

Aufgabe 1 - *Slotted Page*.

1 Punkt

In die folgende Tabelle werden Werte eingefügt:

```
CREATE TABLE books (
  bid INTEGER,
  btitle VARCHAR(20)
);
```

Das Speichern eines `INTEGER` benötigt 4 Byte. Strings, die als `VARCHAR` gespeichert werden, benötigen ein Byte pro Zeichen und zusätzlich ein Byte zur Terminierung. Beispielsweise benötigt die Zeichenfolge `DBMS` 5 Bytes zur Speicherung. Die Tupel werden in eine `Slotted Page` mit folgenden Eigenschaften eingefügt:

- Größe: $2^{13} = 8192$ Bytes,
- Adressierungstyp: **Byte-Adressierung** (es kann jedes Byte adressiert werden)

Die folgenden Operationen werden in dieser Reihenfolge durchgeführt:

```
INSERT INTO books VALUES (42, 'Categoriae');           -- Tupel A
INSERT INTO books VALUES (13, 'Analytica_priora');    -- Tupel B
INSERT INTO books VALUES (37, 'De_sophisticis_elenchis'); -- Tupel C
```

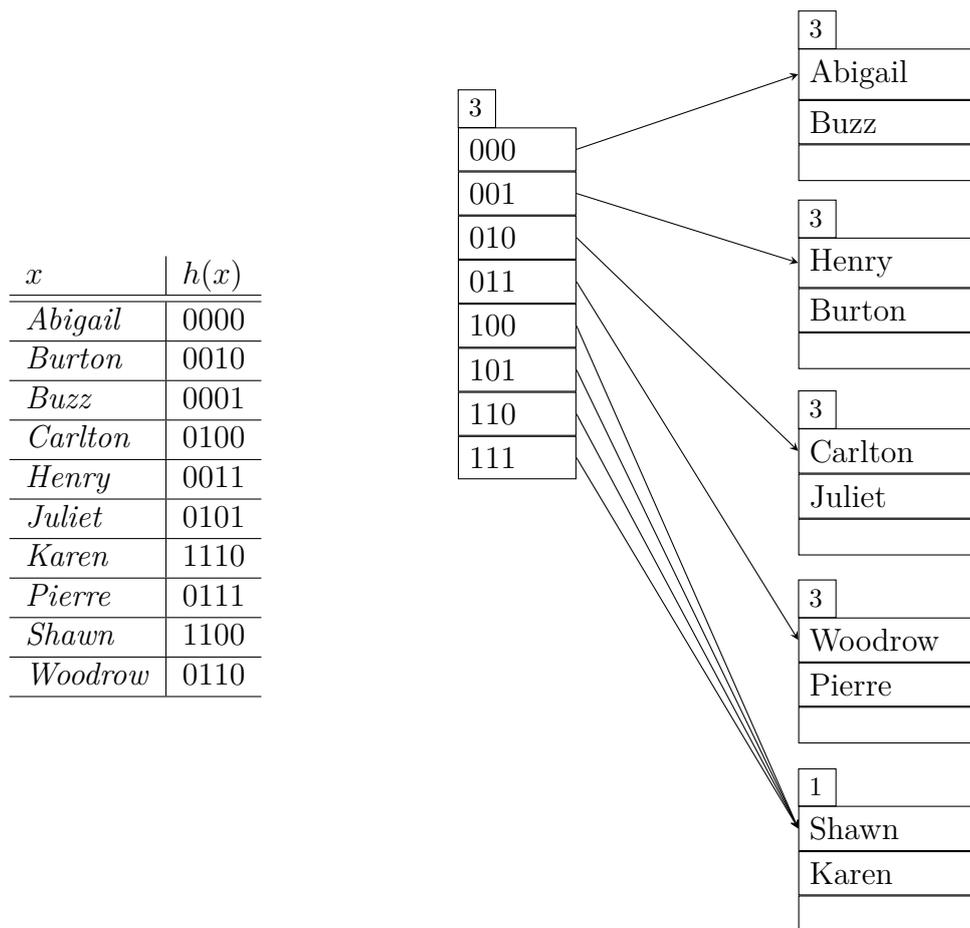
Ergänzen Sie die `Slotted Page` um die **fehlenden Werte/Adressen**, wobei p_i und g_i sich auf den jeweiligen Datensatz d_i beziehen. **(0.125 Punkte pro Wert/Adresse)**

a	f	g_1	p_1	g_2	p_2	g_3	p_3	...	d_3	d_2	d_1
									C	B	A

Aufgabe 2 - Erweiterbares Hashing.

1 Punkt

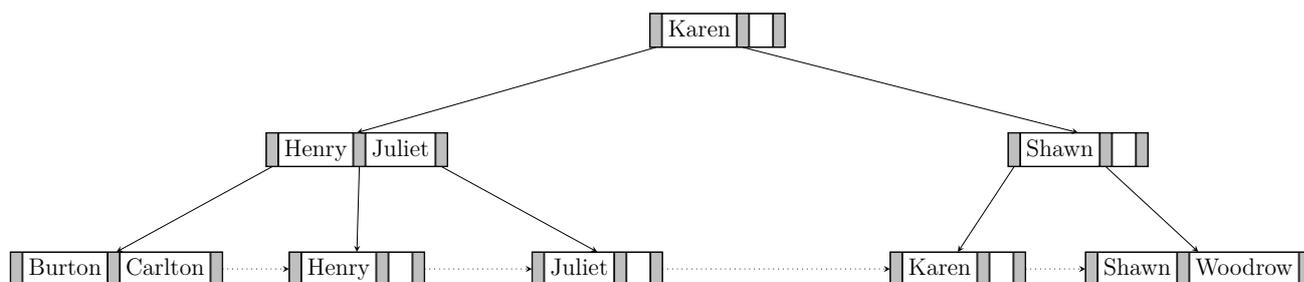
Die Hashfunktion $h(x)$ liefert die in der Tabelle angegebenen Binärwerte. Es soll das Tupel *Henry* aus dem gegebenen Hashcontainer gelöscht werden. Ein Bucket im Hashcontainer kann bis zu 3 Tupel speichern. Das Verzeichnis soll so klein wie möglich gehalten werden. **Illustrieren Sie den resultierenden Hashcontainer.**



Aufgabe 3 - B⁺-Baum-Löschen.

1 Punkt

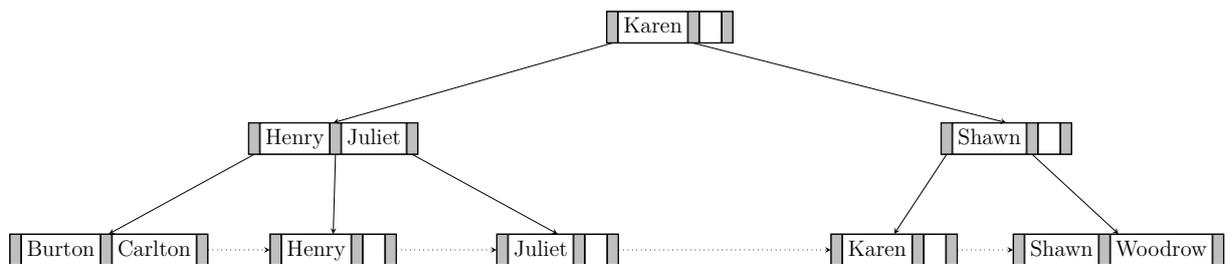
Gegeben ist ein B⁺-Baum mit $m = 3$. Zeichnen Sie den B⁺-Baum der nach dem Löschen von **Karen** entsteht.



Aufgabe 4 - B^+ -Baum-Einfügen.

1 Punkt

Gegeben ist ein B^+ -Baum mit $m = 3$. Zeichnen Sie den B^+ -Baum der nach dem Einfügen von **Dexter** entsteht.



Aufgabe 5 - Indexstrukturen.

1 Punkt

Zeichnen Sie für die folgende Tabelle einen **3-stufigen Sekundärindex** auf dem Attribut **Player**. Die **innere Indexstufe** soll **dense** und die **äußeren beiden Indexstufen** sollen **sparse** sein. Ein **Datenblock** kann **2 Tupel** speichern. In einen **Indexblock** können **3 Einträge** gespeichert werden.

Player	Team
Bell	Steelers
Brees	Saints
Brown	Steelers
Donald	Rams
Edelman	Patriots
Fournette	Jaguars
Gurley	Rams
Houston	Chiefs
Jones	Falcons
Kuechly	Panthers
Lattimore	Saints
Leonard	Colts
Mack	Bears
Michel	Colts
Sanu	Falcons
Thomas	Saints

Aufgabe 6 - *Externes Merge-Sort*.1 Punkt

Führen Sie **externes Merge-Sort** auf der folgenden Relation $R[A]$ aus.
Jeder **Block fasst 4 Tupel**. Die Größe des **Puffer** beträgt 4 **Blöcke**.

100
35
10
45
95
20
25
5
30
90
40
0
15
50
80
60
55
75
65
70

Aufgabe 7 - Join-Algorithmen.**1 Punkt**

Die Relation $R[\underline{A}, \underline{B}, C]$ hat $n_R = 3000$ Tupel auf $b_R = 400$ Blöcken, die Relation $S[\underline{B}, D]$ hat $n_S = 2000$ Tupel auf $b_S = 450$ Blöcken. Es stehen $M = 15$ Blöcke im Puffer zur Verfügung. Es soll ein natürlicher Join mittels eines Hash-Join-Algorithmus auf die Relationen durchgeführt werden.

Beantworten Sie die folgenden Fragen und begründen Sie.

- Welche Relation wird als Probe-Input verwendet, welche als Build-Input?
- Was ist die minimale Anzahl von Partitionen, die erstellt werden müssen?
- Schätzen Sie ab, wieviele Block-Zugriffe ein Hash Join braucht.

Aufgabe 8 - Effiziente Anfragebearbeitung.**1 Punkt**

Gegeben sei eine Relation $R[A, B, C]$ mit folgenden Eigenschaften:

- $|R| = 10 \cdot 10^{12}$ Tupel,
- Pro Datenblock werden $25 \cdot 10^3$ Tupel gespeichert,
- Attribut A hat ganzzahlige Werte gleichverteilt im Bereich $[1; 20 \cdot 10^6]$,
- Attribut B hat ganzzahlige Werte gleichverteilt im Bereich $[1; 20 \cdot 10^9]$,
- Attribut C hat ganzzahlige Werte gleichverteilt im Bereich $[1; 20 \cdot 10^3]$,
- Duplikate werden mittels Tuple Identifier (TID) aufgelöst.
- Folgende Indizes existieren:
 - einstufiger sequentieller sparse Index auf Attribut A mit $5 \cdot 10^6$ Einträgen pro Indexblock,
 - dense B^+ -Baum-Index auf Attribut B , $m = 1.024$, minimale Höhe.

Es soll folgende Anfrage beantwortet werden:

$$\sigma_{A=7 \cdot 10^3 \vee B=7 \cdot 10^3}(R)$$

Geben Sie die **Strategie (0.5 Punkte)** an und berechnen Sie die **Anzahl der Blockzugriffe (0.5 Punkte)** um die Anfrage **möglichst effizient** zu beantworten (1 Knotenzugriff im B^+ -Baum entspricht 1 Blockzugriff).

Aufgabe 9 - Join Kardinalität.

1 Punkt

Gegeben sind folgende Relationen:

- $|R[A, B, C]| = 1000, V(R, A) = 100, V(R, B) = 200, V(R, C) = 300$
- $|S[A, D, E]| = 4000, V(S, A) = 50, V(S, D) = 200, V(S, E) = 300$
- $|T[D, E, F]| = 2000, V(T, D) = 200, V(T, E) = 400, V(T, F) = 600$

Die Werte in den Tupeln sind gleichverteilt und unabhängig.

Schätzen Sie die Kardinalität des Ergebnisses der folgenden Anfrage ab ($\sigma_{A=100}(R) \neq \emptyset$).

$$(\sigma_{A=100}(R)) \bowtie S \bowtie T$$

Aufgabe 10 - Anfrageoptimierung, Join-Reihenfolge.**1 Punkt**

Gegeben seien 3 Relationen $R[A, B, C]$, $S[A, D, E]$ und $T[A, F, G]$:

- $|R| = 1.200$ Tupel, $V(R, A) = 50$, $V(R, B) = 100$, $V(R, C) = 200$
- $|S| = 3.000$ Tupel, $V(S, A) = 20$, $V(S, D) = 1.000$, $V(S, E) = 600$
- $|T| = 5.000$ Tupel, $V(T, A) = 100$, $V(T, F) = 1.200$, $V(T, G) = 1.800$

Weiters sei die folgende SQL-Anfrage gegeben:

```
SELECT R.A, S.D, T.G
FROM R, S, T
WHERE R.A = S.A AND R.A = T.A
```

1. Zeichnen Sie die **algebraische Normalform als Operatorbaum** für die gegebene SQL-Anfrage. **(0.5 Punkte)**
2. Wenden Sie die heuristische Optimierung an, um den Operatorbaum zu optimieren. **Es soll auch die Join-Reihenfolge optimiert werden! (0.5 Punkte)**