Number Theory - 3

Optimized Power Function

We have to find out the **pow (x, n)**.

Basic Approach : We can solve this problem recursively by **x * pow (x, n)**.

Optimized Approach :

If n is even :

Val = pow (x, n)

Pow (x, n) = Val * Val

Example: $x^8 = x^4 * x^4$

If n is odd :

Val = pow (x, n/2)

pow (x, n) = Val * Val * x

Example: $x^9 = x^4 * x^4 * x$

Time complexity of this approach will be O(log(n)).

Modular Exponentiation

If in the previous topic of optimized power function, we have to find $\mathbf{2^{1024}}$ then it will be out of the range of integers or long long.

We can prevent the answers to go out of range using modulo arithmetic

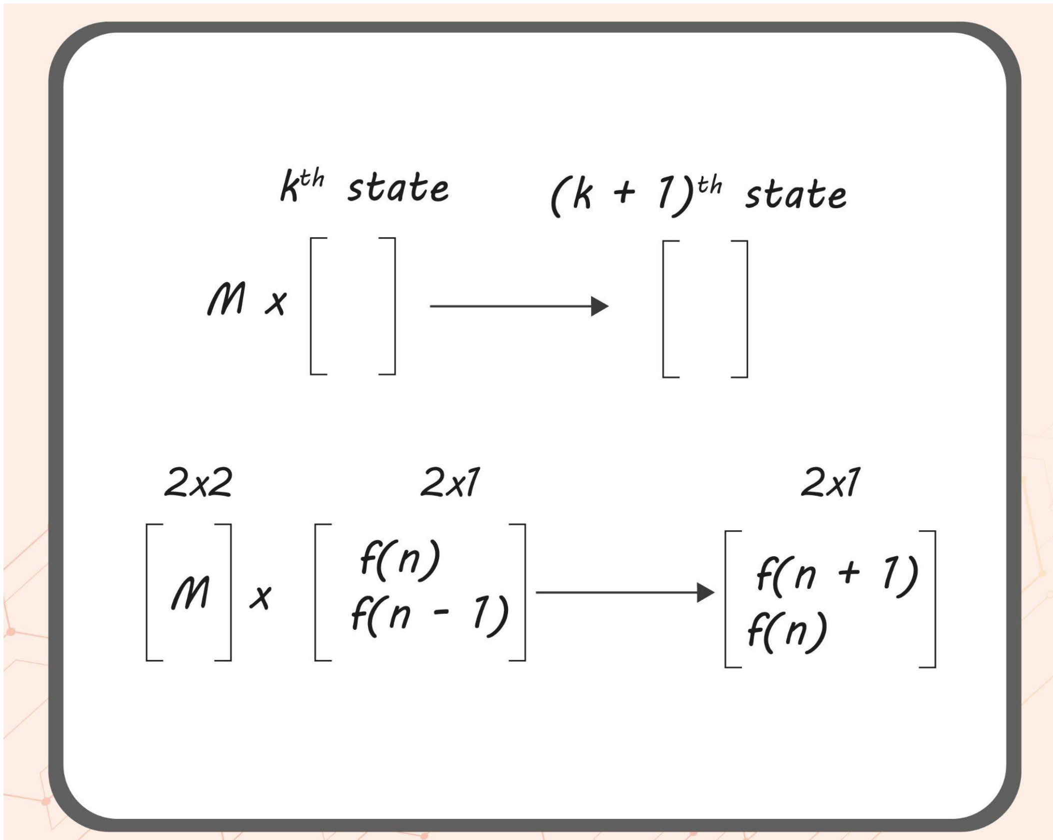
(a * b) % c = ((a % c) * (b % c)) % c

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**.


kth state
(k + 1)th state
M x
2x2
2x1
2x1
M
x
f(n)
f(n - 1)
f(n + 1)
f(n)

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} f(n+1) \\ f(n) \end{bmatrix}$$

$$a \times f(n) + b \times f(n-1) = f(n+1)$$

$$c \times f(n) + d \times f(n-1) = f(n)$$

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**.

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

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

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

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

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

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

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)**.

$$
\begin{array}{ccc} & k^{th} \text{ state} & & (k+1)^{th} \text{ state} \\ M \times & \begin{bmatrix} \quad \\ \quad \end{bmatrix} & = & \begin{bmatrix} \quad \\ \quad \end{bmatrix} \end{array}
$$

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

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

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

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

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

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

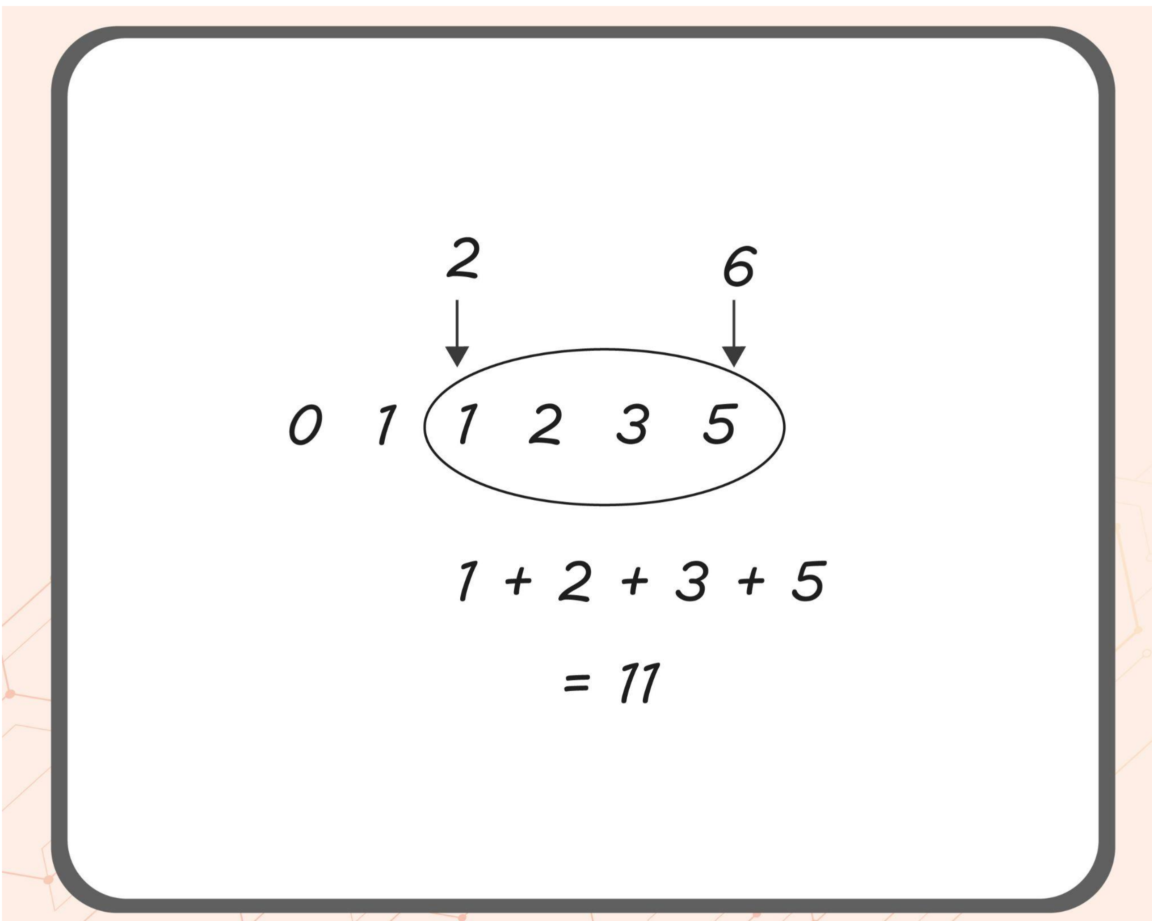
$$\begin{bmatrix} a & 0 & c & d & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f(n) \\ f(n-1) \\ f(n-2) \\ f(n-3) \\ e \end{bmatrix} = \begin{bmatrix} f(n+1) \\ f(n) \\ f(n-1) \\ f(n-2) \\ e \end{bmatrix}$$

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 $\mathbf{S_N = S_{N-1} + F_N}$

$$F_n = F_{\cancel{n}-1} + F_{n-2}$$

$$F_{\cancel{n}-1} = F_{\cancel{n}-2} + F_{n-3}$$

$$F_{\cancel{n}-2} = F_{\cancel{n}-3} + F_{n-4}$$

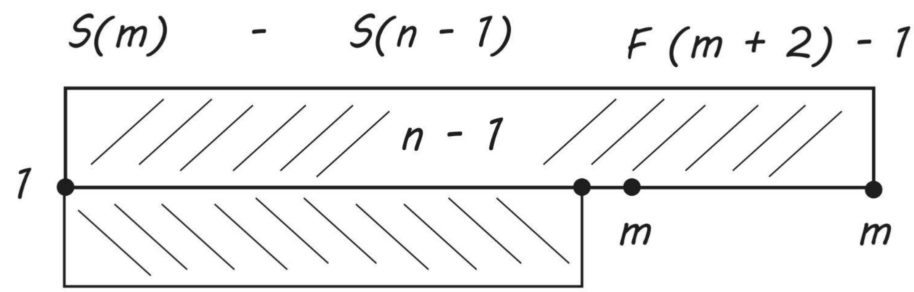
$$\vdots \qquad \vdots$$

$$F_1 = 1$$

→ $\mathbf{F_N = F_{N-2} + F_{N-3} + \ldots\ldots + F_1 + 1}$

→ $\mathbf{F_{N-1} = F_{N-2} + F_{N-3} + \ldots\ldots + F_1 = S_{N-2}}$

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



$F(n = 1) - 1$

$\therefore (F(m + 2) - 1) - F(n + 1) - 1)$

$= F(m + 2) - F(n + 1)$