**Assignment No: 05**

**Title: Binary Search Tree**

**Course Outcome:** **CO1(C214447.1):** Analyze algorithms and to determine algorithm correctness and time efficiency class.

**CO2(C214447.2):** Implement abstract data type (ADT) and data structures for given application.

**CO3(C214447.3):** Design algorithms based on techniques like brute -force, divide and conquer, greedy, etc).

**CO5(C214447.5):** Analyze of algorithms with respect to time and space complexity.

**Date of Completion: 13/12/2021**

**Assessment Grade / Marks:**

**Assessor’s Sign with Date:**

**Assignment No: 05**

**Title:** Binary Search Tree Operations

**Aim:** Implementation of Binary Search Tree and Performing Operations.

**Objective:** Implementation of Binary Search Tree and Performing Operations.

**Problem Statement:**Implement binary search tree and perform following operations:

a) Insert (Handle insertion of duplicate entry)

b) Delete

c) Search

d) Display tree (Traversal)

e) Display - Depth of tree

f) Display - Mirror image

g) Create a copy

h) Display all parent nodes with their child nodes

i) Display leaf nodes

j) Display tree level wise

**Course Outcome:** CO Number: Applicable CO : Blooms Taxonomy Category

**Requirements:** (Components / Digital Kits / Platform / Software / Hardware)

**Platform :-** Online GDB Compiler

**Theory / Procedure / Diagrams / Circuits:**

**Binary Search Tree :-** Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers.It is called a binary tree because each tree node has a maximum of twochildren.It is called a search tree because it can be used to search for the presence of a number in O(log(n)) time.

The properties that separate a binary search tree from a regular binary tree is:

* All nodes of left subtree are less than the root node
* All nodes of right subtree are more than the root node
* Both subtrees of each node are also BSTs i.e. they have the above two properties.

**Some Basic Operations of Binary Search Tree Are as Follows:-**

**1. Search Operation :-** The algorithm depends on the property of BST that if each left subtree has values below root and each right subtree has values above the root.If the value is below the root, we can say for sure that the value is not in the right subtree; we need to only search in the left subtree and if the value is above the root, we can say for sure that the value is not in the left subtree; we need to only search in the right subtree.

**2. Insert Operation :-** Inserting a value in the correct position is similar to searching because we try to maintain the rule that the left subtree is lesser than root and the right subtree is larger than root.We keep going to either right subtree or left subtree depending on the value and when we reach a point left or right subtree is null, we put the new node there.

**3. Deletion :-** There are three cases for deleting a node from a binary search tree.

**Case I:**

In the first case, the node to be deleted is the leaf node. In such a case, simply

delete the node from the tree.

**Case II:**

In the second case, the node to be deleted lies has a single child node. In such a case follow the steps below:

Replace that node with its child node.

Remove the child node from its original position.

**Case III:**

In the third case, the node to be deleted has two children. In such a case follow the steps below:

1. Get the inorder successor of that node.

2. Replace the node with the inorder successor.

3. Remove the inorder successor from its original position.

**Binary Search Tree Applications:**

1. In multilevel indexing in the database

2. For dynamic sorting

3. For managing virtual memory areas in Unix kernel

**Algorithm / Methods / Steps: (if applicable):**

MAIN FUNCTION()

STEP 1: Create a 'BSTNode' structure with three members key,left and right.

STEP 2: Create a ‘Stack’ structure with two members Root and next.

PUSH FUNCTION()

STEP 1: Create a newNode

STEP 2: Set newNode-->Root = Root

STEP 3: Set newNode-->next = \*Top

STEP 4: Set \*Top = newNode

POP FUNCTION()

STEP 1: Define a Node pointer 'temp' and set it to 'Top'

STEP 2: Set \*Top = (\*Top)->next;

STEP 3: Return temp

SEARCH FUNCTION()

STEP 1: IF (node == NULL)

STEP 2: Return NULL

STEP 3: END IF

STEP 4: IF (key < (node->key)

STEP 5: Return Search(node->left,key)

STEP 6: END IF

STEP 7: IF (key > (node->key)

STEP 8: Return Search(node->right,key)

STEP 9: END IF

STEP 10: ELSE

STEP 11: Return node

INSERT FUNCTION()

STEP 1:IF (node==NULL)

STEP 2:Create ‘NewNode’ pointer

STEP 3:Set NewNode->key=key

STEP 4:Set NewNode->left = NewNode->right=NULL

STEP 5:Return NewNode

STEP 6:END IF

STEP 7:IF (key <= (node->key))

STEP 8:Set node->left = Insert(node->left,key)

STEP 9:END IF

STEP 10:ELSE IF (key > (node->key))

STEP 11:Set node->right = Insert(node->right,key)

STEP 12:Return node

DELETE FUNCTION()

STEP 1: IF (key < root->key)

STEP 2:Set root->left = Delete(root->left,key)

STEP 3:END IF

STEP 4:ELSE IF (key > root->key)

STEP 5:Set root->right = Delete(root->right,key);

STEP 6:ELSE

STEP 7:IF (root->left == NULL)

STEP 8:Set struct BSTNode \*temp = root->right

STEP 9:free(root)

STEP 10:Return temp

STEP 11:END IF

STEP 12:ELSE IF (root->right == NULL)

STEP 13:Set struct BSTNode \*temp = root->left;

STEP 14:free(root)

STEP 15:Return temp

STEP 16:END ELSEIF

STEP 17:Set struct BSTNode \*temp = InOrder\_Successor(root->right);

STEP 18:Set root->key = temp->key

STEP 19:Set root->right = Delete(root->right,temp->key)

STEP 20:Return root

INORDER\_SUCCESSOR FUNCTION()

STEP 1:Define pointer ‘current’ and Set = node

STEP 2:WHILE (current && current->left != NULL)

STEP 3:Set current = current->left

STEP 4:END WHILE

STEP 5:Return current

INORDER\_TRAVERSAL\_RECURSIVE FUNCTION()

STEP 1: IF (Root != NULL)

STEP 2: Get Inorder\_Traversal\_Recursion (Root->left)

STEP 3: Get Root->key

STEP 4: Get Inorder\_Traversal\_Recursion (Root->right)

STEP 5: END IF

PREORDER\_TRAVERSAL\_RECURSIVE FUNCTION()

STEP 1: IF (Root != NULL)

STEP 2: Get Root->key

STEP 3: Get Preorder\_Traversal\_Recursion (Root->left)

STEP 4: Get Preorder\_Traversal\_Recursion (Root->right)

STEP 5: END IF

POSTORDER\_TRAVERSAL\_RECURSIVE FUNCTION()

STEP 1: IF (Root != NULL)

STEP 2: Get Postorder\_Traversal\_Recursion (Root->left);

STEP 3: Get Postorder\_Traversal\_Recursion (Root->right);

STEP 4: Get Root->key

STEP 5: END IF

INORDER\_TRAVERSAL\_NON\_RECURSIVE FUNCTION()

STEP 1: IF (Root != NULL)

STEP 2: Set ‘Top’ pointer to’NULL’

STEP 3: Define ‘temp’ Pointer

STEP 4: Define ‘current Node’ and Set it = Root

STEP 5: WHILE (current\_Node != NULL || Top != NULL)

STEP 6: WHILE (current\_Node != NULL)

STEP 7: Push (current\_Node, &Top)

STEP 8: Set current\_Node = current\_Node->left

STEP 9: END WHILE

STEP 10: IF (current\_Node == NULL && Top != NULL)

STEP 11: Set temp = Pop (&Top)

STEP 12: Get temp->key

STEP 13: Set current\_Node = temp->right

STEP 14: END WHILE

STEP 15: END IF

PREORDER\_TRAVERSAL\_NON\_RECURSIVE FUNCTION()

STEP 1: IF (Root != NULL)

STEP 2: Set ‘Top’ pointer to’NULL’

STEP 3: Define ‘temp’ Pointer

STEP 4: Push (Root, &Top)

STEP 5: WHILE (Top != NULL)

STEP 6: Get temp->key

STEP 7: IF (temp->right != NULL)

STEP 8: Push (temp->right, &Top)

STEP 9: END IF

STEP 10: IF (temp->left != NULL)

STEP 11: Push (temp->left, &Top)

STEP 12: END IF

STEP 13: END WHILE

STEP 14: END IF

POSTORDER\_TRAVERSAL\_NON\_RECURSIVE FUNCTION()

STEP 1: IF (Root != NULL)

STEP 2: Define ‘temp’ Pointer

STEP 3: Define ‘Num 1’ and ‘Num 2’ Pointer AND Set them = NULL

STEP 4: Push (Root, &Num1)

STEP 5: WHILE (Num1 != NULL)

STEP 6: Set temp = Pop (&Num1)

STEP 7: Push (temp, &Num2)

STEP 8: IF (temp->left != NULL)

STEP 9: Push (temp->left, &Num1)

STEP 10: END IF

STEP 11: IF (temp->right != NULL)

STEP 12: Push (temp->right, &Num1);

STEP 13: END IF

STEP 14: END WHILE

STEP 15: WHILE (Num2 != NULL)

STEP 16: Get Pop (&Num2)->key

STEP 17: END WHILE

STEP 18: END IF

HEIGHT FUNCTION()

STEP 1:IF(root == NULL)

STEP 2:Return 0

STEP 3:END IF

STEP 4:ELSE

STEP 5:Set int leftHeight = Height(root->left)

STEP 6:Set int rightHeight = Height(root->right)

STEP 7:IF (leftHeight > rightHeight)

STEP 8:Return (leftHeight + 1)

STEP 9:END IF

STEP 10:ELSE

STEP 11: Return(rightHeight + 1)

PRINT\_LEVEL FUNCTION()

STEP 1:IF (root == NULL)

STEP 2:Return

STEP 3:END IF

STEP 4:ELSE IF (level==0)

STEP 5:Get root->key

STEP 6:ELSE

STEP 7:Get printLevel(root->left,level-1)

STEP 8:Get printLevel(root->right,level-1)

LEVEL\_ORDER\_TRAVERSAL FUNCTION()

STEP 1:Int h = Height(root)

STEP 2:FOR (int i=0;i<=h;i++)

STEP 3:Get printLevel(root,i)

STEP 4:END FOR

DISPLAY\_LEAF\_NODES FUNCTION()

STEP 1:IF (root == NULL)

STEP 2:Return

STEP 3:END IF

STEP 4:IF (root->left == NULL && root->right ==NULL)

STEP 5:Get root->key

STEP 6:END IF

STEP 7:IF (root->left)

STEP 8:Get DisplayLeafNodes(root->left)

STEP 9:END IF

STEP 10:IF (root->right)

STEP 11:Get DisplayLeafNodes(root->right)

STEP 12:END IF

DISPLAY\_MIRROR FUNCTION()

STEP 1:IF (root == NULL)

STEP 2:Return

STEP 3:END IF

STEP 4:ELSE

STEP 5:Get DisplayMirror(root->left)

STEP 6:Get DisplayMirror(root->right)

STEP 7:Swap (root->left) AND (root->right)

Input: (Test Cases / Data sets / Database Links)

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

1

Output: (Results / Visualization)

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

1

====================================================================

Enter the element you want to Insert :

13

====================================================================

=================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

1

====================================================================

Enter the element you want to Insert :

5

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

1

====================================================================

Enter the element you want to Insert :

10

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

1

====================================================================

Enter the element you want to Insert :

21

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

1

====================================================================

Enter the element you want to Insert :

8

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

4

====================================================================

Display BST

Recursive Traversal:

1. In-Order Traversal :5 8 10 13 21

2. Pre-Order Traversal :13 5 10 8 21

3. Post-Order Traversal :8 10 5 21 13

Non-Recursive Traversal:

1. In-Order Traversal :5 8 10 13 21

2. Pre-Order Traversal :13 5 10 8 21

3. Post-Order Traversal :8 10 5 21 13

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

2

====================================================================

Enter the element you want to Delete :

21

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

4

====================================================================

Display BST

Recursive Traversal:

1. In-Order Traversal :5 8 10 13

2. Pre-Order Traversal :13 5 10 8

3. Post-Order Traversal :8 10 5 13

Non-Recursive Traversal:

1. In-Order Traversal :5 8 10 13

2. Pre-Order Traversal :13 5 10 8

3. Post-Order Traversal :8 10 5 13

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

3

====================================================================

Enter the element you want to Search :

13

Element 13 Is Found

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

1

====================================================================

Enter the element you want to Insert :

21

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

4

====================================================================

Display BST

Recursive Traversal:

1. In-Order Traversal :5 8 10 13 21

2. Pre-Order Traversal :13 5 10 8 21

3. Post-Order Traversal :8 10 5 21 13

Non-Recursive Traversal:

1. In-Order Traversal :5 8 10 13 21

2. Pre-Order Traversal :13 5 10 8 21

3. Post-Order Traversal :8 10 5 21 13

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

5

====================================================================

Level Order Traversal of Tree : 13 5 21 10 8

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

6

====================================================================

Height of the Tree : 4

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

7

====================================================================

Display all Leaf Nodes : 8 21

====================================================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

====================================================================

1.Insert

2.Delete

3.Search

4.Display

5.Level Order Traversal (BFS)

6.Height of the Tree

7.Display Leaf Nodes

8.Mirror Image of Binary Tree

Enter your choice :

8

====================================================================

Mirror Image of Binary Tree : 21 13 10 8 5

**Inference:** Hence we have studied the construction of Binary Search Tree and performed

various operations.