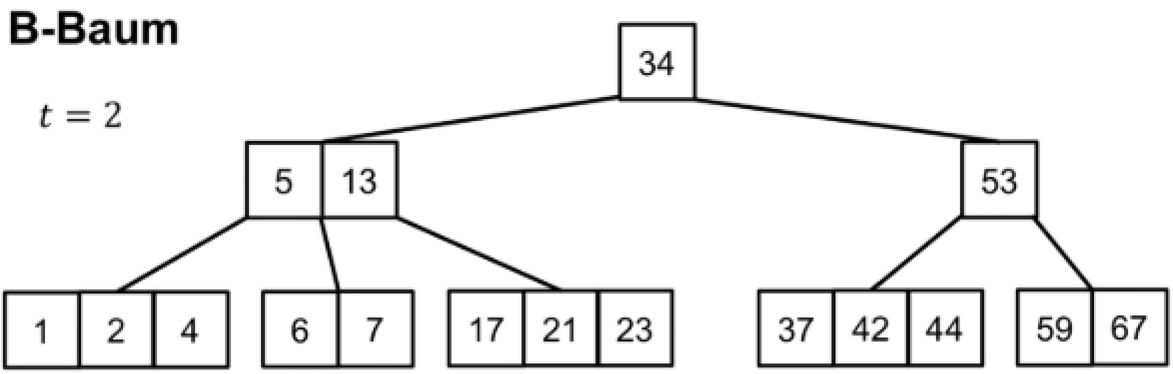
B-Bäume

Ein B-Baum von Grad t ist ein Baum, bei dem

1. jeder Knoten außer der Wurzel zwischen $t-1$ und $2t-1$ Werte `key[0], key[]1.....` hat, die Wurzel hat zwischen 1 und $2t-1$ Werte
2. Die Werte innerhalb eines Knoten sind aufsteigen geordnet
3. die Blätter haben alle die gleiche Höhe

- jeder innerer Knoten mit n Werten n+1 Kinder hat, so dass für alle Kinder k_j aus dem j-ten Kind gilt: $k_0 \leq key[0] \leq k_1 \leq key[1] \cdots \leq k_{n-1} \leq key[n-1] \leq k_n$ also von links nach rechts geordnet, dabei gilt links < rechts



- t = 2, also min 1 und max 3
  `x.n` = Anzahl Werte eines Knoten x
  `x.key[0],....x.key[x.n-1]` = geordnete Werte in Knoten x
  `x.child[0],...,x.child[x.n]` = Zeiger auf Kinder in Knoten x

Höhe B-Baum

- mindestens 1 Wert in Wurzel
- mindestens 2 Knoten in Tiefe 1 mit jeweils mindestens t kindern
- mindestens 2t Knoten in nächster Tiefe mit jeweils mindestens t Kindern
- mindestens $2t^2$ Knoten in nächster Tiefe mit jeweils mindestens t kindern
- usw
- In jedem Knoten außer Wurzel mindestens t - 1 Werte
- Anzahl Werte n im Baum im Vergleich zur Höhe h: $n \geq 2t^h - 1$, also $\log_t \frac{n+1}{2} \geq h$
- ~ Ein B-Baum vom Grad t mit n Werten hat maximale Höhe $h \leq \log_t \frac{n+1}{2}$
- je Größer t, desto flacher der Baum

Anwendung:

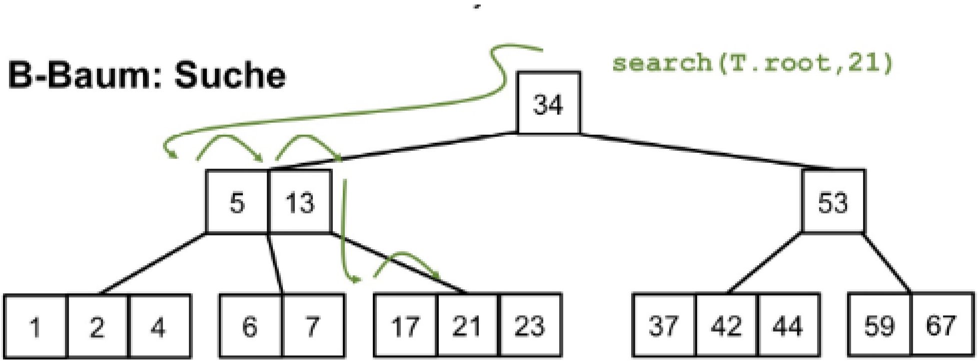
- MySQL speichert Werte in B-Bäumen
- Lesen/Schreiben in Blöcken: mehrere Werte (z.B.Index-Einträge) auf einmal

Suche

`search(x,k)`

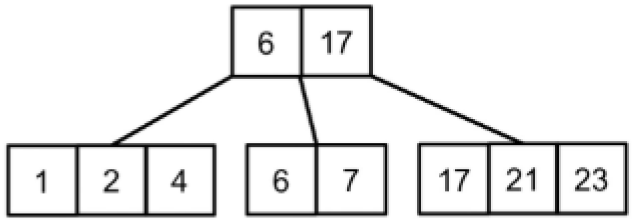
```
WHILE x != nil DO
    i=0; // initialisierung des Startindex
    WHILE i < x.n AND x.key[i] < k DO i = i+1;
    IF i < x.n AND x.key[i]==k THEN // Schlüssel gefunden
        return (x,i);
    ELSE
        x=x.child[i]; // Anderer Fall, gehe zu Kindknoten
return nil;
```

- `x.n` gibt die Anzahl der Schlüssel im Knoten x an
- `x.key[i]` ist der `i`-te Schlüssel im Knoten `x`.
- Laufzeit $O(t * h) = O(\log_t n)$



Baumkunde

- B-Baum vom Grad t: max $2t$, min. t Kinder pro Knoten $\neq$ Wurzel
  - Alternativ: max t, min $\frac{t}{2}$ Kinder pro Knoten $\neq$ Wurzel
- 2-3-4 Baum oder (2,4)- Baum : B-Baum mit $t = 2$
- B+-Baum: alle Werte in Blättern, innerer Knoten enthalten Werte erneut
  Vorteil: innere Knoten speichern nur kurzen Schlüssel, nicht auch nich Daten(-zeiger)
  Nachteil: findet Werte erst im Blatt



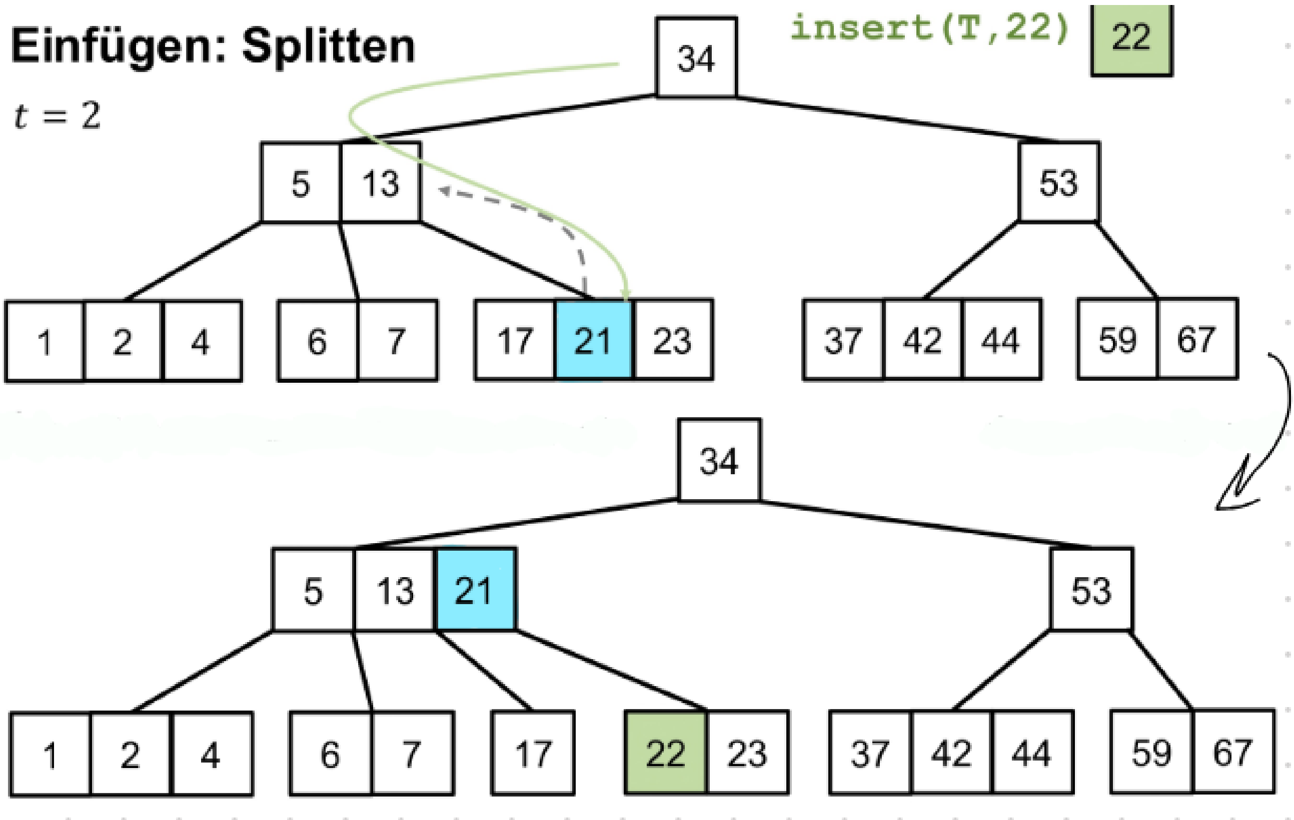
Einfügen

Idee

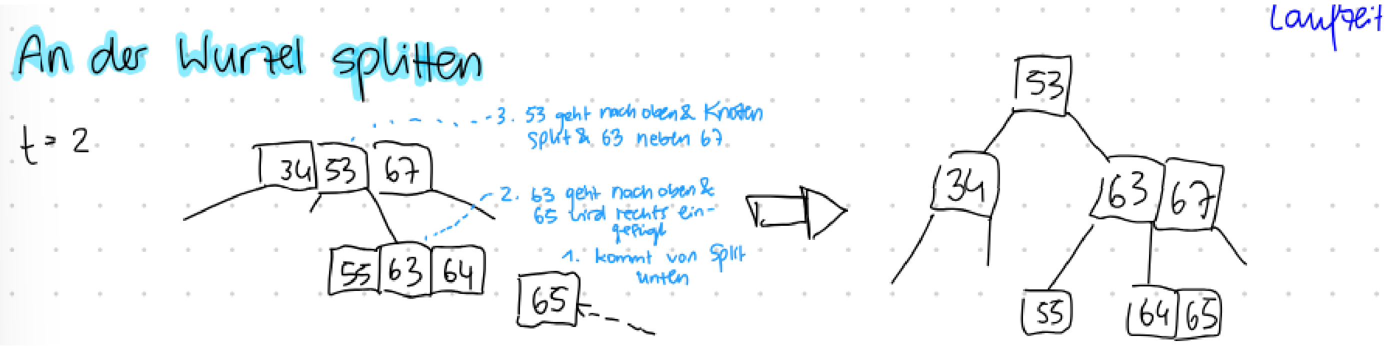
- Einfügen erfolgt immer in einem Blatt
- Wenn Blatt weniger als $2t - 1$ Werte hat, dann einfügen und fertig
- wenn nicht :

Splitten

- Wenn Blatt bereits $2t - 1$ Werte hat, dann teile es in zwei Blätter mit je $t - 1$ Werten, füge mittleren Wert im Elternknoten ein
- Wenn dadurch Elternknoten mehr als $2t - 1$ Werte hat, rekursiv nach oben
- Splitten an der Wurzel : Neue Wurzel wird erzeugt, Höhe des Baumes wächst um 1. B-Baum-Einfügen splittet beim Suchen und läuft nur einmal hinab, sonst werden teure Disk-Operationen zweimal ausgeführt, einmal beim ab, einmal beim aufsteigen



An der Wurzel splitten



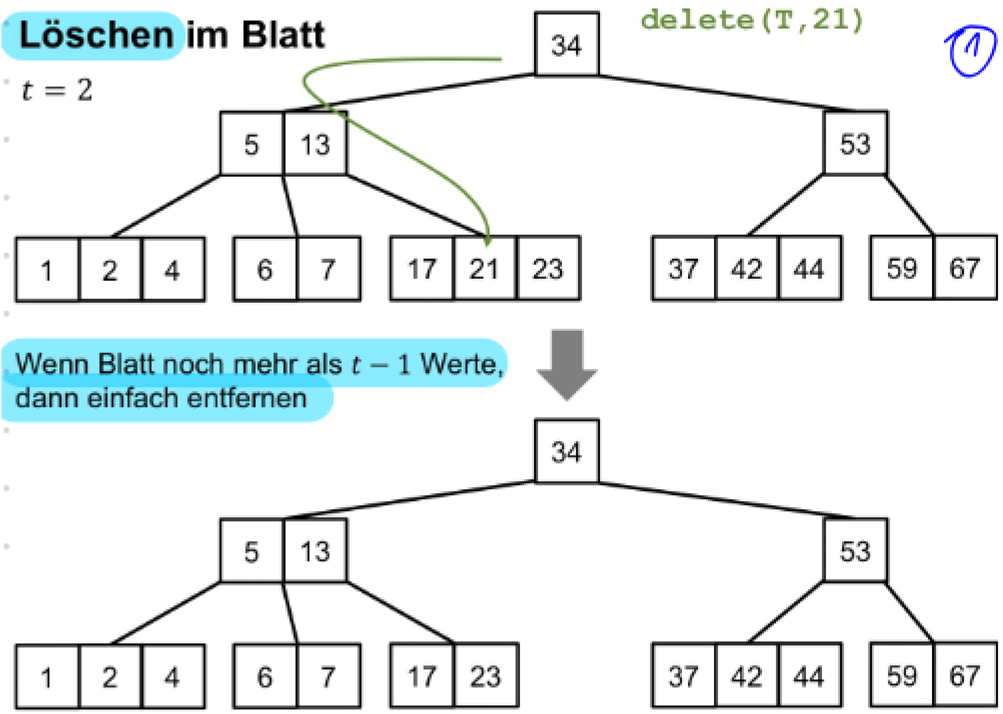
`insert(T,z)`

```
Wenn Wurzel schon 2t-1 Werte hat, dann splitte Wurzel
Suche Rekursiv Einfügeposition
    Wenn zu besuchendes Kind 2t-1 Werte hat,splitte es erst
füge z in Blatt ein
```

Laufzeit: $O(t * h) = O(\log_t n)$

Löschen

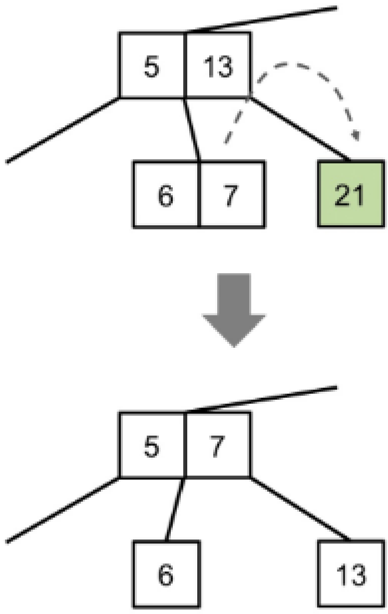
- Wenn Blatt mehr als $t - 1$ Werte, kann der Wert einfach entfernt werden



Löschen im Blatt

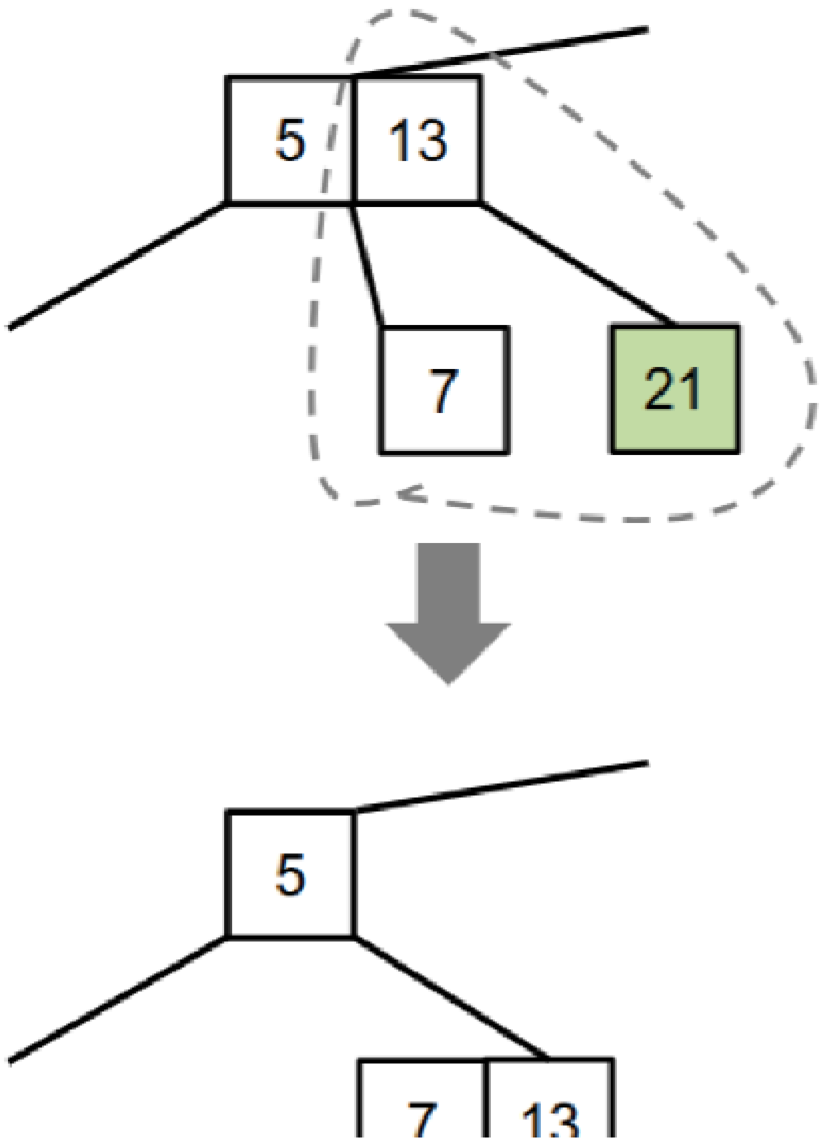
- Wenn $t - 1$ Werte im Blatt mit zu löschendem Wert sind, linker oder rechter Geschwisterknoten hat min. t Wertem dann rotiere Werte von Geschwisterknoten und Elternknoten

$t = 2$



jeweils mind. $t - 1$ Werte

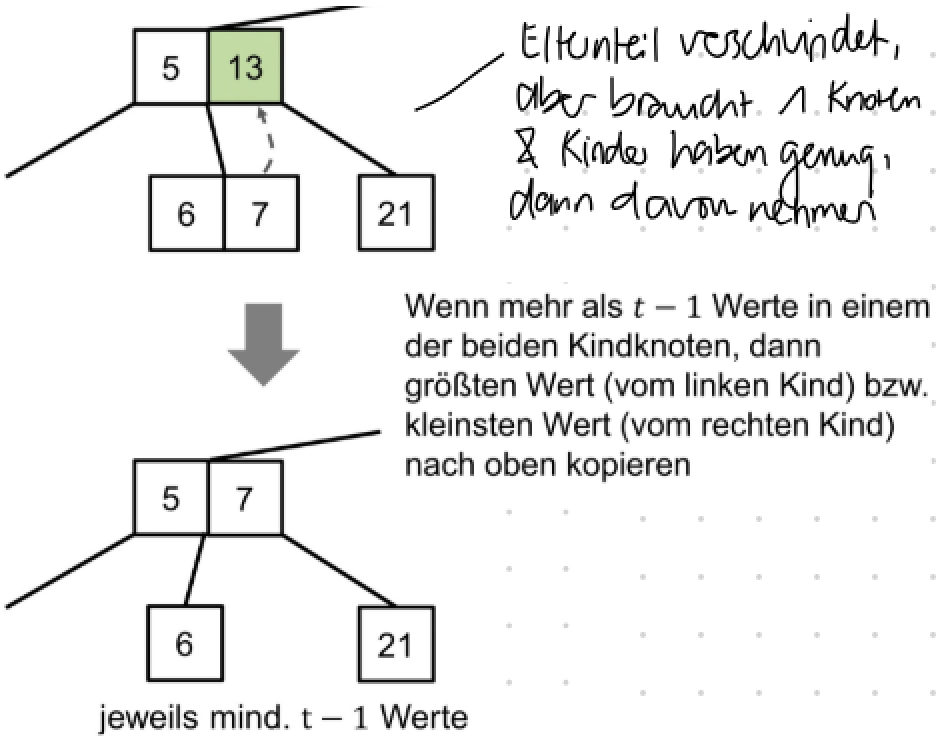
- Wenn $t - 1$ Werte im Blatt mit zu löschendem Wert sind, linker oder rechter Geschwisterknoten hat mind. t Werte, dann rotiere Werte von Geschwisterknoten und Elternknoten



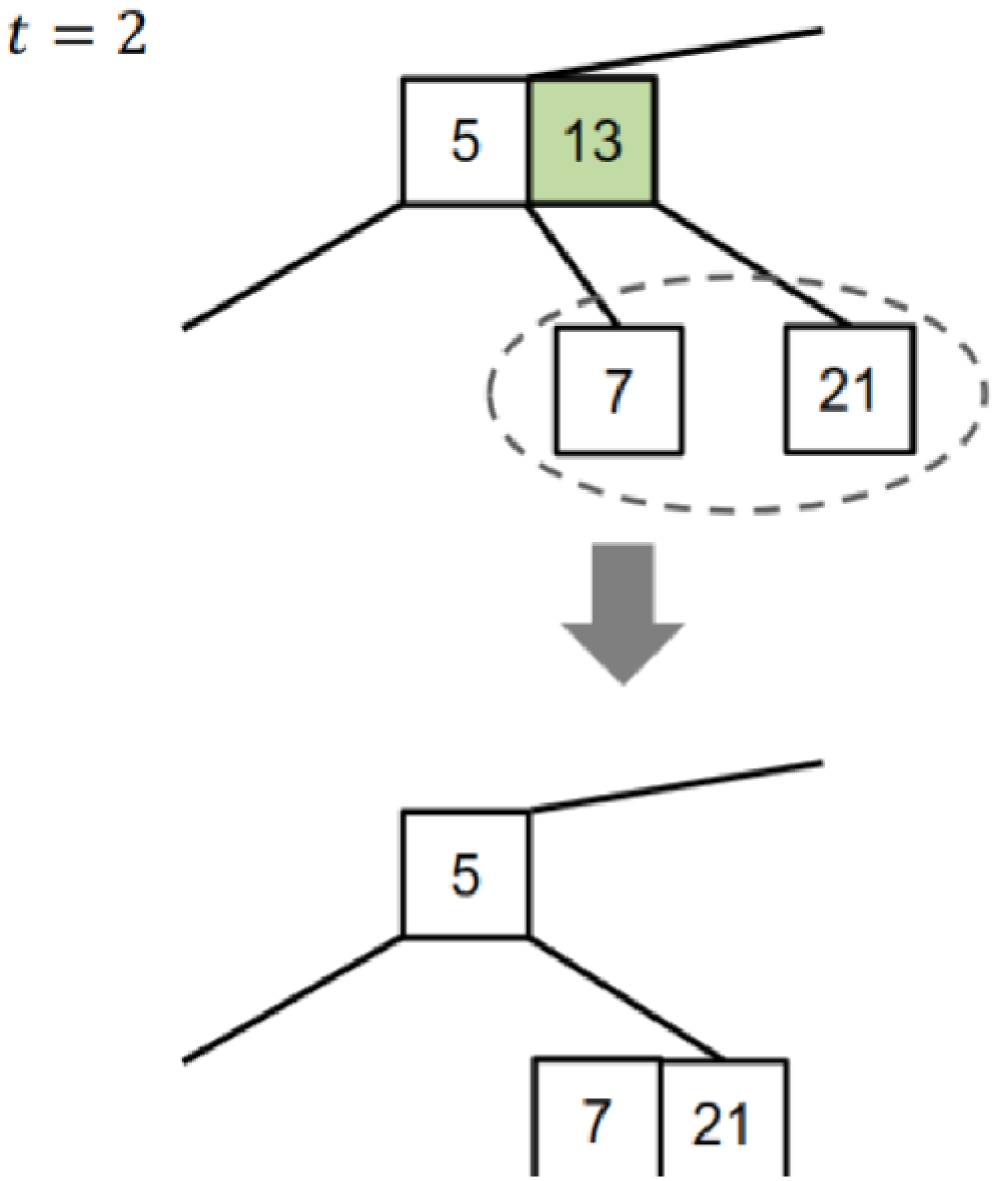
-
  -maximal $t-2+t-1+1=2t-2$ Werte

Löschen im inneren Knoten

- Verschieben:
  Wenn sich mehr als $t-1$ Werte in einem der beiden Kindknoten befinden, dann größten Wert (vom linken Kind) bzw vom kleinstem Wert (vom rechten Kind) nach oben Kopieren

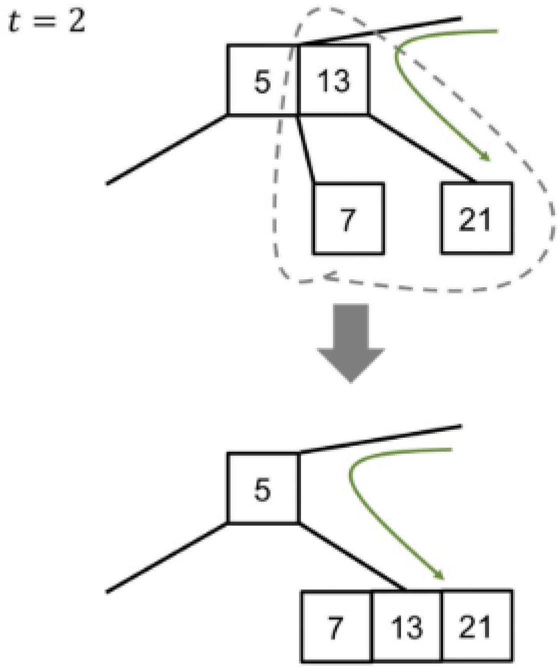


- Verschmelzen
  Wenn sich jeweils $t-1$ Werte in beiden Kindknoten befinden, dann Kindknoten verschmelzen, eventuell hat Elternknoten nun zu wenige Werte



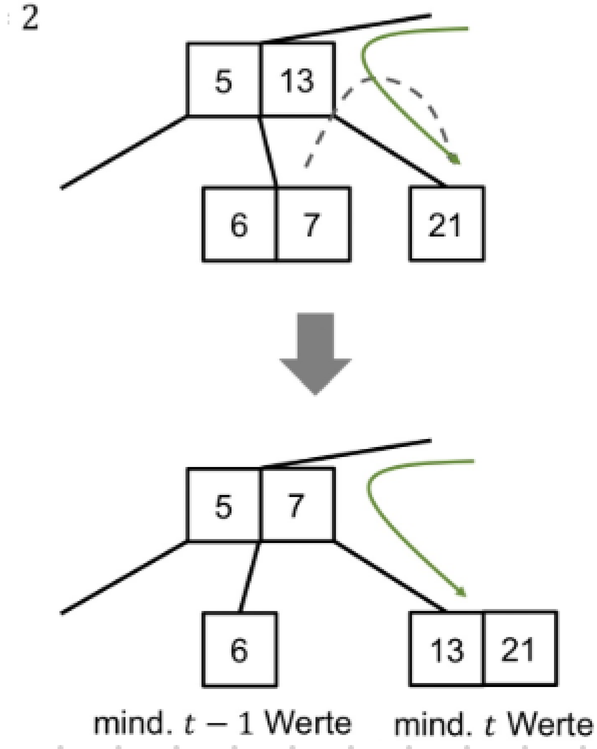
Allgemeines Verschmelzen ohne Löschen

zu besuchendes Kind, rechter, linker Geschwisterknoten (sofern existent) haben $t-1$ Werte, dann ist Verschmelzen ohne weitere Änderungen möglich, wenn der Elternknoten vorher mindestens t Werte hat



Allgemeines Rotieren/Verschieben ohne Löschen

Zu besuchendes Kind hat nur $t-1$ Werte, aber ein Geschwisterknoten hat mehr als t-1, dann kann man dies ohne Änderungen oberhalbtun.
<



informelles Löschen

`delete(T,k)`

```
Wenn Wurzel nur 1 Wert hat und beide Kinder t-1 Werte, verschmelze Wurzel und Kinder (reduziert
Höhe um 1)
```

```
        Wenn zu besuchendes Kind nur t-1 Werte,
        verschmelze es oder rotiere/verschiebe
    Entferne Wert k in innereren Knoten/Blatt
```

Worst-Case-Laufzeiten

- Einfügen, Löschen, Suchen: $\Theta(\log_t n)$
- O- Notation versteckt konstanten Faktor t für Suche innerhalb eine Knoten: $t * \log_t n = t * \left(\frac{\log_2 n}{\log_2 t}\right)$ ist in der Regel größer als $\log_2 n$, also nur vorteilhaft, wenn Daten eingelesen werden