Matrix Multiplication Algorithm Time Complexity

Last modified: August 25, 2021

by Subham Datta
(https://www.baeldung.com/cs/author/subhamdatta)

Algorithms (https://www.baeldung.com/cs/category/algorithms)

Math and Logic (https://www.baeldung.com/cs/category/core-concepts/math-logic)

Complexity (https://www.baeldung.com/cs/tag/complexity)

1. Overview

Matrix multiplication is an important operation in mathematics. It is a basic linear algebra tool and has a wide range of applications in several domains like physics, engineering, and economics.

In this tutorial, we'll discuss two popular matrix multiplication algorithms: the naive matrix multiplication and the Solvay Strassen algorithm. We'll also present the time complexity analysis of each algorithm.

2. Introduction to Matrix Multiplication

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \cdots & A_{1p} \\ A_{21} & A_{22} & A_{23} & \cdots & A_{2p} \\ A_{31} & A_{32} & A_{33} & \cdots & A_{3p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{s1} & A_{s2} & A_{s3} & \cdots & A_{sp} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} & B_{13} & \cdots & B_{1t} \\ B_{21} & B_{22} & B_{23} & \cdots & B_{2t} \\ B_{31} & B_{32} & B_{33} & \cdots & B_{3t} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ B_{p1} & B_{p2} & B_{p3} & \cdots & B_{pt} \end{bmatrix}$$

Now when we multiply the matrix A by the matrix B, we get another matrix – let's name it D. The order of the matrix D would be $s \times t$.

Let's now look into elements the matrix D:

$$D = AB = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \cdots & A_{1p} \\ A_{21} & A_{22} & A_{23} & \cdots & A_{2p} \\ A_{31} & A_{32} & A_{33} & \cdots & A_{3p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{s1} & A_{s2} & A_{s3} & \cdots & A_{sp} \end{bmatrix} . \begin{bmatrix} B_{11} & B_{12} & B_{13} & \cdots & B_{1t} \\ B_{21} & B_{22} & B_{23} & \cdots & B_{2t} \\ B_{31} & B_{32} & B_{33} & \cdots & B_{3t} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ B_{p1} & B_{p2} & B_{p3} & \cdots & B_{pt} \end{bmatrix} = \begin{bmatrix} D_{11} & D_{12} & D_{13} & \cdots & D_{1t} \\ D_{21} & D_{22} & D_{23} & \cdots & D_{2t} \\ D_{31} & D_{32} & D_{33} & \cdots & D_{3t} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ D_{s1} & D_{s2} & D_{s3} & \cdots & D_{st} \end{bmatrix}$$

Each entries D_{ij} in the matrix D can be calculated from the entries of the matrix A and B by finding pairwise summation:

3. Matrix Multiplication Properties

Let $A1$, $A2$ and $A3$ be three matrices of the same dimensions.

According to the associative property in multiplication, we can write $A1(A2A3) = (A1A2)A3$. **This property states that we can change the grouping surrounding matrix multiplication, and it'll not affect the output of the matrix multiplication.**

We can distribute the matrices in a similar way we distribute the real numbers. Using distributive property in multiplication we can write: $A1(A2 + A3) = A1A2 + A1A3$.

There are some special matrices called an identity matrix or unit matrix which has 1 in the main diagonal and 0 elsewhere. Some examples of identity matrices are:

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \\ 0 & 0 & 1 \end{bmatrix}, \quad I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

There is a very interesting property in matrix multiplication. When a $\mathrm{M} \times \mathrm{N}$ matrix is multiplied on the right by a $\mathrm{N} \times \mathrm{N}$ identity matrix, the output matrix would be same as $\mathrm{M} \times \mathrm{N}$ matrix. This property is called multiplicative identity. For example:

$I_4 A_{(4\times4)} = A_{(4\times4)} I_4 = A_{(4\times4)}$

It is important to note that matrix multiplication is not commutative. Suppose we multiply two matrices A and B of the same order $m \times n$ then $AB \neq BA$. This is the general case. But if A and B both are diagonal matrix (https://en.wikipedia.org/wiki/Diagonal_matrix) and have the same dimensions, they hold the commutative property.

4. The Naive Matrix Multiplication Algorithm

4.1. Pseudocode

Let's see the pseudocode of the naive matrix multiplication algorithm first, then we'll discuss the steps of the algorithm:

Algorithm 1: The Naive Matrix Multiplication Algorithm

```
Data: S[A][B], P[G][H]
Result: Q[][]
if B == G then
    for m = 0; m < A; m++ do
        for r = 0; r < H; r++ do
            Q[m][r] = 0;
            for k = 0; k < G; k++ do
                Q[m][r] += S[m][k] * P[k][r];
            end
        end
    end
```


To find an implementation of it, we can visit our article on Matrix Multiplication in Java (/java-matrix-multiplication).

4.2. Time Complexity Analysis

The naive matrix multiplication algorithm contains three nested loops (/java-breaking-out-nested-loop). For each iteration of the outer loop, the total number of the runs in the inner loops would be equivalent to the length of the matrix. Here, integer operations take $\mathcal{O}(1)$ time. **In general, if the length of the matrix is $\mathbf{N}$, the total time complexity would be $\mathcal{O}(\mathbf{N} * \mathbf{N} * \mathbf{N}) = \mathcal{O}(\mathbf{N^3})$.**

Now the question is, can we improve the time complexity of the matrix multiplication? We'll discuss an improved matrix multiplication algorithm in the next section.

5. The Solvay Strassen Algorithm

5.1. Algorithm

In the year 1969, Volker Strassen made remarkable progress, proving the complexity $\mathcal{O}(N^3)$ was not optimal by releasing a new algorithm, named after him.

along with a nice math trick to solve the matrix multiplication problem with low computation.

Let's take two input matrices S and P of order 2×2. In general, the dimension of the input matrices would be $N \times N$:

$$S = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad P = \begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

First step is to divide each input matrix into four submatrices of order $\mathbf{N/2 \times N/2}$:

$$S = \left(\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right), P = \left(\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right)$$

Next step is to perform 10 addition/subtraction operations:

$$T[1] = (f - h)$$
$$T[2] = (a + b)$$

$$T[3] = (c + d)$$

$$T[4] = (g - e)$$
$$T[5] = (a + d)$$
$$T[6] = (e + h)$$
$$T[7] = (b - d)$$
$$T[8] = (g + h)$$
$$T[9] = (a - c)$$
$$T[10] = (e + f)$$

The third step of the algorithm is to calculate 7 multiplication operations recursively using the previous results. Here each $R[i], i \in \{1, .., 7\}$ is of size $N/2 \times N/2$:

$$R[1] = S[1][1]T[1] = a(f - h)$$
$$R[2] = P[2][2]T[2] = h(a + b)$$
$$R[3] = P[1][1]T[3] = e(c + d)$$
$$R[4] = S[2][2]T[4] = d(g - e)$$
$$R[5] = T[5]T[6] = (a + d)(e + h)$$
$$R[6] = T[7]T[8] = (b - d)(g + h)$$
$$R[7] = T[9]T[10] = (a - c)(e + f)$$

$$Q_{11} = R[5] + R[4] - R[2] + R[6]$$
$$Q_{12} = R[1] + R[2]$$
$$Q_{21} = R[3] + R[4]$$
$$Q_{22} = R[5] + R[1] - R[3] - R[7]$$

Now let's put everything together in matrix form:

$$\left(\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right)\left(\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right) = \begin{pmatrix} R[5] + R[4] - R[2] + R[6] & R[1] + R[2] \\ R[3] + R[4] & R[1] + R[5] - R[3] -$$

So as we can see, this algorithm needs to perform 7 multiplication operations, unlike the naive algorithm, which needs 8 multiplication operations.

It is important to note that this algorithm works only on square matrices (https://en.wikipedia.org/wiki/Square_matrix) with the same dimensions.

5.2. Time Complexity Analysis

This solution is based on recursion (/java-recursion). In the first step, we divide the input matrices into submatrices of size $N/2 \times N/2$. This step can be performed in $\mathcal{O}(1)$ times.

In step 2, we calculate 10 addition/subtraction operations which takes $\mathcal{O}(N^2)$ time.

In step 3, we make 7 recursive calls to calculate $R1$ to $R7$. The output of this step would be 7 matrix of order $N \times 2$. This step takes $7T(N/2)$ time.

Finally, by adding and subtracting submatrices of $R[i]$, we get our resultant matrix Q. The time complexity of this step would be $\mathcal{O}(N^2)$.

Therefore the total time complexity of this algorithm would be:

$$\mathbf{T(N) = 7T(N/2) + \mathcal{O}(N^2) = \mathcal{O}(N^{\log_2 7}) \approx \mathcal{O}(N^{2.8074})}$$

6. Comparison Between Two Algorithms

Let's summarize two matrix multiplication algorithms in this section and let's put the key points in a table:

Parameter	The Naive Algorithm	The Solvay Strassen Algorithm
Time Complexity	$\mathcal{O}(N^3)$	$\mathcal{O}(N^{2.8074})$
Approach	Iterative	Divide-and-conquer
Application	Square as well as non-square matrices	Suitable for square matrices only

7. Conclusion

In this tutorial, we've discussed two algorithms for matrix multiplication: the naive method and the Solvay Strassen algorithm in detail.

We also presented a comparison including the key points of these two algorithms.

Comments are closed on this article!

CATEGORIES

ALGORITHMS (/CS/CATEGORY/ALGORITHMS)
ARTIFICIAL INTELLIGENCE (/CS/CATEGORY/AI)
CORE CONCEPTS (/CS/CATEGORY/CORE-CONCEPTS)
DATA STRUCTURES (/CS/CATEGORY/DATA-STRUCTURES)
GRAPH THEORY (/CS/CATEGORY/GRAPH-THEORY)
LATEX (/CS/CATEGORY/LATEX)
NETWORKING (/CS/CATEGORY/NETWORKING)
SECURITY (/CS/CATEGORY/SECURITY)

SERIES

DRAWING CHARTS IN LATEX (/CS/CATEGORY/SERIES)

ABOUT

ABOUT BAELDUNG (HTTPS://WWW.BAELDUNG.COM/ABOUT)
THE FULL ARCHIVE (/CS/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (HTTPS://WWW.BAELDUNG.COM/EDITORS)

TERMS OF SERVICE (HTTPS://WWW.BAELDUNG.COM/TERMS-OF-SERVICE)
PRIVACY POLICY (HTTPS://WWW.BAELDUNG.COM/PRIVACY-POLICY)
COMPANY INFO (HTTPS://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)