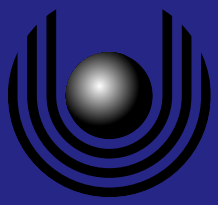
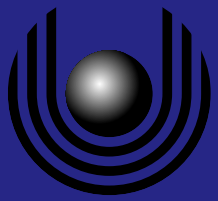


Indexe in Dateien

Thomas Behr

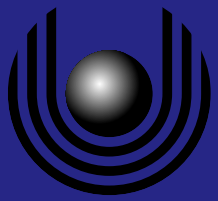
Fakultät für Mathematik und Informatik
Datenbanksysteme für neue Anwendungen
FernUniversität in Hagen

19.November 2015



Motivation

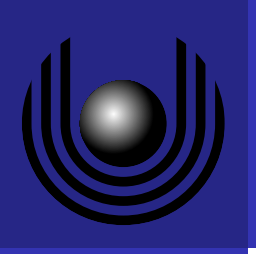
- Indexe ermöglichen schnellen Zugriff auf bestimmte Tupel einer Relation
- Indexe sind nach einem Attribut der Relation organisiert
- verschiedene Indexstrukturen ermöglichen unterschiedliche Arten von Suchen
- hier Betrachtung von B^+-Baum und R-Baum
- beide ermöglichen Punkt und Bereichssuche
- B^+-Baum: Indizierung von Standarddaten
- R-Baum: Indizierung geometrischer Daten
- beide Strukturen schon lange Bestandteil von Secondo



Gibt es schon . . . Nichts mehr zu tun?

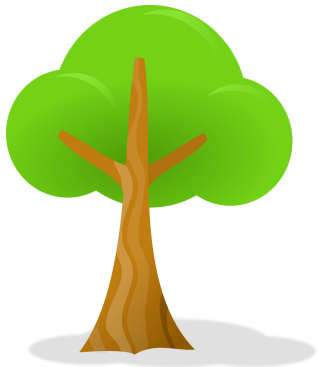
Weit gefehlt.

- existierende Indexe sind Datenbankobjekte
- Zugriff erfolgt inklusive Transaktionseigenschaften
- Übertragung auf andere Computer nicht möglich
- schlecht für parallele Bearbeitung
- Indexe in Dateien statt in der Datenbank

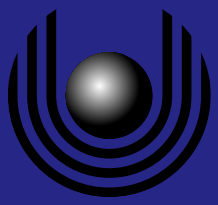


B^+-Bäume

Aufgabe Phase2
Thomas Behr
Motivation
B^+-Bäume
R-Bäume
Cache
Die Aufgabe

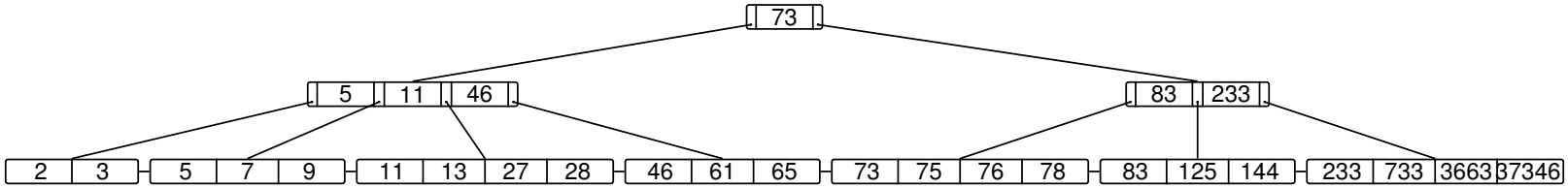


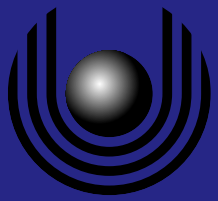
- Vielwegsuchbaum
- Einträge sortiert
- alle Knoten außer Wurzel mind. halbvoll
- balanciert
- Datensatzverweise nur in den Blättern
- innere Knoten nur zur Navigation
- Blätter doppelt verkettet
- Duplikate möglich
- Knotengröße = Seitengröße
- Datensatzidentifikator: TupleId
- Indexattribut
  - aus Standardalgebra
    oder
  - in Kind `INDEXABLE`



Beispiel

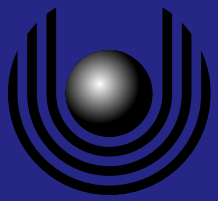
Aufgabe Phase2
Thomas Behr
Motivation
B$^+$-Bäume
R-Bäume
Cache
Die Aufgabe





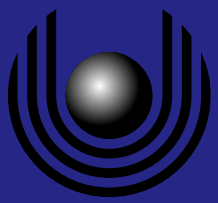
Dateiaufbau

- Header
  - Marker
  - Seitengröße
  - Seitennummer des Wurzelknotens
  - Seitennummer der Freiseitenliste
  - indizierter Typ
  - …
- Knoten
  - Einträge
  - Seitennummern der Söhne
    oder
  - TupleId
- restliche Seite leer



Freiseitenliste

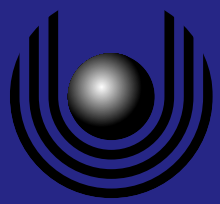
- gelöschte Knoten werden nicht aus Datei entfernt
- werden in Freiseitenliste eingetragen
- Header hat Verweis auf erste Seitennummer
- Seite enthält Seitennummer der nächsten freien Seite oder 0
- einfügen/entnehmen einer Seite stets am Listenanfang
- neuer Knoten immer aus Freiseitenliste, nur wenn leer neue Seite am Ende der Datei



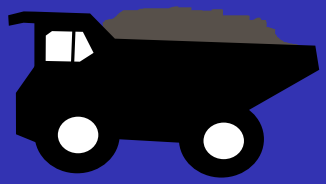
Einfügen

- folge Pfad zu Blatt, in das Eintrag passt
- füge Element und Verweis ein
- wenn Knoten nicht übergelaufen ist: **fertig**

  sonst
- versuche Ausgleich mit Nachbarn
- falls Ausgleich nicht möglich: teile Knoten
- neuer Eintrag im Vater
  möglicherweise Ausgleich/Teilung erforderlich
  kann sich bis zur Wurzel fortsetzen
  Teilung der Wurzel: neue Wurzel mit nur einem Eintrag

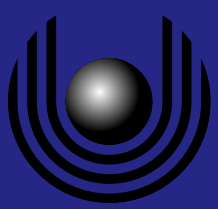


Bulkload



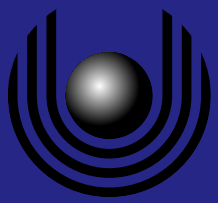
Aufgabe Phase2
Thomas Behr
Motivation
B$^+$-Bäume
R-Bäume
Cache
Die Aufgabe

- hintereinander einfügen in Baum teuer (Umstrukturierungen)
- Einfügen einer sortierten Folge: schnellerer Aufbau
  - fülle Blatt bis Blatt voll ist
  - wenn aktuelles Blatt voll: erzeuge Vater und starte mit neuem Knoten
  - fahre fort
  - wenn Vater überläuft, erzeuge auf dessen Ebene neuen Knoten
  - Kette rechtester Knoten
  - am Ende: Ausgleich dieser Kette mit Nachbarknoten



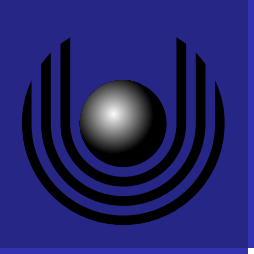
Löschen 

- suche zu löschenden Eintrag und lösche diesem aus Blatt
- Knoten hat noch genügend Elemente: **fertig**
- versuche Ausgleich mit Nachbarn
- Ausgleich nicht möglich: verschmelze mit Nachbarn
  Eintrag aus Vater wird gelöscht
  Unterlauf kann sich bis zur Wurzel fortsetzen
  Wenn Wurzel leer, verschwindet diese

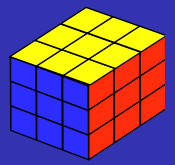


Suche

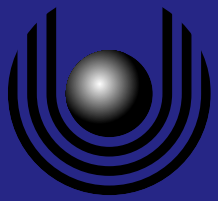
- Intervallsuche
  - gehe zum Blatt, das linkes Intervallende enthält
  - verfolge die Kette der Blätter bis zum rechten Intervallende
- Varianten
  - exact match
  - left range
  - right range



Umstrukturierung

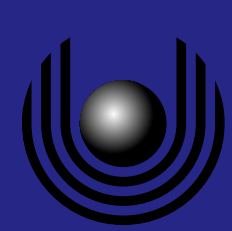


- Seiten in Datei nicht optimal angeordnet
- insbesondere Sprünge beim Durchlaufen der Blattliste
- unbenutzte Seiten in Datei
- Operation zum Bereinigen der Datei (neue Datei wird erstellt)

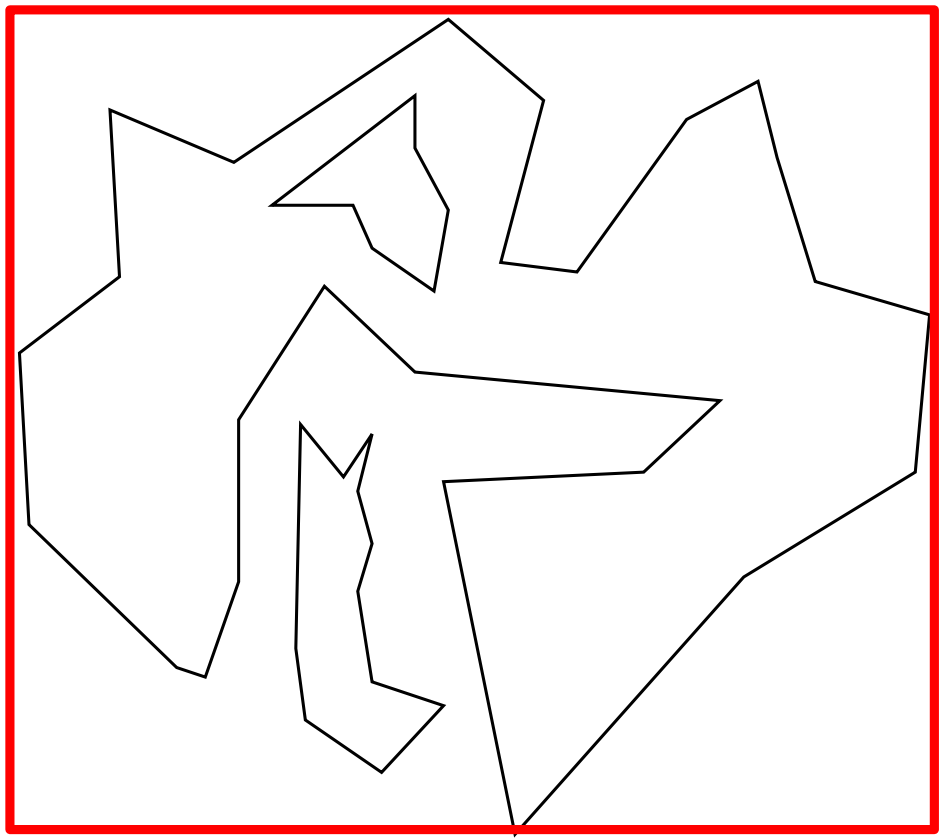


R-Bäume

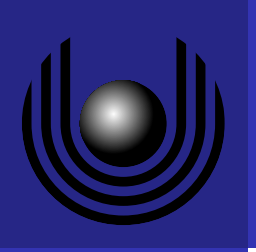
- ähnlich wie B^+-Baum aufgebaut
- es werden Rechtecke indiziert
- komplexe Geometrien durch Bounding Box approximiert
- Verweise zu Tupeln nur in den Blättern
- Vaterknoten enthält Bounding Box der Rechtecke im Sohn
- Rechtecke dürfen überlappen
- Knotengröße = Seitengröße
- Minimalfüllung kann kleiner als die Hälfte sein (Benutzervorgabe)
- keine Ordnung der Elemente in den Knoten
- theoretisch beliebige Dimension



Bounding Box

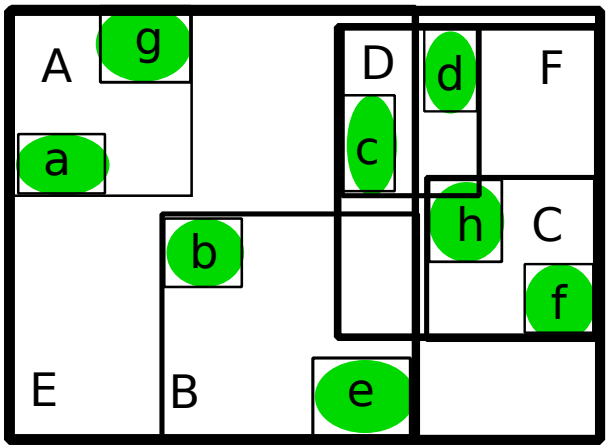


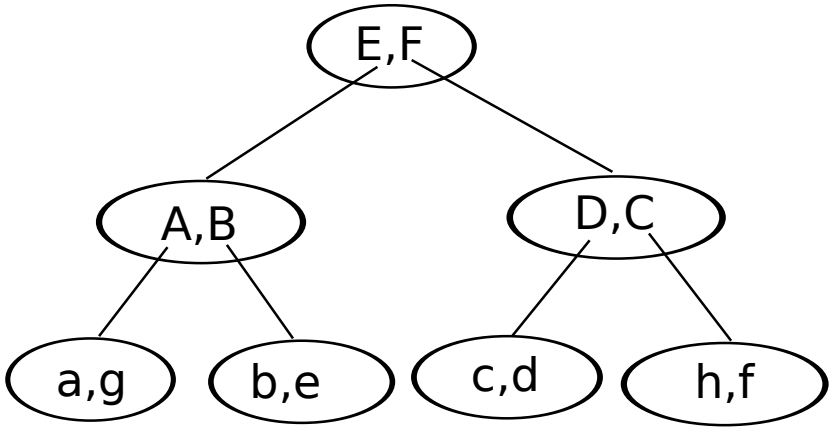
B-Box in 2 Dimensionen

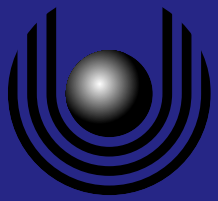


Beispiel

Aufgabe Phase2
Thomas Behr
Motivation
B$^+$-Bäume
R-Bäume
Cache
Die Aufgabe

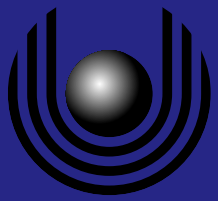






Einfügen

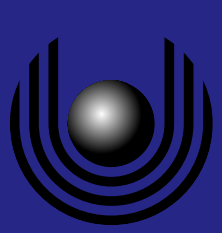
- solange kein Blatt erreicht: wähle besten Sohn
- füge neuen Eintrag in Blatt ein
- bei Überlauf wird Blatt sofort geteilt
- Teilung kann sich bis zur Wurzel hinziehen



Besten Sohn finden

1. Sohn mit geringstem Flächenzuwachs
2. Sohn mit kleinster Fläche
3. Sohn mit wenigsten Einträgen
4. wähle willkürlich

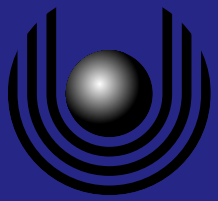
Schritte werden verfolgt, bis eine eindeutige Lösung gefunden ist.



Teilen eines Knotens

Mehrere Möglichkeiten: hier Quadratic Split

- erzeuge zwei neue, leere Knoten
- wähle die beiden Einträge, für deren BBox-Vereinigung maximal ist, und verteile sie auf beide Knoten
- für alle restlichen Elemente
  - falls Minimalfüllung es erfordert: füge alle Elemente in einen Knoten ein
  - sonst
    - berechne für alle Elemente den Flächenzuwachs in beiden Knoten
    - wähle das Element mit der größten absoluten Differenz
    - füge in besten Knoten ein



Suchen

Varianten:

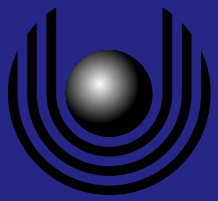
- Punktsuche, Bereichssuche

Algorithmus:

- Starte an der Wurzel
- vom aktuellen Knoten aus, verfolge alle Pfade, bei denen die Boxen der Blätter das Anfragerechteck schneiden
- Wenn Blatt: durchlaufe Blatt und gib Ergebnisse aus

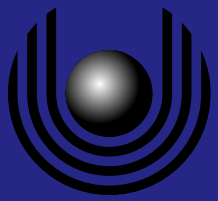
Implementierung

- Rekursion scheinbar angemessen, jedoch durch Stromverarbeitung nicht sinnvoll
- verwende Iterator-Klasse, die einen Stack verwaltet
- Tiefensuche



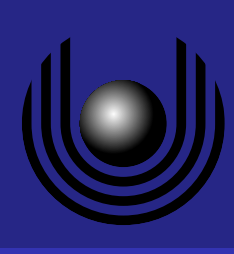
Löschen

- suche Eintrag und lösche ihn
- bei Unterlauf, entferne den ganzen Knoten (merke Ebene)
- entferne Eintrag im Vaterknoten
- kann sich bis zur Wurzel hinziehen
- füge Elemente auf ihren alten Ebenen wieder ein ähnlich wie beim ’normalen’ Einfügen
- falls Wurzel nur noch einen Sohn: entferne Wurzel



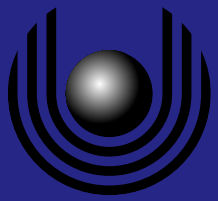
bulkload

- Tupelstrom so sortiert, dass beieinanderliegende Rechtecke hintereinander auftreten
- Algorithmus wie beim B$^+$-Baum
- Zusätzlich: Abstand gegeben, bei dessen Erreichen ein neuer Knoten begonnen wird auch wenn der aktuelle nicht voll ist



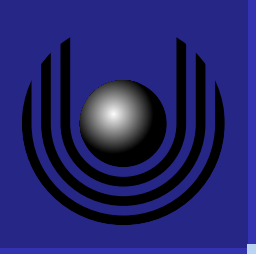
Cache

- Dateizugriffe langsam
- begrenzte Anzahl an Seiten im Hauptspeicher vorhalten
- LRU Strategie
- Herausschreiben aller geänderter Daten möglich (flush)
- für beide Baumarten zu verwenden



Aufgabe

- Implementierung des Dateicache
- Implementierung der beiden vorgestellten Strukturen mit dazugehörigen Operationen
- Erstellung von Secondo-Operatoren



Zeit für Fragen