**DAY-26 (3-12-2024)**

* **TREES:**
* A tree is an ideal data structure for representing hierarchical data
* Tree is a special case of graph having no loops ,no circuits and no self loops
* Traversal

1. Inorder 🡪 left-root-right
2. Preorder🡪 root-left-right
3. Postorder 🡪 left-right-root

Different types of trees:

1. **Binary tree** 🡺 it is a rooted tree in which no vertex has more than two children

Terminology of trees:

1. Node:
2. Degree of a node: number of children
3. Leaf: no children is present
4. Level: different levels and it grow from the root to leaf , root is having the level 1
5. Parent
6. Children
7. Siblings: same parent
8. Degree of tree
9. Ancestor

* **Full or strict binary tree:** having 0 or 2 children
* **Perfect binary tree:** if all its internal nodes has exactly 2 children. All the leaf nodes at the same level or depth
* **Complete binary tree:** a complete binary is a binary tree in which every level, except possibly at last completely filled and all nodes are as far left as possible .it is means to be balanced tree
* All perfect binary trees are complete binary trees but reverse is not true
* **Balanced binary tree:** for all nodes difference of left and right subtree height is not more than one
* Questions:

1. Write a c program to insert nodes in a binary tree -inorder,pre order,post order traversal
2. Write a c program to insert nodes in a binary search tree
3. **Write a c program to find level of the tree**
4. write a c program to max/min elements of a tree

* TRAVERSALS:

**Depth first search traversal (DFS)**:

* Preorder traversal (vertex,left ,right subtree)
* Inorder traversal (left,vertex,right) 🡪 it is already sorted
* Postorder traversal(left,right,vertex)

**Breadth first search traversal**

Example:

5 10 3 4 8 1 2 13

5

3 10

1 4 8 13

2

Left subtree have to be lesser than the root node ,right subtree have to be greater than the root node

* Inorder traversal (LNR) 🡺 1 2 3 4 5 8 10 13

If we came across node(root) then we have print it ,LNR(LEFT,NODE,RIGHT) should do it for all the nodes

* Preorder (NLR)🡺 5 3 1 2 4 10 8 13 ( the first element in the preorder which is middle of the tree or root element of the list)
* Postorder(LRN) 🡺 2 1 4 3 8 13 10 5 ( the first element in the postorder is the last level of the tree)

Program for the inorder traversal

// Online C compiler to run C program online

#include <stdio.h>

#include<stdlib.h>

typedef struct node

{

int data;

struct node \*l;

struct node \*r;

}NODE;

NODE \*createnode(int val)

{

NODE \*nn=(NODE\*)malloc(sizeof(NODE));

nn->data=val;

nn->l=NULL;

nn->r=NULL;

return nn;

}

NODE \*insertNode(NODE \*root,int val)

{

if(root==NULL)

{

return createnode(val); //return nn

}

if(root->data<val)

{

// right subtree

root->r=insertNode(root->r,val);

}

else if(root->data>val)

{

// left subtree

root->l=insertNode(root->l,val);

}

else

{

printf("\n %d duplicate value found.hence ignore",val);

}

return root;

}

void inorder(NODE \*root)

{

if(root==NULL)

{

return;

}

inorder(root->l);

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

inorder(root->r);

}

int main() {

NODE \*root=NULL;

root=insertNode(root,5);

root=insertNode(root,10);

root=insertNode(root,3);

root=insertNode(root,8);

root=insertNode(root,1);

root=insertNode(root,2);

root=insertNode(root,13);

printf("inorder traversal");

inorder(root);

printf("\n\n");

return 0;

}

Program for preorder and postorder:

// Online C compiler to run C program online

#include <stdio.h>

#include<stdlib.h>

typedef struct node

{

int data;

struct node \*l;

struct node \*r;

}NODE;

NODE \*createnode(int val)

{

NODE \*nn=(NODE\*)malloc(sizeof(NODE));

nn->data=val;

nn->l=NULL;

nn->r=NULL;

return nn;

}

NODE \*insertNode(NODE \*root,int val)

{

if(root==NULL)

{

return createnode(val); //return nn

}

if(root->data<val)

{

// right subtree

root->r=insertNode(root->r,val);

}

else if(root->data>val)

{

// left subtree

root->l=insertNode(root->l,val);

}

else

{

printf("\n %d duplicate value found.hence ignore",val);

}

return root;

}

void inorder(NODE \*root)

{

if(root==NULL)

{

return;

}

inorder(root->l);

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

inorder(root->r);

}

void preorder(NODE \*root)

{

if(root==NULL)

{

return;

}

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

preorder(root->l);

preorder(root->r);

}

void postorder(NODE \*root)

{

if(root==NULL)

{

return;

}

postorder(root->l);

postorder(root->r);

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

}

int main() {

NODE \*root=NULL;

root=insertNode(root,5);

root=insertNode(root,10);

root=insertNode(root,3);

root=insertNode(root,8);

root=insertNode(root,1);

root=insertNode(root,2);

root=insertNode(root,13);

printf("inorder traversal");

inorder(root);

printf("\n\n");

printf("preorder traversal");

preorder(root);

printf("\n\n");

printf("postorder traversal");

postorder(root);

printf("\n\n");

return 0;

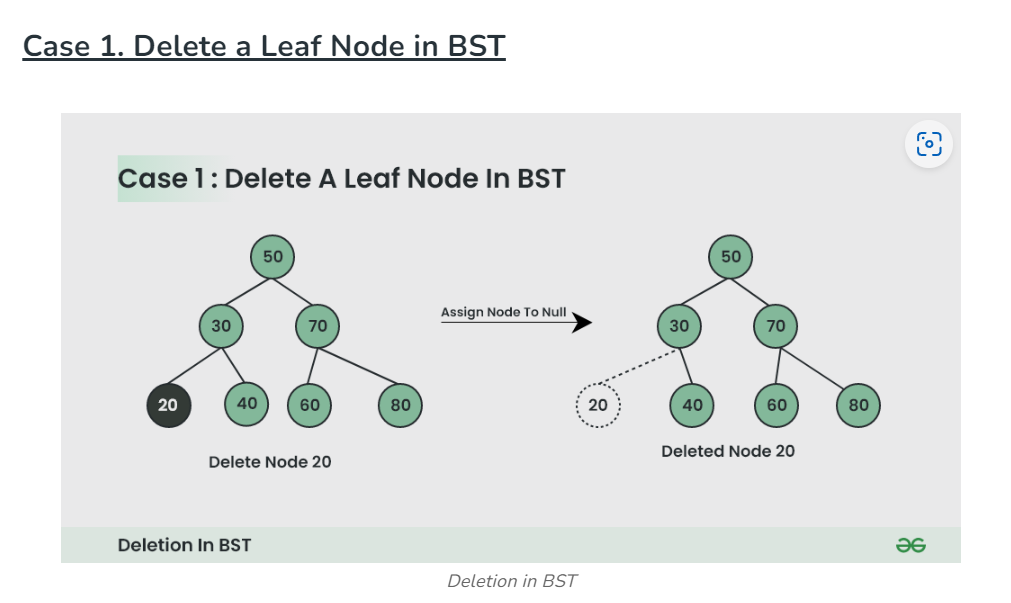
}

DELETE A NODE IN BST: to delete a node there are 3 possibilities

1)delete a leaf node

2) delete a node with 1 child (either left or right)

3) delete a node with 2 child



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Note:** Inorder successor is needed only when the right child is not empty. In this particular case, the inorder successor can be obtained by finding the minimum value in the right child of the node.

Program:

#include <stdio.h>

#include <stdlib.h>

typedef struct node{

int v;

struct node \*left;

struct node \*right;

}NODE;

NODE \*createNode(int val)

{

NODE \*nn = (NODE \*)malloc(sizeof(NODE));

nn->v = val;

nn->left = NULL;

nn->right = NULL;

return nn;

}

NODE \*insertNode(NODE \*root, int val)

{

// printf("\ninserted nn\n");

if (root == NULL )

{

//new tree/tree is not present hence

//1 st node is the root node

// printf("\nCreated nn\n");

return(createNode(val));

}

else if(val < root->v )

{

//left sub tree

root->left = insertNode(root->left,val);

}

else if(val > root->v)

{

//right sub tree

root->right = insertNode(root->right,val);

}

else

{

printf("\n%d Duplicate Value detected. Hence ignored\n",val);

}

return root;

}

void inOrderTrav(NODE \*root)

{

//left->root->right

if(root==NULL)

return;

inOrderTrav(root->left);

printf(" %d ",root->v);

inOrderTrav(root->right);

}

void preOrderTrav(NODE \*root)

{

//root->left-right

if(root==NULL)

return;

printf(" %d ",root->v);

preOrderTrav(root->left);

preOrderTrav(root->right);

}

void postOrderTrav(NODE \*root)

{

//left->right->root

if(root==NULL)

return;

postOrderTrav(root->left);

postOrderTrav(root->right);

printf(" %d ",root->v);

}

// Function to find the minimum value node in a tree

NODE \*findMin(NODE \*root) {

while (root && root->left != NULL) {

root = root->left;

}

return root;

}

// Function to delete a node in the BST

NODE \*deleteNode(NODE \*root, int val) {

if (root == NULL) {

return root; // Tree is empty or value not found

}

// Find the node to be deleted

if (val < root->v) {

root->left = deleteNode(root->left, val);

} else if (val > root->v) {

root->right = deleteNode(root->right, val);

} else {

// Node to be deleted is found

// Case 1: Node has no child (leaf node)

if (root->left == NULL && root->right == NULL) {

free(root);

return NULL;

}

// Case 2: Node has one child (left or right)

else if (root->left == NULL) {

NODE \*temp = root->right;

free(root);

return temp;

} else if (root->right == NULL) {

NODE \*temp = root->left;

free(root);

return temp;

}

// Case 3: Node has two children

else {

NODE \*temp = findMin(root->right); // Find in-order successor

root->v = temp->v; // Replace value with successor

root->right = deleteNode(root->right, temp->v); // Delete successor

}

}

return root;

}

// Free the allocated memory of the tree nodes

void freeTree(NODE \*root) {

if (root == NULL) return;

freeTree(root->left);

freeTree(root->right);

free(root);

}

int main()

{

NODE \*root=NULL;

root = insertNode(root,5);

insertNode(root,10);

insertNode(root,3);

insertNode(root,4);

insertNode(root,8);

insertNode(root,1);

insertNode(root,2);

insertNode(root,13);

insertNode(root,5);

/\*

5

3 10

1 4 8 13

2 5

inorder(LNR)

1 2 3 4 5 8 10 13

preOder(NLR)

5 3 1 2 4 10 8 13

postOrder(LRN)

2 1 4 3 8 13 10 5

2 1 4 3 5 8 13 10 5

\*/

//traversals

printf("\ninOrder Traversal\n");

inOrderTrav(root);

printf("\npreOrder Traversal\n");

preOrderTrav(root);

printf("\npostOrder Traversal\n");

postOrderTrav(root);

deleteNode(root,10);

//traversals

printf("\ninOrder Traversal\n");

inOrderTrav(root);

printf("\npreOrder Traversal\n");

preOrderTrav(root);

printf("\npostOrder Traversal\n");

postOrderTrav(root);

freeTree(root);

printf("\n\n");

return 0;

}