Algorithmen und Datenstrukturen

Master

Hashing

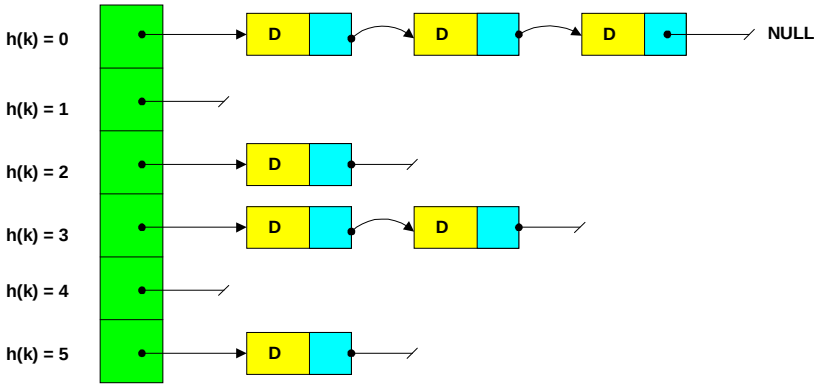
Motivation

- Sortiere Elemente in einem Array
- Statt Suche verwende mathematische Funktion zur Ermittlung des Slots
  - Schneller, weil in-cache
- Problem: welche Funktion soll verwendet werden?

Hash Tables

- Alternative zu Arrays
- Statt Index wird ein Schlüssel verwendet, der durch eine hash function berechnet wird
- Anzahl der Einträge in einer Hash Table ist üblicherweise klein im Vergleich zu möglichen Schlüsseln
- Hash Funktion kann für mehrere Einträge den gleichen Wert liefern (collision)

Hash Table



Hash Table

- Bestimmte Anzahl von Einträgen (buckets)
  - Beim Einfügen wird der Key berechnet. Anschließend wird Element in die entsprechende Liste gehängt.
  - Keine Beschränkung der möglichen Einträge, weil hinter jedem Bucket eine Liste hängt.
  - Wenn eine Liste zu lang wird, sinkt Performance -> Lösung: uniform hashing

Hashing

- Ziel: möglichst gute Gleichverteilung
  - Nachteil: ungefähre Anzahl der Einträge muß bekannt sein
  - Wenn zu viele Einträge muss Hash Table neu aufgebaut/organisiert werden
- Zwei Methoden
  - Divisionsmethode
  - Multiplikationsmethode

Hashing

- Divisionsmethode:
  - Bei m möglichen Einträgen wird der Rest der Division Schlüssel k / Anzahl der Einträge m bestimmt

$$h(k) = k \bmod m$$

  - Für m sollte eine große Primzahl gewählt werden

Hashing

- Multiplikationsmethode
  - Multipliziere Schlüssel k mit einem Faktor 0<A<1, nimm davon den Bruchteil, multipliziere mit der Anzahl der Einträge m und runde die Zahl ab

$$h(k) = m(kA \bmod 1)$$

$$A = (\sqrt{5} - 1)/2 = 0.618$$

Universelles Hashing

- Solange Anzahl der Schlüssel kleiner ist als die Anzahl der Speicherplätze kommt es zu keinen Kollisionen
- Wenn doch, müssen Überläufer in einer Liste gespeichert werden
- Entfernen eines Schlüssels ist hier kein Problem

Offenes Hashing

- Keine externen Listen
- Bei Kollision wird ein Alternativ-Slot im Hash-Array bestimmt (offene Stelle)
  - Definiere Reihenfolge von Slots, die bei Kollision betrachtet werden (Sondierungsfolge)
  - Wenn erster freier Slot gefunden wird, wird der Kandidat dort gespeichert
  - Verschiedene Strategien für Sondierungsfolge (linear, quadratic, double hashing, …)

Offenes Hashing

- Problem beim Löschen – Sondierfolge wird unterbrochen
  - Lösche nicht wirklich sondern markiere den Slot Eintrag als gelöscht – kann bei neuerlicher Kollision ersetzt werden
- Offenes Hashing dann, wenn hauptsächlich eingefügt und gesucht, aber selten gelöscht wird.

Dynamisches Hashing

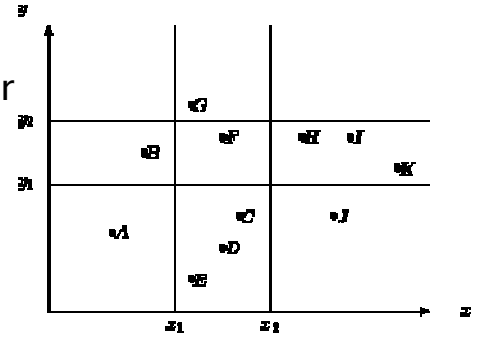
- Für stark wachsende oder schrumpfende Datenbestände
- Bedeutung für Daten auf externen Speichermedien
  - Hash Tabelle ist blockweise organisiert
  - Für einen Block wird eine Hashfunktion verwendet
  - Beim Einfügen eines neuen Blocks wird ein Teil der Liste mit neuer Hash Funktion reorganisiert
  - Muss mir merken, welche Teile mit welcher Hash Funktion belegt sind

Mehrdimensionale Schlüssel

- Beispiel: Name + Adresse + Tel.Nr
  - Finde Namen -> OK
  - Finde alle Einwohner einer Strasse -> ???
  - Finde alle Einwohner mit Telefonnummer in einem bestimmten Intervall -> ???
- Speichere Schlüssel in mehrdimensionalem Grid
  - Punkte nahe beieinander für Bereichsabfragen

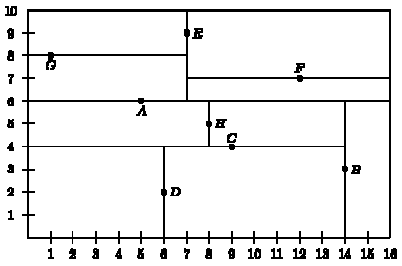
Beispiel: 2 Schlüssel

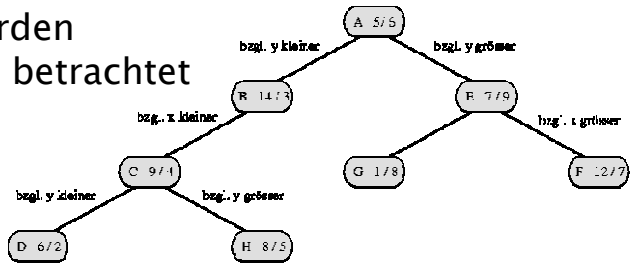
- 2 Dimensionen
- Beliebig erweiterbar
- Für Suche:
  - Projektion von x
  - Projektion von y
  - Dann Schnittmenge



k-d-Baum

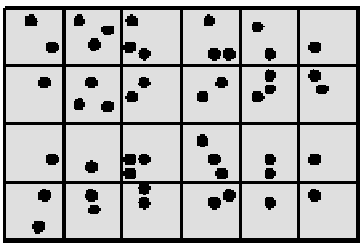
- Verallgemeinerter binärer Baum
- Jeder Knoten hat einen linken und rechten Nachfolger
- Schlüssel werden abwechselnd betrachtet





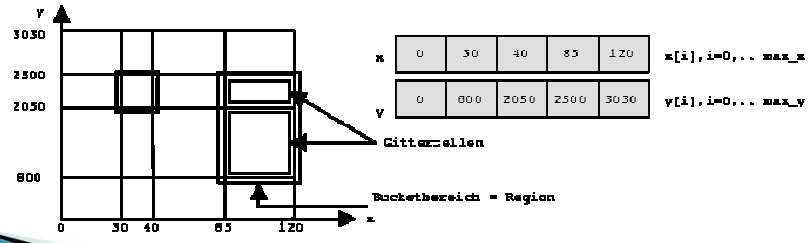
Gitter mit konstantem Abstand

- Wie Vektor bei eindimensionalem Problem
- Problem bei ungleicher Datenverteilung



Das Gridfile

- Datenbereich wird flexibel in Buckets geteilt
- Zugriff auf Daten in zwei Schritten
  - Bestimme Bucket-Adresse
  - Lies Bucket mit Zeigern auf Daten

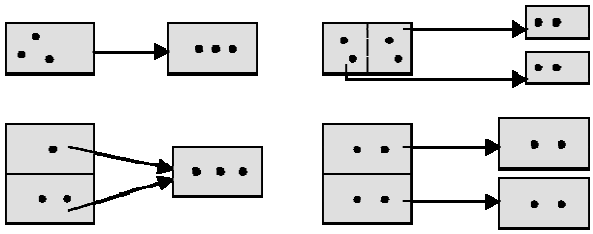


Das Gridfile

- Für k Dimensionen
  - k Skalen zum Einstieg in das Grid Directory
  - Grid Directory hält Verzeichnis der Buckets
  - Bucket hält max. n Datensätze
- Anwendung bei Datenbanken

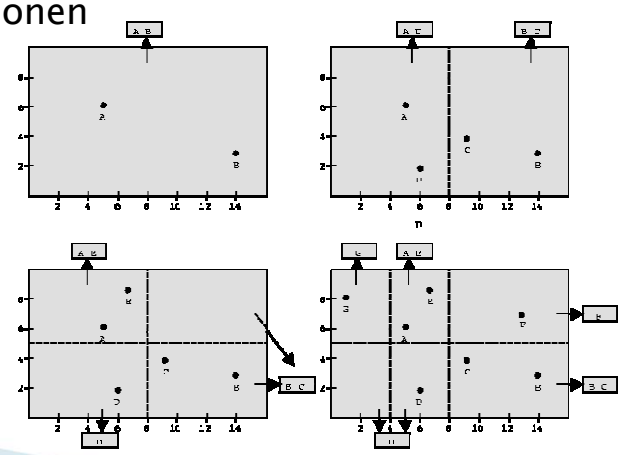
Wachsen und Schrumpfen

- Die grundsätzliche Idee besteht dain, die Skalen dem Inhalt der Buckets anzupassen



Wachsen und Schrumpfen

- Das Aufspalten erfolgt abwechselnd in den Dimensionen



Wachsen und Schrumpfen

- Beim Löschen von Datensätzen können Regionen wieder zusammengeführt werden, wenn sie die gleiche Breite haben
- Mischen nur wenn Einzelregion $< 30\%$ belegt und wenn neue Region dann $< 70\%$ belegt
  - Sonst Gefahr einer baldigen neuerlichen Teilung

Beispiel 4

- C# stellt eine Hashtable Klasse zur Verfügung (System.Collections)

- Schreibe ein Programm, das einen Text einliest und die Wörter
  - in einer Hashtable
  - in einem modifizierten CArray (für Strings)

einträgt. Untersuche mit der bereits bekannten Timing-Klasse das Suchverhalten für jeweils 20 verschiedene Wörter.