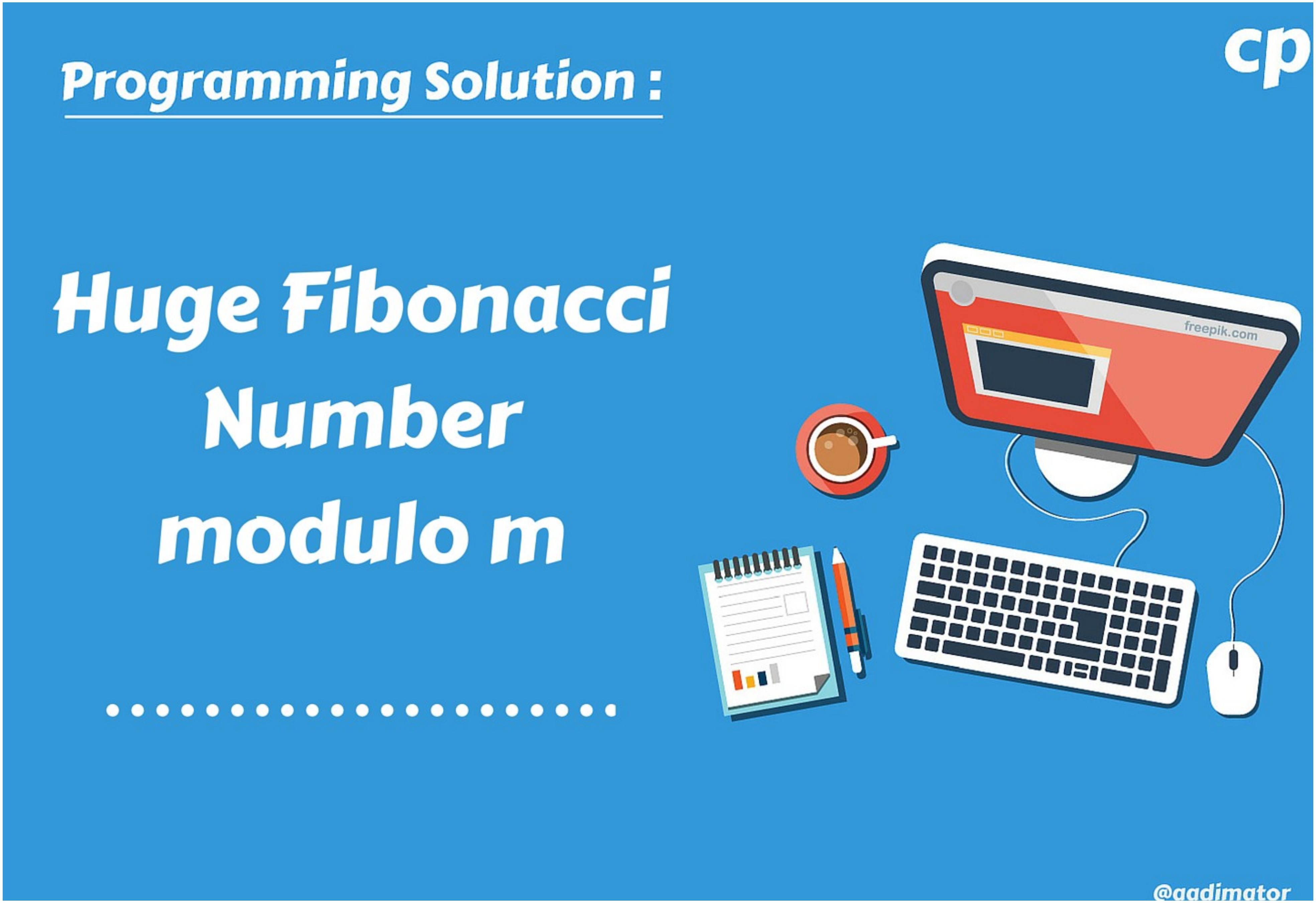
Programming Solution :
Huge Fibonacci Number modulo m
cp
freepik.com
@aadimator

Huge Fibonacci Number modulo m—aadimator

Huge Fibonacci Number modulo m

Aadam Follow
Jul 21, 2016 · 3 min read

This problem was taken from the *Coursera* **Data Structures and Algorithms Specialization**, specifically from the **Algorithmic Toolbox Course**, Week 2, that I've recently completed. If you are taking that course or plan to take that course, please **don't look ahead** at the solution as it will be against the Honor Code and won't do you any good.

· · ·

Problem Introduction

The Fibonacci numbers are defined as follows: $\mathbf{F(0) = 0}$, $\mathbf{F(1) = 1}$, and $\mathbf{F(i) = F(i-1) + F(i-2)}$ for $i \geq 2$.

Problem Description

Task: Given two integers **n** and **m**, output **F(n) mod m** (that is, the remainder of **F(n)** when **divided by m**).
Input Format: The input consists of two integers **n** and **m** given on the same line (separated by a space).
Constraints: $1 \leq \mathbf{n} \leq 10^{18}$, $2 \leq \mathbf{m} \leq 10^{5}$.

Output Format: Output **F(n) mod m**
Time Limits: C: 1 sec, C++: 1 sec, Java: 1.5 sec, Python: 5 sec. C#: 1.5 sec, Haskell: 2 sec, JavaScript: 3 sec, Ruby: 3 sec, Scala: 3 sec.
Memory Limit: 512 Mb

Sample

Input:
281621358815590 30524
Output:
11963

What To Do

In this problem, the given number **n** may be really huge. Hence an algorithm looping for **n** iterations will not fit into one second for sure. Therefore we need to avoid such a loop.
To get an idea how to solve this problem without going through all **F(i)** for **i** from *0 to n*, let's see what happens when **m** is small—say, *m = 2* or *m = 3*.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F_i	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610
$F_i \bmod 2$	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0
$F_i \bmod 3$	0	1	1	2	0	2	2	1	0	1	1	2	0	2	2	1

Take a detailed look at this table. Do you see? Both these sequences are periodic! For *m = 2*, the period is *011* and has length *3*, while for *m = 3* the period is *01120221* and has length *8*. Therefore, to compute, say, **F(2015) mod 3** we just need to *find the remainder of 2015* when *divided by 8*. Since $2015 = 251 \cdot 8 + 7$, we conclude that $F(2015) \bmod 3 = F_7 \bmod 3 = 1$.

This is true in general: for any integer $m \geq 2$, the sequence **F(n) mod m** is periodic. The period always starts with *01* and is known as **Pisano period**.

. . .

My Solution:

```
#include <iostream>

long long get_pisano_period(long long m) {
    long long a = 0, b = 1, c = a + b;
    for (int i = 0; i < m * m; i++) {
        c = (a + b) % m;
        a = b;
        b = c;
        if (a == 0 && b == 1) return i + 1;
    }
}

long long get_fibonacci_huge(long long n, long long m) {
    long long remainder = n % get_pisano_period(m);

    long long first = 0;
    long long second = 1;

    long long res = remainder;

    for (int i = 1; i < remainder; i++) {
        res = (first + second) % m;
        first = second;
        second = res;
    }
```

Explanation:

I couldn’t find a suitable definition for Pisano period so that I could make a naive algorithm. Well, it turns out, I was looking at the wrong place. The whole time, it was right in front of my eyes but I couldn’t see it. So, without further ado, here's the definition. This was a very tricky one. I had to search a lot and read quite a few posts in the Course forum to understand the algorithm.

The algorithm to compute the get_fibonacci_huge was given in the What To Do section. "Therefore, to compute, say, **F(2015) mod 3** we just need to *find the remainder of 2015* when *divided by 8*. Since *2015 = 251·8+7*, we conclude that *F(2015) mod 3 = F 7 mod 3 = 1*." We just have to generalize it.

First we need to find the *remainder of **F(n)** divided by the length of Paisano period given **m***. To find the *length of Paisano period for **m***, simply find the remainder of F(0) mod m to F(m*m) mod m and stop as soon as you encounter **01**, as they indicate that the next iteration is being started, and return the length up to that point.

Now, you only need to find the F(remainder), which is going to be a lot less than F(n) and simply return it.

. . .

If you can think of any other way to improve this algorithm or if you could point out something that you think I did wrong, you're more than w*elcome* to reach out to me by responding to this and pointing out the mistakes. If you like this solution, please hit the "Recommend" button below, it'll mean a lot to me. Thanks.