# Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)

**BG geeksforgeeks.org**/strassens-matrix-multiplication

March 29, 2014

Given two square matrices A and B of size n x n each, find their multiplication matrix.

#### Naive Method

Following is a simple way to multiply two matrices.

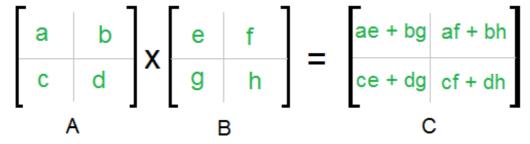
```
void multiply( int A[][N], int B[][N], int C[][N])
{
for ( int i = 0; i < N; i++)
{
    for ( int j = 0; j < N; j++)
    {
        C[i][j] = 0;
    for ( int k = 0; k < N; k++)
        {
        C[i][j] += A[i][k]*B[k][j];
        }
    }
}</pre>
```

Time Complexity of above method is  $O(N^3)$ .

## **Divide and Conquer**

Following is simple Divide and Conquer method to multiply two square matrices.

- 1) Divide matrices A and B in 4 sub-matrices of size  $N/2 \times N/2$  as shown in the below diagram.
- 2) Calculate following values recursively. ae + bg, af + bh, ce + dg and cf + dh.



A, B and C are square metrices of size N x N a, b, c and d are submatrices of A, of size N/2 x N/2  $\,$ 

e, f, g and h are submatrices of B, of size N/2 x N/2

In the above method, we do 8 multiplications for matrices of size  $N/2 \times N/2$  and 4 additions. Addition of two matrices takes  $O(N^2)$  time. So the time complexity can be written as

 $T(N) = 8T(N/2) + O(N^2)$ 

From <u>Master's Theorem</u>, time complexity of above method is  $O(N^3)$  which is unfortunately same as the above naive method.

## Simple Divide and Conquer also leads to $O(N^3)$ , can there be a better way?

In the above divide and conquer method, the main component for high time complexity is 8 recursive calls. The idea of **Strassen's method** is to reduce the number of recursive calls to 7. Strassen's method is similar to above simple divide and conquer method in the sense that this method also divide matrices to sub-matrices of size N/2 x N/2 as shown in the above diagram, but in Strassen's method, the four sub-matrices of result are calculated using following formulae.

$$p1 = a(f - h)$$
  $p2 = (a + b)h$   
 $p3 = (c + d)e$   $p4 = d(g - e)$   
 $p5 = (a + d)(e + h)$   $p6 = (b - d)(g + h)$   
 $p7 = (a - c)(e + f)$ 

The A x B can be calculated using above seven multiplications. Following are values of four sub-matrices of result C

 $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \mathbf{X} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} p5 + p4 - p2 + p6 & p1 + p2 \\ \hline p3 + p4 & p1 + p5 - p3 - p7 \end{bmatrix}$ 

A, B and C are square metrices of size N x N

- a, b, c and d are submatrices of A, of size N/2 x N/2
- e, f, g and h are submatrices of B, of size N/2 x N/2
- p1, p2, p3, p4, p5, p6 and p7 are submatrices of size N/2 x N/2

#### **Time Complexity of Strassen's Method**

Addition and Subtraction of two matrices takes O(N<sup>2</sup>) time. So time complexity can be written as

```
T(N) = 7T(N/2) + O(N^2)
```

From Master's Theorem, time complexity of above method is  $O(N^{Log7})$  which is approximately  $O(N^{2.8074})$ 

Generally Strassen's Method is not preferred for practical applications for following reasons.

- 1) The constants used in Strassen's method are high and for a typical application Naive method works better.
- 2) For Sparse matrices, there are better methods especially designed for them.
- 3) The submatrices in recursion take extra space.
- 4) Because of the limited precision of computer arithmetic on noninteger values, larger errors accumulate in Strassen's algorithm than in Naive Method (Source: <u>CLRS Book</u>)

```
import numpy as np
def split(matrix):
row, col = matrix.shape
row2, col2 = row / / 2 , col / / 2
return matrix[:row2, :col2], matrix[:row2, col2:], matrix[row2:, :col2], matrix[row2:,
col2:1
def strassen(x, y):
if len (x) = 1 :
return x * y
a, b, c, d = split(x)
e, f, g, h = split(y)
p1 = strassen(a, f - h)
p2 = strassen(a + b, h)
p3 = strassen(c + d, e)
p4 = strassen(d, g - e)
p5 = strassen(a + d, e + h)
p6 = strassen(b - d, g + h)
p7 = strassen(a - c, e + f)
c11 = p5 + p4 - p2 + p6
c12 = p1 + p2
c21 = p3 + p4
c22 = p1 + p5 - p3 - p7
c = np.vstack((np.hstack((c11, c12)), np.hstack((c21, c22))))
return c
```

#### **Easy way to remember Strassen's Matrix Equation**

Watch Video At: https://youtu.be/E-QtwPi6201

# **References:**

Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

https://www.youtube.com/watch?v=LOLebQ8nKHA

https://www.youtube.com/watch?v=QXY4RskLQcl

Improved By: PrayushDawda