

Exkurs: Datenstruktur Quad Tree

Prof. Dr. Petra Mutzel



Lehrstuhl für
Algorithm Engineering LS11
Universität Dortmund

27. VO WS07/08 04. Februar 2008

Mehrdimensionale Suchstrukturen

• Gegeben:

- Menge S von N Punkten in R^k
- Familie U von Untermengen von R^k (Ranges)
- $\delta \in U$

• Gesucht:

- Vorverarbeitung von S , so dass Abfragen der Art:
"Berichte alle Punkte in $S \cap \delta$ " effizient berichtet werden können.

• Beispiel: Datenbankabfragen

Anwendungsbereiche

- Datenbanken
- Computergraphik / Computer Vision
- Computer-Aided Design
- Geographische Informationssysteme
- Bildverarbeitung
- Mustererkennung
- Document-Retrieval
- Data Mining
- ...

Charakterisierung (1)

- Welche Datentypen werden gespeichert?
 - S ist ungeordnete Menge (z.B. Index)
 - S ist kartes. Produkt $S^1 \times S^2 \times \dots \times S^k$ geordn. Mengen
- Dimension:
 - k ist kleiner gleich 10
- Operationen:
 - Find, Insert, Delete, (Pred., Succ., Min, Max)
- Welche Speichermedien?
 - intern vs. extern

Charakterisierung (2)

- Welche Objekttypen werden gespeichert?
 - Punkte, Container (z.B. Quader in 3D), komplexere
 - Lage fixiert oder beweglich?

- Welche Abfragen und wie oft?
 - Ist Punkt enthalten?
 - Aufzählung aller Punkte, die in gewünschtem k -dim. Bereich liegen
 - Welche Punkte liegen in der Nähe eines Punktes?
 - Finde die n nächsten Nachbarn eines Punktes
 - Exakte vs. partielle Abfragen
 - Einmalige vs. viele Abfragen

Wir betrachten folgende Abfragen:

- Punkt-Abfrage (Point Query):
 - Ist ein gegebener Datenpunkt in $S \in R^k$ enthalten, und falls ja, dann finde diesen.

- Bereichsabfrage (Range Query):
 - Berichte alle Punkte aus S , deren k Schlüssel in den gewünschten Bereichen liegen.

Einfache Datenstrukturen

- Sequentielle Liste
 - Aufwand: $O(Nk)$

| Name | X-key | Y-key |
|------|-------|-------|
| D | 5 | 45 |
| C | 35 | 40 |
| O | 25 | 35 |
| M | 50 | 10 |

- Invertierte Liste (Knuth 1973)
 - Sortierte Liste pro Schlüssel
 - Durchschnittlicher Aufwand: $O(N^{1-1/k})$
 - Aufwand: $O(Nk)$

| X-key | Y-key |
|-------|-------|
| D | M |
| O | O |
| C | C |
| M | D |

``Fixed Grid`` Methode

- Suchraum wird in gleiche Teile (*Buckets*) der Größe r aufgeteilt, wobei r der Suchradius ist
- Realisiert als k -dim. Array mit einem Eintrag per Bucket; jeder Bucket enthält Punkte in Form einer einfachen Liste
- Durchschnittlicher Suchaufwand für Bereichssuche (Bentley 1977): $O(F 2^k)$, wobei F die Anzahl der berichteten Punkte ist
- Effizient, wenn fixer Radius und Datenpunkte gleichmäßig im Raum verteilt sind (Kartographie)
- Teilt den Raum auf

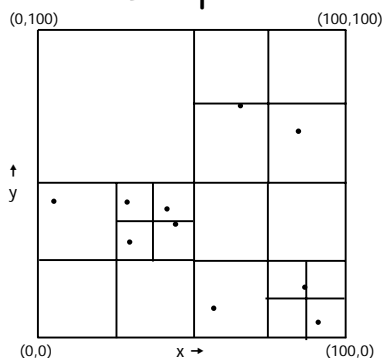
Region Quadtrees

- Repräsentation eines 2-dim. Binärbildes (Region Data)
- Rekursive Teilung eines 0/1-Bereiches in vier gleich große Quadranten, STOP falls Block nur 0 oder nur 1 enthält
- Suchbaum mit Grad 4 (s. Beispiel)
- Jedes Kind eines Knotens repräsentiert Quadranten (NW,NE,SW,SE)
- Blätter → Aufteilung nicht weiter notwendig
- Blätter sind entweder ``weiß`` oder ``schwarz``, innere Knoten sind ``grau``

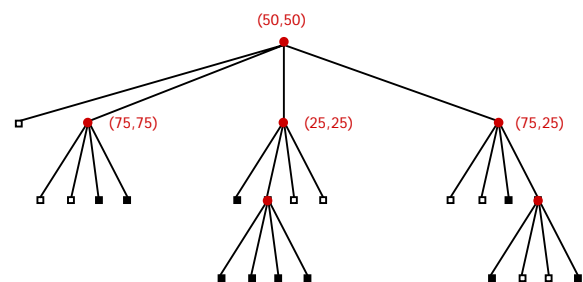
Point-Region Quadtrees

- Repräsentation von Punkten in einem k -dim. Bereich
- Rekursive Teilung eines quadratischen Bereiches in vier gleich große Quadranten, STOP falls Block nur 0 oder nur 1 Punkt enthält
- Jedem Feld wird ein Knoten in einem Suchbaum mit Maximal-Grad 4 zugeordnet
- Jedes Kind eines Knotens repräsentiert Quadranten (NW,NE,SW,SE)
- Blätter → Aufteilung nicht weiter notwendig
- Blätter sind entweder ``weiß`` (falls kein Punkt enthalten ist) oder ``schwarz`` (sonst), innere Knoten sind ``grau``

Point-Region Quadrees: Beispiel



PR Quadtree für Beispiel



Aufbau eines PR Quadtree

Top-Down Aufbau:

- Starte mit Feld B, das alle Knoten enthält
- Sei v_B der zugehörige Knoten im Suchbaum
- Falls B mehr als einen Knoten enthält, dann
 - erzeuge 4 Kinder von v_B im Suchbaum
 - weise jedem Kind-Feld B^i alle Knoten aus B zu, welche in B^i enthalten sind
 - entferne leere Kind-Felder v_{B^i} im Baum

Alternativ: Insert-Aufbau:

- Starte mit leerem Feld B und füge iterativ die Knoten ein
- Einfügen geht ähnlich wie bei binären Suchbäumen: suche das richtige Feld, Suche endet an Blatt, füge ein.

Morton Order

Morton 1966, space filling curve

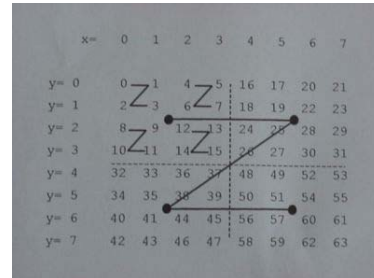
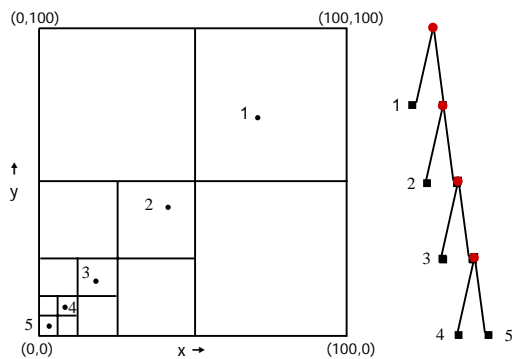


Abb. aus Wikipedia

Laufzeit ?

Baum hat Tiefe $N \rightarrow$ Laufzeiten beider Aufbau-Algorithmen: $O(N^2)$



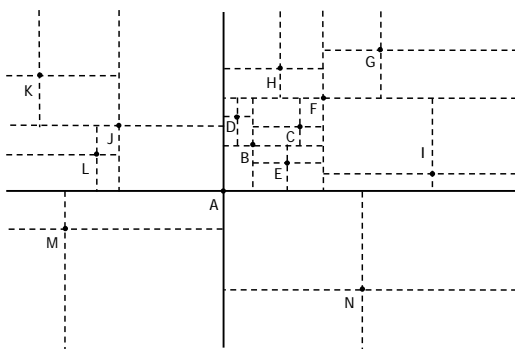
Point Quadrees

- Eingeführt von Finkel & Bentley 1974
- Multidimensionale Verallgemeinerung von binären Suchbäumen
- Verheiratung von "Fixed Grid" mit binären Suchbäumen
- Rekursive Teilung an Datenpunkten in jeweils vier Teile: NW, NE, SW, SE
- Hier Annahme:
 - $k=2$, Verallg. einfach
 - Jeder Punkt wird nur einmal besetzt

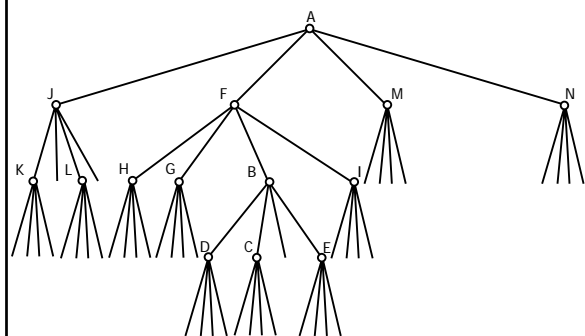
Petra Mutzel: Automatisches Zeichnen von Graphen, WS07/08

16

Beispiel:



Point Quadtree zu Beispiel

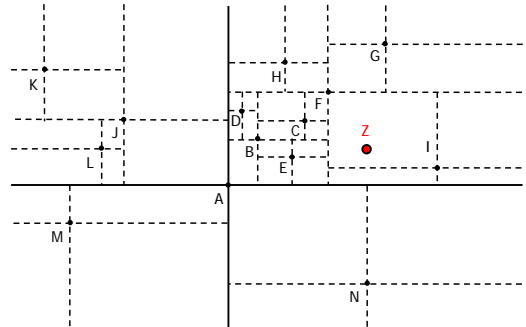


Point Quadtrees: Operation Insert

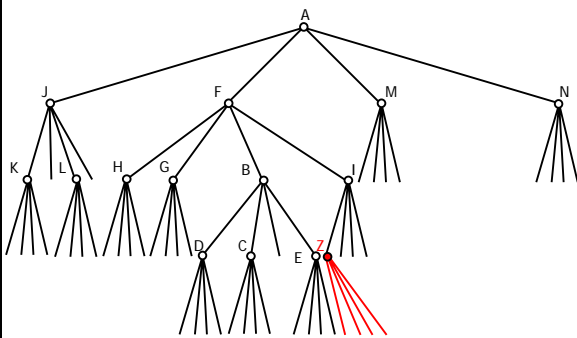
- Jeder innere Knoten enthält
 - Zeiger zu Kindern NW,NE,SW,SE
 - CHILD(P,I): gibt Kind im Quadranten I von Knoten P an
 - XCOORD, YCOORD: Koordinaten von Punkt
 - NAME: Information über Punkt (z.B. Name)

- Ähnlich wie für binäre Suchbäume:
 - Suche den Punkt (nach x und y-key)
 - Wenn Blatt erreicht ist, dann bestimme Position, an die eingefügt werden muss.

Beispiel: Insert Z



Point Quadtree zu Beispiel: Insert Z



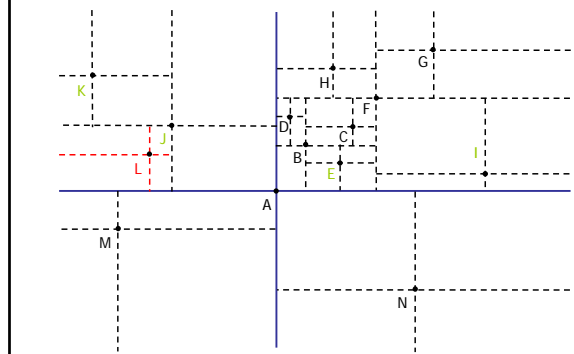
Point Quadtrees: Analyse Aufbau

- Aufbau eines Point Quadtrees:
 - Aufwand ist äquivalent zur Gesamtpfadlänge = Kosten, um nach allen Elementen einmal zu suchen
- Gesamtpfadlänge:
 - Hängt von Reihenfolge der Einfügungen der Punkte ab
 - Empirisch: $N \log_4 N$ (Finkel & Bentley)
 - Worst Case: $\theta(N^2)$
- Aufwand für Insert und Search
 - Empirisch: $O(\log_4 N)$
 - Worst Case: $O(N)$
 - Re-Balancing Methoden sind möglich

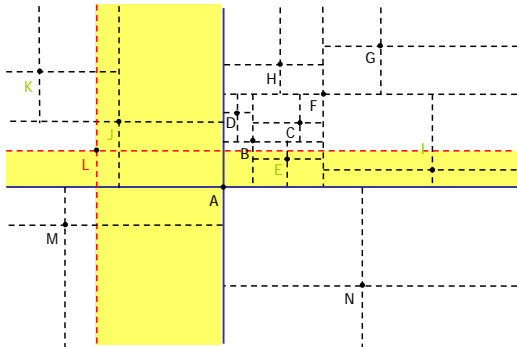
Point Quadtrees: Deletion

- Problem:
 - Geht nicht so leicht wie bei binären Suchbäumen
 - Beispiel: Deletion of A

Beispiel: Deletion



Beispiel: Deletion



Point Quadtrees: Deletion

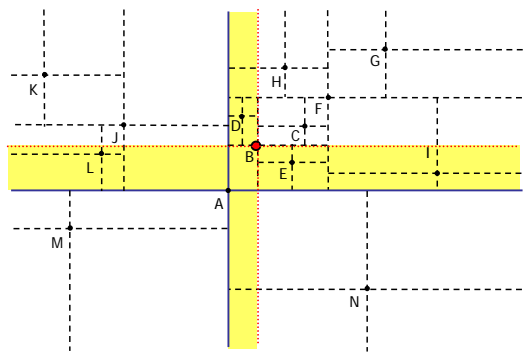
- Problem:
 - Unterbäume des gelöschten Knotens müssen eventuell neu eingefügt werden, denn sie sind nicht mehr im richtigen Quadranten bzgl. der neuen Wurzel
 - Original-Vorschlag war daher: alle diese Unterbäume neu einfügen
 - Besser: Vorschlag von Samet:

Point Quadtrees: Deletion

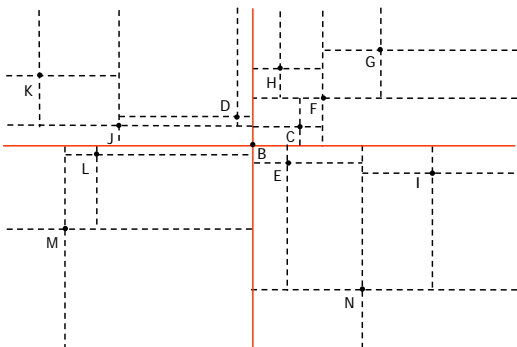
- Problem:
 - Unterbäume der Wurzel müssen neu eingefügt werden
 - Alle Knoten mit deren Unterbäumen, die in der Zwischenregion liegen, müssen neu eingefügt werden.
- Idee:
 - Wähle in jedem Unterquadranten des zu entfernenden Knotens einen Kandidaten aus, der am nächsten bei x oder y-Koordinate ist.
 - Wähle aus diesen vier Kandidaten dann den besten aus.

Delete Punkt A

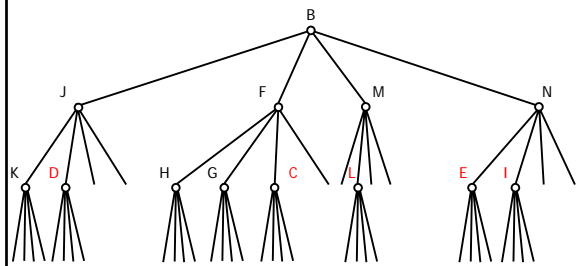
Beispiel:



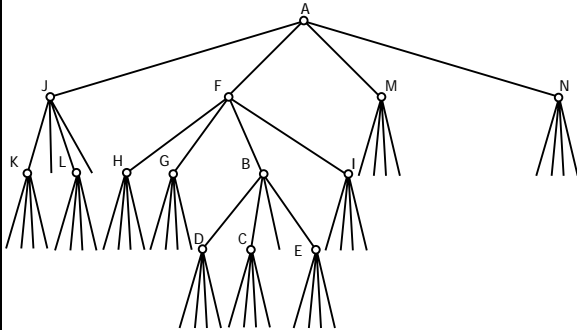
Beispiel nach Deletion von A



Quadtree zu Beispiel nach Deletion von A



Point Quadtree vor Delete A



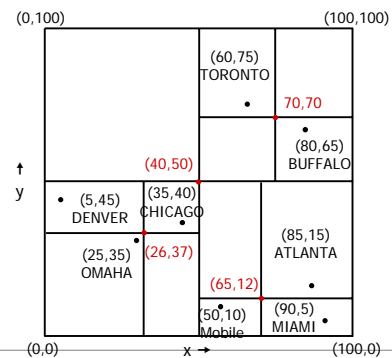
Point Quadtrees: Analyse Deletion

- Theoretisch (Bentley 1988):
 - Aufwand bei gleichmäßig verteilten Daten für die Anzahl der Neueinfügungen geht um 83% zurück gegenüber der Neueinfügung aller Teilbäume.
- Empirisch (Bentley 1988):
 - Empirisch: $N \log_2 N$ vs. deutlich größer in Original
 - Gesamtpfadlänge verringert sich leicht vs. deutlich Verlängerung in Original
- Worst Case:
 - $O(N^2)$
- Deletion sehr komplex!
- Alternative: Pseudo Quadtrees

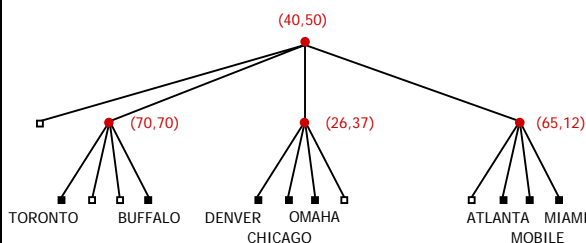
Pseudo Quadtrees

- Overmars und van Leeuwen 1982
- Idee:
 - Rekursive Aufteilung des Raumes an Punkten, die nicht Datenpunkte sind, in Quadranten, Unterquadranten, etc., bis jeder Unterquadrant höchstens einen Datenpunkt enthält.

Pseudo Quadtrees: Beispiel



Pseudo Quadtree für Beispiel



Pseudo Quadtrees:

- Aufbau:
 - Für je N Datenpunkte im k -dim. Raum existiert ein Partitionierungspunkt, so dass jeder Quadrant höchstens $\lceil N/(k+1) \rceil$ Datenpunkte enthält.
- Analyse:
 - Dann besitzt der Pseudo Quadtree eine Tiefe von höchstens $\lceil \log_{k+1} N \rceil$ und kann in Zeit $O(N \log_{k+1} N)$ gebaut werden.

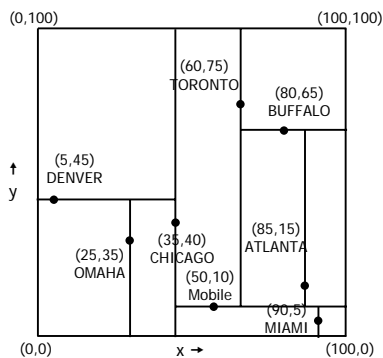
Point Quadrees: Diskussion

- Nachteile bei höheren Dimensionen:
 - An jedem Knoten des Baumes sind k Vergleiche notwendig (um den Quadranten zu bestimmen)
 - Hoher Speicherplatzverbrauch: Jedes Blatt benötigt k viele NULL Pointer, auch jeder innere Knoten besitzt immer wieder NULL Pointer
 - Speicherplatzverbrauch pro Knoten: $k+2^k+1$ Wörter für Koordinaten, Kinder und Info

K-D Trees:

- Bentley 1975
- Idee:
 - Binärer Suchbaum mit der Eigenschaft, dass in jeder Tiefe nach einer anderen Dimension orthogonal aufgeteilt wird.
 - Z.B. $k=2$: nach x -Koordinaten auf den Schichten mit gerader Nummer (Beginn bei Schicht 0), nach y -Koordinaten auf den ungeraden Schichten.
 - Aufteilung basiert auf den Datenpunkten
 - BSP Trees (Fuchs, Kedem, Naylor 1980): K-D Trees, bei denen nicht orthogonal aufgeteilt wird (beliebige Hyperebenen)

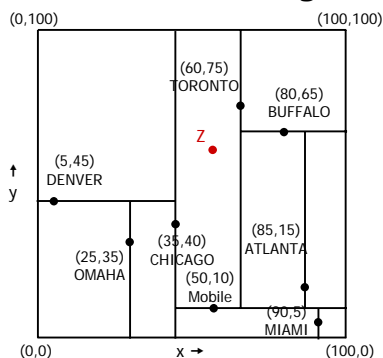
K-D Tree: Beispiel



K-D Trees:

- Datenstruktur:
 - LEFT, RIGHT: linkes und rechtes Kind (referenziert als SON(P,I) bzw. LOSON(P) und HISON(P))
 - XCOORD, YCOORD, ...
 - NAME
 - DISC: Diskriminator bzgl. k -tem Schlüssel
 - Abmachung für Diskriminatoren: gleiche Schlüsselwerte befinden sich im rechten Teilbaum

K-D Tree: Einfügen



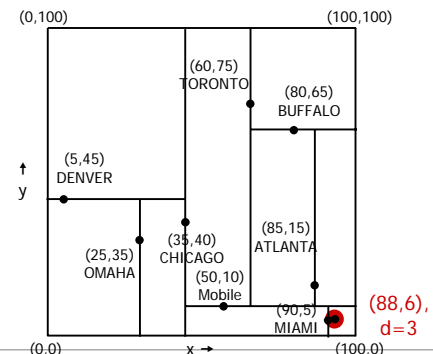
K-D Trees: INSERT

- Analog zu binären Suchbäumen:
 - Wir suchen den Punkt abwechselnde basierend auf den k Schlüsseln
 - Wenn das Blatt erreicht ist, haben wir die Einfüge-Position gefunden
- Analyse:
 - Form des Baumes hängt von Einfügereihenfolge ab
 - Durchschnittliche Tiefe: $O(\log_2 N)$
 - Worst Case Tiefe: $O(N)$, Aufbau: $O(N^2)$
 - Optimierung ähnlich wie bei Quad Trees
 - Alternative: Adaptive K-D Tree

K-D Trees: Bereichssuche

- Ausgabe aller Knoten (x,y) , die sich innerhalb des Gebietes mit Radius d (euklidisch) um (a,b) befinden, d.h. $(a-x)^2+(b-y)^2 \leq d^2$

K-D Trees: Bereichssuche



K-D Trees: Bereichssuche

- Analyse:
 - Worst Case für vollständigen K-D Tree:
 - $O(k N^{1-1/k})$

Diskussion K-D Trees

- An jedem Knoten muss nur jeweils ein Schlüsselvergleich durchgeführt werden.
- Speicherplatz:
 - Blätter: es gibt nur maximal zwei NULL-Pointer
 - Benötigter Speicherplatz pro innerer Knoten: $1+1+1+k$ für LEFT, RIGHT, NAME, DISC + k Wörter für k Schlüssel
- Adaptive K-D Trees:
 - Innere Knoten benötigen nur 5 Wörter
- Nachteil gegenüber Quadtree:
 - Quadtree ist eine parallele Datenstruktur (k Schlüsselvergleiche), K-D Trees nicht