DAY 6

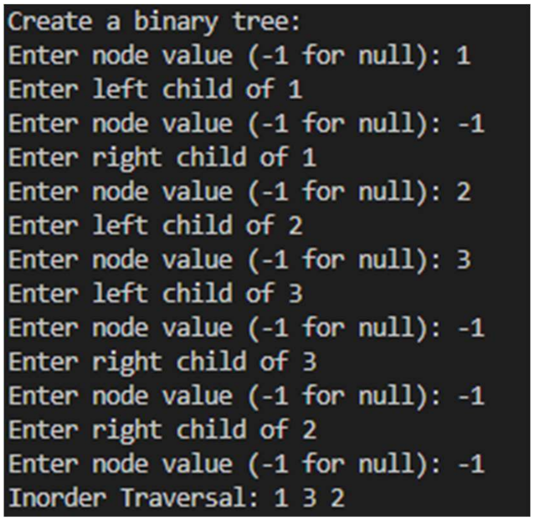
Student Name: Abhishek **UID: 22BCS15315**
Branch: BE-CSE **Section/Group: 620 - A**
Date of Performance:26/12/24

Problem 1

1. **Aim: Binary Tree Inorder Traversal**
2. **Code:**

```
#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 inorderTraversal(TreeNode* root, vector<int>& result) {
    if (root == NULL) return;
    inorderTraversal(root->left, result);
    result.push_back(root->val);
    inorderTraversal(root->right, result);
}
TreeNode* createTree() {
    int val;
    cout << "Enter node value (-1 for null): ";
    cin >> val;
    if (val == -1) {
        return NULL;
    }
    TreeNode* node = new TreeNode(val);
    cout << "Enter left child of " << val << endl;
    node->left = createTree();
    cout << "Enter right child of " << val << endl;
    node->right = createTree();
    return node;
}
int main() {
    cout << "Create a binary tree:" << endl;
    TreeNode* root = createTree();
    vector<int> result;
    inorderTraversal(root, result);
    cout << "Inorder Traversal: ";
    for (int val : result) {
        cout << val << " ";
    }
    cout << endl;
    return 0;
}
```

3. Output:



```
Create a binary tree:
Enter node value (-1 for null): 1
Enter left child of 1
Enter node value (-1 for null): -1
Enter right child of 1
Enter node value (-1 for null): 2
Enter left child of 2
Enter node value (-1 for null): 3
Enter left child of 3
Enter node value (-1 for null): -1
Enter right child of 3
Enter node value (-1 for null): -1
Enter right child of 2
Enter node value (-1 for null): -1
Inorder Traversal: 1 3 2
```

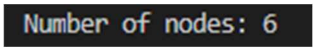
Problem 2

1. Aim: Count Complete Tree Nodes

2. Code:

```
#include <iostream>
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};
class Solution {
public:
    int countNodes(TreeNode* root) {
        if (!root) return 0;
        int height = getHeight(root);
        if (height == 0) return 1;
        int left = 0, right = (1 << height) - 1;
        while (left < right) {
            int mid = left + (right - left) / 2;
            if (exists(mid, height, root)) {
                left = mid + 1;
            } else {
                right = mid;
            }
        }
        return (1 << height) - 1 + left;
    }
private:
    int getHeight(TreeNode* node) {
        int height = 0;
        while (node) {
            height++;
            node = node->left;
        }
        return height - 1;
    }
    bool exists(int index, int height, TreeNode* node) {
        int left = 0, right = (1 << height) - 1;
        for (int i = 0; i < height; i++) {
            int mid = left + (right - left) / 2;
            if (index <= mid) {
                node = node->left;
                right = mid;
            } else {
                node = node->right;
                left = mid + 1;
            }
        }
        return node != nullptr;
    }
};
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);
    Solution solution;
    std::cout << "Number of nodes: " << solution.countNodes(root) << std::endl;
    return 0;
}
```

3. Output:



```
Number of nodes: 6
```

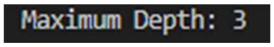
Problem 3

1. Aim: .Binary Tree - Find Maximum Depth

2. Code:

```
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};
int maxDepth(TreeNode* root) {
    if (root == NULL) {
        return 0;
    }
    return 1 + max(maxDepth(root->left), maxDepth(root->right));
}
TreeNode* createTree(const vector<int>& nodes, int index) {
    if (index >= nodes.size() || nodes[index] == NULL) {
        return NULL;
    }
    TreeNode* root = new TreeNode(nodes[index]);
    root->left = createTree(nodes, 2 * index + 1);
    root->right = createTree(nodes, 2 * index + 2);
    return root;
}
int main() {
    vector<int> input = {3, 9, 20, NULL, NULL, 15, 7};
    TreeNode* root = createTree(input, 0);
    int depth = maxDepth(root);
    cout << "Maximum Depth: " << depth << endl;
    return 0;
}
```

3. Output:



```
Maximum Depth: 3
```

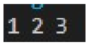
Problem 4

1. Aim: Binary Tree Preorder Traversal

2. Code:

```
include <vector>
#include <iostream>
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};
class Solution {
public:
    void preorderHelper(TreeNode* root, std::vector<int>& result) {
        if (root == nullptr) {
            return;
        }
        result.push_back(root->val); // Visit the root
        preorderHelper(root->left, result); // Traverse left subtree
        preorderHelper(root->right, result); // Traverse right subtree
    }
    std::vector<int> preorderTraversal(TreeNode* root) {
        std::vector<int> result;
        preorderHelper(root, result);
        return result;
    }
};
int main() {
    TreeNode* root = new TreeNode(1);
    root->right = new TreeNode(2);
    root->right->left = new TreeNode(3);
    Solution solution;
    std::vector<int> result = solution.preorderTraversal(root);
    for (int val : result) {
        std::cout << val << " ";
    }
    return 0;
}
```

3. Output:



```
1 2 3
```

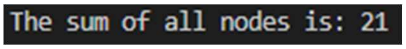
Problem 5

1. Aim: Binary Tree - Sum of All Nodes

2. Code:

```
#include <iostream>
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
int sumOfNodes(TreeNode* root) {
    if (root == nullptr) {
        return 0;
    }
    return root->val + sumOfNodes(root->left) + sumOfNodes(root->right);
}
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->right = new TreeNode(6);
    int totalSum = sumOfNodes(root);
    std::cout << "The sum of all nodes is: " << totalSum << std::endl;
    delete root->left->left; // 4
    delete root->left->right; // 5
    delete root->right->right; // 6
    delete root->left; // 2
    delete root->right; // 3
    delete root; // 1
    return 0;
}
```

3. Output:



```
The sum of all nodes is: 21
```

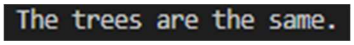
Problem 6

1. Aim: Same Tree

2. Code:

```
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
bool isSameTree(TreeNode* p, TreeNode* q) {
    if (p == nullptr && q == nullptr) {
        return true;
    }
    if (p == nullptr || q == nullptr || p->val != q->val) {
        return false;
    }
    return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
}
#include <iostream>
int main() {
    TreeNode* p = new TreeNode(1);
    p->left = new TreeNode(2);
    p->right = new TreeNode(3);
    TreeNode* q = new TreeNode(1);
    q->left = new TreeNode(2);
    q->right = new TreeNode(3);
    if (isSameTree(p, q)) {
        std::cout << "The trees are the same." << std::endl;
    } else {
        std::cout << "The trees are not the same." << std::endl;
    }
    return 0;
}
```

3. **Output:**



```
The trees are the same.
```

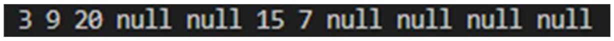
Problem 7

1. **Aim: Construct Binary Tree from Preorder and Inorder Traversal**
2. **Code:**

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <queue>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};
class Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        unordered_map<int, int> inorderIndexMap;
        for (int i = 0; i < inorder.size(); ++i) {
            inorderIndexMap[inorder[i]] = i;
        }
        return buildTreeHelper(preorder, 0, preorder.size() - 1,
                               inorderIndexMap, 0, inorder.size() - 1);
    }
private:
    TreeNode* buildTreeHelper(vector<int>& preorder, int preStart, int
preEnd,
                              unordered_map<int, int>& inorderIndexMap,
                              int inStart, int inEnd) {
        if (preStart > preEnd || inStart > inEnd) {
            return nullptr;
        }
        int rootValue = preorder[preStart];
        TreeNode* root = new TreeNode(rootValue);
        int rootIndex = inorderIndexMap[rootValue];
        int leftSize = rootIndex - inStart;
        root->left = buildTreeHelper(preorder, preStart + 1, preStart + leftSize,
                                     inorderIndexMap, inStart, rootIndex - 1);
        root->right = buildTreeHelper(preorder, preStart + leftSize + 1, preEnd,
                                      inorderIndexMap, rootIndex + 1, inEnd);

        return root;
    }
};
void printLevelOrder(TreeNode* root) {
    if (!root) return;
    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
        TreeNode* node = q.front();
        q.pop();
        if (node) {
            cout << node->val << " ";
            q.push(node->left);
            q.push(node->right);
        } else {
            cout << "null ";
        }
    }
    cout << endl;
}
int main() {
    Solution solution;
    vector<int> preorder = {3, 9, 20, 15, 7};
    vector<int> inorder = {9, 3, 15, 20, 7};
    TreeNode* root = solution.buildTree(preorder, inorder);
    printLevelOrder(root);
    return 0;
}
```

3. **Output:**



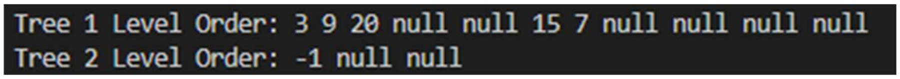
```
3 9 20 null null 15 7 null null null null
```

Problem 8

1. **Aim: Construct Binary Tree from Inorder and Postorder Traversal**
2. **Code:**

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
class Solution {
public:
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        unordered_map<int, int> inorderIndexMap;
        for (int i = 0; i < inorder.size(); ++i) {
            inorderIndexMap[inorder[i]] = i;
        }
    int postIndex = postorder.size() - 1;
    return constructTree(postorder, inorderIndexMap, postIndex, 0,
inorder.size() - 1);
  }
private:
  TreeNode* constructTree(vector<int>& postorder, unordered_map<int,
int>& inorderIndexMap,
              int& postIndex, int inStart, int inEnd) {
    if (inStart > inEnd) return nullptr;
    int rootValue = postorder[postIndex--];
    TreeNode* root = new TreeNode(rootValue);
    int inIndex = inorderIndexMap[rootValue];
    root->right = constructTree(postorder, inorderIndexMap, postIndex,
inIndex + 1, inEnd);
    root->left = constructTree(postorder, inorderIndexMap, postIndex,
inStart, inIndex - 1);
    return root;
  }
};
void printLevelOrder(TreeNode* root) {
  if (!root) return;
  vector<TreeNode*> queue = {root};
  while (!queue.empty()) {
    vector<TreeNode*> nextQueue;
    for (TreeNode* node : queue) {
      if (node) {
        cout << node->val << " ";
        nextQueue.push_back(node->left);
        nextQueue.push_back(node->right);
      } else {
        cout << "null ";
      }
    }
    queue = nextQueue;
  }
}
int main() {
  Solution solution;
  vector<int> inorder1 = {9, 3, 15, 20, 7};
  vector<int> postorder1 = {9, 15, 7, 20, 3};
  TreeNode* root1 = solution.buildTree(inorder1, postorder1);
  cout << "Tree 1 Level Order: ";
  printLevelOrder(root1);
  cout << endl;
  vector<int> inorder2 = {-1};
  vector<int> postorder2 = {-1};
  TreeNode* root2 = solution.buildTree(inorder2, postorder2);
  cout << "Tree 2 Level Order: ";
  printLevelOrder(root2);
  cout << endl;
  return 0;
}}
```

3. Output:



```
Tree 1 Level Order: 3 9 20 null null 15 7 null null null null
Tree 2 Level Order: -1 null null
```

Problem 9

1. Aim: Invert Binary Tree.

2. Code:

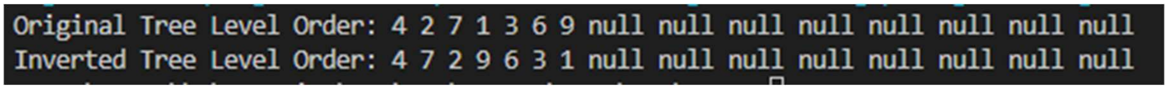
```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
```

```
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
class Solution {
public:
    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(const vector<int>& nodes) {
    if (nodes.empty() || nodes[0] == -1) return nullptr;
    TreeNode* root = new TreeNode(nodes[0]);
    queue<TreeNode*> q;
    q.push(root);
    int i = 1;
    while (i < nodes.size()) {
        TreeNode* current = q.front();
        q.pop();

        if (nodes[i] != -1) {
            current->left = new TreeNode(nodes[i]);
            q.push(current->left);
        }
        ++i;

        if (i < nodes.size() && nodes[i] != -1) {
            current->right = new TreeNode(nodes[i]);
            q.push(current->right);
        }
        ++i;
    }
    return root;
}
void printLevelOrder(TreeNode* root) {
    if (!root) return;
    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
        TreeNode* current = q.front();
        q.pop();
        if (current) {
            cout << current->val << " ";
            q.push(current->left);
            q.push(current->right);
        } else {
            cout << "null ";
        }
    }
    cout << endl;
}
int main() {
    Solution solution;
    vector<int> treeNodes = {4, 2, 7, 1, 3, 6, 9};
    TreeNode* root = createTree(treeNodes);
    cout << "Original Tree Level Order: ";
    printLevelOrder(root);
    TreeNode* invertedRoot = solution.invertTree(root);
    cout << "Inverted Tree Level Order: ";
    printLevelOrder(invertedRoot);
    return 0;
}
```

3. Output:



```
Original Tree Level Order: 4 2 7 1 3 6 9 null null null null null null null null
Inverted Tree Level Order: 4 7 2 9 6 3 1 null null null null null null null null
```

Problem 10

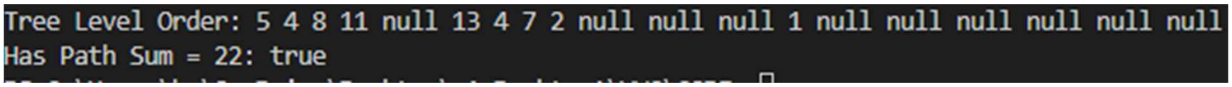
1. Aim: Path Sum

2. Code:

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
class Solution {
public:
    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(const vector<int>& nodes) {
    if (nodes.empty() || nodes[0] == -1) return nullptr;
    TreeNode* root = new TreeNode(nodes[0]);
    queue<TreeNode*> q;
    q.push(root);
    int i = 1;
    while (i < nodes.size()) {
        TreeNode* current = q.front();
        q.pop();

        if (nodes[i] != -1) {
            current->left = new TreeNode(nodes[i]);
            q.push(current->left);
        }
        ++i;
        if (i < nodes.size() && nodes[i] != -1) {
            current->right = new TreeNode(nodes[i]);
            q.push(current->right);
        }
        ++i;
    }
    return root;
}
void printLevelOrder(TreeNode* root) {
    if (!root) return;
    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
        TreeNode* current = q.front();
        q.pop();
        if (current) {
            cout << current->val << " ";
            q.push(current->left);
            q.push(current->right);
        } else {
            cout << "null ";
        }
    }
    cout << endl;
}
int main() {
    Solution solution;
    vector<int> treeNodes = {5, 4, 8, 11, -1, 13, 4, 7, 2, -1, -1, -1, 1};
    int targetSum = 22;
    TreeNode* root = createTree(treeNodes);
    cout << "Tree Level Order: ";
    printLevelOrder(root);
    bool result = solution.hasPathSum(root, targetSum);
    cout << "Has Path Sum = " << targetSum << ": " << (result ? "true" :
"false") << endl;
    return 0;
}
```

3. Output:



```
Tree Level Order: 5 4 8 11 null 13 4 7 2 null null null 1 null null null null null null
Has Path Sum = 22: true
```