	L	kompr. Schlüssel
HARALD	*1*	*4*	*0*	*4*	*HARA*
HARTMUT	*4*	*2*	*3*	*0*	
HEIN	*2*	*5*	*1*	*4*	*EIN \0*
HEINRICH					
HEINZ					
HELMUT					
HOLGER					

Verfahrensparameter:

V = Position im Schlüssel, in der sich der zu komprimierende Schlüssel vom Vorgänger unterscheidet

N = Position im Schlüssel, in der sich der zu komprimierende Schlüssel vom Nachfolger unterscheidet

F = V - 1 (Anzahl der Zeichen des komprimierten Schlüssels, die mit dem Vorgänger übereinstimmen)

L = MAX (N-F, 0) Länge des komprimierten Schlüssels

Durchschnittliche komprimierte Schlüssellänge ca. 1.3 - 1.8

Bsp.: Präfix-Suffix-Komprimierung

Schlüssel (unkomprimiert)		*V*	*N*	*F*	*L*	*Wert*
CITY_OF_NEW_ORLEANS	... GUTHERIE, ARLO	1	6	0	6	CITY_O
CITY_TO_CITY	... RAFFERTTY, GERRY	6	2	5	0	
CLOSET_CHRONICLES	... KANSAS	2	2	1	1	L
COCAINE	... CALE, J.J	2	3	1	2	OC
COLD_AS_ICE	... FOREIGNER	3	6	2	4	LD_A
COLD_WIND_TO_WALHALLA	... JETHRO_TULL	6	4	5	0	
COLORADO	... STILLS, STEPHEN	4	5	3	2	OR
COLOURS	... DONOVAN	5	3	4	0	
COME_INSIDE	... COMMODORES	3	13	2	11	ME_INSIDE__
COME_INSIDE_OF_MY_GUITAR	... BELLAMY_BROTHERS	13	6	12	0	
COME_ON_OVER	... BEE_GEES	6	6	5	1	O
COME_TOGETHER	... BEATLES	6	4	5	0	
COMING_INTO_LOS_ANGELES	... GUTHERIE, ARLO	4	4	3	1	I
COMMOTION	... CCR	4	4	3	1	M
COMPARED_TO_WHAT?	... FLACK, ROBERTA	4	3	3	0	
CONCLUSION	... ELP	3	4	2	2	NC
CONFUSION	... PROCOL_HARUM	4	1	3	0	

2-3-Bäume

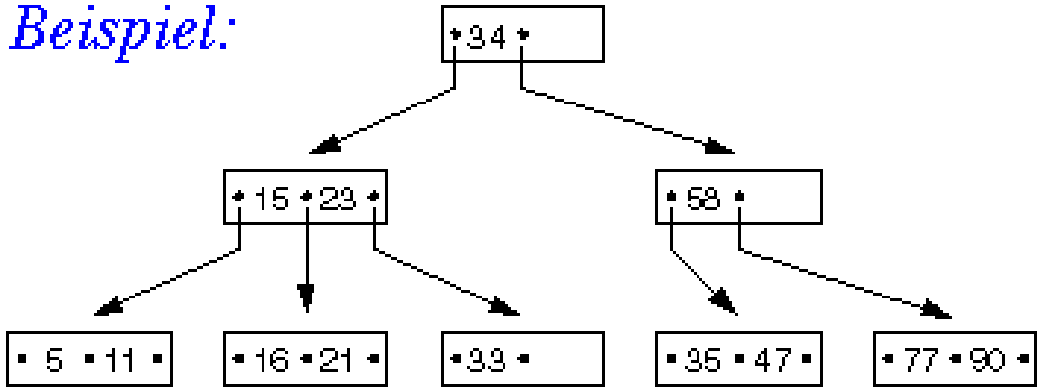
B-Bäume können auch als Hauptspeicher-Datenstruktur verwendet werden

- möglichst kleine Knoten wichtiger als hohes Fan-Out
- 2-3 Bäume: B-Bäume der Klasse (1,h), d.h. mit minimalen Knoten
- Also: Es gelten alle für den B-Baum entwickelten Such- und Modifikationsalgorithmen

Ein 2-3-Baum ist ein m-Wege-Suchbaum (m=3), der entweder leer ist
oder die Höhe h ≥ 1 hat und folgende Eigenschaften besitzt:

- Alle Knoten haben einen oder zwei Einträge (Schlüssel).
- Alle Knoten außer den Blattknoten besitzen 2 oder 3 Söhne.
- Alle Blattknoten sind auf derselben Stufe.

Beispiel:

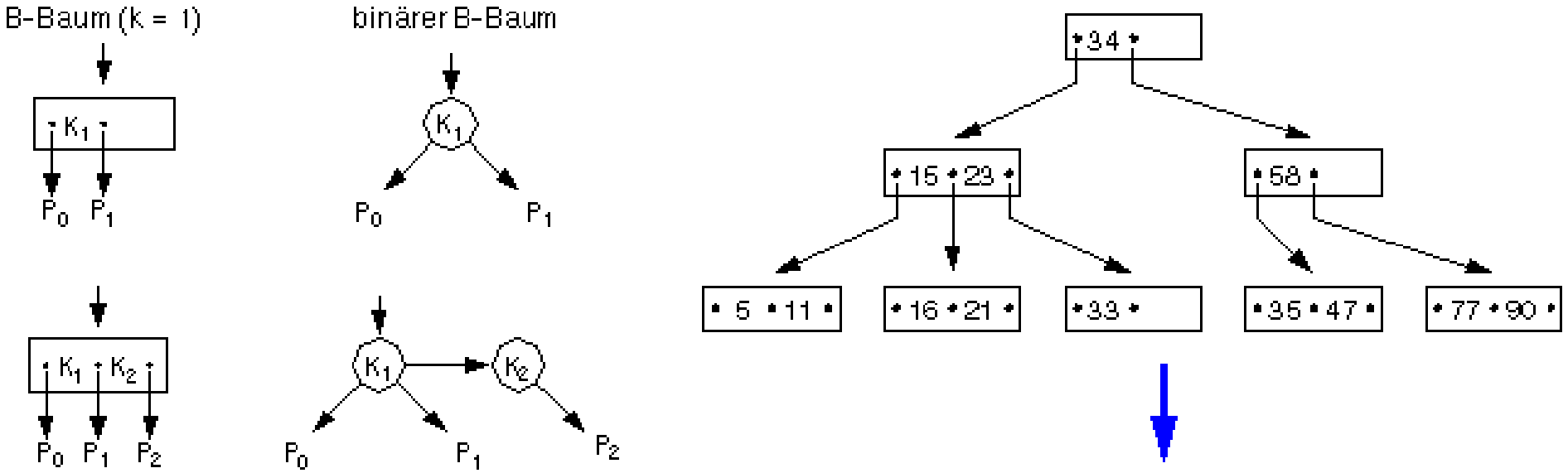


Beobachtungen

- 2-3-Baum ist balancierter Baum
- ähnliche Laufzeitkomplexität wie AVL-Baum
- schlechte Speicherplatznutzung (besonders nach Höhenänderung)

Binäre B-Bäume

- Verbesserte Speicherplatznutzung gegenüber 2-3-Bäumen durch Speicherung der Knoten als gekettete Listen mit einem oder zwei Elementen:



- Variante: symmetrischer binärer B-Baum

Digitale Suchbäume I

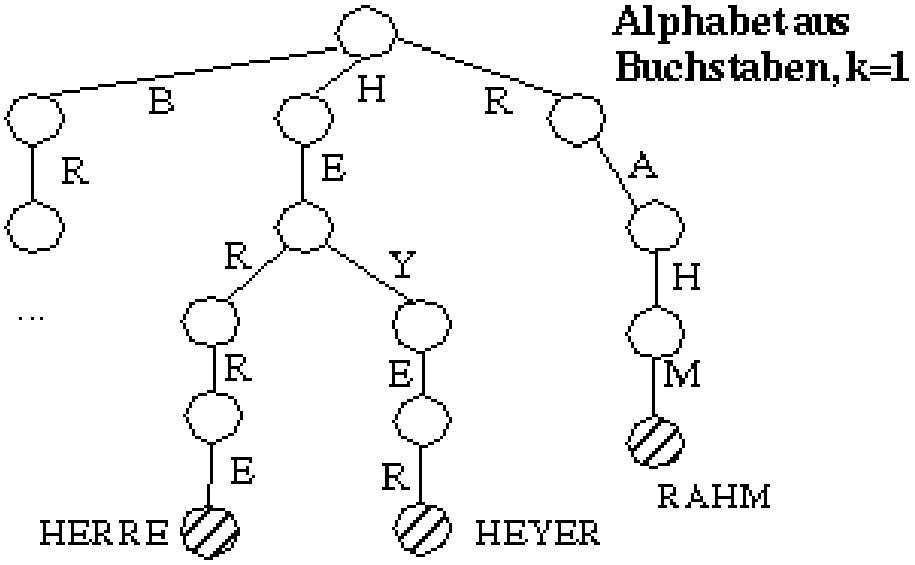
Prinzip digitaler Suchbäume (kurz: Digitalbäume)

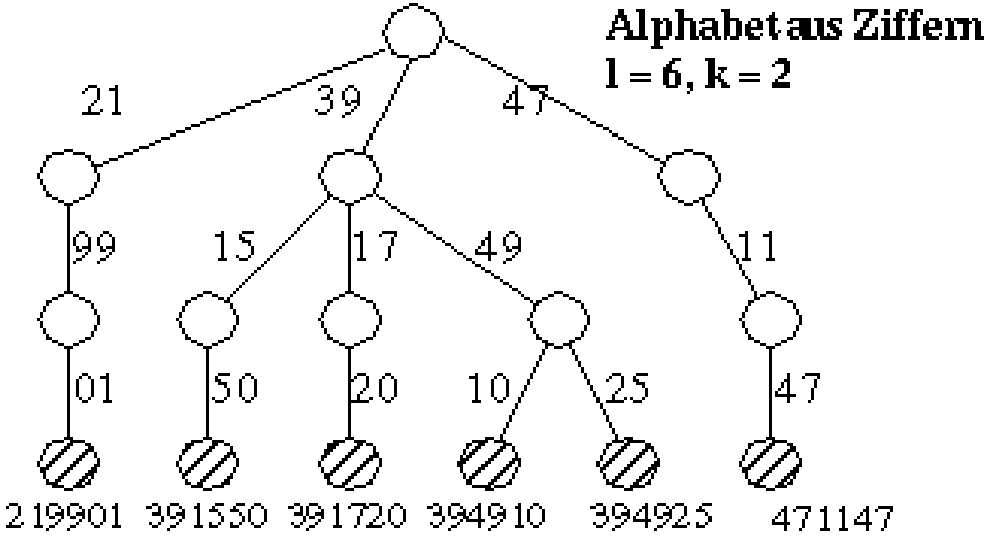
- Zerlegung des Schlüssels - bestehend aus Zeichen eines Alphabets - in Teile
- Aufbau des Baumes nach Schlüsselteilen
- Suche im Baum durch Vergleich von Schlüsselteilen
- jede unterschiedliche Folge von Teilschlüsseln ergibt eigenen Suchweg im Baum
- alle Schlüssel mit dem gleichen Präfix haben in der Länge des Präfixes den gleichen Suchweg
- vorteilhaft u.a. bei variabel langen Schlüsseln, z.B. Strings

Digitale Suchbäume II

Was sind Schlüsselteile ?

- Schlüsselteile können gebildet werden durch Elemente (Bits, Ziffern, Zeichen) eines Alphabets oder durch Zusammenfassungen dieser Grundelemente (z. B. Silben der Länge k)
- Höhe des Baumes = l/k + 1, wenn l die max. Schlüssellänge und k die Schlüsselteillänge ist





m-ärer Trie I

Spezielle Implementierung des Digitalbaumes: Trie

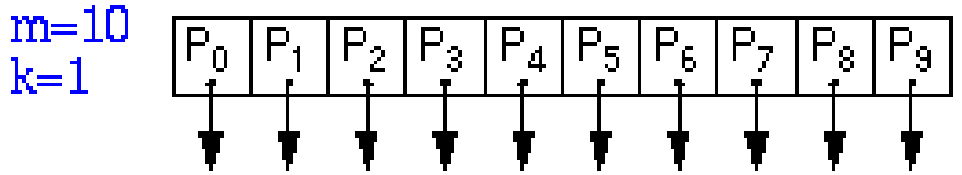
- Trie leitet sich von Information Re*trie*val ab (E.Fredkin, 1960)

Spezielle m-Wege-Bäume, wobei Kardinalität des Alphabets und Länge k der Schlüsselteile den Grad m festlegen

- bei Ziffern: m = 10
- bei Alpha-Zeichen: m = 26; bei alphanumerischen Zeichen: m = 36
- bei Schlüsselteilen der Länge k potenziert sich Grad entsprechend, d. h. als Grad ergibt sich m^k

Trie-Darstellung

- Jeder Knoten eines Tries vom Grad m ist im Prinzip ein eindimensionaler Vektor mit m Zeigern
- Jedes Element im Vektor ist einem Zeichen (bzw. Zeichenkombination) zugeordnet. Auf diese Weise wird ein Schlüsselteil (Kante) implizit durch die Vektorposition ausgedrückt.
- Beispiel: Knoten eines 10-ären Trie mit Ziffern als Schlüsselteilen



- implizite Zuordnung von Ziffer/Zeichen zu Zeiger.

  P_i gehört also zur Ziffer i. Tritt Ziffer i in der betreffenden Position auf, so verweist P_i auf den Nachfolgerknoten. Kommt i in der betreffenden Position nicht vor, so ist P_i mit NULL belegt

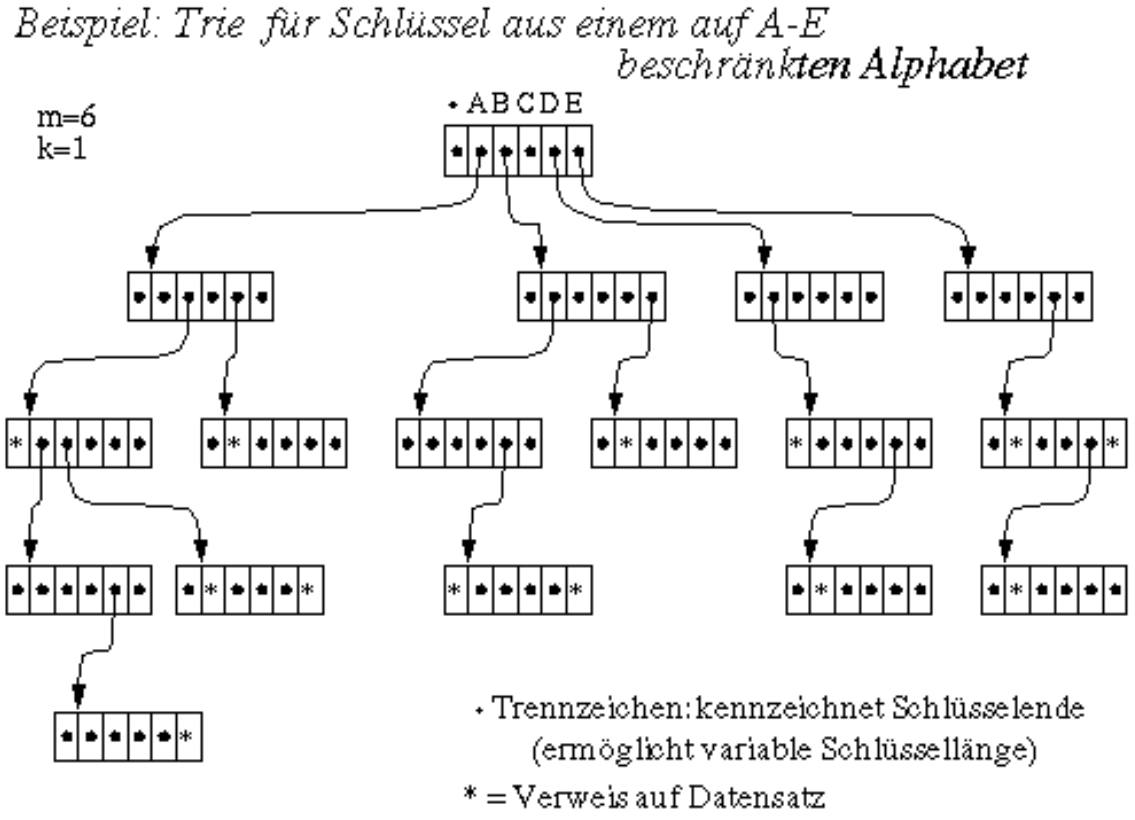
- Wenn der Knoten auf der j-ten Stufe eines 10-ären Trie liegt, dann zeigt P_i auf einen Unterbaum, der nur Schlüssel enthält, die in der j-ten Position die Ziffer i besitzen

Grundoperationen auf Tries I

Direkte Suche:

In der Wurzel wird nach dem 1. Zeichen des Suchschlüssels verglichen. Bei Gleichheit wird der zugehörige Zeiger verfolgt. Im gefundenen Knoten wird nach dem 2. Zeichen verglichen usw.

- Aufwand bei
  erfolgreicher Suche:
  l_i /k (+ 1 bei Präfix)
- effiziente Bestimmung
  der Abwesenheit
  eines Schlüssels (z. B. CAD)



Grundoperationen auf Tries II

- **Einfügen**: Wenn Suchpfad schon vorhanden, wird NULL-Zeiger in *-Zeiger umgewandelt, sonst Einfügen von neuen Knoten (z. B. CAD)

- **Löschen**: Nach Aufsuchen des richtigen Knotens wird ein *-Zeiger auf NULL gesetzt. Besitzt daraufhin der Knoten nur NULL-Zeiger, wird er aus dem Baum entfernt (rekursive Überprüfung der Vorgängerknoten)

Eigenschaften von Tries

Beobachtungen:

- Höhe des Trie wird durch den längsten abgespeicherten Schlüssel bestimmt
- Gestalt des Baumes hängt von der Schlüsselmenge, also von der Verteilung der Schlüssel, nicht aber von der Reihenfolge ihrer Abspeicherung ab
- Knoten, die nur NULL-Zeiger besitzen, werden nicht angelegt

Dennoch schlechte Speicherplatzausnutzung

- dünn besetzte Knoten
- viele Einweg-Verzweigungen (v.a. in der Nähe der Blätter)

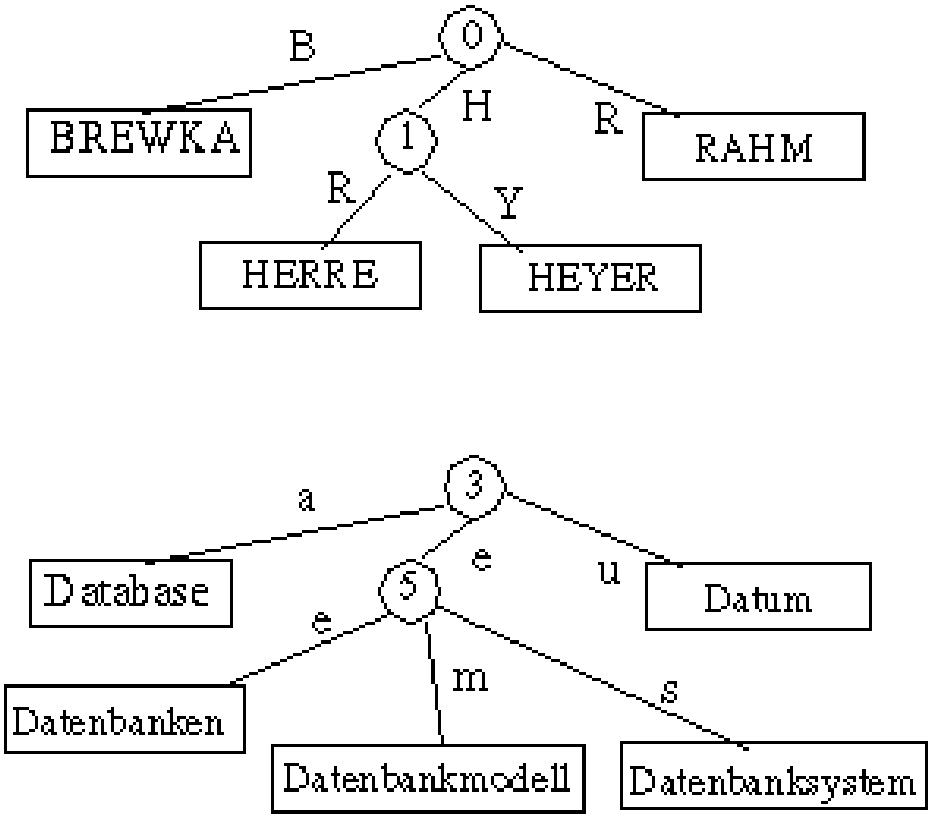
Möglichkeiten der Kompression

- Sobald ein Zeiger auf einen Unterbaum mit nur einem Schlüssel verweist, wird der (Rest-) Schlüssel in einem speziellen Knotenformat aufgenommen und Unterbaum eingespart -> vermeidet Einweg-Verzweigungen
- nur besetzte Verweise werden gespeichert (erfordert Angabe des zugehörigen Schlüsselteils)

PATRICIA-Tree (Practical Algorithm To Retrieve Information Coded In Alphanumeric)

Merkmale

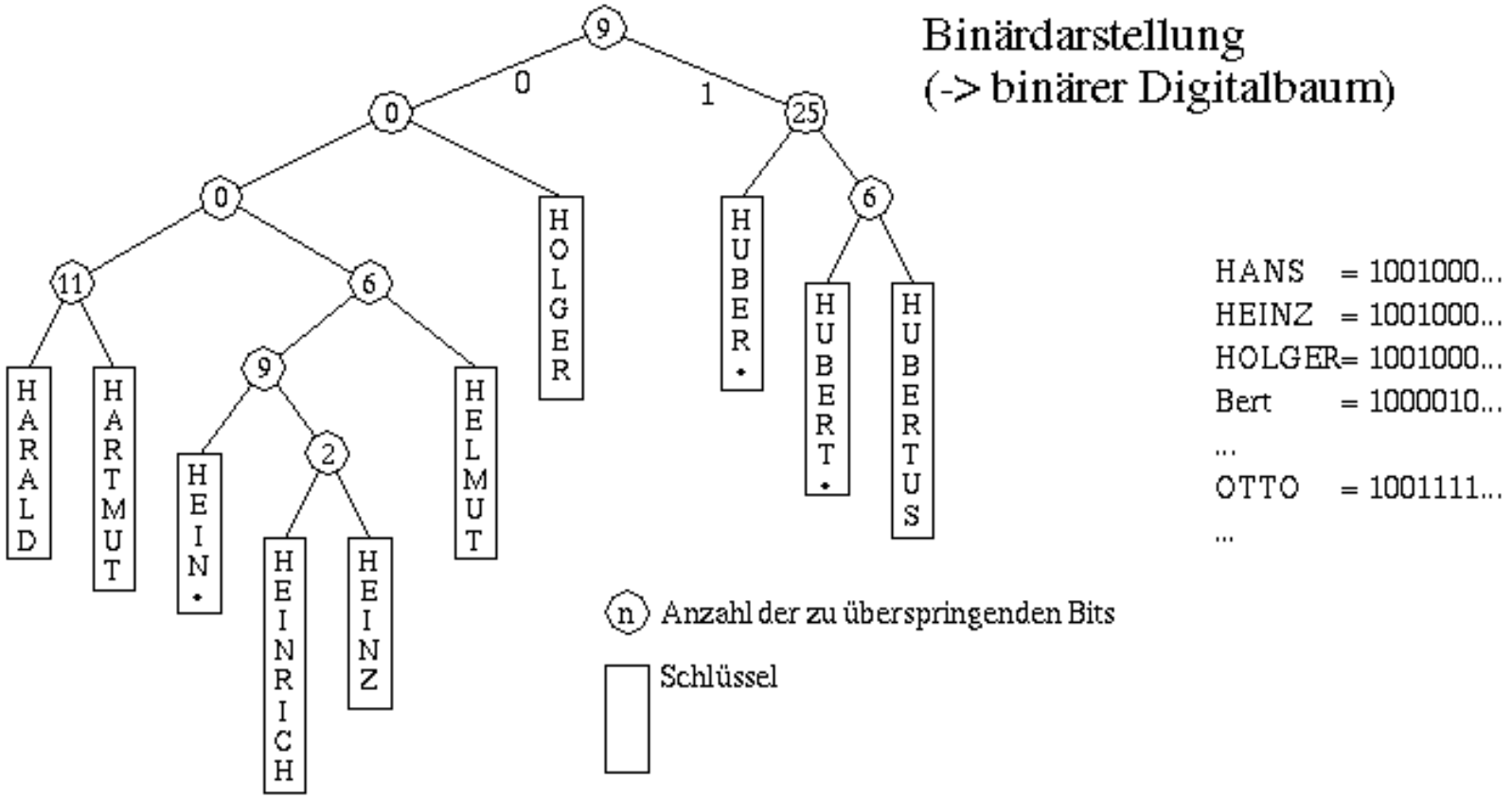
- Binärdarstellung für Schlüsselwerte -> binärer Digitalbaum
- Speicherung der Schlüssel in den Blättern
- innere Knoten speichern, wie viele Zeichen (Bits) beim Test zur Wegeauswahl zu überspringen sind
- Vermeidung von Einweg-verzweigungen, in dem bei nur noch einem verbleibenden Schlüssel direkt auf entsprechendes Blatt verwiesen wird



Eigenschaften von PATRICIA-Trees

- speichereffizient
- sehr gut geeignet für variabel lange Schlüssel und (sehr lange) Binärdarstellungen von Schlüsselwerten
- bei jedem Suchschlüssel muss die Testfolge von der Wurzel beginnend ganz ausgeführt werden, bevor über Erfolg oder Misserfolg der Suche entschieden werden kann

Beispiel: PATRICIA-Tree



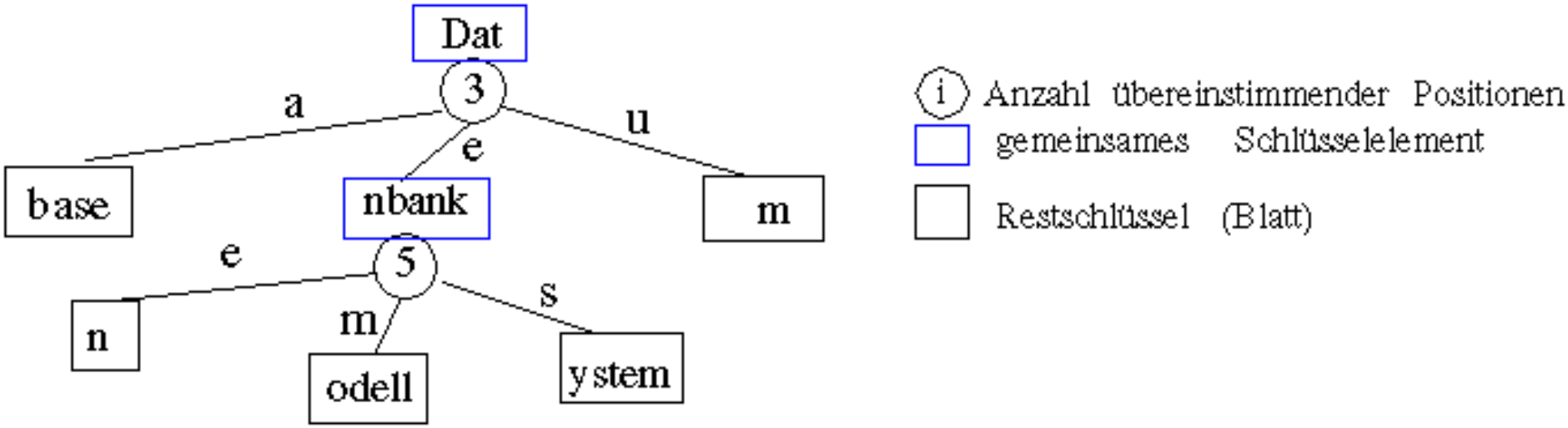
- Suche nach dem Schlüssel HEINZ = X'10010001000101100100110011101011010'
- Suche nach ABEL = X'10000011000010100010110011100' ?

Beobachtung: Erfolgreiche und erfolglose Suche endet in einem Blattknoten

Präfix- bzw. Radix-Baum

(Binärer) Digitalbaum als Variante des PATRICIA-Baumes

- Speicherung variabel langer Schlüsselteile in den inneren Knoten, sobald sie sich als Präfixe für die Schlüssel des zugehörigen Unterbaums abspalten lassen
- komplexere Knotenformate und aufwendigere Such- und Aktualisierungsoperationen
- erfolglose Suche lässt sich oft schon in einem inneren Knoten abbrechen



Zusammenfassung I

Konzept des Mehrwegbaumes:

- Aufbau sehr breiter Bäume von geringer Höhe
- Bezugsgröße: Seite als Transporteinheit zum Externspeicher
- Seiten werden immer größer, d. h., das Fan-out wächst weiter

B- und B*-Baum gewährleisten eine balancierte Struktur

- unabhängig von Schlüsselmenge
- unabhängig ihrer Einfügereihenfolge

Wichtigste Unterschiede des B*-Baums zum B-Baum:

- strikte Trennung zwischen Datenteil und Indexteil. Datenelemente stehen nur in den Blättern des B*-Baumes
- Schlüssel innerer Knoten haben nur Wegweiserfunktion. Sie können auch durch beliebige Trenner ersetzt oder durch Komprimierungsalgorithmen verkürzt werden
- kürzere Schlüssel oder Trenner in den inneren Knoten erhöhen Verzweigungsgrad des Baumes und verringern damit seine Höhe
- die redundant gespeicherten Schlüssel erhöhen den Speicherplatzbedarf nur wenig (<1%)
- Löschalgorithmus ist einfacher
- Verkettung der Blattseiten ergibt schnellere sequentielle Verarbeitung

Zusammenfassung II

Standard-Zugriffspfadstruktur in DBS: B*-Baum

verallgemeinerte Überlaufbehandlung verbessert Seitenbelegung

Schlüsselkomprimierung

- Verbesserung der Baumbreite
- Präfix-Suffix-Komprimierung sehr effektiv
- Schlüssellängen von 20-40 Bytes werden im Mittel auf 1.3-1.8 Bytes reduziert

Binäre B-Bäume: Alternative zu AVL-Bäumen als Hauptspeicher-Datenstruktur

Digitale Suchbäume: Verwendung von Schlüsselteilen

- Unterstützung von Suchvorgängen u.a. bei langen Schlüsseln variabler Länge
- wesentliche Realisierungen: PATRICIA-Baum / Radix-Baum

Algorithmen und Datenstrukturen 1

Wintersemester 2006 / 2007
Zusammenfassung

1. Effizienz: Zeit und Speicher

Die Abarbeitung von Programmen (Software) beansprucht 2 Ressourcen: Zeit und Hardware (wichtig: Speicher).

Wie steigt dieser Ressourcenverbrauch bei größeren Problemen (d.h. mehr Eingabedaten)?

Festlegung der Größenordnung der Komplexität in Abhängigkeit der Eingabegröße: Best Case, Worst Case, Average Case

Meist Abschätzung oberer Schranken (Worst Case): *Groß-Oh-Notation*

Zeitkomplexität T(n) eines Algorithmus ist von der Größenordnung n, wenn es Konstanten n_0 und $c > 0$ gibt, so dass für alle Werte von $n > n_0$ gilt:

$$T(n) \leq c \cdot n$$

2. Suchen

Sequentielle Suche

- Default-Ansatz zur Suche
- lineare Kosten O(n)

Binärsuche

- setzt Sortierung voraus
- Teile-und-Herrsche-Strategie
- wesentlich schneller als sequentielle Suche
- Komplexität: O(log n)

Weitere Suchverfahren auf sortierten Arrays für Sonderfälle

- Sprungsuche
- exponentielle Suche
- Interpolationssuche

3. Listen

Verkettete Listen

- dynamische Datenstrukturen mit geringem Speicheraufwand und geringem Änderungsaufwand
- Implementierungen: einfach vs. doppelt verkettete Listen
- hohe Flexibilität
- hohe Suchkosten

Skip-Listen

- logarithmische Suchkosten
- randomisierte statt perfekter Skip-Listen zur Begrenzung des Änderungsaufwandes

ADT-Beispiele: Stack, Queue, Priority Queue

- spezielle Listen mit eingeschränkten Operationen (LIFO bzw. FIFO)
- formale ADT-Spezifikation zur Festlegung der implementierungsunabhängigen Eigenschaften
- effiziente Implementierbarkeit der Operationen: O(1)
- zahlreiche Anwendungsmöglichkeiten

4. Sortieren

Kosten allgemeiner Sortierverfahren wenigstens O(n log n)

Elementare Sortierverfahren: Kosten $O(n^2)$

- einfache Implementierung; ausreichend bei kleinem n
- gute Lösung: Insertion Sort

O(n log n)-Sortierverfahren: Heap-Sort, Quick-Sort, Merge-Sort

- Heap-Sort und Merge-Sort sind Worst-Case-optimal (O (n log n))
- in Messungen erzielte Quick-Sort in den meisten Fällen die besten Ergebnisse

Begrenzung der Kosten für Umkopieren durch indirekte Sortierverfahren

Vorteilhafte Eigenschaften

- Ausnutzen einer weitgehenden Vorsortierung
- Stabilität

Externes Sortieren

- fortgesetztes Zerlegen und Mischen
- Mehrwege-Merge-Sort reduziert Externspeicherzugriffe

5. Bäume

Definitionen

- Baum, orientierter Baum (Wurzel-Baum), geordneter Baum, Binärbaum
- vollständiger, fast vollständiger, strikter, ausgeglichener, ähnlicher, äquivalenter Binärbaum
- Höhe, Grad, Stufe / Pfadlänge, Gewicht

Speicherung von Binärbäumen

- verkettete Speicherung
- Feldbaum-Realisierung
- sequentielle Speicherung

Baum-Traversierung

- Preorder (WLR): Vorordnung
- Inorder (LWR): Zwischenordnung
- Postorder (LRW): Nachordnung

Gefädelte Binärbäume: Unterstützung der (iterativen) Baum-Traversierung durch Links/Rechts-Zeiger auf Vorgänger/Nachfolger in Traversierungsreihenfolge

6. Binäre Suchbäume

Natürliche binäre Suchbäume

- Grundoperationen: Einfügen, sequentielle Suche, direkte Suche, Löschen
- Bestimmung der mittleren Zugriffskosten

Balancierte Binärbäume

AVL-Baum

- Einfügen mit Rotationstypen
- Löschen mit Rotationstypen

Gewichtsbalancierte Binärbäume

Positionssuche mit balancierten Bäumen (Lösung des Auswahlproblems)

7. Mehrwegbäume I

Konzept des Mehrwegbaumes:

- Aufbau sehr breiter Bäume von geringer Höhe
- Bezugsgröße: Seite als Transporteinheit zum Externspeicher
- Seiten werden immer größer, d. h., das Fan-out wächst weiter

B- und B*-Baum gewährleisten eine balancierte Struktur

- unabhängig von Schlüsselmenge
- unabhängig ihrer Einfügereihenfolge

Wichtigste Unterschiede des B*-Baums zum B-Baum:

- strikte Trennung zwischen Datenteil und Indexteil. Datenelemente stehen nur in den Blättern des B*-Baumes
- Schlüssel innerer Knoten haben nur Wegweiserfunktion. Sie können auch durch beliebige Trenner ersetzt oder durch Komprimierungsalgorithmen verkürzt werden
- kürzere Schlüssel oder Trenner in den inneren Knoten erhöhen Verzweigungsgrad des Baumes und verringern damit seine Höhe
- die redundant gespeicherten Schlüssel erhöhen den Speicherplatzbedarf nur wenig (<1%)
- Löschalgorithmus ist einfacher
- Verkettung der Blattseiten ergibt schnellere sequentielle Verarbeitung

Mehrwegbäume II

Standard-Zugriffspfadstruktur in DBS: B*-Baum

verallgemeinerte Überlaufbehandlung verbessert Seitenbelegung

Schlüsselkomprimierung

- Verbesserung der Baumbreite
- Präfix-Suffix-Komprimierung sehr effektiv
- Schlüssellängen von 20-40 Bytes werden im Mittel auf 1.3-1.8 Bytes reduziert

Binäre B-Bäume: Alternative zu AVL-Bäumen als Hauptspeicher-Datenstruktur

Digitale Suchbäume: Verwendung von Schlüsselteilen

- Unterstützung von Suchvorgängen u.a. bei langen Schlüsseln variabler Länge
- wesentliche Realisierungen: PATRICIA-Baum / Radix-Baum