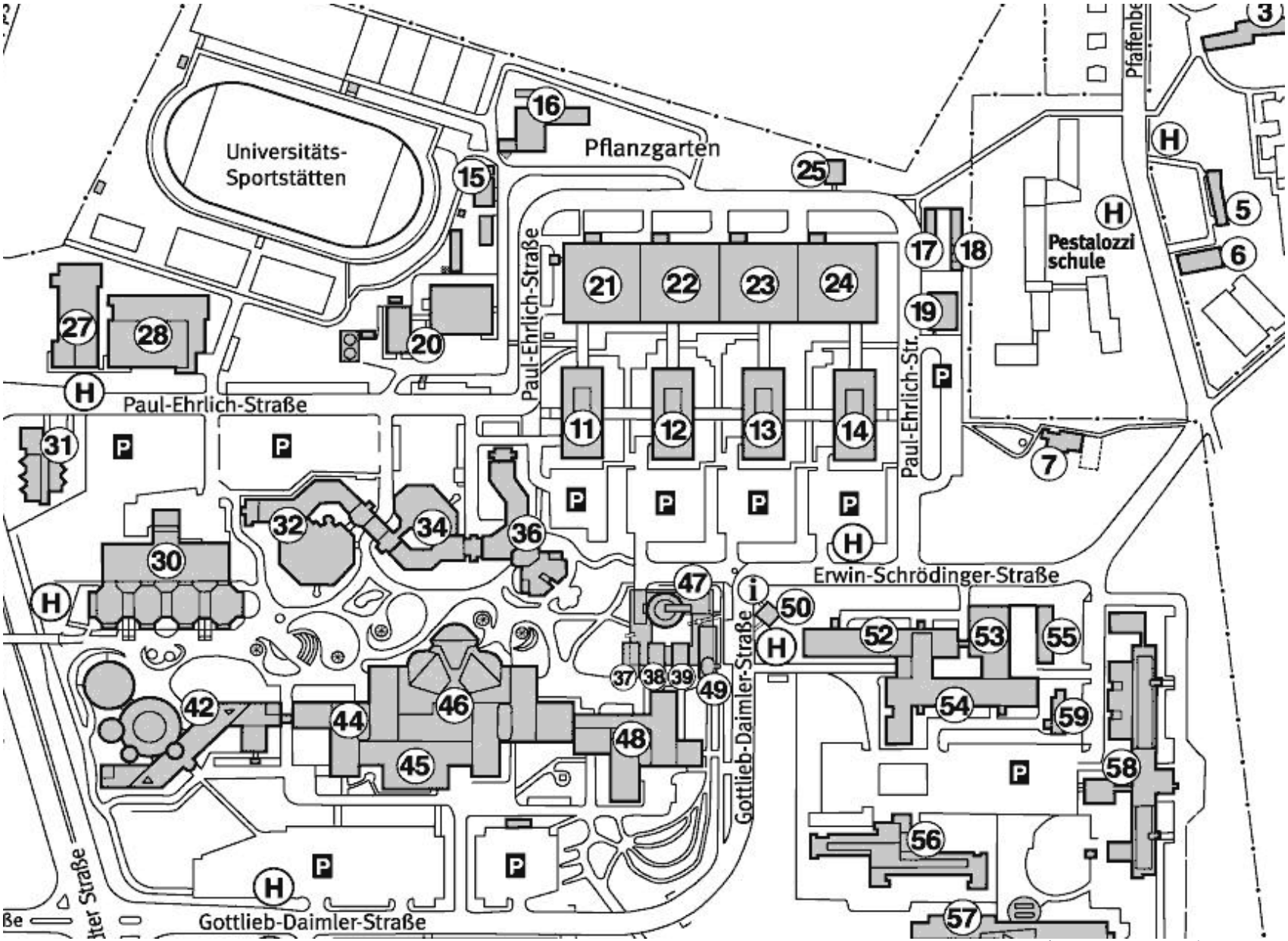
Datenbanksysteme

Wintersemester 2016/17

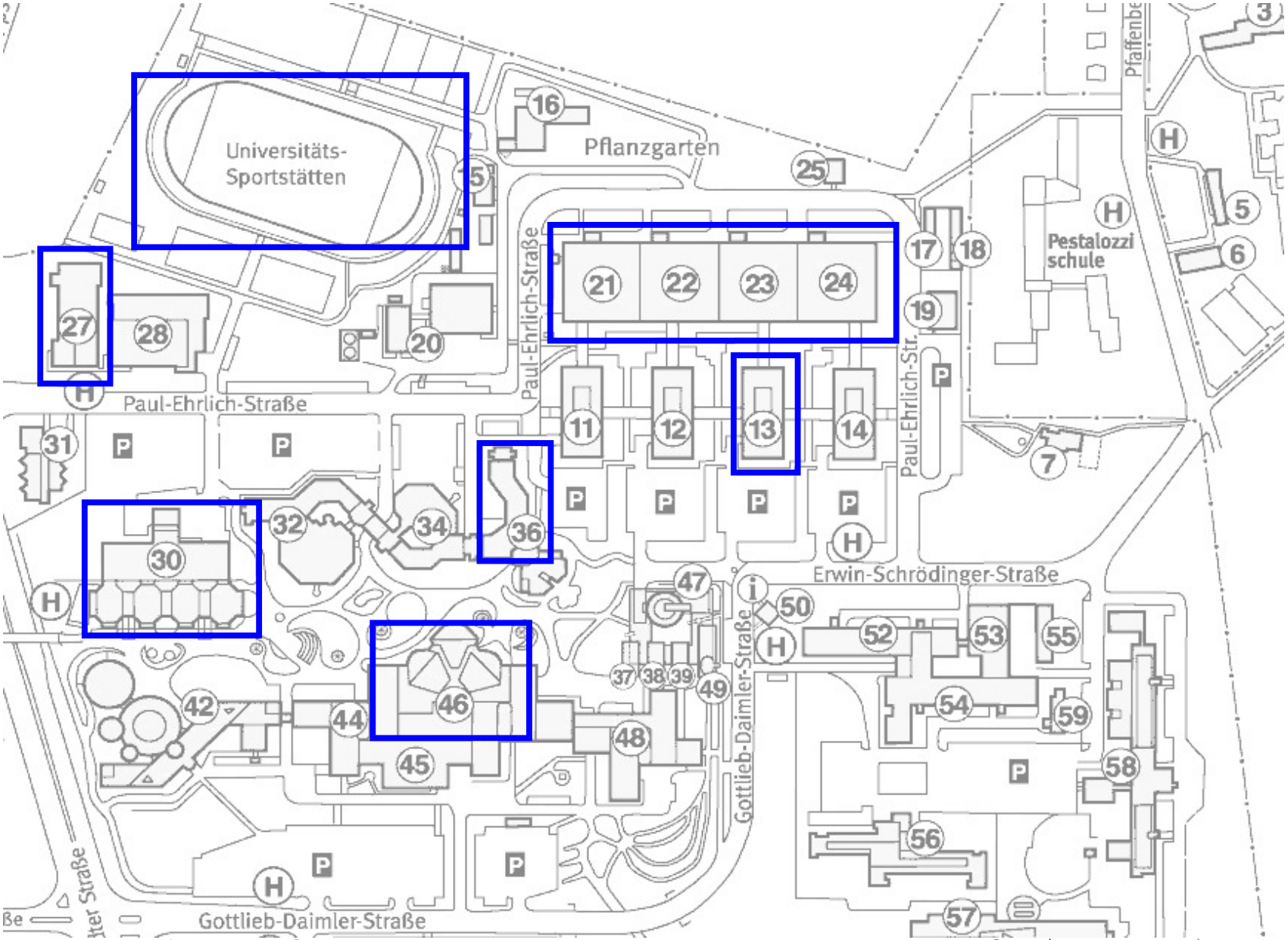
Prof. Dr.-Ing. Sebastian Michel
TU Kaiserslautern

smichel@cs.uni-kl.de

Segmente und Rechtecke



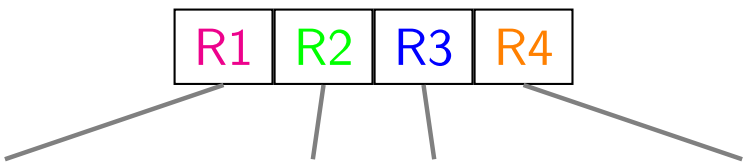
Segmente und Rechtecke

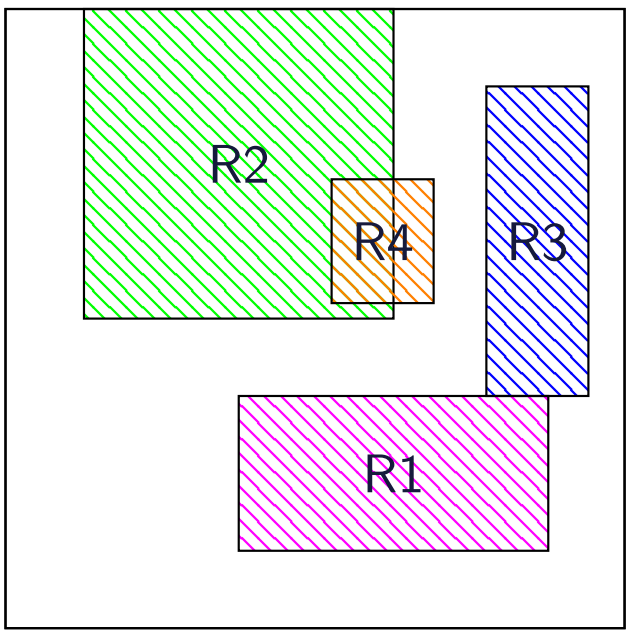


R-Baum - Idee

MBR - Minimum Bounding Rectangle (Minimal Umschließendes Rechteck)

- Jeder Eintrag eines Knotens verweist auf einen Teilbereich des Datenraums
- Beschrieben durch MBRs, die die eingeschlossenen Objekte (Rechtecke) minimal umschreiben

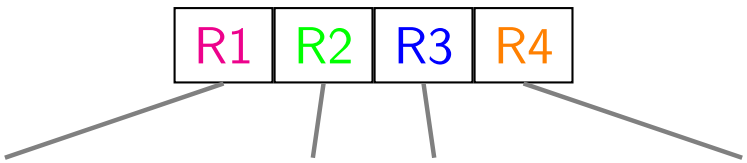


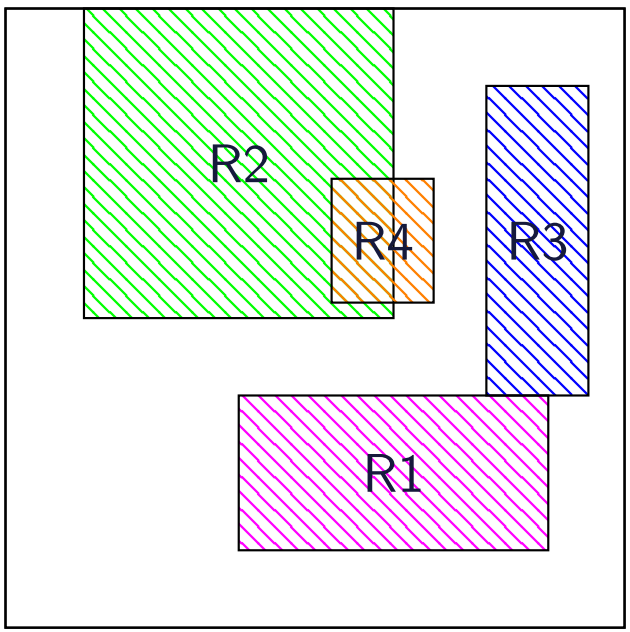


R-Baum - Idee (2)

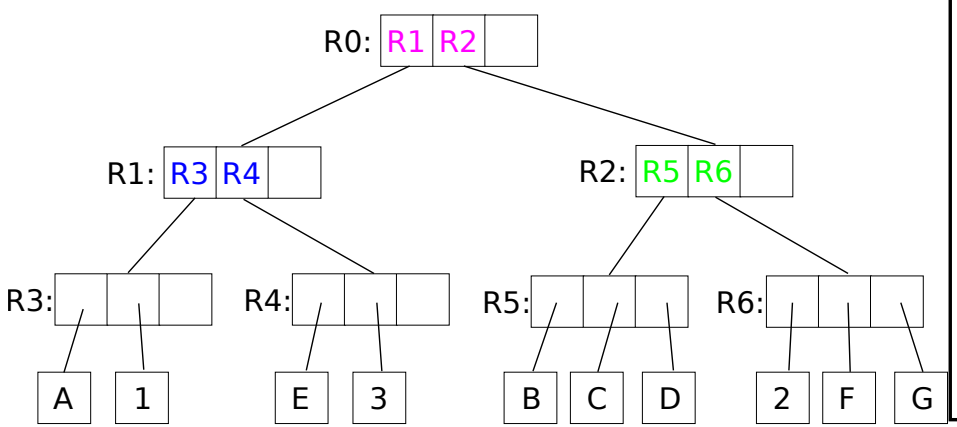
Wegweiserfunktion und überlappende MBRs

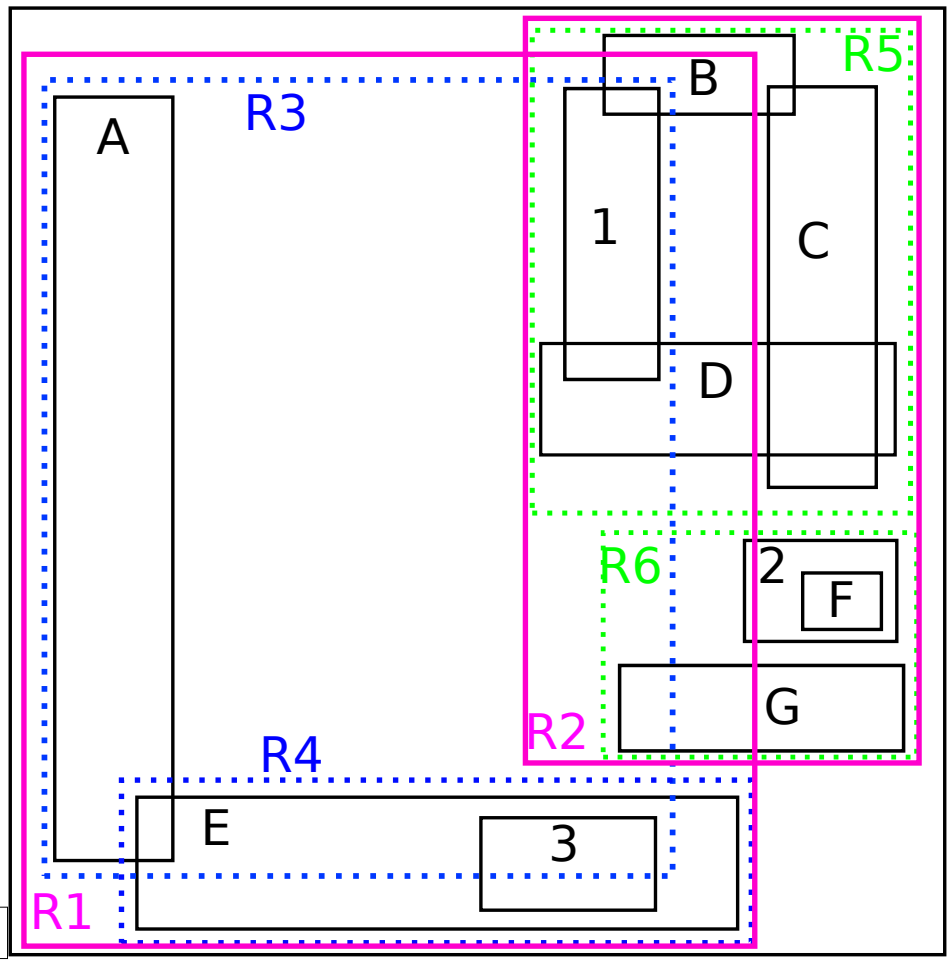
- Bei Suche in dieser "Struktur" müssen für ein gegebenes Rechteck alle Kinderknoten besucht werden, deren MBRs mit dem Anfragerechteck überlappen
- MBRs können sich überlappen. Wozu führt dies bei der Suche?





R-Baum - Beispiel





Source: Buch von H. Samet

R-Baum - Eigenschaften

Eigenschaften

- (m,M) R-Baum
- Jeder Knoten hat zwischen $m \leq \lceil M/2 \rceil$ und M Einträge
- Die Wurzel hat mindestens 2 Einträge, es sei denn sie ist ein Blatt

Vergleich mit B+ Baum

- Klar: Ähnliches Prinzip. Auch der B+ Baum teilt in jedem (inneren) Knoten anhand der Einträge den Datenraum auf
- Aber: Im B+ Baum überlappen sich diese Regionen nicht

Suche im R-Baum

Ähnlich zur Suche im B+ Baum

- Hier können allerdings auch mehrere Kinderknoten besucht werden (müssen!)

Suche

- Ziel: Finde alle Rechtecke, die mit gegebenem Rechteck S überlappen
- Wurzel ist T

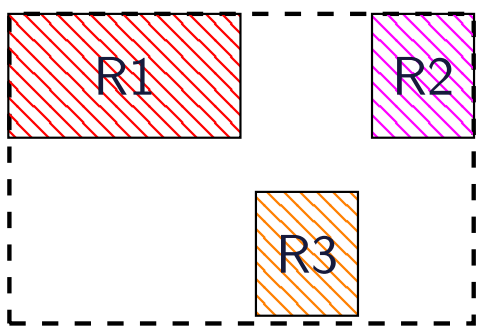
S1 [Suche nach Teilbäumen] Falls T kein Blatt ist, suche in allen Einträgen E in T ob das MBR von E mit T überlappt. Ja? Dann suche weiter im Teilbaum auf den Eintrag E verweist.

S2 [Suche in Blättern] Ist T ein Blatt: Schaue alle Einträge E an. Diejenigen, die mit S überlappen sind Teil des Ergebnisses der Suche.

R-Baum - Einfügen

Einfügen

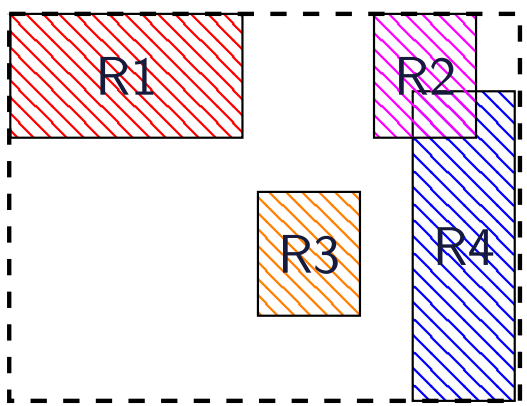
- Ist ein Blatt noch nicht voll, so wird ein Rechteck einfach eingefügt und die entsprechende Bounding-Box (MBR) des Elternknoten-Eintrags wird angepasst (wenn nötig)



R-Baum - Einfügen

Einfügen

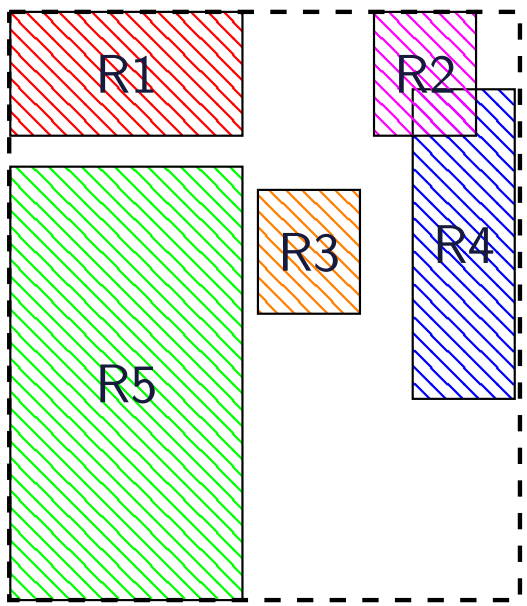
- Ist ein Blatt noch nicht voll, so wird ein Rechteck einfach eingefügt und die entsprechende Bounding-Box (MBR) des Elternknoten-Eintrags wird angepasst (wenn nötig)



R-Baum - Einfügen

Einfügen

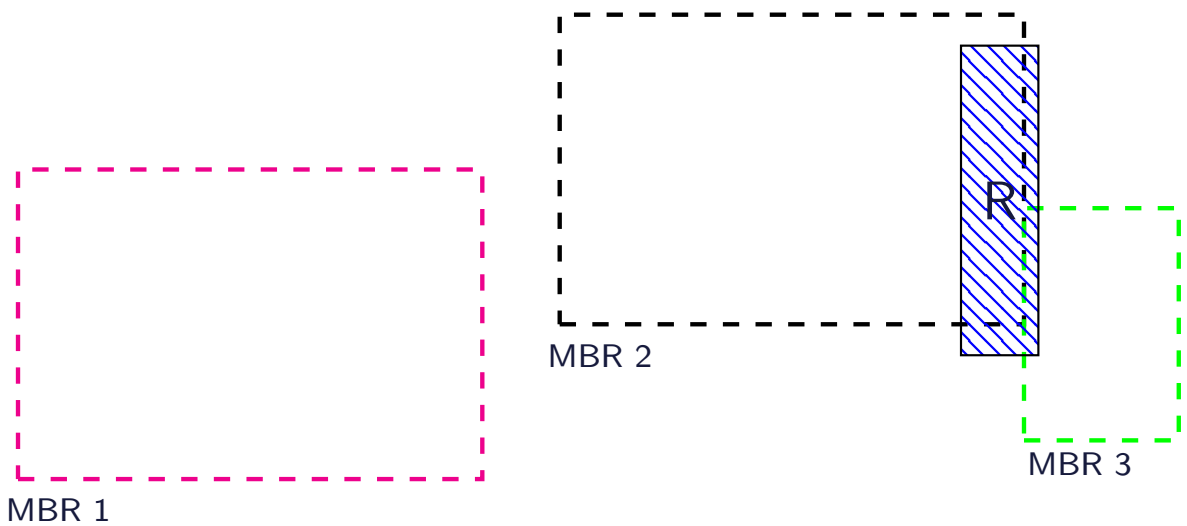
- Ist ein Blatt noch nicht voll, so wird ein Rechteck einfach eingefügt und die entsprechende Bounding-Box (MBR) des Elternknoten-Eintrags wird angepasst (wenn nötig)



R-Baum - Einfügen (2)

Einfügen

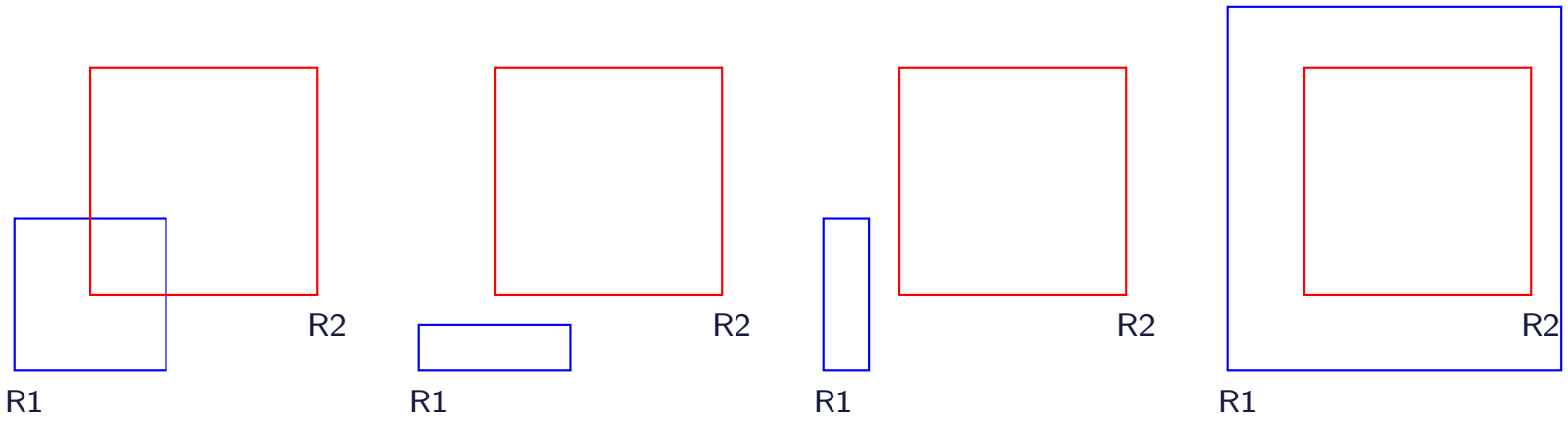
- Es ist nicht immer klar, in welchen Knoten ein Objekt eingefügt werden sollte. Hier: MBR 1, MBR 2 oder MBR 3



Freiheit beim Einfügen

- In der Tat kann ein Objekt überall eingefügt werden!

Überlappung und MBR Berechnung



Zwei Rechtecke $R1 = (r1MinX, r1MinY, r1MaxY, r1MaxY)$ und $R2 = (r2MinX, r2MinY, r2MaxY, r2MaxY)$ überlappen sich, falls

$$r1MaxX \geq r2MinX \wedge r1MinX \leq r2MaxX \wedge$$
$$r1MaxY \geq r2MinY \wedge r1MinY \leq r2MaxY$$

MBR Berechnung

Offensichtlich ist das Minimum-Bounding-Rectangle definiert als Rechteck, das den maximalen bzw. minimalen Wert aller Rechtecke (in jeder Dimension) annimmt.

Einfügen

- **Prinzipiell kann ein neues Rechteck in jeden Knoten eingefügt werden**
- Wenn dieser voll ist, muss gesplittet werden
- Falls nicht muss evtl. zumindest dessen MBR angepasst werden

Beobachtungen und Anforderung

- Die Erweiterung der Bounding-Boxen ist kritisch für die Performance des R-Baums
- Minimiere Überlappung
- Minimiere Ausdehnung

Einfügen - Algorithmus

Ursprünglich von A. Gutman* vorgeschlagener Algorithmus:

Algorithmus: Suche nach Blatt zum Einfügen (ChooseLeaf)

CS1 Setze N:= Wurzel

CS2 Falls N ein Blatt ist: Gebe N zurück

Falls N kein Blatt ist:

Suche den Eintrag von N, dessen Rechteck (MBR) den geringsten Flächenzuwachs benötigt, um das neue Rechteck zu umschließen. Falls es mehrere Möglichkeiten gibt, nimm den Eintrag, dessen MBR die kleinste Fläche hat

CS3 Setze N auf diesen Kinderknoten und wiederhole Schritt CS2

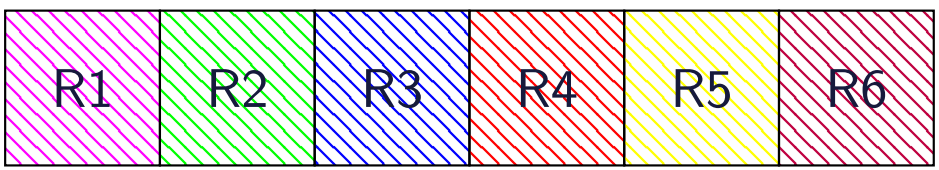
*) Antonin Guttman: R-Trees: A Dynamic Index Structure for Spatial Searching. SIGMOD, 1984.

R-Baum - Überlaufbehandlung

- Falls ein Knoten/Blatt voll ist und ein neuer Eintrag dort nicht mehr gespeichert werden kann, muss wie beim B+ Baum gesplittet werden.
- Der große Unterschied hier ist aber, dass man beliebig splitten kann (nicht nur in der Mitte wie beim B+ Baum)

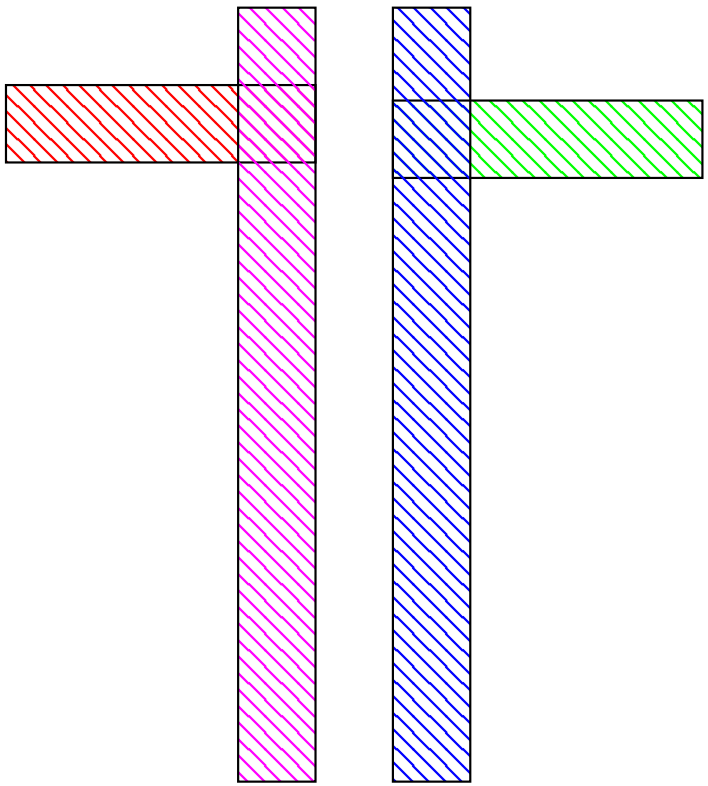
Das Split-Problem

- Gegeben $M + 1$ Einträge eines Knotens.
- Welche zwei Teilmengen dieser Einträge sollen als neuer und alter Knoten im Baum eingetragen werden?



R-Baum -Split-Problem: Beispiel

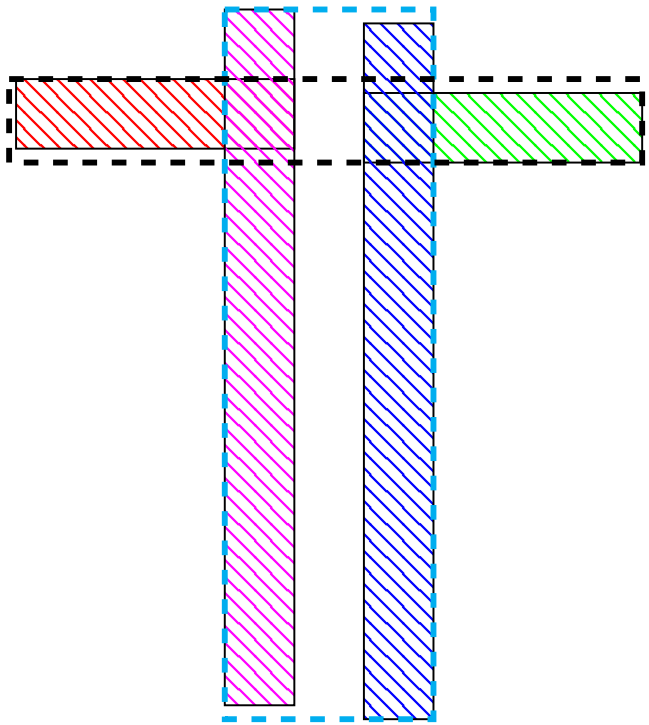
- Wie könnten diese 4 Rechtecke (sinnvoll) in zwei Knoten aufgespalten werden?



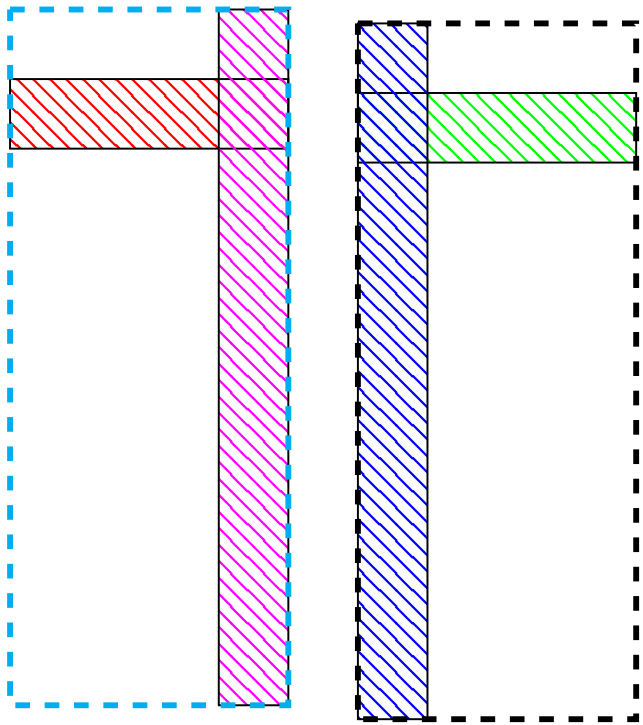
R-Baum -Split-Problem: Beispiel (2)

- Zwei mögliche Splits:

Guter Split



Schlechter Split



- Wieso? Die Fläche des schlechten Splits ist viel größer als die des guten Splits.

R-Baum - Split Problem

- Gegeben $M+1$ Einträge (Rechtecke), wie sollen diese in zwei Knoten aufgeteilt werden?
- In der Originalarbeit von Gutman werden "approximately" 2^{M-1} Möglichkeiten und eine realistische/vernünftige Größe von $M = 50$ angegeben
- D.h. der naive Ansatz, sich alle möglichen Teilmengen anzuschauen und dann die beste Aufteilung zu nehmen, funktioniert nicht (zu teuer!)

R-Baum - Split Problem: Ansatz mit quadratischen Kosten

Quadratische Kosten

- Sucht Split mit kleinstmöglicher Fläche
- Kosten sind quadratisch in M und linear in der Anzahl der Dimensionen d.
- Idee:
  - Suche Paar von Einträgen, die am meisten Fläche verursachen würden falls in einem gemeinsamen Knoten, und packe diese in je einen der Knoten (getrennt)
  - Dann: Betrachte alle übrige Einträge und nehme den Eintrag, für den der Flächenzuwachs den größtmöglichen Unterschied zwischen den Knoten (Gruppen) besitzt
  - Dieser Eintrag wird dann natürlich dem Knoten mit dem kleineren Zuwachs zugewiesen
  - Dies wird wiederholt bis alle Knoten zugewiesen sind.

R-Baum - Split Problem: Ansatz Quadratische Kosten (2)

Aus dem Originalpapier:

Algorithm **Quadratic Split** Divide a set of $M+1$ index entries into two groups

QS1 [Pick first entry for each group] Apply Algorithm **PickSeeds** to choose two entries to be the first elements of the groups Assign each to a group

QS2 [Check if done] If all entries have been assigned, stop If one group has so few entries that all the rest must be assigned to it in order for it to have the minimum number m, assign them and stop

QS3 [Select entry to assign] Invoke Algorithm **PickNext** to choose the next entry to assign Add it to the group whose covering rectangle will have to be enlarged least to accommodate it Resolve ties by adding the entry to the group with smaller area, then to the one with fewer entries, then to either Repeat from QS2

Algorithm **PickSeeds** Select two entries to be the first elements of the groups

PS1 [Calculate inefficiency of grouping entries together] For each pair of entries E_1 and E_2, compose a rectangle J including $E_1 I$ and $E_2 I$ Calculate d = area(J) - area($E_1 I$) - area($E_2 I$)

PS2 [Choose the most wasteful pair] Choose the pair with the largest d

Algorithm **PickNext** Select one remaining entry for classification in a group.

PN1 [Determine cost of putting each entry in each group] For each entry E not yet in a group, calculate d_1 = the area increase required in the covering rectangle of Group 1 to include $E I$ Calculate d_2 similarly for Group 2

PN2 [Find entry with greatest preference for one group] Choose any entry with the maximum difference between d_1 and d_2

Anpassen des Baumes - Methode AdjustTree

- Wird aufgerufen, nachdem neuer Eintrag hinzugefügt wurde
- Behandelt Anpassung des MBR im Elternknoten.
- Propagiert Änderungen, sowohl Handhabung eines Splits sowie Anpassung der MBRs (von unten nach oben)

Algorithm **AdjustTree** Ascend from a leaf node L to the root, adjusting covering rectangles and propagating node splits as necessary

AT1 [Initialize.] Set $N=L$ If L was split previously, set NN to be the resulting second node

AT2 [Check if done] If N is the root, stop

AT3 [Adjust covering rectangle in parent entry] Let P be the parent node of N, and let E_N be N's entry in P Adjust $E_N.I$ so that it tightly encloses all entry rectangles in N.

AT4 [Propagate node split upward] If N has a partner NN resulting from an earlier split, create a new entry E_{NN} with $E_{NN}.p$ pointing to NN and $E_{NN}.I$ enclosing all rectangles in NN Add E_{NN} to P if there is room. Otherwise, invoke **SplitNode** to produce P and PP containing E_{NN} and all P's old entries

AT5 [Move up to next level.] Set $N=P$ and set $NN=PP$ if a split occurred. Repeat from AT2.

Beispiele

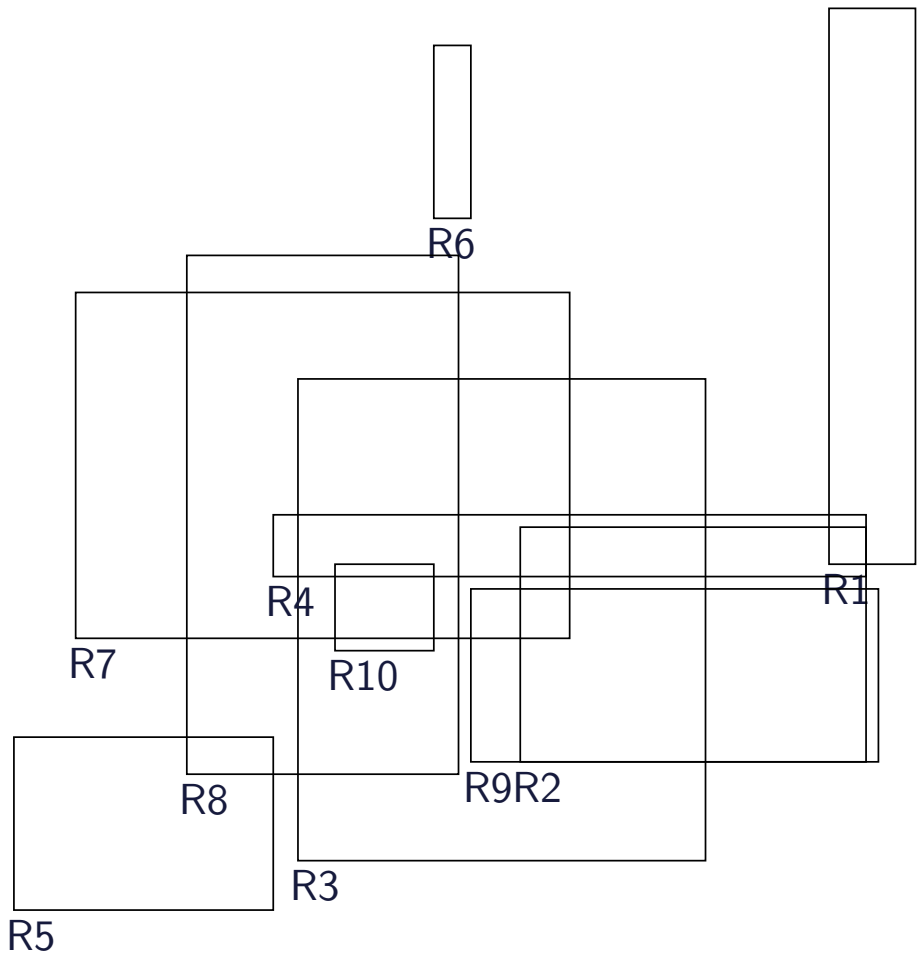
Implementierung/Setup

- Implementierung eines R-Baums aus `https://github.com/aled/jsi`
- Zufällig generierte Rechtecke
- Parameter:
  - MaxNodeEntries=4
  - MinNodeEntries=2

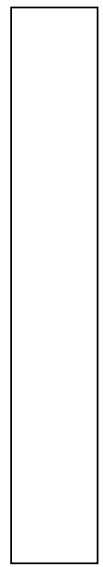
Interpretation

- Tatsächliche (Daten) Rechtecke sind mit R_i benannt
- Einträge in inneren Knoten mit N_j, beliebig, IDs aus Java Objekten (im Zweifelsfall gar nicht interpretieren!)
- Korrespondieren zu MBRs (bunt dargestellt, aber nicht einzeln benannt)

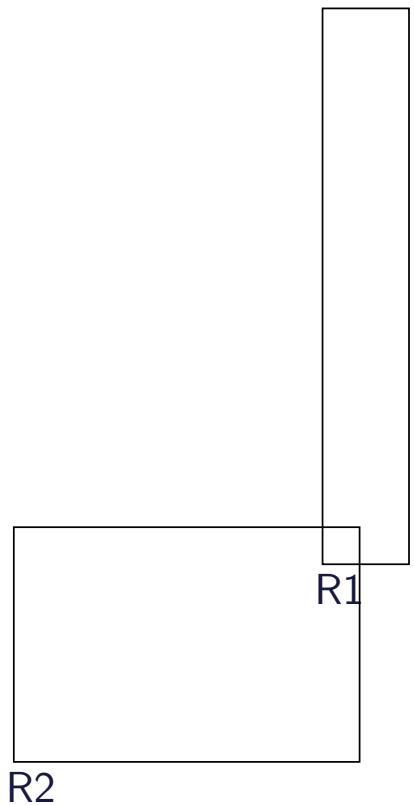
1. Beispieldatensatz: 10 Rechtecke



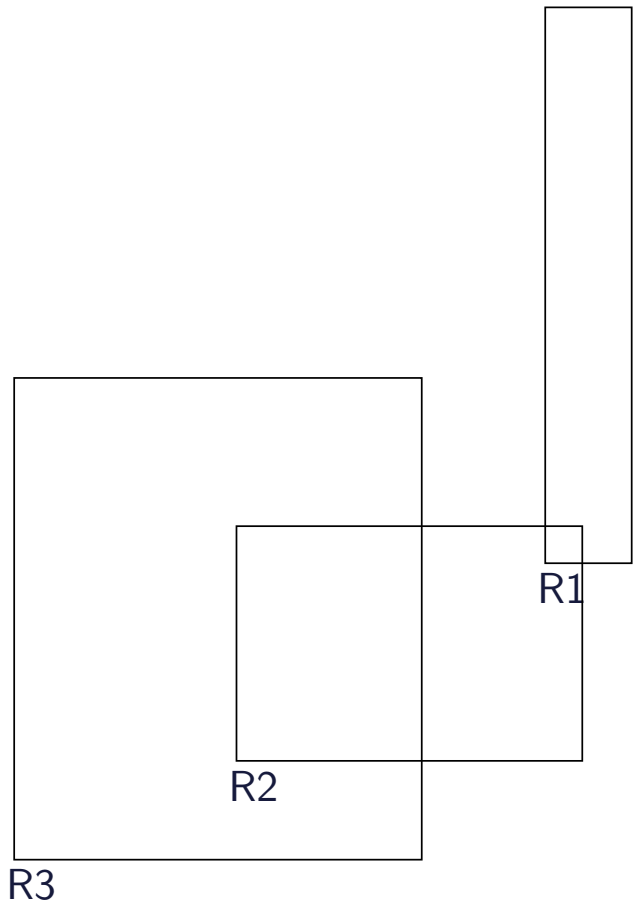
Einfügen von R1



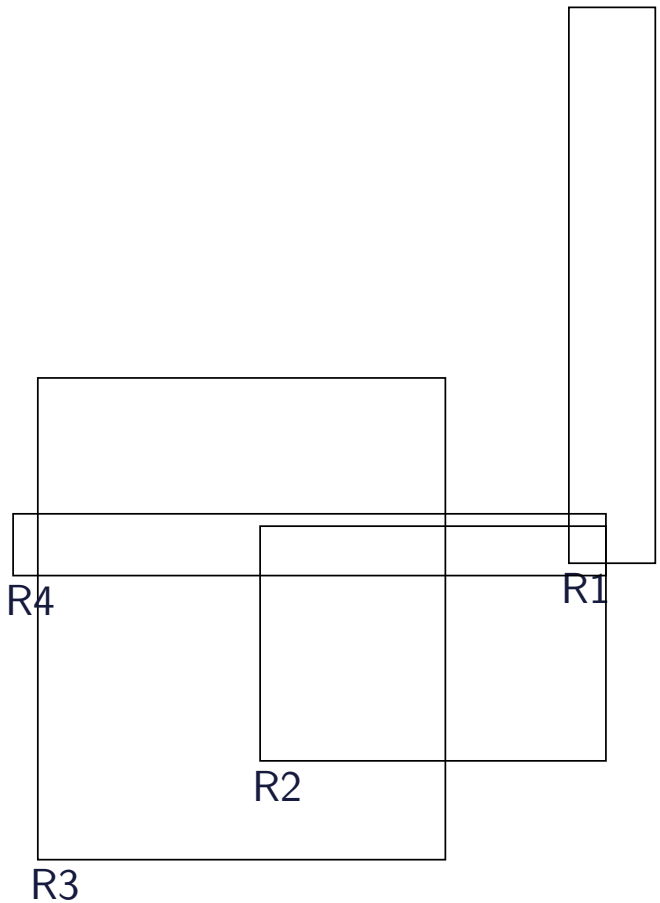
Einfügen von R2



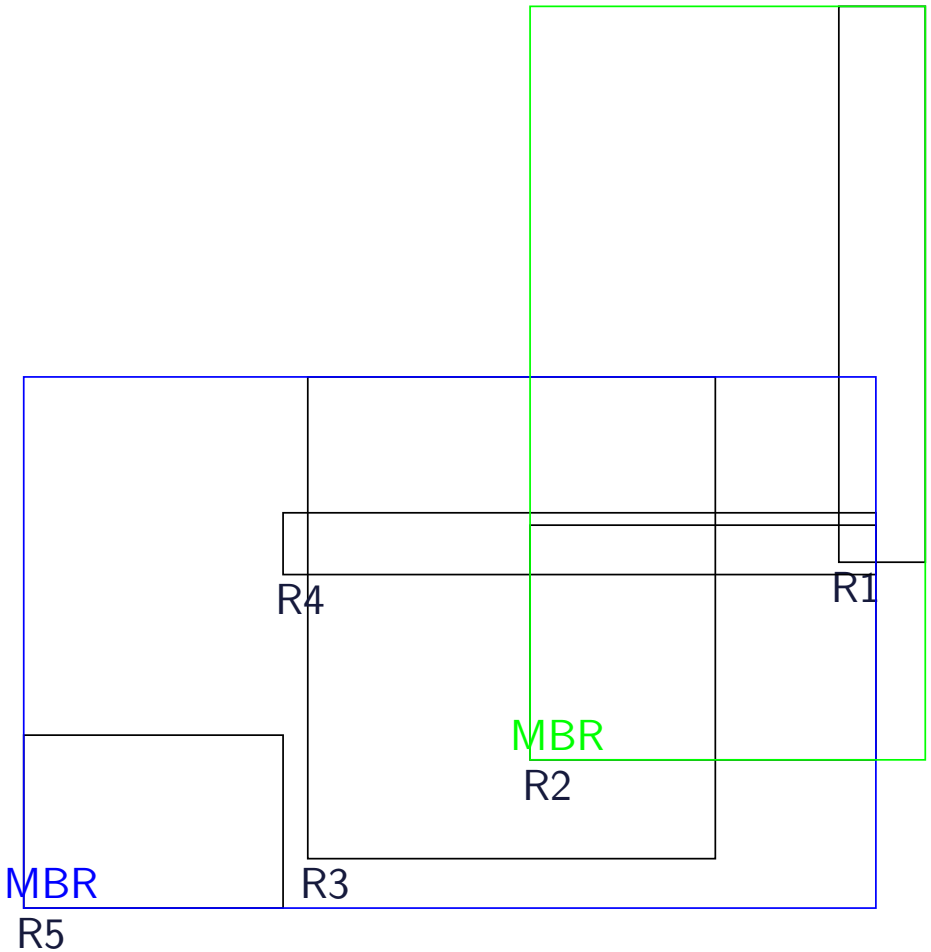
Einfügen von R3



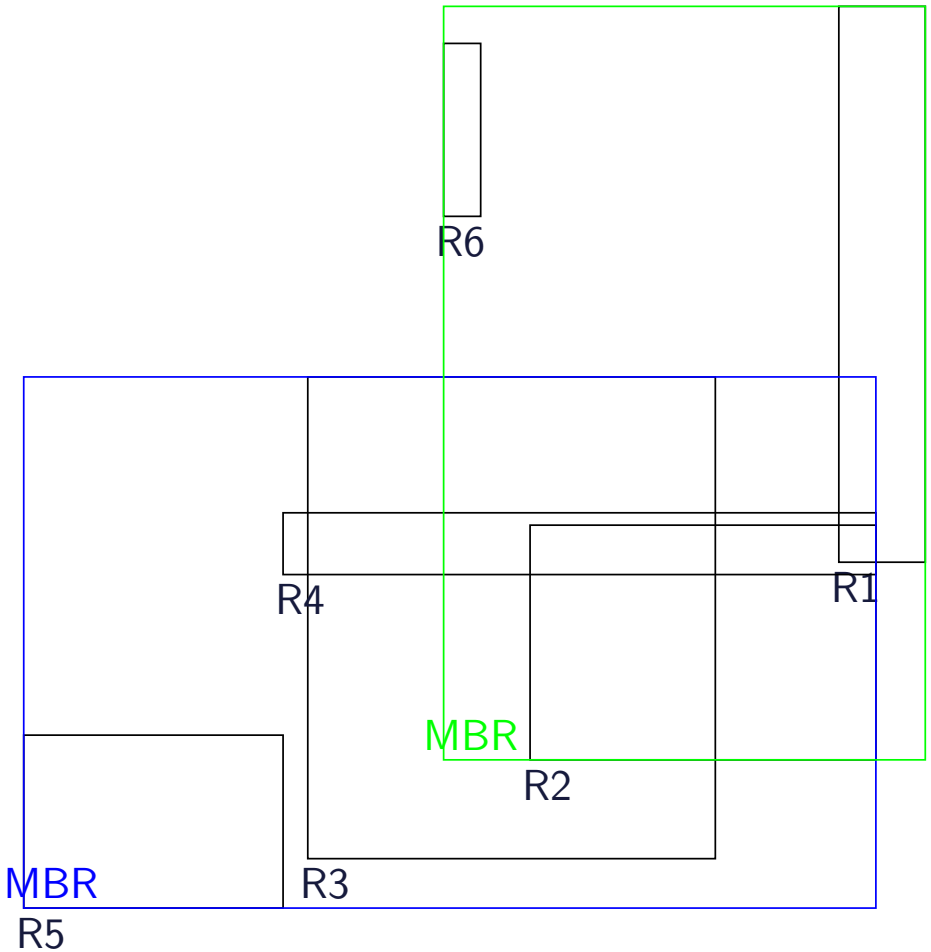
Einfügen von R4



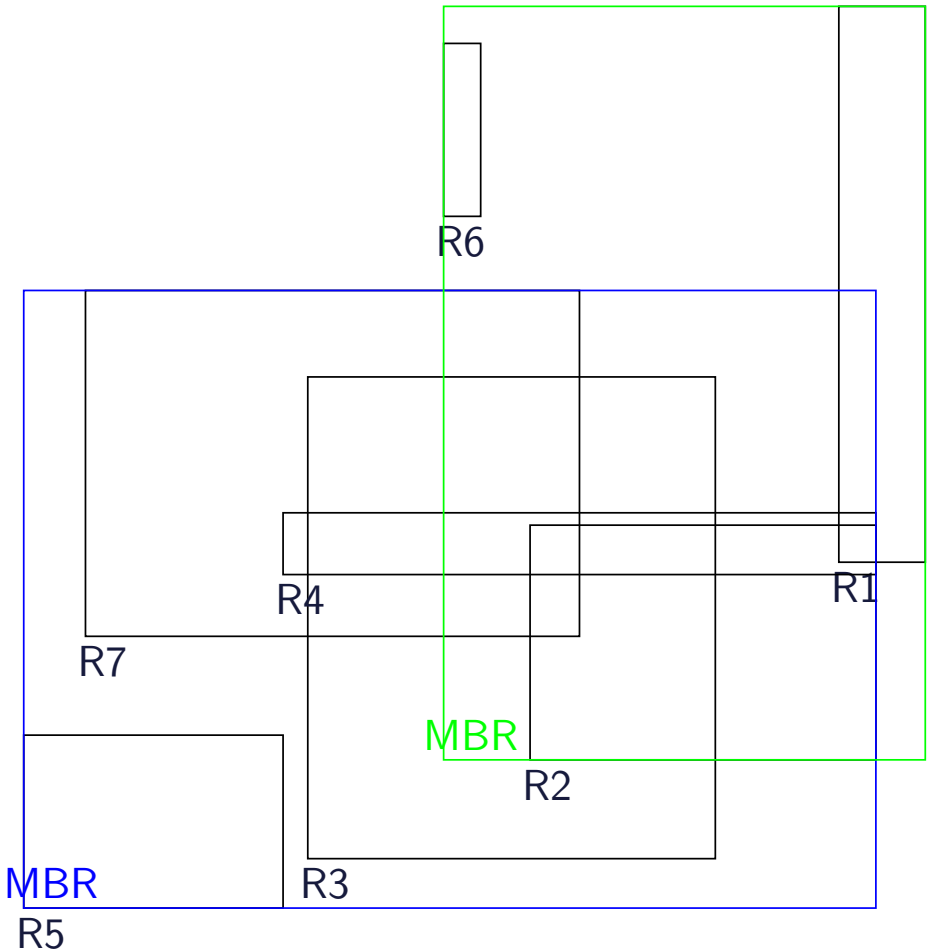
Einfügen von R5



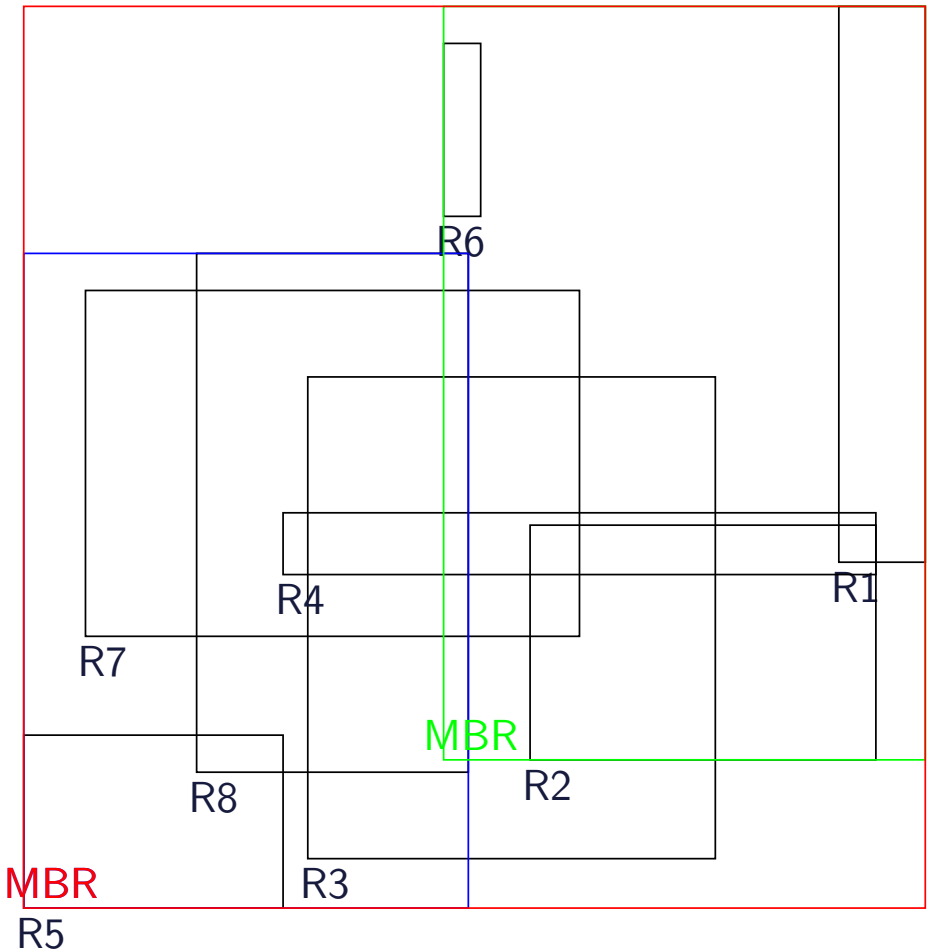
Einfügen von R6



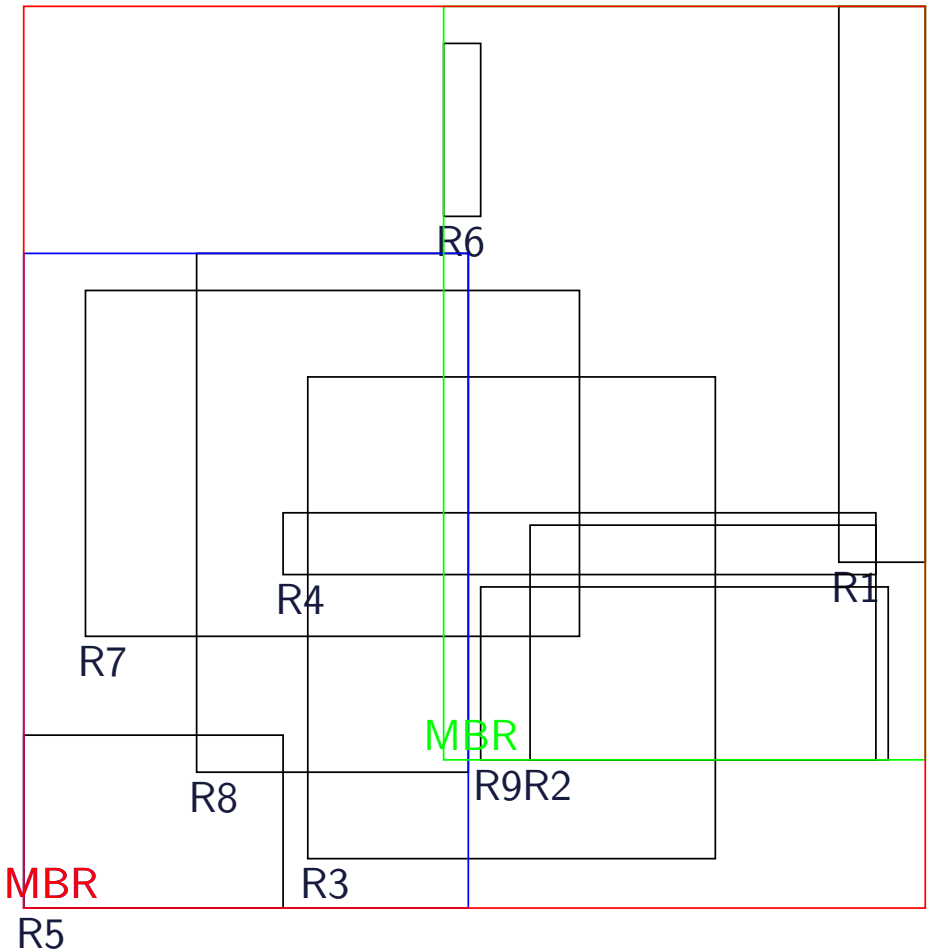
Einfügen von R7



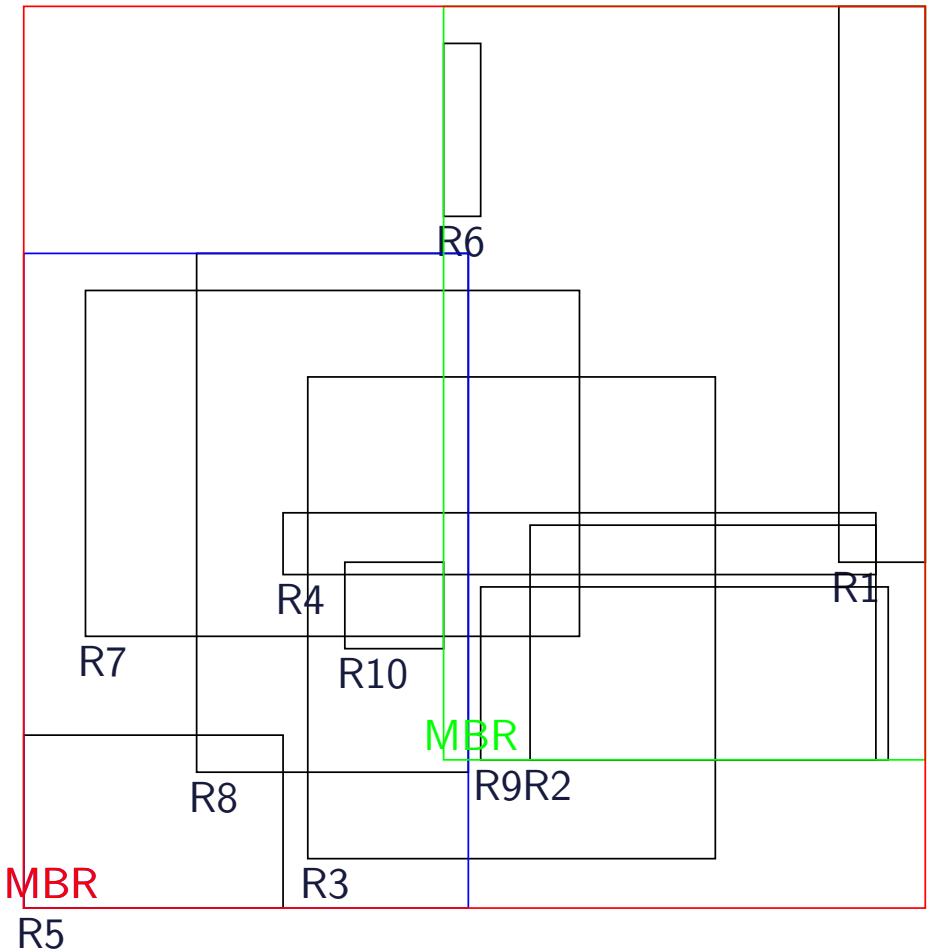
Einfügen von R8



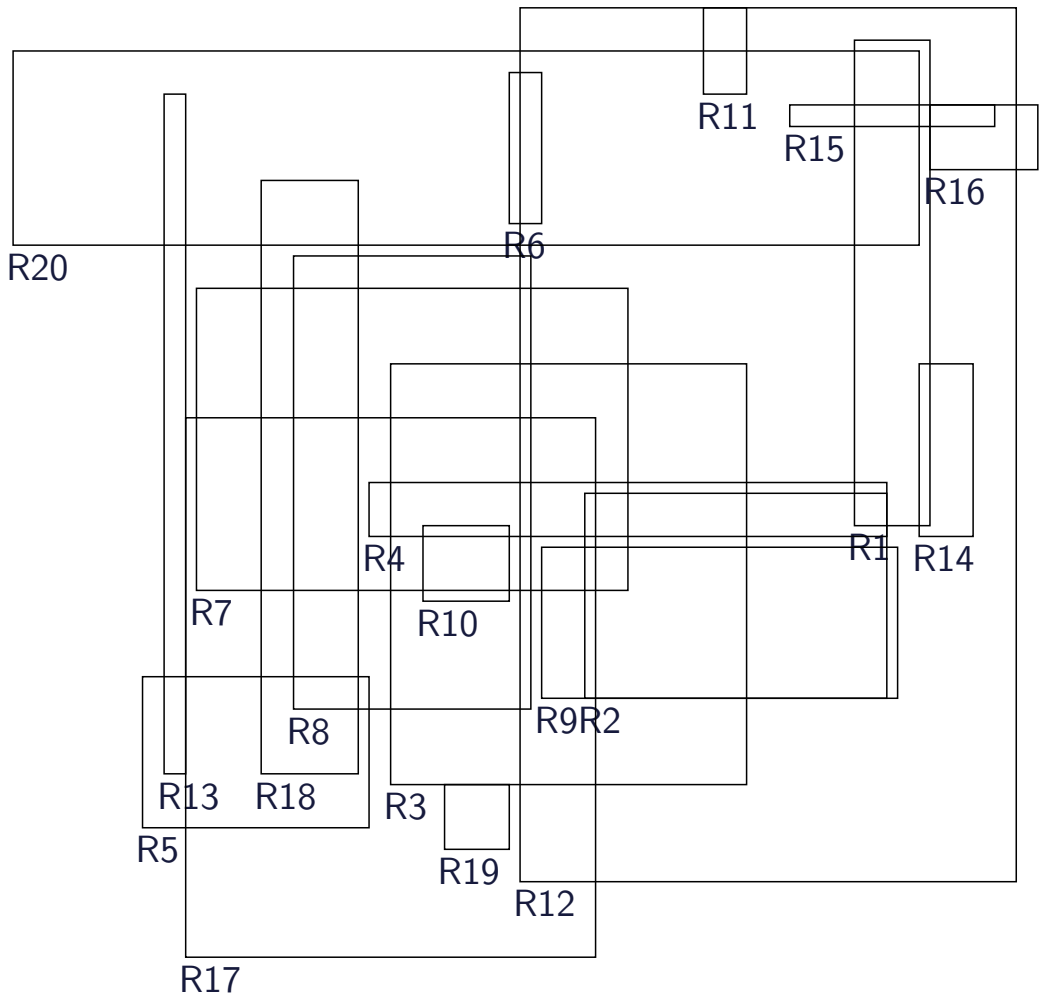
Einfügen von R9



Einfügen von R10



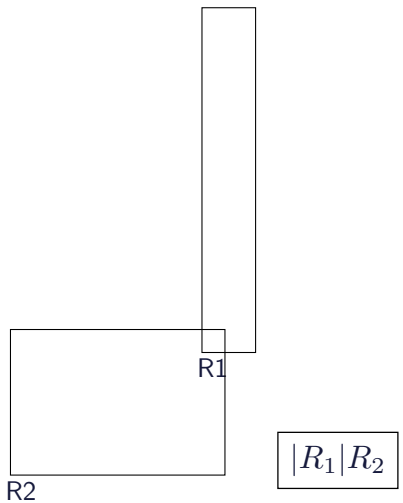
2. Beispieldatensatz: 20 Rechtecke



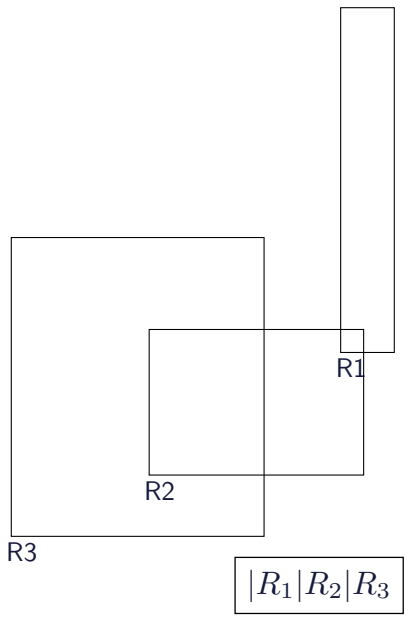
Einfügen von R1

$|R_1$

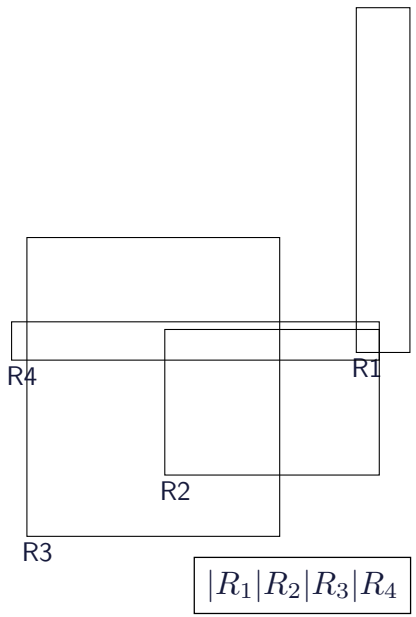
Einfügen von R2



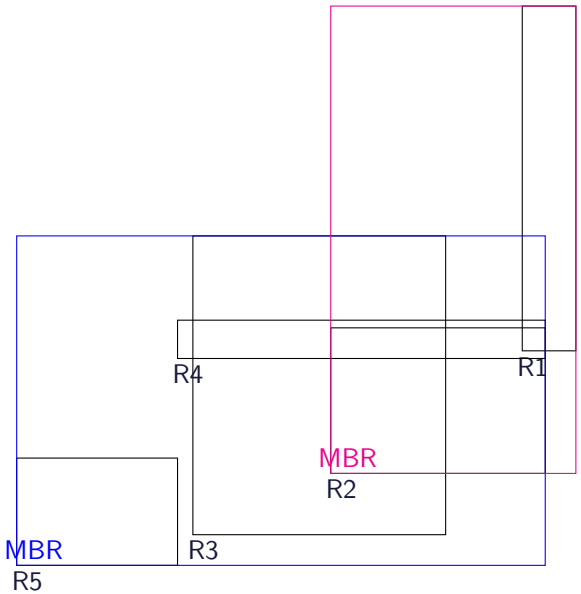
Einfügen von R3

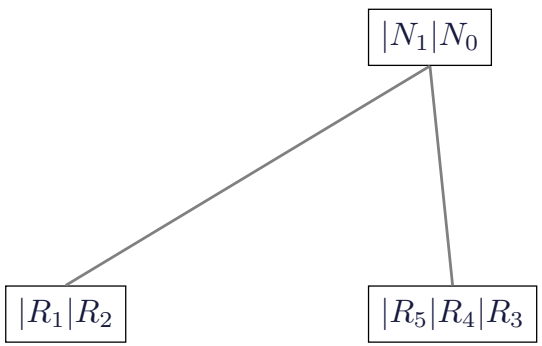


Einfügen von R4

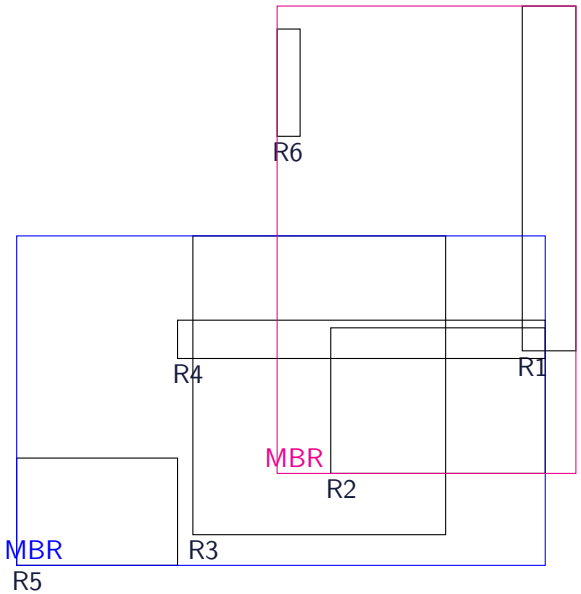


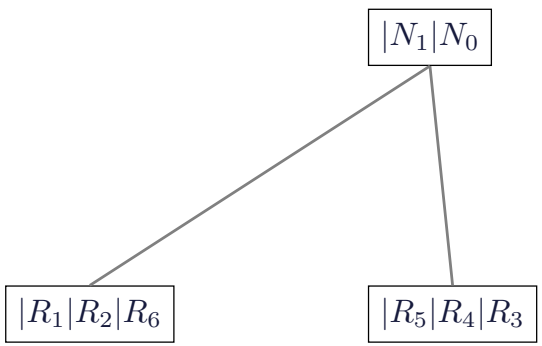
Einfügen von R5



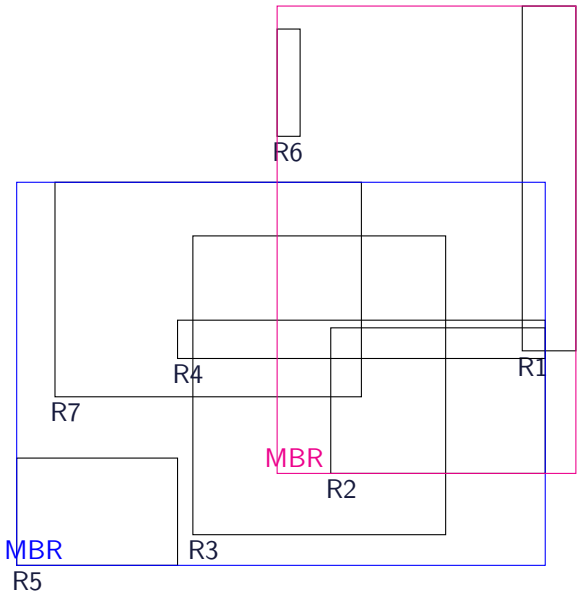


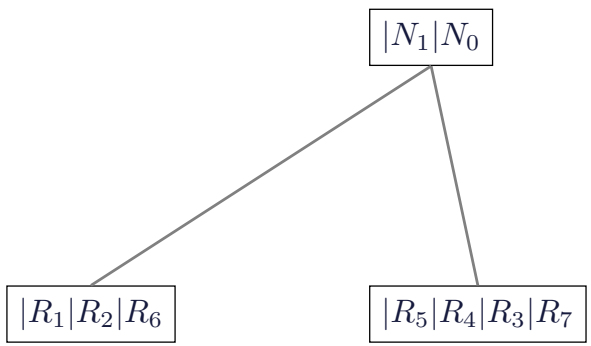
Einfügen von R6



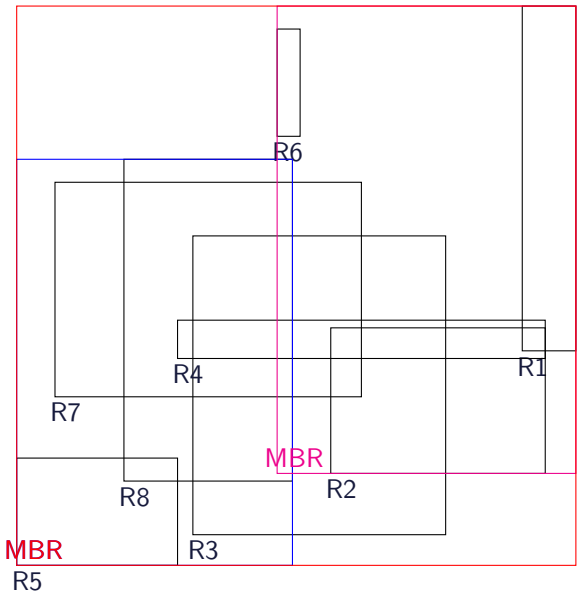


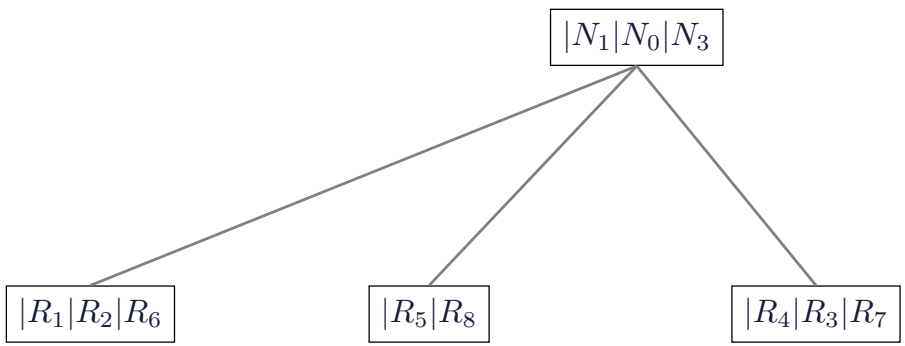
Einfügen von R7



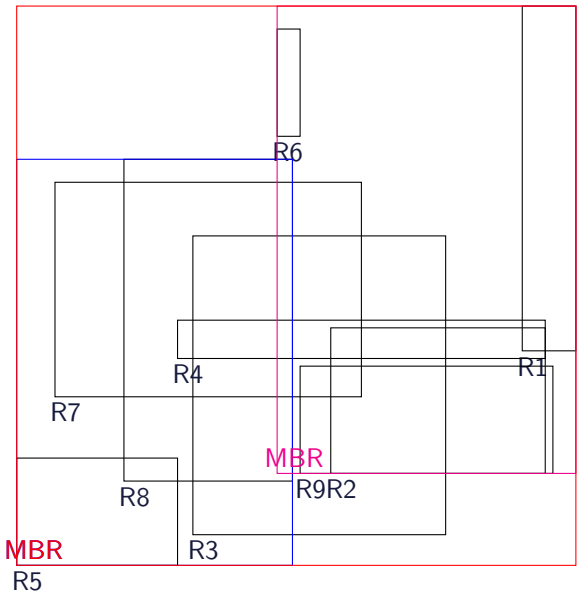


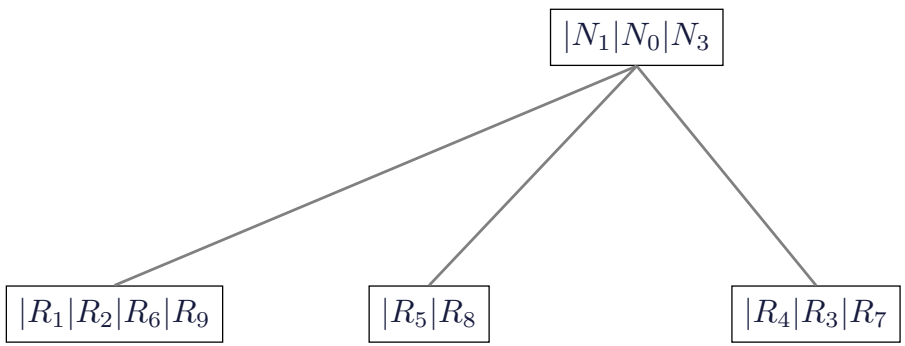
Einfügen von R8



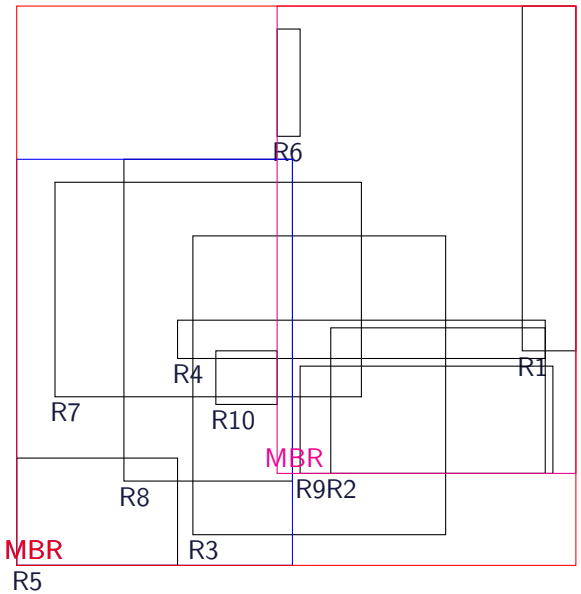


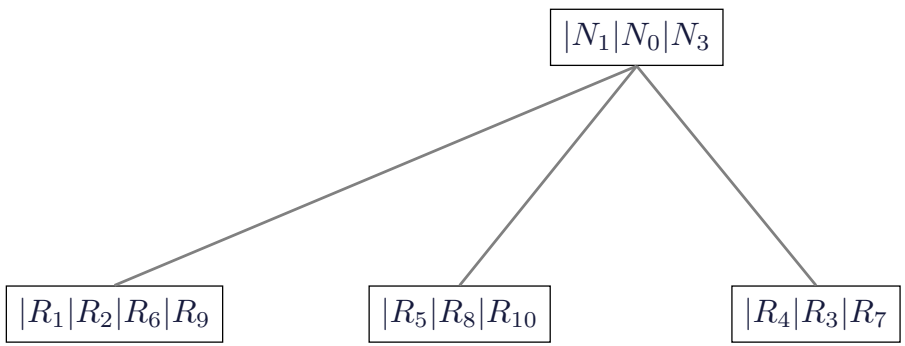
Einfügen von R9



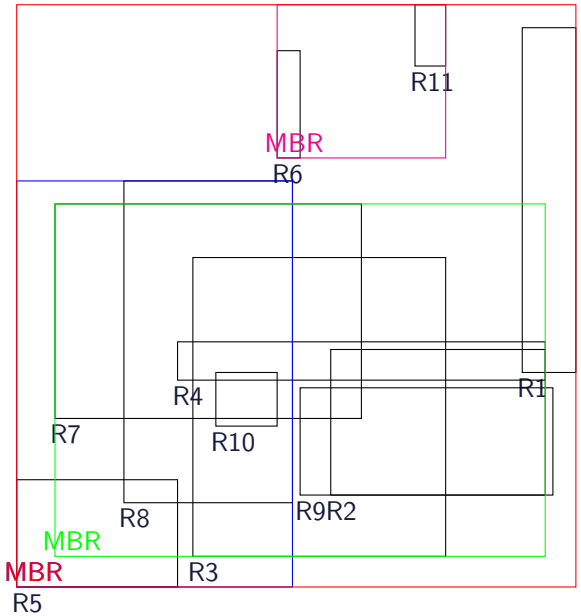


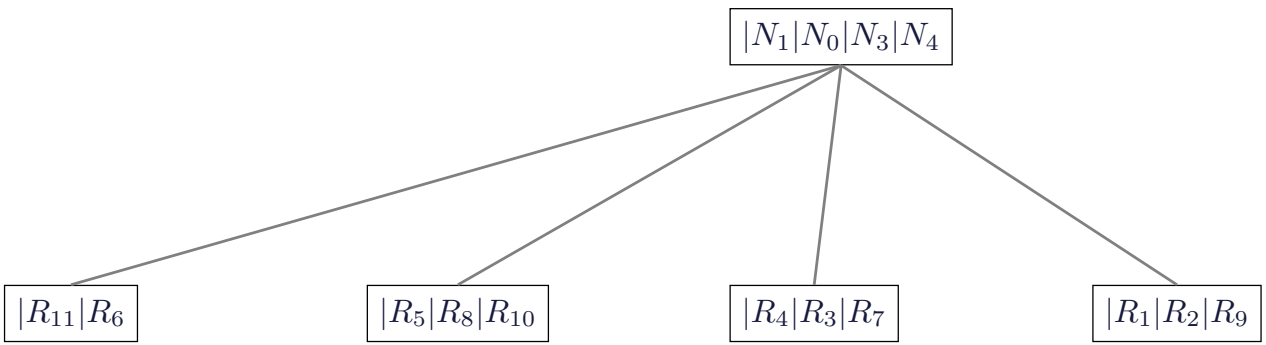
Einfügen von R10



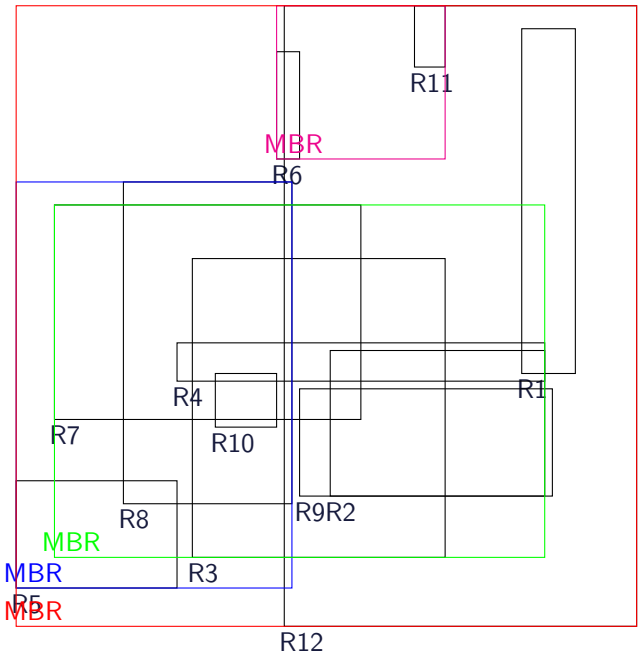


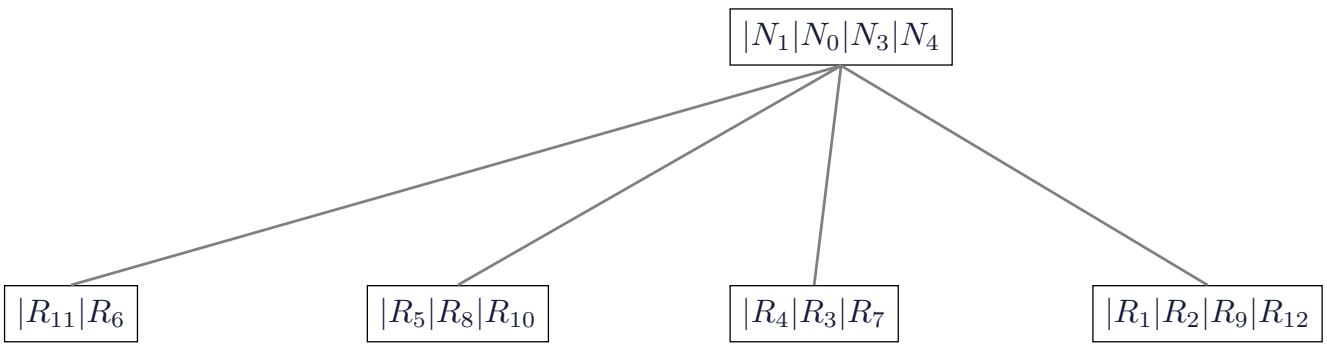
Einfügen von R11



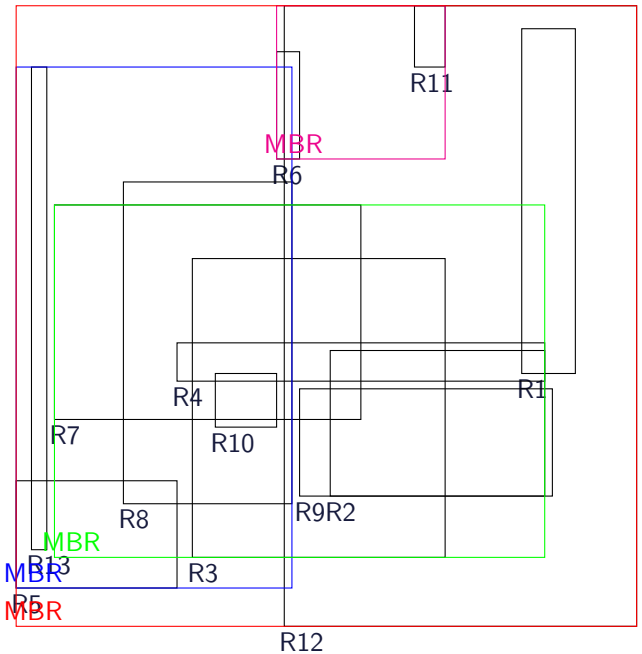


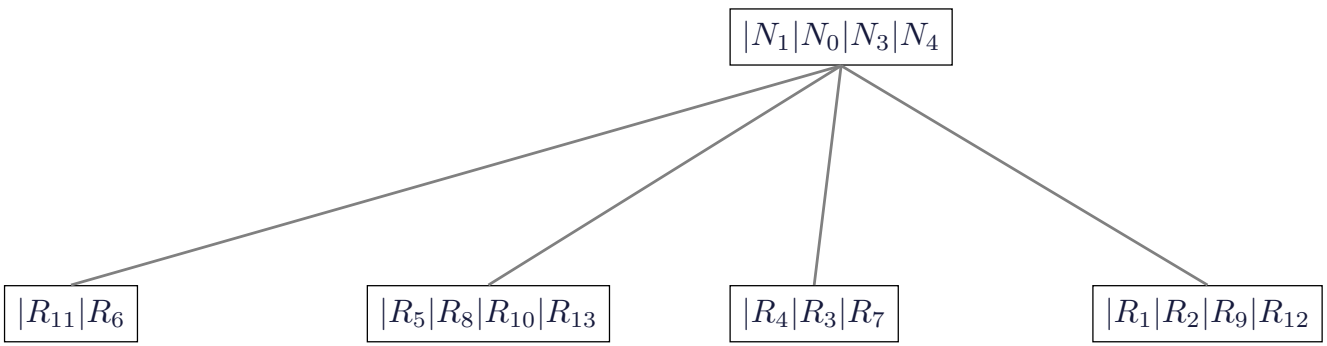
Einfügen von R12



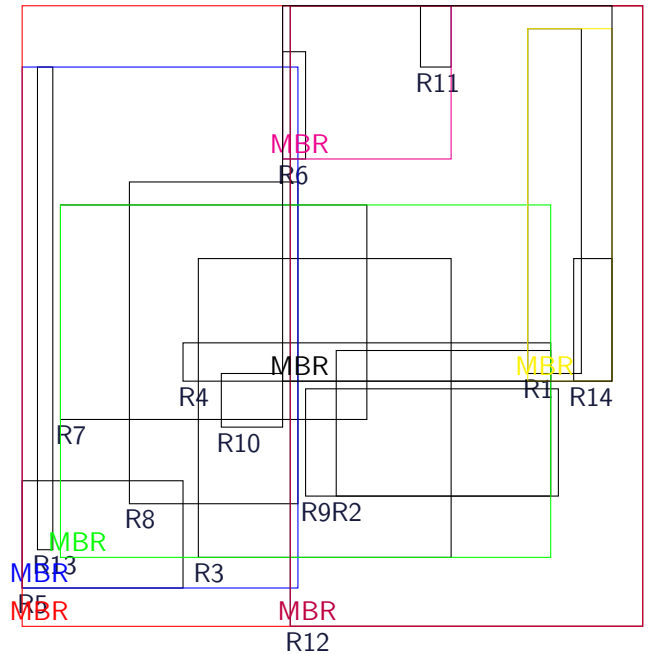


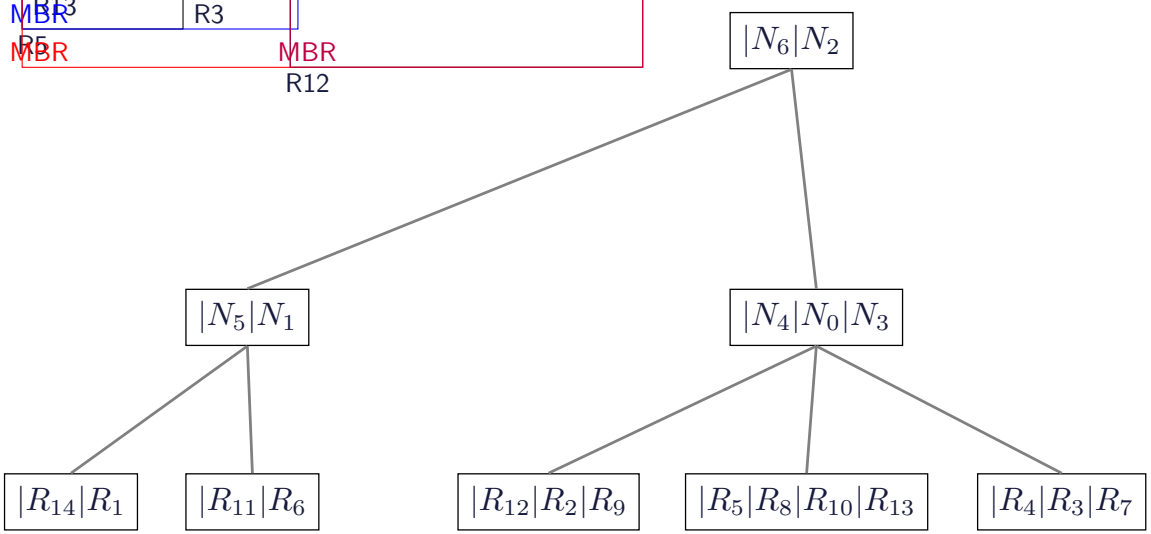
Einfügen von R13



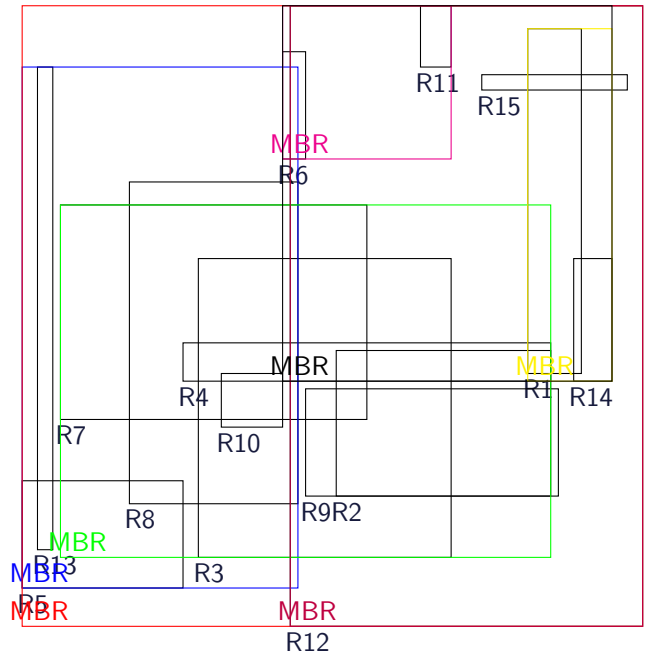


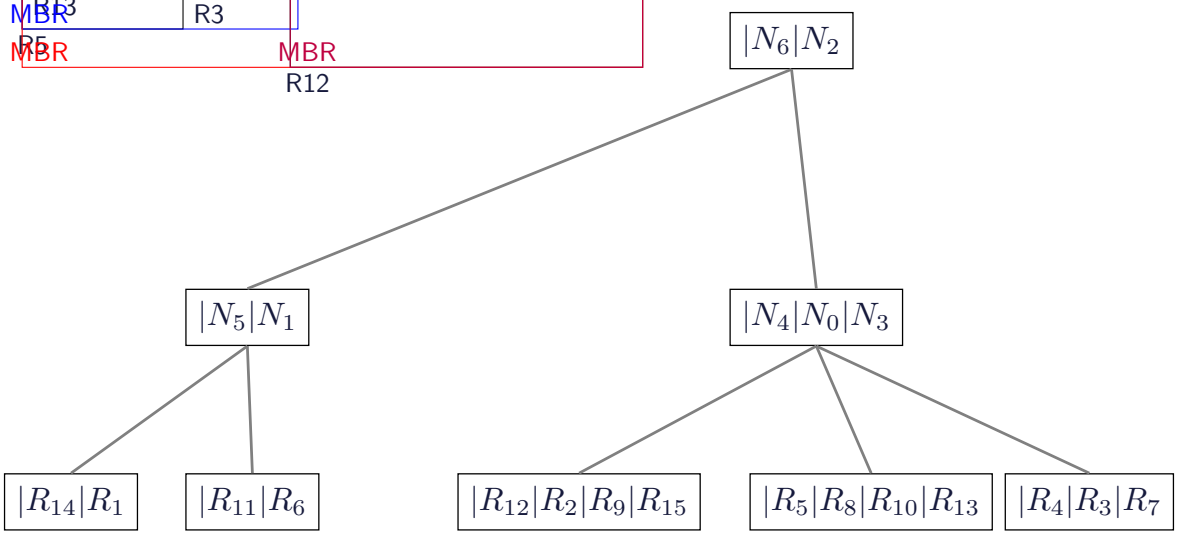
Einfügen von R14



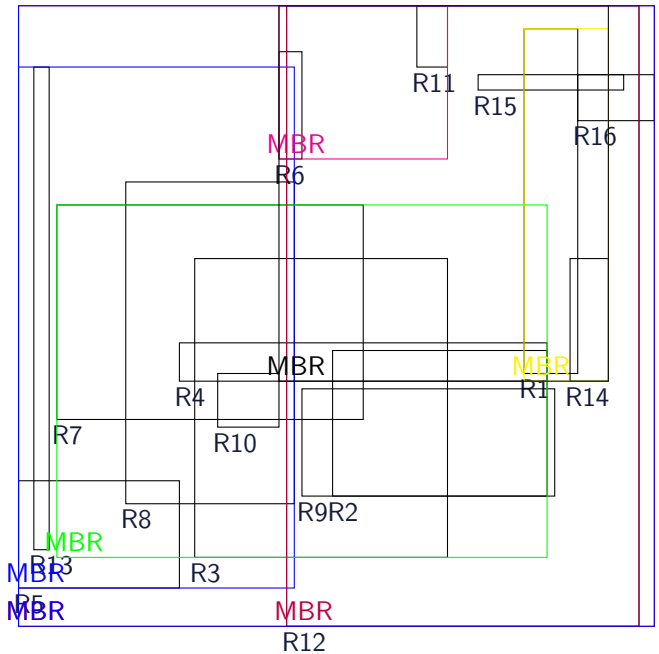


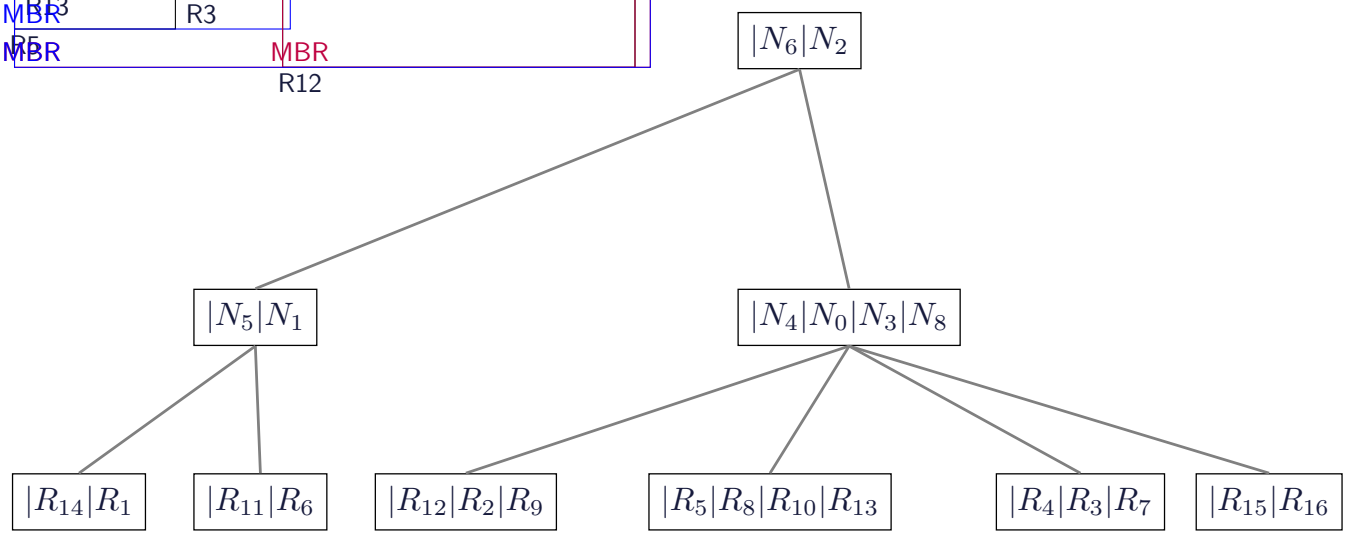
Einfügen von R15



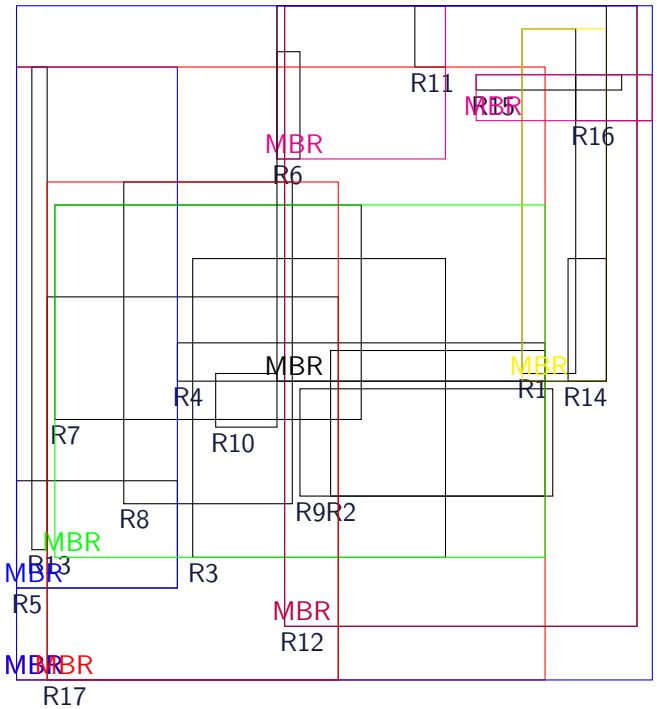


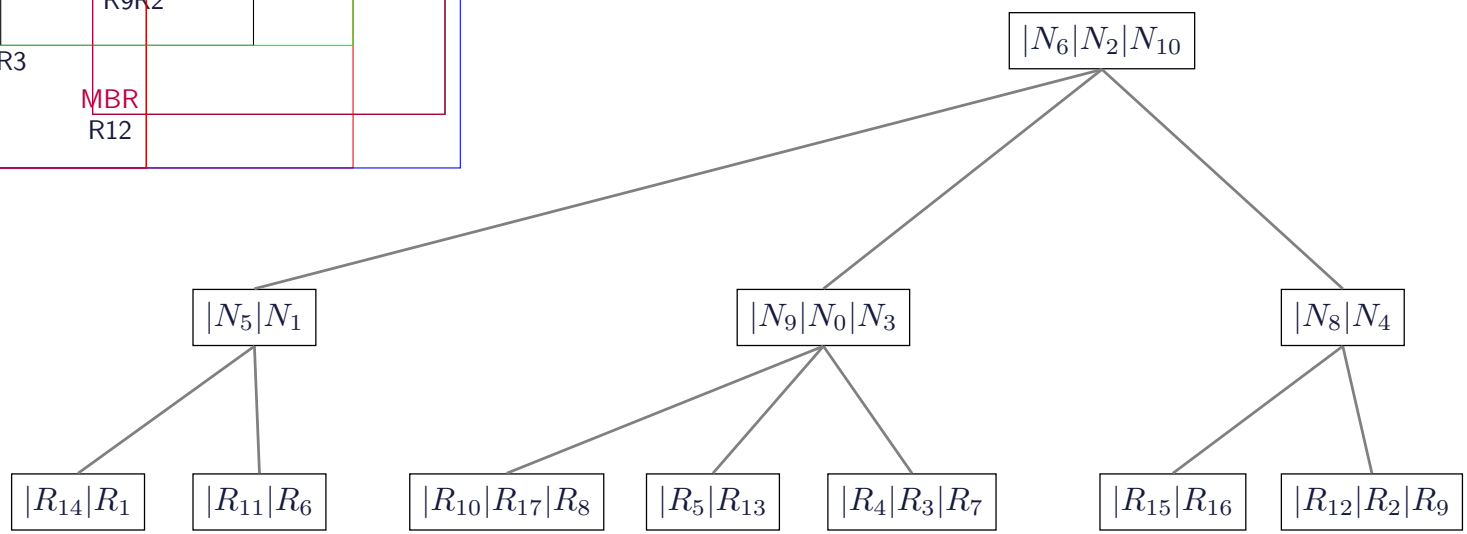
Einfügen von R16



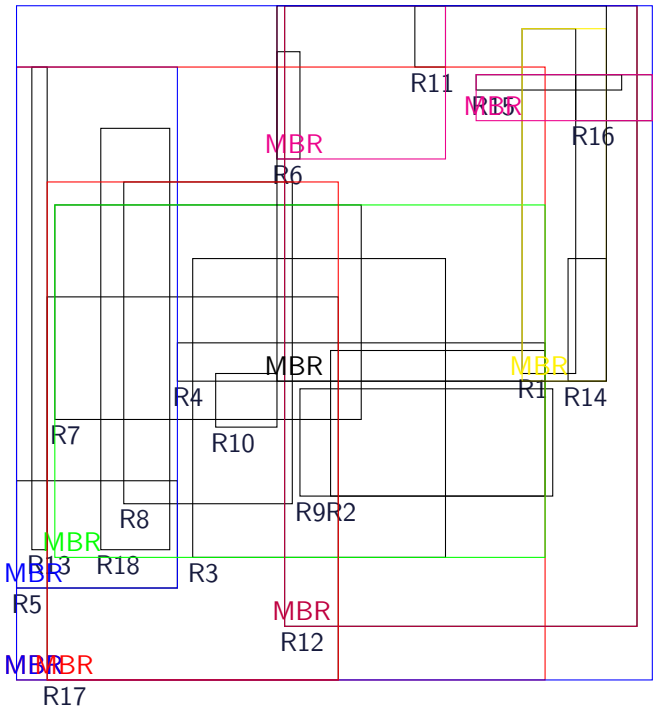


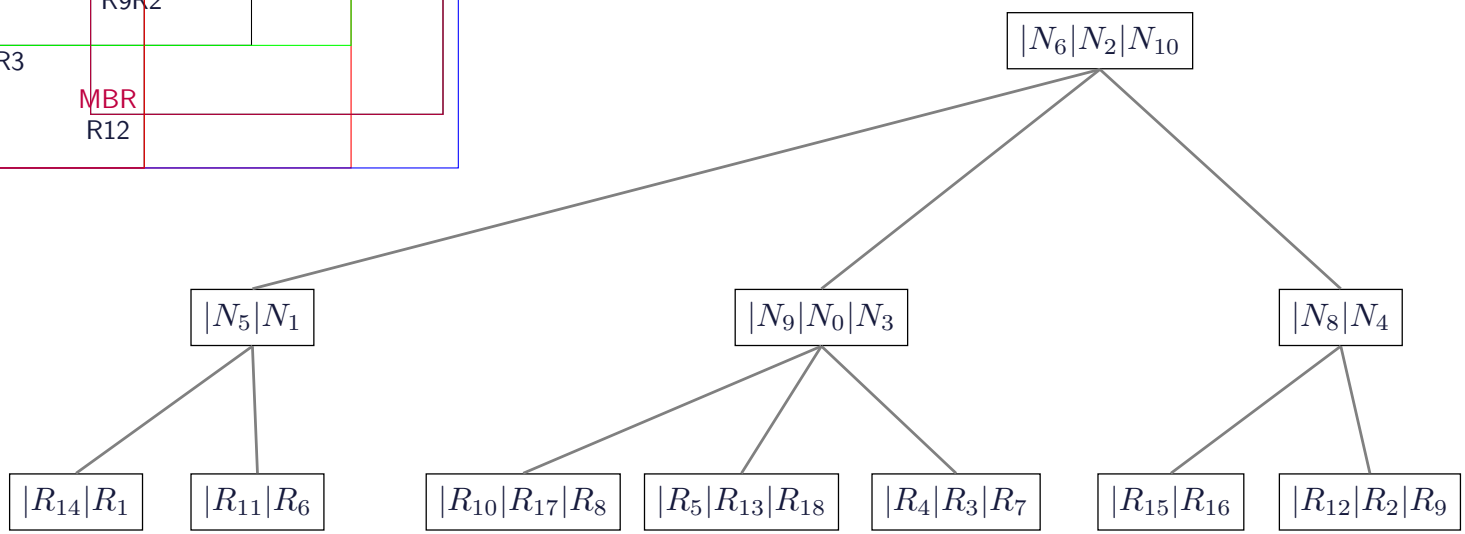
Einfügen von R17



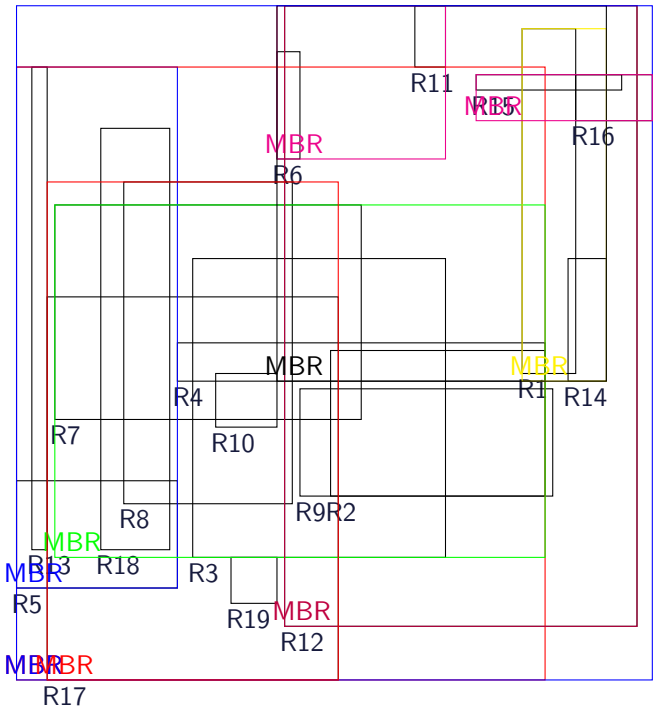


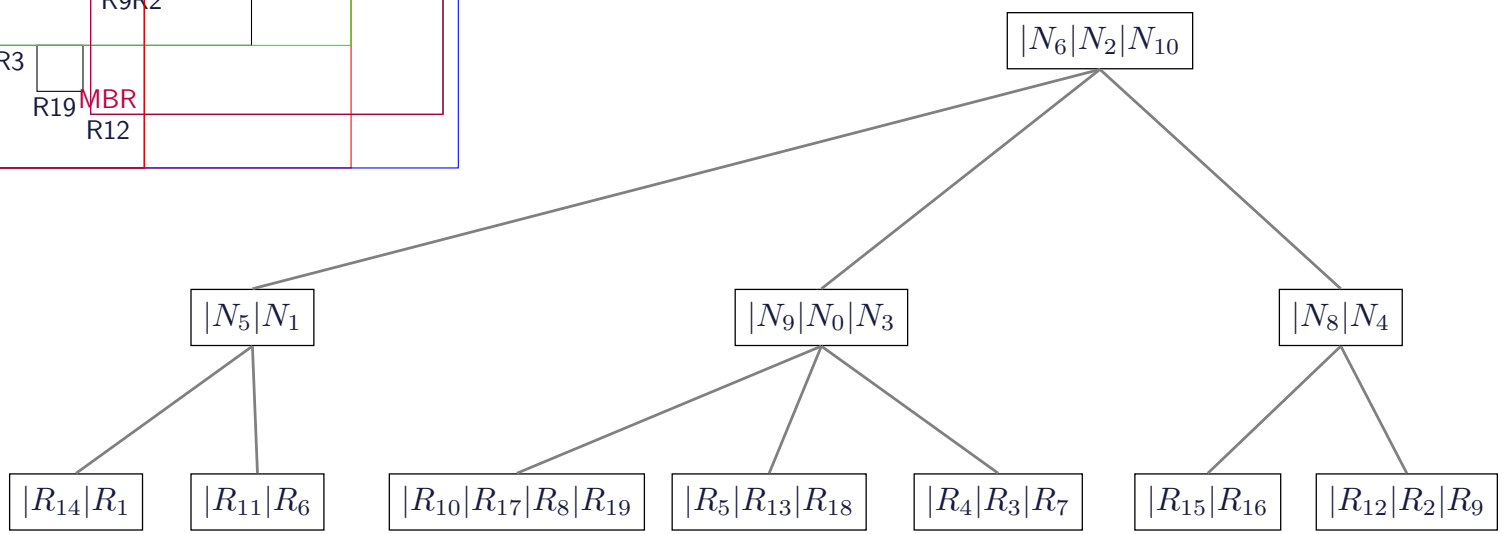
Einfügen von R18



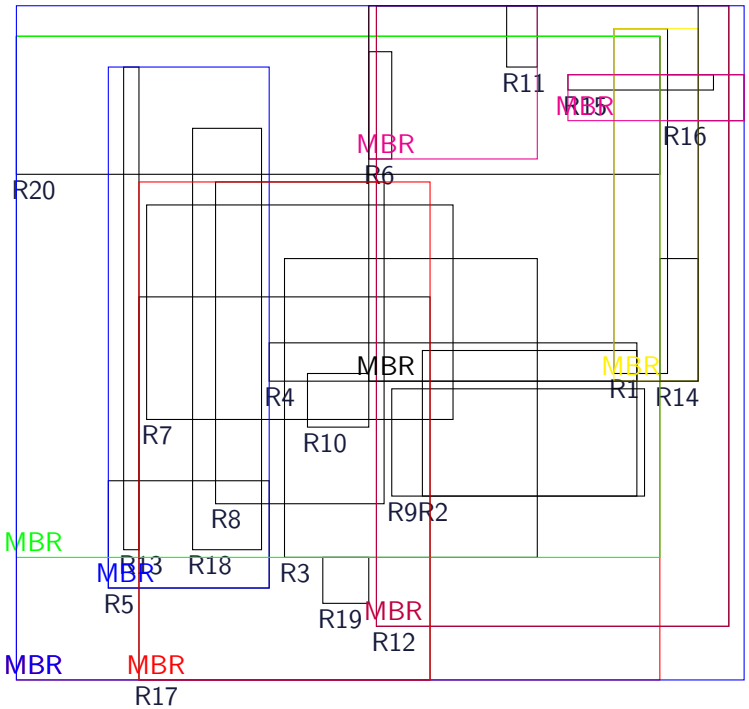


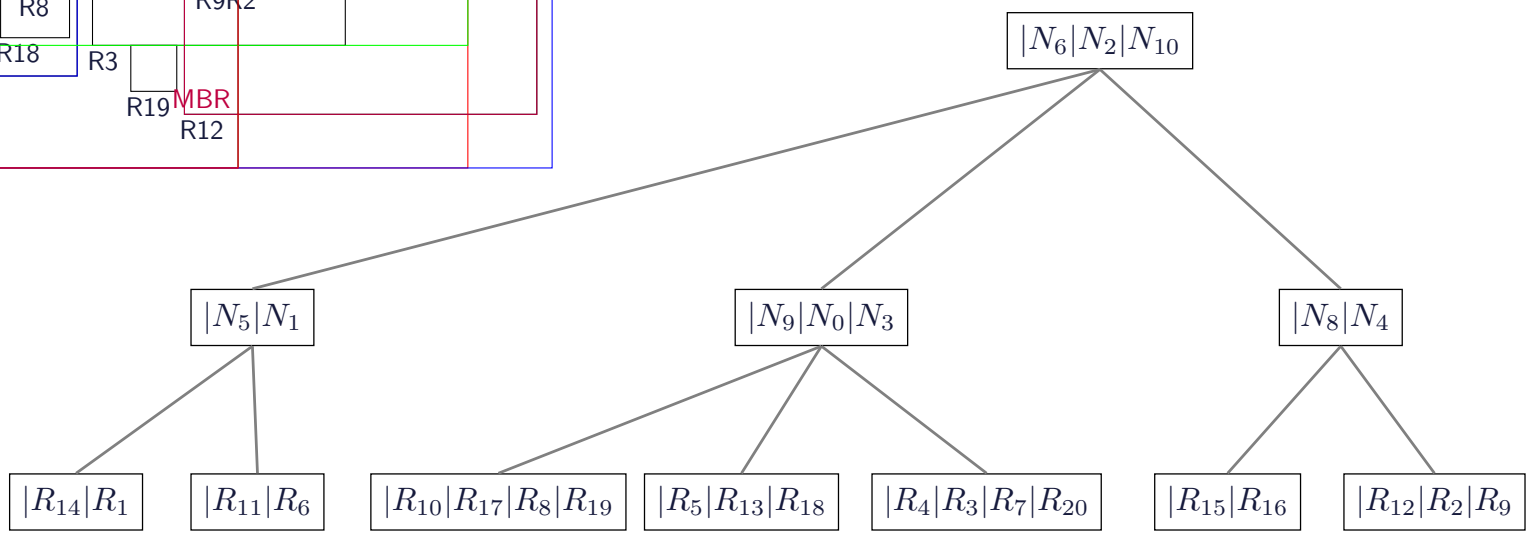
Einfügen von R19





Einfügen von R20





R*-Baum

- Verschiedene nachfolgende Arbeiten zur Verbesserung des R-Baums mittels "besserer" Split-Algorithmen
- Bekannteste dieser Arbeiten ist (m.E.) der R*-Baum von Beckmann et al.

R*-Baum

- Entwicklung von Kriterien und effizienter Algorithmen zum Aufteilen von Knoten im R-Baum
- Insbesondere wird nicht nur der Flächenzuwachs betrachtet sondern Grad der Überlappung

Paper: Beckmann, Kriegel, Schneider, Seeger: The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. SIGMOD, 1990.

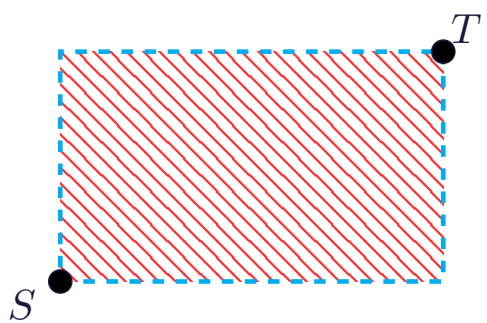
NN Suche im R-Baum

Wir betrachten nun das Problem der Nächsten Nachbar Suche im Falle des R-Baums.

Rechteck im d-dimensionalen Raum

$$R = (S,T)$$

mit $S = [s_1, s_2, \ldots, s_d]$
und $T = [t_1, t_2, \ldots, t_d]$ und $s_i \leq t_i \quad \forall 1 \leq i \leq d$



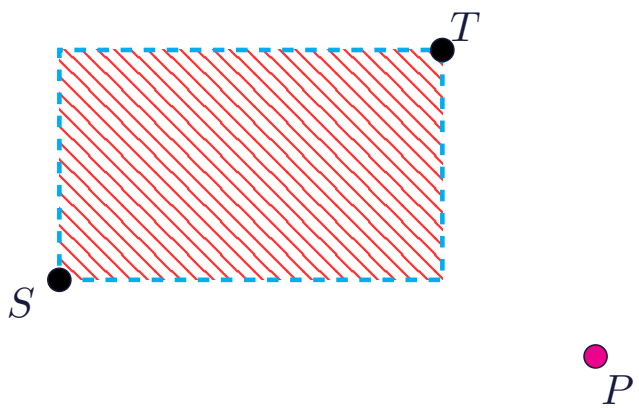
NN Suche im R-Baum

Wir betrachten nun das Problem der Nächsten Nachbar Suche im Falle des R-Baums.

Rechteck im d-dimensionalen Raum

$$R = (S,T)$$

mit $S = [s_1, s_2, \ldots, s_d]$
und $T = [t_1, t_2, \ldots, t_d]$ und $s_i \leq t_i \quad \forall 1 \leq i \leq d$



Was ist die minimale Distanz eines **Punktes** P zu irgendeinem von R umschlossenen Punkt?

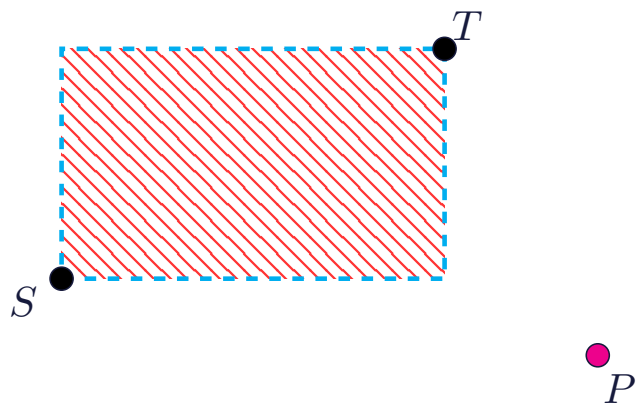
MINDIST

Wir folgen in der Notation dem Papier von N. Roussopoulos et al. Insbesondere, das Distanzmaß hier ist das Quadrat der Eukl. Distanz.

MINDIST(P,R)

Für einen Punkt P und ein Rechteck R definieren wir die Distanz $MINDIST(P,R)$ als

$$MINDIST(P,R) = \sum_{i=1}^{d} |p_i - r_i|^2$$



mit

$$r_i = \begin{cases} s_i & \text{if } p_i < s_i \\ t_i & \text{if } p_i > t_i \\ p_i & \text{sonst} \end{cases}$$

Literatur: N. Roussopoulos, S. Kelley, F. Vincent. Nearest Neighbor Queries. SIGMOD 1995.

MINDIST (2)

Die minimale Distanz eines Punktes P zu einem ausgedehnten (spatial) Objekt o, geschrieben als $||(P,o)||$ ist

$$||(P,o)|| = min_{[x_1,...x_d]\in o}(\sum_{i=1}^{d} |p_i - x_i|^2)$$

MBRs und MINDIST

Gegeben ein Punkt P und ein MBR R welches eine Menge von Objekten $O = \{o_i\}$ umfasst. Dann gilt

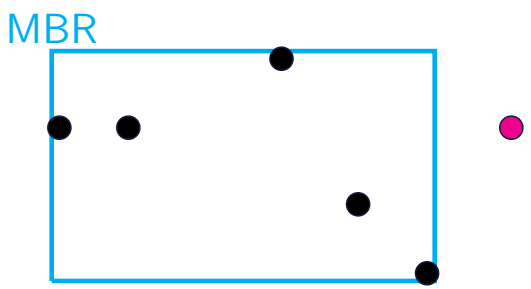
$$\forall o \in O \quad MINDIST(P,R) \leq ||(P,o||$$

MINDIST (3)

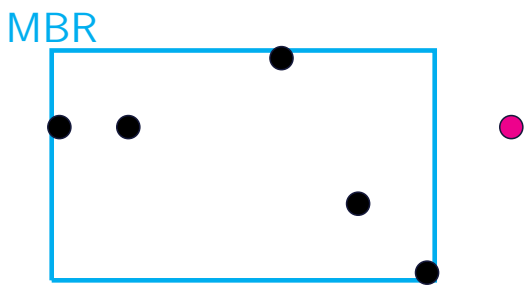
MINDIST wird benutzt, um die Distanz zum am nächsten zu P liegende Objekt in R abzuschätzen.

Suche im R-Baum

- Beim Besuch eines Knotens im R-Baum muss entschieden werden, welches MBR als nächstes besucht werden soll.
- Für jedes MBR betrachten wir MINDIST, da dies eine Näherung der Distanz des nächsten Nachbarn ist.
- Wie gut funktioniert das?



MINDIST (4)

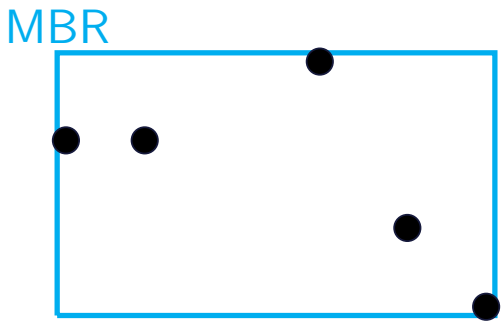


- Schätzung durch MINDIST kann weit daneben liegen
- Was können wir sonst noch über MBRs sagen?

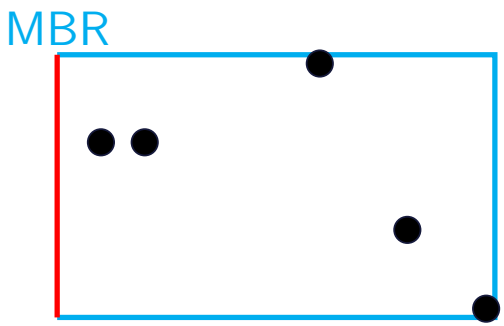
Beobachtung

- Das MBR ist laut Definition das minimale Rechteck, das die enthaltenen Punkte umschließt.
- D.h. auf jeder Kante (für $d = 2$) muss ein Punkt liegen. Für $d = 3$ entsprechend jedes Rechteck, ...
- Wieso? Weil es sonst nicht minimal wäre.
- Rote Kante in Abbildung unten ist z.B. nicht korrekt für ein MBR, müsste "nach rechts verschoben" werden um Rechteck minimal zu machen.

Korrektes MBR (minimal):



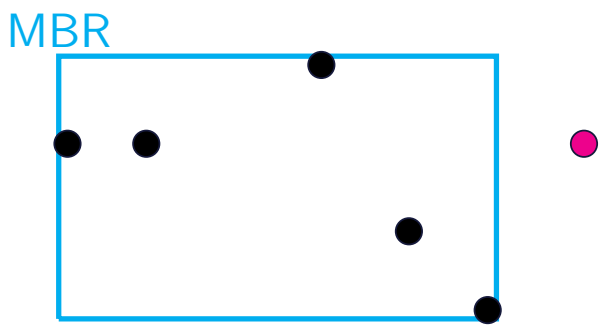
Nicht minimales "MBR":



MINMAXDIST: Idee

Können wir für ein MBR eine obere Schranke für die Distanz des NN Objekts angeben?

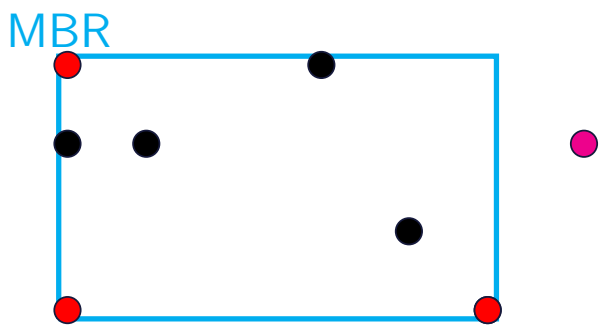
- Wir wissen ja, dass sich auf jeder Kante des MBRs ein Punkt befinden muss.
- Wo liegen diese Punkte maximal weit weg vom Anfragepunkt?



MINMAXDIST: Idee

Können wir für ein MBR eine obere Schranke für die Distanz des NN Objekts angeben?

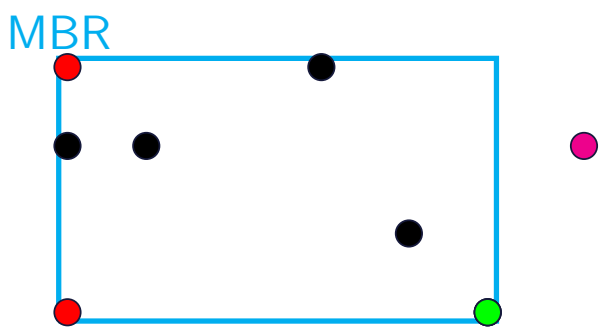
- Wir wissen ja, dass sich auf jeder Kante des MBRs ein Punkt befinden muss.
- Wo liegen diese Punkte maximal weit weg vom Anfragepunkt?



MINMAXDIST: Idee

Können wir für ein MBR eine obere Schranke für die Distanz des NN Objekts angeben?

- Wir wissen ja, dass sich auf jeder Kante des MBRs ein Punkt befinden muss.
- Wo liegen diese Punkte maximal weit weg vom Anfragepunkt?
- Nun betrachten wir den Punkt dieser Punkte mit der kleinsten Distanz zum Anfragepunkt.
- Wieso ist dieser Punkt interessant? Er beschreibt eine obere Schranke für die Distanz des NN. Und zwar die kleinste obere Schranke, die wir angeben können.



MINMAXDIST: Berechnung

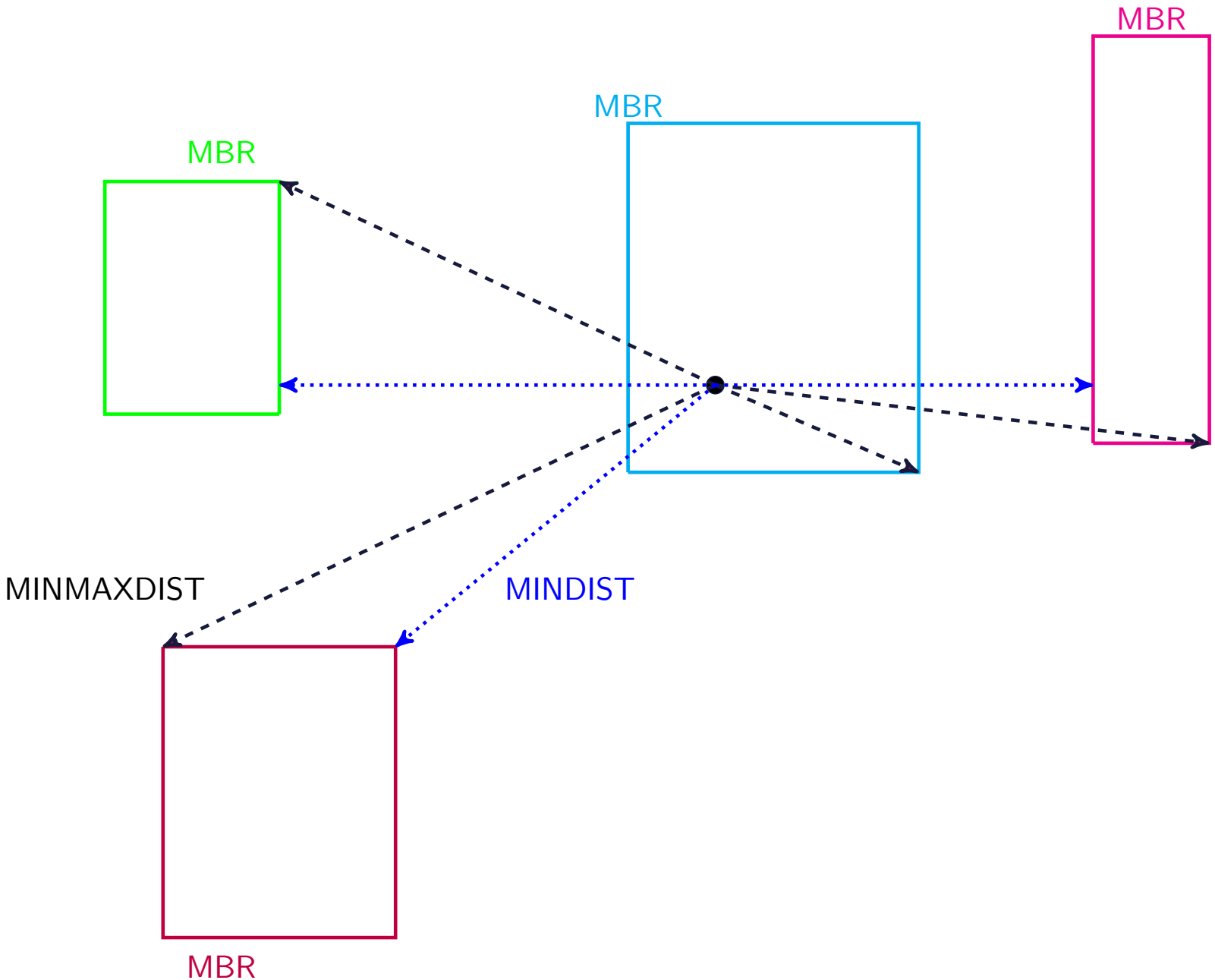
Gegeben ein Punkt P und ein MBR $R = (S,T)$, dann wird **MINMAXDIST(P,R)** berechnet durch

$$\text{MINMAXDIST}(P,R) = min_{1 \leq k \leq d}(|p_k - rm_k|^2 + \sum_{i \neq k, 1 \leq i \leq d} |p_i - rM_i|^2)$$

mit

$$rm_k = \begin{cases} s_k & \text{if } p_k \leq \frac{s_k + t_k}{2} \\ t_k & \text{sonst} \end{cases}$$

$$rM_i = \begin{cases} s_i & \text{if } p_i \geq \frac{s_i + t_i}{2} \\ t_i & \text{sonst} \end{cases}$$


MBR
MBR
MBR
MINMAXDIST
MINDIST
MBR

MINMAXDIST: Interpretation

Gegeben ein Punkt P und ein MBR R welches eine Menge $O = \{o_i\}$ von Objekten umfasst. Dann gilt

$$\exists o \in O \quad ||(P,o)|| \leq MINMAXDIST(P,R)$$

- **Das heißt wir können uns sicher sein, dass es ein Objekt gibt welches auf keinen Fall weiter als MINMAXDIST(P,R) entfernt ist.**

- **Wie können wir die Eigenschaften MINDIST UND MINMAXDIST nun ausnutzen, um einige MBRs gar nicht anschauen zu müssen** (d.h. den Teilbaum nicht betrachten zu müssen)?

Suchraum Pruning

Im Folgenden sind drei Strategien aufgeführt, um die Suche im R-Baum einzuschränken.

Strategie 1

- Ein MBR M mit MINDIST(P, M) größer als die MINMAXDIST(P,M') für ein anderes MBR M' braucht nicht besucht zu werden, da es den NN nicht enthalten kann. **"Downward Pruning"**

Suchraum Pruning (2)

Strategie 2

- Ist die Distanz von P zu einem Objekt O größer als MINMAXDIST(P,M) für ein MBR M, so kann Objekt ignoriert werden, da M ein Objekt O' enthält, das näher an P ist.

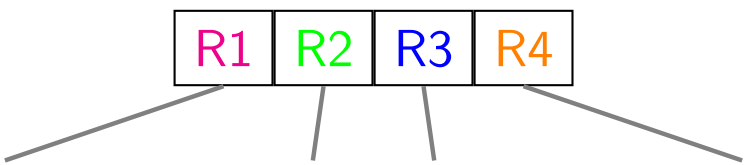
Strategie 3

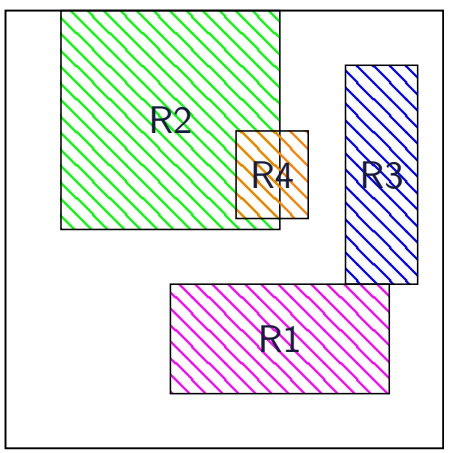
- Jedes MBR M mit MINDIST(P,M) größer als die Distanz von P zu einem Objekt O braucht nicht betrachtet zu werden, da es kein Objekt enthalten kann, das näher an P ist als O. **"Upward Pruning"**

Suche - Ablauf

Die Suche nach dem NN beginnt natürlich an der Wurzel des R-Baums und läuft dann top-down den Baum hinunter.

- Wie in der "normalen" Suche nach überlappenden Rechtecken haben wir hier nun potentiell wieder mehrere Möglichkeiten welche Teilbäume (MBRs!) wir besuchen müssen.
- Wir Suchen hier nach einem Punkt, den nächsten Nachbarn (NN) zum Anfragepunkt
- **Wir sollten hier natürlich so wenig wie möglich MBRs besuchen, aber welche?**





Suchalgorithmus benutzt eine **Prioritäts-Warteschlange** (priority queue). Verschiedene Möglichkeiten der Priorisierung (aka. Sortierung):

Anhand von MINDIST

- MINDIST betrachtet den minimalen Abstand des Anfragepunktes zum MBR
- Dies ist eine **optimistische Priorisierung**. Wie gesehen kann diese Schätzung weit daneben liegen.

Anhand von MINMAXDIST

- MINMAXDIST ist eine **pessimistische Priorisierung**.
- Garantiert immerhin, dass Distanz zu enthaltenen Punkten nicht größer als MINMAXDIST sein kann.

Man kann natürlich Fälle konstruieren wo die eine oder die andere Strategie der Priorisierung besser funktioniert.

Algorithmus: NNSuche

NN.point und NN.dist sind NN bzw. Distanz (initialisiert mit nil und ∞)

Für inneren Knoten (also kein Blatt)

Füge Teilbäume (beschrieben durch MBR) in Queue ein
Sortiere Queue anhand von Priorisierungs-Verfahren
Wende Pruning Strategie 1 und 2 an
while Queue **not empty**
N := Queue.pop
Wende Algorithmus NNSuche rekursiv auf N an
Wende Pruning Strategie 3 an

Für Blattknoten

for each Objekt o **in** Node
dist := $d(q, o)$
if (dist $<$ NN.dist)
NN.dist := dist
NN.point := o