Exercise 8 B-Tree

8. Write a C++ Program for implementation of B-tree (Balanced tress) performing following operations
a) Insertion b) Deletion

Objective: The objective of this exercise is to enable you to program a B-Trees and to perform operation on it.

Procedure and description:
In B-trees, internal (non-leaf) nodes can have a variable number of child nodes within some pre-defined range. When data is inserted or removed from a node, its number of child nodes changes. In order to maintain the pre-defined range, internal nodes may be joined or split.

a) Insertion:
Insertion in the B-tree starts with the similar process of searching for an element in the tree, when the search ends at the leaf node element needs to be inserted with the following procedure:

Algorithm:
Step 1: element will be inserted if the set is not full. Full means node contains a maximum of (m-1) elements, given the order of the B-tree to be m.

Step 2: Otherwise find the median and split the set, and push the median one level up, accommodate the median with the parent node if it is not full, otherwise repeat the process, sometimes this leads to rearrangement of elements in root level or the formation of new root itself.

Expected Output:
After execution of program enter choice as insertion operation.

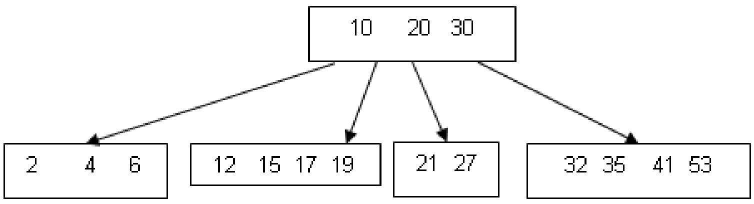

Fig: Initial B-tree before insertion (Here order of tree: 4)

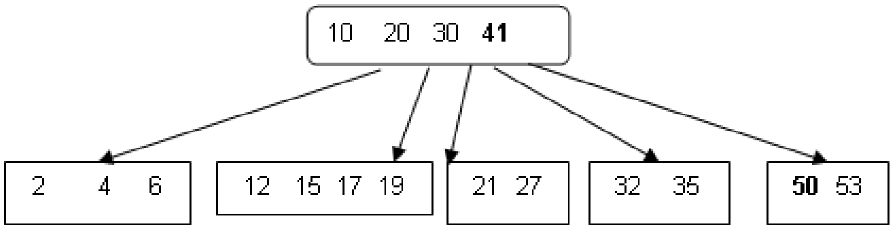

Fig: After inserting element 50

b) Deletion:

Deletion of an element from a b-tree is possible; however, special care must be taken to ensure that the properties of a b-tree are maintained. While deleting an element it is desirable that the elements in the leaf node are removed, but we also get a situation to delete internal node. While deleting an element from the leaf node:

Algorithm:

Step 1: if the leaf node contains more than the minimum number of elements, then the element can be removed.

Step 2: if the node contains just the minimum number of elements then search for an element from left or right sibling to replace the vacancy.

Step 3: If the left siblings have more than the minimum number of elements, pull the largest element up into the parent node and move down the intervening entry from the parent node to the leaf node where the element is to be deleted.

Step 4: otherwise, pull the smallest element of the right sibling to the parent node and move down the intervening parent element to the leaf node.

Step 5: if both the sibling nodes have only minimum numbers of entries then create the new leaf node out of the two leaf nodes and the intervening element of the parent node, ensuring that the total number does not exceed the minimum limit for a node.

Step 6: While barrowing the intervening element from the parent node, it leads the node may be to the minimum number, and then we proceed the process upwards results to reduction of height of the tree.

Expected Output:

After execution of program enter choice as Deletion operation

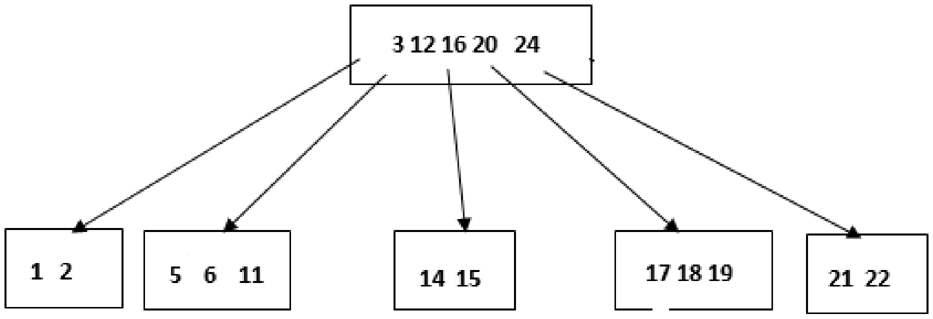

Fig: Initial b-tree before deletion (Here order of tree: 5)

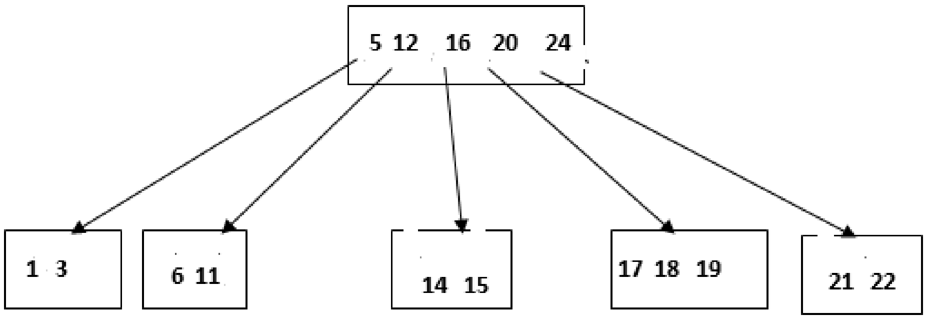

Fig: After deletion of element 2