Divide-and-Conquer: Matrix Multiplication
Strassen's Algorithm

Matrix Multiplication Problem

Matrix Multiplication. Given two matrices:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nk} \end{pmatrix}$$

and

$$B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k1} & b_{k2} & \dots & b_{km} \end{pmatrix}$$

return the matrix

$$C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1m} \\ c_{21} & c_{22} & \dots & c_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nm} \end{pmatrix}$$

where

$$c_{ij} = \sum_{r=1}^{k} a_{ir} \cdot b_{rj}.$$

Step 1: Straightforward Solution.

Algorithm MatrixMultiply in Figure 1 solves the Matrix Multiplication problem in a straighforward manner.

```
Algorithm MatrixMultiply(n,k,m,A[1..n][1..k], B[1..k][1..m])

begin
  C[1..n][1..m];        // define the result matrix
  for i = 1 to n do
    for j = 1 to m do
      c ← 0;
      for s = 1 to k do
        c ← c + A[i][s] * B[s][j];
      end for
      C[i][j] ← c;
    end for
  end for
  return(C);
end
```

Figure 1: Algorithm MatrixMultiply.

Analysis. Correctness is straightforward: the algorithm implements faithfully the definition of the matrix multiplication.

Runtime. Let us assume that $n = \Theta(N)$, $m = \Theta(N)$ and $k = \Theta(N)$. Let us estimate the runtime complexity of Algorithm MatrixMultiply by counting the most expensive operations in the algorithm: the multiplications.

The $c \leftarrow c + A[i][s] * B[s][j]$ assignment statement will be executed exactly $n \cdot m \cdot k$ times. With the assumptions about, we obtain our bound on the runtime of the algorithm: $T(N) = \Theta(N^3)$.

A Divide-And-Conquer Algorithm for Matrix Multiplication

Note. For the sake of simplicity (but without loss of generality) assume that we are multiplying to square $n \times n$ matrices A and B, i.e., $m = n$ and $k = n$.

Key Observation. *Matrix Multiplication can be performed blockwise.*

Let

$$A = \begin{pmatrix} X & Y \\ Z & W \end{pmatrix}, B = \begin{pmatrix} P & Q \\ R & S \end{pmatrix},$$

where $X = \begin{pmatrix} a_{11} & \dots & a_{1\frac{n}{2}} \\ \vdots & \ddots & \vdots \\ a_{\frac{n}{2}1} & \dots & a_{\frac{n}{2}\frac{n}{2}} \end{pmatrix}$, $Y = \begin{pmatrix} a_{1\frac{n}{2}+1} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{\frac{n}{2}\frac{n}{2}+1} & \dots & a_{\frac{n}{2}n} \end{pmatrix}$, and so on.

Then, in fact,

$$A \cdot B = \begin{pmatrix} XP + YR & XQ + YS \\ ZP + WR & ZQ + WS \end{pmatrix}$$

Here XP, YR, XQ, YS, ZP, WR, ZQ and WS are products of the respective matrices X, Y, Z, W, P, Q, R, S and the $+$ operator is the element-by-element matrix addition.

Using this observation, we can devise a divide-and-conquer algorithm for multiplying matrices shown in Figure 2.

This algorithm uses the following supplementary function MatrixSum():

```
Algorithm MatrixSum(n, A[1..n][1..n], B[1..n][1..n])

begin
  C[1..n][1..n];

  for i = 1 to n do
    for j = 1 to n do
      C[i][j] ← A[i][j] + B[i][j];
    end for;
  end for;

  return(C);
end
```

Analysis. Consider the running time of the Algorithm MatrixSum. The assignment operation in that algorithm is performed n^2 times, so, the running time of the algorithm is $O(n^2)$ ($\Theta(n^2)$, in fact).

Now, we can devise the recurrence relation to represent the running time of Algorithm MMDC.

Algorithm MMDC reduces solving problem of multiplying of two $n \times n$ matrices to eight problems of multiplying $\frac{n}{2} \times \frac{n}{2}$ matrices, and computing four $O(n^2)$ matrix sums. Therefore, the recurrence relation for Algorithm MMDC is

$$T(n) = 8T(\frac{n}{2}) + O(n^2).$$

To solve this recurrence relation, observe that in terms of the Master Theorem $a = 8$, $b = 2$ and $\log_b(a) = 3$ and $f(n) = O(n^2) = o(n^{\log_a(b)-\epsilon}) = o(n^{3-0.2}$ for $\epsilon = 0.2$. Therefore, by the Master Theorem,

$$T(n) = O(n^3).$$

This does not improve upon the straightforward algorithm, but as we saw before with finding second largest number problem, this gives us a set up to devise a better algorithm that would not be possible without Divide-and-Conquer.

Strassen's Algorithm

In 1969, *Volker Strassen*, a German mathematician, observed that *we can eliminate **one** matrix multiplication operation* from each round of the divide-and-conquer algorithm for matrix multiplication.

Consider again two $n \times n$ matrices

$$A = \begin{pmatrix} X & Y \\ Z & W \end{pmatrix}, B = \begin{pmatrix} P & Q \\ R & S \end{pmatrix},$$

and recall that

$$A \cdot B = \begin{pmatrix} XP + YR & XQ + YS \\ ZP + WR & ZQ + WS \end{pmatrix}$$

Strassen's Algorithm is based on observing that $XP + YR$, $XQ + YS$, $ZP + WR$ and $ZQ + WS$ can be computed with only **seven** (instead of eight as in Algorithm MMDC) matrix multiplication operations, as follows.

First, compute the following **seven** matrices:

$$P_1 = X(Q - S)$$
$$P_2 = (X + Y)S$$
$$P_3 = (Z + W)P$$
$$P_4 = W(R - P)$$
$$P_5 = (X + W)(P + S)$$
$$P_6 = (Y - W)(R + S)$$
$$P_7 = (X - Z)(P + Q)$$

Note: Computing each of the $P_1, \ldots, P_7$ matrices requires one matrix multiplication operation per matrix.

Second: observe the following equalities:

$$P_5+P_4-P_2+P_6 = (X+W)(P+S)+W(R-P)-(X+Y)S+(Y-W)(R+S) =$$

$$XP+XS+WP+WS+WR-WP-XS-YS+YR-WR+YS-WS = \mathbf{XP} + \mathbf{YR}$$

$$P_1 + P_2 = X(Q - S) + (X + Y)S = XQ - XS + XS + YS = \mathbf{XS} + \mathbf{YS}$$

$$P_3 + P_4 = (Z + W)P + W(R - P) = ZP + WP + WR - WP = \mathbf{ZP} + \mathbf{WR}$$

$$P_1+P_5-P_3-P_7 = X(Q-S)+(X+W)(P+S)-(Z+W)P-(X-Z)(P+Q) =$$

$$XQ-XS+XP+XS+WP+WS-ZP-WP-XP+ZP-XQ+ZQ = \mathbf{ZQ} + \mathbf{WS}$$

That is,

$$A \cdot B = \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{pmatrix}$$

Analysis. We note that a direct implementation of Strassen's Algorithm involves **seven** recursive calls to multiplication problems of size $\frac{n}{2} \times \frac{n}{2}$, but also involves significantly more calls to MatrixSum algorithm that runs in quadratic time. Nevertheless, the $f(n)$ function in terms of the Master Theorem remains $f(n) = O(n^2)$, while the entire recurrence relation becomes

$$T(n) = 7T(\frac{n}{2}) + O(n^2)$$

By Master Theorem, because $n^2 = o(n^{\log_2 7-\epsilon})$, the running time of the Strassen's Algorithm is

$$T(n) = O(n^{\log_2 7}) \approx O(n^{2.81}).$$

Note. This is **not** a tight upper bound on the algorithmic complexity of matrix multiplication. The current best algorithmic bound is $O(n^{2.3728})$. This algorithm, however, and other algorithms similar to it have a very large multiplicative constant associated with the computation, that it is not practical to use.

Algorithm MMDC(n, A[1..n][1..n], B[1..n][1..n])

begin

`// Base Case.` **if** $n = 1$ **then**
return $A[1][1] \cdot B[1][1]$; **end if**
`// Divide the input matrices. Assume this is done in-place,`
`// i.e., X,Y,Z,W,P,Q,R,S are "pointers" to portions of A and B`

$X[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow A[1..\frac{n}{2}][1..\frac{n}{2}]$;
$Y[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow A[1..\frac{n}{2}][\frac{n}{2}+1..n]$;
$Z[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow A[\frac{n}{2}+1..n][1..\frac{n}{2}]$;
$W[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow A[\frac{n}{2}+1..n][\frac{n}{2}+1..n]$;
$P[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow B[1..\frac{n}{2}][1..\frac{n}{2}]$;
$Q[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow B[1..\frac{n}{2}][\frac{n}{2}+1..n]$;
$R[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow B[\frac{n}{2}+1..n][1..\frac{n}{2}]$;
$S[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow B[\frac{n}{2}+1..n][\frac{n}{2}+1..n]$;

`// Solve subporblems`

$XP[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow \mathsf{MMDC}(\frac{n}{2}, X, P)$;
$XQ[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow \mathsf{MMDC}(\frac{n}{2}, X, Q)$;
$YR[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow \mathsf{MMDC}(\frac{n}{2}, Y, R)$;
$YS[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow \mathsf{MMDC}(\frac{n}{2}, Y, S)$;
$ZP[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow \mathsf{MMDC}(\frac{n}{2}, Z, P)$;
$ZQ[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow \mathsf{MMDC}(\frac{n}{2}, Z, Q)$;
$WR[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow \mathsf{MMDC}(\frac{n}{2}, W, R)$;
$WS[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow \mathsf{MMDC}(\frac{n}{2}, W, S)$;

`// Assemble the solution from subproblems`

$C1[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow \mathsf{MatrixSum}(\frac{n}{2}, XP, YR)$;
$C2[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow \mathsf{MatrixSum}(\frac{n}{2}, XQ, YS)$;
$C3[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow \mathsf{MatrixSum}(\frac{n}{2}, ZP, WR)$;
$C4[1..\frac{n}{2}][1..\frac{n}{2}] \leftarrow \mathsf{MatrixSum}(\frac{n}{2}, ZQ, WS)$;

$$C[1..n][1..n] = \begin{pmatrix} C1 & C2 \\ C3 & C4 \end{pmatrix};$$

return(C);

end

Figure 2: Matrix Multiplication using Divide-andC-Conquer approach.