TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS15/16

Harald Lang, Linnea Passing (gdb@in.tum.de)
http://www-db.in.tum.de/teaching/ws1516/grundlagen/

Blatt Nr. 10

Hausaufgabe 1

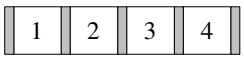
(a) Fügen Sie in einen anfänglich leeren B$^+$-Baum mit $k = 3$ und $k^* = 2$ die Zahlen eins bis fünfundzwanzig in aufsteigender Reihenfolge ein. In den Blattknoten werden TIDs verwendet. Was sind TIDs, wann lohnt sich ihre Verwendung, was ist die Alternative zu TIDs?

(b) Erläutern Sie die Vorgehensweise bei der Bearbeitung der folgenden Anfrage „Finde alle Datensätze mit einem Schlüsselwert zwischen 5 und 15."

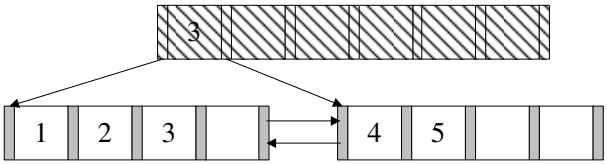
Lösung:

(a) Bei B$^+$-Bäumen unterscheiden sich die Kapazitäten von inneren Knoten und Blattknoten (angegeben durch k und k^*. Im Folgenden werden innere Knoten zur leichteren Unterscheidung schraffiert.

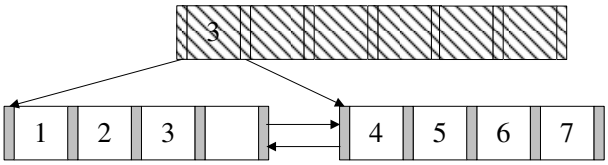
Nachdem man die Zahlen 1 bis 4 eingefügt hat, liegt folgender B-Baum vor (da die Wurzel in diesem Fall ein Blatt ist können höchstens 4 Einträge eingefügt werden):



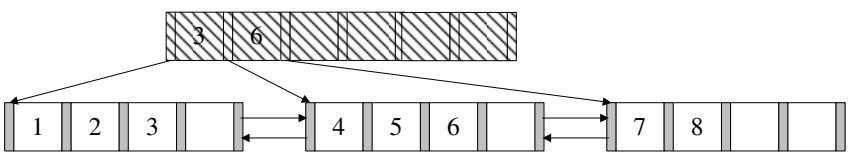
Beim Einfügen von 5 wird der Knoten gespalten. Der Referenzschlüssel 3 wandert in die neue Wurzel, deren Kapazität 6 ist. Die neuen Blattknoten haben eine Kapazität von 4 und sind untereinander verlinkt.



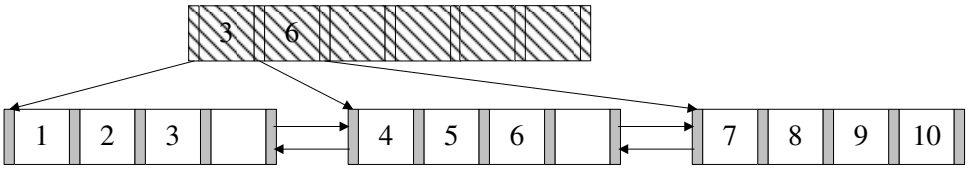
Die nächsten beiden Einträge lassen sich wieder ohne Probleme einfügen.



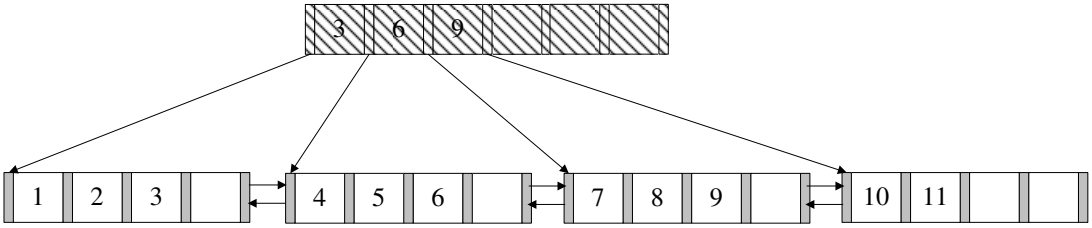
Beim Einfügen der 8 kommt es erneut zum Überlauf. Die 6 wandert in die Wurzel.



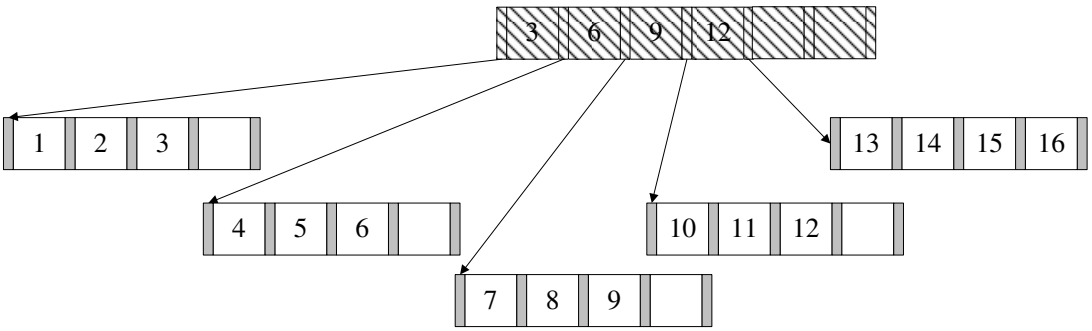
9 und 10 lassen sich wieder ohne Probleme einfügen. Bei 11 kommt es zum Überlauf.



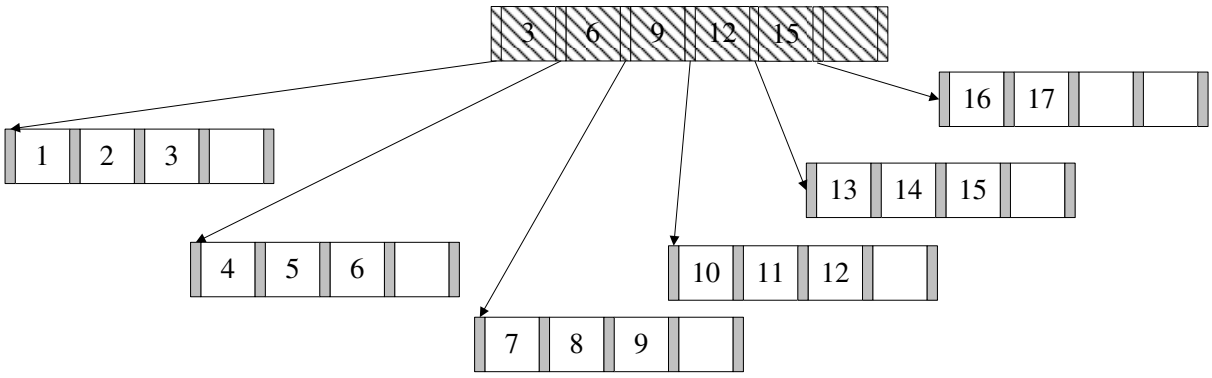
Nach dem Aufspalten erhält man dann:



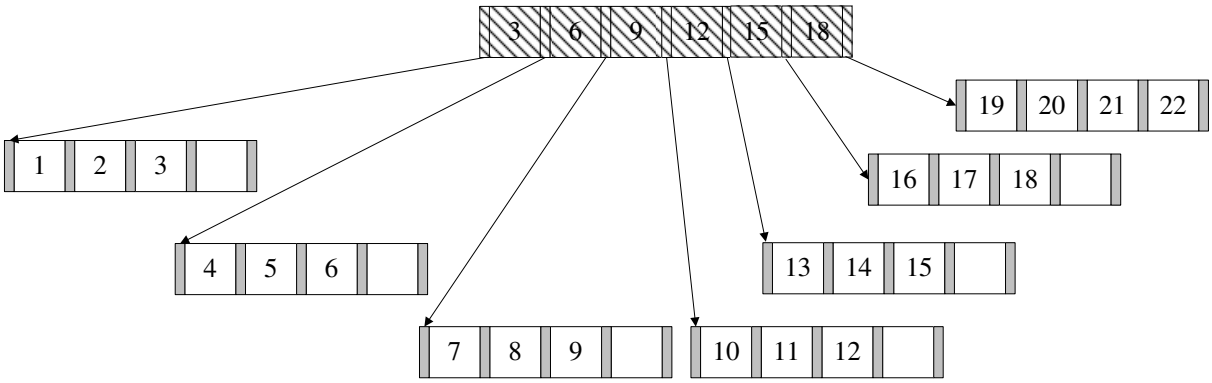
Es werden nun die nächsten Zahlen bis 16 analog eingefügt. (Die Pointer zwischen den Blattknoten existieren weiterhin, werden hier jedoch nicht mehr dargestellt.)



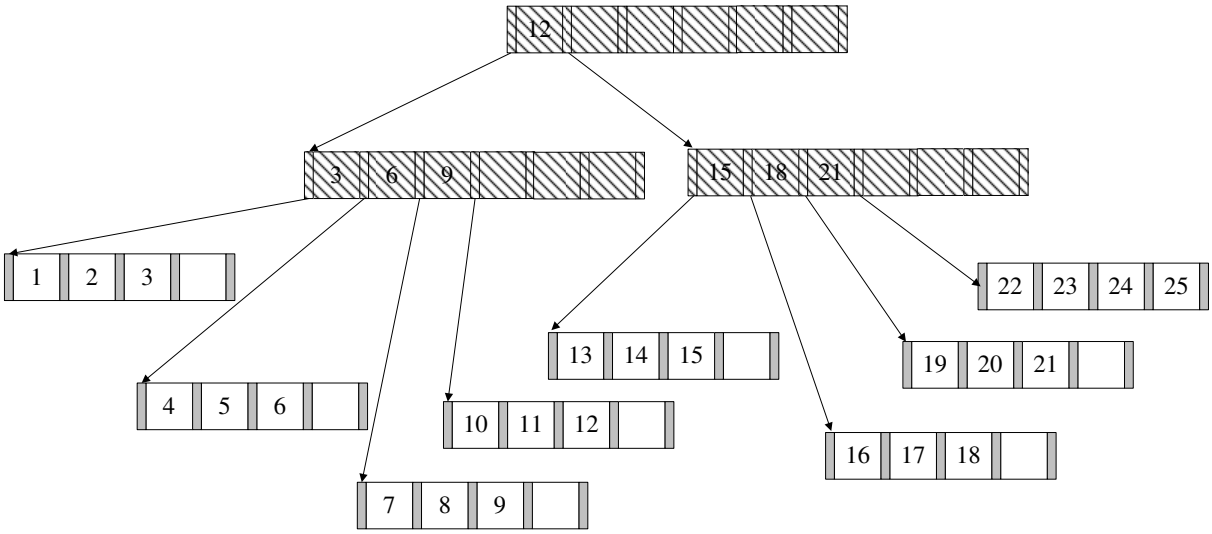
Bei 17 kommt es dann wieder zum Überlauf.



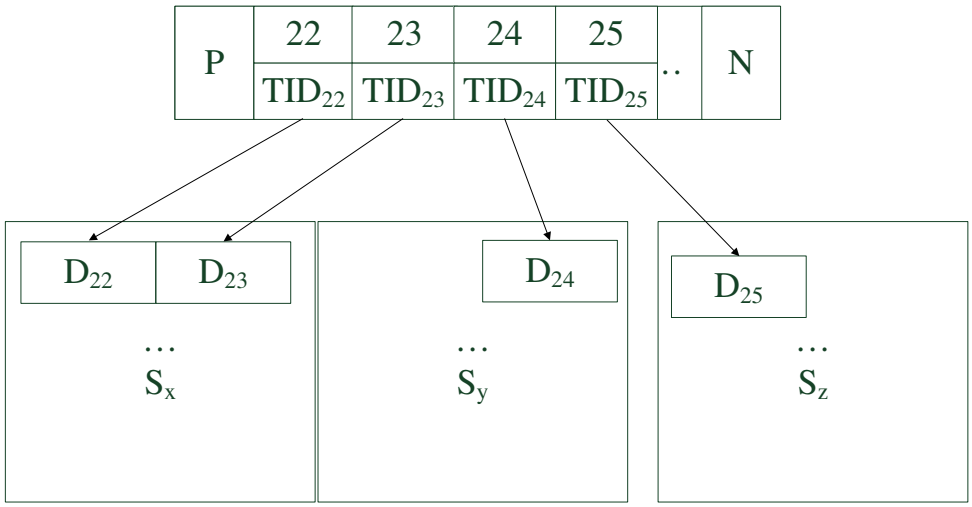
Die nächsten Zahlen werden wieder analog eingefügt. Bei 20 kommt es zum Überlauf.



Nun fügt man noch die Zahlen 21 und 22 ein. Bei 23 kommt es erneut zum Überlauf. Die 21 wird in den Wurzelknoten kopiert, wodurch auch hier ein Überlauf stattfindet, so dass der Baum in seiner Höhe wächst. Nach dem Einfügen von 24 und 25 sieht der Baum wie folgt aus:



In den Blättern können nun Datensätze oder TIDs gespeichert werden. TIDs sind „Zeiger“ auf Tuple. Werden TIDs vewendet, wird der Index kompakter und dadurch der Baum weniger hoch. Dafür ist eine weitere Indirektion zum Auffinden der Daten erforderlich, die bei der Suche nach einem Tupel verfolgt werden muss. Der letzte Knoten würde (im Detail) so aussehen:



S_x, S_y und S_z sind dabei beliebige Seiten im Speicher.

(b) Um eine Bereichsanfrage zu beantworten geht man wie folgt vor:

1. Zunächst sucht man nach der unteren Schranke der Anfrage, in diesem Fall nach der 5. Dies geschieht genauso wie beim B-Baum. Die Suche endet in einem Blattknoten.
2. Anschließend liest man alle sukzessiven Einträge bis zur oberen Schranke der Anfrage, in diesem Fall 15. Hierbei nutzt man die *Next*-Verlinkungen der Blattknoten untereinander.

Natürlich wäre es umgekehrt auch möglich, nach der oberen Schranke zu suchen und dann den *Previous*-Pointern zu folgen.

Hausaufgabe 2

Bestimmen Sie k für einen B-Baum, der die folgenden Informationen aller Menschen auf der Erde (ca. 10 Milliarden) enthalten soll: Namen, Land, Stadt, PLZ, Straße und Hausnummer (insgesamt ca. 100 Byte). Dabei ist die Steuernummer eindeutig und 64 Bit lang und wird im B-Baum als Suchschlüssel verwendet. Gehen Sie bei der Berechnung davon aus, dass eine Speicherseite 16KiB groß ist und ein Knoten des B-Baums möglichst genau auf diese Seite passen sollte.

Lösung:

Zunächst gibt uns die Information über die zu erwartende Anzahl von Einträgen im B-Baum einen Hinweis auf die größe eines Verweises in den Speicher. Wir betrachten gängige Architekturen und sehen, dass eine Speicherung auf einer Maschine, deren Adressierbarer Speicher lediglich 2^{32} byte groß ist, nicht ohne weiteres möglich ist. Wir gehen im Verlauf der Aufgabe also von einer 64 bit Architektur aus, bei der ein Zeiger eine Größe von 8 byte hat.

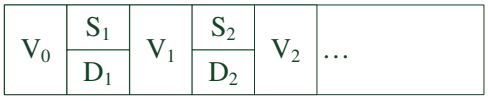


Abbildung 1: Struktur eines B-Baum Knotens

Um nun k zu bestimmen, muss die von k abhängige Größe eines Knotens maximiert werden, so dass dieser gerade noch auf eine Seite der Größe 16KiB passt. Die Struktur eines solchen Knotens ist in Abbildung 1 dargestellt. Zu beachten ist, dass ein komplett gefüllter Knoten genau $2k$ Schlüssel/Daten Paare, jedoch $2k + 1$ Verweise speichern muss.

Die Berechnung ergibt sich wie folgt:

$$\begin{aligned} 2k * (\text{Schlüsselgröße} + \text{Datengröße}) + (2k + 1) * \text{Zeigergröße} &\leq 16KiB \\ 2k * (8\text{bytes} + 100\text{bytes}) + (2k + 1) * 8\text{bytes} &\leq 16384\text{bytes} \\ 2k * 116\text{bytes} + 8\text{bytes} &\leq 16384\text{bytes} \\ 2k &\leq 16376\text{bytes}/116\text{bytes} \\ k \leq (16376\text{bytes}/116\text{bytes})/2 &\approx 70,59 \end{aligned}$$

Da der Knoten „innerhalb“ einer Speicherseite liegen soll und daher nicht überlappen darf, sollte k idealerweise auf 70 gesetzt werden, keinesfalls auf 71.

Hausaufgabe 3

Fügen Sie nacheinander die folgenden Einträge in eine anfangs leere erweiterbare Hashtabelle, welche 2 Einträge pro Bucket aufnehmen kann, ein. Es soll effizient nach der `KundenNr` gesucht werden können.

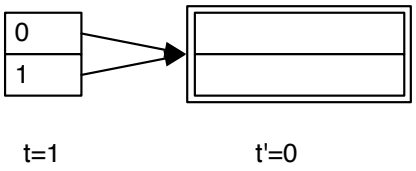
KundenNr	Name
10	Müller
25	Meier
30	Schmidt
18	Krause
40	Schulz
45	Kaufmann

Lösung:

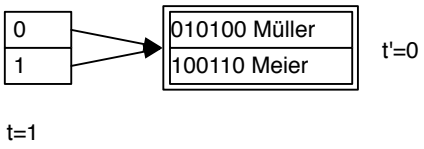
Werte mit binärer KundenNr sowie invers binärer Kundennummer, die für das Einfügen in den Hash genutzt wird:

KundenNr	Name	Binär	Umgekehrt Binär
10	Müller	001010	010100
25	Meier	011001	100110
30	Schmidt	011110	011110
18	Krause	010010	010010
40	Schulz	101000	000101
45	Kaufmann	101101	101101

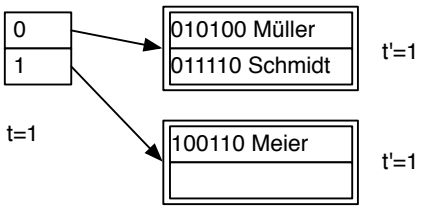
Zunächst eine leere erweiterbare Hashtabelle:



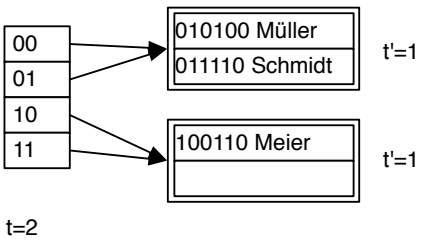
Wir fügen nun die ersten zwei Einträge ein, wonach die Hashtabelle wie folgt aussieht:



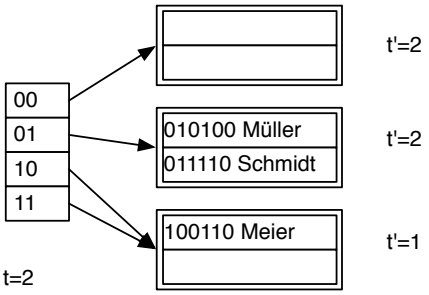
Der nächste Eintrag führt zu einem Überlauf. Da $t' < t$ können wir den Bucket teilen, dies führt zur folgenden Hashtabelle:



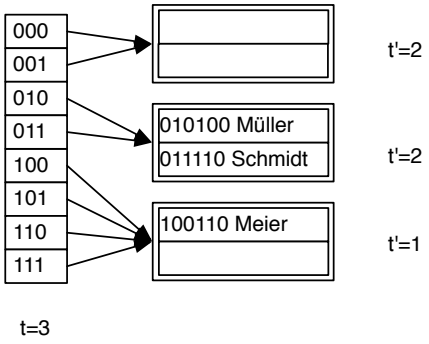
Das Einfügen von `Krause` führt erneut zu einem Überlauf, der Bucket kann aber nicht direkt geteilt werden, da $t' = t$ gilt. Das Verzeichnis wird verdoppelt:



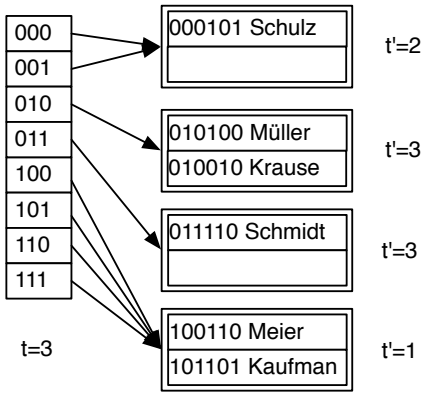
Nun kann der Bucket geteilt werden:



Das Einfügen ist leider immer noch nicht möglich und wieder hilt $t' = t$, weswegen das Verzeichnis erneut verdoppelt werden muss:



Nun kann der Bucket geteilt und alle Einträge eingefügt werden:



Hausaufgabe 4

Gegeben sei eine erweiterbare Hashtabelle mit globaler Tiefe t. Wie viele Verweise zeigen vom Verzeichnis auf einen Behälter mit lokaler Tiefe t'?

Lösung:

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.

$\Rightarrow$ Es gibt somit $2^{t-t'}$ Einträge im Verzeichnis, die auf denselben Behälter mit lokaler Tiefe t' verweisen.

Der Lehrstuhl für Datenbanksysteme wünscht
frohe Weihnachten und schöne Feiertage!