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

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS23/24

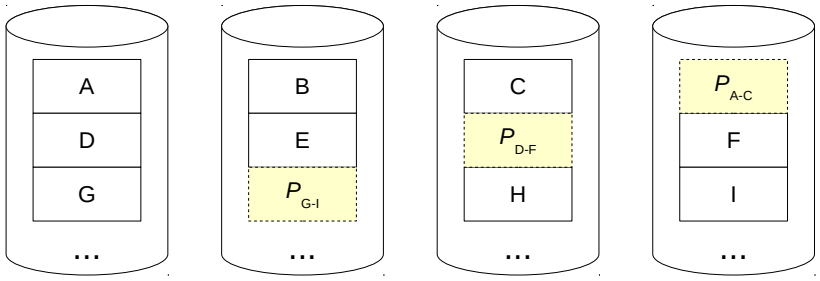
Christoph Anneser, Michael Jungmair, Stefan Lehner, Moritz Sichert, Lukas Vogel
(gdb@in.tum.de)

https://db.in.tum.de/teaching/ws2324/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? Begründen Sie!

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:

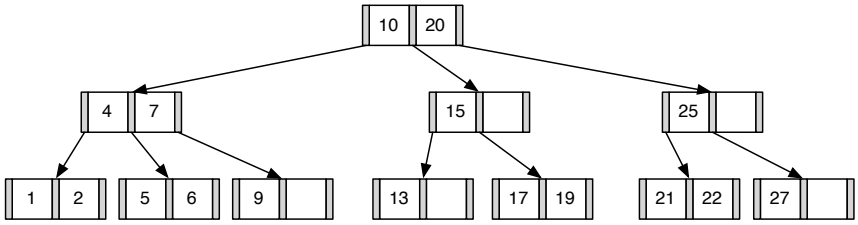
$$\begin{aligned} t_{total} &= t_{seek} + t_{read} \\ &= 10\ \text{ms} + \frac{10^9 \cdot 8\ \text{B}}{160 \cdot 10^6\ \frac{\text{B}}{s}} \\ &= 10\ \text{ms} + \frac{8 \cdot 10^9\ \text{B}}{16 \cdot 10^7\ \frac{\text{B}}{s}} \\ &= 10\ \text{ms} + 50\ \text{s} \\ &\approx 50\ \text{s} \end{aligned}$$

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 $(8 \cdot 10000 \text{ B})/(160 \cdot 10^6 \text{ B/s}) = 0.5$ ms ist vernachlässigbar. Insgesamt läuft das Programm also ungefähr 100 s.

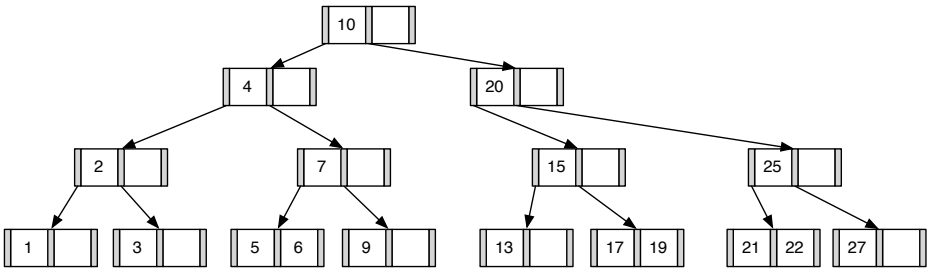
Trotz der „Optimierung“ hat diese Variante also eine längere(!) Laufzeit, als der „naive“ Ansatz aus Teilaufgabe b).

Hausaufgabe 3



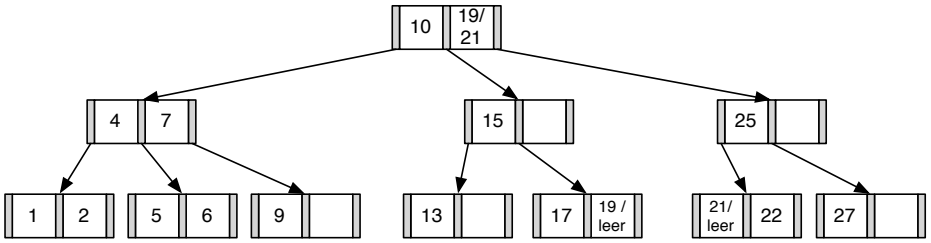
1. Fügen Sie die 3 in den gezeigten B-Baum ein. Zeichnen Sie das Endergebnis. Zeichnen Sie jeweils den kompletten Baum oder machen Sie **deutlich**, falls Teile des Baumes unverändert bleiben. Verwenden Sie den aus der Vorlesung bekannten Algorithmus.

   Lösung: Das Ergebnis sieht wie folgt aus:



2. Entfernen Sie aus dem **ursprünglichen Baum** den Eintrag 20. Zeichnen Sie das Ergebnis der Operation. Sollte es mehrere richtige Lösungen geben, genügt es, wenn Sie hier eine angeben. Zeichnen Sie jeweils den kompletten Baum oder machen Sie **deutlich**, falls Teile des Baumes unverändert bleiben. Verwenden Sie den aus der Vorlesung bekannten Algorithmus.

   Lösung: Das Ergebnis sieht wie folgt aus:



Hausaufgabe 4

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 16 KiB groß ist und ein Knoten des B-Baums möglichst genau auf diese Seite passen sollte.

(1 KiB = 2^{10} Byte)

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.

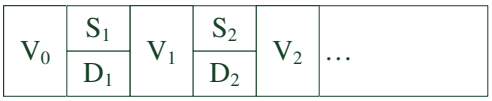


Abbildung 1: Struktur eines B-Baum Knotens

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 16 KiB 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:

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

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

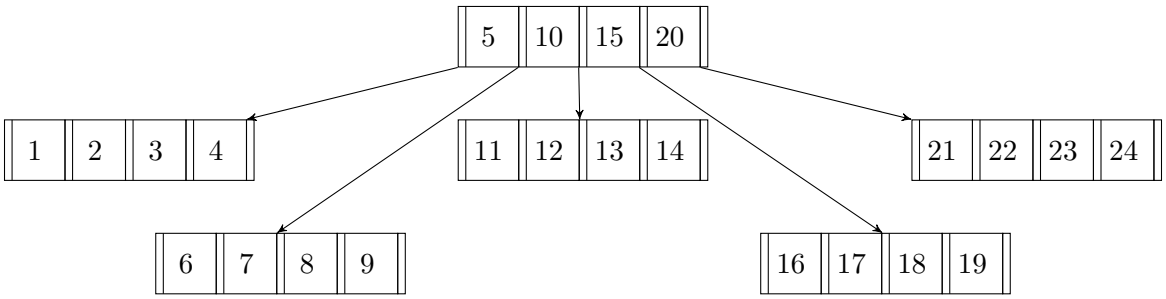
Hausaufgabe 5

Geben Sie eine Permutation der Zahlen 1 bis 24 an, so dass beim Einfügen dieser Zahlenfolge in einen (anfangs leeren) B-Baum mit Grad $k = 2$ ein Baum minimaler Höhe entsteht. Skizzieren Sie den finalen Baum.

Lösung:

Damit der Baum eine minimale Höhe erreicht, muss der Füllgrad aller Knoten möglichst hoch sein. Bei $k = 2$ passen vier Elemente in jeden Knoten. Bei 24 einzufügenden Zahlen

bedeutet das also, dass ein minimaler Baum insgesamt aus $\frac{24}{4} = 6$ Knoten besteht, die alle komplett gefüllt sind. Aufgrund der Invarianzen eines B-Baums muss dies ein Baum der Höhe 2 sein. Der einzig mögliche minimale Baum ist der folgende:



Damit die Zahlen 5, 10, 15 und 20 in die Wurzel geschrieben werden, muss die Einfügereihenfolge so gewählt werden, dass beim Teilen eines Knotens wegen Überlauf genau diese vier Zahlen als mittlere Elemente verwendet werden. Außerdem muss darauf geachtet werden, dass die Knoten nach dem Aufteilen wieder voll gefüllt werden.

Beispielsweise kann beim Einfügen mit den Zahlen 1, 2, 5, 6 und 7 in dieser Reihenfolge angefangen werden, wodurch die 5 dann in die Wurzel geschoben wird. Um die 10 in die Wurzel zu schreiben, können dann die Zahlen 10, 11, 12 eingefügt werden. Für die 15 dementsprechend die Zahlen 15, 16, 17 sowie für die 20 die Zahlen 20, 21, 22. Die übrigen Zahlen können dann in beliebiger Reihenfolge eingefügt werden.

Eine mögliche Permutation der Einfügereihenfolge, um einen Baum minimaler Höhe zu erhalten ist also: 1, 2, 5, 6, 7, 10, 11, 12, 15, 16, 17, 20, 21, 22, 3, 4, 8, 9, 13, 14, 18, 19, 23, 24.

Bonusaufgabe 6

Implementieren Sie einen B+-Baum in C++. Es sollten mindestens die Funktionen *insert* und *lookup* unterstützt werden. Zur Vereinfachung können Sie annehmen, dass lediglich Schlüssel-Wert-Paare bestehend aus Integern eingefügt werden.

Diese Aufgabe ist für diejenigen gedacht, die sich über den Vorlesungsstoff hinaus mit dem Thema Datenbanken beschäftigen wollen. Sie wird nicht in der Übung besprochen und ist nicht klausurrelevant. Falls Sie Feedback wünschen, können Sie Ihre Lösung gerne an `gdb@in.tum.de` *senden.*