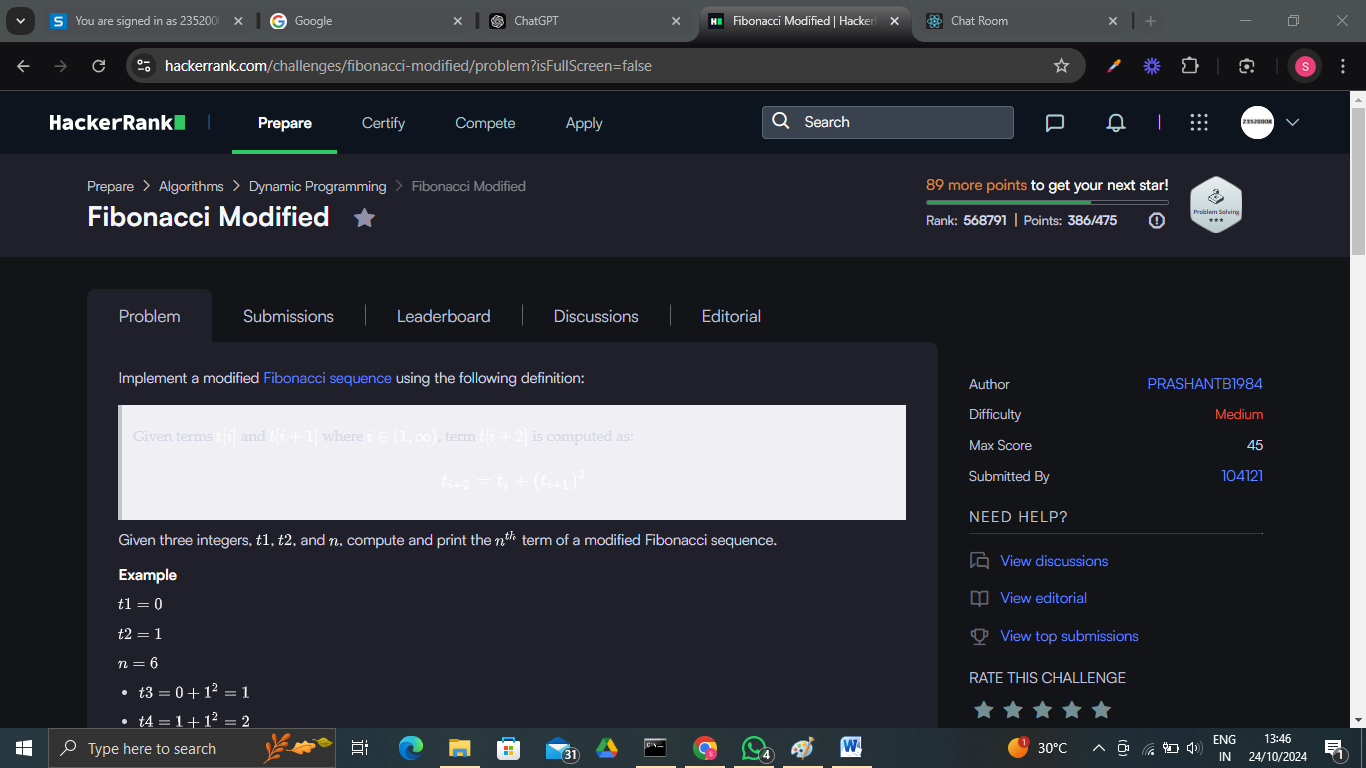
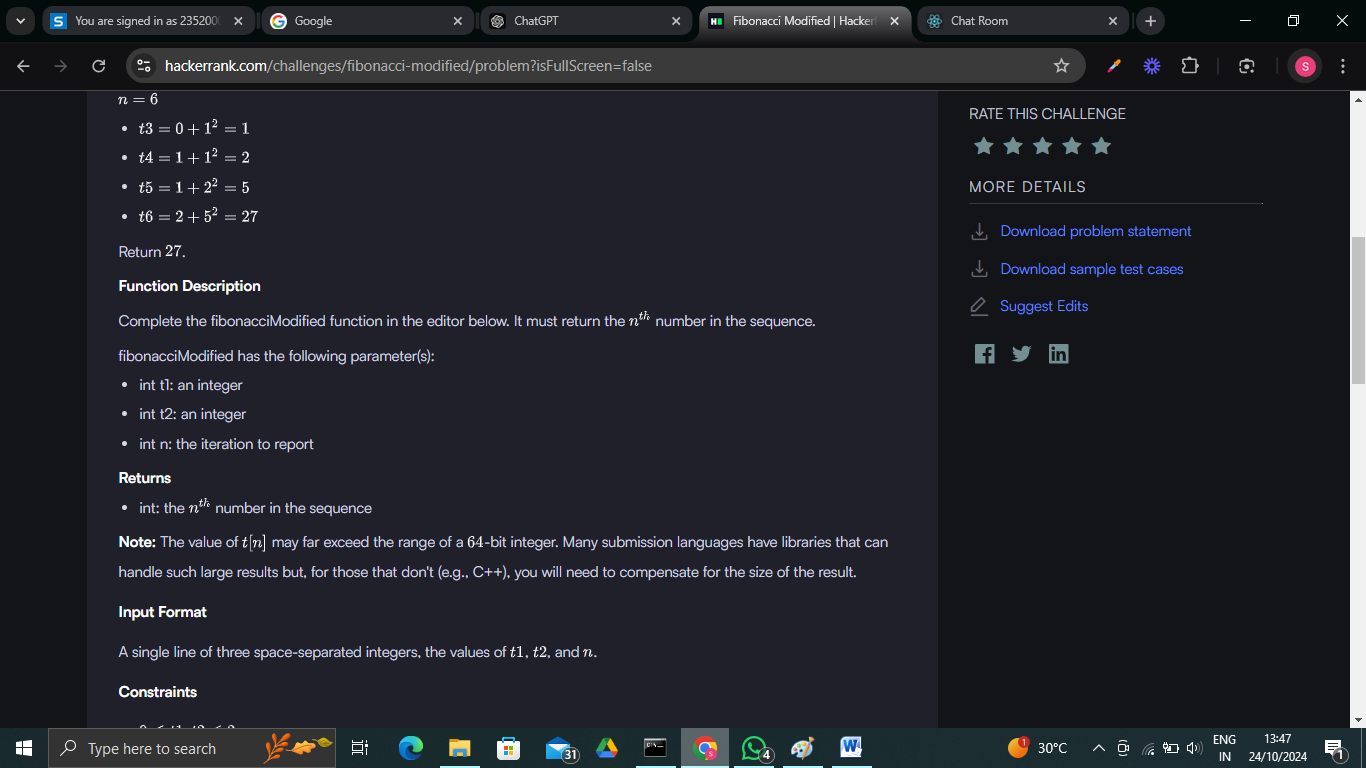
**Design and Analysis of Algorithm**

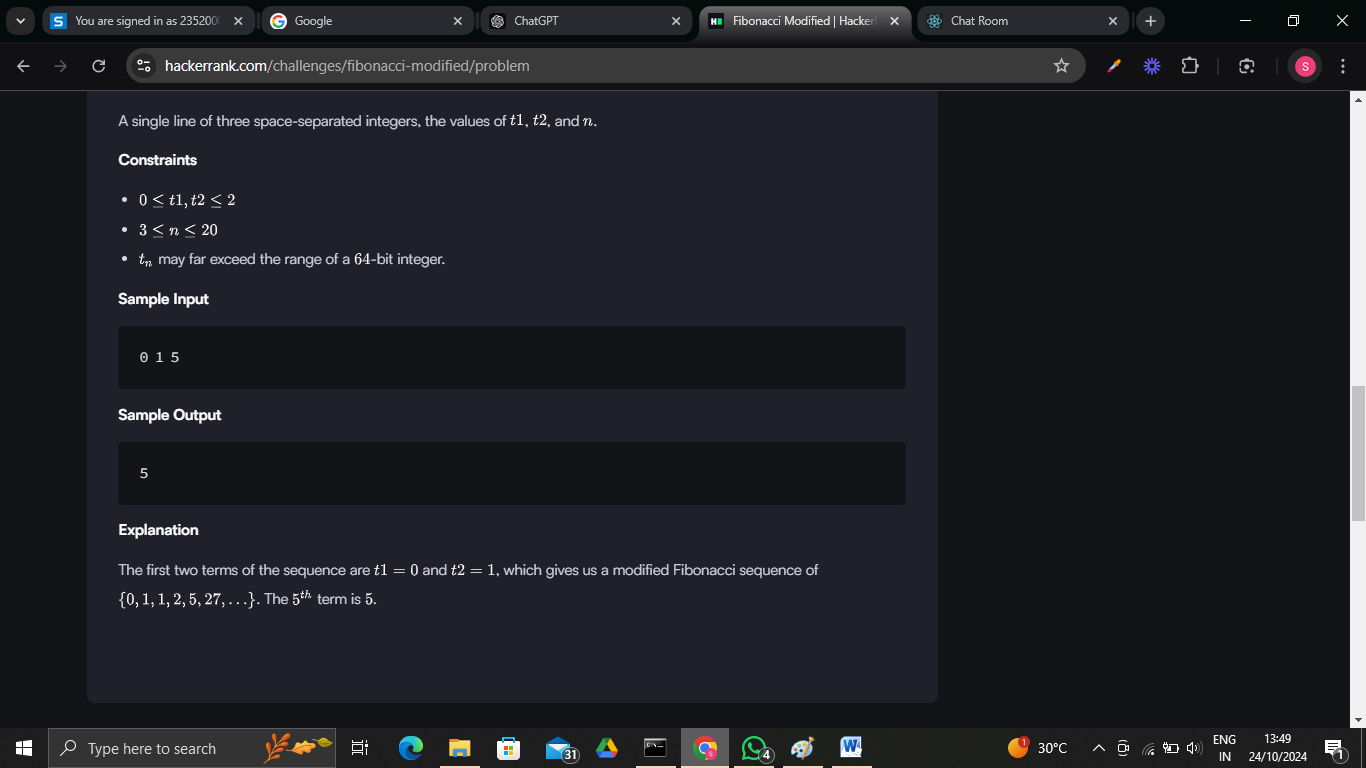
**ISE 2**

**PRN: 23520008**

**Problem 1:**







Code:

import java.io.\*;

import java.math.BigInteger;

class Result {

*/\**

*\* Complete the 'fibonacciModified' function below.*

*\**

*\* The function is expected to return a BigInteger.*

*\* The function accepts following parameters:*

*\*  1. INTEGER t1*

*\*  2. INTEGER t2*

*\*  3. INTEGER n*

*\*/*

    public static BigInteger fibonacciModified(int t1, int t2, int n) {

*// Initialize the first two terms as BigInteger*

        BigInteger first = BigInteger.valueOf(t1);

        BigInteger second = BigInteger.valueOf(t2);

*// If n == 1 or n == 2, return t1 or t2 respectively*

        if (n == 1) return first;

        if (n == 2) return second;

*// Variable to store the current term*

        BigInteger current = BigInteger.ZERO;

*// Calculate the sequence up to the nth term*

        for (int i = 3; i <= n; i++) {

*// Current term: t\_n = t\_(n-2) + (t\_(n-1))^2*

            current = first.add(second.multiply(second));

*// Move to the next term in sequence*

            first = second;

            second = current;

        }

        return current;

    }

}

public class Solution {

    public static void main(String[] args) throws IOException {

        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

        String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

        int t1 = Integer.parseInt(firstMultipleInput[0]);

        int t2 = Integer.parseInt(firstMultipleInput[1]);

        int n = Integer.parseInt(firstMultipleInput[2]);

        BigInteger result = Result.fibonacciModified(t1, t2, n);

        bufferedWriter.write(result.toString());

        bufferedWriter.newLine();

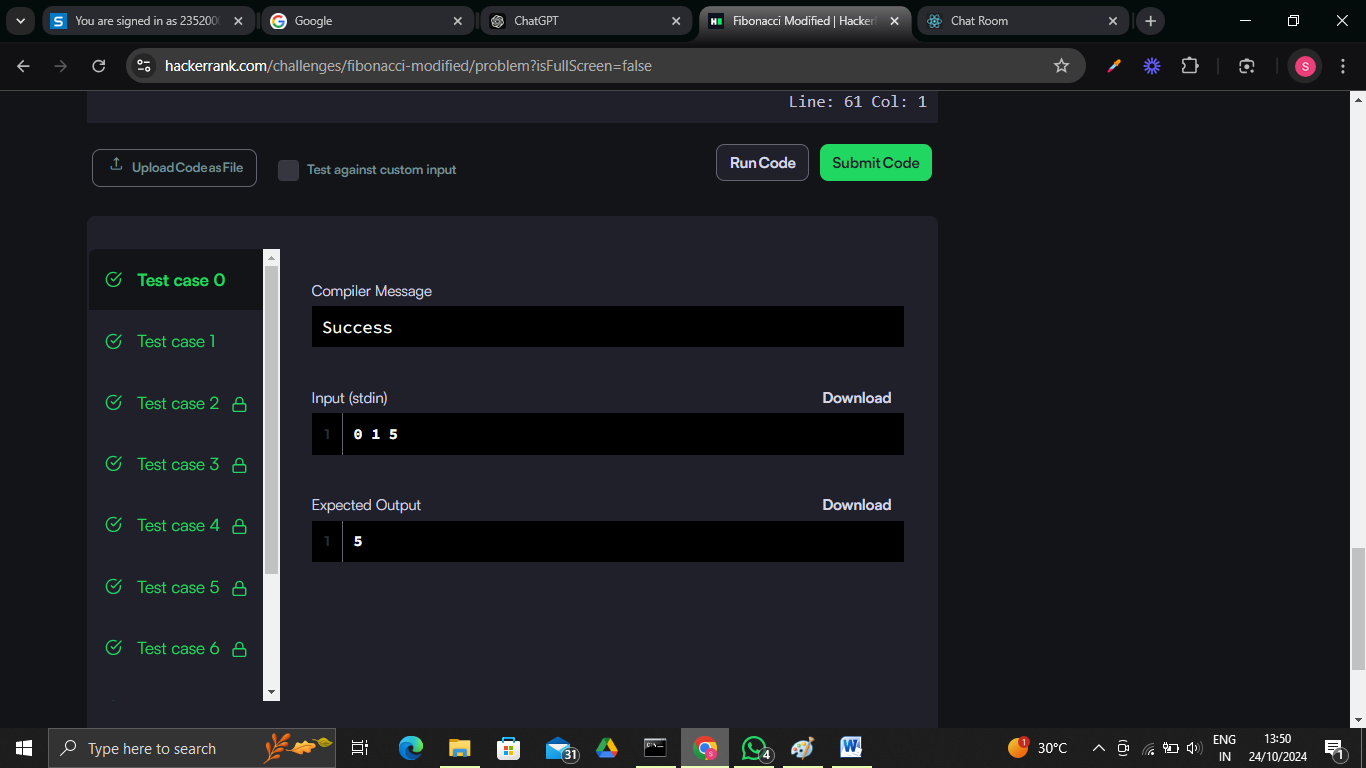
        bufferedReader.close();

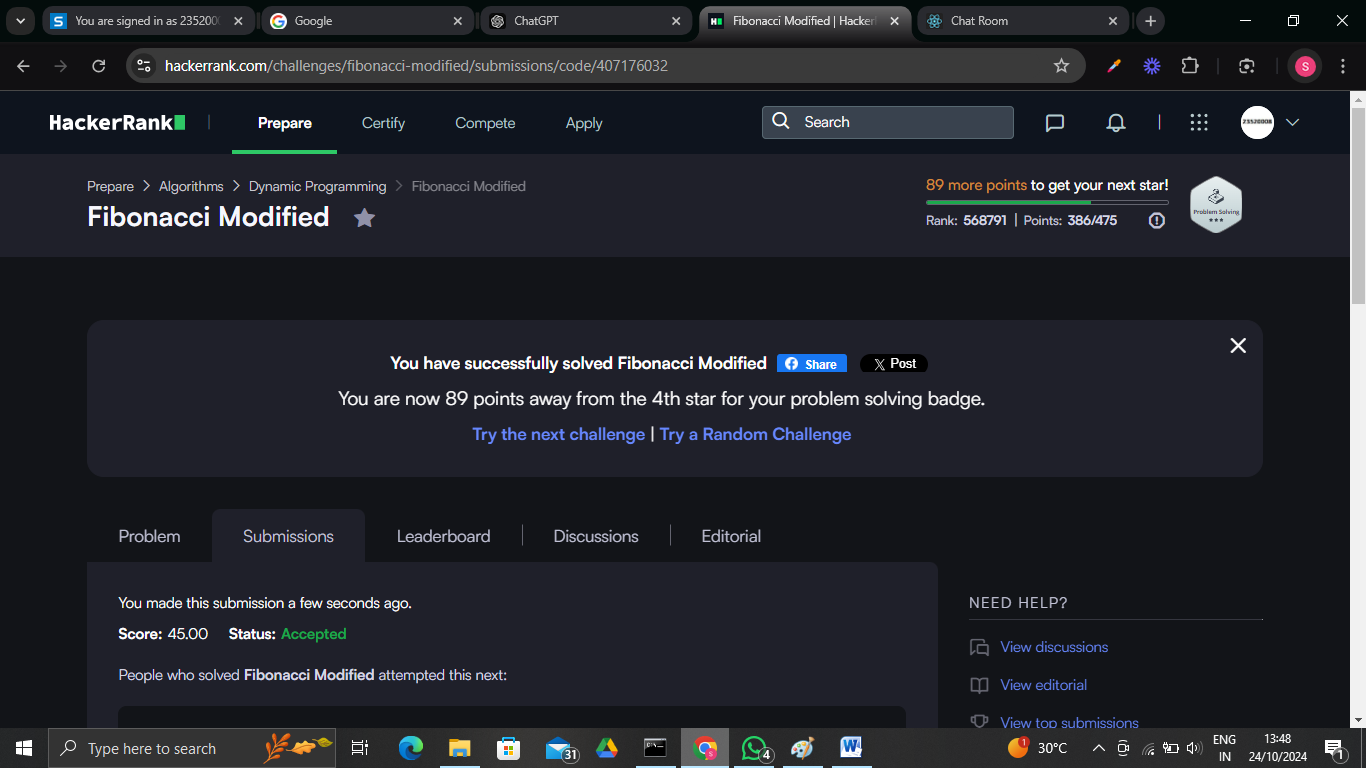
        bufferedWriter.close();

    }

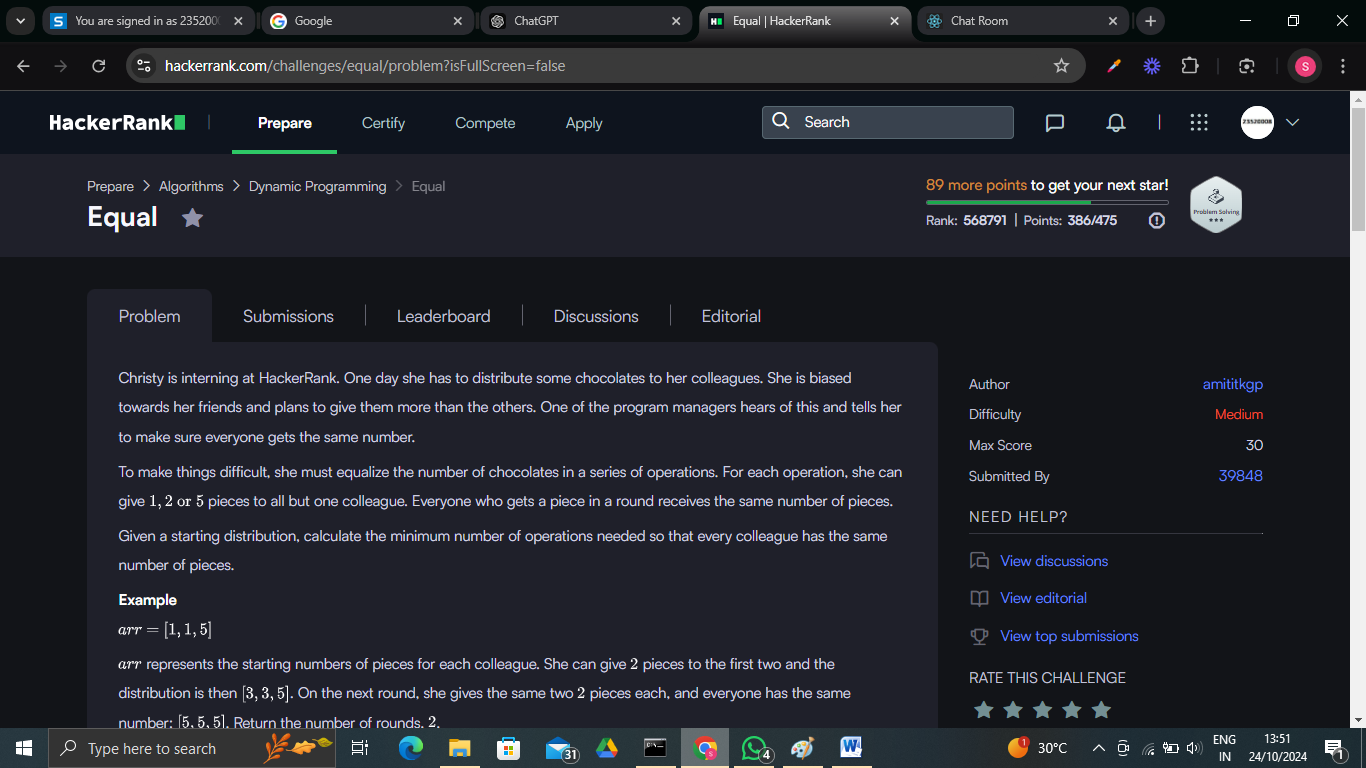
}

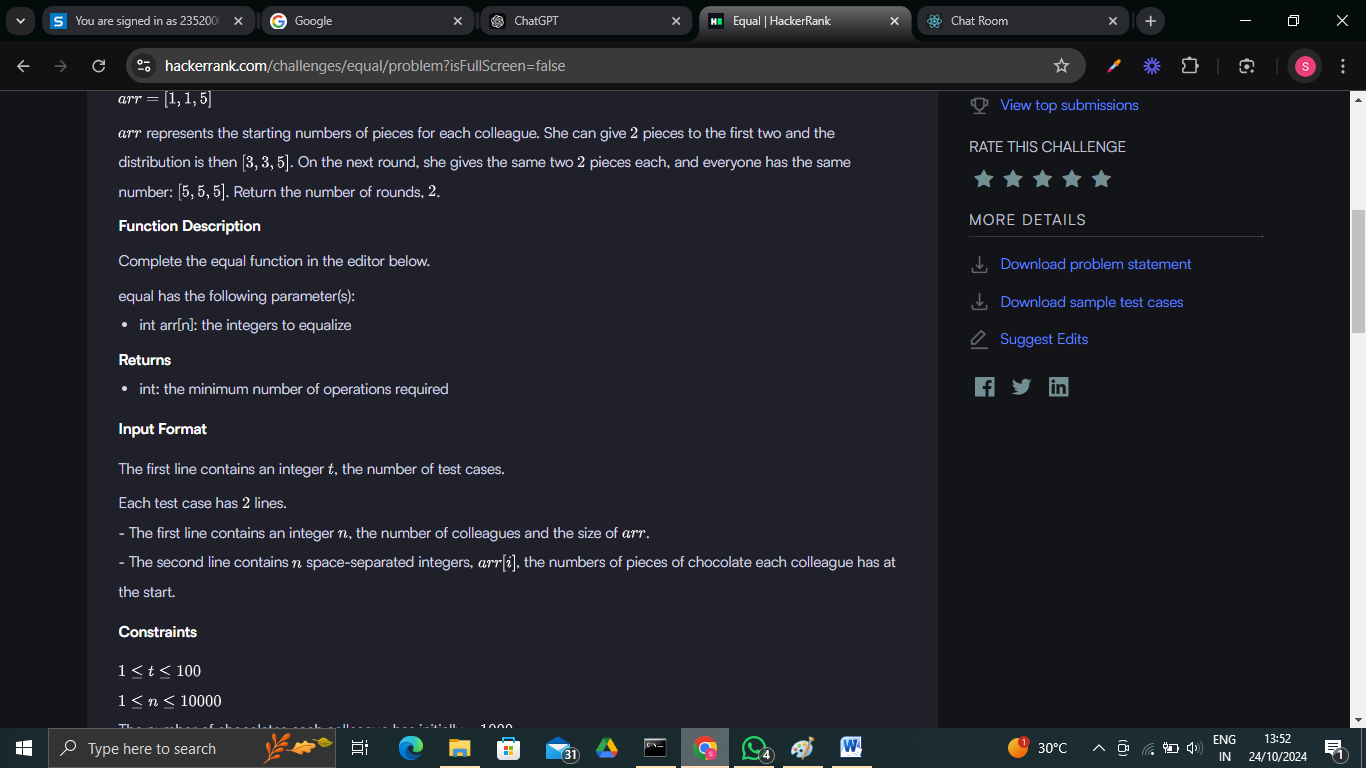
Output:

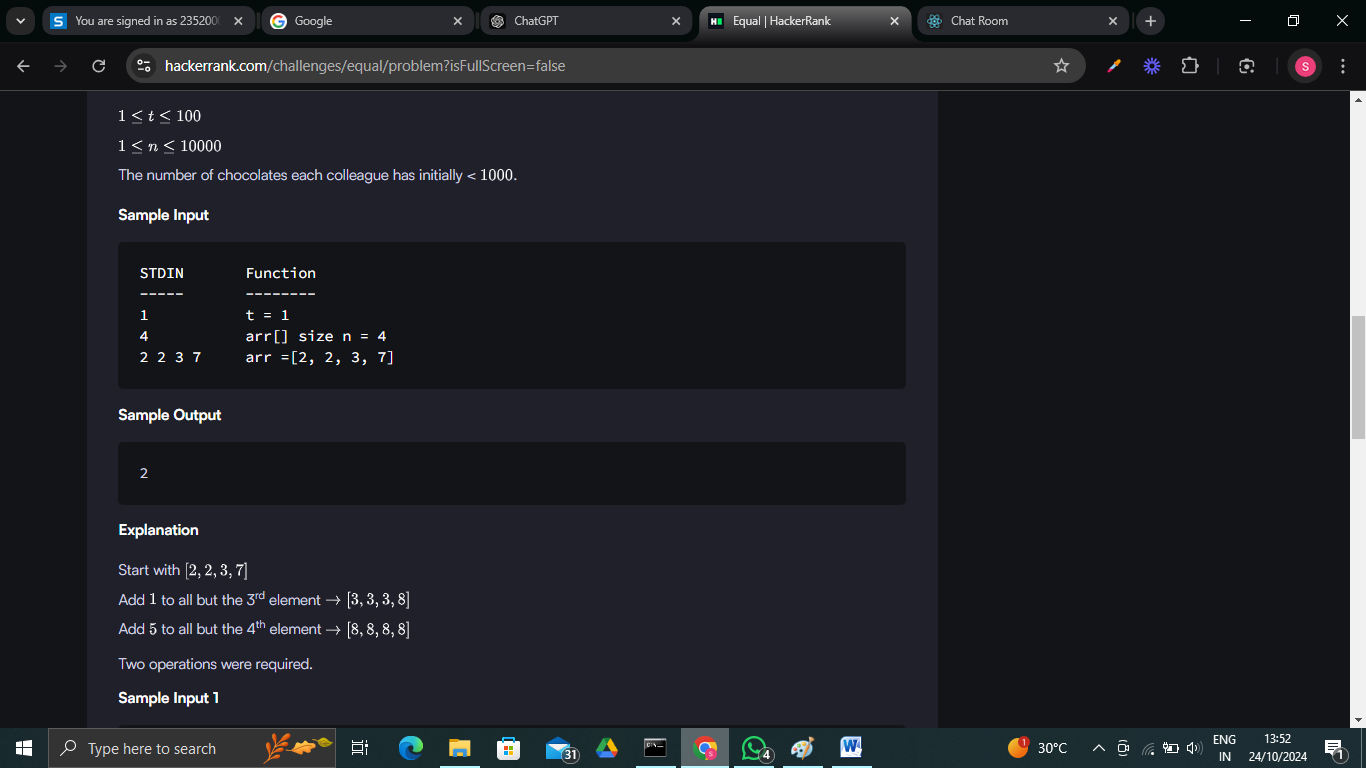




**Problem 2:**







**Code:**

import java.io.\*;

import java.util.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.toList;

class Result {

*/\**

*\* Complete the 'equal' function below.*

*\**

*\* The function is expected to return an INTEGER.*

*\* The function accepts INTEGER\_ARRAY arr as parameter.*

*\*/*

    public static int equal(List<Integer> arr) {

        int minChocolates = Collections.min(arr); *// Find the minimum value in the list*

        int minOperations = Integer.MAX\_VALUE;

*// Try normalizing to minChocolates, minChocolates - 1, and minChocolates - 2*

        for (int base = 0; base <= 2; base++) {

            int currentOperations = 0;

            for (int chocolates : arr) {

                int delta = chocolates - (minChocolates - base); *// Difference from the current base level*

                currentOperations += delta / 5; *// Add operations for reducing by 5*

                delta %= 5;

                currentOperations += delta / 2; *// Add operations for reducing by 2*

                delta %= 2;

                currentOperations += delta;     *// Add operations for reducing by 1*

            }

*// Track the minimum number of operations required*

            minOperations = Math.min(minOperations, currentOperations);

        }

        return minOperations;

    }

}

public class Solution {

    public static void main(String[] args) throws IOException {

        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

        int t = Integer.parseInt(bufferedReader.readLine().trim());

        IntStream.range(0, t).forEach(tItr -> {

            try {

                int n = Integer.parseInt(bufferedReader.readLine().trim());

                List<Integer> arr = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

                    .map(Integer::parseInt)

                    .collect(toList());

                int result = Result.equal(arr);

                bufferedWriter.write(String.valueOf(result));

                bufferedWriter.newLine();

            } catch (IOException ex) {

                throw new RuntimeException(ex);

            }

        });

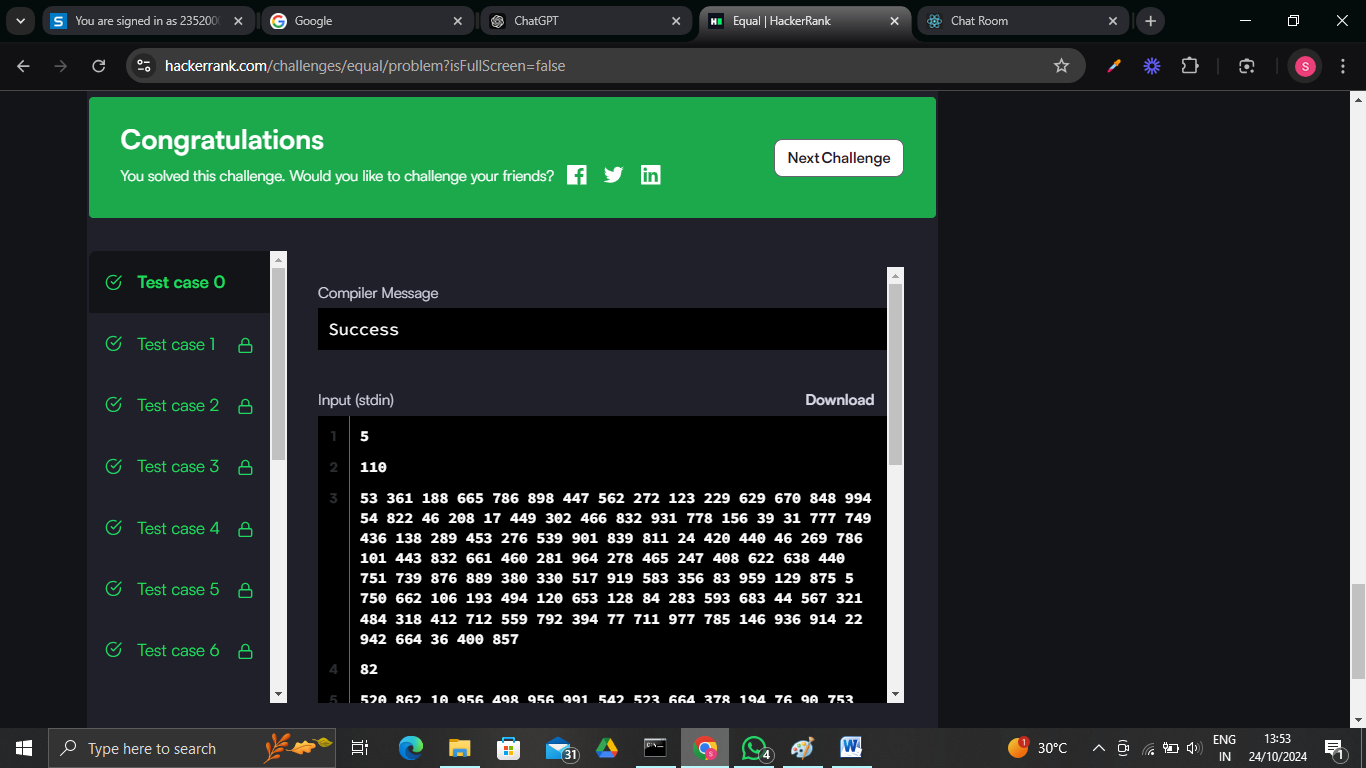
        bufferedReader.close();

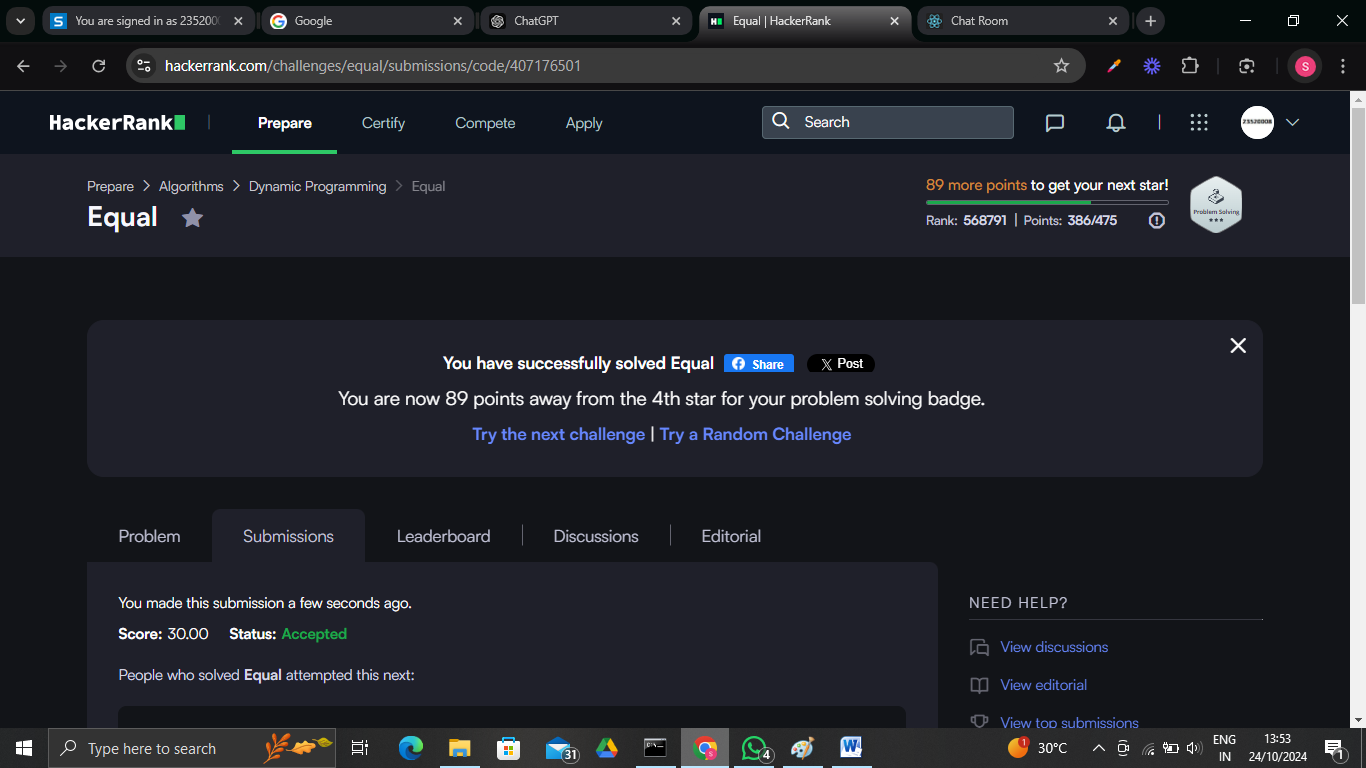
        bufferedWriter.close();

    }

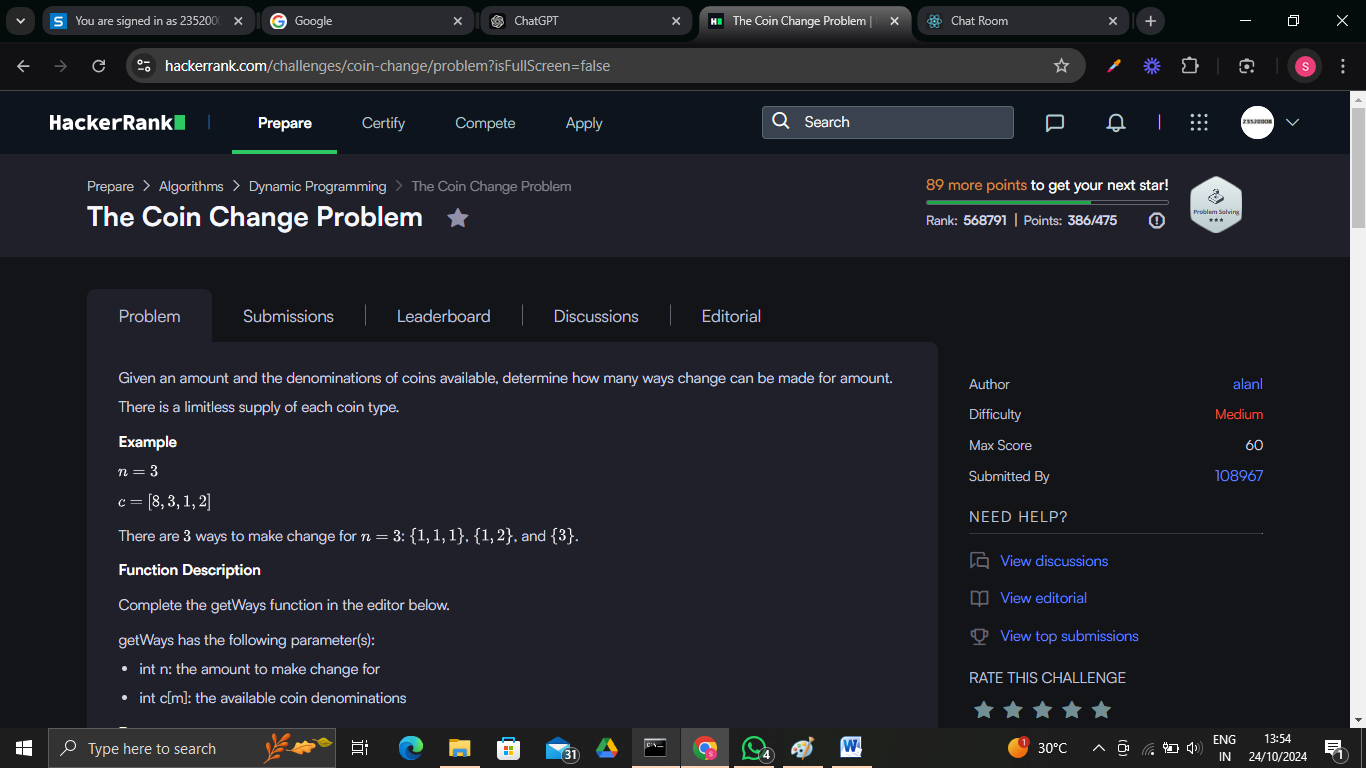
}

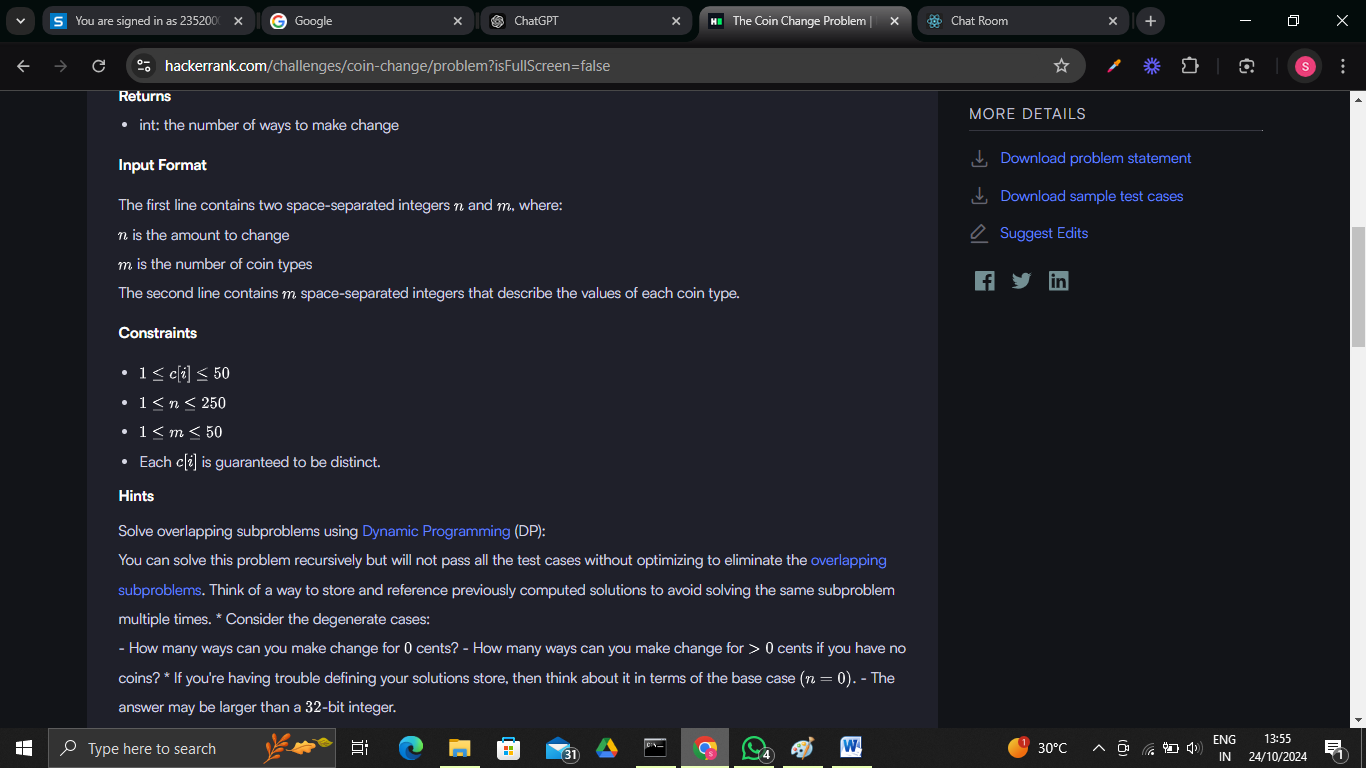
**Output:**





**Problem 3:**





**Code:**

import java.io.\*;

import java.util.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.toList;

class Result {

*/\**

*\* Complete the 'getWays' function below.*

*\**

*\* The function is expected to return a LONG\_INTEGER.*

*\* The function accepts following parameters:*

*\*  1. INTEGER n*

*\*  2. LONG\_INTEGER\_ARRAY c*

*\*/*

    public static long getWays(int n, List<Long> c) {

        long[] dp = new long[n + 1]; *// DP array to store the number of ways to make change*

        dp[0] = 1; *// There's one way to make 0 amount (use no coins)*

*// Iterate over each coin*

        for (long coin : c) {

*// Update dp array for all amounts that can include this coin*

            for (int j = (int) coin; j <= n; j++) {

                dp[j] += dp[j - (int) coin]; *// Update the number of ways to make amount j*

            }

        }

        return dp[n]; *// The answer is the number of ways to make amount n*

    }

}

public class Solution {

    public static void main(String[] args) throws IOException {

        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

        String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

        int n = Integer.parseInt(firstMultipleInput[0]); *// Amount to change*

        int m = Integer.parseInt(firstMultipleInput[1]); *// Number of coin types*

        List<Long> c = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

            .map(Long::parseLong)

            .collect(toList());

*// Print the number of ways of making change for 'n' units using coins having the values given by 'c'*

        long ways = Result.getWays(n, c);

        bufferedWriter.write(String.valueOf(ways));

        bufferedWriter.newLine();

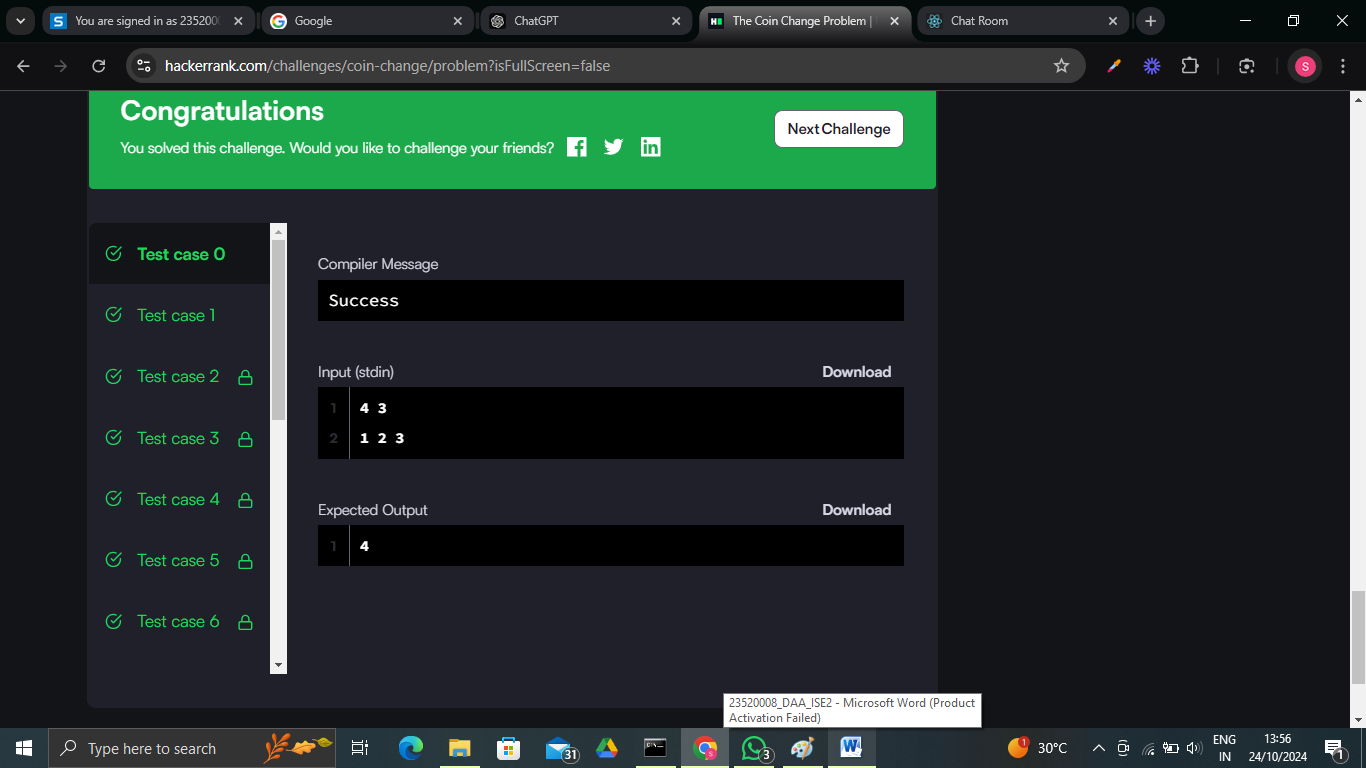
        bufferedReader.close();

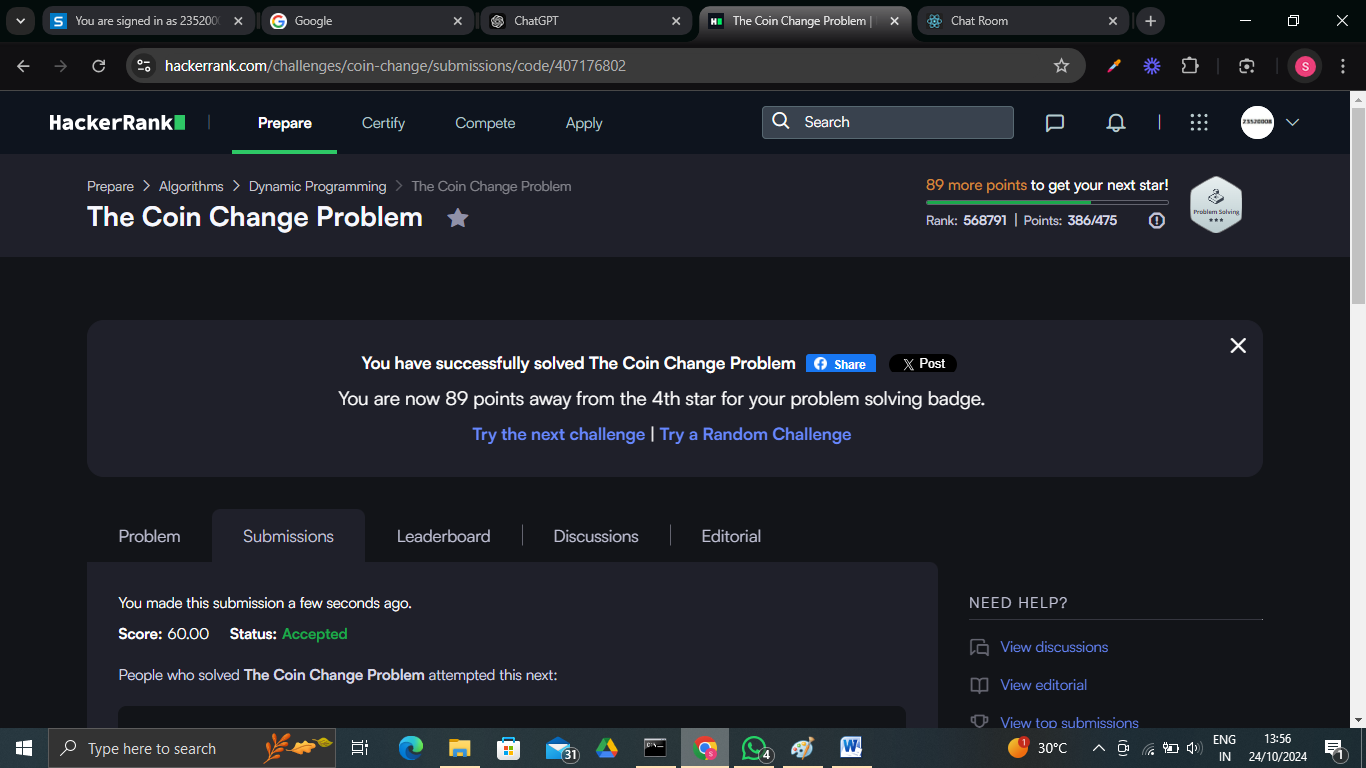
        bufferedWriter.close();

    }

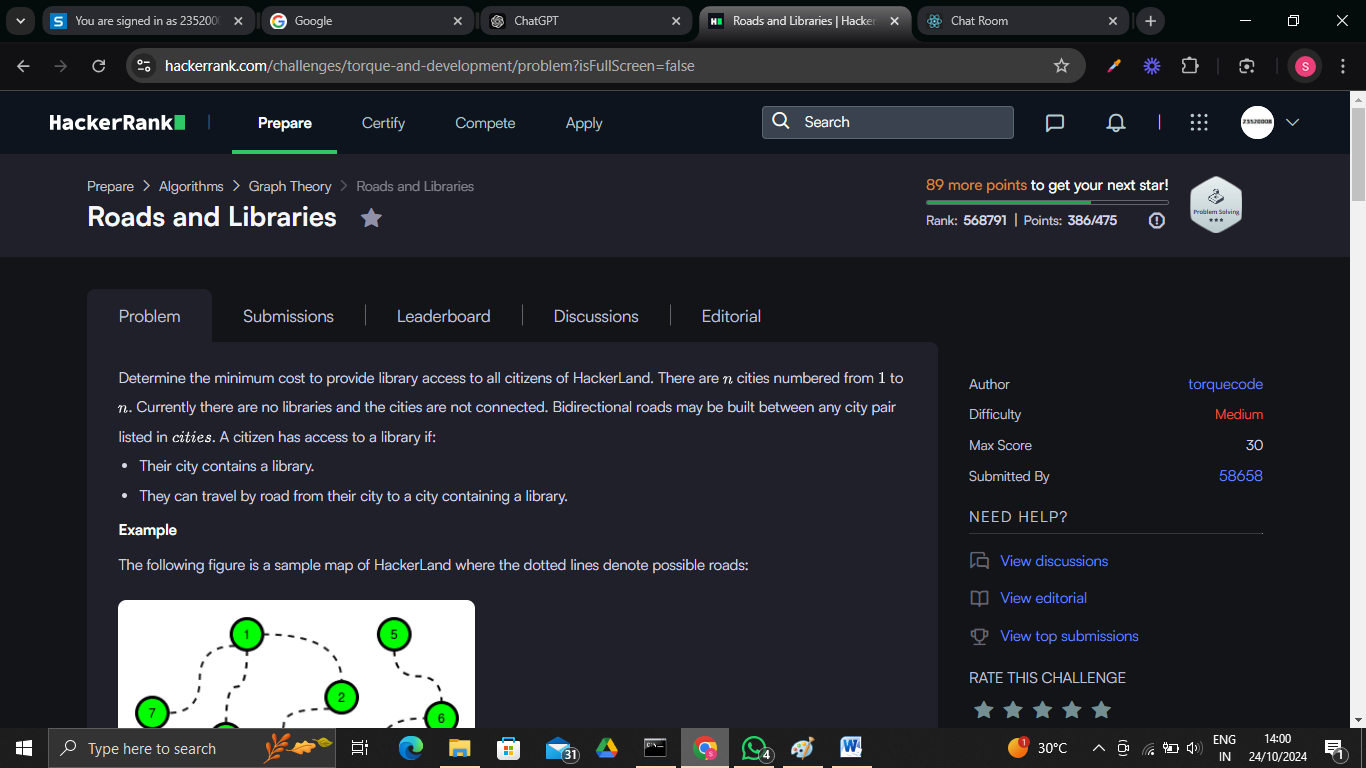
}

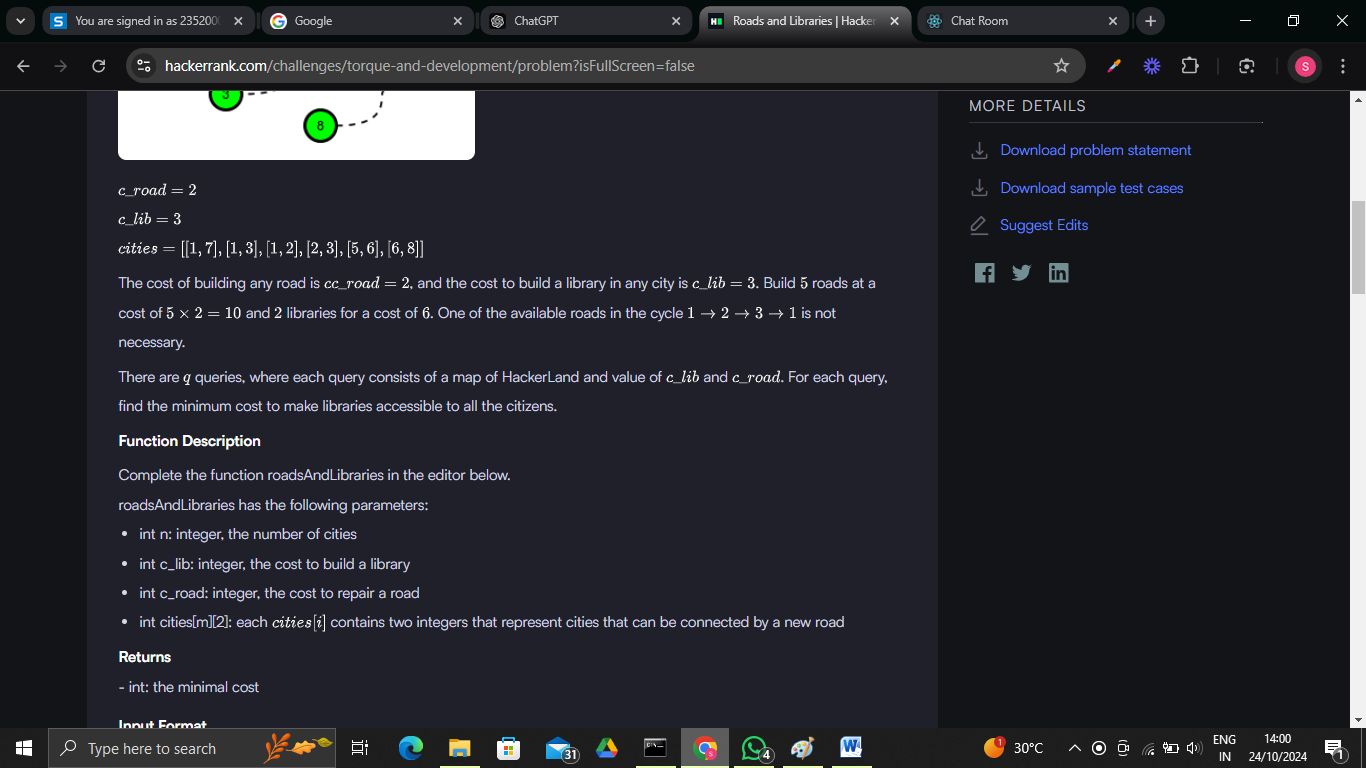
**Output:**

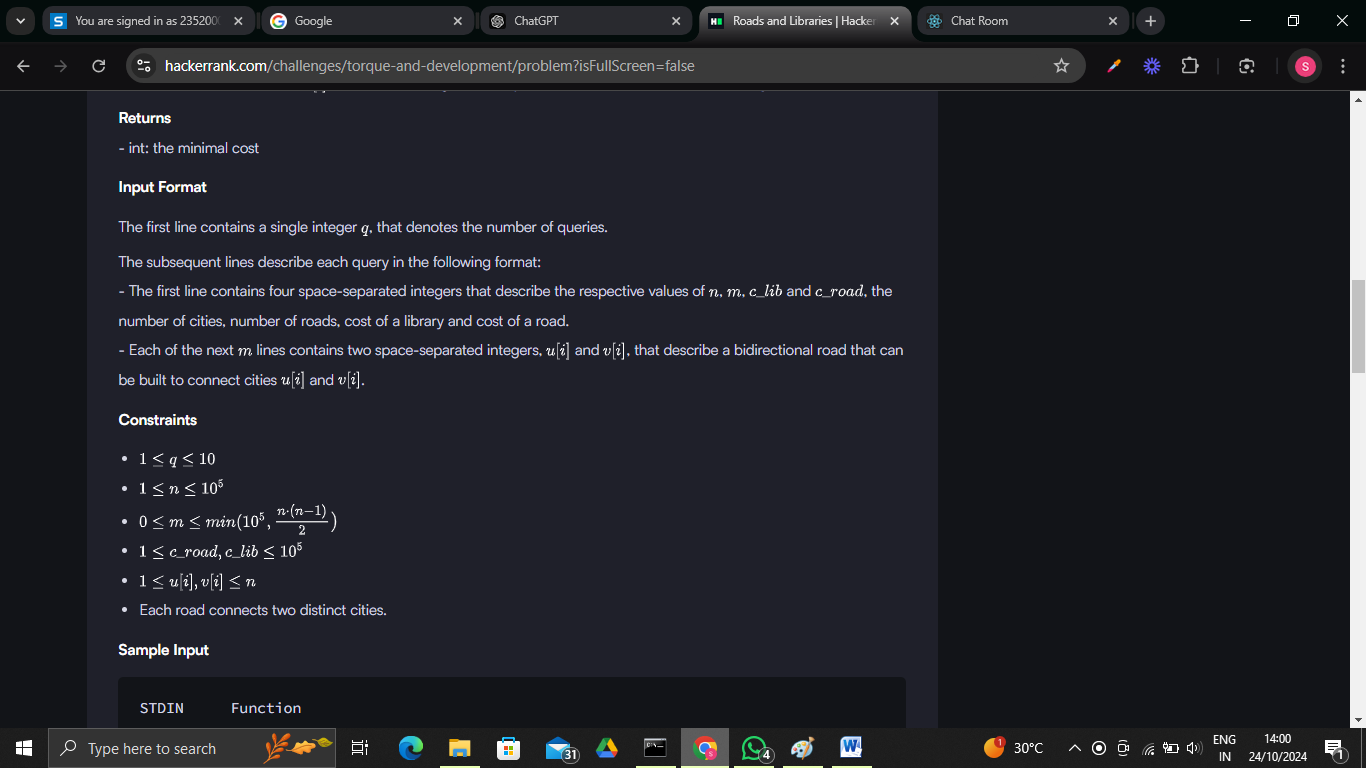




**Problem 4:**







**Code:**

import java.io.\*;

import java.util.\*;

class Result {

    public static long roadsAndLibraries(int n, int c\_lib, int c\_road, List<List<Integer>> cities) {

*// If it's cheaper to build libraries than roads, build a library in each city.*

        if (c\_lib <= c\_road) {

            return (long) n \* c\_lib;

        }

*// Build adjacency list for the cities graph.*

        List<List<Integer>> adj = new ArrayList<>();

        for (int i = 0; i < n; i++) {

            adj.add(new ArrayList<>());

        }

        for (List<Integer> cityPair : cities) {

            int city1 = cityPair.get(0) - 1;

            int city2 = cityPair.get(1) - 1;

            adj.get(city1).add(city2);

            adj.get(city2).add(city1);

        }

*// To keep track of visited cities.*

        boolean[] visited = new boolean[n];

        long totalCost = 0;

*// Find all connected components using DFS.*

        for (int i = 0; i < n; i++) {

            if (!visited[i]) {

*// For each new connected component, calculate the cost.*

                int cityCount = dfs(i, adj, visited);

*// One library for the component and roads for the rest cities in the component.*

                totalCost += c\_lib + (cityCount - 1) \* (long) c\_road;

            }

        }

        return totalCost;

    }

*// DFS to find the size of each connected component.*

    private static int dfs(int city, List<List<Integer>> adj, boolean[] visited) {

        Stack<Integer> stack = new Stack<>();

        stack.push(city);

        visited[city] = true;

        int count = 1;

        while (!stack.isEmpty()) {

            int currCity = stack.pop();

            for (int neighbor : adj.get(currCity)) {

                if (!visited[neighbor]) {

                    visited[neighbor] = true;

                    stack.push(neighbor);

                    count++;

                }

            }

        }

        return count;

    }

}

public class Solution {

    public static void main(String[] args) throws IOException {

        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

        int q = Integer.parseInt(bufferedReader.readLine().trim());

        for (int qItr = 0; qItr < q; qItr++) {

            String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

            int n = Integer.parseInt(firstMultipleInput[0]);

            int m = Integer.parseInt(firstMultipleInput[1]);

            int c\_lib = Integer.parseInt(firstMultipleInput[2]);

            int c\_road = Integer.parseInt(firstMultipleInput[3]);

            List<List<Integer>> cities = new ArrayList<>();

            for (int i = 0; i < m; i++) {

                List<Integer> citiesRow = new ArrayList<>();

                String[] citiesRowTempItems = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

                for (int j = 0; j < 2; j++) {

                    int citiesItem = Integer.parseInt(citiesRowTempItems[j]);

                    citiesRow.add(citiesItem);

                }

                cities.add(citiesRow);

            }

            long result = Result.roadsAndLibraries(n, c\_lib, c\_road, cities);

            bufferedWriter.write(String.valueOf(result));

            bufferedWriter.newLine();

        }

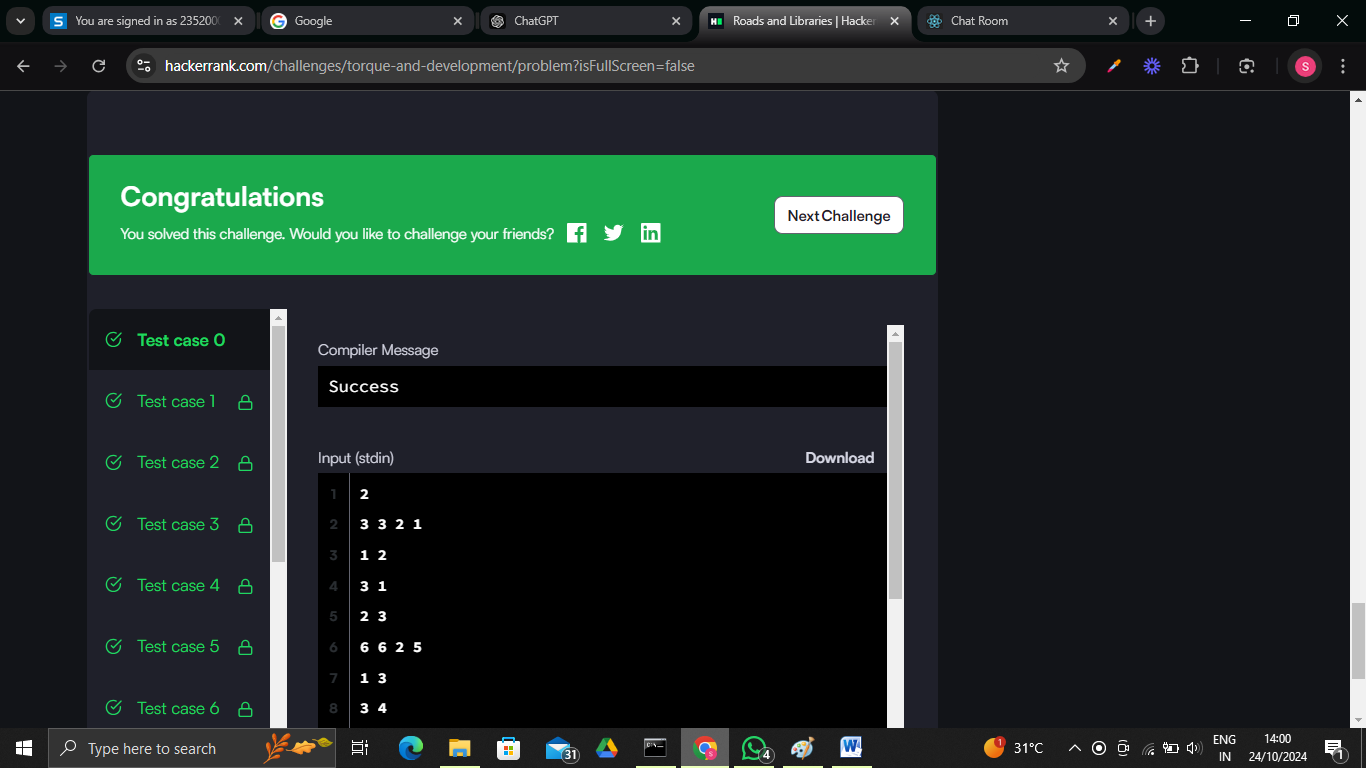
        bufferedReader.close();

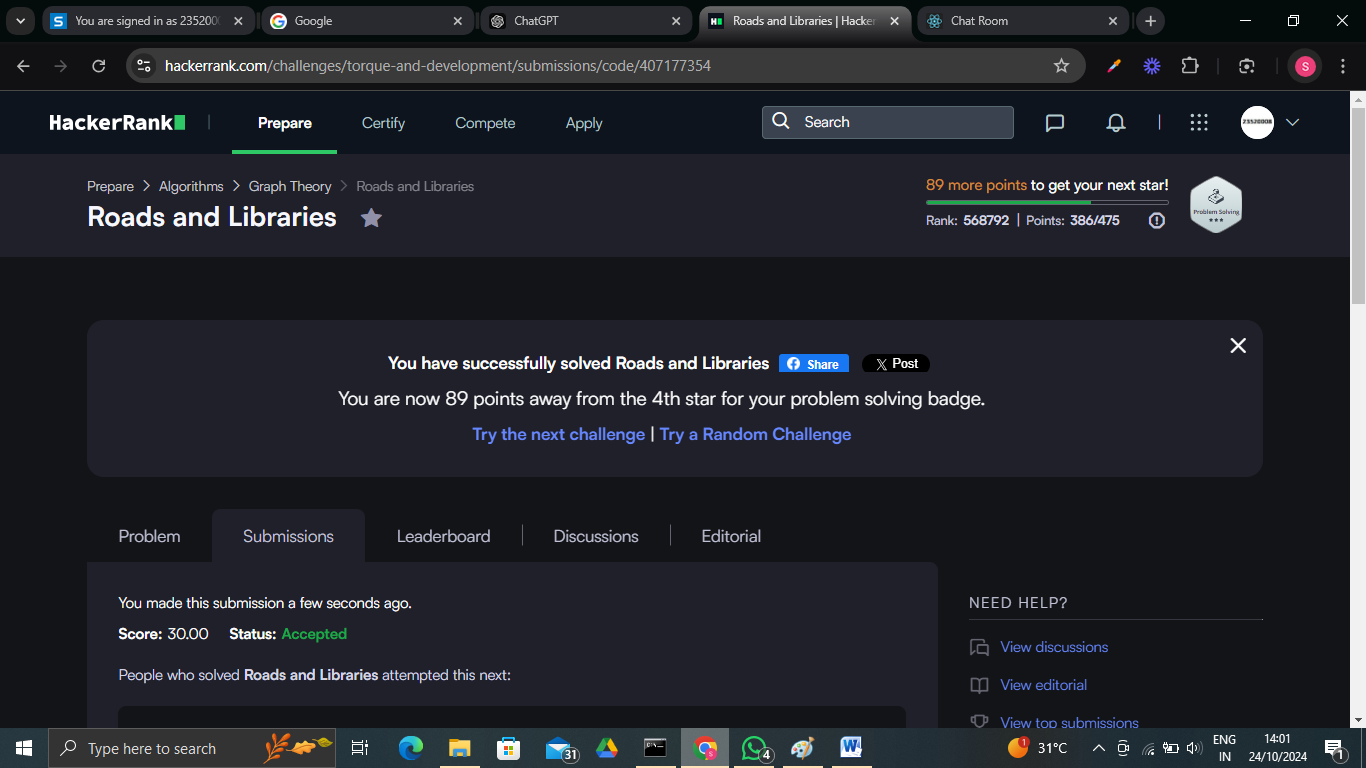
        bufferedWriter.close();

    }

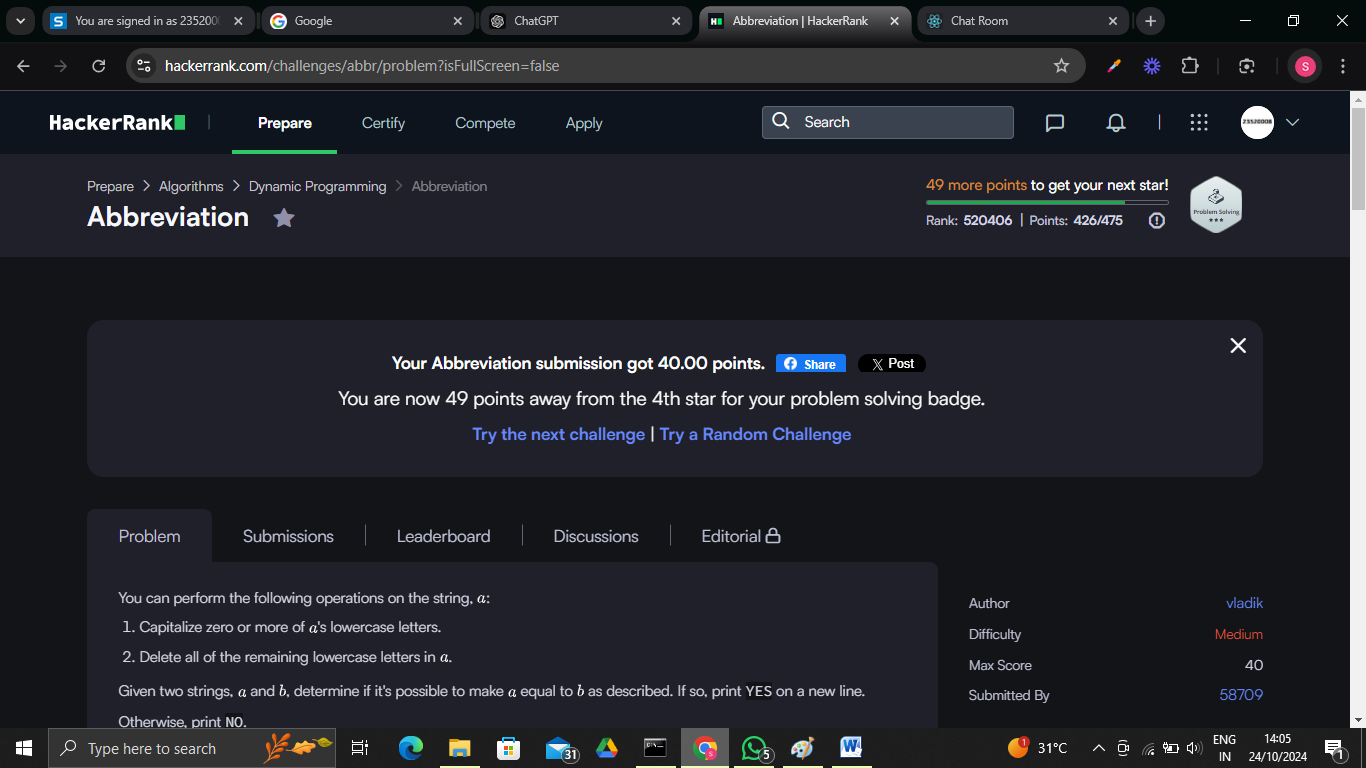
}

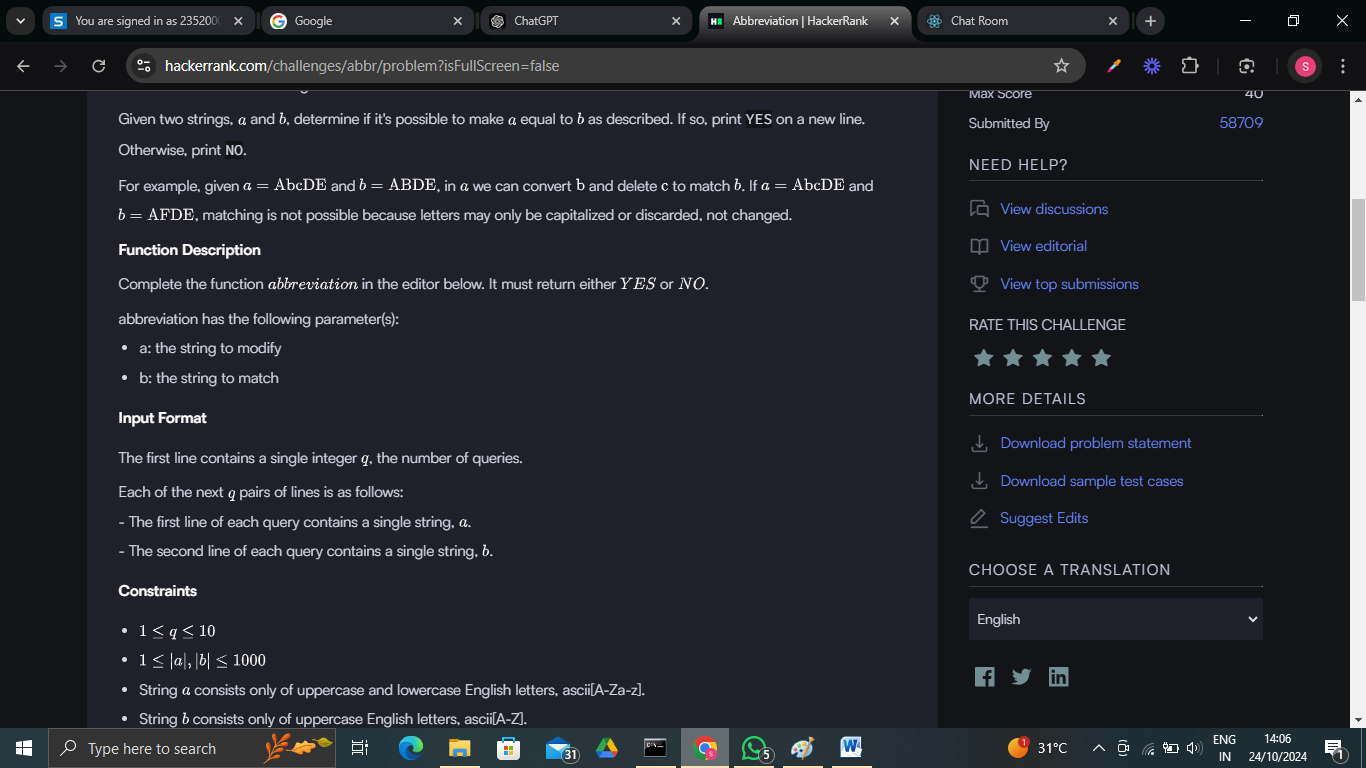
**Output:**

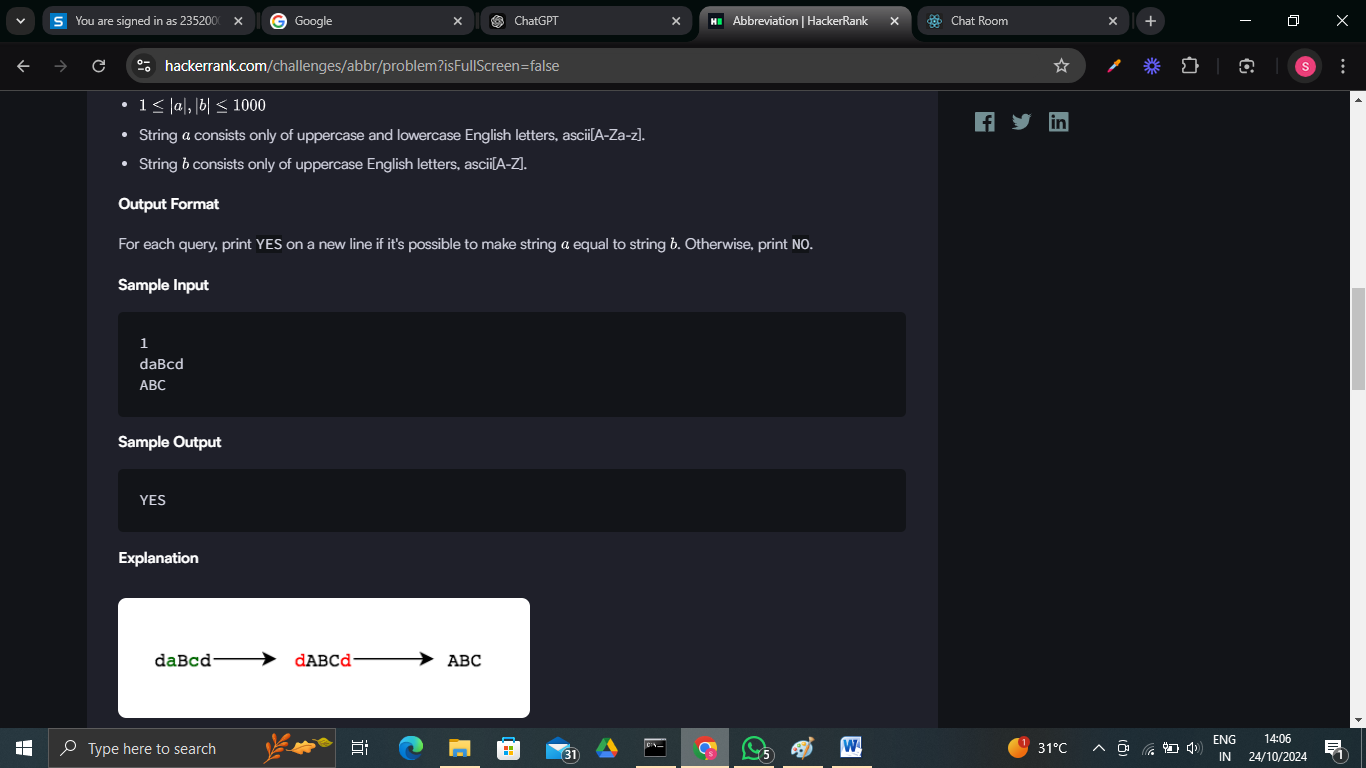




**Problem 5:**







**Code:**

import java.io.\*;

import java.util.stream.\*;

class Result {

*/\**

*\* Complete the 'abbreviation' function below.*

*\**

*\* The function is expected to return a STRING.*

*\* The function accepts following parameters:*

*\*  1. STRING a*

*\*  2. STRING b*

*\*/*

    public static String abbreviation(String a, String b) {

        int m = a.length();

        int n = b.length();

*// DP array to keep track of whether we can form b[0...j] from a[0...i]*

        boolean[][] dp = new boolean[m + 1][n + 1];

        dp[0][0] = true; *// Both strings empty is a valid transformation*

*// Fill the first column: If we can remove all lower case from a[0...i]*

        for (int i = 1; i <= m; i++) {

            dp[i][0] = dp[i - 1][0] && Character.isLowerCase(a.charAt(i - 1));

        }

*// Fill the rest of the DP table*

        for (int i = 1; i <= m; i++) {

            for (int j = 1; j <= n; j++) {

                char currentA = a.charAt(i - 1);

                char currentB = b.charAt(j - 1);

*// If characters match (case insensitive) or we can delete the current character*

                if (Character.toUpperCase(currentA) == currentB) {

                    dp[i][j] = dp[i - 1][j - 1] || (dp[i - 1][j] && Character.isLowerCase(currentA));

                } else if (Character.isLowerCase(currentA)) {

                    dp[i][j] = dp[i - 1][j] || (dp[i - 1][j] && Character.isLowerCase(currentA));

                }

            }

        }

        return dp[m][n] ? "YES" : "NO"; *// Return YES or NO based on the last cell*

    }

}

public class Solution {

    public static void main(String[] args) throws IOException {

        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

        int q = Integer.parseInt(bufferedReader.readLine().trim());

        IntStream.range(0, q).forEach(qItr -> {

            try {

                String a = bufferedReader.readLine();

                String b = bufferedReader.readLine();

                String result = Result.abbreviation(a, b);

                bufferedWriter.write(result);

                bufferedWriter.newLine();

            } catch (IOException ex) {

                throw new RuntimeException(ex);

            }

        });

        bufferedReader.close();

        bufferedWriter.close();

    }

}

Output:

