Grundlagen Datenbanken: Übung 10

Tanmay Deshpande

Gruppe 20 & 21

ge94vem@mytum.de

QR-Code für die Folien

Wiederholung

Woche 10

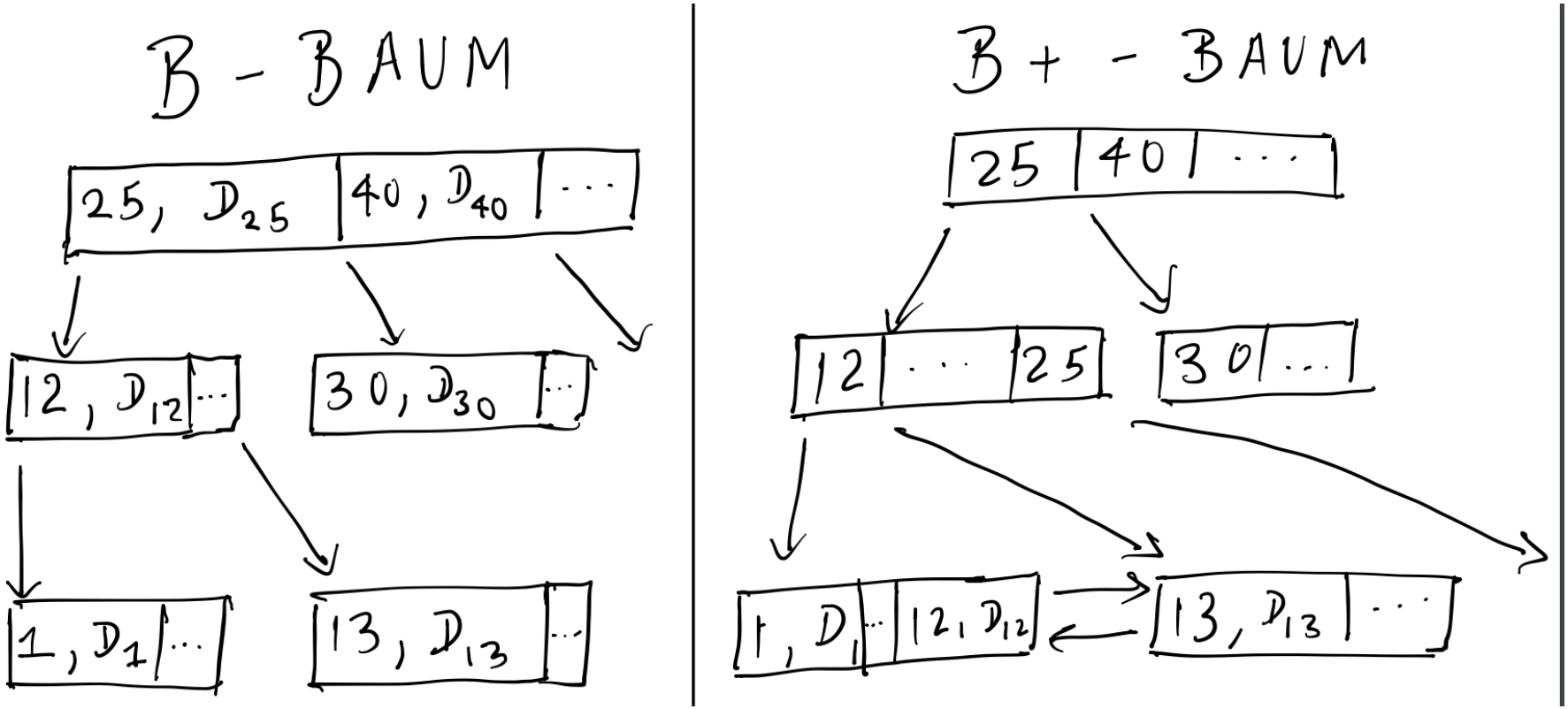
B+ Bäume: Einführung

- Oft wird pro Knoten von B-Bäumen eine Seite im Speicher vergeben
- Je großer die Daten werden, desto kleiner wird *k*
- Anstatt Schlüssel-Daten Paare zu speichern, werden in B+ Bäumen nur Referenzschlüssel in inneren Knoten gespeichert
- Schlüssel-Daten Paare nur in Blätterknoten (als verkettete Liste verwaltet)

Vorteile:

- Da wir nur Schlüssel speichern, passen mehr Elemente in einem Knoten (*k* großer) = Weniger I/O Operationen auf die Festplatte
- Sequenzielle Suche wegen verketteter Liste

B+ Bäume: Beispiel

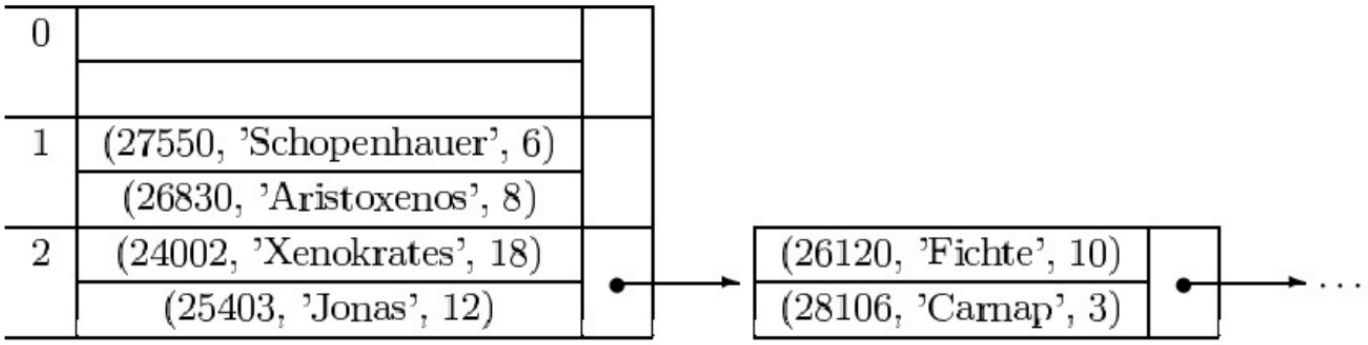


B+ Bäume: Eigenschaften

- Wie B- Bäume, aber anstatt Grad *k* zu haben, sie haben Grad (*k, k**)
- (*k, k**) bedeutet:
  für innere Knoten gilt : $k \leq Anzahl\ Elemente \leq 2k$
  für Blätterknoten gilt : $k^* \leq Anzahl\ Elemente \leq 2k^*$
- Außerdem gilt, doppelte Einträge sind in inneren Knoten erlaubt
- Das linke Kind eines Schlüssels R enthält alle Schlüssel ≤ R
  anstatt nur < R

Statisches Hashing

- Alternative Idee für eine Indexstruktur: anstatt Schlüssel hierarchisch in einem Baum zu speichern, um Suche effizienter zu machen, nutze eine Hashfunktion
- $h(x)$ bildet den Tupel-ID x auf die Seitennummer ab, auf die der Tupel gespeichert ist
- Bei Kollisionen (Seite schon belegt), verweise auf eine andere Seite
- Problem: kann zu linearer Suche werden



$h(x)$
$= x \; mod \; 3$

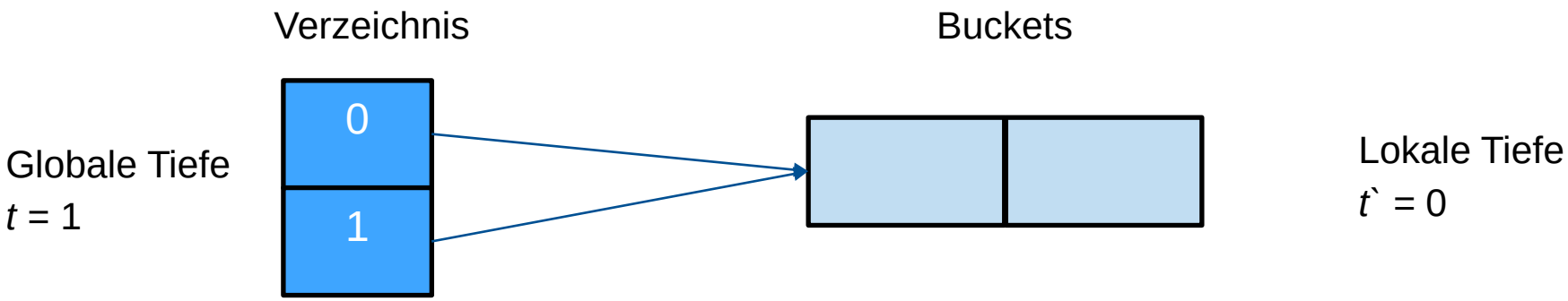
Dynamisches Hashing

- Die Hashfunktion $h(x)$ ordnet jedes x einen Bitstring zu
- Der Bitstring verweist auf einen Bucket
- Buckets haben festgelegte Kapazitäten
- Falls eine Kollision auftritt (der Bucket ist bereits voll) – betrachte mehr Stellen des Bitstrings

Dynamisches Hashing: Beispiel

Nr.	Daten	h(x)
1	D1	1000…
5	D5	1010…
8	D8	0001…
11	D11	1101…

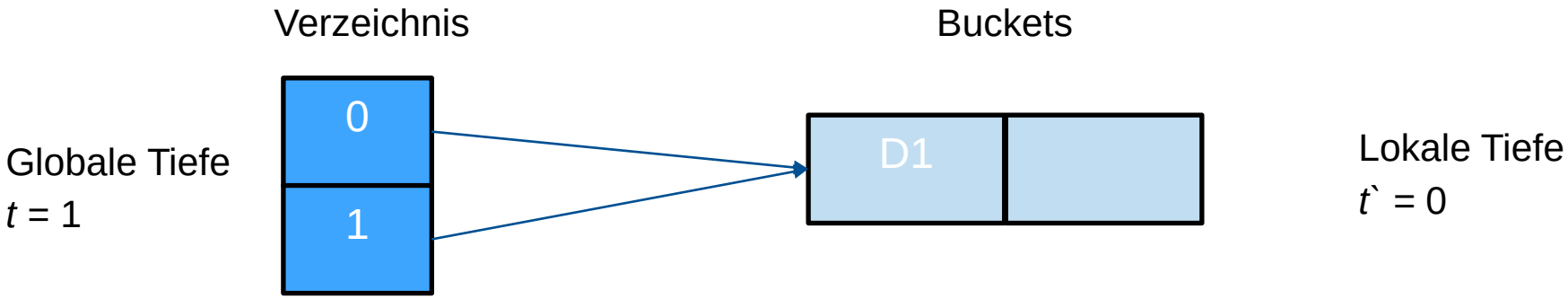
- Wir fügen Schlüssel in ein anfangs leerer erweiterbarer Hashtabelle ein
- Bucketkapazität = 2
- Globale Tiefe = Länge des Bitstrings
- Lokale Tiefe = Bits, die genutzt werden, um Bucket zu bestimmen
- $h(x)$ = Umgedrehte Binärdarstellung der Zahl
- Regel: $t` \leq t$



Dynamisches Hashing: Beispiel

Nr.	Daten	h(x)
1	D1	1000…
5	D5	1010…
8	D8	0001…
11	D11	1101…

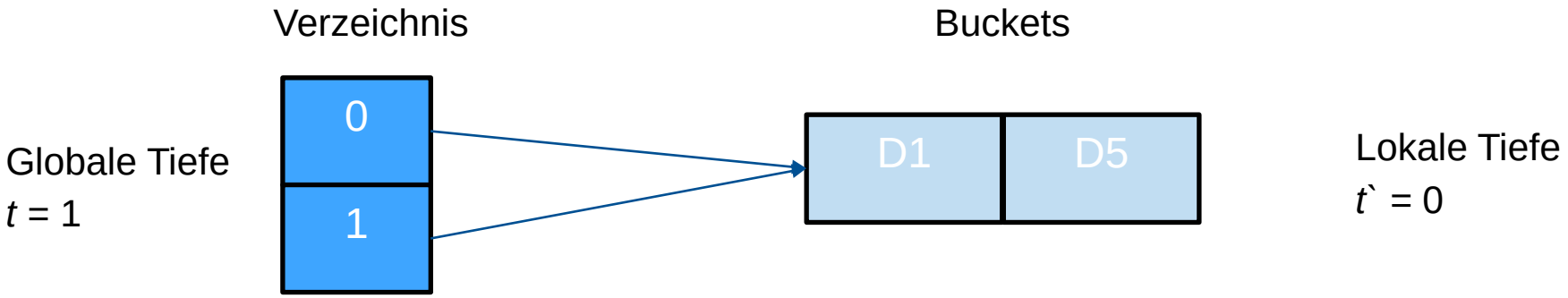
- D1 passt in dem Bucket



Dynamisches Hashing: Beispiel

Nr.	Daten	h(x)
1	D1	1000…
5	D5	1010…
8	D8	0001…
11	D11	1101…

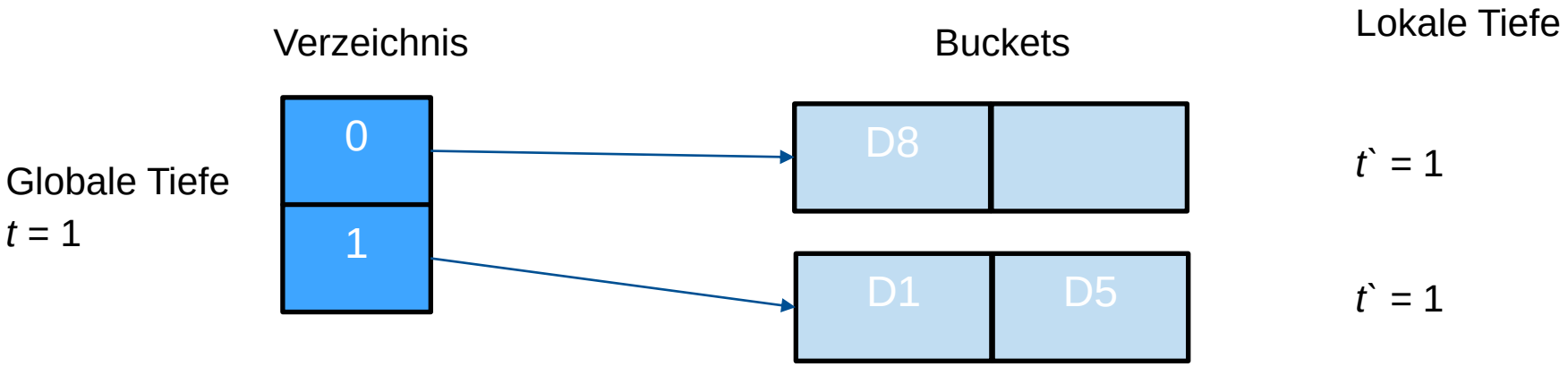
- D5 passt auch



Dynamisches Hashing: Beispiel

Nr.	Daten	h(x)
1	D1	1000…
5	D5	1010…
8	D8	0001…
11	D11	1101…

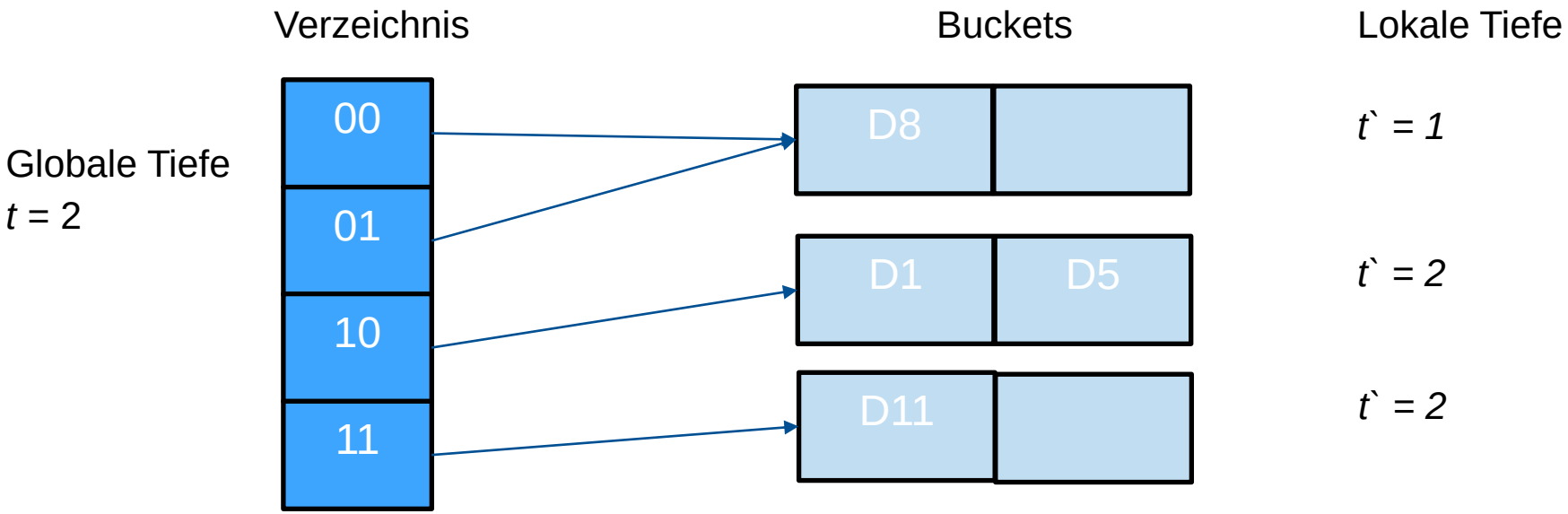
- D8 passt nicht. Der Bucket ist bereits voll! Wir müssen einen neuen Bucket anlegen und die lokale Tiefe erhöhen
- Das dürfen wir machen, da $t` < t$



Dynamisches Hashing: Beispiel

Nr.	Daten	h(x)
1	D1	1000…
5	D5	1010…
8	D8	0001…
11	D11	1101…

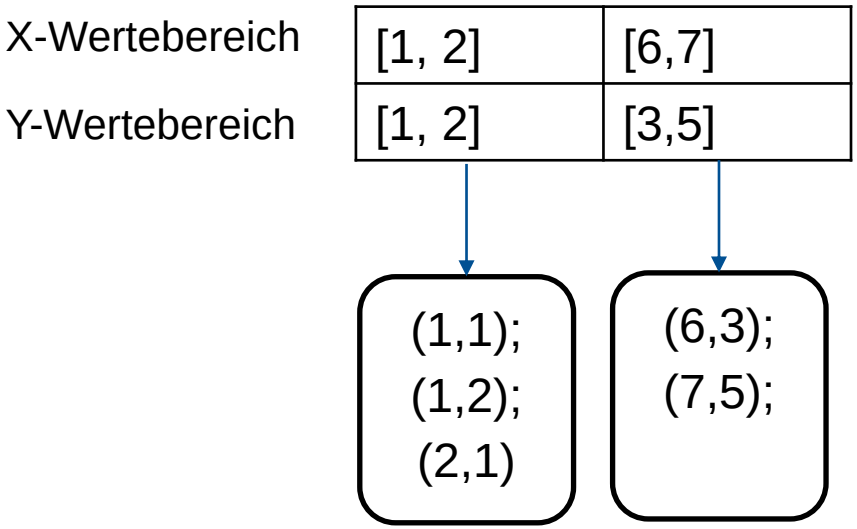
- D9 passt nicht
- Wir müssen einen neuen Bucket anlegen und die lokale Tiefe erhöhen
- Aber dann wäre $t` > t$
- Deshalb erhöhen wir zuerst die globale Tiefe

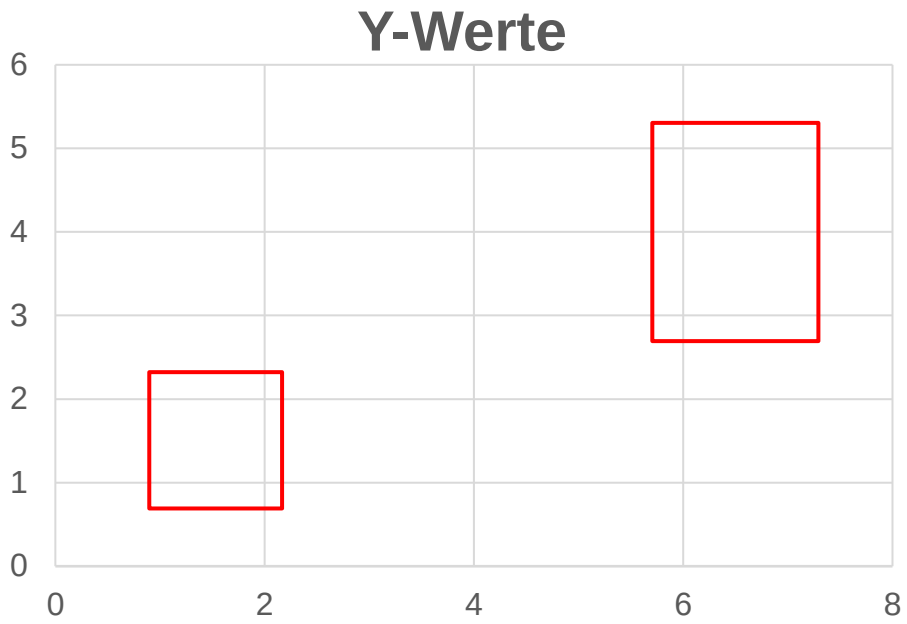


R-Bäume

- Mehrdimensionale Bereichsanfragen sind oft gewünscht
  Bsp: $1 \leq x \leq 5$ und $5 \leq y \leq 10$
- B- und B+- Bäume sind nur eindimensional sortiert
- Bei Hashing sind generell keine Bereichsanfragen möglich
- Lösung:
  Daten in Bereichen unterteilt, die eine feste Anzahl an Punkte speichern können

R-Bäume



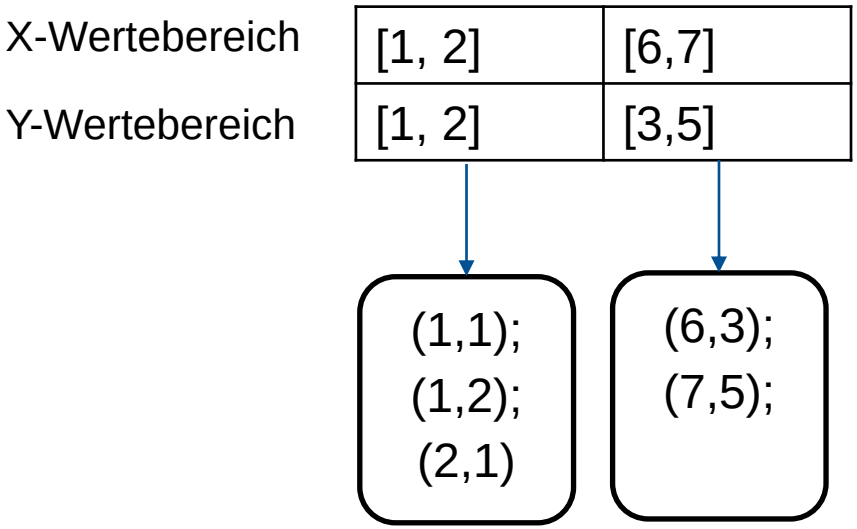


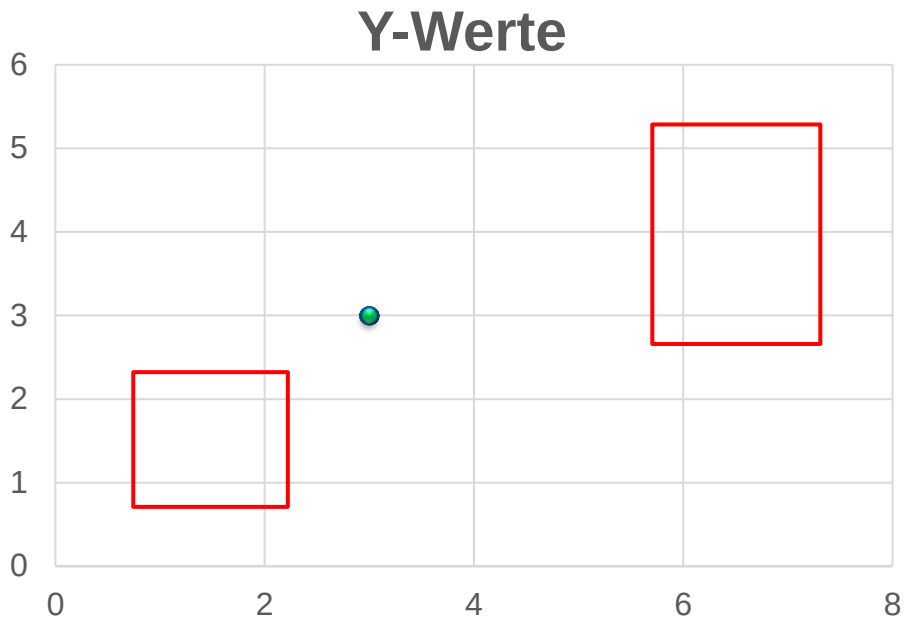
Innere Knoten (eckig): Wertebereiche der verschiedenen Dimensionen. Sie haben einen Nachfolger

Blätterknoten (gerundet): Die eigentliche Werte

In diesem Beispiel, beide haben Kapazität 3

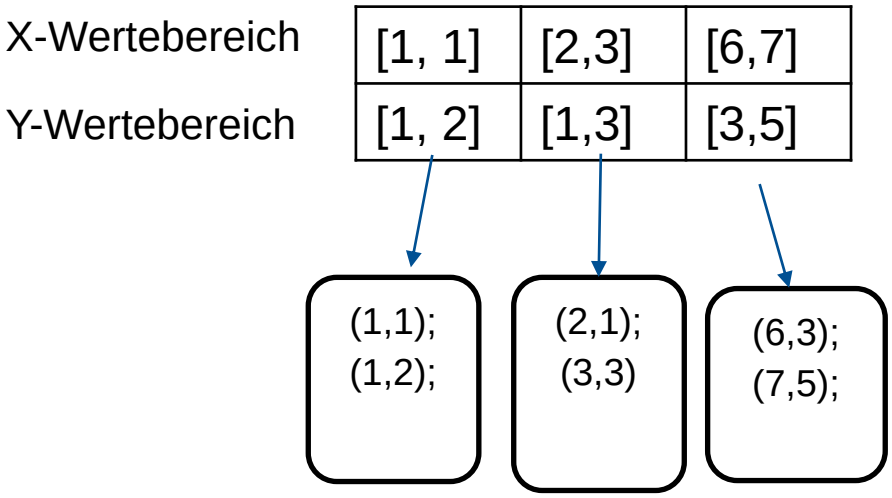
R-Bäume

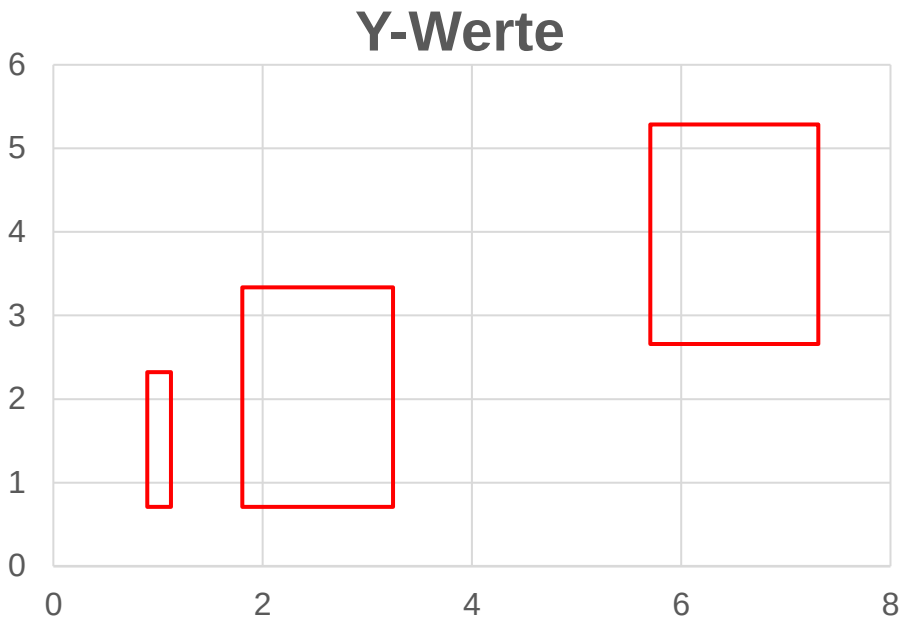




Idee: Wenn ein Box überfüllt wird, spalte in 2 Boxen auf, sodass die Boxen möglichst klein sind

R-Bäume





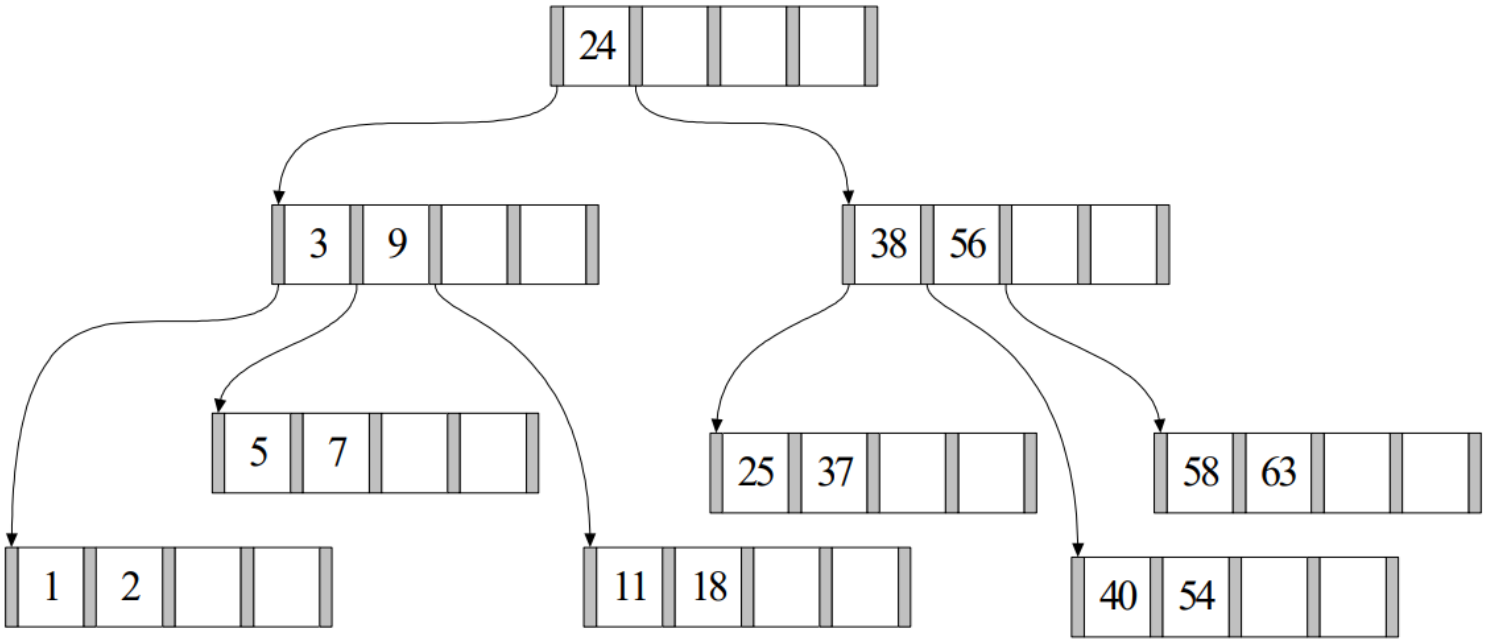
Hinweis: Wenn Wurzel überläuft, wächst es nach oben wie ein B-Baum

Aufgaben

Woche 10

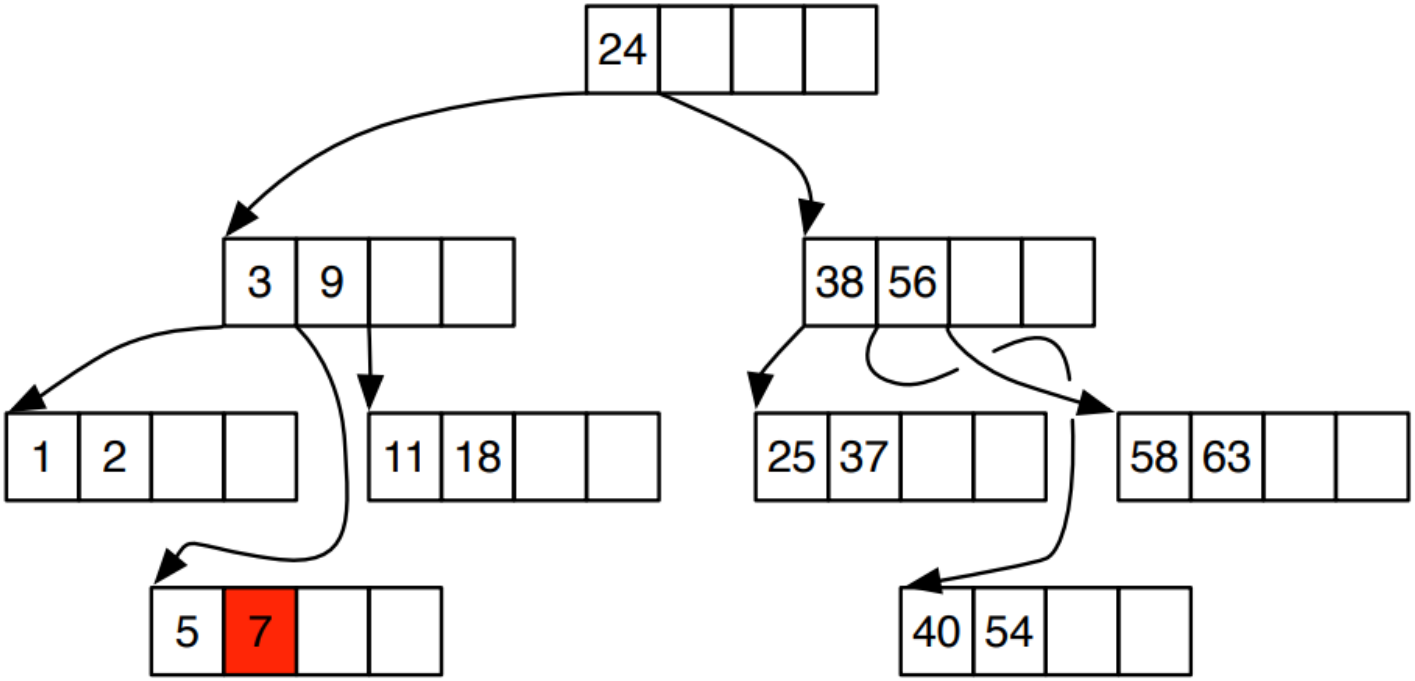
Aufgabe 01

Folgende Ausprägung eines B-Baums mit $k = 2$ sei gegeben:

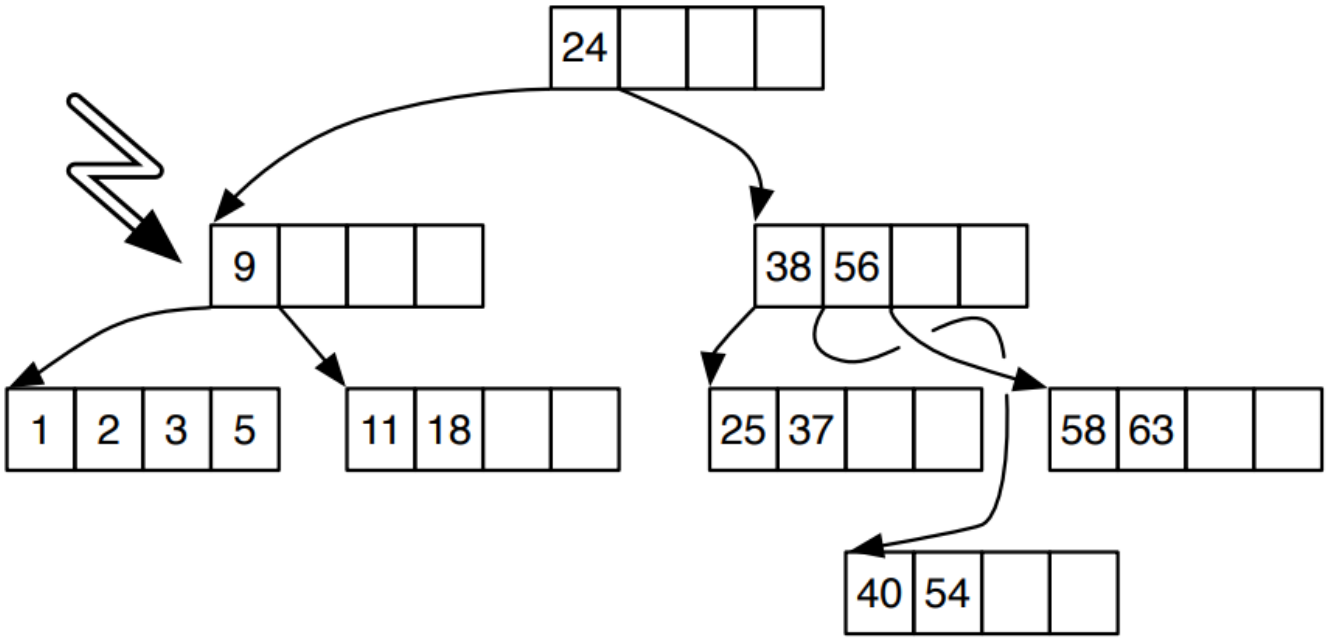


Geben Sie den Baum nach Löschen der 7 an.

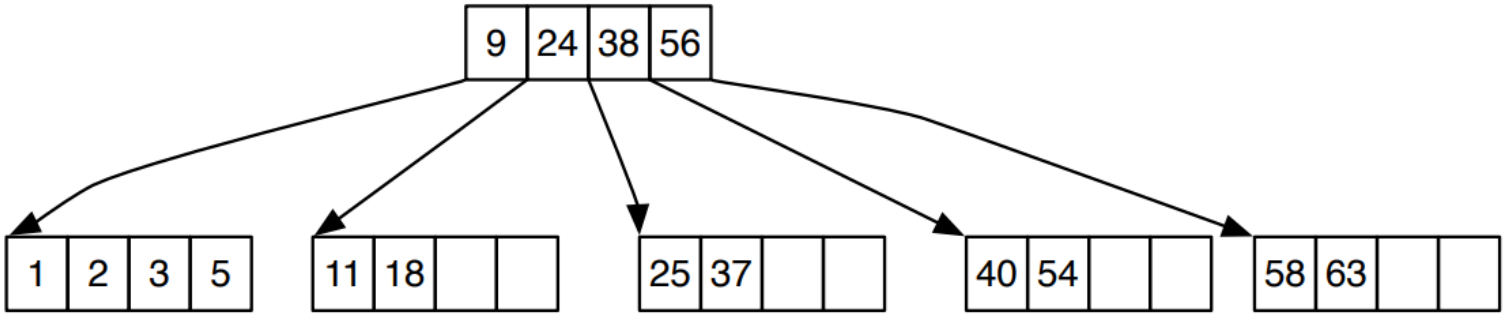
Lösungsvorschlag 01



Lösungsvorschlag 01 (contd…)



Lösungsvorschlag 01 (contd…)



Aufgabe 02

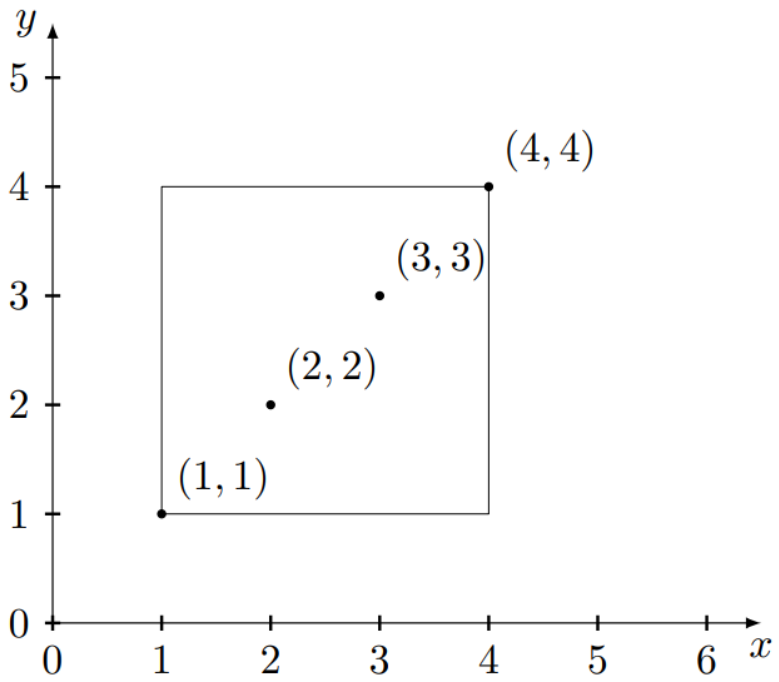
- Fügen Sie in einen anfangs leeren R-Baum mit Knotenkapazität 4 folgende Datenpunkte nacheinander ein:
  (1, 1),(2, 2),(3, 3),(4, 4),(5, 3),(5, 4),(6, 5),(4, 2)

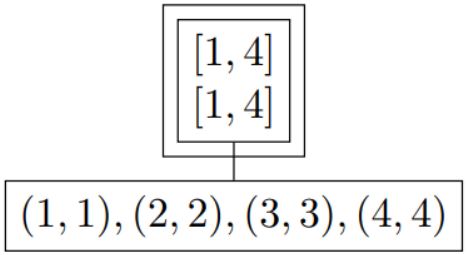
- Splitten Sie die Knoten dabei so, dass die summierte Fläche der durch den Split enstandenen Boxen möglichst klein ist.

- Illustrieren Sie die einzelnen Phasen im Aufbau des R-Baums. Zeichnen Sie hierzu den Baum und den Datenraum unmittelbar vor jedem Split und im Endzustand.

Lösungsvorschlag 02

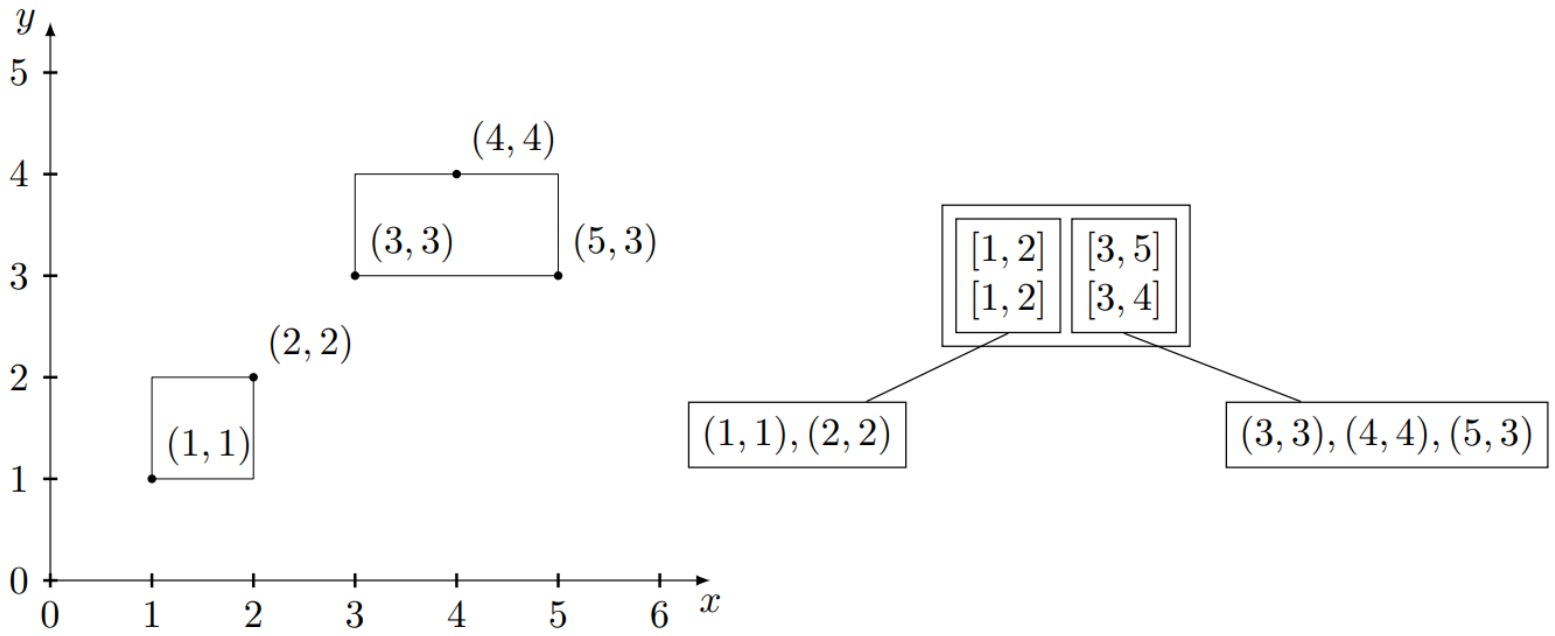
- Wir fügen die Werte (1, 1),(2, 2),(3, 3) und (4, 4) in den ersten Knoten ein.





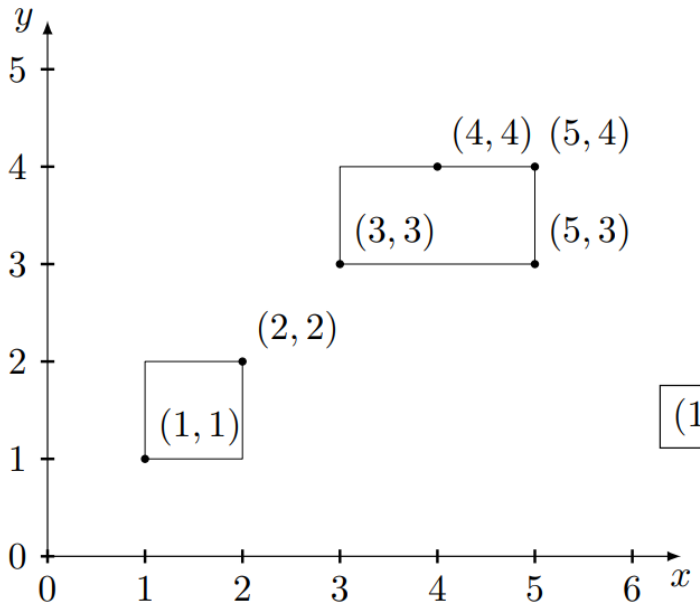
Lösungsvorschlag 02 (contd…)

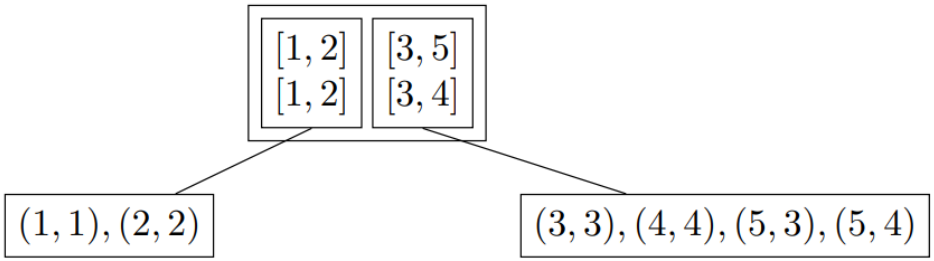
- Da der Knoten nun voll ist, müssen wir ihn für den Wert (5, 3) aufspalten. Um die Gesamtfläche gering zu halten, fügen wir den neuen Wert dann in das zweite Blatt ein



Lösungsvorschlag 02 (contd…)

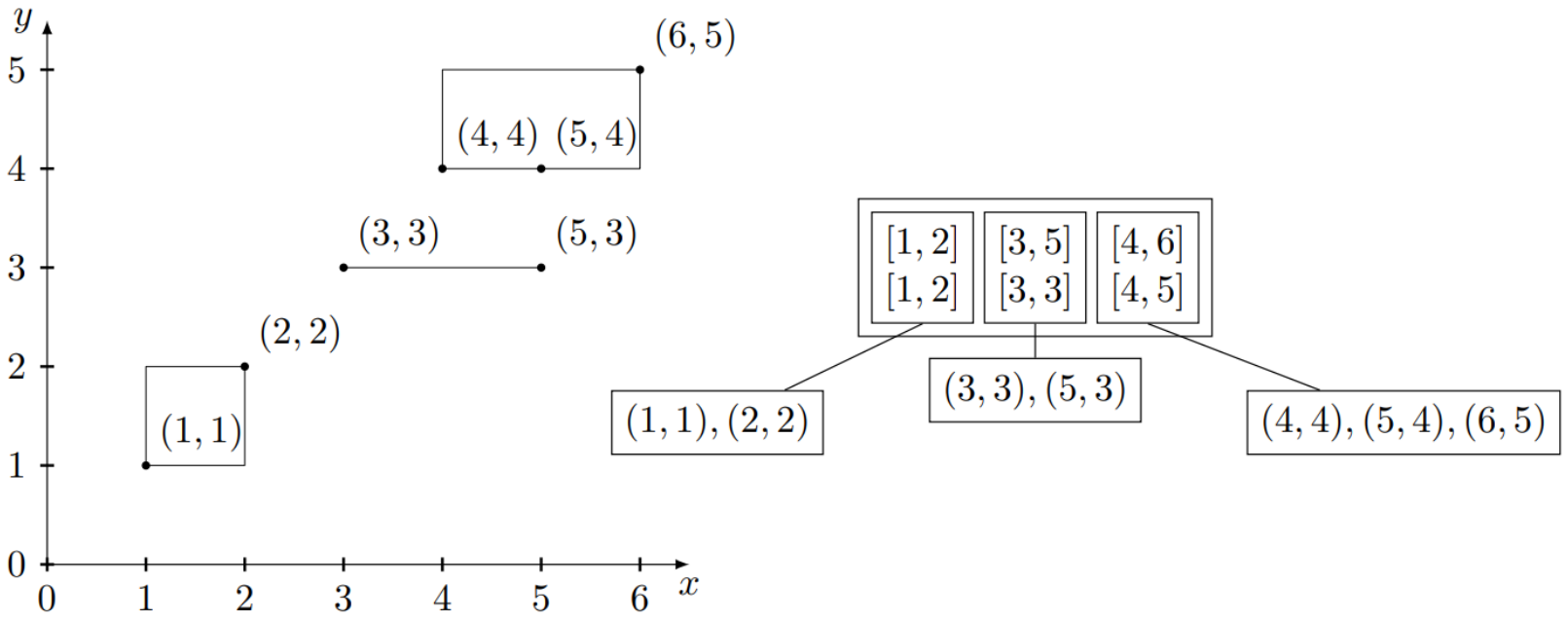
- In dieses Blatt passt auch noch der Wert (5, 4):





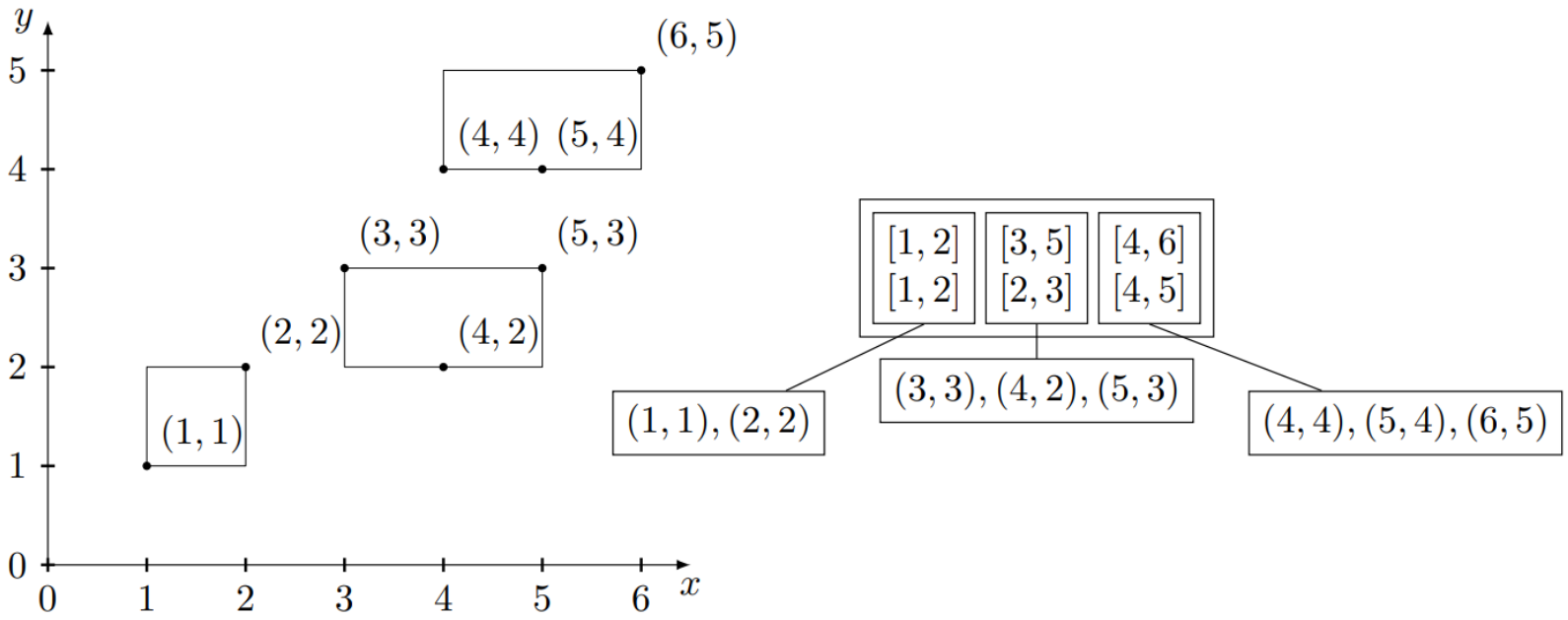
Lösungsvorschlag 02 (contd…)

- Bevor wir (6, 5) einfügen, spalten wir den zweiten Bereich auf



Lösungsvorschlag 02 (contd…)

- (4, 2) könnte entweder im ersten oder zweiten Bereich passen (die Fläche erhöht sich um 2 in beiden Fällen)



Aufgabe 03

- Fügen Sie die folgenden Tupel in eine anfangs leere erweiterbare Hashtabelle, welche 2 Einträge pro Bucket aufnehmen kann, ein. Dabei soll die Matrikelnummer als Suchschlüssel verwendet werden.

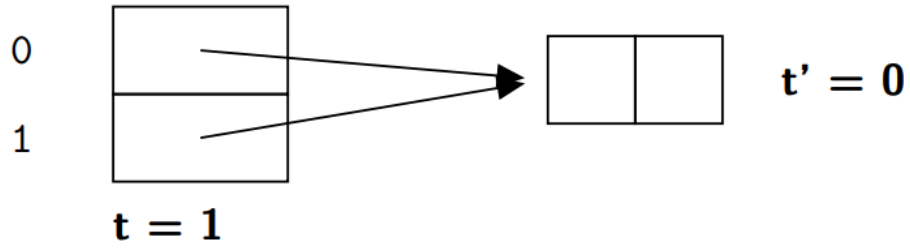
MatrNr	Name
2	Müller
8	Schmidt
19	Fischer
16	Huber
20	Bauer
34	Schneider
30	Wagner

- Verwenden Sie als Hashverfahren die inverse binäre Repräsentation der Matrikelnummer, wie in der Vorlesung beschrieben.

Lösungsvorschlag 03

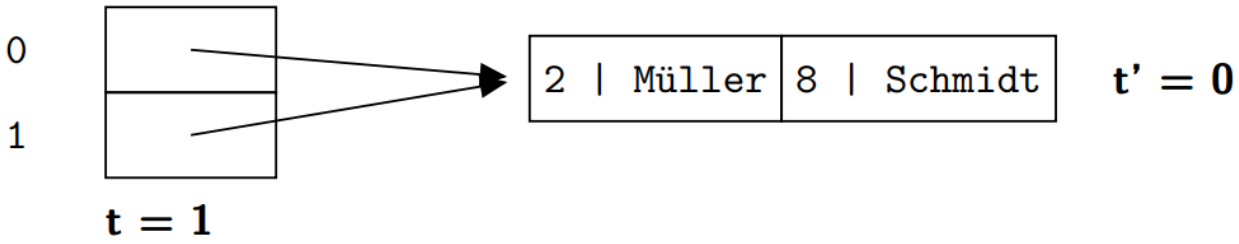
MatrNr	Name	Binär	Invers Binär
2	Müller	000010	010000
8	Schmidt	001000	000100
19	Fischer	010011	110010
16	Huber	010000	000010
20	Bauer	010100	001010
34	Schneider	100010	010001
30	Wagner	011110	011110

Zunächst eine leere erweiterbare Hashtabelle:



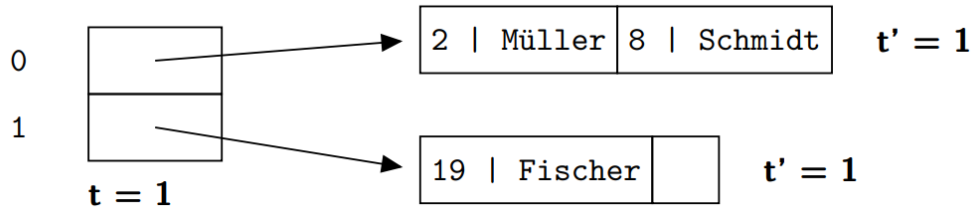
Lösungsvorschlag 03 (contd…)

MatrNr	Name	Binär	Invers Binär
2	Müller	000010	010000
8	Schmidt	001000	000100
19	Fischer	010011	110010
16	Huber	010000	000010
20	Bauer	010100	001010
34	Schneider	100010	010001
30	Wagner	011110	011110



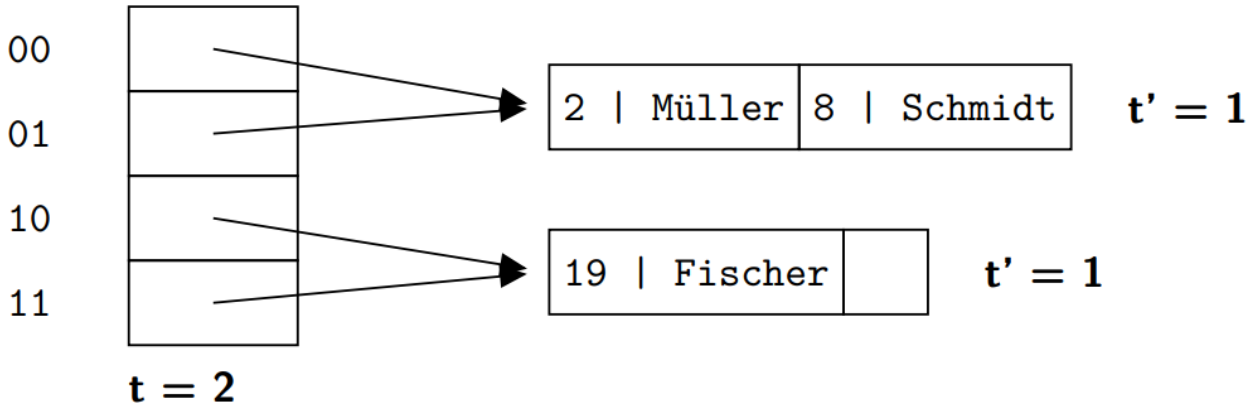
Lösungsvorschlag 03 (contd…)

MatrNr	Name	Binär	Invers Binär
2	Müller	000010	010000
8	Schmidt	001000	000100
19	Fischer	010011	110010
16	Huber	010000	000010
20	Bauer	010100	001010
34	Schneider	100010	010001
30	Wagner	011110	011110



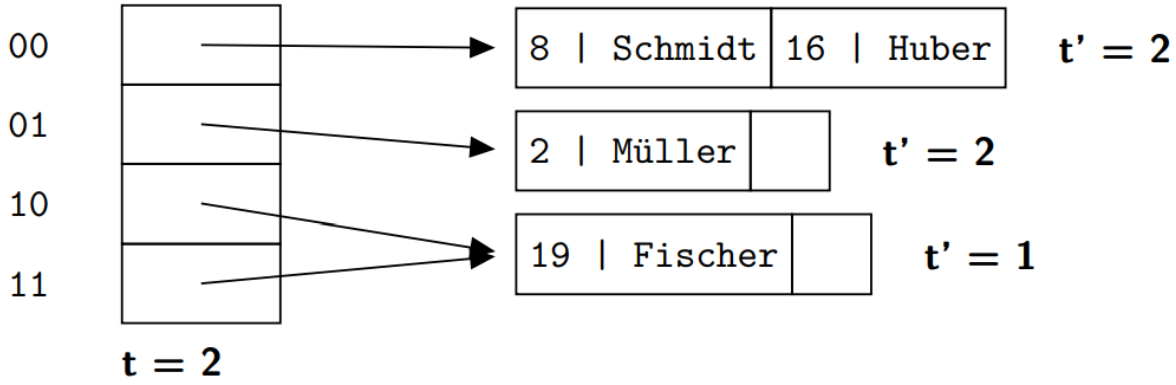
Lösungsvorschlag 03 (contd…)

MatrNr	Name	Binär	Invers Binär
2	Müller	000010	010000
8	Schmidt	001000	000100
19	Fischer	010011	110010
16	Huber	010000	000010
20	Bauer	010100	001010
34	Schneider	100010	010001
30	Wagner	011110	011110



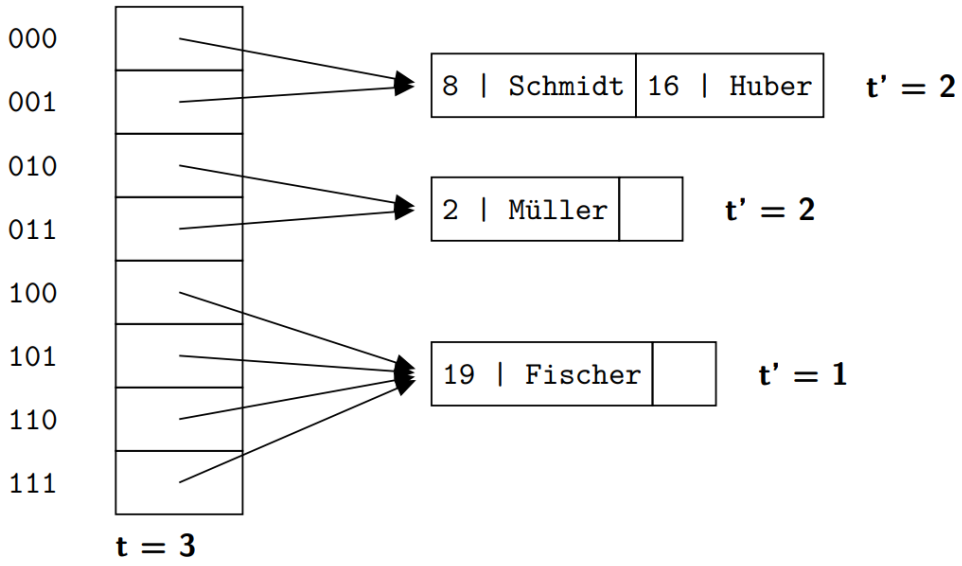
Lösungsvorschlag 03 (contd…)

MatrNr	Name	Binär	Invers Binär
2	Müller	000010	010000
8	Schmidt	001000	000100
19	Fischer	010011	110010
16	Huber	010000	000010
20	Bauer	010100	001010
34	Schneider	100010	010001
30	Wagner	011110	011110



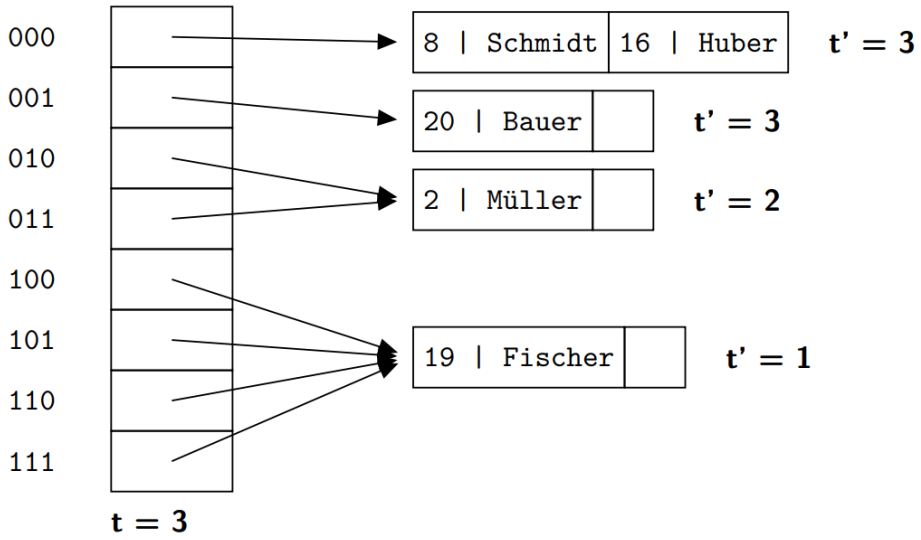
Lösungsvorschlag 03 (contd…)

MatrNr	Name	Binär	Invers Binär
2	Müller	000010	010000
8	Schmidt	001000	000100
19	Fischer	010011	110010
16	Huber	010000	000010
20	Bauer	010100	001010
34	Schneider	100010	010001
30	Wagner	011110	011110



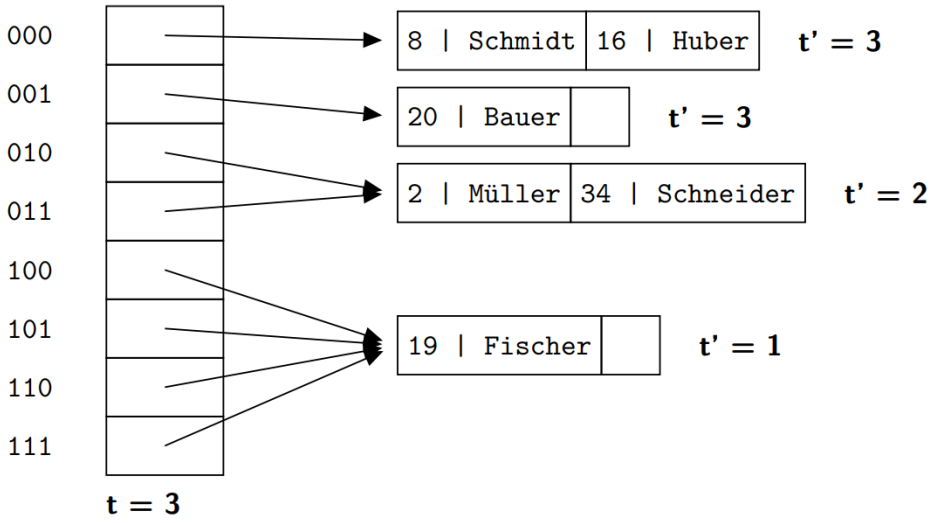
Lösungsvorschlag 03 (contd…)

MatrNr	Name	Binär	Invers Binär
2	Müller	000010	010000
8	Schmidt	001000	000100
19	Fischer	010011	110010
16	Huber	010000	000010
20	Bauer	010100	001010
34	Schneider	100010	010001
30	Wagner	011110	011110



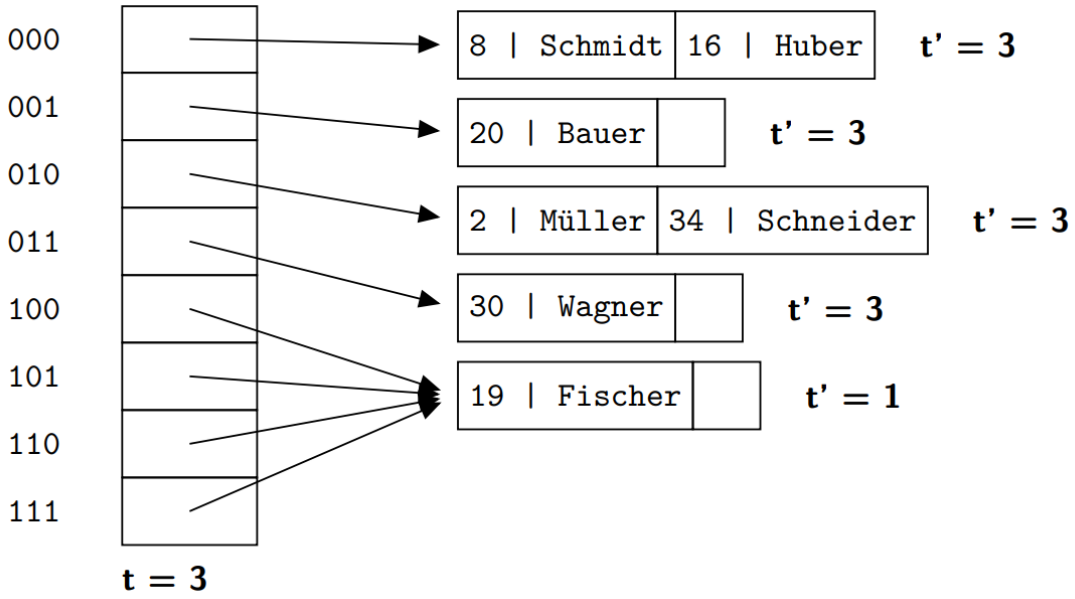
Lösungsvorschlag 03 (contd…)

MatrNr	Name	Binär	Invers Binär
2	Müller	000010	010000
8	Schmidt	001000	000100
19	Fischer	010011	110010
16	Huber	010000	000010
20	Bauer	010100	001010
34	Schneider	100010	010001
30	Wagner	011110	011110



Lösungsvorschlag 03 (contd…)

MatrNr	Name	Binär	Invers Binär
2	Müller	000010	010000
8	Schmidt	001000	000100
19	Fischer	010011	110010
16	Huber	010000	000010
20	Bauer	010100	001010
34	Schneider	100010	010001
30	Wagner	011110	011110



Aufgabe 04

- Gegeben sei eine erweiterbare Hashtabelle mit globaler Tiefe t. Wie viele Verweise zeigen vom Verzeichnis auf einen Behälter mit lokaler Tiefe t`?

Lösungsvorschlag 04

- In dem Verzeichnis einer Hashtabelle mit globaler Tiefe t werden t Bits eines Hashwerts für die Identifizierung eines Verzeichniseintrags verwendet.
- Für einen Behälter mit lokaler Tiefe t` sind hingegen nur die ersten t` Bits dieses Bitmusters relevant.
- Mit anderen Worten bedeutet dies, dass alle Einträge, die einen Behälter mit lokaler Tiefe t` referenzieren, in den ersten t` Bits übereinstimmen.
- Da alle Bitmuster bis zur Länge t in dem Directory aufgeführt sind, unterscheiden sich diese Einträge in den letzten t − t` Bits.
- Es gibt somit $2^{t-t`}$ Einträge im Verzeichnis, die auf denselben Behälter mit lokaler Tiefe t` verweisen.

Aufgabe 05

- Es sollen alle ca. 10 Milliarden Menschen in einer erweiterbaren Hashtabelle verwaltet werden.
- In jede Seite passen ca. 200 Einträge, durchschnittlich sind die Seiten halb voll.
- Je Verweis werden 4 Byte benötigt, da die Musterlösung aus einer Zeit stammt, in der es defakto nur Maschinen mit 32 bit CPU Architektur gab.
- Wie viel Speicherplatz verbraucht das Verzeichnis mindestens?

Lösungsvorschlag 05

- Das Verzeichnis enthält die Verweise auf alle Seiten (= Buckets), in dem die Einträge gehalten werden.
- Da pro Seite durchschnittlich 100 Einträge Platz haben, benötigen wir insgesamt $10^{10}/100 = 10^8$ Seiten. Um 10^8 Seiten zu referenzieren benötigen wir mindestens $\log_2 10^8$ Bits.
- Da dies eine positive ganze Zahl sein muss, ist die Anzahl der benötigten Bits $X = \lceil \log_2 10^8 \rceil$.
- Hiermit können 2^X Verweise im Verzeichnis abgelegt werden, da die Anzahl der Verweise in einem Verzeichnis immer einer 2er-Potenz entspricht.
- Pro Verweis werden 4 Byte benötigt, so dass das Verzeichnis eine Größe von $2^X \cdot$ 4 Byte, also ungefähr 512 MB hat.

Bonusaufgabe 06 (nicht klausurrelevant)

- Implementieren Sie einen B+-Baum in C++. Es sollten mindestens die Funktionen insert und lookup unterstützt werden. Zur Vereinfachung können Sie annehmen, dass lediglich Schlüssel-Wert-Paare bestehend aus Integern eingefügt werden.

- Lösungen an gdb@in.tum.de senden!