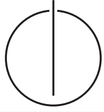


TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Dr. Thomas Neumann

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS19/20

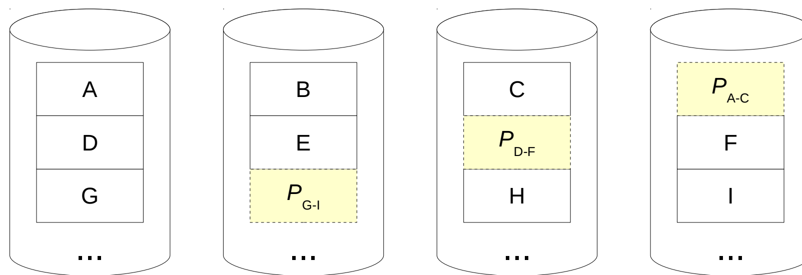
Christoph Anneser, Moritz Sichert, Lukas Vogel (gdb@in.tum.de)

https://db.in.tum.de/teaching/ws1920/grundlagen/

Blatt Nr. 09

Hausaufgabe 1

Die folgende Abbildung zeigt einen Festplattenverbund bestehend aus vier Laufwerken, auf welchen die Datenblöcke A bis I gespeichert sind. Die Blöcke P_i enthalten Paritätsinformationen.



a) Um welches RAID-Level handelt es sich?

b) Wieviele Festplatten können ausfallen, ohne dass mit Datenverlust zu rechnen ist? Geben Sie eine allgemeine Lösung für einen Verbund bestehend aus n Festplatten an.

c) Kann die Ausfallsicherheit erhöht werden? Begründung?

d) Welchen weiteren Vorteil bietet das gezeigte RAID-System neben der Ausfallsicherheit?

e) Nach einem Festplattendefekt enthalten die Datenblöcke die folgenden Binärdaten. Rekonstruieren Sie die Datenblöcke der $Disk_2$ mithilfe der XOR-Verknüpfung.

$Disk_0$	$Disk_1$	$Disk_2$	$Disk_3$
A = 1111	B = 1001	C = _ _ _ _	$P_{\text{A-C}}$ = 1110
D = 0101	E = 1100	$P_{\text{D-F}}$ = _ _ _ _	F = 1100
G = 0011	$P_{\text{G-I}}$ = 1110	H = _ _ _ _	I = 0011

Lösung:

a) 5

b) 1, unabhängig von n.

c) Ja, z.B. mit einem RAID-6 (Ausfall zweier Platten kann kompensiert werden) oder RAID-15 (das RAID-5 wird zusätzlich nochmal gespiegelt).

d) Höherer Datendurchsatz.

e) Die Rekonstruktion der Datenblöcke unterscheidet sich rechnerisch nicht von der Berechnung der Parität.

$Disk_0$	$Disk_1$	$Disk_2$	$Disk_3$
A = 1111	B = 1001	C = **1000**	$P_{\text{A-C}}$ = 1110
D = 0101	E = 1100	$P_{\text{D-F}}$ = **0101**	F = 1100
G = 0011	$P_{\text{G-I}}$ = 1110	H = **1110**	I = 0011

Hausaufgabe 2

Gegeben sei ein Array von 1.000.000.000 8-Byte-Integer-Werten und ein Programm, das alle Werte aufsummiert.

Das Programm wird auf einem System mit 16 GB Hauptspeicher und einer herkömmlichen Magnetfestplatte (Größe 1 TB), auf der alle Werte sequentiell gespeichert sind, ausgeführt. Ein Random Access auf die Festplatte dauert 10 ms, beim sequentiellen Lesen hat sie einen Durchsatz von 160 MB/s. Das Summieren zweier Werte im Hauptspeicher dauert 1 ns.

(1 MB = 10^6 B und 1 TB = 10^{12} B)

a) Gehen Sie davon aus, dass alle Werte bereits im Hauptspeicher liegen. Wie lange läuft das Programm?

b) Nun liegen alle Werte ausschließlich auf der Festplatte. Wie lange läuft das Programm jetzt?

c) Auf der Festplatte liegt jetzt zusätzlich nach jedem 100.000. Wert die Summe der 100.000 davorliegenden Werte. Wie lange läuft das Programm, wenn es nur diese Summen aufsummiert?

Lösung:

a) Jede Zahl wird auf eine laufende Gesamtsumme addiert. Insgesamt müssen also 10^9 Additionen ausgeführt werden. Bei einer Zeit von 1 ns pro Addition ergibt dies eine Gesamtlaufzeit von $10^9 \cdot 10^{-9}$ s = 1 s.

b) Da die Werte sequentiell auf der Festplatte liegen, muss der Lesekopf nicht jeden Wert einzeln ansteuern sondern nur einmal zum ersten Wert finden und dann die Daten sequentiell auslesen. Hier wird die Lesezeit also vom maximalen Durchsatz der Platte dominiert. Insgesamt ergibt sich also:

$$\begin{aligned} t_{total} &= t_{seek} + t_{read} \\ &= 10 \text{ ms} + \frac{10^9 \cdot 8 \text{ B}}{160 \cdot 10^6 \ \frac{\text{B}}{s}} \\ &= 10 \text{ ms} + \frac{8 \cdot 10^9 \text{ B}}{16 \cdot 10^7 \ \frac{\text{B}}{s}} \\ &= 10 \text{ ms} + 50 \text{ s} \\ &\approx 50 \text{ s} \end{aligned}$$

Dazu kommt noch die in Teilaufgabe a) berechnete Additionszeit selbst. Die Gesamtlaufzeit beträgt also 51 Sekunden.

c) Hier kann sich das Programm die Einzeladditionen sparen und muss lediglich die $10^9/10^5 = 10000$ Zwischenwerte addieren. Diese liegen nun allerdings nicht mehr sequentiell hintereinander, sondern $100000 \cdot 8$ B = 800 KB auseinander. Das ist wesentlich größer als ein typischer Festplattensektor. Jeder Lesezugriff auf einen solchen Zwischenwert ist also ein Random Access. Wir erhalten so eine aufsummierte Zugriffszeit von $10000 \cdot 10$ ms = 100 s. Die Additionszeit mit 10000 ns = 10 μs und die Übertragungszeit mit $(8 \cdot 10000)/(160 \cdot 10^6$ B/s$) = 0.5$ ms ist vernachlässigbar. Insgesamt läuft das Programm also ungefähr 100 s.

Trotz der „Optimierung" hat diese Variante also eine längere(!) Laufzeit, als der „naive" Ansatz aus Teilaufgabe b).

Hausaufgabe 3

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?

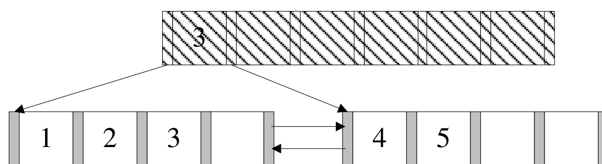
b) Erläutern Sie die Vorgehensweise bei der Bearbeitung der folgenden Anfrage „Finde alle Datensätze mit einem Schlüsselwert zwischen 5 und 15."

Lösung:

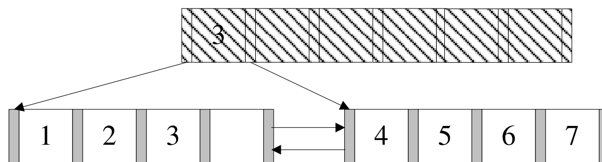
a) Bei B$^+$-Bäumen unterscheiden sich die Kapazitäten von inneren Knoten und Blattknoten (angegeben durch k und k^*). Im Folgenden werden innere Knoten zur leichteren Unterscheidung schraffiert.

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):

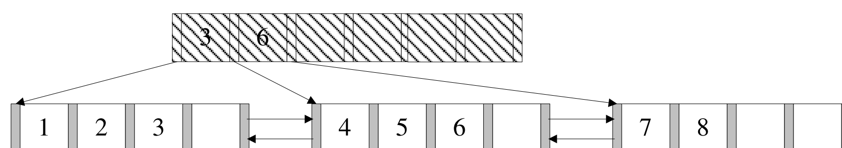
Beim Einfügen von 5 wird der Knoten gespalten. Der Referenzschlüssel 3 wandert in die neue Wurzel, deren Kapazität 6 ist. Die neuen Blattknoten haben eine Kapazität von 4 und sind untereinander verlinkt.



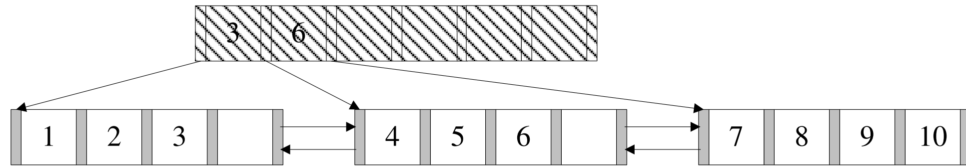
Die nächsten beiden Einträge lassen sich wieder ohne Probleme einfügen.



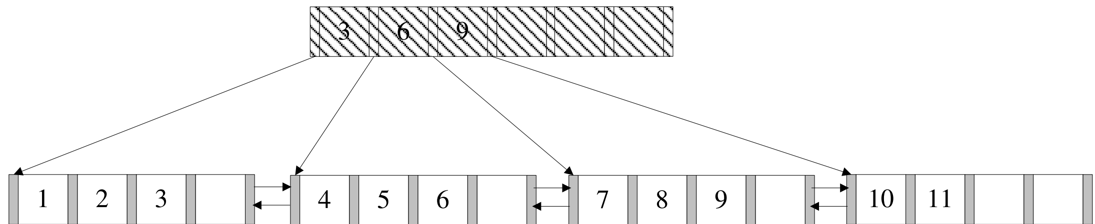
Beim Einfügen der 8 kommt es erneut zum Überlauf. Die 6 wandert in die Wurzel.



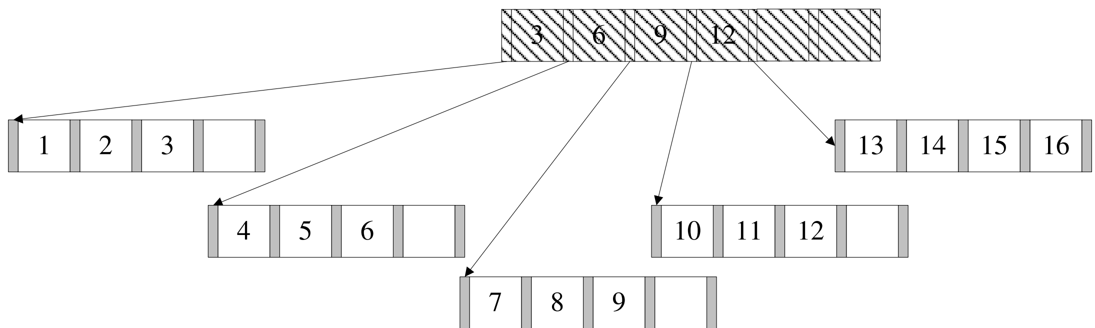
9 und 10 lassen sich wieder ohne Probleme einfügen. Bei 11 kommt es zum Überlauf.



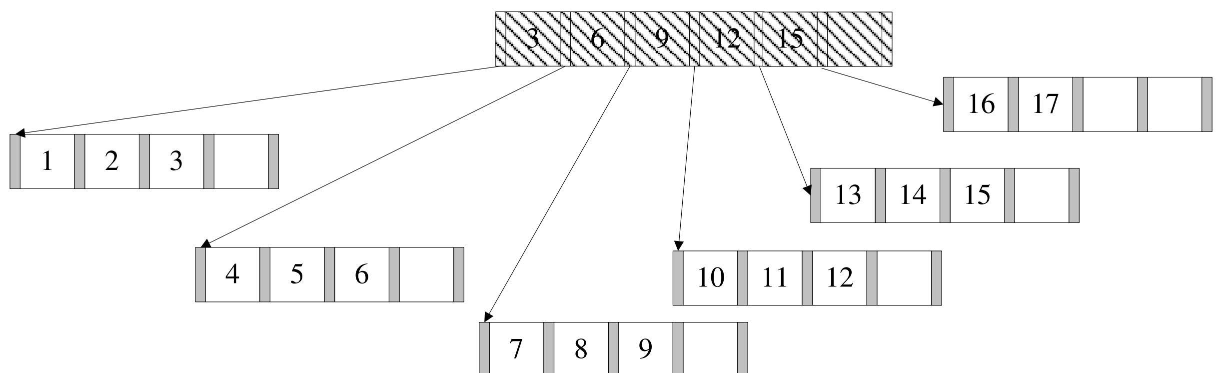
Nach dem Aufspalten erhält man dann:



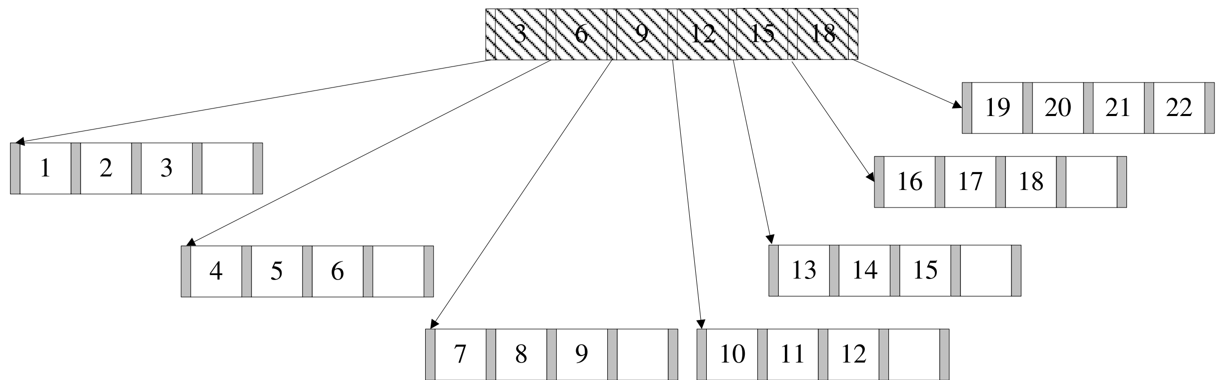
Es werden nun die nächsten Zahlen bis 16 analog eingefügt. (Die Pointer zwischen den Blattknoten existieren weiterhin, werden hier jedoch nicht mehr dargestellt.)



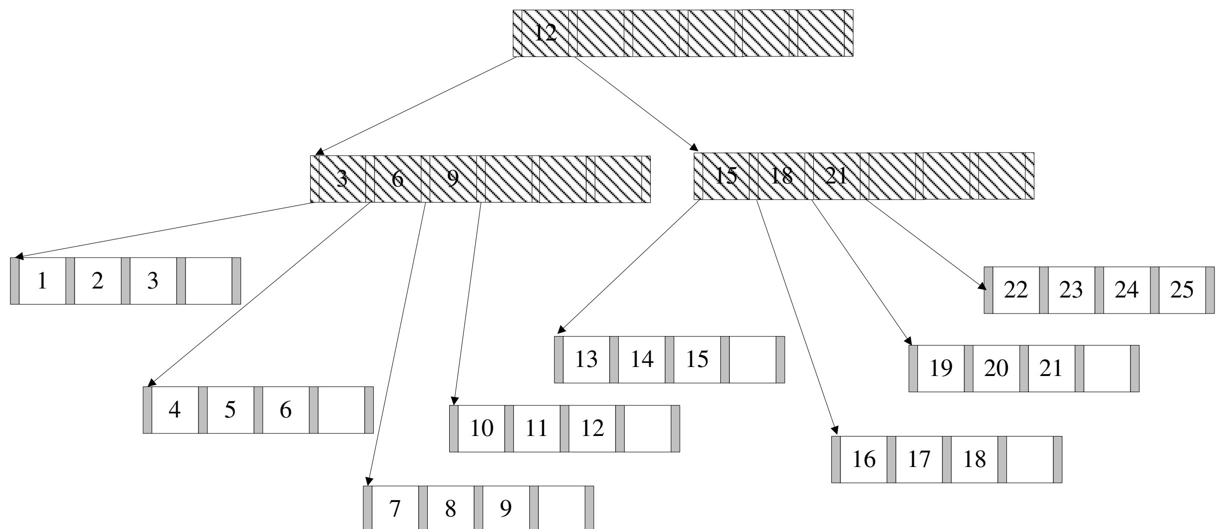
Bei 17 kommt es dann wieder zum Überlauf.



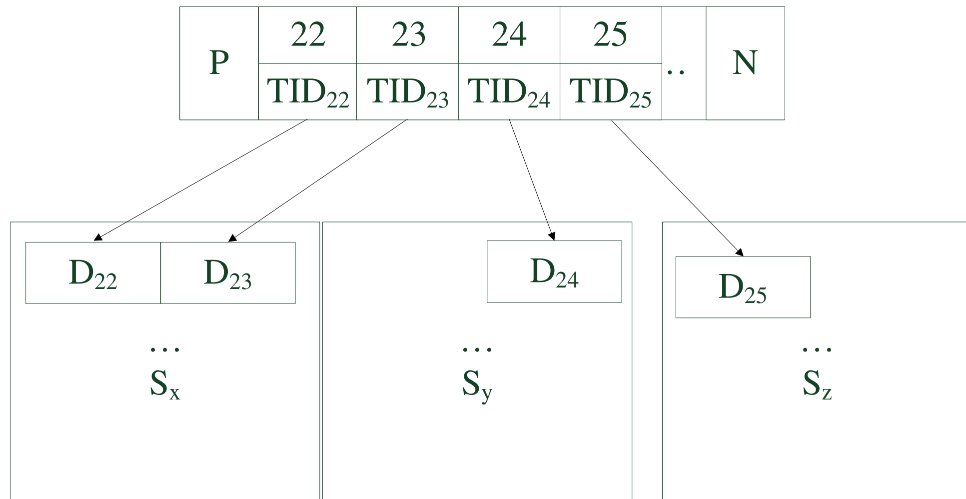
Die nächsten Zahlen werden wieder analog eingefügt. Bei 20 kommt es zum Überlauf.



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 Tupel. 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 4

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.

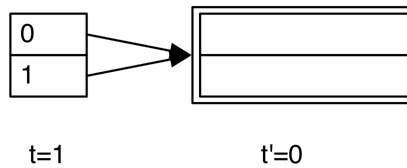
KundenNr	Name
10	Müller
25	Meier
30	Schmidt
18	Krause
40	Schulz
45	Kaufmann

Lösung:

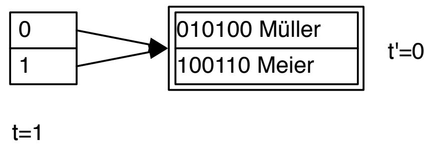
Werte mit binärer KundenNr sowie invers binärer Kundennummer, die für das Einfügen in den Hash genutzt wird:

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

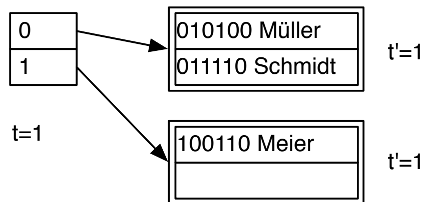
Zunächst eine leere erweiterbare Hashtabelle:



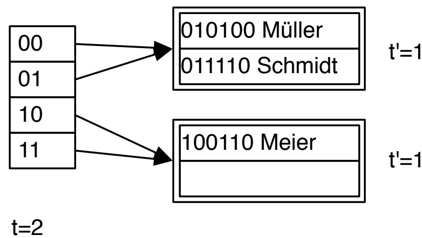
Wir fügen nun die ersten zwei Einträge ein, wonach die Hashtabelle wie folgt aussieht:



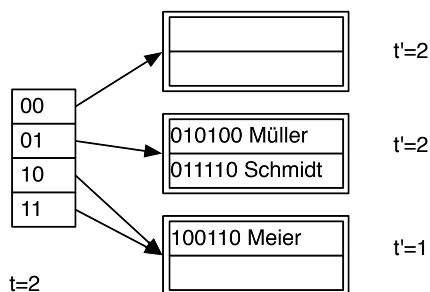
Der nächste Eintrag führt zu einem Überlauf. Da $t' < t$, können wir den Bucket teilen. Dies führt zur folgenden Hashtabelle:



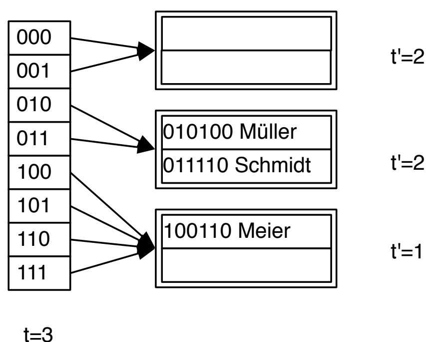
Das Einfügen von `Krause` führt erneut zu einem Überlauf. Da $t' = t$, können wir den Bucket aber nicht direkt teilen. Das Verzeichnis wird verdoppelt:



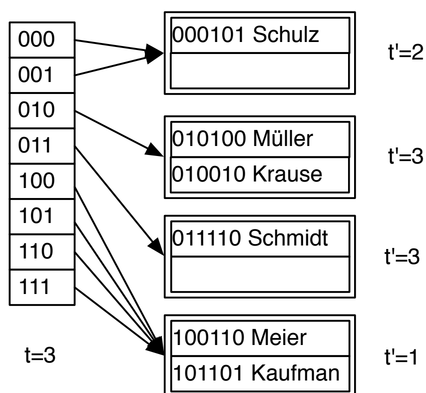
Nun kann der Bucket geteilt werden:



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



Nun kann der Bucket geteilt und alle Einträge eingefügt werden:



Hausaufgabe 5

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.