Informatik 2 mit BlueJ

Ein Kurs für die Stufe Q1

von Ulrich Helmich

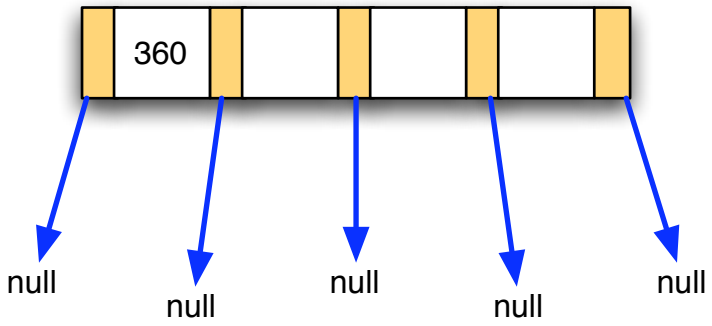
zuletzt überarbeitet am 16. Juni 2020 (B-Bäume: Quelltext ergänzt)

Abitur:

19.11 BAYER-Bäume

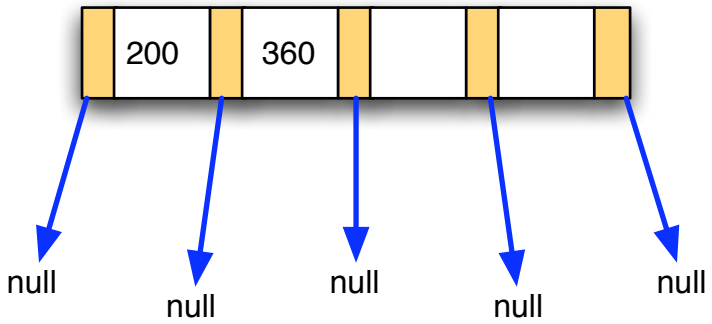
19.11.1 Grundidee

Wie kann man die Vorteile von Arrays und die von Suchbäumen miteinander in Verbindung bringen? Neben den AVL-Bäumen spielen vor allem die BAYER-Bäume oder kurz B-Bäume eine wichtige Rolle bei alltäglichen Anwendungen wie zum Beispiel großen Datenbanken. Wir wollen auch hier keine Java-Quelltexte entwickeln, sondern uns einfach mal etwas oberflächlich mit dem Thema beschäftigen. Bauen wir doch einen solchen B-Baum einfach mal auf. Ein Knoten des B-Baums soll dabei vier Zahlen speichern können - was das bedeutet, werden Sie gleich noch sehen.



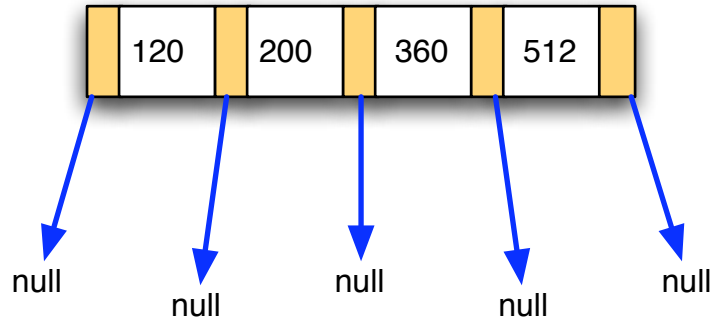
***19-23** Nach Einfügen der 360*

Die Abbildung 19-23 zeigt den B-Baum nach dem Einfügen der Zahl 360. Der B-Baum besteht aus einem einzigen Knoten, der aber aus 9 Komponenten besteht, nämlich 4 Zahlen und 5 Zeigern. Doch dazu später. Fügen wir nun die nächste Zahl ein, die 200:



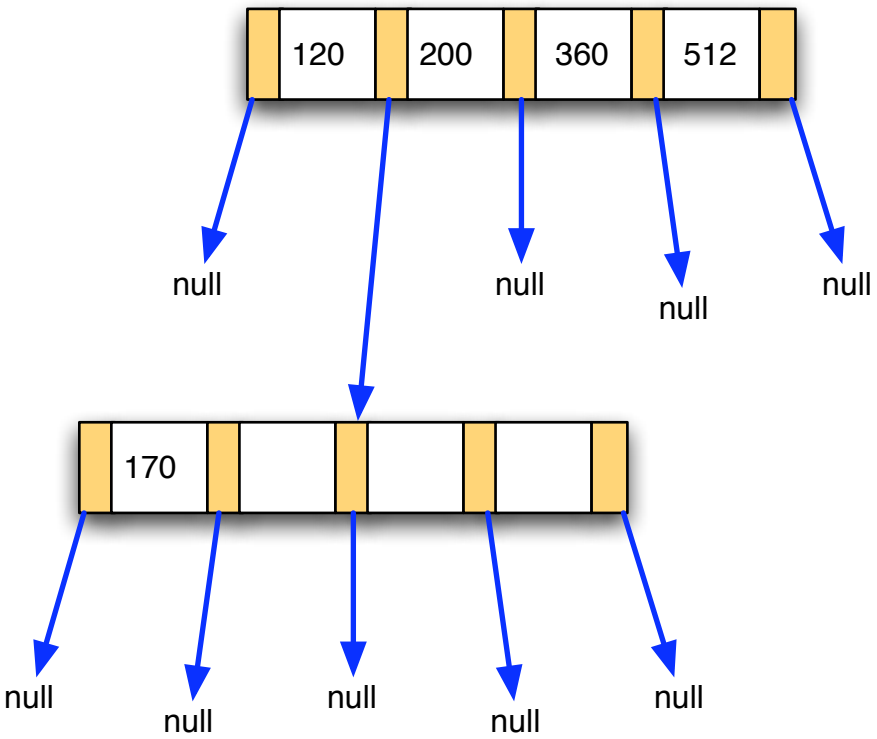
***19-24** Nach Einfügen der 200*

Die 200 wird jetzt vor die 360 gesetzt, weil die neue Zahl kleiner ist als die alte Zahl. Sonst tut sich nichts weiter; der B-Baum besteht immer noch aus nur einem Knoten.



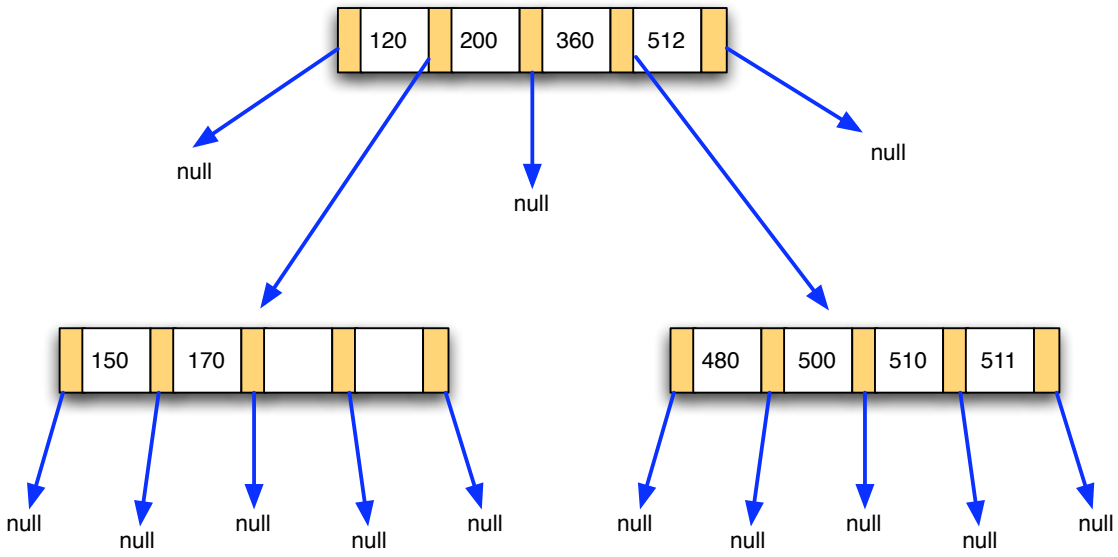
19-25 *Nach Einfügen der 120 und der 512*

Jetzt wird es langsam interessant. Der Knoten ist voll, er kann keine weiteren Zahlen mehr aufnehmen. Innerhalb des Knotens sind die Zahlen jedoch aufsteigend sortiert, wir haben hier quasi einen Array vorliegen. Bei echten Anwendungen werden natürlich viel größere Knoten mit 20, 30 oder noch mehr Elementen eingesetzt. In Wirklichkeit sind diese Elemente auch keine Zahlen, sondern Zeiger auf komplexe Objekte. Aber wir studieren weiter das Verhalten unseres vereinfachten B-Baums und schauen, was passiert, wenn die Zahl 170 eingefügt wird:



19-26 *Nach Einfügen der 170*

Die 170 passt nicht mehr in den ersten Knoten, daher wird nach einer passenden Einfügeposition gesucht und zwischen 120 und 200 gefunden. An den entsprechenden Zeiger des ersten Knotens wird also ein neuer zweiter Knoten angehängt und mit der Zahl 120 gefüllt. Die anderen drei Plätze des zweiten Knotens sind noch frei.



***19-27** Nach Einfügen weiterer Zahlen*

19.11.2 Eine mögliche Datenstruktur in Java

Wie könnte eine mögliche Datenstruktur für B-Bäume in Java aussehen? Wir wollen diese Datenstruktur von vornherein flexibel gestalten, es sollen also Knoten beliebiger Breite möglich sein.

```
public class Knoten
{
   private Knoten[] next;
   private int[] value;
   private int count;
   public int order;
   ...
```

Hier ein Vorschlag für eine solche Datenstruktur. Die Zahlen werden als int-Array **value**, die Nachfolger-Zeiger als Array **next** vom Typ **Knoten** angelegt. Das Attribut **order** speichert die Zahl der Elemente pro Knoten. In unserem eben behandelten Beispiel hätte **order** also den Wert 4, da vier int-Zahlen in einem Knoten gespeichert wurden. Wichtig ist auch das Attribut **count**, in diesem Attribut merkt sich ein Knoten-Objekt, wie viele Elemente bereits in dem Knoten enthalten sind. Bei einem neu erzeugten Knoten-Objekt hat **count** natürlich zunächst den Wert 0. Ist das Objekt voll belegt, hat **count** den Wert **order**-1.

19.11.3 Ein Konstruktor für die Klasse Knoten

```
public Knoten(int n, int x)
{
   order = n;
   next  = new Knoten[order+1];
   value = new int[order];

   for (int i=0; i<order; i++)
   {
      next[i] = null;
      value[i] = 0;
   }
   next[order] = null;
   value[0] = x;
   count = 1;
}
```

Der Konstruktor erhält zwei Parameter, nämlich die Breite `n` des Knotens (beispielsweise 4), und die erste Zahl `x`, die in den Knoten eingefügt werden soll.

Das Attribut **order** speichert dann die Knotenbreite, dann wird der Array **next** mit **order+1** Nachfolgerknoten angelegt, die natürlich zunächst mal leer sind (in der for-Schleife werden diese Zeiger auf `null` gesetzt). Schließlich wird der Array **value** für die eigentlichen Elemente erzeugt und mit Nullen initialisiert.

Nach der for-Schleife wird der ganz rechte Nachfolger-Zeiger des Knotens erzeugt und initialisiert, und schließlich wird die Zahl `x` in das erste Array-Element geschrieben und der Zähler **count** um 1 höher gesetzt. An dieser Stelle ist der Quelltext noch nicht optimal, da nicht überprüft wird, ob **count** nicht größer ist als order.

19.11.4 Eine Einfüge-Methode

```
public void insertValue(int x)
{
   if (count < order)
   {
      value[count] = x;
      count++;
      sort();
   }
   else
   {
      int i=0;
      while ((i < count) && (x > value[i])) i++;
      if (next[i] == null)
         next[i] = new Knoten(order,x);
      else
         next[i].insertValue(x);
   }
}
```

Zunächst wird geschaut, ob noch ein weiteres Element in den Knoten hineinpasst. Ist das der Fall (`count < order`), dann wird der Wert `x` in den ersten noch freien Speicherplatz geschrieben (`value[count] = x`) und **count** anschließend inkrementiert. Danach muss der Knoten neu sortiert werden. Dies geschieht durch den Aufruf der **sort()**-Methode, auf die wir gleich noch kommen.

Ist der Knoten bereits voll besetzt, so muss der passende Nachfolgerknoten gefunden werden. Dazu wird der Wert der Zahl `x` nach und nach mit den Werten der im Knoten vorhandenen Zahlen verglichen. Solange `x` größer ist als die vorhandenen Werte, wird weiter gesucht (siehe while-Schleife). Ist der richtige **next**-Zeiger gefunden, so muss nachgeschaut werden, ob der Nachfolgeknoten bereits existiert oder ob ein neuer Nachfolgerknoten erst angelegt werden muss.

Existiert der Nachfolgerknoten bereits, so wird dieser mit der Anweisung **insertValue(x)** aufgerufen. Muss erst ein neuer Knoten angelegt werden, so erfolgt dies mit der Anweisung `next[i] = new Knoten(order,x)`.

19.11.5 Die sort-Methode für einen Knoten

```
private void sort()
{
   for (int i=0; i<count; i++)
      for (int j=0; j<count-1; j++)
         if (value[j] > value[j+1])
         {
            int temp = value[j];
            value[j] = value[j+1];
            value[j+1] = temp;
         }
}
```

Wie man sofort sieht, handelt es sich hier um einen ganz einfachen Bubblesort. Da die Zahl der Knotenelemente nicht allzu groß ist, reicht hier der einfach zu implementierende Bubblesort völlig aus. Ein Quicksort für 5, 10 oder auch 20 Elemente wäre "mit Kanonen auf Spatzen schießen".

19.11.6 Die show-Methode für einen Knoten

```
public void show(int level)
{
   for (int i=0; i<=count; i++)
   {
      if (next[i] != null)
         next[i].show(level+1);
      if (i<count)
         System.out.println(level+" / "+value[i]);
   }
}
```

Hier handelt es sich um eine typische inorder-Methode. Zunächst wird "nach links herabgestiegen". Wenn da nichts mehr zu erreichen ist, wird wieder aufgestiegen und nebenbei werden die unterwegs "gefundenen" Elemente ausgegeben. Der Ausgabe-Befehl schreibt nicht nur den Wert des jeweiligen Elements in die Konsole, sondern auch den Level, also die Ebene, auf der sich der Knoten befindet.

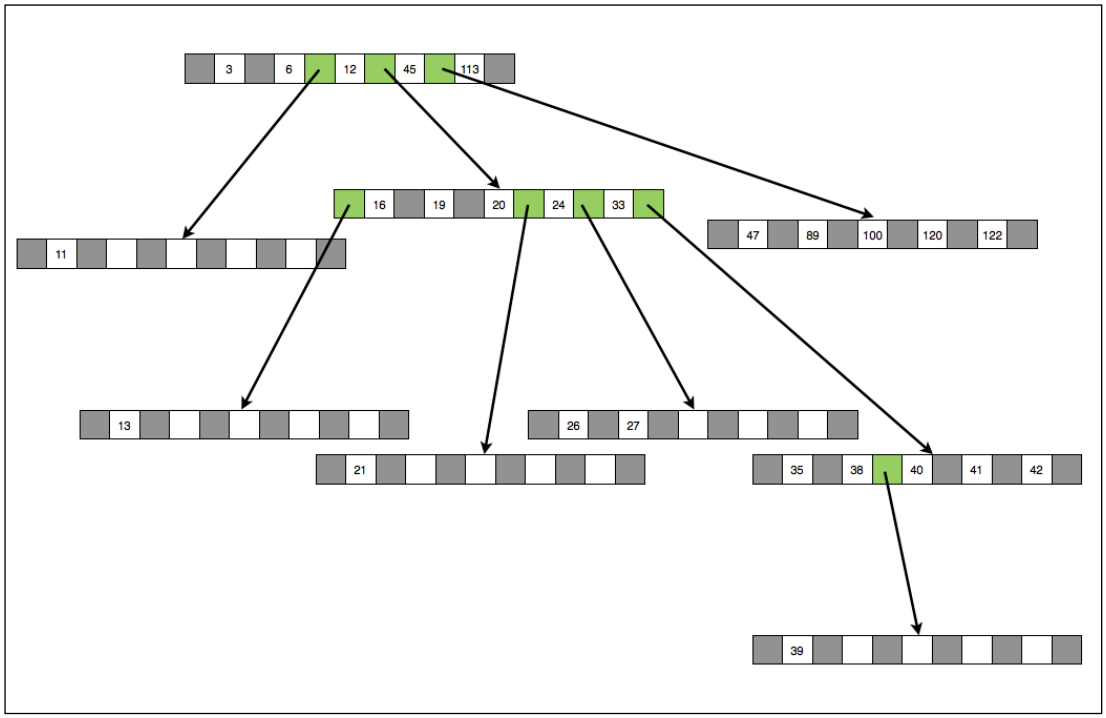
19.11.7 Ein Testprogramm für die Klasse Knoten

```
public class Test
{
    Knoten kn;

    public Test()
    {
      kn = new Knoten(5,12);

      kn.insertValue(113);
      kn.insertValue(45);
      kn.insertValue(3);
      kn.insertValue(6);
      kn.insertValue(11);
      kn.insertValue(100);
      kn.insertValue(89);
      kn.insertValue(47);
      kn.insertValue(16);
      kn.insertValue(24);
      kn.insertValue(33);
      kn.insertValue(19);
      kn.insertValue(20);
      kn.insertValue(21);
      kn.insertValue(13);
      kn.insertValue(27);
      kn.insertValue(26);
      kn.insertValue(35);
      kn.insertValue(40);
      kn.insertValue(42);
      kn.insertValue(38);
      kn.insertValue(41);
      kn.insertValue(120);
      kn.insertValue(122);
      kn.insertValue(39);
      kn.show(1);
    }
}
```

Mit dieser Klasse wurde die Klasse **Knoten** von mir getestet. Der Quelltext funktionierte problemlos. Dann habe ich mit Hand die 25 Elemente in einen B-Baum der Ordnung 5 eingefügt. Hier ist das Ergebnis:



19-28 *Der B-Baum aus der Klasse Test nach Einfügen aller Zahlen.*

Die **next**-Zeiger, die nicht auf null zeigen, sind grün markiert. Schauen wir uns nun die Ausgabe des Testprogramms an:

```
1 / 3
1 / 6
2 / 11
1 / 12
3 / 13
2 / 16
2 / 19
2 / 20
3 / 21
2 / 24
3 / 26
3 / 27
2 / 33
3 / 35
3 / 38
4 / 39
3 / 40
3 / 41
3 / 42
1 / 45
2 / 47
2 / 89
2 / 100
1 / 113
2 / 120
2 / 122
```

Die Ausgabe funktioniert genau so, wie sie theoretisch funktionieren sollte. Zunächst werden die beiden ersten Zahlen des Wurzel-Knotens ausgegeben, da hier noch keine Zeiger existieren, die

auf andere Knoten zeigen. Nun kommt der Zeiger zwischen der 6 und der 12, der auf einen Knoten der Ebene 2 verweist. Also wird zunächst dieser Knoten ausgegeben. Hier steht nur die Zahl 11, daher wird "2 / 11" angezeigt. Nun geht es zurück zum Wurzelknoten, und die 12 wird ausgegeben. Wieder kommt ein Zeiger auf einen untergeordneten Knoten. Nun wird aber nicht die 16, das erste Element dieses Knotens ausgegeben, sondern der erste Zeiger des Knotens zeigt auf einen weiteren Knoten in der Ebene 3. Also muss zunächst dieser abgearbeitet werden. Das einzige Element dieses Knotens wird dann mit "3 / 13" ausgegeben. Und so geht das weiter, bis schließlich alle Elemente in geordneter Reihenfolge ausgegeben worden sind.

B-Bäume wurden übrigens 1972 von Rudolf BAYER entwickelt. Sie werden in den Indexdateien von großen Datenbanken eingesetzt, üblich sind Ordnungen von 1024 oder mehr, d.h., dass in einem einzigen Knoten 1024 Elemente gespeichert werden können.

Damit wollen wir das Kapitel Bäume beenden. Zu dem Thema könnte man sicherlich noch viel mehr sagen, doch leider ist die Zeit in der Schule ja sehr begrenzt, und viele andere Themen der Informatik sind für das Abitur relevant.

19.11.8 Ein weiteres Testprogramm für die Klasse Knoten

```
import java.util.Random;

public class TestAuto
{
    Knoten kn;
    Random z = new Random();

    public TestAuto()
    {
      kn = new Knoten(10,100);

      for (int i=1; i<100; i++)
         kn.insert(z.nextInt(200)+1);

      kn.show(1);
    }
}
```

Diese Testklasse erzeugt 100 Zufallszahlen und "packt" sie in einen B-Baum beliebiger Ordnung. Anschließend erfolgt eine semi-graphische Darstellung des Baumes. Dazu wurde die show()-Methode der Klasse Knoten leicht ergänzt:

```
public void show(int level)
{
    for (int i=0; i<=count; i++)
    {
        if (next[i] != null)
            next[i].show(level+1);
        if (i<count)
        {
            for (int j=1; j<=level; j++)
               System.out.print("\t\t");
            if (i == 0)
               System.out.println("*"+value[i]);
            else
               System.out.println(" "+value[i]);
        }
    }
}
```

Die Zahlen des Wurzelknotens werden linksbündig dargestellt, die Zahlen der nächsten Knotenebene werden zwei Tabulatoren nach rechts eingerückt gezeigt. Die erste Zahl eines jeden Knotens wird außerdem noch mit einem Stern * gekennzeichnet.

Mit diesem Testprogramm kann man wunderbar herumspielen und sich die verschiedenen erzeugten B-Bäume anschauen. Hier zwei Beispiele.

B-Baum der Ordnung 5 mit 50 Zufallszahlen:

```
                    *1
          *1
                              *5
                               6
                               8
                               26
                    *26
                     36
                              *44
                               49
                     50
                              *61
                     62
                              *64
                               68
                     69
                              *73
                               82
                               82
                               88
                               91
           91
           91
           100
                    *101
                              *104
                     120
                     121
                              *126
                               129
                     129
                     129
          129
                    *133
                              *137
                               146
                               149
                               154
                               157
                     158
                     167
                              *171
                     172
                              *176
                               180
                               184
                               187
                     194
                              *198
                               200
```

B-Baum der Ordnung 3 mit 50 Zufallszahlen:

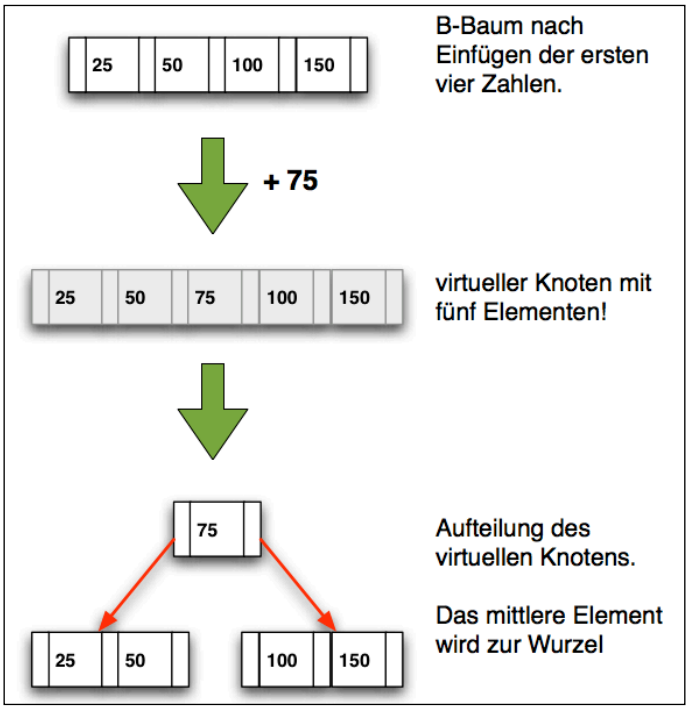
```
                    *2
                              *10
                               14
                     15
                     16
                              *19
          *21
                                    *42
                              *44
                               53
                               54
                    *57
                              *58
                               67
                     75
                                    *76
                              *88
                               89
                               90
                     91
                              *92
           100
                    *103
           111
                              *114
                    *116
                                    *118
                                     120
                              *126
                                    *127
                                     137
                                     138
                               140
                                    *142
                               147
                     147
                                    *151
                                     152
                                                *162
                                                 164
                                     169
                              *171
                               173
                               173
                     175
                                    *180
                              *190
                               191
                                    *192
                               197
```

19.12 "Richtige" B-Bäume

Die B-Bäume, die wir bisher behandelt haben, waren stark vereinfachte B-Bäume. Ich möchte dieses Skript mit einer kurzen Behandlung "echter" B-Bäume ausklingen lassen.

Typisch für einen echten B-Baum ist die so genannte Ordnung n. Darunter versteht man die Zahl der Elemente, die ein Knoten beherbergen kann. Genauer gesagt, ist jeder Knoten mit mindestens n Elementen und maximal 2n Elementen besetzt. Dies gilt nicht für die Wurzel des B-Baums, diese kann auch nur 1 Element enthalten.

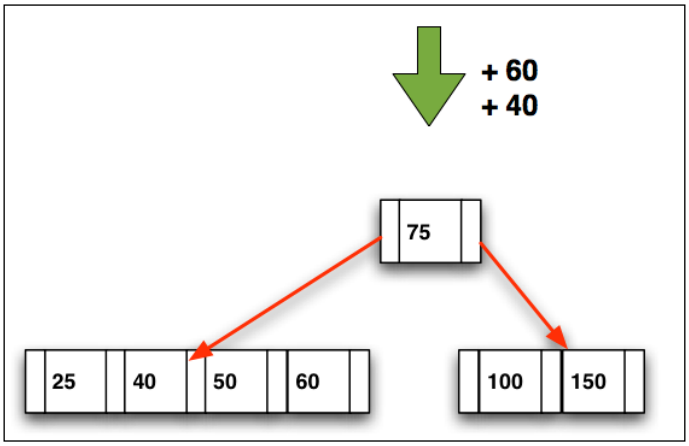
Stellen wir uns einen B-Baum der Ordnung n=2 vor. Dann muss jeder Knoten mindestens 2 Elemente enthalten, und maximal können 4 Elemente aufgenommen werden.



19-29 *Der B-Baum nach dem Einfügen der 5. Zahl*

Zunächst werden die vier Zahlen 50, 25, 150 und 100 eingefügt, und zwar in dieser Reihenfolge. Man erkennt, wie die vier Zahlen in den ersten Knoten einsortiert werden. Bis hier unterscheidet sich der "echte" B-Baum also noch nicht von unserem vereinfachten B-Baum.

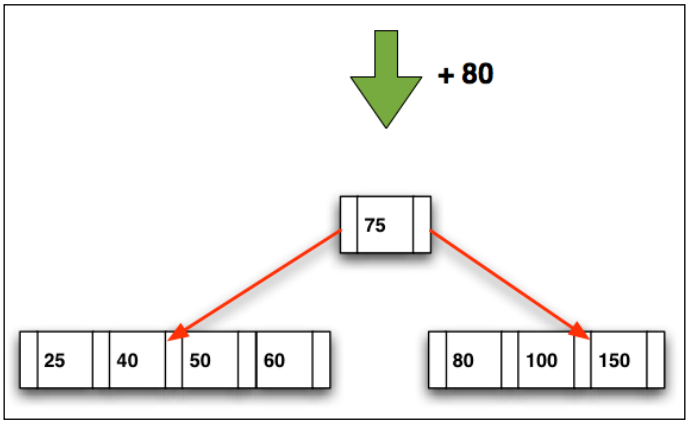
Nun soll die Zahl 75 als fünfte Zahl eingefügt werden. Diese Zahl passt nicht mehr in den Knoten. Bei dem vereinfachten B-Baum hätte man jetzt einen Nachfolger zwischen der 50 und der 100 platziert und dort die 75 als erstes Element eingefügt. Bei dem richtigen B-Baum geht das etwas anders. Stellen wir uns vor, der Knoten könne *doch* ein fünftes Element aufnehmen. In diesem virtuellen Knoten säße die 75 dann genau in der Mitte der fünf Elemente. Nun wird der Knoten in drei neue Knoten geteilt. Die 75 wird zur neuen Wurzel, die 25 und die 50 zum linken Nachfolger und die 100 und die 150 zum rechten Nachfolger.



19-30 *Die 40 und die 60 sind eingefügt worden*

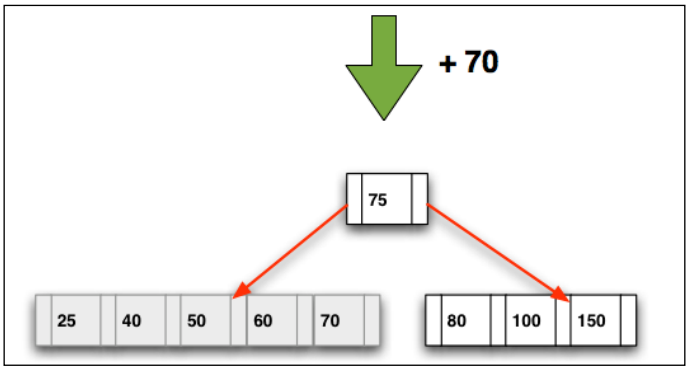
Die 40 und die 60 sind beide kleiner als die 75 in der Wurzel, also werden sie links weiter eingefügt. In dem linken Nachfolgerknoten ist noch Platz für die beiden Zahlen, damit ist die Sache klar. Ein Sortiervorgang sorgt dann dafür, dass der linke Nachfolger wieder aufsteigend sortiert ist.

Wir wollen jetzt die 80 in den Baum einfügen, was auch kein Problem sein sollte, da der rechte Nachfolger ebenfalls Platz für zwei weitere Zahlen hat.



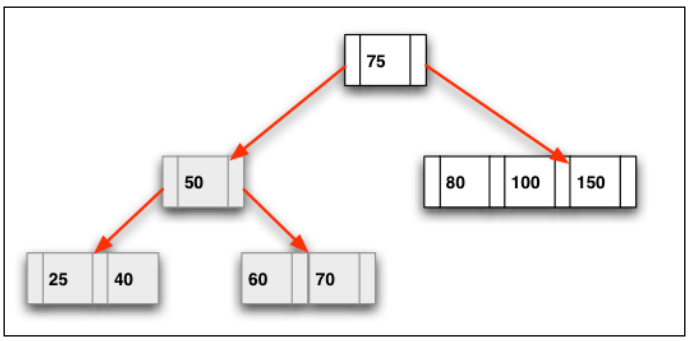
19-31 *Die 80 wurde eingefügt*

Die 70 soll als nächste Zahl eingefügt werden. Da sie kleiner ist als die 75 in der Wurzel des B-Baums, wird zunächst im linken Teilbaum weitergesucht. Bei unserem vereinfachten "B"-Baum hätten wir die 70 einfach in der Wurzel untergebracht, die ja noch Platz für drei Elemente hat. Beim richtigen B-Baum wird dagegen zunächst in den Blättern nach einer Stelle zum Einfügen gesucht. Der rechte Teilbaum ist mit vier Elementen voll besetzt. Die 70 kann nur dann in den rechten Teilbaum eingefügt werden, wenn dieser geteilt wird.



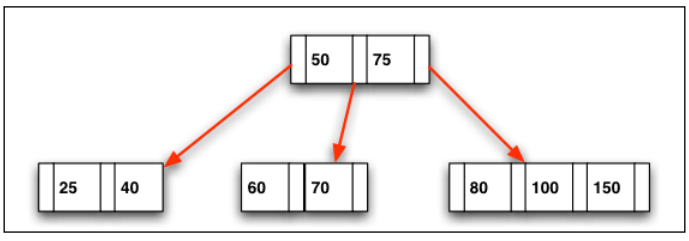
19-32 *Die 70 soll eingefügt werden*

Wenn wir die 70 in den linken Nachfolgeknoten einfügen, stellen wir uns wieder einen überbesetzten virtuellen Knoten vor, nur dass diesmal die 70 nicht in der Mitte des Knotens steht, sondern am rechten Rand. Das macht aber gar nichts. Wir teilen den Knoten wieder in drei Teilknoten. Die 50 in der Mitte bildet die Wurzel des Teilknotens, die 25 und die 40 ergeben den linken Nachfolger, und die 60 und 70 den rechten Nachfolger:



19-33 *"Umbau" des Knotens*

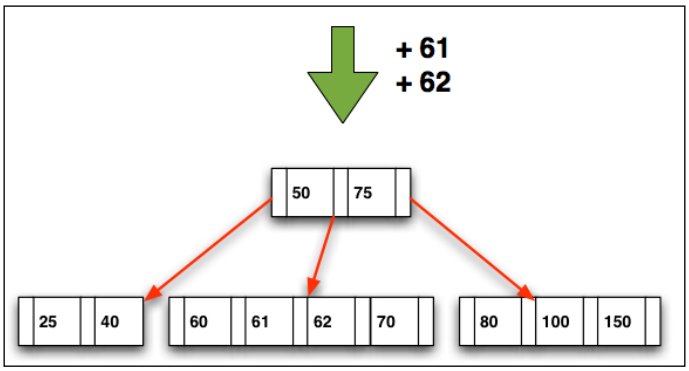
Leider sind wir noch immer nicht fertig mit dem "Umbauen" des Baums. Ein richtiger B-Baum ist gleichzeitig auch ein ausgeglichener Baum, und nun bietet sich noch die Möglichkeit, den Baum so umzustrukturieren, dass alle Blätter sich auf der gleichen Ebene befinden und nicht mehr auf zwei Ebenen wie in der Abbildung oben.



19-34 *Der Baum ist fertig umstrukturiert*

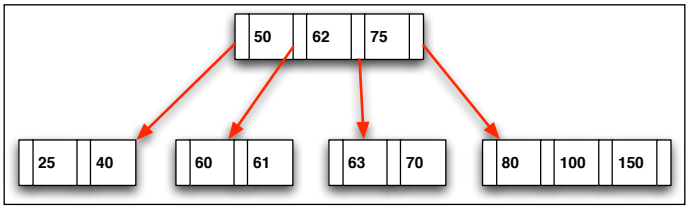
Der Knoten mit der 50 "wandert" nach oben und wird mit dem Vorgänger vereinigt. Dieser vereinigte Vorgänger hat dann drei Nachfolger, die wiederum Platz für weitere Elemente bieten.

Ein B-Baum wächst "von unten nach oben". Neue Elemente werden zunächst in die Blätter eingebaut. Sind diese voll, werden die Blätter geteilt, und die mittleren Elemente wandern eine Ebene nach oben, wo sie in den Mutterknoten integriert werden. Ist auch dieser voll, wiederholt sich der ganze Vorgang, bis schließlich die Wurzel des Baums erreicht ist. Wenn auch die Wurzel voll ist, so wird diese nach dem gleichen Prinzip geteilt, und es entsteht eine neue Wurzel, die dann nur ein Element enthält.



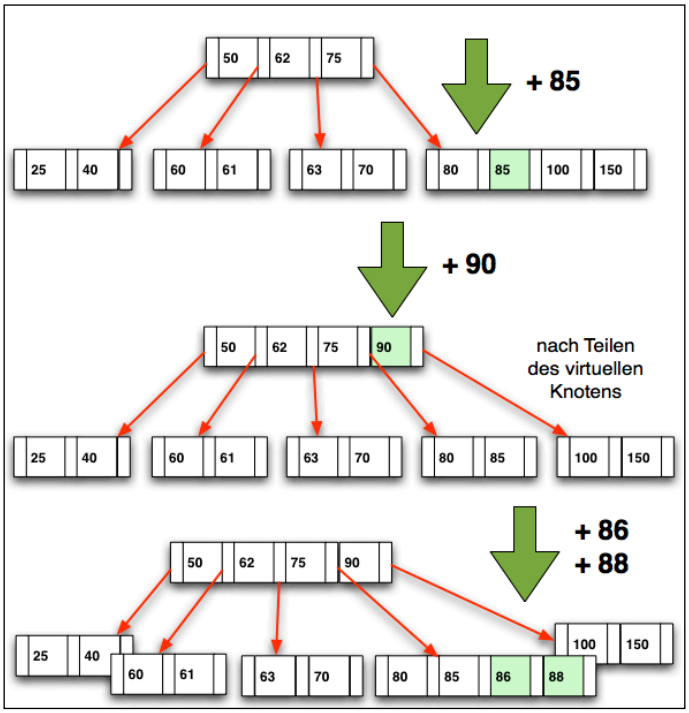
19-35 *Der Baum nach dem Einfügen von 62 und 61*

Hier sehen wir unseren B-Baum nach Einfügen der Zahlen 62 und 61. Der mittlere Knoten der Ebene 2 hatte noch Platz für zwei Elemente, also war das Einfügen überhaupt kein Problem.



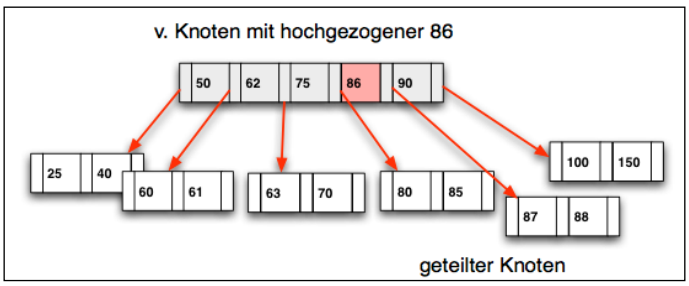
19-36 *Die 63 wurde eingefügt*

Beim Einfügen der 63 wird der mittlere der drei Nachfolgerknoten "überladen" und dann geteilt. Das mittlere Element, die 62, "rutscht" dabei eine Ebene höher in den Vorgängerknoten, der noch ein weiteres Element aufnehmen kann.

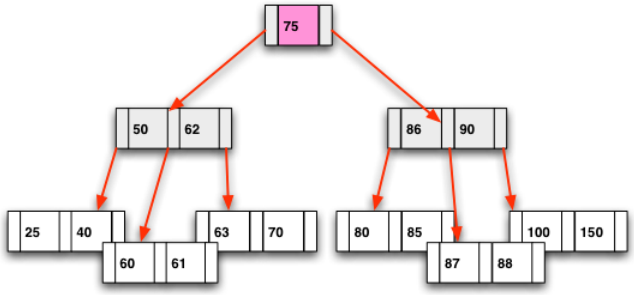


19-37 *Vier weitere Zahlen wurden eingefügt*

Nach dem Einfügen von vier weiteren Zahlen ist der Wurzelknoten vollbesetzt. Jetzt wird es "interessant", weil die Zahl 87 eingefügt werden soll, und die passt in den dritten Nachfolgerknoten (80, 85, 86, 88) nicht mehr hinein. Dieser muss also geteilt werden, und der mittlere Knoten muss dann eine Ebene höher wandern. Der Vorgängerknoten ist aber auch schon voll. Also wird auch dieser geteilt.



19-38 *Der Nachfolgerknoten wurde geteilt, das mittlere Element hochgezogen*



19-39 *Der B-Baum ist wieder fertig und auch ausgeglichen*