Lab sheet 8

Binary Search Tree

NAME :Patel Rajkumar Pankajbhai

Batch : CSE – D

Roll .No : U4CSE2049

1. Create a class BST and implement the insert function.

Code :

private Node Insert(Node *currentNode*, *int* *value* )

{

if (*currentNode* == null)

{

return **new** Node(*value*) ;

}

if (*value* < *currentNode*.value) {

*currentNode*.left = Insert(*currentNode*.left, *value*) ;

}

else if (*value* > *currentNode*.value)

{

*currentNode*.right = Insert(*currentNode*.right, *value*) ;

}

return *currentNode* ;

}

1. Implement inorder traversal in class BST

Code :

private *void* pinorder(Node *root*) {

if (*root*==null) {

return;

}

pinorder(*root*.left) ;

System.out.print(*root*.value + " ");

pinorder(*root*.right) ;

}

1. Create Driver class and insert the elements 30, 35, 40,50,12,17,45,90,23,56 in order to an initially empty BST.

Code :

public static *void* main(String[] *args*) {

BST b = **new** BST() ;

b.insert(30); b.insert(17);

b.insert(45); b.insert(90);

b.insert(23); b.insert(56);

b.insert(35); b.insert(40);

b.insert(50); b.insert(12);

}

1. Perform inorder traversal of the tree created in question 3.

Output :

Text

Description automatically generated

1. Implement postorder and preorder traversals in class BST.

Code :

private *void* ppreorder(Node *root*) {

if (*root* == null ) {

return ;

}

System.out.print(*root*.value + " ");

ppreorder(*root*.left);

ppreorder(*root*.right);

}

private *void* printPostorder(Node *node*)

{

if (*node* == null)

return;

printPostorder(*node*.left);

printPostorder(*node*.right);

System.out.print(*node*.value + " ");

}

1. Implement the delete function in class BST

Code :

private *void* del(Node *root*) {

if (*root*.left==null&&*root*.right!=null) {

*root*=*root*.right;

*root*.right=null;

}

else if (*root*.left!=null&&*root*.right==null) {

*root*=*root*.left;

*root*.left=null;

}

else if (*root*.left!=null&&*root*.right!=null) {

*root*.value=in\_suc(*root*);

}

}

1. Delete the element 17 from the tree created in question 3.

Output :

Text

Description automatically generated

1. Print the postorder and preorder traversals of the tree

Output :

Text

Description automatically generated

1. Implement the search function in class BST.

Code :

public Node search(*int* *value*) {

Node k = root ;

while (k.value!=*value*) {

if (k.value<*value*) {

k=k.right;

}

else {k=k.left;}

}

return k ;

}

1. Search for the given elements and see the result: 40, 90, 32, 92, 56.

Text

Description automatically generated

1. Using search function, display minimum and maximum elements in the tree.
2. Implement a function to find the height of the tree

Code :

private *int* maxDepth(Node *node*)

{

if (*node* == null)

return -1;

else

{

*int* lDepth = maxDepth(*node*.left);

*int* rDepth = maxDepth(*node*.right);

if (lDepth > rDepth)

return (lDepth + 1);

else

return (rDepth + 1);

}

}

1. Find the height of the tree created in question 3.

Output :

Graphical user interface, text

Description automatically generated

1. Implement a function to find the kth largest element in BST
2. Find the 5th largest element of the tree
3. Given a Binary Tree, convert it to a Binary Search Tree. The conversion must be done in such a way that keeps the original structure of Binary Tree.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*