B & B+ Tree

m-ary tree

In graph theory, an **m-ary** tree (also known as **n-ary**, **k-ary** or **k-way** tree) is a rooted tree in which each node has no more than **m** children. A binary tree is the special case where m = 2, and a ternary tree is another case with m = 3 that limits its children to three.

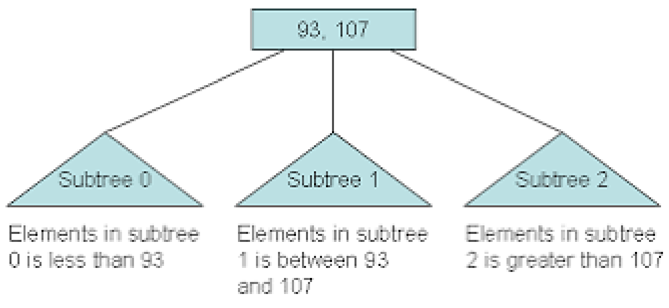
B tree

According to Knuth's definition, a B-tree of order **m** is a tree which satisfies the following properties:

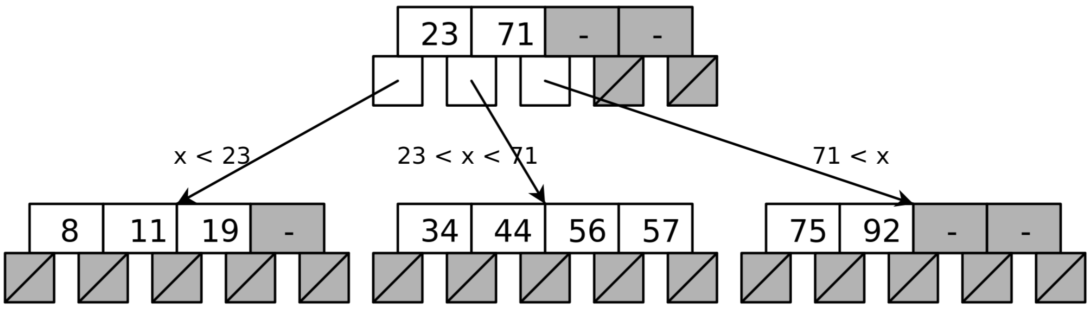
1. Every node has at most m children.
2. Every internal node has at least ⌈m/2⌉ children.
3. Every non-leaf node has at least two children.
4. All leaves appear on the same level and carry no information.
5. A non-leaf node with k children contains k−1 keys.

Each internal node's keys act as separation values which divide its subtrees. For example, if an internal node has 3 child nodes (or subtrees) then it must have 2 keys: a1 and a2. All values in the leftmost subtree will be less than a1, all values in the middle subtree will be between a1 and a2, and all values in the rightmost subtree will be greater than a2.

Internal nodes: Internal nodes (also known as inner nodes) are all nodes except for leaf nodes and the root node. They are usually represented as an ordered set of elements and child pointers.
The root node: The root node's number of children has the same upper limit as internal nodes, but has no lower limit. The root may be the only node in the tree with no children at all.
Leaf nodes: In Knuth's terminology, the "leaf" nodes are the actual data objects / chunks. The internal nodes that are one level above these leaves only store keys (at most m-1, and at least (m/2)-1 if they are not the root) and pointers (one for each key) to nodes carrying the data objects / chunks.


93, 107
Subtree 0
Subtree 1
Subtree 2
Elements in subtree 0 is less than 93
Elements in subtree 1 is between 93 and 107
Elements in subtree 2 is greater than 107

Another visual example:



B+ Tree

A B+ tree is an extension of a B tree which makes the search, insert and delete operations more efficient. As we know that B trees allow both the data pointers and the key values in internal nodes as well as leaf nodes, this certainly becomes a drawback for B trees, as the ability to insert the nodes at a particular level is decreased thus increase the node levels in it.

B+ Tree vs. B Tree

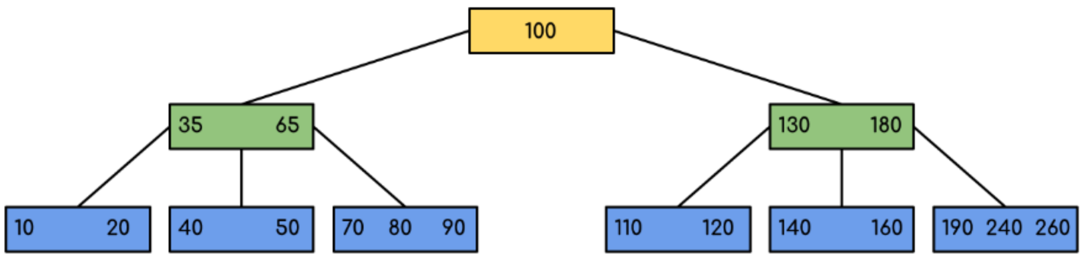
Here, are the main differences between B+ Tree vs. B Tree

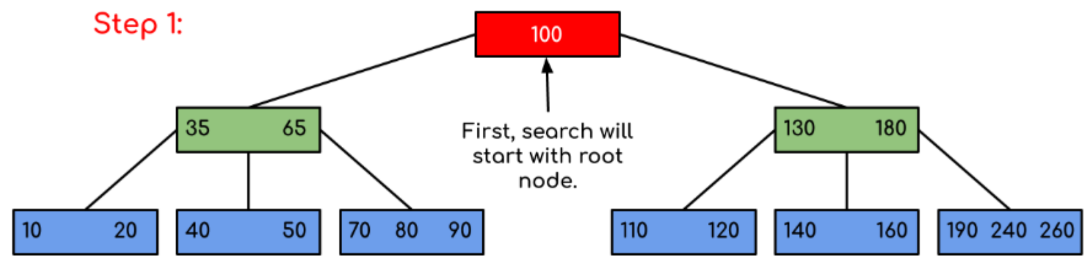
B+ Tree	**B Tree**
Search keys can be repeated.	Search keys cannot be redundant.
Data is only saved on the leaf nodes.	Both leaf nodes and internal nodes can store data
Data stored on the leaf node makes the search more accurate and faster.	Searching is slow due to data stored on Leaf and internal nodes.
Deletion is not difficult as an element is only removed from a leaf node.	Deletion of elements is a complicated and time-consuming process.
Linked leaf nodes make the search efficient and quick.	You cannot link leaf nodes.,

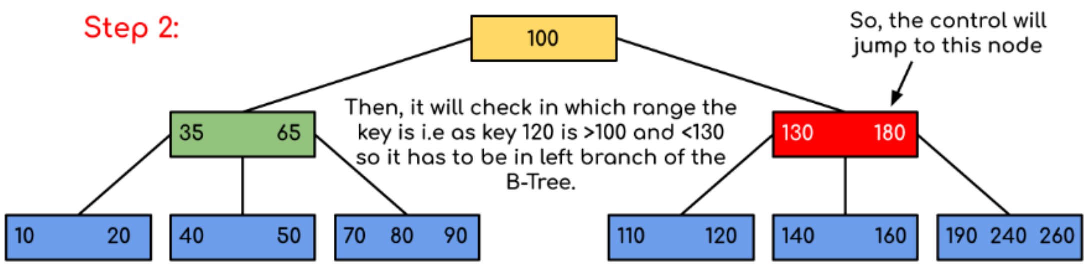
Searching a node in a B-Tree

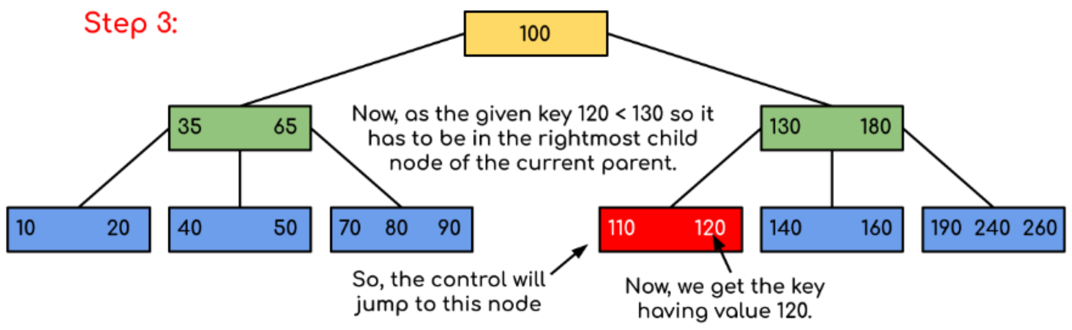
1. Start from the root and recursively traverse down.
2. For every visited non-leaf node,
    a. Perform a binary search on the records in the current node.
    b. If the node has the key, we simply return the node.
    c. Otherwise, we recur down to the appropriate child (The child which is just before the first greater key) of the node.
3. If we reach a leaf node and don't find k in the leaf node, then return NULL.

Input: Search 120 in the given B-Tree.






Step 2:
100
So, the control will jump to this node
Then, it will check in which range the key is i.e as key 120 is >100 and <130 so it has to be in left branch of the B-Tree.
35 65
130 180
10 20
40 50
70 80 90
110 120
140 160
190 240 260


Step 3:
100
35 65
Now, as the given key 120 < 130 so it has to be in the rightmost child node of the current parent.
130 180
10 20
40 50
70 80 90
110 120
140 160
190 240 260
So, the control will jump to this node
Now, we get the key having value 120.

Insert Operation in B-Tree

Since B Tree is a self-balancing tree, you cannot force insert a key into just any node.

The following algorithm applies:

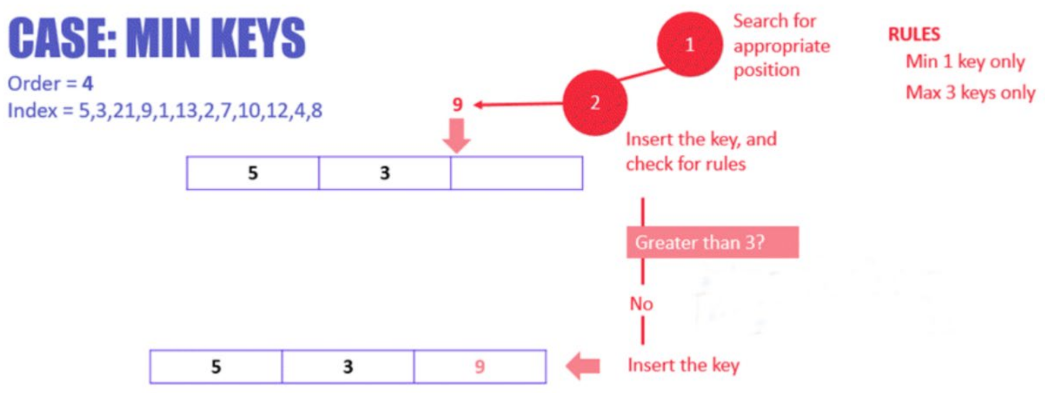
1. Run the search operation and find the appropriate place of insertion.
2. Insert the new key at the proper location, but if the node has a maximum number of keys already:
3. The node, along with a newly inserted key, will split from the middle element.
4. The middle element will become the parent for the other two child nodes.
5. The nodes must re-arrange keys in ascending order.

Computation of no. of keys and children

Say, if the Order (m) = 5
Then,

- Min children = ⌈m/2⌉ = 3
- Max children = m = 5
- Min keys = ⌈m/2⌉ - 1 = 2
- Max keys = m – 1 = 4

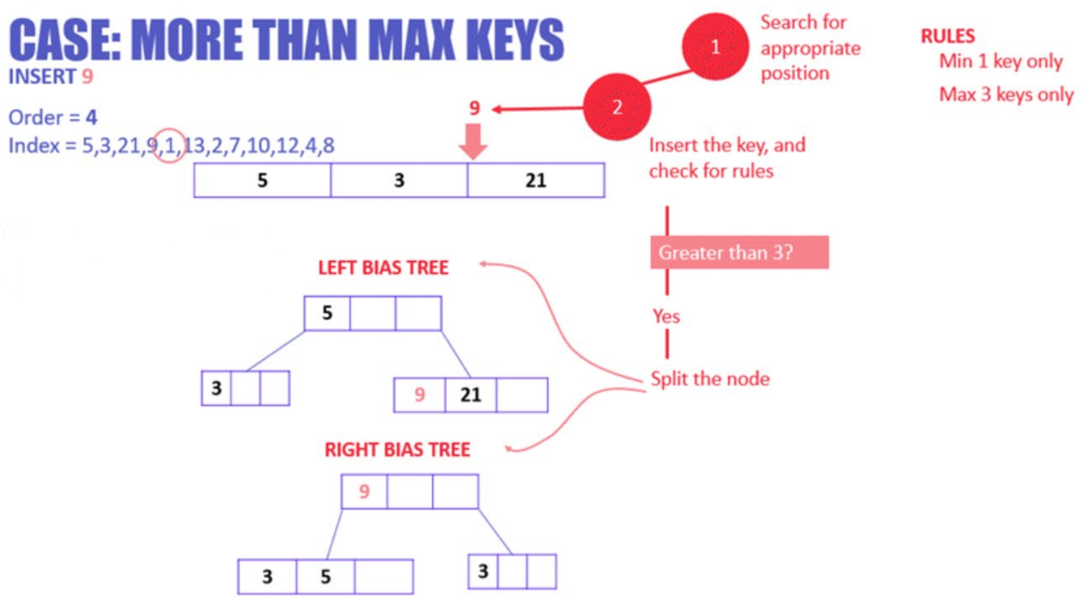
Example 01:


CASE: MIN KEYS
Order = 4
Index = 5,3,21,9,1,13,2,7,10,12,4,8
1
Search for appropriate position
RULES
Min 1 key only
Max 3 keys only
9
2
Insert the key, and check for rules
5
3
Greater than 3?
No
5
3
9
Insert the key

<u>In the above example:</u>

1. Search the appropriate position in the node for the key
2. Insert the key in the target node, and check for rules
3. After insertion, does the node have more than equal to a minimum number of keys, which is 1? In this case, yes, it does. Check the next rule.
4. After insertion, does the node have more than a maximum number of keys, which is 3? In this case, no, it does not. This means that the B Tree is not violating any rules, and the insertion is complete.

Example 02:

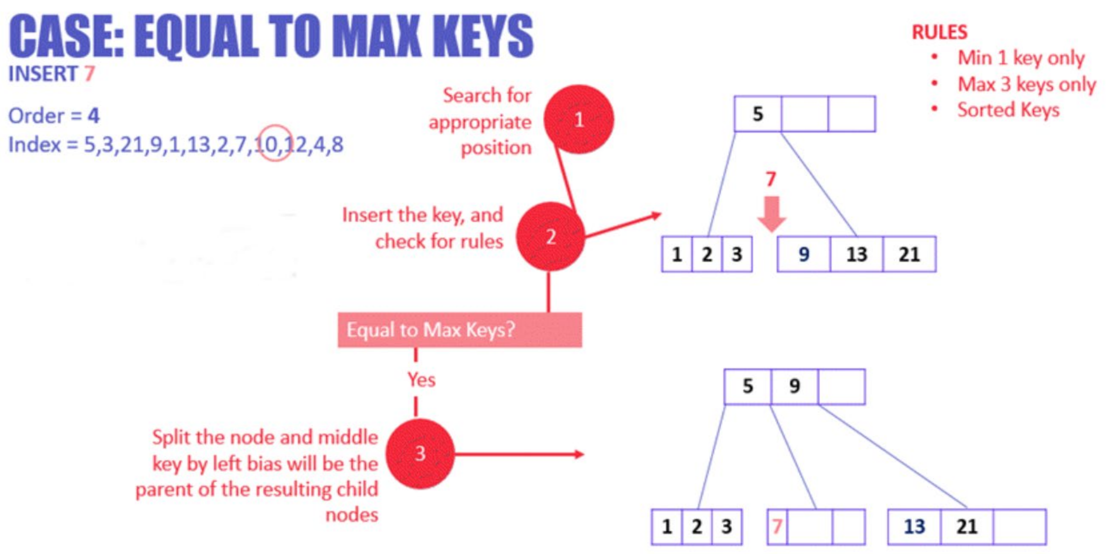


<u>In the above example:</u>

1. The node has reached the max number of keys
2. The node will split, and the middle key will become the root node of the rest two nodes.
3. In case of even number of keys, the middle node will be selected by left bias or right bias.

Similarly, 13 and 2 can be inserted easily in the node as they fulfil less than max keys rule for the nodes.

Example 03:


CASE: EQUAL TO MAX KEYS
INSERT 7
Order = 4
Index = 5,3,21,9,1,13,2,7,10,12,4,8
RULES
Min 1 key only
Max 3 keys only
Sorted Keys
1 Search for appropriate position
2 Insert the key, and check for rules
Equal to Max Keys?
Yes
3 Split the node and middle key by left bias will be the parent of the resulting child nodes
5
7
1 2 3
9 13 21
5 9
1 2 3
7
13 21

<u>In the above example:</u>

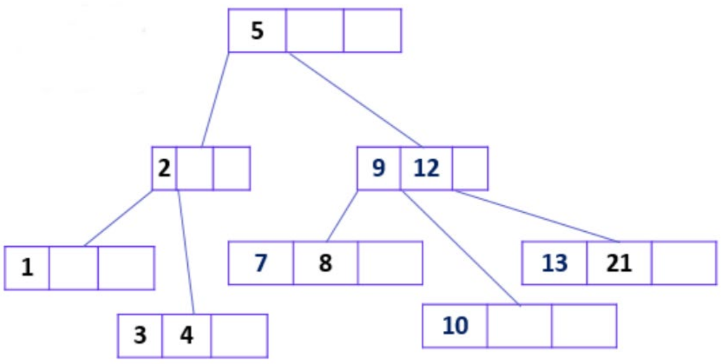
1. The node has keys equal to max keys.
2. The key is inserted to the target node, but it violates the rule of max keys.

3. The target node is split, and the middle key by left bias is now the parent of the new child nodes.
4. The new nodes are arranged in ascending order.

Similarly, based on the above rules and cases, the rest of the values can be inserted easily in the B Tree.

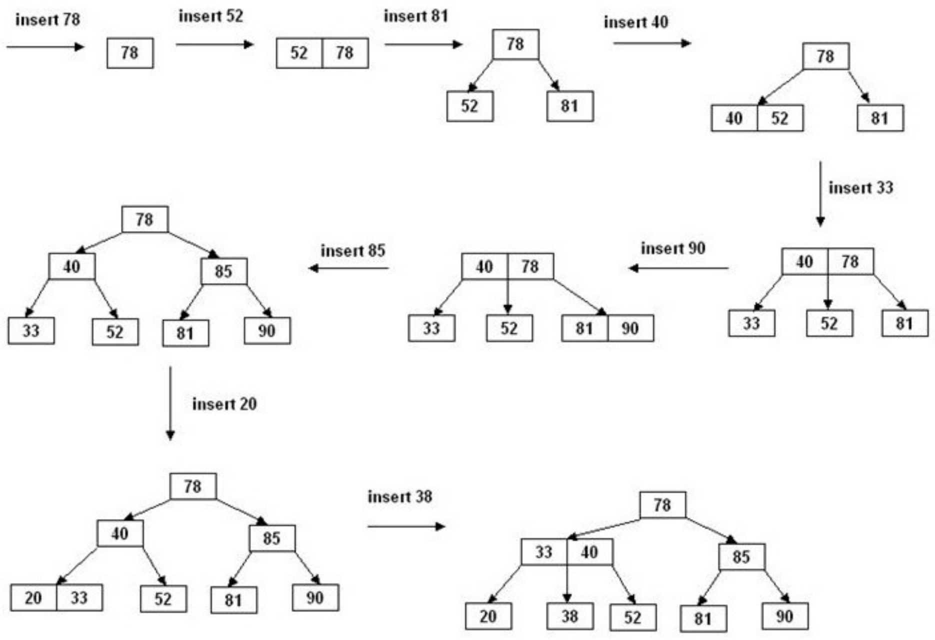
EXAMPLE SOLVED

Order = **4**
Index = 5,3,21,9,1,13,2,7,10,12,4,8



Another complete example:

- Example: Insert the keys 78, 52, 81, 40, 33, 90, 85, 20, and 38 in this order in an initially empty B-tree of order 3


insert 78
78
insert 52
52 78
insert 81
78
52
81
insert 40
78
40 52
81
insert 33
40 78
33
52
81
insert 90
40 78
33
52
81 90
insert 85
78
40
85
33
52
81
90
insert 20
78
40
85
20 33
52
81
90
insert 38
78
33 40
85
20
38
52
81
90