Baumverfahren

- Stufenanzahl dynamisch verändern
- wichtigste Baumverfahren: *B-Bäume* und ihre Varianten
- B-Baum-Varianten sind noch „allgegenwärtiger“ in heutigen Datenbanksystemen als SQL
- SQL nur in der relationalen und objektrelationalen Datenbanktechnologie verbreitet; B-Bäume überall als Grundtechnik eingesetzt

B-Bäume

- Ausgangspunkt: ausgeglichener, balancierter Suchbaum
- *Ausgeglichen* oder *balanciert*: alle Pfade von der Wurzel zu den Blättern des Baumes gleich lang
- Hauptspeicher-Implementierungsstruktur: binäre Suchbäume, beispielsweise AVL-Bäume von Adelson-Velskii und Landis
- Datenbankbereich: Knoten der Suchbäume zugeschnitten auf Seitenstruktur des Datenbanksystems
- mehrere Zugriffsattributwerte auf einer Seite
- *Mehrweg-Bäume*

Prinzip des B-Baums

- *B-Baum* von Bayer (B für balanciert, breit, buschig, Bayer, **NICHT:** binär)
- dynamischer, balancierter Indexbaum, bei dem jeder Indexeintrag auf eine Seite der Hauptdatei zeigt

Mehrwegebaum völlig ausgeglichen, wenn

1. alle Wege von Wurzel bis zu Blättern gleich lang
2. jeder Knoten gleich viele Indexeinträge

vollständiges Ausgleichen zu teuer, deshalb B-Baum-Kriterium:

Jede Seite außer der Wurzelseite enthält zwischen m und $2m$ Daten

Eigenschaften des B-Baums (1)

n Datensätze in der Hauptdatei $\Rightarrow$ in $log_m(n)$ Seitenzugriffen von der Wurzel zum Blatt

- Durch Balancierungskriterium wird Eigenschaft nahe an der vollständigen Ausgeglichenheit erreicht (1. Kriterium vollständig erfüllt, 2. Kriterium näherungsweise)
- Kriterium garantiert 50% Speicherplatzausnutzung
- einfache, schnelle Algorithmen zum Suchen, Einfügen und Löschen von Datensätzen (Komplexität von $O(\log_m(n))$)

Eigenschaften des B-Baums (2)

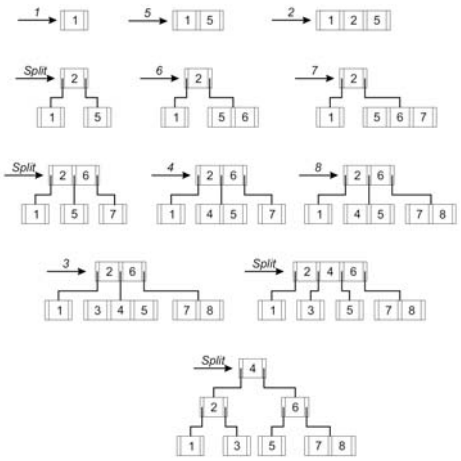
- B-Baum als Primär- und Sekundärindex geeignet
- Datensätze direkt in die Indexseiten $\Rightarrow$ Dateiorganisationsform
- Verweist man aus Indexseiten auf Datensätze in den Hauptseiten $\Rightarrow$ Sekundärindex

Definition B-Baum

Ordnung eines B-Baumes ist minimale Anzahl der Einträge auf den Indexseiten außer der Wurzelseite
Bsp.: B-Baum der Ordnung 8 faßt auf jeder inneren Indexseite zwischen 8 und 16 Einträgen
Def.: Ein Indexbaum ist ein B-Baum der Ordnung m, wenn er die folgenden Eigenschaften erfüllt:

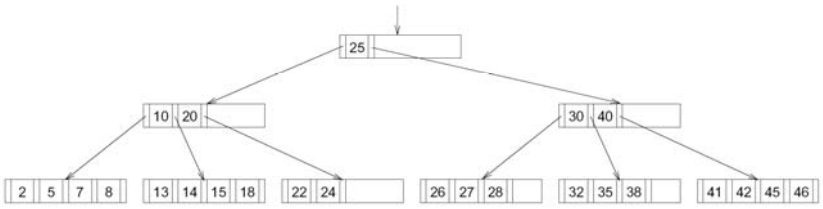
1. Jede Seite enthält höchstens $2m$ Elemente.
2. Jede Seite, außer der Wurzelseite, enthält mindestens m Elemente.
3. Jede Seite ist entweder eine Blattseite ohne Nachfolger oder hat $i+1$ Nachfolger, falls i die Anzahl ihrer Elemente ist.
4. Alle Blattseiten liegen auf der gleichen Stufe.

Beispiel: Einfügen in B-Baum



Suchen in B-Bäumen

- **lookup** wie in statischen Indexverfahren
- Startend auf Wurzelseite Eintrag im B-Baum ermitteln, der den gesuchten Zugriffsattributwert w überdeckt $\Rightarrow$ Zeiger verfolgen, Seite nächster Stufe laden
- Suchen: 38, 20, 6



Einfügen in B-Bäume

Einfügen eines Wertes w

- mit **lookup** entsprechende Blattseite suchen
- passende Seite $n < 2m$ Elemente, w einsortieren
- passende Seite $n = 2m$ Elemente, neue Seite erzeugen,
  - ersten m Werte auf Originalseite
  - letzten m Werte auf neue Seite
  - mittleres Element auf entsprechende Indexseite nach oben
- eventuell dieser Prozeß rekursiv bis zur Wurzel

Löschen in B-Bäumen (1)

bei weniger als m Elementen auf Seite: Unterlauf
Löschen eines Wertes w: Bsp.: 24; 28, 38, 35

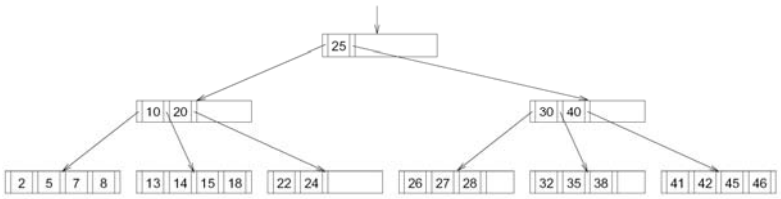
- mit **lookup** entsprechende Seite suchen
- w auf Blattseite gespeichert $\Rightarrow$ Wert löschen, eventuell Unterlauf behandeln
- w nicht auf Blattseite gespeichert $\Rightarrow$ Wert löschen, durch lexikographisch nächstkleineres Element von einer Blattseite ersetzen, eventuell auf Blattseite Unterlauf behandeln

Löschen in B-Bäumen (2)

Unterlaufbehandlung

- Ausgleichen mit der benachbarten Seite (benachbarte Seite n Elemente mit $n > m$)
- oder Zusammenlegen zweier Seiten zu einer (Nachbarseite $n = m$ Elemente), das „mittlere“ Element von Indexseite darüber dazu, auf Indexseite eventuell Unterlauf behandeln

Beispiel: Löschen

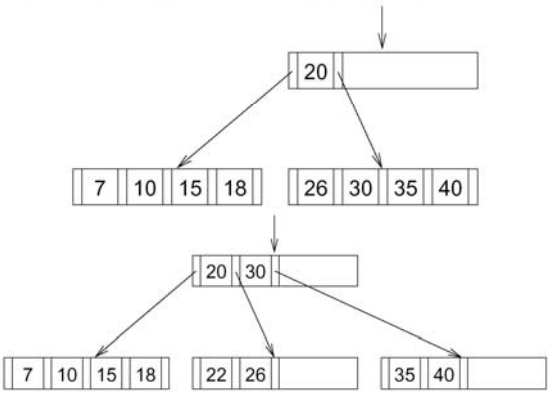


m = 2

Löschen: a) 24
b) 28, 38, 35

Beispiel: Einfügen und Löschen

Einfügen des Elementes 22; Löschen von 22



Komplexität der Operationen

- Aufwand beim Einfügen, Suchen und Löschen im B-Baum immer $O(log_m(n))$ Operationen
- entspricht genau der „Höhe“ des Baumes
- Konkret: Seiten der Größe 4 KB, Zugriffsattributwert 32 Bytes, 8-Byte-Zeiger: zwischen 50 und 100 Indexeinträge pro Seite; Ordnung dieses B-Baumes 50
- 1.000.000 Datensätze: $\log_{50}(1.000.000) = 4$ Seitenzugriffe im schlechtesten Fall
- Wurzelseite jedes B-Baumes normalerweise im Puffer: drei Seitenzugriffe

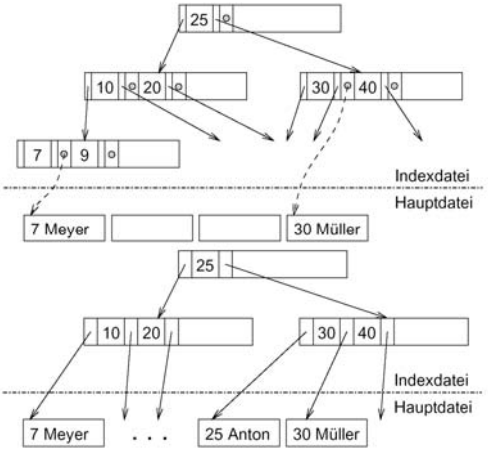
B-Baum-Varianten

- B^+-Bäume: Hauptdatei als letzte (Blatt-)Stufe des Baumes integrieren
- B^*-Bäume: Aufteilen von Seiten vermeiden durch „Shuffle“
- Präfix-B-Bäume: Zeichenketten als Zugriffsattributwerte, nur Präfix indexieren

B^+-Baum

- in der Praxis am häufigsten eingesetzte Variante des B-Baumes: effizientere Änderungsoperationen, Verringerung der Baumhöhe
- integriert Datensätze der Hauptdatei auf den Blattseiten des Baumes
- in inneren Knoten nur noch Zugriffsattributwert und Zeiger auf nachfolgenden Seite der nächsten Stufe

B-Baum vs. B$^+$-Baum



Ordnung und Operationen

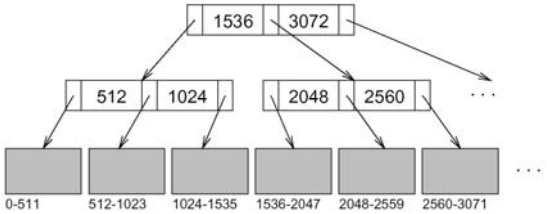
- *Ordnung* für B$^+$-Baum: (x, y), x Mindestbelegung der Indexseiten, y Mindestbelegung der Datensatz-Seiten
- **delete** gegenüber B-Baum effizienter („Ausleihen“ eines Elementes von der Blattseite entfällt)
- Zugriffsattributwerte in inneren Knoten können sogar stehenbleiben
- häufig als Primärindex eingesetzt
- B$^+$-Baum ist dynamische, mehrstufige, indexsequentiellen Datei

B*- und B#-Baum

- Problem beim B-Baum: häufiges Aufspalten von Seiten und geringe Speicherplatzausnutzung von nahe 50%
- B*-Baum, B#-Baum:
  - statt Aufteilen von Seiten bei Überlauf zunächst Neuverteilen der Datensätze auf eventuell nicht voll ausgelastete Nachbarseiten
  - falls nicht möglich: zwei Seiten in drei aufteilen (ermöglicht durchschnittliche Speicherplatzausnutzung von 66% statt 50%)

B+-Baum für BLOBs

- Statt Zugriffsattributwerte in B+-Baum: Positionen oder Offsets im BLOB indexieren
- BLOB-B+-Baum: Positions-B+-Baum



- Auch für andere große Speicherobjekte (wie in objektorientierten Datenbanken üblich) geeignet