12/08/2024

Page No.
Date

# Day 15 of DSA.

## Tree

Check Box

- [x] Binary Tree : Basic properties and operations
- [x] Binary Search Tree : Insertion, deletion, searching.
- [x] Implementation basic operations of binary tree
- [x] Implement insertion, deletion, searching in BST.

Page No.
Date

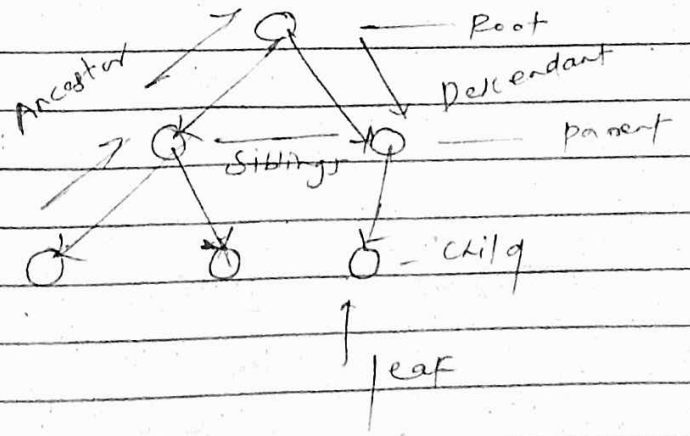
## * Binary Tree *.

← Non-Linear Data structure

1. Node
2. Root
3. parent
4. child
5. siblings
6. Ancestor
7. Descendant
8. Leaf

* Implementation OR Initialization

```
Node
{
    int data;
    Node * left;
    Node * right;
};
```



Page No.
Date

```
Node * buildTree(Node *root)
{
    cout << "Enter the data :" << endl;
    int data;
    cin >> data;
    root = new node(data);

    if (data == -1) {
        return NULL;
    }

    cout << " Enter the data for inserting
            in left of " << data << endl;
    root->left = buildTree(root->left);
    cout << " Enter the data for inserting in
            right of " << data << endl;
    root->right = buildTree(root->right);
    return root;
}
```

✓

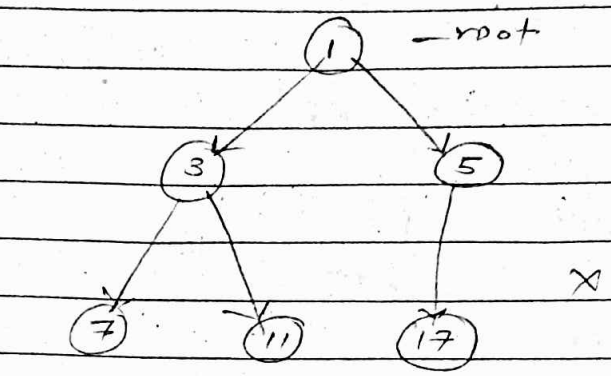
Page No.
Date

## Level order Traversal

```
void levelOrderTraversal(node* root){
    queue<node*> q;
    q.push(root);
    q.push(NULL);

    while(!q.empty()){
        node* temp = q.front();
        q.pop();

        if(temp == NULL){
            cout << endl;
            if(!q.empty()){
                q.push(NULL);
            }
        }
        else{
            cout << temp->data << " ";
            if(temp->left){
                q.push(temp->left);
            }
            if(temp->right){
                q.push(temp->right);
            }
        }
    }
}
```

LNR NLR LRN

Inorder / Preorder / postorder Traversal




✓ 7 3 11 1 17 5 = Inorder

✓ 1 3 7 11 5 17 - preorder

✓ 7 11 3 17 5 1 - postorder

Page No.
Date

## Build From level order → Queue:

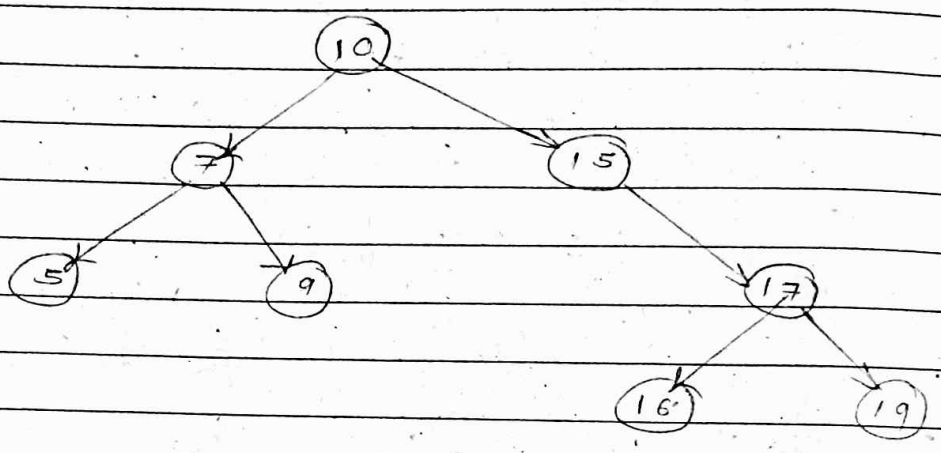
```
node * buildFromLevelOrder(node * root){
    queue<node *> q;
    cout << "Enter data for root" << endl;
    int data;
    cin >> data;
    root = new node(data);
    q.push(root);
    while(!q.empty()){
        node * temp = q.front();
        q.pop();

        cout << "Enter left node for: " << temp->data <<
                endl;
        int leftData;
        cin >> leftData;

        if(leftData != -1){
            temp->left = new node(leftData);
            q.push(temp->left);
        }

        cout << "Enter right node for: " << temp->data <<
                endl;
        int rightData;
        cin >> rightData;

        if(rightData != -1)
        {
            temp->right = new node(rightData);
            q.push(temp->right);
        }
```

Page No.
Date

* Binary Search Tree.

Left side - small elements.

Right side - greater elements




* Initialization:-

```
class Node
{
    public:
        int data;
        Node * left;
        Node * right;

    Node(int d){
        this -> data = d;
        this -> left = NULL;
        this -> right = NULL;
    }
};
```

Page No.
Date

(1) Insert

```
Node* insertIntoBST(Node *root, int d)
{
    if (root == NULL)
    {
        root = new Node(d);
        return root;
    }
    if (d > root->data)
    {
        root->right = insertIntoBST(root->right, d);
    }
    else
    {
        root->left = insertIntoBST(root->left, d);
    }
    return root;
}
```

(2) Search

```
bool searchInBST(Node* &root, int x)
{
    if (root == NULL)
    {
        return False;
    }
    if(root->data == x)
    {
        return true;
    }
    if(root->data > x)
    {
        return searchInBST(root->left, x);
    }
    else{
        return searchInBST(root->right, x);
    }
}
```

Page No.
Date

③ Min value:-

```
temp = root
while ( temp -> left != NULL)
{
    temp = temp -> left;
}
return temp
```

④ max value:-

```
temp = root
while ( temp -> right != NULL)
{
    temp = temp -> right;
}
return temp;
```

* delete.

(1) 0 child :-

Just delete that node

(2) 1 child :-

Just Remove that Node
& Link its upper & lower
Node

(3) 2 child :-

Find out min value from right side
& then put into that
element & Remove it:
(min value)