WINTER DOMAIN CAMP

NAME: Vikram Kumar **UID:22BCS12220**

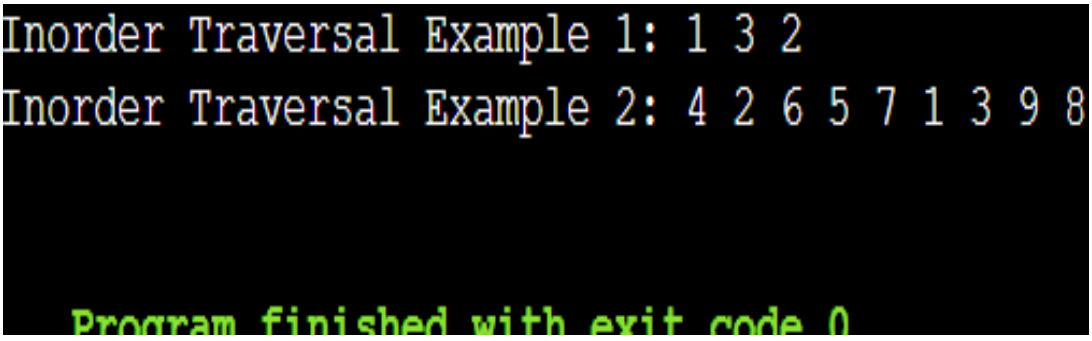
SECTION: KPIT-901-B **DATE: 27/12/2024**

Q1. Binary tree inorder traversal(Easy)

Sol.

```cpp
//binary tree inorder traversal(easy)
#include <iostream>
#include <vector>
using namespace std;
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) {} };
void inorderTraversalHelper(TreeNode* root, vector<int>& result) {
    if (root == nullptr) return;
    inorderTraversalHelper(root->left, result);
    result.push_back(root->val);
    inorderTraversalHelper(root->right, result); }
vector<int> inorderTraversal(TreeNode* root) {
    vector<int> result;
    inorderTraversalHelper(root, result);
    return result;}
void printVector(const vector<int>& vec) {
    for (int val : vec) {
        cout << val << " "; }
    cout << endl;}
  int main() {
    TreeNode* root1 = new TreeNode(1);
    root1->right = new TreeNode(2);
    root1->right->left = new TreeNode(3);
    vector<int> result1 = inorderTraversal(root1);
    cout << "Inorder Traversal Example 1: ";
    printVector(result1);
    TreeNode* root2 = new TreeNode(1);
    root2->left = new TreeNode(2, new TreeNode(4), new TreeNode(5, new TreeNode(6), new TreeNode(7)));
    root2->right = new TreeNode(3, nullptr, new TreeNode(8, new TreeNode(9), nullptr));
    vector<int> result2 = inorderTraversal(root2);
    cout << "Inorder Traversal Example 2: ";
    printVector(result2);
    return 0;}
```

OUTPUT:



Q2. Construct binary tree from preorder and inorder traversal(medium) Sol.

```cpp
//Construct binary tree form preorder and inorder traversal(medium.)
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;
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) {}
};
TreeNode* buildTreeHelper(const vector<int>& preorder, int preorderStart, int preorderEnd,
                          const vector<int>& inorder, int inorderStart, int inorderEnd,
                          unordered_map<int, int>& inorderIndexMap) {
    if (preorderStart > preorderEnd || inorderStart > inorderEnd) {
        return nullptr;
    }
    int rootVal = preorder[preorderStart];
    TreeNode* root = new TreeNode(rootVal);
    int rootIndex = inorderIndexMap[rootVal];
    int leftSubtreeSize = rootIndex - inorderStart;
    root->left = buildTreeHelper(preorder, preorderStart + 1, preorderStart + leftSubtreeSize,
                                 inorder, inorderStart, rootIndex - 1, inorderIndexMap);
    root->right = buildTreeHelper(preorder, preorderStart + leftSubtreeSize + 1, preorderEnd,
                                  inorder, rootIndex + 1, inorderEnd, inorderIndexMap);
    return root;
}
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,
                           inorder, 0, inorder.size() - 1, inorderIndexMap);
}
```

```cpp
void printLevelOrder(TreeNode* root) {
    if (!root) return;
    vector<TreeNode*> currentLevel{root};
    while (!currentLevel.empty()) {
        vector<TreeNode*> nextLevel;
        for (TreeNode* node : currentLevel) {
            if (node) {
                cout << node->val << " ";
                nextLevel.push_back(node->left);
                nextLevel.push_back(node->right);
            } else {
                cout << "null ";
            }
        }
        currentLevel = nextLevel;
    }
    cout << endl;
}

int main() {
    vector<int> preorder1 = {3, 9, 20, 15, 7};
    vector<int> inorder1 = {9, 3, 15, 20, 7};
    TreeNode* root1 = buildTree(preorder1, inorder1);
    cout << "Tree for Example 1: ";
    printLevelOrder(root1);

    vector<int> preorder2 = {-1};
    vector<int> inorder2 = {-1};
    TreeNode* root2 = buildTree(preorder2, inorder2);
    cout << "Tree for Example 2: ";
    printLevelOrder(root2);

    return 0;
}
```

OUTPUT:

```
Tree for Example 1: 3 9 20 null null 15 7 null null null null
Tree for Example 2: -1 null null


..Program finished with exit code 0
Press ENTER to exit console.
```

Q3. Binary tree from inorder and

postorder traversal (medium).

Sol.

```cpp
//Binary tree rfom inorder and postorder travesal(medium).
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;
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) {}
};
TreeNode* buildTreeHelper(const vector<int>& inorder, int inorderStart, int inorderEnd,
                          const vector<int>& postorder, int postorderStart, int postorderEnd,
                          unordered_map<int, int>& inorderIndexMap) {
    if (inorderStart > inorderEnd || postorderStart > postorderEnd) {
        return nullptr;
    }
    int rootVal = postorder[postorderEnd];
    TreeNode* root = new TreeNode(rootVal);

    int rootIndex = inorderIndexMap[rootVal];
    int leftSubtreeSize = rootIndex - inorderStart;
    root->left = buildTreeHelper(inorder, inorderStart, rootIndex - 1,
                                 postorder, postorderStart, postorderStart + leftSubtreeSize - 1, inorderIndexMap);
    root->right = buildTreeHelper(inorder, rootIndex + 1, inorderEnd,
                                  postorder, postorderStart + leftSubtreeSize, postorderEnd - 1, inorderIndexMap);

    return root;
}

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;
    }
```

```
    return buildTreeHelper(inorder, 0, inorder.size() - 1,
                           postorder, 0, postorder.size() - 1, inorderIndexMap);
}

void printLevelOrder(TreeNode* root) {
    if (!root) return;

    vector<TreeNode*> currentLevel{root};
    while (!currentLevel.empty()) {
        vector<TreeNode*> nextLevel;
        for (TreeNode* node : currentLevel) {
            if (node) {
                cout << node->val << " ";
                nextLevel.push_back(node->left);
                nextLevel.push_back(node->right);
            } else {
                cout << "null ";

            }
        }
        currentLevel = nextLevel;

    }
    cout << endl;

}
int main() {
    vector<int> inorder1 = {9, 3, 15, 20, 7};
    vector<int> postorder1 = {9, 15, 7, 20, 3};
    TreeNode* root1 = buildTree(inorder1, postorder1);
    cout << "Tree for Example 1: ";
    printLevelOrder(root1);
    vector<int> inorder2 = {-1};
    vector<int> postorder2 = {-1};
    TreeNode* root2 = buildTree(inorder2, postorder2);
    cout << "Tree for Example 2: ";
    printLevelOrder(root2);
    return 0; }
```

OUTPUT:

```
Tree for Example 1: 3 9 20 null null 15 7 null null null null
Tree for Example 2: -1 null null


...Program finished with exit code 0
```

Q4.Populating next right pointers in each node(hard) Sol.

```cpp
//Populating next right pointers in each node(Hard).
#include <iostream>
#include <vector>
#include <unordered_map>
#include <queue>
using namespace std;
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) {}
};
TreeNode* buildTreeHelper(const vector<int>& inorder, int inorderStart, int inorderEnd,
                          const vector<int>& postorder, int postorderStart, int postorderEnd,
                          unordered_map<int, int>& inorderIndexMap) {
    if (inorderStart > inorderEnd || postorderStart > postorderEnd) {
        return nullptr;
    }
    int rootVal = postorder[postorderEnd];
    TreeNode* root = new TreeNode(rootVal);
    int rootIndex = inorderIndexMap[rootVal];
    int leftSubtreeSize = rootIndex - inorderStart;
    root->left = buildTreeHelper(inorder, inorderStart, rootIndex - 1,
                                 postorder, postorderStart, postorderStart + leftSubtreeSize - 1, inorderIndexMap);
    root->right = buildTreeHelper(inorder, rootIndex + 1, inorderEnd,
                                  postorder, postorderStart + leftSubtreeSize, postorderEnd - 1, inorderIndexMap);
    return root;
}
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;
    }
    return buildTreeHelper(inorder, 0, inorder.size() - 1,
                           postorder, 0, postorder.size() - 1, inorderIndexMap);
}
```

```cpp
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() {
    vector<int> inorder1 = {9, 3, 15, 20, 7};
    vector<int> postorder1 = {9, 15, 7, 20, 3};
    TreeNode* root1 = buildTree(inorder1, postorder1);
    cout << "Tree for Example 1: ";
    printLevelOrder(root1);

    vector<int> inorder2 = {-1};
    vector<int> postorder2 = {-1};
    TreeNode* root2 = buildTree(inorder2, postorder2);
    cout << "Tree for Example 2: ";
    printLevelOrder(root2);
    return 0;
}
```

Output:

```
Tree for Example 1: 3 9 20 null null 15 7 null null null null
Tree for Example 2: -1 null null


...Program finished with exit code 0
Press ENTER to exit console.
```

Q5.Number of good path.(hard)

Sol.

```cpp
//No. of good path(hard).
#include <iostream>
#include <vector>
#include <unordered_map>
#include <algorithm>
using namespace std;
class UnionFind {
public:
    vector<int> parent, rank;

    UnionFind(int n) {
        parent.resize(n);
        rank.resize(n, 1);
        for (int i = 0; i < n; ++i) {
            parent[i] = i;
        }
    }

    int find(int x) {
        if (parent[x] != x) {
            parent[x] = find(parent[x]);
        }
        return parent[x];
    }
    void unite(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);
        if (rootX != rootY) {
            if (rank[rootX] > rank[rootY]) {
                parent[rootY] = rootX;
            } else if (rank[rootX] < rank[rootY]) {
                parent[rootX] = rootY;
            } else {
                parent[rootY] = rootX;
                rank[rootX]++;
            }
        }
    }
```

```cpp
};
int numberOfGoodPaths(vector<int>& vals, vector<vector<int>>& edges) {
    int n = vals.size();
    unordered_map<int, vector<int>> valueToNodes;
    for (int i = 0; i < n; ++i) {
        valueToNodes[vals[i]].push_back(i);
    }
    vector<vector<int>> graph(n);
    for (auto& edge : edges) {
        int a = edge[0], b = edge[1];
        graph[a].push_back(b);
        graph[b].push_back(a);
    }

    UnionFind uf(n);
    vector<bool> visited(n, false);
    int goodPaths = 0;
    for (auto& [value, nodes] : valueToNodes) {
        for (int node : nodes) {
            visited[node] = true;
            for (int neighbor : graph[node]) {
                if (visited[neighbor] && vals[neighbor] <= value) {
                    uf.unite(node, neighbor);
                }
            }
        }
        unordered_map<int, int> componentCount;
        for (int node : nodes) {
            int root = uf.find(node);
            componentCount[root]++;
        }
        for (auto& [root, count] : componentCount) {
            goodPaths += (count * (count + 1)) / 2;
        }
    }

    return goodPaths;
}
```

```cpp
int main() {
    vector<int> vals1 = {1, 3, 2, 1, 3};
    vector<vector<int>> edges1 = {{0, 1}, {0, 2}, {2, 3}, {2, 4}};
    cout << "Example 1 Output: " << numberOfGoodPaths(vals1, edges1) << endl;
    vector<int> vals2 = {1, 1, 2, 2, 3};
    vector<vector<int>> edges2 = {{0, 1}, {1, 2}, {2, 3}, {2, 4}};
    cout << "Example 2 Output: " << numberOfGoodPaths(vals2, edges2) << endl;
    vector<int> vals3 = {1};
    vector<vector<int>> edges3 = {};
    cout << "Example 3 Output: " << numberOfGoodPaths(vals3, edges3) << endl;

    return 0;
}
```

OUTPUT:

```
Example 1 Output: 5
Example 2 Output: 7
Example 3 Output: 1


...Program finished with ex
```