Date _____ Page _____

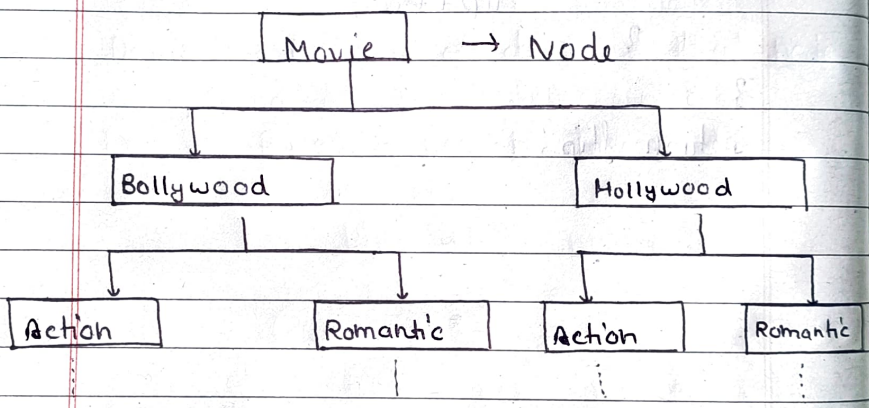
Day - 149

Trees

* **Trees:**

⇒ It is a type of DS that represent a hierarchial relationship between data element called nodes.

⇒ We are using Trees everyday in our life i.e. —



⇒ This is Trees DS.

⇒ So, here hierarchy is maintaining at every level.

⇒ Every nodes have relationship between them.

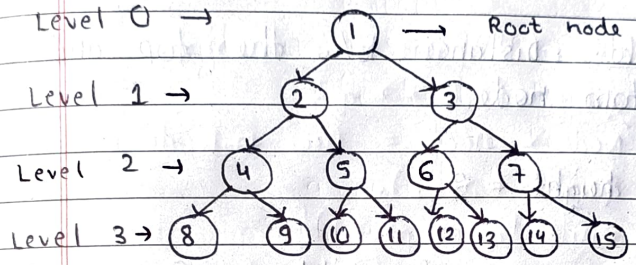
⇒ Trees helps in reducing the searching time.

Date: Page:

* **Binary Tree:**

⇉ It is defined as a Tree DS where each node has atmost 2 children.

Terms in Trees



⇉ **Root:** Topmost node of the tree or a node that don't have any parent.

⇉ **Parent:** Node that is above of any node.

⇉ **Leaf:** Node that don't have any children.

⇉ **Child:** Node originate from a node is the child of that node.

⇉ **Ancestor:** All the above connected nodes of that node are the ancestor of that node.

Date Page

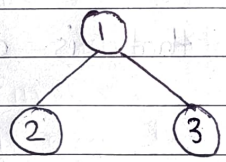
⇉ Descendent: All the below connected nodes of a node.

⇉ Sibling: Nodes that have same parent are siblings.

⇉ Height: Distance b/w the top and the bottom node.
So,
height = 3

⇉ Degree: No. of children of that node.

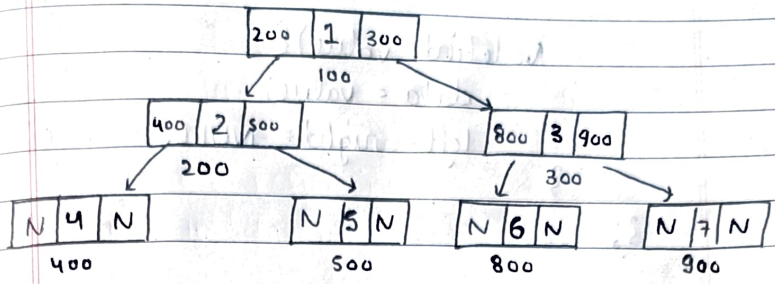
Ex. of Binary Search Tree



⇉ No. of edges in n-node BST:
Total edges = n-1

* Representation of Tree:

Left	Data	Right



Internal Representation

⇉ When anyone give -1 then we don't have to create child of that node.

⇉ We will make the tree from left to right.

⇉ So, how will we know which node we have to select for their left & right child.

⇉ For that, when ~~we~~ any node come, we will add it to the queue.

⇉ Now, we will select front element of the queue for child.

Code

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

Date Page

```
Node(int value){
    data = value;
    left = right = NULL;
}
};

int main(){ int x ; cin >> x;
    queue <Node *> q;
    Node * root = new Node(x);
    q.push(root);
    while(!q.empty()){
        Node * temp = q.front();
        q.pop();
        cin >> x;      // left child
        if( x != -1){
            temp->left = new Node(x);
            q.push(temp->left);}
        cin >> x;      // right child
        if( x != -1){
            temp->right = new Node(x);
            q.push(temp->right);
        }
    }
}
```