Примеры алгоритмов. Асимптотические оценки. Семинар 1.

Подготовил: *Горбунов Э.*

Ключевые слова: O-обозначения, рекуррентные соотношения, основная теорема о рекуррентных оценках, алгоритмы "разделяй и властвуй", алгоритмы Карацубы и Штрассена, линейный алгоритм поиска k-й порядковой статистики

Организационные вопросы по курсу

1. Вся важная информация по курсу будет выкладываться здесь: eduardgorbunov.github.io/amc2019.html
2. Вам предстоит написать *три* контрольные. Предположительные даты контрольных: 24.03 (midterm), 12.05 (final) и, если кому-то повезёт, 19.05 (утешительная). Отмечу, что контрольные будут проходить по воскресеньям. Ориентировочная продолжительность контрольной: 3 часа.
3. Студент, получивший положительную оценку по первым двум курсовым контрольным ($\geqslant$ удовл.(3)), на утешительную контрольную не идёт и имеет право получить итоговую оценку по курсу, равную max(полусумма курсовых контрольных, оценка семинариста). В моей группе оценка семинариста вычисляется по следующей формуле:

$$\begin{aligned}\text{Оценка семинариста} &= 0.2 * \text{Оценка за первую контрольную} + 0.3 * \text{Оценка за вторую контрольную} \\ &+ 0.35 * \text{Оценка за мини-контрольные} + 0.15 * \text{ДЗ}.\end{aligned}$$

   Мини-контрольные будут проводиться в начале каждого занятия, начиная с четвёртого семинара, по теме предыдущего занятия.
4. Студент, получивший хотя бы на одной курсовой контрольной неудовлетворительную оценку (в том числе пропустивший контрольную по уважительной причине), обязан писать утешительную контрольную.
5. Если утешительная контрольная написана на положительную оценку, то вопрос о зачёте решается по результатам последующего устного опроса при показе работы.
6. Если утешительная контрольная не написана на положительную оценку, то для получения зачёта нужно следить за графиком пересдач. На них проводятся короткие письменные контрольные и устный опрос.

Сдача домашних заданий

После каждого семинара я буду присылать домашнее задание на следующую неделю. Обычно на его выполнение вам будет даваться ровно 7 дней. На всякий случай время дедлайна будет указано в домашнем задании. Задания можно присылать на мою почту ed-gorbunov@yandex.ru, указывая тему письма amvhw. Решения должны быть оформлены в виде файла с расширением pdf (желательно при помощи системы LaTeX), а также к письму должна быть приложена оригинальная редактируемая версия (doc или tex файл). Название файла должно быть следующим: hw + номер задания (2 цифры) + Ваша фамилия + номер вашей группы (например, hw01petrov678.pdf. Если вам требуется вставить в решение рисунок чего-либо, то можно вставлять картинки в решения (но запрещается вставлять фотографии решений целиком).

Вопросы можно присылать мне на почту или в сети ВКонтакте https://vk.com/eduard_gorbunov.

Литература

Основная

1. **[АХУ]** *Ахо А., Хопкрофт Д., Ульман Д.* Построение и Анализ Вычислительных Алгоритмов. М.: Мир, 1979.

2. **[ГД]** *Гери М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.

3. **[ДПВ]** *Дасгупта С., Пападимитриу Х., Вазирани У.* Алгоритмы. М.: МЦНМО, 2014.

4. **[Кормен 1]** *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: Построение и Анализ. М.: МЦНМО, 2002.

5. **[Кормен 2]** *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: Построение и Анализ. (2-е изд.) М.: Вильямс, 2005.

6. **[Кормен 3]** *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: Построение и Анализ. (3-е изд.) М.: Вильямс, 2013.

7. **[ХМУ]** *Хопкрофт Д., Мотвани Р., Ульман Д.* Введение в теорию автоматов, языков и вычислений. М.,: Вильямс, 2002.

8. **[AB]** *Arora S., Barak B.* Computational Complexity: A Modern Approach. theory.cs.princeton.edu/complexity/book.pdf

Дополнительная

1. *Верещагин Н., Шень А.* Вычислимые Функции. М.: МЦНМО, 1999. (Электронный вариант: www.mccme.ru/free-books)

2. **[Виноградов]** *Виноградов И.* Основы теории чисел. М.-Л.: Гостехиздат, 1952

3. *Вялый М., Журавлев Ю., Флеров Ю.* Дискретный анализ. Основы высшей алгебры. М.: МЗ Пресс, 2007.

4. **[К-Ш-В]** *Китаев А., Шень А., Вялый М.* Классические и квантовые вычисления. М.: МНЦМО-ЧеРо, 1999.

5. **[Кнут-1, Кнут-2, Кнут-3, Кнут-4] (цифра отвечает номеру тома** *Кнут Д.* Искусство программирования для ЭВМ. Существует несколько изданий на русском. Первое было выпущено издательством "Мир" в семидесятых. В настоящее время выпускается издательством "Вильямс". Есть многочисленные сетевые варианты.

6. **[К-Ф]** *Кузюрин Н., Фомин С.* Эффективные алгоритмы и сложность вычислений. М.: МФТИ, 2007.

7. **[П-С]** *Пападимитриу Х., Стайглитц К.* Комбинаторная оптимизация. Алгоритмы исложность. М.: Мир, 1985.

8. **[Хинчин]** *Хинчин А.* Цепные дроби. М.: Наука, 1979.

9. **[Ш]** *Шень А.* Программирование. Теоремы и задачи. М.: МЦНМО, 2007. (Электронный вариант: www.mccme.ru/free-books)

Нотация

Одной из важных характеристик алгоритма является время его работы. Тем не менее время работы алгоритма зависит от того, что или кто исполняет этот алгоритм. На практике часто оказывается, что время работы пропорционально числу проделанных элементарных операций (сложение чисел, их умножение и т.д.), то есть можно сказать, что время работы не превосходит числа таких итераций, помноженного на некоторую константу. Поэтому для оценки времени работы алгоритмов очень удобно использовать O-обозначения.

Пусть $f(n)$ и $g(n)$ — неотрицательные функции.

- $f(n) = O(g(n))$ означает, что порядок роста g при $n \to \infty$ *не меньше* порядка роста f, т. е. $\exists c > 0\ \exists n_0 : \forall n > n_0 \hookrightarrow f(n) \leq cg(n)$;
- $f(n) = \Omega(g(n)) \Leftrightarrow g(n) = O(f(n))$ (порядок роста f не меньше порядка роста g);
- $f(n) = o(g(n))$, если $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0$ (f имеет меньший порядок роста, чем g);
- $f(n) = \omega(g(n)) \Leftrightarrow g(n) = o(f(n))$ (g имеет меньший порядок роста, чем f);
- $f(n) = \Theta(g(n))$, если $f(n) = O(g(n))$ и $f(n) = \Omega(g(n))$ (порядки роста f и g одинаковы).

$\lfloor a \rfloor$ обозначает наибольшее целое, не превосходящее число a.

$\lceil a \rceil$ обозначает наименьшее целое, больше либо равное a.

Упражнения

1. Пусть функции $f(n)$ и $g(n)$ положительны при достаточно больших n. Верны ли следующие утверждения?
   (a) Если $f(n) = O(g(n))$, то $g(n) = O(f(n))$.
   (b) $f(n) + g(n) = \Theta(\min\{f(n), g(n)\})$.
   (c) $f(n) = O(g(n))$ влечёт $\log(f(n)) = O(\log(g(n)))$, если $\log g(n) > 0$ и $f(n) \geqslant 1$ для достаточно больших n.
   (d) $f(n) = O(g(n))$ влечёт $2^{f(n)} = O(2^{g(n)})$.
   (e) $f(n) = O(f^2(n))$.
   (f) $f(n) = \Theta(f(\frac{n}{2}))$.
   (g) $f(n) + o(f(n)) = \Theta(f(n))$.
2. Докажите основные свойства со страницы из Википедии про O-большое и o-малое.

Суммы и их свойства

Если алгоритм содержит цикл, то время его работы складывается из времён работы отдельных шагов. Поэтому для оценки времени работы алгоритма важно уметь работать с суммами и оценивать их.

1. **Почленное интегрирование и дифференцирование.** Напомним два важных факта из математического анализа.

   Теорема. Пусть $\{a_n(x)\}_{n=1}^{\infty}$ — последовательность функций, такая что $\forall n \hookrightarrow a_n(x)$ — интегрируемая по Риману на отрезке $[a, b]$ функция. Пусть ряд $\sum\limits_{n=1}^{\infty} a_n(x)$ сходится равномерно на $[a, b]$. Тогда сумма ряда $S(x) = \sum\limits_{n=1}^{\infty} a_n(x)$ является интегрируемой по Риману на отрезке $[a, b]$ функция и верна формула

$$\sum_{n=1}^{\infty} \int_a^b a_n(x)dx = \int_a^b \left(\sum_{n=1}^{\infty} a_n(x) \right) dx. \tag{1}$$

Теорема. Пусть $\{a_n(x)\}_{n=1}^{\infty}$ — последовательность функций, такая что $\forall n \hookrightarrow a_n(x)$ — непрерывно дифференцируемая на множестве E функция, где E — промежуток (отрезок или луч или интервал или полуинтервал или вся действительная ось). Пусть ряд $\sum\limits_{n=1}^{\infty} a'_n(x)$ сходится равномерно на E и пусть $\exists x_0 \in E : \sum\limits_{n=1}^{\infty} a_n(x)$ сходится. Тогда верна формула

$$\sum_{n=1}^{\infty} a'_n(x) = \left(\sum_{n=1}^{\infty} a_n(x)\right)'. \tag{2}$$

Например, из $\sum\limits_{k=0}^{\infty} x^k = \frac{1}{1-x}$ при $|x| < 1$ можно почленным интегрированием и домножением на x можно получить

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}.$$

2. **Суммы разностей.** Суммы вида
$$\sum_{k=1}^{n}(a_k - a_{k-1}) = a_n - a_0$$
называют суммами разностей, или телескопическими суммами. Иногда бывает так, что сумму можно представить в таком виде. Например,
$$\sum_{k=1}^{n} \frac{1}{k(k+1)} = \sum_{k=1}^{n}\left(\frac{1}{k} - \frac{1}{k+1}\right) = 1 - \frac{1}{n}.$$

3. **Индукция.** Если удалось угадать формулу для суммы, её можно проверить с помощью математической индукции.

4. **Почленные сравнения.** Основная идея — оценить каждый член сверху (или снизу) и получить оценку сверху (или снизу).

5. **Разбиение на части.** Основная идея — разбить сумму на части и оценить каждую часть по отдельности.

6. **Сравнение с интегралами.** Для монотонно возрастающей функции f мы можем заключить сумму между двумя интегралами:
$$\int_{m-1}^{n} f(x)dx \leqslant \sum_{k=m}^{n} f(k) \leqslant \int_{m}^{n+1} f(x)dx.$$
Аналогичное неравенство для монотонно убывающей функции:
$$\int_{m}^{n+1} f(x)dx \leqslant \sum_{k=m}^{n} f(k) \leqslant \int_{m-1}^{n} f(x)dx.$$
Этим методом можно получить хорошие оценки для частичных сумм гармонического ряда: нижнюю оценку
$$\sum_{k=1}^{n} \frac{1}{k} \geqslant \int_{1}^{n+1} \frac{dx}{x} = \ln(n+1)$$
и верхнюю оценку
$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \sum_{k=2}^{n} \frac{1}{k} \leqslant 1 + \int_{1}^{n} \frac{dx}{x} = 1 + \ln n.$$

Рекуррентные соотношения

Рекуррентное соотношение — это описание некоторой функции через неё же саму. Например, $f(0) = 0$, $f(n) = f(n-1)+1\ \forall n > 0$ — это рекуррента, задающая функцию $f(n) = n$. Решить рекуррентное соотношение означает найти функцию, которую задаёт это соотношение, в явном виде. Ниже собраны основные способы решения рекуррентных соотношений. Иногда точное решение не требуется, а достаточно получить оценку в терминах $O(\cdot), \Theta(\cdot), \Omega(\cdot), \omega(\cdot)$. Рекомендую также почитать главу 4 из книги Кормена про рекуррентные соотношения и файл **99recurrences.pdf** (он есть на piazza, а так же в группе в ВК) про то, как решать рекурренты.

1. **Метод подстановки.** Самый универсальный способ решить/оценить рекурренту — угадать вид/оценку функции и доказать это по индукции. В этом и заключается метод подстановки.

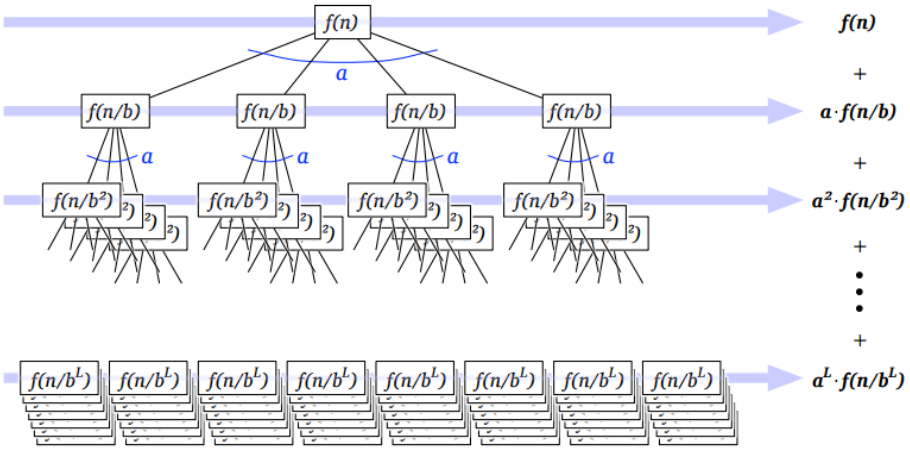
   Упражнения.

   (a) Решить методом подстановки рекурренту о Ханойской башне: $T(n) = 2T(n-1) + 1$, где $T(0) = 0$.

   (b) Оценить $T(n) = 2T(\lfloor \frac{n}{2} \rfloor) + n$ методом подстановки.

2. **Метод итераций.** Если не получается угадать решение, то можно просто подставлять рекурренту саму в себя (итерировать соотношение) и получить ряд, который можно оценить. К сожалению, этот метод часто приводит к сложным выкладкам, которые можно избежать, если использовать более хитрые методы.

3. **Деревья рекурсии.** Чтобы понять принцип работы с деревьями рекурсии достаточно внимательно посмотреть на рисунок ниже.



4. **Основная теорема о рекуррентных оценках (The Master Theorem).** Обсудим общий метод решения рекуррентных соотношений вида

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), \tag{3}$$

   где $a \geqslant 1$ и $b > 1$ — некоторые константы, а f — положительная функция (для достаточно больших n). Соотношение вида (3) возникает, если алгоритм разбивает задачу на a частей размера $\frac{n}{b}$ и решает каждую задачу рекурсивно. Затраты на разбиение на подзадачи и объединение результатов описывается функцией $f(n)$. Ниже приведены несколько примеров таких алгоритмов.

   (a) Быстрое возведение в степень (бинарное возведение в степень): $T(n) = T(\frac{n}{2}) + \Theta(1)$.

   (b) Сортировка слиянием (Merge-Sort): $T(n) = 2T(\frac{n}{2}) + \Theta(n)$.

   (c) Алгоритм Карацубы умножения чисел: $T(n) = 3T(\frac{3}{4}) + \Theta(n)$.

(d) Алгоритм Штрассена умножения матриц: $T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$.

Такие алгоритмы часто называют алгоритмами типа "разделяй и властвуй"

Теорема. Пусть $a \geqslant 1$ и $b > 1$ — константы, $f(n)$ — функция, $T(n)$ определено при неотрицательных n формулой

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

где под $\frac{n}{b}$ понимается либо $\lceil \frac{n}{b} \rceil$, либо $\lfloor \frac{n}{b} \rfloor$. Тогда:

(a) если $f(n) = O(n^{\log_b a - \varepsilon})$ для некоторого $\varepsilon > 0$, то $T(n) = \Theta(n^{\log_b a})$;

(b) если $f(n) = \Theta(n^{\log_b a})$, то $T(n) = \Theta(n^{\log_b a} \log_b n)$;

(c) если $f(n) = \Omega(n^{\log_b a + \varepsilon})$ для некоторого $\varepsilon > 0$ и если $af(\frac{n}{b}) < cf(n)$ для некоторой константы $c < 1$ и достаточно больших n, то $T(n) = \Theta(f(n))$.

После первого прочтения может возникнуть вопрос: о чём эта теорема? Почему сравниваются именно $f(n)$ и $n^{\log_b a}$? На самом деле, если не вдаваться в детали, то эта теорема о том, кто побеждает: функция $f(n)$ или $a^{\log_b n} = n^{\log_b a}$ (последнее — максимальный член арифметической прогресии, получаемой при использовании метода подстановки). Действительно, подставляя выражения для $T(n)$ в себя, неформально можно записать

$$aT\left(\frac{n}{b}\right) + f(n) \approx f(n) + a \cdot f\left(\frac{n}{b}\right) + a^2 \cdot f\left(\frac{n}{b^2}\right) + \ldots + a^{\log_b n} f(1) + \Theta(1). \tag{4}$$

В первом случае для достаточно больших n $f(n) < cn^{\log_b a - \varepsilon} = ca^{\log_b n}$. Поэтому (4)

$$aT\left(\frac{n}{b}\right) + f(n) \lessapprox cn^{\log_b a - \varepsilon} \cdot \sum_{k=0}^{\log_b n} \frac{a^k}{b^{k(\log_b a - \varepsilon)}} = cn^{\log_b a - \varepsilon} \cdot O\left(\frac{a^{\log_b n}}{b^{\log_b n (\log_b a - \varepsilon)}}\right) = O(n^{\log_b a}),$$

так как $a^{\log_b n} = n^{\log_b a}$. Оценка снизу следует из неотрицательности слагаемых (т.к. в сумме есть член пропорциональный $a^{\log_b n} = n^{\log_b a}$. Данное доказательство совершенно неформальное, но при желании его можно строго записать. Аналогично разбираются и остальные случаи. Полное доказательство можно прочитать в книге Кормена в главе про рекуррентные соотношения.

5. **Метод Акра-Баззи***. Данный раздел не обязателен. Рассмотрим рекурренту типа "разделяй и властвуй" общего вида

$$T(n) = \sum_{i=1}^{k} a_i T(b_i n + h_i(n)) + f(n) \quad \text{для } n \geqslant n_0, \tag{5}$$

причём выполнены следующие условия:

- k — константа
- $a_i > 0$ и $0 < b_i < 1$ — тоже некоторые константы для всех i
- $|f(x)| = O(x^c)$, где c — константа
- $|h_i(x)| = O\left(\frac{x}{(\log x)^2}\right)$ для всех i
- x_0 — константа.

В таком случае существует единственное число p, что $\sum\limits_{i=1}^{k} a_i b_i^p = 1$. Оказывается, что выполняется следующая оценка:

$$T(n) = \Theta\left(n^p \left(1 + \int_1^n \frac{f(x)}{x^{p+1}} dx\right)\right). \tag{6}$$

6. **Операторный метод***.Данный раздел не обязателен. Советую прочитать про метод операторов в документе **99recurrences.pdf**. Кратко привожу здесь основные моменты в использовании данного метода.

Operator	Definition
addition	$(f+g)(n) := f(n)+g(n)$
subtraction	$(f-g)(n) := f(n)-g(n)$
multiplication	$(\alpha \cdot f)(n) := \alpha \cdot (f(n))$
shift	$\boldsymbol{E}f(n) := f(n+1)$
k-fold shift	$\boldsymbol{E}^k f(n) := f(n+k)$
composition	$(\boldsymbol{X}+\boldsymbol{Y})f := \boldsymbol{X}f + \boldsymbol{Y}f$
	$(\boldsymbol{X}-\boldsymbol{Y})f := \boldsymbol{X}f - \boldsymbol{Y}f$
	$\boldsymbol{X}\boldsymbol{Y}f := \boldsymbol{X}(\boldsymbol{Y}f) = \boldsymbol{Y}(\boldsymbol{X}f)$
distribution	$\boldsymbol{X}(f+g) = \boldsymbol{X}f + \boldsymbol{X}g$

Operator	Functions annihilated	
$\boldsymbol{E}-1$	α	
$\boldsymbol{E}-a$	αa^n	
$(\boldsymbol{E}-a)(\boldsymbol{E}-b)$	$\alpha a^n + \beta b^n$	[if $a \neq b$]
$(\boldsymbol{E}-a_0)(\boldsymbol{E}-a_1)\cdots(\boldsymbol{E}-a_k)$	$\sum_{i=0}^{k} \alpha_i a_i^n$	[if a_i distinct]
$(\boldsymbol{E}-1)^2$	$\alpha n + \beta$	
$(\boldsymbol{E}-a)^2$	$(\alpha n+\beta)a^n$	
$(\boldsymbol{E}-a)^2(\boldsymbol{E}-b)$	$(\alpha n+\beta)a^b + \gamma b^n$	[if $a \neq b$]
$(\boldsymbol{E}-a)^d$	$\left(\sum_{i=0}^{d-1} \alpha_i n^i\right)a^n$	

If $\boldsymbol{X}$ annihilates f, then $\boldsymbol{X}$ also annihilates $\boldsymbol{E}f$.
If $\boldsymbol{X}$ annihilates both f and g, then $\boldsymbol{X}$ also annihilates $f \pm g$.
If $\boldsymbol{X}$ annihilates f, then $\boldsymbol{X}$ also annihilates αf, for any constant α.

If $\boldsymbol{X}$ annihilates f and $\boldsymbol{Y}$ annihilates g, then $\boldsymbol{X}\boldsymbol{Y}$ annihilates $f \pm g$.

1. Write the recurrence in operator form
2. Extract an annihilator for the recurrence
3. Factor the annihilator (if necessary)
4. Extract the *generic solution* from the annihilator
5. Solve for coefficients using base cases (if known)

7. **Общее решение линейной однородной рекурренты.**[1] Допустим нужно решить уравнение $x_n = b_{k-1}x_{n-1} + \ldots + b_0 x_{n-k}$. Для этого выпишем его характеристический многочлен:

$$\lambda^k - \sum_{i=0}^{k-1} b_i \lambda^i = \prod_i (\lambda - \lambda_i)_i^m, \tag{7}$$

где λ_i — корни характеристического многочлена, а m_i — их кратности. Тогда общее решение исходного уравнения имеет вид

$$x_n = \sum_i P_i(n)\lambda_i^n, \tag{8}$$

где $P_i(n)$ — произвольные многочлены степени не выше $m_i - 1$. Вообще говоря, если есть комплексные корни, то и многочлены нужно рассматривать с комплексными коэффициентами. Однако, если все коэффициенты исходного уравнения вещественны, то для каждого комплексного корня $\rho e^{i\varphi}$ число, комплексно сопряжённое ему, тоже является корнем многочлена (то есть $\rho e^{-i\varphi}$ тоже корень). От них можно линейным преобразованием перейти к $\rho\cos\varphi = \rho\left(\dfrac{e^{i\varphi}+e^{-i\varphi}}{2}\right)$ и $\rho\sin\varphi = \rho\left(\dfrac{e^{i\varphi}-e^{-i\varphi}}{2i}\right)$, так как множество решений линейной рекурренты образуют линейное пространство. Соответственно, если мы захотим запараметризовать все вещественные решения, в общем решении нужно будет вместо $P_1(n)\lambda_i^n = P_1(n)\rho^n e^{in\varphi}$ и $P_2(n)\bar{\lambda}_i^n = P_2(n)\rho^n e^{-in\varphi}$ рассматривать слагаемые $P_1(n)\rho^n\cos n\varphi$ и $P_2(n)\rho^n \sin n\varphi$, где многочлены рассматриваются с вещественными коэффициентами.

Алгоритм Карацубы умножения чисел

Подготовил: Кульков А.

Со школы все знают алгоритм, работающий за $O(n^2)$: умножение в столбик. Долгое время предполагалось, что ничего быстрее придумать нельзя. Первым эту гипотезу опроверг Карацуба, хотя считается, что преобразование Фурье в своих работах использовал ещё Гаусс, однако к возможности его использования для данной задачи пришли значительно позже.

Алгоритм Карацубы лаконичен и прост. Пусть мы перемножаем $A = a_0 + a_1 x$ и $B = b_0 + b_1 x$. Тогда:

$$A \cdot B = a_0 b_0 + (a_0 b_1 + a_1 b_0)x + a_1 b_1 x^2 =$$
$$= a_0 b_0 + [(a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1]x + a_1 b_1 x^2$$

Пусть для простоты числа нам даны в двоичной системе счисления и имеют длину n. Тогда если мы возьмём $x = 2^k, k \approx n/2$, то мы сведём задачу к трём вызовам той же задачи, но в два раза меньшего размера: для $a_0 b_0$, $a_1 b_1$ и $(a_0 + a_1)(b_0 + b_1)$. Для времени работы в таком случае будет иметь место оценка

[1] Теория однородных линейных рекуррент, в частности, доказательство того, почему решение нужно искать именно так, почти дословно повторяет соответствующее доказательство для линейных однородных дифференциальных уравнений. Для экономии времени мы не будем на этом останавливаться.

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n) = \Theta\left(n^{\log_2 3}\right) = \Theta(n^{1.584\ldots}).$$

Алгоритм Штрассена умножения матриц

Подготовил: Кульков А.

Пусть нам нужно найти матрицу $C = AB$, где A и B – матрицы размера $n \times n$. По определению можно считать эту матрицу по формуле $C_{ij} = \sum\limits_{k=1}^{n} A_{ik}B_{kj}$. Время работы такого алгоритма будет равна $\Theta(n^3)$. Алгоритм Штрассена был предложен в 1969 году и долгое время считался лучшим результатом по данной задаче.

Будем считать, что n – степень двойки, если это не верно, можно дополнить матрицы нулевыми столбцами и строками, чтобы это выполнялось. Теперь представим наши матрицы в следующем блочном виде:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Где A_{ij}, B_{ij}, C_{ij} – матрицы размера $\frac{n}{2}$. В таких обозначениях можно записать

$$C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j}$$

Это сводит задачу умножения больших матриц к умножению мелких. Однако суммарно требует 8 умножений матриц и 4 сложения, по два умножения и одному сложению на каждый из четырёх C_{ij}. При этом переходить мы будем от матриц порядка n к матрицам порядка $\frac{n}{2}$, то есть, общее время работы будет равно

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2) = \Theta(n^{\log_2 8}) = \Theta(n^3)$$

Чтобы решить эту проблему, было предложено ввести следующие вспомогательные матрицы:

$$\begin{gathered} P_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \\ P_2 = (A_{21} + A_{22})B_{11} \\ P_3 = A_{11}(B_{12} - B_{22}) \\ P_4 = A_{22}(B_{21} - B_{11}) \\ P_5 = (A_{11} + A_{12})B_{22} \\ P_6 = (A_{21} - A_{11})(B_{11} + B_{12}) \\ P_7 = (A_{12} - A_{22})(B_{21} + B_{22}) \end{gathered}$$

Можно проверить, что все C_{ij} можно выразить через P_k:

$$\begin{gathered} C_{11} = P_1 + P_4 - P_5 + P_7 \\ C_{12} = P_3 + P_5 \\ C_{21} = P_2 + P_4 \\ C_{22} = P_1 - P_2 + P_3 + P_6 \end{gathered}$$

Таким образом, нам потребуется всего 7 умножений матриц и 18 сложений. Время работы будет:

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2) = \Theta(n^{\log_2 7}) = \Theta(n^{2.81\ldots}).$$

Линейный алгоритм поиска k-й порядковой статистики

Подготовил: Кульков А.

Полезно будет прочитать соответствующий раздел в книге Кормена.

Один из известных линейных алгоритмов поиска k-ой статистики (BFPRT-Алгоритм) имеет следующий вид: разобьём массив на группы из пяти элементов. В каждой из групп выберем медиану за $O(1)$. Затем найдём медиану этих медиан (т.е. $n/2$-ю статистику), обозначим её s и разделим множество на два по ключам относительно неё. Половина пятёрок будут иметь медиану не больше s, а другая половина имеет медиану не меньше s, значит, хотя бы $\frac{1}{2} \cdot \frac{3}{5}n = \frac{3}{10}n$ элементов будут в каждом из множеств. Сравним теперь количество элементов, которое обозначим через a, в левой половине полученного массива с требумой статистикой k. Если элементов больше либо равно k, найдём k-ую статистику в левой половине. В противном случае либо s – ответ если $k - a = 1$, либо нужно вызваться в правую половину с числом $k - a - 1$. Так как размер обеих половин будет не больше, чем $\frac{7n}{10}$, получим в итоге следующее соотношение:

$$T(n) = \Theta(n) + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) = \Theta(n).$$

Другие полезные факты

1. **Гармонический ряд** имеет вид $1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} + \ldots$, а его n-я частичная сумма равна

$$H_n = \sum_{k=1}^{n} \frac{1}{k} = \ln n + \gamma + O\left(\frac{1}{n}\right), \tag{9}$$

   где $\gamma \approx 0,5772\ldots$ — константа Эйлера-Маскерони.

2. **Формула Стирлинга.**

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + O\left(\frac{1}{n}\right)\right) \tag{10}$$