Структуры данных и анализы. База

silvia.lesnaia

20 ноября 2025 г.

1 Определение алгоритма. Свойства алгоритмов (пять основных свойств: дискретность, детерминированность, понятность, массовость, результативность). Причины возникновения формального определения алгоритма. Основные типы алгоритмических моделей (перечислить).

1.1 Алгоритм

конечный набор правил, который определяет последовптельность операций для решения конкретного множества задач и обладает пятью важными чертами: конченость, определенность, ввод вывод, эффективность (Д. Кнут)

1.2 Основные свойства алгоритмов

Дискретность — алгоритм должен решать задачу, проделывая конечное множество элементарных шагов, каждый из которых выполняется во времени дискретно.

Детерминирванность (определённость): в кажждый момент времени шаг работы алгоритма однозначно определяется состоянием системы.

Понятность : алгоритм должен включать команды, понятные исполнителю.

Завершаемость (результативность, конечность) : при корректно заданных исходных данных алгоритм должен давать результат за конечное число шагов.

Массовость (универсальность): задача может не иметь универсального решения.

1.3 Причины возникновения формального определения алгоритма

Доказательство алгоритмической неразрешимости требовало формального уточнения понятия алгоритма. В отличие от создания конкретных алгоритмов, доказательство невозможности должно содержать высказывание обо всех мыслимых алгоритмах. Без формального определения невозможно было говорить о всех возможных алгоритмах и, следовательно, доказать, что задача алгоритмически неразрешима.

1.4 Основные типы алгоритмических моделей

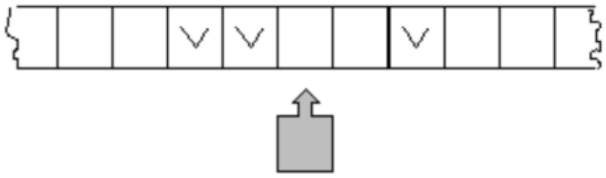
Алгоритм как некое детерминированное устройство - абстрактные машины. Машина Тьюринга и машина Поста.

Алгоритм как процедура вычисления некой числовой функции. Рекурсивные функции Черча.

Алгоритм как последовательность преобразований цепочек в каком-либо алфавите. (Комбинаторные операции над словами). Нормальные алгоритмы Маркова.

2 Машина Поста. Описание, примеры. Тезис Поста

2.1 Машина Поста



Машина Поста — четвёрка $M = (T, H, P, S_0)$, где

- T — бесконечная лента, разделённая на ячейки, в каждой из которых может быть или отсутствовать метка
- H — каретка, способная перемещаться по ленте, читать и писать
- P — программа из конечного множества пронумерованных команд
- S_0 — начальное состояние, определяемое положением каретки

Команды машины Поста:

$\rightarrow$	Шаг вправо
$\leftarrow$	Шаг влево
v	Записать отметку
$\times$	Стереть отметку

$?\,a : b$	`если метки нет {`
	`  goto a // a - номер команды`
	`} else {`
	`  goto b // b - номер команды`
	`}`

!	Стоп

2.2 Описание, примеры

Инструкции программы занумерованы и выполняются последовательно, оператор условного перехода ? осуществляет перемещение к конкретному номеру. Программа может завершить работу тремя путями:

Останов по команде ! называется результативным и указывает на корректность алгоритма

Останов при выполнении недопустимой команды (запись метки, где она есть, или стирание, где её нет)

Зацикливание

Пример: переместим головку к первой пустой позиции справа 1. -> 2 2. ? 3;

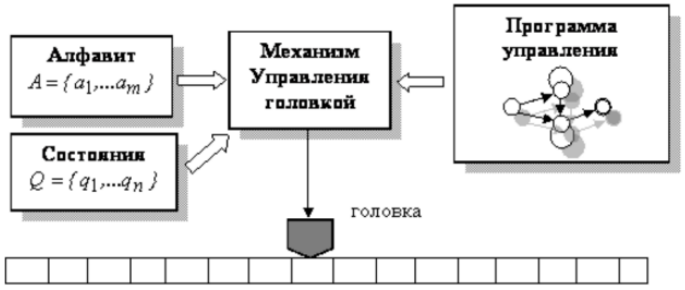
2.3 Тезис Поста

Всякий алгоритм представим в форме машины Поста — Эмиль Леон Пост Потому что алгоритм по Посту — это программа для машины Поста (с)

3 Машина Тьюринга. Описание, способы задания (список команд, таблица переходов, граф переходов). Конфигурации МТ. Вычислимость по Тьюрингу.

3.1 Машина Тьюринга

3.1 Машина Тьюринга



Машина Тьюринга — семёрка $M = (Q, \Sigma, A, \delta, q_0, F, \varepsilon)$, где:

- Q — конечное множество состояний управляющего устройства (внутренний алфавит)
- $\Sigma \subset A$ — множество входных символов ленты, подмножество внешнего алфавита A
- A — внешний алфавит, конечное множество допустимых символов ленты
- $\delta : Q \times A \to Q \times A \times \{L, R, E\}$ — функция переходов.
  - Запись $q_i a_i \to q_j a_m d$, где $q_i, q_j \in Q, a_i, a_m \in A, d \in \{L, R, E\}$ означает: машина из состояния q_i при прочтении с ленты символа a_i должна перейти в состояние q_i, записать символ a_m и сдвинуться влево, вправо или никуда в зависимости от d
- $q_0 \in Q$ — начальное состояние
- $F \subseteq Q$ — множество заключительных состояний
- $\varepsilon \in A \setminus \Sigma$ — пустой символ

3.2 Описание, способы задания (список команд, таблица переходов, граф переходов)

3.2 Способы задания МТ

1. Список команд

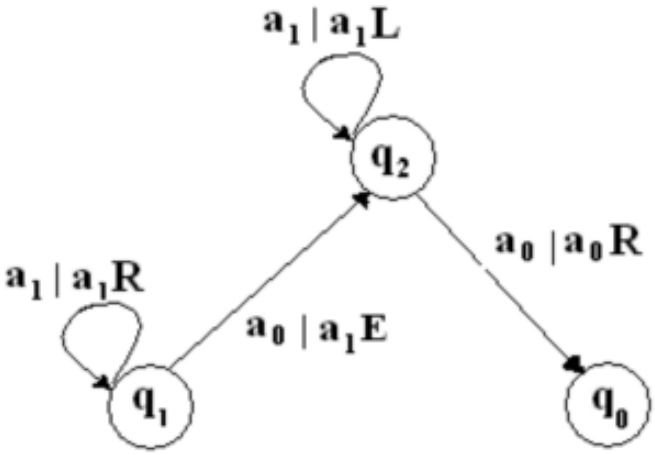
$A = \{a_0 = \lambda, a_1 = 1\}$ и $Q = \{q_0, q_1, q_2\}$

- $q_1\lambda \to q_2 1E$
- $q_1 1 \to q_1 1R$
- $q_2 1 \to q_2 1L$
- $q_2\lambda \to q_0 \lambda R$

2. Таблица переходов

	a_0	a_q
q_1	$q_2 a_1 E$	$q_1 a_1 R$
q_2	$q_0 a_0 R$	$q_2 a_2 L$

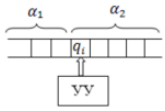
Граф переходов



3.3 Конфигурации машины Тьюринга

Конфигурация совокупность внутреннего состояния, состояния ленты, положения головки на ленте.

Визуально её можно представить следующим образом:



Машина обозревает первый символ слова α_2и слева написано слово α_1 МТ находится в состоянии q_i.

Также конфигурацию можно записать машинным словом $\alpha_1 q_i \alpha_2$ в алфавите $A \cup Q$.

Машина Тьюринга реализует процесс изменения конфигураций: начальная конфигурация K_0 порождает последовательность конфигураций $K_0 \to K_1 \to ... \to K_i \to K_i + 1 \to ....$

Машина **применима** к конфигурации $K_0 \overset{\text{df}}{\Leftrightarrow}$ последовательность конфигураций конечна.

В противном случае машина **неприменима**.

В **стандартном виде** начальная конфигурация имеет вид $K_0 = q_1\alpha$, равно как заключительная $- K_f = q_0\alpha$.

Привести конфигурации к стандартному виду можно, добавив дополнительные состояния и команды, приводящие положение головки к стандартному виду. В результате мы получим новую машину Тьюринга, которая будет эквивалентна исходной.

3.4 Вычислимость по Тьюрингу

3.4 Вычислимость по Тьюрингу

$]f : A^* \to A^*$

Определение. Машина Тьюринга M **вычисляет** $f \overset{\text{df}}{\Leftrightarrow} \forall \alpha \in A^*$ выполняется:

$$f(\alpha) = \begin{cases} \beta, \text{ если } M(\alpha) = \beta, \text{ т. е. } (K_0 = q_1\alpha) \wedge (K_1 = q_0\beta) \\ \text{не определена, если } M(\alpha) \text{ не останавливается} \end{cases}$$

Определение. f **вычислима по Тьюрингу** $\overset{\text{df}}{\Leftrightarrow} \exists M$ — машина Тьюринга, правильно вычисляющая f

4 Введение в теорию рекурсивных функций. Простейшие функции. Оператор суперпозиции. Оператор примитивной рекурсии. Оператор минимизации. Примитивно-рекурсивные и частично-рекурсивные функции. Тезис Черча. Примеры и способы задания рекурсивных функций.

4.1 Простейшие функции

Числовые фукнции, значение которых можно установить посредством некоторого алгоритма, называются вычислимыми функциями.

Пример:

s(x) = x + 1 - функци непосредственного следования, также опрелённая для всех целых нотрицательных начений своего аргуумента;

Z($x_1, x_2, ...x_n$) = 0 нуль-функция, которая определена для всех неотрицательных значений аргумента;

4.2 Оператор суперпозиции

Оператором суперпозиции S называется подстановка в функцию от n переменных m функций от n одних и тех же переменных. Она дает новую от n переменных.

В операции суперпозиции S^{m+1} индекс сверху указывает на число функций.

Таким образом, при помози оператора суперпозиции и функции выбора можно выразить любую подстановку функции в функцию.

Свойства операции суперпозиции:

1.Операция суперпозиции сохраняет свойство всюду определенности функций

2.Операция суперпозиции сохраняет свойство алгоритмической вычислимости функций.

Примеры суперпозиции функции

f(x) = 0 и g(x) = x + 1 получим фунцию h(x) = g(f(x)) = 0 + 1 = 1

4.3 Оператор примитивной рекурсии

Оператор примитивной рекурси задается следующим образом: Рекурсия ведется по одному аргументу, все остальные считаются параметрами

Функция полученная с помощью операции примитивной рекурсии

f = R(g,h)

R - оператор примитивной рекурсии.

4.4 Основные свойства операции примитивной рекурсии.

Всякая ПРФ является всюду определённой функцией. Всякая ПРФ является алгоритмически вычислимой.

Определение. Функция примитивно-рекурсивная если она является элементарной или может быть получена из элементарных функций с помозью конечного числа применений операторов тождества, суперпозиции и примитивной рекурсии.

Если некоторые функции являются примитивно-рекурсивными, то в результате применения к ним операторов суперпозиции или примитивной рекурсии можно получить новые ПРФ

Существует три возможности доказательства того, что функция является примитивнорекурсивной:

1. Показать, что заданная функция является простейшей.

2. Показать, что заданная функция построена с помощью оператора суперпозиции.

3. Показать, что заданная функция построена с помощью оператора примитивной рекурсии.

4.5 Оператор минимизации

Оператор минимизации M (μ-оператор) Пусть задана некоторая частичная функция $g(z, \overline{x})$. В результате применения оператора минимизации мы получаем новую функцию, которая вычисляется следующим образом:

$$f(\overline{x}) = y \Leftrightarrow \forall i < y \ (g(i, \overline{x}) \ \text{ опр.} \wedge g(i, \overline{x}) \neq 0) \wedge (g(y, \overline{x}) = 0)$$

μ-оператор записывается следующим образом:

$$f(\overline{x}) \simeq \mu y(g(y, \overline{x}) = 0)$$

Короче говоря, $f(\overline{x})$ — минимальное y такое, что $g(y, \overline{x}) = 0$.

4.6 Тезис Чёрча

Определение. Частично-рекурсивная функция — функция, которая может быть получина из простейших функции с помощью конечного числа операторов суперпозиции, примитивной рекурсии и минимизации. Определение. Общерекурсивная функция — всюду определённая частично-рекурсивная функция

Тезис Чёрча (для частично рекурсивных функций) Класс алгоритмически вычислимых функций совпадает с классом всех частично рекурсивных функций. Принятие данного тезиса позволяет истолковывать доказательство, что некоторая функция не является частичо рекурсивной, как доказательство отсутствия алгоритма вычисления ее значений

Всякая функция, вычислимая с помощью машины Тьюринга, является частично рекусивной. Всякая частично рекурсивная функция вычислима на машине Тьюринга

4.7 Примеры и способы задания рекурсивных функций

Рекурсивная функция может быть задана:

Прямо, когда даётся базовый и рекурсивный случай

Косвенно, когда две функции вызывают друг друга

5 Нормальные алгоритмы Маркова.

Класс нормально вычислимых функций совпадает с классами частично рекурсивных функций и вычислимых по Тьюрингу

Определение. Марковская подстановка — операция над словами, задаваемая парой слов (P, Q), которая заменяет в слове R первое вхождение P на Q, обозначается P -> Q. Пустое слово обозначается как $\wedge$.

Определение. Схема нормального алгоритма Маркова — это упорядоченный конечный список подстановок в алфавите A

Нормальный алгоритм перерабатывает слово V в слово W.

Принцип нормализации Маркова — функция, заданная в некотором алфавите, нормально вычислима $\rightarrow \exists$ алгоритм нахождения её значений

6 Алгоритмически неразрешимые задачи. Примеры алгоритмически неразрешимых задач. Методы доказательства алгоритмической неразрешимости. Проблема останова МТ и самоприменимости (доказательство неразрешимости).

6.1 Алгоритмически неразрешимые задачи

Для алгоритмически невычислимых функций не существует алгоритма решения. Это означает невозможность решения всех задач одного класса одним и тем же приёмом, при этом конкретные задачи могут быть решаемыми.

6.2 Примеры алгоритмически неразрешимых задач

Пример. h(n) возвращает 0 или 1 - найдутся ли в числе π n идущих подряд девяток. Функция алгоритмически неразрешима, так как число π иррационально и по природе числа неизвестно, существует ли решение для всех n

6.3 Методы доказательства алгоритмической неразрешимости

Прямой метод использует диагональный метод Кантора. Заключается он в следующем: из предполоожения о разрешимости данной проблемы в ходе рассуждений приходят к противоречию.

Косвенный метод состоит в следующем: показывается, что разрешимость исследуемой проблемы влечёт разрешимость проблемы, о которой уже известно, что она неразрешима. Метод сведения часто бывает более удобным, чем прямой метод. Применяя метод сведения, обычно ссылаются на искусственные задачи, которые не представляют самостоятельного интереса, но для которых легко непосредственно доказать их неразрешимость

6.4 Проблема останова МТ в самоприменимости (доказательство неразрешимости)

Определение. Самоприменимость — частный случай проблемы остановки. Означает свойство алгоритма успешно завершаться на данных, представляющих собой формальную запись этого же алгоритма

Пример. Тождественнные преобразования строк в алфавите A — самоприменимый алгоритм.

Теорема о проблеме останова МТ. ∄ алгоритма, способного по описанию произвольного алгоритма и входных данных, определить, остановится ли алгоритм на этих данны

7 Трудоемкость алгоритма. Функция трудоемкости. Классификация алгоритмов по виду функции трудоемкости.

7.1 Трудоёмкость алгоритма и её функция

Определение. Трудоёмкость алгоритма — количество элементарных операций, совершаемых алгоритмом для решения конкретной проблемы, заданной N + M + Sd + Sr словами в памяти, где

N — количество слов для описания проблемы

M — программа-решение из M машинных инструкций

Sd — память для хранения промежуточных результатов

Sr — память для организации вычислительного процесса

Определение. Функция трудоёмкости $T_a(\mathrm{N})$ — отношение, связывающее входные данные алгоритма с количеством элементарных операций. Комплексный анализ алгоритма может быть выполнен на основе комплексной оценки ресурсов формальной системы, требуемых алгоритмом для решения конкретных проблем. Очевидно, что для различных областей применения веса ресурсов будут различны, что приводит к следующей комплексной оценке алгоритма:

7.2 Классификация алгоритмов по виду функции трудоёмкости

7.2.1 Количественно-зависимые

Алгоритмы, T_a которых зависит только от размерности входа:

$$T_a(D) = T_a(|D|) = T_a(N)$$

Пример: умножение матриц

7.2.2 Параметрически-зависимые

T_a зависит от конкретных значений обрабатываемых слов:

$$T_a(D) = T_a(d_1, ..., d_n) = T_a(P_1, ..., P_m),\ m \leq n$$

Пример: рекурсивное вычисление x^k

7.3 Количественно-параметрические

Комбинация первых двух вариантов:

$$T_a(D) = T_a(N, d_1, ..., d_n) = T_a(N, P_1, ..., P_m),\ m \leq n$$

7.4 Порядко-зависимые

Количество операций зависит от порядка расположения исходных объектов.

$$T_a\left(D_{p_i}\right) \neq T_a\left(D_{p_j}\right)$$

Пример: алгоритмы поиска минимума и максимума в массиве

8 Асимптотический анализ сложности алгоритмов; сравнение наилучших, средних и наихудших оценок; O-, o-, $\omega-$ и θ-нотации. Свойства.

8.1 O-, Theta-, Omega-нотации

8.1 O-, Theta-, Omega-нотации

1. Оценка $O(g(n))$ (О большое) — функции, растущие медленне чем g

$$\exists c > 0,\ n_0 > 0 :\ \forall n > n_0\ 0 \leq T(n) \leq cg(n)$$

Вобще, запись $O(g(n))$ обозначает класс функций таких, что все они растут не быстрее, чем функция $g(n)$ с точностью до постоянного множителя, поэтому иногда говорят, что $g(n)$ мажорирует функцию $T(n)$.

2. Оценка $\Theta(g(n))$ (тетта) — функции, растующие с той же скоростью, что и g.

Пусть $T(n)$ и $g(n)$ — положительные функции положительного аргумента, $n \geq 1$, c_1, c_2 — некоторые константы.

$$c_1 \cdot g(n) \leq T(n) \leq c_2 \cdot g(n)$$

Обозначение: $T(n) = \Theta(g(n))$.

Читается как "Тэта большое от g от n".

3. Оценка Ω (омега) — функции, растущие быстрее g

В отличие от оценки O, оценка Ω является оценкой снизу — т.е. определяет класс функций, которые растут не медленнее, чем $g(n)$ с точностью до постоянного множителя:

$$\exists c > 0, n_0 > 0 : 0 \leq cg(n) \leq t(n)\ \forall n > n_0$$

Функции, обозначающиеся одними и теми же буквами:
- В верхнем регистре накладывают нестрогое ограничение
- В нижнем — строгое

8.2 Свойства асимтотических оценок

Транзитивность:

$$T(n) = \Theta(g(n)) \text{и} g(n) = \Theta(h(n)) \Rightarrow T(n) = \Theta(h(n))$$

Рефлексивность:

$$T(n) = \Theta(T(n))$$

Симметричность:

$$T(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$$

Для O и Ω выполняются аналогичные свойства.

9 Порядок роста. Основные классы эффективности

9.1 Порядок роста

Вид f(n)	**Характеристика класса алгоритмов**
1	Большинство инструкций большинства функций запускается один или несколько раз. Если все инструкции программы обладают таким свойством, то время выполнения программы постоянно.
$\log N$	Когда время выполнения программы является логарифмичесим, программа становится медленнее с ростом N. Такое время выполнения обычно присуще программам, которые сводят большую задачу к наборе меньших подзадач, уменьшая на каждом шаге размер задачи на некоторый постоянный фактор.
N	Когда время пвыолнения программы является линейным, это обычно значит, что каждый входной элемент подвергается небольшой обработке.
$N \log N$	Возникает тогда, когда алгоритм решает задачу, разбивая её на меньшие подзадачи, решая их независимо и затемм объединяя решения
N^2	Когда время выполнения алгоритма является квадратичным, он полезен для практического использования при решении относительно небольших задач квадратичное время выполеия обычно появляется в алгоритмах, которые обрабатывают все пары
N^3	Похожий алгоритм, который обрабатывает тройки элементов данных (возможно, в цикле тройного уровня вложенности)
2^n	Лишь несколько алгоритмов с экспоненциальным временем выполнения имеет практическое примеение, хотя такие алгоритми возникюат естественным образом при попытках прямого решения задачи, например полного перебора. Время выполнения увеличивается экспоненциально с ростом N
$N!$	

9.2 Основные классы эффективности

9.2.1 Класс 0 Это класс быстрых алгоритмов с постоянным временем выполенния из функция трудомекости O(1). Промежуточное состояние занимают алгоритмы со сложностью O(log n)

9.2.2 Класс P

Задачи со сложностью O(1):

вставка и удаление элемента в односвязном и двусявязном списке

добавление вершины или ребра в графе

Задачи со соложностью O(log n):

Двоичный поиск в линейном упорядоченном массиве;

Задачи с полиномиальной сложностью:

Задача сортировки

Задача поиска эйлерова цикла на графе

9.2.3 Класс E

Класс экспоненциальных алгоритмов со степенью трудоёмкости O(2 n)

Задача коммивояжёра

9.2.4 Класс F

Класс надэкспоненциальных алгоритмов с факториаьлной трудоёмкостью

10 Класс NP. Проблема равенства P и NP. Класс NPC

10.1 Класс NP

Класс NP включает в себя четыре вида задач:

Задачи, которые нельзя отнести ни к классу P, ни к классу E

Задачи, которые недетерменированная машина Тьюринага может решить за полиномиальное время, тогда как для детерменированной машины Тьюринга полиномиальный алгоритм неизвестен.

Для этих задач до сих пор не разрабботан эффективный (т. е. полиномиальный) алгоритм, но и не доказано, что таких алгоритмов не существует.

К классу NP относятся все задачи, решение которых можно проверить за полиномиальное время. Проверка осуществляется следующим образом: абстрактная сущность — оракул — предлагает решения, которые после проверки верификатором за полиномиальное время приобретают “юридическую” силу

10.2 Проблема равенства P и NP

Поскольку детерминированная машина Тьюринга может рассматриваться как специальный случай недетерминированной машины Тьюринга, в которой отсутствует стадия угадывания, а стадия проверки совпадает с ДМТ,

класс NP включает в себя класс P, а также некоторые проблемы, для решения которых известны лишь алгоритмы, экспоненциально зависящие от размера входа (то есть неэффективные для больших входов)

Вопрос о равенстве этих двух классов считается одной из самых сложных открытых проблем в области теоретической информатики.

Доказательств нет, но считается, что p $\subset$ NP

10.3 Класс NPC

называется NP-полной (NPC).

Теорема. $\nexists$ полиномиально решаемая NPC-задача -> NPC = NP

11 Анализ нерекурсивных алгоритмов по фактическому количеству элементарных операций. Анализ трудоемкости основных алгоритмических конструкций.

При анализе нерекурсинвых алгоритмов важно понимать, что они состоят из трёх основных конструкций:

1. Следование, когда имеет место линейная последовательность. Трудоёмкость вычисляется как сумма трудоёмкостей составных блоков: T = t1 + t2 + ...

2. Ветвление, когда имеет место условное выполнение одного из нескольких блоков кода. Оператор if можно представить в виде T = p * t_{then} + (1 - p) * t_{else}, где p - вероятность истинности условия.

3. Цикл, когда имеет место многократное выполнение одного и того же блока кода. В таком случае трудоёмкость этого блока умножается на количество итераций. Блоки могут представлять собой либо эти же конструкции (строя таким образом вложенность), либо элементарные операции, которыми считаются:

1. Присваивание
2. Одномерная индексация
3. Арифметические операции
4. Сравнение
5. Логические операции

Трудоёмкость в данном подходе вычисляется как общее количество элементарных операций.

12 Оценивание порядков роста времени работы нерекурсивных алгоритмов. Правило суммы и произведения. Оценка основных алгоритмических конструкций.

12.1 Правило суммы и произведения

Правило сумм. T1(n) + T2(n) = O(f(n)) + O(g(n)) = O(max(f(n), g(n)))

Правило произведений. T1(n) * T2(n) = O(f(n) * g(n))

12.2 Оценка основных алгоритмических конструкций

Элементарные операции выполняются за O(1)

Последовательность операций a, b, c выполняется за O(max(a, b, c))

Условие выполняется за O(max(then, else))

Цикл из n итераций над a выполняется за O(n * a)

13 Анализ рекурсивных алгоритмов. Рекуррентные соотношения. Метод индукции. Метод подстановки.

13.1 Рекуррентные соотношения

Анализ сложности состоит в определении трудоёмкости. Определение. Рекуррентное соотношение — это уравнение или неравенство, описывающее зависимость следующих значений от предыдущих.

13.2 Метод индукции

я его анализа по методу математической индукции мы находим мажорирующую функцию f(n). Для этого мы предполгаем, что в базовом случае для достаточно большого n_0 неравенство выполняется, и выводим индуктивный шаг

13.3 Метод подстановки.

Метод подстановки состоит из трёх шагов:

1. Делается догадка о виде решения

2. С помощью метода математической индукции доказывается, что решение правильное

3. Вычисляются константы

14 Анализ рекурсивных алгоритмов. Рекуррентные соотношения. Первая основная теорема. Следствие.

Первая основная теорема. Пусть дано рекурсивное соотношение

$$t(n) = \begin{cases} c, \text{если } n = 1 \\ at\left(\frac{n}{k}\right) + bn^{\tau}, \text{если } n > 1 \end{cases}$$

где $a > 0, k > 1$ — целые константы, $b \geq 0, c \geq 0, \tau \geq 0 \in \mathbb{R}$.

Тогда при $n = k^m, m \in Z^+$ решением соотношения является функция

$$t(k^m) = \begin{cases} a^m c + bk^{m\tau} m, \text{если } a = k^{\tau} \\ a^m c + bk^{m\tau} \frac{\left(\frac{a}{k^{\tau}}\right)^m - 1}{\left(\frac{a}{k^{\tau}}\right) - 1}, \text{если } a \neq k^{\tau} \end{cases}$$

Следствие. В предположениях первой основной теоремы при больших значениях n и $\forall b > 0, c \geq 0$ справедливы оценки

$$t(n) = \begin{cases} O(n^{\tau} \log_k n), \text{если } a = k^{\tau} \\ O(n^{\tau}), \text{если } a < k^{\tau} \\ O(n^{\log_k a}), \text{если } a > k^{\tau} \end{cases}$$

При $b = 0, c > 0$:

$$t(n) = O\left(a^{\log_k n}\right) = O\left(n^{\log_k a}\right)$$

15 Математический анализ рекурсивных алгоритмов. Рекуррентные соотношения. Вторая основная теорема. Следствия.

Вторая основная теорема. Пусть дано реккурентное соотношение

$$t(n) = \begin{cases} c, & \text{если } 0 \leqslant n \leqslant k-1, \\ at(n-k) + bn^{\tau}, & \text{если } n \geqslant k, \end{cases} \tag{1}$$

где $a > 0, k \geqslant 1$ — целые константы, $b, c, \tau \geqslant 0$ — вещественные константы.
Тогда при $n = km,\ m \in \mathbb{Z}_+$ верны неравенства:

$$c + bk^{\tau-1}n \leqslant t(n) \leqslant c + \frac{b}{k}n^{\tau+1}, \quad \text{если } a = 1,$$

$$a^{n/k}c + bk^{\tau}\frac{a^{n/k}-1}{a-1} \leqslant t(n) \leqslant a^{n/k}c + bn^{\tau}\frac{a^{n/k}-1}{a-1}, \quad \text{если } a \neq 1.$$

Следствие. В предположениях второй теоремы при $\tau = 0$ решением рекуррентного соотношения (1) является функция

$$t(n) = \begin{cases} c + \frac{b}{k}n, & \text{если } a = 1, \\ a^{n/k}c + b\frac{a^{n/k}-1}{a-1}, & \text{если } a \neq 1. \end{cases}$$

Следствие. При $\tau = 0,\ \forall c \geqslant 0,\ n \to \infty$ для решения реккурентного соотношения (1) верны асимптотические оценки:

$$t(n) = \begin{cases} O(n), & \text{если } a = 1,\ b > 0, \\ O(a^{n/k}), & \text{если } a \neq 1,\ b > 0. \end{cases}$$

В частности, $\tau = b = 0,\ a = 1 \Rightarrow\ t(n) \equiv O(1)$.

Следствие. При $\tau, c \geqslant 0,\ b > 0,\ a = 1,\ n \to \infty$ для решения реккурентного соотношения (1) верна асимптотическая оценка:

$$t(n) = O(n^{\tau+1})$$

Следствие. При $c \geqslant 0,\ b > 0,\ a > 1,\ t \in \mathbb{Z}_+,\ n \to \infty$ для решения реккурентного соотношения (1) верна асимптотическая оценка:

$$t(n) = O(a^{n/k})$$

16 Математический анализ рекурсивных алгоритмов. Анализ дерева рекурсии.

Вторая основная теорема. Пусть дано реккурентное соотношение

$$t(n) = \begin{cases} c, & \text{если } 0 \leqslant n \leqslant k-1, \\ at(n-k) + bn^\tau, & \text{если } n \geqslant k, \end{cases} \tag{1}$$

где $a > 0, k \geqslant 1$ — целые константы, $b, c, \tau \geqslant 0$ — вещественные константы.
Тогда при $n = km,\ m \in \mathbb{Z}_+$ верны неравенства:

$$c + bk^{\tau-1}n \leqslant t(n) \leqslant c + \frac{b}{k}n^{\tau+1}, \quad \text{если } a = 1,$$

$$a^{n/k}c + bk^\tau \frac{a^{n/k}-1}{a-1} \leqslant t(n) \leqslant a^{n/k}c + bn^\tau \frac{a^{n/k}-1}{a-1}, \quad \text{если } a \neq 1.$$

Следствие. В предположениях второй теоремы при $\tau = 0$ решением рекуррентного соотношения (1) является функция

$$t(n) = \begin{cases} c + \frac{b}{k}n, & \text{если } a = 1, \\ a^{n/k}c + b\frac{a^{n/k}-1}{a-1}, & \text{если } a \neq 1. \end{cases}$$

Следствие. При $\tau = 0,\ \forall c \geqslant 0,\ n \to \infty$ для решения реккурентного соотношения (1) верны асимптотические оценки:

$$t(n) = \begin{cases} O(n), & \text{если } a = 1,\ b > 0, \\ O(a^{n/k}), & \text{если } a \neq 1,\ b > 0. \end{cases}$$

В частности, $\tau = b = 0,\ a = 1 \Rightarrow\ t(n) \equiv O(1)$.

Следствие. При $\tau, c \geqslant 0,\ b > 0,\ a = 1,\ n \to \infty$ для решения реккурентного соотношения (1) верна асимптотическая оценка:

$$t(n) = O(n^{\tau+1})$$

Следствие. При $c \geqslant 0,\ b > 0,\ a > 1,\ t \in \mathbb{Z}_+,\ n \to \infty$ для решения реккурентного соотношения (1) верна асимптотическая оценка:

$$t(n) = O(a^{n/k})$$

17 Эмпирический анализ алгоритмов.

Эмпирический анализ алгоритмов — это метод оценки эффективности алгоритма на практике путём проведения экспериментов и измерения его производительности на различных входных данных. В отличие от теоретического (математического) анализа, который даёт асимптотические оценки сложности, эмпирический анализ показывает реальное поведение алгоритма в конкретных условиях.

Основные этапы эмпирического анализа алгоритмов:

Определение цели эксперимента. Например, сравнить скорость работы нескольких алгоритмов или проверить гипотезу о сложности.

Выбор метрик измерения. Обычно измеряют время выполнения, коли-

чество операций, использование памяти.

Подготовка входных данных. Важно учитывать разные типы данных: случайные, упорядоченные, худшие случаи, реальные данные.

Реализация алгоритма и запуск тестов. Желательно использовать одинаковые условия для всех алгоритмов (одинаковое оборудование, язык программирования, компилятор).

Сбор и анализ результатов. На основе собранных данных строят графики, вычисляют средние значения, делают выводы.

18 Основные методы построения алгоритмов. Алгоритмы «Разделяй и властвуй». Примеры.

19

20

21

22

23

24

25

26

27