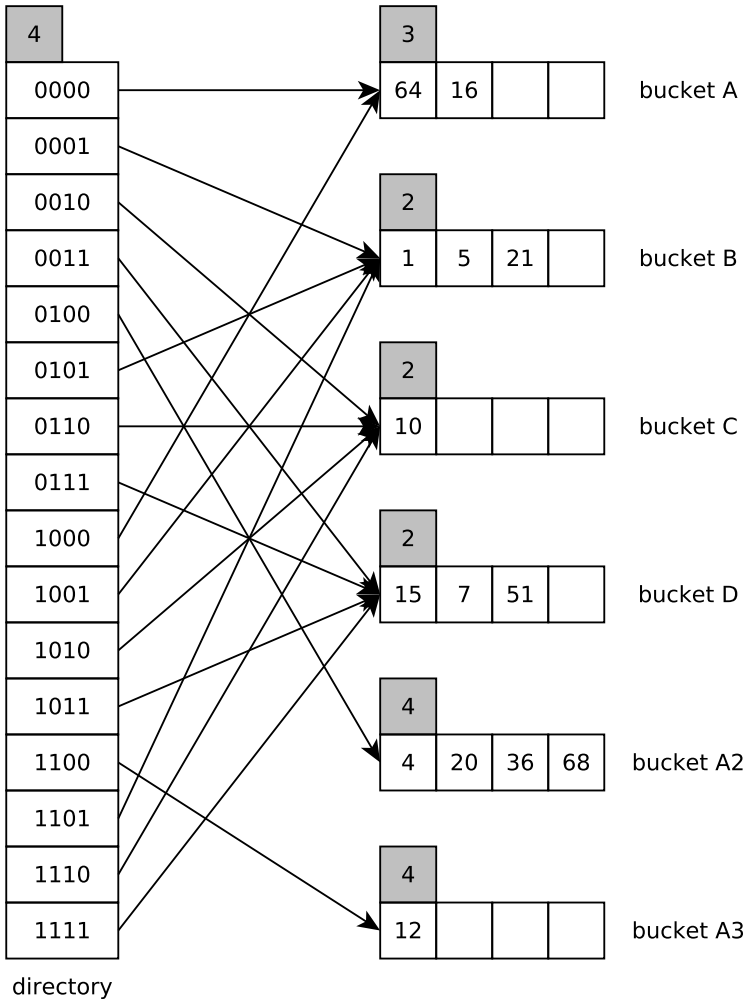
Aufgabe 1

Gegeben sei der Extendible Hashing Index *in Abbildung* **??**. *Auf dieser Grundlage beantworten Sie bitte folgende Fragen:*

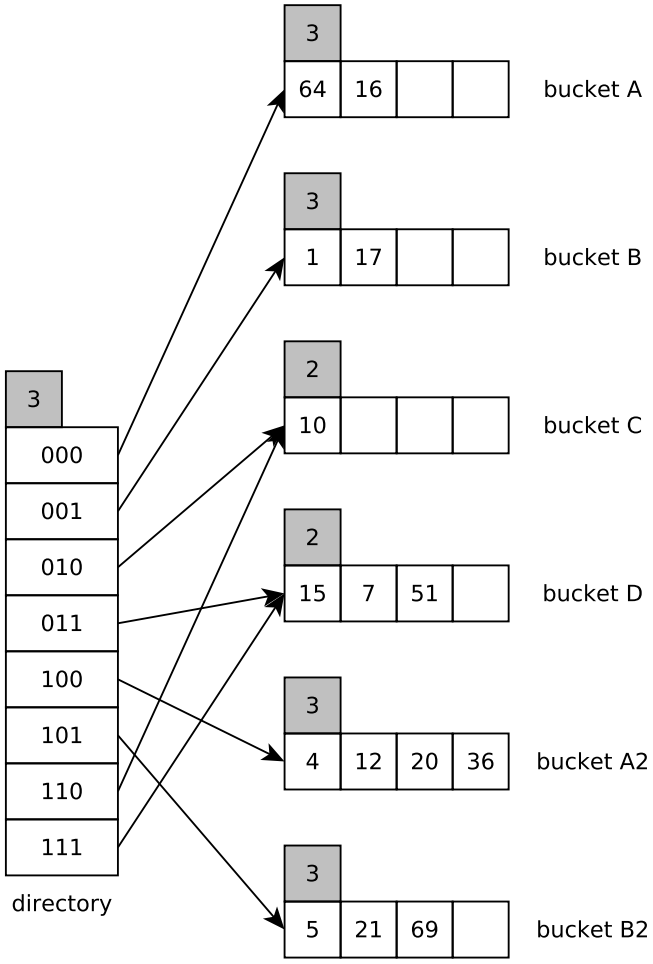
1. *Was können Sie über den Eintrag, der als Letzter in diesen Index eingetragen wurde sagen, wenn Sie wissen, dass bis dato kein Eintrag gelöscht wurde? Könnte dieser Eintrag einen Split ausgelöst haben?*

   Nein, da bisher nur A gesplittet wurde und hier aber mehr als $d + 1 = 5$ Elemente vorhanden sind, weswegen der Split mehrere Eintragungen zurück liegt.

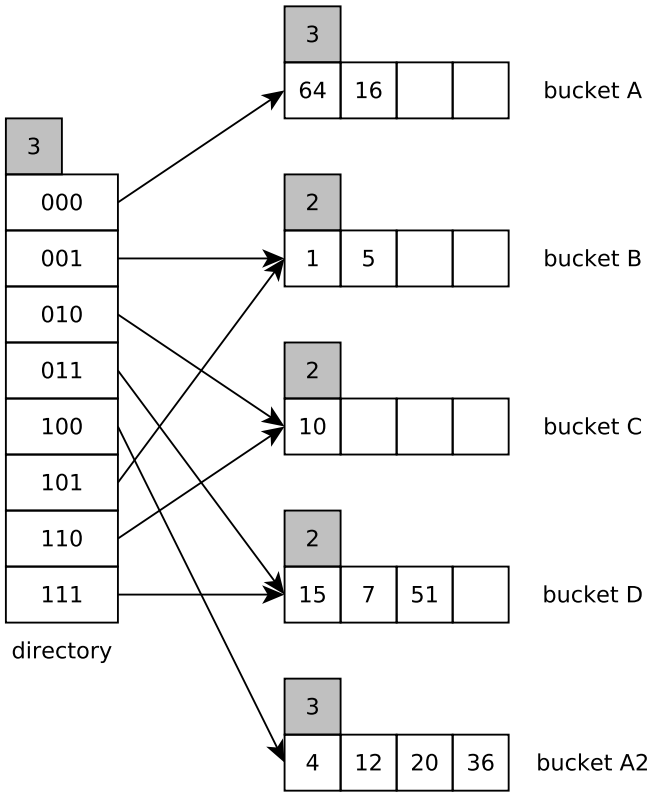
2. *Zeichnen Sie den Index aus Abbildung* 1*, nachdem Sie einen Eintrag v mit dem Hash-Wert* 68 *eingetragen haben.*



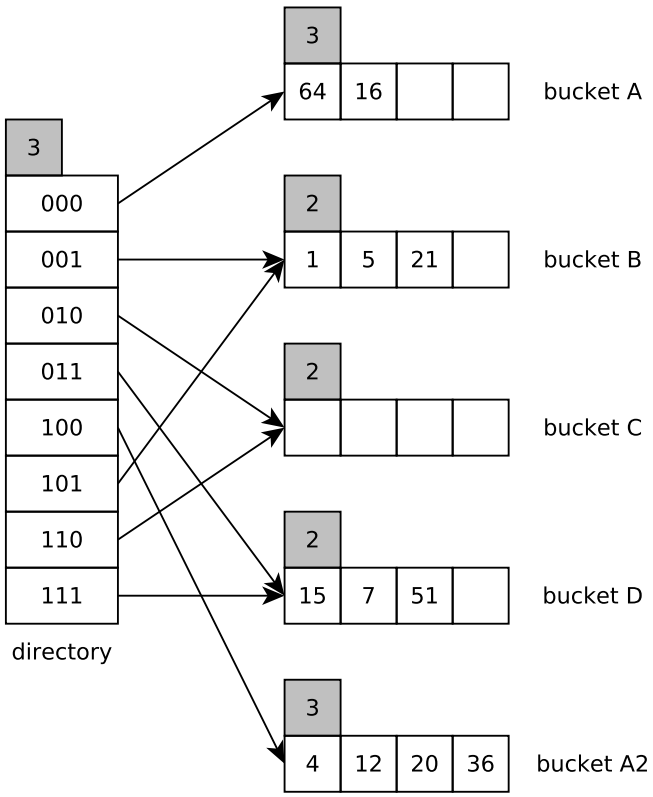
3. *Fügen Sie im originalen Index von Abbildung* 1 *die Werte* v_1 *und* v_2 *mit den Hash-Werten* 17 *und* 69 *ein und zeichnen Sie den Index auf.*



4. *Löschen Sie im Index von Abbildung* 1 *den Eintrag mit dem Hash-Wert* 21.



5. *Zeichnen Sie den Index aus Abbildung* 1*, nachdem Sie den Eintrag mit dem Hash-Wert 10 gelöscht haben. Wird dadurch eine Verschmelzung ausgelöst? Wenn nicht, nennen Sie die Gründe dafür. (Beachten Sie, dass hier wieder der vollständige Löschalgorithmus zum Einsatz kommt.)*



Es wird keine Verschmelzung ausgelöst, da eine Verschmelzung Konflikte für Bucket A und $A2$ auslösen. Anders ausgedrückt, die *mininal verlangte* globale Größe 3 ist größer, als die lokale Größe 2.

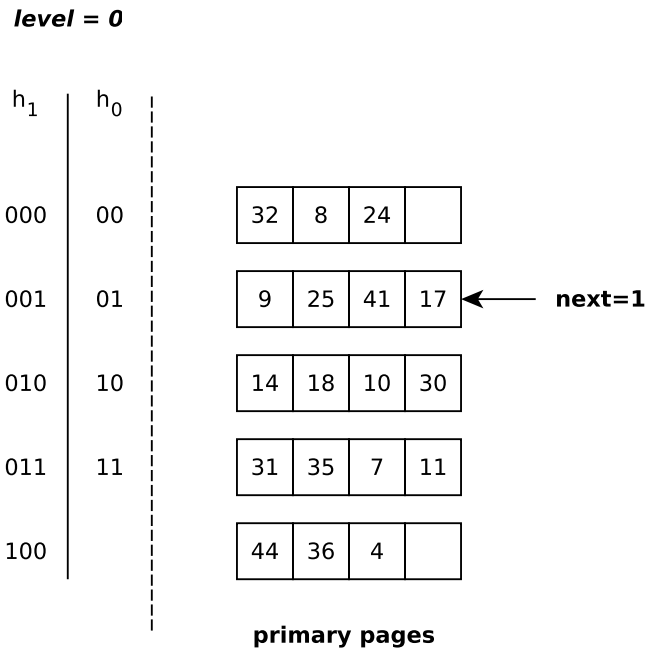
Aufgabe 2

Gegeben Sei der Linear Hashing Index *in Abbildung 2. Das Kriterium für einen Split ist das Anlegen einer neuen* overflow page.
Auf dieser Grundlage beantworten Sie bitte folgende Fragen:

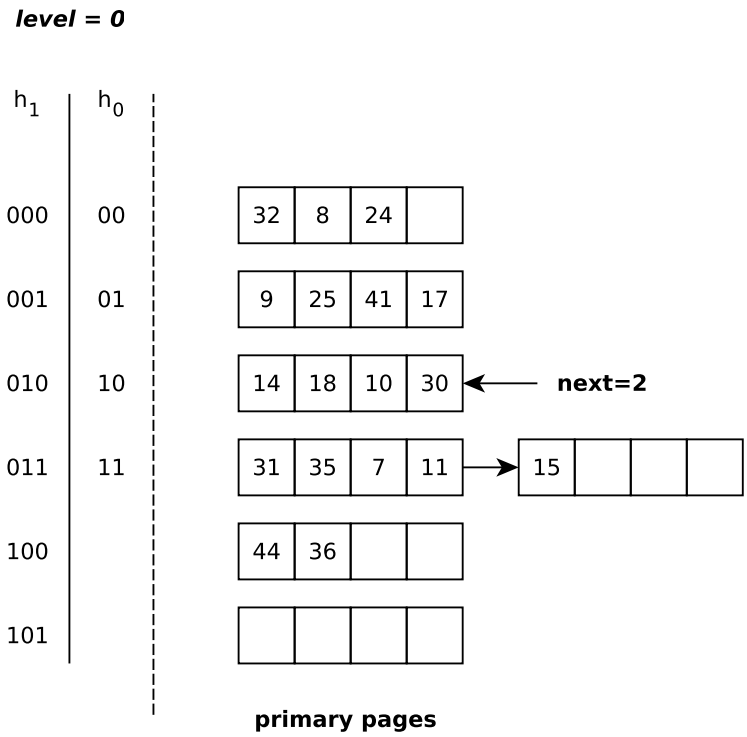
1. *Was können Sie über den Eintrag, der als Letzter in diesen Index eingetragen wurde sagen, wenn Sie wissen, dass bis dato kein Eintrag gelöscht wurde? Könnte dieser Eintrag einen Split ausgelöst haben?*

   Ein durch den letzten Eintrag ausgelöster Split ist möglich, da die Page 0 und 4 zusammen $n + 1 = 5$ Elemente ergeben, auch da der `next`$= 1$ ist.

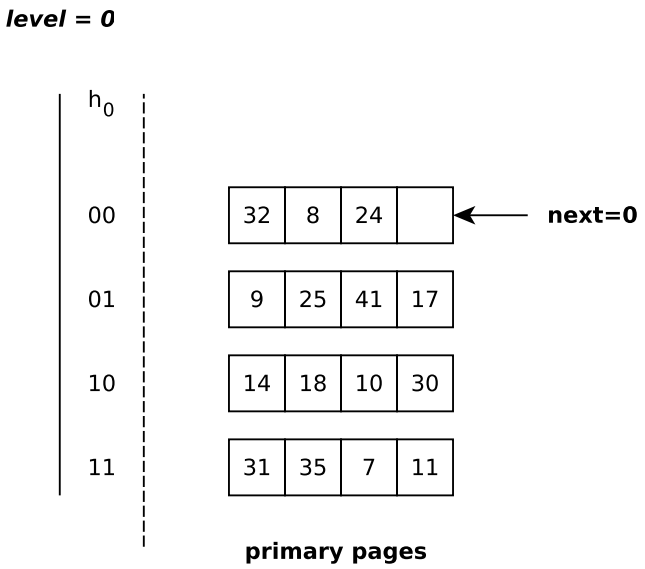
2. *Zeichnen Sie den Index aus Abbildung* 2*, nachdem Sie einen Eintrag v mit dem Hash-Wert* 4 *eingetragen haben.*



3. *Fügen Sie im originalen Index von Abbildung* 2 *den Wert* v *mit den Hash-Wert* 15 *ein und zeichnen Sie den Index auf.*



4. *Löschen Sie im Index von Abbildung* 2 *die Einträge mit den Hash-Werten* 36 *und* 44. *Legen Sie dabei ëmpty primary bucketäls* underflow *Kriterium zu Grunde.*



Aufgabe 3

Gegeben sei eine Relation $R(a, b, c, d)$, die mit einer Million Tupeln gefüllt ist. R ist als Heap-Datei mit unclustered Indizes organisiert. Die Tupel in R seien in beliebiger Reihenfolge angeordnet. a sei ein Kandidatenschlüssel für R, mit Werten zwischen 0 *bis* 999999*. Für jede der folgenden Anfragen, geben Sie den Ansatz an welcher, die wenigsten I/O Operationen benötigt. Begründen Sie* kurz*.*

- *Scannen der gesamten Heap-Datei.*
- *Benutzen eines B+ Baumes auf das Attribut a.*
- *Benutzen eines Hash-Index auf das Attribut a.*

Die Anfragen lauten wie folgt:

1.
```
SELECT *
FROM R
```

2.
```
SELECT *
FROM R
WHERE a < 50
```

3.
```
SELECT *
FROM R
WHERE a = 50
```

4.
```
SELECT *
FROM R
WHERE a > 50 AND a < 100
```

1. Scannen der gesamten Heap-Datei.
   Da ohnehin alle Elemente von R gelesen werden überwiegt der sequential Scan in der Performanz gegenüber einem zusätzlichen Mehraufwand beim Durchlaufen der anderen Strukturen.
2. Benutzen eines B+ Baumes auf das Attribut a.
   Bei Range-Queries haben B+ Bäume den geringsten Aufwand, da über die Baumstruktur eine schnelle Suche möglich ist und Querverweise das sequentielle Lesen ermöglichen. Sie bilden damit einen Kompromiss zwischen Heap-Scan und Index-Zugriff.

3. Benutzen eines Hash-Index auf das Attribut a.
   Es wird nur ein Element gelesen, womit sich der Aufwand auf den Index-Zugriff beschränkt, hierbei ist der Hash-Index mit den wenigsten I/O's berechnet.
4. Benutzen eines B+ Baumes auf das Attribut a.
   Bei Range-Queries haben B+ Bäume den geringsten Aufwand, da über die Baumstruktur eine schnelle Suche möglich ist und Querverweise das sequentielle Lesen ermöglichen. Sie bilden damit einen Kompromiss zwischen Heap-Scan und Index-Zugriff.