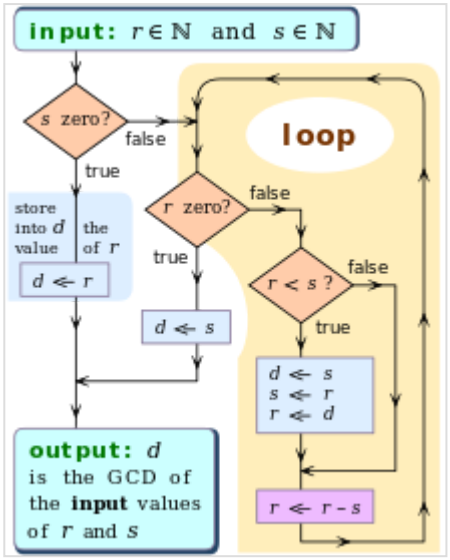
WIKIPEDIA

Алгоритм

В математике и информатике **алгоритм** (/) представляет собой конечную последовательность строгих.....[1] Алгоритмы используются в качестве спецификаций для выполнения вычислений и обработки данных. Более продвинутые алгоритмы могут использовать условные обозначения, чтобы перенаправлять выполнение кода различными маршрутами (называемыми автоматическим принятием решений) и выводить достоверные выводы (называемые автоматическими рассуждениями), в конечном итоге достигая автоматизации. Метафорическое использование человеческих характеристик в качестве дескрипторов машин уже практиковалось Аланом Тьюрингом с такими терминами, как "память", "поиск" и "стимул".[2]



Блок-схема использования последовательных вычитаний для нахождения наибольшего общего делителя чисел *r* и *s*

Напротив, эвристический - это подход к решению задач, который может быть указан не полностью или не гарантировать правильных или оптимальных результатов, особенно в проблемных областях, где нет четко определенного правильного или оптимального результата.[3] Например, рекомендательные системы социальных сетей таким образом полагаются на эвристику, что, хотя в популярных СМИ 21 века их широко называют "алгоритмами", они не могут давать правильных результатов из-за характера проблемы.

Как эффективный метод, алгоритм может быть выражен в пределах конечного объема пространства и времени[4] и на четко определенном формальном языке[5] для вычисления функции.[6] Начиная с начального состояния и начального ввода (возможно, пустого),[7] инструкции описывают вычисление, которое при выполнении проходит через конечное[8] количество четко определенных последовательных состояний, в конечном итоге производя "вывод"[9] и завершается в конечном конечном состоянии. Переход из одного состояния в другое не обязательно детерминированный; некоторые алгоритмы, известные как рандомизированные алгоритмы, включают случайный ввод.[10]

История

Древние алгоритмы

С древних времен были подтверждены пошаговые процедуры решения математических задач. Сюда входит вавилонская математика (около 2500 г. до н.э.),[11] египетская математика (около 1550 г. до н.э.), [11] индийская математика (около 800 г. до н.э. и позже; например, Сутры Шульбы, школа Кералы и Брахмасфутасиддханта),[12][13] Оракул Ифа (https://www.jstor.org/stable/3027363) (около 500 г. до н.э.), греческая математика (около 240 г. до н.э., например, решето Эратосфена и Алгоритм Евклида),[14] и арабская математика (9 век, например, криптографические алгоритмы для взлома кода на основе частотного анализа).[15]

Аль-Хорезми и термин *алгоритм*

Около 825 года Мухаммад ибн Муса аль-Хоризми написал *китаб аль-исаб аль-хинди* ("Книга индийских вычислений") и *китаб аль-джам ва'ль-тафрик аль-исаб аль-Хинди* ("Сложение и вычитание в индийской арифметике"). Оба этих текста в оригинале на арабском языке в настоящее время утрачены. (Однако его другая книга по алгебре сохранилась.)[16]

В начале 12 века появились латинские переводы упомянутых текстов аль-Хорезми, включающие индуистско–арабскую систему счисления и арифметику: *Liber Algorismi de practica arismetrice* (приписывается Иоанну Севильскому) и *Liber Algorismi de numero Indorum* (приписывается Аделарду из Бата).[17] Таким образом, *альгоаризми* или *algorismi* - это латинизация имени Аль-Хорезми; текст начинается с фразы *Dixit Algorismi* ("Так говорил Аль-Хорезми").[18]

В 1240 году Александр Вильдье пишет текст на латыни под названием *Carmen de Algorismo*. Он начинается словами:

> *Haec algorismus ars praesens dicitur, in qua / Talibus Indorum fruimur bis quinque figuris.*

что переводится как:

> Алгоритмизм - это искусство, с помощью которого в настоящее время мы используем те индийские цифры, число которых дважды пять.

Стихотворение длиной в несколько сотен строк вкратце описывает искусство вычисления с помощью индийских кубиков нового стиля (*Tali Indorum*), или индуистских цифр.[19]

Эволюция слова в английском языке

Около 1230 года засвидетельствовано английское слово *algorism*, а затем Чосером в 1391 году. Английский язык перенял французский термин.[20][21]

В 15 веке под влиянием греческого слова ἀριθμός (*arithmos*, "число"; *ср.* "арифметика") латинское слово было изменено на *algorithmus*.

В 1656 году в словаре английского языка *Glossographia* говорится:[22]

> *Алгоритмизм* ([латинский] *algorismus*) искусство или использование шифров или нумерации с помощью шифров; умение вести бухгалтерский учет.
>
> *Augrime* ([латинский] *algorithmus*) умение вести бухгалтерский учет или нумерацию.

В 1658 году, в первом издании *"Нового мира английских слов"*, говорится:[23]

> *Алгоритм*, (слово, составленное из *арабского* и *испанского* языков) искусство вычисления с помощью шифров.

В 1706 году, в шестом издании *"Нового мира английских слов"*, говорится:[24]

> *Алгоритм*, искусство вычисления или счета с помощью чисел, которое содержит пять основных правил арифметики, *а именно*. *Нумерация*, *сложение*, *вычитание*, *умножение* и *деление*; к которым может быть добавлено *извлечение корней*: это также называется *Logistica Numeralis*.

Алгоритмизм, практическая операция в нескольких частях *показной арифметики* или *алгебры*; иногда десятью цифрами это принимается за практику обычной арифметики.

В 1751 году в "*Спутнике молодого алгебраиста"* Дэниел Феннинг противопоставляет термины "*алгоризм"* и "*алгоритм*" следующим образом:[25]

Алгоритм означает первые принципы, а *алгоритмизм* - практическую часть, или знание того, как применить алгоритм на практике.

По крайней мере, с 1811 года подтверждено, что термин *алгоритм* означает "пошаговую процедуру" в английском языке.[26][27]

В 1842 году в Словаре науки, литературы и искусства говорилось:

АЛГОРИТМ, означает искусство вычислений применительно к какому-то конкретному предмету или каким-то определенным способом; как алгоритм чисел; алгоритм дифференциального исчисления.[28]

Использование компьютера

Диаграмма Ады Лавлейс из "Примечания G", первого опубликованного компьютерного алгоритма

В 1928 году частичная формализация современной концепции алгоритмов началась с попыток решить *Entscheidungsproblem* (проблему принятия решений), поставленную Дэвидом Гильбертом. Более поздние формализации были сформулированы как попытки определить "эффективную вычислимость"[29] или "эффективный метод".[30] Эти формализации включали рекурсивные функции Геделя–Гербранда Клини 1930, 1934 и 1935 годов, лямбда-исчисление Алонзо Черча 1936 года, формулировку 1 Эмиля Поста 1936 года и машины Тьюринга Алана Тьюринга в 1936-37 и 1939 годах.

Неформальное определение

Одним из неформальных определений является "набор правил, который точно определяет последовательность операций",[31] который будет включать все компьютерные программы (включая программы, которые не выполняют числовых вычислений) и (например) любую предписанную бюрократическую процедуру[32] или рецепт кулинарной книги.[33]

В общем, программа является алгоритмом только в том случае, если она в конечном итоге останавливается[34] — хотя бесконечные циклы иногда могут оказаться желательными.

Прототипическим примером алгоритма является алгоритм Евклида, который используется для определения максимального общего делителя двух целых чисел; пример (есть и другие) описан в блок-схеме выше и в качестве примера в более позднем разделе.

Булос, Джеффри и 1974, 1999 предлагают неформальное значение слова "алгоритм" в следующей цитате:

Ни один человек не может писать достаточно быстро, или достаточно долго, или достаточно мало † (†"все меньше и меньше без ограничений ... вы бы попытались написать "на молекулах, на атомах, на электронах"), чтобы перечислить все члены перечислимо бесконечного множества, записывая их имена одно за другим в некоторой нотации. Но

> люди могут сделать что-то не менее полезное в случае определенных перечислимо бесконечных множеств: они могут дать *явные инструкции для определения n-го члена множества* для произвольного конечного *n*. Такие инструкции должны быть даны совершенно явно, в форме, в которой *им могла бы следовать вычислительная машина* или *человек, способный выполнять только очень элементарные операции с символами*.[35]

"Перечислимо бесконечное множество" - это такое, элементы которого могут быть приведены во взаимно однозначное соответствие с целыми числами. Таким образом, Булос и Джеффри говорят, что алгоритм подразумевает инструкции для процесса, который "создает" выходные целые числа из *произвольного* "входного" целого числа или целых чисел, которые, теоретически, могут быть сколь угодно большими. Например, алгоритм может представлять собой алгебраическое уравнение, такое как *y* = *m* + *n* (т. Е. Две произвольные "входные переменные" *m* и *n*, которые выдают результат *y*), но попытки различных авторов определить это понятие указывают на то, что это слово подразумевает гораздо больше, что-то порядка (для примера сложения):

> Точные инструкции (на языке, понятном "компьютеру")[36] для быстрого, эффективного, "хорошего"[37] процесса, который определяет "ходы" "компьютера" (машины или человека, оснащенного необходимой внутренней информацией и возможностями)[38] для поиска, декодирования и последующей обработки произвольных входных целых чисел / символов *m* и *n*, символов + и = ... и "эффективно"[39] производить в "разумные" сроки[40] вывод целого числа *y* в указанном месте и в указанном формате.

Концепция *алгоритма* также используется для определения понятия разрешимости — понятия, которое является центральным для объяснения того, как возникают формальные системы, начиная с небольшого набора аксиом и правил. В логике время, необходимое для выполнения алгоритма, не может быть измерено, поскольку оно, по-видимому, не связано с обычным физическим измерением. Из таких неопределенностей, характеризующих текущую работу, вытекает отсутствие определения *алгоритма*, которое подходило бы как к конкретному (в некотором смысле), так и к абстрактному использованию термина.

Большинство алгоритмов предназначены для реализации в виде компьютерных программ. Однако алгоритмы реализуются и другими средствами, например, в биологической нейронной сети (например, в человеческом мозге, реализующем арифметику, или в насекомом, ищущем пищу), в электрической цепи или в механическом устройстве.

Формализация

Алгоритмы необходимы для того, как компьютеры обрабатывают данные. Многие компьютерные программы содержат алгоритмы, которые детализируют конкретные инструкции, которые компьютер должен выполнять — в определенном порядке — для выполнения определенной задачи, такой как расчет зарплаты сотрудников или печать табелей успеваемости учащихся. Таким образом, алгоритмом можно считать любую последовательность операций, которая может быть смоделирована с помощью полной по Тьюрингу системы. Среди авторов, утверждающих этот тезис, Мински (1967), Сэвидж (1987) и Гуревич (2000):

> Мински: "Но мы также будем утверждать, что с Тьюрингом ... что любая процедура, которую "естественно" можно было бы назвать эффективной, на самом деле может быть реализована (простой) машиной. Хотя это может показаться экстремальным, аргументы ... в его пользу трудно опровергнуть".[41] Гуревич: "... неофициальный аргумент Тьюринга в пользу его тезиса оправдывает более сильный тезис: каждый алгоритм может быть смоделирован машиной Тьюринга ... согласно Сэвиджу [1987], алгоритм - это вычислительный процесс, определяемый машиной Тьюринга".[42]

Машины Тьюринга могут определять вычислительные процессы, которые не завершаются. Неформальные определения алгоритмов обычно требуют, чтобы алгоритм всегда завершался. Это требование делает задачу определения того, является ли формальная процедура алгоритмом, невозможной в общем случае — из-за основной теоремы теории вычислимости, известной как проблема остановки.

Обычно, когда алгоритм связан с обработкой информации, данные могут быть считаны из источника ввода, записаны на устройство вывода и сохранены для дальнейшей обработки. Сохраненные данные рассматриваются как часть внутреннего состояния объекта, выполняющего алгоритм. На практике состояние хранится в одной или нескольких структурах данных.

Для некоторых из этих вычислительных процессов алгоритм должен быть строго определен и конкретизирован таким образом, чтобы он применялся во всех возможных обстоятельствах, которые могут возникнуть. Это означает, что любые условные шаги должны выполняться систематически, в каждом конкретном случае; критерии для каждого случая должны быть четкими (и вычислимыми).

Поскольку алгоритм представляет собой точный список точных шагов, порядок вычислений всегда имеет решающее значение для функционирования алгоритма. Обычно предполагается, что инструкции перечислены явно и описываются как начинающиеся "сверху" и идущие "вниз" — идея, которая более формально описывается *потоком управления*.

До сих пор обсуждение формализации алгоритма предполагало предпосылки императивного программирования. Это наиболее распространенная концепция, которая пытается описать задачу в дискретных, "механических" терминах. С этой концепцией формализованных алгоритмов связана операция присваивания, которая устанавливает значение переменной. Он основан на интуиции "памяти" в виде блокнота. Пример такого задания можно найти ниже.

Для получения некоторых альтернативных концепций того, что представляет собой алгоритм, см. Функциональное программирование и логическое программирование.

Выражающие алгоритмы

Алгоритмы могут быть выражены во многих видах обозначений, включая естественные языки, псевдокод, блок-схемы, диаграммы драконов, языки программирования или управляющие таблицы (обрабатываемые интерпретаторами). Выражения алгоритмов на естественном языке, как правило, многословны и неоднозначны и редко используются для сложных или технических алгоритмов. Псевдокод, блок-схемы, drakon-диаграммы и контрольные таблицы - это структурированные способы выражения алгоритмов, которые позволяют избежать многих двусмысленностей, распространенных в утверждениях, основанных на естественном языке. Языки программирования в первую очередь предназначены для выражения алгоритмов в форме, которая может быть выполнена компьютером, но они также часто используются как способ определения или документирования алгоритмов.

Возможно большое разнообразие представлений, и можно выразить данную программу машины Тьюринга в виде последовательности машинных таблиц (подробнее см. Конечный автомат, таблицу перехода состояний и таблицу управления), в виде блок-схем и диаграмм драконов (подробнее см. Диаграмму состояний) или в виде рудиментарного машинного кода или ассемблерного кода, называемых "наборами четверок" (подробнее см. Машину Тьюринга).

Представления алгоритмов можно разделить на три общепринятых уровня описания машины Тьюринга следующим образом:[43]

1 Высокоуровневое описание
> "... проза для описания алгоритма, игнорирующего детали реализации. На этом уровне нам не нужно упоминать, как машина управляет своей лентой или головкой ".

2 Описание реализации
> "... проза, используемая для определения того, как машина Тьюринга использует свою головку и как она хранит данные на своей ленте. На этом уровне мы не приводим подробностей о состояниях или функции перехода ".

3 Формальное описание

Наиболее подробный, "самый низкий уровень", дает "таблицу состояний" машины Тьюринга.

Пример простого алгоритма "Add m + n", описанного на всех трех уровнях, см. в Примерах.

Дизайн

Разработка алгоритма относится к методу или математическому процессу для решения задач и разработки алгоритмов. Разработка алгоритмов является частью многих теорий решений, таких как принцип "разделяй и властвуй" или динамическое программирование в рамках исследования операций. Методы проектирования и реализации конструкций алгоритмов также называются шаблонами проектирования алгоритмов,[44] примеры включают шаблон метода шаблона и шаблон декоратора.

Одним из наиболее важных аспектов разработки алгоритма является эффективность использования ресурсов (время выполнения, использование памяти); обозначение big O используется, например, для описания роста времени выполнения алгоритма по мере увеличения размера его входных данных.

Типичные этапы разработки алгоритмов:

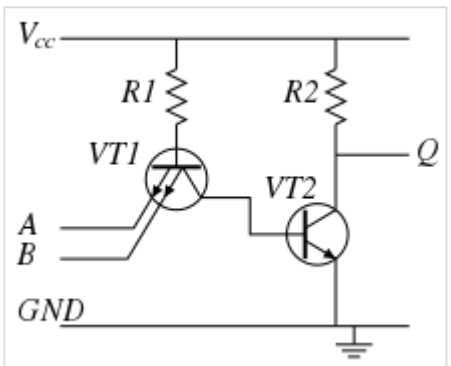
1. Определение проблемы
2. Разработка модели
3. Спецификация алгоритма
4. Разработка алгоритма
5. Проверка корректности алгоритма
6. Анализ алгоритма
7. Реализация алгоритма
8. Программа тестирования
9. Подготовка документации

Компьютерные алгоритмы

"Элегантные" (компактные) программы, "хорошие" (быстрые) программы: понятие "простота и элегантность" неофициально встречается у Кнута и именно у Чайтена:

> Кнут: " ... нам нужны *хорошие* алгоритмы в некотором слабо определенном эстетическом смысле. Один критерий ... это продолжительность времени, затраченного на выполнение алгоритма .... Другими критериями являются адаптируемость алгоритма к компьютерам, его простота и элегантность и т.д."[45]

> Чайтин: "... программа "элегантна", под чем я подразумеваю, что это минимально возможная программа для получения результата, который она выполняет"[46]

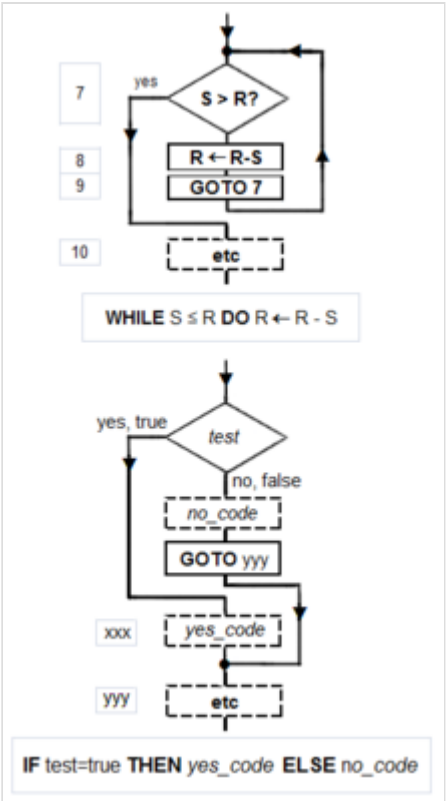


Логический алгоритм NAND, реализованный электронным способом в чипе 7400

Чайтин предваряет свое определение словами: "Я покажу, что вы не можете доказать, что программа "элегантна" " — такое доказательство решило бы проблему остановки (там же).

Алгоритм в сравнении с функцией, вычислимой с помощью алгоритма: для данной функции может существовать несколько алгоритмов. Это верно даже без расширения доступного программисту набора инструкций. Роджерс замечает , что "Это так ... важно различать понятие *алгоритма*, т.е. процедуры, и понятие *функции, вычислимой с помощью алгоритма*, т.Е. отображения, получаемого с помощью процедуры. Одна и та же функция может иметь несколько разных алгоритмов".[47]

К сожалению, может существовать компромисс между добротностью (скоростью) и элегантностью (компактностью) — элегантной программе может потребоваться больше шагов для завершения вычислений, чем менее элегантной. Ниже приведен пример, использующий алгоритм Евклида.



Примеры блок-схем канонических структур Бема-Якопини: последовательность (прямоугольники, спускающиеся по странице), WHILE-DO и IF-THEN-ELSE. Три структуры состоят из примитивного условного перехода (`IF test THEN GOTO step xxx`, показан в виде ромба), безусловного перехода (прямоугольник), различных операторов присваивания (прямоугольник) и HALT (прямоугольник). Вложение этих структур внутри блоков присваивания приводит к сложным диаграммам (ср. Tausworthe 1977: 100, 114).

Компьютеры (и вычислители), модели вычислений: Компьютер (или человеческий "компьютер"[48]) - это ограниченный тип машины, "дискретное детерминированное механическое устройство"[49], которое слепо следует своим инструкциям.[50] Примитивные модели Мельзака и Ламбека[51] свели это понятие к четырем элементам: (i) дискретные, различимые *местоположения*, (ii) дискретные, неразличимые *счетчики*[52] (iii) агент и (iv) список инструкций, которые *эффективны* относительно возможностей агента.[53]

Мински описывает более подходящую вариацию модели "счеты" Ламбека в своих "Очень простых основах вычислимости".[54] Машина Мински последовательно выполняет свои пять (или шесть, в зависимости от способа подсчета) инструкций, если только условный переход IF-THEN или безусловный переход GOTO не нарушают последовательность выполнения программы. Помимо HALT, машина Мински включает в себя три операции *присваивания* (замены, подстановки)[55]: НОЛЬ (например, содержимое ячейки заменяется на 0: L ← 0), ПРЕЕМНИК (например, L ← L + 1) и УМЕНЬШЕНИЕ (например, L ← L – 1).[56] Программисту редко приходится писать "код" с таким ограниченным набором команд. Но Мински показывает (как и Мелзак и Ламбек), что его машина является полной по Тьюрингу только с четырьмя общими *типами* инструкций: условный переход, безусловный переход, назначение / замена / substitution и HALT. Однако для полноты по Тьюрингу также требуется несколько различных инструкций присваивания (например, DECREMENT, INCREMENT и ZERO / CLEAR / EMPTY для машины Мински); их точная спецификация в некоторой степени зависит от разработчика. Безусловный переход удобен; его можно создать, инициализировав выделенное местоположение нулем, например, инструкцией " Z ← 0 "; после этого инструкция, ЕСЛИ Z = 0, ТО переход xxx является безусловным.

Моделирование алгоритма: компьютерный язык: Кнут советует читателю, что "лучший способ изучить алгоритм - это попробовать его ... немедленно возьмите ручку и бумагу и проработайте пример".[57] Но как насчет симуляции или выполнения реальной вещи? Программист должен перевести алгоритм на язык, который симулятор / компьютер / вычислитель может *эффективно* выполнять. Стоун приводит пример этого: при вычислении корней квадратного уравнения компьютер должен знать, как извлечь квадратный корень. Если они этого не делают, то алгоритм, чтобы быть эффективным, должен предоставлять набор правил для извлечения квадратного корня.[58]

Это означает, что программист должен знать "язык", который эффективен по отношению к целевому вычислительному агенту (компьютеру / вычислительнице).

Но какую модель следует использовать для моделирования? Ван Эмде Боас замечает: "даже если мы основываем теорию сложности на абстрактных, а не на конкретных машинах, произвольность выбора модели сохраняется. Именно на этом этапе появляется понятие "*симуляция*".[59] При измерении скорости важен набор инструкций. Например, подпрограмма в алгоритме Евклида для вычисления остатка выполнялась бы намного быстрее, если бы у программиста была доступна инструкция "модуль", а не просто вычитание (или хуже: просто "уменьшение" Мински).

Структурированное программирование, канонические структуры: Согласно тезису Черча–Тьюринга, любой алгоритм может быть вычислен с помощью модели, известной как полная по Тьюрингу, и, согласно демонстрациям Мински, полнота по Тьюрингу требует только четырех типов команд — условного перехода, безусловного перехода, присваивания, ОСТАНОВКИ. Кемени и Курц отмечают, что, хотя "недисциплинированное" использование безусловных GOTO и условных IF-THEN GOTO может привести к "спагетти-коду", программист может писать структурированные программы, используя только эти инструкции; с другой стороны, "также возможно, и не слишком сложно, писать плохо структурированные программы на структурированном языке".[60] Таусворт дополняет три канонические структуры Бема-Якопини:[61] ПОСЛЕДОВАТЕЛЬНОСТЬ, IF-THEN-ELSE и WHILE-DO еще двумя: ВЫПОЛНЕНИЕ ВО ВРЕМЯ и В СЛУЧАЕ.[62] Дополнительным преимуществом структурированной программы является то, что она поддается доказательствам корректности с использованием математической индукции.[63]

Канонические символы блок-схемы[64]: Графическое пособие, называемое блок-схемой, предлагает способ описания и документирования алгоритма (и соответствующей ему компьютерной программы). Подобно программному потоку машины Мински, блок-схема всегда начинается вверху страницы и продолжается вниз. Его основных символов всего четыре: направленная стрелка, показывающая ход выполнения программы, прямоугольник (SEQUENCE, GOTO), ромб (IF-THEN-ELSE) и точка (OR-tie). Канонические структуры Бема-Якопини состоят из этих примитивных форм. Подструктуры могут "гнездиться" в прямоугольниках, но только если из надстройки есть единственный выход. Символы и их использование для построения канонических структур показаны на диаграмме.

Примеры

Пример алгоритма

Один из простейших алгоритмов - найти наибольшее число в списке чисел случайного порядка. Для поиска решения требуется просмотреть каждое число в списке. Из этого следует простой алгоритм, который может быть изложен в высокоуровневом описании английской прозой следующим образом:

Описание высокого уровня:

1. Если в наборе нет чисел, то нет и самого большого числа.
2. Предположим, что первое число в наборе является наибольшим числом в наборе.
3. Для каждого оставшегося числа в наборе: если это число больше текущего наибольшего числа, считайте это число наибольшим числом в наборе.
4. Когда в наборе не осталось чисел для повторения, считайте, что текущее наибольшее число является наибольшим числом в наборе.

(Квази) формальное описание: Написанное прозой, но гораздо более близкое к языку высокого уровня компьютерной программы, ниже приводится более формальное кодирование алгоритма в псевдокоде или пиджин-коде:

```
Алгоритм с наибольшим числом
Входные данные: список чисел L.
Выходные данные: наибольшее число в списке L.
```

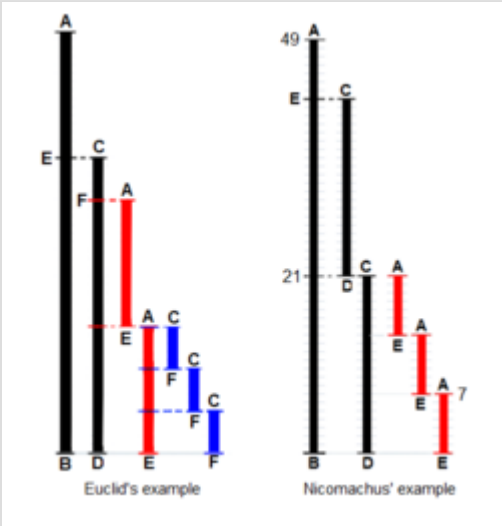
```
L.size if = 0 возвращает null
наибольший ← L[0]
для каждого элемента в L, сделайте
    если элемент > самый большой, тогда
        самый большой ← элемент
возвращает самый большой
```

- "←" обозначает назначение. Например, "*самый большой* ← *элемент*" означает, что значение *самого большого* изменяется на значение *элемента*.

- "**return**" завершает работу алгоритма и выводит следующее значение.

Алгоритм Евклида

В математике алгоритм Евклида или алгоритм Евклида - эффективный метод вычисления наибольшего общего делителя (GCD) двух целых чисел (numbers), наибольшего числа, которое делит их оба без остатка. Он назван в честь древнегреческого математика Евклида, который впервые описал его в своих *элементах* (ок. 300 г. до н.э.).[65] Это один из старейших широко используемых алгоритмов. Он может быть использован для приведения дробей к их простейшей форме и является частью многих других теоретико-числовых и криптографических вычислений.



Пример-диаграмма алгоритма Евклида из Т.Л. Хита (1908) с добавлением более подробной информации. Евклид не выходит за рамки третьего измерения и не приводит числовых примеров. Никомах приводит пример 49 и 21: "Я вычитаю меньшее из большего; остается 28; затем я снова вычитаю из этого те же 21 (поскольку это возможно); остается 7; Я вычитаю это из 21, остается 14; из чего я снова вычитаю 7 (поскольку это возможно); остается 7, но 7 нельзя вычесть из 7 ". Хит комментирует, что "Последняя фраза любопытна, но ее значение достаточно очевидно, как и значение фразы о завершении "на одно и то же число"".(Хит 1908:300).

Евклид (https://www.worldhistory.org/Euclid/) ставит задачу таким образом: "Учитывая два числа, не являющихся простыми по отношению друг к другу, найти их наибольшую общую меру". Он определяет "число как множество, состоящее из единиц": счетное число, положительное целое число, не включающее ноль. "Измерять" – это последовательно (q раз) помещать меньшую длину s вдоль большей длины l до тех пор, пока оставшаяся часть r не станет меньше более короткой длины s.[66] Современными словами, остаток $r = l - q \times s$, q - частное, или остаток r - это "модуль", целочисленно-дробная часть, оставшаяся после деления.[67]

Чтобы метод Евклида был успешным, начальные длины должны удовлетворять двум требованиям: (i) длины не должны быть равны нулю, И (ii) вычитание должно быть "правильным"; т. Е. Тест должен гарантировать, что меньшее из двух чисел вычитается из большего (или они могут быть равны, чтобы их вычитание дало ноль).

Первоначальное доказательство Евклида добавляет третье требование: две длины не должны быть простыми по отношению друг к другу. Евклид предусмотрел это, чтобы он мог построить доведение до абсурда доказательство того, что общая мера двух чисел на самом деле является *наибольшей*.[68] Хотя алгоритм Никомаха такой же, как у Евклида, когда числа являются простыми по отношению друг к другу, это дает число "1" для их общей меры. Итак, если быть точным, то на самом деле это алгоритм Никомаха.

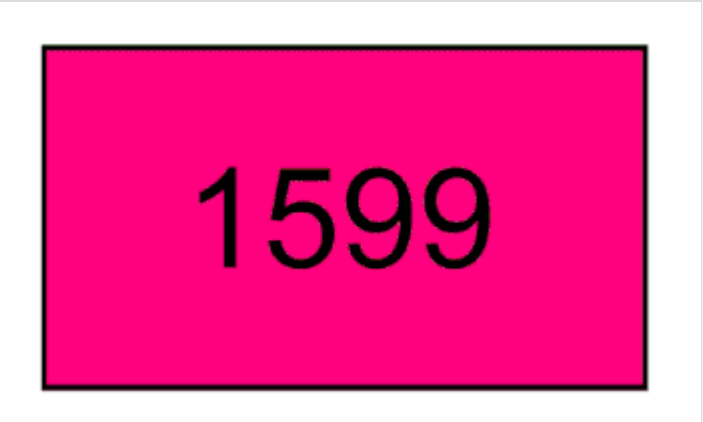
Компьютерный язык для алгоритма Евклида

Для выполнения алгоритма Евклида требуется всего несколько *типов* инструкций — некоторые логические тесты (условный переход), безусловный переход, присвоение (замена) и вычитание.

- *Местоположение* обозначается заглавными буквами, например S, A и т.д.
- Изменяющееся количество (number) в местоположении пишется строчными буквами и (обычно) ассоциируется с названием местоположения. Например, местоположение L в начале может содержать число *l* = 3009.

Неэлегантная программа для алгоритма Евклида

Следующий алгоритм сформулирован как четырехэтапная версия Кнута Евклида и Никомаха, но вместо использования деления для нахождения остатка он использует последовательные вычитания меньшей длины *s* из оставшейся длины *r* до тех пор, пока *r* не станет меньше *s*. Высокоуровневое описание, выделенное жирным шрифтом, адаптировано из Кнута 1973: 2-4:



Графическое выражение алгоритма Евклида для нахождения наибольшего общего делителя для 1599 и 650

```
1599 = 650×2 + 299
650 = 299×2 + 52
299 = 52×5 + 39
52 = 39×1 + 13
39 = 13×3 + 0
```

ВВОД:

```
1 [В двух местах L и S поместите числа l и s, которые
представляют две длины]: ВВЕДИТЕ L, S 2 [Инициализировать R:
сделать оставшуюся длину r равной начальной / initial / input
длине l]:
R ← L
```

E0: [Обеспечить *r* ≥ *s*.]

```
3 [Убедитесь, что меньшее из двух чисел находится в S, а
большее - в R]: ЕСЛИ R > S, ТО содержимое L является большим
числом, поэтому пропустите шаги обмена - 4, 5 и 6: ПЕРЕЙДИТЕ к
шагу 7, ИНАЧЕ поменяйте содержимое R и S. 4 L ← R (этот первый
шаг избыточен, но полезен для последующего обсуждения). 5 R ← S
6 S ← L
```

E1: [Найти остаток]: Пока оставшаяся длина *r* в R не станет меньше более короткой длины *s* в S, повторно вычтите число измерения *s* в S из оставшейся длины *r* в R.

```
7 ЕСЛИ S> R, ТО измерение завершено, поэтому ПЕРЕХОДИМ к 10 ЕЩЕ одному измерению, 8 R ← R − S 9
[Остаточный цикл]:
ПЕРЕХОД 7.
```

E2: [Остаток равен нулю?]: ЛИБО (i) последнее измерение было точным, остаток в R равен нулю, и программа может быть остановлена, ЛИБО (ii) алгоритм должен продолжаться: последнее измерение оставило остаток в R меньшим, чем число измерений в S.

```
10 ЕСЛИ R = 0, ТО
готово
ПЕРЕХОДИМ К шагу 15
ещё
ПЕРЕЙДИТЕ К шагу 11,
```

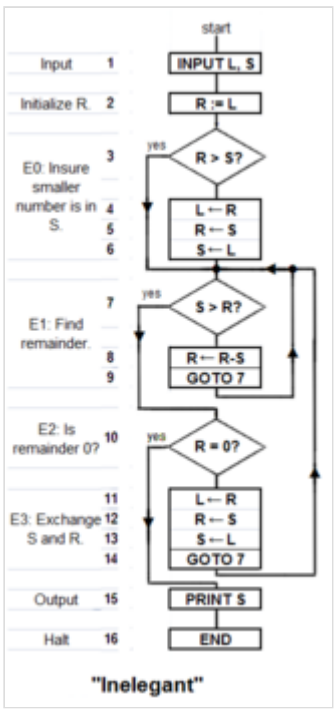
E3: [Взаимозаменить *s* и *r*]: Суть алгоритма Евклида. Используйте остаток *r* для измерения того, что ранее было меньшим числом *s*; L служит временным местоположением.

```
11 L ← R 12 R ← S 13 S ← L 14 [Повторить процесс измерения]:
ПЕРЕХОД 7
```

ВЫВОД:

```
15 [Готово. S содержит наибольший общий делитель]:
ВЫВЕДИТЕ S
```

Выполнено:



"Неэлегантный" - это перевод версии алгоритма Кнута с циклом остатка на основе вычитания, заменяющим его использование деления (или инструкции "модуля"). Получено из Knuth 1973: 2-4. В зависимости от двух чисел "Неэлегантный" может вычислить g.c.d. за меньшее количество шагов, чем "Элегантный".

```
16 ОСТАНОВКА, END, STOP.
```

Элегантная программа для алгоритма Евклида

[*уточнить*] Блок-схему "Elegant" можно найти в верхней части этой статьи. На (неструктурированном) языке Basic шаги пронумерованы, а инструкция `LET [] = []` представляет собой инструкцию по назначению, обозначаемую ←.

```
5 Алгоритм Рема Евклида для наибольшего общего делителя
6 PRINT "Введите два целых числа, больших 0"
10 ВВЕДИТЕ A,B
20 ЕСЛИ B=0 , ТО GOTO 80
30 ЕСЛИ A > B ТОГДА GOTO 60
40 ПУСТЬ B=B-A
50  ПЕРЕХОД 20
60 ПУСТЬ A=A-B
70 ПЕРЕХОД 20
80 ВЫВЕДИТЕ
90 КОНЦА
```

Как работает "Elegant": вместо внешнего "цикла Евклида" "Elegant" вычисляет остаток от деления с использованием по модулю и сдвигает переменные A и B на каждой итерации. Следующий алгоритм может использоваться с языками программирования из семейства C:

- Стандартная функция abs изменяет отрицательное целое число на положительное
- Когда входные данные A или B имеют значение 0, алгоритм останавливается и результат равен 0
- Если входные данные A больше входных данных B, алгоритм автоматически поменяет переменные A и B местами во время первой итерации по модулю
- Итерация (цикл выполнения while) начинается и останавливается только тогда, когда переменной B присваивается значение 0:
  - % вычисляет по модулю деления A и B, что уменьшает число (например: 23 = 3 × 6 + остаток 5)
  - A приравнивается к B
  - B приравнивается к результату по модулю
  - Итерация продолжается до тех пор, пока B больше 0
- Когда итерация останавливается, переменная A всегда содержит наибольший общий делитель

```
// Евклида алгоритм наибольший общий делитель
инт euclidAlgorithm (тип int a, внутр Б) {
    а = АБС(а);
    б = АБС(Б);
    если (А == 0 || Б == 0) {
       возвращают 0;
    }
    делать {
       инт Рес = А % Б;
       А = Б;
       Б = рез;
    } в то время как (Б > 0);
  вернуться с;
```

Тестирование алгоритмов Евклида

Выполняет ли алгоритм то, чего хочет его автор? Несколько тестовых примеров обычно дают некоторую уверенность в базовой функциональности. Но тестов недостаточно. Для тестовых примеров в одном источнике[69] используются 3009 и 884. Кнут предложил 40902, 24140. Другой интересный случай - это два относительно простых числа 14157 и 5950.

Но "исключительные случаи"[70] должны быть выявлены и протестированы. Будет ли "Неэлегантный" работать должным образом, когда R> S, S > R, R = S? То же самое для "Элегантного": B> A, A > B, A = B? (Да всем). Что происходит, когда одно число равно нулю, оба числа равны нулю? ("Неэлегантный"

вычисляется вечно во всех случаях; "Элегантный" вычисляется вечно, когда A = 0.) Что произойдет, если ввести *отрицательные* числа? Дробные числа? Если входные числа, то есть область действия функции, вычисляемая алгоритмом / программой, должны включать только положительные целые числа, включая ноль, то нулевые сбои указывают на то, что алгоритм (и программа, которая создает его экземпляр) является частичной функцией, а не полной функцией,,,. Заметным сбоем из-за исключений является отказ ракеты Ariane 5 рейса 501 (4 июня 1996 г.).

Доказательство корректности программы с помощью математической индукции: Кнут демонстрирует применение математической индукции к "расширенной" версии алгоритма Евклида и предлагает "общий метод, применимый для доказательства корректности любого алгоритма".[71] Таусворт предлагает, чтобы мерой сложности программы была продолжительность доказательства ее корректности.[72]

Измерение и совершенствование алгоритмов Евклида

Элегантность (компактность) против совершенства (скорости): Имея всего шесть основных инструкций, "Элегантный" является явным победителем по сравнению с "Неэлегантным" с тринадцатью инструкциями. Однако "Неэлегантный" *быстрее* (он останавливается за меньшее количество шагов). Анализ алгоритма[73] указывает, почему это так: "Элегантный" выполняет *два* условных теста в каждом цикле вычитания, тогда как "Неэлегантный" выполняет только один. Поскольку алгоритм (обычно) требует много циклов, *в среднем* тратится много времени на выполнение теста "B = 0?", который необходим только после вычисления остатка.

Можно ли улучшить алгоритмы?: Как только программист оценивает программу как "подходящую" и "эффективную", то есть она вычисляет функцию, задуманную ее автором, возникает вопрос, можно ли ее улучшить?

Компактность "Неэлегантного" можно улучшить, исключив пять шагов. Но Чайтин доказал, что сжатие алгоритма не может быть автоматизировано с помощью обобщенного алгоритма;[74] скорее, это можно сделать только эвристически; т. Е. Путем исчерпывающего поиска (примеры можно найти на Busy beaver), методом проб и ошибок, сообразительности, проницательности, применения индуктивного рассуждения и т.д. Обратите внимание, что шаги 4, 5 и 6 повторяются на шагах 11, 12 и 13. Сравнение с "Elegant" дает подсказку, что эти шаги вместе с шагами 2 и 3 можно исключить. Это сокращает количество основных инструкций с тринадцати до восьми, что делает его "более элегантным", чем "Elegant", на девять шагов.

Скорость выполнения "Elegant" можно улучшить, переместив тест "B = 0?" за пределы двух циклов вычитания. Это изменение требует добавления трех инструкций (B = 0?, A = 0?, GOTO). Теперь "Элегантный" быстрее вычисляет числа для примеров; всегда ли это так для любых заданных A, B и R, S потребует детального анализа.

Алгоритмический анализ

Часто важно знать, какой объем конкретного ресурса (например, времени или хранилища) теоретически требуется для данного алгоритма. Были разработаны методы анализа алгоритмов для получения таких количественных ответов (оценок); например, алгоритм, который суммирует элементы списка из n чисел, потребует времени $O(n)$, используя обозначение big O. В любое время алгоритму необходимо запоминать только два значения: сумму всех элементов на данный момент и его текущую позицию во входном списке. Следовательно, говорят, что для него требуется пространство, равное $O(1)$, если пространство, необходимое для хранения входных чисел, не подсчитано, или $O(n)$ если оно подсчитано.

Разные алгоритмы могут выполнять одну и ту же задачу с помощью разного набора инструкций за меньшее или большее время, пространство или "усилия", чем другие. Например, алгоритм бинарного поиска (по стоимости $O(\log n)$) превосходит последовательный поиск (по стоимости $O(n)$) при использовании для поиска по таблицам в отсортированных списках или массивах.

Формальный или эмпирический

Анализ и изучение алгоритмов - это дисциплина информатики, которая часто практикуется абстрактно, без использования конкретного языка программирования или реализации. В этом смысле анализ алгоритмов напоминает другие математические дисциплины в том смысле, что он фокусируется на базовых свойствах алгоритма, а не на специфике какой-либо конкретной реализации. Обычно для анализа используется псевдокод, поскольку это самое простое и общее представление. Однако, в конечном счете, большинство алгоритмов обычно реализуются на определенных аппаратных / программных платформах, и их алгоритмическая эффективность в конечном итоге проверяется с использованием реального кода. Для решения "одноразовой" задачи эффективность конкретного алгоритма может не иметь существенных последствий (если только n не является чрезвычайно большим), но для алгоритмов, предназначенных для быстрого интерактивного, коммерческого или научного использования с длительным сроком службы, это может иметь решающее значение. Масштабирование от малого n до большого n часто приводит к появлению неэффективных алгоритмов, которые в остальном безвредны.

Эмпирическое тестирование полезно, поскольку оно может выявить неожиданные взаимодействия, влияющие на производительность. Тесты могут использоваться для сравнения до / после потенциальных улучшений алгоритма после оптимизации программы. Однако эмпирические тесты не могут заменить формальный анализ, и их нетривиально выполнять справедливым образом.[75]

Эффективность выполнения

Чтобы проиллюстрировать потенциальные улучшения, возможные даже в хорошо зарекомендовавших себя алгоритмах, недавнее значительное новшество, касающееся алгоритмов БПФ (широко используемых в области обработки изображений), может сократить время обработки до 1000 раз для таких приложений, как медицинская визуализация.[76] В общем, повышение скорости зависит от особых свойств задачи, которые очень распространены в практических приложениях.[77] Ускорения такой величины позволяют вычислительным устройствам, широко использующим обработку изображений (таким как цифровые камеры и медицинское оборудование), потреблять меньше энергии.

Классификация

Существуют различные способы классификации алгоритмов, каждый из которых имеет свои достоинства.

По реализации

Один из способов классификации алгоритмов - это средства реализации.

Рекурсия

Рекурсивный алгоритм - это тот, который многократно вызывает (ссылается) сам на себя до тех пор, пока не будет выполнено определенное условие (также известное как условие завершения), что является методом, общим для функционального программирования. Итеративные алгоритмы используют повторяющиеся конструкции, такие как циклы, а иногда и дополнительные структуры данных, такие как стеки, для решения поставленных задач. Некоторые задачи естественным образом подходят для той или иной

```
int gcd(int A, int B) {
    if

        gcd(
        A -B,B);elseвозвращаетgcd
    (
        else )A,B-A);
}
```

реализации. Например, Ханойские башни хорошо понятны с помощью рекурсивной реализации. Каждая рекурсивная версия имеет эквивалентную (но, возможно, более или менее сложную) итеративную версию, и наоборот.

Рекурсивная C реализация алгоритма Евклида из приведенной блок-схемы

Последовательный, параллельный или распределенный

Алгоритмы обычно обсуждаются в предположении, что компьютеры выполняют по одной инструкции алгоритма за раз. Эти компьютеры иногда называют последовательными компьютерами. Алгоритм, разработанный для такой среды, называется последовательным алгоритмом, в отличие от параллельных алгоритмов или распределенных алгоритмов. Параллельные алгоритмы - это алгоритмы, использующие преимущества компьютерных архитектур, в которых несколько процессоров могут работать над задачей одновременно. Распределенные алгоритмы - это алгоритмы, в которых используется несколько машин, соединенных компьютерной сетью. Параллельные и распределенные алгоритмы разделяют задачу на более симметричные или асимметричные подзадачи и собирают результаты обратно воедино. Например, центральный процессор может быть примером параллельного алгоритма. Потребление ресурсов в таких алгоритмах - это не только процессорные циклы на каждом процессоре, но и накладные расходы на связь между процессорами. Некоторые алгоритмы сортировки можно эффективно распараллелить, но их накладные расходы на связь являются дорогостоящими. Итеративные алгоритмы, как правило, распараллеливаются, но некоторые задачи не имеют параллельных алгоритмов и называются по своей сути последовательными задачами.

Детерминированный или недетерминированный

Детерминированные алгоритмы решают проблему с точным решением на каждом шаге алгоритма, тогда как недетерминированные алгоритмы решают проблемы путем угадывания, хотя типичные догадки делаются более точными за счет использования эвристики.

Точный или приблизительный

Хотя многие алгоритмы достигают точного решения, алгоритмы аппроксимации стремятся к приближению, более близкому к истинному решению. Приближение может быть достигнуто либо с помощью детерминированной, либо случайной стратегии. Такие алгоритмы имеют практическую ценность для многих сложных задач. Одним из примеров приближенного алгоритма является задача о рюкзаке, где есть набор заданных предметов. Его цель - упаковать рюкзак, чтобы получить максимальную общую стоимость. Каждый предмет имеет определенный вес и определенную ценность. Общий вес, который можно переносить, не превышает некоторого фиксированного числа X. Таким образом, решение должно учитывать веса предметов, а также их стоимость.[78]

Квантовый алгоритм

Они работают на реалистичной модели квантовых вычислений. Этот термин обычно используется для тех алгоритмов, которые кажутся по своей сути квантовыми или используют какую-либо существенную особенность квантовых вычислений, такую как квантовая суперпозиция или квантовая запутанность.

По парадигме проектирования

Другой способ классификации алгоритмов - по методологии их проектирования или парадигме. Существует определенное количество парадигм, каждая из которых отличается от другой. Кроме того, каждая из этих категорий включает множество различных типов алгоритмов. Вот некоторые общие парадигмы:

Перебор или исчерпывающий поиск

Перебор - это метод решения проблем, который предполагает систематическое перебирание всех возможных вариантов до тех пор, пока не будет найдено оптимальное решение. Этот подход может занять очень много времени, поскольку требует перебора всех возможных комбинаций переменных. Однако он часто используется, когда другие методы недоступны или слишком сложны. Грубую силу можно использовать для решения множества задач, включая нахождение кратчайшего пути между двумя точками и взлом паролей.

Разделяй и властвуй

Алгоритм "разделяй и властвуй" многократно сводит экземпляр задачи к одному или нескольким меньшим экземплярам одной и той же задачи (обычно рекурсивно) до тех пор, пока экземпляры не станут достаточно маленькими, чтобы их можно было легко решить. Одним из таких примеров разделяй и властвуй является сортировка слиянием. Сортировка может быть выполнена для каждого сегмента данных после разделения данных на сегменты, а сортировка целых данных

может быть получена на этапе объединения путем объединения сегментов. Более простой вариант "разделяй и властвуй" называется *алгоритмом уменьшения и завоевания*, который решает идентичную подзадачу и использует решение этой подзадачи для решения более масштабной проблемы. Разделяй и властвуй делит проблему на несколько подзадач, поэтому этап завоевания более сложный, чем алгоритмы уменьшения и завоевания. Примером алгоритма уменьшения и завоевания является алгоритм бинарного поиска.

Поиск и перечисление

Многие задачи (например, игра в шахматы) можно смоделировать как задачи на графах. Алгоритм исследования графа определяет правила перемещения по графу и полезен для решения подобных задач. В эту категорию также входят алгоритмы поиска, перечисление ветвей и границ и обратный поиск.

Рандомизированный алгоритм

Такие алгоритмы делают некоторый выбор случайным образом (или псевдослучайно). Они могут быть очень полезны при поиске приближенных решений задач, в которых поиск точных решений может быть непрактичным (см. Эвристический метод ниже). Известно, что для некоторых из этих задач самые быстрые приближения должны включать некоторую случайность.[79] Могут ли рандомизированные алгоритмы с сложностью за полиномиальное время быть самыми быстрыми алгоритмами для некоторых задач - открытый вопрос, известный как проблема P против NP. Существует два больших класса таких алгоритмов:

1. Алгоритмы Монте-Карло с высокой вероятностью возвращают правильный ответ. Например. RP - это подкласс таких алгоритмов, которые выполняются за полиномиальное время.
2. Алгоритмы Лас-Вегаса всегда возвращают правильный ответ, но время их выполнения ограничено только вероятностно, например, ZPP.

Снижение сложности

Этот метод предполагает решение сложной задачи путем преобразования ее в более известную задачу, для которой у нас есть (надеюсь) асимптотически оптимальные алгоритмы. Цель состоит в том, чтобы найти сокращающий алгоритм, сложность которого не зависит от результирующего сокращенного алгоритма. Например, один алгоритм выбора для нахождения медианы в несортированном списке включает в себя сначала сортировку списка (дорогостоящая часть), а затем выделение среднего элемента в отсортированном списке (дешевая часть). Этот метод также известен как *трансформируй и властвуй*.

Обратное отслеживание

При таком подходе несколько решений создаются постепенно и отказываются от них, когда определяется, что они не могут привести к действительному полному решению.

Проблемы оптимизации

Для задач оптимизации существует более конкретная классификация алгоритмов; алгоритм для таких задач может подпадать под одну или несколько общих категорий, описанных выше, а также под одну из следующих:

Линейное программирование

При поиске оптимальных решений линейной функции, связанной линейными ограничениями на равенство и неравенство, ограничения задачи могут быть использованы непосредственно при получении оптимальных решений. Существуют алгоритмы, которые могут решить любую задачу из этой категории, такие как популярный симплексный алгоритм.[80] Проблемы, которые могут быть решены с помощью линейного программирования, включают задачу о максимальном потоке для ориентированных графов. Если задача дополнительно требует, чтобы одно или несколько неизвестных были целым числом, то она классифицируется в целочисленном программировании. Алгоритм линейного программирования может решить такую задачу, если можно доказать, что все ограничения для целых значений являются поверхностными, т. Е. Решения в любом случае удовлетворяют этим ограничениям. В общем случае используется специализированный алгоритм или алгоритм, который находит приближенные решения, в зависимости от сложности задачи.

Динамическое программирование

Когда задача показывает оптимальные подструктуры — это означает, что оптимальное решение проблемы может быть построено из оптимальных решений подзадач — и перекрывающиеся подзадачи, что означает, что одни и те же подзадачи используются для решения множества

различных экземпляров задачи, более быстрый подход, называемый *динамическим программированием*, позволяет избежать повторного вычисления решений, которые уже были вычислены. Например, алгоритм Флойда–Варшалла, кратчайший путь к цели из вершины взвешенного графа можно найти, используя кратчайший путь к цели из всех соседних вершин. Динамическое программирование и запоминание идут рука об руку. Основное различие между динамическим программированием и "разделяй и властвуй" заключается в том, что подзадачи более или менее независимы в "Разделяй и властвуй", тогда как в динамическом программировании подзадачи перекрываются. Разница между динамическим программированием и простой рекурсией заключается в кэшировании или запоминании рекурсивных вызовов. Когда подзадачи независимы и нет повторений, запоминание не помогает; следовательно, динамическое программирование не является решением всех сложных проблем. Используя запоминание или поддерживая таблица уже решенных подзадач, динамическое программирование сводит экспоненциальный характер многих задач к полиномиальной сложности.

Жадный метод

Жадный алгоритм похож на алгоритм динамического программирования в том смысле, что он работает путем изучения подструктур, в данном случае не проблемы, а данного решения. Такие алгоритмы начинаются с некоторого решения, которое может быть дано или было сконструировано каким-либо образом, и улучшают его, внося небольшие изменения. Для некоторых задач они могут найти оптимальное решение, в то время как для других они останавливаются на локальных оптимумах, то есть на решениях, которые не могут быть улучшены алгоритмом, но не являются оптимальными. Наиболее популярное использование жадных алгоритмов - для нахождения минимального остовного дерева, где с помощью этого метода возможно нахождение оптимального решения. Дерево Хаффмана, Крускал, Прим, Соллин - это жадные алгоритмы, которые могут решить эту задачу оптимизации.

Эвристический метод

В задачах оптимизации эвристические алгоритмы могут использоваться для нахождения решения, близкого к оптимальному, в случаях, когда поиск оптимального решения непрактичен. Эти алгоритмы работают, по мере продвижения становясь все ближе и ближе к оптимальному решению. В принципе, если запускать их бесконечно долго, они найдут оптимальное решение. Их заслуга в том, что они могут найти решение, очень близкое к оптимальному, за относительно короткое время. К таким алгоритмам относятся локальный поиск, запретный поиск, имитированный отжиг и генетические алгоритмы. Некоторые из них, такие как имитированный отжиг, являются недетерминированными алгоритмами, в то время как другие, такие как запретный поиск, являются детерминированными. Когда известна оценка погрешности неоптимального решения, алгоритм далее классифицируется как алгоритм аппроксимации.

По области обучения

Каждая область науки имеет свои проблемы и нуждается в эффективных алгоритмах. Связанные проблемы в одной области часто изучаются вместе. Некоторыми примерами классов являются алгоритмы поиска, алгоритмы сортировки, алгоритмы слияния, числовые алгоритмы, графовые алгоритмы, строковые алгоритмы, вычислительные геометрические алгоритмы, комбинаторные алгоритмы, медицинские алгоритмы, машинное обучение, криптография, алгоритмы сжатия данных и методы синтаксического анализа.

Поля имеют тенденцию накладываться друг на друга, и достижения алгоритма в одной области могут улучшить результаты в других, иногда совершенно не связанных, областях. Например, динамическое программирование было изобретено для оптимизации потребления ресурсов в промышленности, но в настоящее время используется при решении широкого круга задач во многих областях.

По сложности

Алгоритмы можно классифицировать по количеству времени, которое им требуется для выполнения, по сравнению с размером их входных данных:

- Постоянное время: если время, необходимое алгоритму, одинаково, независимо от размера входных данных. Например. доступ к элементу массива.

- Логарифмическое время: если время является логарифмической функцией размера входных данных. Например. алгоритм бинарного поиска.
- Линейное время: если время пропорционально размеру входных данных. Например. обходу списка.
- Полиномиальное время: если время является степенью размера входных данных. Например. алгоритм пузырьковой сортировки имеет квадратичную временную сложность.
- Экспоненциальное время: если время является экспоненциальной функцией размера входных данных. Например. Поиск методом перебора.

В некоторых задачах может быть несколько алгоритмов разной сложности, в то время как в других задачах может не быть алгоритмов или неизвестных эффективных алгоритмов. Существуют также сопоставления некоторых задач с другими задачами. В связи с этим было сочтено более подходящим классифицировать сами задачи, а не алгоритмы, по классам эквивалентности на основе сложности наилучших возможных алгоритмов для них.

Непрерывные алгоритмы

Прилагательное "непрерывный" применительно к слову "алгоритм" может означать:

- Алгоритм, работающий с данными, представляющими непрерывные величины, даже если эти данные представлены дискретными приближениями — такие алгоритмы изучаются в численном анализе; или
- Алгоритм в форме дифференциального уравнения, который непрерывно обрабатывает данные, запущенный на аналоговом компьютере.[81]

Алгоритм = Логика + Управление

В логическом программировании алгоритмы рассматриваются как имеющие "логический компонент, который определяет знания, которые будут использоваться при решении задач, и управляющий компонент, который определяет стратегии решения проблем, с помощью которых используются эти знания".[82]

Алгоритм Евклида иллюстрирует этот взгляд на алгоритм.[83][84] Вот представление логического программирования, использующее:- для представления "if" и отношение gcd (A, B, C) для представления функции gcd (A, B) = C:

```
НОД(A, A, A).
нод(A, B, C) :- A > B, нод(A-B, B, C).
нод(A, B, ). ) :- B > A, gcd(A, B-A, C).
```

На языке логического программирования Ciao отношение gcd может быть представлено непосредственно в функциональной нотации:

```
НОД(A, A) := A.
нод(A, B) := нод(A-B, B) :- A > B.
нод(A, B) := нод( A, B-A) :- B > A.
```

Реализация Ciao преобразует функциональную нотацию в реляционное представление в Prolog, извлекая встроенные вычитания, A-B и B-A, как отдельные условия:

```
нод(A, A, A).
нод(A, B, C):- A > B, A' - это A-B, нод(A', B, C).
нод(A, B, C):- B > A, B' - это B-A, нод(A, B, C).
```

Полученная программа имеет чисто логическое (и "декларативное") прочтение как рекурсивное (или индуктивное) определение, которое не зависит от того, как логика используется для решения задач:

```
НОД из A и A равен A.
НОД для A и B равен C, если A > B и A' равно A-B, а нод для A' и B равно C.
НОД для A и B равен C, если B > A и B' равно B-A, а НОД для A и B' равно C.
```

Различные стратегии решения проблем превращают логику в разные алгоритмы. Теоретически, учитывая пару целых чисел A и B, для генерации всех экземпляров отношения gcd можно использовать прямое (или "восходящее") рассуждение, заканчивающееся, когда генерируется желаемый gcd из A и B. Конечно, дальновидные рассуждения в данном случае совершенно бесполезны. Но в других случаях, таких как определение последовательности Фибоначчи[82] и журнала данных, опережающее мышление может быть эффективной стратегией решения проблем. (Смотрите, например, логическую программу для вычисления чисел Фибоначчи в Algorithm = Logic + Control).

В отличие от неэффективности прямого рассуждения в этом примере, обратное (или "нисходящее") рассуждение с использованием разрешения SLD превращает логику в алгоритм Евклида:

```
Чтобы найти gcd C двух заданных чисел A и B:
Если A = B, то C = A.
Если A > B, то пусть A' = A-B и найдите gcd из A' и B, который равен C.
Если B > A, то пусть B' = B-A и найдите gcd из A и B', который равен C.
```

Одним из преимуществ представления алгоритма в виде логического программирования является то, что его чисто логическое прочтение облегчает проверку правильности алгоритма относительно стандартного нерекурсивного определения gcd.[83] Вот стандартное определение, написанное на Prolog:

```
НОД(A, B, C) : - делит(C, A), делит(C, B),
    forall((делит(D, A), делит(D, B)), D  =< C).

 делит(C, число) :-
   между(1, Число, C), 0 -это число mod C.
```

Это определение, которое является спецификацией алгоритма Евклида, также выполняется в Prolog: Обратное рассуждение рассматривает спецификацию как алгоритм грубой силы, который перебирает все целые числа C от 1 до A, проверяя, делит ли C как A, так и B, а затем для каждого такого C снова перебирает все целые числа D от 1 до A, пока не найдет C такое, что C больше или равно всем D, которые также делят как A, так и B. Хотя этот алгоритм безнадежно неэффективен, он показывает, что формальные спецификации часто могут быть написаны на формируются логические программы, и они могут быть выполнены с помощью Prolog, чтобы проверить, что они правильно представляют неформальные требования.

Юридические вопросы

Алгоритмы сами по себе обычно не подлежат патентованию. В Соединенных Штатах претензия, состоящая исключительно из простых манипуляций с абстрактными понятиями, числами или сигналами, не является "процессами" (USPTO 2006), поэтому алгоритмы не подлежат патентованию (как в деле Готтшалк против Бенсона). Однако практическое применение алгоритмов иногда патентоспособно. Например, в деле Даймонд против Дира применение простого алгоритма с обратной связью для облегчения отверждения синтетического каучука было признано патентоспособным. Патентование программного обеспечения является спорным,[85] и существуют критикуемые патенты, касающиеся алгоритмов, особенно алгоритмов сжатия данных, таких как патент LZW от Unisys.

Кроме того, некоторые криптографические алгоритмы имеют ограничения на экспорт (см. Экспорт криптографии).

История: Развитие понятия "алгоритм"

Древний Ближний Восток

Самые ранние свидетельства существования алгоритмов найдены в вавилонской математике древней Месопотамии (современный Ирак). На шумерской глиняной табличке, найденной в Шуруппаке недалеко от Багдада и датированной ок. 2500 г. до н.э., описан самый ранний алгоритм деления.[11] Во времена династии Хаммурапи ок. 1800 – ок. 1600 г. до н.э. На вавилонских глиняных табличках были описаны алгоритмы вычисления формул.[86] Алгоритмы также использовались в вавилонской астрономии. Вавилонские глиняные таблички описывают и используют алгоритмические процедуры для вычисления времени и места значительных астрономических событий.[87]

Алгоритмы для арифметики также встречаются в древней египетской математике, восходящей к Римскому математическому папирусу ок. 1550 г. до н.э..[11] Алгоритмы позже использовались в древней эллинистической математике. Двумя примерами являются сито Эратосфена, которое было описано в *Введении в арифметику* Никомаха,[88][14]: гл. 9.2 и алгоритм Евклида, который был впервые описан в *Элементах Евклида* (ок. 300 г. до н.э.).[14]: гл. 9.1

Дискретные и различимые символы

Подсчетные знаки: Чтобы отслеживать свои стада, мешки с зерном и деньги, древние использовали подсчет: накапливали камни или метки, нацарапанные на палочках, или делали отдельные символы на глине. Благодаря вавилонскому и египетскому использованию знаков и символов, в конечном итоге появились римские цифры и счеты (Дилсон, стр. 16-41). Метки подсчета занимают видное место в арифметике унарной системы счисления, используемой в машине Тьюринга и в вычислениях после машины Тьюринга.

Манипулирование символами как "заменителями" чисел: алгебра

Мухаммад ибн Муса аль-Хорезми, персидский математик, написал *Аль-Джабр* в 9 веке. Термины "алгоритмизм" и "алгоритм" происходят от имени аль-Хоризми, в то время как термин "алгебра" взят из книги *Аль-Джабр*. В Европе слово "алгоритм" первоначально использовалось для обозначения наборов правил и методов, используемых Аль-Хорезми для решения алгебраических уравнений, а затем было обобщено для обозначения любого набора правил или методов.[89] В конечном итоге это привело к появлению у Лейбница понятия математического уравнителя (ок. 1680):

> На добрых полтора столетия опередив свое время, Лейбниц предложил алгебру логики, алгебру, которая определяла бы правила манипулирования логическими понятиями так, как обычная алгебра определяет правила манипулирования числами.[90]

Криптографические алгоритмы

Первый криптографический алгоритм расшифровки зашифрованного кода был разработан Аль-Кинди, арабским математиком 9 века, в *рукописи О расшифровке криптографических сообщений*. Он дал первое описание криптоанализа с помощью частотного анализа, самого раннего алгоритма взлома кода.[15]

Механические устройства с дискретными состояниями

Часы: Болтер называет изобретение часов с приводом от веса "Ключевым изобретением [Европы в Средние века]", в частности, спускового механизма verge[91], который обеспечивает нам тиканье механических часов. "Точный автомат"[92] сразу привел к появлению "механических автоматов", начиная с 13 века, и, наконец, к "вычислительным машинам" — разностному движку и аналитическим машинам Чарльза Бэббиджа и графини Ады Лавлейс середины 19 века.[93] Лавлейс приписывают первое создание алгоритма, предназначенного для обработки на компьютере — аналитической машины Бэббиджа, первого устройства, которое считается настоящим компьютером с Тьюрингом, а не просто калькулятором, — и в результате ее иногда называют "первым программистом в истории", хотя полная реализация второго устройства Бэббиджа будет реализована только спустя десятилетия после ее жизни.

Логические машины 1870 – Стэнли Джевонс "логические счеты" и "логическая машина": Техническая проблема заключалась в сокращении булевых уравнений при представлении в форме, подобной той, что сейчас известна как карты Карно. Джевонс (1880) сначала описывает простые "счеты" из "деревянных пластинок, снабженных булавками, сконструированных таким образом, что любая часть или класс [логических] комбинаций могут быть выбраны механически ... Однако совсем недавно я свел систему к полностью механической форме и, таким образом, воплотил весь косвенный процесс вывода в то, что можно назвать *логической машиной*" Его машина была оснащена "определенными подвижными деревянными стержнями" и "у подножия находятся 21 клавиша, как у пианино [и т.д.] ...". С помощью этой машины он мог анализировать "силлогизм или любой другой простой логический аргумент".[94]

Эту машину он продемонстрировал в 1870 году перед членами Королевского общества.[95] Однако другой логик Джон Венн в своей работе 1881 года " *Символическая логика "* критически оценил эти усилия: "Я сам не очень высоко оцениваю интерес или важность того, что иногда называют логическими машинами ... мне не кажется, что какие-либо известные в настоящее время или, вероятно, обнаруженные изобретения действительно заслуживают названия "логических машин"; подробнее см. в "Характеристиках алгоритмов". Но чтобы не отставать, он тоже представил "план, несколько аналогичный, как я понимаю, плану профессора. *Счеты* Джевона ... [И] [а] коэффициент усиления, соответствующий логической машине профессора. Можно описать следующее устройство. Я предпочитаю называть это просто машиной с логическими диаграммами ... но я полагаю, что он мог бы очень полно выполнять все, что можно рационально ожидать от любой логической машины".[96]

Жаккардовый ткацкий станок, перфокарты Холлерита, телеграфия и телефония – электромеханическое реле: Белл и Ньюэлл (1971) указывают, что Жаккардовый ткацкий станок (1801), предшественник карт Холлерита (перфокарт, 1887), и "технологии телефонной коммутации" были корнями дерева, ведущего к разработке первых компьютеров.[97] К середине 19 века телеграф, предшественник телефона, использовался во всем мире, его дискретная и различимая кодировка букв в виде "точек и тире" была обычным звуком. К концу 19 века использовалась бегущая строка (ок. 1870-х гг.), как и карточки Холлерита при переписи населения США 1890 года. Затем появился телетайп (ок. 1910) с использованием на перфоленте кода Бодо на ленте.

Сети телефонной коммутации с электромеханическими реле (изобретены в 1835 году) стояли за работой Джорджа Стибица (1937), изобретателя цифрового суммирующего устройства. Работая в Bell Laboratories, он наблюдал "обременительное" использование механических калькуляторов с шестеренками. "Однажды вечером 1937 года он пошел домой, намереваясь проверить свою идею... Когда переделка была закончена, Стибиц сконструировал двоичное суммирующее устройство".[98]

Математик Мартин Дэвис отмечает особую важность электромеханического реле (с его двумя "бинарными состояниями" - *разомкнутым* и *замкнутым*).:

Только с разработкой, начавшейся в 1930-х годах, электромеханических калькуляторов, использующих электрические реле, были построены машины, имеющие масштабы, которые представлял Бэббидж ".[99]

Математика в течение 19 века вплоть до середины 20 века

Символы и правила: Математика Джорджа Буля (1847, 1854), Готтлоба Фреге (1879) и Джузеппе Пеано (1888-1889) быстро свела арифметику к последовательности символов, с которыми манипулируют по правилам. Книга Пеано "*Принципы арифметики, представленные новым методом*" (1888) была "первой попыткой аксиоматизации математики на символическом языке".[100]

Но Хейенорт отдает Фреге (1879) должное: Фреге - "возможно, самая важная отдельная работа, когда-либо написанная по логике. ... в которой мы видим " "язык формул", то есть *характерный язык*, язык, написанный специальными символами "для чистой мысли", то есть свободный от риторических прикрас ... построенный из определенных символов, которыми манипулируют в соответствии с определенными правилами".[101] Работа Фреге была дополнительно упрощена и дополнена Альфредом Нортом Уайтхедом и Бертран Рассел в своих "Принципах математики" (1910-1913).

Парадоксы: В то же время в литературе появился ряд тревожных парадоксов, в частности, парадокс Бурали-Форти (1897), парадокс Рассела (1902-03) и парадокс Ричарда.[102] Полученные в результате соображения привели к работе Курта Геделя (1931) — он специально цитирует парадокс лжеца — который полностью сводит правила рекурсии к числам.

Эффективная вычислимость: В попытке решить задачу Entscheidungsproblem, точно определенную Гильбертом в 1928 году, математики впервые приступили к определению того, что подразумевается под "эффективным методом", или "эффективным вычислением", или "эффективной вычислимостью" (т. Е. Вычислением, которое было бы успешным). В быстрой последовательности появилось следующее: Алонзо Черч, Стивен Клини и Дж.Б. Россер λ-исчисление[103] отточенное определение "общей рекурсии" из работы Геделя, основанное на предложениях Жака Эрбрана (ср. лекции Геделя в Принстоне 1934 года) и последующих упрощениях Клини.[104] Доказательство Черча[105] того, что Entscheidungsproblem была неразрешимой, Эмиль Пост определяет эффективную вычисляемость как работника, бездумно следующего списку инструкций по перемещению влево или вправо через последовательность комнат и, находясь там, либо помечающего, либо стирающего бумагу, либо наблюдающего за бумагой и принимающего решение "да-нет" относительно следующей инструкции.[106] Доказательство Аланом Тьюрингом того, что проблема Entscheidungsproblem неразрешима с помощью его "а- [автоматической] машины"[107] - по сути, почти идентично "формулировке" Поста, Дж. Баркли Россер определение "эффективного метода" в терминах "машины".[108] Предложение Клина о предшественнике "Тезиса Черча", который он назвал "Тезис I",[109] а несколько лет спустя Клин переименовал свой тезис в "Тезис Черча"[110] и предложил "Тезис Тьюринга".[111]

Эмиль Пост (1936) и Алан Тьюринг (1936-37, 1939)

Эмиль Пост (1936) описал действия "компьютера" (человека) следующим образом:

> "... задействованы две концепции: *символьное пространство*, в котором должна выполняться работа, ведущая от проблемы к ответу, и фиксированный неизменяемый *набор направлений*.

Его пространство символов было бы

> "двусторонняя бесконечная последовательность пробелов или блоков ... Решатель задач или рабочий должен перемещаться и работать в этом пространстве символов, будучи способным находиться и работать только в одном блоке одновременно. ... поле должно допускать только два возможных условия, т. Е. Быть пустым или без опознавательных знаков и иметь в себе одну метку, скажем, вертикальный штрих.

"Необходимо выделить одно поле и назвать его начальной точкой. ... конкретная задача должна быть задана в символической форме конечным числом полей [т. Е. входных данных], отмеченных обводкой. Аналогично, ответ [т. Е. выходные данные] должен быть задан в символической форме с помощью такой конфигурации отмеченных полей...

"Набор указаний, применимых к общей задаче, устанавливает детерминированный процесс применительно к каждой конкретной задаче. Этот процесс завершается только тогда, когда дело доходит до направления типа (C) [т.Е. остановки] ".[112] Подробнее на машине пост–Тьюринга

РаботаАлана Тьюринга[113] предшествовала работе Стибица (1937); неизвестно, знал ли Стибиц о работе Тьюринга. Биограф Тьюринга полагал, что использование Тьюрингом модели, похожей на пишущую машинку, проистекало из юношеского интереса: "Алан в детстве мечтал изобрести пишущие машинки; у миссис Тьюринг была пишущая машинка, и он вполне мог начать с вопроса, что подразумевалось под названием пишущей машинки "механическая"" ".[114] Учитывая распространенность в то время азбуки Морзе, телеграфии, тикерной ленты и телетайпов, вполне возможно, что все это оказало влияние на Тьюринга в течение его жизни. его молодость.

Статуя Алана Тьюринга в парке Блетчли

Тьюринг — его модель вычислений теперь называется машиной Тьюринга — начинает, как и Пост, с анализа человеческого компьютера, который он сводит к простому набору базовых движений и "состояний разума". Но он идет еще дальше и создает машину как модель вычисления чисел.[115]

"Вычисления обычно выполняются путем написания определенных символов на бумаге. Мы можем предположить, что эта бумага разделена на квадраты, как детская книжка по арифметике ... Тогда я предполагаю, что вычисления выполняются на одномерной бумаге, то есть на ленте, разделенной на квадраты. Я также предположу, что количество символов, которые могут быть напечатаны, конечно...

"Поведение компьютера в любой момент определяется символами, которые он наблюдает, и его "состоянием ума" в этот момент. Мы можем предположить, что существует граница B для количества символов или квадратов, которые компьютер может наблюдать в один момент. Если он хочет наблюдать больше, он должен использовать последовательные наблюдения. Мы также предположим, что количество состояний сознания, которые необходимо принимать во внимание, конечно...

"Давайте представим, что операции, выполняемые компьютером, должны быть разделены на "простые операции", которые настолько элементарны, что нелегко представить их дальнейшее разделение".[116]

Редукция Тьюринга дает следующее:

"Следовательно, простые операции должны включать:

"(a) изменения символа на одном из наблюдаемых квадратов
"(b) изменение одного из наблюдаемых квадратов на другой квадрат в пределах L квадратов одного из ранее наблюдавшихся квадратов.

"Возможно, некоторые из этих изменений обязательно вызывают изменение состояния ума. Следовательно, наиболее общую операцию следует рассматривать как одну из следующих:

"(A) возможное изменение (a) символа вместе с возможным изменением состояния ума.
"(B) возможное изменение (b) наблюдаемых квадратов вместе с возможным изменением состояния ума"

"Теперь мы можем сконструировать машину для выполнения работы этого компьютера".[116]

Несколько лет спустя Тьюринг расширил свой анализ (тезис, определение) таким убедительным его выражением:

"Говорят, что функция "эффективно вычисляема", если ее значения могут быть найдены с помощью какого-либо чисто механического процесса. Хотя интуитивно понять эту идею довольно легко, тем не менее желательно иметь какое-то более определенное, математически выражаемое определение ... [он обсуждает историю определения в значительной степени так, как представлено выше, в отношении Геделя, Хербранда, Клини, Черча, Тьюринга и Поста] ... Мы можем понимать это утверждение буквально, понимая под чисто механическим процессом тот, который мог бы выполняться машиной. Можно дать математическое описание в определенной нормальной форме структур этих машин. Развитие этих идей приводит к авторскому определению вычислимой функции и отождествлению вычислимости с эффективной вычислимостью...

> "† Мы будем использовать выражение "вычислимая функция" для обозначения функции, вычисляемой машиной, и мы позволяем "эффективно вычислимой" относиться к интуитивной идее без особой идентификации с каким-либо из этих определений".[117]

Дж. Б. Россер (1939) и С. К. Клини (1943)

Дж. Баркли Россер определил "эффективный [математический] метод" следующим образом (добавлен курсив):

> ""Эффективный метод" используется здесь в довольно особом смысле метода, каждый шаг которого точно определен и который наверняка даст ответ за конечное число шагов. С этим особым значением на сегодняшний день были даны три разных точных определения. [его сноска № 5; см. Обсуждение непосредственно ниже]. Самый простой из них (благодаря Post и Turing), по сути, гласит, что *существует эффективный метод решения определенных наборов задач, если можно построить машину, которая затем решит любую проблему из набора без вмешательства человека, за исключением вставки вопроса и (позже) чтения ответа*.. Все три определения эквивалентны, поэтому не имеет значения, какое из них используется. Более того, тот факт, что все три эквивалентны, является очень сильным аргументом в пользу правильности любого из них. " (Россер 1939:225-226)

Сноска Россера № 5 ссылается на работу (1) Черча и Клина и их определение λ-определимости, в частности, использование Черчем этого в его *Неразрешимой проблеме теории элементарных чисел* (1936); (2) Хербранда и Геделя и их использование рекурсии, в частности, использование Геделем в его знаменитой статье *О формально неразрешимых предложениях Principia Mathematica и связанных системах I* (1931); и (3) Поста (1936) и Тьюринга (1936-37) в его знаменитой статье. их механизм - модели вычислений.

Стивен К. Клини определил как свой ныне знаменитый "Тезис I", известный как тезис Черча-Тьюринга. Но он сделал это в следующем контексте (выделено жирным шрифтом в оригинале):

> "12. *Алгоритмические теории*... При создании полной алгоритмической теории мы описываем процедуру, выполнимую для каждого набора значений независимых переменных, которая обязательно завершается и таким образом, чтобы по результату мы могли прочитать определенный ответ "да" или "нет" на вопрос: "истинно ли значение предиката?"" (Клини 1943: 273)

История после 1950 года

Ряд усилий был направлен на дальнейшее уточнение определения "алгоритма", и работа продолжается из-за проблем, связанных, в частности, с основами математики (особенно с тезисом Черча–Тьюринга) и философией разума (особенно с аргументами об искусственном интеллекте). Подробнее см. в разделе Характеристики алгоритма.

Смотрите также

- Абстрактная машина
- АЛГОЛ

- Разработка алгоритмов
- Характеристики алгоритма
- Алгоритмических предубеждений
- Состав алгоритма
- Алгоритмические сущности
- Алгоритмический синтез
- Алгоритмический метод
- Алгоритмическая топология
- Мусор входит, мусор выходит
- *Введение в алгоритмы* (учебник)
- Правительство алгоритм
- Список алгоритмов
- Список общих тем алгоритма
- Регулирование алгоритмов
- Теория вычислений
  - Теория вычислимости
  - Теория вычислительной сложности
- Вычислительная математика

Примечания

1. "Определение АЛГОРИТМА" (https://www.merriam-webster.com/dictionary/algorithm). *Онлайн-словарь Merriam-Webster*. Архивировано (https://web.archive.org/web/20200214074446/https://www.merriam-webster.com/dictionary/algorithm) с оригинала 14 февраля 2020 года. Проверено 14 ноября 2019 года.
2. Блэр, Энн, Дугид, Пол, Гинг, Аня-Сильвия и Графтон, Энтони. Информация: Исторический компаньон, Принстон: Издательство Принстонского университета, 2021. стр. 247
3. Дэвид А. Гроссман, Офир Фридер, *Поиск информации: алгоритмы и эвристика*, 2-е издание, 2004, ISBN 1402030045
4. "Любой классический математический алгоритм, например, может быть описан конечным числом английских слов" (Rogers 1987:2).
5. Четко определено относительно агента, выполняющего алгоритм: "Существует вычислительный агент, обычно человек, который может реагировать на инструкции и выполнять вычисления" (Rogers 1987: 2).
6. "алгоритм - это процедура вычисления *функции* (относительно некоторых выбранных обозначений для целых чисел) ... это ограничение (числовых функций) не приводит к потере общности", (Роджерс 1987: 1).
7. "Алгоритм имеет ноль или более входных данных, т.Е. Величин, которые задаются ему изначально перед запуском алгоритма" (Knuth 1973: 5).
8. "Процедура, обладающая всеми характеристиками алгоритма, за исключением того, что ей, возможно, не хватает конечности, может быть названа "вычислительным методом" (Knuth 1973: 5).
9. "Алгоритм имеет один или несколько выходных данных, то есть величин, которые имеют определенное отношение к входным данным" (Knuth 1973: 5).
10. Вопрос о том, является ли процесс со случайными внутренними процессами (не включая входные данные) алгоритмом, является спорным. Роджерс полагает, что: "вычисление выполняется дискретно, поэтапно, без использования непрерывных методов или аналоговых устройств ... выполняется детерминированно, без обращения к случайным методам или устройствам, например, игральным костям" (Rogers 1987: 2).
11. Шабер, Жан-Люк (2012). *История алгоритмов: от камешка до микрочипа*. Springer Science & Business Media. стр. 7-8. ISBN 9783642181924.
12. Sriram, M. S. (2005). "Algorithms in Indian Mathematics" (https://books.google.com/books?id=qfJdDwAAQBAJ&pg=PA153). In Emch, Gerard G.; Sridharan, R.; Srinivas, M. D. (eds.). *Contributions to the History of Indian Mathematics*. Springer. p. 153. ISBN 978-93-86279-25-5.
13. Hayashi, T. (2023, January 1). Brahmagupta (https://www.britannica.com/biography/Brahmagupta). Encyclopedia Britannica.
14. Кук, Роджер Л. (2005). *История математики: краткий курс*. Джон Уайли и сыновья. ISBN 978-1-118-46029-0.
15. Дули, Джон Ф. (2013). *Краткая история криптологии и криптографических алгоритмов*. Springer Science & Business Media. стр. 12-3. ISBN 9783319016283.

16. Бернетт, Чарльз (2017). "Арабские цифры" (https://books.google.com/books?id=zTQrDwAAQBAJ&pg=PA39). В книге Томаса Ф. Глика (ред.). *Возрождение Ратледжа: средневековая наука, технология и медицина (2006): Энциклопедия*. Тейлор и Фрэнсис. стр. 39 (https://web.archive.org/web/20230328222536/https://books.google.com/books?id=zTQrDwAAQBAJ&pg=PA39). ISBN 978-1-351-67617-5. Архивировано (https://web.archive.org/web/20230328222536/https://books.google.com/books?id=zTQrDwAAQBAJ&pg=PA39) с оригинала 28 марта 2023 года. Проверено 5 мая 2019 года.
17. "алгоритмизм" (https://www.oed.com/search/dictionary/?q=algorism). *Оксфордский словарь английского языка* (онлайн-издание). Издательство Оксфордского университета. (Требуется подписка или членство в участвующем учреждении (https://www.oed.com/public/login/loggingin#withyourlibrary).)
18. Brezina, Corona (2006). *Аль-Хорезми: изобретатель алгебры* (https://books.google.com/books?id=955jPgAACAAJ). Издательская группа Розена. ISBN 978-1-4042-0513-0.
19. "Абу Джафар Мухаммад ибн Муса аль-Хорезми" (http://members.peak.org/~jeremy/calculators/alKwarizmi.html). *members.peak.org*. Архивировано (https://web.archive.org/web/20190821232118/http://members.peak.org/~jeremy/calculators/alKwarizmi.html) с оригинала 21 августа 2019 года. Проверено 14 ноября 2019 года.
20. Mehri, Bahman (2017). "От Аль-Хорезми к алгоритму". *Олимпиады по информатике*. **11** (2): 71-74. doi:10.15388/ioi.2017.special.11 (https://doi.org/10.15388%2Fioi.2017.special.11).
21. "алгоритмический" (https://www.thefreedictionary.com/algorismic). *Бесплатный словарь*. Архивировано (https://web.archive.org/web/20191221200124/https://www.thefreedictionary.com/algorismic) с оригинала 21 декабря 2019 года. Проверено 14 ноября 2019 года.
22. Блаунт, Томас (1656). *Глоссография или словарь ...* (https://leme.library.utoronto.ca/lexicons/478/details#details) Лондон: Хамфри Мозли и Джордж Собридж.
23. Филлипс, Эдвард (1658). *Новый мир английских слов, или общий словарь, содержащий толкования таких трудных слов, которые заимствованы из других языков...* (https://quod.lib.umich.edu/e/eebo/A54746.0001.001/1:10.1.9?rgn=div3;view=fulltext)
24. Филлипс, Эдвард; Керси, Джон (1706). *Новый мир слов: или Универсальный словарь английского языка. Содержащий описание оригинального или собственного смысла и различные значения всех трудных слов, заимствованных из других языков ... Вместе с кратким и понятным объяснением всех терминов, относящихся к любому из искусств и наук ... к этому добавляется толкование имен собственных .* (https://babel.hathitrust.org/cgi/pt?id=nyp.33433002977779&view=1up&seq=35&size=125&q1=algorithm). Напечатано для Дж. Филлипса и др.
25. Феннинг, Дэниел (1751). *Спутник юного алгебраиста, или новое и простое руководство по алгебре; представлено the doctrine of vulgar fractions: предназначено для использования в школах ... иллюстрировано множеством числовых и буквенных примеров ...* (https://babel.hathitrust.org/cgi/pt?id=mdp.39015063909108&view=1up&seq=17&q1=algorithm) Напечатано для G. Keith & J. Robinson. p. xi (https://babel.hathitrust.org/cgi/pt?id=mdp.39015063909108&view=1up&seq=17&q1=algorithm).
26. *Электрическое обозрение 1811-07: Том 7* (https://archive.org/details/sim_eclectic-review_1811-07_7/page/586). Издательство Open Court Publishing Co. Июль 1811. стр. [1] (https://archive.org/details/sim_eclectic-review_1811-07_7). "Тем не менее, ему нужен новый алгоритм, комплексный метод, с помощью которого теоремы могут быть установлены без двусмысленности и околичностей, [...]"
27. "algorithm" (https://www.oed.com/search/dictionary/?q=algorithm). *Oxford English Dictionary* (Online ed.). Oxford University Press. (Subscription or participating institution membership (https://www.oed.com/public/login/loggingin#withyourlibrary) required.)
28. Already 1684, in Nova Methodus pro Maximis et Minimis, Leibnitz used the Latin term "algorithmo".
29. Kleene 1943 in Davis 1965:274
30. Rosser 1939 in Davis 1965:225
31. Stone 1973:4
32. Simanowski, Roberto (2018). *The Death Algorithm and Other Digital Dilemmas* (https://books.google.com/books?id=RJV5DwAAQBAJ). Untimely Meditations. Vol. 14. Translated by Chase, Jefferson. Cambridge, Massachusetts: MIT Press. p. 147. ISBN 9780262536370. Archived (https://web.archive.org/web/20191222120705/https://books.google.com/books?id=RJV5DwAAQBAJ) from the original on December 22, 2019. Retrieved May 27, 2019. "[...] the next level of abstraction of central bureaucracy: globally operating algorithms."

33. Dietrich, Eric (1999). "Algorithm". In Wilson, Robert Andrew; Keil, Frank C. (eds.). *The MIT Encyclopedia of the Cognitive Sciences* (https://books.google.com/books?id=-wt1aZrGXLYC). MIT Cognet library. Cambridge, Massachusetts: MIT Press (published 2001). p. 11. ISBN 9780262731447. Retrieved July 22, 2020. "An algorithm is a recipe, method, or technique for doing something."
34. Stone requires that "it must terminate in a finite number of steps" (Stone 1973:7–8).
35. Boolos and Jeffrey 1974,1999:19
36. cf Stone 1972:5
37. Knuth 1973:7 states: "In practice, we not only want algorithms, but we also want *good* algorithms ... one criterion of goodness is the length of time taken to perform the algorithm ... other criteria are the adaptability of the algorithm to computers, its simplicity, and elegance, etc."
38. cf Stone 1973:6
39. Stone 1973:7–8 states that there must be, "...a procedure that a robot [i.e., computer] can follow in order to determine precisely how to obey the instruction". Stone adds finiteness of the process, and definiteness (having no ambiguity in the instructions) to this definition.
40. Knuth, loc. cit
41. Minsky 1967, p. 105
42. Gurevich 2000:1, 3
43. Sipser 2006:157
44. Goodrich, Michael T.; Tamassia, Roberto (2002). *Algorithm Design: Foundations, Analysis, and Internet Examples* (http://ww3.algorithmdesign.net/ch00-front.html). John Wiley & Sons, Inc. ISBN 978-0-471-38365-9. Archived (https://web.archive.org/web/20150428201622/http://ww3.algorithmdesign.net/ch00-front.html) from the original on April 28, 2015. Retrieved June 14, 2018.
45. Knuth 1973:7
46. Chaitin 2005:32
47. Rogers 1987:1–2
48. In his essay "Calculations by Man and Machine: Conceptual Analysis" Seig 2002:390 credits this distinction to Robin Gandy, cf Wilfred Seig, et al., 2002 *Reflections on the foundations of mathematics: Essays in honor of Solomon Feferman*, Association for Symbolic Logic, A.K. Peters Ltd, Natick, MA.
49. cf Gandy 1980:126, Robin Gandy *Church's Thesis and Principles for Mechanisms* appearing on pp. 123–148 in J. Barwise et al. 1980 *The Kleene Symposium*, North-Holland Publishing Company.
50. A "robot": "A computer is a robot that performs any task that can be described as a sequence of instructions." cf Stone 1972:3
51. Lambek's "abacus" is a "countably infinite number of locations (holes, wires, etc.) together with an unlimited supply of counters (pebbles, beads, etc.). The locations are distinguishable, the counters are not". The holes have unlimited capacity, and standing by is an agent who understands and is able to carry out the list of instructions" (Lambek 1961:295). Lambek references Melzak who defines his Q-machine as "an indefinitely large number of locations ... an indefinitely large supply of counters distributed among these locations, a program, and an operator whose sole purpose is to carry out the program" (Melzak 1961:283). B-B-J (loc. cit.) add the stipulation that the holes are "capable of holding any number of stones" (p. 46). Both Melzak and Lambek appear in *The Canadian Mathematical Bulletin*, vol. 4, no. 3, September 1961.
52. Если путаницы не возникнет, слово "счетчики" можно исключить, и можно сказать, что местоположение содержит одно "число".
53. "Мы говорим, что инструкция *эффективна*, если существует процедура, которой робот может следовать, чтобы точно определить, как выполнять инструкцию". (Стоун 1972:6)
54. см. Minsky 1967: глава 11 "Компьютерные модели" и глава 14 "Очень простые основы вычислимости", стр. 255-281, в частности,
55. ср. Кнут 1973: 3.
56. Но ему всегда предшествует IF-THEN, чтобы избежать неправильного вычитания.
57. Кнут 1973:4
58. Стоун 1972: 5. Методы извлечения корней нетривиальны: см. Методы вычисления квадратных корней.
59. Leeuwen, Jan (1990). *Справочник по теоретической информатике: алгоритмы и сложность. Том A* (https://books.google.com/books?id=-X39_rA3VSQC). Elsevier. стр. 85. ISBN 978-0-444-88071-0.

60. Джон Г. Кемени и Томас Э. Курц 1985 *Назад к базовому: история, искажение и будущее языка*, Издательская компания Addison-Wesley Publishing Company, Inc. Рединг, Массачусетс, 0-201-13433-0 ISBN.
61. Таусворт 1977:101
62. Таусворт 1977:142
63. Кнут 1973, раздел 1.2.1, расширенный Таусвортом 1977 на страницах 100ff и главе 9.1
64. ср. Tausworthe 1977
65. Хит 1908: 300; Издание Хокинга 2005 года в Дувре происходит от Heath.
66. " "Пусть CD, измеряя BF, оставляет FA меньше самого себя". Это аккуратная аббревиатура для выражения " измеряйте вдоль BA последовательные длины, равные CD, пока не будет достигнута точка F, так что оставшаяся длина FA будет меньше CD; другими словами, пусть BF будет наибольшим точным кратным CD, содержащимся в BA" (Heath 1908: 297)
67. О современных методах лечения, использующих деление в алгоритме, см. Харди и Райт 1979: 180, Кнут 1973: 2 (том 1), а также более подробное обсуждение алгоритма Евклида в Knuth 1969: 293-297 (том 2).
68. Euclid covers this question in his Proposition 1.
69. "Euclid's Elements, Book VII, Proposition 2" (http://aleph0.clarku.edu/~djoyce/java/elements/bookVII/propVII2.html). Aleph0.clarku.edu. Archived (https://web.archive.org/web/20120524074919/http://aleph0.clarku.edu/~djoyce/java/elements/bookVII/propVII2.html) from the original on May 24, 2012. Retrieved May 20, 2012.
70. While this notion is in widespread use, it cannot be defined precisely.
71. Кнут 1973:13-18. Он приписывает "формулировку доказательства алгоритма в терминах утверждений и индукции" Р. В. Флойду, Питеру Науру, К. А.Р. Хоару, Х.Х. Голдстайну и Дж. фон Нейману. Таусворт 1977 заимствует пример Евклида Кнута и расширяет метод Кнута в разделе 9.1 *Формальные доказательства* (стр. 288-298).
72. Таусворт 1997:294
73. ср. Кнут 1973: 7 (том. I) и его более подробный анализ на стр. 1969: 294-313 (том II).
74. Сбой происходит, когда алгоритм пытается сжаться. Успех решит проблему остановки.
75. Kriegel, Hans-Peter; Schubert, Erich; Zimek, Arthur (2016). "(Черное) искусство оценки во время выполнения: сравниваем ли мы алгоритмы или реализации?". *Системы знаний и информации*. **52** (2): 341-378. doi: 10.1007/s10115-016-1004-2 (https://doi.org/10.1007%2Fs10115-016-1004-2). ISSN 0219-1377 (https://www.worldcat.org/issn/0219-1377). S2CID 40772241 (https://api.semanticscholar.org/CorpusID:40772241).
76. Джиллиан Конахан (январь 2013). "Лучшая математика делает сети передачи данных более быстрыми" (http://discovermagazine.com/2013/jan-feb/34-better-math-makes-faster-data-networks). discovermagazine.com. Архивировано (https://web.archive.org/web/20140513212427/http://discovermagazine.com/2013/jan-feb/34-better-math-makes-faster-data-networks) с оригинала 13 мая 2014 г.. Проверено 13 мая 2014 года.
77. Хайтам Хассани, Петр Индик, Дина Катаби и Эрик Прайс, "Симпозиум ACM-SIAM по дискретным алгоритмам (SODA)", (http://siam.omnibooksonline.com/2012SODA/data/papers/500.pdf) Архивировано (https://web.archive.org/web/20130704180806/http://siam.omnibooksonline.com/2012SODA/data/papers/500.pdf) 4 июля 2013 года в Wayback Machine, Киото, январь 2012 года. Смотрите также веб-страницу sFFT, (http://groups.csail.mit.edu/netmit/sFFT/) заархивированную (https://web.archive.org/web/20120221145740/http://groups.csail.mit.edu/netmit/sFFT/) 21 февраля 2012 г. на Wayback Machine.
78. Kellerer, Hans; Pferschy, Ulrich; Pisinger, David (2004). *Проблемы с рюкзаком | Ханс Келлерер | Спрингер* (https://www.springer.com/us/book/9783540402862). Спрингер. doi:10.1007/978-3-540-24777-7 (https://doi.org/10.1007%2F978-3-540-24777-7). ISBN 978-3-540-40286-2. S2CID 28836720 (https://api.semanticscholar.org/CorpusID:28836720). Архивировано (https://web.archive.org/web/20171018181055/https://www.springer.com/us/book/9783540402862) с оригинала 18 октября 2017 г.. Проверено 19 сентября 2017 года.

79. Например, объем выпуклого многогранника (описанный с использованием оракула принадлежности) может быть аппроксимирован с высокой точностью рандомизированным алгоритмом полиномиального времени, но не детерминированным: см. Дайер, Мартин; Фриз, Алан; Каннан, Рави (январь 1991). "Случайный алгоритм с полиномиальным временем для аппроксимации объема выпуклых тел". *J. ACM*. **38** (1): 1-17. CiteSeerX 10.1.1.145.4600 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.145.4600). doi:10.1145/102782.102783 (https://doi.org/10.1145%2F102782.102783). S2CID 13268711 (https://api.semanticscholar.org/CorpusID:13268711).
80. Джордж Б. Данциг и Мукунд Н. Тапа. 2003. *Линейное программирование 2: теория и расширения*. Springer-Verlag.
81. Цыпкин (1971). *Адаптация и обучение в автоматических системах* (https://books.google.com/books?id=sgDHJlafMskC&pg=PA54). Академическая пресса. стр. 54. ISBN 978-0-08-095582-7.
82. Kowalski, Robert (1979). "Алгоритм = Логика + управление" (https://doi.org/10.1145%2F359131.359136). *Коммуникации ACM*. **22** (7): 424–436. doi:10.1145/359131.359136 (https://doi.org/10.1145%2F359131.359136). S2CID 2509896 (https://api.semanticscholar.org/CorpusID:2509896).
83. Warren, D.S., 2023. Writing correct prolog programs. In Prolog: The Next 50 Years (pp. 62-70). Cham: Springer Nature Switzerland.
84. Kowalski, R., Dávila, J., Sartor, G. and Calejo, M., 2023. Logical English for law and education. In Prolog: The Next 50 Years (pp. 287–299). Cham: Springer Nature Switzerland.
85. "The Experts: Does the Patent System Encourage Innovation?" (https://www.wsj.com/articles/SB10001424127887323582904578487200821421958). *The Wall Street Journal*. May 16, 2013. ISSN 0099-9660 (https://www.worldcat.org/issn/0099-9660). Retrieved March 29, 2017.
86. Knuth, Donald E. (1972). "Ancient Babylonian Algorithms" (https://web.archive.org/web/20121224100137/http://steiner.math.nthu.edu.tw/disk5/js/computer/1.pdf) (PDF). *Commun. ACM*. **15** (7): 671–677. doi:10.1145/361454.361514 (https://doi.org/10.1145%2F361454.361514). ISSN 0001-0782 (https://www.worldcat.org/issn/0001-0782). S2CID 7829945 (https://api.semanticscholar.org/CorpusID:7829945). Archived from the original (http://steiner.math.nthu.edu.tw/disk5/js/computer/1.pdf) (PDF) on December 24, 2012.
87. Aaboe, Asger (2001). *Episodes from the Early History of Astronomy*. New York: Springer. pp. 40–62. ISBN 978-0-387-95136-2.
88. Ast, Courtney. "Eratosthenes" (http://www.math.wichita.edu/history/men/eratosthenes.html). Wichita State University: Department of Mathematics and Statistics. Archived (https://web.archive.org/web/20150227150653/http://www.math.wichita.edu/history/men/eratosthenes.html) from the original on February 27, 2015. Retrieved February 27, 2015.
89. Chabert, Jean-Luc (2012). *A History of Algorithms: From the Pebble to the Microchip*. Springer Science & Business Media. p. 2. ISBN 9783642181924.
90. Davis 2000:18
91. Bolter 1984:24
92. Bolter 1984:26
93. Bolter 1984:33–34, 204–206.
94. All quotes from W. Stanley Jevons 1880 *Elementary Lessons in Logic: Deductive and Inductive*, Macmillan and Co., London and New York. Republished as a googlebook; cf Jevons 1880:199–201. Louis Couturat 1914 *the Algebra of Logic*, The Open Court Publishing Company, Chicago and London. Republished as a googlebook; cf Couturat 1914 in:75–76 gives a few more details; he compares this to a typewriter as well as a piano. Jevons states that the account is to be found at January 20, 1870 *The Proceedings of the Royal Society*.
95. Jevons 1880:199–200
96. All quotes from John Venn 1881 *Symbolic Logic*, Macmillan and Co., London. Republished as a googlebook. cf Venn 1881:120–125. The interested reader can find a deeper explanation in those pages.
97. Bell and Newell diagram 1971:39, cf. Davis 2000
98. * Melina Hill, Valley News Correspondent, *A Tinkerer Gets a Place in History*, Valley News West Lebanon NH, Thursday, March 31, 1983, p. 13.
99. Davis 2000:14
100. van Heijenoort 1967:81ff
101. van Heijenoort's commentary on Frege's *Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought* in van Heijenoort 1967:1

102. Dixon 1906, cf. Kleene 1952:36–40
103. cf. footnote in Alonzo Church 1936a in Davis 1965:90 and 1936b in Davis 1965:110
104. Kleene 1935–6 in Davis 1965:237ff, Kleene 1943 in Davis 1965:255ff
105. Church 1936 in Davis 1965:88ff
106. cf. "Finite Combinatory Processes – formulation 1", Post 1936 in Davis 1965:289–290
107. Turing 1936–37 in Davis 1965:116ff
108. Rosser 1939 in Davis 1965:226
109. Kleene 1943 in Davis 1965:273–274
110. Kleene 1952:300, 317
111. Kleene 1952:376
112. Turing 1936–37 in Davis 1965:289–290
113. Turing 1936 in Davis 1965, Turing 1939 in Davis 1965:160
114. Hodges, p. 96
115. Turing 1936–37:116
116. Turing 1936–37 in Davis 1965:136
117. Turing 1939 in Davis 1965:160

Bibliography

- Axt, P (1959). "On a Subrecursive Hierarchy and Primitive Recursive Degrees" (https://doi.org/10.2307%2F1993169). *Transactions of the American Mathematical Society*. **92** (1): 85–105. doi:10.2307/1993169 (https://doi.org/10.2307%2F1993169). JSTOR 1993169 (https://www.jstor.org/stable/1993169).
- Bell, C. Gordon and Newell, Allen (1971), *Computer Structures: Readings and Examples*, McGraw–Hill Book Company, New York. ISBN 0-07-004357-4.
- Blass, Andreas; Gurevich, Yuri (2003). "Algorithms: A Quest for Absolute Definitions" (http://research.microsoft.com/~gurevich/Opera/164.pdf) (PDF). *Bulletin of European Association for Theoretical Computer Science*. **81**. Archived (https://ghostarchive.org/archive/20221009/http://research.microsoft.com/~gurevich/Opera/164.pdf) (PDF) from the original on October 9, 2022. Includes a bibliography of 56 references.
- Bolter, David J. (1984). *Turing's Man: Western Culture in the Computer Age* (1984 ed.). Chapel Hill, NC: The University of North Carolina Press. ISBN 978-0-8078-1564-9., ISBN 0-8078-4108-0
- Boolos, George; Jeffrey, Richard (1999) [1974]. *Computability and Logic* (https://archive.org/details/computabilitylog0000bool_r8y9) (4th ed.). Cambridge University Press, London. ISBN 978-0-521-20402-6.: cf. Chapter 3 *Turing machines* where they discuss "certain enumerable sets not effectively (mechanically) enumerable".
- Burgin, Mark (2004). *Super-Recursive Algorithms*. Springer. ISBN 978-0-387-95569-8.
- Campagnolo, M.L., Moore, C., and Costa, J.F. (2000) An analog characterization of the subrecursive functions. In *Proc. of the 4th Conference on Real Numbers and Computers*, Odense University, pp. 91–109
- Church, Alonzo (1936). "An Unsolvable Problem of Elementary Number Theory". *The American Journal of Mathematics*. **58** (2): 345–363. doi:10.2307/2371045 (https://doi.org/10.2307%2F2371045). JSTOR 2371045 (https://www.jstor.org/stable/2371045). Reprinted in *The Undecidable*, p. 89ff. The first expression of "Church's Thesis". See in particular page 100 (*The Undecidable*) where he defines the notion of "effective calculability" in terms of "an algorithm", and he uses the word "terminates", etc.
- Church, Alonzo (1936). "A Note on the Entscheidungsproblem". *The Journal of Symbolic Logic*. **1** (1): 40–41. doi:10.2307/2269326 (https://doi.org/10.2307%2F2269326). JSTOR 2269326 (https://www.jstor.org/stable/2269326). S2CID 42323521 (https://api.semanticscholar.org/CorpusID:42323521). Church, Alonzo (1936). "Correction to a Note on the Entscheidungsproblem". *The Journal of Symbolic Logic*. **1** (3): 101–102. doi:10.2307/2269030 (https://doi.org/10.2307%2F2269030). JSTOR 2269030 (https://www.jstor.org/stable/2269030). S2CID 5557237 (https://api.semanticscholar.org/CorpusID:5557237). Reprinted in *The Undecidable*, p. 110ff. Church shows that the Entscheidungsproblem is unsolvable in about 3 pages of text and 3 pages of footnotes.
- Daffa', Ali Abdullah al- (1977). *The Muslim contribution to mathematics*. London: Croom Helm. ISBN 978-0-85664-464-1.

- Davis, Martin (1965). *The Undecidable: Basic Papers On Undecidable Propositions, Unsolvable Problems and Computable Functions* (https://archive.org/details/undecidablebasic0000davi). New York: Raven Press. ISBN 978-0-486-43228-1. Davis gives commentary before each article. Papers of Gödel, Alonzo Church, Turing, Rosser, Kleene, and Emil Post are included; those cited in the article are listed here by author's name.
- Davis, Martin (2000). *Engines of Logic: Mathematicians and the Origin of the Computer*. New York: W.W. Nortion. ISBN 978-0-393-32229-3. Davis offers concise biographies of Leibniz, Boole, Frege, Cantor, Hilbert, Gödel and Turing with von Neumann as the show-stealing villain. Very brief bios of Joseph-Marie Jacquard, Babbage, Ada Lovelace, Claude Shannon, Howard Aiken, etc.
- This article incorporates public domain material from Paul E. Black. "algorithm" (https://xlinux.nist.gov/dads/HTML/algorithm.html). *Dictionary of Algorithms and Data Structures*. NIST.
- Dean, Tim (2012). "Evolution and moral diversity" (https://doi.org/10.4148%2Fbiyclc.v7i0.1775). *Baltic International Yearbook of Cognition, Logic and Communication*. **7**. doi:10.4148/biyclc.v7i0.1775 (https://doi.org/10.4148%2Fbiyclc.v7i0.1775).
- Dennett, Daniel (1995). *Darwin's Dangerous Idea* (https://archive.org/details/darwinsdangerous0000dennn). Vol. 2. New York: Touchstone/Simon & Schuster. pp. 32 (https://archive.org/details/darwinsdangerous0000denn/page/32)–36. Bibcode:1996Cmplx...2a..32M (https://ui.adsabs.harvard.edu/abs/1996Cmplx...2a..32M). doi:10.1002/(SICI)1099-0526(199609/10)2:1<32::AID-CPLX8>3.0.CO;2-H (https://doi.org/10.1002%2F%28SICI%291099-0526%28199609%2F10%292%3A1%3C32%3A%3AAID-CPLX8%3E3.0.CO%3B2-H). ISBN 978-0-684-80290-9. {{cite book}}: |journal= ignored (help)
- Dilson, Jesse (2007). *The Abacus* (https://archive.org/details/abacusworldsfirs0000dils) ((1968, 1994) ed.). St. Martin's Press, NY. ISBN 978-0-312-10409-2., ISBN 0-312-10409-X
- Yuri Gurevich, *Sequential Abstract State Machines Capture Sequential Algorithms* (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.146.3017&rep=rep1&type=pdf), ACM Transactions on Computational Logic, Vol 1, no 1 (July 2000), pp. 77–111. Includes bibliography of 33 sources.
- van Heijenoort, Jean (2001). *From Frege to Gödel, A Source Book in Mathematical Logic, 1879–1931* ((1967) ed.). Harvard University Press, Cambridge. ISBN 978-0-674-32449-7., 3rd edition 1976[?], ISBN 0-674-32449-8 (pbk.)
- Hodges, Andrew (1983). *Alan Turing: The Enigma*. Vol. 37. New York: Simon and Schuster. pp. 107–108. Bibcode:1984PhT....37k.107H (https://ui.adsabs.harvard.edu/abs/1984PhT....37k.107H). doi:10.1063/1.2915935 (https://doi.org/10.1063%2F1.2915935). ISBN 978-0-671-49207-6. {{cite book}}: |journal= ignored (help), ISBN 0-671-49207-1. Cf. Chapter "The Spirit of Truth" for a history leading to, and a discussion of, his proof.
- Kleene, Stephen C. (1936). "General Recursive Functions of Natural Numbers" (https://web.archive.org/web/20140903092121/http://gdz.sub.uni-goettingen.de/index.php?id=11&PPN=GDZPPN002278499&L=1). *Mathematische Annalen*. **112** (5): 727–742. doi:10.1007/BF01565439 (https://doi.org/10.1007%2FBF01565439). S2CID 120517999 (https://api.semanticscholar.org/CorpusID:120517999). Archived from the original (http://gdz.sub.uni-goettingen.de/index.php?id=11&PPN=GDZPPN002278499&L=1) on September 3, 2014. Retrieved September 30, 2013. Presented to the American Mathematical Society, September 1935. Reprinted in *The Undecidable*, p. 237ff. Kleene's definition of "general recursion" (known now as mu-recursion) was used by Church in his 1935 paper *An Unsolvable Problem of Elementary Number Theory* that proved the "decision problem" to be "undecidable" (i.e., a negative result).
- Kleene, Stephen C. (1943). "Recursive Predicates and Quantifiers" (https://doi.org/10.2307%2F1990131). *Transactions of the American Mathematical Society*. **53** (1): 41–73. doi:10.2307/1990131 (https://doi.org/10.2307%2F1990131). JSTOR 1990131 (https://www.jstor.org/stable/1990131). Reprinted in *The Undecidable*, p. 255ff. Kleene refined his definition of "general recursion" and proceeded in his chapter "12. Algorithmic theories" to posit "Thesis I" (p. 274); he would later repeat this thesis (in Kleene 1952:300) and name it "Church's Thesis"(Kleene 1952:317) (i.e., the Church thesis).
- Kleene, Stephen C. (1991) [1952]. *Introduction to Metamathematics* (Tenth ed.). North-Holland Publishing Company. ISBN 978-0-7204-2103-3.
- Knuth, Donald (1997). *Fundamental Algorithms, Third Edition*. Reading, Massachusetts: Addison–Wesley. ISBN 978-0-201-89683-1.
- Knuth, Donald (1969). *Volume 2/Seminumerical Algorithms, The Art of Computer Programming First Edition*. Reading, Massachusetts: Addison–Wesley.

- Kosovsky, N.K. *Elements of Mathematical Logic and its Application to the theory of Subrecursive Algorithms*, LSU Publ., Leningrad, 1981
- Kowalski, Robert (1979). "Algorithm=Logic+Control" (https://doi.org/10.1145%2F359131.359136). *Communications of the ACM*. **22** (7): 424–436. doi:10.1145/359131.359136 (https://doi.org/10.1145%2F359131.359136). S2CID 2509896 (https://api.semanticscholar.org/CorpusID:2509896).
- A.A. Markov (1954) *Theory of algorithms*. [Translated by Jacques J. Schorr-Kon and PST staff] Imprint Moscow, Academy of Sciences of the USSR, 1954 [i.e., Jerusalem, Israel Program for Scientific Translations, 1961; available from the Office of Technical Services, U.S. Dept. of Commerce, Washington] Description 444 p. 28 cm. Added t.p. in Russian Translation of Works of the Mathematical Institute, Academy of Sciences of the USSR, v. 42. Original title: Teoriya algerifmov. [QA248.M2943 Dartmouth College library. U.S. Dept. of Commerce, Office of Technical Services, number OTS 60-51085.]
- Minsky, Marvin (1967). *Computation: Finite and Infinite Machines* (https://archive.org/details/computationfinit0000mins) (First ed.). Prentice-Hall, Englewood Cliffs, NJ. ISBN 978-0-13-165449-5. Minsky expands his "...idea of an algorithm – an effective procedure..." in chapter 5.1 *Computability, Effective Procedures and Algorithms. Infinite machines.*
- Post, Emil (1936). "Finite Combinatory Processes, Formulation I". *The Journal of Symbolic Logic*. **1** (3): 103–105. doi:10.2307/2269031 (https://doi.org/10.2307%2F2269031). JSTOR 2269031 (https://www.jstor.org/stable/2269031). S2CID 40284503 (https://api.semanticscholar.org/CorpusID:40284503). Reprinted in *The Undecidable*, pp. 289ff. Post defines a simple algorithmic-like process of a man writing marks or erasing marks and going from box to box and eventually halting, as he follows a list of simple instructions. This is cited by Kleene as one source of his "Thesis I", the so-called Church–Turing thesis.
- Rogers, Hartley Jr. (1987). *Theory of Recursive Functions and Effective Computability*. The MIT Press. ISBN 978-0-262-68052-3.
- Rosser, J.B. (1939). "An Informal Exposition of Proofs of Godel's Theorem and Church's Theorem". *Journal of Symbolic Logic*. **4** (2): 53–60. doi:10.2307/2269059 (https://doi.org/10.2307%2F2269059). JSTOR 2269059 (https://www.jstor.org/stable/2269059). S2CID 39499392 (https://api.semanticscholar.org/CorpusID:39499392). Reprinted in *The Undecidable*, p. 223ff. Herein is Rosser's famous definition of "effective method": "...a method each step of which is precisely predetermined and which is certain to produce the answer in a finite number of steps... a machine which will then solve any problem of the set with no human intervention beyond inserting the question and (later) reading the answer" (p. 225–226, *The Undecidable*)
- Santos-Lang, Christopher (2015). "Moral Ecology Approaches to Machine Ethics" (http://grinfree.com/MoralEcology.pdf) (PDF). In van Rysewyk, Simon; Pontier, Matthijs (eds.). *Machine Medical Ethics*. Intelligent Systems, Control and Automation: Science and Engineering. Vol. 74. Switzerland: Springer. pp. 111–127. doi:10.1007/978-3-319-08108-3_8 (https://doi.org/10.1007%2F978-3-319-08108-3_8). ISBN 978-3-319-08107-6. Archived (https://ghostarchive.org/archive/20221009/http://grinfree.com/MoralEcology.pdf) (PDF) from the original on October 9, 2022.
- Scott, Michael L. (2009). *Programming Language Pragmatics* (3rd ed.). Morgan Kaufmann Publishers/Elsevier. ISBN 978-0-12-374514-9.
- Sipser, Michael (2006). *Introduction to the Theory of Computation* (https://archive.org/details/introductiontoth00sips). PWS Publishing Company. ISBN 978-0-534-94728-6.
- Sober, Elliott; Wilson, David Sloan (1998). *Unto Others: The Evolution and Psychology of Unselfish Behavior* (https://archive.org/details/untoothersevolut00sobe). Cambridge: Harvard University Press. ISBN 9780674930469.
- Stone, Harold S. (1972). *Introduction to Computer Organization and Data Structures* (1972 ed.). McGraw-Hill, New York. ISBN 978-0-07-061726-1. Cf. in particular the first chapter titled: *Algorithms, Turing Machines, and Programs*. His succinct informal definition: "...any sequence of instructions that can be obeyed by a robot, is called an *algorithm*" (p. 4).
- Tausworthe, Robert C (1977). *Standardized Development of Computer Software Part 1 Methods*. Englewood Cliffs NJ: Prentice–Hall, Inc. ISBN 978-0-13-842195-3.
- Turing, Alan M. (1936–37). "On Computable Numbers, With An Application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society*. Series 2. **42**: 230–265. doi:10.1112/plms/s2-42.1.230 (https://doi.org/10.1112%2Fplms%2Fs2-42.1.230). S2CID 73712 (https://api.semanticscholar.org/CorpusID:73712).. Corrections, ibid, vol. 43(1937) pp. 544–546. Reprinted in *The Undecidable*, p. 116ff. Turing's famous paper completed as a Master's dissertation while at King's College Cambridge UK.

- Turing, Alan M. (1939). "Systems of Logic Based on Ordinals". *Proceedings of the London Mathematical Society*. **45**: 161–228. doi:10.1112/plms/s2-45.1.161 (https://doi.org/10.1112%2Fplms%2Fs2-45.1.161). hdl:21.11116/0000-0001-91CE-3 (https://hdl.handle.net/21.11116%2F0000-0001-91CE-3). Reprinted in *The Undecidable*, pp. 155ff. Turing's paper that defined "the oracle" was his PhD thesis while at Princeton.
- United States Patent and Trademark Office (2006), *2106.02 **>Mathematical Algorithms: 2100 Patentability* (http://www.uspto.gov/web/offices/pac/mpep/documents/2100_2106_02.htm), Manual of Patent Examining Procedure (MPEP). Latest revision August 2006
- Zaslavsky, C. (1970). Mathematics of the Yoruba People and of Their Neighbors in Southern Nigeria. The Two-Year College Mathematics Journal, 1(2), 76–99. https://doi.org/10.2307/3027363

Further reading

- Bellah, Robert Neelly (1985). *Habits of the Heart: Individualism and Commitment in American Life* (https://books.google.com/books?id=XsUojihVZQcC). Berkeley: University of California Press. ISBN 978-0-520-25419-0.
- Berlinski, David (2001). *The Advent of the Algorithm: The 300-Year Journey from an Idea to the Computer* (https://archive.org/details/adventofalgorith0000berl). Harvest Books. ISBN 978-0-15-601391-8.
- Chabert, Jean-Luc (1999). *A History of Algorithms: From the Pebble to the Microchip*. Springer Verlag. ISBN 978-3-540-63369-3.
- Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest; Clifford Stein (2009). *Introduction To Algorithms* (3rd ed.). MIT Press. ISBN 978-0-262-03384-8.
- Harel, David; Feldman, Yishai (2004). *Algorithmics: The Spirit of Computing*. Addison-Wesley. ISBN 978-0-321-11784-7.
- Hertzke, Allen D.; McRorie, Chris (1998). "The Concept of Moral Ecology". In Lawler, Peter Augustine; McConkey, Dale (eds.). *Community and Political Thought Today*. Westport, CT: Praeger.
- Knuth, Donald E. (2000). *Selected Papers on Analysis of Algorithms (http://www-cs-faculty.stanford.edu/~uno/aa.html) Archived (https://web.archive.org/web/20170701190647/http://www-cs-faculty.stanford.edu/~uno/aa.html) July 1, 2017, at the Wayback Machine*. Stanford, California: Center for the Study of Language and Information.
- Knuth, Donald E. (2010). *Selected Papers on Design of Algorithms (http://www-cs-faculty.stanford.edu/~uno/da.html) Archived (https://web.archive.org/web/20170716225848/http://www-cs-faculty.stanford.edu/~uno/da.html) July 16, 2017, at the Wayback Machine*. Stanford, California: Center for the Study of Language and Information.
- Wallach, Wendell; Allen, Colin (November 2008). *Moral Machines: Teaching Robots Right from Wrong*. US: Oxford University Press. ISBN 978-0-19-537404-9.
- Bleakley, Chris (2020). *Poems that Solve Puzzles: The History and Science of Algorithms* (https://books.google.com/books?id=3pr5DwAAQBAJ). Oxford University Press. ISBN 978-0-19-885373-2.

External links

- "Algorithm" (https://www.encyclopediaofmath.org/index.php?title=Algorithm). *Encyclopedia of Mathematics*. EMS Press. 2001 [1994].
- Algorithms (https://curlie.org/Computers/Algorithms/) at Curlie
- Weisstein, Eric W. "Algorithm" (https://mathworld.wolfram.com/Algorithm.html). *MathWorld*.
- Dictionary of Algorithms and Data Structures (https://www.nist.gov/dads/) – National Institute of Standards and Technology

Algorithm repositories

- The Stony Brook Algorithm Repository (http://www.cs.sunysb.edu/~algorith/) – State University of New York at Stony Brook
- Collected Algorithms of the ACM (http://calgo.acm.org/) – Associations for Computing Machinery
- The Stanford GraphBase (http://www-cs-staff.stanford.edu/~knuth/sgb.html) Archived (https://web.archive.org/web/20151206222112/http://www-cs-staff.stanford.edu/%7Eknuth/sgb.html) December 6, 2015, at

the Wayback Machine – Stanford University

-