Алгоритмы и структуры данных

Лекция 1. Сложность алгоритма

Преподаватель: Тазиева Рамиля Фаридовна

Основные понятия

Алгоритм – это точное предписание, определяющее вычислительный процесс, ведущий от варьируемых начальных данных к исходному результату.

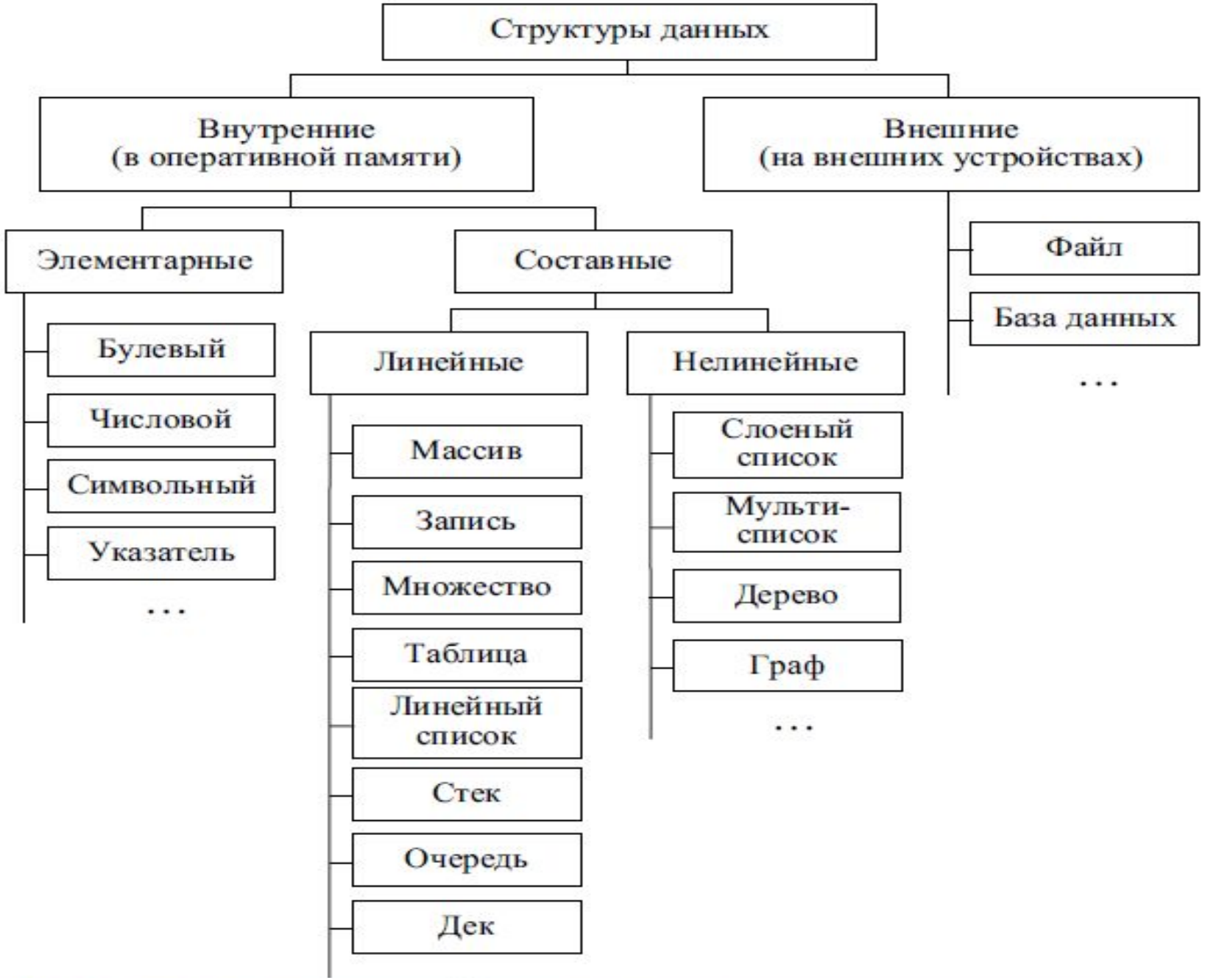
Под **структурой данных** в общем случае понимают множество элементов данных и связей между ними.

Физическая структура данных – способ физического представления данных в памяти ЭВМ.

Рассмотрение структуры данных без учета ее представления в памяти называется **абстрактной** или **логической структурой**.

Информация по каждому типу однозначно определяет:

- структуру хранения данных указанного типа, т.е. выделение памяти, представление данных в ней и метод доступа к данным;
- множество допустимых значений, которые может иметь тот или иной объект описываемого типа;
- набор допустимых операций, которые применимы к объекту описываемого типа.


Структуры данных
Внутренние
(в оперативной памяти)
Внешние
(на внешних устройствах)
Элементарные
Составные
Файл
База данных
...
Булевый
Числовой
Символьный
Указатель
...
Линейные
Нелинейные
Массив
Запись
Множество
Таблица
Линейный список
Стек
Очередь
Дек
Слоеный список
Мульти-список
Дерево
Граф
...

Сложность алгоритма

Алгоритм должен удовлетворять следующим требованиям:

1) быть простым для понимания, перевода в программный код и отладки;
2) эффективно использовать вычислительные ресурсы и выполняться по возможности быстро.

Сложность алгоритма – это величина, отражающая порядок величины требуемого ресурса (времени или дополнительной памяти) в зависимости от размерности задачи.

Сложность алгоритма:

Пространственная сложность V(n). Временная сложность T(n).

Пример:

Число тактов необходимых для работы алгоритма $T(n)=11n^2+19n*\log n+3n+4$,

Временная сложность алгоритма T(n) имеет порядок $O(n^2)$.

n	3n	19n logn	$11n^2$
1,00	3,00	0	11,00
16,00	48,00	366,05	2 816,00
256,00	768,00	11 713,68	720 896,00
4 096,00	12 288,00	281 128,30	184 549 376,00
65 536,00	196 608,00	5 997 403,75	47 244 640 256,00
1 048 476,00	3 145 428,00	119 935 810,66	12 092 321 148 336,00
16 775 616,00	50 326 848,00	2 302 770 205,03	3 095 634 213 974 020,00

Теоретическая оценка сложности алгоритма

1. Если операция выполняется а фиксированное число шагов, не зависящее от количества данных, то принято писать O(1).
2. Время выполнения операций присваивания, чтения, записи обычно имеют порядок O(1).
3. Время выполнения операций в данной последовательности совпадает с наибольшим временем выполнения операции в последовательности (правило сумм: если $T_1(n)$ имеет порядок O(f(n)), а $T_2(n)$ - порядок O(g(n)), то $T_1(n)+T_2(n)$ имеет порядок O(max(f(n),(g(n))).
4. Время выполнения конструкции ветвления (if-then-else) состоит из времени вычисления логического выражения (обычно имеет порядок O(1)) и наибольшего из времени, необходимого для выполнения операций, исполняемых при истинном значении логического выражения и при ложном значении логического выражения.
5. Время выполнения цикла состоит из времени вычисления условия прекращения цикла(обычно имеет порядок O(1)) и произведения количества выполняемых итераций цикла на наибольшее возможное время выполнения операций тела цикла.
6. При наличии в алгоритме операции безусловного перехода, необходимо учитывать изменения последовательности операций, осуществляемых с использованием этих операций безусловного перехода.

О-символика

Пусть даны две функции f(n) и g(n) натурального аргумента n, значениями которых являются действительные числа.

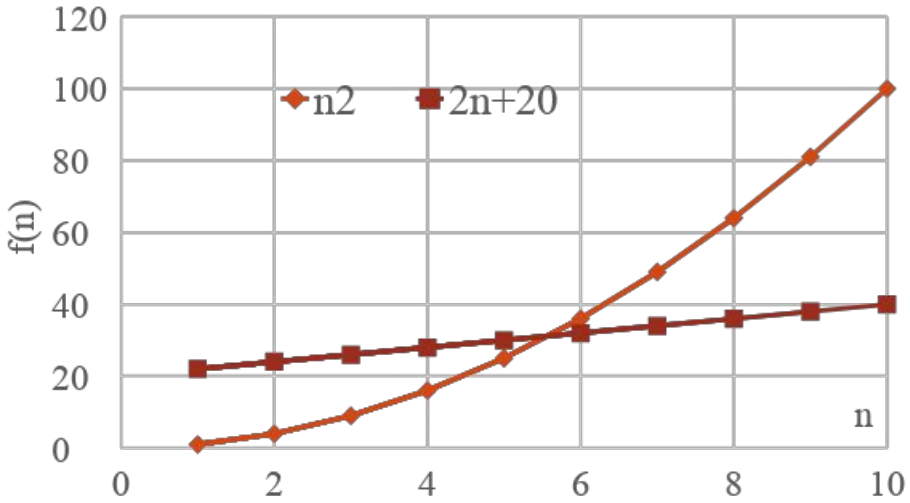
Говорят, что f=O(g) («f растет не быстрее g»), если существует такая константа c>0, что f(n)≤c·g(n) для всех n.

Пример

Алгоритм 1 : $f_1(n) = n^2$ **Алгоритм 2** : $f_2(n) = 2n + 20$.

$$\frac{2n+20}{n^2} = \frac{2}{n} + \frac{20}{n^2} \leqslant 22$$

$$\frac{n^2}{2n+20} \geqslant \frac{n^2}{22n} = \frac{n}{22}$$



Алгоритм 3 : $f_3(n) = n + 1$.

$$\frac{f_2(n)}{f_3(n)} = \frac{2n+20}{n+1} \leqslant 20 \quad \text{и} \quad \frac{f_3(n)}{f_2(n)} \leqslant 1.$$

$f_2 = O(f_3)$ и $f_3 = O(f_2)$

$f_2 = \Theta(f_3)$ и $f_1 = \Omega(f_3)$.

Обозначение $O(\cdot)$ можно считать аналогом $\leq$. Аналоги для $\geq$ и $=$ такие:

$f = \Omega(g)$ (f растёт не медленнее g, с точностью до константы) означает $g = O(f)$;

$f = \Theta(g)$ (f и g имеют одинаковый порядок роста) означает, что $f = O(g)$ и $g = O(f)$.

Правила

Правила замен:

1. Постоянные множители можно опускать. Например, $14n^2$ можно заменить на n^2.
2. n^a растёт быстрее n^b для $a > b$. Например, в присутствии слагаемого n^2 можно пренебречь слагаемым n.
3. Любая экспонента растёт быстрее любого многочлена (полинома). Например, 3^n растёт быстрее n^5.
4. Любой полином растёт быстрее любого логарифма.

Упражнение:
Сравните сложность следующих алгоритмов используя символики O (аналог «≤»), Θ (аналог «=»), Ω (аналоги для «≥»):

	$f(n)$	$g(n)$		$f(n)$	$g(n)$
A	n-100	n-200	**E**	10 log n	$\log(n^2)$
B	$n^{1/2}$	$n^{2/3}$	**F**	$n^{1/2}$	$5^{\log 2n}$
C	n log n	10n log (10n)	**G**	$n2^n$	3^n
D	log(2n)	log(3n)	**H**	2^n	2^{n+1}

Числовые алгоритмы

Операция сложения

Пример: 53+35

Сложность алгоритма: O(n).

перенос:	1			1	1	1	
		1	1	0	1	0	1 (53)
		1	0	0	0	1	1 (35)
	1	0	1	1	0	0	0 (88)

Операция умножения

Пример: 13*11

				×	1	1	0	1
					1	0	1	1
					1	1	0	1 (1101 умножить на 1)
+				1	1	0	1	(1101 умножить на 1, сдвинутое на 1)
+			0	0	0	0		(1101 умножить на 0, сдвинутое на 2)
+		1	1	0	1			(1101 умножить на 1, сдвинутое на 3)
	1	0	0	0	1	1	1	1 (143 в двоичной системе счисления)

Метод Аль-Хорезми

11	13	
5	26	
2	52	(вычёркиваем)
1	104	
	143	(ответ)

Для последовательного сложения n чисел, потребуется (n-1) шагов:

$$\underbrace{O(n)+O(n)+\ldots+O(n)}_{n-1 \text{ раз}}=O(n^2)$$

Сложность алгоритма: $\Omega(n^2)$.

Метод «разделяй и властвуй»

Принципы:

1. Задача разбивается на несколько более простых *подзадач* (subproblems) того же типа.
2. Подзадачи решаются рекурсивно.
3. Из ответов для подзадач строится ответ для исходной задачи.

Пример :

1. Пусть x и y – это два n-битовых числа (n есть степень двойки).
2. Разобьём каждое из чисел x и y на две половины длины n/2:

$$x = \boxed{x_L}\boxed{x_R} = 2^{n/2}x_L + x_R,$$

$$y = \boxed{y_L}\boxed{y_R} = 2^{n/2}y_L + y_R.$$

Например, если x = 10110110_2 , то x_L =1011, x_R =0110 и x =1011 · 2^4+0110. (операция умножения – левый сдвиг).

3. Заметим, что

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R.$$

4. Сложение и домножение на степень двойки (сдвиг влево) производятся за линейное время от длины чисел O(n). Произведения $x_L y_L$, $x_L y_R$, $x_R y_L$, $x_R y_R$ можно вычислить за четыре рекурсивных вызова.

5. Обозначив через $T(n)$ общее время работы алгоритма на n-битовых числах, приходим к следующему рекуррентному соотношению:

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n).$$

6. Для ускорения алгоритма нам и понадобится замечание Гаусса

$(x_L + x_R)(y_L + y_R) = x_L y_L + x_L y_R + x_R y_L + x_R y_R$. $x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R.$

7. Вычисляя x·y, можно обойтись тремя произведениями (n/2)-битовых чисел: $x_L y_L$, $x_R y_R$, $(x_L + x_R)(y_L + y_R)$. Время работы соответствующего алгоритма удовлетворяет соотношению:

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n).$$

8. Чтобы оценить время работы алгоритма, посмотрим на дерево рекурсии.

на уровне k 3^k задач
размер задач $n/2^k$

Уровень	Сложность
верхний k=0	O(n)
Нижний $k = \log_2 n$	$O(3^{\log 2n})$ или $O(n^{\log 23}) = O(n^{1,59})$

общее количество операций на k-м уровне: $3^k \cdot O\left(\frac{n}{2^k}\right) = \left(\frac{3}{2}\right)^k \cdot O(n).$

Основная теорема

Если бы не трюк Гаусса, то коэффициент ветвления был бы равен 4. У дерева тогда было бы $4^{\log 2n} = n^2$ листьев, и время работы алгоритма было бы квадратичным.

Представим себе алгоритм, работающий по методу «разделяй и властвуй». Пусть он сводит данную задачу размера ***n*** к ***a*** подзадачам размера ***n/b*** и из найденных ответов строит ответ для исходной задачи. Будем считать, что на разбиение и сборку уходит время $O(n^d)$ (в рассмотренном выше алгоритме умножения $a=3$, $b=2$, $d=1$). Рекуррентное соотношение на время работы такого алгоритма:

$$T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d).$$

Основная теорема. *Пусть* $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ *для некоторых констант* $a > 0$, $b > 1$, $d \geqslant 0$. *Тогда*

$$T(n) = \begin{cases} O(n^d), & \text{если } d > \log_b a, \\ O(n^d \log n), & \text{если } d = \log_b a, \\ O(n^{\log_b a}), & \text{если } d < \log_b a. \end{cases}$$

Упражнения

Задание 1

Упражнения

Определите скорость роста функций по рекуррентным соотношениям.

- $T(n) = 2T(n/2) + n^3$
- $T(n) = T(9n/10) + n$
- $T(n) = 7T(n/3) + n^2$
- $T(n) = 7T(n/2) + n^2$

Основная теорема. *Пусть* $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ *для некоторых констант* $a > 0$, $b > 1$, $d \geqslant 0$. *Тогда*

$$T(n) = \begin{cases} O(n^d), & \text{если } d > \log_b a, \\ O(n^d \log n), & \text{если } d = \log_b a, \\ O(n^{\log_b a}), & \text{если } d < \log_b a. \end{cases}$$

Определить скорость роста алгоритма бинарного поиска.