**DSA LAB**

**Lab Assignment number 11**

**Name:** Aamir Ansari  **Batch:** A **Roll no:** 01

**Aim:** Implementation of various operations on binary search tree

**Algorithms:**

Create Node:

getNewNode (data)

Step 1: [INITIALIZE] newNode

Step 2: SET newNode -> data = data

Step 3: SET newNode -> left = NULL

Step 4: SET newNode -> right = NULL

Stem 5: return newNode

Step 6: EXIT

Insertion of node:

Insert (ROOT, VAL)

Step 1: IF ROOT = NULL, then

Allocate memory for ROOT

SET ROOT->DATA = VAL

SET ROOT->LEFT = ROOT ->RIGHT = NULL

ELSE

IF VAL < ROOT->DATA

ROOT->LEFT= Insert(ROOT->LEFT, VAL)

ELSE

ROOT->RIGHT=Insert(ROOT->RIGHT, VAL)

[END OF IF]

[END OF IF]

Step 2: End

Deletion of node:

Delete (ROOT, VAL)

Step 1: IF ROOT = NULL, then

return ROOT

IF VAL < ROOT->DATA

ROOT->LEFT=Delete(ROOT->LEFT, VAL)

ELSE IF VAL > ROOT->DATA

ROOT->RIGHT=Delete(ROOT->RIGHT, VAL)

ELSE

// if node is leaf node or single child node

IF ROOT->LEFT = NULL

TEMP=ROOT->RIGHT

FREE ROOT

RETURN TEMP

ELSE IF ROOT->RIGHT=NULL

TEMP=ROOT->LEFT

FREE ROOT

RETURN TEMP

ELSE

// If node has both left and right child

SET TEMP = findLargestNode(ROOT->LEFT) //inorder predecessor

SET ROOT->DATA = TEMP->DATA

ROOT->LEFT=Delete (ROOT->LEFT, TEMP->DATA)

[END OF IF]

[END OF IF]

Step 2: RETURN ROOT

Step 3: End

Searching for data:

searchElement (ROOT, VAL)

Step 1: IF ROOT ->DATA = VAL OR ROOT = NULL, then

Return ROOT

ELSE

IF VAL < ROOT ->DATA

Return searchElement(ROOT->LEFT,VAL)

ELSE

Return searchElement(ROOT->RIGHT,VAL)

[END OF IF]

[END OF IF]

Step 2: End

Height:

Height (ROOT)

Step 1: IF ROOT = NULL, then

Return 0

ELSE

SET LeftHeight = Height(ROOT ->LEFT)

SET RightHeight = Height(ROOT ->RIGHT)

IF LeftHeight > RightHeight

Return LeftHeight + 1

ELSE

Return RightHeight + 1

[END OF IF]

[END OF IF]Step 2: End

In-order Traversal:

inorderTraversal(root)

STEP 1: IF ROOT != NULL

inorderTraversal(root->left);

printf("%d\t", root->data);

inorderTraversal(root->right);

Step 2: EXIT

Pre-order Traversal:

preorderTraversal(root)

STEP 1: IF ROOT != NULL

printf("%d\t", root->data);

preorderTraversal(root->left);

preinorderTraversal(root->right);

Step 2: EXIT

Post-order Traversal:

postorderTraversal(root)

STEP 1: IF ROOT != NULL

postorderTraversal(root->left);

postorderTraversal(root->right);

printf("%d\t", root->data);

Step 2: EXIT

Count nodes:

totalNodes (ROOT)

Step 1: IF ROOT = NULL, then

Return 0

ELSE

Return totalNodes(ROOT ->LEFT) + totalNodes(ROOT ->RIGHT) + 1

[END OF IF]

Step 2: End

Count Leaf nodes:

countLeafNodes(ROOT)

Step 1: IF ROOT = NULL THEN

return 0

[END IF]

Step 2: IF ROOT -> left = ROOT -> RIGHT = NULL THEN

return 1

ELSE

return countLeafNodes(ROOT->left) + countLeafNodes(ROOT->right)

[END IF]

Step 3: EXIT

Count Non-leaf Nodes:

countNonLeafNodes(ROOT)

Step 1: return countAllNodes(ROOT) – countLeafNodes(ROOT)

Step 2: EXIT

Find Minimum:

findMin(ROOT)

Step 1: Repeat step 2 while ROOT->LEFT != NULL

Step 2: SET ROOT = ROOT -> LEFT

Step 3: return ROOT

Step 4: EXIT

Find Maximum:

findMax(ROOT)

Step 1: Repeat step 2 while ROOT->RIGHT != NULL

Step 2: SET ROOT = ROOT -> RIGHT

Step 3: return ROOT

Step 4: EXIT

Mirror image:

mirrorImage(ROOT)

Step 1: [INITIALIZE] ptr

Step 2: IF ROOT != NULL

Step 3: mirrorImage(root–>left)

Step 4: mirrorImage(root–>right)

Step 5: ptr=ROOT–>left

Step 6: ptr–>left = ptr–>right

Step 7: ROOT–>right = ptr

Step 8: EXIT

Deleting complete tree:

deleteTree(ROOT)

Step 1: IF ROOT != NULL , THEN

deleteTree (ROOT ->LEFT)

deleteTree (ROOT ->RIGHT)

Free (ROOT)

[END OF IF]

Step 2: End