Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Generalized Search Trees for Database Systems

Advanced Topics in Databases

Johannes Dyck

Gliederung

1. Einführung
   1. Autoren
   2. Motivation
2. Der GiST
   1. Kernkonzepte
   2. Methoden und Algorithmen (am Beispiel)

Autoren

- Joseph M. Hellerstein
  - Havard
  - University of California, Berkeley
  - University of Wisconsin, Madison
- Jeffrey F. Naughton
  - University of Wisconsin, Madison
  - Stanford
- Avi Pfeffer
  - Havard

[Quellen siehe letzte Folie]

Motivation

- Kompletter Scan einer Datenbank dauert lange
- Indizes bieten eine Lösung
- Häufige Indexstruktur: B^+-Baum
  - Bereichsanfragen
  - Gleichheitsanfragen
- Genügen B^+-Bäume?

Motivation

- Räumliche oder geographische Daten
  - R-Bäume
- Mengenwertige Attribute
  - RD-Bäume
- Lösungsansätze
  - Hochspezialisierte Suchbäume für spezifische Datentypen
  - Spezialisierte, bzgl. der Datentypen erweiterbare Suchbäume
  - Generische, anpassbare Suchbäume

Der GiST

- Abstrakter Datentyp
- Konfigurierbar für …
  - … verschiedene Datentypen
  - … verschiedene Anfragen
- Nutzer stellt sechs Schlüsselmethoden bereit
  - Bestimmt Datentypen und unterstützte Anfragen
- Sämtliche Algorithmen bauen auf diesen Methoden auf

Kernkonzept

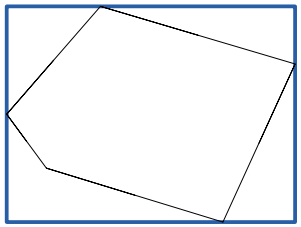
- Grundstruktur ähnlich B^+-Baum
- Anfragen werden als Prädikate formuliert
- Schlüssel stellen Prädikate dar
  - Sämtliche Tupel unter einem Schlüssel erfüllen Prädikat
  - Überlappung mehrerer Schlüssel möglich
- Tupel können durch ihre Werte Prädikate instanziieren

Prädikate

- In B+-Bäumen
  - Contains([2,12), *v*) (Schlüssel)
  - Equal(4, *v*)
- In R-Bäumen
  - Contains(((2,7),(6,3)), *b*) (Schlüssel)
  - Overlap(((2,7),(6,3)), *b*)
  - Equal(((2,7),(6,3)), *b*)
- In RD-Bäumen
  - Contains({2, 3, 9}, *s*) (Schlüssel)
  - Overlap({2, 3, 9}, *s*)
  - Equal({2, 3, 9}, *s*)

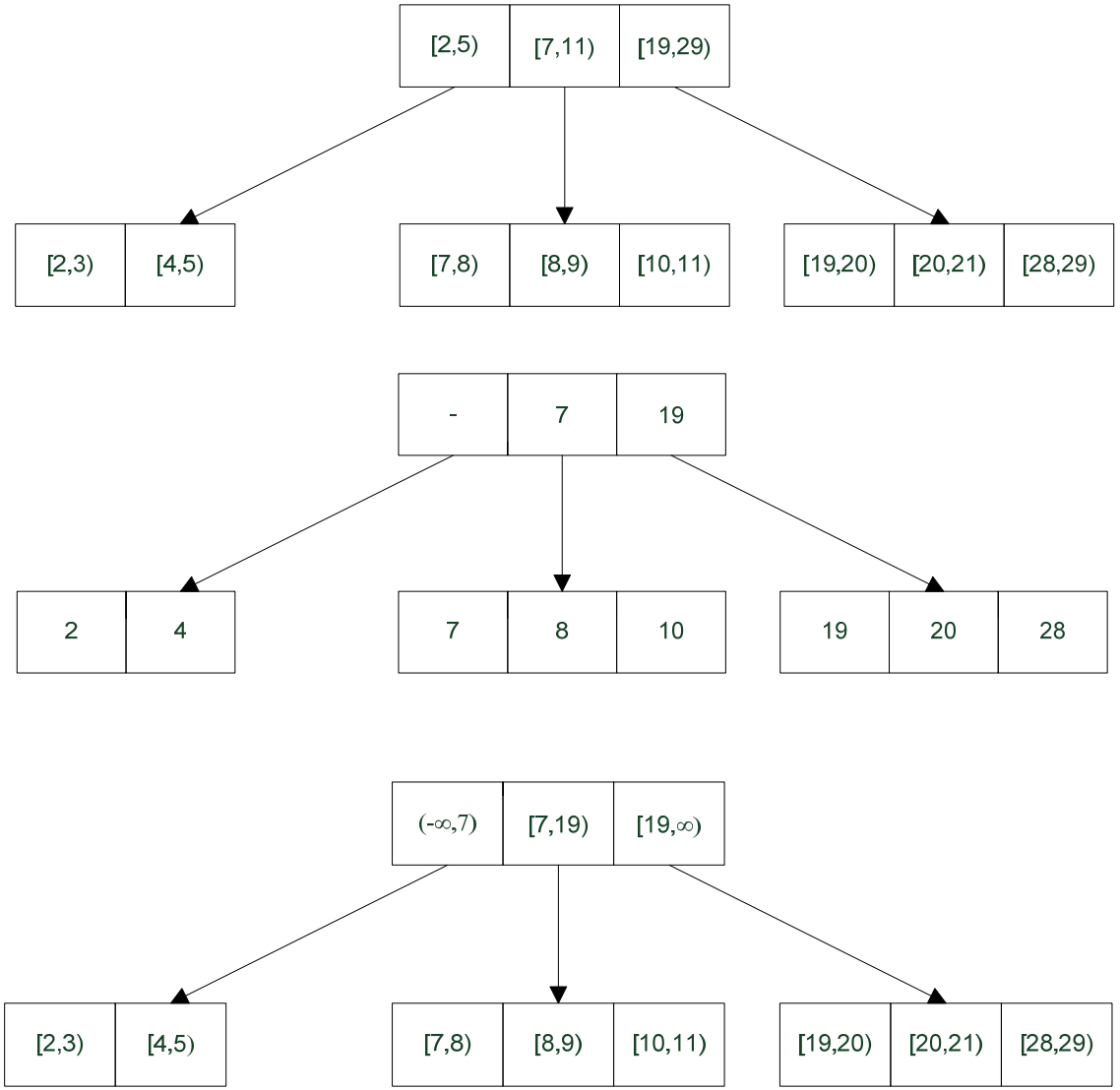
Komprimierung und Dekomprimierung

- Methode: Compress
  - Komprimierte Darstellung von Schlüsseln
  - Einfachster Fall: Identität
  - Schlüssel werden komprimiert gespeichert



- Methode: Decompress
  - Dekomprimierung zur Auswertung der Prädikate
  - Einfachster Fall: Identität
  - Informationsverlust möglich
  - Reduzierte Performanz

Komprimierung und Dekomprimierung



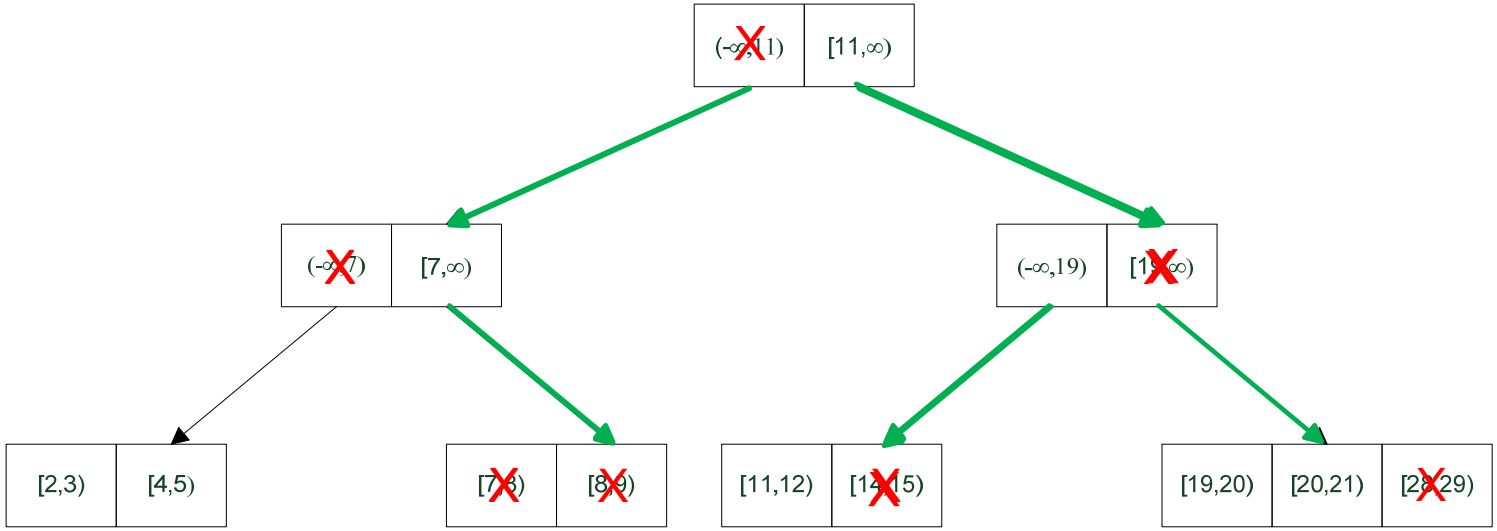
Algorithmus: Search

- Suche nach allen Tupeln, die gegebenes Prädikat erfüllen

- Beginne bei Wurzel
  - Prüfe, ob Schlüssel Tupel enthalten kann, die Prädikat erfüllen
  - Konsistenz von Schlüssel und Anfrageprädikat
  - Falls ja, folge Zeiger
  - Prüfe am Ende Tupel selbst

- Methode: Consistent
  - Gegeben: Schlüssel und Anfrageprädikat
  - Bei Widerspruch falsch, ansonsten wahr
  - Kann wahr ergeben, obwohl kein Tupel vorhanden

Algorithmus: Search



- Contains([10,12), *v*)
  - Contains([-∞,11)) und Contains([10,12)) sind konsistent
- Equal(11, *v*)
- Contains([8,20), *v*)

Search in linear geordneten Mengen

- Erweiterung für geordneten Wertebereich
  - Compare
  - PickSplit teilt Knoten gemäß Compare
  - Keine Überlappungen
- FindMin
  - Konsistenten Eintrag, der am weitesten links steht, verfolgen
  - Im Blatt ersten solchen Schlüssel von links zurückgeben
- Next
  - Nächsten konsistenten Schlüssel rechts zurückgeben

Konsistenz

- R-Bäume
  - Überlappen sich zwei Rechtecke?
  - Ist ein Rechteck in einem anderen enthalten?
- RD-Bäume
  - Sind Contains({2, 3, 9}) und Equal({3}) konsistent?
- Noch allgemeiner
  - Sind $(p \wedge q \wedge r) \vee z$ und $\neg p \vee (r \wedge z)$ konsistent?

Algorithmus: Insert

- Geeigneten Teilbaum finden
  - Baum in Richtung Blatt durchlaufen
  - Jeweils den Teilbaum wählen, für den Penalty minimal wird
- Methode: Penalty
  - Für einen gegebenen Teilbaum, bewerte Einfügen des Schlüssels
  - Beschreibt typischerweise den Zuwachs im Prädikat
- Füge Knoten ein
- Knoten teilen, falls nötig

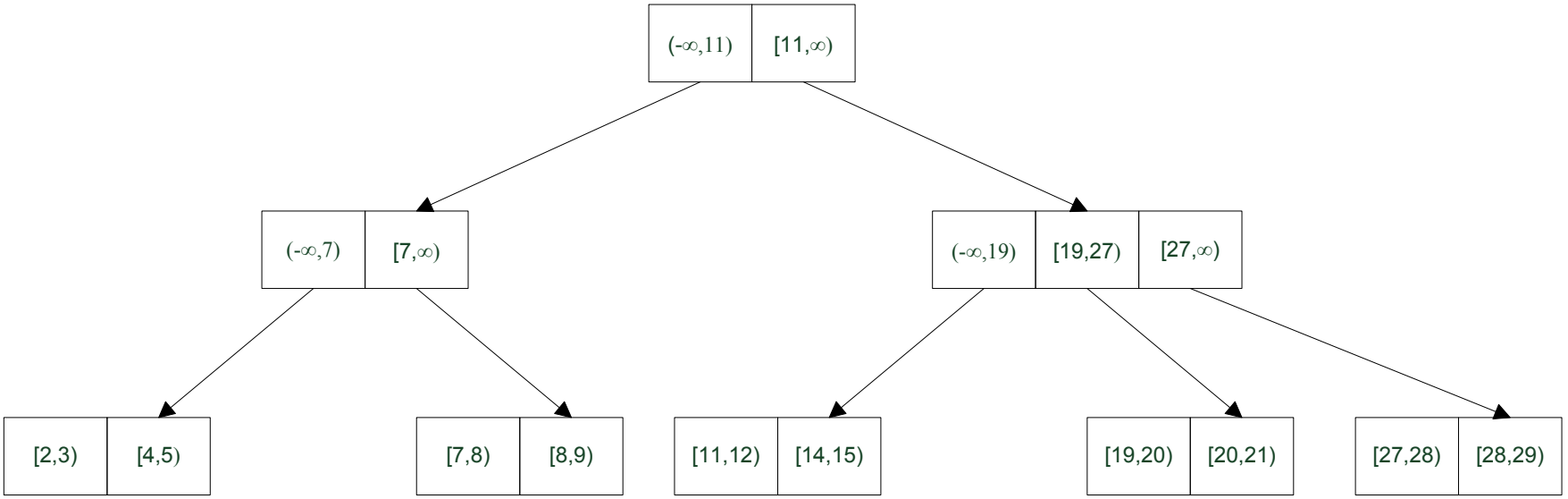
Algorithmus: Split

- Teile Knoten, in den eingefügt wurde, gemäß PickSplit
- Methode: PickSplit
  - Teilt eine Menge von Einträgen in zwei Teilmengen
  - Entscheidet über den *minimum fill factor*
- Füge neuen Knoten ein, bilde Prädikat durch Union
- Methode: Union
  - Gegeben: eine Menge von Schlüsseln
  - Bilde Prädikat, das alle diese Schlüssel erfüllen

Algorithmus: AdjustKeys

- Durch Einfügen müssen Prädikate angepasst werden
- Beginne bei Knoten, in den eingefügt wurde
- Erstelle korrektes Prädikat durch Union
- Breche ab, …
  - … wenn bei Wurzel angekommen
  - … Darstellung vorher bereits korrekt

Beispiel für Insert



- Einfügen von 27, also [27,28)
- Penalty berechnet die Länge, um die ein Intervall erweitert würde
- PickSplit teilt in zwei Hälften
- Einfügen eines neuen Schlüssels
- Änderungen nach oben verfolgen

Ausblick

- Algorithmus: Delete
- Implementierung
  - IO-Kosten
  - Nebenläufigkeit, Recovery
  - Variable Schlüssel
  - Bulk Loading
- Weitere Themen
  - Wann sind welche Indizes sinnvoll?
  - Indizierung ungewöhnlicher Daten
  - Schlüsselkomprimierung
  - Schätzungen für Indexzugriff über GiST
  - Optimierungen

Quellen

Joseph M. Hellerstein, Jeffrey F. Naughton and Avi Pfeffer. Generalized Search Trees for Database Systems. Technical Report #1274. University of Wisconsin at Madison, July 1995.

http://db.cs.berkeley.edu/jmh/

http://pages.cs.wisc.edu/~naughton/

http://www.eecs.harvard.edu/~avi/