R-Baum und seine Spezialisierungen: R*- und R+-Baum

Joana Bendoraityte
mai00jgx@studserv.uni-leipzig.de

19. März 2004

Zusammenfassung

Die Arbeit ist im Rahmen der Vorlesung "Geoinformationssysteme 2" im Wintersemester 2003/2004 an der Universität Leipzig (Abteilung Datenbanken) angefertigt.

Der Beleg beschreibt den R-Baum und zwei seiner Spezialisierungen oder Versionen: R*- und R+-Baum. Es werden die Struktur, die Algorithmen der Suche, des Einfügens, des Löschens und des Knotensplittens betrachtet. Jeder Algorithmus ist sowohl wörtlich beschrieben als auch in einer halbformaler Beschreibung angegeben. Nach der Erläuterung jedes Algorithms folgt ein Beispiel.

Inhaltsverzeichnis

1 Einführung

Der R-Baum geht auf A. Guttman (1984) zurück. Dabei werden achsparallele Rechtecke zur Beschreibung der Objekte für den räumlichen Zugriff verwendet (statt der räumlichen Objekte selbst). Der R-Baum ist für mehrdimensionale Räume und Phänomene räumlicher Ausdehnung geeignet. Es ist ein höhenbalanzierter Baum ähnlich einem Binärbaum, dessen Indexeinträge in den Blättern Zeiger zu den realen Objekten beinhalten. Die Knoten korrespondieren zu den physikalischen Plattenpages. Er erfordert keine periodische Reorganisation. Zur Suche nach räumlichen Objekten ist nur eine kleine Anzahl von Knoten zu analysieren - im Gegensatz zu hierarchischen Baumstrukturen kann allerdings die parallele Suche in verschiedenen Ästen notwendig sein.

R-Baum wird angewendet, in:

- Computer Aided Design (CAD)
- Geografischen Informations Sytemen(GIS)
- Computer-Vision und Robotics
- Kartographie

Die wichtigen Modifikationen des R-Baums sind R*- und R+-Bäume. Im Folgenden werden die Struktur, die Eigenschaften und Algorithmen dieser drei Indexstrukturen vorgestellt.

In diesem Artikel sind alle Beispiele zweidimensional. Das ist nur deswegen, um die Erklärungen primitiv aber deutlich darzustellen. R-Bäume können je nach Bedarf in den Räumen mit mehr als zwei Dimensionen angewandt werden.

2 R-Baum (R-Tree)

2.1 Struktur

Der R-Baum ist ein perfekt balancierter Baum ähnlich dem B-Baum. Und wie bei diesem soll jeder Knoten (außer der Wurzel) einen Mindest-Füllgrad aufweisen und die Blätter sollen alle auf der selben Höhe liegen.

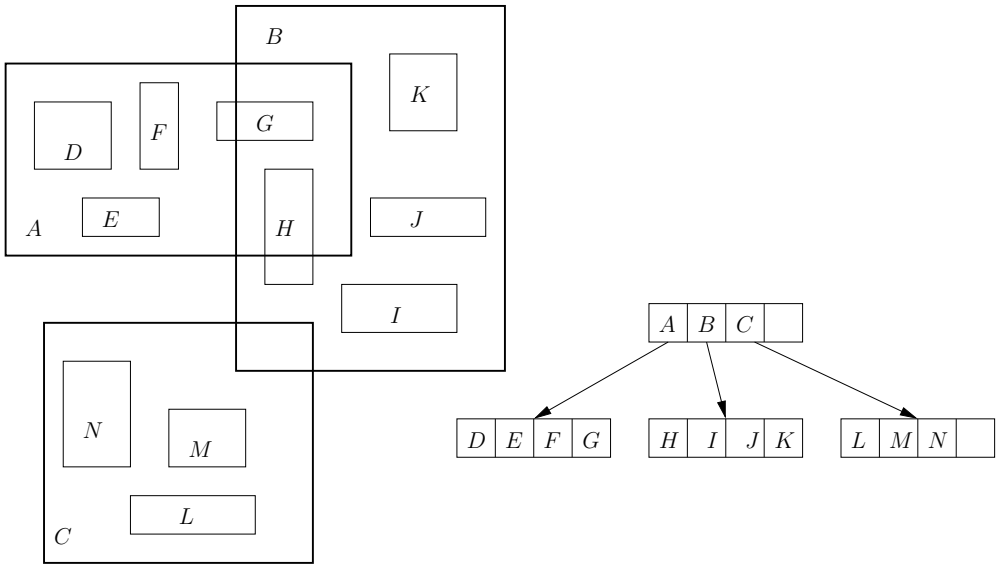

Abbildung 1: R-Baum

Vorgelegt sei ein Objekt O in einer Ebene mit einem rechtwinkligen kartesischen Koordinatensystem. Ein achsenparalleles Rechteck R heißt zu O umbeschrieben, wenn O $\subseteq R$ gilt. Ein zu O umbeschriebenes Rechteck R' heißt kleinstes umbeschriebenes Rechteck von O, wenn gilt $R' \subseteq R$ für alle zu O umbeschriebenen Rechtecke R. (Betrachtet man Objekte in Räumen höher Dimension k, spricht man von umbeschriebenen k-dimensionalen Quadern) Wir verwenden unabhängig von der Dimension die vom Englischen "minimal bounding box" abgeleitete Abkürzung MBB. Die MBB sind selbst wieder k-dimensionale Objekte. Werden einige zu einer neuen Gesamtheit zusammengefasst, kann man dieser Zusammenfassung wieder einer MBB zuordnen. Diese Struktur setzt sich so bis zur Wurzel fort. Schließlich beeinhaltet die Wurzel ein MBB über alle Objekte.

Im R-Baum gibt es zwei Knotentypen: Blattknoten und innerer Knoten. Ein Eintrag des Blattes ist ein Paar aus einem MBB und einem Bezeichner oder Verweis auf das Datenobjekt (z.B. eine ID-Nummer): (MBB, oid). Ein Eintrag des inneren Knotens ist ein Paar (MBB, nodeid). Nodeid ist ein Verweis, der auf einen Sohnknoten zeigt. Die Struktur muss folgende Bedingungen erfüllen:

- Pro Knoten (außer der Wurzel) maximal M, wenigstens m Einträge, wobei m $\in$ [0, M/2];
- Wenn die Wurzel kein Blatt ist, hat sie mindestens 2 Söhne;

- Jeder Eintrag in einem inneren Knoten hat als MBB den kleinsten umbeschriebenen Quader, der alle MBB des betreffenden Sohnes enthält.
- Jeder Eintrag in einem Blatt hat als MBB einen Verweis auf das Datenobjekt;
- Alle Blätter befinden sich auf der gleichen Ebene.

Die Abbildung 1 bietet ein Beispiel eines einfachen R-Baums an.

Die maximale Höhe des R-Baumes, der N Einträge hat, ist $\log_m N - 1$. Und die minimale Höhe ist $\log_M N - 1$.

Weiter werden die Algorithmen zum Suchen (Search), Einfügen (Insert) und Löschen (Delete) beschrieben. In den Einfügen- und Löschenalgorithmen wird der ***SplitNode*** Algorithmus verwendet, der später im Kapitel 2.5 erklärt wird.

2.2 Suchen

Die Suche ist ähnlich der in einem B-Baum. Zu unterscheiden sind Gebiet- und Punktsuche. Die Vorgehensweise ist in beiden Fällen gleichartig. Deswegen erkläre ich nur die Gebietsuche ausführlich und die Punktsuche kurz.

Gebietsuche. Annahme: zu suchen ist das Gebiet S. Beginnend bei der Wurzel werden alle Einträge des aktuellen Knotens auf einen Schnitt mit S untersucht. Bei allen Einträgen, bei denen $S \cap$ MBB $\neq \varnothing$, wird der Sohnknoten rekursiv durchsucht. Ist man an einem Blattknoten angekommen, so werden alle Einträge betrachtet. Im Falle $S \cap$ MBB $\neq \varnothing$ werden, über die in den Blätttern enthaltenen Zeiger zu den Datenobjekten, die gefundenen Datenobjekte als Ergebnis der Suche zurückgegeben.

Halbformale Beschreibung *Search*. **T** ist der Wurzelknoten des R-Baums. S ist die zu findende MBB. Es sind alle Einträge, deren MBB sich mit S überschneiden, zu finden.

```
procedure Search (T) {
  if ( T ist kein Blatt )
  then {
```

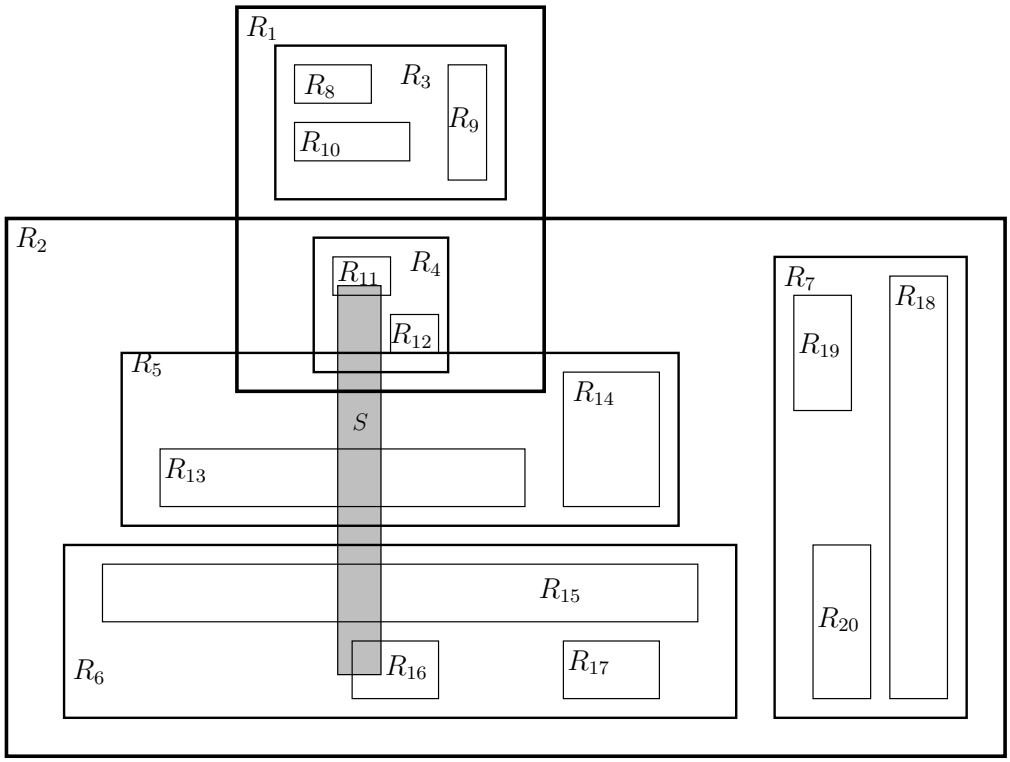

Abbildung 2: Suche

```
    for all Einträge (MBB, nodeid) von T do {
      if ( S∩MBB ≠ ∅ )
      then Search (nodeid);
    }
  }
  else {
    for all Einträge (MBB, oid) von T do {
      if ( S∩MBB ≠ ∅ )
      then das Datenobjekt, auf das oid verweist, gehört zur Ergebnismenge;
    }
  }
}
```

Punktsuche. Annahme: zu suchen ist der Punkt p. Beginnend bei der Wurzel wird untersucht, für welche Einträge $p \in$ MBB gilt. Die Einträge, die Bedingung erfüllen, werden rekursiv weiter analysiert. Da die MBB im Baum auch übereinander liegen können, besteht durchaus die Möglichkeit,

daß mehrere Söhne untersucht werden müssen.Die Suche endet auf der Blattebene. Die MBB, die sich in einem Blatt befinden und $p \in$ MBB, gehören zum Ergebnis.

Beispiel des Suchenalgorithmus (Gebietsuche). In der Abbildung 2 ist das Rechteck S das Sucheziel, wobei der Baum mit M = 4 und m = 2 ist.

Der Algorithmus sucht nach der Rechtecken, die sich mit S überschneiden. In der Abbildung 3 ist der vom Algorithmus gewählte Ablaufweg gezeigt.

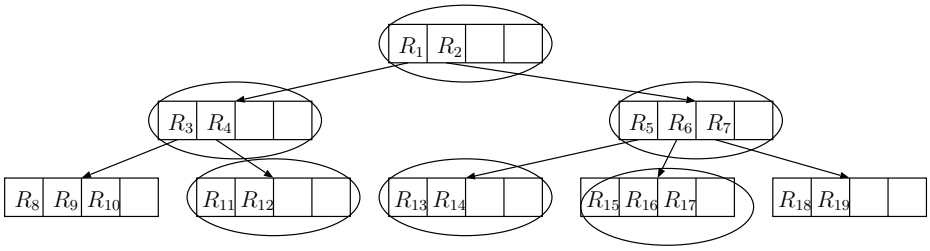

Abbildung 3: Suche

Da sich das Rechteck S mit den Wurzeleinträgen R_1 und R_2 überschneidet, untersucht der Algorithmus beide Einträge weiter. In R_1 gibt es ein Rechteck R_4, das sich mit S deckt. Die Einträge des Rechtecks R_4 werden jetzt überprüft. So erreicht der Algorithmus den Blattknoten, der nach geeigneten Einträgen untersucht wird. R_{11} ist der einzige passende Kandidat zur Ergebnismenge. In R_2 gibt es zwei Rechtecke, die sich mit S schneiden: R_5 und R_6. Die weitere Untersuchung wird vom Algorithmus in R_5 und R_6 geführt. Dies liefert die Einträge R_{13} bzw. R_{15} und R_{16}. Somit bilden R_{11}, R_{13}, R_{15} und R_{16} das Ergebnis der Suche.

2.3 Einfügen

Da die gesamte Dateninformation beim R-Baum in den Blättern gespeichert ist, muss ein geeignetes Blatt zuerst gefunden werden, in das das Objekt eingefügt werden soll. Um dieses zu finden, wird beginnend bei der Wurzel immer der Unterbaum desjenigen Eintrags, dessen MBB durch das Einfügen den geringsten Flächenzuwachs bekommt, weiterverfolgt. Ist dieser bei zwei MBB gleich, so entscheidet man sich für jene mit der geringeren Fläche, anschließend für die mit den wenigeren Einträgen. Hier sieht man, daß die Struktur eines R-Baumes nicht deterministisch ist, denn sie hängt von der

Reihenfolge der eigefügten Daten ab. Ist das Blatt gefunden, kann der Eintrag eingefügt werden. Die Intervallgrenzen des Blattes und die aller Vaterknoten müssen nun noch angepasst werden.

Halbformale Beschreibung *Insert*. Der neue Eintrag E = (MBB_E, oid) soll in den R-Baum eingefügt werden.

```
procedure Insert (E) {
  Wähle mit der Funktion ChooseLeaf das Blatt B,
  wohin E eingetragen wird;

  if (B hat ausreichend Platz)
  then Füge E hinzu;
  else {
    Führe die Funktion SplitNode(B) (siehe 2.5) aus, um 2 Knoten
    (B und BB) zu erhalten. Der Eintrag E und alle Einträge aus B
    werden jetzt auf B und BB aufgeteilt;

    if (B ist Wurzel)
    then Erstelle die neue Wurzel, deren Söhne B und BB sind;
  }
  AdjustTree (B);
}
```

Halbformale Beschreibung *ChooseLeaf*. Diese Funktion selektiert das Blatt, in das der neue Eintrag zugefügt wird.

```
procedure ChooseLeaf () {
  K := Wurzel;
  while ( K ist kein Blatt ) {
    K := Sohnknoten, dessen MBB die kleinste Vergrößerung
    benötigt, um MBB_E aufzunehmen;
    (Wenn das nicht eindeutig ist, wähle MBB mit der kleinsten
    Fläche)
  }
  return K;
}
```

Halbformale Beschreibung *AdjustTree*. Es werden von einem Blatt **B** aufsteigend zur Wurzel die MBB angepasst und Knotenteilungen vollzogen, wenn dies nötig ist.

```
procedure AdjustTree (B) {
  K := B;
  while ( K ist kein Wurzel ) {
    V := Vaterknoten von K;
    E_K = (MBB_K, K_ID) := Eintrag in V, der auf K zeigt;
    Passe MBB_K an, so daß es alle MBB in K eng einschließt;
    if ( K hat einen Partner KK, der aus dem früheren SplitNode
    entstanden ist )
    then {
      E_KK = (MBB_KK, KK_ID) := Eintrag in V, der auf KK zeigt;
      Passe MBB_KK an, so daß es alle MBB in KK eng einschließt;
      if (V hat mehr als M Einträge)
      then {
        Führe die Funktion SplitNode(V) aus, um 2 Knoten
        (V und VV) zu erhalten. Der Eintrag E_KK und alle Einträge aus V
        werden jetzt auf V und VV aufgeteilt;
        if (V ist Wurzel)
        then Erstelle die neue Wurzel, deren Söhne V und VV sind;
      }
    }
    if ( es wurden keine Änderungen beim Anpassen und kein
    SplitNode(V) vorgenommen)
    then return;
    else K := V;
  }
}
```

Beispiel Das Rechteck R_{21} wird eingefügt (Abb. 4). (R-Baum: M=3, m=1.)

Um die beste Position für das neue Rechteck zu finden, wird zuerst ***ChooseLeaf*** aufgerufen. Der Ablaufsweg dieser Routine ist in der Abbildung 5 veranschaulicht.

In der Wurzel ist der Eintrag R_1 unter unseren Interessen, da R_{21} in R_1 liegt. Weiter wählt ***ChooseLeaf*** R_3, da R_3 die geringere Vergrößerung als R_4 verlangt. So kommt die Routine zum Blattknoten. Da er voll ist, muss der Knoten wie in der Abbildung 6 geteilt werden, um R_{21} hinzufügen dürfen.

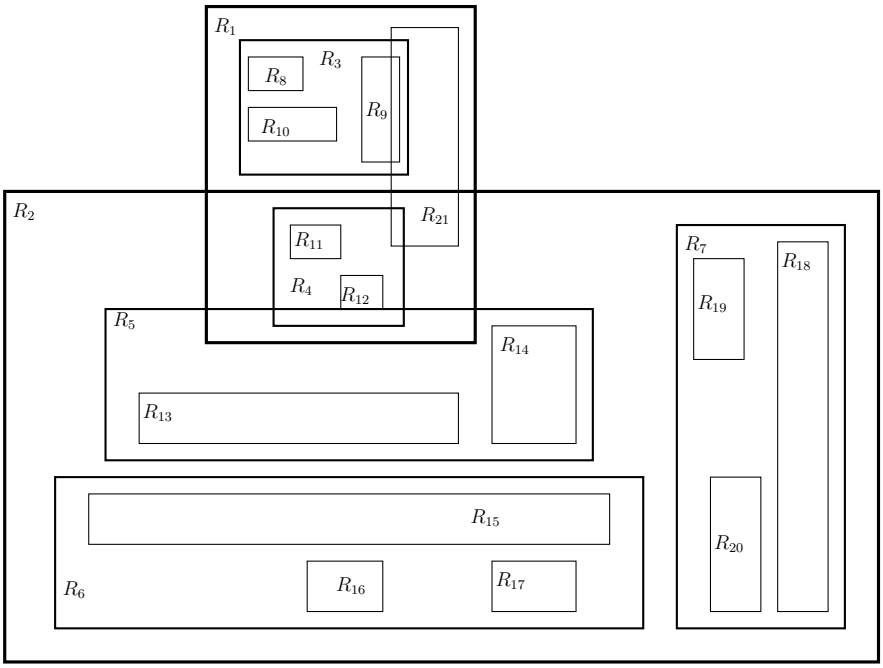

Abbildung 4: Einfügen

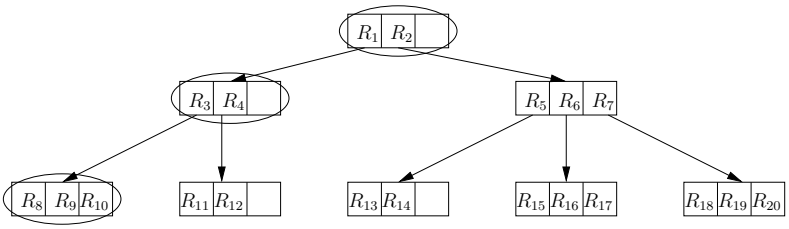

Abbildung 5: Ablaufsweg von *ChooseLeaf*

SplitNode versucht die Rechtecke möglichst klein machen: R_{10} und R_8 werden ins Rechteck R_3 gelegt; R_9 und R_{21} entsprechend in ein neues Rechteck R_3' im Vaterknoten. R_3 wird der Funktion ***AdjustTree***, die aufwärts weitergeht, übermitellt. Der aktuelle Knoten hat genug Platz, um R_3' hinzuzufügen. Deswegen ist kein Splitten da nötig. Schließlich kommt man zur Wurzel, und der Algorithmus endet. Das Einfügen ist nun fertig, die Abbildung 7 zeigt das Ergebnis.

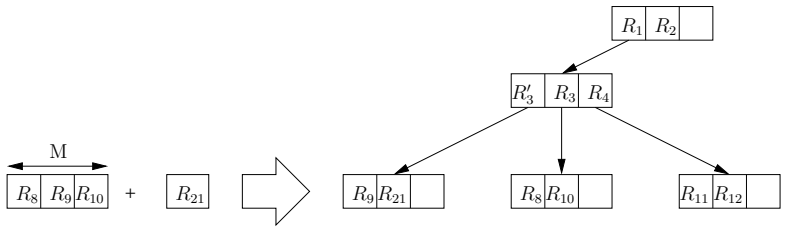

Abbildung 6: Knotensplitten

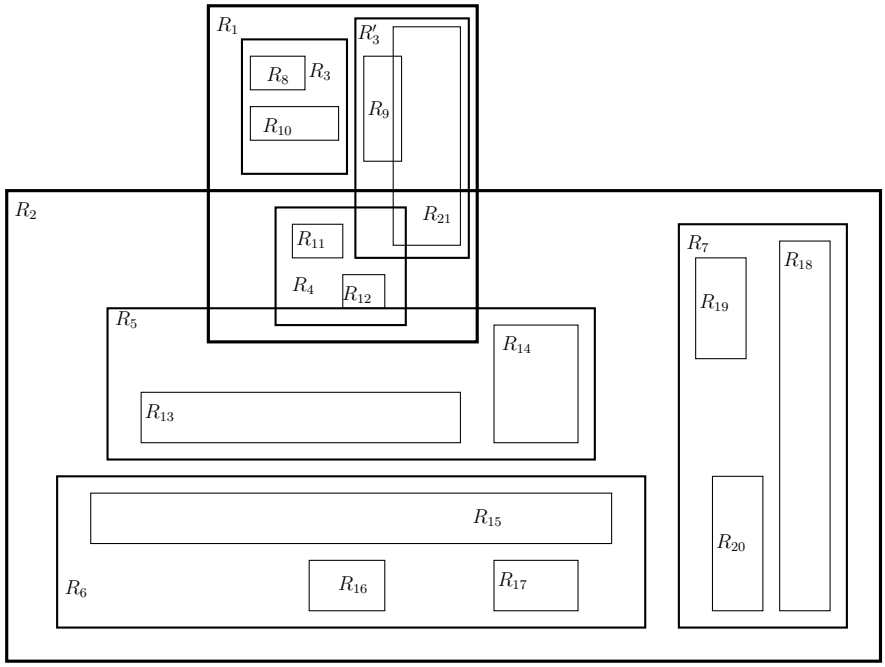

Abbildung 7: nach Einfügen

2.4 Löschen

Um ein Objekt zu löschen, muss es zuerst gefunden werden. Anschließend wird es einfach entfernt. Falls das Blatt, aus dem der Eintrag gelöscht wurde, immer noch m Einträge enthält, so müssen nur die Intervallgrenzen des Blattes und aller darüber liegenden Knoten angepasst werden.

Besaß das Blatt allerdings vor dem Löschen bereits nur m Einträge, so ist nach dem Löschen die minimale Anzahl an Einträgen unterschritten (underflow); der Knoten muss aufgelöst und bestehende Einträge auf andere Knoten verteilt werden. Dies geschieht folgendermaßen: alle Einträge aus dem unterbesetzten Knoten werden aus dem Baum entfernt und zwischengespeichert.

Da dieser Knoten nun nicht mehr existiert, kann für den Eintrag im Vaterknoten die Gefahr bestehen, daß der Vaterknoten weniger als m Einträge besitzt. Der ganze Vorgang wiederholt sich möglicherweise. Wie auch schon beim Einfügen, kann dies bis zur Wurzel hin fortsetzen. Sobald alle MBB angepasst wurden, werden die zwischengespeicherten Knoten mit dem schon bekannten Einfügenalgorithmus wieder im Baum plaziert. Dabei ist allerdings darauf zu achten, daß Knoten aus einer höheren Ebene auch wieder höher im Baum eingepasst werden, damit der Baum höhenbalanciert bleibt. Als letzter Schritt ist noch die Wurzel zu betrachten: besitzt die Wurzel nur noch einen Eintrag, so wird sie komplett entfernt und der Sohn wird zur neuen Wurzel. Dies ist die einzige Möglichkeit für einen R-Baum, an Höhe zu verlieren.

Halbformale Beschreibung *Delete* Der Eintrag E wird gelöscht.

```
procedure Delete (E) {
  B := null;
  B := FindLeaf (Wurzel);
  if ( B == null )
  then Breche ab;
  Entferne E aus B;
  CondenseTree (B);
}
```

Halbformale Beschreibung *FindLeaf*.

```
E := der zu löschende Eintrag;
K := Wurzel;
procedure FindLeaf (K) {
  if ( K ist kein Blatt )
  then {
    for all Einträge (MBB, nodeid) in K do {
      if ( MBB schließt K ein )
      then FindLeaf (K) ;
    }
  } else {
    for all Einträge (MBB, oid) in K do {
      if ( (MBB, oid) == E )
      then return (K);
```

```
  }
}
```

Halbformale Beschreibung *CondenseTree*.

```
Q := leere Liste, in die gelöschte Knoten eingefügt werden;
procedure CondenseTree (K) {
  while ( K ist kein Wurzel ) {
    V := Vaterknoten von K;
    E_V := (MBB_K, K_ID); /* Eintrag im V, der auf K verweist */
    if ( K hat weniger als m Einträge )
    then {
      Lösche E_V;
      Füge K in Q ein;
    } else {
      Passe MBB_K an, sodaß MBB_K alle Einträge von K eng einschließt;
    }
    K := V;
  }
  for all Knoten in Q do {
    if ( Knoten ist Blatt )
    then {
      for all Einträge im Knoten do
        Insert (Eintrag);
    } else {
      for all Einträge im Knoten do
        Füge den Eintrag in den Baum ein, sodaß sich die Einträge aus
        höherer Level auch höher im Baum befinden;
    }
  }
}
```

Beispiel Um ein nichttriviales Beispiel für das Löschen zu geben, müssen die Eigenschaften des Baumes aus dem vorigen Kapitel (2.3) verändert werden. Wir setzen M = 4 und m = 2 (Abb. 8).

Die Struktur des Beispieldatensatzes ändert sich dadurch nicht. Nun soll das Rechteck R_{12} gelöscht werden. Der Algorithmus muss R_{12} erst finden. Dann muss R_{12} aus R_4 gelöscht werden. Auf R_4 wird ***CondenseTree*** aufgerufen. Da R_4 weniger als m Einträge hat, wird es zur leer initialisiertem

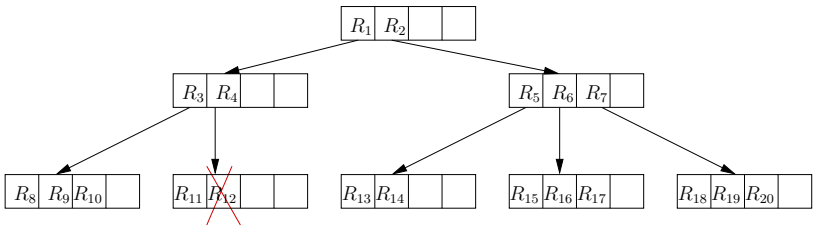

Abbildung 8: Veränderter Beispieldatensatz vor dem Löschen von R_{12}.

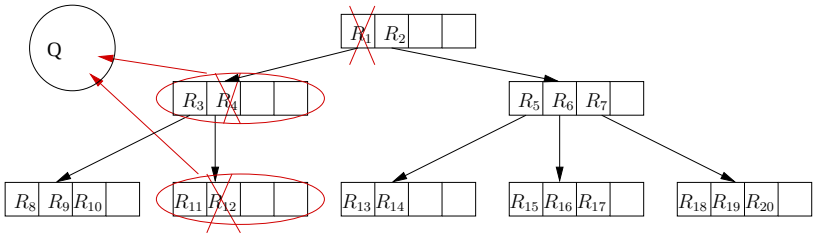

Abbildung 9: Speichern von Knoten, deren Referenzen gelöscht wurden.

Menge Q hinzugefügt und sein Eintrag in R_1 gelöscht. R_1 wird überprüft und, nachdem es zu Q hinzugefügt wurde, auch gelöscht (Abb. 9). Als nächstes wäre die Wurzel an der Reihe, diese bleibt aber unverändert. Jetzt werden zunächst die Einträge, die nicht aus Blättern stammen, wieder eingefügt, und zwar in der korrekten Baumhöhe, so daß ihre Blätter auf dem allgemeinen Blattniveau stehen. Im Beispiel ist das nur der Eintrag R_3. Dann kommen noch die Blatteinträge an die Reihe, hier R_{11} (Abb 10). Danach geht ***CondenseTree*** zu Ende. Die Wurzel hat nur noch einen Eintrag R_1, der zur neuen Wurzel gemacht wird. Die Abbildung 11 zeigt, wie der Baum nach dem Löschen von R_{12} aussieht.

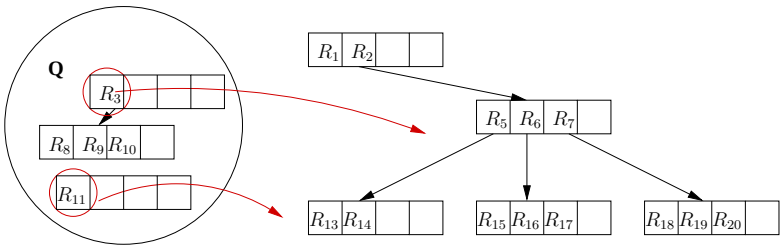

Abbildung 10: Wiedereinfügen gespeicherter Einträge.

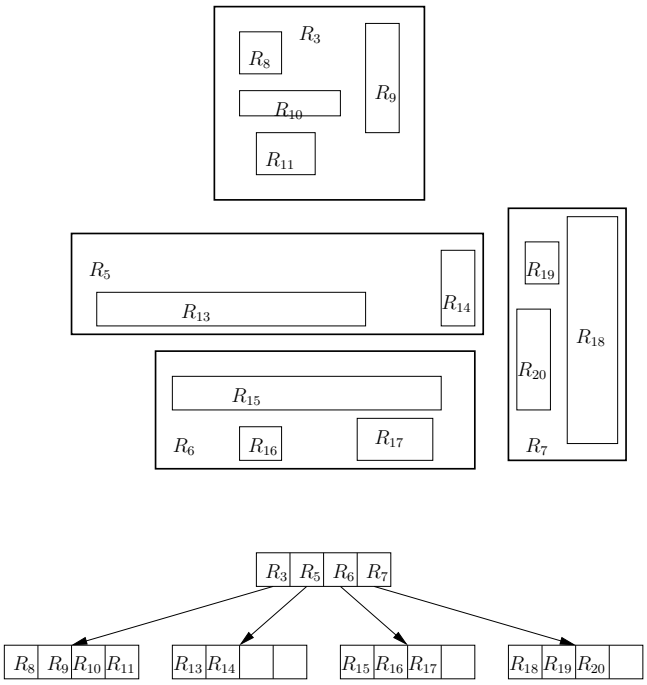

Abbildung 11: Der Baum nach dem Löschen von R_{12}.

2.5 Splitten des Knotens

Beim Einfügen und Löschen wurde oft die Methode ***SplitNode*** angewendet aber nicht erklärt. Die Funktion ist notwendig, wenn ein Knoten überlaufen ist. Die Aufgabe des Splittens ist, von M auf M+1 angewachsenen Einträge auf zwei Knoten zu verteilen. Wie schon beim Einfügen oder Löschen wird auch beim Splitten versucht, die MBB möglichst klein zu halten. Auf diese Weise wird die Wahrscheinlichkeit minimiert, dass beide Knoten bei einer Suche geprüft werden müssen.

Zwei Beispiele vom ***SplitNode*** -Algoritmus zeigt die Abbildung 12. Die linke Seite präsentiert ein schlechtes Splitten, weil seine äußerste MBB viel Platz benötigt und die Rechtecke R_1 und R_2 nicht möglichst klein sind. Im Gegensatz dazu steht das Teilen rechts.

Guttman gibt drei verschiedene Versionen des Algorithmus an, die sich in ihrer Laufzeit und Qualität der Ergebnisse unterscheiden.

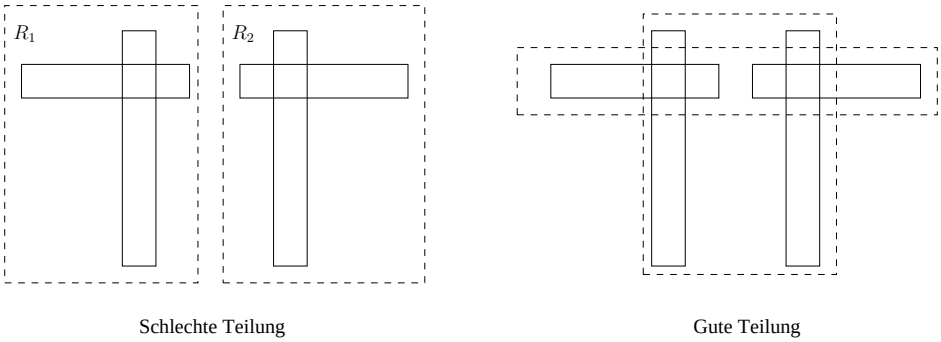

Abbildung 12: Splitten eines Knotens

2.5.1 Exhaustive

Ein naiver Ansatz, um die optimale Gruppierung zu finden, ist, alle Gruppierungen zu generieren und die beste auszuwählen. Die Anzahl der Gruppierungen ist 2^{M-1} und bei einer vernünftigen Wahl von M sehr groß. Deshalb ist dieser Algorithmus für eine nutzbare Implementierung eines R-Baums zu langsam. Die Kosten sind exponentiell.

2.5.2 Quadratic-Cost

Als ersten Schritt weist man jedem der beiden neuen Knoten einen Eintrag zu. Diese beiden Einträge werden nach folgendem Verfahren ausgewählt: Es wird für alle möglichen Paare die Fläche berechnet, die diese aufspannen, also die MBB, die sie erzeugen würden, befänden sie sich im selben Knoten. Die beiden Objekte, für welche die größte Fläche berechnet wurde, werden als erste Einträge den beiden neuen Knoten zugewiesen.

Für die Aufteilung der restlichen Objekte gibt es wiederum ein Verfahren: Als erstes muss dasjenige Objekt ausfindig gemacht werden, das als nächstes zugewiesen werden soll. Dafür wird für jedes noch verbleibende Objekt der Flächenzuwachs berechnet, den dieses durch Hinzufügen zu jedem der beiden MBB verursachen würde. Am Ende wird das Objekt als nächster Eintrag ausgewählt, bei dem die Differenz der beiden Flächenzuwachse am größten ist. Es wird dem Knoten hinzugefügt, dessen MBB am geringsten vergrößert werden muss. Dies wird so lange fortgeführt, bis alle Datenobjekte aufgeteilt sind oder ein Knoten so wenige Einträge besitzt, daß er alle verbleibenden Objekte benötigt, um die minimale Anzahl m zu erreichen. Die Laufzeit ist quadratisch für M und linear für die Anzahl der Dimensionen.

Halbformale Beschreibung *QuadraticSplit*. Die Methode unterteilt M+1 Einträge des überfüllten Knotens in zwei Gruppen, die jeweils Einträge des neuen und alten Knotens sind.

```
procedure QuadraticSplit ( Knoten, der M+1 Einträge besitzt ) {
  (E1, E2) := PickSeeds( Knoten, der M+1 Einträge besitzt );
  Füge E1 in eine Gruppe und E2 in die andere hinzu;
  while ( nicht alle Einträge sind zugewiesen ) do {
    if (eine der beiden Gruppen hat so wenig Einträge, daß sie alle
      restliche benötigt, um insgesamt m zu haben )
    then Füge alle restliche in die Gruppe zu;
    else {
      E := PickNext( Einträge, die noch nicht zugewiesen sind);
      Vergrößerung1 := Flächezuwachs der Gruppe1 nach dem Einfügen
      von E;
      Vergrößerung2 := Flächezuwachs der Gruppe2 nach dem Einfügen
      von E;
      if ( Vergrößerung1 ≠ Vergrößerung2 )
      then Füge E in die Gruppe zu, deren Vergrößerung am kleinsten ist;
      else if ( Fläche der Gruppe1 ≠ Fläche der Gruppe2 )
        Füge E in die Gruppe zu, deren Fläche am kleinsten ist;
      else Füge E in die Gruppe zu, die am wenigsten Einträge hat.
    }
  }
}
```

Halbformale Beschreibung *PickSeeds*. Die Methode wählt zwei Einträge, die die ersten in je einer Gruppe sein sollen.

```
procedure PickSeeds ( Knoten, der M+1 Einträge besitzt ) {
  D := 0;  /* der größte Abstand */
  for all ( Paare (Ei, Ej), wobei Ei, Ej aus dem überfüllten Knoten sind;
      i ≠ j; i, j ∈ [1, M + 1] ) do {
    Bilde MBB J, die MBBi und MBBj umfasst;
    d := Fläche(J) - Fläche(MBBi) - Fläche(MBBj);
    if ( d > D )
    then D := d;
  }
  return das Paar (Ei, Ej), dessen d = D;
}
```

Halbformale Beschreibung *PickNext*. Die Funktion berechnet für alle verbliebenen Einträge den Flächenzuwachs und den nächten Eintrag, der in eine der beiden Gruppen zugewiesen wird.

```
procedure PickNext ( Einträge, die noch nicht zugewiesen sind ) {
  for all ( Einträge ) do {
    d1 := Flächevergrößerung, die nach der Aufnahme von MBB in die
    Gruppe 1 entsteht;
    d2 := Flächevergrößerung, die nach der Aufnahme von MBB in die
    Gruppe 2 entsteht;
  }
  return E, dessen | d1 − d2 | am größten ist;
}
```

2.5.3 Linear-Cost

Dieser Algorithmus sucht in jeder Dimension nach den beiden Objekten mit dem größten Abstand voneinander. Am Ende wird von all diesen Paaren das mit dem letzendlich größten Abstand gewählt und die beiden Objekte auf verschiedene Knoten verteilt. Im Gegensatz zum Quadratic-Cost Algorithmus gibt es beim Linear-Cost Algorithmus kein Verfahren mit dem der nächste hinzufügende Eintrag gewählt wird, die Wahl erfolgt willkürlich. Da die etwas schlechtere Qualität dieses Algorithmus sich nur geringfügig auf die Suchleistung auswirkt, wird er den anderen beiden meist vorgezogen. Die Kosten sind linear in M und in Dimensionen.

Halbformale Beschreibung *Linear-Cost*.

```
procedure Linear-Cost ( Knoten, M+1 Einträge besitzt ) {
  Suche nach zwei Einträgen, deren MBB in jeder Dimension mit dem
  größten Abstand voneinander entfernt sind;
  Füge einen in die Gruppe1 und anderen in die Gruppe 2;
  Verteile die restlichen nicht zugewiesenen Einträge beliebig (willkürlich);
}
```

3 R*-Baum

R*-Baum ist eine kombinierte Optimierungsmethode von Bereich, Breite und Überlappung von jedem MBB in Knoten. Er ist eine Spezialisierung vom R-Baum bzw. ein erweiterter R-Baum.

Die einzigen Unterschiede zwischen dem R-Baum und dem R*-Baum sind nur bei Einfügen- und Splittenalgorithmen. Der Struktur von den beiden ist nicht zu unterscheiden.

Einziges Kriterium beim R-Baum für die Optimierung der Einfügeoperation:

- (O1) Minimiere leere Fläche zwischen Directory-MBB (MBB des Eintrags im Vaterknoten) und den darin enthaltenen MBB.

Beim R*-Baum gibt es vier Optimierungskriterien:

- (O1) Minimiere leere Fläche zwischen Directory-MBB und den darin enthaltenen MBB (wie beim R-Baum);
- (O2) Minimiere die Überlappungen von Directory-MBB;
- (O3) Minimiere den Rand der Directory-MBB;
- (O4) Optimiere die Speichernutzung.

(O1) und (O2) berwirken, daß beim Durchsuchen des Baumes früher entschieden werden kann, welcher Pfad zu nehmen ist.

(O4) bewirkt, daß der Baum nicht zu hoch wird, damit der Suchalgorithmus sehr schnell laufen kann.

Die Kriterien beeinflussen sich auch gegenseitig:

- Für (O1) und (O2) muss die Form der MBB flexibel gewählt werden können, dies wirkt (O3) entgegen.
- Beachtung von (O1) kann (O2) günstig beeinflussen etc.

3.1 Einfügen

Man ruft den ***ChooseLeaf*** (s. die Seite 20) Algorithmus auf, um einen geeigneten Knoten nach geringstem Flächen- und Überlappungswert zu finden. Die neue MBB wird eingef"gt. Wenn der Knoten mehr als M Einträge besitzt, ruf man ***OverflowTreatment***, um die Entfernung zwischen den Zentren seiner MBB und dem Zentrum der MBB, die den Knoten umfasst, berechnet. Man löscht den Eintrag mit der größten Entfernung, und passt die MBB des Knotens an. Der gelöschte Eintrag wird nach der in diesem Abschnitt beschriebenen Vorgehensweise wieder eingefügt. In einem Knoten darf man nicht 2 mal hintereinander den Algorithmus ***OverflowTreatment*** aufrufen. An Stelle dessen soll ***SplitNode*** ausgeführt werden. Dann passt man alle sich schneidende MBB im Einfügungspfad an. Falls es die Wurzel bewirkt, wird eine neue Wurzel erzeugt bzw. die Höhe des Baumes um eins erhöht.

Halbformale Beschreibung *Insert*. E ist einzufügen.

```
procedure Insert ( E ) {
  K := ChooseLeaf ();
  Füge E in K zu;
  if ( K hat M+1 Einträge )
  then OverflowTreatment ( K );
  if ( SplitNode( K wurde ausgeführt )
  then {
    OverflowTreatment ( Vaterknoten von K );
    if ( K ist Wurzel )
    then Erstelle eine neue Wurzel, die auf K und den nach dem
    Splitten entstandenen neuen Knoten zeigt;
  }
}
```

Halbformale Beschreibung *OverflowTreatment*.

```
procedure OverflowTreatment ( K ) {
  if ( K ist keine Wurzel &
    OverflowTreatment ist erstes Mal mit K aufgerufen )
  then Reinsert ( K );
  else SplitNode( K );
}
```

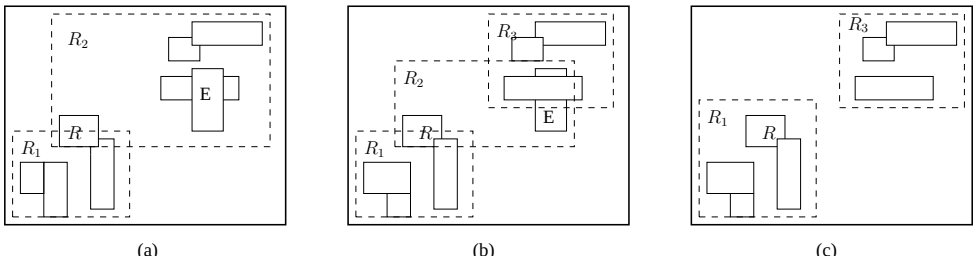

Abbildung 13: Wiedereinfügen-Strategie: (a) Einfügen von E; (b) Split wie im R-Baum; (c) Wiedereinfügen von R im R*-Baum

Halbformale Beschreibung *Reinsert*

```
procedure Reinsert ( K ) {
  for all Einträge von K do
    d := Entfernung zwischen dem Zentrum der MBB und dem der MBB,
    die K umfasst;
  Sortiere die Einträge absteigend nach d;
  Lösche die ersten p Einträge von K und passe das MBB von K an;
  for all gelöschte Einträge do
    Insert ( der gelöschte Eintrag );
}
```

Am effizientesten ist es, wenn p = 30% der Einträge von **K**.

In der Abbildung 13 ist die Strategie der Methode ***Reinsert*** im R*-Baum dargestellt. Das Datenrechteck E soll in den Baum (M=4, m=2) eingefügt werden. Die MBB R_2 wird überfüllt. Der ***SplitNode*** Algorithmus des R-Baums würde R_2 lokal reorganisieren (Abb. 13(b)). Der R*-Raum meidet, den Knoten sofort zu splitten. Da wird erstens ein Rechteck, das am weitesten vom Zentrum des R_2 entfernt ist, gefunden. In unseren Fall ist das R. ***Reinsert*** löscht R aus R_2 und fügt erneut in den Baum. R wird in R_1 aufgenommen (Abb. 13(c)). Jetzt kann E in R_2 hingefügt werden, ohne ***SplitNode*** zu verursachen.

ChooseLeaf Zuerst muss ein geeignetes Blatt gefunden werden, in das das Objekt eingefügt werden soll. Um dieses zu finden wird beginnend bei der Wurzel immer der Unterbaum desjenigen Eintrags weiterverfolg. Dieser Algortihmus wird dann in 2 Teile verzweigt. Wenn der Zeiger auf innere Knoten zeigt, wählt man den Eintrag, dessen Rechteck durch das Einfügen

den geringsten Flächenzuwachs erfährt. Wenn der Zeiger auf Blättern zeigt, wählt man den Eintrag, dessen MBB durch das Einfügen den geringsten Überlappungsvergrößerung braucht. Ist das Blatt gefunden, kann der Eintrag eingefügt werden.

Algorithmus ***SplitNode***(lese im nächsten Abschnitt) des R*-Baumes macht den Baum gleichmäßig verteilt und die Überlappungen werden berücksichtigt und kleinstmöglichst verhindert, damit die Suche schneller ausgeführt ist.

Halbformale Beschreibung *ChooseLeaf*.

```
procedure ChooseLeaf ( ){
  K := Wurzel;
  while ( K ist kein Blatt ) {
    if ( Sohnknoten von K ist ein Blatt )
    then {
      Wähle den Eintrag, dessen MBB die kleinste Überlappungs-
      vergrößerung benötigt, um die neue Daten-MBB aufzunehmen.
      if ( mehrere solche Einträge sind gefunden )
      then {
        Wähle den Eintrag, dessen MBB die kleinste Vergrößerung benötigt;
        if ( mehrere solche Einträge sind gefunden )
        then Wähle den den Eintrag, dessen MBB am kleinsten ist;
      }
    }
    else {
      Wähle den Eintrag, dessen MBB die kleinste Vergrößerung
      benötigt, um die neuen MBB einzufügen;
      if ( mehrere solche Einträge sind gefunden )
      then Wähle den Eintrag, dessen MBB am kleinsten ist
    }
    K := Sohnknoten, auf den nodeid (aus dem gewählten Eintrag) zeigt;
  }
  return K;
}
```

3.2 Splitten des Knotens

Wenn der Knoten mehr als M Einträge hat, werden die nach allen Koordinaten bzgl. unteren und oberen Wert ihrer MBB sortiert. Dann berechnet man

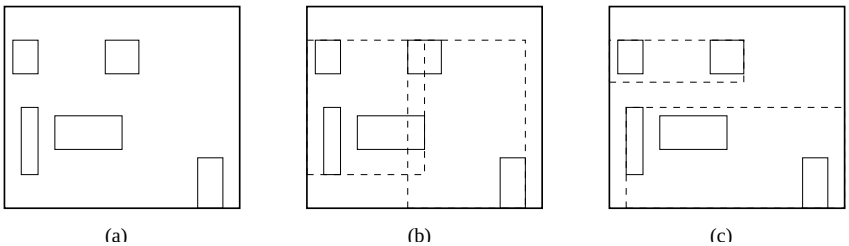

Abbildung 14: Splitten eines Knotens (M=4, m=2): (a) der überfüllte Knoten; (b) Splitten-Algorithmus eines R-Baumes; (c) Splitten-Algorithmus eines R*-Baumes

die Summe aller Umfangswerte der verschiedenen Verteilung von gewählter Achse. Man wählt die Achse mit der minimalen Summe als Splitachse. Entlang der gewählten Achse wählt man die Verteilung mit dem minimalen Überlappungswert. Zuletzt kann man direkt die Einträge nach der gewählten Verteilung auf zwei Gruppen verteilen. (Abb. 14)

Pseude-Code *SplitNode*.

```
procedure SplitNode ( K ) {
  A := ChooseSplitAxis ( K ); /* Achse */
  Bestimme alle Verteilungen entlang der A;
  Wähle die Verteilung mit dem minimalen Überlappungswert;
  if ( mehrere solche Verteilungen sind gefunden )
  then Wähle die Verteilung mit minimalem Flächenwert;
  Verteile die Einträge auf zwei Gruppen;
}
```

Halbformale Beschreibung *ChooseSplitAxis*.

```
procedure ChooseSplitAxis ( K ) {
  for all Dimensionen do {
    Sortiere die Einträge zunächst nach dem unteren, dann nach dem
    oberen Wert ihrer MBB;
    Bestimme alle Verteilungen;
    S := Summe aller Umfangswerte der verschiedenen Verteilungen;
  }
  return Achse mit dem kleinsten S;
}
```

Die Anzahl der Verteilungen ist M-2m+2 pro Sortierung.

4 R+-Baum

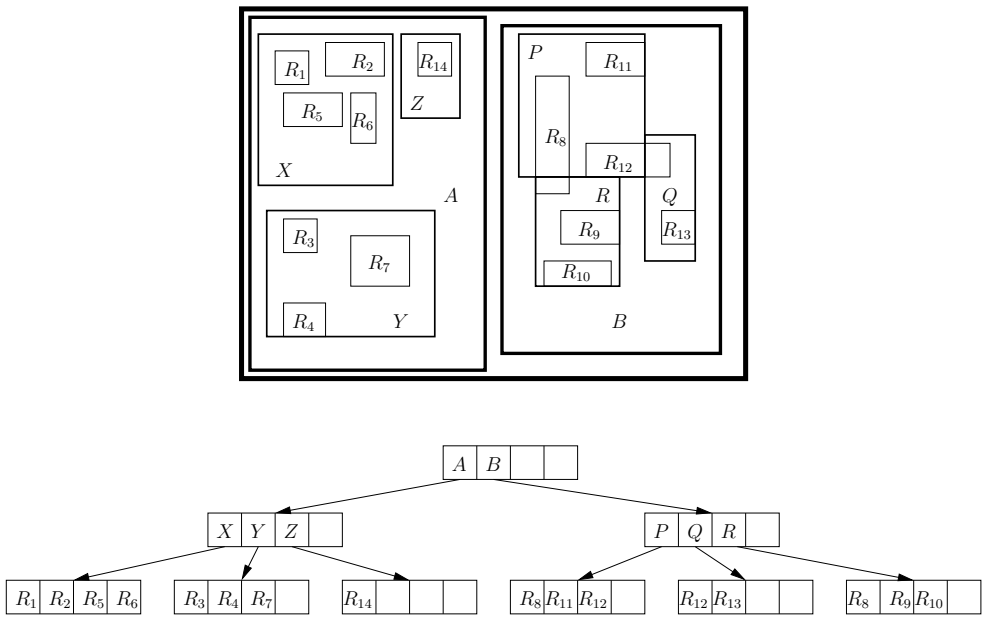

Abbildung 15: R+-Baum

4.1 Struktur

R+-Baum vermeidet überlappende Suchraum-Regionen in den Nicht-Blattknoten, wodurch der Suchraum disjunkt aufgeteilt wird. R+-Baum ist ein Vertreter des Clipping-Ansatzes (allerdings kein reines Clipping). Die Struktur dieses Baums ist wie folgt:

- Die Wurzel hat mindestens zwei Einträge, außer wenn sie ein Blattknoten ist.
- Die MBB der Knoten, die sich in einer gleichen Baumebene befinden, dürfen einander nicht überschneiden.
- Wenn Knoten N kein Blatt ist, umfasst sein Rechteck alle Rechtecke, die sich in einem Unterbaum mit der Wurzel N befinden.

- Die Objekte, die nicht vollständig in eine Region fallen, werden in allen Blattknoten, mit denen sie sich überschneiden, gespeichert. D.h. solche MBB werden mehrmals (redundant) gespeichert.

Die Struktur des R+-Baums ist wie die eines R-Baums, außer daß R+-Baum keine Überlappungen zwischen Rechtecken in der gleichen Baumebene zulässt. Ein Beispiel des R+-Baums zeigt die Abbildung 15.

4.2 Algorithmen

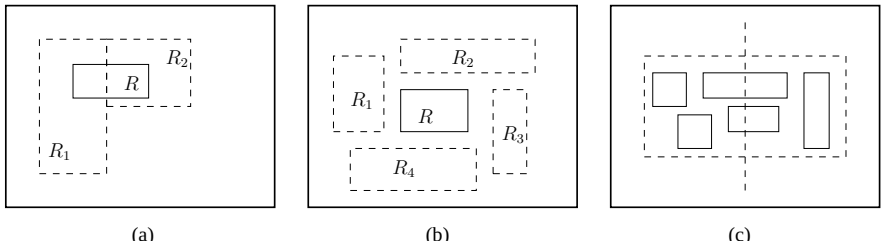

Abbildung 16: Einfügen von R: (a) Clipping in Teilrechtecke; (b) Deadlock; (c) abwärts Splitten

Die Algorithmen sind denen eines R-Baumes ähnlich.

Suche Da R+-Baum keine Überlappungen in inneren Knoten aufweist, ist die Suche viel schneller als im R-Baum. Vor allem bei Punktanfragen kommt es bis zu $> 50\%$ Zugriffsersparnis.

Einfügen Eine Daten-MBB E wird in alle Suchräume bzw. korrespondierende Blattknoten, mit denen sich eine Überschneidung ergibt, aufgenommen. Um keine Überlappungen zwischen MBB der Knoten auf der gleichen Ebene zu schaffen, verwendet der R+-Baum Clipping. Wenn sich E mit einigen MBB $\{mbb_1, mbb_2, ..., mbb_N\}$, die jeweils einen Blattknoten umfassen, überschneiden, wird E entlang der MBB (logisch) "zerschnitten". Dann wird E in jedes Baltt, dessen MBB$\in \{mbb_1, mbb_2, ..., mbb_N\}$, eingefügt. (Abb. 16(a))

Wenn E nicht vollständig von einem MBB bzw. einigen MBB überdeckt werden kann, muss man eine oder mehrere MBB vergrößern. Dies funktioniert ähnlich ***AdjustTree*** (siehe S. 8) im R-Baum. Zusätzlich muss man

aufpassen, daß keine sich schneidende MBB in einer Baumebene entstehen. Es gibt Fälle, wenn die Vergrößerung der MBB nicht möglicht ist, da sich eine Überschneidung dann nicht vermeiden liese. Die Abbildung 16(b) zeigt ein Beispiel: wenn R_1, R_2, R_3 oder R_4 vergrößert wird, bekommen wir die Überlappung der MBB. Einige MBB sollen anhand der Methode ***Reinsert*** neu organisiert werden.

Das Splitten ist manchmal die einzige Lösung der Knotenüberfüllung. Das Splitten eines Knotens ist wieder ähnlich dem des R-Baumes. Das Splitten setzt sich abwärts fort, um die Überschneidungen von MBB auf der gleichen Ebene zu meiden. Z.B.: wenn A Vater von B ist und B Vater von C, müssen diese dann ebenfalls gesplittet werden. Deswegen ist der Algorithmus des Splittens viel komplizierter (Abb. 16(c)).

Löschen Der Baum wird durchsucht, in welchem Blatt sich das zu entfernende Objekt befindet. Dann wird es aus dem Baum entfernt. Es gibt keine minimale Anzahl m.

Literatur

[1] Sebastian Käbisch. *R-Tree*. Proseminar: Algorithms and Datastructures for Database Systems, University of Passau, 2003

[2] Phillipe Rigaux, Michel Scholl, Agnes Voisard. *Spatial Databases with Applications to GIS*. Morgan Kaufmann Publishers, 2002

[3] Nobert Bartelme. *Geoinformatik*. Berlin, Heidelberg, 1995

[4] Ralf Bill. *Hardware, Software und Daten*. Karlsruhe, 3. Aufl., 1997

[5] *Einführung R-Baum*, 2003
www-i9.informatik.rwth-aachen.de/Lehre/PRAKTIKUM-SS03/Material/02%20R-Baum.pdf

[6] Gabriele Wilke-Müller. *R-Baum eine dynamische Index-Struktur für räumliche Suche*, 2003
www.inf.uni-konstanz.de/dbis/teaching/ws0304/datatypes/download/slides-r-tree.ppt

[7] Jemmy Halim. *R-Baum & R*-Baum*, 2003
www.informatik.uni-bonn.de/~tb/Lehre/ws02/psIdx/Folien/Halim.ppt

[8] Tilmann Rabl. *R-Baume*, 2001
www.db.fmi.uni-passau.de/lehre/WS01-02/proinhalt.phtml

[9] E. Rahm. *Implementierung vonDatenbanksystemen 1*, 2002
dbs.uni-leipzig.de/en/skripte/IDBS1/HTML/kap7-13.html