Santhosh Nuthulapati	santhosh.nuthulapati@uci.edu	snuthula
Zhihui Xia	zhihuix1@uci.edu	zhihiux1
Shuqing Ye	shuqingy2@uci.edu	shuqingy2

Read and **understand** programming **problem** statements. Ask questions.

- **Identify edge cases** for the problem
- **Define** effective **test case**(s) and expected result(s) for program
- **Design** one **algorithmic solution** on paper (or whiteboard)
- **Analyze** the **time** and space **complexity** of the solution
- **Write** nearly correct **code on paper** (or whiteboard) to solve problem
- **Explain** your algorithm/**program** to others
- **Simulate test case** and verify your program produces correct results
- Maybe, **implement, test**, and demonstrate correct function of the solution

Identify edge cases and **Define** effective **test case**(s) and expected result(s) for program:

1. Root == nullptr
   Expected output: 0

2. Root has no left or right child : [1, nullptr, 1]
   Expected output: 1

3. Root has no left and right child [1]
   Expected output: 0

4. Input = [5,4,5,1,1,5]
   Output = 2

5. Input = [1,4,5,4,4,5]
   Output = 2

Design one **algorithmic solution** on paper, and write down the pseudocode (or whiteboard, or a text file.), put the screenshot or text here:

```
max_length = 0

dfs(root):
    return 0 if root is null

    left = dfs(root.left)
    right = dfs(root.right)

    if root.left and root.left.val == root.val: left += 1
    else left = 0

    if root.right and root.right.val == root.val: right += 1
    else right = 0

    max_length = max(max_length, left + right)
    return max(left, right)

dfs(root)
return max_length
```

Analyze the **time** and space **complexity** of the solution:

Time: T(n) = O(n)

Space: O(1) + O(height of the tree) (Call Stack)

Explain your algorithm/program in simple words:

We do a DFS on left and right subtrees for each node. (Which returns the maximum length in left/right subtree respectively)

Increment the left subtree path if the left child's value equals the node's value.

Similarly for the right subtree path if the right child's value equals the node's value.

We check if the global maximum is greater than the sum of left and right paths, if yes, we update the global maximum value.

Global maximum at the end of recursive DFS is the final solution.

Simulate test case and verify your program produces correct results:

1. Root == nullptr : directly return 0
2. Root.left == nullptr or Root.right == nullptr

    The value we get from dfs(left) or dfs(right) will be 0.

3. Root has no left and right child

    Max_length is not updated, return 0

4. root = [5,4,5,1,1,5]

```
dfs(root[0])
    left = dfs(root[1]):
        left = dfs(root[3]) = 0
            left = dfs(nullptr) = 0
            right = dfs(nullptr) = 0
            left = 0
            Right = 0
            Max_length = 0
        right = dfs(root[4]) = 0
            left = dfs(nullptr) = 0
            right = dfs(nullptr) = 0
            left = 0
            Right = 0
            Max_length = 0
        left = 0
        right = 0
        Max_length =0
    right = dfs(root[2]) = 1
        left = 0
        right= dfs(root[5])
            left = dfs(nullptr) = 0
            right = dfs(nullptr) = 0
            left = 0
            Right = 0
            Max_length = 0
        left = 0
        right = right + 1 = 1  //Root[2].val == root[5].value
        max_length = 1
    left = 0
```

```
        right = right + 1 = 2 // (root[0].val == root[2].val)
        max_length = max(max_length, right+left) = 2
return max_length=2
```