Matrix Exponentiation

The concept of matrix exponentiation in its most general form is very useful in solving questions that involve calculating the n^{th} of a linear recurrence relation in time of the order of log(n).

Matrix
A matrix is basically a rectangular arrangement of numbers into rows and columns.

'M' is a matrix of n rows and m columns(n x m), where a_{ij} represents the entry present at i^{th} row(1<=i<=n) and j^{th} column(1<=j<=m).

In programming, a matrix is represented as a 2-D array. **For example:** A matrix 'M' with 4 rows and 3 columns is represented as a 2-D array as:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$$

Matrix multiplication
The product of two matrices is defined if the number of columns in the first matrix is equal to the number of rows in the second matrix. If the product is defined, the resulting matrix will have the number of rows equal to the rows of the first matrix and the number of columns equal to the columns of the second matrix.
For example: If 'A' is a 5 x 4 matrix and 'B' is a 4 x 3 matrix, then the product A.B is defined, and the resulting matrix 'C' = A.B will be a 5 x 3 matrix.

Let 'A' be a n x m matrix and 'B' be a m x k matrix, since the product is defined, then the resulting matrix 'C' of order n x k is given as:

$\mathbf{c_{ik}} = \sum_{x=1}^{m} \mathbf{a_{ix}} * \mathbf{b_{xk}}$, where a_{ix} represents the entry in 'A' at i^{th} row and x^{th} column and b_{xj} represents the entry in 'B' at x^{th} row and k^{th} column.
For example:

Let 'A' be a matrix of dimensions 2 x 2:

$$\begin{pmatrix} 2 & 3 \\ 1 & 5 \end{pmatrix}$$

Let 'B' be a matrix of dimensions 2 x 3:

$$\begin{pmatrix} 4 & 2 & 1 \\ 2 & 4 & 3 \end{pmatrix}$$

We define a matrix C = A.B of order 2 x 3:

$$\begin{pmatrix} 2.4 + 3.2 & 2.2 + 3.4 & 2.1 + 3.3 \\ 1.4 + 5.2 & 1.2 + 5.4 & 1.1 + 5.3 \end{pmatrix}$$

The resulting matrix C:

$$\begin{pmatrix} 14 & 16 & 11 \\ 14 & 22 & 16 \end{pmatrix}$$

Number of operations in matrix multiplication

From the above example, matrix 'C' has n.k elements, and for the computation of each element we perform m operations, so the total number of operations required are **O(n*k*m)**.

Pseudocode:

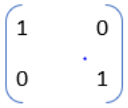
```
/*
        The function takes input matrices A and B of dimensions 'n x m' and 'm x k' respectively.
*/
function matrixMultiply(A,B,n,m,k)

        //  Define a matrix C of dimensions n x k
        C[n][k]

        for i = 0 to n - 1
                for j = 0 to k - 1
                        C[i][j] = 0
                        for r = 0 to m - 1
                                C[i][j] = C[i][j] + A[i][r] * B[r][j]

        return C
```

Time complexity: O(n*k*m), as we have three for loops, for every entry c_{ik} of C, we are performing m operations.

Properties of matrix multiplication

- Matrix multiplication is **not commutative**, in other words, the order of multiplication of matrices matters, so A.B ≠ B.A
- Matrix multiplication is **associative**, so A.(B.C) = (A.B).C, given the product is defined over the matrices A, B, C.
- Matrix multiplication is **distributive**, so A.(B + C) = A.B + B.C, given the product is defined over the matrices A, B, C.

- Given a 'n x n' matrix 'M'(also known as a 'square' matrix), then multiplying the matrix with an Identity matrix I_n results in the same matrix 'M'. This property is known as **multiplicative identity** property. The identity matrix is a square matrix(matrix with the same number of rows and columns) denoted by I_n having order 'n x n', with all entries on the main diagonal as '1' and other entries are '0'.
  For example: The identity matrix of dimension 2 x 2(I_2) is given as:



Matrix exponentiation
Let 'M' be a square matrix having n rows and n columns, we define matrix exponentiation as:
M^x = M * M * M * ... * M (x times), where x is a non-negative integer.
Note: $M^0 = I_n$ (Identity matrix of order n)

Pseudocode:

```
/*
        The function takes input matrices A and B of dimensions 'n x m' and 'm x k' respectively.
*/
function matrixMultiply(A,B,n,m,k)

        // Define a matrix C of dimensions n x k
        C[n][k]

        for i = 0 to n - 1
                for j = 0 to k - 1
                        C[i][j] = 0
                        for r = 0 to m - 1
                                C[i][j] = C[i][j] + A[i][r] * B[r][j]

        return C


//   The function takes input square matrix M of dimension 'n x n'
function matrixPower(M,n,x)

        //   If exponent is 0, return Identity matrix I_n
        if x equals 0
                return I_n

        //   Recurse for lower powers
        return matrixMultiply(M, matrixPower(M, x - 1), n, n, n)
```

Time complexity: O(n^3 * x), as we are making x recursive calls and in each recursive call we are performing multiplication of matrices of order 'n x n'.

We can optimize the above approach by using an idea similar to binary exponentiation. where we split the exponent in its binary representation.
For example: Let 'A' be the square matrix of order n(n x n), and the exponent be 11, so:
$A^{11} = A^{1011} = A^8 . A^2 . A^1$
Basically we took the binary representation of 11(1011_2) which helped us in representing 11 as sum of powers of 2:
$11 = 2^3 + 2^1 + 2^0 = 8 + 2 + 1$
So, $A^{11} = A^8 . A^2 . A^1$

Since exponent 'x' has only floor($\log_2 x$)+1 digits in its binary representation, so we only need to perform O($\log_2 x$) multiplications.

Hence, we can modify our recursive approach as:

$$M^x \Rightarrow \begin{cases} 1 & \text{if } x = 0 \\ (M^{x/2})^2 & \text{if } x > 0 \text{ and } x \text{ is even} \\ M*(M^{x/2})^2 & \text{if } x > 0 \text{ and } x \text{ is odd} \end{cases}$$

Pseudocode:

```
/*
        The function takes input matrices A and B of dimensions 'n x m' and 'm x k' respectively.
*/
function matrixMultiply(A, B, n, m, k)

        //  Define a matrix C of dimensions n x k
        C[n][k]

        for i = 0 to n - 1
                for j = 0 to k - 1
                        C[i][j] = 0
                        for r = 0 to m - 1
                                C[i][j] = C[i][j] + A[i][r] * B[r][j]

        return C


//  The function takes input square matrix M of dimension 'n x n'
function matrixExpo(M, n, x)

        //  If exponent is 0, return Identity matrix I_n
        if x equals 0
                return I_n

        //  If x is divisible by 2
```

```
    res = matrixPower(M, x/2), n, n, n)
    return matrixMultiply(res, res, n, n, n)

    //   If x is not divisible by 2
    res = matrixPower(M, x/2), n, n, n)
    res = matrixMultiply(res, res, n, n, n)
    return matrixMultiply(res, M, n, n, n)
```

Time complexity: O($\log_2 x * n^3$), as we make $\log_2 x$ recursive calls and in each recursive call we are performing multiplication of matrices of order 'n x n'.