Лекция 7 от 02.02.2016

Умножение чисел. Алгоритм Карацубы

Пусть $x = \overline{x_1 x_2 \dots x_n}$ и $y = \overline{y_1 y_2 \dots y_n}$. Распишем их умножение в столбик:

$$
\begin{array}{r}
\times \begin{array}{r} x_1\, x_2 \dots x_n \\ \underline{y_1\, y_2 \dots y_n} \end{array} \\
z_{11}\, z_{12}\, \dots z_{1n} \\
z_{21}\;\; z_{22} \dots z_{2n}\quad \\
+ \qquad \dots\dots\dots\dots\dots \qquad\quad \\
\underline{z_{n1}\, z_{n2} \dots z_{nn} \qquad\qquad} \\
z_{11}\, z_{12} \dots\dots\dots\dots\dots z_{2n}\, z_{2n+1}
\end{array}
$$

Какова сложность такого умножения? Всего n строк. На получение каждой строки тратится $O(n)$ операций. Тогда сложность этого алгоритма — $nO(n) = O(n^2)$. Теперь вопрос: *а можно ли быстрее?* Один из величайших математиков XX века, А.Н. Колмогоров, считал, что это невозможно.

Попробуем воспользоваться стратегией «Разделяй и властвуй». Разобьём числа в разрядной записи пополам. Тогда

$$
\times \begin{cases} x = 10^{n/2}a + b \\ y = 10^{n/2}c + d \end{cases}
$$
$$\Downarrow$$
$$xy = 10^n ac + 10^{n/2}(ad + bc) + bd$$

Как видно, получается 4 умножения чисел размера $\frac{n}{2}$. Так как сложение имеет сложность $\Omega(n)$, то

$$T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n)$$

Чему равно $T(n)$? Воспользуемся основной теоремой. Напомним: в общем виде неравенство имеет вид:

$$T(n) \leqslant aT\left(\frac{n}{b}\right) + cn^d$$

В нашем случае $a = 4, b = 2, d = 1$. Заметим, что $4 > 2^1 \implies a > b^d$. Тогда $T(n) = O(n^{\log_2 4}) = O(n^2)$.

Как видно, it's not very effective. Хотелось бы свести число умножений на каждом этапе к трём, так как это понизит сложность до $O(n^{\log_2 3}) \approx O(n^{1.58})$Но как?

Вернёмся к началу. Разложим $(a + b)(c + d)$

$$(a + b)(c + d) = ac + (ad + bc) + bd \implies ad + bc = (a + b)(c + d) - ac - bd$$

Подставим это в начальное выражение для xy:

$$xy = 10^n ac + 10^{n/2}((a + b)(c + d) - ac - bd) + bd$$

Отсюда видно, что достаточно посчитать три числа размера $\frac{n}{2}$: $(a+b)(c+d)$, ac и bd. Тогда:

$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n) \implies T(n) = O(n^{\log_2 3})$$

Полученный алгоритм называется *алгоритмом Карацубы.* На данный момент доказано, что для любого $\varepsilon > 0$ существует алгоритм, который совершает умножение двух чисел с сложностью $O(n^{1+\varepsilon})$. Также стоит упомянуть *алгоритм Шёнхаге-Штрассена*, работающий за $O(n \log n \log\log n)$

Перемножение матриц. Алгоритм Штрассена

Пусть у нас есть квадратные матрицы

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \text{ и } B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{pmatrix}$$

Сколько операций нужно для умножения матриц? Умножим их по определению. Матрицу $C = AB$ заполним следующим образом:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Всего в матрице n^2 элементов. На получение каждого элемента уходит $O(n)$ операций (умножение за константное время и сложение n элементов). Тогда умножение требует $n^2 O(n) = O(n^3)$ операций.

А можно ли быстрее? Попробуем применить стратегию «Разделяй и властвуй». Представим матрицы A и B в виде:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \text{ и } B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

где каждая матрица имеет размер $\frac{n}{2}$. Тогда матрица C будет иметь вид:

$$C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

Как видно, получаем 8 перемножений матриц порядка $\frac{n}{2}$. Тогда

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

По основной теореме получаем, что $T(n) = O\left(n^{\log_2 8}\right) = O(n^3)$.

Можно ли уменьшить число умножений до 7? *Алгоритм Штрассена* утверждает, что можно. Он предлагает ввести следующие матрицы (даже не спрашивайте, как до них дошли):

$$\begin{cases} M_1 = (A_{11} + A_{22})(B_{11} + B_{22}); \\ M_2 = (A_{21} + A_{22})B_{11}; \\ M_3 = A_{11}(B_{12} - B_{22}); \\ M_4 = A_{22}(B_{21} + B_{11}); \\ M_5 = (A_{11} + A_{12})B_{22}; \\ M_6 = (A_{21} - A_{11})(B_{11} + B_{12}); \\ M_7 = (A_{12} - A_{22})(B_{21} + B_{22}); \end{cases}$$

Тогда

$$\begin{cases} C_1 & = M_1 + M_4 - M_5 + M_7; \\ C_2 & = M_3 + M_5; \\ C_3 & = M_2 + M_4; \\ C_4 & = M_1 - M_2 + M_5 + M_6; \end{cases}$$

Можно проверить что всё верно (оставим это как ~~наказание~~ упражнение читателю). Сложность алгоритма:

$$T(n) = 7T\left(\frac{n}{2}\right) + O(n^2) \implies T(n) = O\left(n^{\log_2 7}\right)$$

На данный момент один из самых быстрых алгоритмов имеет сложность $\approx O(n^{2.3})$ (*алгоритм Виноградова*). Но этот алгоритм быстрее только в теории — из-за астрономически огромной константы.