

## DAY 6

### 1) Binary Tree Inorder Traversal

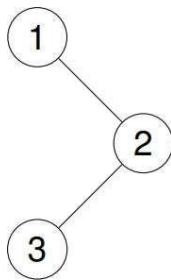
Given the root of a binary tree, return the inorder traversal of its nodes' values.

**Example 1:**

**Input:** root = [1,null,2,3]

**Output:** [1,3,2]

**Explanation:**



**SOLUTION:**

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};
void inorderRecursive(TreeNode* root, vector<int>& result) {
```

```

    if (root == NULL) return;
    inorderRecursive(root->left, result);
    result.push_back(root->val);
    inorderRecursive(root->right, result);
}

vector<int> inorderIterative(TreeNode* root)
{
    vector<int> result;
    stack<TreeNode*> stack;
    TreeNode* current = root;

    while (current != NULL || !stack.empty())
    {
        while (current != NULL) {
            stack.push(current);
            current = current->left;
        }
        current = stack.top();
        stack.pop();
        result.push_back(current->val);
        current = current->right;
    }

    return result;
}

void printVector(const vector<int>& vec)
{
    for (int val : vec) {
        cout << val << " ";
    }
    cout << endl;
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->right = new TreeNode(2);

```

```

root->right->left = new TreeNode(3);
vector<int> resultRecursive;
inorderRecursive(root, resultRecursive);
cout << "Recursive Inorder Traversal: ";
printVector(resultRecursive);
vector<int> resultIterative = inorderIterative(root);
cout << "Iterative Inorder Traversal: ";
printVector(resultIterative);

return 0;
}

```

## OUTPUT:

```

Recursive Inorder Traversal: 1 3 2
Iterative Inorder Traversal: 1 3 2

=== Code Execution Successful ===

```

## 2. 2) Count Complete Tree Nodes

Given the root of a complete binary tree, return the number of the nodes in the tree. According to Wikipedia, every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. It can have between  $1$  and  $2^h$  nodes inclusive at the last level  $h$ .

Design an algorithm that runs in less than  $O(n)$  time complexity.

### Example 1:

**Input:** root = [1,2,3,4,5,6]

**Output:** 6

## SOLUTION:

```

#include <iostream>
#include <cmath>
using namespace std;

```

```

struct TreeNode
{
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

```

```

int computeHeight(TreeNode* root)
{
    int height = 0;
    while (root)
    {
        height++;
        root = root->left;
    }
    return height;
}

```

```

int countNodes(TreeNode* root)
{
    if (!root) return 0;
    int leftHeight = computeHeight(root->left);
    int rightHeight = computeHeight(root->right);
    if (leftHeight == rightHeight) {
        return (1 << leftHeight) + countNodes(root->right);
    } else {
        return (1 << rightHeight) + countNodes(root->left); // 2^rightHeight +
nodes in the left subtree
    }
}

```

```

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
}

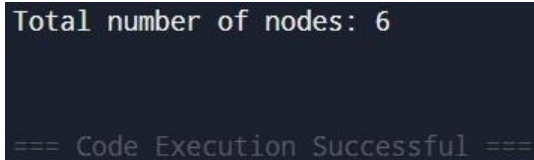
```

```

root->right->left = new TreeNode(6);
int totalNodes = countNodes(root);
cout << "Total number of nodes: " << totalNodes << endl;
return 0;
}

```

OUTPUT:



```

Total number of nodes: 6

=== Code Execution Successful ===

```

### 3) Binary Tree - Find Maximum Depth

A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

**Example 1:**

**Input:** [3,9,20,null,null,15,7]

**Output:** 3

**SOLUTION:**

```

#include <iostream>
#include <algorithm>
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

int maxDepth(TreeNode* root)
{
    if (!root) return 0;
    int leftDepth = maxDepth(root->left);
    int rightDepth = maxDepth(root->right);
    return max(leftDepth, rightDepth) + 1;
}

```

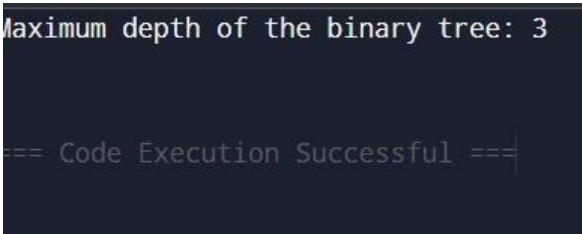
```

}
int main() {
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(9);
    root->right = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);
    int depth = maxDepth(root);
    cout << "Maximum depth of the binary tree: " << depth << endl;
    delete root->right->right;
    delete root->right->left;
    delete root->right;
    delete root->left;
    delete root;

    return 0;
}

```

## OUTPUT:



```

Maximum depth of the binary tree: 3
=== Code Execution Successful ===

```

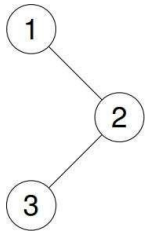
## 4) Binary Tree Preorder Traversal

Given the root of a binary tree, return the preorder traversal of its nodes' values.

**Example 1:**

**Input:** root = [1,null,2,3]

**Output:** [1,2,3]

**Explanation:****SOLUTION:**

```
#include <iostream>
#include <vector>
#include <stack>
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
};

class Solution
{ public:
    std::vector<int> preorderTraversal(TreeNode* root)
    { std::vector<int> result;
      if (!root) return result;
      std::stack<TreeNode*> stack;
      stack.push(root);

      while (!stack.empty())
      { TreeNode* node = stack.top();
        stack.pop();
```

```

        result.push_back(node->val);
        if (node->right) stack.push(node->right);
        if (node->left) stack.push(node->left);
    }
    return result;
}
};

TreeNode* createTree() {
    TreeNode* root = new TreeNode(1);
    root->right = new TreeNode(2);
    root->right->left = new TreeNode(3);
    return root;
}

int main()
{
    Solution
    solution;
    TreeNode* root = createTree();
    std::vector<int> result = solution.preorderTraversal(root);
    std::cout << "Preorder Traversal: ";
    for (int val : result)
    {
        std::cout << val << "
        ";
    }
    std::cout << std::endl;

    return 0;
}

```

## OUTPUT:

```
Preorder Traversal: 1 2 3
```

```
=== Code Execution Successful ===
```



### 5) Binary Tree - Sum of All Nodes

Given the root of a binary tree, you need to find the sum of all the node values in the binary tree.

#### Example 1:

**Input:** root = [1, 2, 3, 4, 5, null, 6]

**Output:** 21

**Explanation:** The sum of all nodes is  $1 + 2 + 3 + 4 + 5 + 6 = 21$ .

#### SOLUTION:

```
#include <iostream>
#include <vector>
#include <stack>
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
};

class Solution
{ public:
    int sumOfNodes(TreeNode* root)
    { if (!root) return 0;
      return root->val + sumOfNodes(root->left) + sumOfNodes(root->right);
    }
};

TreeNode* createTree() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
```

```

    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->right = new TreeNode(6);
    return root;
}

int main()
{
    Solution
    solution;
    TreeNode* root = createTree();
    int sum = solution.sumOfNodes(root);
    std::cout << "Sum of all nodes: " << sum << std::endl;
    return 0;
}

```

## OUTPUT:

```

Sum of all nodes: 21

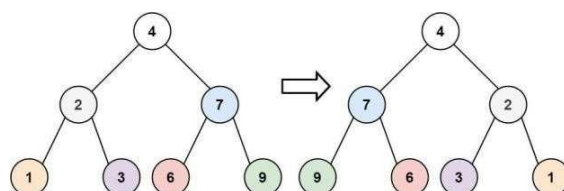
=== Code Execution Successful ===

```

## 6) Invert Binary Tree

Given the root of a binary tree, invert the tree, and return its root.

### Example 1:



**Input:** root = [4,2,7,1,3,6,9]

**Output:** [4,7,2,9,6,3,1]

**SOLUTION:**

```
#include <iostream>
#include <vector>
#include <stack>
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
};
class Solution
{ public:
    int sumOfNodes(TreeNode* root)
    { if (!root) return 0;
      return root->val + sumOfNodes(root->left) + sumOfNodes(root->right);
    }
    TreeNode* invertTree(TreeNode* root)
    { if (!root) return nullptr;
      TreeNode* temp = root->left;
      root->left = root->right;
      root->right = temp;
      invertTree(root->left);
      invertTree(root->right);
      return root;
    }
};
TreeNode* createTree() {
    TreeNode* root = new TreeNode(4);
    root->left = new TreeNode(2);
```

```

    root->right = new TreeNode(7);
    root->left->left = new TreeNode(1);
    root->left->right = new TreeNode(3);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(9);
    return root;
}

void printTree(TreeNode* root)
{
    if (!root) return;
    printTree(root->left);
    std::cout << root->val << " ";
    printTree(root->right);
}

int main()
{
    Solution
    solution;
    TreeNode* root = createTree();
    std::cout << "Original tree (in-order): ";
    printTree(root);
    std::cout << std::endl;
    root = solution.invertTree(root);
    std::cout << "Inverted tree (in-order): ";
    printTree(root);
    std::cout << std::endl;
    return 0;
}

```

## OUTPUT:

```

Original tree (in-order): 1 2 3 4 6 7 9
Inverted tree (in-order): 9 7 6 4 3 2 1

```

```

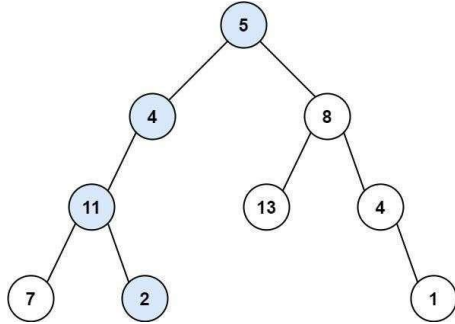
=== Code Execution Successful ===

```

## 7) Path Sum

Given a binary tree and a sum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum. Return false if no such path can be found.

### Example 1:



### SOLUTION:

```
#include <iostream>
#include <vector>
#include <stack>
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
};
class Solution
{ public:
    int sumOfNodes(TreeNode* root)
    { if (!root) return 0;
      return root->val + sumOfNodes(root->left) + sumOfNodes(root->right);
    }
};
```

```

TreeNode* invertTree(TreeNode* root)
{
    if (!root) return nullptr;
    TreeNode* temp = root->left;
    root->left = root->right;
    root->right = temp;
    invertTree(root->left);
    invertTree(root->right);
    return root;
}

bool hasPathSum(TreeNode* root, int targetSum)
{
    if (!root) return false;
    if (!root->left && !root->right) return root->val == targetSum;
    int remainingSum = targetSum - root->val;
    return hasPathSum(root->left, remainingSum) || hasPathSum(root->right,
remainingSum);
}

};

TreeNode* createTree() {
    TreeNode* root = new TreeNode(5);
    root->left = new TreeNode(4);
    root->right = new TreeNode(8);
    root->left->left = new TreeNode(11);
    root->left->left->left = new TreeNode(7);
    root->left->left->right = new TreeNode(2);
    root->right->left = new TreeNode(13);
    root->right->right = new TreeNode(4);
    root->right->right->right = new TreeNode(1);
    return root;
}

void printTree(TreeNode* root)
{
    if (!root) return;
    printTree(root->left);

```

```

        std::cout << root->val << " ";
        printTree(root->right);
    }
int main()
{
    Solution
    solution;
    TreeNode* root = createTree();
    std::cout << "Original tree (in-order): ";
    printTree(root);
    std::cout << std::endl;

    int targetSum = 22;
    bool result = solution.hasPathSum(root, targetSum);
    std::cout << "Has path sum " << targetSum << ": " << (result ? "true" :
    "false") << std::endl;

    return 0;
}

```

## OUTPUT:

```

Original tree (in-order): 7 11 2 4 5 13 8 4 1
Has path sum 22: true

```

```

=== Code Execution Successful ===

```

### 1. 8) Populating Next Right Pointers in Each Node

Given a binary tree

```

struct Node {
    int val;
    Node *left;

```

```

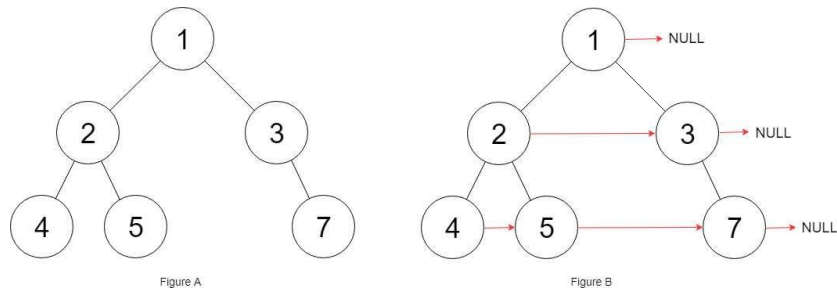
Node *right;
Node *next;
}

```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

Example 1:



Input: root = [1,2,3,4,5,null,7]

Output: [1,#,2,3,#,4,5,7,#]

Explanation: Given the above binary tree (Figure A), your function should populate each next pointer to point to its next right node, just like in Figure B. The serialized output is in level order as connected by the next pointers, with '#' signifying the end of each level.

## SOLUTION:

```

#include <iostream>
using namespace std;
struct Node {
    int val;
    Node *left;
    Node *right;
    Node *next;
    Node(int x) : val(x), left(NULL), right(NULL), next(NULL) {}
};

```



```

class Solution
{ public:
    void connect(Node* root)
    { if (root == NULL) {
        return;
    }
    Node* current = root;
    while (current != NULL) {
        Node* levelStart = current;
        Node* prev = NULL;
        while (levelStart != NULL)
            { if (levelStart->left) {
                if (prev) {
                    prev->next = levelStart->left;
                }
                prev = levelStart->left;
            }
            if (levelStart->right)
                { if (prev) {
                    prev->next = levelStart->right;
                }
                prev = levelStart->right;
            }
            levelStart = levelStart->next;
        }
        current = current->left;
    }
};

void printLevels(Node* root)
    { while (root) {

```

```

Node* current = root;
while (current) {
    cout << current->val << " ";
    current = current->next;
}
cout << "# ";
root = root->left;
}
cout << endl;
}
int main() {
    // Example: root = [1,2,3,4,5,null,7]
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->right = new Node(7);
    Solution solution;
    solution.connect(root);
    printLevels(root);
    return 0;
}

```

## OUTPUT:

```

1 # 2 3 # 4 5 7 #

=== Code Execution Successful ===

```

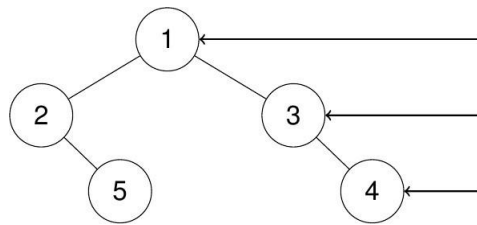
## 2. 9) Binary Tree Right Side View

Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

### Example 1:

**Input:** root = [1,2,3,null,5,null,4]

**Output:** [1,3,4]



**Explanation:**

### SOLUTION:

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
struct Node {
    int val;
    Node *left;
    Node *right;
    Node(int x) : val(x), left(NULL), right(NULL) {}
};
class Solution
{ public:
    vector<int> rightSideView(Node* root)
    { vector<int> result;
      if (root == NULL)
        { return result;

```

```

    }
    queue<Node*> q;
    q.push(root);
    while (!q.empty()) {
        int levelSize = q.size();
        for (int i = 0; i < levelSize; ++i)
            { Node*    currentNode    =
              q.front(); q.pop();
              if (i    ==    levelSize    -    1)
                  { result.push_back(currentNode->val);
                    }
              if (currentNode->left)
                  { q.push(currentNode-
                        >left);
                    }
              if (currentNode->right)
                  { q.push(currentNode-
                        >right);
                    }
              }
        }
    }

    return result;
}
};

void printRightSideView(const vector<int>& rightSide)
    { for (int val : rightSide) {
      cout << val << " ";
      }
    cout << endl;
}

int main() {
    Node* root = new Node(1);

```

```

root->left = new Node(2);
root->right = new Node(3);
root->left->right = new Node(5);
root->right->right = new Node(4);
Solution solution;
vector<int> rightSide = solution.rightSideView(root);
printRightSideView(rightSide);
return 0;
}

```

### OUTPUT:

```

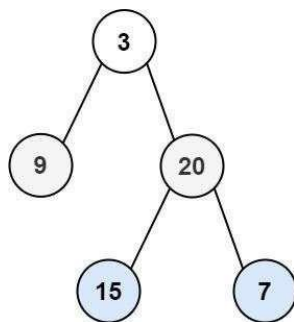
1 3 4
=== Code Execution Successful ===

```

### 3. 10) Binary Tree Zigzag Level Order Traversal

Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).

#### Example 1:



**Input:** root = [3,9,20,null,null,15,7]

**Output:** [[3],[20,9],[15,7]]

**SOLUTION:**

```
#include <iostream>
#include <queue>
#include <vector>
#include <deque>
using namespace std;
struct Node {
    int val;
    Node *left;
    Node *right;
    Node(int x) : val(x), left(NULL), right(NULL) {}
};
class Solution
{ public:
    vector<vector<int>> zigzagLevelOrder(Node* root)
    { vector<vector<int>> result;
      if (root == NULL)
          { return result;
          }
      queue<Node*> q;
      q.push(root);
      bool leftToRight = true;
      while (!q.empty()) {
          int levelSize = q.size();
          deque<int> currentLevel;
          for (int i = 0; i < levelSize; ++i)
              { Node* currentNode =
                q.front(); q.pop();
                if (leftToRight)
                    { currentLevel.push_back(currentNode->val);
                    } else {
```

```

        currentLevel.push_front(currentNode->val);
    }
    if (currentNode->left)
        { q.push(currentNode-
        >left);
        }
    if (currentNode->right)
        { q.push(currentNode-
        >right);
        }
    }
    result.push_back(vector<int>(currentLevel.begin(),
currentLevel.end()));
    leftToRight = !leftToRight;
}
return result;
}
};

```

```

void printZigzagOrder(const vector<vector<int>>& zigzagOrder)
{ for (const auto& level : zigzagOrder) {
    for (int val : level)
        { cout << val << "
        ";
        }
    cout << endl;
}
}

```

```

int main() {
    // Example: root = [3,9,20,null,null,15,7]
    Node* root = new Node(3);
    root->left = new Node(9);
    root->right = new Node(20);
}

```

```
root->right->left = new Node(15);
```



```
root->right->right = new Node(7);  
Solution solution;  
vector<vector<int>> zigzagOrder = solution.zigzagLevelOrder(root);  
printZigzagOrder(zigzagOrder);  
return 0;  
}
```

## OUTPUT:

```
3  
20 9  
15 7  
  
=== Code Execution Successful ===
```