ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ

ПЛАН ЛЕКЦИИ

1. Принципы анализа алгоритмов
2. Допущения, принятые при проведении анализа
3. Формальная классификация входных данных
4. Функции трудоёмкости алгоритмов
5. Асимптотический анализ. Виды оценок
6. Асимптотический анализ. Методы
7. Замечания

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
1. Принципы анализа алгоритмов (1)

1. **Инвариантность проводимого анализа**

Независимость от реализации на конкретной ЭВМ

2. **Анализу подвергаются алгоритмы, решающие одну и ту же проблему**

Особенности "диалектов" языков высокого уровня типа *Pascal*, *C++*, *Java* и др. при этом не учитываются , поскольку структуры операторов управления функционально одинаковы: циклы, операторы ветвления et c.

3. **Алгоритмы анализируются на классах данных**

Пример – алгоритм выбора максимума из N чисел.

```
big = list [1]
for i = 2 to N
   if  (list [i] > big) then big = list [i]
end for
```

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
1. Принципы анализа алгоритмов (2)

Если список ***list*** упорядочен по убыванию, присваивание будет 1, а если по возрастанию, то N.

При $N = 10 \Rightarrow 10! = 3\,628\,800$ возможных комбинаций, из которых 362 880 случаев, когда максимум 1-й в последовательности, столько же, когда 2-й и т.д.

На каждом из классов алгоритм покажет свои характеристики.

Важно! *Если классы построены правильно*, то *характеристики* алгоритма при работе *на классе будут неизменны.*

4. **Учёт ресурса типа "память"**

Недавняя проблема 2000: если необходимо хранить много дат, то два числа – балласт (так считалось в 50-е годы XX века).

Оптимизация по занимаемой программой памяти и по быстродействию программы (α и ω).

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
1. Принципы анализа алгоритмов (3)

5. **Поиск наилучшего, наихудшего и среднего случаев**

Наилучший случай – ситуация, для которой обработка некоторого класса данных даёт самый лучший показатель в рамках выбранной оценки.

Обычно тривиален.

Наихудший случай – ситуация, для которой обработка некоторого класса данных даёт самый плохой показатель в рамках выбранной оценки.

Позволяет определить максимальное время работы алгоритма.

Средний случай – результат статистической обработки показателей, полученных для разных классов данных

$$A(n) = \sum_{i=1}^{m} P_i(n) \cdot t_i(n),$$

n – размер входного данного, m – число классов данных, $P_i(n)$ – вероятность принадлежности данных к группе (классу), t_i – время необходимое для решения задач i-го класса.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
2. Допущения, принятые при проведении анализа (1)

1. **В качестве вычислительного устройства**, реализующего алгоритм, принимается абстрактная машина с процессором фон-Неймановской архитектуры (*John von Neumann*), которая поддерживает произвольный доступ к памяти и множество элементарных операций, соответствующий набору конструкций языков “высокого уровня”
2. **Каждая команда** выполняется не более соответствующего фиксированного времени
3. **Исходные данные** представляются в виде машинных слов по β (байт/бит) каждое.

   Конкретный входной набор задаётся N словами памяти, отсюда объём на входе алгоритма $N_\beta = N \times \beta$.

   Часто используется термин *длина входа алгоритма N,* который отражает линейную размерность входных данных

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
2. Допущения, принятые при проведении анализа (2)

4. **Программный код представляется последовательностью машинных слов**

   Программный код, по аналогии с входными данными, характеризуется величиной $M_\beta = M \times \beta_M$, где M – число машинных команд (инструкций), а β_M – размер памяти, отводимой под хранение отдельной инструкции.

5. **Факультативно** (опционально) **учитываются дополнительные ресурсы**
- S_d – объём памяти для хранения промежуточных данных;
- S_r – объём памяти, необходимой для организации вычислительного процесса (стеки, память для организации рекурсивных вызовов и возвратов, область буферов для осуществления операций ввода вывода et c.).

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
2. Допущения, принятые при проведении анализа (3)

6. **Комплексный показатель оценки алгоритма**

$$\Psi_A = c_1 \cdot F_A(N) + c_2 \cdot M_\beta + c_3 \cdot N_\beta + c_4 \cdot S_d + c_5 \cdot S_r,$$

где $F_A(N)$ – трудоёмкость алгоритма, c_i, $i = 1 \ldots 5$ – веса, приданные отдельным показателям.

Трудоёмкость алгоритма $F_A(N)$ – количество "элементарных" операций, совершаемых алгоритмом для решения конкретной проблемы в данной формальной системе.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
3. Формальная классификация входных данных (1)

Количество операций, выполняемых над входом одинаковой длины N, зависит от конкретного набора данных (см., например, алгоритм выбора максимума из N чисел).

Пусть D_A – множество наборов данных (проблем), заданное в формальной системе.

$D_N \in D_A$ – конкретный набор входных данных, $|D_N| = N$.

$D_N = \{D \in D_A : |D_N| = N\}$

$M_{DN} = |D_N|$ – мощность множества D_N.

Алгоритм, решая различные задачи для размерности N, выполнит разное число операций:

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
3. Формальная классификация входных данных (2)

$F_a^{\wedge}(N)_{D \in D_N} = \max\{F_a(D)\}$ – худший случай для D_N;

$F_a^{\vee}(N)_{D \in D_N} = \min\{F_a(D)\}$ – наилучший случай для D_N;

$\overline{F}_a(N) = \frac{1}{M_{DN}} \sum_{D \in D_N} \{F_a(D_N)\}$ – средний случай или среднее число операций, совершаемых алгоритмом A для решения задач размерностью N.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
4. Функции трудоёмкости алгоритмов (1)

Функции трудоёмкости есть результаты аппроксимации зависимостей трудоёмкости $F_a(N)$ от длины входа алгоритма. Классификация функций трудоёмкости организована по виду входа.

1. **Количественно-зависимые по трудоёмкости алгоритмы** (класс N)

Для данной категории алгоритмов функция трудоёмкости *зависит только от объёма* входных данных, а не от их конкретных значений.

$$F_a(D) = F_a(|D|) = F_a(N).$$

Примеры:

- стандартные алгоритмы для работы с матрицами, векторами, массивами;
- операции ввода/вывода;
- вычисление статистических показателей выборок и т.д.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
4. Функции трудоёмкости алгоритмов (2)

2. **Параметрически-зависимые по трудоёмкости алгоритмы** (класс *PR*)

Размерность, как правило, *фиксирована*, а трудоёмкость *определяется конкретными значениями* компонентов вектора данных. Это, например алгоритмы расчётов математических функций на базе степенных рядов с заданной точностью

$$F_a(D) = F_a(d_1, d_2, ..., d_n) = F_a(p_1, p_2, \ldots, p_m)\ ,\ m \le n,$$

n – длина ввода, m – число параметров, оказывающих влияние на трудоёмкость.

Примеры

- возведение в степень k последовательными умножениями $F_a(x, k) = F_a(k)$, $n = 2$, $m = 1$;
- вычисление тригонометрической функции с точностью ε : $F_a(x, \varepsilon) = F_a(D)$, $n = 2$, $m = 2$

$$\sin x = \sum (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!};$$

- вычисление факториала $F_a(k) = F_a(k)$, $n = 1$, $m = 1$.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
4. Функции трудоёмкости алгоритмов (3)

3. **Количественно-параметрические по трудоёмкости алгоритмы**
(класс *NPR*: *NPRL – Low*, *NPRE – Equivalent, NPRH – High*)

Алгоритмы данного класса имеют функцию трудоёмкости, которая одновременно зависит как от количества (объёма) данных, так и от конкретных значений элементов данных.

$$F_a(D) = \mathrm{F_a}(|\mathrm{D}|, P_1, P_2, \ldots, P_m) = \mathrm{F_a}(N, P_1, P_2, \ldots, P_m)$$

Примером служат алгоритмы численных методов :

- имеется внешний цикл по точкам аргумента, который зависит от их количества;
- внутри цикла присутствует процедура точного расчёта (или вложенный цикл), являющийся параметрически зависимым.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
4. Функции трудоёмкости алгоритмов (4)

4. **Порядково-зависимые по трудоёмкости алгоритмы** (класс *NPRS*)

Для данных алгоритмов *функция* трудоёмкости *определяется порядком следования* данных в потоке.

Пусть множество D представлено из элементов ($d_1, d_2, \dots, d_n$) и $|D| = N$.

На основании D определим множество D_P всех упорядоченных последовательностей из N членов на ($d_1, d_2, \dots, d_n$), для которого $|D_P| = n!$:

$$D_P = \{{}_1D_P, {}_2D_P, \dots, {}_iD_P, \dots, {}_iD_P, \dots\}$$

Определение. Если $F_a({}_iD_P) \neq F_a({}_jD_P)$, где $F_a({}_iD_P) \in D_P$ и $F_a({}_jD_P) \in D_P$, то алгоритм является *порядково-зависимым* по трудоёмкости.

Примеры:

- алгоритмы поиск минимального (максимального) значений в массиве;
- алгоритмы сортировки и упорядочивания et c.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
5. Асимптотический анализ. Виды оценок (1)

Асимптотический анализ применяется для оценки (прогноза) количества операций, лежащих в основе определения сложности.

Целью является *оценка скорости роста* числа операций *при возрастании объёма* водных данных.

Синонимы:

- оценка скорости роста;
- скорость роста алгоритма;
- сложность алгоритма.

Пусть $f(n)$ и $g(n)$ *положительные* функции *положительного* аргумента, $1 \leq n$.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ

5. Асимптотический анализ. Виды оценок (2)

1. Оценка снизу $\boldsymbol{\Omega}$ (Омега большое)

$f(n) = \Omega[g(n)]$ или $f(n) \sim\geq g(n)$, если существуют целые N и C, такие, что

$$f(n) \geq C \cdot g(n) \text{ для всех } n \geq N.$$

- Определён класс функций, которые растут не медленнее, чем $g(n)$ с точностью до постоянного множителя, начиная с некоторого объёма данных N.
- Запись $\Omega(k \times \ln k)$ означает класс функций, которые растут не медленнее, нежели $g(k) = k \times \ln k$.
- Традиционно в данном классе заключены все полиномы, степени больше единичной и все степенные функции, с основанием, большим единицы.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ

5. Асимптотический анализ. Виды оценок (3)

2. Оценка сверху $\boldsymbol{O}$ (О большое)

$f(n) = O[g(n)]$ или $f(n) \sim\leq g(n)$, если существуют целые N и C, такие, что

$$f(n) \leq C \cdot g(n) \text{ для всех } n \geq N.$$

- Оценка определяет класс функций, которые *растут не быстрее*, чем функция $g(n)$ с точностью до некоторого постоянного множителя;
- Функция $g(n)$ *мажорирует* $f(n)$.

Пример. Все функции:

$$f(n) = n^{-1};$$
$$f(n) = 3 \cdot n + 17;$$
$$f(n) = n \times \ln n;$$
$$f(n) = 6 \cdot n^2 + 24 \cdot n + 77$$

мажорируются функцией $g(n) = n^2$, поэтому справедлива оценка $O(n^2)$.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
5. Асимптотический анализ. Виды оценок (4)

3. Оценка Θ (тэта)

$f(n) = \Theta[g(n)]$ или $f(n) \sim= g(n)$, если существуют целые N, C_1 и C_2, такие, что

$$C_1 \cdot g(n) \le f(n) \le C_2 \cdot g(n) \text{ для всех } n \ge N.$$

- Оценка Θ интерпретируется как одновременное выполнение оценок Ω и O.
- Формально $\Theta[g(n)]$ есть пересечение Ω и O.
- Запись $f(n) = \Theta(1)$ – означает, что функция либо равна константе, либо ограничена константой на бесконечности.

Пример.

Для функции $f(n) = 4 \cdot n^2 + n \cdot \ln n + 174$ оценка составит $\Theta(n^2)$, а для функции

$f(n) = 7 + n^{-1}$ оценка $\Theta(n^{-1/2})$ предпочтительнее, чем $\Theta(1)$.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
6. Асимптотический анализ. Методы (1)

Существо вопроса: для $f(n) = n^3 - 30 \cdot n$ считается, что сложность растёт как $g(n) = n^3$. Причина в том, что разница при $n = 100$, разница между $f(n)$ и $g(n)$ составляет 0,3 %.

Основоположники методов Д. Грин (Daniel H. Greene) и Д. Кнут сформулировали, что цель асимптотического анализа “состоит в нахождении хорошего приближения к точному решению”

1. **Метод раскрутки**

Название дали переводчик монографии “Mathematics for the Analysis of Algorithms” Б.Б. Походзей и редактор Ю.В. Матиясевич.

В оригинале *boot strapping*, буквально означает “натягивание сапога на ногу за лямки”

- Метод применяют, когда анализируемая функция удовлетворяет некому неявному уравнению.
- Последовательно подставляя в уравнение асимптотическое приближение к функции, полученное на предыдущем шаге, добиваемся улучшения асимптотической оценки.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
6. Асимптотический анализ. Методы (2)

Пример. Уравнение

$$f(t)\cdot \mathbf{e}^{f(t)} = t \qquad (1)$$

может быть делением с последующим логарифмированием преобразовано к виду

$$f(t) = \ln t - \ln f(t). \qquad (2)$$

Поскольку $f(t) > 1$ при $t > e$ получим оценку

$$f(t) = O(\ln t). \qquad (3)$$

Подстановка (3) в (2) даёт

$$f(t) = \ln t + O(\ln \ln t). \qquad (4)$$

Для дальнейшего улучшения снова подставим (4) в (2), получим:

$$f(t) = \ln t - \ln \ln t - \ln\left[1 + O\left(\frac{\ln \ln t}{\ln t}\right)\right] = \ln t - \ln \ln t + O\left(\frac{\ln \ln t}{\ln t}\right) \qquad (5)$$

Продолжая подстановки, можно получить решение с потребной точностью…

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
6. Асимптотический анализ. Методы (3)

2. Метод расчленения

- Применяется к сумам и интегралам.
- Сумма многокомпонентная.
- Ни один из компонентов не является пренебрежимым на всей области суммирования или интегрирования.

Пусть имеем функцию сложности вида

$$f(n) = \sum_{3 \le d \le \frac{n}{2}} \frac{1}{d\left(\frac{n}{d}\right)^d} \qquad (1)$$

Разобьём сумму на интервалы. Пусть первый интервал $d \in [3; 8]$. Для него

$$\sum_{3 \le d \le 8} \frac{1}{3\left(\frac{n}{8}\right)^3} \qquad (2)$$

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
6. Асимптотический анализ. Методы (4)

Сложность этой функции $O(n^{-3})$. Замена d на 3 либо 8 производится из соображения увеличения слагаемого. Второй интервал $d \in [8; \sqrt{n}]$ даёт

$$\sum_{8 \le d \le \sqrt{n}} \frac{1}{8\left(\frac{n}{\sqrt{n}}\right)^8} \tag{3}$$

Оценка сложности в этом случае $O(n^{-4}\sqrt{n})$ Данная оценка есть результат суммирования $O(\sqrt{n})$ членов, каждый из которых равен $O(n^{-4})$

Третий интервал $d \in \left[\sqrt{n}; \frac{n}{2}\right]$. Для этого случая сумма

$$\sum_{\sqrt{n} \le d \le \frac{n}{2}} \frac{1}{\sqrt{n} \cdot 2^{\sqrt{n}}} \tag{4}$$

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
6. Асимптотический анализ. Методы (5)

имеет оценку сложности $O\left(\frac{\sqrt{2}}{2^{\sqrt{n}}}\right)$. Если сопоставить оценки сложности,

полученные на различных интервалах, то оценка $O(n^{-4})$ доминирует все остальные оценки, поэтому функция оценки сложности есть

$$O\left\{\sum_{3 \le d \le \frac{n}{2}} \frac{1}{d\left(\frac{n}{d}\right)^d}\right\} = O(n^{-3})$$

Самое сложное в методе – разбиение исходного интервала суммирования на подынтервалы.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
7. Замечания (1)

1. Каждый из классов $\Omega[g(n)]$, $\Theta[g(n)]$ и $O[g(n)]$ является множеством. В ходе асимптотического анализа принято писать, например, $f(n) = \Theta[g(n)]$, хотя более корректно будет $f(n) \in \Theta[g(n)]$, поскольку $f(n)$ является элементом множества.
2. Класс $O[g(n)]$ чрезвычайно важен для анализа. Пусть имеем алгоритмы $A1$ и $A2$. Если классы сложности $O[g_{A1}(n)]$ и $O[g_{A2}(n)]$ совпадают, то это значит, что алгоритм $A2$ решает поставленную задачу не лучше $A1$.
3. Проверить принадлежит ли функция классу $O[g(n)]$ можно либо по определению, либо найдя предел:

$$\lim_{n\to\infty}\frac{g(n)}{f(n)} = C.$$

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
7. Замечания (2)

4. Важен корректный выбор функции

Пусть имеем функции:

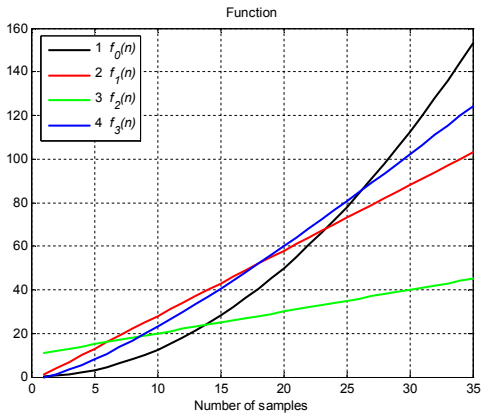
$$f_0 = n^2 / 8; \quad (1)$$

$$f_1 = 3\cdot n - 2; \quad (2)$$

$$f_2 = n + 10; \quad (3)$$

$$f_3 = n\cdot \ln n. \quad (4)$$

Поведение функций



ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
7. Замечания (3)

Пусть $f(n)$ и $g(n)$ функции, из которых нужно выбрать асимптотическую оценку алгоритма, либо асимптотические оценки алгоритмов A_1 и A_2.

Мера $\rho[f(n), g(n)]$ – количественная оценка расхождения функций $f(n)$ и $g(n)$.

Порог значимости h – величина, мерой расхождения $\rho[f(n), g(n)]$ не более которой можно пренебречь.

Свойства меры:

- $\rho[f(n), g(n)] = -\rho[g(n), f(n)]$ – симметрия меры;
- $\rho[f(n), g(n)] = 0 \Rightarrow f(n) = g(n)$ – сходимость меры.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
7. Замечания (4)

1. Пусть $\rho[f(n), g(n)] \le h$.

   Функции $f(n)$ и $g(n)$ асимптотически эквивалентны, оба алгоритма A_1 и A_2 характеризуются одинаковой трудоёмкостью.

2. Пусть $h - \rho[f(n), g(n)] < 0$.

   Функция $f(n)$ доминирует $g(n)$, которая возрастает не быстрее $f(n)$, то есть, имеем нижнюю оценку трудоёмкости $f(n) = \Omega[g(n)]$, а алгоритм A_2 лучше, чем A_1.

3. Пусть $\rho[f(n), g(n)] + h < 0$.

   Функция $f(n)$ доминируется $g(n)$, поэтому имеем нижнюю оценку трудоёмкости $f(n) = O[g(n)]$, и алгоритм A_1 предпочтительнее A_2.

Важно точность сравнения определяется

- заданием величины порога h;
- методикой расчёта меры расхождения $\rho[f(n), g(n)]$.

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
7. Замечания (5)

Пример Дж. МакКоннела. Имеется некий алгоритм подсчёта

```
for all 256 символов do
    ОБНУЛИТЬ СЧЁТЧИК
end for
while not eof do
    ВВЕСТИ ОЧЕРЕДНОЙ СИМВОЛ
    УВЕЛИЧИТЬ СООТВЕТСТВУЮЩИЙ СЧЁТЧИК
end while
```

ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ
7. Замечания (6)

При обработке файла длиной N грубым подсчётом установлено:

- операций присваивания 256 (без учёта работы цикла);
- проверок условий $N + 258$ (с учётом проверок условия выхода из цикла);
- инкрементов $N + 256$ (без учёта работы цикла).

Если $N = 500$, то алгоритм выполняет 1771 операций, из которых 770 на инициализацию, то есть 43%. Если $N = 50\,000$, то операций 100 771, а доля инициализации составит менее 1%.