CSE331: Introduction to Algorithms
Notes on Lecture 9: Matrix Multiplication

Antoine Vigneron

October 5, 2020

1 Analysis of Strassen's algorithm

The running time of Strassen's algorithm satisfies the recurrence relation

$$T(n) = 7T(n/2) + \Theta(n). \tag{1}$$

The master theorem yields $T(n) = \Theta(n^{\log 7})$. (As usual in this course, $\log n$ means $\log_2 n$.) We now prove it using the recursion tree and the substitution method.

1.1 Recursion Tree

The recursion tree is depicted in Figure 1. The size of the subproblems decrease by a factor 2 each time we go down one level, and at the leafs the size is 1. So if h denotes the height of the tree, we have $n/2^h = 1$ and thus $h = \log n$. The number of subproblems is multiplied by 7 with each level, and thus the number of subproblems at depth i is 7^i. So the number of leaves is $7^h = 7^{\log n} = 2^{\log(7)\log(n)} = n^{\log 7}$.

At depth $i < h$, the size of the subproblems is $n/2^i$, and there are 7^i subproblems, hence the total cost of the nodes at depth i is $cn\left(\frac{7}{4}\right)^i$. So the total cost of the recursion tree is

$$T(n) = T(1) \cdot n^{\log 7} + cn^2 \sum_{i=0}^{\log(n)-1} \left(\frac{7}{4}\right)^i$$

where the first term accounts for the leaves of the tree. Observe that

$$\sum_{i=0}^{\log(n)-1} \left(\frac{7}{4}\right)^i = \frac{\frac{7}{4}^{\log n} - 1}{\frac{7}{4} - 1} = \frac{4}{3}\left(\frac{7^{\log n}}{4^{\log n}} - 1\right) = \frac{4}{3}\left(\frac{n^{\log 7}}{n^2} - 1\right)$$

and thus

$$T(n) = T(1) \cdot n^{\log 7} + \frac{4}{3}cn^{\log 7} - \frac{4}{3}cn^2.$$

So the recursion tree method suggests that $T(n) = \Theta(n^{\log 7})$.

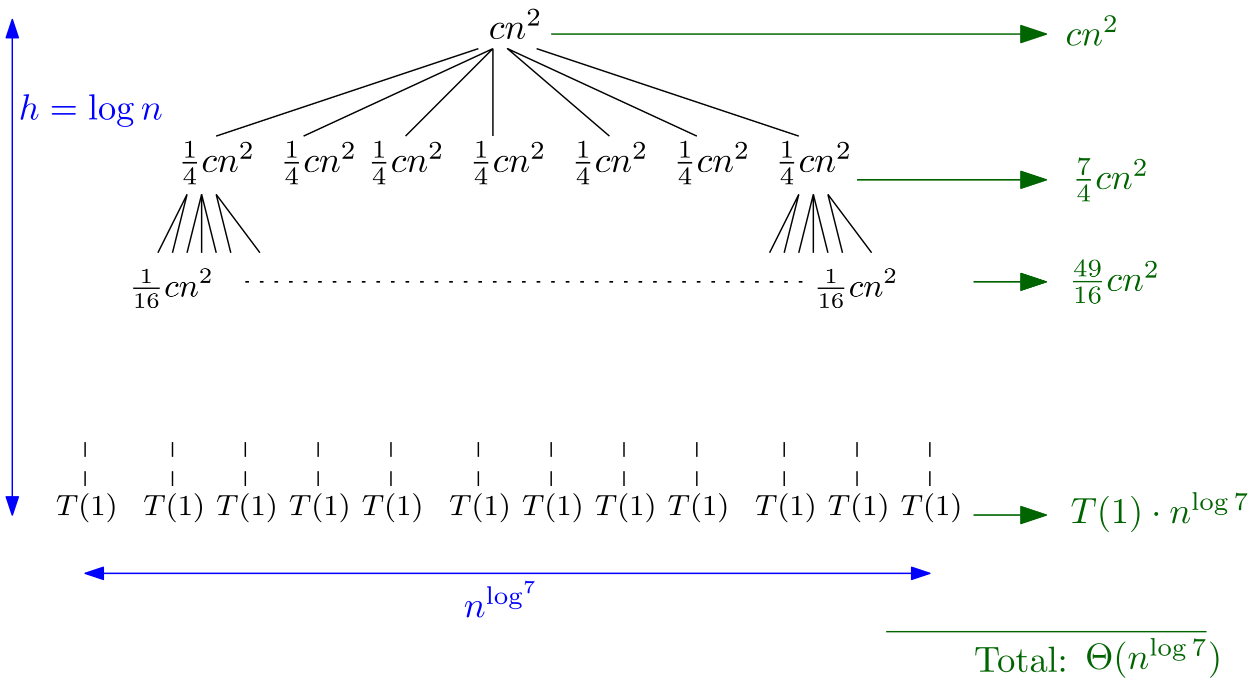

Figure 1: Recursion tree for $T(n) = 7T(n/2) + cn^2$

1.2 Substitution Method

We only prove that $T(n) = O(n^{\log 7})$, so Equation (1) can be rewritten

$$T(n) \leqslant 7T(n/2) + cn^2$$

for some constant $c > 0$. We want to prove by induction that

$$T(n) \leqslant dn^{\log 7} + en^2 \tag{2}$$

for some constants d and e. So we assume that

$$T(m) \leqslant dm^{\log 7} + em^2 \qquad \forall m < n.$$

and we want to prove that Inequality (2) follows. By setting $m = n/2$, it follows that

$$\begin{aligned} T(n) &\leqslant 7d\left(\frac{n}{2}\right)^{\log 7} + 7e\left(\frac{n}{2}\right)^2 + cn^2 \\ &= dn^{\log 7} + \left(\frac{7e}{4} + c\right)n^2 \end{aligned}$$

In order for inequality (2) to be satisfied, it suffices to have $\frac{7}{4}e + c \leqslant e$, and thus $e \leqslant -\frac{4}{3}c$. We also need the base case $T(1) \leqslant d + e$ to be satisfied. So we choose $e = -\frac{4}{3}c$ and $d = T(1) + \frac{4}{3}c$, and we have proved that

$$T(n) \leqslant \left(T(1) + \frac{4}{3}c\right)n^{\log 7} - \frac{4}{3}cn^2 \qquad \forall n \geqslant 1$$

which implies $T(n) = O(n^{\log 7})$.

2 Odd size matrices

It is not clear how to apply Strassen's algorithm when the size of the matrices is odd. For instance, suppose $n = 5$ and we split matrix B between rows 3 and 4, and between columns 3 and 4. Then A_{11} is a 3×3 matrix, B_{12} is a 3×2 matrix and B_{22} is a 2×2 matrix. So the expression $P_1 = A_{11} \times (B_{12} - B_{22})$ does not make sense.

One way to fix the problem is, as explained in class, to add rows and columns of zeroes to the matrix until the size is a power of 2, so we replace n with $n' = 2^h$ where h is the integer such that $2^{h-1} < n \leqslant 2^h$.

Example: We transform a 5×5 matrix into an 8×8 matrix.

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 6 & 7 & 8 \\ 5 & 6 & 7 & 8 & 9 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 0 & 0 & 0 \\ 2 & 3 & 4 & 5 & 6 & 0 & 0 & 0 \\ 3 & 4 & 5 & 6 & 7 & 0 & 0 & 0 \\ 4 & 5 & 6 & 7 & 8 & 0 & 0 & 0 \\ 5 & 6 & 7 & 8 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

As $n' < 2n$, the running time is still $\Theta((2n)^{\log_2 7}) = \Theta(n^{\log_2 7})$.

A better way of doing it is the following: If n is odd, then we add one row and one column of zeroes, and then recurse on matrices of size $(n+1)/2$. This is done at each level of recursion where the size is odd. In this way, we add fewer zeroes.