LECTURE #2

CS 170

Spring 2021

Last time: integer arithmetic

- addition: carry method is $O(n)$ and is optimal
- multiplication: - long multiplication is $O(n^2)$ and is <u>NOT</u> optimal
  - Karatsuba multiplication is $O(n^{\log_2 3})$ (& even faster methods are known)

The main idea was divide and conquer:

$$X \cdot Y = X_H Y_H \cdot 10^n + (X_H Y_L + X_L Y_H) \cdot 10^{n/2} + X_L Y_L$$

$$= X_H Y_H \cdot 10^n + \left((X_H + X_L)\cdot(Y_H + Y_L) - X_H Y_H - X_L Y_L\right) \cdot 10^{n/2} + X_L Y_L$$

This gives a recurrence $T(n) = 3 \cdot T\left(\frac{n}{2}\right) + O(n) = O(n^{\log_2 3})$.

Today:

- solving recurrences like the above, in general
- fast multiplication of matrices

Useful facts:

- $(a^b)^c = (a^c)^b = a^{bc}$
- $a^{\log_b n} = (b^{\log_b a})^{\log_b n}$
  $= (b^{\log_b n})^{\log_b a} = n^{\log_b a}$

Review an example: $T(n) = 3 \cdot T\left(\frac{n}{2}\right) + c \cdot n \quad T(1) = c \cdot 1$ (WLOG two constants are equal)

$$c \cdot n$$

$$c \cdot \frac{n}{2} \quad c \cdot \frac{n}{2} \quad c \cdot \frac{n}{2}$$

$$c \cdot \frac{n}{4} \quad \cdots$$

$$c \cdot 1 \quad c \cdot 1$$

problem size	work/problem	# of problems	total work
n	$c \cdot n$	1	$c \cdot n$
$n/2$	$c \cdot n/2$	3	$\frac{3}{2} c \cdot n$
$n/4$	$c \cdot n/4$	9	$\left(\frac{3}{2}\right)^2 c \cdot n$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n/2^i$	$c \cdot n/2^i$	3^i	$\left(\frac{3}{2}\right)^i c \cdot n$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	$c \cdot 1$	$3^{\log_2 n}$	$\left(\frac{3}{2}\right)^{\log_2 n} c \cdot n$

$$c \cdot n \cdot \left(1 + \left(\frac{3}{2}\right) + \cdots + \left(\frac{3}{2}\right)^{\log_2 n}\right)$$

$$= O\left(n \left(\frac{3}{2}\right)^{\log_2 n}\right) = O\left(n \, \frac{n^{\log_2 3}}{n^{\log_2 2}}\right) = O\left(n^{\log_2 3}\right)$$

We now do a similar reasoning for a general case.

In general: $T(n) = a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d \qquad T(1) = c$

Master Theorem for Recurrences

$\frac{a}{b^d} < 1 \rightarrow T(n) = O(n^d)$

$\frac{a}{b^d} = 1 \rightarrow T(n) = O(n^d \log n)$

$\frac{a}{b^d} > 1 \rightarrow T(n) = O(n^{\log_b a})$

proof: Let us analyze the tree of work.

$c \cdot n^d$

$c \cdot \left(\frac{n}{b}\right)^d \quad \dots \quad c \cdot \left(\frac{n}{b}\right)^d$

$c \cdot \left(\frac{n}{b^2}\right)^d \quad \dots$

$\vdots$

c

problem size	work/problem	# of problems	total work
n	$c \cdot n^d$	1	$c \cdot n^d$
n/b	$c \cdot (n/b)^d$	a	$(a/b^d) \cdot c \cdot n^d$
n/b^2	$c \cdot (n/b^2)^d$	a^2	$(a/b^d)^2 \cdot c \cdot n^d$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n/b^i	$c \cdot (n/b^i)^d$	a^i	$(a/b^d)^i \cdot c \cdot n^d$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	$c \cdot 1$	$a^{\log_b n}$	$(a/b^d)^{\log_b n} \cdot c \cdot n^d$

$$\left.\begin{array}{l} c \cdot n^d \\ (a/b^d) \cdot c \cdot n^d \\ \vdots \\ (a/b^d)^{\log_b n} \cdot c \cdot n^d \end{array}\right\} \; c \cdot n^d \cdot \left(1 + \frac{a}{b^d} + \dots + \left(\frac{a}{b^d}\right)^{\log_b n}\right)$$

We are left to bound $c \cdot n^d \cdot \left(1 + \frac{a}{b^d} + \cdots + \left(\frac{a}{b^d}\right)^{\log_b n}\right)$ in the three cases.

Recall: $1 + p + p^2 + \cdots + p^k = \frac{p^{k+1} - 1}{p-1}$

① **root-heavy:** if $\frac{a}{b^d} < 1$ (ie, $d > \log_b a$) then

$$T(n) = c \cdot n^d \cdot \frac{1 - \left(\frac{a}{b^d}\right)^{\log_b n + 1}}{1 - \frac{a}{b^d}} = O(n^d)$$

② **balanced:** if $\frac{a}{b^d} = 1$ (ie, $d = \log_b a$) then

$$T(n) = c \cdot n^d \cdot \underbrace{(1 + \cdots + 1)}_{\log_b n + 1} = O(n^d \log_b n) = O(n^d \log n)$$

③ **leaf-heavy:** if $\frac{a}{b^d} > 1$ (ie, $d < \log_b a$) then

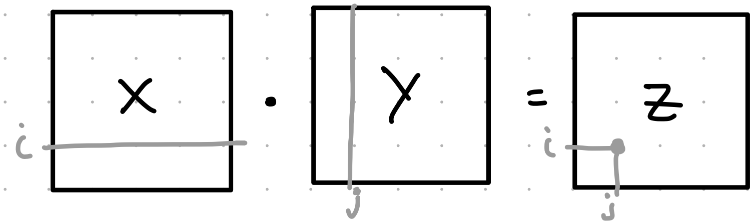
$$T(n) = c \cdot n^d \cdot \frac{\left(\frac{a}{b^d}\right)^{\log_b n + 1} - 1}{\frac{a}{b^d} - 1} = O\left(n^d \left(\frac{a}{b^d}\right)^{\log_b n}\right) = O\left(n^d \frac{n^{\log_b a}}{n^d}\right) = O(n^{\log_b a})$$

$\blacksquare$

Examples: $T(n) = 4T\left(\frac{n}{2}\right) + O(n) \Rightarrow a = 4, b = 2, d = 1 \Rightarrow \frac{a}{b^d} = \frac{4}{2^1} > 1 \Rightarrow O(n^{\log_b a}) = O(n^2)$

$T(n) = 3T\left(\frac{n}{2}\right) + O(n) \Rightarrow a = 3, b = 2, d = 1 \Rightarrow \frac{a}{b^d} = \frac{3}{2} > 1 \Rightarrow O(n^{\log_b a}) = O(n^{\log_2 3})$.

Matrix multiplication



- input: $n \times n$ matrices X and Y
- output: product matrix $Z := X \cdot Y$

$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj} = \langle X[i,*], Y[*,j] \rangle$$

In general NOT commutative (it can be that $XY \neq YX$).

- **Naive algorithm:**

$$\left.\begin{array}{lll} \text{For } i=1,\dots,n: & & n \text{ times} \\ \quad \text{For } j=1,\dots,n: & & n \text{ times} \\ \quad\quad \text{Compute } Z_{ij} & & O(n) \end{array}\right\} O(n^3)$$

- **Recursive algorithm:** leverage blockwise multiplication

$$X \cdot Y = \begin{array}{|c|c|} \hline A & B \\ \hline C & D \\ \hline \end{array} \cdot \begin{array}{|c|c|} \hline E & F \\ \hline G & H \\ \hline \end{array} = \begin{array}{|c|c|} \hline AE+BG & AF+BH \\ \hline CE+DG & CF+DH \\ \hline \end{array}$$

$$T(n) = 8 \cdot T\left(\frac{n}{2}\right) + O(n^2)$$

Via Master Theorem for recurrences:
$a=8, b=2, d=2 \Rightarrow \frac{a}{b^d} = \frac{8}{4} > 1 \Rightarrow n^{\log_b a}$

$$\left.\begin{array}{l} \text{– depth of tree is } \log_2 n \\ \text{– work at the leaves is } 8^{\log_2 n} = n^3 \end{array}\right\} O(n^3)$$

same as before ☹

Strassen Matrix Multiplication (1969)

Improves on the simple divide and conquer approach to beat $O(n^3)$.

$P_1 = A\cdot(F-H)$

$P_2 = (A+B)\cdot H$

$P_3 = (C+D)\cdot E$

$P_4 = D\cdot(G-E)$

$P_5 = (A+D)\cdot(E+H)$

$P_6 = (B-D)\cdot(G+H)$

$P_7 = (A-C)\cdot(E+F)$

$$X\cdot Y = \begin{array}{|c|c|} \hline P_5+P_4-P_2+P_6 & P_1+P_2 \\ \hline P_3+P_4 & P_1+P_5-P_3-P_7 \\ \hline \end{array}$$

Designing Strassen's Algorithm

Joshua A. Grochow* and Cristopher Moore†

September 1, 2017

Abstract

In 1969, Strassen shocked the world by showing that two $n \times n$ matrices could be multiplied in time asymptotically less than $O(n^3)$. While the recursive construction in his algorithm is very clear, the key gain was made by showing that 2×2 matrix multiplication could be performed with only 7 multiplications instead of 8. The latter construction was arrived at by a process of elimination and appears to come out of thin air. Here, we give the simplest and most transparent proof of Strassen's algorithm that we are aware of, using only a simple unitary 2-design and a few easy lines of calculation. Moreover, using basic facts from the representation theory of finite groups, we use 2-designs coming from group orbits to generalize our construction to all $n \geq 2$ (although the resulting algorithms aren't optimal for $n \geq 3$).

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

- depth of tree is $\log_2 n$
- work at leaves is $7^{\log_2 n} = n^{\log_2 7}$

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

Via Master Theorem for recurrences:

$a=7, b=2, d=2 \Rightarrow \frac{a}{b^d} = \frac{7}{4} > 1 \Rightarrow n^{\log_b a}$

Examples of divide and conquer:

- Karatsuba: faster integer multiplication
- Strassen: faster matrix multiplication

Beyond Strassen

$\omega < 3$ to be non-trivial

Suppose you figure out how to multiply $k \times k$ matrices in k^{ω} multiplications.
E.g. Strassen multiplies 2×2 matrices in $7 = 2^{\log_2 7}$ multiplications.

Then via divide and conquer you get an algorithm in time:

$$T(n) = k^{\omega} \cdot T\left(\frac{n}{k}\right) + O(n^2)$$

$$\Rightarrow T(n) = O\left(n^{\log_k k^{\omega}}\right) = O(n^{\omega}).$$

Hence you can "lift" any finite-size improvement into an asymptotic improvement.
Researchers have found improvements for very large matrices ($k = 10^{100}$) via yet other recursive techniques, leading to improvements in time for matrix multiplication.

The quest for the best matrix multiplication algorithm is open.

The fastest algorithm known at present runs in time $\approx O(n^{2.373})$.

A Refined Laser Method and Faster Matrix Multiplication

Josh Alman[*] Virginia Vassilevska Williams[†]

October 13, 2020

Abstract

The complexity of matrix multiplication is measured in terms of ω, the smallest real number such that two $n \times n$ matrices can be multiplied using $O(n^{\omega+\epsilon})$ field operations for all $\epsilon > 0$; the best bound until now is $\omega < 2.37287$ [Le Gall'14]. All bounds on ω since 1986 have been obtained using the so-called laser method, a way to lower-bound the 'value' of a tensor in designing matrix multiplication algorithms. The main result of this paper is a refinement of the laser method that improves the resulting value bound for most sufficiently large tensors. Thus, even before computing any specific values, it is clear that we achieve an improved bound on ω, and we indeed obtain the best bound on ω to date:

$$\omega < 2.37286.$$