Лекции по предмету
Алгоритмы 2

Группа лектория ФКН ПМИ 2016-2017
Данила Кутенин

2016/2017 учебный год

Содержание

Программа. Орг моменты

Внимание: программа дополняется после каждой лекции.

1. Матроиды.
2. Быстрое преобразование Фурье.
3. Алгоритм Карацубы, алгоритм Штрассена.

Формула такая же, как и в прошлом году:

$0.3 \cdot O_{\text{контесты}} + 0.25 \cdot O_{\text{семинарские листки}} + 0.15 \cdot O_{\text{кр}} + 0.3 \cdot O_{\text{экзамен}} + \text{Б}$.

Округление вверх.

Лекция 01 от 02.09.2016. Матроиды

Пока чуть отдаленно от матроидов.

У нас есть конечное множество A, которое в будущем мы будем называть *носителем*. Пусть $F \subset 2^A$, и F мы будем называть *допустимыми* множествами.

Также у нас есть весовая функция $c(w)$ $\forall w \in A$. Для каждого $B \in F$ мы определим *стоимость* множетсва, как $\sum\limits_{w \in B} c(w)$. Наша задача заключается в том, чтобы найти максимальный вес из всех допустимых множеств.

Пример 1 (Задача о рюкзаке). *У каждого предмета есть вес и стоимость. Мы хотим унести как можно больше вещей максимальной стоимости с весом не более k.*

Вес не более k нам задает ограничение, то есть множество F. А максимизация унесенной суммы нам и задаёт задачу.

Матроид

Множество F теперь будет всегда обозначаться как I.

Матроидом называется множество подмножеств множества A таких, что выполняются следующие 3 свойства:

1. $\varnothing \in I$

2. $B \in I \Rightarrow \ \forall D \subset B \Rightarrow D \in I$

3. Если $B, D \in I$ и $|B| < |D| \Rightarrow \exists w \in D \setminus B$ такой, что $B \cup w \in I$

Дальнейшее обозначение матроидов — $\langle A, I \rangle$.

Определение 1. *Базой матроида называют множество всех таких элементов $B \in I$, что* **не** *существует B', что $B \subset B', |B'| > |B|$ и $B' \in I$. Обозначение $\mathfrak{B}$.*

Свойство 1. *Все элементы из базы имеют одну и ту же мощность. И все элементы из I, имеющие эту мощность, будут в базе.*

Доказательство очевидно из определения.

Пример 2 (Универсальный матроид)**.** *Это все подмножества B множества A такие, что $|B| \leqslant k$ при $k \geqslant 0$. Все свойства проверяются непосредственно.*

База такого матроида — все множества размера k.

Пример 3 (Цветной матроид)**.** *У элементов множества A имеются цвета. Тогда $B \in I$, если все элементы множества B имеют разные цвета. Свойства проверяются непосредственно, в 3 свойстве надо воспользоваться принципом Дирихле.*

База такого матроида — множества, где присутствуют все цвета.

Пример 4 (Графовый матроид на n вершинах)**.** *$\langle E, I\rangle$. Множество ребер $T \in I$, если T не содержит циклов.*

Докажем 3 свойство:

Доказательство. Пусть у нас есть T_1 и T_2 такие, что $|T_1| < |T_2|$. Разобьём граф, построенный на T_1 на компоненты связности. Так как ребер ровно $|T_1|$ на n вершинах, то компонент связности будет $n - |T_1|$. В другом случае компонент связности будет $n - |T_2| < n - |T_1|$. То есть во 2-ом графе будет меньше компонент связности, а значит по принципу Дирихле найдётся ребро, которое соединяет 2 компоненты связности в 1-ом графе.

Этот алгоритм чем-то отдаленно напоминает алгоритм Краскала. □

Базой в таком матроиде являются все остовные деревья.

Пример 5 (Матричный матроид)**.** *Носителем здесь будут столбцы любой фиксированной матрицы. I — множество всех подмножеств из линейно независимых столбцов. Все свойства выводятся из линейной алгебры (3-е из метода Гаусса, если быть точным).*

Пример 6 (Трансверсальный матроид)**.** *$G = \langle X, Y, E\rangle$ — двудольный граф с долями X, Y. Матроид будет $\langle X, I\rangle$ такой, что $B \in I$, если существует паросочетание такое, что множество левых концов этого паросочетания совпадает с B.*

Докажем 3 свойство:

Доказательство. Пусть есть 2 паросочетания на $|B_1|$ и $|B_2|$ ($|B_1| < |B_2|$) вершин левой доли. Тогда рассмотрим симметрическую разность этих паросочетаний. Так как во 2-ом паросочетании ребер больше, то существует чередующаяся цепь, а значит при замене ребер на этой чередующейся цепи с новой добавленной вершиной (а она найдётся по принципу Дирихле) получим паросочетание с ещё 1 добавленной вершиной. □

Базой в таком матроиде будут вершины левой доли максимального паросочетания.

Приводимость одной базы к другой

Лемма 1. *Пусть $B, D \in \mathfrak{B}$. Тогда существует последовательность $B = B_0, B_1, \ldots, B_k = D$ такие, что $|B_i \triangle B_{i+1}| = 2$, где $\triangle$ обозначает симметрическую разность множеств.*

Доказательство. Будем действовать по шагам. Если текущее $B_i \neq D$, тогда возьмём произвольный элемент w из $B_i \setminus D$. Тогда по 2-ому пункту определения матроида следует, что $B_i \setminus w \in I$. Так как $|B_i \setminus w| < |D|$, то существует $u \in D$ такой, что $(B_i \setminus w) \cup u \in I$. И теперь $B_{i+1} \leftarrow (B_i \setminus w) \cup u$. Мы сократили количество несовпадающих элементов с D на 1, симметрическая разность B_i и B_{i+1} состоит из 2 элементов — w и u. □

Наконец, мы подошли к основной теореме лекции — жадный алгоритм или теорема Радо-Эдмондса.

Жадный алгоритм на матроиде

Доказательство будет в несколько этапов.

Для начала определимся с обозначениями. $M = \langle A, I \rangle, n = |A|, w_i$ — элементы множестваA. Решаем обычную задачу на максимизацию необходимого множества.

Теорема 1 (Жадный алгоритм. Теорема Радо-Эдмондса)**.** *Если отсортировать все элементы A по невозрастанию стоимостей весовой функции: $c_1 \geqslant c_2 \geqslant \ldots \geqslant c_n$, то такой алгоритм решает исходную задачу о нахождении самого дорогого подмножества:*

Algorithm 1 Жадный алгоритм на матроиде.

```
B ← ∅
for c_i do
    if B ∪ w_i ∈ I then
        B ← B ∪ w_i
```

Доказательство. Теперь поймём, что наш алгоритм в итоге получит какой-то элемент из базы. Пусть B_i — множество, которое мы получим после i шагов цикла нашего алгоритма. Действительно, если это не так, что существует множество из базы, которое его накрывает: формально $\exists D \in I : B_n \subset D$ и $|B_n| < |D|$, так как можно взять любой элемент из базы и добавлять в B_n по 1 элементу из пункта 3 определения матроида. Тогда у нас существует элемент w_i, который мы не взяли нашим алгоритмом, но $B_{i-1} \cup w_i \in I$, так как $B_{i-1} \cup w_i \subset B_n \cup w_i \subset D$, то есть это лежит в I по пункту 2 определения матроида. Значит мы должны были взять w_i, противоречие.

Рассмотрим последовательность d_i из 0 и 1 длины n такую, что $d_i = 1$ только в том случае, если мы взяли алгоритмом i-ый элемент. А оптимальное решение задачи пусть будет e_i — тоже последовательность из 0 и 1. Последовательности будут обозначаться d и e соответственно.

Если на каком-то префиксе последовательности d единиц стало меньше, чем в e, то возьмём все элементы, которые помечены последовательностью e единицами. Пусть это множество будет E. Аналогично на этом префиксе последовательности d определим множество D. $|D| < |E|, D \in I, E \in I$, поэтому мы можем дополнить D каким-то элементом из E, которого не было в D. То есть на этом префиксе у d стоит 0 (пусть это будет место i), но заметим, что на i-ом шаге мы обязаны были брать этот элемент, из-за рассуждений аналогичным рассуждению про базу (2 абзаца вверх).

Получаем, что на каждом префиксе d единиц не меньше, чем на этом же префиксе последовательности e. Значит 1-ая единица в d встретится не позже, чем в e, 2-ая единица в d не позже, чем 2-ая в e и т.д. по рассуждениям по индукции.

$\square$

На лекции была теория про ранги. В доказательстве можно обойтись без неё, просто приложу то, что сказал Глеб. Может быть понадобиться в задачах.

Рангом множества $B \subset A$ (обозн. $r(B)$) называют максимальное число k такое, что $\exists C \subset B$ такое, что $|C| = k, C \in I$.

Эта функция обладает таким свойством: для любого элемента $w \in A$ следует, что $r(B \cup w) \leqslant r(B) + r(w)$. Давайте поймём, почему так:

Если $r(B \cup w) = r(B)$, то всё хорошо, так как $r(w) \geqslant 0$. Если $r(B \cup w) = r(B) + 1$ (других вариантов не бывает из определения), то тогда $w \in I$, так как в $B \cup w$ найдётся такое $C \subset (B \cup w)$, что $|C| = r(B \cup w), w \in C$ (иначе C годилось бы для B и $r(B \cup w) = r(B)$), значит $r(w) = 1$, так как $C \in I$, а $\{w\} \subset C$.

Лекция 2 от 06.09.2016. Быстрое преобразование Фурье

Чтобы быть успешным программистом, надо знать 3 вещи:

- Сортировки;
- Хэширование;
- Преобразование Фурье.

Глеб

В этой лекции будет разобрано дискретное преобразование Фурье (Discrete Fourier Transform).

Применение преобразования Фурье

Допустим, что мы хотим решить такую задачу:

Пример 1. *Даны 2 бинарные строки A и B длины n и m соответственно. Мы хотим найти, какая подстрока в A наиболее похожа на B. Наивная реализация решает эту задачу в худшем случае за $O(n^2)$. Преобразование Фурье поможет решить эту задачу за $O(n \log n)$, а именно научимся решать другую задачу:*

Цель. *Хотим научиться перемножать многочлены одной степени $A(x) = a_0 + a_1x + \ldots + a_{n-1}x^{n-1}$ и $B(x) = b_0 + b_1x + \ldots + b_{n-1}x^{n-1}$ так, что $C(x) = A(x)B(x)$, то есть считать свёртку (найти все коэффициенты, если по-другому) $\sum\limits_{i=0}^{n-1} \sum\limits_{j=0}^{i} a_j b_{i-j} x^i$ за $O(n \log n)$.*

Вернёмся к нашему примеру. Поймём как с помощью нашей **цели** *решать задачу про бинарные строки.*

Пусть $A = a_0 \ldots a_{n-1}, B = b_0 \ldots b_{n-1}$. Их можно считать одной длины (просто добавим нулей в конец b при надобности). Теперь задача переформулировывается как нахождение максимального скалярного произведение B и некоторых циклических сдвигов A (до $n-m+1$).

Инвертируем массив B и припишем в конец n нулей, а к массиву A припишем самого себя. Посмотрим на все коэффициенты перемножения:

$$c_k = \sum_{i+j=k} a_i b_j$$

Но $b_i = 0$ при $i \geqslant n$, поэтому при $k \geqslant n$:

$$c_k = \sum_{i=0}^{n-1} b_i a_{k-i}$$

Выбрав нужные коэффициенты, мы решили эту задачу.

Алгоритм быстрого преобразования Фурье

Основная идея алгоритма заключается в том, чтобы представить каждый многочлен через набор n точек и значений многочлена в этих точках, быстро (за $O(n \log n)$) вычислить значения в каких-то n точках для обоих многочленов, потом перемножить за $O(n)$ сами значения. Потом применить обратное преобразование Фурье и получить коэффициенты $C(x) = A(x)B(x)$.

Итак, для начала будем считать, что $n = 2^k$ (просто добавим нулей до степени двойки).

Рассмотрим циклическую группу корней из 1 — $W_n = \{e^{i\frac{2\pi k}{n}} \ \forall\ k = 0, \dots, n-1\}$. Обозначим за $w_n = e^{i\frac{2\pi}{n}}$. Одно из самых главных свойств, что $w_n^p \cdot w_n^q = w_n^{p+q}$, которым мы будем пользоваться в дальнейшем.

Воспользуемся идеей метода «разделяй и властвуй».

Пусть $A(x) = a_0 + \dots a_{n-1}x^{n-1}$.

Представим $A(x) = A_l(x^2) + xA_r(x^2)$ так, что

$$A_l(x^2) = a_0 + a_2x^2 + \dots + a_{n-2}x^{n-2}$$
$$A_r(x^2) = a_1 + a_3x^2 + \dots + a_{n-1}x^{n-2}$$

Определение 1. Назовём *Фурье-образом* многочлена $P(x) = p_0 + \dots + p_{m-1}x^{m-1}$ вектор из m элементов — $\langle P(1), P(w_m), P(w_m^2), \dots, P(w_m^{m-1})\rangle$.

Теперь рекурсивно запускаемся от многочленов меньшей степени. Так как для любого целого неотрицательного k следует, что $2k$ четное число, то $w_n^{2k} = w_{n/2}^k \in W_{n/2}$, то есть мы можем уже использовать значения Фурье-образа для вычисления $A(x)$.

Если мы сможем за линейное время вычислить сумму $A_l(x^2) + xA_r(x^2)$, то суммарное время работы будет $O(n \log n)$, так как $A_l(x), A_r(x)$ имеют степень в 2 раза меньше, чем $A(x)$.

Действительно это легко сделать из псевдокода, который приведен ниже:

Algorithm 2 FFT

function FFT(A) $\triangleright$ A — массив из комплексных чисел, функция возвращает Фурье-образ
  $n \leftarrow \text{length}(A)$
  if $n == 1$ **then**
    return A
  $A_l \leftarrow \langle a_0, a_2, \dots, a_{n-2}\rangle$
  $A_r \leftarrow \langle a_1, a_3, \dots, a_{n-1}\rangle$
  $\hat{A}_l \leftarrow \text{FFT}(A_l)$
  $\hat{A}_r \leftarrow \text{FFT}(A_r)$
  for $k \leftarrow 0$ to $\frac{n}{2} - 1$ **do**
    $A[k] \leftarrow \hat{A}_l[k] + e^{i\frac{2\pi k}{n}}\hat{A}_r[k]$
    $A[k + \frac{n}{2}] \leftarrow \hat{A}_l[k] - e^{i\frac{2\pi k}{n}}\hat{A}_r[k]$ $\triangleright$ Здесь минус перед комплексным числом из-за того, что мы должны найти другой угол, удвоенный которого на окружности будет $\frac{2\pi(k+n/2)}{n}$
  return A

Теперь поговорим про обратное FFT. Этого материала не было на лекции на момент написания:

Фактически, мы вычислили такую вещь за $O(n \log n)$:

$$\begin{pmatrix} w_n^0 & w_n^0 & w_n^0 & w_n^0 & \cdots & w_n^0 \\ w_n^0 & w_n^1 & w_n^2 & w_n^3 & \cdots & w_n^{n-1} \\ w_n^0 & w_n^2 & w_n^4 & w_n^6 & \cdots & w_n^{2(n-1)} \\ w_n^0 & w_n^3 & w_n^6 & w_n^9 & \cdots & w_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n^0 & w_n^{n-1} & w_n^{2(n-1)} & w_n^{3(n-1)} & \cdots & w_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

где y_i — Фурье-образ многочлена $A(x)$.

Фактически нам надо найти обратное преобразование. Магическим образом обратная матрица к квадратной матрице выглядит почти также:

$$\frac{1}{n}\begin{pmatrix} w_n^0 & w_n^0 & w_n^0 & w_n^0 & \cdots & w_n^0 \\ w_n^0 & w_n^{-1} & w_n^{-2} & w_n^{-3} & \cdots & w_n^{-(n-1)} \\ w_n^0 & w_n^{-2} & w_n^{-4} & w_n^{-6} & \cdots & w_n^{-2(n-1)} \\ w_n^0 & w_n^{-3} & w_n^{-6} & w_n^{-9} & \cdots & w_n^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n^0 & w_n^{-(n-1)} & w_n^{-2(n-1)} & w_n^{-3(n-1)} & \cdots & w_n^{-(n-1)(n-1)} \end{pmatrix}$$

$$\frac{1}{n}\begin{pmatrix} w_n^0 & w_n^0 & w_n^0 & w_n^0 & \cdots & w_n^0 \\ w_n^0 & w_n^{-1} & w_n^{-2} & w_n^{-3} & \cdots & w_n^{-(n-1)} \\ w_n^0 & w_n^{-2} & w_n^{-4} & w_n^{-6} & \cdots & w_n^{-2(n-1)} \\ w_n^0 & w_n^{-3} & w_n^{-6} & w_n^{-9} & \cdots & w_n^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n^0 & w_n^{-(n-1)} & w_n^{-2(n-1)} & w_n^{-3(n-1)} & \cdots & w_n^{-(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

Откуда получаем: $a_k = \frac{1}{n}\sum_{j=0}^{n-1} y_j w_n^{-kj}$.

Теперь напишем псевдокод обратного алгоритма:

Algorithm 3 FFT_inverted

function FFT_INVERTED(A) ▷ A — Фурье-образ, возвращает коэффициенты многочлена
  $n \leftarrow \text{length}(A)$
  if $n == 1$ **then**
    return A
  $A_l \leftarrow \langle a_0, a_2, \ldots, a_{n-2} \rangle$
  $A_r \leftarrow \langle a_1, a_3, \ldots, a_{n-1} \rangle$
  $\hat{A}_l \leftarrow \text{FFT\_inverted}(A_l)$
  $\hat{A}_r \leftarrow \text{FFT\_inverted}(A_r)$
  for $k \leftarrow 0$ to $\frac{n}{2} - 1$ **do**
    $A[k] \leftarrow \hat{A}_l[k] + \mathrm{e}^{i\frac{-2\pi k}{n}} \hat{A}_r[k]$ ▷ Здесь угол идёт с минусом
    $A[k+\frac{n}{2}] \leftarrow \hat{A}_l[k] - \mathrm{e}^{i\frac{-2\pi k}{n}} \hat{A}_r[k]$
    $A[k] \leftarrow A[k]/2$ ▷ Поделим на 2 $\log n$ раз, а значит поделим на n в итоге
    $A[k+\frac{n}{2}] \leftarrow A[k+\frac{n}{2}]/2$ ▷ Аналогично строчке выше
  return A

Лекция 3 от 16.09.2016. Алгоритм Карацубы, алгоритм Штрассена

Перемножение 2 длинных чисел с помощью FFT

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

$$\begin{array}{r}
\times \begin{array}{r} x_1\, x_2 \ldots x_n \\ \underline{y_1\, y_2 \ldots y_n} \end{array} \\
z_{11}\, z_{12}\, \ldots z_{1n} \\
z_{21}\, z_{22} \ldots z_{2n} \qquad \\
+ \qquad \ldots\ldots\ldots\ldots\ldots \qquad\qquad \\
\underline{z_{n1}\, z_{n2} \ldots z_{nn} \qquad\qquad\quad} \\
z_1\, z_2 \ldots\ldots\ldots\ldots\ldots z_{2n}
\end{array}$$

Понятно, что наивное умножение 2 длинных чисел будет иметь сложность $O(n^2)$.

Давайте научимся перемножать 2 числа быстрым преобразованием Фурье за $O(n \log n)$.

Пусть $a = \overline{a_{n-1} \ldots a_0}, b = \overline{b_{n-1} \ldots b_0}$.

Тогда введём многочлены $f(x) = \sum\limits_{i=0}^{n-1} a_i x^i, g(x) = \sum\limits_{i=0}^{n-1} b_i x^i$.

За $O(n \log n)$ мы можем найти $h(x) = (f(x) \cdot g(x)) = \sum\limits_{i=0}^{2n-2} c_i x^i$.

После этого надо аккуратно провести переносы разрядов таким образом:

Algorithm 4 Умножение 2 длинных чисел.

function Умножение 2 длинных чисел($h(x)$) $\triangleright$ $h(x)$ — перемножение 2 многочленов $f(x)$ и $g(x)$.
  $carry \leftarrow 0$
  for $i = 0$ to $2n - 2$ **do**
    $h_i \leftarrow h_i + carry$
    $carry \leftarrow \left\lfloor \frac{h_i}{10} \right\rfloor$
    $h_i \leftarrow h_i \mod 10$

Но этот метод плохо применим на практике из-за того, что быстрое преобразование Фурье имеет очень большую константу.

Алгоритм Карацубы

Какое-то время человечество не знало алгоритмов перемножения быстрее, чем за $O(n^2)$. А.Н. Колмогоров считал, что это вообще невозможно. В один момент собрались математики на мехмате МГУ и решили доказать, что это невозможно. Но один из аспирантов (Анатолий Алексеевич Карацуба) Колмогорова пришёл и сказал, что у него получилось сделать это быстрее. Давайте посмотрим, как:

Будем считать, что $n = 2^k$ (если это не так, дополним нулями, сложность вырастет лишь в константу раз).

Для начала просто 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 $\Theta(n)$, то

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

Чему равно $T(n)$? Если посмотреть на дерево исходов или воспользоваться индукцией, то получим, что $T(n) = O(n^2)$, что, конечно, неэффективно.

Анатолий Алексеевич проявил недюжие способности и предложил следующее:

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:

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

Докажем, что $T(n) = O(n^{\log_2 3})$.

Рассмотрим дерево исходов: в каждой вершине дерева мы выполняем не более Cm действий, где C —какая-то фиксированная константа, а m — размер числа на данном шаге, поэтому

$$T(n) \leqslant Cn\left(1 + \frac{3}{2} + \ldots + \frac{3^{\log_2 n}}{2^{\log_2 n}}\right),\text{ так как на каждом шаге мы запускаемся 3 раза от задачи в 2 раза}$$

Откуда $T(n) \leqslant Cn \cdot \frac{\frac{3}{2}^{\log_2 n} - 1}{1/2} = 2Cn^{\log_2 3} = O(n^{\log_2 3}) \approx O(n^{1.5849})$

Полученный алгоритм называется алгоритмом Карацубы.

Перемножение матриц. Алгоритм Штрассена

После идеи А.А. Карацубы, появились многие алгоритмы, использующие ту же идею. Одним из этих алгоритмов является алгоритм Штрассена. Будем считать, что $n = 2^k$ снова (оставляем читалелю самим подумать, как дополнить матрицы $m \times t, t \times u$, чтобы потом лего восстановить ответ)

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.

Попробуем применить аналогичную 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 индукции 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:

$$T(n) = 7T\left(\frac{n}{2}\right) + O(n^2) \implies T(n) = O\left(n^{\log_2 7}\right) \approx O(n^{2.8073})$$

Доказательство времени работы такое же, как и в алгоритме Карацубы.

Также существует модификация алгоритма Штрассена, где используется лишь 15 сложений матриц на каждом шаге, вместо 18 предъявленных выше.

Эквивалентность асимптотик некоторых алгоритмов

Этот раздел не войдёт в экзамен.

Здесь мы поговорим об обращении и перемножении 2 матриц. Докажем, что асимптотики этих алгоритмов эквивалентны.

Теорема 1 (Умножение не сложнее обращения)**.** *Если можно обратить матрицу размеров $n \times n$ за время $T(n)$, где $T(n) = \Omega(n^2)$, и $T(3n) = O(T(n))$ (условие регулярности), то две матрицы размером $n \times n$ можно перемножить за время $O(T(n))$*

Доказательство. Пусть A и B матрицы одного порядка размера $n \times n$. Пусть

$$D = \begin{pmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{pmatrix}$$

Тогда легко понять, что

$$D^{-1} = \begin{pmatrix} I_n & -A & AB \\ 0 & I_n & -B \\ 0 & 0 & I_n \end{pmatrix}$$

Матрицу D мы можем построить за $\Theta(n^2)$, которое является $O(T(n))$, поэтому с условием регулярности получаем, что $M(n) = O(T(n))$, где $M(n)$ — асимптотика перемножения 2 матриц. $\square$

С обратной теоремой предлагаем ознакомиться в книге Кормена или Ахо, Хопкрофта и Ульмана.