# Алгоритмы и структуры данных

## Лекция 1. Алгоритмы и их свойства

Антон Штанюк (к.т.н, доцент)

18 февраля 2021 г.

Нижегородский государственный технический университет им. Р.Е. Алексеева
Институт радиоэлектроники информационных технологий
Кафедра "Компьютерные технологии в проектировании и производстве"

# Содержание


Понятие алгоритма

Способы записи алгоритмов

Теоретические подходы к понятию алгоритма

- Нормальные алгорифмы Маркова
- $\lambda$-исчисление
- Машина Тьюринга

Список литературы

# Понятие алгоритма

# Понятие алгоритма

### Определение алгоритма

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

Слово алгоритм арабского происхождения и связано с именем математика и астронома *Абу Абдулла Абу Джафар Мухамад ибн Муса ал-Хорезми*, который в 825 г. н.э. описал придуманную в Индии позиционную десятичную систему исчисления.

# Свойства алгоритмов

1. Дискретность. Процесс решения протекает в виде последовательности отдельных действий, следующих друг за другом.
2. Элементарность действий. Каждое действие является настолько простым, что оно не допускает возможности неоднозначного толкования.
3. Определенность. Каждое действие определено и после выполнения каждого действия однозначно определяется, какое действие будет выполнено следующим.
4. Связанность. На каждом следующем шаге используются результаты предыдущих.
5. Конечность. Алгоритм заканчивает работу после конечного числа шагов.
6. Результативность. В момент прекращения работы алгоритма известно, что является результатом.
7. Массовость. Алгоритм описывает некоторое множество процессов, применимых при различных входных данных.

# Понятие алгоритма

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.

Для работы алгоритма требуется исполнитель - человек или механизм, понимающий способ записи и интерпретации действий.

# Алгоритмизация

Алгоритмизация - процесс систематического составления алгоритмов для решения поставленных задач. Она является необходимым шагом в процессе разработки программного обеспечения.

Для составления алгоритмов нужны творческие способности, знание теории и способов записи алгоритма. Для исполнения алгоритма необходимо неукоснительно следовать каждому шагу.

# Способы записи алгоритмов

# Способы записи алгоритма

- Словесный (естественный язык, ЕЯ)
- Ограниченный естественный язык (ОЕЯ)
- Схематический (графический)
- Псевдоязык (в том числе языки программирования)

# Использование естественного языка

Наиболее простой способ - ЕЯ (естественный язык)

**Пример алгоритма для приготовления кипятка**

1. взять чайник
2. открыть крышку
3. если есть вода, перейти к шагу 5
4. налить воды
5. поставить на огонь
6. дождаться струи пара

# Использование естественного языка

Достоинства:

1. Понятен всем носителям языка
2. Не требует специального синтаксиса
3. Легко поддаётся редактированию

Недостатки:

1. Неоднозначен (некоторые слова и выражения могут толковаться по-разному)
2. Трудно изучаем для иностранцев, не владеющих языком
3. Несёт в себе избыточность в плане количества информации

# Использование ограниченного естественного языка

Мы можем устранить неоднозначность, оставив только *однозначные* слова.

**Пример**

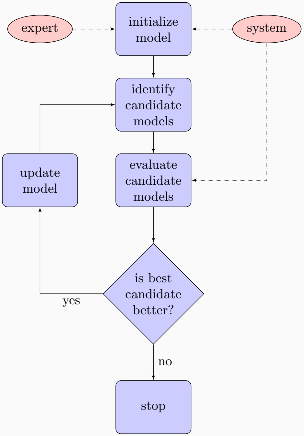
1. Посмотреть налево
2. Посмотреть направо
3. Перейти дорогу

Каждое слово может быть *командой* или *параметром*. В примере слова **Перейти, Посмотреть** относятся к командам, а остальные - к параметрам.

Хороший пример командного языка: армейский. В условиях боевых действий нет времени задумываться о смысле заданий.

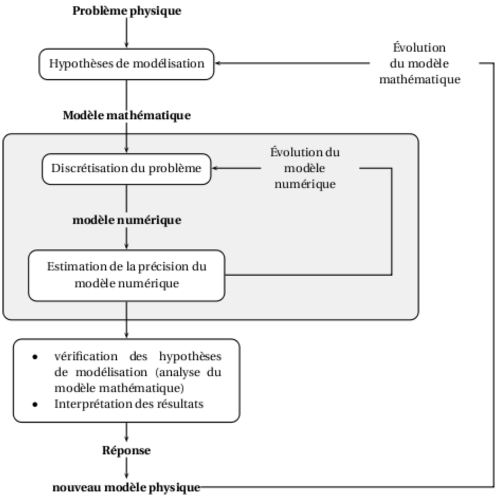
# Использование схем и диаграмм

Схемы, рисунки и диаграммы часто бывают более наглядны, но могут требовать и пояснений.



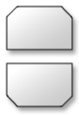
# Использование схем и диаграмм

В данной схеме сочетается языки блок-схем (flowchart) и французкий.



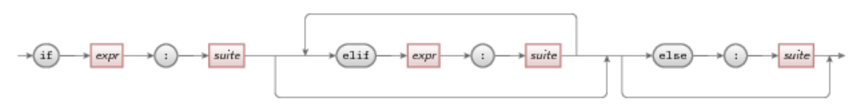
Элементы блок-схем строятся (согласно **ГОСТ 19.701-90**) из следующих элементов:

| | |
|---|---|
| Пуск-останов |  |
| Процесс |  |
| Решение |  |
| Предпроцесс |  |

| | |
|---|---|
| Ввод-вывод |  |
| Цикл |  |
| Соединитель |  |
| Комментарий |  |

# Использование схем и диаграмм

Синтаксические диаграммы:



# Использование псевдоязыка

Псевдоязык очень напоминает алгоритмический язык программирования, но использует обобщённые для многих языков конструкции

**procedure** Euclid($a, b$) ▷ НОД a и b
    $r \leftarrow a \bmod b$
    **while** $r \neq 0$ **do** ▷ ответ получен, если r равно 0
        $a \leftarrow b$
        $b \leftarrow r$
        $r \leftarrow a \bmod b$
    **end while**
    **return** $b$ ▷ НОД - b
**end procedure**

# Использование псевдоязыка

Блок IF

**if** $quality \geq 9$ **then**
    $a \leftarrow perfect$
**else if** $quality \geq 7$ **then**
    $a \leftarrow good$
**else if** $quality \geq 5$ **then**
    $a \leftarrow medium$
**else if** $quality \geq 3$ **then**
    $a \leftarrow bad$
**else**
    $a \leftarrow unusable$
**end if**

Блок WHILE

$sum \leftarrow 0$
$i \leftarrow 1$
**while** $i \leq n$ **do**
    $sum \leftarrow sum + i$
    $i \leftarrow i + 1$
**end while**

Блок REPEAT

$sum \leftarrow 0$
$i \leftarrow 1$
**repeat**
    $sum \leftarrow sum + i$
    $i \leftarrow i + 1$
**until** $i > n$

# Использование псевдоязыка

Блок FOR

$sum \leftarrow 0$
**for** $i \leftarrow 1, n$ **do**
    $sum \leftarrow sum + i$
**end for**

$s \leftarrow 0$
$p \leftarrow 0$
**for** $i \leftarrow 1,\ 10$ **do**
    $s \leftarrow s + i$
    $p \leftarrow p + s$
**end for**

# Теоретические подходы к понятию алгоритма

# Теоретическое определение алгоритма

**Определение *А.Н. Колмогорова:***

Алгоритмом принято называть систему вычислений, которая для некоторого класса математических задач из записи *А* «условий» задачи позволяет при помощи однозначно определенной последовательности операций, совершаемых «механически», без вмешательства творческих способностей человека, получить запись *В* «решения» задачи.

Таким образом, алгоритм - это процедура, которая:

1. Применяется к строго определенным исходным данным А.
2. Представляется в виде последовательности простых шагов; каждый шаг состоит в непосредственной обработке возникшего к этому шагу состояния S в состояние $S^* = \Omega(S)$.
3. Преобразование S в $S^*$ производится однозначно заданным способом.
4. Процесс переработки А продолжается до тех пор, пока либо не произойдет безрезультатная остановка, либо не появится сигнал о получении решения В. При этом не исключается возможность непрекращающегося процесса переработки.

# Теоретическое определение алгоритма

$$A_0 = A$$

$$A_1 = \Omega(A_0)$$

$$A_2 = \Omega(A_1)$$

$$\ldots$$

$$A_n = \Omega(A_{n-1})$$

$$\ldots$$

$$B = A_k$$

Каждый алгоритм задает функцию, которая по набору исходных данных выдает результат применения алгоритма к этим данным. Такая функция называется *вычислимой*.

# Подходы к понятию алгоритма

Были предложены математические трактовки алгоритма:

1. аппарат частично-рекурсивных функций;
2. нормальные алгорифмы Маркова;
3. лямбда-исчисление Чёрча;
4. машина Тьюринга.

Для современного программиста интерес представляют два последних подхода.

Лямбда-исчисление легло в основу **функционального подхода** к программированию.

Машина Тьюринга заложила фундамент архитектуры ЭВМ и привела к появлению **императивного подхода**.

# Определение

Для определения нормального алгорифма Маркова (НАМ) вводится алфавит, с помощью которого описывается и алгоритм и обрабатываемые данные. В алфавит также включается пустой символ $\epsilon$, который может означать **пустое слово**.

Всякий НАМ определяется множеством пар слов алфавита, которые называются **подстановками**. В паре слов подстановки левое (первое) слово непустое, а правое может быть и непустым и пустым. Слова разделяются стрелками. В качестве данных берется непустая строка символов. Схема подстановки: $\alpha \to \beta$.

# Работа НАМ

Работа НАМ состоит из последовательности шагов, которые описываются следующем образом:

1. В последовательности подстановок ищем самую первую подстановку, левое слово которой входит в строку данных.
2. В строке данных ищем самое левое (первое) вхождение левого слова $\alpha$ найденной подстановки
3. Это вхождение заменяем на правое слово $\beta$ найденной подстановки (преобразование данных).

# Работа НАМ

Шаги повторяются до тех пор, пока

- возникает ситуация когда шаг не может быть выполнен из-за отсутствия подстановки - *правило остановки*;
- устанавливается, что процесс подстановок не может остановиться.

Следует обратить особое внимание на тот факт, что на каждом шаге формулы в НАМ всегда просматриваются начиная с самой первой.

# Пример 1

Рассмотрим алгоритм увеличения десятичного целого числа на единицу. В алфавите у нас будут присутствовать цифры от 0 до 9 и служебный символ *.

Работа алгорифма описывается следующим набором правил:

1. $0* \to 1$
2. $1* \to 2$
3. $2* \to 3$
4. $3* \to 4$
5. $4* \to 5$
6. $5* \to 6$
7. $6* \to 7$
8. $7* \to 8$
9. $8* \to 9$
10. $9* \to *0$
11. $** \to 1$
12. $* \to \epsilon$

# Пример 1

Рассмотрим работу НАМ на примере чисел *30,79,99*:

$$*30* \xrightarrow{1} *31 \xrightarrow{12} 31$$

$$*79* \xrightarrow{10} *7*0 \xrightarrow{8} *80 \xrightarrow{12} 80$$

$$*99* \xrightarrow{10} *9*0 \xrightarrow{10} **00 \xrightarrow{11} 100$$

## Пример 2

Пусть дано слово, состоящее из символов $A = a, b$ произвольной длины. Преобразовать его таким образом, чтобы сначала шли символы $a$, а затем $b$.

Единственное правило НАМ в данном случае: $ba \to ab$

Пока во входном слове справа хотя бы от одного символа $b$ есть символ $a$, эта формула будет переносить $a$ налево от этого $b$. Формула перестает работать, когда справа от $b$ нет ни одного $a$, это и означает, что все $a$ оказались слева от $b$. Например:

$$babba \to abbba \to abbab \to ababb \to aabbb$$

Алгоритм остановился на последнем слове, т.к. к нему уже неприменимо правило.

# $\lambda$-исчисление

Разработано А. Чёрчем в 1936 году. Представляет собой способ описания вычислимых функций и нашло яркое применение в современной **функциональной** парадигме программирования (например, в языке *Haskell*).

Алонзо Чёрч (1903-1995)

В основе $\lambda$-исчисления лежат два фундаментальных понятия:

1. абстракция (построение функции по заданному выражению): $F = \lambda x.t[x]$
2. аппликация (применение функции к заданному аргументу): $Fx$

# Запись функции

Функции записываются следующим способом (пример):

$\lambda x.x + 1$ - означает функцию $f(x) = x + 1$

$(\lambda x.x + 1)2$ - а это вызов (аппликация) функции с аргументом 2

Для вычисления последнего выражения используют $\beta$-редукцию, то есть подстановку вместо $x$ его значения. Формально это можно записать так:

$$(\lambda x.t)a = t[x := a]$$

# Вычисление выражений

Покажем, как можно вычислить выражение:

$$(\lambda x.x + 1)3 = (3 + 1) = 4$$

# Каррирование

Функция двух переменных $x$ и $y$ $f(x, y) = x + y$ может быть рассмотрена как функция одной переменной $x$, возвращающая функцию одной переменной $y$:

$$\lambda x.\lambda y.x + y$$

Такой процесс применим к функциям любой ”арности” и называется *каррированием*.

# Определение $\lambda$ терма

- **Переменные**: обозначаются буквами $x, y, z, \ldots$.
- **Константы**: фиксированные значения $\lambda$-нотации.
- **Комбинации**:
  - Применение функции к аргументу $fs$.
  - Комбинации левоассоциативны (скобки могут опускаться).
- **Абстракции**:
  - Абстракция терма $t$ по переменной $x$ обозначается как $\lambda x.t$.
  - $x$ может входить, а может и не входить в $t$.
- Других термов нет.

# Примеры $\lambda$ термов

$$x$$

$$(xz)$$

$$(\lambda x.(xz))((\lambda x.(xz))y)$$

$$(\lambda y.((\lambda x.(xz))y))((\lambda y.((\lambda x.(xz))y))w)$$

$$(\lambda z.(\lambda w.((\lambda y.((\lambda x.(xz))y))w)))$$

# Ассоциативность операций

- **Аппликация** ассоциативна *влево*: $Fabc = (((Fa)b)c)$
- **Абстракция** ассоциативна *вправо*: $\lambda xyz.t = (\lambda x(\lambda y(\lambda z(t))))$

# Свободные и связанные переменные

Абстракция $\lambda x.t[x]$ связывает переменную $x$ в терме $t$.

В функции $\lambda x.x + y$ переменная $y$ является *свободной*.

# Полезные примеры $\lambda$-выражений

| | |
|---|---|
| $\lambda x.x$ | функция идентичности (*id*) |
| $\lambda x.xy$ | функция *id*, применяемая к *y* |
| $\lambda x.\lambda y.y$ | функция, возвращающая функцию *id* |

# Преобразования (редукции)

Основные преобразования $\lambda$-выражений:

- $\alpha$**-редукция**: $\lambda x.t \to \lambda y.t[x := y]$, при условии, что $y$ не принадлежит множеству свободных переменных.
- $\beta$**-редукция**: $(\lambda x.t)a \to t[x := a]$.
- $\eta$**-редукция**: $\lambda x.tx \to t$.

# Альфа-редукция

В процессе подстановки необходимо следить, чтобы не появились новые связанные переменные. Например:

$$(\lambda xy.x)y = \lambda y.y$$

- ошибка!

Для исключения подобных ситуаций проводим переименование свободной переменной, чтобы она не стала связанной.

$$(\lambda xz.x)y = \lambda yz.y$$

- это называется $\alpha$-редукцией (преобразованием).

# Beta-редукция

$\beta$ - редукция позволяет подставить значения переменных в выражение.

А вот этот терм невозможно преобразовать:

$$(\lambda x.xx)(\lambda x.xx) = (\lambda x.xx)(\lambda x.xx) = \ldots$$

# Стратегии редукции

- **Аппликативная стратегия (по значению)**: преобразование начинается с самого ”внутреннего” подвыражения.
- **Нормальная стратегия (по имени)**: преобразование начинается с самого левого подвыражения.

Пример: $(\lambda xy.x)z((\lambda x.xx)(\lambda x.xx))$

Разные стратегии дают разные результаты. Какой правильный?

# Примеры из языка Scheme (диалект Lisp)

| Пример |
| --- |
| (define x (+ 3 5)) |
| (* x x) |
| (* 8 8) - аппликативный порядок |
| (* (+ 3 5) (+ 3 5)) - нормальный порядок |

# Алан Тьюринг

Машина Тьюринга - модель вычислительного механизма (1936)

А. Тьюринг
(1912–1954)

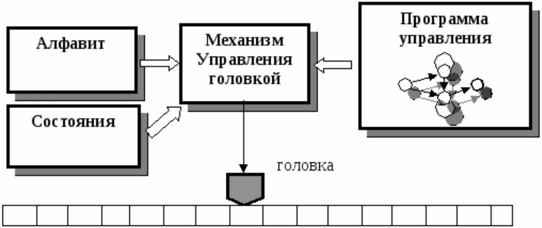


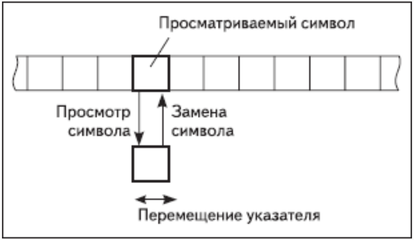

Рис. 1. Машина Тьюринга

# Строение МТ

В состав МТ входят:

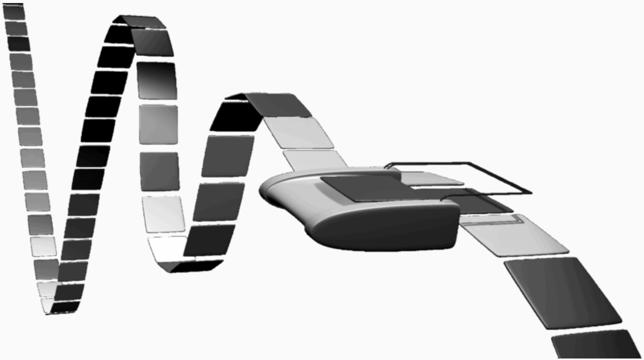
1. *Память* в виде бесконечной в обе стороны ленты, разбитой на ячейки, в каждую из которых может быть записан один символ из алфавита $A = \{a_0, a_1, \ldots, a_n\}$, причем $a_0$ является *пустым* символом.
2. *Головка записи/чтения*, способная читать и записывать символ в текущей ячейке.
3. *Автомат*, сдвигающий ленту влево или вправо относительно головки. Автомат подчиняется множеству команд $D = \{L, R, H\}$, что означает ”влево”, ”вправо”, ”на месте”.
4. *Устройство управления (УУ)*, осуществляющее выработку команд на движение ленты, запись символов в текущую ячейку памяти в зависимости от текущего символа ячейки и текущего состояния машины, задаваемого символом из алфавита $Q = \{q_0, q_1, q_1, \ldots, q_m\}$.

# Строение МТ



Просматриваемый символ
Просмотр символа
Замена символа
Перемещение указателя

# Современное изображение МТ



# Принцип работы

В начальный момент на ленте находится непустое слово (входное слово), УУ находится в начальном состоянии $q_0$, а головка записи/чтения находится над левым символом входного слова.

На каждом шаге работы головка считывает текущий символ с ленты и в зависимости от него и текущего состояния могут выполняться следующие действия:

- в текущую ячейку записывается новый символ;
- изменяется состояние УУ;
- автомат осуществляет сдвиг ленты по команде или остается на месте.

## Принцип работы

Если в результате некоторого количества шагов МТ переходит в состояние, при котором состояние УУ не меняется, символ не меняется и движения ленты не происходит, то говорят, что МТ переходит в заключительное состояние *останова*. При этом слово, оставшееся на ленте и будет является результатом работы алгоритма.

Действия МТ записываются в виде таблицы правил, которые имеют вид

$$q_i a_j \to q_{i1} a_{j1} d_k$$

Набор правил и является программой, по которой работает МТ.

# Добавление единицы

Реализуем функцию увеличения числа на единицу:

$$f(n) = n + 1$$

Запись правил (программы)

1. $\mathbf{q_0} * \to \mathbf{q_1 R}$ - если текущий символ $*$, а состояние $q_0$, то перейти в состояние $q_1$ и сдвинуться вправо.
2. $\mathbf{q_1 1} \to \mathbf{q_1 R}$ - если текущее состояние $q_1$, а символ в ячейке $1$, то не изменяя состояние сдвинуться вправо.
3. $\mathbf{q_1} * \to \mathbf{q_2 1}$ - если текущее состояние $q_1$, а символ в ячейке $*$, то записать в ячейку $1$ и остаться на месте.
4. $\mathbf{q_2 1} \to \mathbf{q_2 L}$ - если текущее состояние $q_2$, а символ в ячейке $1$, то сдвинуться влево.
5. $\mathbf{q_2} * \to \mathbf{q_3} *$ - состояние $q_3$ является состоянием *останова*

# Добавление единицы

Итак, если на вход МТ подается лента с записанным входным словом

$$*111*$$

,

то после работы программы на ленте останется слово

$$*1111$$

# Сложение двух чисел

1. $\mathbf{q_0} * \to \mathbf{q_1 R}$
2. $\mathbf{q_1 1} \to \mathbf{q_1 R}$
3. $\mathbf{q_1} * \to \mathbf{q_2 1}$
4. $\mathbf{q_2 1} \to \mathbf{q_2 R}$
5. $\mathbf{q_2} * \to \mathbf{q_3 L}$
6. $\mathbf{q_3 1} \to \mathbf{q_4} *$
7. $\mathbf{q_4} * \to \mathbf{q_5 L}$
8. $\mathbf{q_5 1} \to \mathbf{q_5 L}$
9. $\mathbf{q_5} * \to \mathbf{q_6} *$

Состояние $q_6$ в этой программе является состоянием останова.

# Реализация умножения

Рассмотрим программу для умножения двух чисел в унарной системе счисления.

Входное слово имеет вид

$$*111..111 \times 111...11 = *$$

где символы $*$ обозначают границы входного слова, единицы задают разряды перемножаемых чисел, а *x* - знак умножения (и разделитель чисел). Полученное в результате перемножения число будет записано справа от символа равенства $=$.

# Реализация умножения

| | |
|---|---|
| q0∗→q0R | q4a→q4aR |
| q01→q0R | q4=→q4=R |
| q0×→q1×R | q41→q41R |
| q11→q2aR | q4∗→q51R |
| q21→q21L | q5∗→q2∗L |
| q2a→q2aL | q6a→q61R |
| q2=→q2=L | q6×→q7×R |
| q2×→q3×L | q7a→q7aR |
| q31→q4aR | q71→q2aR |
| q3a→q3aL | q7=→q8=L |
| q3∗→q6∗R | q8a→q81L |
| q4×→q4×R | q8×→q9H |

# Реализация умножения

Так, при входном слове

$$*111\times 11=*$$

Получаем выходное слово

$$*111\times 111=111111*$$

за 152 шага.

# Машина Тьюринга в природе

Для производства белков в клетке с помощью сложно устроенного фермента — РНК-полимеразы — считывается информация с ДНК, своего рода информационной ленты машины Тьюринга. Здесь, правда, не происходит перезапись ячеек самой ленты, но в остальном процесс весьма похож: РНК-полимераза садится на ДНК и двигается по ней в одном направлении, при этом она синтезирует нить РНК — нуклеиновой кислоты, сходной с ДНК. Готовая РНК, отсоединяясь от фермента, несёт информацию к клеточным органеллам, в которых производятся белки.

Ещё более похож на машину Тьюринга процесс исправления ошибок в ДНК — её репарация. Здесь ДНК-полимераза вместе с другими белками двигается по ленте ДНК и считывает обе её половинки (геномная ДНК, как известно, представляет собой две переплетенных нити, несущих одну и ту же информацию). Если информация в половинках не совпадает, ДНК-полимераза принимает одну из них за образец и «правит» другую.

# Список литературы

# Список литературы i


- Кормен Т.,Лейзерсон Ч., Ривест Р.,
  *Алгоритмы: построение и анализ*.
  МЦНМО, Москва, 2000.
- Серджвик Р.
  *Фундаментальные алгоритмы на C++. Части 1-4*.
  Diasoft, 2001.
- Ахо А., Хопкрофт Д., Ульман Д.
  *Структуры данных и алгоритмы*.
  «Вильямс», 2000.
- Круз Р.Л.
  *Структуры данных и проектирование программ.*
  М.: Бином, 2014.
- Топп У., Форд У.
  *Структуры данных в C++*.
  М.: Бином, 1999.