Solution@15-12-23

2601. Prime Subtraction Operation

```
import java.util.Scanner;


public class PrimeSubtractionOperation {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);


      // Input size of the array
      System.out.print("Enter the size of the array: ");
      int n = scanner.nextInt();


      // Input array elements
      System.out.print("Enter the array elements: ");
      int[] nums = new int[n];
      for (int i = 0; i < n; i++) {
         nums[i] = scanner.nextInt();
      }


      // Check if it's possible to make the array strictly increasing
      boolean result = canMakeIncreasing(nums);
      System.out.println("Result: " + result);
   }


   private static boolean canMakeIncreasing(int[] nums) {
```

```
    int n = nums.length;

    // Iterate through the array and check if it's possible to make it strictly increasing
    for (int i = 1; i < n; i++) {
        if (nums[i] <= nums[i - 1]) {
            // If the current element is not greater than the previous one, try to subtract a prime
            int diff = nums[i - 1] - nums[i] + 1;
            if (!isPrime(diff)) {
                return false; // If unable to find a prime to subtract, return false
            }
            nums[i] = nums[i - 1] + 1; // Subtract the prime to make the array strictly increasing
        }
    }

    return true; // If the array is strictly increasing after the operations, return true
}

private static boolean isPrime(int num) {
    if (num <= 1) {
        return false;
    }
    for (int i = 2; i <= Math.sqrt(num); i++) {
        if (num % i == 0) {
            return false;
        }
```

```
        }
        return true;
    }
}
```

Output

```
PS C:\Users\Thakur Sahab\Desktop\java> java PrimeSubtractionOperation
Enter the size of the array: 4
Enter the array elements: 6 10 11 12
Result: true
```

2667. Create Hello World Function

```
import java.util.function.Supplier;

public class HelloWorldFunction {

    public static void main(String[] args) {

        Supplier<String> f1 = createHelloWorld();
        System.out.println(f1.get()); // Output: "Hello World"


    }

    public static Supplier<String> createHelloWorld() {
```

```
        return () -> "Hello World";
    }
}
```

Output:-

```
PS C:\Users\Thakur Sahab\Desktop\java> javac HelloWorldFunction.java
PS C:\Users\Thakur Sahab\Desktop\java> java HelloWorldFunction
Hello World
```

2666. Allow One Function Call

```
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

public class OnceFunctionCall {

    public static void main(String[] args) {
        // Example 1
        Function<int[], Integer> fn1 = (a) -> a[0] + a[1] + a[2];
        int[][] calls1 = {{1, 2, 3}, {2, 3, 6}};
        System.out.println(once(fn1, calls1)); // Output: [{"calls":1,"value":6}]

        // Example 2
        Function<int[], Integer> fn2 = (a) -> a[0] * a[1] * a[2];
        int[][] calls2 = {{5, 7, 4}, {2, 3, 6}, {4, 6, 8}};
        System.out.println(once(fn2, calls2)); // Output: [{"calls":1,"value":140}]
```

```
    }

    public static String once(Function<int[], Integer> fn, int[][] calls) {
        Map<String, Object> result = new HashMap<>();
        int callCount = 0;
        Integer value = null;

        for (int[] args : calls) {
            if (callCount == 0) {
                // The first time the function is called, store the result and increment callCount
                value = fn.apply(args);
                callCount++;
            }
        }

        result.put("calls", callCount);
        result.put("value", value);

        return "[" + result.toString() + "]";
    }
}
```

Output:-

```
PS C:\Users\Thakur Sahab\Desktop\java> javac OnceFunctionCall.java
PS C:\Users\Thakur Sahab\Desktop\java> java OnceFunctionCall
[{calls=1, value=6}]
```

[{calls=1, value=140}]

60. Permutation Sequence

```
import java.util.ArrayList;
import java.util.List;

public class PermutationSequence {

    public static void main(String[] args) {
        int n = 3;
        int k = 4;

        String result = getPermutation(n, k);
        System.out.println(result); // Output: "231"
    }

    public static String getPermutation(int n, int k) {
        List<Integer> numbers = new ArrayList<>();
        StringBuilder result = new StringBuilder();

        // Populate the numbers list
        for (int i = 1; i <= n; i++) {
            numbers.add(i);
        }
```

```
        // Calculate the factorial
        int[] factorial = new int[n];
        factorial[0] = 1;
        for (int i = 1; i < n; i++) {
            factorial[i] = i * factorial[i - 1];
        }
        // Adjust k to be 0-based
        k--;

        generatePermutation(numbers, k, factorial, result);
        return result.toString();
    }

    private static void generatePermutation(List<Integer> numbers, int k, int[] factorial, StringBuilder result) {
        if (numbers.isEmpty()) {
            return;
        }

        int n = numbers.size();
        int index = k / factorial[n - 1];
        result.append(numbers.remove(index));

        k %= factorial[n - 1];
```

```
        generatePermutation(numbers, k, factorial, result);
    }
}
```

Output:-

```
PS C:\Users\Thakur Sahab\Desktop\java> javac PermutationSequence.java
PS C:\Users\Thakur Sahab\Desktop\java> java PermutationSequence
231
```

136. Single Number

```
public class SingleNumber {

    public static void main(String[] args) {
        int[] nums = {4, 2, 1, 2, 1};

        int result = singleNumber(nums);
        System.out.println(result); // Output: 4
    }

    public static int singleNumber(int[] nums) {
        int result = 0;

        // XOR all elements in the array
        for (int num : nums) {
            result ^= num;
```

```
        }


        return result;
    }
}
```

Output:-

```
PS C:\Users\Thakur Sahab\Desktop\java> javac SingleNumber.java
PS C:\Users\Thakur Sahab\Desktop\java> java SingleNumber
4
```

22. Generate Parentheses

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class GenerateParentheses {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the value of n: ");
        int n = scanner.nextInt();

        List<String> result = generateParenthesis(n);
        System.out.println(result);
```

```
        scanner.close();
    }

    public static List<String> generateParenthesis(int n) {
        List<String> result = new ArrayList<>();
        generateParenthesisHelper(n, n, "", result);
        return result;
    }

    private static void generateParenthesisHelper(int left, int right, String current, List<String> result) {
        if (left == 0 && right == 0) {
            result.add(current);
            return;
        }

        if (left > 0) {
            generateParenthesisHelper(left - 1, right, current + "(", result);
        }

        if (right > left) {
            generateParenthesisHelper(left, right - 1, current + ")", result);
        }
    }
}
```

Output

PS C:\Users\Thakur Sahab\Desktop\java> javac GenerateParentheses.java

PS C:\Users\Thakur Sahab\Desktop\java> java GenerateParentheses

Enter the value of n: 6

[(((((()))))), ((((()())))), ((((())()))), ((((()))())), ((((())))()), ((((()))))(), (((()(())))), (((()()()))), (((()())())), (((()()))()), (((()())))(), (((())(()))), (((())()())), (((())())()), (((())()))(), (((()))(())), (((()))()()), (((()))())(), (((())))(()), (((())))()(), ((()((())))), ((()(()()))), ((()(())())), ((()(()))()), ((()(())))(), ((()()(()))), ((()()()())), ((()()())()), ((()()()))(), ((()())(())), ((()())()()), ((()())())(), ((()()))(()), ((()()))()(), ((())((()))), ((())(()())), ((())(())()), ((())(()))(), ((())()(())), ((())()()()), ((())()())(), ((())())(()), ((())())()(), ((()))((())), ((()))(()()), ((()))(())(), ((()))()(()), ((()))()()(), (()(((())))), (()((()()))), (()((())())), (()((()))()), (()((())))(), (()(()(()))), (()(()()())), (()(()())()), (()(()()))(), (()(())(())), (()(())()()), (()(())())(), (()(()))(()), (()(()))()(), (()()((()))), (()()(()())), (()()(())()), (()()(()))(), (()()()(())), (()()()()()), (()()()())(), (()()())(()), (()()())()(), (()())((())), (()())(()()), (()())(())(), (()())()(()), (()())()()(), (())(((()))), (())((()())), (())((())()), (())((()))(), (())(()(())), (())(()()()), (())(()())(), (())(())(()), (())(())()(), (())()((())), (())()(()()), (())()(())(), (())()()(()), (())()()()(), ()((((())))), ()(((()()))), ()(((())())), ()(((()))()), ()(((())))(), ()((()(()))), ()((()()())), ()((()())()), ()((()()))(), ()((())(())), ()((())()()), ()((())())(), ()((()))(()), ()((()))()(), ()(()((()))), ()(()(()())), ()(()(())()), ()(()(()))(), ()(()()(())), ()(()()()()), ()(()()())(), ()(()())(()), ()(()())()(), ()(())((())), ()(())(()()), ()(())(())(), ()(())()(()), ()(())()()(), ()()(((()))), ()()((()())), ()()((())()), ()()((()))(), ()()(()(())), ()()(()()()), ()()(()())(), ()()(())(()), ()()(())()(), ()()()((())), ()()()(()()), ()()()(())(), ()()()()(()), ()()()()()()]

77. Combinations

```
import java.util.ArrayList;

import java.util.List;



public class Combinations {



   public static void main(String[] args) {

      int n = 4;

      int k = 2;



      List<List<Integer>> result = combine(n, k);

      System.out.println(result);
```

```
    }

    public static List<List<Integer>> combine(int n, int k) {
        List<List<Integer>> result = new ArrayList<>();
        combineHelper(n, k, 1, new ArrayList<>(), result);
        return result;
    }

    private static void combineHelper(int n, int k, int start, List<Integer> current, List<List<Integer>>
result) {
        if (k == 0) {
            result.add(new ArrayList<>(current));
            return;
        }

        for (int i = start; i <= n; i++) {
            current.add(i);
            combineHelper(n, k - 1, i + 1, current, result);
            current.remove(current.size() - 1);
        }
    }
}
```

Output

```
PS C:\Users\Thakur Sahab\Desktop\java> javac Combinations.java
PS C:\Users\Thakur Sahab\Desktop\java> java Combinations
[[1, 2], [1, 3], [1, 4], [2, 3], [2, 4], [3, 4]]
```

129. Sum Root to Leaf Numbers

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

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

public class SumRootToLeafNumbers {

    public static void main(String[] args) {
        // Example usage:
        // Construct a binary tree: 1 -> 2 -> 3
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);

        int result = sumNumbers(root);
        System.out.println(result); // Output: 25 (12 + 13)
    }

    public static int sumNumbers(TreeNode root) {
```

```
        return dfs(root, 0);
    }

    private static int dfs(TreeNode node, int currentSum) {
        if (node == null) {
            return 0;
        }

        currentSum = currentSum * 10 + node.val;

        // If it's a leaf node, return the sum
        if (node.left == null && node.right == null) {
            return currentSum;
        }

        // Recursive calls for left and right children
        int leftSum = dfs(node.left, currentSum);
        int rightSum = dfs(node.right, currentSum);

        return leftSum + rightSum;
    }
}
```

Output:

```
PS C:\Users\Thakur Sahab\Desktop\java> javac SumRootToLeafNumbers.java
PS C:\Users\Thakur Sahab\Desktop\java> java SumRootToLeafNumbers
```

150. Evaluate Reverse Polish Notation

```
import java.util.Stack;

public class EvaluateReversePolishNotation {

    public static void main(String[] args) {
        // Example usage:
        String[] tokens = {"2", "1", "+", "3", "*"};
        int result = evalRPN(tokens);
        System.out.println(result); // Output: 9
    }

    public static int evalRPN(String[] tokens) {
        Stack<Integer> stack = new Stack<>();

        for (String token : tokens) {
            if (isOperator(token)) {
                // Pop the top two operands and perform the operation
                int operand2 = stack.pop();
                int operand1 = stack.pop();
                int result = performOperation(operand1, operand2, token);
                stack.push(result);
            } else {
```

```
            // Convert the token to an integer and push to the stack
            stack.push(Integer.parseInt(token));
        }
    }

    return stack.pop();
}

private static boolean isOperator(String token) {
    return token.equals("+") || token.equals("-") || token.equals("*") || token.equals("/");
}

private static int performOperation(int operand1, int operand2, String operator) {
    switch (operator) {
        case "+":
            return operand1 + operand2;
        case "-":
            return operand1 - operand2;
        case "*":
            return operand1 * operand2;
        case "/":
            return operand1 / operand2; // Note: Integer division truncates toward zero
        default:
            throw new IllegalArgumentException("Invalid operator");
    }
```

```
    }
}
```

Output:-

```
PS C:\Users\Thakur Sahab\Desktop\java> javac EvaluateReversePolishNotation.java
PS C:\Users\Thakur Sahab\Desktop\java> java EvaluateReversePolishNotation
9
```

139. Word Break

```
import java.util.List;

public class WordBreak {

    public static void main(String[] args) {
        // Example usage:
        String s = "leetcode";
        List<String> wordDict = List.of("leet", "code");
        boolean result = wordBreak(s, wordDict);
        System.out.println(result); // Output: true
    }

    public static boolean wordBreak(String s, List<String> wordDict) {
        int n = s.length();
        boolean[] dp = new boolean[n + 1];
        dp[0] = true;

        for (int i = 1; i <= n; i++) {
```

```
        for (String word : wordDict) {
            int len = word.length();
            if (i >= len && dp[i - len] && s.substring(i - len, i).equals(word)) {
                dp[i] = true;
                break;
            }
        }
    }

    return dp[n];
  }
}
```

Output

```
PS C:\Users\Thakur Sahab\Desktop\java> javac WordBreak.java
PS C:\Users\Thakur Sahab\Desktop\java> java WordBreak
True
```

193. Valid Phone Numbers

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class ValidPhoneNumbers {

  public static void main(String[] args) {
```

```
        String[] phoneNumbers = {
            "(123) 456-7890",
            "456-789-0123",
            "123-45-6789",
            "invalid-phone-number"
        };

        for (String phoneNumber : phoneNumbers) {
            if (isValidPhoneNumber(phoneNumber.trim())) {
                System.out.println(phoneNumber + " is a valid phone number.");
            } else {
                System.out.println(phoneNumber + " is not a valid phone number.");
            }
        }
    }

    public static boolean isValidPhoneNumber(String phoneNumber) {
        String regex = "^\\((\\d{3})\\) (\\d{3}-\\d{4})$|^\\d{3}-\\d{3}-\\d{4}$";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(phoneNumber);
        return matcher.matches();
    }
}
```

Output

```
PS C:\Users\Thakur Sahab\Desktop\java> javac ValidPhoneNumbers.java
```

```
PS C:\Users\Thakur Sahab\Desktop\java> java ValidPhoneNumbers
(123) 456-7890 is a valid phone number.
456-789-0123 is a valid phone number.
123-45-6789 is not a valid phone number.
invalid-phone-number is not a valid phone number.
```

199. Binary Tree Right Side View

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

class TreeNode {
   int val;
   TreeNode left;
   TreeNode right;

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

public class BinaryTreeRightSideView {

   public static void main(String[] args) {
      // Example usage:
```

```
    // Construct a binary tree: 1 -> 2 -> 3 -> 4
    TreeNode root = new TreeNode(1);
    root.left = new TreeNode(2);
    root.right = new TreeNode(3);
    root.left.right = new TreeNode(4);

    List<Integer> result = rightSideView(root);
    System.out.println(result); // Output: [1, 3, 4]
}

public static List<Integer> rightSideView(TreeNode root) {
    List<Integer> result = new ArrayList<>();

    if (root == null) {
        return result;
    }

    Queue<TreeNode> queue = new LinkedList<>();
    queue.offer(root);

    while (!queue.isEmpty()) {
        int size = queue.size();

        for (int i = 0; i < size; i++) {
            TreeNode current = queue.poll();
```

```
            // Record the rightmost node at each level
            if (i == size - 1) {
                result.add(current.val);
            }

            if (current.left != null) {
                queue.offer(current.left);
            }

            if (current.right != null) {
                queue.offer(current.right);
            }
        }
    }

    return result;
  }
}
```

Output

```
PS C:\Users\Thakur Sahab\Desktop\java> javac BinaryTreeRightSideView.java
PS C:\Users\Thakur Sahab\Desktop\java> java BinaryTreeRightSideView
[1, 3, 4]
```