Matrix Exponentiation and Extended Euclidean Algorithm

MNNIT Computer Coding Club


think
idea
try
do
do again
and again
keep doing
success!

Objectives

For Extended Euclidean Algorithm

1. Understand the extended euclidean algorithm
2. Applications

For Matrix Exponentiation

1. Understand Matrix Exponentiation.
2. How to use this for solving problems?

Extended Euclidean Algorithm

Pre-requisites

- Euclidean Algorithm
- Knowledge about gcd

What exactly is extended from the trivial Euclidean algorithm

The standard Euclidean algorithm simply aims to find the gcd of two numbers.

The extended euclidean algorithm not only finds the gcd of two numbers (a , b) but also finds the value of x , y satisfying the following equation.

$$a \cdot x + b \cdot y = gcd(a,b)$$

Algorithm Proof and working

To find : x , y satisfying **ax + by = gcd(a , b)**

Let x1 , y1 be known such that **b*x1 + (a mod b)*y1 = gcd(a , b)**

Reiterating the second equation we get

b * x1 + (a - floor(a/b) * b) * y1 = gcd(a , b)

a * y1 + b * (x1 - floor(a/b)*y1) = gcd(a , b)

Thus comparing coefficients we get **x = y1** and **y = x1 - floor(a/b)*y1**

This leads to a recursive solution to find x, y simultaneously with gcd where base case scenario is (x = 1 , y = 0) when b = 0

Code

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
```

Applications

1. Solving Linear Diophantine Equations
2. Finding modular multiplicative inverse.

Linear Diophantine Equations

A linear Diophantine equation is just a fancy name for the following equation ax + by = c

Cases

1. If a = b = 0 then if c = 0 then infinite solutions else no solution (degenerate case)
2. If **c % gcd(a , b) == 0** then infinite solutions with one particular and other general solutions

   Let (x1 , y1) be the solution of ax + by = gcd(a , b) then particular solution is (x = **x1*c/gcd(a,b)** , y = **y1*c/gcd(a,b)**) and general solutions are **(x + k*b/gcd(a,b) , y - k * a/gcd(a,b))**

3. If **c % gcd(a , b) != 0** then no solution

Modular Multiplicative inverse

Aim : To find (1/a) mod m.

If gcd(a , m) > 1 then no inverse is possible

else

Consider the equation **ax + my = 1**

From extended euclidean algorithm we can find (x , y).

Taking mod m both sides.

ax mod m = 1 mod m thus **x is the inverse** of a modulo m.

Matrix Exponentiation

References

1. Binary Exponentiation - Competitive Programming Algorithms (cp-algorithms.com)
2. Matrix exponentiation | HackerEarth
3. I, ME AND MYSELF !!!: Matrix Exponentiation (zobayer.blogspot.com)

Prerequisite

1. Binary Exponentiation.
2. Recursion.
3. Matrix.

Calculate a^n

1. Naive approach

```
int res = 1;
for(int i=0;i<n;i++) {
   res = res*a;
}
```

2. Binary Exponentiation.

Binary Exponentiation

Binary exponentiation (also known as exponentiation by squaring) is a trick which allows to calculate a^n using only O(logn) multiplications (instead of O(n) multiplications required by the naive approach).

$$a^n = \begin{cases} 1 & \text{if } n == 0 \\ \left(a^{\frac{n}{2}}\right)^2 & \text{if } n > 0 \text{ and } n \text{ even} \\ \left(a^{\frac{n-1}{2}}\right)^2 \cdot a & \text{if } n > 0 \text{ and } n \text{ odd} \end{cases}$$

Implementation - Binary Exponentiation

1. Recursive:

```
long long binpow(long long a, long long b) {
    if (b == 0)
        return 1;
    long long res = binpow(a, b / 2);
    if (b % 2)
        return res * res * a;
    else
        return res * res;
}
```

Implementation - Binary Exponentiation

2. Iterative:

```
long long binpow(long long a, long long b) {
    long long res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

Matrix

A matrix is a rectangular array or table of numbers, symbols, or expressions, arranged in rows and columns.

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}$$

Identity Matrix

Identity matrix of size n is the n × n square matrix with ones on the main diagonal and zeros elsewhere.

$$I_n = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}$$

Matrix Multiplication

Consider two matrices:

1. Matrix A have n rows and k columns;
2. Matrix B have k rows and m columns (notice that number of rows in B is the same as number of columns in A).

Then we define operation:

C = A * B (matrix multiplication)

$$c_{ij} = \sum_{r=1}^{k} a_{ir} * b_{rj}$$

Matrix Exponentiation

Suppose you have a matrix A with n rows and n columns (we'll call such matrices "square matrix of size n"). We can define matrix exponentiation as:

$A^x = A * A * A * \ldots * A$ (x times)

with special case of x = 0:

$A^0 = I_n$

Implementation - Matrix Exponentiation

The brute-force solution would be:

```
function matrix_power_naive(A, x):
  result = I_n
  for i = 1..x:
    result = result * A
  return result
```

It runs in $\Theta(n^3 * x)$: we do x matrix multiplications on square matrices of size n, and each multiplication runs in $\Theta(n^3)$.

Can we use same trick?

Yes, we can use same idea as follows:

```
function matrix_power_final(A, x):
  result = I_n
  while x > 0:
    if x % 2 == 1:
      result = result * A
    A = A * A
    x = x / 2
  return result
```

Finding Nth Fibonacci Number

Fibonacci numbers F_n are defined as follows:

1. $F_0 = F_1 = 1$;
2. $F_i = F_{i-1} + F_{i-2}$ for $i \geq 2$.

We want to find F_N modulo 1000000007, where N can be up to 10^{18}.

Approach

We have a recursive relation, and what we want to do, is to find a matrix *M* which can lead us to the desired state from a set of already known states. Let, we know *k* states of a given recurrence relation, and want to find the *(k+1)*th state. Let *M* be a *k x k* matrix, and we build a matrix *A*:[*k x 1*] matrix from the known states of the recurrence relation, now we want to get a matrix *B*:[*k x 1*] which will represent the set of next states, i.e. *M x A = B*, as shown below:

$$M \times \begin{vmatrix} f(n) \\ f(n-1) \\ f(n-2) \\ \cdots\cdots \\ f(n-k) \end{vmatrix} = \begin{vmatrix} f(n+1) \\ f(n) \\ f(n-1) \\ \cdots\cdots \\ f(n-k+1) \end{vmatrix}$$

Back to Fibonacci Problem

For Fibonacci problem, we have:

Matrix A

$$\begin{vmatrix} f(n) \\ f(n-1) \end{vmatrix}$$

Matrix B

$$\begin{vmatrix} f(n+1) \\ f(n) \end{vmatrix}$$

Find M

Let us assume:

$$M = \begin{pmatrix} x & y \\ z & w \end{pmatrix}$$

We want to find x, y, z and w. We will use $M \times A = B$ for this purpose.

$$\begin{bmatrix} x & y \\ z & w \end{bmatrix} \times \begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} f(n+1) \\ f(n) \end{bmatrix}$$

$$\begin{bmatrix} x \cdot f(n) + y \cdot f(n-1) \\ z \cdot f(n) + w \cdot f(n-1) \end{bmatrix} = \begin{bmatrix} f(n+1) \\ f(n) \end{bmatrix}$$

$\Rightarrow$ $x \cdot f(n) + y \cdot f(n-1) = f(n+1)$

& $z \cdot f(n) + w \cdot f(n-1) = f(n)$.

We already have, $f(n+1) = f(n) + f(n-1)$

$\Rightarrow$ $x = 1$ and $y = 1$

Also, $f(n) = f(n)$.

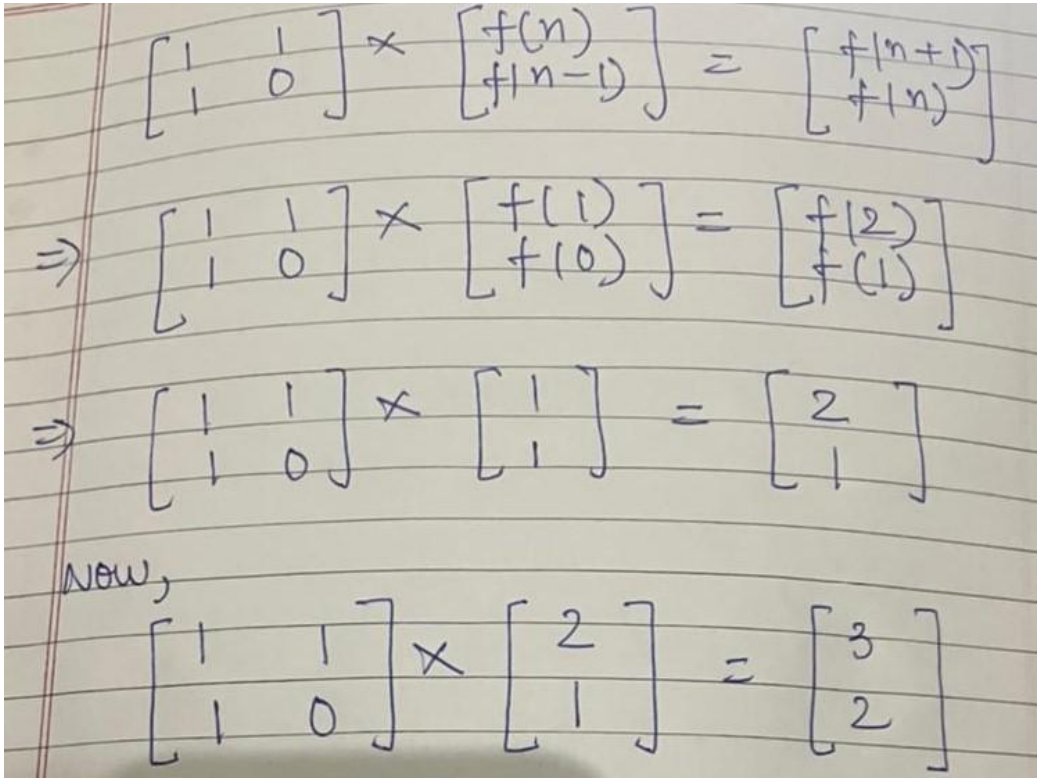
$\Rightarrow$ $z = 1$ and $w = 0$.

$\Rightarrow$ $M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$

What we have?

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

How this helps?



or

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} f(2) \\ f(1) \end{bmatrix} = \begin{bmatrix} f(3) \\ f(2) \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} f(1) \\ f(0) \end{bmatrix} = \begin{bmatrix} f(3) \\ f(2) \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^2 \times \begin{bmatrix} f(1) \\ f(0) \end{bmatrix} = \begin{bmatrix} f(3) \\ f(2) \end{bmatrix}.$$

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

$$\Rightarrow \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^2 \times \begin{bmatrix} f(n-1) \\ f(n-2) \end{bmatrix} = \begin{bmatrix} f(n+1) \\ f(n) \end{bmatrix}$$

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

let's assume $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

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

$$\Rightarrow \boxed{f(n+1) = a \cdot f(0) + b \cdot f(1)}$$

Slight Modification

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

Solution

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

New Problem

find $f(n) = a * f(n-1) + c * f(n-3)$.

Ooops! a few minutes ago, all we saw were contiguous states, but here, the state $f(n-2)$ is missing. Now? what to do?

Approach

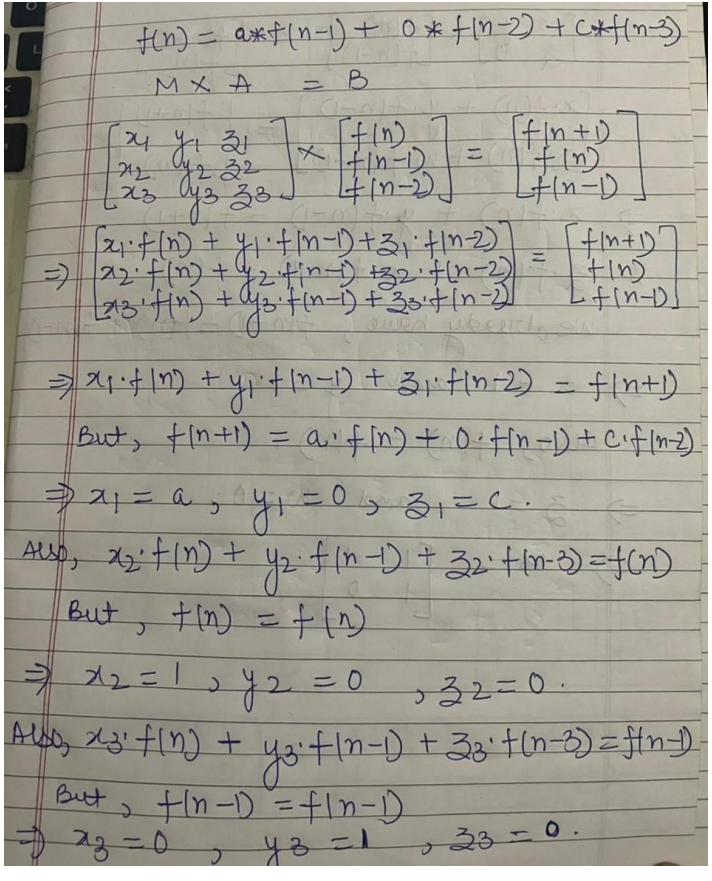
Actually, this is not a problem, we can convert the relation as follows:

$f(n) = a * f(n-1) + 0 * f(n-2) + c * f(n-3)$

Solution

$$\begin{vmatrix} a & 0 & c \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{vmatrix} \times \begin{vmatrix} f(n) \\ f(n-1) \\ f(n-2) \end{vmatrix} = \begin{vmatrix} f(n+1) \\ f(n) \\ f(n-1) \end{vmatrix}$$

Explanation



New Problem

find $f(n) = f(n-1) + f(n-2) + c$ where c is any constant.

Approach

We will consider three states:

$$M \times \begin{vmatrix} f(n) \\ f(n-1) \\ c \end{vmatrix} = \begin{vmatrix} f(n+1) \\ f(n) \\ c \end{vmatrix}$$

Solution

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

Homework

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

$$\begin{vmatrix} a & 0 & c & d & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{vmatrix}$$

Sum of Fibonacci numbers

Let’s go back to Fibonacci numbers:

1. $F_0 = F_1 = 1$;
2. $F_i = F_{i-1} + F_{i-2}$ for $i \geq 2$.

Introduce a new sequence: $P_i = F_0 + F_1 + \ldots + F_i$ (i.e. sum of first i Fibonacci numbers).

The problem is: find P_N modulo 1000000007 for N up to 10^{18}.

$$P(n) = f(0) + f(1) + \dots + f(n)$$

$$P(n) = P(n-1) + f(n).$$

$$M \times A = B$$

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

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix} \times \begin{bmatrix} P(n) \\ f(n-1) \\ f(n) \end{bmatrix} = \begin{bmatrix} P(n+1) \\ f(n) \\ f(n+1) \end{bmatrix}$$

$$\begin{bmatrix} x_1 \cdot P(n) + y_1 \cdot f(n-1) + z_1 \cdot f(n) \\ x_2 \cdot P(n) + y_2 \cdot f(n-1) + z_2 \cdot f(n) \\ x_3 \cdot P(n) + y_3 \cdot f(n-1) + z_3 \cdot f(n) \end{bmatrix} = \begin{bmatrix} P(n+1) \\ f(n) \\ f(n+1) \end{bmatrix}$$

$$\Rightarrow x_1 \cdot P(n) + y_1 \cdot f(n-1) + z_1 \cdot f(n) = P(n+1)$$

But, $P(n+1) = P(n) + f(n+1)$

$$\Rightarrow P(n+1) = P(n) + f(n) + f(n-1).$$

$$\Rightarrow x_1 = 1, \; y_1 = 1 \text{ and } z_1 = 1.$$

Also, $x_2 \cdot P(n) + y_2 \cdot f(n-1) + z_2 \cdot f(n) = f(n)$

$\Rightarrow x_2 = 0, y_2 = 0, z_2 = 1$

Also, $x_3 \cdot P(n) + y_3 \cdot f(n-1) + z_3 \cdot f(n) = f(n+1)$

But $f(n+1) = f(n) + f(n-1)$

$\Rightarrow x_3 = 0, y_3 = 1$ and $z_3 = 1$

$$\therefore M = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

New Problem

Find $g(n) = 2g(n-1) + 2g(n-2) + f(n)$, where, $f(n) = 2f(n-1) + 2f(n-2)$.

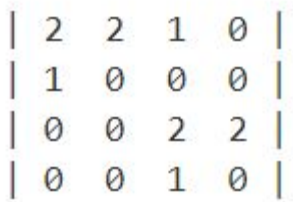
Matrix A

$$\begin{vmatrix} g(n) \\ g(n-1) \\ f(n+1) \\ f(n) \end{vmatrix}$$

Matrix B

$$\begin{vmatrix} g(n+1) \\ g(n) \\ f(n+2) \\ f(n+1) \end{vmatrix}$$

Find M?



New Problem

Find f(n) = f(n-1) + f(n-2) + n.

Approach

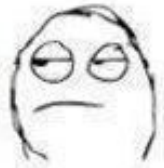
Define g(n) = n => g(n) = g(n-1) + 1, g(1) = 1.

Then, f(n+1) = f(n) + f(n-1) + g(n+1).

Homework

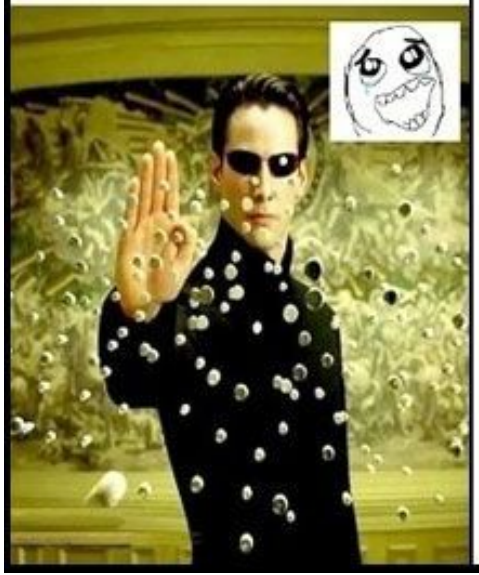
Lockdown Loss | CodeChef

Teacher



Today, we are going to learn about matrix

Expectation



Reality

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$


" THE FEARS WE DON'T FACE
BECOME OUR LIMITS "
Unknown
epic quotes

Thank You!