Aufgabe 12

Die Seitenersetzungsstrategie *Clock* wird oft auch *Second-Chance* genannt. Unter diesem Namen führten sie Prof. Polze und Prof. Liggesmeyer in ihren Vorlesungen ein.

Dies ist die Pufferbelegung zu Beginn, wobei der veränderte Seitenrahmen grau hinterlegt und der als *current* behandelte blau markiert ist:

Seitenrahmen	**1**	**2**	**3**	**4**	**5 (current)**	**6**
`pin_count`	1	0	0	2	0	*(leer)*
Blocknummer	3	7	5	6	12	*(leer)*
`referenced`	0	0	0	0	1	*(leer)*

a) Block 25 wird angefordert:
Der Seitenrahmen 6 ist leer, in ihn wird der Block 25 geladen. Da er benutzt wird, muss sein `pin_count` auf 1 erhöht werden. Das `referenced` Bit wird beim Neuladen stets gelöscht.

Seitenrahmen	**1**	**2**	**3**	**4**	**5 (current)**	**6**
`pin_count`	1	0	0	2	0	*1*
Blocknummer	3	7	5	6	12	*25*
`referenced`	0	0	0	0	1	*0*

b) Block 7 wird angefordert:
Seite 7 ist bereits vorhanden (im Seitenrahmen 2), daher wird nur der `pin_count` erhöht und das `referenced` Bit gelöscht.

Seitenrahmen	**1**	**2**	**3**	**4**	**5 (current)**	**6**
`pin_count`	1	*1*	0	2	0	1
Blocknummer	3	7	5	6	12	25
`referenced`	0	0	0	0	1	0

c) Eine Anwendung gibt Block 3 frei:
Der `pin_count` des Seitenrahmen 1 wird auf Null, das `referenced` Bit auf Eins gesetzt:

Seitenrahmen	**1**	**2**	**3**	**4**	**5 (current)**	**6**
`pin_count`	*0*	1	0	2	0	1
Blocknummer	3	7	5	6	12	25
`referenced`	*1*	0	0	0	1	0

d) Eine Anwendung gibt Block 6 frei:
Der `pin_count` des Seitenrahmen 4 wird um 1 verringert und auf Eins gesetzt.

Seitenrahmen	**1**	**2**	**3**	**4**	**5 (current)**	**6**
`pin_count`	0	1	0	*1*	0	1
Blocknummer	3	7	5	6	12	25
`referenced`	1	0	0	0	1	0

e) Block 25 wird erneut angefordert:
Block 25 ist bereits geladen worden, daher erhöht sich der `pin_count` auf 2.

Seitenrahmen	**1**	**2**	**3**	**4**	**5 (current)**	**6**
`pin_count`	0	1	0	1	0	*2*
Blocknummer	3	7	5	6	12	25
`referenced`	1	0	0	0	1	0

f) Block 2 wird angefordert:
Alle Seitenrahmen sind belegt, eine Verdrängung muss vorgenommen werden. Ausgehend vom als *current* gesetzten Seitenrahmen 5 wird untersucht, welcher Seitenrahmen ausgelagert werden kann. Seitenrahmen 6 hat einen `pin_count` der größer als Null ist. Zwar trifft dies für den als nächstes betrachteten Seitenrahmen 1 nicht mehr zu, allerdings ist dessen `referenced` Bit gesetzt. Nachdem es gelöscht wurde, darf zwar nicht Rahmen 2 verdrängt werden (`pin_count` größer Null), aber Seitenrahmen 3 erfüllt alle Bedingungen für eine Verdrängung:

Seitenrahmen	**1**	**2**	**3 (current)**	**4**	**5**	**6**
`pin_count`	0	1	*1*	1	0	2
Blocknummer	3	7	*2*	6	12	25
`referenced`	*0*	0	0	0	1	0

g) Block 4 wird angefordert:
Erneut ist eine Verdrängung notwendig. Die `pin_count`-Bedingung wird vom Seitenrahmen 5 erfüllt, da dessen `referenced` Bit aber gesetzt ist, muss es gelöscht werden und der nächste Rahmen betrachtet werden. Für Rahmen 6 gilt die `pin_count`-Bedingung nicht, jedoch kann Rahmen 1 überschrieben werden:

Seitenrahmen	**1 (current)**	**2**	**3**	**4**	**5**	**6**
`pin_count`	1	1	1	1	0	2
Blocknummer	4	7	2	6	12	25
`referenced`	0	0	0	0	0	0

Aufgabe 13

Die gegebenen Werte sind (*B* - Anzahl Seiten, *R* – Einträge pro Seite, *D* – mittlerer Dateizugriff, *C* – Berechnung auf Record, *H* – Auswertung Hashfunktion):

$$B = \frac{1000\ kB}{4\ kB} = 250$$

$$R = \frac{10.000}{B} = 40$$

$$D = 10\ ms$$

$$C = 0\ ms$$

$$H = 0\ ms$$

a) Für *Delete* gebe ich zwei Zeiten an: für das Löschen eines Wertes und für das Löschen eines Bereiches. Sollte gar die ID der/des zu löschenden Tupel vorhanden sein, so kann *Search* komplett weggelassen werden, was die Zeit drastisch reduziert (*Heap* und *Hashed file* auf 0,01s, *Sorted file* auf 2,5s).

	Heap file	**Sorted file**	**Hashed file**
Scan	*BD* = 2,5s	*BD* = 2,5s	*1,25BD* = 3,125 s
Einzelwertanfrage	*0,5BD* = 1,25s	$D \cdot log_2 B$ ≈ 0,08s	*D* = 0,01s
Bereichsanfrage	*BD* = 2,5s	$D \cdot (log_2 B + matched\ pages)$ ≈ 0,01s·(8+matched pages)	*1,25BD* = 3,125 s
Insert	*2D* = 0,02s	*Search + BD* ≈ 2,58s	*2D* = 0,02s
Delete	*Search + D* = 1,26s bzw. 2,51s	*Search + BD* ≈ 2,58s bzw. 2,58s+0,01s·matched pages	*Search + D* = 0,02s bzw. 3,135s

b) Die besten Antwortzeiten für Einzelwertanfragen liefert ein Hashed File (lediglich abhängig von *D*), daher fällt meine Wahl darauf.

c) In diesem Fall hilft ein Index auf das angefragte Attribut weiter. Da Leseoperationen mit einem Dense Index schneller erfolgen als mit einem Sparse Index und Bereichsanfragen die vorsortierte Anordnung der Datensätze hintereinander nutzen können, wäre eine Sorted-File-Struktur zu bevorzugen. Es sind nur wenige Extremsituationen konstruierbar, wo fast alle Seiten Treffer enthalten, so dass log_2B+matched pages)>B gilt und ein Heap-File besser wäre.

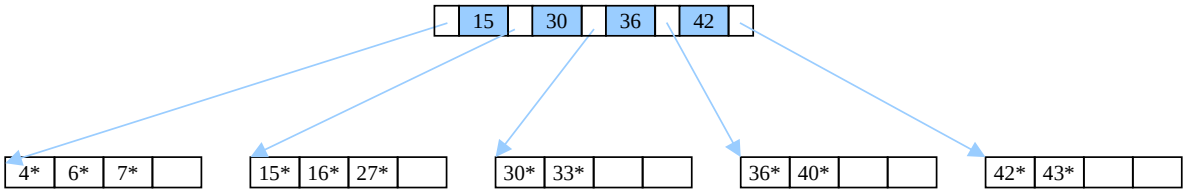
d) Ein Vergleich des Zeitaufwandes für Einzelwertanfragen und Bereichsanfragen kommt zu dem Schluss, dass ein Sorted-File eindeutig am schnellsten ist. Rechnerisch überprüfe ich dies, indem ich den Mittelwert der Antwortzeiten beider Anfragemethoden bilde:

	Heap file	**Sorted file**	**Hashed file**
(Einzelwertanfrage + Bereichsanfrage) / 2	*(0,5BD + BD) / 2* = 1,875s	*(D·log₂B +* *D·(log₂B+matched pages)) / 2* ≈ 0,08s + 0,05s·matched pages	*(D + 1,25BD) / 2* = 1,5675 s

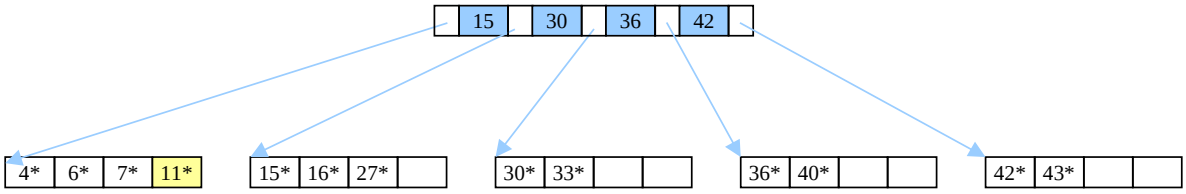
Der Wert für matched pages kann maximal 250 erreichen, so dass selbst im ungünstigsten Fall ein Sorted-File 1,33s benötigt.

Aufgabe 14

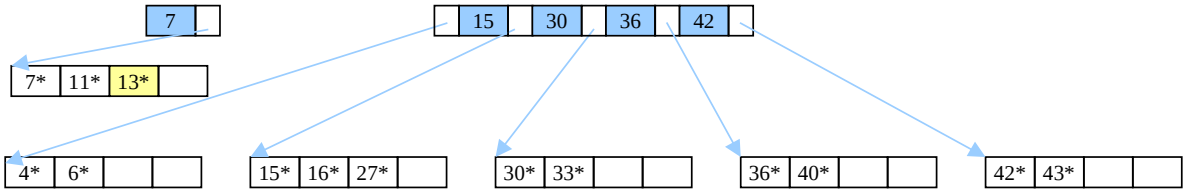
Anfänglich hat die Datenbank diese Struktur als B+ Baum:



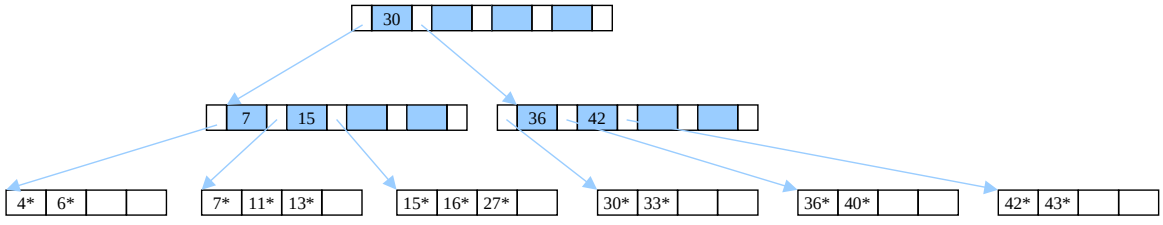
Um den Wert 11 einzufügen, muss man das linke Blatt füllen:



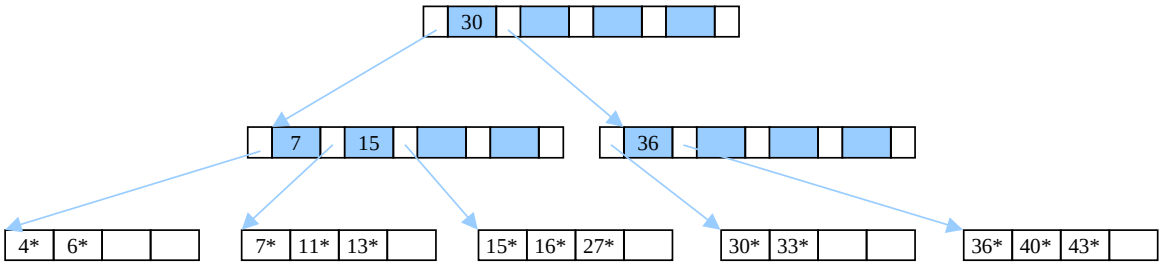
Will man den Wert 13 einfügen, so stellt man fest, dass dafür eigentlich das linke Blatt zuständig wäre. Leider ist es bereits gefüllt, so dass eine Aufspaltung notwendig wird. Der temporär erzeugte B+ Baum sieht dann so aus:



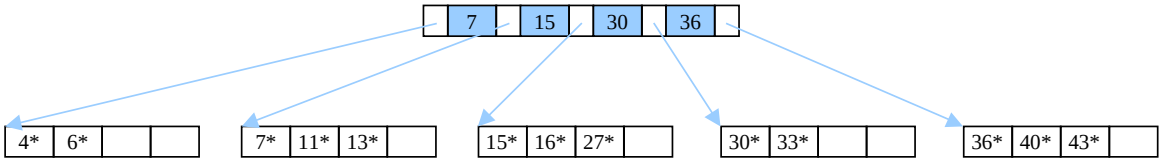
Die 7 ist nun als Dateneintrag *und* als Indexeintrag vorhanden. Letzterer muss in den Wurzelknoten eingefügt werden, was wiederum zu einem Überlauf führt, der eine Spaltung nach sich zieht. Von den fünf Indexeinträgen in der Wurzel wird die 30 als mittelster Wert ausgewählt, um den neuen Wurzelknoten zu bilden. Einfügen von 13 führt also zu dem Baum:



Die Entfernung von 42 bedeutet, dass im rechten Blatt der Füllgrad von 50% unterschritten wird, was zur Auflösung dieses Blattes führt. Eine Umverteilung ist nicht möglich, da alle benachbarten Blätter ebenfalls nur zu 50% gefüllt sind. Als einziger Ausweg ist eine Verschmelzung (Merge) möglich. Dabei wird aber der Indexeintrag 42 überflüssig. Temporär sieht der Baum so aus:



Man stellt fest, dass die 36 als Indexeintrag ebenfalls das 50%-Füllkriterium unterschreiten und aufgelöst werden muss. Dazu wird der Wurzelknoten aufgelöst und die 30 „heruntergezogen“. Der neue Wurzelknoten ist die Verschmelzung der beiden Knoten, die bisher zwischen der Wurzel und den Blättern lagen. Der endgültige B+ Baum nach Löschung von 42 hat die Gestalt:



Die Struktur eines ISAM-Bäume wird bei der Erzeugung festgelegt und kann nachträglich nicht oder kaum geändert werden. Sollten die anfänglich getroffenen Annahmen über die zu speichernden Daten falsch sein, so müssen viele Overflow-Seiten angelegt werden, die die Zugriffszeiten drastisch erhöhen.
B+ Bäume besitzen, im Gegensatz zu ISAM-Bäumen, eine dynamische Struktur, die höhenbalanciert ist. Wertverändernde Zugriffe (Einfügen, Löschen) verändern diese Struktur, was einen gewissen Aufwand zur Folge hat, aber für konstant schnelle Lesezugriffe sorgt.
Beide Strukturen zählen zu den effizientesten Datenbankstrukturen und sind weitverbreitet. Ihre Indexe sind clustered und sparse.