05/05/2020

Threaded Binary Tree:

A binary tree is threaded by making all right child pointers that would normally be null point to the in-order successor of the node (if it exists), and all left child pointers that would normally be null point to the inorder predecessor of the node.

The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion.

Types of Threaded binary trees:

1. Single threaded: each node is threaded towards either the in-order predecessor or successor (left or right).

(a) **Left in-threaded binary tree:**

A left NULL pointer can be used as threads to store the address of inorder predecessor of the node then the binary tree is called a left in-threaded binary tree.

(b) **Right in-threaded binary tree:**

A right NULL pointer can be used as threads to store the address of inorder successor of the node then the binary tree is called a right in-threaded binary

2. Double threaded : each node is threaded towards both the in-order predecessor and successor (left and right)

i.e. If both left and right NULL pointers are used as threads to store the address of inorder predecessor and inorder successor then the binary tree is called a fully in-threaded tree or in-threaded or double threaded tree.

Example.

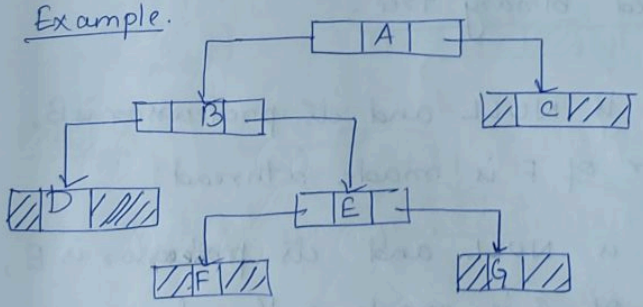


fig: Binary tree without threads.

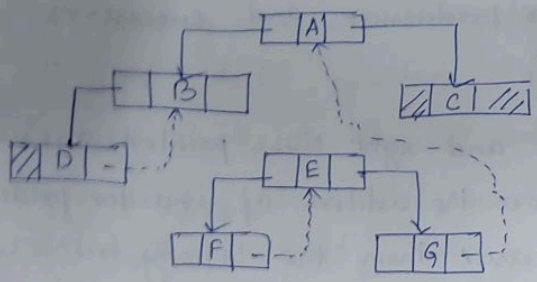
Here, Inorder traversal of the tree is DBFEGAC.

D's right link is NULL and its successor is B, so right pointer of D is made a thread.

F's right link is NULL and its successor is E, so right pointer of F is made a thread.

G's right link is NULL and its successor is A, so right pointer of G is made a thread.

Right in-threaded binary tree.



Inorder successor : D B F E G A C .

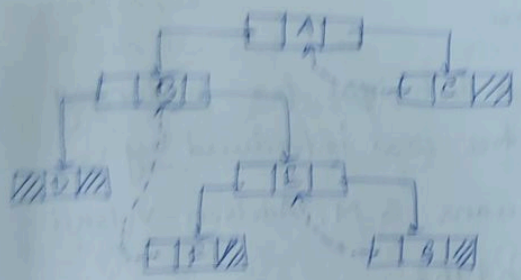
Left in-threaded binary tree.

F's left link is NULL and its predecessor is B, so left pointer of F is made a thread.

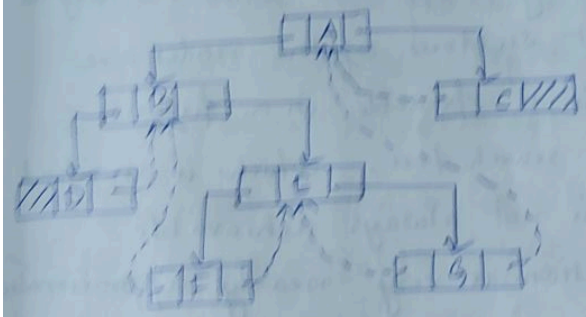
G's left link is NULL and its predecessor is E, so left pointer of G is made a thread.

C's left link is NULL and its predecessor is A, so left pointer of C is made a thread.

Inorder predecessor : D B F E G A C



Fully in threaded Binary tree.



D's left link is NULL but it has no predecessor and c's right link is NULL but it has no successor so both these pointer fields are NULL in the fully threaded tree.

B-tree Data Structure. 9/5/2020.

B-tree is a special type of search tree in which a node contains more than one value (key) and more than two children.
B-tree was developed in the year 1972 by Bayer and McCreight with the name Height Balanced m-way search tree. Later it was named as B-tree.

So, B-tree can be defined as follows -
B-tree is a self balanced search tree in which every node contains multiple keys and has more than two children.

Here, the number of keys in a node and number of children of a node depends on the order of B-tree. Every B-tree has an order.

B-tree of order m has the following properties -

(1). All leaf nodes must be at same level.

(2) All nodes except root must have atleast $\lceil m/2 \rceil - 1$ and maximum of $m-1$ keys.

(3) All non leaf nodes except root (i.e. all internal nodes) must have atleast $m/2$ children.

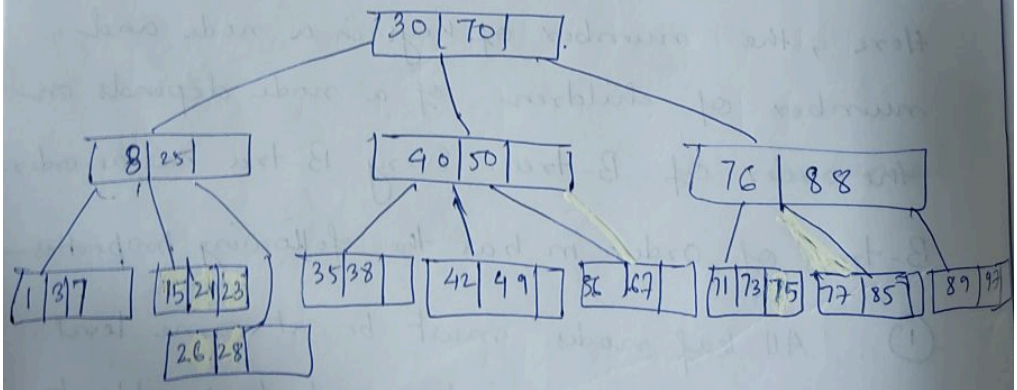
(4) If the root node is a non leaf node, then it must have atleast 2 children.

(5) A non leaf node with n-1 keys must have n number of children.

(6) All the key values in a node must be in ascending order.

For example.

B-tree of order 4 contains a maximus of 3 key values in a node and maximum of 4 children for a node.



Operations on a B-tree.

The following operations are performed on a B-tree.

1. Search
2. Insertion.
3. Deletion.

Search operation in B-tree.

The search operation in B-tree is similar to the search operation in Binary Search tree. In B-tree search process starts from the root node but here we make an n-way decision every time. where 'n' is the total number of children the node has.

The search operation is performed as follows:

Step 1: Read the search element from the user.

Step 2: Compare the search element with first key value of root node in the tree.

Step 3: If both are matched, then display "Given node is found" and terminate the function.

Step 4: If both are not matched, then check whether search element is smaller or larger than that key value.

Step 5: If search element is smaller, then continue the search process in left subtree.

Step 6: If search element is larger, then compare the search element with next key value in the same node and repeat step 3, 4, 5 and 6 until we find the exact match or until the search element is compared with last key value in the leaf node.

Step 7: If the last key value in the leaf node is also not matched then display "Element is not found" and terminate the function.

Insertion Operation in B-tree.

In a B-tree, a new element must be added only at the leaf node. That means, the new key value is always attached to the leaf node only. The insertion operation is performed as follows:

Step 1: Check whether tree is Empty.

Step 2: If tree is empty, then create a new node with new key value and insert it into the tree as a root node.

Step 3: If tree is not empty, then find the suitable leaf node to which the new key value is added using Binary Search Tree logic.

Step 4: If that leaf node has empty position, add the new key value to that leaf node in ascending order of key value within the node.

Step 5: If that leaf node is already full, split that leaf node by sending middle value to its parent node. Repeat the same until the sending value is fixed into a node.

Step 6: If the spliting is performed at root node then the middle value becomes new root node for the tree and the height of the tree is increased by one.

—x—

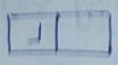
Date : 11/5/2022

Example

Construct a B tree of order 3 by inserting numbers from 1 to 10.

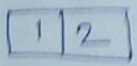
Insert 1.

Since '1' is the first element into the tree that is inserted into a new node. It acts as the root node.



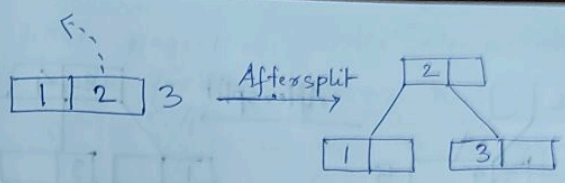
Insert 2.

Element 2 is added to existing leaf node. Here, we have only one node and that node acts as root and also leaf. This leaf node has an empty position. So, new element 2 can be inserted at that empty position.



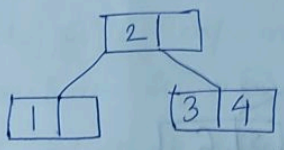
Insert 3.

Element 3 is added to existing leaf node. Here we have only one node and that node acts as root and also leaf. This leaf node doesn't has parent. So, this middle value becomes a new root node for the tree.



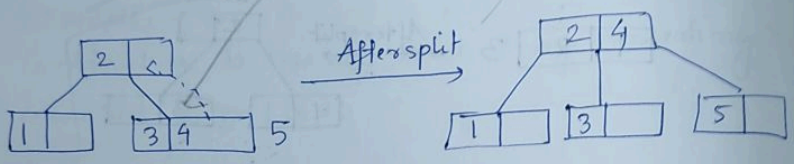
Insert 4.

Element 4 is larger that root node 2 and it is not a leaf node. So, we move to the right of 2. We reach to a leaf node with value 3 and it has an empty position. So new element 4 can be inserted at that empty position.



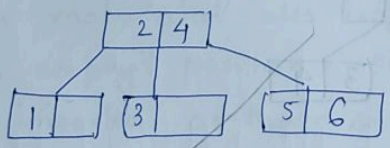
Insert 5.

Element 5 is larger than root node '2' and it is not a leaf node. So, we move to the right of '2'. We reach to a leaf node and it is already full. So, we split that node by sending middle value 4 to its parent node 2. There is an empty position in its parent node. So value 4 is added to node with value 2 and new element 5 added as new leaf node.



Insert (6)

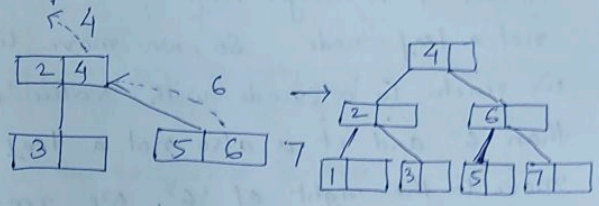
Element '6' is larger than root node '2' and '4'. and it is not a leaf node. So, we move to the right of '4'. We reach to a leaf node with value '5' and it has an empty position. So, new element (6) can be inserted at that empty position.



Insert (7)

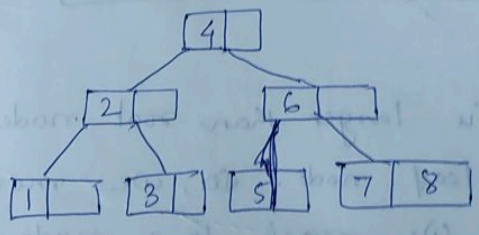
Element '7' is larger than root node '2' and '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a leaf node and it is already full. So, we split that node by sending middle value (6) to its parent node (2 & 4). But the parent (2 & 4) is also full. So, again we split the node (2 & 4) by sending middle value '4' to its parent but this node

doesn't have parent. So, the element '4' becomes new root node for the tree.



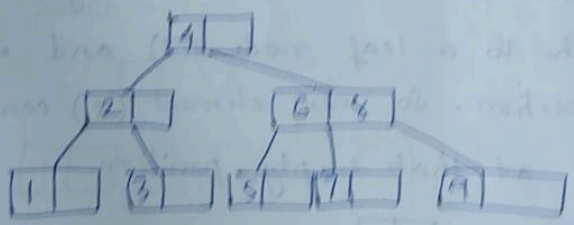
Insert(8)

Element '8' is larger than root node '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a node with value '6'. '8' is larger than '6' and it is also not a leaf node. So we move to the right of '6'. We reach to a leaf node (7) and it has an empty position. So new element (8) can be inserted at that empty position.



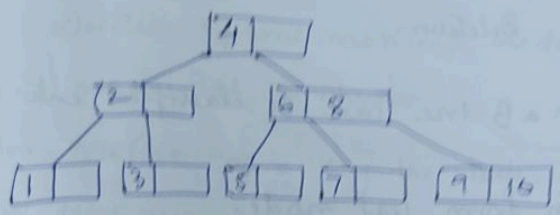
Insert (9)

Element '9' is larger than root node '4' and it is not a leaf node. So, we move to the right of '4', we reach to a node with value '6', '9' is larger than '6' and it is also not a leaf node. So, we move to the right of '6', we reach to a leaf node (7 8). This leaf node is already full, So, we split this node by sending middle value (8) to its parent node. The parent node (6) has an empty position. So, '8' is added at that position. And new element is added as a new leaf node.



Insert (10)

Element '10' is larger than root node '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a node with values '6' and '8'. '10' is larger than '6' & '8', and it is not a leaf node. So, we move to the right of '8' - We reach to a leaf node (9) - this leaf node has an empty position. So, new element '10' is added at that empty position.



which is the required B-tree.

—/—

9. ~~Ans~~ (1) Construct a B-tree of order 5 by inserting the following key values sequentially.

35 63 24 10 12 39 29 72 11 8 4 18 78 14 80 70 21

9. (2) Explain B+ tree with an example.

9. (3) What is the difference between B and B+ tree.

B+ tree Deletion.

Deletion in a B+ tree can be classified into two cases -

(A) Deletion from leaf node.

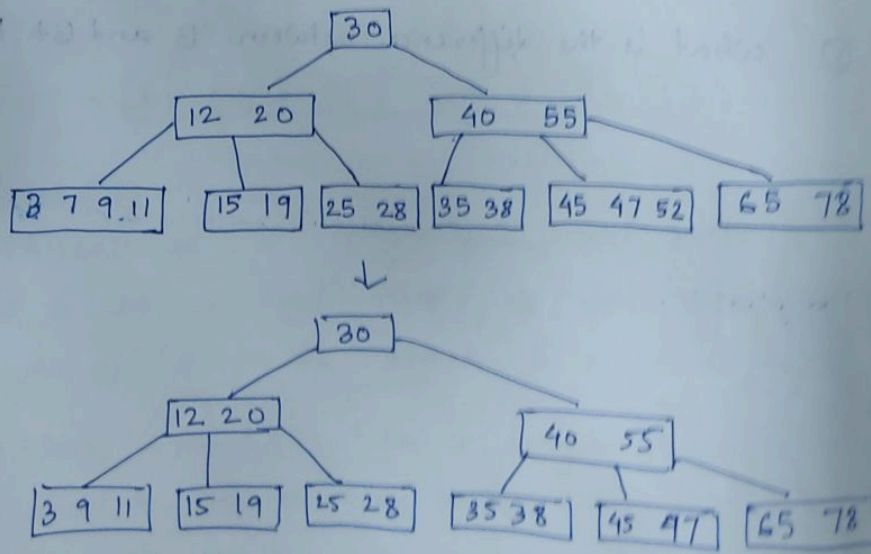
(B) Deletion from non leaf node.

(A) Deletion from leaf node.

(1) If node has more than MIN keys.

In this case, deletion is very simple and key can be very easily deleted from the node by shifting other keys of the node.

(a) Delete 7, 52 from tree which is given below-



After deleting '7', 9 and 11 can be shifted left to fill the gap. ~~extracted key~~ The key 52 is the rightmost key so other keys in the node need not be shifted.

(B) Deletion from nonleaf node.

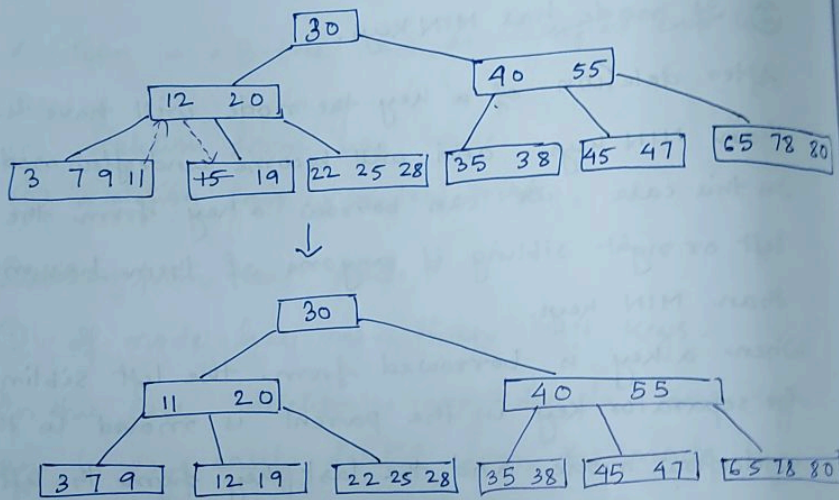
(2) If node has MIN keys.

After deletion of a key the node will have less than MIN keys, and will become underflow node. In this case, we can borrow a key from the left or right sibling if anyone of them has more than MIN keys.

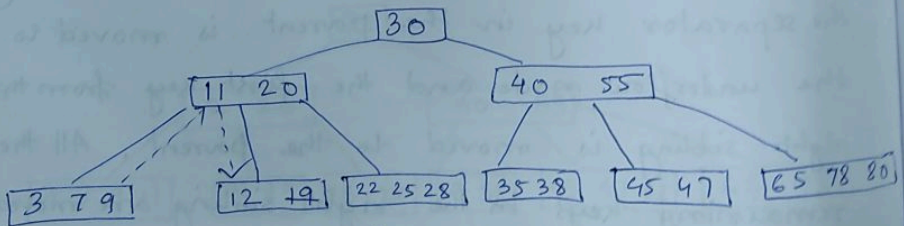
When a key is borrowed from the left sibling the separator key in the parent is moved to the underflow node and the last key from the left sibling is moved to the parent.

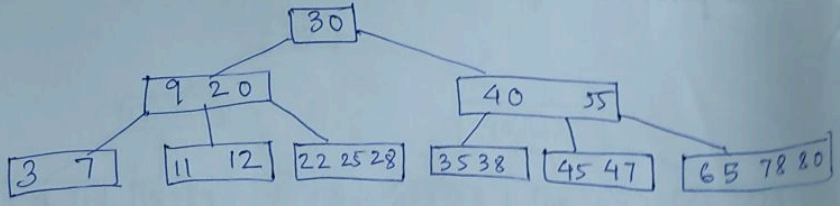
When a key is borrowed from the right sibling the separator key in the parent is moved to the underflow node and the first key from the right sibling is moved to the parent. All the remaining keys in the right sibling are moved one position left.

(b) Delete 15 from the following tree.

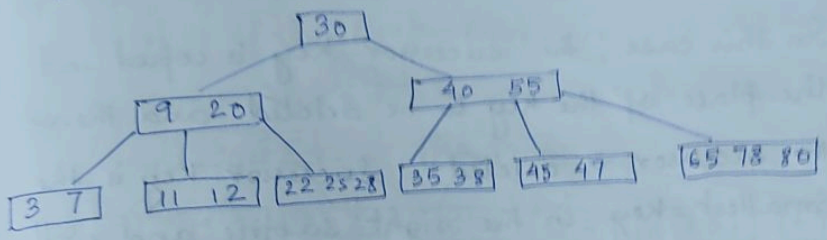


c) Delete 19 from the following tree.

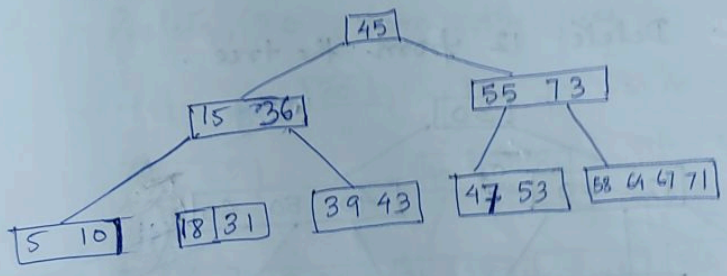




H.W. ① Delete 45 from the tree



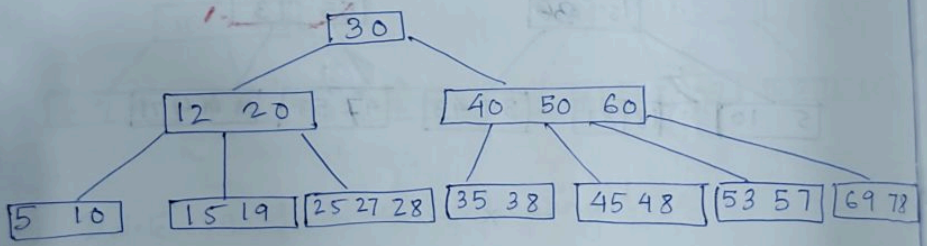
② Delete 31 from the tree.



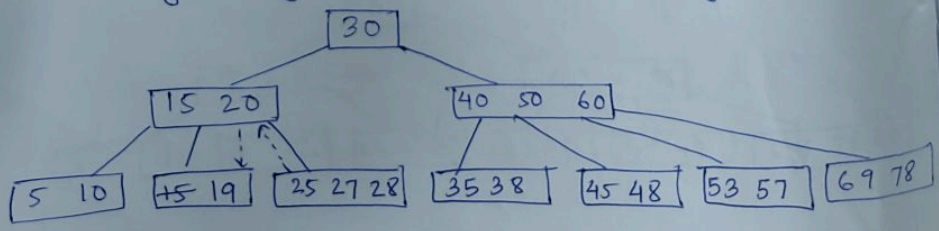
Deletion from non leaf node -

In this case, the successor key is copied at the place of the key to be deleted and then the successor is deleted. Successor key is the smallest key in the right subtree and will always be in the leaf node.

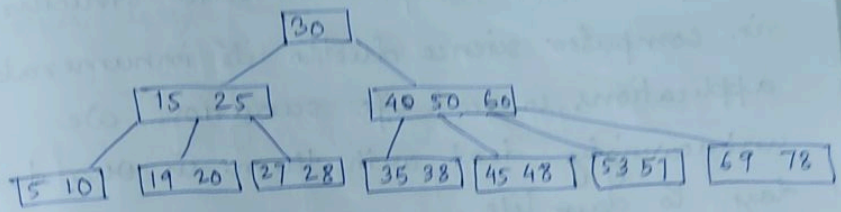
Ex.1. Delete 12 from the tree.



The successor key of 12 is 15.
So we will copy 15 at the place of 12 and now our task reduces to deletion of 15 from the leaf node. This deletion is performed by borrowing a key from the right sibling.



The tree after deletion of 12 is.



H.W. Delete 30 from the tree.