Экзамен по АиСД 2024/25

@tiom4eg, @alekseyka2006 ...

March 30, 2025

Contents

1 Асимптотика

$g(n) \in O(f(n)) \implies \exists C > 0 \forall n : g(n) \leq C \cdot f(n)$
$g(n) \in o(f(n)) \implies \forall C > 0 \exists N : \forall n > N : g(n) < C \cdot f(n)$
$g(n) \in \Omega(f(n)) \implies \exists C > 0 \forall n : g(n) \geq C \cdot f(n)$
$g(n) \in \omega(f(n)) \implies \forall C > 0 \exists N : \forall n > N : g(n) > C \cdot f(n)$
$g(n) \in \Theta(f(n)) \implies g(n) \in \Omega(f(n)) \land g(n) \in O(f(n))$
Амортизированная - достигается в среднем по всем операциям, real-time - гарантированно достигается на каждой операции, ожидаемая - достигается по матожиданию

2 Теория вероятностей

Случайные величины $A_1, \ldots, A_n$ независимы, если для всех подмножеств $\{A_{i_k}\}$ выполняется $P(\bigcap\limits_{j=1}^{k} A_{i_j}) = \prod\limits_{j=1}^{k} P(A_{i_j})$

Матожидание: $\mathbb{E}[X] = \sum\limits_{\omega} X(\omega) \cdot p(\omega)$, $\mathbb{E}[aX + bY] = \sum\limits_{\omega}(a \cdot X(\omega) + b \cdot Y(\omega)) \cdot p(\omega) = \mathbb{E}[aX] + \mathbb{E}[bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]$, $\mathbb{E}[XY] = \sum\limits_{\omega} X(\omega)Y(\omega)p_X(\omega)p_Y(\omega)$ (для независимых X, Y) $= \sum\limits_{\omega} X(\omega)p_X(\omega) \cdot \sum\limits_{\omega} Y(\omega)p_Y(\omega) = \mathbb{E}[X]\mathbb{E}[Y]$

Дисперсия: $\mathbb{D}[X] = \mathbb{E}[X - \mathbb{E}[X]]^2 = \mathbb{E}[X^2 - 2X\mathbb{E}[X] + \mathbb{E}[X]^2] = \mathbb{E}[X^2] + \mathbb{E}[X]^2 - 2\mathbb{E}[X]\mathbb{E}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$, линейность примерно так же расписывается

Теорема (Неравенство Маркова):

Пусть случайная величина $X : \Omega \to \mathbb{R}_+$ определена на вероятностном пространстве $(\Omega, F, \mathbb{R})$, и ее математическое ожидание $\mathbb{E}|\xi|$ конечно. Тогда:

$$\forall\, x > 0 \;\; \mathbb{P}(|\xi| \geqslant x) \leqslant \frac{\mathbb{E}|\xi|}{x}$$

где:

x — константа соответствующая некоторому событию в терминах математического ожидания
ξ — случайная величина
$\mathbb{P}(|\xi| \geqslant x)$ — вероятность отклонения модуля случайной величины от x
$\mathbb{E}|\xi|$ — математическое ожидание случайной величины

Доказательство:

▷

Возьмем для доказательства следующее понятие:

Пусть A — некоторое событие. Назовем индикатором события A случайную величину I, равную единице если событие A произошло, и нулю в противном случае. По определению величина $I(A)$ имеет распределение Бернулли с параметром:

$$p = \mathbb{P}(I(A) = 1) = \mathbb{P}(A),$$

и ее математическое ожидание равно вероятности успеха $p = \mathbb{P}(A)$. Индикаторы прямого и противоположного событий связаны равенством $I(A) + I(\overline{A}) = 1$. Поэтому

$$|\xi| = |\xi| \cdot I(|\xi| < x) + |\xi| \cdot I(|\xi| \geqslant x) \geqslant |\xi| \cdot I(|\xi| \geqslant x) \geqslant x \cdot I(|\xi| \geqslant x).$$

Тогда:

$$\mathbb{E}|\xi| \geqslant \mathbb{E}(x \cdot I(|\xi| \geqslant x)) = x \cdot \mathbb{P}(|\xi| \geqslant x).$$

Разделим обе части на x:

$$\mathbb{P}(|\xi| \geqslant x) \leqslant \frac{\mathbb{E}|\xi|}{x}$$

◁

Теорема (Неравенство Чебышева):

Если $\mathbb{E}\xi^2 < 1$, то $\forall x > 0$ будет выполнено

$$\mathbb{P}(|\xi - \mathbb{E}\xi| \geqslant x) \leqslant \frac{\mathbb{D}\xi}{x^2}$$

где:

$\mathbb{E}\xi^2$ — математическое ожидание квадрата случайного события.
$E\xi$ — математическое ожидание случайного события
$P(|\xi - \mathbb{E}\xi| \geqslant x)$ — вероятность отклонения случайного события от его математического ожидания хотя бы на x
$\mathbb{D}\xi$ — дисперсия случайного события

Доказательство:

▷

Для $x > 0$ неравенство $|\xi - \mathbb{E}\xi| \geqslant x$ равносильно неравенству $(\xi - \mathbb{E}\xi)^2 \geqslant x^2$, поэтому

$$\mathbb{P}(|\xi - \mathbb{E}\xi| \geqslant x) = \mathbb{P}((\xi - \mathbb{E}\xi)^2 \geqslant x^2) \leqslant \frac{\mathbb{E}(\xi - \mathbb{E}\xi)^2}{x^2} = \frac{\mathbb{D}\xi}{x^2}$$

◁

3 Quicksort

Доказательство асимптотики при случайном выборе разделяющей точки (будем считать, что все элементы уникальны): пусть T_n - асимптотика для n. $T_n = (n-1) + \frac{1}{n}\sum\limits_{i=0}^{n-1} T_i + T_{n-1-i} = (n-1) + \frac{2}{n}\sum\limits_{i=0}^{n-1} T_i \implies nT_n =$

$n(n-1)+2\sum\limits_{i=0}^{n-1} T_i$

$nT_n-(n-1)T_{n-1}=n(n-1)+2\sum\limits_{i=0}^{n-1} T_i-((n-1)(n-2)+2\sum\limits_{i=0}^{n-2} T_i)=n(n-1)-(n-1)(n-2)+2T_{n-1} \implies nT_n=(n+1)T_{n-1}+2n-2 \implies \frac{T_n}{n+1}=\frac{T_{n-1}}{n}+\frac{2}{n+1}-\frac{2}{n(n+1)} \leq \frac{T_{n-1}}{n}+\frac{2}{n+1}=\frac{T_{n-2}}{n-1}+\frac{2}{n+1}+\frac{2}{n}-\frac{2}{n(n-1)} \leq \dots \leq \frac{T_1}{2}+\sum\limits_{i=1}^{n+1}\frac{2}{i}=O(n\log n)$

4 Median of Medians/Quickselect

По сути, мы хотим находить такую разделяющую точку, что она будет всегда работать достаточно хорошо. Для этого, разобьём все элементы на блоки по 5 элементов, в них отсортируем элементы (по сути, за $O(1)$), далее найдём медиану среди всех медианных элементов в блоках. Заметим, что будет выполнено следующее: в тех блоках, в которых медиана будет меньше медианы медиан, первые три элемента также гарантированно будут меньше, то есть про них мы можем сказать, что они точно окажутся слева от разделяющего элемента. Аналогично, в блоках, в которых медиана больше медианы медиан, последние три элемента гарантированно окажутся справа. Поскольку в обоих случаях блоков будет $\frac{3n}{10}$, получаем, что размер каждой части разбиения будет находиться между $\frac{3n}{10}$ и $\frac{7n}{10}$. Предположим, что каждый раз мы будем попадать в худший из случаев и идти в блок размера $\frac{7n}{10}$. Тогда, $T(n)=T(\frac{n}{5})+T(\frac{7n}{10})+O(n)=O(n)$ (по мастер-теореме)

5 Кучи

Инварианты d-арной кучи: значение в каждой вершине не больше, чем в любом из её детей, все слои, кроме последнего, заполнены полностью (содержат k^i вершин), в последнем слое заполнен какой-то префикс мест.

Для балансировки кучи будем использовать операции `sift_up` и `sift_down`. `sift_up` будет просто поднимать вершину, пока она больше своего родителя, `sift_down` же, наоборот, будет опускать вершину из корня (свапая её с максимальным ребёнком), пока не восстановится инвариант кучи.

Пусть все дети у вершины, которую сейчас просеиваем вниз, образуют в своих поддеревьях корректные d-арные кучи. Тогда, либо наша уже корректная (если текущая вершина не больше своих детей), либо мы свапаем её с минимальным ребёнком, тогда инвариант "вершина не больше всех своих детей" сохранится у всех вершин, кроме, возможно, просеиваемой. Примерно так же можно доказать, что `sift_up` работает корректно при добавлении нового элемента.

Добавление элемента: добавляем новую вершину на последний слой в первое свободное место, просеиваем её вверх. Удаление минимума - в корень ставим значение какой-либо другой вершины (обычно, последней) и просеиваем вниз. Построение за $O(n)$: закинем все вершины в кучу неважно как, после чего вызовем для всех вершин `sift_down` в порядке снизу вверх. Корректность обоснована тем, что при вызове процедуры от некоторой вершины поддеревья её детей уже будут корректными d-арными кучами. Поскольку `sift_down`, как и `sift_up`, работает за высоту кучи, получим сложность $\leq \sum\limits_{i=0}^{\log_d n}(i+1)\frac{n}{d^i}=O(n)$.

6 Skip list

TODO

7 Амортизированные очереди, деки (с поддержкой минимума)

Исходно у нас есть структура стек, которую понятно, как реализовывать. Мы хотим реализовать очередь/дек с эффективным использованием памяти. Реализуем очередь на двух стеках: st_{head}, st_{tail}. В качестве потенциала возьмём $|st_{tail}|$. При добавлении элемента будем добавлять его наверх st_{tail}, выполнится одна операция, потенциал увеличится на 1, поэтому $a_i=1+1=2$. При удалении элемента посмотрим на то, пусть ли st_{head}. Если нет - удалим элемент с его верхушки, потенциал не изменится $\implies a_i=1+0=1$. Если же он пуст, то перенесём в него все элементы из st_{tail} по одному, понятно, что верхний элемент в st_{tail} будет нижним в st_{head} и наоборот, это будет корректным порядком для st_{head}. Будет выполнено $|st_{tail}|$ операций сложности $O(1)$ для переноса и потенциал уменьшится на $|st_{tail}|$, после чего будет извлечён элемент с верхушки st_{head} как и в прошлом случае, значит $a_i=k-k+1=1$. Доказали, что $a_i=O(1)$, $\Phi_i=O(n)$, поскольку в любой момент размер никакого из стеков не мог превысить общее число операций (n), значит, средняя стоимость операции равна $O(1)$.

Дек можно реализовать с помощью трёх стеков. Первые два будут как и раньше st_{head}, st_{tail}, потенциалом будет $3\cdot\max(|st_{head}|, |st_{buf}|)$. Третьим будет st_{buf}, который мы будем использовать как вспомогательный буфер при операции ребаланса. При `push_front` или `push_back` будем добавлять соответствующий элемент наверх нужного стека, при `pop_front` или `pop_back`, если нужный стек - непустой, будем забирать из него верхний элемент. Остается последний случай - когда мы пытаемся забрать верхний элемент из стека, который является пустым. Предположим, что пустым является st_{head}, а st_{tail} непуст. Перекинем половину элементов st_{tail} в st_{buf}, оставшиеся перекинем в st_{head} (они при этом поменяют свой порядок на обратный, то есть нужный), затем из st_{buf} вернём элементы в st_{tail}. После таких действий элементы сохранят нужный порядок, при этом потенциал поделится примерно пополам. Значит, $a_i = 1.5\max(|st_{head}|,\ |st_{buf}|) - 0.5\Phi_i + O(1) = O(1)$, остальные операции имеют стоимость $O(1)$ аналогично очереди.

Чтобы всё это могло поддерживать минимум, будем реализовывать соответствующие структуры на стеках с поддержкой минимума. Стек с поддержкой минимума будет просто поддерживать стек рекордов вместе с элементами. Понятное дело, что добавление поддержки стека рекордов - гарантированное $O(1)$, так что амортизация не поломается.

8 Фибоначчиева куча

TODO

9 Хеши и хеш-таблицы

Пусть есть пространство ключей U и семейство хэш-функций $H : U \to [m]$. H называется универсальным семейством хэш-функций для U, если $\forall x, y \in U : |\{h \in H : h(x) = h(y)\}| \le \frac{|H|}{m}$. Семейство хэш-функций называется k-независимым, если $\forall (x_1, \ldots, x_k) \in U^k\ (x_i \ne x_j),\ \forall (y_1, \ldots, y_k) \in [m]^k : P[h(x_1) = y_1 \wedge \cdots \wedge h(x_k) = y_k] = m^{-k}$

При открытой адресации всё хранится в одном массиве, разрешение коллизий происходит следующим образом - выбирается правило, по которому изменяется значение хеша, пока бакет с соответствующим хешем занят. Например, можно прибавлять 1 со взятием по модулю размера хеш-таблицы. В закрытой адресации поступаем по другому - в каждом бакете храним какую-то структуру (например, односвязный список), в которой будут храниться ключи с одинаковым хешем и по которой будем осуществлять поиск/вставку при запросе к данному ключу.

Фильтр Блума - храним битсет длины m, выбираем k хеш функций (которые должны быть независимыми в совокупности), для элемента x ставим единички в биты $h_1(x),\ h_2(x), \ldots,\ h_k(x)$. Вероятность того, что какой-то бит останется нулевым после добавления n элементов, равна $(1 - \frac{1}{m})^{kn} \approx e^{-\frac{kn}{m}}$. Ложноположительное срабатывание - все биты, принадлежащие хешу какого-то ключа, оказываются единичными, вероятность равна $(1 - e^{-\frac{kn}{m}})^k$

Двухуровневая схема идеального хеширования: выбирается случайная функция $U \to [N]$, заменяется до тех пор, пока $\sum |B_i|^2 > 2N$ (где B_i - бакет i), после чего для бакета i выделяем $|B_i|^2$ ячеек и находим функцию h_i, которая является идеальной. Пруфы ниже (на английском):

Theorem 10.6 *If we pick the initial h from a universal set H, then*

$$\Pr[\sum_i (n_i)^2 > 4N] < 1/2.$$

Proof: We will prove this by showing that $\mathbf{E}[\sum_i (n_i)^2] < 2N$. This implies what we want by Markov's inequality. (If there was even a 1/2 chance that the sum could be larger than $4N$ then that fact by itself would imply that the expectation had to be larger than $2N$. So, if the expectation is less than $2N$, the failure probability must be less than 1/2.)

Now, the neat trick is that one way to count this quantity is to count the number of ordered pairs that collide, including an element colliding with itself. E.g, if a bucket has `{d,e,f}`, then `d` collides with each of `{d,e,f}`, `e` collides with each of `{d,e,f}`, and `f` collides with each of `{d,e,f}`, so we get 9. So, we have:

$$\begin{aligned}
\mathbf{E}[\sum_i (n_i)^2] &= \mathbf{E}[\sum_x \sum_y C_{xy}] \quad (C_{xy} = 1 \text{ if } x \text{ and } y \text{ collide, else } C_{xy} = 0) \\
&= N + \sum_x \sum_{y \ne x} \mathbf{E}[C_{xy}] \\
&\le N + N(N-1)/M \quad (\text{where the } 1/M \text{ comes from the definition of universal}) \\
&< 2N. \quad (\text{since } M = N) \quad \blacksquare
\end{aligned}$$

10.5.1 Method 1: an $O(N^2)$-space solution

Say we are willing to have a table whose size is quadratic in the size N of our dictionary S. Then, here is an easy method for constructing a perfect hash function. Let H be universal and $M = N^2$. Then just pick a random h from H and try it out! The claim is there is at least a 50% chance it will have no collisions.

Claim 10.5 *If H is universal and $M = N^2$, then $\Pr_{h \sim H}$(no collisions in S) $\geq 1/2$.*

Proof:

- How many pairs (x, y) in S are there? **Answer:** $\binom{N}{2}$
- For each pair, the chance they collide is $\leq 1/M$ by definition of "universal".
- So, Pr(exists a collision) $\leq \binom{N}{2}/M < 1/2$. ■

This is like the other side to the "birthday paradox". If the number of days is a lot *more* than the number of people squared, then there is a reasonable chance *no* pair has the same birthday.

So, we just try a random h from H, and if we got any collisions, we just pick a new h. On average, we will only need to do this twice. Now, what if we want to use just $O(N)$ space?

10 ДД/Splay

TODO (TO DIE)

11 Внешняя память

В реальном мире основным боттлнеком являются дисковые операции, поэтому в этом разделе мы постараемся оптимизировать уже известные алгоритмы под работу с внешней памятью. Стандартные обозначения: B - размер блока памяти, который считывается за одну операцию, M - размер имеющейся оперативной памяти (в которой операции будут выполняться за символическое $O(0)$), N - число операций или размер данных.

Самая базовая задача - пройтись по какому-то массиву в памяти и что-то в нём найти/посчитать какую-то статистику. Если массив лежит во внешней памяти подряд, то будем считывать в оперативную память по B элементов за раз, обновлять статистику, считывать следующий блок и так далее. Если мы хотим произвести какую-то трансформацию (например, превратить массив чисел в массив префсумм), то можно модифицировать загруженный блок и вернуть его на старое место также одной операцией. Данную операцию далее будем называть $Scan(N)$ и она работает за $O(\frac{N}{B})$ операций.

Реализуем стек во внешней памяти. Изначально будем в RAM хранить пустой блок размера B. Как поддерживать операции стека на нём - очевидно. Дождёмся первого момента, когда этот блок полностью заполнится и положим его в память, при этом сохраним его в RAM. Далее, те элементы, которые не влезают в первый блок, будем добавлять во второй, пока и он не переполнится, после чего мы положим его в память вслед за прошлым положенным, поменяем с первым блоком и очистим второй. То есть, мы хотим хранить в RAM последний полный блок стека + текущий неполный. Если мы в какой-то момент опустошим первый блок, то полезем в память, чтобы забрать оттуда новый блок. Почему это работает за $O(\frac{1}{B})$ на операцию в худшем случае? Очевидно, что нужно хотя бы B операций с момента прошлой записи блока в память, чтобы мы заполнили ещё один блок и записали в память его. Проблему могут доставлять операции `pop`. Но из-за того, что мы храним в RAM, гарантируется, что после последнего доступа в внешнюю память с целью забрать блок до нового доступа пройдет хотя бы B операций (поскольку после очередного чтения из памяти, в RAM хранится хотя бы B элементов, чтобы запросить новый доступ, их все нужно удалить).

Имея реализацию стека, можно написать очередь на двух стеках (из переднего стека только забираем элементы, в задний только добавляем) и дек на двух стеках (здесь они уже используются на полную мощность). Примерно такими же манипуляциями, как и для стека, можно доказать, что все работает за $O(\frac{1}{B})$ на операцию.

Наконец, опишем идею для реализации сортировки во внешней памяти за $O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{M})$ (эту же идею можно переделать, чтобы получить кучу с такой же асимптотикой). По сути, будем строить дерево merge сортировки, только вместо 2 детей будем объединять $\frac{M}{B}$ за раз (больше не влезет в RAM). В листьях дерева будем хранить отсортированные блоки размера M - подгрузим их в RAM, отсортируем и загрузим обратно. Теперь, пусть мы хотим смержить $\frac{M}{B}$ кусков. Будем подгружать их в память блоками размера B. Далее, в RAM можем мержить их примерно как угодно, например можно завести приоритетную очередь и доставать оттуда по одному минимальному элементу. Когда в каком-то из блоков закончятся элементы, возьмём следующий (или ничего не возьмём, если элементов не осталось). Заполненные блоки, полученные в результате мержа, закинем друг за другом во внешнюю память, пометив, что в этом месте у нас лежит массив очередной вершины дерева сортировки. Поскольку каждый блок размера B исходного массива будет обработан в $\log_{\frac{M}{B}} \frac{N}{M}$ вершинах дерева (поскольку именно такая высота будет у дерева сортировки), получим желаемую асимптотику. Процедура сортировки с такой асимптотикой является оптимальной и обозначается как $Sort(N)$.

12 2-3 дерево и B-деревья

TODO

13 Персистентность

Персистентные структуры данных — это структуры данных, которые при внесении в них изменений сохраняют доступ ко всем своим предыдущим состояниям.

Есть несколько «уровней» персистентности: частичная — к каждой версии можно делать запросы, но изменять можно только последнюю, полная — можно делать запросы к любой версии и менять любую версию, конфлюэнтная — помимо этого можно объединять две структуры данных в одну (например, сливать вместе кучи или деревья поиска). Также стоит отметить, что амортизация и персистентность вместе не работают, поскольку мы можем найти операцию, которая выполняется долго и попросить повторить её много раз, что сломает асимптотику.

Для того, чтобы реализовать персистентный стек, достаточно вспомнить, что стек это, по сути, односвязный список, к тому же во время запросов мы "дёргаем" только последний его элемент. Тогда, в качестве версии стека будем хранить значение его верхнего элемента и ссылка на версию стека без последнего элемента (такая точно будет существовать, поскольку последний элемент когда-то был добавлен, и в этот момент стек, очевидно, состоял из всех элементов, кроме добавленного). При добавлении элемента создадим новую версию со ссылкой на текущую, при удалении перейдём в ту версию, на которую ведёт ссылка (изначально будет существовать пустой стек с версией 0). Всё это, очевидно, работает за гарантированное $O(1)$.

Персистентное дерево отрезков основывается на идее того, что за один запрос дерева отрезков мы проходимся по не более чем $O(\log n)$ вершинам (на самом деле, это работает для вообще всех структур без амортизации, например для ДД). Тогда при изменении информации в вершине будем создавать копию текущей вершины, изменять данные уже в ней (чтобы не испортить старые версии) и возвращать указатель на новую версию вершины, чтобы обновить указатели на детей в той вершине, из которой пришли в текущую. Это обеспечивает полную персистентность за $O(\log n)$.

Теперь покажем, как можно реализовать персистентную очередь. Автору данной секции очень не понравилось легендарное решение с 6 (или 5) стеками, поэтому он воспользуется другой идеей, которая использует тот факт, что у нас есть именно персистентные стеки. Итак, пусть у нас есть персистентный стек st_{all}, в котором хранятся все когда-либо добавленные элементы. Заведём два стека st_{head} и st_{future}. В st_{head} будут лежать элементы с начала очереди, в st_{future} будет набираться будущая версия st_{head} (а ещё, эти стеки тоже должны быть персистентными, чтобы копирование работало за $O(1)$). Итак, что мы будем делать - каждый раз, когда st_{future} пустой, мы будем с конца добавлять в него по одному актуальные элементы st_{all} (они, очевидно, образуют некоторый суффикс). Когда в st_{future} наберутся все нужные элементы, переложим его в st_{head} и начнём набирать новый st_{future}. При операции удаления элемента из начала очереди, будем брать верхний элемент из st_{head}. Теперь докажем, что если после выполнения каждой операции над очередью добавлять один элемент в st_{future}, то st_{head} никогда не опустеет раньше времени и всё будет работать корректно. Итак, после добавления первого элемента, мы сразу же добавим его в st_{future}, который переложим в st_{head}. Запомним это состояние как "обнуление" и "обнулением" будем называть всякое такое состояние, в котором st_{head} только что было присвоено новое значение, а st_{future} был очищен. Докажем, что между обнулениями st_{head} не обнулится, а в st_{future} будут добавлены только нужные элементы. Второе сразу следует из того, что мы проверяем добавляемый в st_{future} элемент на актуальность. С первым всё тоже несложно - пусть после последнего обнуления в st_{head} осталось n элементов. Понятно, что st_{future}, который превратился в st_{head}, набирался на протяжении n операций, среди которых было не более n добавлений новых элементов. Таким образом, даже если все операции после этого обнуления будут удалением элемента из начала очереди, мы успеем заполнить st_{future} и положить новую версию в st_{head} до того, как нам поступит запрос на удаление из непустого стека (заметим, что запросы добавления элемента делают нам только лучше, поскольку число элементов, которые надо положить в st_{future} фиксируется на момент обнуления). Корректность доказана, асимптотика будет $O(1)$, поскольку мы выполняем всего одно добавление элемента в st_{future} за операцию плюс выполняем ещё $O(1)$ других операций, которые занимают $O(1)$ времени (поскольку копирование персистентных стеков и откат в них занимает $O(1)$ времени).

14 LCA/RMQ

TODO

15 Link-cut дерево

TODO (TO DIE)

16 Переборы (рюкзак, клики, MITM, SOS)

TODO

17 Альфа-бета отсечение

TODO

18 Численные методы (Ньютон, тернарный поиск, БПФ и применения)

TODO (TO DIE)

19 Мастер-теорема и Карацуба

TODO

20 Геометрия

TODO (TO DIE PAINFULLY)

21 Монте-Карло

TODO

22 Метод Ньютона

Ищем корень функции $f(x)$. Пусть x_0 – начальное приближение. Вычислим x_{n+1} как $x_{x+1} = x_n - \frac{f(x_n)}{f'(x_n)}$. Геометрический смысл: x_{n+1} – точка пересечения касательной к графику $f(x)$ в точке x с осью Ox

22.1 Оценка сходимости

Пусть f имеет корень в точке a, пусть $d_n = x_n - a$ (неточность приближения). Разложим $f(x)$ в точке x_n в многочлен Тейлора первой степени с остаточным членом в форме Лагранжа:

$$f(x) = f(x_n) + f'(x_n)\,(x - x_n) + \frac{f''(c_n)}{2}(x - x_n)^2 \qquad c_n \in (x, x_n) \tag{1}$$

Подставим $x = a$ в тождество

$$f(a) = 0 = f(x_n) + f'(x_n)(a - x_n) + \frac{f''(c_n)}{2}(a - x_n)^2 \tag{2}$$

Разделим на $f'(x_n)$

$$0 = \frac{f(x_n)}{f'(x_n)} + (a - x_n) + \frac{f''(c_n)}{2f'(x_n)}(a - x_n)^2 \tag{3}$$

Перенесём $\frac{f(x_n)}{f'(x_n)} + (a - x_n)$ налево

$$x_{n+1} - a = (x_n - a) - \frac{f(x_n)}{f'(x_n)} = \frac{f''(c_n)}{2f'(x_n)}(a - x_n)^2 \tag{4}$$

Значит

$$d_{n+1} = d_n^2 \cdot \frac{f''(c_n)}{2f'(x_n)} \tag{5}$$

Если значение $\frac{f''(c_n)}{2f'(x_n)}$ ограничено (константой, не зависящей от n), то имеем место квадратичная сходимость

Пусть $f(x)$ имеет корень в точке a, дважды непрерывно дифференцируема в окрестности a и $f'(a) \neq 0$. Тогда в некоторой окрестности a значение $\frac{f''(c_n)}{2f'(x_n)}$ ограничено, значит имеет место квадратичная сходимость

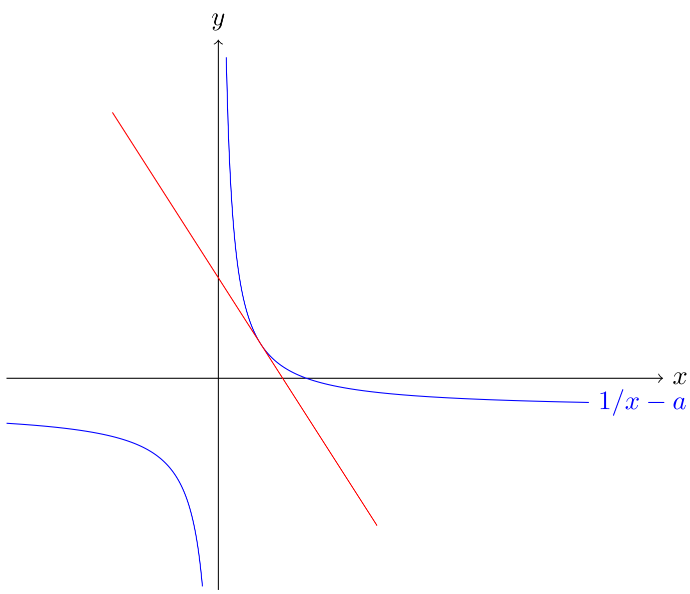
22.2 Пример для $1/a$

$$f(x) = 1/x - a \qquad f(1/a) = 0 \tag{6}$$

$$f'(x) = -\frac{1}{x^2} \qquad f''(x) = \frac{2}{x^3} \tag{7}$$

$$x_{n+1} = x_n - \frac{1/x_n - a}{-1/x_n^2} = x_n + \left(x_n - a \cdot x_n^2\right) = x_n(2 - ax_n) \tag{8}$$

$$d_{n+1} = d_n^2 \cdot \frac{f''(c_n)}{2f'(x_n)} \approx d_n^2 \cdot \frac{f''(1/a)}{2f'(1/a)} = -d_n^2 \cdot a \tag{9}$$



Из-за выпуклости $f(x)$ на $(0, +\infty)$ при $x_0 \in (0, 1/a)$ последовательность $\{x_n\}$ сойдётся к $1/a$ (так как при $x_n \in (0, 1/a)$ выполнено $x_n \leq x_{n+1} \leq 1/a$, так как касательная пересекает Ox строго правее x_n, но левее $1/a$ из-за выпуклости). При $x_0 \notin (0, 1/a]$ последовательность $\{x_n\}$ может как и сойтись к $1/a$, как и разойтись к $-\infty$

x_0 стоит брать чуть меньшим $1/a$

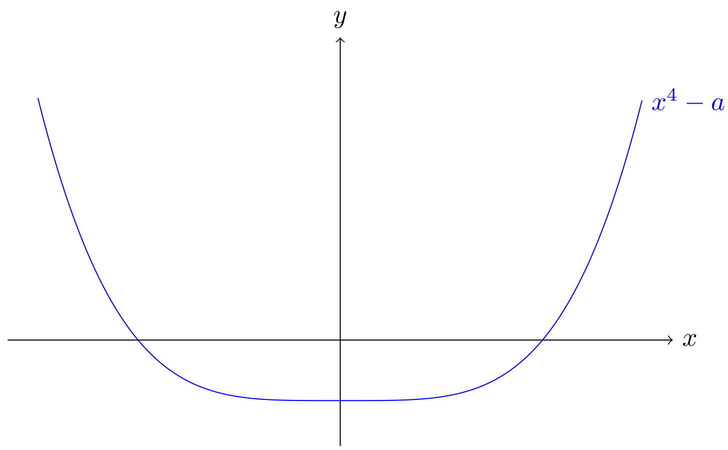
22.3 Пример для $\sqrt[4]{a}$

$$f(x) = x^4 - a \qquad f(\sqrt[4]{a}) = 0 \tag{10}$$

$$f'(x) = 4x^3 \qquad f''(x) = 12x^2 \tag{11}$$

$$x_{n+1} = x_n - \frac{x_n^4 - a}{4x_n^3} = x_n - \left(\frac{1}{4} \cdot x_n - \frac{a}{4x_n^3}\right) = \frac{3}{4} \cdot x_n + \frac{a}{4x_n^3} \tag{12}$$

$$d_{n+1} = d_n^2 \cdot \frac{f''(c_n)}{2f'(x_n)} \approx d_n^2 \cdot \frac{f''(\sqrt[4]{a})}{2f'(\sqrt[4]{a})} = d_n^2 \cdot \frac{3}{2\sqrt[4]{a}} \quad (13)$$



Из-за выпуклости $f(x)$ при $x_0 \in (\sqrt[4]{a}, +\infty)$ последовательность $\{x_n\}$ сойдётся к $\sqrt[4]{a}$ по аналогичным причинам. При $x_0 \in (-\infty, -\sqrt[4]{a})$ последовательность сойдётся к $-\sqrt[4]{a}$

x_0 стоит брать чуть большим $\sqrt[4]{a}$

23 Тернарный поиск

Пусть $f(x)$ имеет глобальный минимум в точке x^*, причём строго убывает на $(-\infty, x^*]$ и строго возрастает на $[x^*, +\infty)$.

Пусть известно, что $x^* \in [l, r]$ (исходно положим $(l, r) = (-C, C)$). Пусть $m_0 = (2l + r)/3$ и $m_1 = (l + 2r)/3$ (m_0, m_1 делят отрезок $[l, r]$ в отношении $1 : 1 : 1$). Вычислим значения $f(m_0), f(m_1)$

- Если $f(m_0) \leq f(m_1)$, то $x^* \in [l, m_1]$. Заменим (l, r) на (l, m_1)
- Если $f(m_0) \geq f(m_1)$, то $x^* \in [m_0, r]$. Заменим (l, r) на (m_0, r)

Повторим, пока не будет выполнено $r < l + \varepsilon$, где ε – требуемая точность

За итерацию длина отрезка $[l, r]$ уменьшается ровно в 1.5 раза, значит всего потребует не более чем $2\log_{1.5}\frac{C}{\varepsilon} + O(1) = O(\log\frac{C}{\varepsilon})$ вычислений функции $f(x)$.

23.1 Трюк с золотым сечением

Пусть $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$ – корень уравнения $x^2 - x - 1 = 0$. Будем брать $m_0 = \frac{1}{2\varphi+1}((\varphi + 1) \cdot l + \varphi \cdot r)$ и $m_1 = \frac{1}{2\varphi+1}(\varphi \cdot l + (\varphi + 1) \cdot r)$ (m_0, m_1 делят отрезок $[l, r]$ в отношении $\varphi : 1 : \varphi$).

Тогда, так как $\frac{1+2\varphi}{1+\varphi} = \varphi$, то на каждой итерации вышеописанного алгоритма, кроме первой, одно из значений $f(m_0), f(m_1)$ будет уже вычислено. Причём длина отрезка $[l, r]$ будет уменьшаться в $\frac{1+2\varphi}{1+\varphi} = \varphi$ раз. Поэтому суммарно будет произведено $\log_\varphi\frac{C}{\varepsilon} + O(1)$ вычислений функции $f(x)$. Что примерно в $\frac{2\ln(1.618)}{\ln(1.5)} \approx 2.37$ раз меньше $2\log_{1.5}\frac{C}{\varepsilon}$

24 Быстрое преобразование Фурье

24.1 Прямое преобразование

Пусть $P(x) \in \mathbb{C}[x]$ – некоторой многочлен с комплексными коэффициентами. Хотим вычислить значения $P(x)$ в корнях $x^n - 1$ (то есть корнях из единицы), для $n = 2^k$.

Будем делать это рекурсивно. Пусть требуется вычислить значения многочлена $P(x)$ в корнях $x^n - z$, где $z \in \mathbb{C} \setminus \{0\}$ и $\deg P(x) < n$

- Если $n = 1$, то многочлен $P(x)$ – константа, а единственный корень $x^n - z = x - z$ это z. Значит $P(z) = [x^0]P(x)$

- Если $n \neq 1$, то $n = 2k$ для некоторого натурального k. Разложим $x^n - z$ как
$$x^n - z = x^{2k} - z = (x^k - \sqrt{z})(x^k + \sqrt{z}) \tag{14}$$
Очевидно, что множество корней $x^{2k} - z$ это в точности объединение множеств корней $x^k - \sqrt{z}$ и $x^k + \sqrt{z}$. Вычислим $P_1(x) = P(x) \bmod (x^k - \sqrt{z})$ и $P_2(x) = P(x) \bmod (x^k + \sqrt{z})$ (это делается за $O(n)$). Затем рекурсивно вычислим значения $P_1(x)$ в корнях $x^k - \sqrt{z}$ и значения $P_2(x)$ в корнях $x^k + \sqrt{z}$

При реализации стоит заранее вычислить значение константы $\sqrt{z}$ для каждого рекурсивного вызова. Например можно предподсчитать последовательность
$$w_n = \begin{cases} 1 & \text{если } n = 0 \\ \cos\frac{\pi}{2n} + i\sin\frac{\pi}{2n} & \text{если } n = 2^k \\ w_{n-2^k} \cdot w_{2^k} & \text{иначе, где } k = \lfloor \log_2 n \rfloor \end{cases}$$
и в i-ом рекурсивном вызову на уровне d использовать $\sqrt{z} = w_i$ (в нумерации с нуля)

24.2 Обратное преобразование

Для обращения преобразования достаточно обратить вышеописанный алгоритм.
Рассмотрим рекурсивный вызов вышеописанного алгоритма

- Если $n = 1$, то обращать нечего, так как никаких вычислений не выполнялось
- Если $n \neq 1$, то $n = 2k$ для некоторого натурального k. Заметим, что значение $P(x) \bmod (x^{2k} - z)$ однозначно определяется значениями $P_1(x) = P(x) \bmod (x^k - \sqrt{z})$ и $P_2(x) = P(x) \bmod (x^k + \sqrt{z})$, так как многочлены $x^k - \sqrt{z}$ и $x^k + \sqrt{z}$ взаимно просты.

  Вспомним формулу для явного нахождения решения системы сравнений из китайской теоремы об остатках:
$$\begin{cases} X \equiv A \pmod{C} \\ X \equiv B \pmod{D} \end{cases} \tag{15}$$
$$X \equiv_{CD} A \cdot \operatorname{inv}(D, C) \cdot D + B \cdot \operatorname{inv}(C, D) \cdot C \tag{16}$$
  где $\operatorname{inv}(x, y)$ – обратное к x по модулю y

  Подставим $X = P(x),\ A = P_1(x),\ B = P_2(x),\ C = x^k - \sqrt{z},\ D = x^k + \sqrt{z}$

  Тогда $\operatorname{inv}(D, C) = \operatorname{inv}(D \bmod C, C) = \operatorname{inv}(2\sqrt{z}, C) = \frac{1}{2\sqrt{z}}$ и $\operatorname{inv}(C, D) = -\frac{1}{2\sqrt{z}}$

  Значит
$$P(x) = P_1(x) \cdot \frac{1}{2\sqrt{z}} \left(x^k + \sqrt{z}\right) - P_2(x) \cdot \frac{1}{2\sqrt{z}} \left(x^k - \sqrt{z}\right) = \tag{17}$$
$$\frac{1}{2} \left(P_1(x) \cdot \left(1 + x^k \cdot \frac{1}{\sqrt{z}}\right) + P_2(x) \cdot \left(1 - x^k \cdot \frac{1}{\sqrt{z}}\right) \right) \tag{18}$$
  что позволяет вычислить $P(x) \bmod x^n - z$ за $O(n)$

24.3 Альтернативная интерпретация

Вычисление $P_1(x)$ и $P_2(x)$ в 24.1 это взятие $P(x)$ по модулю $x^k - \sqrt{z}$ и $x^k + \sqrt{z}$ соответственно. Пусть $w = \sqrt{z}$. Представим $P(x)$ в виде $P(x) = A(x) + x^k B(x)$ (то есть $A(x), B(x)$ – младшая и старшая половина коэффициентов соответственно). Тогда выполнено
$$\begin{bmatrix} P_1(x) \\ P_2(x) \end{bmatrix} = \begin{bmatrix} 1 & w \\ 1 & -w \end{bmatrix} \times \begin{bmatrix} A(x) \\ B(x) \end{bmatrix} \tag{19}$$
Значит обратное преобразование может быть вычислено по формуле
$$\begin{bmatrix} A(x) \\ B(x) \end{bmatrix} = \begin{bmatrix} 1 & w \\ 1 & -w \end{bmatrix}^{-1} \times \begin{bmatrix} P_1(x) \\ P_2(x) \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ \frac{1}{w} & -\frac{1}{w} \end{bmatrix} \times \begin{bmatrix} P_1(x) \\ P_2(x) \end{bmatrix} \tag{20}$$
Прямое и обратное преобразование можно вычислять без использования дополнительной памяти, просто сразу записывая на место $A(x)$ и $B(x)$ значения $P_1(x), P_2(x)$ и наоборот.

24.4 Одновременное вычисление для двух многочленов из $\mathbb{R}[x]$

Даны два многочлена $A(x), B(x) \in \mathbb{R}[x]$, хотим вычислить их значения в корнях $(x^n - 1)$ за одно преобразование Фурье размера n (n – степень двойки). Заметим, что для $P(x) \in \mathbb{R}[x]$ выполнено $P(\overline{z}) = \overline{P(z)}$. Пусть $P(x) = A(x) + i \cdot B(x)$. Пусть w – какой-то из корней $(x^n - 1)$.

$$P(w) = A(w) + i \cdot B(w) \qquad P(\overline{w}) = A(\overline{w}) + i \cdot B(\overline{w}) \tag{21}$$

$$\begin{aligned} P(w) + \overline{P(\overline{w})} &= \\ A(w) + i \cdot B(w) + \overline{A(\overline{w}) + i \cdot B(\overline{w})} &= \\ A(w) + i \cdot B(w) + \overline{A(\overline{w})} + \overline{i \cdot B(\overline{w})} &= \\ A(w) + i \cdot B(w) + A(w) + \overline{i} \cdot \overline{B(\overline{w})} &= \\ A(w) + i \cdot B(w) + A(w) - i \cdot B(w) &= \\ 2A(w) & \end{aligned} \qquad \begin{aligned} P(w) - \overline{P(\overline{w})} &= \\ A(w) + i \cdot B(w) - \overline{A(\overline{w}) - i \cdot B(\overline{w})} &= \\ A(w) + i \cdot B(w) - \overline{A(\overline{w})} - \overline{i \cdot B(\overline{w})} &= \\ A(w) + i \cdot B(w) - A(w) - \overline{i} \cdot \overline{B(\overline{w})} &= \\ A(w) + i \cdot B(w) - A(w) + i \cdot B(w) &= \\ 2i \cdot B(w) & \end{aligned} \tag{22}$$

Что позволяет зная $P(w), P(\overline{w})$ вычислить $A(w), B(w)$ за $O(1)$. Все корни, кроме ± 1, разбиваются на пары сопряжённых.

24.5 Свёртка

Хотим вычислить произведение многочленов $A(x), B(x) \in \mathbb{C}[x]$. Пусть $k = \left\lceil \log_2 (\deg A(x) + \deg B(x) + 1) \right\rceil$ и $n = 2^k$. Пусть $C(x) = A(x)B(x)$ – искомый многочлен.

Заметим, что так как $\deg C(x) = \deg A(x) + \deg B(x) < n$, то достаточно найти $C(x) \bmod (x^n - 1)$, так как $C(x) \bmod (x^n - 1) = C(x)$. Значение $C(x) \bmod (x^n - 1)$ однозначно определяется значениями $C(x)$ в корнях $(x^n - 1)$. Используя 24.1 мы можем найти значения $A(x), B(x)$ в этим корнях, а затем и значения $C(x)$ в них, так как для любого $z \in \mathbb{C}$ выполнено $C(z) = A(z)B(z)$. Затем, используя 24.2, мы можем восстановить $C(x) \bmod (x^n - 1)$ по значениям в корнях

24.6 Прикладывания

Пусть даны две последовательности $a_0, a_1, a_2, ..., a_{n-1}$ и $b_0, b_1, ..., b_{m-1}$, причём $m \leq n$. Хотим для каждого k от 0 до $n - m$ найти сумму

$$c_k = \sum_{i=0}^{m-1} a_{k+i} \cdot b_i \tag{23}$$

То есть как бы *приложить* последовательность b к последовательности a, и вычислить сумму поэлементных произведений. Для этого просто вычислим произведение $A(x)B(x)$, где

$$A(x) = \sum_{i=0}^{n-1} x^i \cdot a_i \qquad B(x) = \sum_{i=0}^{m-1} x^i \cdot b_{m-1-i} \tag{24}$$

То есть свёртку последовательности a с развёрнутой последовательностью b.

Заметим, что последовательные $n-m+1$ коэффициентов, начиная с коэффициента при x^{m-1} и будет результатом. Причём в свёртке можно использовать $k = \lceil \log_2 n \rceil$, а не $k = \lceil \log_2 (n + m - 1) \rceil$, так как умножение по модулю $(x^n - 1)$ это *циклическая* свёртка, и *переполнение* не затронет результат, так как им будет затронуты не более чем $m - 2$ первых членов

24.7 Поиск по шаблону

Чтобы сделать поиск по шаблону, для каждого прикладывания вычислим

$$\sum_{x,y} (x - y)^2 \tag{25}$$

где сумма берётся по всем парам прикладываемых к друг другу символов. Сумма равна 0 тогда и только тогда, когда все символы в прикладываемых парах совпадают. Для вычисления суммы раскроем скобки и каждое слагаемое посчитаем отдельно (для слагаемых с $\deg_x > 0$ и $\deg_y > 0$ понадобится свёртка)

Если разрешается иметь отличие на не более чем 1, то аналогичным образом вычислим

$$\sum_{x,y} (x-y-1)^2 (x-y)^2 (x-y+1)^2 \tag{26}$$

Такая сумма равна 0, тогда и только тогда, когда все пары символов отличаются не более, чем на 1

25 Деление чисел длины n за $O(n \log n)$

Для простоты будем считать, что основание системы счисления 10. Обозначим за $|a| = \lfloor \log_{10} a \rfloor + 1$ – количество цифр в числе a.

Пусть мы хотим разделить число A на число B, то есть найти $Q = \left\lfloor \frac{A}{B} \right\rfloor$. Пусть $n = |A|, m = |B|$ и $m \leq n$. Заметим, что в частном будет примерно (с точностью до ± 1) $n - m$ цифр. Пусть $k = n+1$ и вычислим приближённое частное Q'

$$Q' = \left\lfloor \frac{A \cdot \left\lfloor \frac{10^k}{B} \right\rfloor}{10^k} \right\rfloor \tag{27}$$

Число $\left\lfloor \frac{10^k}{B} \right\rfloor$ вычисляется применением метода Ньютона на функции $f(x) = \frac{10^k}{x} - B$

$$f(x) = \frac{10^k}{x} - B \qquad f'(x) = -\frac{10^k}{x^2} \tag{28}$$

$$x \mapsto x - \frac{f(x)}{f'(x)} = x - \frac{\frac{10^k}{x} - B}{-\frac{10^k}{x^2}} = x + \frac{10^k \cdot x - B \cdot x^2}{10^k} = \tag{29}$$

$$\frac{2x \cdot 10^k - B \cdot x^2}{10^k} \approx \left\lfloor \frac{2x \cdot 10^k - B \cdot x^2}{10^k} \right\rfloor \tag{30}$$

Все вычисления можно производить в целых числах, но нужно быть осторожным, при слишком маленьком результате итерация Ньютона может не смочь *перепрыгнуть* от одного целого числа к другому. Так как найденное значение частного будет лишь приближением, нужно произвести *коррекцию* ошибки на $O(1)$ (частный случай описан ниже).

Оценим ошибку

$$\left\lfloor \frac{10^k}{B} \right\rfloor = \frac{10^k}{B} - \theta \qquad 0 \leq \theta < 1 \tag{31}$$

$$\implies Q' = \left\lfloor \frac{A \cdot \left\lfloor \frac{10^k}{B} \right\rfloor}{10^k} \right\rfloor = \left\lfloor \frac{A \cdot \left(\frac{10^k}{B} - \theta \right)}{10^k} \right\rfloor = \left\lfloor \frac{A}{B} - \frac{A \cdot \theta}{10^k} \right\rfloor \tag{32}$$

Так как $A < 10^k$, то *ошибка*, то есть $\frac{A \cdot \theta}{10^k}$ не превосходит 1, значит приближённое частное либо равно настоящему, либо на 1 меньше. Формально $Q' \leq Q \leq Q + 1$. Вычислив $R' = A - Q' \cdot B$ и сравнив R' с B, можно понять, какой из случаев выполнился, а значит и вычислить Q.