Лекция

Теория алгоритмов

"Если бы строители строили здания так же, как программисты пишут программы, первый залетевший дятел разрушил бы цивилизацию"

Второй закон Вейлера

- Современная история математики большую роль формированию алгебры в том виде, который мы имеем сейчас, относит к трудам ал-Хорезми, арабского учёного,

- «Слово “алгорифм” в форме “алгоризм” часто употреблялось в качестве заглавия изложений индийского счисления в рукописях XII–XV вв. и в книгах XV–XVI вв.».
- Свои трактаты ал-Хорезми начинал со слов «dixit algorizmi», что означает «Говорит ал-Хорезми

- Понятие алгоритма принадлежит к числу понятий столь фундаментальных, что не может быть выражено через другие ,а должно рассматриваться как неопределяемое .
- Алгоритм — это точное предписание, которое задаёт вычислительной процесс, начинающийся с произвольного исходного данного и направленный на получение полностью определяемого этим исходным данным результата .(определение Маркова)

Закон Хоара (о больших задачах) : "Внутри каждой большой задачи сидит маленькая, которая старается выбраться наружу"

Многие люди думают, что это значит:

Большие задачи на самом деле маленькие

Что имеется в виду:

Большие задачи составлены из меньших задач

Предостережение:

Найти маленькую задачу в центре большой — недостаточно для решения — это только начало исследований...

- Формулировка Колмогорова содержит две существенные идеи:
- итеративность алгоритмического процесса
- локальность каждого отдельного шага.

- **Теория алгоритмов - раздел математики, изучающий общие свойства алгоритмов.**
- **Определение**

Алгоритм - предписание, однозначно задающее процесс преобразования исходной информации в виде последовательности элементарных дискретных шагов, приводящих за их конечное число к результату.

Исторический обзор

- Первым дошедшим до нас алгоритмом в его интуитивном понимании – конечной последовательности элементарных действий, решающих поставленную задачу, считается предложенный Евклидом в III веке до нашей эры алгоритм нахождения наибольшего общего делителя двух чисел (алгоритм Евклида).

Алгоритм Евклида .

- Алгоритм Евклида - это способ нахождения НОД двух натуральных чисел
- Пример
- Пусть $a = 525$, $b = 231$. Найдем НОД

(алгоритм Евклида)

- $525 = 231 \cdot 2 + 63$
- $231 = 63 \cdot 3 + 42$
- $63 = 42 \cdot 1 + 21$
- $42 = 21 \cdot 2$
- Таким образом, $(525, 231) = 21$.

- Пусть даны числа *a* и *b*; *a* >= 0, *b*>= 0, считаем, что *a* > *b* .

Алгоритм:

- **1. Ввести** *a* и *b* .
- **2. Если** *b* = 0 **,** то **Ответ:** *a* . **Конец** .
- **3. Заменить** *r* := "остаток от деления *a* на *b* ", *a* := *b* , *b* := *r* .
- **4. Идти** на **2.**

Алгоритм Евклида - это способ нахождения НОД двух натуральных чисел a и b.

Предположим для определенности, что $a \geq b$. Разделим a на b с остатком:

$a = bq + r$, q и r — целые, $0 \leq r < b$.

Теперь имеется две возможности:

1) $r = 0$. Тогда ясно, что $(a, b) = b$.

2) $r > 0$

Алгоритм Евклида - это способ нахождения НОД двух натуральных чисел a и b.

Предположим для определенности, что $a \geq b$. Разделим a на b с остатком:

$a = bq + r$, q и r — целые, $0 \leq r < b$.

Теперь имеется две возможности:

1) $r = 0$. Тогда ясно, что $(a, b) = b$.

2) $r > 0$

Алгоритм Евклида - это способ нахождения НОД двух натуральных чисел a и b.

Предположим для определенности, что $a \geq b$. Разделим a на b с остатком:

$$a = bq + r,\ q \text{ и } r \text{ — целые},\ 0 \leq r < b.$$

Теперь имеется две возможности:

1) $r = 0$. Тогда ясно, что $(a, b) = b$.

2) $r > 0$

Тогда нужно воспользоваться следующим замечательным соотношением:

$$(a,b) = (b,r),$$

вытекающим из того, что каждый общий делитель a и b делит r и каждый общий делитель b и r делит a

(это видно из определения деления с остатком), так что множество общих делителей a и b совпадает с множеством общих делителей b и r, в частности, совпадают наибольшие общие делители.

Имеем:

$$a \geq b > r.$$

Теперь вместо (a,b) надо находить (b,r). Деля b на r и обозначая остаток через r_1, мы получаем:

остаток: теперь r, мы получили:

$(a,b) = (b,r) = (r,r_1), \quad a \geq b > r > r_1.$

Если $r_1 = 0$, то $(a,b) = (b,r) = r$. Если же $r_1 \neq 0$, то надо делить r на r_1 и так далее, пока не получится остаток, равный 0.

В конце концов это произойдет, поскольку остатки все время уменьшаются.

Последний отличный от нуля остаток и будет равен (a,b). Окончательно запишем весь процесс так:

$$a = bq + r$$

$$b = rq_1 + r_1$$

$$r = r_1q_2 + r_2$$

$$r_1 = r_2q_3 + r_3$$

$$\cdots$$

$$r_{n-2} = r_{n-1}q_n + r_n$$

$$r_{n-1} = r_nq_{n+1}$$

$$a \geq b > r > r_1 > r_2 > \ldots > r_{n-1} > r_n > 0$$

$$(a, b) = r_n$$

Древний Вавилон

- Зародились начала алгебры, т.е. решения систем линейных и квадратичных уравнений.
- «Я вычел из площади сторону моего квадрата, это 14,30» -
- *современный язык* $x2—x=14,30$.
- «Ты берёшь 1, коэффициент.
- Ты делишь пополам 1, это 0;30.
- Ты умножаешь 0;30 на 0;30, это 0;15.
- Ты складываешь [это] с 14,30 и это есть 14,30;15, что является квадратом для 29;30.
- Ты складываешь 0;30, которое ты умножал, с 29;30, получается 30, сторона квадрата»,
- Перевести!!!

- Первые фундаментальные работы по теории алгоритмов были опубликованы независимо в 1936 году годы
- Аланом Тьюрингом,
- Алоизом Черчем и
- Эмилем Постом
- В 1950-е годы существенный вклад в теорию алгоритмов внесли работы Колмогорова и Маркова.

Направления в теории алгоритмов

- Классическая теория алгоритмов
- формулировка задач в терминах формальных языков,
- понятие задачи разрешения,
- введение сложностных классов,
- открытие класса NP-полных задач и его исследование);

- Теория асимптотического анализа алгоритмов
- понятие сложности и трудоёмкости алгоритма,
- критерии оценки алгоритмов,
- методы получения асимптотических оценок,

- Теория практического анализа вычислительных алгоритмов
- получение явных функции трудоёмкости,
- интервальный анализ функций,
- практические критерии качества алгоритмов,
- методика выбора рациональных алгоритмов.

Основные свойства алгоритма:

- 1. *Дискретность*. Процесс решения протекает в виде последовательности отдельных действий, следующих друг за другом.
- 2. *Элементарность действий.* Каждое действие не допускает возможности неоднозначного толкования.
- 3. *Определенность.* Каждое действие определено и после выполнения каждого действия однозначно определяется, какое действие будет выполнено следующим.
- 4. *Конечность.* Алгоритм заканчивает работу после конечного числа шагов.

- 5. *Результативность*. В момент прекращения работы алгоритма известно, что является результатом.
- 6. *Массовость.* Алгоритм описывает некоторое множество процессов, применимых при различных входных данных.
- Алгоритм считается правильным, если при любых допустимых данных он заканчивает работу и выдает результат, удовлетворяющий требованиям задачи.
- Алгоритм однозначен, если при применении к одним и тем же входным данным он дает один и тот же результат

Три класса алгоритмов

- *1. Вычислительные алгоритмы -* работают с простыми видами данных (числа, матрицы).
- *2. Информационные алгоритмы* представляют собой набор сравнительно небольших процедур, но работающих с большими объемами информации
- *Управляющие алгоритмы* характерны тем, что данные к ним поступают от внешних процессов, которыми они управляют. Результатом работы этих алгоритмов являются различные управляющие воздействия.

Оценка сложности алгоритма

- измеряется двумя параметрами: **T**(временная сложность)
- **S** (пространственная сложность, или требования к памяти).
- **T** и **S** обычно представляются в виде функций от **n**, где **n** - размер входных данных.

Пример:

Если временная сложность алгоритма описывается как

- T(n) = 4n2 + 7n + 12, то вычислительная сложность определяется, как O(n2).
- Временная сложность, определяемая таким образом не зависит от реализации

Классификация алгоритмов по сложности

- 1. *Постоянный* - сложность оценивается как O(1).
- 2. *Линейный* - оценка равна O(n).
- 3. *Квадратный* - O(n2)
- 4. *Кубический, полиноминальный* - O(n3),O(nm).
- 5. *Экспоненциальный* - O(tp(n)), t- константа, p(n) - некоторая полиномиальная функция.
- 6. *Факториальный* - O(n!). Обладает наибольшей временной сложностью среди всех известных типов.

Время работы алгоритма

Полиномиальные алгоритмы

n^2 $n \log n$ n^3 n^{1000}

Псевдополиномиальные алгоритмы

$n^{\log n}$ $n^{\log^2 n}$ $c^{\log^7 n}$

Субэкспоненциальные алгоритмы

$2^{\sqrt{n}}$ $5^{(n^{0.98})}$

Экспоненциальные алгоритмы

2^n 8^n $n!$ n^n

Сравните 1.1^n и $10n^3$ для n=100

Базовый алгоритм:

Время работы 2^n, на практике работает для $n \leq n_0$

Аппаратный подход (закон Мура)

За полтора года компьютеры удвоили производительность

$$n_0 \to n_0 + 1$$

Мозговой подход

Удалось придумать алгоритм работающий за 1.41^n

$$n_0 \to 2n_0$$

- Проблемы, которые невозможно решить за полиномиальное время, называют *нерешаемыми*, потому что нахождение их решений быстро становится невозможным.
- Нерешаемые проблемы иногда называют трудными.
- Замечание некоторые проблемы принципиально неразрешимы, то есть даже отвлекаясь от временной сложности, невозможно создать алгоритм их решения

Примеры трудных задач

- **• Задача комивояжера.**

Комивояжер должен объехать N городов с целью осуществления продажи своих товаров.

Все N городов соединены дорогами по принципу "каждый с каждым".

Известна стоимость проезда между двумя любыми городами.

Найти оптимальный маршрут движения так, чтобы побывать во всех городах и при этом иметь минимальные затраты на дорогу.

- **Решение** (метод грубого перебора).
- Произвольно пронумеруем N городов целыми числами от 1 до N, причем базовый город имеет номер N.
- Каждый тур (один из возможных маршрутов) однозначно соответствует перестановке целых чисел 1, 2, ..N - 1.
- Для каждой перестановки строим тур и определяем его стоимость.
- Обрабатывая все перестановки запоминаем маршрут, который имеет на текущий момент самую низкую стоимость.

- Если находится маршрут с меньшей стоимостью, то все дальнейшие сравнения осуществляем с ним.
- Алгоритм является факториальным, с оценкой O(n!).
- В задаче требуется найти (N - 1)! перестановок целых чисел.
- Если даже требуется только один шаг для каждой перестановки, то эта часть алгоритма потребует O[(n - 1)!] шагов, поэтому любая верхняя граница для общего времени работы должна быть O(n!).

- **Проблема тройного брака.**
- В комнате n мужчин, n женщин и n чиновников. Есть список разрешенных браков, записи которого состоят из одного мужчины, одной женщины и одного регистрирующего чиновника.
- Если дан этот список троек, то возможно ли построить n браков так, чтобы любой либо сочетался браком только с одним человеком или регистрировал только один брак?

- Быстрыми являются **линейные алгоритмы**, которые обладают сложностью порядка n и называются также алгоритмами порядка O(n), где n - размерность входных данных
- Пример
- К линейным алгоритмам относится алгоритм нахождения суммы десятичных чисел, состоящих из n1 и n2 цифр. Сложность этого алгоритма - O(n1 + n2).

- **Определение Полиномиальный алгоритм** (или алгоритм полиномиальной временной сложности, или алгоритм принадлежащим классу **P**) - алгоритм, у которого временная сложность равна O(nk), где k - целое число > 0.
- Пример
- алгоритмы - деление, извлечение квадратного корня, решение систем линейных уравнений и др. - попадают в более общий класс полиномиальных алгоритмов.

Класс NP

Неформально: задачи, которые могут быть решены перебором $2^{poly(n)}$ вариантов

Принадлежность классу **NP** — это положительная характеристика!

Пример: разложение числа на множители

NP-полные задачи

Неформально: наиболее трудные задачи внутри класса **NP**

Примеры: Коммивояжер, Гамильтонов цикл, Сумма размеров

NP-трудные задачи

Неформально: задачи к которым можно полиномиально свести любою задачу из класса **NP**

Не обязательно лежат в **NP**!

Методы для решения NP-полных задач

- Рекурсия
- Хитрый перебор
- Локальный поиск
- Случайный порядок действий
- Вероятностное сведение к простой задаче
- Случайное блуждание

Задача ВЫПОЛНИМОСТЬ (SAT)

Дано: булева формула в КНФ

Определить: существует ли выполняющая подстановка?

Выполнима ли формула: $(\neg a \vee \neg b)\&(a \vee b \vee c)\&(a \vee \neg b)$?

Задача 3-ВЫПОЛНИМОСТЬ (3-SAT)

Каждая скобка содержит не более трех переменных

Факт: 3-SAT является NP-полной

Идея:

Сводить решение задачи для n переменных к меньшему количеству переменных

Алгоритм Мониена и Шпекенмайера [1985]

1. Фиксируем скобку (пусть $(x \vee y \vee z)$)
2. Решаем задачу для $x = 1$
3. Если ответ "невыполнима", решаем задачу для $x = 0, y = 1$
4. Если ответ "невыполнима", решаем задачу для $x = 0, y = 0, z = 1$

Рекурентное уравнение:

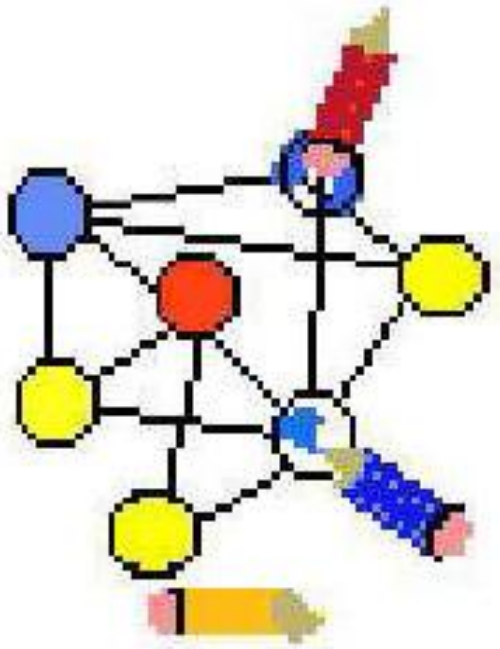
$T(n) \leq T(n-1) + T(n-2) + T(n-3)$

Факт: $T(n) = O(1.84^n)$

Задача 3-раскрашиваемость

Дано: неориентированный граф

Определить: существует ли правильная раскраска вершин в три цвета?



Рекурсия для раскраски

Тупая идея: полный перебор — $O(3^n)$

Простой рекурсивный алгоритм:

Фиксируем цвет первой вершины
Перебираем не более чем ДВА варианта раскраски какого-нибудь соседа для уже раскрашенных
Сложность - $O(2^n)$

Можно сделать хитрее:

Факт: 2-раскрашиваемость полиномиально разрешима
В один из цветов покрашено не более $n/3$ вершин
Переберем все варианты для множества синих вершин:

$$\sum_{i=0}^{n/3} C_n^i \leq 1.89^n$$

Сумма размеров

Данные:

Натуральные числа $w_1, \ldots, w_n$ и натуральное s

Выяснить:

Существует ли подмножество $w_1, \ldots, w_n$,
сумма элементов которого дает ровно s?

Пример:

17 43 23 38 14 20 36 47 ; 100

Есть ли подмножество с суммой 100?

Факт: задача сумма размеров являются NP-полными

Применение к сумме размеров

Разделим на две части, сосчитаем все подмножества в частях, получим 2-табличную сумму размера $2^{n/2}$

Недостаток: требование по памяти — $2^{n/2}$

$$\underbrace{17 \quad 43 \quad 23 \quad 38}_{} \quad \underbrace{14 \quad 20 \quad 36 \quad 47}_{} \; ; 100$$

$$\begin{array}{ll} \{17, 43, 23, & 100 - 14 = 86 \\ 38, 60, 40, 55, & 100 - 20 = 80 \\ 66, 81, 61, 83, & 100 - 36 = 64 \\ 98, 78\} & 100 - 47 = 53 \\ & 100 - 34 = 66 \end{array}$$

!

Класс E: задачи, экспоненциальные по природе

- К экспоненциальным задачам относятся задачи, в которых требуется построить множество всех подмножеств данного множества, все полные подграфы некоторого графа или же все поддеревья некоторого графа.

Задачи не попадающие ни в класс P, ни в класс E

- **задача о выполнимости**: существует ли для данной булевской формулы, находящейся в КНФ, такое распределение истинностных значений, что она имеет значение И?
- **задача коммивояжера**;
- **решение систем уравнений с целыми переменными;**
- **составление расписаний, учитывающих определенные условия**;
- **размещение** обслуживающих центров (телефон, телевидение, срочные службы) для максимального числа клиентов при минимальном числе центров;

- **оптимальная загрузка** емкости (рюкзак, поезд, корабль, самолёт) при наименьшей стоимости;
- **оптимальный раскрой** (бумага, картон, стальной прокат, отливки), оптимизация маршрутов в воздушном пространстве, инвестиций, станочного парка;
- **задача распознавания простого числа**;

- *Детерминированные алгоритмы* - во всех них для любого данного состояния существует не больше одного вполне определенного "следующего" состояния.
- детерминированный алгоритм в каждый момент времени может делать только что-либо одно.
- В недерминированном алгоритме для любого данного состояния может быть больше одного допустимого следующего состояния; другими словами, недетерминированный алгоритм в каждый момент времени может делать больше одной вещи.

- Недетерминированные алгоритмы не являются в каком-то смысле вероятностными или случайными алгоритмами;
- они являются алгоритмами, которые могут находиться одновременно во многих состояниях.

Нормальные алгоритмы Маркова

- **Определение** Алфавит **-** конечное, непустое множество элементов называемых буквами. Различные сочетания букв образуют слова.
- **Определение Слово -** это любая конечная последовательность знаков алфавита.
- Марков любую последовательность букв называл «словами».
- **Определение** Число символов в слове называется его **длиной.**
- **Определение** Слово, длина которого равна нулю, называется ***пустым***

- **Определение Нормальная схема подстановок -** это конечный набор, состоящий из пар слов, где левое слово переходит в правое (но не наоборот).

Пример

Алфавит содержит символы русского языка: $A = \{а,б \ldots я\}$. Найти систему подстановок, обеспечивающих преобразования: *нуть* →*муть*, *поло* →*мала*. Найти результат применения такого алгоритма к исходным словам *папа*, *пузо*

Решение

Система подстановок достаточно очевидна: *п* →*м*, *о* →*а*.

Применение алгоритма: папа →мапа →мама пузо →музо →муза

Пример

Пусть задан алфавит $A = \{*, 1\}$ и единственная подстановка: $*1 \to 1$;

Найти результат обработки, если исходным является слово $P = 11*111*1$

Решение

Применение нормального алгоритма с указанной подстановкой к данному слову дает последовательность (подчеркиванием выделяется преобразуемая комбинация):

$$\underline{11}*\underline{1}11*1 \to 11111*\underline{1} \to 111111$$

т.е. алгоритм находит количество единиц в исходном слове (суммирует числа в унарной системе счисления).

Нормальный алгоритм Маркова

- задается *алфавитом* А и *нормальной схемой подстановок*,
- где алфавит - конечное, непустое множество элементов называемых буквами.

  Различные сочетания букв образуют слова

Нормальный алгоритм Маркова задается.

1. алфавитом $A = \{a_0, a_1, \ldots a_N\}$
2. нормальной схемой подстановки $P = \{A_i \rightarrow B_i\}$, определяющих множество формул подстановки
3. множеством завершающих формул $R = \{A_1 \Rightarrow B_1\}$, где $R \subseteq P$.

Основная гипотеза теории алгоритмов в форме Маркова приобретает такой вид:

Всякий алгоритм нормализуем.

- **Тезис Маркова** : всякий алгоритм в алфавите А представим в виде нормального алгоритма в этом же алфавите.

- Это тезис потому, что его невозможно доказать, т.к. в нем фигурируют с одной стороны, интуитивное расплывчатое понятие "всякий алгоритм", а с другой стороны - точное понятие "нормальный алгоритм".

Алгоритм как абстрактная машина

- **Алгоритмические процессы – это процессы, которые может осуществлять определенным образом устроенная машина, моделирующая тем самым выполнение отдельных операций человеком.**

Общие требования к машинам

- характер их функционирования должен быть *дискретным,* т.е. состоять из отдельных шагов , каждый из которых выполняется только после завершения предыдущего;
- действия должны быть *детерминированы*, т.е. шаги выполняются в строгом порядке, а их результат определяется самим шагом и результатами предыдущих шагов;
- перед началом работы машине предоставляются исходные данные из области определения алгоритма;

- за конечное число шагов работы машины должен быть получен результат (или информация о том, что считать результатом);
- машина должна быть *универсальной*, т.е. такой, чтобы с ее помощью можно было бы выполнить любой алгоритм.

- Концепция алгоритма как абстрактной машины была выдвинута одновременно (1936 – 1937 гг.) английским математиком Аланом Тьюрингом и Эмилем Постом