1 Quadtree-Strukturen, Buckets, Linearer Quadtree

1.1 *Kapiteleinführung*

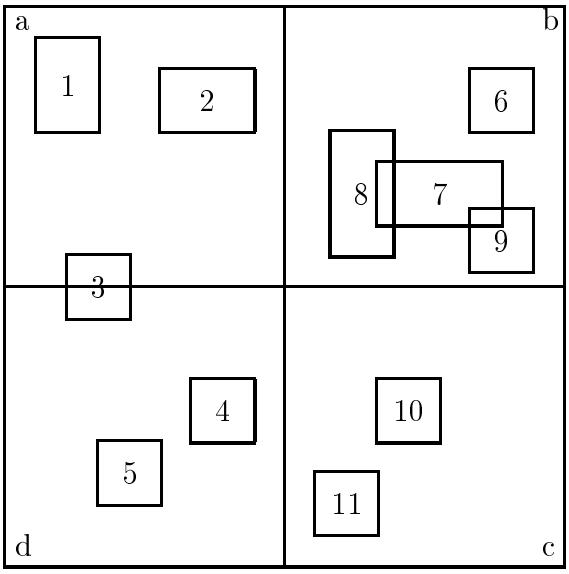
Die Quadtree-Datenstruktur wird wegen ihrer Einfachheit im Allgemeinen für die Beschleunigung des Zugriffs auf die Objekte der 2-D (D steht für dimensional) Ebene benutzt. Es ist auch die Verallgemeinerung auf die 2^D-Bäume möglich. Im 3D-Raum entsteht so die Octree-Datenstruktur.

Im folgenden skizzieren wir eine Variante für die Indexierung einer Sammlung von Rechtecken(mbb's genannt), die auf der Festplatte gespeichert sind. Diese Rechtecke sind neben Punkten für uns von Interesse, weil sie als minimale umbeschriebene Rechtecke (minimal bounding box - mbb) von 2-dimensionalen geometrischen Objekten einen schnellen Vortest ermöglichen, ob z.B ein Punkt in einem Objekt liegen kann.

Am Anfang betrachten wir ein rechteckiges Gebiet in einer Ebene zusammen mit den Objekten als Grundgebiet. Jedes Objekt wird durch einen Objekt-Identifikator repräsentiert. Dieser Objekt-Identifikator wird benutzt, um dadurch Objekte zu referenzieren. Wir sehen die Suche nach einem Objekt als erfolgreich an, wenn wir den zugehörigen Objekt-Identifikator (Objekt-ID) kennen. Jedem Objekt ordnen wir ein kleinstes umbeschriebenes Rechteck (mbb) zu. Mit dem mbb wird die Geometrie des Objektes näherungsweise erfaßt. Ein Eintrag (Datensatz) für ein Objekt besteht aus dem Paar (mbb, oid).

Die Datensätze sollen in Buckets, die jeweils einer Plattenseite entsprechen, gespeichert werden. Da die Größe eines Buckets (Seitenkapazität) beschränkt ist, können maximal N Einträge in ein Bucket aufgenommen werden. Wenn ein Bucket voll ist(es hat N Einträge) und ein neuer Eintrag hinzugenommen werden muss, dann werden noch 3 neue Buckets alloziert und alle Einträge auf diese 4 Buckets verteilt (jedem Bucket wird jeweils ein Viertel des ursprünglichen Gebiets zugeordnet). Jedes dieser vier Buckets wird graphisch als ein Quadrant des vom ursprunglichen Bucket erfassten Gebiets dargestellt. Dadurch hat man 4 Quadranten, die jeweils als NordWest (NW), NordOst (NO), SüdWest (SW) und SüdOst (SO) bezeichnet werden.

Das unten dargestellte Bild (Bild 1.1) zeigt als graphisches Beispiel die Partitionierung der willkürlich von uns vorgegebenen Sammlung und ihren asso-



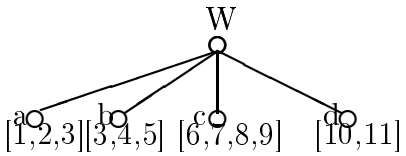


Abbildung 1: Quadtree-Anfangssituation

zierten Baum. In der Darstellung des Baumes sind bei den Blattknoten die in den Blattseiten gespeicherten Rechtecke angegeben. Die Seitengröße wird in unserem Beispiel gleich 4 angenommen.

1.2 *Grundoperationen (Einfügen, Suchen, Löschen) von Rechtecken*

Bei den Grundoperationen wird hier nur Geometrie betrachtet, es versteht sich, dass es immer den Datensatz (mbb,oid) betrifft.

Suchen:

Beim Suchen wird versucht festzustellen, ob das gesuchte Rechteck in dem Baum vorhanden ist. Es wird in den Blättern gesucht, dessen zugeordnetes Gebiet (hier Blattquadranten genannt) es schneidet. Dabei werden alle Pfade verfolgt, die zu den vom Rechteck geschnittenen Blattquadranten führen (es wird in dem Baum nach allen vom Rechteck geschnittenen Blattquadranten gesucht).
Dabei wird Seite(Bucket) (sagen wir p) gelesen, die den Blättern zugewiesen wurde.

Einfügen:

Einfügen = erfolglose Suche + Einfügen eines Rechtecks.

D.h. während des Einfügen wird genauso vorgegangen wie beim Suchen mit dem einzigen Unterschied , dass nach der Feststellung des Nicht-Vorhandenseins eines Rechtecks in dem Baum, wird das jeweilige Rechteck in jedes Blatt eingefügt, dessen zugeordnetes Gebiet (hier Blattquadranten genannt) es schneidet(vergleiche Bild 1.1 mit Bild 1). Beim Einfügen eines Rechteckes werden alle Pfade verfolgt, die zu den vom Rechteck geschnittenen Blattquadranten führen (es wird in dem Baum nach allen vom Rechteck geschnittenen Blattquadranten gesucht).

Beim Einfügen ergibt sich eine der folgenden Situationen:

1. Entweder p (Seite, in die das Objekt einzufügen ist) hat weniger als N Einträge, dann wird der neue Eintrag hinzugefügt.

2. Oder p hat N Einträge (ist voll und der neue Eintrag kann nicht eingefügt werden), dann wird

- der Quadrant in vier untereinander gleichgroße Subquadranten durch Halbieren der Seiten geteilt,
- drei neue Festplattenseiten allokiert,
- alle Einträge werden zwischengespeichert und zusammen mit dem neuen Eintrag in die 4 Seiten eingefügt

Im Endeffekt habe wir also 4 Buckets, die jetzt alte und neue Einträge enthalten. Dabei kann es vorkommen, dass ein oder mehrere Buckets wieder N+1 Einträge erhalten. Das Verfahren der Teilung ist rekursiv zu wiederholen, bis der neue Eintrag eingefügt werden konnte, d.h. bis die entstandene Buckets höchstens je N Einträge haben.

Die Rekursion bricht in der Regel ab, da die Subquadranten immer kleiner werden. Damit sinkt die Wahrscheinlichkeit, dass die Buckets der Subquadranten auch alle Einträge des Ausgangsrechtecks enthalten.

Die Rekursion wird nicht abbrechen, wenn ein Bucket N Einträge enthält, deren mbb alle mit dem Rechteck zusammenfallen, das dem Bucket entspricht. In diesem selten auftretenden Fall versagt die Datenstruktur. Er sollte bei der Programmierung aber beachtet werden. Gleiches gilt auch wenn die mbb's der N Objekte ineinander geschachtelt sind.

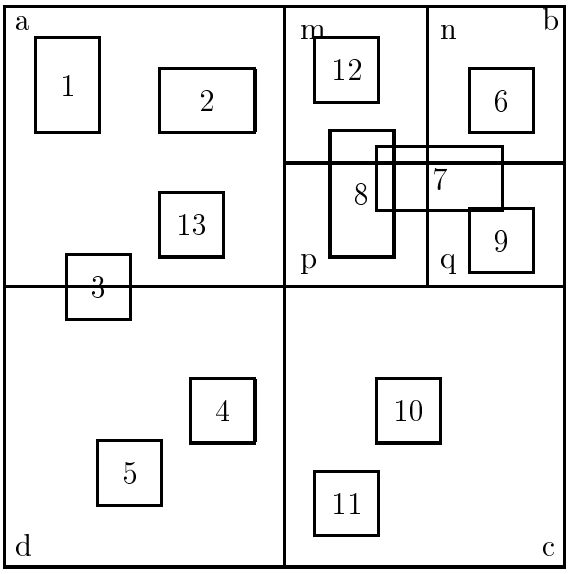
Beim Suchen wird genauso vorgegangen wie beim Einfügen mit dem einzigen Unterschied, dass bei der Suche kein einziger Eintrag hinzugefügt wird, sondern es wird nur festgestellt, ob ein Eintrag in dem Baum vorhanden ist oder nicht.

Löschen:

Der zu löschende Eintrag wird gesucht und aus allen Buckets, in denen er vorkommt, gelöscht.
Beim Herausstreichen des zu löschenden Eintrags sind 2 Fälle möglich:

1. Entweder nach dem Abzählen in den vier Brüder-Buckets ensteht ein einzelnes Bucket (passiert in dem Falle, wenn nach dem Zusammensetzen aller davor in Buckets existierenden Einträgen die Kapazität von N nicht überschritten wird).

2. Oder es wird einfach ein Eintrag aus dem Bucket bzw. Buckets entfernt, in denen es sich befand.



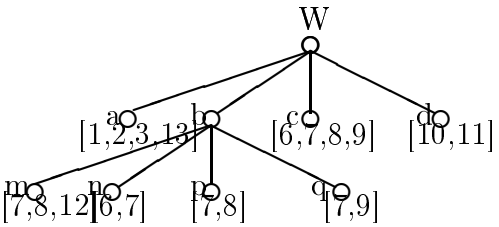


Abbildung 2: Quadtree nach dem Einfügen vom Element mbb 12

Im folgenden führen wir ein Beispiel mit der verbalen Beschreibung und graphischen Darstellung (unter Zuilfenahme von der Abbildung 2 , wobei als Ausganspunkt Abbildung 1 gilt) an. Dabei wird eine Bucket-Größe von 4 Einträgen unterstellt.

1. Das Einfügen vom Rechteck 13 führt zu keiner Aufteilung des Quadranten. Es wird einfach zum Blattquadranten a hinzugefügt.

2. Das Einfügen vom Rechteck 12 führt zum Aufteilen des Quadrants b in vier Subquadranten m,n,p,q. Dieses Rechteck wird zu denjenigen neu gebildeten Blätterquadranten hinzugefügt, die er jeweils überschneidet.

In unserem Falle wird es nur zum Subquadrant m hinzugefügt. Der Quadrant b wird in unserem Fall in Subquadranten aufgeteilt, weil die von uns am Anfang vereinbarte Seitekapazität von vier (Anzahl von Rechtecken, die im Quadranten enthalten sein müssen,also höchstens vier, um Quadrant nicht in Subquadranten aufzuteilen) überschritten wurde.

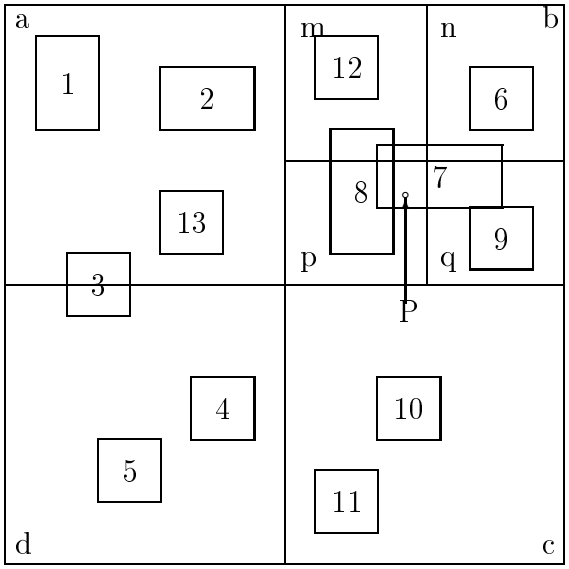
1.3 *Vor- und Nachteile*

Diese Variante vom Quadtree entspricht einigen Anforderungen von SAM(Spatial Access Methods) nicht, und zwar:

- Die Haupursache für den Mismatch ist Fan-Out(die maximale Anzahl von Söhne bei jedem Quadrant), das gleich 4 ist, und das zum Belegen von nur einem kleinen Teil der Festplattenseite führt. Deshalb lassen sich der Quadtree bzw. seine innere Knoten nicht so einfach effizient auf die Festplattenseiten abbilden.
- Die Baumstrukturen mit dem groï¿½n Fan-Out(solche wie B- oder R-Baum, die im Kaptitel 13 diskutiert werden) lassen sich effizienter auf die Festplattenseite abbilden und sind auch sehr gut für die Zugriffsmethoden auf den sekundären Speicher geeignet.
- Die Zeit, die für die Quadtree-Anfrage benötigt wird, ist vergleichbar mit der Baumtiefe, die ihrerseits so groß sein kann, da der Baum nicht balanciert ist.
- Eine Möglichkeit für die effiziente Abbildung der inneren Baumknoten auf Festpalltenseiten besteht nur dann, wenn die Sammlung statisch ist. Im dynamischen Falle verschlechtert sich die Perfomance rasant.
- Ausserdem leidet der Quadtree genauso wie ***Gridfiles***(siehe Kapitel 8) unter der Duplizierung von Objekten in verschiedenen Blättern.

1.4 *Rüeckblick: Punktanfrage, Fensteranfrage, z-Ordering*

An dieser Stelle möchte ich noch einen kleinen Rückblick auf die ***Punktanfrage*** machen (siehe Kapitel 10 “Quadtree für Punkte“), ganz kurz die



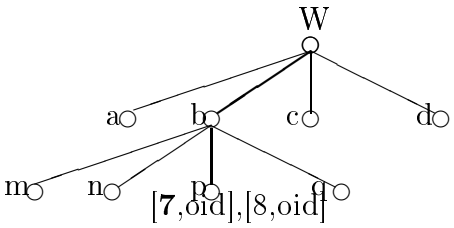


Abbildung 3: Punktanfrage

Fensteranfrage und die Raumfüllungskurve (in unserem Falle wird nur die Z-ordering-Kurve benötigt) beschreiben , da im letzten Teil dieses Kapitels und zwar bei dem linearen Quadtree dieses Wissen zum weiteren Verstehen benötigt wird.

Punktanfrage:

Gegeben sei ein Punkt P durch Koordinaten (x,y).
Gesucht sei ein Objekt o mit P $\in$ mbb(o).

Punktanfrage lässt sich sehr leicht mit dem Qaudtree realisieren. Es wird ein Pfad von der Wurzel bis zu dem Blatt des Baumes verfolgt. Dabei wird auf jeder Ebene aus den vier möglichen Quadranten derjenige gewählt, der das Punktargument enthält. Das Blatt wird genauso gescannt und gelesen wie beim ***Grid*** (siehe Kapitel 8 "Festgitter und Gridfile"). Auf dem Bild 1.4 ist die Punktanfrage zusammen mit dem von der Wurzel bis zum Blatt des Baumes verfolgten Pfad und dem Ergebnis (Objekt 7) dargestellt.

Fensteranfrage:

Gegeben sei ein Anfragefenster (Rechteck) F Gesucht werden alle Objekte, deren mbb mit F einen nichtleeren Durchschnitt haben.
Für die ***Fensteranfrage*** müssen also alle Blattquadranten weiter untersucht werden, die sich das Fensterargument (Anfragerechteck) schneiden. Die dort gespeicherte Datensätze werden sequentiell auf einen nichtleeren Schnitt ihres mbb mit F untersucht.

Eine ***Flächenfüllende Kurve*** legt eine totale Ordnung(eine sequentielle Anordnung) der Zellen von einem 2D-Gitter fest. Diese Ordnung ist sehr nützlich, wenn dadurch die Nähe der Zellen erhaltenbleibt. Das bedeutet, dass zwei Zellen, die in der Ebene benachbart sind, sich auch nah in der totalen Ordnung befinden werden. Selbst wenn es nicht immer der Fall ist, führen einige Kurven zu einer vernüftigen Annäherung dieser "Nähe-Eigenschaft". Die Ordnung soll unabhängig von einer möglichen Verfeinerung des Gitters stabil bleiben.

An dieser Stelle möchte ich eine der bekanntesten Raumfüllungskurven darstellen - die Z-Oder-Kurve (auch Z-Ordering-Kurve genannt). Zur Beschreibung der Z-Oder-Kurve benötigen wir ein Gitter, das darunter liegen soll. Dieses Gitter ist ein Array von N×N Zellen, wobei $N = 2^d$ ist. Es kann als ein kompletter Quadtree mit der Tiefe gleich d angesehen werden.

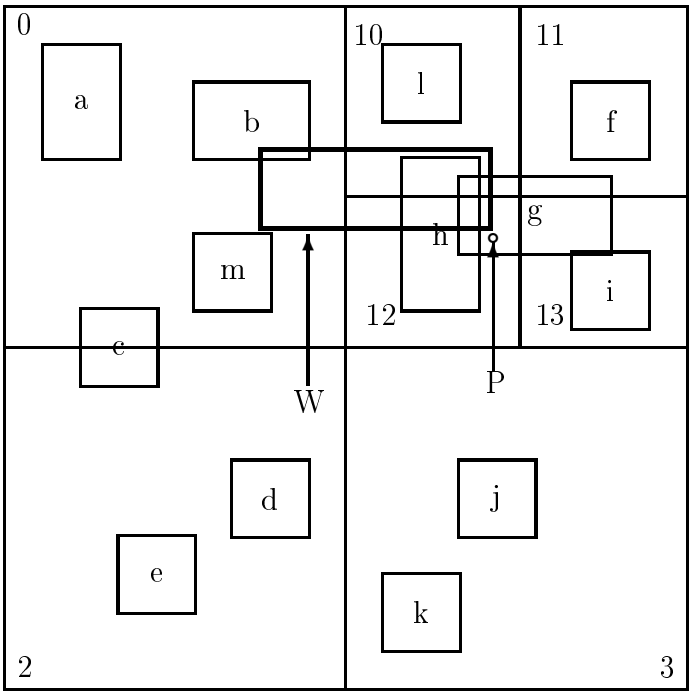
Z-Order:

Ein Label ist assoziert mit jedem Knoten des kompletten Quadtree und wird dabei als ein String über dem Alphabet (0, 1, 2, 3) gebildet. Das Label für die Wurzel ist ein leeres String. Die Söhne NW (NO, SW, SO) von dem internen Knoten k werden als k.0 (k.1, k.2, k.3) notiert, wobei "." als die String-Konkatenation bezeichnet wird.

Wenn die Zellen mit der Stringlänge d gelabelt sind, dann können wir sie gemäï¿½ihrer Labels in der lexikographischen Ordnung sortieren. Nehmen wir als Beispiel die Tiefe d=3 und sortieren die Labels in der aufsteigende Reihenfolge. Nach der Sortierung befindet sich die Zelle 212 vor der Zelle 300 und nach der Zelle 21. Diese z-artige Ordnung (siehe unten die Abbildung 4) hat zum Name Z-Order beigetragen.

Beim Quadtree-Labeling gibt es einige allgemeine Aussagen:

- Die Ordnung von Blättern entspricht dem Scannen von Blättern von links nach rechts.
- Die Labels haben verschiedene Gröï¿½n. Also wenn ein Blatt B eine Gitterzelle Z enthält, dann ist das Label b von B ein Präfix von dem Label z von Z und wir haben b $<$ z, wo das Zeichen "$<$" die aufsteigende lexikographische Ordnung auf den Strings bedeutet. Zum Beispiel $3 < 31 < 312 < 32$.
- Die Größe des Labels (bei der Darstellung im Dezimalsystem) ist gleich der Tiefe des Blattes in diesem Baum, dem dieses Label zugeordnet wurde.
- Das Label von einem Blatt kann als der Pfad von der Wurzel zu diesem Blatt angesehen werden.



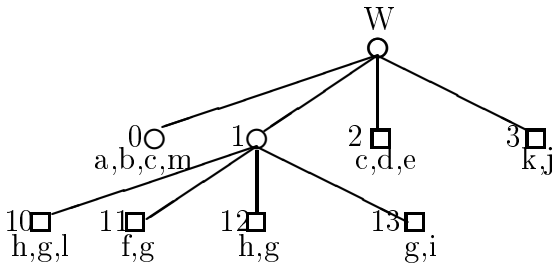


Abbildung 4: Z-Order Beispiel

2 *Linearer Quadtree*

2.1 *Linearer Quadtree mit dem B***-Baum*

Wenn ein Eintrag [mbb,oid] einem Quadtree-Blatt mit dem Label b zugewiesen und in einer Seite mit Adresse p gespeichert wurde, dann wird in einem B*-Baum die Sammlung von Paaren (b,p) indexiert, die aufs Label b verweisen. (siehe Abbildung 5)

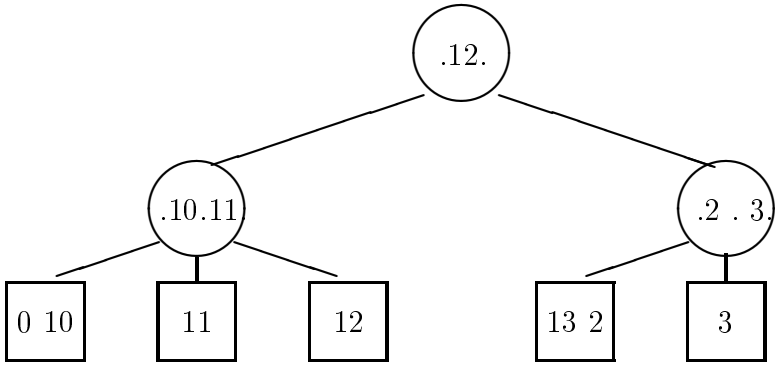


Abbildung 5: (2,3)B*-Baum

Eine solche Struktur führt zu einer guten Packung von Quadtree-Labels in den Bläettern vom B*-Baum. Sie ist vorhanden sowohl während des Einfügens als auch während des Löschens von Objekten in die Kollektion (z. B. sind die Einträge, die den Blättern mit den Lables 0 und 10 entsprechen, in dem selben Knoten). Solches Schema hat das gleiche Redundanzproblem wie die früher präsentierte Variante des Quadtree. D.h. Knoten, die verschiedene Qaudtree-Quadranten überlappen, sind in den Seiten duplieziert, die diesen Blättern zugewiesen sind (z. B. g ist 4 Mal in der gleichen Seiten gespeichert). Im folgenden schauen wir uns lineare Strukturen an, die Duplizierung von Knoten vermeiden.

2.2 *Punktanfrage mit dem linearen Quadtree*

Um die Identifikatoren von Objekten zu bekommen, deren Knoten(mbb) einen Punkt P enthält, benötigt man folgende 3 Schritte:

1. *Punktlabel berechnen.*

   Berechne das Label b (ein String der Länge d) von der Gitterzelle, die den Punkt P enthält. Das wird mit Hilfe der Methode *Punktlabel* des unten angegebenen Algorithmus bewerkstelligt (siehe LQ-PunktAnfrage).

2. *Retrieval vom Quadtree-Blatt in dem B*-Baum.*

   Sei B das Label vom Quadtree-Blatt, das P enthält. Wenn B=b, dann befindet sich das Qaudtree-Blatt in der Tiefe d. Wenn aber B ein Präfix von b ist, dann ist seine Länge kleiner als d und der Blattquadrant enthält die Gitterzelle, die P enthält. Genaugenommen entspricht dieser Quadrant dem Eintrag in dem B*- Baum, dessen Schlüssel der gröï¿½e Wert ist, der kleiner als oder gleich b ist. Die Suche nach solchen Einträgen ist keine Basisfunktion von B*- Bäumen. Diese Funktion wird in unserem Algorithmus MaxInf(b) genannt (siehe LQ-PunktAnfrage).

3. *Zugriff aufs Blatt und Scannen vom Blatt*

   Wenn der Eintrag im B*- Baum gefunden wurde, dann muss auf die Seite mit der Adresse p zugegriffen, alle Paare [mbb,oid] in dieser Seite gescannt und anschließend geprüft werden, welcher der Knoten(mbb) den Punkt P enthält. Das Ergebnis ist die Liste von Objektidentifikatoren.

Das oben beschrieben Algorithmus ist auf dem oben stehenden Bild 1.4 dargestellt.

- *Schritt 1:*

  Aus den Koordinaten vom Punkt P findet man, dass die Gitterzelle, die P enthält, das Label 120 hat.

- *Schritt 2:*

  Mit Hilfe von der Funktion MaxInf stellt man schnell fest, dass das maximale Label, das kleiner als 120, das Label 12 ist.

- *Schritt 3:*

  In diesem Schritt bleibt es nur noch auf die Seite zuzugreifen , zu scannen und die Knoten aufs Enthalten vom Punkt P zu prüfen.

LQ-PunktAnfrageAlogrithmus:

Quelle: Buch “The design and analysis of spatial data structures"

```
LQ-PunktAnfrage(P:punkt):set(oid)
    begin
    Ergebnis=0
//Schritt 1: berechne das Label vom Punkt
     b = PunktLabel(P)
//Schritt 2: der Eintrag [B,p] wurde beim Traversieren vom B*- Baum mit
//dem Schlüssel b bekommen
     [L, p]=MaxInf(b)
//Schritt3: bekomme die seite und gebe die Objekten zurück
    Seite = LeseSeite(p)
    for each e in Seite do
     if (e.mbb contains P) then Ergebnis+=e.oid
    end for
    return Ergebnis
    end
```

2.3 *FensterAnfrage mit dem linearen Quadtree*

Der Gedanke ist das Interval I von Labels zu berechnen, das alle Qaudranten abdeckt, die das Fenster W schneidet. Dann stellt man eine Bereichsanfrage auf dem B*- Baum in dem Interval I.

Der Berechnungsvorgang wird in 3 Schritten abgewickelt, die im folgenden beschrieben werden:

1. Berechne das Label b vom NW-Fenstervertex. Das geschieht genauso wie in dem Schritt 1 vom PunktAnfrageAlgorithmus. Dann berechne MaxInf(b) als im Schritt 2 von diesem Algorithmus. Das ergibt dann

[B,p], wobei das Label B die unterste Grenze vom Interval ist. Als Nächstes berechnen wir das Label b' vom SO-Fenstervertex mit MaxInf. Das führt zu [B',p'], wobei B' ist die oberste Grenze.

2. Stelle eine Bereichsanfrage im Interval [B,B'] auf dem B* -Baum, welche dann alle Einträge [b,p] zurückgibt, deren Label b in diesem Interval liegt.

3. Für jeden B*- Baumeintrag e=[b,p] berechne den Quadranten, der als e.b markiert ist, mit der Funktion Quadrant(e.b). Wenn der Quadrant(e,b) das Fenster überschneidet, dann greife auf die Quadrantenseite e.p zu und teste jeden von Quadtree-Einträgen auf die ï¿½erlappung mit W.

Sehen wir jetzt das Bild 1.4 auf dem das Anfragefenster mit dem Argument W abgebildet ist.

1. *Schritt 1:*

   Das Label vom NW-Fenstervertex von W ist 012 und das Label von SO-Fenstervertex ist 121. Dann suchen wir im B* - Baum Bidl 11.5 mit der Funktion MaxInf das gröï¿½e Quadtreelabel, das kleiner als 012 und 121 ist. Wir bekommen MaxInf(012)=01 und MaxInf(121)=12.

2. *Schritt 2:*

   In diesem Schritt schreiben wir eine Bereichsanfrage auf dem B*- Baum (sihe Bild 1.4) in dem Interval [01,12]

3. *Schritt 3:*

   Für jeden Eintrag, der während des Scannens gefunden wurde, greifen wir auf die Seite nur dann zu, wenn W den Quadranten(e.b) überlappt.

LQ-FensterAnfrageAlogrithmus:

Quelle: Buch “The design and analysis of spatial data structures“

```
LQ-FensterAnfrage(W:rectangle):set(oid)
begin
    Ergebnis=0
//Schritt 1: Aus den Fenstervertexen W.nw und W.se berechnet man
//           das Interval [B,B']. Dazu wird die 2-malige Suche im B*- Baum
//           benötigt.
    b =PunktLabel(W.nw);[B,p]=MaxInf(b);
    b'=PunktLabel(W.se);[B',p']=MaxInf(b');
//Schritt 2: Man berechnet die Menge Q von B*- Baumeinträgen [b,p]
           in b ∈ [B,B']
    Q=BereichAnfrage([B,B'])
//Schritt 3: Man greift auf jeden Eintrag in der Menge Q zu, dessen Quadrant
//           W überlappt.
      for each q in Q do
        if (Quadrant(q.b) overlaps W) then
          Setie = LeseSeite(q.p)
// Es wird die Quadtree-Seite gescannt
           for each e in Seite do
             if (e.mbb overlaps W) then Ergebnis+=(e.oid)
           end for
         end if
       end for
//Es werden Ergebnisse sortiert und Duplikate eliminiert
    Sort(Ergebnis);RemoveDuplikate(Ergebnis);
     return Ergebnis
end
```

Literatur: Samet Buch “The design and analysis of spatial data structures“