B-Baum

- Bisher sind wir davon ausgegangen, dass Knoten ohne Problem adressiert werden können.
- Ein Problem entsteht, wenn wir zum Betrachten eines Knotens eine aufwändige Speicheroperation durchführen müssen:
  - Im Arbeitsspeicher können wir byteweise adressieren.
  - Von der Festplatte (Sekundärspeicher) werden immer größere Einheiten (Seiten einer Festplatte) adressiert und auf einmal in den Arbeitsspeicher geladen. Diese Blöcke umfassen in der Regel die vielfache Größe eines Knotens.
- Ein Zugriff auf die Festplatte ist in etwa um den Faktor 10^3 langsamer als der Zugriff auf den Arbeitsspeicher.

B-Baum

- **Ziel:** Vermeide es, unnötige Daten von der Festplatte zu lesen und öfter als nötig auf diese zuzugreifen.
- **Ansatz:**
  - Gruppiere so viele Schlüssel in einem Knoten, dass diese möglichst exakt den Platz einer Seite füllen.
  - Innerhalb eines Knotens müssen die Schlüssel so abgelegt werden, dass sie eine effiziente Suche ermöglichen.
  - Wir können nun einen Knoten vom Sekundärspeicher in den Arbeitsspeicher lesen, und dann diesen Knoten ohne einen weiteren Zugriff auf den Sekundärspeicher durchsuchen.
- Eine balancierte Baumstruktur, die dieses ermöglicht, ist der B-Baum.

B-Baum

B-Baum / B^+ -Baum (*B= balanced*)

- Verallgemeinerung der 2-3 Bäume
- dynamischer, eindimensionaler Zugriffspfad
- Festlegung der Knotengröße des Baumes auf der Seitenkapazität des Sekundärspeichers
  => 1 Knoten = 1 Seite auf Sekundärspeicher
- Max. Seitenzugriffe durch Höhe des Baumes begrenzt
- Balanciert

B-Baum

Für einen B-Baum der Ordnung *m* gilt:

1. Jeder Wert von der Wurzel zu einem Blatt hat die gleiche Länge.
2. Die Wurzel ist entweder ein Blatt oder hat mind. zwei Söhne.
3. Jeder Knoten hat mindestens $\lceil m/2 \rceil$ und höchstens m Söhne.
4. Jeder Knoten mit k Söhne speichert $k-1$ Schlüssel.
5. Einträge in allen Knoten sind immer sortiert.
6. Ein Blatt belegt immer jeweils einen (Speicher-)Block.
7. Sind $T_0, \ldots, T_k$ die k Teilbäume der Söhne eines Knotens und $s_1, \ldots, s_k$ seine Schlüssel, dann speichert:
   - T_0 nur Schlüssel kleiner s_1
   - T_k nur Schlüssel größer s_k
   - T_i nur Schlüssel zwischen s_i und s_{i+1} für alle $1 \leq i < k$

Die Verweise bei den Blättern sind leer (NIL-Pointer)

$\lceil m/2 \rceil$ = die kleinste ganze Zahl, die größer oder gleich ist wie m/2.

z.B. für m=5 -> m/2 = 2,5 -> $\lceil m/2 \rceil$ =3

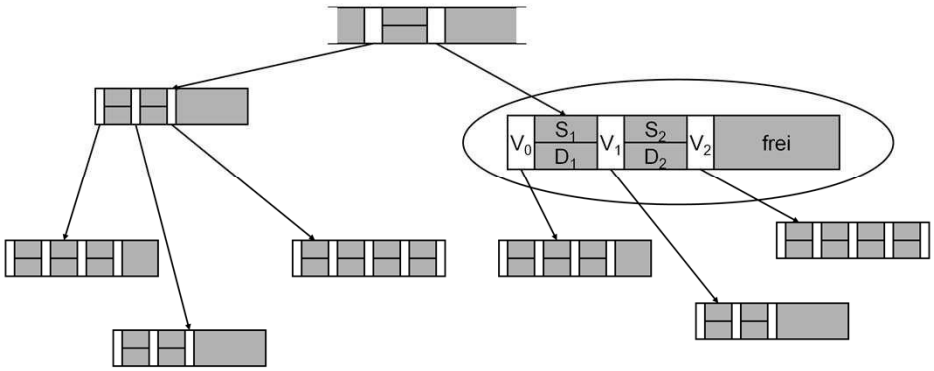
B-Baum

Für einen B-Baum mit dem Verzweigungsgrad *k* gilt:

1. Jeder innerer Knoten, außer der Wurzel, hat mind. *k* und max. *2k* Einträge

2. Alle Knoten mit *n* Einträge, außer den Blättern, haben *n*+1 Kinder

B-Baum



V_i : Verweis auf Kind-Knoten
S_i : Schlüsselwert
D_i : Datensatz / -wert und TID (Tupel-Identifikator)

B-Baum

- Passt ein Knoten genau auf eine Plattenseite, dann ist der Nachfolgerzeiger ein Verweis auf den Sekundärspeicher.
- Solche Verweise sind Systemabhängig und können von Sekundärspeicher zu Sekundärspeicher sehr unterschiedlich aussehen.
- Wir gehen daher von der Existenz eines Datentyps

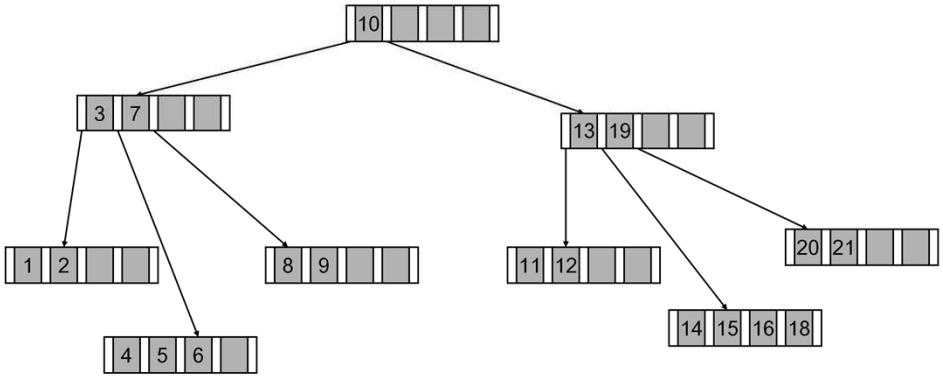
  pointer to page

  aus.
- Ein Knoten kann wie folgt realisiert werden:

```
1: type knoten=
2:    record
3:          keys : array[1..m-1] of schluesseltyp;
4:          sons : array[1..m] of pointer to page;
5:    end;
```

B-Baum

Ein Beispielbaum (m=5)



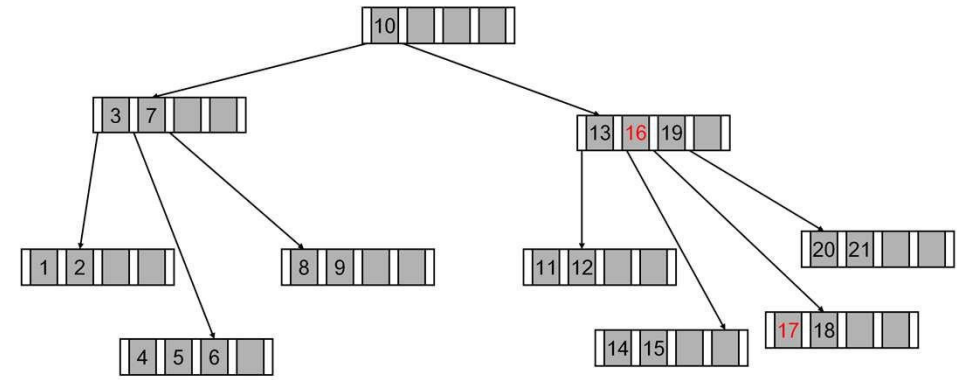
B-Baum

Einfüge-Algorithmus:

1. Führe eine Suche nach dem Schlüssel durch; diese endet (scheitert) an der Einfügestelle
2. Füge den Schlüssel dort ein.
3. Ist der Knoten überfüllt, teile ihn
   - Erzeuge einen neuen Knoten und belege ihn mit den Einträgen des überfüllten Knotens, deren Schlüssel größer ist als der des mittleren Eintrags.
   - Füge den mittleren Eintrag im Vaterknoten des überfüllten Knotens ein.
   - Verbinde den Verweis rechts des neuen Eintrags im Vaterknoten mit dem neuen Knoten
4. Ist der Vaterknoten jetzt überfüllt?
   - Handelt es sich um die Wurzel, so lege eine neue Wurzel an.
   - Wiederhole Schritt 3 mit dem Vaterknoten

B-Baum

Einfügen einer 17



B^+-Baum

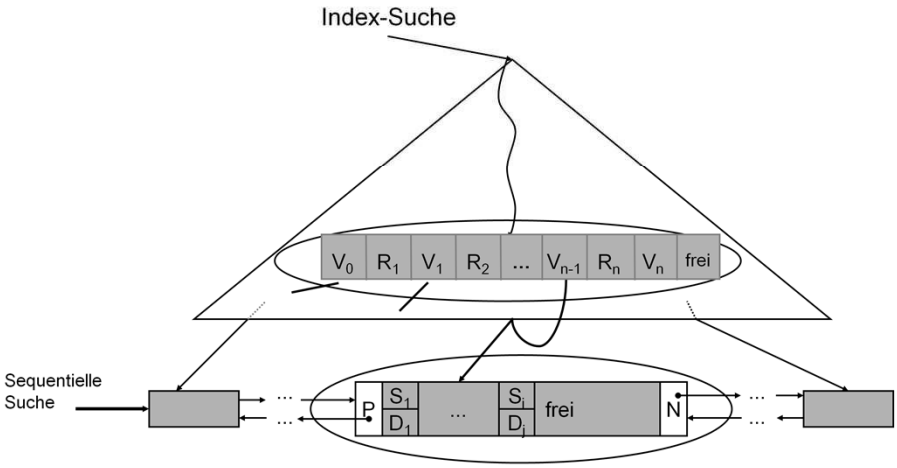
B^+-Baum:

- Knoten beinhalten nur Referenzen
- Referenzen in den Knoten sind sog. Wegweiser („raod map“)
- Referenzschlüssel müssen nicht unbedingt einem realen Schlüssel entsprechen
- Daten nur in den Blättern
- Blätter mit vor- und rückwärts Verweis für schnelle sequentielle Suche

Vorteile:

- geringe Höhe durch einen hohen Verzweigungsgrad
- weniger Seitenzugriffe als B-Baum, um ein Datum zu finden

B+-Baum


Index-Suche
V0 R1 V1 R2 ... Vn-1 Rn Vn frei
Sequentielle Suche
P
S1 D1
...
Si Di
frei
N

B^+-Baum

B^+-Baum vom Typ (*k*,*k) hat folgende Eigenschaften:**

1. Jeder Weg entlang der Wurzel bis zu einem Blatt ist immer gleich lang
2. Jeder Knoten, außer der Wurzel, hat mind. *k* und max. *2k* Einträge.
   Blätter haben mind. *k** und max. *2k** Einträge.
   Die Wurzel hat max. *2k* Einträge oder ist selber ein Blatt mit max. *2k** Einträge.
3. Jeder Knoten mit *n* Einträge, außer den Blättern, hat *n* + 1 Kinder.
4. Seien R_1, ..., R_n die Referenzschlüssel eines inneren Knotens (auch der Wurzel) mit *n*+1 Kindern. Seien V_0, ..., V_n die Verweise auf diese Kinder.
   (a) V_0 verweist auf den Teilbaum mit Schlüsseln kleiner oder gleich R_1.
   (b) V_i (i = 1, ..., n - 1) verweist auf den Teilbaum, dessen Schlüssel zwischen R_i und R_{i+1} liegen. (einschliesslich R_{i+1}).
   (c) V_n verweist auf den Teilbaum mit Schlüsseln größer als R_n.

Anzahl der Seitenzugriffe

Bäume benötigen Seitenzugriffe in der Ordnung von:

$$\log_k (n)$$

Hash-Tabellen in der Regel:

1 bis 2 Seitenzugriffe

k = Verzweigungsgrad
n = Anzahl der eingetragenen Datensätze