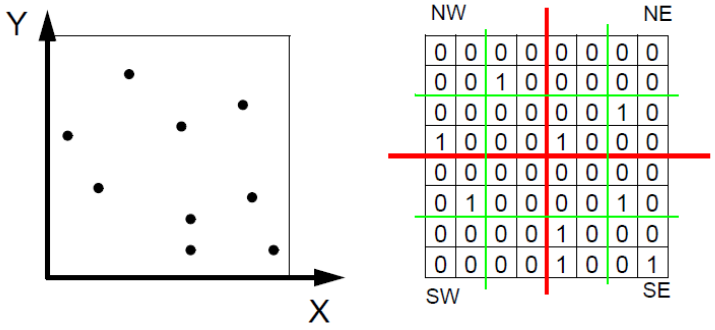
4.4 MX-Quadtrees (I)

MatriX Quadtree

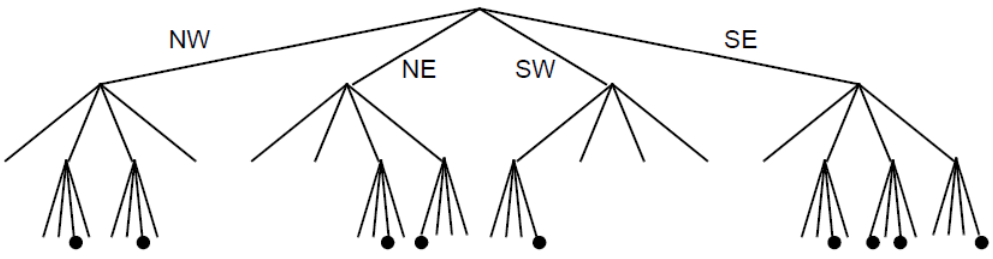
- Verwaltung 2-dimensionaler Punkte
- Punkte als 1-Elemente in einer quadratischen Matrix mit Wertebereich {0,1}
- rekursive Aufteilung des Datenraums in die Quadranten NW, NE, SW und SE
- feste Auflösung des Datenraums in $2^p \cdot 2^p$ Gitterzellen



4.4 MX-Quadtrees (II)

Baumstruktur

- *Interne Knoten* besitzen 4 Verweise auf Söhne (NW, NE, SW, SE)
- *Blattknoten* enthalten 0 oder 1 Datensatz
- Datensätze befinden sich alle auf demselben Level
- Für jeden internen Knoten gibt es (mindestens) einen Teilbaum mit Datensatz
- Datenraum mit $2^p \cdot 2^p$ *Gitterzellen:*
  p = Höhe des MX-Quadtrees (Abstand eines Datensatzes zur Wurzel)



⇒ in jeder Gitterzelle kann sich nur ein Punkt befinden

4.4 MX-Quadtrees (III)

Gegeben

- Breite des Gitters $2 \cdot W$
- Zentrum des Gitters (W,W)

Gesucht

- Quadrant eines Punkts (X,Y)

Algorithmus

```
MX_Compare (X, Y, W);
        IF X < W THEN
                IF Y < W THEN RETURN 'SW'
                ELSE RETURN 'NW'
        ELSE IF Y < W THEN RETURN 'SE'
                ELSE RETURN 'NE');
```


4.4 MX-Quadtrees (IV)

Algorithmus Punktanfrage

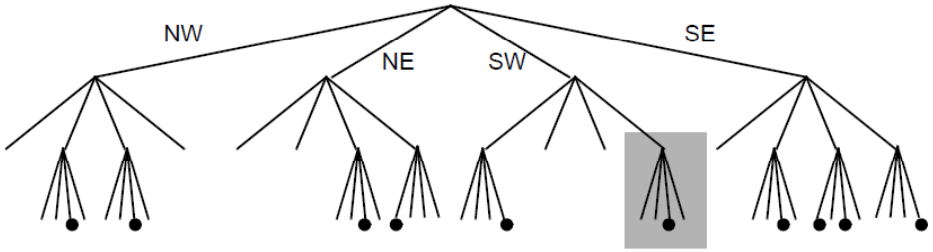
```
Point_Query (X, Y, W, Node);
        Q:=MX_Compare(X,Y,W);
        Q-Son:= Reference to Quadrant Q of Node;
        IF Q-Son = NULL THEN RETURN NULL
        ELSE
                IF W = 1 THEN RETURN Data of Q-Son of Node
                ELSE Point_Query(X MOD W,Y MOD W, W/2,Q-Son);
```

- Erster Aufruf mit Node = Wurzel des MX-Quadtrees und $W = 2^{p-1}$ des MX-Quadtrees
- Punktanfrage ist auf einen Pfad des MX-Quadtrees beschränkt

4.4 MX-Quadtrees (V)

Einfügen

0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0
1	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0
0	0	0	0	1	0	0	0
0	0	1	0	1	0	0	1



Eigenschaften

- falls in dem Blatt schon ein Datensatz vorhanden ist, wird er durch neuen
  Datensatz überschrieben
- Einfügereihenfolge hat keinen Einfluß auf Datenstruktur

4.4 MX-Quadtrees (VI)

Algorithmus Einfügen

```
MX_Insert (X, Y, Data, W, Node);
IF W = 1 THEN
        Q:=MX_Compare(X,Y,W);
        Q-Son:= Reference to Quadrant Q of Node;
        IF NULL(Q-Son) THEN Create NW, NE, SW and SE-Son of Node;
        Insert (X,Y,Data) into Q-Son of Node;
ELSE
        Q:=MX_Compare(X,Y,W);
        Q-Son:= Reference to Quadrant Q of Node;
        IF NULL(Q-Son) THEN Create NW, NE, SW and SE-Son of Node;
        MX_Insert(X MOD W,Y MOD W,Data,W/2,Q-Son)
```

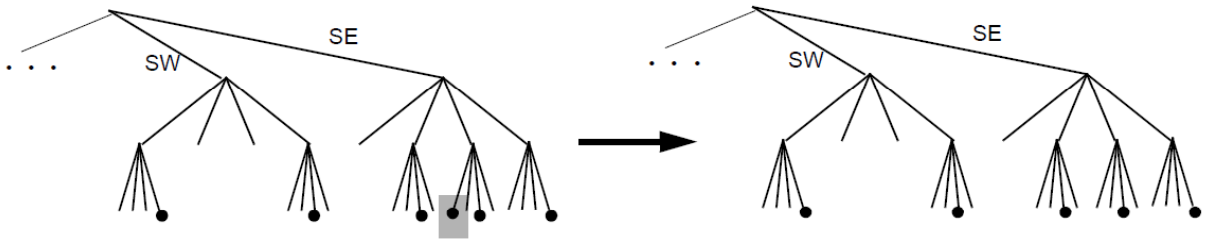
4.4 MX-Quadtrees (VII)

Algorithmus Einfügen

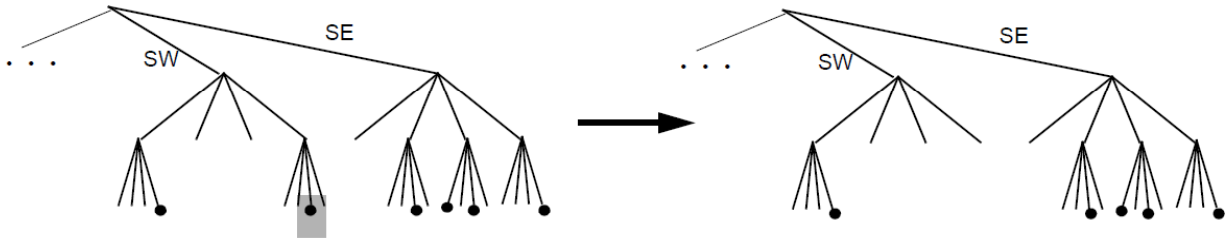
- Erster Aufruf mit Node = Wurzel des MX-Quadtrees und W = 2^{p-1} des MX-Quadtrees
- Einfügen ist auf einen Pfad des MX-Quadtrees (plus die Brüder) beschränkt

4.4 MX-Quadtrees (VIII)

Löschen



Löschen mit Kollabieren

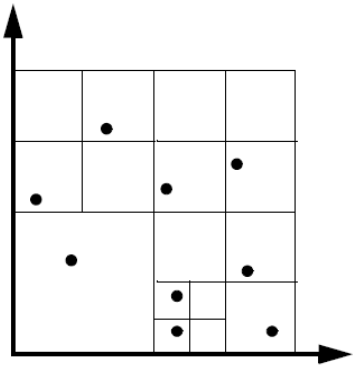


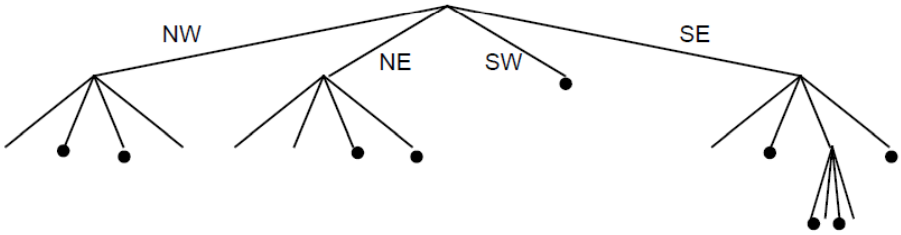
- Löschen ist auf die Knoten eines Pfades und die jeweiligen Brüder beschränkt

4.4 PR-Quadtrees (I)

Point Region Quadtree

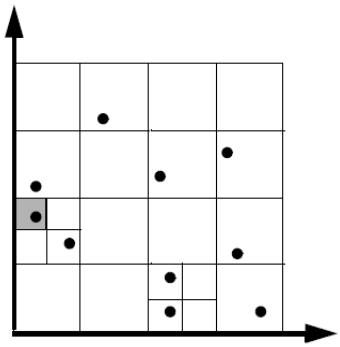
- variable Auflösung des Datenraums
- Komprimierung eines internen Knotens eines MX-Quadtrees, falls im Teilbaum nur ein Datensatz vorhanden
- Dann wird der Datensatz direkt in dem internen Knoten abgespeichert und dessen vier Kinderknoten werden freigegeben.
- Jeder interne Knoten besitzt mindestens zwei Punkte in den darunterliegenden Teilbäumen

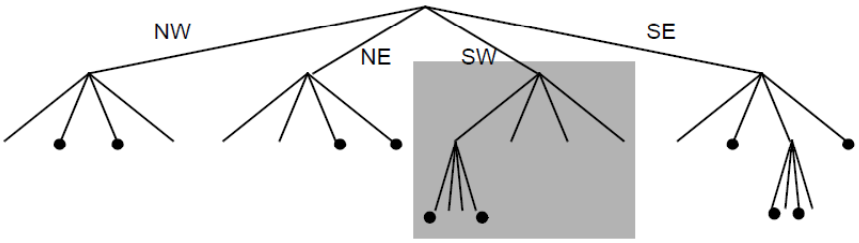




4.4 PR-Quadtrees (II)

Einfügen





- Suche den Einfügeknoten *N*
- Falls *N* ein leerer Knoten, so füge den Datensatz in *N* ein
- Andernfalls teile den Datenraum des Teilbaums von *N solange rekursiv* auf, bis die beiden Punkte in unterschiedlichen Quadranten (Knoten) liegen

4.4 PR-Quadtrees (III)

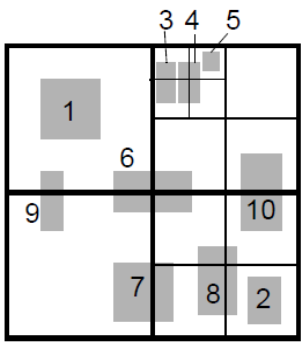
Eigenschaften

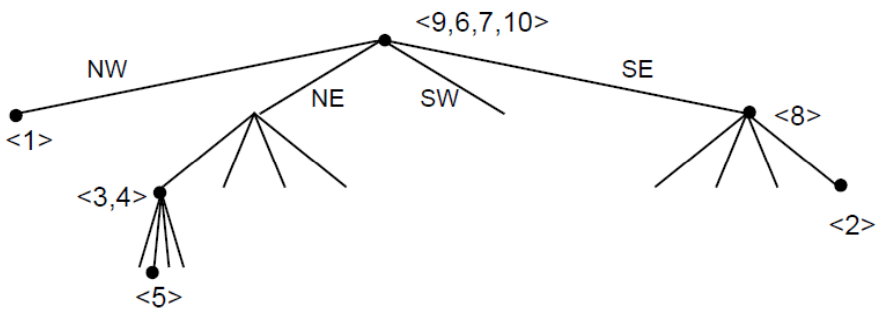
- Einfügereihenfolge hat keinen Einfluß auf Datenstruktur
- Entstammen die Punkte einem Datenraum mit $2^p \cdot 2^p$ Gitterzellen, so kann der Speicherplatzbedarf für n Punkte $O(n \cdot p)$ betragen.

4.4 Quadtrees für Rechtecke (I)

Idee 1

- Rechtecke (MURs) werden durch die minimal umgebende Zelle eines Quadtrees repräsentiert
- Speichere zu jedem Knoten eine Liste von Rechtecken, die vollständig in der dem
  Knoten zugeordneten Region liegen, aber nicht in der Region eines darunterliegenden Kinderknotens
- Beispiel:





4.4 Quadtrees für Rechtecke (II)

Idee 1 (cont.)

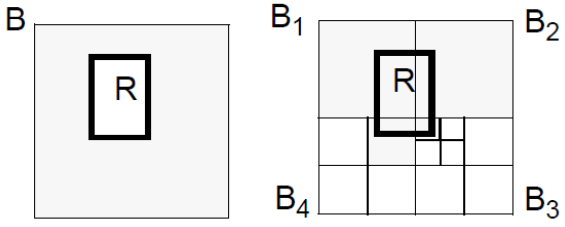
⇒ unbeschränkte Länge der Listen (schwierige Organisation auf Sekundärspeicher)

⇒ viele Geo-Objekte, die nicht die Anfrage erfüllen, aber deren minimal umgebende Quadtree-Zellen die Anfrage erfüllen *(schlechte Approximation)*

4.4 Quadtrees für Rechtecke (III)

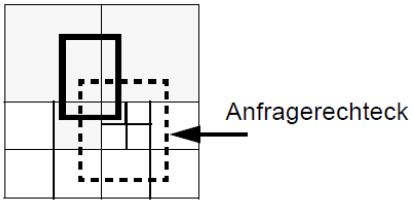
Idee 2

- repräsentiere ein Rechteck durch mehrere Quadtree-Zellen
- Repräsentation eines Rechtecks R:
  - B sei die zu R gehörige minimal umgebende Quadtree-Zelle;
  - B_1, B_2, B_3 und B_4 seien die Zellen der darunterliegenden Kinderknoten.
  - Dann repräsentiere R durch die Zellen des Quadtrees, die $R \cap B_i$, $1 \leq i \leq 4$, minimal umgeben.



Repräsentation durch eine Zelle

Repräsentation durch mehrere Zellen



Anfrage liefert mehrmals die gleiche Antwort

\+ bessere Approximation des Rechtecks R, d.h. weniger Fehltreffer

\- die gleiche Antwort wird ggf. mehrfach gefunden

4.4 Quadtrees für Polygone

Ziel

- Abspeicherung von Linien und Polygonen direkt in einem Quadtree
- Clustering nicht nur der MURs, sondern der EBs selbst
- Minimierung des Speicherplatzbedarfs

PM-Quadtrees

- Rekursive Aufteilung der Menge von Eckpunkten / Kanten eines Polygons in Teilmengen, die durch eine Datenstruktur fester Grösse repräsentiert werden können
- Diese Datenstrukturen werden in einem Blattknoten des Quadtrees abgespeichert

Varianten

- Repräsentierung der Eckpunkte: PM1-Quadtrees, PM_2-Quadtrees, PM_3-Quadtrees
- Repräsentierung der Kanten: PMR-Quadtrees

4.4 PM1-Quadtree (I)

Idee

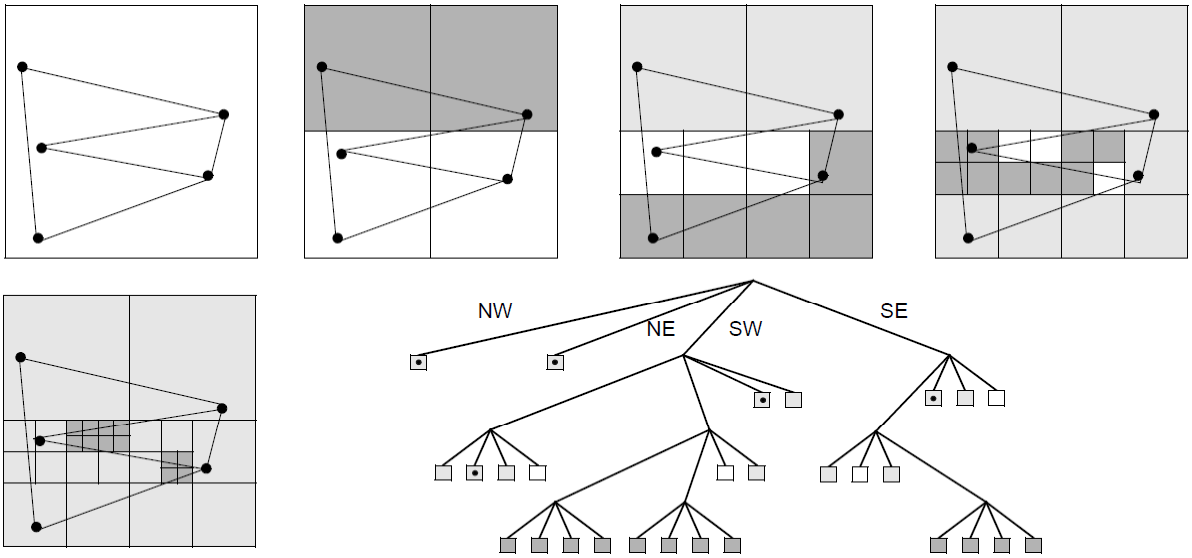
- Abspeicherung der Polygone durch ihre Eckpunkte, ohne dabei eine Approximation durch minimal umgebende Quadtree-Zellen oder minimal umgebende Rechtecke zu benutzen

Baumstruktur

- Eine *Blattzelle* ist eine Zelle (Quadrant) des Gitters, die durch ein Blatt des Quadtrees repräsentiert wird
- Höchstens ein Eckpunkt eines Polygons liegt in der Zelle eines Blattknotens
- Falls ein Blattknoten *B des* PM1-Quadtrees einen Eckpunkt E enthält, so müssen alle Kanten in *B* den Punkt *E* als Eckpunkt besitzen
- Falls eine Blattzelle *B* keinen Eckpunkt enthält, so darf durch *B* nur eine Kante führen

4.4 PM_1-Quadtree (II)

Beispiel



- Leistung hängt wesentlich von der Nähe zwischen Punkten und Kanten ab (Kanten in einem Eckpunkt mit kleinem Winkel ⇒ schlechtes Leistungsverhalten)

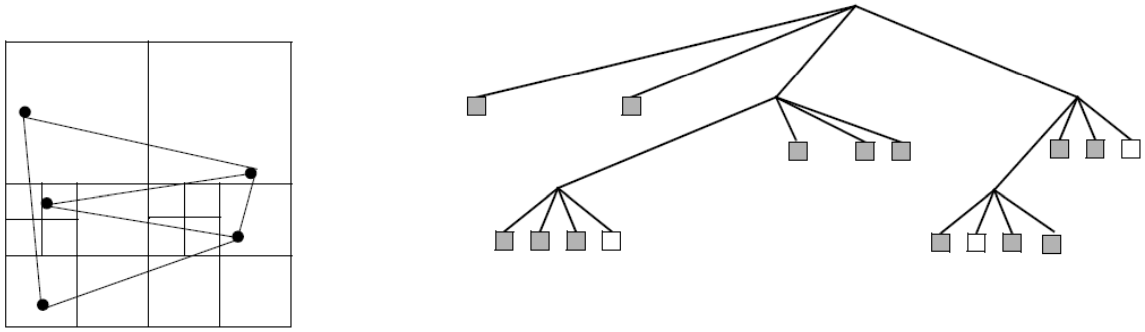
+ sehr einfache Datenstruktur für Blattknoten

4.4 PM_2-Quadtree

Baumstruktur

- wie bei PM_1-Quadtree
- Änderung: Falls ein Blattknoten *B* keinen Eckpunkt enthält, so dürfen durch *B* mehrere Kanten mit einem gemeinsamen Eckpunkt führen (anstatt nur einer Kante)

Beispiel



+ geringere Höhe des Quadtrees

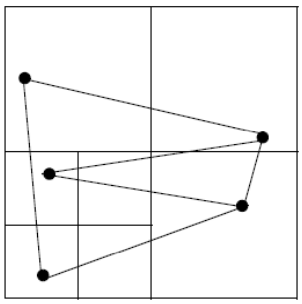
- komplexere Datenstruktur für Blattknoten

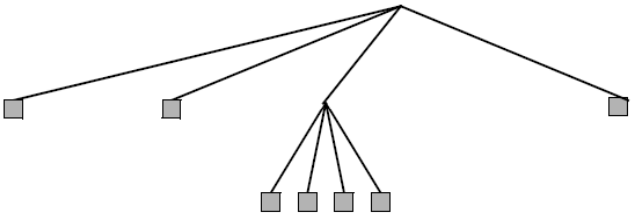
4.4 PM_3-Quadtree

Baumstruktur

- Höchstens ein Eckpunkt eines Polygons liegt in einer Blattzelle
- Eine Blattzelle kann Kanten mit beliebigen Eckpunkten enthalten

Beispiel





+ noch geringere Höhe des Quadtrees

- noch komplexere Datenstruktur für Blattknoten

4.4 PMR-Quadtree (I)

Baumstruktur

- Kanten werden in alle geschnittenen Blattzellen eingefügt
- C >1 versch. Kanten können in einer Blattzelle abgespeichert werden

Einfügen

- Füge eine neue Kante in alle Blattzellen ein, die sich mit der Kante schneiden
- Falls die Zelle übergelaufen ist (mehr als c Kanten), so spalte die Zellen (ggf. rekursiv) in 4 Teile auf
- Falls es nicht möglich ist, den Überlauf zu beseitigen, dann schreibe die überzähligen Kanten in eine Überlaufzelle, die mit der ursprünglichen Blattzelle verkettet ist

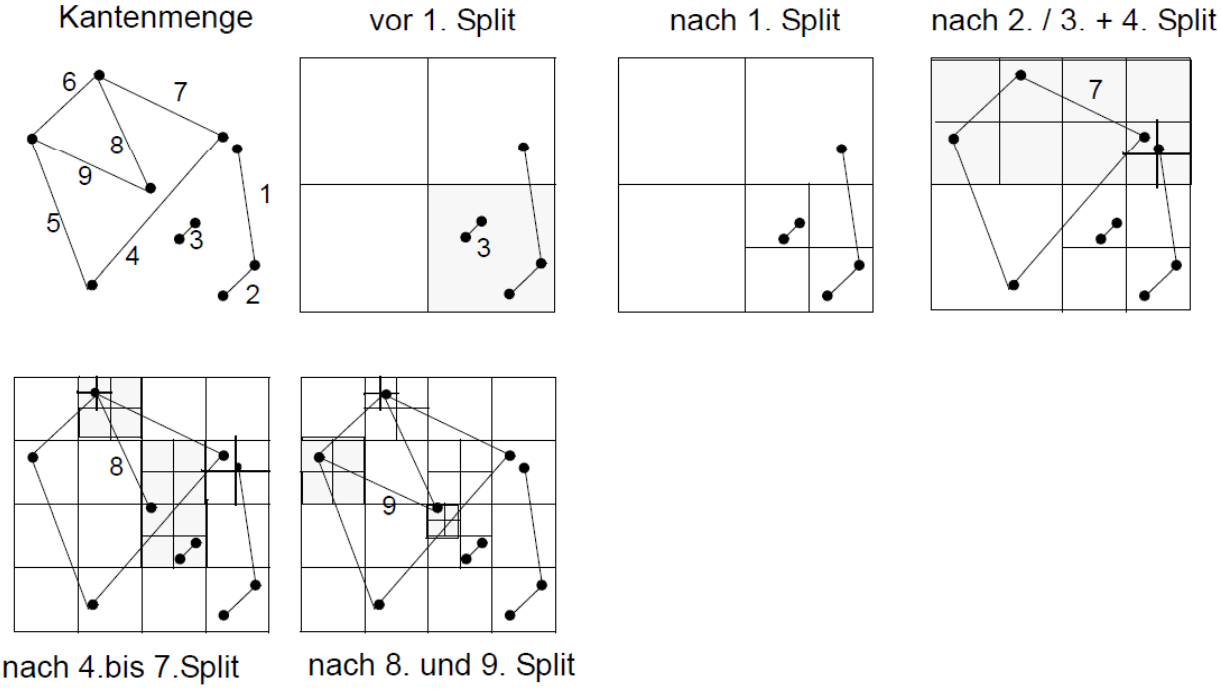
Experimentelle Untersuchung

- Der PMR-Quadtree ist den PM_x-Quadtrees bezüglich der Speicherplatzausnutzung überlegen

4.4 PMR-Quadtree (II)

Beispiel

$c = 2$



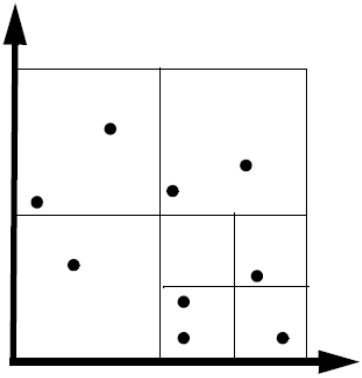
4.4 Abb. auf Sekundärspeicher (I)

Abbildung der Zugriffsstrukturen auf Sekundärspeicher

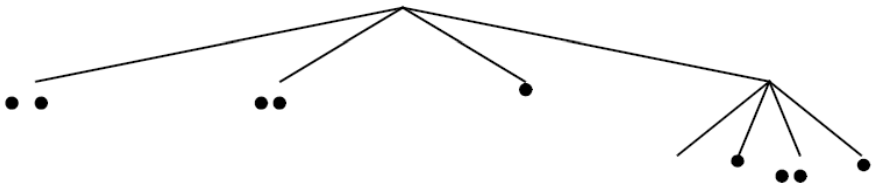
- R-Baum: Knoten = Seite
- Quadtree: ? (in einem Knoten befinden sich nur wenige Einträge)

1. Ansatz: Anpassung der Kapazität der Blattzellen

- Erhöhe die Kapazität der Blattzellen



PR-Quadtree mit einer Blattknotenkapazität 2



– Organisation der internen Knoten bleibt problematisch

4.4 Abb. auf Sekundärspeicher (II)

2. Ansatz: Einbettung in eindimensionalen Raum

- Nur gefüllte (schwarze) Blattzellen werden betrachtet
- Eine Blattzelle entspricht einer Datenseite (höhere Kapazität)
- Jede dieser Zellen erhält eine Ordnungsnummer
- Die Zellen werden durch eine herkömmliche, eindimensionale Zugriffsstruktur (z.B. B-Baum) verwaltet

Anforderungen

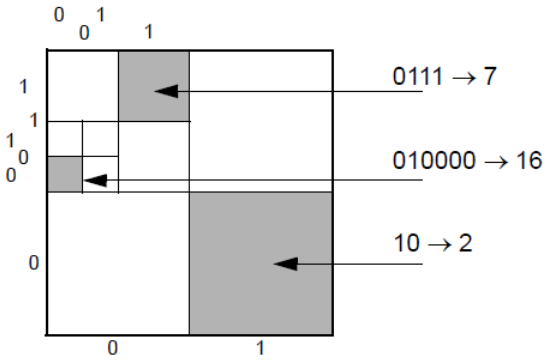
- Einfache Berechnung der Ordnungsnummer
- Erhalt von räumlicher Nachbarschaft in dieser neuen Ordnung (Annahme: räumlich benachbarte Objekte werden oft gemeinsam angefragt)

Fragestellung

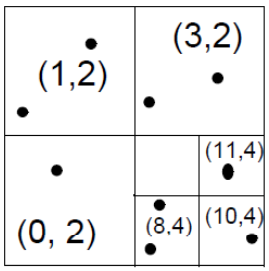
- Wie sieht eine geeignete Ordnung aus?

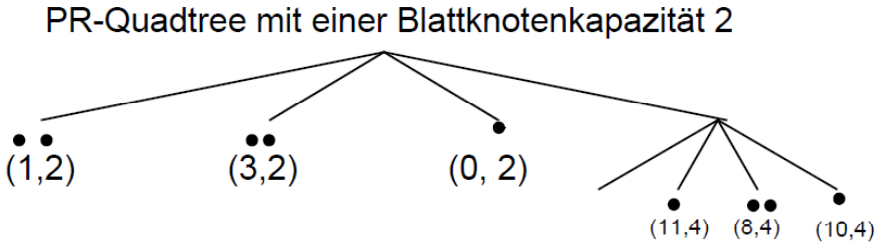
4.4 Linear Quadtree mit Z-Ordnung (I)

- *Codierung* von Quadtree-Zellen:



1. Mischen der beiden Bitfolgen, 2. Interpretation als Dezimalzahl
- *Level eines* Codes = Anzahl der Bits
- *Z-Wert* = (Dezimalwert des Codes, Level)





4.4 Linear Quadtree mit Z-Ordnung (II)

- Lineare Ordnung zur Verwaltung im B^+-Baum
  - Seien (c_1, l_1) und (c_2, l_2) zwei Z-Werte und sei $l = \min\{l_1, l_2\}$.
  - Dann ist die Ordnungsrelation $\leq_Z$ wie in 4.2 definiert:

$$(c_1, l_1) \leq (c_2, l_2) \text{falls } c_1 \text{ div } 2^{l_1 - l} \leq c_2 \text{div } 2^{l_2 - l}$$

  - Beispiele:
    $(1,2) \leq_Z (3,2)$, $(3,4) \leq_Z (3,2)$, $(1,2) \leq_Z (10,4)$

- Wenn eine Blattzelle (= Datenseite) des Quadtrees überläuft, die durch den Z-Wert
  (c,l) repräsentiert wird, dann Split der Seite in 4 Seiten gemäß Quadtree-Strategie
- diese Seiten besitzen die Z-Werte

  (4*c, l + 2), (4*c + 1, l + 2), (4*c + 2, l + 2), (4*c + 3, l + 2)

4.4 Quadtrees: Zusammenfassung

- Quadtrees sind die am häufigsten verwendeten räumlichen Zugriffsstrukturen in Geo-Informationssystemen
- Fülle von Varianten (siehe [Samet])
- Quadtrees werden eingesetzt für die Organisation 2-dimensionaler Punkte, Rechtecke, Streckenzüge und Polygone (für 3-dimensionale Objekte: Octtree)
- Repräsentation von Polygonen durch minimal umgebende Quadtree-Zellen (mitoder ohne Clipping), durch Eckpunkte oder durch Kanten
- Quadtrees können benutzt werden, um Anfragen wie die Punkt-Anfrage, die Fenster-Anfrage und den Spatial Join zu beantworten
- Quadtrees sind ursprünglich als eine Datenstruktur für den Hauptspeicher entworfen worden, können aber durch Verwendung raumfüllender Kurven auch für Sekundärspeicher genutzt werden