2300031751
LALITHYA

DAA skill week 12

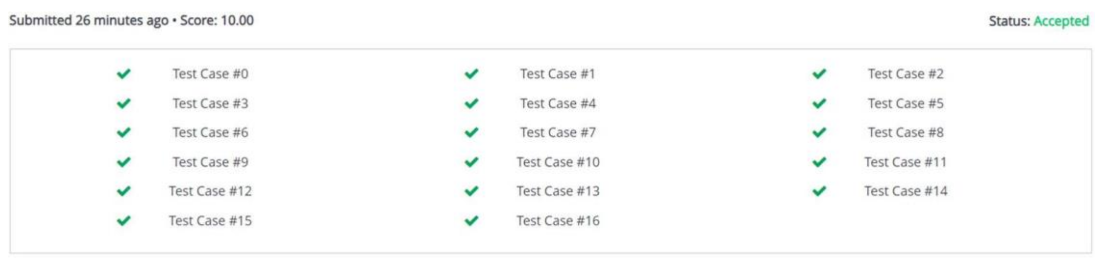
1) Lego Blocks:-

```
import java.io.*;

import java.math.*;

import java.security.*;

import java.text.*;

import java.util.*;

import java.util.concurrent.*;

import java.util.regex.*;


class Result {

  private static final int MOD = 1_000_000_007;


  public static int legoBlocks(int n, int m) {

    // Step 1: Calculate the number of ways to build each row

    int[] rowWays = new int[m + 1];

    rowWays[0] = 1;  // Base case, one way to build width 0


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

      rowWays[i] = rowWays[i - 1];

      if (i >= 2) rowWays[i] = (rowWays[i] + rowWays[i - 2]) % MOD;

      if (i >= 3) rowWays[i] = (rowWays[i] + rowWays[i - 3]) % MOD;

      if (i >= 4) rowWays[i] = (rowWays[i] + rowWays[i - 4]) % MOD;

    }


    // Step 2: Calculate the number of ways to build the entire wall as a solid structure

    int[] totalWays = new int[m + 1];

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

      totalWays[i] = power(rowWays[i], n, MOD);
```

```
        }

        // Step 3: Remove invalid configurations (subtract non-solid partitions)
        int[] solidWays = new int[m + 1];
        solidWays[1] = totalWays[1];  // Base case, width 1 is always solid

        for (int i = 2; i <= m; i++) {
            solidWays[i] = totalWays[i];
            for (int j = 1; j < i; j++) {
                solidWays[i] = (solidWays[i] - (solidWays[j] * totalWays[i - j]) % MOD + MOD) % MOD;
            }
        }

        return solidWays[m];
    }

    // Function to compute (x^y) % mod using fast exponentiation
    private static int power(int x, int y, int mod) {
        long result = 1;
        long base = x;
        while (y > 0) {
            if (y % 2 == 1) {
                result = (result * base) % mod;
            }
            base = (base * base) % mod;
            y /= 2;
        }
        return (int) result;
    }
}
```

```
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());


    for (int tItr = 0; tItr < t; tItr++) {
      String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");


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


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


      int result = Result.legoBlocks(n, m);


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


    bufferedReader.close();
    bufferedWriter.close();
  }
}
```

Test cases:-


Submitted 26 minutes ago • Score: 10.00

Status: Accepted

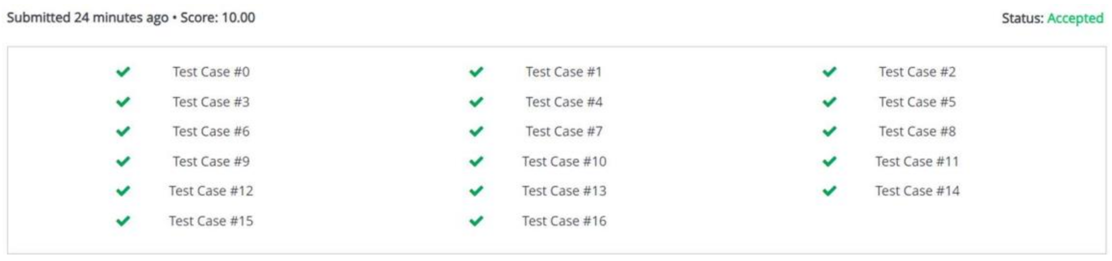
✓ Test Case #0	✓ Test Case #1	✓ Test Case #2
✓ Test Case #3	✓ Test Case #4	✓ Test Case #5
✓ Test Case #6	✓ Test Case #7	✓ Test Case #8
✓ Test Case #9	✓ Test Case #10	✓ Test Case #11
✓ Test Case #12	✓ Test Case #13	✓ Test Case #14
✓ Test Case #15	✓ Test Case #16	

2) Stock Maximize:-

```
import java.io.*;

import java.util.*;


class Result {


   /*

    * Complete the 'stockmax' function below.

    *

    * The function is expected to return a LONG_INTEGER.

    * The function accepts INTEGER_ARRAY prices as parameter.

    */

   public static long stockmax(List<Integer> prices) {

      long totalProfit = 0;

      int maxPriceSoFar = 0;
```

```
        // Iterate over prices in reverse order
        for (int i = prices.size() - 1; i >= 0; i--) {
            // If current price is greater than maxPriceSoFar, update it
            if (prices.get(i) > maxPriceSoFar) {
                maxPriceSoFar = prices.get(i);
            }
            // Calculate profit if we were to buy at the current price
            totalProfit += maxPriceSoFar - prices.get(i);
        }

        return totalProfit;
    }
}

public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(System.in));
        BufferedWriter bufferedWriter = new BufferedWriter(new
OutputStreamWriter(System.out));

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

        for (int tItr = 0; tItr < t; tItr++) {
            int n = Integer.parseInt(bufferedReader.readLine().trim());

            String[] pricesTemp = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");
            List<Integer> prices = new ArrayList<>();

            for (int i = 0; i < n; i++) {
                int pricesItem = Integer.parseInt(pricesTemp[i]);
                prices.add(pricesItem);
            }

            long result = Result.stockmax(prices);

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

        bufferedReader.close();
        bufferedWriter.close();
    }
}
```

Test cases:-



Submitted 24 minutes ago • Score: 10.00

Status: Accepted

✓ Test Case #0	✓ Test Case #1	✓ Test Case #2
✓ Test Case #3	✓ Test Case #4	✓ Test Case #5
✓ Test Case #6	✓ Test Case #7	✓ Test Case #8
✓ Test Case #9	✓ Test Case #10	✓ Test Case #11
✓ Test Case #12	✓ Test Case #13	✓ Test Case #14
✓ Test Case #15	✓ Test Case #16	