Applications of Matrix exponentiation

1. **Nth term of Fibonacci sequence**

   A Fibonacci sequence is defined as: $F_n = F_{n-1} + F_{n-2}$, n>=2 and $F_0 = 0$, $F_1 = 1$. For calculating the nth term of the Fibonacci sequence denoted by F_n, one of the simplest approaches is to iteratively compute the Fibonacci numbers till n.

   Pseudocode:

```
//  The function takes input n and returns the nth term of the Fibonacci sequence
function Fibonacci(n)

        //  Base case
        if n equals 0 or 1
                return n

        //  Initializing a and b to F0 and F1
        a = 0
        b = 1

        for i = 2 to n
                c = a + b
                a = b
                b = c

        return c
```

Time complexity: O(n), as we need n iterations to compute the nth term of the Fibonacci sequence

The above approach is not feasible for calculating the nth term of Fibonacci sequence for large n(say $n = 10^{18}$) as the number of operations required will be $O(10^{18})$.

Let us represent our linear recurrence relation $F_n = F_{n-1} + F_{n-2}$ in the form of a matrix of dimensions 2 x 1 as

$$\begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix}$$

Now we want to find a matrix 'M'(coefficient matrix) such that M multiplied by F_{n-1} results in the next term represented as

$$\begin{bmatrix} F_{n} \\ F_{n-1} \end{bmatrix}$$

Hence,

$$M * \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix} = \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

Now, the first question is what will be the dimensions of the matrix M?
Let the dimensions of the matrix be n x m, from the above equation it is clear that (n x m).(2 x 1) = (2 x 1) and for the product to be defined m = 2(number of columns = no of rows) and n = 2(as the resulting matrix is having 2 rows).

Hence our equation becomes,

$$\begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} * \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix} = \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

The second question is what will be the values of the matrix M?
Let us try to write the result in the form of matrix multiplication, we have
$F_n = F_{n-1}.m_{11} + F_{n-2}.m_{12}$ -(1)
$F_{n-1} = F_{n-1}.m_{21} + F_{n-2}.m_{22}$ -(2)

From equation(1) it is clear that m_{11} = 1 and m_{12} = 1 as our recurrence relation is defined as $F_n = 1.F_{n-1} + 1.F_{n-2}$
From equation(2) it is clear that m_{21} = 1 and $m_{22,}$ = 0 in order to make both sides of the equation equal.
Hence our equation becomes,

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} * \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix} = \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

Now since we have found the matrix 'M', we can represent $\begin{pmatrix} F_2 \\ F_1 \end{pmatrix}$ as:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} * \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} F_2 \\ F_1 \end{pmatrix}$$

We can also compute $\begin{pmatrix} F_3 \\ F_2 \end{pmatrix}$ as:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} * \begin{pmatrix} F_2 \\ F_1 \end{pmatrix} = \begin{pmatrix} F_3 \\ F_2 \end{pmatrix}$$

$$\begin{pmatrix} F_2 \\ F_1 \end{pmatrix}$$

Substituting the value of from (1) in (2), we get

$$\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix} * \begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix} * \begin{pmatrix}1\\ 0\end{pmatrix} = \begin{pmatrix}F_3\\ F_2\end{pmatrix}$$

Hence, we can generalize the above equation to compute the nth term of the sequence as:
where F_n will represent the nth term of the Fibonacci sequence.

$$\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix} * \begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix} \textbf{ * .... n - 1 times } * \begin{pmatrix}1\\ 0\end{pmatrix} = \begin{pmatrix}F_n\\ F_{n-1}\end{pmatrix}$$

How can we compute M^n efficiently?
Using matrix exponentiation we can calculate M^n efficiently and hence the time complexity to find the nth term of the Fibonacci sequence becomes $O(\log_2 n * 2^3)$, which is much better than O(n) and allows us to compute the nth term of the Fibonacci sequence efficiently for large n (say $n = 10^{18}$).

Pseudocode:

```
//  The function takes input n and returns the nth term of the Fibonacci sequence
function Fibonacci(n)

        //  Defining '2 x 1' matrix - "base", defining the base case
        base = [(1 0)]

        //  Defining matrix "M"
        M = [(1 1),(1 0)]

        //  For nth term of the fibonacci sequence, we need to multiply "M" n - 1 times
        res = matrixExpo(M, 2, n - 1)

        //  Finally getting the resultant matrix after multiplying by base matrix
        res = matrixMultiply(res,base,2,2,1)

        /*
                Return the the entry corresponding to first row and first column denoting the nth
                term of the Fibonacci sequence
        */
        return res[0][0]
```

Time complexity: $O(\log_2 n * 2^3)$, where n denotes the nth term of the Fibonacci sequence and '2' is the dimension of the coefficient matrix M.
Note: The functions used in the pseudocode are defined above.

2. N th term of linear recurrence relation

Let us define a general linear recurrence relation, where the nth term of the sequence is given as:

$$\mathbf{A_n = c_1.A_{n-1} + c_2.A_{n-2} + c_3.A_{n-3} + \dots. + c_k.A_{n-k} \text{ or } A_n = \sum_{i=1}^{k} c_i.A_{n-i}}$$

The above relation is a recurrence relation as computing the value of A_n is dependent on the terms A_j $(j<n)$ and is linear in nature as the dependence of previous terms(A_j, $j<n$) is linear.

One of the approaches to calculate the nth term of the sequence is to iteratively calculate the next terms and use those values to calculate the value of the next term similar to the iterative version of the Fibonacci sequence.

Instead, let us try to find the nth term of the sequence defined by the above linear recurrence relation using matrix exponentiation. Let us represent A_{n-1}, A_{n-2}, A_{n-3}, A_{n-4}, ….., A_{n-k} in the form
of k x 1 matrix given as:

$$\begin{pmatrix} A_{n-1} \\ A_{n-2} \\ A_{n-3} \\ \dots. \\ A_{n-k} \end{pmatrix}$$

As a part of next step, let’s define a matrix ‘M’(coefficient matrix), as discussed earlier the dimensions of the matrix M will be k x k. Let us try to compute the values of the matrix M, writing the results of matrix multiplication in the form of equations as:

$A_n \;= m_{11}.A_{n-1} + m_{12}.A_{n-2} + m_{13}.A_{n-3} + \dots. + m_{1k}.A_{n-k}$ -(1)
$A_{n-1} = m_{21}.A_{n-1} + m_{22}.A_{n-2} + m_{23}.A_{n-3} + \dots. + m_{2k}.A_{n-k}$ -(2)
.
.
.
$A_{n-k+1} = m_{k1}.A_{n-1} + m_{k2}.A_{n-2} + m_{k3}.A_{n-3} + \dots. + m_{kk}.A_{n-k}$ -(k)

Comparing equation(1) with the linear recurrence relation and other equations for equality we obtain matrix M as:

$$\begin{pmatrix} c_1 & c_2 & c_3 & \dots & c_{k-1} & c_k \\ 1 & 0 & 0 & \dots & & 0 \\ 0 & 1 & 0 & \dots & & 0 \\ 0 & 0 & 1 & \dots & & 0 \\ \dots & & & & & \\ 0 & 0 & 0 & \dots & 1 & 0 \end{pmatrix}$$

Hence, in order to compute the n^{th} term of the sequence, we can use matrix exponentiation as explained in the above section to compute Fibonacci numbers.

Matrix exponentiation is also useful in various **dynamic programming** problems, where the recurrence relation can be expressed as some linear combination of previous terms.

Note: There are various problems that require doing modulo division by some number (say 1000000007), you can simply modify the above functions using modular arithmetic.
As an exercise, try finding out the coefficient matrix 'M' for the following linear recurrence relations:

1. $A_n = 2.A_{n-1} + 3.A_{n-2} + 4.A_{n-3}$
2. $A_n = 6.A_{n-1} + 5.A_{n-2} + 3$
3. $A_n = A_{n-1} + A_{n-2} + A_{n-1}.A_{n-2}$