B-Trees

A B-tree of order *d* is a tree with the following structural properties:

1. The root either is a *leaf* or has between 2 and $2d+1$ children.
2. All nodes that are not leaves, except the root, have between d and $2d+1$ children.
3. All leaves are at the same depth
4. All leaves have between *d* and 2*d* keys.
5. All data are stored in leaves.
6. Interior nodes contain 2*d* keys and 2*d*+1 pointer.
   6.1. Pointers are denoted $p_1, p_2, \ldots, p_{2d+1}$.
   6.2. Keys are denoted, k_1, k_2, k_{2d}.
7. Pointers reference the children of interior nodes.
8. Keys are in ascending order with $k_1 < k_2 < \cdots < k_{2d}$.
9. The value of key k_i is equal to the smallest key in the subordinate tree referenced by pointer p_{i+1}. For example, the value of k_1 is equal to the smallest key in the subordinate tree referenced by pointer p_2. In a similar way, the values of keys $k_2, k_3, \ldots, k_{2d}$, are equal to the smallest keys in subordinate trees referenced by pointers $p_3, p_4, \ldots, p_{2d+1}$, respectively.

1. A B-tree of order *one* is shown in Figure 1.
2. Interior nodes are represented by rounded-boxes containing both pointers and keys in Figure 1.
3. Leaves are shown as square-cornered-boxes having only keys in Figure 1.
4. Since *d*=1, *each* interior node can have at most *three* ($2(1)+1=3$) children and leaves have at most *two* keys.
5. The root (18:-:-) has two children. The root is not a leaf.
6. Interior nodes are those nodes having keys (6:12) and (24:30). The first, (6:12) has three children and the second, (24:30), has three children.
7. Leaves are those nodes having keys [0,3], [6,9], [12,15], [18,21], [24,27], and [30,33].
8. The root, when it is not a leaf, and interior nodes are denoted (k_1,k_2,…,k_{2d}) with dashes (-) in place of actual keys when the key does not exist.
9. Leaves are denoted [k_1,k_2,…,k_{2d}] with dashes (-) in place of actual keys when the key does not exist.

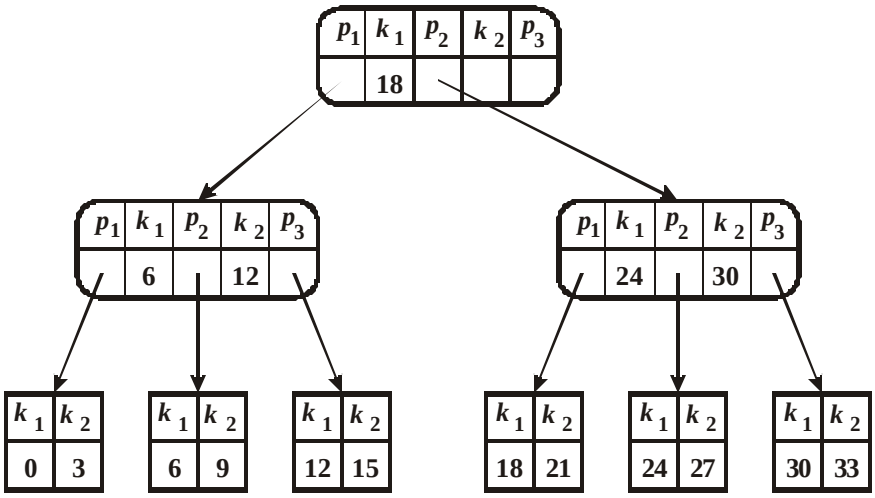

Figure 1. B+-Tree of order 1.

A B-tree can be represented by listing the nodes using a preorder traversal and indenting them according to their depth. For example, the B-tree in Figure 1 can be represented

(18,-)
- **(6,12)**
  - **[0,3]**
  - **[6,9]**
  - **[12,15]**
- **(24,30)**
  - **[18,21]**
  - **[24,27]**
  - **[30,33]**

10. B-trees can also be represented graphically as illustrated in Figure 2. The root, when it is not a leaf, and interior nodes are designated using a box with round corners. Leaves are shown in boxes with square corners. The B-tree in Figure 1 is also shown in Figure 2.

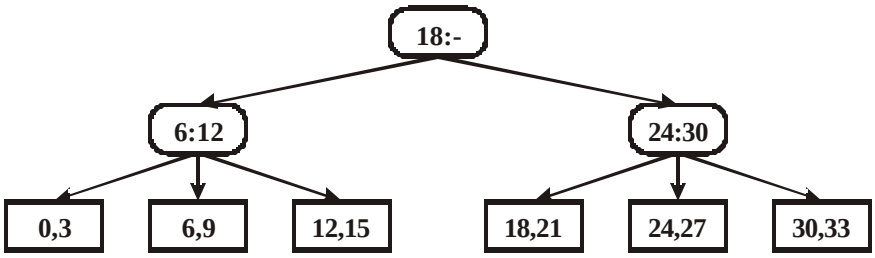

Figure 2. B+-tree of order 1

Insertion

1. Start with an empty B-tree of order 1. Insert key **51**. The resulting tree contains a single leaf.

   [51,-]

2. Insert key **29**. Key 51 is shifted one position to the right preserving the property that keys are always in ascending order.

   [29,51]

3. Insert key **73**. Key 73 is appended to the list of keys in the node. The leaf is now *overfull*.

   [29,51,73]

   3.1. Split the leaf.
   [29,-]
   [51,73]

   3.2. Create an empty interior node that will be the parent of the two leaves.
   (-:-)
   [29,-]
   [51,73]

   3.3. *Copy* the smallest key in the node having the larger keys into the newly created interior node.
   (51:-)
   [29,-]
   [51,73]

   3.4. Assign a pointer to the first leaf **[29,-]** to p_1 and a pointer to the second leaf, **[51,73]** to p_2.

4. Insert key **105**.
   (51:-)
   [29,-]
   [51,73,105]

   4.1. The leaf, **[51,73,105]**, is *overfull*.
   4.2. Split the leaf
   (51:-)
   [29,-]
   [51, -]
   [73,105]

   4.3. Create an empty interior node that will be the parent of the two leaves.
   (51:-)
   [29,-]
   (-,-)
   [51, -]
   [73,105]

4.4. *Copy* the smallest key in the node having the larger keys into the newly created interior node.

(51:-)
[29,-]
(73,-)
[51, -]
[73,105]

4.5. Merge the newly created interior node with the existing interior node.

(51:73)
[29,-]
[51, -]
[73,105]

5. Insert key **15**.

(51:73)
[15,29]
[51, -]
[73,105]

6. Insert key **31**.

(51:73)
[15,29,31]
[51, -]
[73,105]

6.1. The leaf **[15,29,31]**, is *overfull*. Split the leaf into **[15,-]** and **[29,31]**. Hoist the middle key into a new interior node **(29:-)** having pointers to the two leaves **[15,-]** and **[29,31]**.

(51:73)
(29:-)
[15,-]
[29,31]
[51, -]
[73,105]

6.2. The new interior node must be merged with the existing interior node.

(51:73) ? (29:-)
[15,-]
[29,31]
[51, -]
[73,105]

(29:51:73)
[15,-]
[29,31]
[51, -]
[73,105]

6.3. The interior node, **(29:51:73)** is *overfull* and must be split.

(29:51:73)
[15,-]
[29,31]
[51, -]
[73,105]

6.4. Splitting an interior node is different than splitting a leaf. The middle key is *removed* rather than copied. Pointers to the subordinate interior nodes are assigned to pointers, p_1 and p_2 on either side of key **51** in the new root **(51:-)**.

(51:-)
(29:-)
(73:-)

6.5. Subordinate leaves remain attached to their respective interior nodes.

(51:-)
(29:-)
[15,-]
[29,31]
(73:-)
[51, -]
[73,105]