Data Structures Lab (CSC204) **Session: Monsoon 2021-22**
III BTech (CSE) **Date: 26.10.2021**

Assignment-8 (Tree-II)

---

1. Tree is one of the most important data structures that is used for efficiently performing operations like insertion, deletion and searching of values. However, while working with a large volume of data, construction of a well-balanced tree is essential. An AVL tree is a self-balancing binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one. If at any time they differ by more than one, rebalancing is done to restore this property. Searching, insertion, and deletion all take $O(log\ n)$ time in both the average and worst cases, where n is the number of nodes in the tree prior to these operations. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations. Write a program which can perform following operations over AVL trees.

i. Write a function ***insert_into_Tree*** ( ) which can insert user-given values into AVL tree by maintaining its properties. Your program should be able to print the *Pre-order*, *Post-order*, and *In-order* traversal of the AVL tree obtained after performing insertion operations.

Sample Input:
$Enter\ the\ number\ of\ nodes\ to\ be\ inserted$: 9
$Enter\ the\ node-values$: 24, 98, 21, 5, 7, 9, 15, 67, 14

Sample Output:
$Pre-order\ traversal$: 21, 7, 5, 14, 9, 15, 67, 24, 98
$Post-order\ traversal$: 5, 9, 15, 14, 7, 24, 98, 67, 21
$In-order\ traversal$: 5, 7, 9, 14, 15, 21, 24, 67, 98

ii. Write a function ***delete_from_Tree*** ( ) which is able to delete user-given values from the AVL tree by maintaining its inherent properties. After deletion operation, print the *Pre-order*, *Post-order*, and *In-order* traversal of the remaining AVL tree.

Sample Input:
$Enter\ the\ number\ of\ nodes\ to\ be\ deleted$: 2
$Enter\ the\ node-values\ to\ be\ deleted$: 15, 98

Sample Output:
$Pre-order\ traversal$: 21, 7, 5, 14, 9, 67, 24
$Post-order\ traversal$: 5, 9, 14, 7, 24, 67, 21
$In-order\ traversal$: 5, 7, 9, 14, 21, 24, 67

2. B-tree is a self-balancing tree data structure that maintains sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time. The B-tree generalizes the binary search tree, allowing for nodes with more than two children. Unlike other self-balancing binary search trees, the B-tree is well suited for storage systems that read and write relatively large blocks of data, such as disks. It is commonly used in databases and file systems. A B-tree of order m is a tree which satisfies the following properties:

- Every node has at most m children.
- Every non-leaf node (except root) has at least $\lceil m/2 \rceil$ child nodes.
- The root has at least two children and at most m children if it is not a leaf node.
- A non-leaf node with k children contain $k-1$ keys where $k \leq m$.
- All leaves appear at the same level.

Write a program which can perform following tasks over B-tree:

i. Write a function ***insert_into_BTree*** ( ) which can insert the user-given values to formulate a B-tree. Your program should be able to print the traversal of B-tree obtained after applying insertion operations.

Sample input:
$Enter\ the\ order\ of\ the\ B-Tree: 4$
$Enter\ the\ number\ of\ keys\ to\ be\ inserted: 12$
$Enter\ the\ key-values: 13, 42, 87, 64, 22, 38, 21, 90, 4, 46, 11, 8$

Sample Output:
$Traversal\ of\ B-Tree: 4, 8, 11, 13, 21, 22, 38, 42, 46, 64, 87, 90$

ii. Write a function ***search_in_BTree*** ( ) which can search the presence of user-given key-value in the formulated B-tree.

Sample Input:
$Enter\ the\ key-value\ to\ be\ searched: 90$

Sample Output:
$Key-value\ 90\ is\ present\ in\ the\ B-Tree.$

iii. Write a function ***delete_from_BTree*** ( ) which can delete the user-given values from the constructed B-Tree. Your program should be able to print the traversal of the remaining B-Tree obtained after deletion of user-given key-values.

Sample input:

$Enter\ the\ number\ of\ keys\ to\ be\ deleted$: 3

$Enter\ the\ key-values$: 22, 21, 46

Sample Output:

$Traversal\ of\ B-Tree$: 4, 8, 11, 13, 38, 42, 64, 87, 90