---

1 Matrix Multiplication

In order to perform matrix multiplication, the columns of the first matrix MUST be equal to the rows of the second matrix. As an example, let us assign matrix A = i by j and matrix B = j by k. In order to perform A x B, the columns of A (j) must equal the rows of B (j). Multiplying A x B will result in an i by k matrix. Observe in the following example.

$$A = \begin{bmatrix} 1, 2, 3 \\ 4, 5, 6 \end{bmatrix} B = \begin{bmatrix} 7, 8, 9, 1 \\ 2, 3, 4, 5 \\ 6, 7, 8, 9 \end{bmatrix}$$

$$A \times B = \begin{bmatrix} x, x, x, x \\ x, x, x, x \end{bmatrix} or\ [2 \times 4]$$

The end result in this example is a 2 by 4 matrix. To calculate each value of the multiplied matrix, the rows of the first matrix need to be multiplied by the columns of the second matrix. These values are then added together for each row and column combination. Let us continue from our previous example and continue the calculation to find A x B.

$$A = \begin{bmatrix} 1, 2, 3 \\ 4, 5, 6 \end{bmatrix} B = \begin{bmatrix} 7, 8, 9, 1 \\ 2, 3, 4, 5 \\ 6, 7, 8, 9 \end{bmatrix}$$

$$\begin{bmatrix} ((1*7)+(2*2)+(3*6)), ((1*8)+(2*3)+(3*7)), ((1*9)+(2*4)+(3*8)), 38 \\ ((4*7)+(5*2)+(6*6)), ((4*8)+(5*3)+(6*7)), 104, 83 \end{bmatrix}$$

$$A \times B = \begin{bmatrix} 29, 35, 41, 38 \\ 74, 89, 104, 83 \end{bmatrix}$$

1.1 Time Complexity

From section 1, we can derive the time complexity for triple nested for loop to calculate A x B to be T(n)= i x j x k with these values being defined in the section 1. The basic operation in this algorithm is the multiplication between the rows and columns.

1.2 Chained Matrix Multiplication

Now let us try chained matrix multiplication. Chained matrix multiplication involves more than two matrices multiplied together to create a product. Let A = m by p, B = p by q, and C = q by n. From the properties explained in section 1, our new matrix will be of size m by n. If we wanted to multiply A x B x C, we would have:

$$A \times B \times C = [m \times p] \times [p \times q] \times [q \times n] = [m \times n]$$

We also know that matrices follow the associative property. This means that when we multiply A x B x C, we can either do (A x B) x C or A x (B x C). This leads to the same answer but a different number of operations depending on the matrix sizes. Let us derive the number of possible combinations the associative property allows for n number of matrices.

$$n = 2 : A \times B = (A \times B) = 1$$

$$n = 3 : A \times B \times C = (A \times B) \times C, A \times (B \times C) = 2$$

$$n = 4 : A \times B \times C \times D = (A \times B) \times C \times D, A \times (B \times C) \times D, A \times B \times (C \times D), (A \times B) \times (C \times D) = 4$$

$$n = 5 : ... = 16$$

$$n = 6 : ... = 32$$

$$t_n \geq 2^{n-2}$$

We observe that for any n amount of matrices, the number of associative property permutations is always $t_n \geq 2^{n-2}$, with t_n being the number of different possible orderings. This relationship is exponential and we will utilize dynamic programming to calculate the optimal ordering for n matrices to minimize the number of operations. This includes optimal ordering for multiplying any subset of the n matrices, as shown below.

Let there be 6 matrices: A_1 through A_6
Let us suppose that the optimal ordering is given as

$$A_1((((A_2A_3)A_4)A_5)A_6)$$

The principle of optimality states that the subsets within the optimal ordering will also be in their optimal ordering. Given this principle, the subset below is also in optimal ordering for A_2, A_3, and A_4

$$(A_2A_3)A_4$$

1.3 Dynamic Programming Application

Because of the principle of optimality, we are able to apply dynamic programming to problems. To do this, we utilize tables to look up solutions of smaller instances. Examples of this would be utilizing a table with one row to solve the Fibonacci problem dynamically. To multiply n matrices, a n by n table, M, indexed from 1 to n will be used.

When i < j
The table M[i][j] will contain the minimum number of multiplications needed to multiply matrices A_i to A_j

When i = j
M[i][i] = 0