Insertion Techniques in B-Tree

J.K.

⇒ New keys are to be inserted at leaf level.

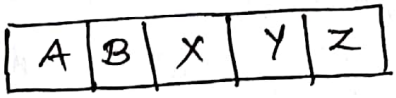
⇒ Starting from root, find out the correct leaf node in which the new key is to be inserted. In this process, if any node along this path is found to be full (ie they have max. no. of keys possible), then split the node to make space for new key.

⇒ A node is splitted when it is having (2t-1) keys values. The splitting is done by shifting the median key from the node to its parent and dividing the node into two as follows.

eg: B-tree with minimum degree(t) = 3

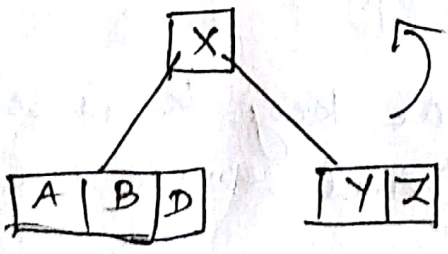
max. keys = 2t-1 = 5

At the instant



then we have to insert D, then node to be splitted because it has max. keys

splitting



Shift median 'x' to up.

Scanned by CamScanner

Deletion technique in B-Tree

⇒ Deletion from a B-tree is more complicated than insertion because we can delete a key from any node - not just a leaf - and when we delete a key from an internal node, we will have to rearrange the node's children.

⇒ As in insertion, we must make sure the deletion doesn't violate the B-tree properties.

⇒ Just as we had to ensure that a node didn't get too big due to insertion, we must ensure that a node doesn't get too small during deletion.

⇒ The deletion procedure deletes the key 'k' from the subtree rooted at 'x'. This procedure guarantees that whenever it calls itself recursively on a node x, the no. of keys in x is atleast the minimum degree t.

⇒ working of deletion process with various cases

Case 1

If the key k is in node x and x is a leaf (with enough keys), then delete the key k from x.

Case 2

If the key k is in node x and x is an internal node, do the following

Scanned by CamScanner

2.a) If the child y that precedes k in node x has atleast t keys, then find the predecessor k' of k in the subtree rooted at y. Recursively delete k' and replace k by k' in x.

2.b) If y has fewer than t keys, then check the child z that follows k in node x. If z has atleast t keys, then find the successor k' of k in the subtree rooted at z. Recursively delete k' and replace k by k' in x.

2.c) If both y and z have only (t-1) keys, then merge k and all of z into y so that x loses both k and the pointer to z. Then free z and recursively delete k from y

Case 3

If the key k is not present in internal node x, determine the root c of x of the appropriate subtree that must contain k, if k is in the tree at all.

If c of x has only (t-1) keys, then execute step 3.a or 3.b.

3.a) If c of x has only (t-1) keys, but has an immediate sibling with atleast t keys, give c of x an extra key by moving a key from x down into c of x, moving a key from c of x's immediate left or right sibling up into x and

Scanned by CamScanner

moving the appropriate child pointer from the sibling into c of x

3.b) If c of x and both of c of x's immediate siblings have (t-1) keys, then merge c of x with one sibling which involves moving a key from x down into the new merged node to become the median key for that node.

Now repeat from step 1 taking

x = c of x.

Scanned by CamScanner