Worksheet - 22

515. Find Largest Value in Each Tree Row

```
class Solution {
// Static array to store the largest value at each level
static Integer[] largest = new Integer[10_001]; of the binary tree
// Variable to track the current level being processed
int largestIdx = 0;
// Method to find the largest value at each level in the binary tree
public List<Integer> largestValues(TreeNode root) {
// Call the depth-first search method starting from the root node at level 0
dfs(root, 0);
// Convert the array of largest values to a list and return it
return Arrays.asList(Arrays.copyOf(largest, largestIdx));
}
// Depth-first search method to traverse the binary tree
private void dfs(TreeNode node, int level) {
// Base case: If the node is null, return
if (node == null)  return;
// If the current level is greater than or equal to the largestIdx (current number of levels processed)
if (level >= largestIdx) {
// Update the largestIdx to the current level plus 1
largestIdx = level + 1;
// Store the value of the current node as the largest value at the current level
largest[level] = node.val;
} else {
// Compare the value of the current node with the largest value at the current level and store the maximum value
largest[level] = Math.max(largest[level], node.val);
}
```

```
// Recursively call the dfs method for the left child of the current node, incrementing the level by 1
dfs(node.left,  level + 1);
// Recursively call the dfs method for the right child of the current node, incrementing the level by 1
dfs(node.right, level + 1);
}
}


public class Main {
public static void main(String[] args) {
// Create a binary tree
TreeNode root = new TreeNode(1);
root.left = new TreeNode(3);
root.right = new TreeNode(2);
root.left.left = new TreeNode(5);
root.left.right = new TreeNode(3);
root.right.right = new TreeNode(9);

// Create an instance of the Solution class
Solution solution = new Solution();

// Call the largestValues method and get the result
List<Integer> largestValues = solution.largestValues(root);

// Print the largest values at each level
System.out.println("Largest values at each level: " + largestValues);
}
}
```

520. Detect Capital

```
class Solution {
public boolean detectCapitalUse(String word) {
// Check if the length of the word is 0 or 1
if (word.length() == 0 || word.length() == 1) {
// Return true as there is only one character or no characters
return true;
}

// Check if the first character is uppercase
if (Character.isUpperCase(word.charAt(0))) {
// Check if the second character is uppercase
boolean isFirstCharacter = Character.isUpperCase(word.charAt(1));
// Iterate through the rest of the characters
for (int i = 2; i < word.length(); i++) {
// Check if the current character is uppercase
boolean currentCharState = Character.isUpperCase(word.charAt(i));
// If the current character's state (uppercase or lowercase) is different from the first character's state, return false
if (currentCharState != isFirstCharacter) {
return false;
}
}
} else {
// If the first character is lowercase, check that all the remaining characters are also lowercase
for (int i = 1; i < word.length(); i++) {
// If any of the remaining characters are uppercase, return false
if (Character.isUpperCase(word.charAt(i))) {
return false;
}
```

```
}
}

// If all the checks pass, return true
return true;
}

class Main {
public static void main(String[] args) {
// Test cases
Solution solution = new Solution();

String word1 = "USA";
System.out.println(word1 + ": " + solution.detectCapitalUse(word1));

String word2 = "leetcode";
System.out.println(word2 + ": " + solution.detectCapitalUse(word2));

String word3 = "Google";
System.out.println(word3 + ": " + solution.detectCapitalUse(word3));

String word4 = "FlaG";
System.out.println(word4 + ": " + solution.detectCapitalUse(word4));
}
}
}
```