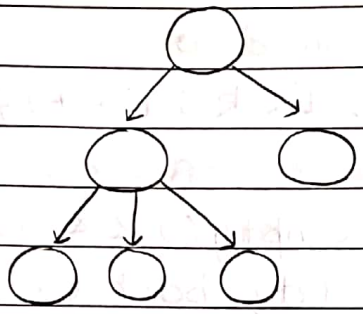
05/05/2023

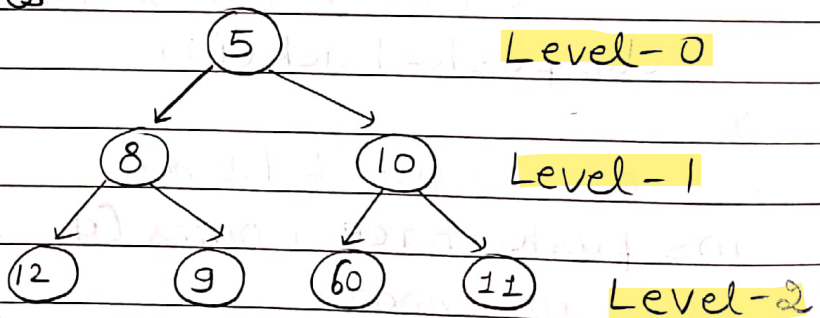
Trees

It is a non-linear data structure.



The above is an example of tree data structure.

Terminologies



The above tree is a binary tree as every node is having maximum 2 children.

1) node

Single entity in a tree is called node. Structure of node of tree is as follows :-

```
class TreeNode {
    int data;
    TreeNode * left;
    TreeNode * right;
};
```

2) Root

Topmost node in the tree is known as the root node. Here 5 is the root node.

Parent of root node can be itself or null depending on the question.

3) Parent
Parent of a particular node is the one from which it is generated. Ex → 8 is the parent of 12 and 9.

4) Sibling
The nodes which are at same level, are known as siblings. Ex → 8 and 10 nodes are siblings. 12 and 9 are siblings. and have same parent

5) Ancestors
Ancestor of 11 is 10 and 5.

6) Descendant
Descendant of 8 is 12 and 9

7) Leaf nodes
The nodes which does not have any child node is known as leaf nodes. Ex → 12, 9, 60 and 11 are leaf nodes.

8) Child
Immediate next node of a particular node is known as child node. Ex → Child of 5 is 8 and 10.

Note → Ancestor of 60 is 10 and 5 whereas parent of 60 is 10 only.
Descendant of 5 is 8, 10, 12, 9, 60 and 11 whereas child node of 5 is 8 & 10 only.

Note ⇒ Online test
- → MCQs Formulae based question
- → Coding test 80-90% famous classical ques
- → 10% question Mixture of data structure

Tree implementation

1) Creation of node

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

2) Just create root node and then recursion will handle.

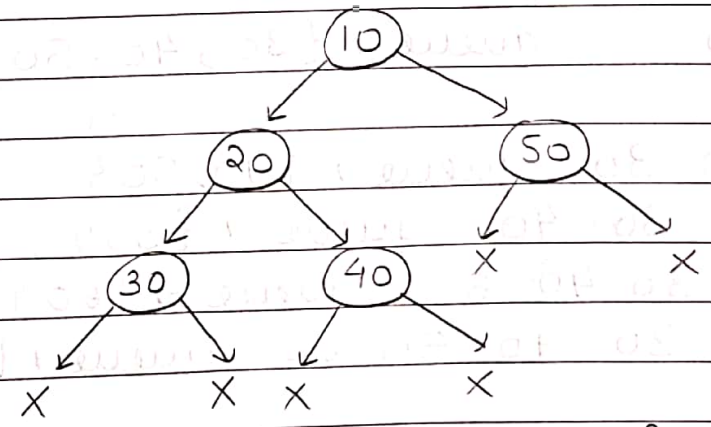
Code

```
class Node {
    public :
        int data;
        Node * left;
        Node * right;
        Node (int data) {
            this -> data = data;
            left = NULL;
            right = NULL;
        }
};
Node * buildTree (int data) {
    // -1 data indicates that we have arrived
    at the leaf node & hence return NULL.
```

↳ Depends on question.

```
if(data == -1){
    return NULL;
}
//Create root node (Solve one case)
Node * root = new Node(data);
//Recursion handles for left & right subtree
int leftData;
cout << "Enter data of left of " <<data<<endl;
cin >> leftData;
root->left = buildTree(leftData);
int rightData;
cout << "Enter data of right of " <<data <<endl;
cin >> rightData;
root->right = buildTree(rightData);
return root; //Return the root node.
}
```

Flow of the code

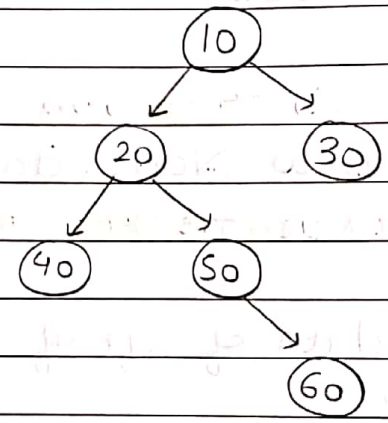


10 >20>30> -1 >-1 >40 > -1 > -1 > 50 > -.1 > -1 } Order of creation of nodes

Traversals in tree

There are various ways in which the tree can be traversed.

1) Level order traversal



10 20 30 40 50 60

This can done with the help of queue

1) Push the root node in the queue.
   queue → {10}
2) Print front element and pop it and push the child of the removed element.

   O/p → 10 queue → {20, 30}

3)

   O/p → 10 20 queue → {30, 40, 50}
   Child of 20

4) O/p → 10 20 30 queue → {40, 50}
5) O/p → 10 20 30 40 queue → {50}
6) O/p → 10 20 30 40 50 queue → {60}
7) O/p → 10 20 30 40 50 60 queue → { }

Now queue is empty & hence stop traversing

Code

```
void levelOrderTraversal (Node * root){
        //Empty tree
```

```
if (root == NULL)
        return ;
queue <Node *> q ;
//Push root node in the queue
q.push(root); * (1st modification)
//Run loop until queue does not become empty
while (! q.empty()) {
    //Fetch front element
    Node * temp = q.front();
    q.pop(); * (2nd modification)
    cout << temp -> data << " ";
    //Left child exist or not
    if (temp -> left) {
        q.push (temp -> left); //Push left child
    }
    //Right child exist or not
    if (temp -> right) {
        q.push (temp -> right); //Push right child
    }
}
}
```

Here the output is getting printed in a single line but we want the o/p in the tree format. After printing the level we have to print endl.
This can be done by doing the marking by pushing NULL which indicates level completed. In this case we have to do endl & then do the marking for the next level.
1st modification
q.push (NULL); // To do the marking

2nd modification

```
if (temp == NULL) {
        //go to next line
        cout <<endl;
        // Marking for next level
        if (!q.empty()) {
            q.push(NULL);
        }
}
else {
    cout << temp->data << " ";
    ; } Same push left & right child if
        exists
}
```