Matrix Multiplication

Problem: Given two $n \times n$ matrices $\mathbf{A}$ and $\mathbf{B}$, find the $n \times n$ matrix $\mathbf{C}$ such that $\mathbf{C} = \mathbf{A} * \mathbf{B}$

The naive algorithm for this task runs in $\Theta(n^3)$ time. We would like to improve on this runtime. Perhaps we could use a

Divide and Conquer Algorithm

Idea: Divide $n \times n$ matrix into 2×2 matrix of $n/2 \times n/2$ submatrices.

$$\mathbf{C} = \mathbf{A} * \mathbf{B} \Rightarrow \begin{bmatrix} \mathbf{r} & \mathbf{s} \\ \mathbf{t} & \mathbf{u} \end{bmatrix} = \begin{bmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{bmatrix} * \begin{bmatrix} \mathbf{e} & \mathbf{f} \\ \mathbf{g} & \mathbf{h} \end{bmatrix}$$

Therefore:

$$\mathbf{r} = \mathbf{ae} + \mathbf{bg}$$
$$\mathbf{s} = \mathbf{af} + \mathbf{bh}$$
$$\mathbf{t} = \mathbf{ce} + \mathbf{dg}$$
$$\mathbf{u} = \mathbf{cf} + \mathbf{dh}$$

So we have 8 recursive multiplies and 4 additions.

Running Time: $T(n) = \Theta(n^3)$

But we can improve this!

Strassen's Algorithm

Define:

$\mathbf{p_1} = \mathbf{a} * (\mathbf{f} - \mathbf{h})$
$\mathbf{p_2} = (\mathbf{a} + \mathbf{b}) * \mathbf{h}$
$\mathbf{p_3} = (\mathbf{c} + \mathbf{d}) * \mathbf{e}$
$\mathbf{p_4} = \mathbf{d} * (\mathbf{g} - \mathbf{e})$
$\mathbf{p_5} = (\mathbf{a} + \mathbf{d}) * (\mathbf{e} + \mathbf{h})$
$\mathbf{p_6} = (\mathbf{b} - \mathbf{d}) * (\mathbf{g} - \mathbf{h})$
$\mathbf{p_7} = (\mathbf{a} - \mathbf{c}) * (\mathbf{e} + \mathbf{f})$

In terms of $\mathbf{p_1}$ through $\mathbf{p_7}$:

$\mathbf{r} = \mathbf{p_5} + \mathbf{p_4} - \mathbf{p_2} + \mathbf{p_6}$
$\mathbf{s} = \mathbf{p_1} + \mathbf{p_2}$
$\mathbf{t} = \mathbf{p_3} + \mathbf{p_4}$
$\mathbf{u} = \mathbf{p_5} + \mathbf{p_1} - \mathbf{p_3} - \mathbf{p_7}$

With these definitions, Strassen's Matrix Multiply Algorithm becomes:

1 Divide: Partition $\mathbf{A}$ and $\mathbf{B}$ into submatrices; add and subtract to form terms.
2 Conquer: 7 recursive multiplications.
3 Combine: Add and subtract terms to form $\mathbf{C}$.

Running Time: $T(n) = \Theta(n^{\lg 7}) = O(n^{2.81})$

Matrix Product Checking

Your friend gives you a program binary that multiplies matrices...or so he claims. You want to verify that your friend's program multiplies matrices correctly, but you don't want to source dive the decompiled binary or spend all the time needed to multiply the two matrices yourself. More formally:

Problem: Given three real $n \times n$ matrices, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, does $\mathbf{C} = \mathbf{A} * \mathbf{B}$?

We can use the following randomized algorithm (due to Freivalds) to solve this in $O(n^2)$ time:

MATRIX-PRODUCT-CHECK$(\mathbf{A}, \mathbf{B}, \mathbf{C})$
1 Pick a vector $r \in \{0,1\}^n$ uniformly at random.
2 **if** $\mathbf{A}(\mathbf{B}r) = \mathbf{C}r$
3 **then** Output "Yes"
4 **else** Output "No"

Claim: If $\mathbf{AB} = \mathbf{C}$ then this algorithm will always output "Yes," and if $\mathbf{AB} \neq \mathbf{C}$ then
Pr[MATRIX-PRODUCT-CHECK$(\mathbf{A}, \mathbf{B}, \mathbf{C})$ outputs "Yes"] $\leq \frac{1}{2}$.

Proof.

The first part of this claim follows from associativity of matrix multiplication. To prove the second part, start by conceptualizing a matrix $\mathbf{D} = \mathbf{AB} - \mathbf{C}$. Because we assume that $\mathbf{AB} \neq \mathbf{C}$, $\mathbf{D} \neq 0$. Therefore there must exist some element in $\mathbf{D}$ that is non-zero; call this non-zero element d_{ij}.

We will now further assume that this algorithm outputs "Yes," despite the fact that $\mathbf{AB} \neq \mathbf{C}$. (We want to examine what values of r could cause this algorithm to mistakenly output "Yes," and deduce that the probability of this r occurring is small.) Examine the ith entry in the product $\mathbf{D}r$. For $\mathbf{D}r$ to be 0 (which corresponds to our algorithm outputting "Yes") the ith entry of $\mathbf{D}r$ must be 0. Therefore

$\{\text{The } i\text{th entry of } \mathbf{D}r\} = \sum_k d_{ik} r_k = 0.$

Since we assume that $d_{ij} \neq 0$, this sum is 0 iff

$r_j = -(\sum_{k \neq j} d_{ik} r_k)/d_{ij}$

Therefore, for a given selection of the other values in r, there is exactly one value of r_j that will make this sum 0. However, we select randomly from two possible values for r_j, so the probability that we select the correct one is $\leq \frac{1}{2}$. It follows that the Pr$[\mathbf{D}r = 0] \leq \frac{1}{2}$ as claimed.