



Не всякую задачу можно решить

Что такое алгоритм, почему одни задачи считаются мгновенно, другие — миллиарды лет, а третьи не решаются никогда



Техника · 13–17 лет · 14 минут

Компьютеры становятся быстрее каждый год, и кажется, что вопрос любой сложности — вопрос времени. Это неверно дважды: есть задачи, которые не успеет решить никакая машина, и задачи, которые доказано неразрешимы вообще.

Три вида задач

Разговоры о компьютерах обычно строятся на молчаливом допущении: машины становятся мощнее, значит, рано или поздно решится что угодно.

Допущение неверно, и неверно двумя разными способами.

Задачи делятся на три группы, и границы между ними проведены не мощностью железа, а математикой.

Первая: решаются быстро. Отсортировать миллион чисел, найти кратчайший путь по карте, проверить, есть ли слово в словаре. Тут всё хорошо: растёт задача — время растёт умеренно.

Вторая: решаются, но не успеем. Решение существует и известно, как его искать. Просто искать придётся дольше, чем существует Вселенная.

Третья: не решаются никогда. Доказано, что алгоритма нет. Не «пока не нашли» — а нет и не будет.

Вторая и третья группы — самое интересное, и они меняют представление о том, что вообще может машина.

Что такое алгоритм

Сначала договоримся о слове, потому что дальше всё на нём держится.

Алгоритм — это точная последовательность действий, приводящая к результату за конечное число шагов. Ключевых требований три: **однозначность** (на каждом шаге понятно, что делать), **конечность** (когда-нибудь кончится), **результативность** (в конце получится ответ).

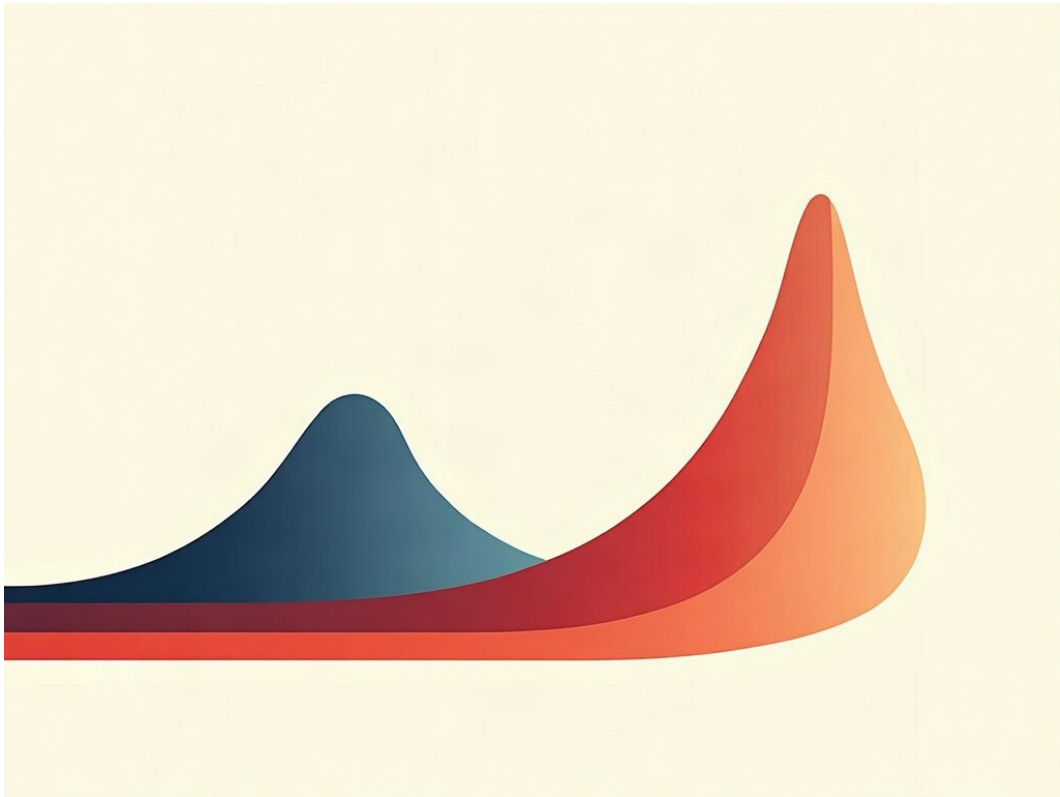
Слово происходит от имени Аль-Хорезми, среднеазиатского математика IX века: его книгу об индийском счёте перевели на латынь, и искажённое имя автора стало обозначать способ вычисления вообще.

Важно, что алгоритм не обязан быть компьютерным. Рецепт борща, инструкция по сборке шкафа, способ деления в столбик — алгоритмы. Компьютер просто выполняет их быстро и не отвлекаясь.

Почему быстрая машина не спасает

Теперь ключевая идея, которую стоит понять один раз и на всю жизнь.

Важно не то, сколько времени алгоритм работает на конкретной задаче. Важно, **как это время растёт, когда задача становится больше.**



Две кривые, между которыми пропасть

Сравним два роста.

Алгоритм А: задача вдвое больше — времени вдвое больше. Было 10 городов и секунда — стало 20 городов и две секунды. Удвоил мощность машины — удвоил посильный размер задачи.

Алгоритм В: каждый новый элемент **удваивает** время. Было 10 элементов и секунда. Стало 20 — тысяча секунд. Стало 30 — миллион. Стало 60 — миллиард миллиардов секунд, это в сотни миллионов раз дольше возраста Вселенной.

И вот главное. Удвой мощность компьютера для алгоритма В — и ты выиграешь **один элемент**. Не вдвое больше, а плюс один. Сделай компьютер в тысячу раз быстрее — выиграешь десять элементов.

Отсюда вывод, который обычно и удивляет: **против такого роста железо бессильно**. Тут нужен другой алгоритм, а не другая машина.

Сложность алгоритма

Мера того, как быстро растёт время работы при увеличении размера задачи. Записывают в виде «О-большое»: $O(n)$ — время пропорционально размеру, $O(n^2)$ — квадрату размера, $O(2^n)$ — удваивается с каждым элементом. Практическая граница между «можно» и «нельзя» проходит примерно там, где показатель уходит из степени в основание: задачи со степенным ростом обычно посильны, с показательным — обычно нет.

Задача, которую все знают

Классический пример задачи из второй группы — про коммивояжёра.

Есть города и расстояния между ними. Надо объехать все по одному разу и вернуться, потратив как можно меньше.

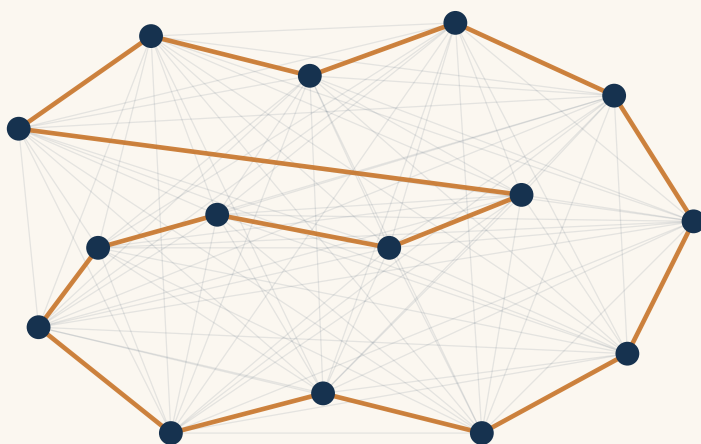
Звучит проще некуда. Решение очевидно: перебрать все маршруты и выбрать короткий.

Теперь посчитаем. Для 5 городов маршрутов десятки. Для 10 — сотни тысяч. Для 20 — больше, чем секунд прошло с начала Вселенной. Для 60 — число, для которого не хватит атомов в наблюдаемой Вселенной, если записывать его камешками.

Заметь: задача не выдуманная и не редкая. Её решают каждый день — развозя посылки, планируя маршруты автобусов, сверля отверстия в плате.

Как же с ней справляются? Никак — в смысле точного решения. С ней **мирятся**:

- Ищут не лучший маршрут, а хороший: на несколько процентов хуже оптимального, зато за секунду.



пятнадцать городов — это уже больше сорока миллиардов маршрутов

перебрать все не выйдет никогда

Пятнадцать городов — и перебрать все маршруты уже невозможно

- Пользуются приёмами, которые обычно работают, хотя гарантий не дают.
- Сокращают перебор, заранее отбрасывая заведомо плохие ветки.

И это типичная инженерная развязка: когда точный ответ недостижим, меняют требование — с «наилучшего» на «достаточно хорошего».

Задачи, у которых решения нет вовсе

Теперь третья группа, и она интереснее всего, потому что речь не о нехватке времени.

В 1936 году Алан Тьюринг доказал, что существует задача, для которой алгоритма нет в принципе. Называется она проблемой остановки.



Программа, которая никогда не остановится, выглядит как программа, которая ещё считает

Формулировка простая. Требуется написать программу, которая, получив на вход любую другую программу и её данные, отвечает: остановится ли та когда-нибудь или будет работать вечно.

Задача полезная: такая программа ловила бы зависания до запуска.

Тьюринг доказал, что её не существует. Ход доказательства можно передать без математики.

Почему такой программы не может быть

Допустим, кто-то написал программу **Решатель**. Она берёт любую программу и отвечает «остановится» или «зациклится», причём всегда правильно.

Теперь построим на её основе вредную программу — назовём её **Упрямец**. Работает она так:

1. Получить на вход программу.
2. Спросить у Решателя, остановится ли она.
3. Если Решатель сказал «остановится» — заикнуться навсегда.
4. Если сказал «заикнуться» — немедленно остановиться.

Упрямец просто делает наоборот. Ничего сложного, и написать его можно, раз Решатель существует.

А теперь подадим Упряму на вход его самого.

Спросим: остановится ли Упрямец, работая над Упрямом?

Допустим, Решатель говорит «остановится». Тогда Упрямец, по своему правилу, заикнется. Значит, Решатель ошибся.

Допустим, Решатель говорит «заикнуться». Тогда Упрямец останавливается. Значит, Решатель ошибся снова.

Решатель ошибается в обоих случаях. Но мы предположили, что он никогда не ошибается. Значит, предположение неверно: **Решателя не существует.**

Заметь, что мы не искали алгоритм и не сдались. Мы доказали невозможность — а это совсем другое знание, гораздо более прочное.

Почему это не повод унывать

Из сказанного легко сделать мрачный вывод. Он был бы неверным.

Невозможность в общем случае не значит невозможность в частных. Программ, про которые нельзя сказать, заикнутся ли они, полно в теории; на практике анализаторы кода прекрасно ловят огромную часть зависаний. Просто они не обещают поймать все.

Приблизительное часто достаточно. Маршрут на процент хуже идеального доставит посылки ничуть не хуже.

Ограничение — это знание. Понимать, что задача неразрешима, полезнее, чем годами искать решение. Огромное количество сил в истории математики было потрачено на поиски того, чего нет, — пока не доказали, что нет.

Границы двигаются иначе, чем кажется. Успехи в трудных задачах приходят не от роста мощности, а от новых подходов: другой алгоритм, другая постановка, отказ от точности ради скорости.

Отменяют ли эти границы новые технологии

Отменяют частично. Квантовые вычисления действительно меняют расклад для некоторых задач: есть алгоритмы, дающие огромное ускорение там, где обычные машины бессильны. Класс посильного не зафиксирован навечно.

Не отменяют главного. Квантовый компьютер быстрее не для всего, а для узкого круга задач особой структуры, и проблему остановки он не решит: её невозможность доказана не про скорость, а про саму логику. Никакое железо не отменит логического противоречия.

Практический вывод для обеих сторон: следить стоит не за мощностью машин, а за появлением новых алгоритмов. Именно там происходят настоящие сдвиги.

Что из этого стоит унести

Несколько мыслей, полезных вне программирования.

Спрашивай, как растёт. Не «сколько это займёт сейчас», а «что будет, когда станет вдвое больше». Вопрос отделяет решения, которые доживут до роста, от тех, что рухнут.

Различай «трудно» и «невозможно». Это разные вещи, и путать их дорого в обе стороны: упорствовать в невозможном и сдаваться перед трудным одинаково неприятно.

Помни, что доказанная невозможность — результат. В отличие от «у меня не получилось», она говорит нечто о мире, а не о тебе.

Меняй требование, если не сходится. Когда точного ответа нет, почти всегда существует вопрос попроще, ответ на который решает исходную задачу практически. Умение найти такой вопрос — и есть инженерия.

Задачи, которые легко проверить и трудно решить

Есть особая порода задач, о которой стоит сказать отдельно, — на ней держится половина современной криптографии.

Заметь асимметрию на простом примере. Головоломку sudoku трудно решать и мгновенно проверять: увидев заполненную сетку, ты за минуту убедишься, что всё верно. То же с разложением числа на множители: найти множители большого числа тяжело, а перемножить найденное и сверить — секунда.

Класс таких задач — «решение ищется долго, проверяется быстро» — обозначают буквами NP. Задачи, которые и решаются быстро, — P.

И вот главный открытый вопрос всей информатики: **совпадают ли эти два класса?** Иначе говоря, верно ли, что у всякой задачи с быстрой проверкой есть и быстрое решение, просто мы его не нашли?

Почти все специалисты считают, что не совпадают, — но доказательства нет уже полвека. За него назначена премия в миллион долларов, и она не выплачена.

Практическая сторона вопроса огромна. Если вдруг окажется, что классы совпадают и быстрый способ есть, рухнет вся защита, построенная на трудности разложения чисел, — то есть банковские переводы, подписи, защищённая связь. Одновременно решатся задачи планирования, укладки, расписаний, на которых человечество теряет миллиарды.

Любопытный поворот: мы живём в мире, безопасность которого основана на нерешённой математической задаче. Не на доказанной невозможности — а на том, что решение очень долго ищут и не находят.

Главное в трёх строчках

- Важна не скорость алгоритма, а то, как растёт время при увеличении задачи: против показательного роста рост мощности машины бессилён — удвоение мощности добавляет один элемент.
- Часть практических задач точно не решается за разумное время, и с ними мирятся, заменяя «наилучший ответ» на «достаточно хороший».
- Есть задачи, для которых алгоритма не существует вовсе, и это доказано: проблема остановки неразрешима не из-за нехватки мощности, а из-за логического противоречия.

Вопросы для размышления

- Почему увеличение мощности компьютера в тысячу раз добавляет к показательной задаче всего десяток элементов?
- Чем доказательство невозможности отличается от утверждения «мы пока не нашли решения»?
- Задача о коммивояжёре не решается точно, но посылки развозятся каждый день. Что именно изменили в постановке, чтобы это стало возможным?