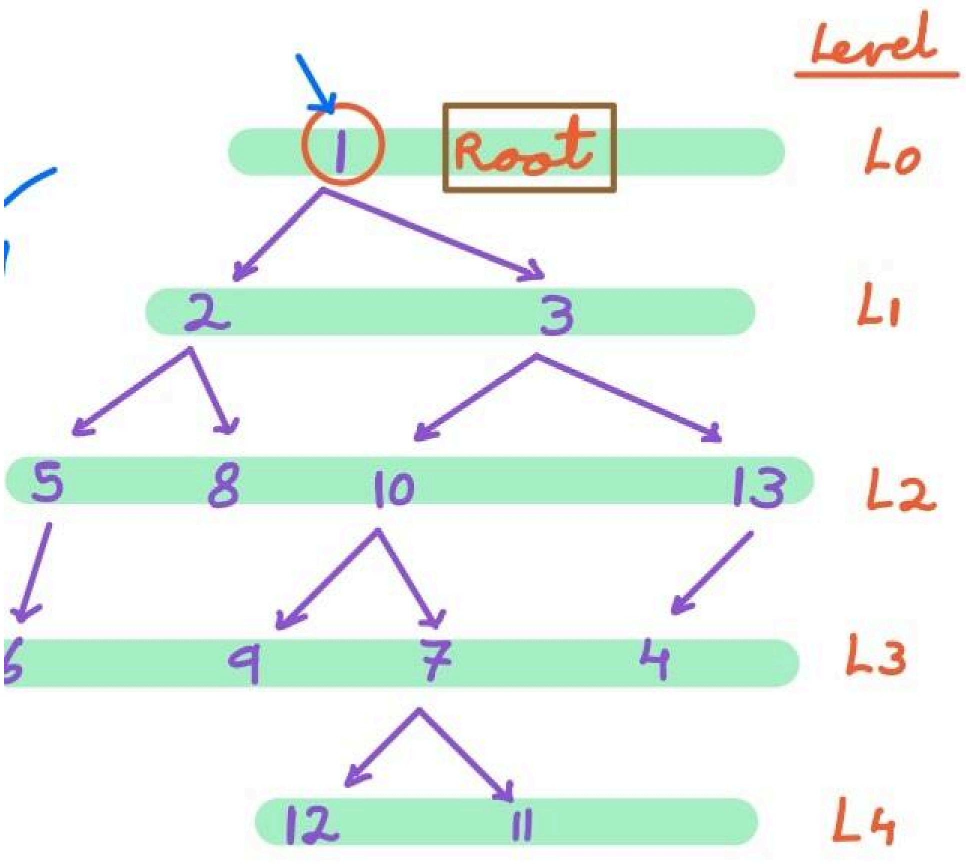
Advanced DSA : Trees 2: Views & Types

Level Order Traversal

Input: 1, 2, 3, 5, 8, 10, 13, 6, 9, 7, 4, 12, 11.

The diagram for the following nodes will be:



```
1
2  3
5  8  10  13
6  9  7  4
12  11
```

Question

Will the last level node always be a leaf node?

Choices

- [✓ YES]
- ☐ NO
- ☐ Cant say

Explanation

Yes, in the context of a binary tree's right view, the last level node will always be a leaf node. This is because the right view of a binary tree focuses on the rightmost nodes at each level as seen from a top-down view.

Question

Which traversal is best to print the nodes from top to bottom?

Choices

- [✓ Level order traversal]
- ☐ Pre order
- ☐ post order

Explanation:

When you want to print nodes from top to bottom, the level-order traversal, also known as Breadth-First Search (BFS), is the best choice. Level-order traversal ensures that nodes at the same level are processed before moving to the next level. This results in a top-to-bottom exploration of the tree.

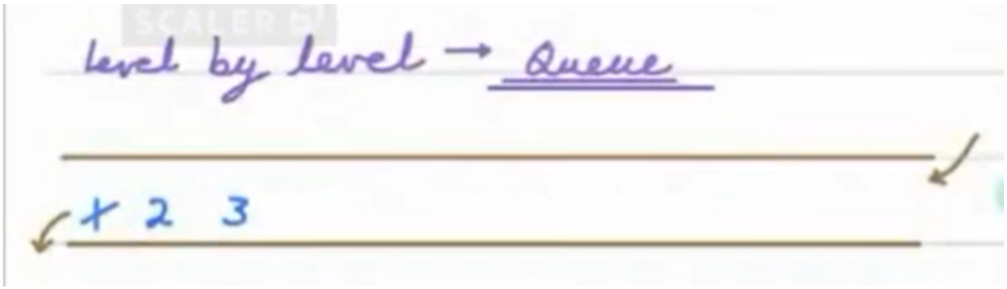
Level order traversal Observations

Observations:

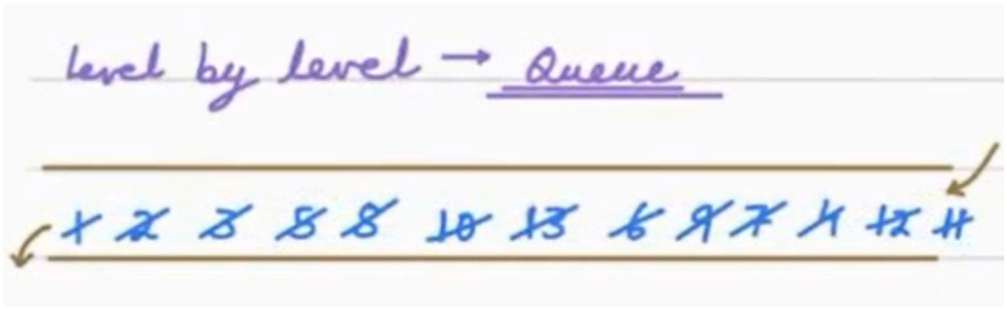
- Level order traversal visits nodes level by level, starting from the root.
- It uses a queue to keep track of the nodes to be processed.

- Nodes at the same level are processed before moving on to the next level.
- This traversal ensures that nodes at higher levels are visited before nodes at lower levels.

Since this will be done level by level hence we will be requiring a queue data structure to solve this problem:



After the whole process the queue data strucutre will look somewhat like this:



Like this(in theabove example) it will be done for all of the nodes.
Let us see the pseudocde to solve the problem in printing in one line only:

Pseudocode:

```
q.enqueue(root) {
    while (!q.eempty()) {
        x = q.dequeue()
        print(x.data)
        if (x.left != null) q.enqueue(x.left)
        if (x.right != null) q.enqueue(x.right)
    }
}
```

Each level will be printed in seperate line:

```
1
2  3
5  8  10  13
6  9  7  4
12  11
```

Observations:

- Level order traversal prints nodes at the same depth before moving to the next level, ensuring that nodes on the same level are printed on separate lines.

Approach:

1. Start with the root node and enqueue it.
2. Initialize last as the root.
3. While the queue is not empty:
    - Dequeue a node, print its data.
    - Enqueue its children (if any).
    - If the dequeued node is the same as last, print a newline and update last.

Dry-Run:

```
      1
     / \
    2   3
   / \
  4   5
```

- Enqueue root node 1 and initialize last as 1.
- Dequeue 1 and print it. Enqueue its children 2 and 3.
- Dequeue 2 and print it. Enqueue its children 4 and 5.
- Dequeue 3 and print it. Since 3 is the last node in the current level, print a newline.
- Dequeue 4 and print it. There are no children to enqueue for 4.
- Dequeue 5 and print it. There are no children to enqueue for 5.

Final Output:

```
1
2
3
4
5
```

Let us see the pseudocode to solve the problem in printing in **seperate** line only:

Pseudocode:

```
q.enqueue(root) {
    last = root;
    while (!q.empty()) {
        x.dequeue()
        print(x.data)
        if (x.left != null) q.enqueue(x.left)
        if (x.right != null) q.enqueue(x.right)
        if (x == last && !q.empty()) {
            print("\n");
            last = q.rear();
        }
    }
}
```

Complexity

Time Complexity: O(N)
Space Complexity: O(N)

Problem 2 Right and left view

Example:

Let us see an example below:

```
    1
   / \
  2   3
 / \   \
4   5   6
         \
          7
```

The right view of this tree would be [1, 3, 6, 7] when viewed from the right side.

To solve this we need to print last node of every level.

Question

Print right view of the given binary tree,

```
  1
 / \
2   3
 \   \
  5   6
     / \
    8   7
   / \
  9   10
```

Choices

- [ ] [1, 3, 6, 7]
- [ ] [1, 3, 6, 8, 9]
- [ ] [1, 3, 6, 7, 8, 9, 10]
- [✓] [1, 3, 6, 7, 10]
- [ ] [1, 2, 5]

Right view Observations

Observations/Idea

- The idea behind obtaining the right view of a binary tree is to perform a level-order traversal, and for each level, identify and print the rightmost node. This process ensures that we capture the rightmost nodes at each level, giving us the right view of the binary tree. We can obtain the right-view of the tree using a breadth-first level-order traversal with a queue and a loop.

Approach:

1. Initialize an empty queue for level order traversal and enqueue the root node.
2. While the queue is not empty, do the following:
   - Get the number of nodes at the current level (levelSize) by checking the queue's size.
   - Iterate through the nodes at the current level.
   - If the current node is the rightmost node at the current level, print its value.
   - Enqueue the left and right children of the current node if they exist.

- Repeat this process until the queue is empty.

Let us see the pseudocode to solve the problem:

Pseudocode:

```
q.enqueue(root)
last = root;
while (!q.empty(1)) {
    x = q.dequeue()
    if (x.left != null) q.enqueue(x.left)
    if (x.right != null) q.enqueue(x.right)
    if (x == last) {
        print(x.data)
        if (!q.empty()) {
            print("\n");
            last = q.rear();
        }

    }
}
```

Complexity

Time Complexity: O(N)
Space Complexity: O(M)

Vertical Order traversal

Examples:
Consider the following binary tree:

```
    1
   / \
  2   3
 / \ / \
4  5 6  7
   / \
  8   9
```

Vertical order traversal of this tree would yield the following output:

```
Vertical Line 1: 4
Vertical Line 2: 2, 8
Vertical Line 3: 1, 5, 6
Vertical Line 4: 3, 9
Vertical Line 5: 7
```

We need to print the vertical lines from top to bottom.

Question

Consider the following binary tree:

```
          1
         / \
        2   3
         \   \
          5   6
             / \
            8   7
           / \
          9   10
```

Pick the vertical order traversal of the given Binary Tree.

Choices

- [✓] [2, 1, 5, 9, 3, 8, 6, 10, 7]
- [ ] [1, 2, 5, 3, 6, 8 ,9, 10, 7]
- [ ] [1, 2, 3, 5, 6, 8, 7, 9, 10]
- [ ] [1, 5, 2, 3, 6, 10, 8, 7, 9]

Explanation:

Vertical order traversal of this tree would yield the following output:

```
Vertical Line 1: 2
Vertical Line 2: 1, 5, 9
Vertical Line 3: 3, 8
Vertical Line 4: 6, 10
Vertical Line 5: 7
```

Vertical Order traversal Observations

Observation:

- Vertical order traversal of a binary tree prints nodes column by column, with nodes in the same column printed together.

:::warning
Please take some time to think about the solution approach on your own before reading further.....
:::

These are the steps to Print vertical order traversal:

Approach:

- Assign horizontal distances to nodes (root gets distance 0, left decreases by 1, right increases by 1).
- Create a map/hash table where keys are distances and values are lists of node values.
- Update the map while traversing: append node values to corresponding distance lists.
- After traversal, print the values from the map in ascending order of distances.

Vertical Order traversal Pseudocode

Pseudocode

Let us see the pseudocode to solve this:

```
procedure levelOrderTraversal(root)
    if root is null
        return

    Create a queue
    Enqueue root

    while queue is not empty
        currentNode = dequeue a node from the queue
        print currentNode's value

        if currentNode has left child
            enqueue left child
        end if

        if currentNode has right child
            enqueue right child
        end if
    end while
end procedure
```

Problem 4 Top View

Example:

Consider the following binary tree:

```
    1
   / \
  2   3
 / \ / \
4  5 6  7
```

The top view of this tree would be [4, 2, 1, 3, 7].

Question

Consider the following binary tree:

```
      1
     / \
    2   3
     \   \
      5   6
         / \
        8   9
```

What is the top view of the given binary tree.

Choices

☐ [5, 2, 1, 3, 6, 9]
☐ [8, 5, 2, 1, 3, 6, 9]
☐ [2, 1, 5, 3, 8, 6, 9]
[✓ [2, 1, 3, 6, 9]]

Explanation:

The Top view of the Given Binary tree is [2, 1, 3, 6, 9].

Top View Observations

Observations:

- Assign Horizontal Distances: Nodes are assigned horizontal distances, with the root at distance 0, left children decreasing by 1, and right children increasing by 1. This helps identify the nodes in the top view efficiently.

Approach:

For this we need to follow these steps:

- Traverse the binary tree.
- Maintain a map of horizontal distances and corresponding nodes.
- Only store the first node encountered at each unique distance.
- Print the stored nodes in ascending order of distances to get the top view.

Pseudocode:

```
procedure topView(root)
    if root is null
        return

    Create an empty map

    Queue: enqueue (root, horizontal distance 0)

    while queue is not empty
        (currentNode, currentDistance) = dequeue a node

        if currentDistance is not in the map
            add currentDistance and currentNode's value to map

        enqueue (currentNode's left child, currentDistance - 1) if left child exists
        enqueue (currentNode's right child, currentDistance + 1) if right child exists

    Print values in map sorted by keys
end procedure
```

Complexity

Time Complexity: O(N)

Space Complexity: O(W)

Types of binary tree

1. **Proper Binary Tree (Strict Binary Tree):**
   Every node has either 0 or 2 children (never 1 child).
   Diagram:

```
    A
  /   \
 B     C
/ \   / \
```

2. **Complete Binary Tree:**
   All levels are filled except possibly the last level, which is filled left to right.
   Diagram:

```
    A
  /   \
 B     C
/ \   /
```

3. **Perfect Binary Tree:**
   All internal nodes have exactly two children, and all leaf nodes are at the same level.
   Diagram:

```
    A
  /   \
 B     C
/ \   / \
```

Question

Perfect Binary Trees are also:

Choices

- [ ] Proper binary tree
- [ ] Complete binary tree
- [✓ both]
- [ ] none

Explanation:

A perfect binary tree is a specialized case of both a proper binary tree and a complete binary tree, where all internal nodes have two children, all leaf nodes are at the same level, and all levels are completely filled.

Problem 5 : Check height balanced tree

Definition

For all nodes if(`height_ofleftchild-height_ofrightchild`) <= 1

Example:

```
        1
       / \
      2   3
     / \
    4   5
   /
  6
```

This tree is not height-balanced because the left subtree of node 2 has a height of 3, while the right subtree of node 2 has a height of 0, and the difference is greater than 1.

:::warning
Please take some time to think about the brute force approach on your own before reading further.....
:::

Brute Force

Approach

- For each node in the binary tree, calculate the height of its left and right subtrees.
- Check if the absolute difference between the heights of the left and right subtrees for each node is less than or equal to 1.
- If step 2 is true for all nodes in the tree, the tree is height-balanced.

Pseudocode:

```
// Helper function to calculate the height of a tree
function calculateHeight(root):
    if root is null:
        return -1
    return 1 + max(calculateHeight(root.left), calculateHeight(root.right))

function isHeightBalanced(node):
    if node is null:
        return true  // An empty tree is height-balanced

    // Check if the current node's subtrees are height-balanced
    leftHeight = calculateHeight(node.left)
    rightHeight = calculateHeight(node.right)

    // Check if the current node is height-balanced
    if abs(leftHeight - rightHeight) > 1:
        return false

    // Recursively check the left and right subtrees
    return isHeightBalanced(node.left) && isHeightBalanced(node.right)

// Example usage:
root = buildTree()  // Build your binary tree
result = isHeightBalanced(root)
```

> NOTE: For a null node: **height = -1**

Complexity

Time Complexity: $O(N^2)$
Space Complexity: O(N)

Question

Which traversal is best to use when finding the height of the tree?

Choices

- ☐ Level order

- [ ] Inorder
- [x] postorder
- [ ] preorder

Explanation:

Postorder traversal works best for calculating the height of a tree because it considers the height of subtrees before calculating the height of parent nodes, which mirrors the hierarchical nature of tree height calculation.

Check height balanced tree Optimised Approach

Observation/Idea:

- To solve the problem of determining whether a binary tree is height-balanced we can consider using a recursive approach where you calculate the height of left and right subtrees and check their balance condition at each step. Keep track of a boolean flag to indicate whether the tree is still balanced.

Approach:

- We use a helper function height(root) to calculate the height of each subtree starting from the root.
- In the height function:
- If the root is null (i.e., an empty subtree), we return -1 to indicate a height of -1.
- We recursively calculate the heights of the left and right subtrees using the height function.
- We check if the absolute difference between the left and right subtree heights is greater than 1. If it is, we set the ishb flag to false, indicating that the tree is not height-balanced.
- We return the maximum of the left and right subtree heights plus 1, which represents the height of the current subtree.
- The ishb flag is initially set to true, and we start the height calculation from the root of the tree.
- If, at any point, the ishb flag becomes false, we know that the tree is not height-balanced, and we can stop further calculations.
- After the traversal is complete, if the ishb flag is still true, the tree is height-balanced.

Example:

```
      1
     / \
    2   3
   / \
  4   5
```

This tree is height-balanced because the height of the left and right subtrees of every node differs by at most 1.

Pseudocode

```
int height(root, ishb) {
    if (root == null) return -1;
    l = height(root.left)
    r = height(root.right)
    if (abs(l - r) > 1) ishb = false;
    return max(l, r) + 1
}
```

Complexity

Time Complexity: O(N)

Space Complexity: O(log N)