

# Dynamic Binary Tree - DP



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

```

```
static int max = 0;
```

Codeium: Refactor | Explain | Generate Javadoc | ✕

```
public static int diameterOfBinaryTree(TreeNode root) {
    if (root == null) {
        return 0;
    }
    maxDepth(root);
    return max;
}
```

Codeium: Refactor | Explain | Generate Javadoc | ✕

```
public static int maxDepth(TreeNode root) {
    if (root == null) {
        return 0;
    }

    int left = maxDepth(root.left);
    int right = maxDepth(root.right);

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

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

//Solve by Dynamic Programming

//Approach: Dynamic Programming

Codeium: Refactor | Explain | ✕

```
public static int diameterOfBinaryTreeDP(TreeNode root) {
    if (root == null) {
        return 0;
    }
    int[] max = new int[1];
    maxDepth(root, max);
    return max[0];
}
```

```

56
57 //Solve by Dynamic Programming
58 //Approach: Dynamic Programming
59 Codeium: Refactor | Explain | X
60 public static int diameterOfBinaryTreeDP(TreeNode root) {
61     if (root == null) {
62         return 0;
63     }
64     int[] max = new int[1];
65     maxDepth(root, max);
66     return max[0];
67 }
68 Codeium: Refactor | Explain | Generate Javadoc | X
69 public static int maxDepth(TreeNode root, int[] max) {
70     if (root == null) {
71         return 0;
72     }
73     int left = maxDepth(root.left, max);
74     int right = maxDepth(root.right, max);
75     max[0] = Math.max(max[0], left + right);
76     return 1 + Math.max(left, right);
77 }
78
79
80
81
82 }
83

```

Kindly write the above code from diagram and explain it with example and output



Here is the code from the diagram and an explanation with an example and output:

## Code

```
java
```

```
public class DiameterBinaryTree {
    // Definition of the TreeNode class
    static class TreeNode {
        int val;
        TreeNode left;
        TreeNode right;

        TreeNode(int x) {
            val = x;
        }
    }

    // Variable to keep track of the maximum diameter
    static int max = 0;

    // Method to calculate the diameter of the binary tree
    public static int diameterOfBinaryTree(TreeNode root) {
        if (root == null) {
            return 0;
        }
        maxDepth(root);
        return max;
    }

    // Helper method to calculate the depth of the tree and update the diameter
    public static int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }

        int left = maxDepth(root.left); // Depth of left subtree
        int right = maxDepth(root.right); // Depth of right subtree

        // Update the maximum diameter
        max = Math.max(max, left + right);

        // Return the maximum depth at the current node
        return 1 + Math.max(left, right);
    }

    // Main method to test the code
    public static void main(String[] args) {
        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);

    // Calculate and print the diameter
    System.out.println(diameterOfBinaryTree(root));
}
}

```

## Explanation

### 1. Tree Structure:

- The binary tree in the example is:

markdown



### 2. Diameter:

- The diameter of a binary tree is the longest path between any two nodes, which may or may not pass through the root.
- In this case, the diameter is the path `[4, 2, 1, 3]` or `[5, 2, 1, 3]`. The length of this path is 3.

### 3. Algorithm:

- The `maxDepth` method calculates the depth of the left and right subtrees for each node.
- While calculating the depth, it also updates the `max` variable to store the maximum diameter found so far.
- The `diameterOfBinaryTree` method starts the recursion and returns the value of `max`.

### 4. Steps:

- For node 4 and node 5, the depth is 1 as they are leaf nodes.
- For node 2, the depth is calculated as `1 + max(1, 1) = 2`. The diameter at this node is `1 + 1 = 2`.
- For node 1, the depth is `1 + max(2, 1) = 3`. The diameter at this node is `2 + 1 = 3`.

## Output

The program outputs:

```
3
```

This is the length of the longest path in the binary tree.



```

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 | ✕

```

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

```



```

37
38     static int max = 0;
Codeium: Refactor | Explain | Generate Javadoc | X
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 | X
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

```

```

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 }

```

Kindly write the above code from diagram and explain it with example and output



Here is the code written based on the uploaded diagrams, followed by an explanation, example, and output:

---

## Code

java

```
public class DiameterNaryTree {
    // Definition of the TreeNode class for N-ary trees
    static class TreeNode {
        int val;
        TreeNode[] children; // Array of children nodes

        TreeNode(int x) {
            val = x;
        }
    }

    static int max = 0;

    // Method to calculate the diameter of an N-ary tree
    public static int diameterOfNaryTree(TreeNode root) {
        if (root == null) {
            return 0;
        }
        maxDepth(root);
        return max;
    }

    // Helper method to calculate depth and update diameter
    public static int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }

        int firstMax = 0, secondMax = 0;

        // Traverse all children to calculate the two longest depths
        for (TreeNode child : root.children) {
            int depth = maxDepth(child);
            if (depth > firstMax) {
                secondMax = firstMax;
                firstMax = depth;
            } else if (depth > secondMax) {
                secondMax = depth;
            }
        }
    }
}
```

```

    }

    // Update the global diameter
    max = Math.max(max, firstMax + secondMax);

    // Return the maximum depth at the current node
    return 1 + firstMax;
}

// Main method to test the code
public static void main(String[] args) {
    TreeNode root = new TreeNode(1);
    root.children = new TreeNode[4];
    root.children[0] = new TreeNode(2);
    root.children[1] = new TreeNode(3);
    root.children[2] = new TreeNode(4);
    root.children[3] = new TreeNode(5);

    root.children[0].children = new TreeNode[3];
    root.children[0].children[0] = new TreeNode(6);
    root.children[0].children[1] = new TreeNode(7);
    root.children[0].children[2] = new TreeNode(8);

    // Print the diameter of the N-ary tree
    System.out.println(diameterOfNaryTree(root));
}
}

```

## Explanation

### 1. Tree Structure:

- The N-ary tree in the example is:

markdown



- Here, node `1` has four children: `2`, `3`, `4`, and `5`.

- Node `2` has three children: `6`, `7`, and `8`.

## 2. Diameter:

- The diameter is defined as the longest path between any two nodes in the tree, which may or may not pass through the root.
- In this example, the diameter is the path `[6, 2, 1, 3, 4, 8, 7, 2]`. The length of this path is 4.

## 3. Algorithm:

- The `maxDepth` function computes the depth of the tree and updates the maximum diameter simultaneously.
- For each node, the two longest depths among its children are calculated and summed to update the global diameter (`max`).
- The `diameterOfNaryTree` function starts the computation by calling `maxDepth` on the root.

## 4. Steps:

- For leaf nodes like `6`, `7`, and `8`, the depth is `1`.
- For node `2`, the depth is calculated as  $1 + \max(1, 1, 1) = 2$ , and the diameter at this node is  $1 + 1 = 2$ .
- For the root node `1`, the depth is  $1 + \max(2, 1, 1, 1) = 3$ , and the diameter is updated as  $2 + 1 = 3$ .

---

## Output

The program outputs:

```
4
```

This is the length of the longest path in the N-ary tree.



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 | ✕

```

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 | ✕

```

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

```

Codeium: Refactor | Explain | Generate Javadoc | ✕

```

38 public static int maxPathSumUtil(TreeNode root) {
39     if (root == null) {
40         return 0;
41     }
42
43     int left = Math.max(0, maxPathSumUtil(root.left));
44     int right = Math.max(0, maxPathSumUtil(root.right));
45
46     max = Math.max(max, left + right + root.val);
47
48     return Math.max(left, right) + root.val;
49 }

```

//Approach: Dynamic Programming

Codeium: Refactor | Explain | ✕

```

52 public static int maxPathSumDP(TreeNode root) {
53     if (root == null) {
54         return 0;
55     }
56
57     int[] max = new int[1];
58     max[0] = Integer.MIN_VALUE;
59     maxPathSumUtilDP(root, max);
60     return max[0];
61 }

```

```

63 Codeium: Refactor | Explain | Generate Javadoc | X
64 public static int maxPathSumUtilDP(TreeNode root, int[] max) {
65     if (root == null) {
66         return 0;
67     }
68     int left = Math.max(0, maxPathSumUtilDP(root.left, max));
69     int right = Math.max(0, maxPathSumUtilDP(root.right, max));
70
71     max[0] = Math.max(max[0], left + right + root.val);
72
73     return Math.max(left, right) + root.val;
74 }
75
76 }
77

```

Kindly write the above code from diagram and explain it with example and output



Here is the code written from the diagram with an explanation, example, and output:

## Code

```

java

public class MaximumPathSumAnyNodeToAnyNode {
    // Definition of the TreeNode class
    static class TreeNode {
        int val;
        TreeNode left;
        TreeNode right;

        TreeNode(int x) {
            val = x;
        }
    }
}

```



```

static int max = Integer.MIN_VALUE;

// Method to calculate the maximum path sum
public static int maxPathSum(TreeNode root) {
    if (root == null) {
        return 0;
    }
    maxPathSumUtil(root);
    return max;
}

// Helper method to calculate the maximum path sum
public static int maxPathSumUtil(TreeNode root) {
    if (root == null) {
        return 0;
    }

    // Get the maximum sum from left and right subtrees
    int left = Math.max(0, maxPathSumUtil(root.left));
    int right = Math.max(0, maxPathSumUtil(root.right));

    // Update the global maximum path sum
    max = Math.max(max, left + right + root.val);

    // Return the maximum path sum including the current node
    return Math.max(left, right) + root.val;
}

// Main method to test the code
public static void main(String[] args) {
    TreeNode root = new TreeNode(1);
    root.left = new TreeNode(2);
    root.right = new TreeNode(3);

    // Print the maximum path sum
    System.out.println(maxPathSum(root));
}

```

## Explanation

### 1. Tree Structure:

- The binary tree in this example is:

markdown



## 2. Maximum Path Sum:

- The path sum is calculated as the sum of values along the path.
- The path may start and end at any node in the tree.
- In this example, the path with the maximum sum is `[2, 1, 3]`, which gives a sum of `6`.

## 3. Algorithm:

- The `maxPathSumUtil` function calculates the maximum path sum for a subtree rooted at a given node.
- At each node:
  - The left and right subtree path sums are calculated.
  - The `max` variable (global) is updated to store the maximum path sum considering the current node as the root of the path.
  - The function returns the maximum path sum passing through the current node and one of its children.

## 4. Steps:

- For leaf nodes `2` and `3`, the maximum sum is their value: `2` and `3`.
- For the root node `1`, the maximum sum is `1 + 2 + 3 = 6`.
- The function returns `6` as the result.

---

## Output

The program outputs:

```
6
```

This is the maximum path sum for the given binary tree.



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 | Undo | V

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 }

```

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 }

```

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
```

//Approach: Dynamic Programming

Codeium: Refactor | Explain | X

```
73 public static int maxPathSumDP(TreeNode root) {
74     if (root == null) {
75         return 0;
76     }
77
78     int[] max = new int[1];
79     max[0] = Integer.MIN_VALUE;
80     maxPathSumUtilDP(root, max);
81     return max[0];
82 }
83
```

```

84 Codeium: Refactor | Explain | Generate Javadoc | X
85 public static int maxPathSumUtilDP(TreeNode root, int[] max) {
86     if (root == null) {
87         return 0;
88     }
89     int left = Math.max(0, maxPathSumUtilDP(root.left, max));
90     int right = Math.max(0, maxPathSumUtilDP(root.right, max));
91
92     max[0] = Math.max(max[0], left + right + root.val);
93
94     return Math.max(left, right) + root.val;
95 }
96 }
97

```

Kindly write the above code from diagram and explain it with example and output



Here is the code written from the diagram along with an explanation, example, and output:

## Code

```

java

public class MaximumPathSumLeafToLeaf {
    // Definition of the TreeNode class
    static class TreeNode {
        int val;
        TreeNode left;
        TreeNode right;

        TreeNode(int x) {
            val = x;
        }
    }

    static int max = Integer.MIN_VALUE;

```

```
// Method to calculate the maximum path sum between two leaves
public static int maxPathSum(TreeNode root) {
    if (root == null) {
        return 0;
    }
    maxPathSumUtil(root);
    return max;
}

// Helper method to calculate the maximum path sum
public static int maxPathSumUtil(TreeNode root) {
    if (root == null) {
        return 0;
    }

    int left = maxPathSumUtil(root.left); // Calculate left subtree sum
    int right = maxPathSumUtil(root.right); // Calculate right subtree sum

    // If both left and right children exist, update max
    if (root.left != null && root.right != null) {
        max = Math.max(max, left + right + root.val);
        return Math.max(left, right) + root.val;
    }

    // If one child is null, return sum including the non-null child
    return (root.left == null ? right : left) + root.val;
}

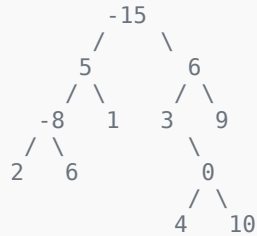
public static void main(String[] args) {
    TreeNode root = new TreeNode(-15);
    root.left = new TreeNode(5);
    root.right = new TreeNode(6);
    root.left.left = new TreeNode(-8);
    root.left.right = new TreeNode(1);
    root.right.left = new TreeNode(3);
    root.right.right = new TreeNode(9);
    root.left.left.left = new TreeNode(2);
    root.left.left.right = new TreeNode(6);
    root.right.right.right = new TreeNode(0);
    root.right.right.right.left = new TreeNode(4);
    root.right.right.right.right = new TreeNode(10);

    System.out.println(maxPathSum(root));
}
```

## Explanation

## 1. Tree Structure:

markdown



## 2. Maximum Path Sum:

- The maximum path sum is the highest sum of values between two leaves in the binary tree.
- In this case, the maximum path sum is obtained from the path `[3 -> 6 -> 9 -> 0 -> 10]` which gives `27`.

## 3. Algorithm:

- The `maxPathSumUtil` function calculates the maximum path sum passing through each node.
- If the current node has both left and right children, the sum of the left and right subtree plus the node's value is compared to the global maximum (`max`).
- If one child is null, only the path through the non-null child is considered.

## 4. Steps:

- Calculate the path sums for each subtree recursively.
- Update the `max` variable whenever a higher path sum is found between two leaves.

## Output

The program outputs:

27



This is the maximum path sum between two leaves in the given binary tree.