

Codeium: Refactor | Explain

```
3 public class DiameterBinaryTree {
4     //Given a binary tree, you need to compute the length
5     // of the diameter of the tree.
6     //The diameter of a binary tree is the length of
7     // the longest path between any two nodes in a tree.
8     //This path may or may not pass through the root.
9     //Example: Given a binary tree
10    //      1
11    //     / \
12    //    2  3
13    //   / \
14    //  4  5
15    //Return 3, which is the length of the path [4,2,1,3] or [5,2,1,3].
16
```

Codeium: Refactor | Explain

```
17 static class TreeNode {
18     int val;
19     TreeNode left;
20     TreeNode right;
21     TreeNode(int x) {
22         val = x;
23     }
24 }
25
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
26 public static void main(String[] args) {
27     TreeNode root = new TreeNode(1);
28     root.left = new TreeNode(2);
29     root.right = new TreeNode(3);
30     root.left.left = new TreeNode(4);
31     root.left.right = new TreeNode(5);
32     System.out.println(diameterOfBinaryTree(root));
33 }
34
35 static int max = 0;
```

35     static int max = 0;

Codeium: Refactor | Explain | Generate Javadoc | X

36     public static int diameterOfBinaryTree(TreeNode root) {

37         if (root == null) {

38             return 0;

39         }

40         maxDepth(root);

41         return max;

42     }

43

Codeium: Refactor | Explain | Generate Javadoc | X

44     public static int maxDepth(TreeNode root) {

45         if (root == null) {

46             return 0;

47         }

48

49         int left = maxDepth(root.left);

50         int right = maxDepth(root.right);

51

52         max = Math.max(max, left + right);

53

54         return 1 + Math.max(left, right);

55     }

56

57     //Solve by Dynamic Programming

58     //Approach: Dynamic Programming

Codeium: Refactor | Explain | X

59     public static int diameterOfBinaryTreeDP(TreeNode root) {

60         if (root == null) {

61             return 0;

62         }

63         int[] max = new int[1];

64         maxDepth(root, max);

65         return max[0];

66     }

67

Codeium: Refactor | Explain | Generate Javadoc | ✕

```
68 public static int maxDepth(TreeNode root, int[] max) {
69     if (root == null) {
70         return 0;
71     }
72
73     int left = maxDepth(root.left, max);
74     int right = maxDepth(root.right, max);
75
76     max[0] = Math.max(max[0], left + right);
77
78     return 1 + Math.max(left, right);
79 }
80
81
82 }
83
```

.DiameterBinaryTree

3

Codeium: Refactor | Explain

```
3 public class DiameterNaryTree {
4     //Given an N-ary tree, find the diameter of the tree.
5     //The diameter of an N-ary tree is the length of the
6     // longest path between any two nodes in a tree.
7     //This path may or may not pass through the root.
8     //Example: Given a binary tree
9     //      1
10    //    /\ \
11    //   2 3 4 5
12    //  /\ \
13    // 6 7 8
14    //Return 4, which is the length of the path [6,2,1,3,4,8,7,2].
15
```

Codeium: Refactor | Explain

```
16 static class TreeNode {
17     int val;
18     TreeNode[] children;
19     TreeNode(int x) {
20         val = x;
21     }
22 }
23
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
24 public static void main(String[] args) {
25     TreeNode root = new TreeNode(1);
26     root.children = new TreeNode[4];
27     root.children[0] = new TreeNode(2);
28     root.children[1] = new TreeNode(3);
29     root.children[2] = new TreeNode(4);
30     root.children[3] = new TreeNode(5);
31     root.children[0].children = new TreeNode[3];
32     root.children[0].children[0] = new TreeNode(6);
33     root.children[0].children[1] = new TreeNode(7);
34     root.children[0].children[2] = new TreeNode(8);
35     System.out.println(diameterOfNaryTreeDP(root));
36 }
37
38 static int max = 0;
```

```
37
38     static int max = 0;
Codeium: Refactor | Explain | Generate Javadoc | ✕
39     public static int diameterOfNaryTree(TreeNode root) {
40         if (root == null) {
41             return 0;
42         }
43         maxDepth(root);
44         return max;
45     }
46
```

```
Codeium: Refactor | Explain | Generate Javadoc | ✕
47     public static int maxDepth(TreeNode root) {
48         if (root == null) {
49             return 0;
50         }
51
52         int firstMax = 0;
53         int secondMax = 0;
54         for (TreeNode child : root.children) {
55             int depth = maxDepth(child);
56             if (depth > firstMax) {
57                 secondMax = firstMax;
58                 firstMax = depth;
59             } else if (depth > secondMax) {
60                 secondMax = depth;
61             }
62         }
63
64         max = Math.max(max, firstMax + secondMax);
65
66         return 1 + firstMax;
67     }
68
```

```

68
69 //Solve by Dynamic Programming
70 //Approach: Dynamic Programming
Codeium: Refactor | Explain | ✕
71 public static int diameterOfNaryTreeDP(TreeNode root) {
72     if (root == null) {
73         return 0;
74     }
75     int[] max = new int[1];
76     maxDepth(root, max);
77     return max[0];
78 }
79

```

```

Codeium: Refactor | Explain | Generate Javadoc | ✕
80 public static int maxDepth(TreeNode root, int[] max) {
81     if (root == null) {
82         return 0;
83     }
84
85     int firstMax = 0;
86     int secondMax = 0;
87     for (TreeNode child : root.children) {
88         int depth = maxDepth(child, max);
89         if (depth > firstMax) {
90             secondMax = firstMax;
91             firstMax = depth;
92         } else if (depth > secondMax) {
93             secondMax = depth;
94         }
95     }
96
97     max[0] = Math.max(max[0], firstMax + secondMax);
98
99     return 1 + firstMax;
100 }
101
102 }
103

```

Codeium: Refactor | Explain

```
3 public class MaximumPathSumAnyNodeToAnyNode {
4     //Given a binary tree, find the maximum path sum.
5     // The path may start and end at any node in the tree.
6     //Example: Given the below binary tree
7     //      1
8     //     / \
9     //    2  3
10    //Return 6. The path 2 -> 1 -> 3 gives the maximum
11    // path sum.
12
```

Codeium: Refactor | Explain

```
13 static class TreeNode {
14     int val;
15     TreeNode left;
16     TreeNode right;
17     TreeNode(int x) {
18         val = x;
19     }
20 }
21
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
22 public static void main(String[] args) {
23     TreeNode root = new TreeNode(1);
24     root.left = new TreeNode(2);
25     root.right = new TreeNode(3);
26     System.out.println(maxPathSum(root));
27 }
28
29 static int max = Integer.MIN_VALUE;
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
30 public static int maxPathSum(TreeNode root) {
31     if (root == null) {
32         return 0;
33     }
34     maxPathSumUtil(root);
35     return max;
36 }
37
```

```

38  Codeium: Refactor | Explain | Generate Javadoc | ✕
39  public static int maxPathSumUtil(TreeNode root) {
40      if (root == null) {
41          return 0;
42      }
43
44      int left = Math.max(0, maxPathSumUtil(root.left));
45      int right = Math.max(0, maxPathSumUtil(root.right));
46
47      max = Math.max(max, left + right + root.val);
48
49      return Math.max(left, right) + root.val;
50  }
51
52  //Approach: Dynamic Programming
53  Codeium: Refactor | Explain | ✕
54  public static int maxPathSumDP(TreeNode root) {
55      if (root == null) {
56          return 0;
57      }
58
59      int[] max = new int[1];
60      max[0] = Integer.MIN_VALUE;
61      maxPathSumUtilDP(root, max);
62      return max[0];
63  }
64
65  Codeium: Refactor | Explain | Generate Javadoc | ✕
66  public static int maxPathSumUtilDP(TreeNode root, int[] max) {
67      if (root == null) {
68          return 0;
69      }
70
71      int left = Math.max(0, maxPathSumUtilDP(root.left, max));
72      int right = Math.max(0, maxPathSumUtilDP(root.right, max));
73
74      max[0] = Math.max(max[0], left + right + root.val);
75
76      return Math.max(left, right) + root.val;
77  }

```

**.MaximumPathSumAnyNodeToAnyNode**



Codeium: Refactor | Explain

```
3 public class MaximumPathSumLeafToLeaf {
4     //If a binary tree is given, how to find Maximum path sum
5     // between two leaves of binary tree.
6     //
7     //All should be numbers
8     //The maximum sum path may or may not go through root.
9     //For example, in the following binary tree, the maximum sum
10    // is 27(3 + 6 + 9 + 0 -1 + 10). Expected time complexity is O(n)
11    // where n is the number of nodes in the given Binary Tree.
12    //      -15
13    //      / \
14    //     5  6
15    //    / \ / \
16    //   -8 1 3 9
17    //  / / \ / \
18    // 2 6  0 6 4
19    //  / \
20    // 0  10
21    //Approach: Dynamic Programming
22
```

Codeium: Refactor | Explain

```
23 static class TreeNode {
24     int val;
25     TreeNode left;
26     TreeNode right;
27     TreeNode(int x) {
28         val = x;
29     }
30 }
31
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
32 public static void main(String[] args) {
33     TreeNode root = new TreeNode(-15);
34     root.left = new TreeNode(5);
35     root.right = new TreeNode(6);
36     root.left.left = new TreeNode(-8);
37     root.left.right = new TreeNode(1);
38     root.right.left = new TreeNode(3);
39     root.right.right = new TreeNode(9);
40     root.left.left.left = new TreeNode(2);
41     root.left.right.left = new TreeNode(6);
42     root.left.right.right = new TreeNode(0);
43     root.right.right.left = new TreeNode(6);
44     root.right.right.right = new TreeNode(4);
45     root.left.right.left.left = new TreeNode(0);
46     root.left.right.left.right = new TreeNode(10);
47     System.out.println(maxPathSum(root));
48 }
49
```

```
50 static int max = Integer.MIN_VALUE;
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
51 public static int maxPathSum(TreeNode root) {
52     if (root == null) {
53         return 0;
54     }
55     maxPathSumUtil(root);
56     return max;
57 }
58
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
59 public static int maxPathSumUtil(TreeNode root) {
60     if (root == null) {
61         return 0;
62     }
63
64     int left = Math.max(0, maxPathSumUtil(root.left));
65     int right = Math.max(0, maxPathSumUtil(root.right));
66
67     max = Math.max(max, left + right + root.val);
68
69     return Math.max(left, right) + root.val;
70 }
71
```

```

70     }
71
72     //Approach: Dynamic Programming
73     Codeium: Refactor | Explain | ✕
74     public static int maxPathSumDP(TreeNode root) {
75         if (root == null) {
76             return 0;
77         }
78
79         int[] max = new int[1];
80         max[0] = Integer.MIN_VALUE;
81         maxPathSumUtilDP(root, max);
82         return max[0];
83     }
84
85     Codeium: Refactor | Explain | Generate Javadoc | ✕
86     public static int maxPathSumUtilDP(TreeNode root, int[] max) {
87         if (root == null) {
88             return 0;
89         }
90
91         int left = Math.max(0, maxPathSumUtilDP(root.left, max));
92         int right = Math.max(0, maxPathSumUtilDP(root.right, max));
93
94         max[0] = Math.max(max[0], left + right + root.val);
95
96         return Math.max(left, right) + root.val;
97     }

```

## .MaximumPathSumLeafToLeaf