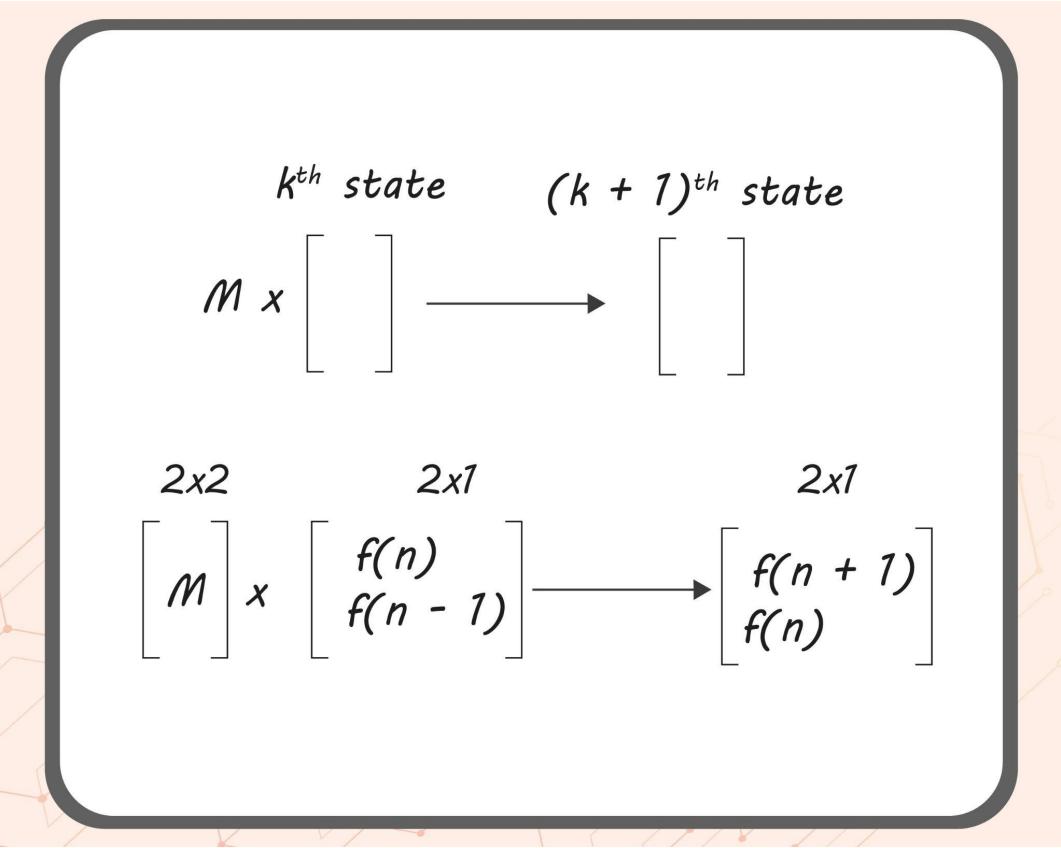
Number Theory - 4

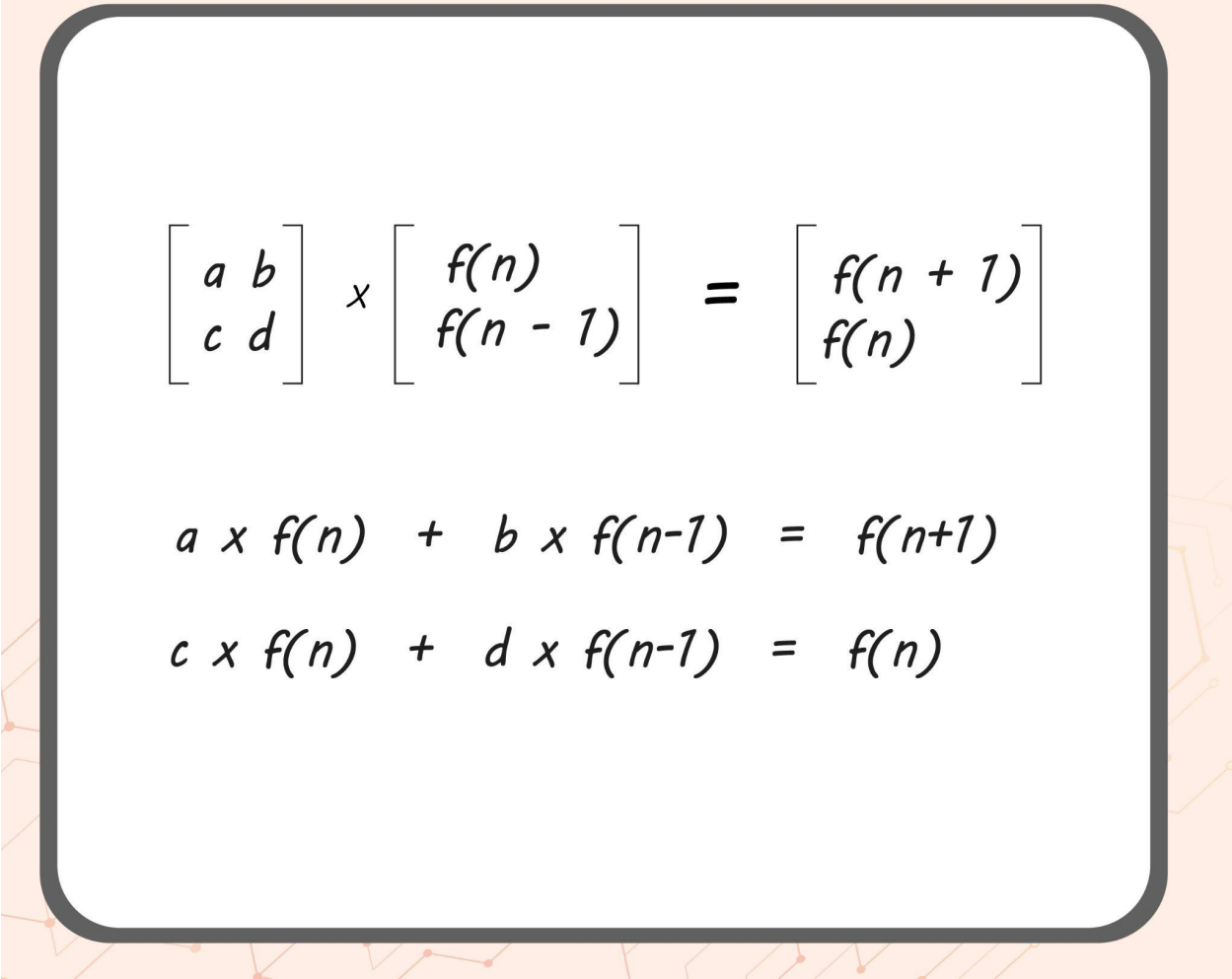
Matrix Exponentiation

Matrix exponentiation says that we have to find a matrix such that our recurrence relation at k^{th} state when multiplied by the matrix gives the $(k + 1)^{th}$ state of the recurrence relation.

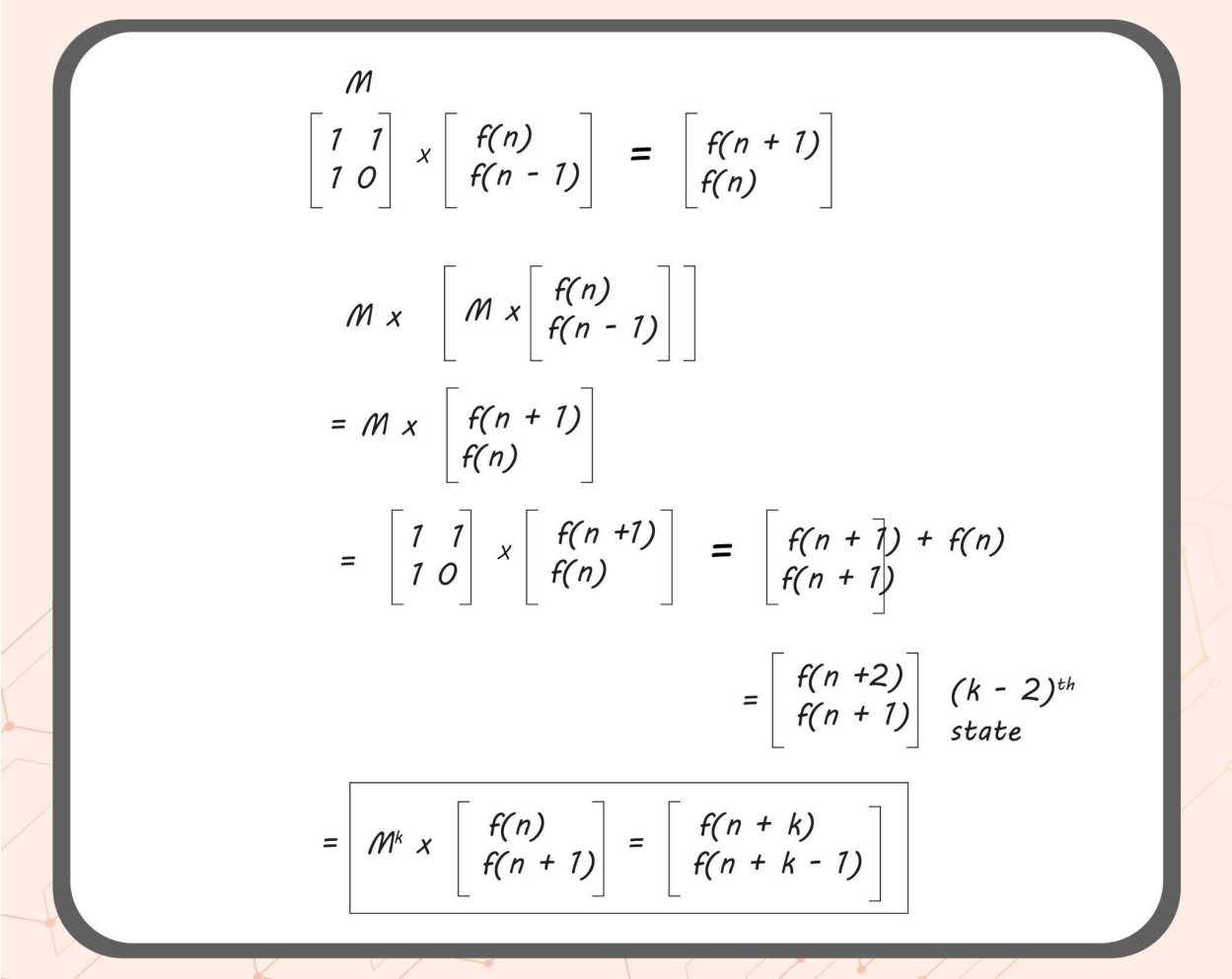
Example : **f (n) = f (n-1) + f (n-2)**

Since there are two unknowns therefore our matrix will be of size **2 X 2**.





Now, since **f (n) = f (n -1) + f (n - 2)** can be written as **f (n + 1) = f (n) + f (n - 1)**, therefore, **a , b = 1**. Also, **c = 1 and d = 0**.



$$M \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} f(n+1) \\ f(n) \end{bmatrix}$$

$$M \times \left[M \times \begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} \right]$$

$$= M \times \begin{bmatrix} f(n+1) \\ f(n) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} f(n+1) \\ f(n) \end{bmatrix} = \begin{bmatrix} f(n+1) + f(n) \\ f(n+1) \end{bmatrix}$$

$$= \begin{bmatrix} f(n+2) \\ f(n+1) \end{bmatrix} \quad (k-2)^{th} \text{ state}$$

$$= \boxed{M^k \times \begin{bmatrix} f(n) \\ f(n+1) \end{bmatrix} = \begin{bmatrix} f(n+k) \\ f(n+k-1) \end{bmatrix}}$$

Time complexity of finding Nth fibonacci using matrix exponentiation will be **O(log(N))**.

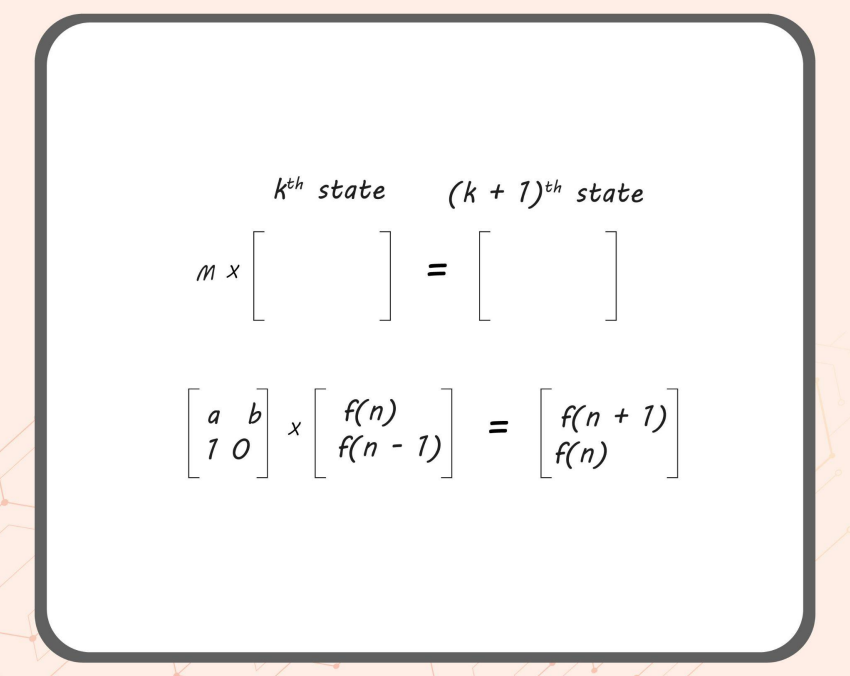
Code:

```
#include<iostream>
using namespace std;


void multiply(int A[2][2],int M[2][2]){

    int firstValue = A[0][0] * M[0][0] + A[0][1] * M[1][0];
    int secondValue = A[0][0] * M[0][1] + A[0][1] * M[1][1];
    int thirdValue = A[1][0] * M[0][0] + A[1][1] * M[1][0];
```

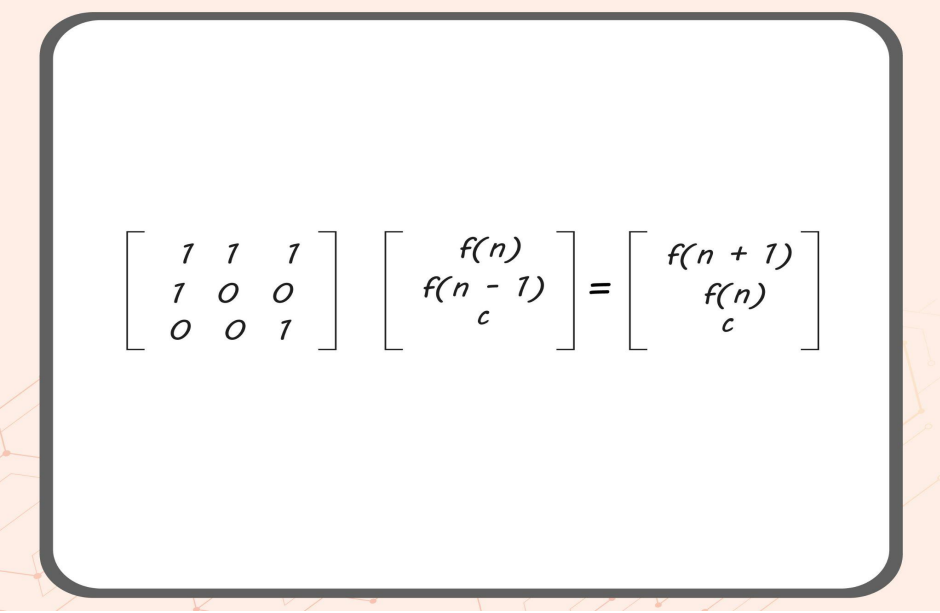
```
    int fourthValue = A[1][0] * M[0][1] + A[1][1] * M[1][1];

    A[0][0] =firstValue;
    A[0][1] = secondValue;
    A[1][0] = thirdValue;
    A[1][1] = fourthValue;

}
void power(int A[2][2],int n){
    if(n==1){
        return;
    }
    power(A,n/2);
    multiply(A,A);
    if(n%2 !=0){
        int F[2][2] = {{1,1},{1,0}};
        multiply(A,F);
    }
}
int getFibonacci(int n){
    if(n==0 || n==1){
        return n;
    }
    int A[2][2] = {{1,1},{1,0}};
    power(A,n-1);
    return A[0][0];
}
int main(){
    int n;
    cin >> n;
    cout << getFibonacci(n)<<endl;
    return 0;
}
```

Some examples of Recurrence Relations

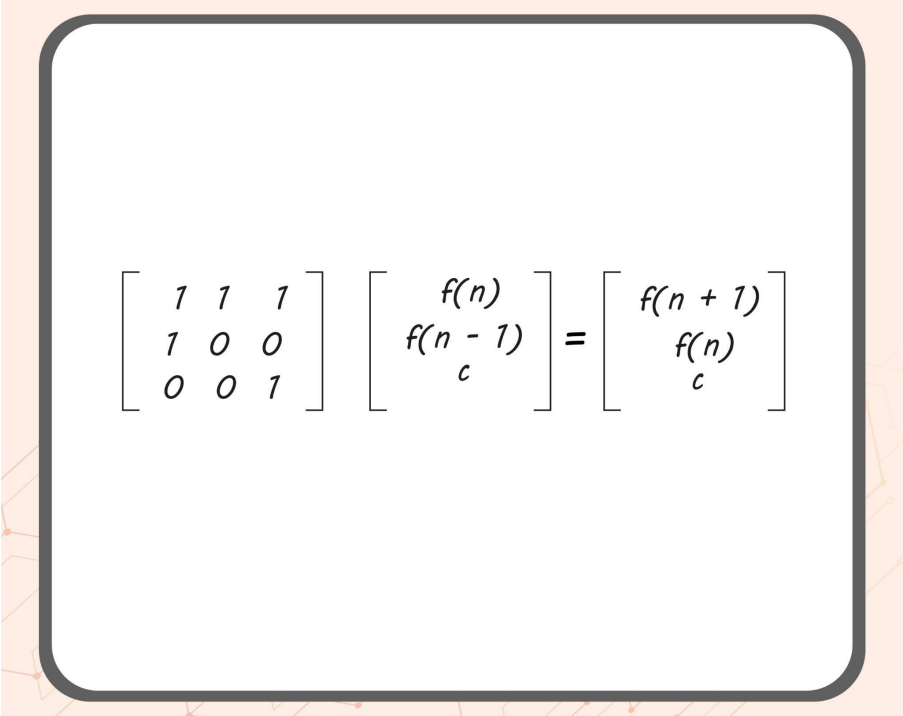
1. **f (n) = a * f (n-1) + b * f (n-2).**



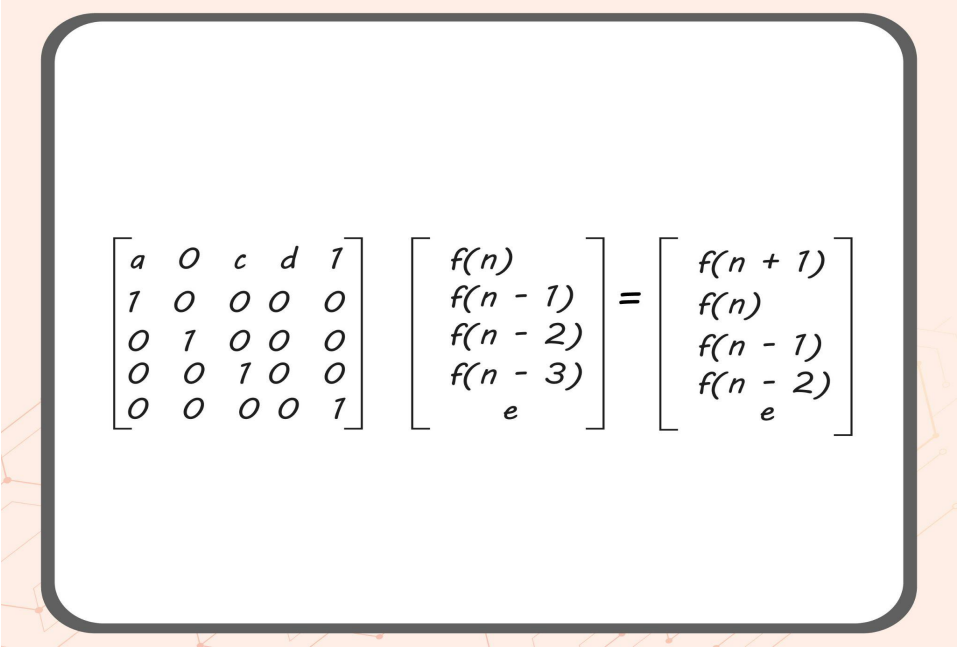
2. **f (n) = f (n-1) + f (n-2) + c**



3. **f (n) = f (n-1) + f (n-3)**



4. **f (n) = a * f (n-1) + b * f (n-2) + c * f (n-3) + d * f (n-4) + e**

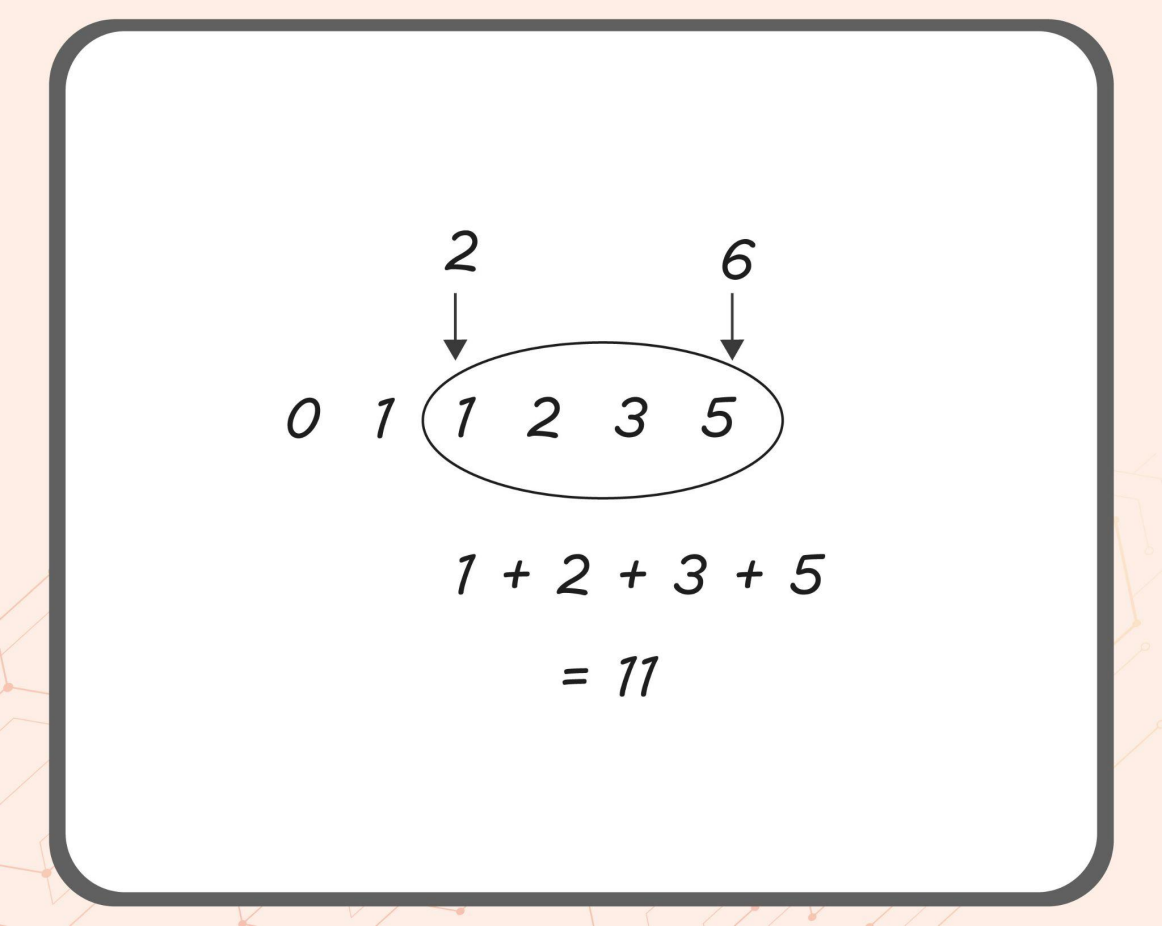


FiboSum

Problem Statement : Find the sum between N^{th} and M^{th} fibonacci numbers.

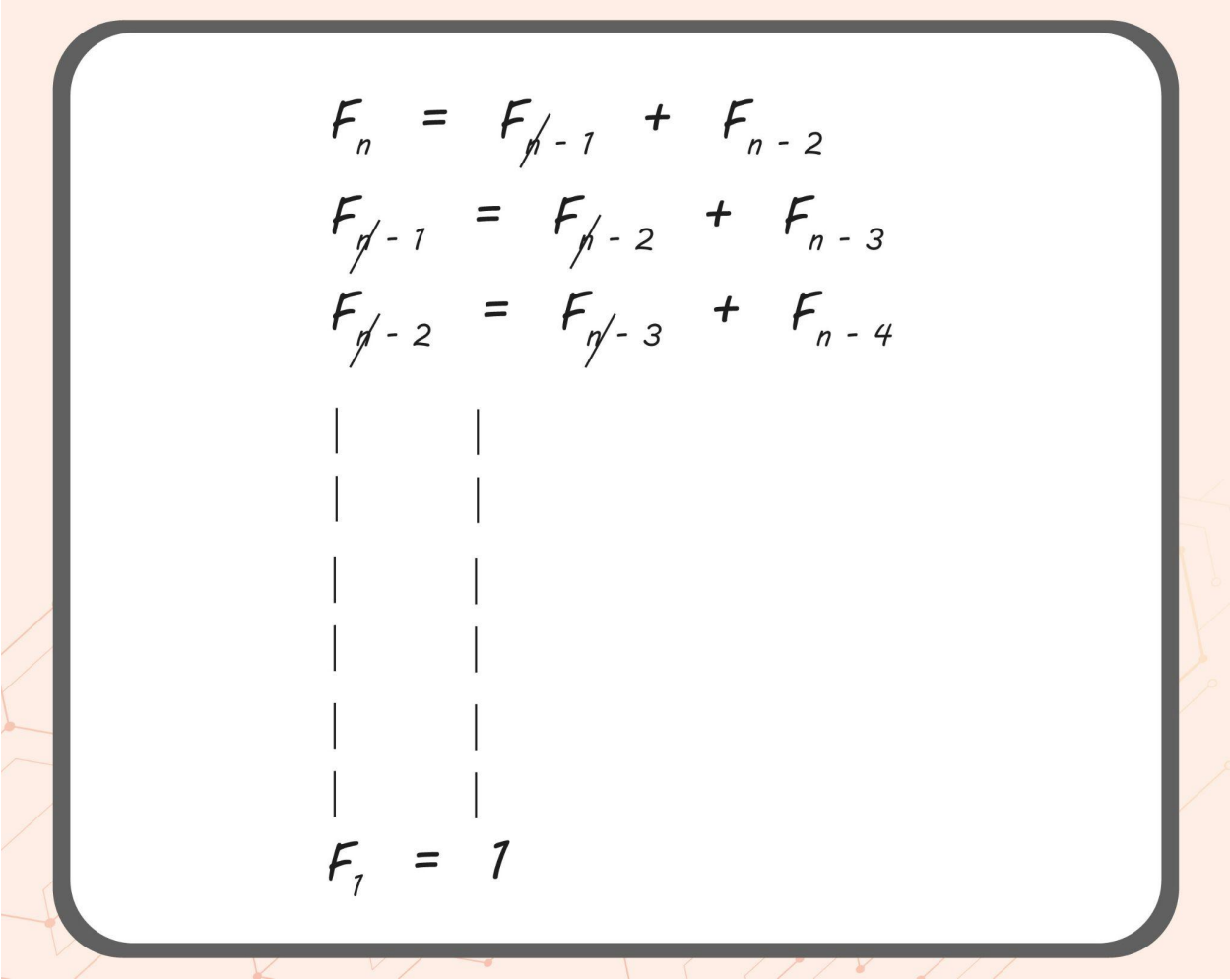
Explanation :

Example : Taking N = 2 and M = 6,



Now, **M, N <= 10^9** and a maximum of 10^8 operations per second are allowed so we can't calculate the sum in **O (M)** or **O (N)**.

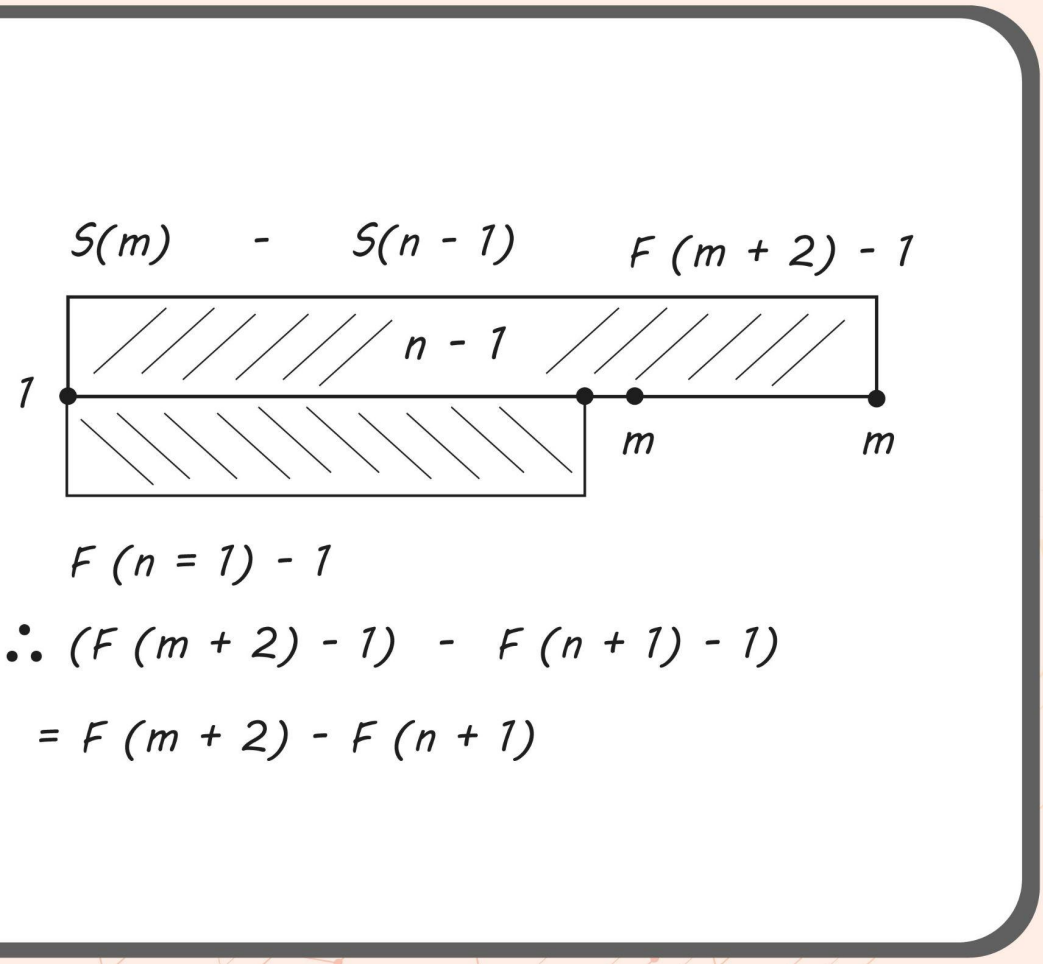
Optimized Approach: Sum of N fibonacci numbers is $S_N = S_{N-1} + F_N$



→ $F_N = F_{N-2} + F_{N-3} + ...+ F_1 + 1$

→ $F_{N-1} = F_{N-2} + F_{N-3} +.....+ F_1 = S_{N-2}$

Therefore, $S_N = F_{N+2} - 1$


S(m) - S(n - 1)
F (m + 2) - 1
n - 1
1
m
m
F (n = 1) - 1
∴ (F (m + 2) - 1) - F (n + 1) - 1)
= F (m + 2) - F (n + 1)

Fermat's Little Theorem

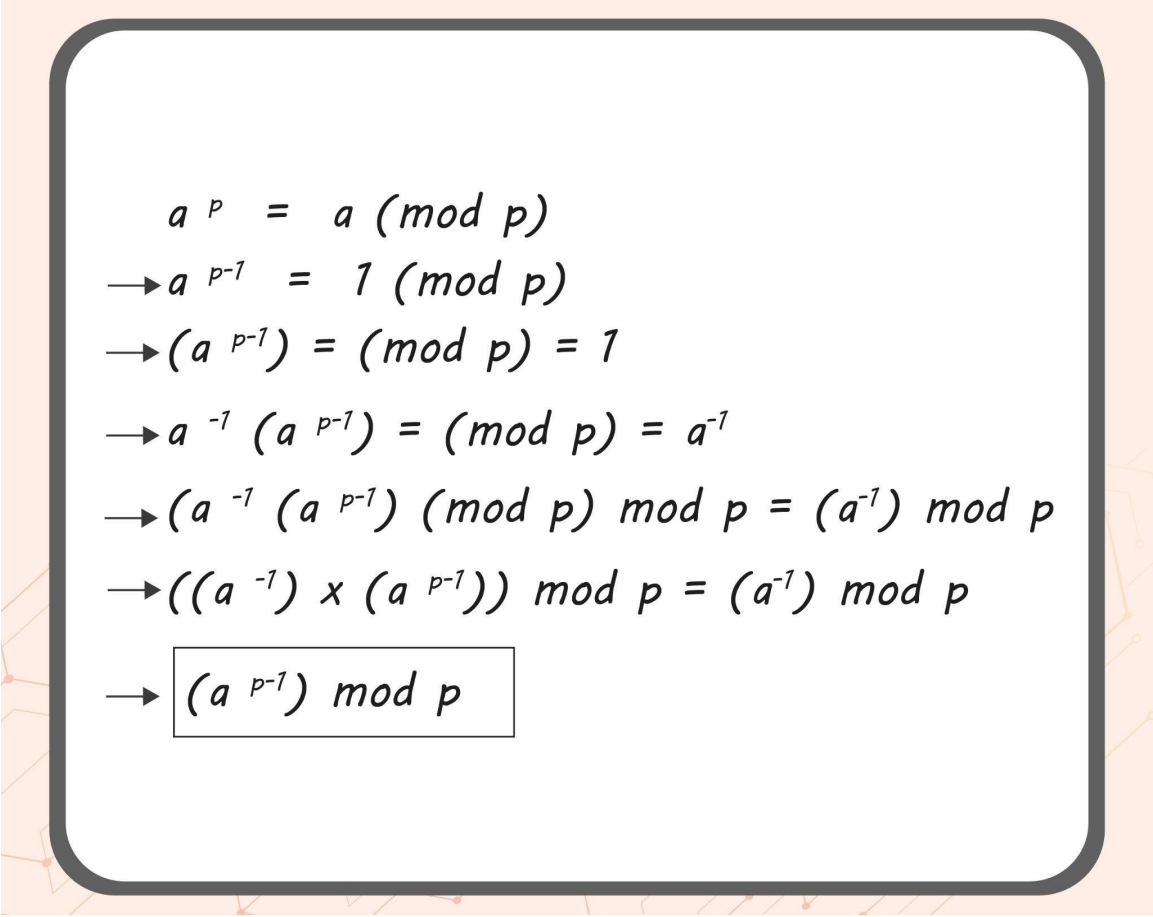
Fermat's little theorem is a fundamental theorem in elementary number theory, which helps compute powers of integers modulo prime numbers.

The theorem states that :

Let **p** be a prime number, and **a** be any integer. Then $\mathbf{a^p - a}$ is always divisible by **p**.

In modular arithmetic notation, this can be written as $\mathbf{a^p \equiv a \bmod p}$.

Example: 11 is prime so $2^{11} - 2 = 2046$ is divisible by 11 by Fermat's little theorem.



$\mathbf{(a^{p-2})}$ mod p can be calculated using modular exponentiation.

Wilson’s Theorem

Wilson's theorem states that a positive integer **p > 1** is a prime if and only if **(p−1)!≡−1(modp)**.

Let’s verify Wilson’s theorem using some examples:

n	(n-1)!	(n-1)! (mod n)	Is Prime?
2	1	1	yes
3	2	2	yes
4	6	2	no
5	24	4	yes
6	120	0	no
7	720	6	yes
8	5040	0	no
9	40320	0	no
10	362880	0	no

Uses :

Consider the problem of computing factorial under modulo of a prime number which is close to the input number, i.e., we want to find the value of “n! % p” such that n < p, p is a prime and n is close to p. For example (25! % 29). From Wilson's theorem, we know that 28! is -1. So we basically need to find [(-1) * inverse(28, 29) * inverse(27, 29) * inverse(26)] % 29. The inverse function inverse(x, p) returns the inverse of x under modulo p.

Income on the N^{th} day

Problem Statement:

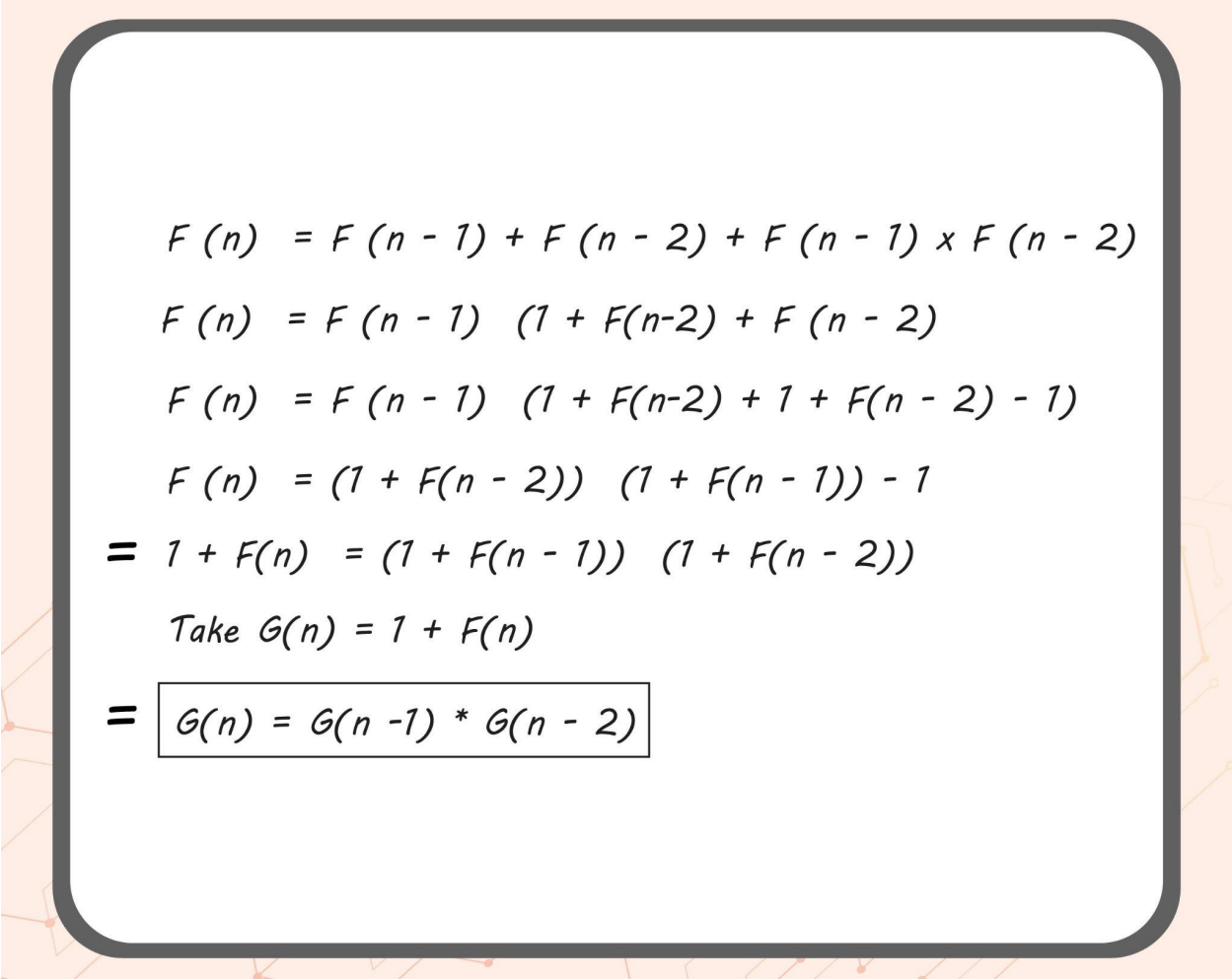
Daulat Ram is an affluent businessman. After demonetization, an IT raid was held at his accommodation in which all his money was seized. He is very eager to gain his money back, he started investing in certain ventures and earned out of them. On the first day, his income was Rs. X, followed by Rs. Y on the second day. Daulat Ram observed his growth as a function and wanted to calculate his income on the n^{th} day.

The function he found out was **F(n) = F(n-1) + F(n-2) + F(n-1)×F(n-2)**

Given his income on day 0 and day 1, calculate his income on the n^{th} day (yeah It's that simple).

Explanation:

Manipulating the given expression a little bit:



Now take **G(0) = a** and **G(1) = b**.

→ **G(2) = ab**

→ **G(3) = ab^2**

→ **G(4) = a^2b^3**

→ **G(5) = a^3b^5**

→ **G(n) = $a^{fib(n-1)}$ $b^{fib(n)}$**

Now that we know the general expression of G(n), we can easily find out the general expression for F(n).