Pflichtmodul Informationssysteme (SS 2018)

Prof. Dr. Jens Teubner

Leitung der Übungen: Thomas Lindemann, Tanja Bock

Lösungsskizze zu Übungsblatt Nr. 14

Ausgabe: 11.07.2018 Abgabe: entfällt

Dieses Übungsblatt bildet eine Sammlung von Aufgaben, die teilweise aus alten Klausuren entnommen wurden. Es dient zur Selbstkontrolle und soll einen Überblick über einige der über das Semester hinweg behandelten Themen verschaffen. Diese Themenübersicht sollte auf keinen Fall als Musterklausur betrachtet werden und repräsentiert nicht die komplette Menge klausurrelevanter Themen.

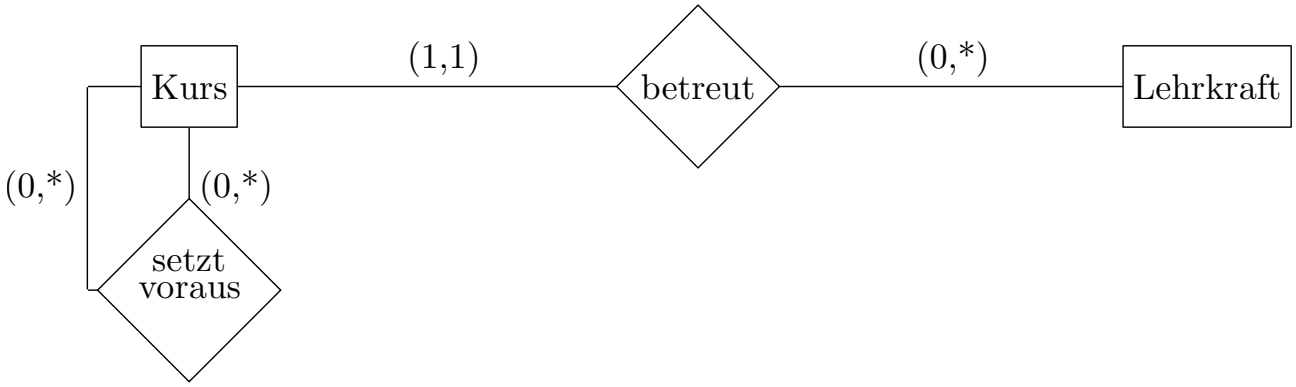
Aufgabe 1 (ER-Modellierung)

In einem Schulungszentrum werden verschiedene Kurse angeboten, wobei ausgewählte Kurse den Inhalt anderer Kurse voraussetzen. Je Kurs gibt es genau eine Lehrkraft, die auch verschiedene Kurse betreuen kann.

1. Erstellen Sie für diesen Diskursbereich ein ER-Diagramm. Verwenden Sie für die Funktionalität der Beziehungstypen die Min-/Max-Notation.
2. Überführen Sie das erhaltene ER-Diagramm in Tabellen des Relationalen Modells und geben Sie Primär- (*primary*) und Fremdschlüsselbedingungen (*foreign keys*) an. Überlegen Sie sich dazu geeignete Attributnamen.

Lösung:

1. ER-Diagramm:



2. Da keine Attribute vorgegeben sind, ergänzen wir selbst die für die Schlüsselbeziehungen nötigen Attribute. Des weiteren ergänzen wir einige Attribute, die die Nutzung der Datensätze einfacher gestalten sollen. Ein Kurs wird durch seine Nummer identifiziert und hat eine Bezeichnung. Eine Lehrkraft wird durch ihre Personalnummer identifiziert und hat einen Namen. Die Relation “setzt voraus” wird über das Paar identifiziert, welches

sich aus den Schlüsselattributen der beiden teilnehmenden Entitäten zusammensetzt. Die Relation "betreut" wird durch die Nummer des Kurses identifiziert (dies reicht aufgrund der Kardinalitätsbeschränkung):

In den Tabellen wird folgende Notation verwendet: Die unterstrichenen Attribute bilden jeweils (zusammengenommen im Falle von SetztVoraus) einen Schlüssel für die entsprechende Relation (Tabelle):

(a) Entitäten:

Kurse	
Kursnummer	Bezeichnung
⋮	⋮

Lehrkräfte	
Personalnummer	Name
⋮	⋮

(b) Relationen:

SetztVoraus	
Kursnummer	Kursnummer2
⋮	⋮

Betreut	
Kursnummer	Personalnummer
⋮	⋮

Hierbei sind Kursnummer und Kursnummer2 Fremdschlüssel, die auf das entsprechende Attribut in Kurse verweisen. Personalnummer ist ein Fremdschlüssel, der auf das entsprechende Attribut in Lehrkräfte verweist.

Aufgabe 2 (Tabellen erzeugen)

Ein Sportverein will eine Datenbank mit folgendem Schema aufbauen:

sch(Mitglieder)=(MitglNr, Name, Alter, Abteilung)
sch(Beiträge)=(Alter, Betrag)
sch(Zahlungen)=(Datum, MitglNr, Betrag)

In *Beiträge* werden die Mitgliedsbeiträge festgehalten, die vom Alter des Mitglieds abhängig sind. Manche Mitglieder bezahlen ihre Beiträge in Raten, daher sind in *Zahlungen* alle bislang eingegangenen Beträge vermerkt.

1. Geben Sie die SQL-Anweisungen an, mit denen das obige Relationen-Schema erzeugt wird. Spezifizieren Sie dabei sämtliche Primär- und Fremdschlüsselbeziehungen.
2. Trainern soll die Möglichkeit gegeben werden, von allen Mitgliedern *Name* und *Alter* abzufragen. Geben Sie SQL-Kommandos an, die zur Erstellung einer View, die genau diese Information enthält, nötig sind.

Lösung:

1. SQL-Anweisungen:

```
CREATE TABLE Mitglieder ( MitglNr INTEGER NOT NULL,
                          Name CHAR(30),
                          Alter INTEGER,
                          Abteilung CHAR(30),
                          PRIMARY KEY (MitglNr)
)
```

```
CREATE TABLE Beitraege ( Alter INTEGER NOT NULL,
                         Betrag INTEGER,
                         PRIMARY KEY (Alter)
)

CREATE TABLE Zahlungen ( Datum DATE NOT NULL,
                         MitglNr INTEGER NOT NULL,
                         Betrag INTEGER,
                         PRIMARY KEY (Datum, MitglNr),
                         FOREIGN KEY (MitglNr) REFERENCES Mitglieder
)
```

2. View:

```
CREATE VIEW NameUndAlter AS SELECT Name, Alter FROM Mitglieder;
```

Aufgabe 3 (Anfragesprachen)

Gegeben sei das Relationen-Schema aus Aufgabe 2:

sch(Mitglieder)=(MitglNr, Name, Alter, Abteilung)
sch(Beiträge)=(Alter, Betrag)
sch(Zahlungen)=(Datum, MitglNr, Betrag)

Formulieren Sie die folgende Anfrage in **Tupel-Relationen-Kalkül (*tuple relational calculu*) und SQL**. Verwenden Sie `DISTINCT` genau dann, wenn im Ergebnis tatsächlich Duplikate auftreten können.

1. Geben Sie *Name*, *MitglNr* und *Betrag* der fälligen Beiträge aller Mitglieder aus der Abteilung 'Fußball' aus.

   Lösung:
   $\{t | \exists m : m \in Mitglieder \wedge \exists b : b \in Beitraege \wedge m.Alter = b.Alter \wedge$
   $m.Abteilung = `Fussball' \wedge t \leftarrow \langle m.Name, m.MitglNr, b.Betrag \rangle\}$

```
SELECT m.Name, m.MitglNr, b.Betrag
FROM Mitglieder m, Beitraege b
WHERE m.Alter = b.Alter AND m.Abteilung = 'Fussball';
```

Formulieren Sie die folgenden beiden Anfragen in **Relationen-Algebra und SQL**. Verwenden Sie auch hier `DISTINCT` genau dann, wenn im Ergebnis tatsächlich Duplikate auftreten können.

2. Welche Mitglieder haben bislang noch gar nichts bezahlt? Geben Sie *MitglNr*, *Name*, *Alter* und den fälligen *Betrag* aus.

   Lösung:
   $\Big(\pi_{MitglNr,Name,Alter}(Mitglieder) - \pi_{MitglNr,Name,Alter}(Mitglieder \bowtie Zahlungen)\Big)$
   $\bowtie$
   $Beitraege$

```
SELECT m.MitglNr, m.Name, m.Alter, b.Betrag
FROM Mitglieder m, Beitraege b
WHERE m.Alter = b.Alter
AND NOT EXISTS (
  SELECT *
  FROM Zahlungen AS z
  WHERE z.MitglNr = m.MitglNr
);
```

3. Geben Sie von den ältesten Mitgliedern *MitglNr* und *Name* aus.

   Lösung:
   Self-Join zwischen Mitglieder, wobei linkes Alter größer ist als rechtes. Damit sind alle, die rechts stehen, nicht die ältesten. Wir ziehen sie daher von der Gesamt-Menge an Mitgliedern ab und es bleiben die ältesten. Der erneute Join mit Mitglieder ergibt die passenden Attribute.

$$\pi_{MitglNr}(Mitglieder) - \pi_{MitglNr \leftarrow m_2}($$
$$\sigma_{a_1 > a_2}($$
$$\pi_{m_1 \leftarrow MitglNr, a_1 \leftarrow Alter}(Mitglieder) \times \pi_{m_2 \leftarrow MitglNr, a_2 \leftarrow Alter}(Mitglieder)$$
$$)$$
$$)$$
$$\bowtie$$
$$\pi_{MitglNr,Name}(Mitglieder)$$

   Alternativlösung: Die Idee ist es, zuerst Mitgliedsnummer und Name aller Mitglieder zu berechnen, die nicht ältestes Mitglied des Vereins sind (ähnlich wie oben). Dann zieht man diese Tupel von der Menge aller Tupel von Mitgliedsnummern und Namen ab. Man kriegt dann:

$$\pi_{MitglNr,Name}(Mitglieder) - \pi_{MitglNr \leftarrow MN_2,Name}(Exp)$$

   Hierbei gelte:

$$Exp = \sigma_{a_1 > a_2}(\pi_{a_1 \leftarrow Alter}(Mitglieder) \times \pi_{MN_2 \leftarrow MitglNr, a_2 \leftarrow Alter,Name}(Mitglieder))$$

```
SELECT MitglNr, Name
FROM Mitglieder m
WHERE NOT EXISTS (
  SELECT *
  FROM Mitglieder aelter
  WHERE aelter.Alter > m.Alter
);
```

Formulieren Sie die folgenden zwei Anfragen nur in **SQL**:

4. Welche *Abteilung* hat die meisten Mitglieder, die bisher noch nichts gezahlt haben?

   Lösung:

```
SELECT m.Abteilung
FROM Mitglieder m WHERE NOT EXISTS (
   SELECT * FROM Zahlungen z1 WHERE z1.MitglNr=m.MitglNr)
GROUP by m.Abteilung HAVING COUNT(*) >= ALL(
   SELECT COUNT(*)
   FROM Mitglieder m1 WHERE NOT EXISTS (
      SELECT * FROM Zahlungen z WHERE z.MitglNr = m1.MitglNr)
   GROUP BY m1.Abteilung);
```

Idee: Wähle die Abteilungen, wo die Anzahl der Mitglieder, die noch nichts gezahlt haben (das sind die, deren Mitgliedsnummer nicht in der Tabelle Zahlungen auftaucht), größer oder gleich der entsprechenden Anzahl jeder einzelnen Abteilung ist.

Eine übersichtlichere Lösung basierend auf Views:

```
CREATE VIEW N AS
  SELECT M.Abteilung
  FROM Mitglieder M
  WHERE NOT EXISTS (
    SELECT *
    FROM Zahlungen Z
    WHERE M.MitglNr = Z.MitglNr
  );


SELECT Abteilung
FROM N
GROUP BY Abteilung
HAVING COUNT(*) >= ALL (
  SELECT COUNT(*)
  FROM N
  GROUP BY Abteilung
);
```

5. Geben Sie für alle Mitglieder, die noch nicht vollständig bezahlt (unter Einbeziehung aller bereits geleisteten Zahlungen) haben, *Name* und den noch verbleibenden Betrag aus.

   Lösung:

```
SELECT m.Name, b.Betrag - g.Gesamt AS Verbleibend
FROM Mitglieder m,
     Beitraege b,
     (SELECT MitglNr, SUM(Betrag) AS Gesamt
      FROM Zahlungen GROUP BY MitglNr) g
WHERE g.MitglNr = m.MitglNr AND
      b.Betrag > g.Gesamt AND
      b.Alter=m.Alter;
```

Aufgabe 4 (Schemanormalisierung)

Gegeben seien das Relationenschema

$$sch(R) = (VWXYZ)$$

sowie die zugehörige Menge

$$\mathcal{F} = \{Z \to X, \quad V \to W, \quad X \to YV, \quad W \to V\}$$

von funktionalen Abhängigkeiten.

1. Geben sie einen Schlüssel für sch(R) an!
2. **Zerlegen Sie R mit Hilfe des BCNF-Zerlegungsalgorithmus (*BCNF decomposition algorithm*) aus der Vorlesung, sodass alle resultierenden Tabellen in BCNF vorliegen.**
   - ▷ Geben Sie vor jedem Durchlauf der **while**-Schleife sowie nach Ablauf des Algorithmus die Schemata aller erzeugten Tabellen an. Listen Sie dabei auch alle zugehörigen Funktionalen Abhängigkeiten (*functional dependencies*) auf.

Lösung: Wir haben $sch(R) = VWXYZ$ und $\mathcal{F} = \{Z \to X, V \to W, X \to YV, W \to V\}$. Ein Schlüssel für R ist Z, da durch $\{Z \to X\}$ X abgeleitet werden kann, gefolgt von YV durch $\{X \to YV\}$, und W durch $\{V \to W\}$. Da immer $\{Z \to Z\}$ gilt, kann durch Kenntnis von Z jedes Attribut in $sch(R)$ abgeleitet werden, woraus folgt das Z ein Schlüssel ist.

Die drei funktionalen Abhängigkeiten $V \to W, X \to YV$ und $W \to V$ verletzen die BCNF-Eigenschaft. Zur Lösung der Aufgabe genügt es, eine Menge von Schemata in BCNF anzugeben. Im folgenden sind alle Möglichkeiten, den Nichtdeterminismus aufzulösen, aufgeführt.

1. Entlang $V \to W$ zerlegen. Es ergibt sich die Menge:

   $$\{(VW, \{V \to W, W \to V\}^+), (VXZY, \{X \to YV, Z \to X\}^+)\}$$

   Beim zweiten Schema verletzt die Abhängigkeit $X \to YV$ die Bedingung.
   Weiter entlang $X \to YV$ zerlegen. Es ergibt sich die Menge:

   $$\{(VW, \{V \to W, W \to V\}^+), (XYV, \{X \to YV\}^+), (ZX, \{Z \to X\}^+)\}$$

   Alle Schemata sind in BCNF

2. Entlang $W \to V$ zerlegen. Es ergibt sich die Menge:

   $$\{(WV, \{V \to W, W \to V\}^+), (WXZY, \{X \to WY, Z \to WXY\}^+)\}$$

   Beim zweiten Schema verletzt die Abhängigkeit $X \to WY$ die Bedingung.
   Weiter entlang $X \to WY$ zerlegen. Es ergibt sich die Menge:

   $$\{(WV, \{V \to W, W \to V\}^+), (XWY, \{X \to WY\}^+), (ZX, \{Z \to X\}^+)\}$$

   Alle Schemata sind in BCNF.

3. Entlang $X \to YV$ zerlegen. Es ergibt sich die Menge:

$$\{(XYV, \{X \to YV\}^+), (WXZ, \{Z \to XW, X \to W\}^+)\}$$

Beim zweiten Schema verletzt die Abhängigkeit $X \to W$ die Bedingung.
Entlang $X \to W$ zerlegen. Es ergibt sich die Menge:

$$\{(XYV, \{X \to YV\}^+), (XW, \{X \to W\}^+), (XZ, \{Z \to X\}^+)\}$$

Alle Schemata sind in BCNF.

Aufgabe 5 (Transaktionskontrolle)

Untersuchen Sie die folgenden Schedules auf Serialisierbarkeit. Geben Sie dazu den vollständigen Abhängigkeitsgraphen (*conflict graph / serialization graph*) an und beschriften Sie die Kanten mit allen vorkommenden Konflikten. Geben Sie, falls möglich, einen äquivalenten seriellen Schedule an.

(a) $\langle r_2(x), r_3(y), w_2(x), r_4(x), w_3(z), r_1(z), w_1(z), r_4(z), w_4(x), r_2(y), w_4(z), w_2(y)\rangle$

(b) $\langle r_3(u), r_2(v), w_3(u), r_2(u), r_1(w), w_2(v), r_1(v), r_3(u), r_3(w), r_2(u), w_3(w), w_2(u)\rangle$

Lösung:

1. $\langle r_2(x), r_3(y), w_2(x), r_4(x), w_3(z), r_1(z), w_1(z), r_4(z), w_4(x), r_2(y), w_4(z), w_2(y)\rangle$

Konfliktrelation:

$r_2(x) \prec w_4(x)$ Label für Kante 1
$r_3(y) \prec w_2(y)$ Label für Kante 2
$w_2(x) \prec r_4(x)$ Label für Kante 1
$w_2(x) \prec w_4(x)$ Label für Kante 1
$w_3(z) \prec r_1(z)$ Label für Kante 3
$w_3(z) \prec w_1(z)$ Label für Kante 3
$w_3(z) \prec r_4(z)$ Label für Kante 4
$w_3(z) \prec w_4(z)$ Label für Kante 4
$r_1(z) \prec w_4(z)$ Label für Kante 5
$w_1(z) \prec r_4(z)$ Label für Kante 5
$w_1(z) \prec w_4(z)$ Label für Kante 5

Konfliktgraph:

- $T_2 \to T_4$ (Kante 1)
- $T_3 \to T_2$ (Kante 2)
- $T_3 \to T_1$ (Kante 3)
- $T_3 \to T_4$ (Kante 4)
- $T_1 \to T_4$ (Kante 5)

Der Graph ist azyklisch. Also gibt es konflikt-äquivalente serielle Ausführung, zB: $\langle T_3, T_1, T_2, T_4\rangle$.

2. $\langle r_3(u), r_2(v), w_3(u), r_2(u), r_1(w), w_2(v), r_1(v), r_3(u), r_3(w), r_2(u), w_3(w), w_2(u)\rangle$

   Konfliktrelation:

   $r_3(u) \prec w_2(u)$ Label für Kante 1
   $w_3(u) \prec r_2(u)$ Label für Kante 1
   $w_3(u) \prec w_2(u)$ Label für Kante 1
   $r_1(w) \prec w_3(w)$ Label für Kante 2
   $w_2(v) \prec r_1(v)$ Label für Kante 3
   r3(u) < w2(u)

   Konfliktgraph:

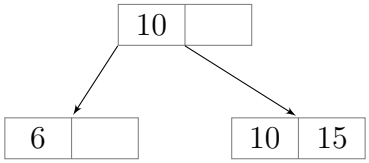
   - $T_3 \to T_2$ (Kante 1)
   - $T_1 \to T_3$ (Kante 2)
   - $T_2 \to T_1$ (Kante 3)

   Der Graph hat einen Zyklus. Also ist der Schedule nicht konflikt-serialisierbar.

Aufgabe 6 (B+ Indexstrukturen)

Fahrer			
fahrer_id	name	team	nation
15	Antonio Pizzonia	Jaguar-Racing	Brasilien
6	Kimi Räikkönen	McLaren-Mercedes	Finnland
10	Heinz-Harald Frentzen	Sauber-Petronas	Deutschland
3	Juan Pablo Montoya	Williams-BMW	Kolumbien
9	Nick Heidfeld	Sauber-Petronas	Deutschland
2	Rubens Baricello	Ferrari	Brasilien
4	Ralf Schumacher	Williams-BMW	Deutschland
17	Jacques Villeneuve	BAR-Honda	Kanada

Der Betreiber einer Formel-Eins-Informationsseite im Internet will die Daten aller Fahrer erfassen. Ein Praktikant gibt die obige Liste in das Datenbanksystem ein. Das Datenbanksystem verwendet zur Indizierung des Attributs fahrer_id einen B+-Baum der Ordnung 1 (2 Schlüsselwerte je Knoten; Fan-out: 3). Nachdem der Praktikant die ersten drei Tupel eingegeben hat, sieht der B+-Baum wie folgt aus:

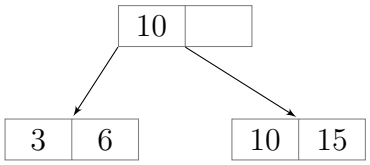


Der Praktikant gibt nun die restlichen Tupel (beginnend mit 'Juan Pablo Montoya') in die Datenbank in der obigen Reihenfolge von oben nach unten ein.

1. Wie sieht der B+-Baum nach jedem einzelnen Schritt aus wenn die Tupel weiter eingegeben werden?
   Zur Bearbeitung der Aufgabe ist die folgende Konvention zu wählen: Kommt es zum Split eines Knoten, verhält sich der B+-Baum wie folgt: Die jeweils größere "Hälfte" wandert in den rechten Knoten. Redistribution kennt das Datenbanksystem nicht.

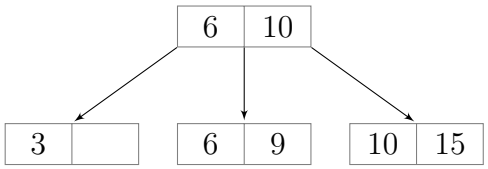
Lösung:

(a)



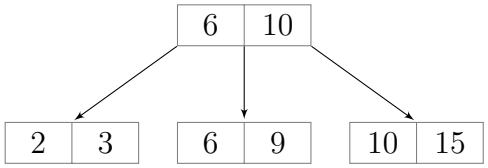
Die 3 passt in den linken Blattknoten. Es sind keine weiteren Aktionen nötig.

(b)



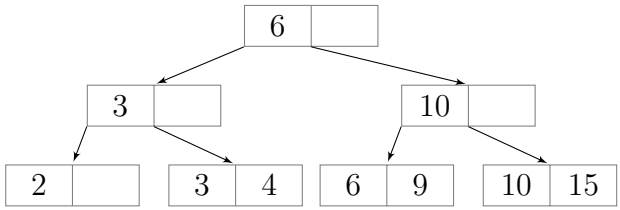
Der linke Blattknoten, in den die 9 eingefügt werden müsste, ist voll und wird daher gesplittet. Von {3,6,9} ist 6 der mittlere Wert und wird daher als Seperator in den Elternknoten übernommen. Außerdem wird es der erste Wert im rechten Knoten.

(c)



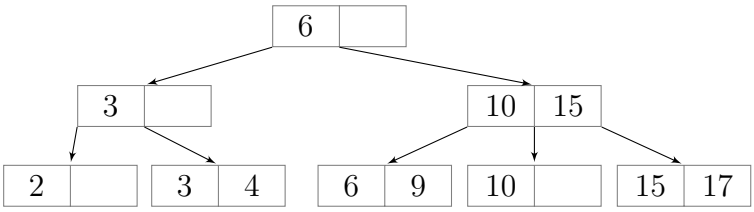
Die 2 passt in den linken Blattknoten. Es sind keine weiteren Aktionen nötig.

(d)



Der linke Blattknoten, in den die 4 eingefügt werden müsste, ist voll und wird daher gesplittet. Von {2,3,4} ist 3 der mittlere Wert und wird daher als Seperator in den Elternknoten übernommen. Die Wurzel ist allerdings ebenfalls voll und muss daher auch gesplittet werden. Von {4,6,10} ist 6 der mittlere Wert und wird daher als Seperator in die neue Wurzel verlagert. Alle Kinder rechts von der 6 (der alten Wurzel) werden dementsprechend in den rechten Teilbaum übernommen.

(e)



Der ganz rechte Blattknoten, in den die 17 eingefügt werden müsste, ist voll und wird daher gesplittet. Von {10,15,17} ist 15 der mittlere Wert und wird daher als Seperator in den Elternknoten übernommen. Außerdem wird es der erste Wert im ganz rechts eingefügten Knoten.

2. Schätzen Sie für B+- und Hash-Index den Aufwand in Anzahl Seitenzugriffen ab, der für eine große Anzahl Datensätze `n` für das Auffinden eines Tupels notwendig ist. Wie groß ist der Aufwand im Regelfall, wie können sich ungünstig verteilte Daten auswirken?

Lösung:

(a) Die Anzahl Seitenzugriffe für eine Navigation durch einen B+-Baum ergibt sich aus seiner Höhe:

$$\#Seitenzugriffe = h = \left\lceil \log_{fanout} n \right\rceil \ ,$$

wobei der *fanout* mindestens $d + 1$ und höchstens $2d + 1$ beträgt:

$$d + 1 \leq fanout \leq 2d + 1 \ .$$

In der Praxis liegt der *fanout* im Durchschnitt genau dazwischen. Dadurch erhält man

$$\#Seitenzugriffe \approx \left\lceil \log_{3/2d+1/2} n \right\rceil \ .$$

(b) Aufgrund seines direkten Zugriffs auf die Hashbuckets ermöglicht der Hash-Index einen Zugriff mit einem Aufwand in $\mathcal{O}(1)$. Dabei ist aber zu beachten, dass der Hash-Index ausschließlich für Zugriffe mit Prüfungen auf Gleichheit oder Ungleichheit genutzt werden kann und nicht für die Sortierung geeignet ist.

3. Beim erstmaligen Laden von großen Tabellen ("Bulk Loading") liegen die Quelldaten oft bereits vorsortiert vor. Welche Nachteile entstehen, wenn beim Ladevorgang der normale Einfüge-Algorithmus verwendet wird? Wie könnte man einen Bulk-Loading-Algorithmus konzipieren, der diese Nachteile behebt (und natürlich nur mit vorsortierten Daten funktioniert)?

Lösung:

(a) Durch das sequentielle Einfügen der vorsortierten Quelldaten müssen sehr häufig Knoten gesplitet werden. Dies führt zu einem hohen Zusatzaufwand und vielen Umstrukturierungen im B+-Baum.

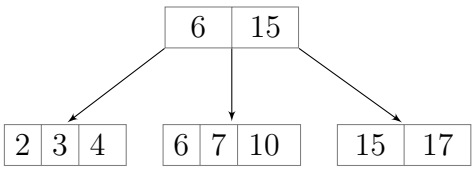
(b) Zur Beschleunigung des Einfügevorgangs könnte ein rekursiver Ansatz verfolgt werden, der die eingegebenen Daten in jedem Rekursionsschritt entsprechend dem Fanout des B+-Baums aufteilt, bis jedes Fragment klein genug ist, um in ein Blatt zu passen. Jede Zerteilung entspricht bei diesem Vorgang einem inneren Knoten im Baum. Für die gegebenen Beispieldaten liefe der Algorithmus wie folgt ab:

i.

2	3	4	6	7	10	15	17

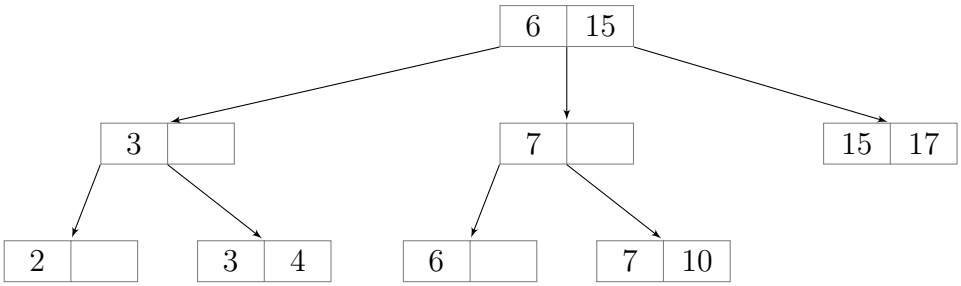
Ausgangssituation mit vorsortierter Eingabe

ii.



Erste Teilung in 3 Unterbäume. In der Wurzel werden die ersten Werte aus dem mittleren und rechten Baum als Seperator gesetzt.

iii.



Erneute Teilung der Unterbäume. Nun sind alle Blätter klein genug und es sind keine weiteren Unterteilungen nötig.