

WINTER DOMAIN CAMP

NAME: Vikram Kumar

SECTION: KPIT-901-B

UID:22BCS12220

DATE: 27/12/2024

Q1. Binary tree inorder traversal(Easy)

Sol.

```
1 //binary tree inorder traversal(easy)
2 #include <iostream>
3 #include <vector>
4 using namespace std;
5 struct TreeNode {
6     int val;
7     TreeNode* left;
8     TreeNode* right;
9     TreeNode() : val(0), left(nullptr), right(nullptr) {}
10    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
11    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {} };
12 void inorderTraversalHelper(TreeNode* root, vector<int>& result) {
13     if (root == nullptr) return;
14     inorderTraversalHelper(root->left, result);
15     result.push_back(root->val);
16     inorderTraversalHelper(root->right, result); }
17 vector<int> inorderTraversal(TreeNode* root) {
18     vector<int> result;
19     inorderTraversalHelper(root, result);
20     return result;}
21 void printVector(const vector<int>& vec) {
22     for (int val : vec) {
23         cout << val << " "; }
24     cout << endl;}
25 int main() {
26     TreeNode* root1 = new TreeNode(1);
27     root1->right = new TreeNode(2);
28     root1->right->left = new TreeNode(3);
29     vector<int> result1 = inorderTraversal(root1);
30     cout << "Inorder Traversal Example 1: ";
31     printVector(result1);
32     TreeNode* root2 = new TreeNode(1);
33     root2->left = new TreeNode(2, new TreeNode(4), new TreeNode(5, new TreeNode(6), new TreeNode(7)));
34     root2->right = new TreeNode(3, nullptr, new TreeNode(8, new TreeNode(9), nullptr));
35     vector<int> result2 = inorderTraversal(root2);
36     cout << "Inorder Traversal Example 2: ";
37     printVector(result2);
38     return 0;}
```

OUTPUT:

```
Inorder Traversal Example 1: 1 3 2
Inorder Traversal Example 2: 4 2 6 5 7 1 3 9 8

Program finished with exit code 0
```

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

```
1 //Construct binary tree form preorder and inorder traversal(medium.)
2 #include <iostream>
3 #include <vector>
4 #include <unordered_map>
5 using namespace std;
6 struct TreeNode {
7     int val;
8     TreeNode* left;
9     TreeNode* right;
10    TreeNode() : val(0), left(nullptr), right(nullptr) {}
11    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
12    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
13 };
14 TreeNode* buildTreeHelper(const vector<int>& preorder, int preorderStart, int preorderEnd,
15                           const vector<int>& inorder, int inorderStart, int inorderEnd,
16                           unordered_map<int, int>& inorderIndexMap) {
17     if (preorderStart > preorderEnd || inorderStart > inorderEnd) {
18         return nullptr;
19     }
20     int rootVal = preorder[preorderStart];
21     TreeNode* root = new TreeNode(rootVal);
22     int rootIndex = inorderIndexMap[rootVal];
23     int leftSubtreeSize = rootIndex - inorderStart;
24     root->left = buildTreeHelper(preorder, preorderStart + 1, preorderStart + leftSubtreeSize,
25                                inorder, inorderStart, rootIndex - 1, inorderIndexMap);
26     root->right = buildTreeHelper(preorder, preorderStart + leftSubtreeSize + 1, preorderEnd,
27                                  inorder, rootIndex + 1, inorderEnd, inorderIndexMap);
28     return root;
29 }
30 TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
31     unordered_map<int, int> inorderIndexMap;
32     for (int i = 0; i < inorder.size(); ++i) {
33         inorderIndexMap[inorder[i]] = i;
34     }
35
36     return buildTreeHelper(preorder, 0, preorder.size() - 1,
37                            inorder, 0, inorder.size() - 1, inorderIndexMap);
38 }
```

```

39 - void printLevelOrder(TreeNode* root) {
40     if (!root) return;
41     vector<TreeNode*> currentLevel{root};
42     while (!currentLevel.empty()) {
43         vector<TreeNode*> nextLevel;
44         for (TreeNode* node : currentLevel) {
45             if (node) {
46                 cout << node->val << " ";
47                 nextLevel.push_back(node->left);
48                 nextLevel.push_back(node->right);
49             } else {
50                 cout << "null ";
51             }
52         }
53         currentLevel = nextLevel;
54     }
55     cout << endl;
56 }
57
58 - int main() {
59     vector<int> preorder1 = {3, 9, 20, 15, 7};
60     vector<int> inorder1 = {9, 3, 15, 20, 7};
61     TreeNode* root1 = buildTree(preorder1, inorder1);
62     cout << "Tree for Example 1: ";
63     printLevelOrder(root1);
64
65     vector<int> preorder2 = {-1};
66     vector<int> inorder2 = {-1};
67     TreeNode* root2 = buildTree(preorder2, inorder2);
68     cout << "Tree for Example 2: ";
69     printLevelOrder(root2);
70
71     return 0;
72 }

```

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.

```
1 //Binary tree from inorder and postorder traversal(medium).
2 #include <iostream>
3 #include <vector>
4 #include <unordered_map>
5 using namespace std;
6 struct TreeNode {
7     int val;
8     TreeNode* left;
9     TreeNode* right;
10    TreeNode() : val(0), left(nullptr), right(nullptr) {}
11    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
12    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
13 };
14 TreeNode* buildTreeHelper(const vector<int>& inorder, int inorderStart, int inorderEnd,
15                           const vector<int>& postorder, int postorderStart, int postorderEnd,
16                           unordered_map<int, int>& inorderIndexMap) {
17     if (inorderStart > inorderEnd || postorderStart > postorderEnd) {
18         return nullptr;
19     }
20     int rootVal = postorder[postorderEnd];
21     TreeNode* root = new TreeNode(rootVal);
22
23     int rootIndex = inorderIndexMap[rootVal];
24     int leftSubtreeSize = rootIndex - inorderStart;
25     root->left = buildTreeHelper(inorder, inorderStart, rootIndex - 1,
26                                postorder, postorderStart, postorderStart + leftSubtreeSize - 1, inorderIndexMap);
27     root->right = buildTreeHelper(inorder, rootIndex + 1, inorderEnd,
28                                  postorder, postorderStart + leftSubtreeSize, postorderEnd - 1, inorderIndexMap);
29
30     return root;
31 }
32
33 TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
34     unordered_map<int, int> inorderIndexMap;
35     for (int i = 0; i < inorder.size(); ++i) {
36         inorderIndexMap[inorder[i]] = i;
37     }
38 }
```

```

39     return buildTreeHelper(inorder, 0, inorder.size() - 1,
40                             postorder, 0, postorder.size() - 1, inorderIndexMap);
41 }
42
43 void printLevelOrder(TreeNode* root) {
44     if (!root) return;
45
46     vector<TreeNode*> currentLevel{root};
47     while (!currentLevel.empty()) {
48         vector<TreeNode*> nextLevel;
49         for (TreeNode* node : currentLevel) {
50             if (node) {
51                 cout << node->val << " ";
52                 nextLevel.push_back(node->left);
53                 nextLevel.push_back(node->right);
54             } else {
55                 cout << "null ";
56             }
57         }
58         currentLevel = nextLevel;
59     }
60     cout << endl;
61 }
62
63
64 }
65 int main() {
66     vector<int> inorder1 = {9, 3, 15, 20, 7};
67     vector<int> postorder1 = {9, 15, 7, 20, 3};
68     TreeNode* root1 = buildTree(inorder1, postorder1);
69     cout << "Tree for Example 1: ";
70     printLevelOrder(root1);
71     vector<int> inorder2 = {-1};
72     vector<int> postorder2 = {-1};
73     TreeNode* root2 = buildTree(inorder2, postorder2);
74     cout << "Tree for Example 2: ";
75     printLevelOrder(root2);
76     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.

```
1 //Populating next right pointers in each node(Hard).
2 #include <iostream>
3 #include <vector>
4 #include <unordered_map>
5 #include <queue>
6 using namespace std;
7 struct TreeNode {
8     int val;
9     TreeNode* left;
10    TreeNode* right;
11    TreeNode() : val(0), left(nullptr), right(nullptr) {}
12    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
13    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
14 };
15 TreeNode* buildTreeHelper(const vector<int>& inorder, int inorderStart, int inorderEnd,
16                           const vector<int>& postorder, int postorderStart, int postorderEnd,
17                           unordered_map<int, int>& inorderIndexMap) {
18     if (inorderStart > inorderEnd || postorderStart > postorderEnd) {
19         return nullptr;
20     }
21     int rootVal = postorder[postorderEnd];
22     TreeNode* root = new TreeNode(rootVal);
23     int rootIndex = inorderIndexMap[rootVal];
24     int leftSubtreeSize = rootIndex - inorderStart;
25     root->left = buildTreeHelper(inorder, inorderStart, rootIndex - 1,
26                                postorder, postorderStart, postorderStart + leftSubtreeSize - 1, inorderIndexMap);
27     root->right = buildTreeHelper(inorder, rootIndex + 1, inorderEnd,
28                                  postorder, postorderStart + leftSubtreeSize, postorderEnd - 1, inorderIndexMap);
29     return root;
30 }
31 TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
32     unordered_map<int, int> inorderIndexMap;
33     for (int i = 0; i < inorder.size(); ++i) {
34         inorderIndexMap[inorder[i]] = i;
35     }
36     return buildTreeHelper(inorder, 0, inorder.size() - 1,
37                            postorder, 0, postorder.size() - 1, inorderIndexMap);
38 }
```

```

39 void printLevelOrder(TreeNode* root) {
40     if (!root) return;
41     queue<TreeNode*> q;
42     q.push(root);
43
44     while (!q.empty()) {
45         TreeNode* node = q.front();
46         q.pop();
47
48         if (node) {
49             cout << node->val << " ";
50             q.push(node->left);
51             q.push(node->right);
52         } else {
53             cout << "null ";
54         }
55     }
56     cout << endl;
57 }
58 int main() {
59     vector<int> inorder1 = {9, 3, 15, 20, 7};
60     vector<int> postorder1 = {9, 15, 7, 20, 3};
61     TreeNode* root1 = buildTree(inorder1, postorder1);
62     cout << "Tree for Example 1: ";
63     printLevelOrder(root1);
64
65     vector<int> inorder2 = {-1};
66     vector<int> postorder2 = {-1};
67     TreeNode* root2 = buildTree(inorder2, postorder2);
68     cout << "Tree for Example 2: ";
69     printLevelOrder(root2);
70     return 0;
71 }

```

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.

```
1 //No. of good path(hard).
2 #include <iostream>
3 #include <vector>
4 #include <unordered_map>
5 #include <algorithm>
6 using namespace std;
7 class UnionFind {
8 public:
9     vector<int> parent, rank;
10
11     UnionFind(int n) {
12         parent.resize(n);
13         rank.resize(n, 1);
14         for (int i = 0; i < n; ++i) {
15             parent[i] = i;
16         }
17     }
18
19     int find(int x) {
20         if (parent[x] != x) {
21             parent[x] = find(parent[x]);
22         }
23         return parent[x];
24     }
25
26     void unite(int x, int y) {
27         int rootX = find(x);
28         int rootY = find(y);
29         if (rootX != rootY) {
30             if (rank[rootX] > rank[rootY]) {
31                 parent[rootY] = rootX;
32             } else if (rank[rootX] < rank[rootY]) {
33                 parent[rootX] = rootY;
34             } else {
35                 parent[rootY] = rootX;
36                 rank[rootX]++;
37             }
38         }
39     }
40 }
```



```

39 };
40 int numberOfGoodPaths(vector<int>& vals, vector<vector<int>>& edges) {
41     int n = vals.size();
42     unordered_map<int, vector<int>> valueToNodes;
43     for (int i = 0; i < n; ++i) {
44         valueToNodes[vals[i]].push_back(i);
45     }
46     vector<vector<int>> graph(n);
47     for (auto& edge : edges) {
48         int a = edge[0], b = edge[1];
49         graph[a].push_back(b);
50         graph[b].push_back(a);
51     }
52
53     UnionFind uf(n);
54     vector<bool> visited(n, false);
55     int goodPaths = 0;
56     for (auto& [value, nodes] : valueToNodes) {
57         for (int node : nodes) {
58             visited[node] = true;
59             for (int neighbor : graph[node]) {
60                 if (visited[neighbor] && vals[neighbor] <= value) {
61                     uf.unite(node, neighbor);
62                 }
63             }
64         }
65         unordered_map<int, int> componentCount;
66         for (int node : nodes) {
67             int root = uf.find(node);
68             componentCount[root]++;
69         }
70         for (auto& [root, count] : componentCount) {
71             goodPaths += (count * (count + 1)) / 2;
72         }
73     }
74
75     return goodPaths;
76 }

```

```

77 int main() {
78     vector<int> vals1 = {1, 3, 2, 1, 3};
79     vector<vector<int>> edges1 = {{0, 1}, {0, 2}, {2, 3}, {2, 4}};
80     cout << "Example 1 Output: " << numberOfGoodPaths(vals1, edges1) << endl;
81     vector<int> vals2 = {1, 1, 2, 2, 3};
82     vector<vector<int>> edges2 = {{0, 1}, {1, 2}, {2, 3}, {2, 4}};
83     cout << "Example 2 Output: " << numberOfGoodPaths(vals2, edges2) << endl;
84     vector<int> vals3 = {1};
85     vector<vector<int>> edges3 = {};
86     cout << "Example 3 Output: " << numberOfGoodPaths(vals3, edges3) << endl;
87
88     return 0;
89 }

```

OUTPUT:

```

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

...Program finished with ex
Press ENTER to exit console

```